Project Class COMP 767

Philippe Helal
ID 260949246
philippe.helal@mail.mcgill.ca

Badreddine Sayah
ID 260940022
badreddine.sayah@mail.mcgill.ca

April 30th, 2020

The video can be found at this link.

1 Introduction

Experience Replay is a simple but essential concept for function approximators like neural networks, such learning algorithms have proven their capacities to learn and approximate complex functions, they require a large amount of data though, implicitly or explicitly assumed to be independent and identically distributed. Experience Replay (ER) helps to satisfy this assumption by breaking up the strong correlations between successive states, this will bring more diversity between samples and more stability in learning the value functions. However, Experience Replay uniformly samples transitions at random, without considering their importance in learning. By contrast, Prioritized Experience Replay (PER) reuse older agent experiences according to their learning values. As stated by the authors, replaying transitions with high expected TD errors at a higher frequency will lead to better performance, speed up and efficiency in the exploitation of the memory buffer.

Our goal is to explore the Prioritized Experience Replay which was proposed by Schaul et. al. [3], we are interested to reproduce some of their results and study the behaviour of this important approach on some Atari games.This report is organized as follow: Section 2 describes the experience replay approach that we have implemented. Experiments and results are presented in Section 3. Section 4 presents some perspectives and our conclusions.

2 Method

In prioritized experience replay, transitions encountered by a reinforcement learning agent are stored in a replay memory and sampled according to their priority; specifically, by their corresponding temporal difference errors. This is the key idea that was proposed by Schaul et. al. in [3] to improve on the first experience replay method, where experiences are replayed at simple random uniform distribution, that would help to break correlated updates and avoid rapid forgetting by replaying older experiences. Nonetheless, ER approach suffers somehow from a lack of efficiency, because it considers that all experiences have an equivalent amount for the learning process. Our implementation can be found at this link.

The PER variant we implemented is *direct proportional prioritization* (following the pseudo-code of Algorithm 1 in the paper [3]). The priority of each batch of experience in this method[1] is measured by the last encountered absolute TD errors: $p_i = |\delta_i| + \epsilon$, ($\epsilon$ is a small constant, set to 1e-5).

As prioritizing with TD errors only may introduce several issues, for instance neglecting experiences with (immediate) low TD errors that could be useful later, sensitivity to noisy rewards and slow learning, Experiences are stochastically sampled following the probability defined as:

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

This will ensure that experiences with a low initial TD error will not have zero probabilities. Moreover the exponent α allows us to control the degree of prioritization and balance between a uniform sampling at random with $\alpha = 0$, and a pure greedy prioritizing with $\alpha = 1$.

We implemented the Double deep Q-network developed (DQN) proposed by Mnih et al. [1] to use it with the porortional PER. This agent combines (deep) neural networks and reinforcement learning. Its architecture consists of 5 layers: 3 convolutions each of which is followed by rectified linear activation function, and two dense layers. The first convolutional layer applies 32 filters with a kernel size of 8x8, the size of the output feature map is down-sampled with a stride of 4, the second layer applies 64 filters of 4x4 and a stride of 2, the last convolutions convolves 64 filters of 3x3 with a stride of 1. The dense layer is composed of 512 units followed by a ReLU activation, and the last dense layer of this network gives a single output for each possible action in the action space.

A weighted importance-sampling was used to reduce the bias introduced by the stochasticity in prioritized experience replay, these weights were computed by $w_j = (\frac{1}{N} \cdot \frac{1}{P_j})^\beta$, as in [3]

[1]Pure greedy prioritization

The parameter β gives control on the amount of bias correction by importance sampling. We decayed this exponent over time by:

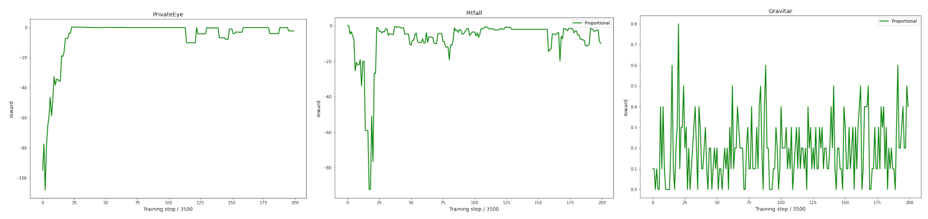
$$\beta = \beta_0 + t\left(\frac{1-\beta_0}{\varepsilon}\right)$$

where t is the current time-step and ε is a small positive constant

3 Experiment settings and results

To test our PER implementation, we picked three Atari games, namely Private Eye, Pitfall and Gravitar [2]. Adapting to the computational resources at our disposal, we ran each game for 700,000 total steps, in contrast to the authors' 200 million. Atari benchmarks contains challenging and realistic problem with complex functions to be approximated, often with a delayed reinforcement signal that may not appear immediately after taking an action [3]

Most of our hyperparameters follows the reported ones: we took mini-batches of 32 experiences, sampled from the memory buffer. The optimization algorithm used is Adam with a learning rate of 1e-4. The replay memory buffer can store 10^5 of the most recently observed transitions. Only the rewards were clipped to a range of [-1, 1] to stabilize the learning process, and not the temporal difference errors. For the proportional variant, considering the best hyperparameters reported in the paper [3], we set α to 0.6 and initial β to 0.4, the hyperparameter used to anneal the factor that controls the amount of the weighted importance sampling correction as mentioned before. Figures below depicts the observed rewards over the training steps for the three games.



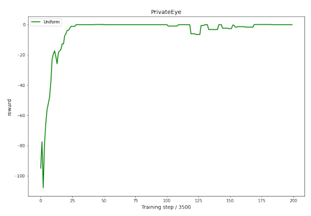
(a) Private Eye (b) Pitfall (c) Gravitar

Figure 1: proportional prioritized experience replay

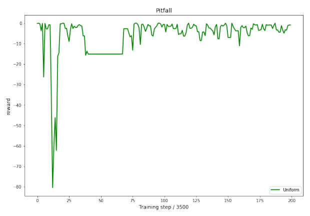
We can observe that Double DQN with this PER variant quickly converges for both games, however for Gravitar, the algorithm is still learning after 700,000 steps.

Setting the exponent α to zero (and/or increasing β) allows us to have a close behaviour to random uniform sampling [3], the plots in Figure 2 shows us the benefits of the prioritization regarding the speed of learning, on the Private

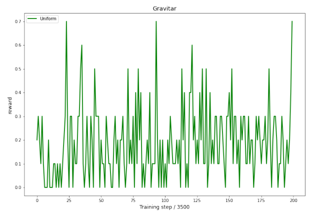
Eye task for example, we can observe that with uniform random sampling the agent required few more training steps. For the Pitfall and Gravitar games we notice much more fluctuations and instability in the learning process. In these runs, we kept the initial β of 0.4 for the β scheduler. other runs were performed with Initial beta of 0 and 1 were tried, the performance with uniform random sampling considerably decreased.



(a) Private Eye



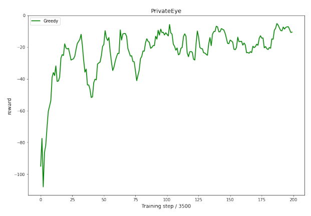
(b) Pitfall



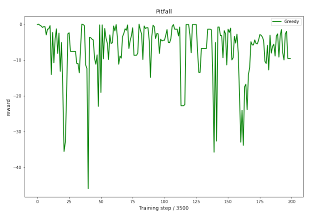
(c) Gravitar

Figure 2: Uniform random sampling ($\alpha = 0$)

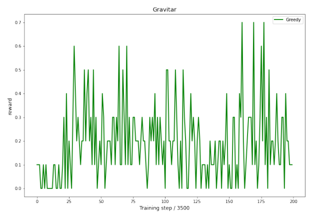
Furthermore, we were interested to try the pure greedy prioritization, thus we set α to 1 to obtain such behaviour. on these Atari games performance is clearly worse than the proportional method.



(a) Private Eye



(b) Pitfall



(c) Gravitar

Figure 3: Pure greedy prioritization ($\alpha = 1$)

4 Discussion

We can see that prioritized experience replay can accelerate the learning process over uniform replay in certain scenarios. Due to the limited resources at our disposal, we were unable to run the experiments for nearly as long as the original paper, in order to see if our implementation perfectly reproduces the authors' results, particularly when the learning curves start to converge. Additionally, implementing the rank-based variant of PER could be interesting as it is hypothesised to be more robust to outliers and hence more efficient.

5 State of contributions

Both authors contributed equally and jointly write the report and code together. We hereby state that the work presented in this report is that of the authors.

References

[1] Volodymyr Mnih et al. *Human-level control through deep reinforcement learning.* 2015.

[2] J Veness MG Bellemare Y Naddaf and M Bowling. *The arcade learning environment An evaluation platform for general agents. Journal of Artificial Intelligence Research.* 2012.

[3] I Antonoglou T Schaul J Quan and D Silver. *Prioritized Experience Replay. Conf. paper at ICLR.* 2016.

The Machine Learning Reproducibility Checklist (v2.0, Apr.7 2020)

For all **models** and **algorithms** presented, check if you include:

- ☐ A clear description of the mathematical setting, algorithm, and/or model.
- ☐ A clear explanation of any assumptions.
- ☐ An analysis of the complexity (time, space, sample size) of any algorithm.

For any **theoretical claim**, check if you include:

- ☐ A clear statement of the claim.
- ☐ A complete proof of the claim.

For all **datasets** used, check if you include:

- ☐ The relevant statistics, such as number of examples.
- ☐ The details of train / validation / test splits.
- ☐ An explanation of any data that were excluded, and all pre-processing step.
- ☐ A link to a downloadable version of the dataset or simulation environment.
- ☐ For new data collected, a complete description of the data collection process, such as instructions to annotators and methods for quality control.

For all shared **code** related to this work, check if you include:

- ☐ Specification of dependencies.
- ☐ Training code.
- ☐ Evaluation code.
- ☐ Pre-trained model(s).
- ☐ README file includes table of results accompanied by precise command to run to produce those results.

For all reported **experimental results**, check if you include:

- ☐ The range of hyper-parameters considered, method to select the best hyper-parameter configuration, and specification of all hyper-parameters used to generate results.
- ☐ The exact number of training and evaluation runs.
- ☐ A clear definition of the specific measure or statistics used to report results.
- ☐ A description of results with central tendency (e.g. mean) & variation (e.g. error bars).
- ☐ The average runtime for each result, or estimated energy cost.
- ☐ A description of the computing infrastructure used.

Reproduced from: *www.cs.mcgill.ca/~jpineau/ReproducibilityChecklist-v2.0.pdf*