Using Experience Replay and Target Networks for Stable Deep Reinforcement Learning

Lennert Jansen, Daniël Nobbe, Jorrim Prins & Yke Rusticus
10488952, 12891517, 11038934 & 11306386
Reinforcement Learning - Reproducibility Project

1 Introduction

In recent years, neural networks have experienced a dramatic rise in popularity, mostly due to the year-over-year increase in available computing power. Its emergence has also induced a significant increase in the use of neural networks in Reinforcement Learning (RL). We examine the Deep Q-Network (DQN) by Mnih et al. [2015], which implements Q-learning [Watkins and Dayan, 1992] with function approximation by a neural network. Divergence of this model can often be observed and we examine two tricks, *experience replay* and *target network*, that Mnih et al. [2015] use to prevent this problem. We perform convergence experiments on OpenAI Gym's Acrobot and CartPole environments [Brockman et al., 2016], trying to answer the following research question: Can experience replay and a target network help prevent divergence for Deep Q-Networks, and if so, to what extent?

Divergence is expected to be caused by the combination of bootstrapping, off-policy learning and function approximation, Sutton and Barto [2018] call this the *deadly triad*. Bootstrapping happens when temporal-difference (TD) learning is used, a method for learning the value $Q(s, a)$ of a certain (state, action) pair. It bootstraps the current value on the value of the next state, as these values should be close. To optimise the exploration/exploitation trade-off, Q-learning is introduced by Watkins and Dayan [1992] as the off-policy variant of TD learning. True action-values $Q(s, a)$ can be approximated with a function, which is beneficial when state or action spaces are high-dimensional or continuous and a tabular approach – storing the value for each (s, a)-pair – is infeasible. Mnih et al. [2015] introduce the DQN as a Q-learning model with its Q-function approximated by a neural network, and train this model to play Atari games. DQN incorporates all three components of the aforementioned deadly triad and can therefore present divergence problems.

Theoretical explanation of the problem is concentrated around linear examples [Baird, 1995, Tsitsiklis and Van Roy, 1996] and non-linear settings have only been explored in experimental settings [Van Hasselt et al., 2018, Achiam et al., 2019]. Consequently, there are not many theoretical guarantees on convergence of DQN (with or without its additional tricks), but it manages to converge to good solutions in previous experiments. We will focus on the effect of experience replay and target networks on convergence of DQN by experimenting on the environments specified above. Having elaborated on the cause of divergence, appropriately called the deadly triad, the following sections elaborate on the techniques behind DQN, prepare experiments to explore convergence properties and analyse the results provided by those experiments.

2 Methods & Approach

We consider RL tasks where an agent interacts with an environment and sequentially observes states where it performs an action, to receive a corresponding reward. The agent aims to develop a policy for choosing actions, such that the cumulative future reward is maximised. This cumulative future reward at time-step t is defined as $R_t = \sum_{k=0}^{T} \left(\gamma^k r_{t+k+1}\right)$, with γ the discount factor and T the terminal time-step. We use Q-learning with neural network approximation (DQN [Mnih et al., 2015]) to approximate the optimal action-value function:

$$Q^*(s, a) = \max_\pi \mathbb{E}[R_t \mid s_t = s, a_t = a, \pi] \tag{1}$$

Q^* is the expected maximum sum of discounted rewards r_t, given a behaviour policy $\pi(a \mid s)$ for state s and action a. The behaviour policy in our experiments is defined as an ϵ-greedy policy with an annealed ϵ. The intuition behind this process is based on the *Bellman equation* [Sutton and Barto, 2018]. The Bellman equation essentially says that the future reward maximising strategy chooses an action that maximises the expectation of current reward and discounted

future action-value pairs, given knowledge of the optimal action-value function for the next state and action (s' and a'). The equation gives rise to a formal updating scheme for Q-learning that should theoretically converge towards the optimal Q^*: $Q^*_{i+1}(s,a) = \mathbb{E}_{s'}[r + \gamma \max_{a'} Q^*_i(s',a')]$. In practice, however, this quickly becomes infeasible for high-dimensional tabular and non-tabular settings, so function approximators are introduced to achieve generalisation.

In our case, we work with a non-linear approximation of the optimal Q-function, $Q(s,a;\theta) \approx Q^*(s,a)$, specifically by using a neural network with parameters θ. The set of environments we consider for our experiments is smaller and made up of simpler tasks than that of Mnih et al. [2015], so we implement a less complex model than the original authors' convolutional neural network. The Q-Network is trained by updating its parameters θ according to the Bellman equation, resulting in the following loss function to be minimised, with θ_i^- the parameters from the preceding iteration:

$$L_i(\theta_i) = \mathbb{E}_{s,a,r,s'}\left[\left(r + \gamma \max_{a'} Q(s',a';\theta_i^-) - Q(s,a;\theta_i)\right)^2\right]. \tag{2}$$

2.1 Experience replay & target network

As mentioned before, the DQN process we have described above suffers from all components of the deadly triad and is known to be sensitive to divergence. This limitation is addressed by Mnih et al. [2015], equipping DQN with two extensions to regular Q-learning. The first modification, experience replay, helps improve stability by breaking the strong correlation between successive observations within an episode, thereby reducing the variance of the updates. It involves saving an experience tuple $e_t = (s_t, a_t, r_t, s_{t+1})$ from every time-step in a dataset $D_t = (e_1, ..., e_t)$. Uniformly drawn experience samples from D are then used for learning. A limitation of experience replay, as proposed by Mnih et al. [2015], is that due to memory restrictions, the dataset of stored experiences must overwrite older transitions for newer ones, without distinguishing between the importance of transitions. Similarly, uniform sampling doesn't prioritise transitions from which we can learn more.

The second extension involves training a separate Q-network from which we generate targets y, the so-called target network. The primary Q-network is copied every C updates to produce the target network $\hat{Q}$ and the following C Q-learning updates will be performed using target values from $\hat{Q}$, after which the process is repeated. Regular Q-learning updates that increase $Q(s_t, a_t)$ tend to increase $Q(s_{t+1}, a)$, thereby increasing the target y and possibly leading to instability or divergence of network weights. Implementation of a target network stabilises learning by lowering the frequency of target updates, slowing down the effect of target updates on Q-updates. For an overview of the described process, see Algorithm 1.

3 Experimental Setup

The experiments we present will focus on the two extensions as presented in section 2. The baseline model of our research will consist of a basic DQN, excluding the extensions we have discussed, with an underlying neural network of one 128-node hidden layer and a ReLU activation function between the hidden layers. Minimising the loss function (see Equation 2) is done with the Adam optimiser [Kingma and Ba, 2014]. This baseline model is extended to accommodate either experience replay or a target network structure, we identify these extensions as the DQN-ER and DQN-TN models respectively. Of course, we want to ultimately find the solution to our divergence problem by implementing experience replay and target network simultaneously, and the implemented model that uses both extensions will be identified as Robust-DQN. The Robust-DQN will, in an optimal situation, be robust to divergence problems. All experiments are evaluated on the duration, the average loss function value and the maximum absolute Q-value of each episode. We will repeat each experiment 10 times with different random seeds to obtain statistically sound results.

3.1 Parameter variation

Experience replay and target network both contain a hyperparameter setting of which the effect can be examined. Two hyperparameters arise from the implementation of experience replay: the size of experience dataset M (*memory capacity*) and the number of randomly picked learning instances (*batch size*). Extension by using a target network also involves setting an extra parameter, the *clone interval* denoted by C. The clone interval defines after how many steps the parameters of the target network are updated. We select appropriate parameters for both the experience replay and target network and compare the resulting performance against the performance for disabled experience replay and/or target network.

In addition to these process-specific parameters that can help provide insight into the working of the two tricks, more general parameters could also have enormous effects on model behaviour. Convergence problems can obviously be correlated to parameter settings regarding the learning rate, discount factor and underlying network size. Some variation in these parameters are explored to gain additional insight into convergence behaviour of the DQN.

3.2 Reward clipping

Different environments use different scales for rewards, Mnih et al. [2015] therefore introduces *reward clipping* to limit the scale of error derivatives and to be able to use equal learning rates for multiple environments. An additional benefit of reward clipping, as demonstrated by Van Hasselt et al. [2018], is that we can set boundaries on realistic estimated Q-values. More specifically, if rewards are clipped within [-1,1] and we use a discount factor γ, every estimated Q-value above $\lim_{T\to\infty}\left(1+\gamma+\gamma^2+\ldots+\gamma^T\right)=\frac{1}{1-\gamma}$ is unrealistic. We use reward clipping for all experiments to distinguish realistic from unrealistic Q-values, providing us with another indicator of divergence.

3.3 Libraries and packages

We perform experiments on the Acrobot-v1 and CartPole-v1 environments from the OpenAI Gym library [Brockman et al., 2016]. This is a widely used library for testing and developing RL agents and includes several other ready-made RL environments, including Atari games. It also includes standardised methods for interacting with those environments. The neural networks are implemented using PyTorch [Paszke et al., 2017], providing the availability to specify networks concisely and use them effectively.

4 Results

4.1 CartPole-v1

In the CartPole-v1 environment, we find specific hyperparameter settings for which the network always converges, regardless of the use of the aforementioned tricks. These settings are given in Table 1, where batch size, memory capacity and clone interval are not defined yet. Since we are interested in divergence, we also test a setting where we find experience replay and a target network to be necessary for convergence, by only changing the discount factor. In each experiment, the batch size is set to $b=\min\{64,M\}$, where M is the memory capacity of experience replay.

	CartPole-v1 (always converges)	**CartPole-v1** (diverges without tricks)	**Acrobot-v1** (diverges)
No. episodes	500	500	100
Discount factor	0.8	0.95	0.999
Learning rate	0.001	0.001	0.001
Min. ϵ	0.05	0.05	0.05
Max. ϵ	1.0	1.0	1.0
Annealing time	1000	1000	1000

Table 1: Hyperparameter settings for training of DQN for the CartPole-v1 and Acrobot-v1 environments. The left-most setting for CartPole-v1 shows convergence with and without tricks, the second only with tricks. The settings for Acrobot-v1 show divergence with and without tricks.

In CartPole-v1, the goal is to take as many action steps as possible before the game terminates. To compare the effect of the target network on the performance in the CartPole-v1 environment, we plot performances for various clone intervals C in Figure 1, both with and without experience replay. In the case where $C=1$, the target network is updated every step, which is equivalent to not using a target network. $\max|Q|$ represents the maximal absolute Q-value per batch averaged over the number of batches in one episode of training. We find that for larger C, the realistic $\max|Q|$ limit is reached more gradually without overshooting, and the loss values are less extreme at the start. However, if C becomes too high (e.g. $C>500$ for Robust-DQN), the model will stop learning.

In Figure 2, results are shown for a setting in which we can identify divergence. We find that using no tricks, or applying experience replay only, results in estimated Q-values diverging from the realistic limit of 5 (corresponding to $\gamma=0.95$). The loss for these settings also seems to only grow over time. One may never know for sure whether this behaviour will continue after a number of extra episodes. However, note that these values are plotted on logarithmic scale, so the curves are actually increasing near-exponentially. Enabling the target network without experience replay does not diverge in 500 episodes, although it also does not converge yet. The Robust-DQN setting shows convergence in the loss and $\max|Q|$ values, and provides a good solution to the problem, as can be seen in in the average number of durations after some training.

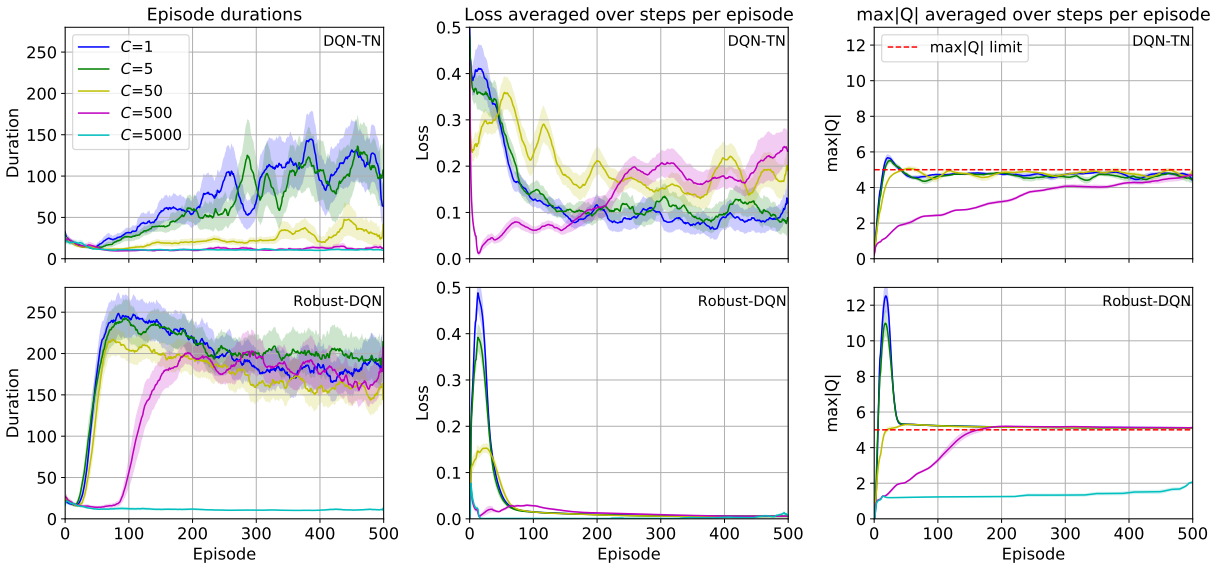

Figure 1: Comparison of update intervals C for CartPole-v1, where $\gamma = 0.8$ and memory size $M = 50$k. None of the settings seem to clearly diverge. All estimated (absolute) Q-values seem to converge below or near the maximum realistic value of 5. The curves are smoothed over 20 consecutive values and filled regions represent $\sqrt{(\frac{Var(Y)}{N})}$, where Y is the relevant value and $N = 10$ is the number of runs. The curves are the means of the independent runs.

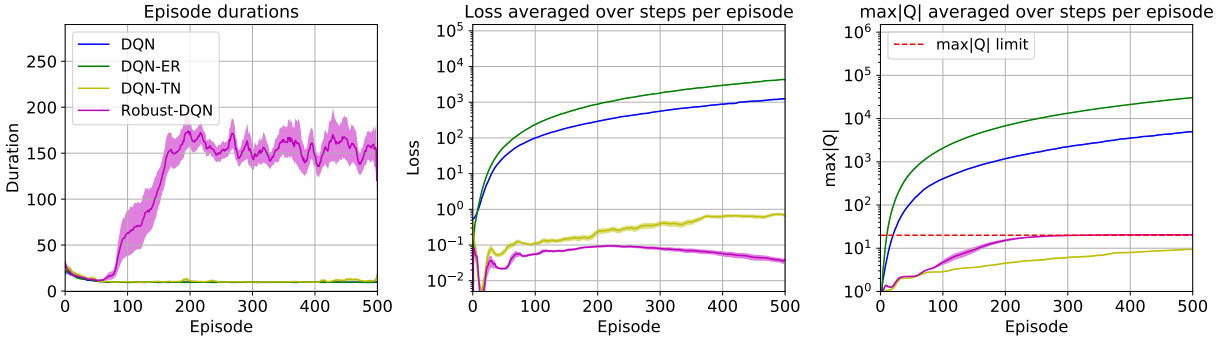

Figure 2: A setting where experience replay and the use of a target network individually make little difference, but together result in stable learning in the CartPole-v1 environment, with $\gamma = 0.95$. Update interval C was chosen to be 500 in case a target network was used, and memory size for experience replay is $M = 50$k . The dashed line indicates the limit of realistic maximum absolute Q values. The curves are smoothed over 10 consecutive values and filled regions represent $\sqrt{(\frac{Var(Y)}{N})}$, where Y is the relevant value and $N = 10$ is the number of runs.

4.2 Acrobot-v1

In the Acrobot-v1 environment, we find general hyperparameter settings for which the Robust-DQN gets good results, but shows some divergence in the $\max |Q|$ values. Note that in this environment, shorter episode lengths are better. The maximum number of steps in an episode is 500, so the maximum realistic absolute Q-value is also 500.

By doing an informal search for the parameters for the tricks, we find an optimal memory capacity of 10,000 and batch size of 64. The optimal target network clone interval is 500.

See Figure 3 for the performance under enabling and disabling of the tricks. Notable results are the bad performance of the vanilla network and the divergence of the models with a target network, which can be seen in the $\max |Q|$ values, and . Based on the rightmost graph, we expect the $\max |Q|$ values of the Robust-DQN to eventually converge, although this is not guaranteed. In terms of performance, the Robust-DQN gives the best and most stable performance. Both the vanilla DQN and the DQN-ER show good initial performance that drops off for more training episodes, which is a sign

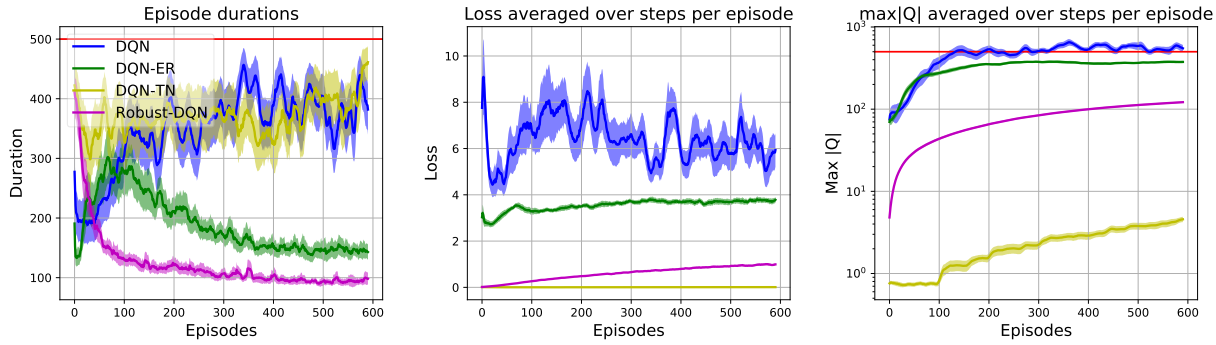

Figure 3: Performance for the four network settings in the Acrobot-v1 environment. All three plots are smoothed over 10 steps, and the transparent fill around each line indicates the value of $\sqrt{(\frac{Var(Y)}{N})}$, where Y is the relevant value and N is the number of runs (10). The realistic maximum duration and $\max|Q|$ value are indicated with red lines.

of divergence. The divergence of these networks eventually stalls, although at a bad solution, and the vanilla DQN even reaches unrealistic $\max|Q|$ values. Eventually, DQN-ER slowly improves again, like the Robust-DQN. In the CartPole-v1 environment, a positive reward is given for each timestep at which the agent manages to keep the pole upright, which gives short action-reward feedback loops. In the Acrobot-v1 environment, however, the agent starts in a 'bad' situation and constantly gets negative rewards until it manages to finish its task, receiving a neutral reward. This gives rise to long action-reward feedback loops, since the agent will only get positive feedback at the end of an episode. This explains why a 0.999 discount rate is necessary for good performance here and why the vanilla DQN diverges to unrealistic $\max|Q|$ values.

5 Conclusion & Discussion

In this paper we present the results of experiments regarding the convergence properties of DQN in two simple OpenAI Gym environments, examining the effect of two modifications – experience replay and target networks. The results show that these extensions are important factors for the stability of network parameters. The simultaneous application of experience replay and a target network, named Robust-DQN, typically outperforms the other configurations in terms of episode duration, average loss per episode, and maximum absolute Q-value. The best performance of Robust-DQN is observed in the CartPole environment, with convincingly superior episode durations and observed convergence. However, the Q-values for the Robust-DQN do not converge within 800 episodes in the Acrobot environment, while the performance does converge to a good level. The vanilla implementation converges in terms of Q-values, but to a poor solution. In general we see that the target network has the largest effect in divergence prevention, while experience replay has the largest effect on real performance.

When interpreting the CartPole results, take into account that suboptimal hyperparameter settings are purposely used to initially find divergence, this is done so the impact of DQN's extensions can be studied. However, it is therefore harder to generalise the implications of these results, as we do not observe a desirably functioning environment to begin with.

A reason we only observe convergence for CartPole (and not for Acrobot) could be the difference in reward structure between the two, as described in section 4.2. While the Acrobot-agent can go a large number of steps without positive reward, the CartPole environment provides intermittent positive rewards. This can make Acrobot a more challenging environment to learn, given roughly the same conditions as CartPole.

We also observe initial poor episode durations for Acrobot when target networks are used (Robust-DQN and DQN-TN), while the target network-free implementations seem to start off better. It indicates that the delay of learning target initially impedes performance. This could owe to the initial target network having little relevant information to aim towards, and possibly needing some warm-up time before its semi-static nature becomes beneficial.

In future research, it would be interesting to devote more time to optimising the network architecture (e.g., numbers of nodes and layers) or extend the training horizons to rule out convergence in any reasonable number of episodes. The latter was not done by us due to limitations in computing power and time.

References

J. Achiam, E. Knight, and P. Abbeel. Towards characterizing divergence in deep q-learning. *arXiv preprint arXiv:1903.08894*, 2019.

L. Baird. *Residual Algorithms: Reinforcement Learning with Function Approximation*. Morgan Kaufman Publishers, 1995.

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

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch, 2017.

R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

J. N. Tsitsiklis and B. Van Roy. Feature-based methods for large scale dynamic programming. *Machine Learning*, 22 (1-3):59–94, 1996.

H. Van Hasselt, Y. Doron, F. Strub, M. Hessel, N. Sonnerat, and J. Modayil. Deep reinforcement learning and the deadly triad. *arXiv preprint arXiv:1812.02648*, 2018.

C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

A Appendix

A.1 Experience replay with varying memory sizes

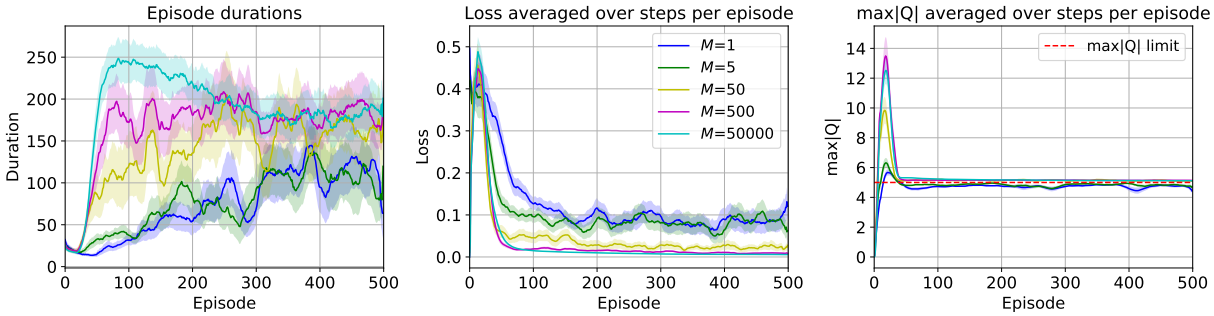

Figure 4: DQN-ER comparison for different memory sizes M for CartPole, where $\gamma = 0.8$. None of the settings seem to diverge. All estimated (absolute) Q-values seem to converge below or near the maximum realistic value of 5. The batch size per run was equal to $\min\{64, M\}$. Results were averaged over 20 neighbouring values to reduce clutter. The curves are smoothed over 20 consecutive values and filled regions represent $\sqrt{(\frac{Var(Y)}{N})}$, where Y is the relevant value and $N = 10$ is the number of runs.

.

A.2 Pseudocode for DQN

Algorithm 1: DQN with target networks and experience replay

initialise memory size M of buffer D for experience replay;
initialise Q-network with random parameters θ;
initialise target network $\hat{Q}$ with parameters $\theta^- = \theta$;
for *episode e = 1, ..., E* **do**
  initialise sequence s_1;
  for *time-step t = 1, ..., T* **do**
    select action a_t with probability ϵ;
    otherwise select greedy action $a_t = \arg\max_a Q(s_t, a; \theta)$ with probability $1 - \epsilon$;
    take action a_t and observe reward r_t and next state s_{t+1};
    store transition tuple (s_t, a_t, r_t, s_{t+1}) in D;
    Uniformly sample mini-batch of transition tuples (s_j, a_j, r_j, s_{j+1}) from D;
    if *episode e terminates at time-step* $j + 1$ **then**
      $y_j = r_j$;
    else
      $y_j = r_j + \gamma \max_{a'} \hat{Q}(s_{j+1}, a'; \theta^-)$;
    end
    do SGD step on $(y_j - Q(s_j, a_j; \theta))^2$ w.r.t. θ;
    if $t \mod C == 0$ **then**
      update target network $\hat{Q} = Q$;
    else
    end
  end
end