The Banana Unity Environment with Deep–Q–Network

Paolo Recchia

1 Introduction to action–value methods

Action–value methods in reinforcement learning aim to find the best action-value function $q_\pi(s,a)$ defined as

Definition. *Action–value function*

$$q_\pi(s,a) = \mathbb{E}_\pi\Bigg[G_t|S_t = s, A_t = a\Bigg] \tag{1}$$

where G_t is the discounted return

$$G_t = \sum_{k=0}^{\infty}\gamma^k R_{t+k}. \tag{2}$$

For a discrete action and state space, the action–value function can be represented as a table. However, in most cases, the action and state space are continuous (as for the banana unity environment, which the state space is continuous and action space is discrete). In addition the shape of the action–value function can be quite complex and a linear approximation may not fit the purpose.

The main idea is to approximate the action–value function with a neural network. In this case the optimization process becomes more straightforward, in fact, the well–known optimization techniques of neural networks can be exploited. This basic idea was implemented with great success in the seminal work of the researchers in Google DeepMind [1]. In this context of Banana Unity Environment we basically follow the same principle.

2 The algorithm

At each iteration, the aim is to find the best set of parameters of the action–value function (namely the Q–network) which minimizes the following loss function

$$\mathcal{L}(\theta) = \Bigg[R + \gamma \max_{a'} Q(s', a', \theta) - Q(s, a, \theta)\Bigg]^2. \tag{3}$$

However two key optimizations are possible.

Experience Replay Because some states–actions pair come less likely than others it's better to collect some experiences (namely a tuple of $e_t = (s_t, a_t, r_{t+1}, s_{t+1})$) in a buffer memory and then, in order to avoid the high correlation among consecutive experiences in this buffer memory, we can take randomly a batch of these observations

Fixed Target As we can see in (3) both the target network and the loss function depend on the same set of parameters θ. To make a reliable learning process the parameters of target Q–network and the parameters θ to learn have to be decoupled. We can then fix the parameters of the target Q–network to a set θ^- and update it after each C step.

After these optimizations the loss function reads as

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s')\sim U(D)} \left[R + \gamma \max_{a'} Q(s', a', \theta^-) - Q(s, a, \theta) \right]^2 . \tag{4}$$

The symbol $\mathbb{E}_{(s,a,r,s')\sim U(D)}$ indicates the expected value calculated on a batch of experiences (s, a, r', s') uniformly sampled from the buffer memory D.

As in [1] the algorithm to train the agent in Banana Unity Environment reads as

Algorithm 1: DQN algorithm with Experience Replay and Fixed Target [1]

```
Initialize buffer memory $D$ to capacity $N$
Initialize Q–network with random weights $\theta$
Initialize target Q$_{target}$ network with weights $\theta^- = \theta$
for episode $i = 1$ to $M$ do
    Reset Banana Unity Environment and set the input state of the first
      step $s_1$
    for time–step $t = 1$ to $T$ do
        $a_t = \varepsilon$–greedy action selection
        The agent execute action $a_t$ and record $r_{t+1}$ and next state $s_{t+1}$
        Store the experience $(s_t, a_t, r_{t+1}, s_{t+1})$ in the buffer memory $D$
        if Every $C$ steps then
            Sample random mini–batch of experiences from $D$
            Evaluate the loss function $\mathcal{L}(\theta)$
            Perform the minimization of the loss function $\mathcal{L}(\theta)$ on the
              network parameters $\theta$
            Reset $Q_{target} = Q$
        end
        Set $s_t = s_{t+1}$
    end
end
```

3 The Banana Unity Environments

The aim of the Banana Unity game is to collect as many yellow banana as possible avoiding the blue ones.

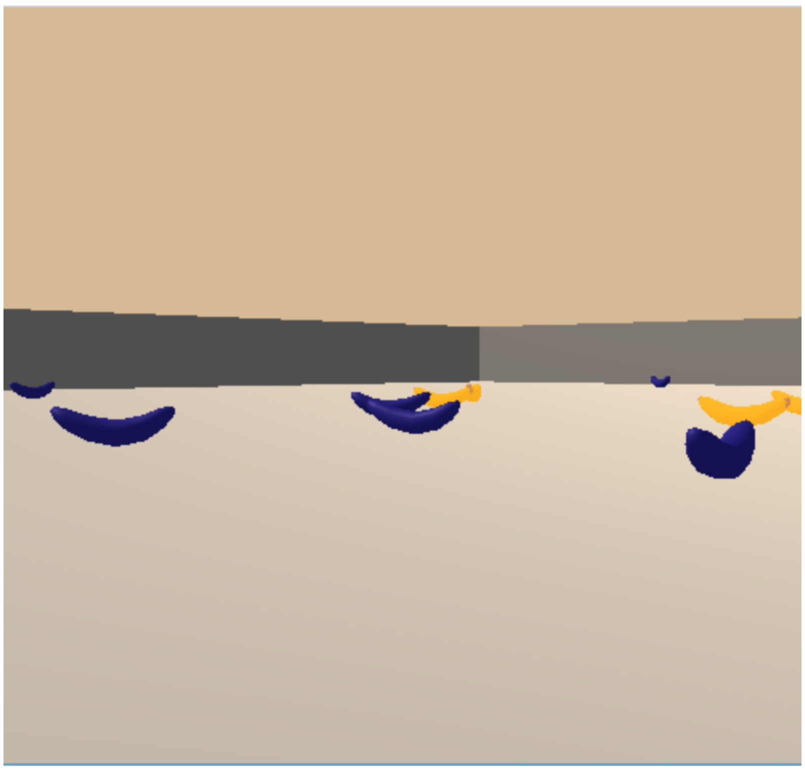
Figure 1: Banana Unity game

The state space has 37 different features as the agent's velocity, along with ray–based perception of objects around agent's forward direction, etc. Each of these features can be a value in a continuous range. The action space is discrete instead, in fact, the agent can take only four possible actions (walk forward, walk backward, turn right, turn left).

A reward of +1 is provided for collecting a yellow banana, and a reward of -1 is provided for collecting a blue banana. The agent will win the game if it gets an average reward of 13 for 100 episodes.

4 The Q–network and the outcome

For the Q–network the following hyper–parameters have been chosen

Discount factor γ $= 0.99$

Number of max episode $= 2000$

Number of max time steps per episode $= 1000$

Buffer memory size N $= 1 \times 10^5$

Batch size for random sampling $= 64$

Input layer of Q–network $= 37$ (the state space size)

Hidden layers: 2 hidden fully connected layers with 64 ReLU nodes

Output layer $= 4$ (the number of actions)

Kind of optimizer: Adam (adaptive moment estimation)

Learning rate of Adam optimizer $= 5 \times 10^{-4}$

Step for updating the target Q–network $C = 4$

As we can see in figure 2, by the aforementioned hyper–parameters, the agent solved the game in less than 700 episode.

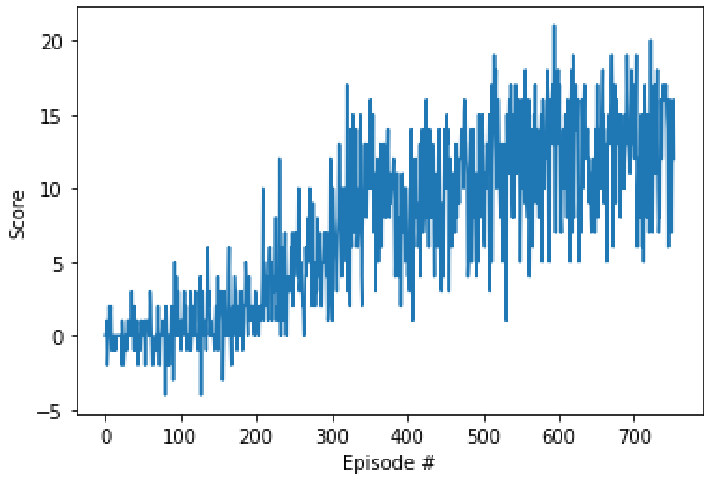

Figure 2: In this test the agent solved the environment in 653 episodes with an average score of 13.01.

5 Insights for future work

The following improvements to the Deep–Q–network can be implemented in future works

Double Q–learning: The vanilla DQN as presented above and in the original paper could lead to an overestimation of the action–value function. In fact, especially in the early step, the learning process is still evolving, then taking the action which maximizes the Q function could be not a 'good' choice. Double Q–learning [2] fixed this issue by decoupling the parameters used to select the best action and the parameters to evaluate the corresponding Q function. Namely this means to reformulate the target in the loss function (3) as

$$R + \gamma Q\left(s', \underset{a'}{\text{argmax}} Q(s', a', \theta), \theta^{-}\right) \tag{5}$$

where θ are the same parameters of the online network and θ^{-} are the fixed ones in our fixed target DQN.

Prioritized experience replay For the learning process some expereinces stored in the memory buffer are more important than others; taking them randomly doesn't take into account the 'weight' of each experience. In addition, since the memory buffer is practically limited in capacity, older and more important experiences can be lost. A way to prioritize the experiences was firstly proposed in [3]. The idea is to prioritize the experiences which lead to a bigger TD–error during the learning, so at each experience we can assign a value of priority

$$p_t = R_{t+1} + \gamma Q(S_{t+1}, a, \theta) - Q(S_t, A_t, \theta). \tag{6}$$

However some experiences could have a priority almost equal to zero though we can still learn something from them, so we can add some noise ε to the value of the priorities too

$$p_t = R_{t+1} + \gamma Q(S_{t+1}, a, \theta) - Q(S_t, A_t, \theta) + \varepsilon. \tag{7}$$

At each experience in the buffer memory we can assign its related priority. Instead to sample uniformly from a batch of experiences, we can then sample by the following probability

$$P(i) = \frac{p_i}{\sum_k p_k}. \tag{8}$$

It worth to underline that some experiences with higher priorities can be replayed many times. In order to avoid the overfitting of this subset of experiences it is helpful to add some randomness on the probabilities (8)

$$P(i) = \frac{p_i^a}{\sum_k p_k^a} \tag{9}$$

if $a = 0$ the experiences won't be prioritized at all (they will be picked uniformly), if $a = 1$ the experiences will be completely prioritized as in (8).

References

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. nature, 518(7540):529–533, 2015.

[2] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In Proceedings of the AAAI conference on artificial intelligence, volume 30, 2016.

[3] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. arXiv preprint arXiv:1511.05952, 2015.