Playing Atari with DQN

Alexey Didenkov
CS 5824: Advanced Machine Learning
Virginia Polytechnic Institute and State University
adidenkov@vt.edu

Abstract

This paper addresses the problem of training Deep Reinforcement Learning agents on Atari environments, evaluating the effect of several improvement strategies on their performance. Specifically, the practices of using prioritized replay buffers and target networks are found to improve the stability of training and better asymptotic behavior, while the presence of a replay buffer is found to be crucial. This evaluation is performed on dueling convolutional networks that receive video input from said environments. Minor evidence is found to suggest that unintentional forgetting in networks may get them out of local minima.

1 Introduction

Reinforcement learning is a vast field with applications in many areas. An important task of it is being able to train models that perform well and learn efficiently on human-level challenging tasks. Learning to play games shows a generalization of models' ability to do just that.

The primary approach of this project follows the results of Mnih et al. [2013]. In particular, the agents are evaluated on Atari games. Due to only receiving video input, they are tasked with learning to extract features. This prioritizes general intelligence and versatility, rewarding agents that can be applied to many games. These games are simplified versions of real-world tasks; they are human-like, and provide at least some degree of a challenge.

1.1 Problem

The primary task of this research is to train Reinforcement Learning agents that perform well on Atari game environments. Then, an evaluation of several strategies will be performed to measure the effect that each has on the overall score. A novelty of this approach is that it performs such analyses on Dueling networks.

1.2 Hypothesis

In particular, the evaluated methods aim to improve the stability of learning through various means. This means making the training curve less noisy and able to increase to a higher plateu. We speculate that such methods may come at a cost of slower initial learning, but should produce better asymptotic behavior.

2 Background

Q-learning is a type of Reinforcement Learnig agents. Such agents learn so-called Q-values; expected values of taking a certain action from a given state. As formulated by Mnih et al. [2013], these can be

Preprint. Under review.

expressed as:

$$Q^*(s,a) = \mathop{\mathbb{E}}_{s' \sim \varepsilon} \Big[r + \gamma \max_{a'} Q^*(s',a') | s,a \Big] \tag{1}$$

Deep neural networks may be used to create more complex agents, which are then known as Deep-Q Networks (DQN).

2.1 Double Q-Learning

Deep Q-learning overestimates action values, so one can use two models that update each other. In particular, in the version used by van Hasselt et al. [2015] one determines the policy, and the other determine the value. The updated Q-value function is then:

$$\mathcal{L}_i(\theta_i) = \mathop{\mathbb{E}}_{(s,a,r,s') \sim U(D)} \left[\left(r + \gamma Q\Big(s', \operatorname*{argmax}_{a'} Q(s',a';\theta_i); \theta_i^-\Big) - \qquad Q(s,a;\theta_i) \right)^2 \right] \tag{2}$$

2.2 Dueling Networks

Wang et al. [2016] propose further splittting the networks; using one for value function estimation, and the other for actions. The first network then evaluates states and the second does the actions. Their joint selection policy is thus:

$$Q(s,a) = V(s) + \Big(A(s,a) - \frac{1}{|\mathcal{A}|} \sum_{a' \in \mathcal{A}} A(s,a') \Big) \tag{3}$$

2.3 Experience Replay

Reinforcement Learning algorithms sample data from environments. As this is an expensive process, past data may be re-used for additional training without requiring new samples. In addition, Schaul et al. [2016] propose a version that assigns importance sampling weights defined as

$$w_i = \left(\frac{1}{N} \frac{1}{P(i)} \right)^\beta \tag{4}$$

This given an additional advantage of being able to replay more important sections in memory, somewhat analogous to learning from mistakes.

3 Results

The implementation used the lambl.ai framework developed by Varuna Jayasiri [2020]. The base model used was a 2D-convolutional net with 3 layers; kernel sizes 8, 4, 3 and strides 4, 2, 1. Environments were sampled with a fixed frameskip of 4; resulting images were converted to grayscale and resized to 84×84. Two Atari environments were evaluated: **Breakout** and **Q*bert**, with three representative metrics: episode length and reward, as well as sampled Q-values.

Default hyperparameters were as follows:

1. Exploration coefficient ϵ: Decays from 1.0 on step 0 to 0.1 on 25k, then to 0.01 on 1M.
2. Replay buffer priority β: Inreases from 0.4 on step 0 to 1.0 on 1M.
3. Replay buffer size: 2^{14}.
4. Learning rate λ: Dynamically updated bectween 10^{-4} and 10^{-3}.
5. Target model update frequency: 250 steps.

This dueling networ constitutes the *Default* model setup. Any modifications to it were subtractive; i.e. the decrease in performance upon removing any specific feature was interpreted as said feature's contribution to the default score.

3.1 Replay buffer

The replay buffer is a feature desined to both improve stability and make models less prone to critically forget past behavior. Several setups were evaluated; some with and others without replay buffers.

1. *Default*: A prioritized replay buffer.
2. *Unprionitized*: The priority coefficient β was set to 0 on step 0. By (4), this makes all sample weitghts identical and acts as a priority-free buffer. Note that β still increased over time, albeit its effect in the early stages was miniscule.
3. *No replay*: The size of the replay buffer was decreased to 1. This gives the network only the most recent point sampled from the environment, and is functionally equivalent to not having a replay buffer.

The results of this experiment can be seen in Figure 1.

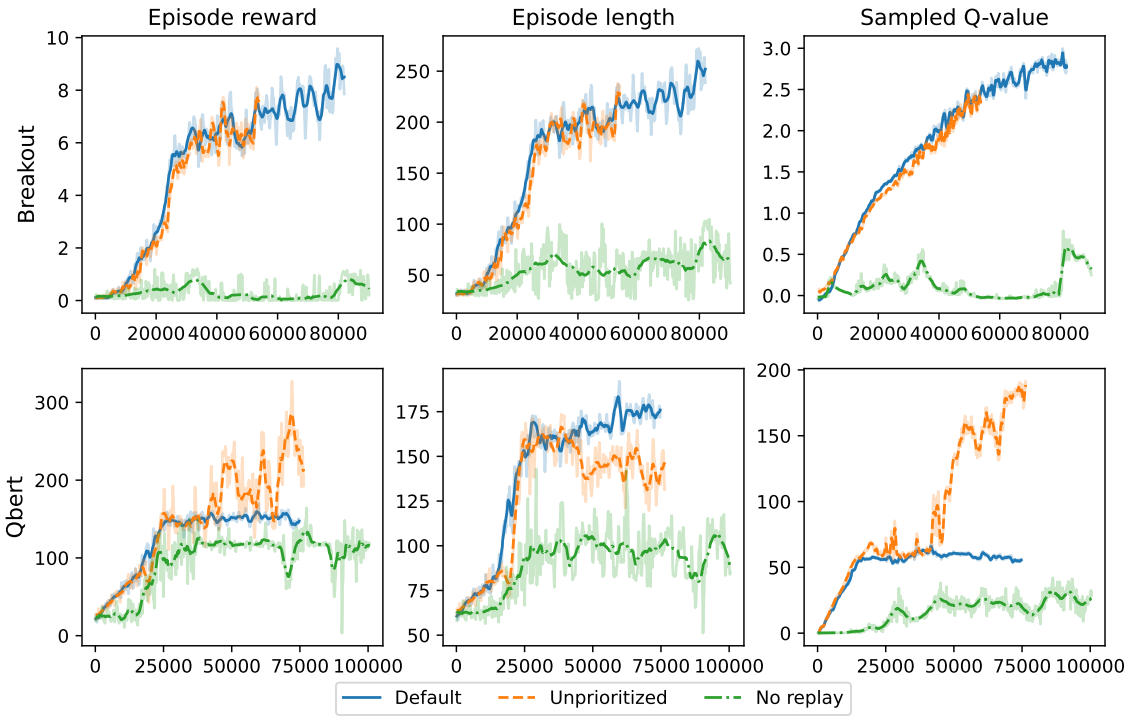

Figure 1: Effect of replay buffer on DQN performance.

3.2 Model update policy

The effect of using a target model was also evaluated. This setup varied updating two separate models vs updating a single one.

1. *Default*: Double-model update; target network is sampled while Q-network is updated.
2. *Single*: Model update frequency was set to 1. This makes the Q-model and target model identical each iteration, and is functionally equivalent to using a single model.

The results of this experiment can be seen in Figure 2.

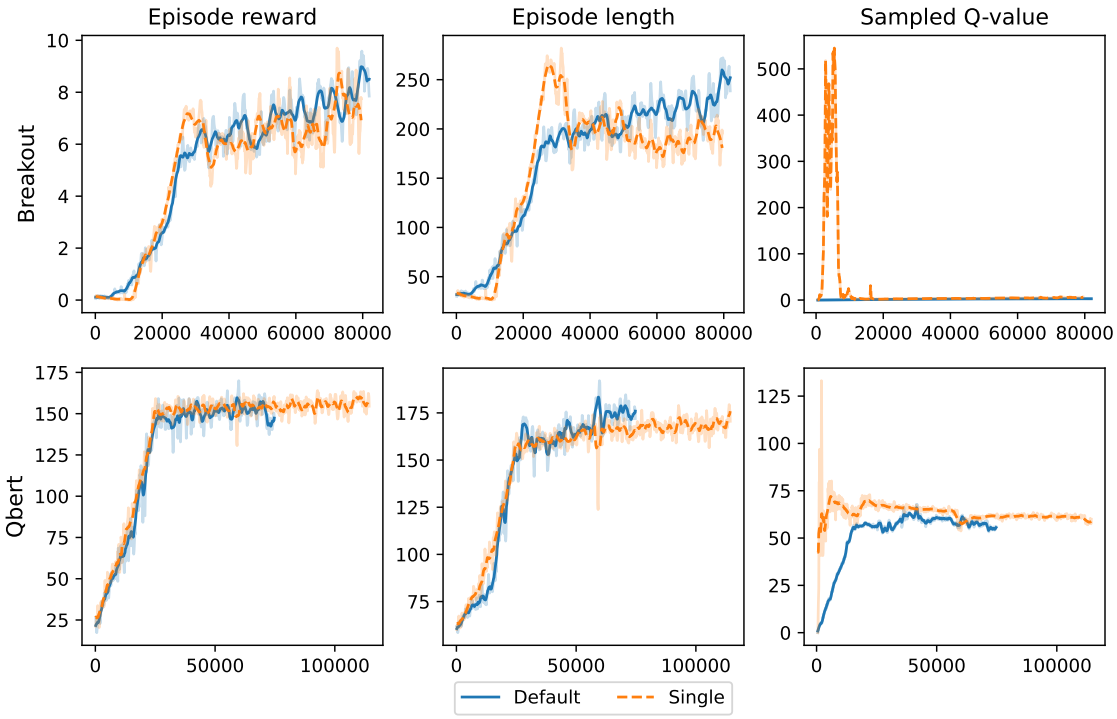

Figure 2: Effect of model update policy on DQN performance.

4 Analysis

The most evidently significant result was the presence of the replay buffer, as removing it consistently decreased model performance. This is seen in Figure 1, with the green dash-dotted line being below others in every metric. This decrease is more significant in the Breakout environment, possibly due to either an increased need to infer environment state from past frames, or the need to coordinate long-term strategies.

The Breakout environment does not show significant difference when removing priorities from the buffer, but Q*bert seems to function differently. The exhibited behavior seems contrary; it increases episode reward while decreasing episode length. This could indicate a policy of risky behavior, and suggests that the prioritized model may be learning in a more consistent manner.

Particularly noteworthy is a rapid increase in episode length on the Q*bert environment around step 20k. A comparison of this with temporal difference error is seen in Figure 2. This reveals that the rapid increase immediately follows a drop in TD error; possibly indicating either a local breakthrough, or an example of the model forgetting its past policy.

Lastly, omitting a target model does not appear to significantly change results. While evidence for this claim is limited, the experiment may suggest that using two networks leads to better asymptotic behavior, despite an intial slower learning curve.

On a side note, the top-right plot of Figure 2 shows an example of a model critically over-estimating its Q-values, suggesting that it may not be a good metric. This metric was used for consistency with Mnih et al. [2013].

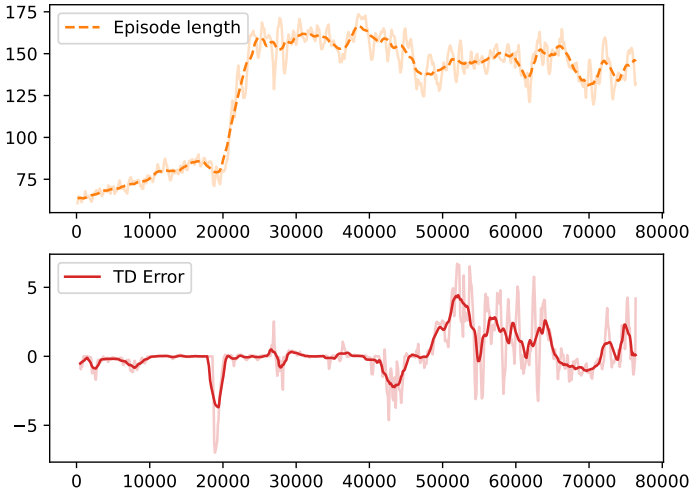

Figure 3: Example of forgetting; unprioritized Q*bert.

5 Conclusions

The results are consistent with the hypothesis; the use of both prioritized replay and a target network has shown better asymptotic behavior in learning. Additionally, the use of a replay buffer was found to be more crucial than originally thought.

5.1 Limitations

The most concerning factor of these results is the score obtained when compared to Mnih et al. [2013]. Evidently, their exploration terminated after 100 epochs of roughly 30 minutes each; i.e. a 50-hour training cycle. Due to session limits, these models were trained for only a tenth as long, and hence achieve comparatively lower scores. While decreasing the significance of these results, this places an additional emphasis on the models' asymptotic behavior.

5.2 Further research

Future research could address the phenomenon of forgetting induced by a priorty-free buffer. An isolated example shown in Figure 3 seems to suggest that forgetting may get models out of local minima.

Contributions

- Alexey - everything.

References

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning. *arXiv:1312.5602 [cs]*, December 2013. URL `http://arxiv.org/abs/1312.5602`. arXiv: 1312.5602.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized Experience Replay. *arXiv:1511.05952 [cs]*, February 2016. URL `http://arxiv.org/abs/1511.05952`. arXiv: 1511.05952.

Hado van Hasselt, Arthur Guez, and David Silver. Deep Reinforcement Learning with Double Q-learning. *arXiv:1509.06461 [cs]*, December 2015. URL `http://arxiv.org/abs/1509.06461`. arXiv: 1509.06461.

Nipun Wijerathne Varuna Jayasiri. labml.ai Annotated Paper Implementations, 2020. URL `https://nn.labml.ai/`.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling Network Architectures for Deep Reinforcement Learning. *arXiv:1511.06581 [cs]*, April 2016. URL http://arxiv.org/abs/1511.06581. arXiv: 1511.06581.