Value-Approximation from Pixels: a Sample Efficiency Perspective on DQN-Like Models

Martina Rossini
M.Sc. in Artificial Intelligence
Alma Mater Studiorum - University of Bologna
martina.rossini3@studio.unibo.it

Abstract

In this work we implement different improvements over the standard DQN algorithm and test them for sample efficiency by training each new agent over the Ms.Pacman game environment for 100k steps, inspired by the Atari 100k benchmark. In particular, we implemented (i) double DQN, (ii) DQN with prioritized replay, (iii) DQN with dueling networks, (iv) DQN with noisy-nets for exploration and, finally, (v) we added image augmentation and multiple updates per step to our best model. We were able to achieve a mean score per episode of 860.802 ± 161.787 and a mean human-normalized score of 0.091.

1 Introduction

Deep RL algorithms trained directly on image pixels are notoriously not sample efficient, and this is true in particular for model-free agents like DQN and its more recent variations. Indeed, the original human-level results obtained on the Atari 2600 benchmark by Mnih et al. (2015) required 50 million training frames, which corresponds to about 38 days of game experience. More recent algorithms like Rainbow (Hessel et al., 2017) were trained for up to 200 million frames and the whole process took approximately 10 days. Requiring such an high number of interactions with the environment can have an unbearable cost in non-simulated environments – thus strongly limiting the application of RL models in real-world industrial and robotics applications.

Indeed, one of the main reasons for sample inefficiency when learning from pixels is the difficulty of training the network to appropriately encode the states and to, at the same time, predict the action values. The shallow CNN encoders which are commonly used produce a poor representation of the states, which limits performance. At the same time Kostrikov et al. (2020) pointed out how naively trying to use deeper network results in clear overfitting. These problems are exacerbated by the usually sparse reward signals and by the correlation between samples, leading to algorithms that require tens of millions of interactions with the environment to converge.

Kaiser et al. (2019) introduced a new benchmark (Atari 100k), where the goal is to solve 26 Atari games while limiting the number of agent steps to 100k. Since it was first introduced in 2019, research efforts in solving this benchmark have mainly focused in three directions: (i) model-based algorithms like EfficientZero (Ye et al., 2021), (ii) algorithms that employ an auxiliary loss like CURL (Srinivas et al., 2020) and SPR (Schwarzer et al., 2021) or (iii) algorithms with a focus on data augmentation and on increasing the number of network updates per interaction with the environment like Dr.Q (Kostrikov et al., 2020) or OTRainbow (Kielak, 2020).

However, there is a distinct lack of comparison of those that are currently considered the best model-free algorithms on this benchmark (i.e.: Dr.Q, CURL) with some actual baselines – like classic and double DQN. This work aims to build custom implementations of the aforementioned algorithms as well as some of their common variations (see Section 2 for more details) and to evaluate their performance after just 100k training steps. Due to computational and timing constraints, for the actual experiments we did not use the whole Atari 100k suite but just the Ms.Pacman game.

1.1 The environment

Ms.Pacman objective is to score as many points as she can while trying to escape from four ghosts that are trying to eat her. She scores 10 points every time she eats one of the dots in the maze, while if she eats an "energy pill" she scores 50 points: moreover, this scares the four ghosts which turn blue for a limited amount of time, during which they can be eaten themselves. Note that Ms.Pacman scores 200 points for the first ghost she manages to eat during the effect of the pill, 400 for the second, etc. Since taking the same action may result in obtaining different rewards, we use reward clipping in order not to give mixed signals to the agent while learning. Notice that in the game you can never lose points (i.e.: you never get a negative reward) but you do lose a life every time Ms.Pacman gets eaten by a ghost and after three lives it's game over. Also notice that Ms.Pacman can receive additional points by eating cherries, bananas and other snacks which appear on the screen in a stochastic manner.

The Atari game screen has a resolution of 210x160 pixels with 128 possible colors at 60Hz. Following the pre-processing established by Mnih et al. (2015), each frame was reshaped to a size of 84x84 and transformed to grayscale. Moreover, since from a single frame you cannot obtain information related to, for instance, directionality, we actually stack together the four most recent frames and pass this stack as input to the network.

2 Algorithms

As briefly mentioned in the Introduction, we tried to solve the Ms.Pacman game using different versions of the DQN value approximation algorithm. In particular, we implemented the basic DQN agent introduced by Mnih et al. (2015) as well as the following popular variations: double DQN, prioritized replay, DQN with dueling architecture and DQN with noisy networks. Finally, we also experimented with data augmentation techniques which, as pointed out by Kostrikov et al. (2020) can improve sample efficiency by a large margin. A brief description of all the aforementioned architectures is given in Sections 2.1 through 2.4.

2.1 DQN and Double DQN

DQN (Mnih et al., 2015) is a value approximation method which combines classical Q-learning with Artificial Neural Networks (ANN) in order to estimate the state-action value function Q_*. Classic TD learning algorithms like Q-learning are also known as tabular methods as they usually represent the state-action space through a table with one row per pair. In many real world applications however, the size of the state-action space makes storing it as a table prohibitive in terms of memory requirements. Thus, DQN uses an ANN (i.e.: a universal approximator), as a parametrized function $\hat{q}$ which learns how to map state-action pairs to their Q-value.

As in traditional TD methods, there is no need to wait for the end of an episode in order to make the update: let's suppose that, at time t, the neural network has parameters θ_t, the state is s_t and the action we select using an ϵ-greedy policy is a_t. We now call r_{t+1} the reward we obtain from the environment when taking action a_t from s_t, while s_{t+1} is the new state. Then, immediately after this step we can estimate the expected cumulative reward as $y_t = r_{t+1} + \gamma \max_a \hat{q}(s_{t+1}, a, \theta_t)$, where γ is a discount factor used to give more weight to current rewards with respect to possible future ones: we use this estimation as the target value of our the update. In practice, the network parameters are learned by using stochastic gradient descent to minimize either the MSE or the Huber loss between estimated target y_t and current Q-value $\hat{q}(s_t, a_t, \theta_t)$. Supposing the loss to be MSE, the function to minimize is

$$\left[r_{t+1} + \gamma \max_a \hat{q}(s_{t+1}, a, \theta_t) - \hat{q}(s_t, a_t, \theta_t)\right]^2 \tag{1}$$

Notice that the network used by Mnih et al. (2015) is composed by three convolutional layers all followed by ReLU activation

- the first one produces 32 channels and has kernel size 8x8 and stride 4
- the second one produces 64 channels, has a kernel of 4x4 and stride 2

- the last one has 64 channels, kernel size 3x3 and stride 1

This convolutional feature extractor is followed by a fully connected layer of size 512 with ReLU activation, which is then connected with the output layer: a dense layer with a neuron for every valid action in the game.

Experience Replay Mnih et al. (2015) noted that, since consecutive samples are strongly correlated, the updates of standard online Q-learning have a big variance – which makes the algorithm quite inefficient and slow to converge. Thus they proposed *experience replay*, where the agent's experiences $e_t = (s_t, a_t, r_{t+1}, s_{t+1})$ at every step t are stored in a replay memory M: the Q-learning updates are then applied over samples of experiences drawn at random from M. This approach also implies that each experience e_t could potentially be used in many weight updates, thus allowing for greater data efficiency. Also notice how learning by experience replay requires the agent to learn off-policy, as the current parameters are different from those that were used to generate the samples.

Target Network Mnih et al. (2015) also introduced the concept of *target network*: they use a second network $\overline{q}$ with parameters $\overline{\theta}_t$ in order to generate the target for the update. The weights of this target network are not learned but simply copied over from the online network $\hat{q}$ every C updates. This helps improving the stability of the algorithm with respect to standard Q-learning: indeed, in Equation 1 every update to $\hat{q}$ also changes the target y_t, and this can cause oscillations or even divergence during training. Thus, the only difference with respect to what we showed above is the estimation of the target $y_t = r_{t+1} + \gamma \max_a \overline{q}(s_{t+1}, a, \overline{\theta}_t)$: this also means that in case of MSE loss the Equation 1 becomes

$$\left[r_{t+1} + \gamma \max_a \overline{q}(s_{t+1}, a, \overline{\theta}_t) - \hat{q}(s_t, a_t, \theta_t)\right]^2 \tag{2}$$

Notice that updating $\overline{q}$ by copying over the weights of $\hat{q}$ every C step is called an *hard update*: the possibly abrupt change this introduces in the estimation of the target y_t may cause fluctuations and slow down convergence. Another option is to perform a so-called *soft update* (Lillicrap et al., 2015), where the target weights are updated after every step as $\overline{\theta}_{t+1} = \tau\overline{\theta}_t + (1-\tau)\theta_{t+1}$, with τ that's usually very close to 1. This gradual update helps the target values change slowly and usually improves stability.

Double DQN van Hasselt et al. (2015) pointed out that the max operator in DQN, which uses the same values to both select and evaluate an action, causes the algorithm to produce overoptimistic value estimates and showed how this hurts performance. To solve the problem, they proposed to use the online network $\hat{q}$ to select the action, while the target network is used to evaluate it. Thus, in Double DQN the update is the same as in standard DQN but the target y_t is estimated as

$$y_t = r_{t+1} + \gamma\hat{q}(s_{t+1}, \arg\max_a \overline{q}(s_{t+1}, a, \theta_t), \overline{\theta}_t) \tag{3}$$

As we can see in Figure 1, this change does indeed reduce harmful overestimation of target values.

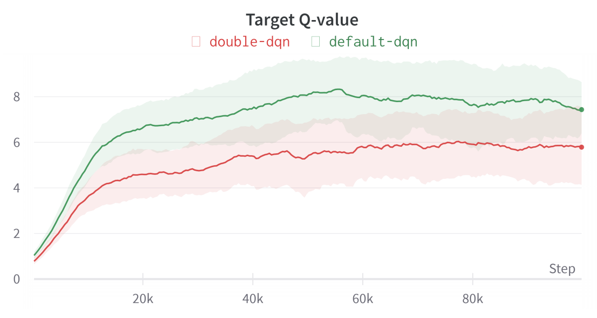

Figure 1: Estimated target y_t, standard DQN vs. double DQN

2.2 Prioritized Replay

In the original DQN algorithm, experiences are uniformly sampled from the replay buffer: this is not ideal as some transitions may be redundant or may not be useful for the agent at the moment – but they may become so when the agent becomes more competent. Thus, Schaul et al. (2015) proposed to replay more frequently the transitions which have an higher TD-error (i.e.: the difference between y_t and the current $\hat{q}(s_t, a_t, \theta_t)$), which is used as a proxy measure for the expected learning progress. As new transitions are added to the replay memory, they are given maximum priority: this makes the algorithm biased towards recent experiences.

Note however that, if we always greedily pick the transitions with higher TD-error, those with low error on first visit may never be replayed. Thus, Schaul et al. (2015) actually propose a stochastic sampling method where the probability of sampling transition i is $P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$, with p_i the priority of transition i and α a parameter that determines the amount of prioritization. Finally, notice that in order to correct the bias we introduce by changing the distribution of the updates, they also propose to add importance sampling weights to the Q-learning update.

2.3 Dueling network

Wang et al. (2015) pointed out that, in many states, some actions are not relevant and thus estimating their value is unnecessary. They thus proposed a novel architecture called *dueling network* which explicitly separates the representations of the action advantages from that of the state values. In practice, the network is composed by a common convolutional feature extractor followed by two different branches composed of dense layers which encode respectively the value $V(s)$ and the advantage $A(a, s)$. The output of the network is given by a special aggregating layer that computes

$$Q(s,a) = V(s) + \left(A(s,a) - \frac{1}{|A|} \sum_{a'} A(s,a') \right)$$

Intuitively, this network can learn which states are not relevant without having to estimate all the action advantages for those states.

2.4 Noisy-Nets and Augmentation

Since heuristics like ϵ-greedy, which perform local perturbations, are unlikely to lead to efficient exploration in many environments, Fortunato et al. (2017) proposed an alternative approach called NoisyNet, where the exploration is driven by some learned perturbations to the network weights. In practice, they inject noise into the linear layers of their architecture, and then select an action simply as $a_t = \arg\max_a \hat{q}(s_t, a, \theta_t)$. The noise is sampled from a certain distribution $\mathcal{D}$, whose parameters (i.e.: mean and standard deviation if $\mathcal{D}$ is Gaussian) are learned along with the standard network weights. They obtained significant performance improvement in many Atari games while reducing the need for hyper-parameter tuning, as the amount of noise is automatically tuned during training.

Augmentation Techniques Because data augmentation techniques have proved to be highly effective in domains like computer vision, Kostrikov et al. (2020) proposed to use standard image transformations in order to perturb the states before they are fed into the network. In particular, they used simple reward-preserving techniques like random shifts and random intensity variations and obtained a significant improvement of performance on the Atari 100k benchmark.

3 Experimental Setup

We trained each agent for 100k steps and ran 5 evaluation episodes every 5k network updates in order to evaluate their performance as the learning progressed. We decided to start by filling up the replay buffer with 20k random transitions, meaning that out agents actually interacted with the environment for a total of 120k times. Each agent was trained 3 times with different random seeds and results were averaged to account for the high variability that characterizes the performance of these algorithms. Once the training phase was concluded, each of the three copies of an agent was evaluated for 125k frames with 5 different starting seeds: results were again averaged and compared with the those of a random agent on the same amount of frames.

Table 1: Model hyper-parameters. Note that not all of them are applicable to every agent: for instance, the "Data augmentation" hyper-parameter is only used for `augmented-dr.q` and `multi-update`.

Parameter	Setting
Data augmentation	Random shifts and intensity
Reward clipping	$[-1, +1]$
Max frames $\times$ episode	108k
Target network update period	1
τ for soft update	0.99
Discount factor	0.99
Optimizer	Adam
Learning rate	0.0000625
Clip gradient norm	10
ϵ annealing steps	50k
ϵ start	1.0
ϵ end	0.01
ϵ eval	0.001
Replay buffer size	100k
β Prioritized replay	0.4
α Prioritized replay	0.6

We trained and tested all the agents we summarized in Section 2, as well as an *extra updates* (Holland et al., 2019) version of out best model, which performs 4 training steps instead of just one for every interaction. The selected hyper-parameters are shown in Table 1: note that, due to computational constraints, we were not able to perform an exhaustive search over the hyper-paramers. We thus used values taken from the literature as much as possible.

All our experiments were conducted using the `ALE/MsPacman-v5` environment on OpenAI Gym (Brockman et al., 2016) without sticky actions and with frame skipping set to four.

4 Results

The algorithms were implemented incrementally following the same order used in Section 2; this means that, for instance, the model we call `dueling-dqn` when exhibiting our results also implements double DQN and prioritized replay. Figure 2 shows the results, in terms of mean episode reward, of

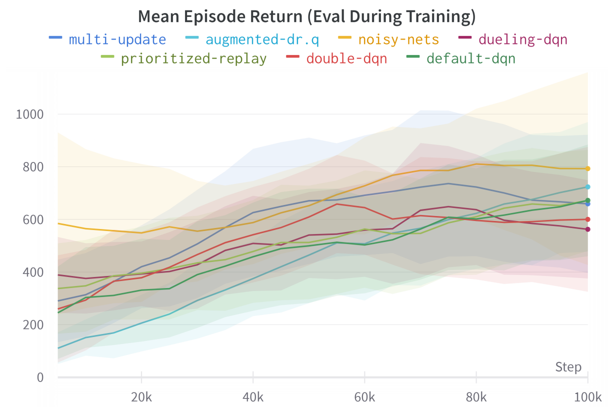

Figure 2: Mean return over 5 evaluation episodes every 5k training steps

our during-training evaluation: notice how the performance of all agents is improving over-time and how the curve for the model `augmented-dr.q` has fewer oscillations than the others, which is an indication that the smoothing effect image augmentation techniques commonly have on DL models may also extend to Reinforcement Learning. Figure 3 depicts the mean episode reward obtained by all our models after 125k evaluation steps; it also includes information about the standard deviation of those results. We can see how, despite the small number of training frames, all models perform considerably better than a random agent (in pink) and how the agent `augmented-dr.q` – which combines all the Section 2 improvements – not only has the highest average score but also has one of the smallest standard deviations. The `noisy-nets` model (i.e.: same as `augmented-dr.q` but without image augmentation) gets comparable results, but its standard deviation is almost three times as large: this implies that image augmentation not only improves sample efficiency but also reduces the variance in the results.

The `multi-update` agent, which is similar to `augmented-dr.q` but performs four training steps per interaction with the environment, is also able to obtain good results. However, tracking its mean episode return during train we notice how its performance seems to degrade after approximately 70k training steps. Analyzing the loss trend, we notice that the curve for the `multi-update` model steadily goes up during the last 30k updates, meaning that its relatively poor behavior could be due to overfitting. Interestingly enough, `double-dqn` alone has almost the same mean episode reward than the `multi-update` model and substantially outperform the `dueling-dqn` algorithm.

Finally, Table 2 reports the human-normalized score of all our agents as well as the CURL, Dr.Q and OTRainbow scores on the Ms.Pacman game (as reported by Kostrikov et al. (2020)) for comparison. We can see how our best score is pretty close to both Dr.Q and OTRainbow's, while CURL's is substantially higher – but then, the agent is also more complex as it uses an auxiliary loss function to help in learning useful representations from from pixels.

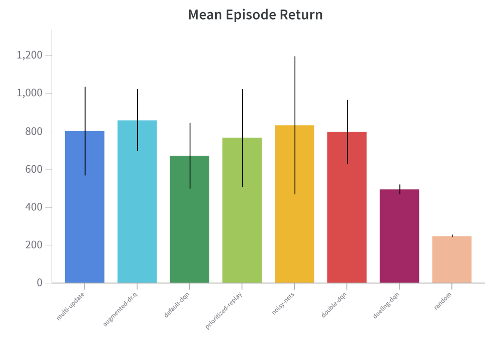

Figure 3: Mean episode return after 125k evaluation frames

Table 2: Mean human-normalized scores, comparison between our models and the literature.

Agent	Score
multi-update	0.083
augmented-dr.q	**0.091**
noisy-nets	0.087
dueling-dqn	0.037
prioritized-replay	0.077
double-dqn	0.082
default-dqn	0.063
Dr.Q	0.098
CURL	**0.178**
OTRainbow	0.095

5 Conclusion and Future Work

These types of algorithms are extremely sensitive to the choice of hyper-parameters, thus a possible future improvement of this work would include further hyper-parameter tuning. Moreover, we saw that the `multi-update` agent was likely overfitting: we thus think that it could be interesting to see whether common DL practices to prevent overfitting like $L_1 \backslash L_2$ regularization, dropout or early stopping could also improve our performances. Moreover, when choosing which image augmentation techniques to employ for their Dr.Q model, Kostrikov et al. (2020) discarded rotations and cutouts after an ablation study performed on the DeepMind contol suite: it could be interesting to see what effect these techniques have on Atari games.

Finally, we think that transforming the frames to grayscale could set the agent at serous disadvantage when playing this game as it becomes exponentially harder to understand that the ghost are turning blue when Ms.Pacman eats an energy pill and that she can only eat them in that precise condition. We thus run the risk of confusing the agent: it sometimes gains a reward when hitting a ghost, while other times it loses a life. Another possible area of improvement would be looking at ways to retain this color information while not increasing too much the number of trainable parameters in the network.

References

G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym, 2016.

M. Fortunato, M. G. Azar, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin, C. Blundell, and S. Legg. Noisy networks for exploration, 2017. URL https://arxiv.org/abs/1706.10295.

M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver. Rainbow: Combining improvements in deep reinforcement learning, 2017. URL https://arxiv.org/abs/1710.02298.

G. Z. Holland, E. J. Talvitie, and M. Bowling. The Effect of Planning Shape on Dyna-style Planning in High-dimensional State Spaces, Mar. 2019. URL http://arxiv.org/abs/1806.01825. arXiv:1806.01825 [cs].

L. Kaiser, M. Babaeizadeh, P. Milos, B. Osinski, R. H. Campbell, K. Czechowski, D. Erhan, C. Finn, P. Kozakowski, S. Levine, A. Mohiuddin, R. Sepassi, G. Tucker, and H. Michalewski. Model-based reinforcement learning for atari, 2019. URL https://arxiv.org/abs/1903.00374.

K. Kielak. Importance of using appropriate baselines for evaluation of data-efficiency in deep reinforcement learning for Atari, Mar. 2020. URL http://arxiv.org/abs/2003.10181. arXiv:2003.10181 [cs, stat].

I. Kostrikov, D. Yarats, and R. Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels, 2020. URL https://arxiv.org/abs/2004.13649.

T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning, 2015. URL https://arxiv.org/abs/1509.02971.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533, Feb. 2015. ISSN 1476-4687. doi: 10.1038/nature14236. URL https://doi.org/10.1038/nature14236.

T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay, 2015. URL https://arxiv.org/abs/1511.05952.

M. Schwarzer, A. Anand, R. Goel, R. D. Hjelm, A. Courville, and P. Bachman. Data-Efficient Reinforcement Learning with Self-Predictive Representations, May 2021. URL http://arxiv.org/abs/2007.05929. arXiv:2007.05929 [cs, stat] version: 4.

A. Srinivas, M. Laskin, and P. Abbeel. CURL: Contrastive Unsupervised Representations for Reinforcement Learning, Sept. 2020. URL http://arxiv.org/abs/2004.04136. arXiv:2004.04136 [cs, stat] version: 4.

H. van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning, 2015. URL https://arxiv.org/abs/1509.06461.

Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas. Dueling network architectures for deep reinforcement learning, 2015. URL https://arxiv.org/abs/1511.06581.

W. Ye, S. Liu, T. Kurutach, P. Abbeel, and Y. Gao. Mastering Atari Games with Limited Data, Dec. 2021. URL http://arxiv.org/abs/2111.00210. arXiv:2111.00210 [cs].