Solving Flappy Bird environment with DQN and its variants

Mattia Bertè
Department of Computer Science
University of Bologna
mattia.berte@studio.unibo.it

Abstract

This project contains a comparison between DQN method and two of its variant, Dueling DQN which introduce the dueling network architecture and DQN plus prioritized experience replay technique. The algorithms were tested on Flappy Bird environment. Repository with code and demo can be found in GitHub [1].

1 Environment

Flappy Bird is a simple game for smartphone, whose goal is keeping the bird alive as most as possible trying to avoid the pipe while it's moving forward. The action space is pretty small, just two actions are possible: tapping the screen or not tapping the screen. When you tap the screen, the bird flaps the wings and moves upward while on the other hand it tends to move downward. There exists two versions of this Gym environment, I used the simpler one. The observation space has just two dimensions:

- horizontal distance between the bird and the next pipe;
- difference between the bird's y position and the next hole's y position.

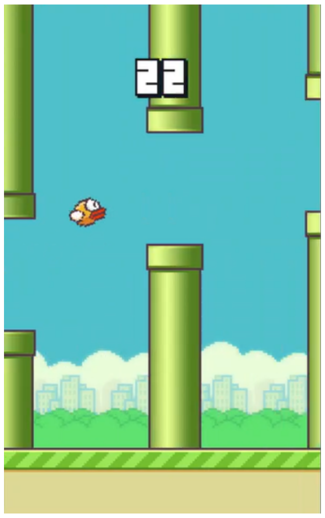

Figure 1: Frame example of Flappy Bird game.

[1]https://github.com/TiaBerte/flappy-bird-dqn

2 Q-learning

Q-learning (Watkins and Dayan, 1992) is a reinforcement learning algorithm whose aim is finding function $\mathcal{Q} : S \times A \rightarrow \mathbb{R}$ that describes the state-action value function. The update scheme is the one in eq. 1.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t(R_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)) \tag{1}$$

Q-learning is an off-policy algorithm since the policy used for interacting with the environment is different from the one used for learning. Indeed DQN, often uses an ϵ-greedy policy for exploring action different from the optimal ones with probability ϵ.
Q-learning in its first basic formulation was able to handle only tabular data. In order to deal with continuous observation space, a parametrized function was required and this problem was solved with the introduction of deep neural network as function approximator. DQN algorithm is indeed just a variation of the classic Q-learning algorithm in which the Q function is parametrized through a neural network. DQN (Mnih et al., 2013) introduced also a fundamental strategy to break the correlation between the training samples which affected the convergence. They added a replay buffer for storing past experience from which transitions are randomly sampled during training.

3 Dueling Network Architecture

(Wang et al., 2016) proposed a variation to the standard network architecture which could be used in both pre-existent reinforcement algorithms and new ones, this variation doesn't affect the structure of the algorithm but just the way in which Q values are computed. The idea is to split the representation of state values V (a scalar value) and action advantages A (a $|A|$-dimensional vector) which are subsequently combined following equation 2.

$$Q(s, a, \theta, \alpha, \beta) = V(s, \theta, \beta) + \left(A(s, a, \theta, \alpha) - \frac{1}{|A|} \sum_{a'} A(s, a', \theta, \alpha) \right) \tag{2}$$

In the initial phase the equation was eq. 3. However after empirical test they noticed that eq. 2 was more stable during the training.

$$Q(s, a, \theta, \alpha, \beta) = V(s, \theta, \beta) + \left(A(s, a, \theta, \alpha) - \max_{a'} A(s, a', \theta, \alpha) \right) \tag{3}$$

They suggested this new approach for specific environment in which in some states the action doesn't affect the environment in useful ways, such as in the Atari game Enduro in which the action of turning is relevant only in presence of other cars. I thought this could be a good strategy for this environment in which the action selection is more relevant in proximity of a pipe.

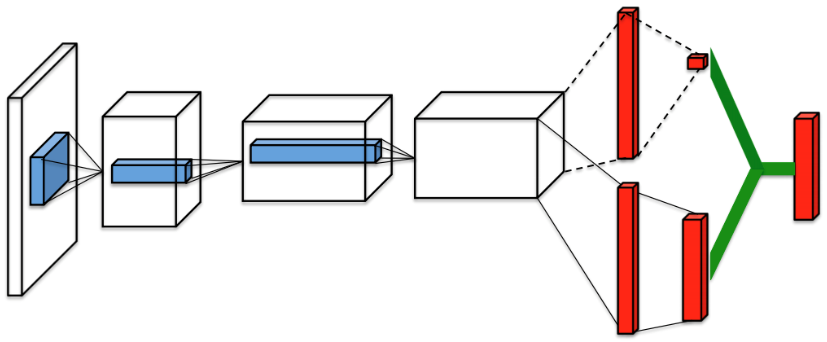

Figure 2: Dueling network architecture. Image from (Wang et al., 2016).

4 Prioritized Experience Replay

(Schaul et al., 2015) proposed a new method for sampling the experience from the buffer. They suggested sampling with a higher probability the transitions from which is possible to learn more. A possible approximator of this value is the temporal difference error which in some reinforcement learning algorithms is already computed for updating the network parameters. However this technique would be very expensive due to the iteration over the whole buffer. To avoid this problem, TD errors are only updated for the transitions that are replayed. The sampling probability is proportional to the so called priority p_i to the power of α.

$$P(i) = \frac{p_i^\alpha}{\sum_k p_i^\alpha} \tag{4}$$

where $p_i = |\delta| + \epsilon$ with δ being TD error and ϵ a small constant to avoid never revisiting states whose error was zero. The exponent α determines how much prioritization is used, with $\alpha = 0$ corresponding to the uniform case.
The prioritized experience introduces bias, indeed prioritizing the transition with an higher error is equivalent to sampling form a distribution different from the one described by the samples in the buffer and it can lead to a solution different from the desired one. It can be corrected using importance-sampling weights (eq. 5). These weights are then used for rescaling the loss. For stability reasons, the weights are normalized with respect to the maximum weight.
The value β is increased till it reaches 1 toward the end.

$$w_i = \left(\frac{1}{N} \cdot \frac{1}{P_i}\right)^\beta \tag{5}$$

5 Experiments

I experimented using the standard DQN algorithm, the one with the dueling architecture and with the application of the prioritized experience replay. I trained the network for 2500 episodes using the hyper-parameters in table 1, which were fine-tuned for the classic DQN. The only variations I introduced was the use of LeakyReLU activation function instead of the ReLU and the use of a soft update instead of the hard one.
The hyper-parameter ϵ related to the ϵ-greedy policy was decreased from 0.1 to 0.01 during the first 90% of the training and then kept it costant.
Regarding the hyper-parameters of the prioritized buffer, I used the one proposed in the paper, $\alpha = 0.6$ and $\beta_0 = 0.4$ which was linearly increased from episode 0 to episode 2250 and then kept equal to 1.

Table 1: Hyper-parameters value

Hyper-parameter	Value
Optimizer	Adam
Learning rate	$5 \cdot 10^{-5}$
Discount (γ)	0.99
Replay buffer size	$1 \cdot 10^5$
Number of hidden layers	2
Number of hidden units per layer	256
Number of samples per mini-batch	256
Non-linearity	LeakyReLU
Target smoothing coefficient (τ)	0.005
Max ϵ	0.1
Min ϵ	0.01
Prioritized Replay α	0.6
Starting prioritized replay β	0.4

During the training, every 10 episodes, the model was tested for 10 episodes and then the rewards were averaged for having a better representation of the model performances. The policy used at evaluation time was a greedy one without possibility to explore (so $\epsilon = 0$).

From the evaluation reward (Figure 3) we can see that prioritized experience replay helped the convergence speed but it lead to a drop in performances. This could be related to the fact that PER was introduce specifically for environment with sparse reward as Cliff Walking environment in which the sequence of action which leads to the success are very rare with respect to the great number of failure.
On the other hand, dueling architecture improved the results with just the cons of requiring an higher number of episodes. It could be related to the fact that this different architecture presents more parameters to train.

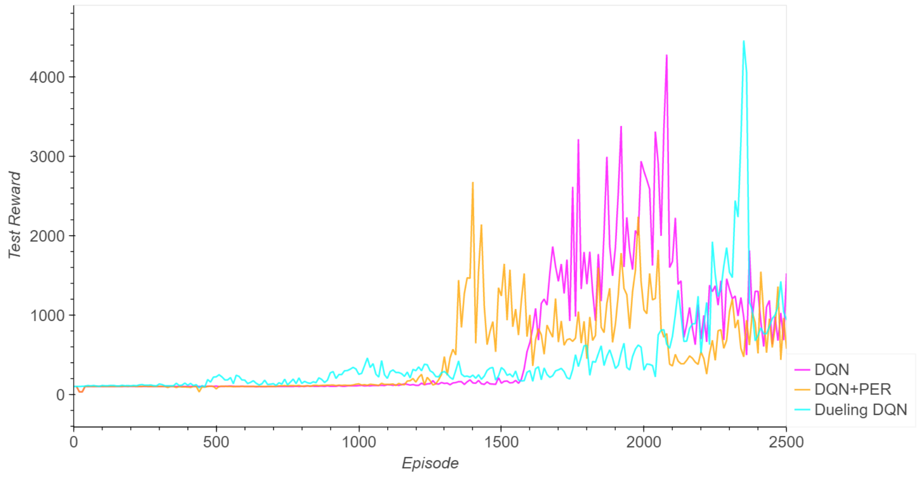

Figure 3: Evaluation reward comparison of the tested algorithms.

Table 2: Best mean reward for each algorithm

Algorithm	Best mean reward
DQN	4280
Dueling DQN	4458
DQN + PER	2677

6 Conclusion

I can conclude that both variations present its pros and cons.
The dueling network architecture seems to be a well suited variations for this environment.
The prioritized buffer replay instead even if helped the convergence does not perform so well in this environment in which the rewards are not sparse.

References

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2015). Prioritized experience replay.

Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M., and De Freitas, N. (2016). Dueling network architectures for deep reinforcement learning. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, page 1995–2003. JMLR.org.

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