Efficient Double DQN with Self-Supervised Learning

Jia Liang Zhou
Department of Engineering
University of Bologna
Bologna, Italy
jialiang.zhou@studio.unibo.it

Abstract

Taking inspiration from existing methods, which leverage self-supervised learning methods to improve the efficiency of Reinforcement Learning algorithms during the training phase, an approach combining BYOL method with a Double DQN agent is proposed.

1 Introduction

This is the report written for the project of the **Autonomous and Adaptive Systems** course held by Professor Mirco Musolesi at University of Bologna.

The code is available here: `https://github.com/tibe97/unibo_aas_project.git`

1.1 Deep Reinforcement Learning

The field of Reinforcement Learning has seen great successes thanks to the deployment of Deep Neural Networks as function approximators between states or state-action pairs to values, representing the optimality of being in a certain state or of taking a certain action in that state. Tabular methods need one entry for each state-action pair, which becomes clearly infeasible with complex environments such as games with a huge number of possible state configurations. Deep Neural Networks instead, only need to store the weights, which can be trained and updated with the computation of a loss, optimized iteratively with Stochastic Gradient Descent algorithm. Here we focus on value-based methods.

1.2 Data-Efficient RL

Typically, in RL agents are trained for millions of steps or interactions, while more recent works have been exploring more data-efficient ways to interact with the environments ([7], [4], [5]). More in particular, *Atari 100k* [7] has been used to benchmark the performance of RL agents with limited data, where the agent is allowed to interact with the environment for 100k steps only. In this work we follow these approaches, also due to the limitations of computational resources, and we try to improve over the baseline performances of standard DQN algorithm [3]. We propose a method combining Double DQN[6] algorithm for reinforcement learning with BYOL [1] algorithm for ***self-supervised*** representation learning. Self-supervised learning in computer vision has proved outperform supervised learning techniques, without the need of using labels, which can be very expensive to produce. The goal of self-supervised learning techniques is to learn good representations without relying on labels, and usually intermediary tasks are introduced as objective, before finetuning or trasfer learning with a smaller set of labeled data. By learning a good representations of the states, given as consecutive images or frames from the games, we aim to improve the policy by learning better estimates for the state-action values.

Preprint. Under review.

2 Related Work

There have been previous attempts to leverage the advantages of self-supervised learning methods in order to improve the quality of the state representations.

Contrastive Unsupervised Representations for Reinforcement Learning (CURL) [5] uses the **MoCo** framework [2] with Rainbow DQN. The states, which in our case are stacks of frames from the Atari games, are randomly augmented into two views. These views are fed to the query and the key encoders, which outputs or embeddings are compared in the InfoNCE loss. This loss makes the embeddings of the augmented views of the same state similar to each other, while pushing away the embeddings of different states in the representation space. This task is indeed Contrastive Learning, where we use positive and negative examples to guide the mapping of different or similar images in the embedding space. The reason of using negative examples is to avoid collapsed solutions: using only positives can result in "easy" solutions, where the network gets stuck at outputting constant embedding vectors which trivially lead to the maximum similarity value and to the smallest contrastive loss. To prevent this, negatives are introduced in order to separate the embeddings of different examples in the representation space, by contrasting positive examples against negative examples.

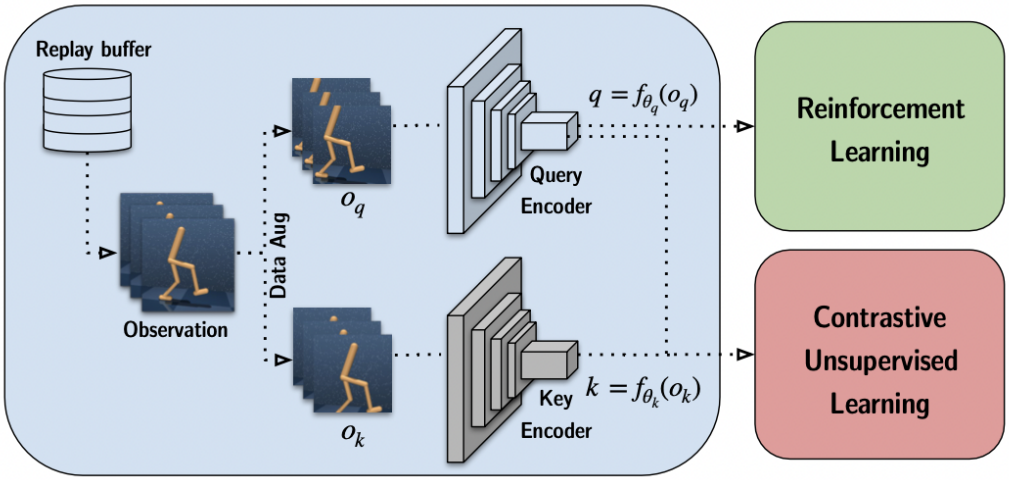

Figure 1: Overview of CURL method: the stack of images undergoes random data augmentations to produce two views of the same example. The two views are then used to compute both Q-values for reinforcement learning and embedding projections for contrastive unsupervised learning.

3 Method

3.1 Double DQN

Double Q-learning [6] solves the known problem of overestimating action values for the Q-learning algorithm. It uses a different network, the target network, to estimate the values of the next state-action pairs, while still computing the actions with the current network, the online network. This decoupling of selection from the estimation reduces the overestimation of the values. The weights of the target network are updated every t steps with a copy of the weights of the online network, so there is no gradient flow in the target network during training. It is also possible to perform soft updates, where the parameters of the target network are gradually updated at every time step.

3.1.1 Bootstrap Your Own Latent (BYOL)

This self-supervised learning method for computer vision proposes a solution to learn from unlabeled data without the need of using negatives examples, as it is common practice in self-supervised learning. In order to avoid collapsed solutions, the authors of BYOL use two separate networks, the online and the target networks, to output the prediction vector and the projection vector separately.

We start first with a set of image augmentations which is randomly applied twice to the same batch of images, in order to produce two different augmented views, v and v'. These two views v and v' are fed to the online network and to the target network to produce respectively $q_\theta(z_\theta)$ and z'_ξ, which are then used to predict one from the other in the loss, computed as follows:

$$L_{BYOL} = 2 - 2(l_2norm(q_\theta(z_\theta)) \cdot l_2norm(z'_\xi))$$

where $l_2norm(\cdot)$ is the l2-normalization function. The dot product between the normalized vectors gives us the cosine similarity between the embeddings. Minimizing this loss is equivalent to maximizing the similarity between $q_\theta(z_\theta)$ and z'_ξ.

During training, the target network is updated as an exponential moving average of the online network. Given the parameters of the online network θ and the parameters of the target network ξ, the formula of the update is:

$$\xi \leftarrow \tau\xi + (1-\tau)\theta$$

where $\tau \in [0, 1]$ is the target decay rate.

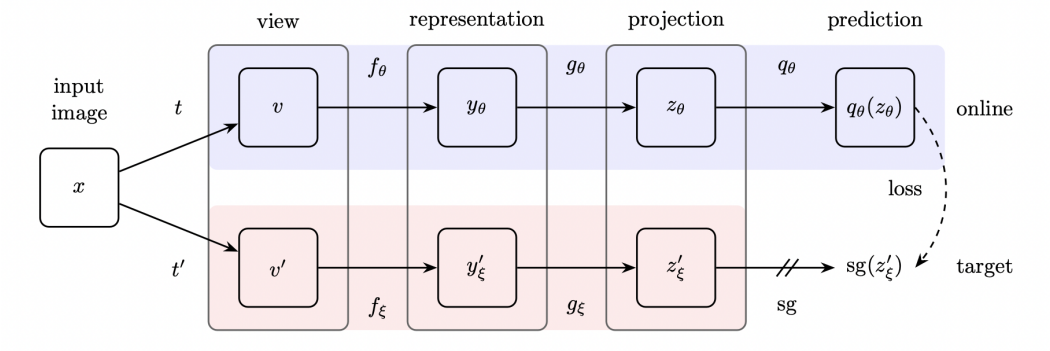

Figure 2: Overview of BYOL method.

3.1.2 Implementation details

The main reason for selecting BYOL as the self-supervised learning framework, over all the other available techniques, is that it uses two separate networks, the online and the target, similarly to Double DQN. In this way we can exploit the two networks for both computation of Q-values and representation learning. The individual losses are summed in final loss, so both the learning tasks are performed at the same time:

$$L = L_{RL} + L_{BYOL}$$

The target network is updated with soft updates, and not every t time steps. The training loop implemented for the experiments follows the pseudo-code presented in the original DQN paper [3].

4 Experiments and Results

4.1 Atari 100k

In this work we only focus environments we discrete action spaces. To train and to evaluate the methods in a more data-efficient way, we use the Atari games limited to 100k environment steps, or 400k frames if using frame-skip of 4. In these experiments only the Breakout game is used for simplicity.

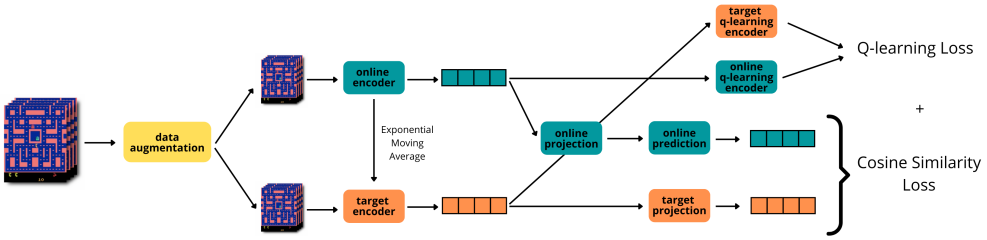

Figure 3: Overview of the proposed method: Q-values are computed from both online and target networks, the same as the prediction and the projection vectors.

Table 1: Hyperparameters used during training.

Breakout	
Parameter	Value
max-steps	100k
eval-interval	500
eval-episodes	10
batch-size	32
replay-buffer-size	5000
base-lr	0.00025
τ (soft-updates)	0.99

4.2 Experimental settings

We trained the agents for 100k steps or interactions with the environment. Random agent and Double DQN agent were used as baselines for the results evaluation.

All these combinations were run during the experiments:

- *Random agent*
- *Double DQN*
- *Double DQN + BYOL*
- *Double DQN + Temporal BYOL* (using stack of frames S and S_{t+1} for the two augmented views

The evaluation of the agents is performed during the training every 500 time steps and they are evaluated for 10 consecutive episodes, averaging the total rewards from all the episodes.

The **backbone** of both the networks follows the same architecture of presented in the original DQN paper [3]. The full network architecture is shown in table 2.

The ***data augmentation*** pipeline includes the following operations in sequence:

- Random cropping + Resizing
- Random contrast change
- Random horizontal flip
- Random brightness change
- Gaussian filtering with kernel size (5,5)

4.3 Results

The rewards collected during the 100k training steps were very noisy, so an Exponential Moving Average smoothing was applied with a factor of 0.9 (figure 4). We can clearly see that the worst

Table 2: Convolutional neural network used for the online and the target networks.

Architecture	
Layer	Specs
Input	size=(84,84,4)
Conv1	filters=32, kernel-size=(8,8), strides=4, activation=ReLU
Conv2	filters=64, kernel-size=(4,4), strides=2, activation=ReLU
Conv3	filters=64, kernel-size=(3,3), strides=1, activation=ReLU
Embedding	size=512
Outputs	
Q-values	size=4, The same as the number of action of Breakout
Projection	size=256, FC-layer with Embedding as input
Prediction	size=256, FC-layer with Projection as input

performance is given by the *Random Agent*, as expected. The *DQN* and *Double DQN* baselines performed slighly better than the *Random Agent*. Even if the *DDQN+BYOL* and *DDQN+BYOL Temporal* achieved the best results, these were not the expected performances.

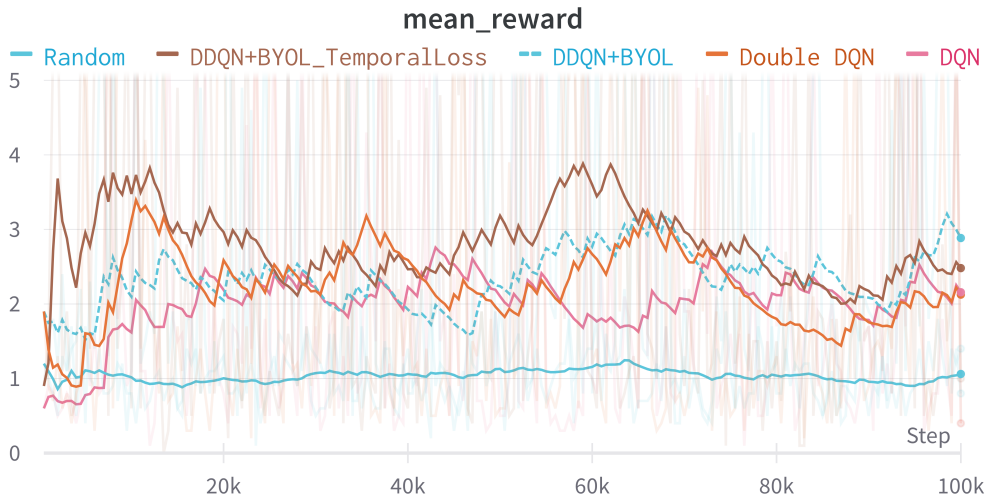

Figure 4: **Atari Breakout**: mean reward of 10 evaluation episodes during training (100k steps).

This low performance is probably due to the *BYOL* module, which did not help to improve the representations of the states. In fact, after few training steps, the *BYOL* loss immediately drops to values very close to 0, both for *DDQN+BYOL* and *DDQN+BYOL Temporal* (figure 5). This may be due to the networks reaching collapsed solutions, probably for the projection and for the prediction heads only, while not affecting too much the backbone shared with the action-values head. If the backbone was affected as well, we would probably not have similar results as standard *DQN*.

Stronger augmentations, lower learning rates and batch normalization were also experimented to prevent collapsing, but no improvements were achieved.

5 Conclusions

In this work a different approach was explored to tackle the problem of learning policies in a data-efficient way. Experiments were conducted on the Atari Breakout environment, but even if it is a not so complex game, the expected results were not achieved. The training process was unstable, with the rewards range varying considerably. Smoothing helped however to visualize and to compare the

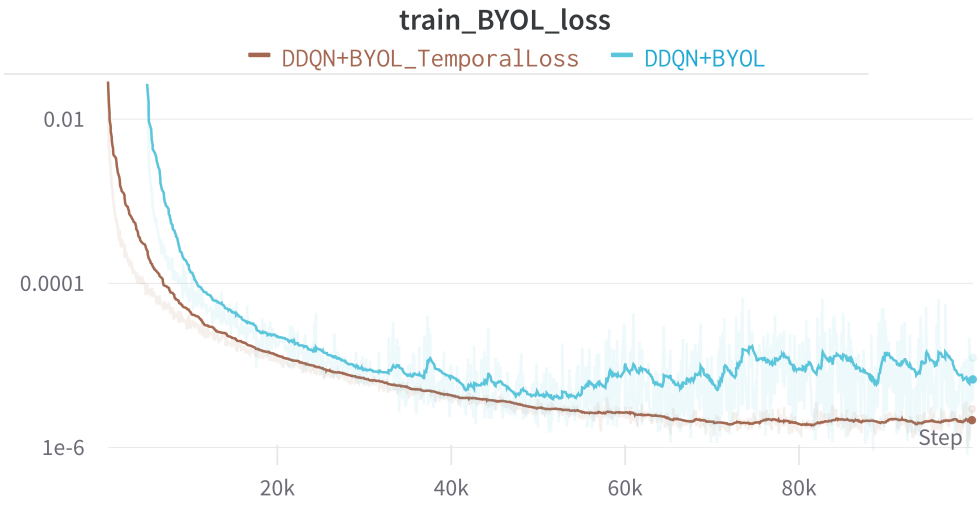

Figure 5: **Atari Breakout**: mean reward of 10 evaluation episodes during training (100k steps).

methods more clearly. It would be useful to investigate further on the causes of the collapse of the *BYOL* module, in order to fully exploit the advantages of self-supervised learning.

References

[1] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H. Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Ávila Pires, Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, Bilal Piot, Koray Kavukcuoglu, Rémi Munos, and Michal Valko. Bootstrap your own latent: A new approach to self-supervised learning. *CoRR*, abs/2006.07733, 2020.

[2] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross B. Girshick. Momentum contrast for unsupervised visual representation learning. *CoRR*, abs/1911.05722, 2019.

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[4] Max Schwarzer, Ankesh Anand, Rishab Goel, R. Devon Hjelm, Aaron C. Courville, and Philip Bachman. Data-efficient reinforcement learning with momentum predictive representations. *CoRR*, abs/2007.05929, 2020.

[5] Aravind Srinivas, Michael Laskin, and Pieter Abbeel. CURL: contrastive unsupervised representations for reinforcement learning. *CoRR*, abs/2004.04136, 2020.

[6] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *CoRR*, abs/1509.06461, 2015.

[7] Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, and Yang Gao. Mastering atari games with limited data. *CoRR*, abs/2111.00210, 2021.