Project1: DQN for the banana environment

UDACITY

Contents

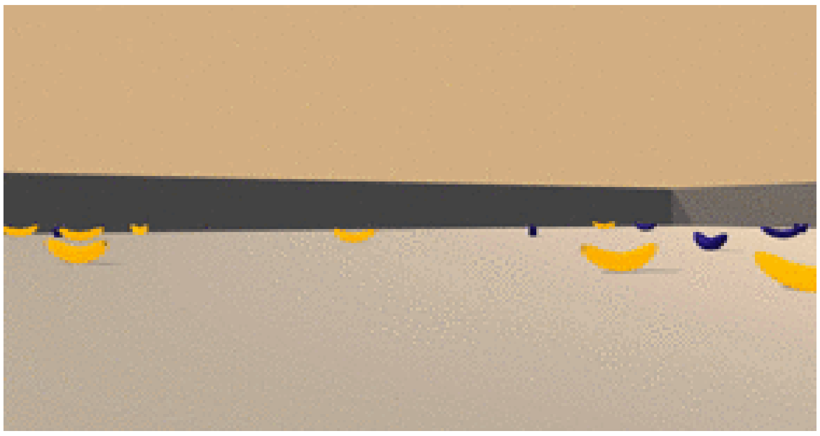


Figure 1: Screenshot of the Udacity banana environment

1 Environment description

The environment is a Unity environment provided by Udacity. It is similar to the Unity ML agents Banana Collector environment, but not identical.
The task is to navigate an agent in a square world. In the environment there are yellow and blue bananas. The goal is to to collect as many yellow bananas as possible, while preventing to collect blue bananas. A screenshot of the environment is shown in figure 1.

Statespace and dynamics

The state space has 37 dimensions $\mathcal{S} \subset \mathbb{R}^{37}$ and contains the agent's velocity in x and y direction, along with 7 ray-based perception of dimension 5 of objects around the agent's forward direction. The five dimensional ray-based perception can be interpreted as follow

$$[\mathbb{I}(\text{Yellow Banana}), , \mathbb{I}(\text{Wall}), \mathbb{I}(\text{Blue Banana}), \mathbb{I}(\text{Agent}), d], \tag{1}$$

whereby d is the distance of the object and $\mathbb{I}(.)$ is the indicator function. The environment dynamic (transition function) is unknown. As there are no other agents the environment can be considered stationary.

Actionspace

The action space of the agent is discrete and consists of the following 4 actions:

$$\mathcal{A} = \{\text{move left}, \text{move right}, \text{move up}, \text{move down}\} \tag{2}$$

Reward and solution criteria

The reward structure of the environment is as follows: 1 collecting a yellow banana, -1 collecting a blue banana. The environment is considered as solved when the agent gets an average score of +13 over 100 consecutive episodes.

2 DQN Algorithm description

Deep Q-learning is an algorithms that takes the value-based Q-learning algorithm (SARSA-max) and combines it with deep neural networks to leverage the power of generalisation. The algorithm was first brough up to play Atari games [Mni+13].
The update equation of Q-learning is given as

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[r_t + \max_a Q(S_t, A_t) \right]. \tag{3}$$

The data needed to run one update step is (s_t, a_t, r_t, s_{t+1}). In contrast to SARSA, the action a_{t+1} is not needed. As the update equation directly corresponds to the Bellman equation it is an off-policy algorithm. Although it was shown that going too far off-policy, can destabilize training. Therefore, as behaviour policy, an epislon greedy policy is used

$$\pi(a|s) = \begin{cases} \max_a Q(s,a) & \text{probability } (1-\epsilon), \\ \text{random } a & \text{probability } \epsilon \end{cases} \tag{4}$$

To favor exploration at the beginning and exploitation over time, ϵ can be decreased as training progresses. In DQN the Q function is approximated using a parametrized neural network $Q_\theta(s, a)$. To make the learning of this network work stable, at least the following extensions are recommended:

- **Target Q-network**: To avoid that the network chasing moving targets - two neural networks are used for learning the Q function. The so called target network, has its weights only updated very slowly and is used to generate the targets for the network that is actually learning.

$$L(\theta, \mathcal{D}) = \sum_i (y_i - Q_\theta(s_i, a))^2, \tag{5}$$

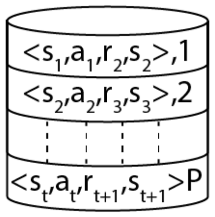
$$y_i = r_i + \max_a Q_\Theta(s_i^{'}, a) \tag{6}$$

$$\text{update } \theta \text{ based on } \nabla_\theta L(\theta, \mathcal{D}) \tag{7}$$

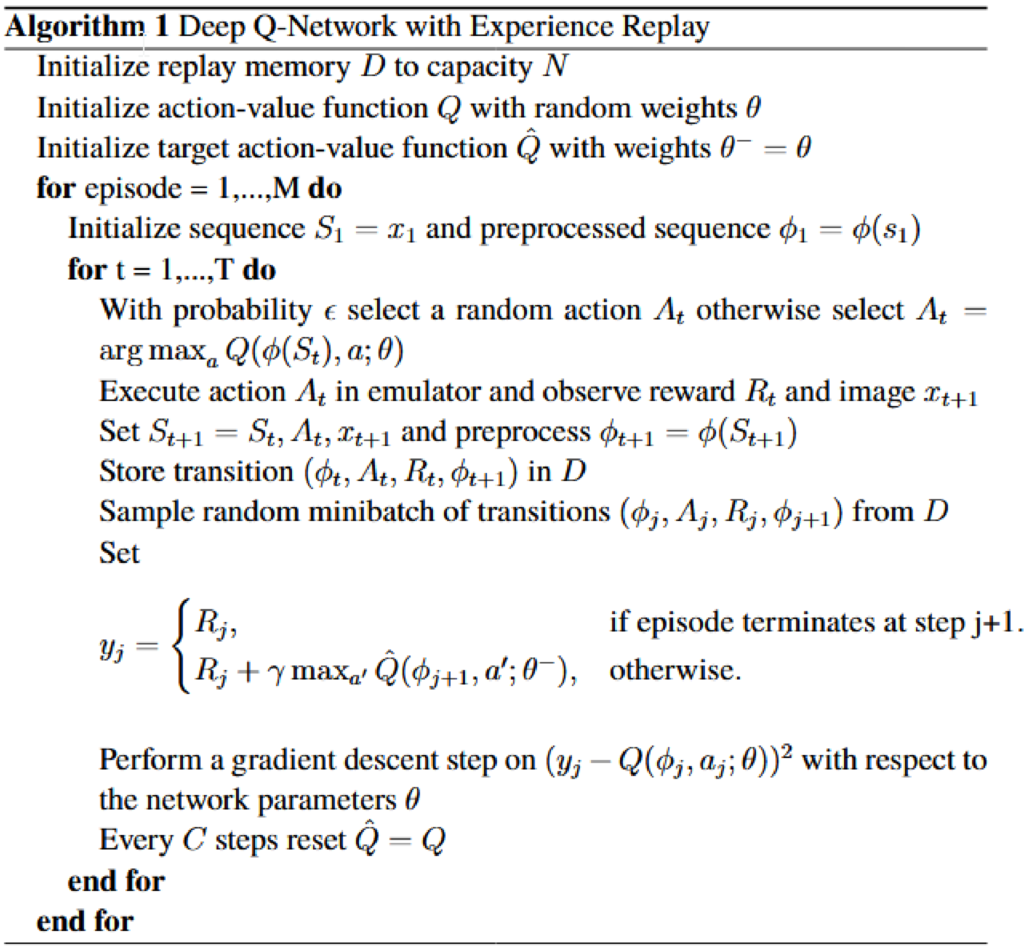
  The weights of the target network are updated very slowly

$$\Theta \leftarrow \tau\Theta + (1-\tau)\theta \tag{8}$$

- **Experience replay buffer**: Set of (s, a, s', r, d) $\mathcal{D}$ previous experiences. For stable behaviour, the replay buffer should be large enough to contain a wide range of exeriences. But is should also only contain most recent data as too much experience can slow down learning.



The full aglorithm is shown in figure 2.



```
**Algorithm 1** Deep Q-Network with Experience Replay
  Initialize replay memory $D$ to capacity $N$
  Initialize action-value function $Q$ with random weights $\theta$
  Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
  for episode = 1,...,M do
    Initialize sequence $S_1 = x_1$ and preprocessed sequence $\phi_1 = \phi(s_1)$
    for t = 1,...,T do
      With probability $\epsilon$ select a random action $A_t$ otherwise select $A_t = \arg\max_a Q(\phi(S_t), a; \theta)$
      Execute action $A_t$ in emulator and observe reward $R_t$ and image $x_{t+1}$
      Set $S_{t+1} = S_t, A_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(S_{t+1})$
      Store transition $(\phi_t, A_t, R_t, \phi_{t+1})$ in $D$
      Sample random minibatch of transitions $(\phi_j, A_j, R_j, \phi_{j+1})$ from $D$
      Set
        $y_j = \begin{cases} R_j, & \text{if episode terminates at step j+1.} \\ R_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-), & \text{otherwise.} \end{cases}$
      Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters $\theta$
      Every $C$ steps reset $\hat{Q} = Q$
    end for
  end for
```

Figure 2: Summary of DQN algorithm

3 Implementation details

As a baseline implementation the code from the udacity ”LunarLander-v2” environment DQN agent was used. Some adjustments were made to cope with the banana environment. The action-value function approximation network architecture is shown in figure 4.

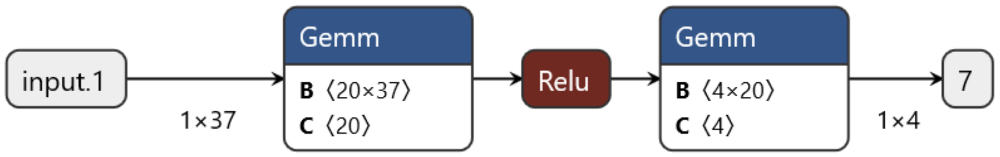


Figure 3: Q-function network architecture. Feed forward network with one hidden layer and ReLU activation. Visualized with: netron.app.

The following hyper-parameters were used for training:

```
BUFFER_SIZE = int(1e5)  # replay buffer size
BATCH_SIZE = 64         # minibatch size
GAMMA = 0.99            # discount factor
TAU = 1e-3              # for soft update of target parameters
LR = 5e-4               # learning rate
UPDATE_EVERY = 4        # how often to update the network
```

4 Training and Result

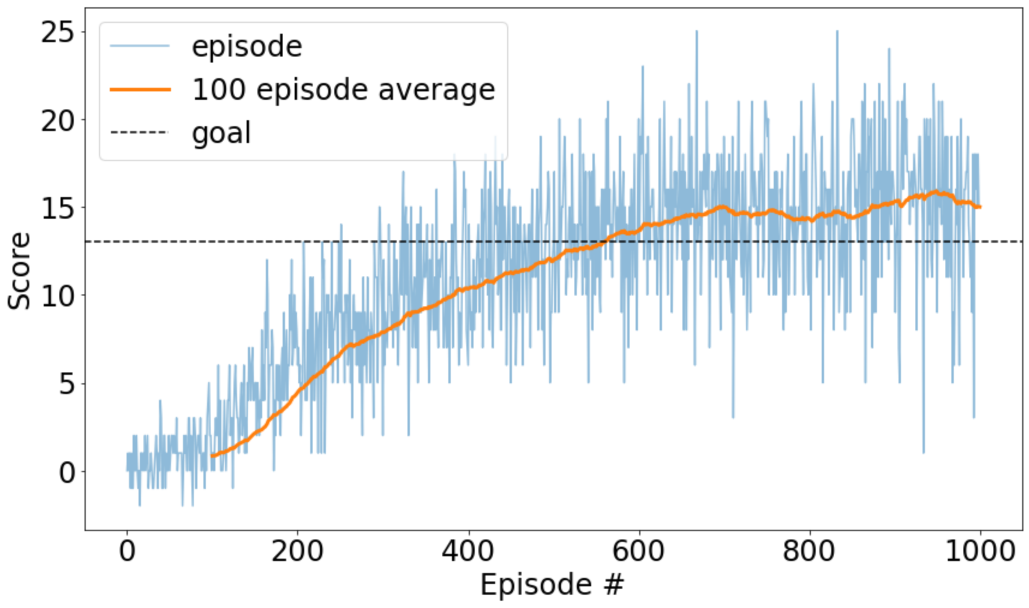


Figure 4: Q-function network architecture. Feed forward network with one hidden layer and ReLU activation. Visualized with: netron.app.

5 Summary and ideas for future work

The training of an agent was pretty straight forward using the baseline algorithm provided in the coding exercise. Just some slight adjustment to the network and the hyperparameters were needed. It would be interesting to further tune the hyperparameters or to add improvements from the literature to the DQN algorithm like:

- Prioritized experience replay [Sch+16]
- Double Q-Learning ([TS], [vHGS15]
- Dueling DQN [Wan+16]
- Distributional DQN [BDM17]
- Noisy DQN [For+19]
- Learning from multi-step bootstrap targets [Mni+16]
- Multiple improvements in combination (Rainbow) [Hes+17]

References

[Mni+13] Volodymyr Mnih et al. *Playing Atari with Deep Reinforcement Learning.* Version 1. Dec. 19, 2013. DOI: 10.48550/arXiv.1312.5602. arXiv: 1312.5602 [cs]. URL: http://arxiv.org/abs/1312.5602 (visited on 12/08/2023). preprint.

[vHGS15] Hado van Hasselt, Arthur Guez, and David Silver. *Deep Reinforcement Learning with Double Q-learning.* Dec. 8, 2015. DOI: 10.48550/arXiv.1509.06461. arXiv: 1509.06461 [cs]. URL: http://arxiv.org/abs/1509.06461 (visited on 12/08/2023). preprint.

[Mni+16] Volodymyr Mnih et al. *Asynchronous Methods for Deep Reinforcement Learning.* June 16, 2016. DOI: 10.48550/arXiv.1602.01783. arXiv: 1602.01783 [cs]. URL: http://arxiv.org/abs/1602.01783 (visited on 12/08/2023). preprint.

[Sch+16] Tom Schaul et al. *Prioritized Experience Replay.* Feb. 25, 2016. DOI: 10.48550/arXiv.1511.05952. arXiv: 1511.05952 [cs]. URL: http://arxiv.org/abs/1511.05952 (visited on 12/08/2023). preprint.

[Wan+16] Ziyu Wang et al. *Dueling Network Architectures for Deep Reinforcement Learning.* Apr. 5, 2016. DOI: 10.48550/arXiv.1511.06581. arXiv: 1511.06581 [cs]. URL: http://arxiv.org/abs/1511.06581 (visited on 12/08/2023). preprint.

[BDM17] Marc G. Bellemare, Will Dabney, and Rémi Munos. *A Distributional Perspective on Reinforcement Learning.* July 21, 2017. DOI: 10.48550/arXiv.1707.06887. arXiv: 1707.06887 [cs, stat]. URL: http://arxiv.org/abs/1707.06887 (visited on 12/08/2023). preprint.

[Hes+17] Matteo Hessel et al. *Rainbow: Combining Improvements in Deep Reinforcement Learning.* Oct. 6, 2017. DOI: 10.48550/arXiv.1710.02298. arXiv: 1710.02298 [cs]. URL: http://arxiv.org/abs/1710.02298 (visited on 12/08/2023). preprint.

[For+19] Meire Fortunato et al. *Noisy Networks for Exploration.* July 9, 2019. DOI: 10.48550/arXiv.1706.10295. arXiv: 1706.10295 [cs, stat]. URL: http://arxiv.org/abs/1706.10295 (visited on 12/08/2023). preprint.

[TS] Sebastian Thrun and Anton Schwartz. “Issues in Using Function Approximation for Reinforcement Learning”. In: ().