THE PROBLEM WITH DDPG: UNDERSTANDING FAILURES IN DETERMINISTIC ENVIRONMENTS WITH SPARSE REWARDS

Anonymous authors
Paper under double-blind review

ABSTRACT

In environments with continuous state and action spaces, state-of-the-art actor-critic reinforcement learning algorithms can solve very complex problems, yet can also fail in environments that seem trivial, but the reason for such failures is still poorly understood. In this paper, we contribute a formal explanation of these failures in the particular case of sparse reward and deterministic environments. First, using a very elementary control problem, we illustrate that the learning process can get stuck into a fixed point corresponding to a poor solution. Then, generalizing from the studied example, we provide a detailed analysis of the underlying mechanisms which results in a new understanding of one of the convergence regimes of these algorithms. The resulting perspective casts a new light on already existing solutions to the issues we have highlighted, and suggests other potential approaches.

1 INTRODUCTION

The Deep Deterministic Policy Gradient (DDPG) algorithm (Lillicrap et al. (2015)) is one of the earliest deep Reinforcement Learning (RL) algorithms designed to operate on potentially large continuous state and action spaces with a deterministic policy, and it is still one of the most widely used. However, it is often reported that DDPG suffers from instability in the form of sensitivity to hyper-parameters and propensity to converge to very poor solutions or even diverge. Various algorithms have improved stability by addressing well identified issues, such as the overestimation bias in TD3 (Fujimoto et al., 2018b) but, because a fundamental understanding of the phenomena underlying these instabilities is still missing, it is unclear whether these ad hoc remedies truly address the source of the problem. Thus, better understanding why these algorithms can fail even in very simple environments is a pressing question.

To investigate this question, we introduce in Section 4 a very simple one-dimensional environment with a sparse reward function where DDPG sometimes fails. Analyzing this example allows us to provide a detailed account of these failures. As presented in Section 5, we reveal the existence of a cycle of mechanisms operating in the sparse reward and deterministic case, leading to the quick convergence to a poor policy. In particular, we show that, when the reward is not discovered early enough, these mechanisms can lead to a *deadlock* situation where neither the actor nor the critic can evolve anymore. Critically, this deadlock persists even when the agent is subsequently trained with rewarded samples.

The study of these mechanisms is backed-up with formal proofs in a simplified context where the effects of function approximation is ignored. Nevertheless, the resulting understanding helps analyzing the practical phenomena encountered when using actors and critics represented as neural networks. From this new light, we revisit in Section 7 a few existing algorithms whose components provide an alternative to the building blocks involved in the undesirable cyclic convergence process, and we suggest alternative solutions to these issues.

2 RELATED WORK

Issues when combining RL with function approximation have been studied for a long time (Baird & Klopf, 1993; Boyan & Moore, 1995; Tsitsiklis & Van Roy, 1997). In particular, it is well known that deep RL algorithms can diverge when they meet three conditions coined as the ”deadly triad” (Sutton & Barto, 2018), that is when they use (1) function approximation, (2) bootstrapping updates and (3) off-policy learning. However, these questions are mostly studied in the continuous state, discrete action case. For instance, several recent papers have studied the mechanism of this instability using DQN (Mnih et al., 2013). In this context, four failure modes have been identified from a theoretical point of view by considering the effect of a linear approximation of the deep-Q updates and by identifying conditions under which the approximate updates of the critic are contraction maps for some distance over Q-functions (Achiam et al., 2019). Meanwhile, van Hasselt et al. (2018) shows that, due to its stabilizing heuristics, DQN does not diverge much in practice when applied to the ATARI domain. In contrast to these papers, here we study a failure mode specific to continuous actor-critic algorithms. It hinges on the fact that one cannot take the maximum over actions, and must rely on the actor as a proxy for the optimal action instead. Therefore, the failure mode identified in this paper cannot be reduced to any of the ones that affect DQN.

More related to our work, several papers have studied failure to gather rewarded experience from the environment due to poor exploration (Colas et al., 2018; Fortunato et al., 2017; Plappert et al., 2017), but we go beyond this issue by studying a case where the reward is actually found but not properly exploited. Finally, some of our insights are related to those of Fujimoto et al. (2018a), who shows that DDPG cannot learn with data uncorrelated to the distribution under the current policy.

3 Background: Deep Deterministic Policy Gradient

The DDPG algorithm (Lillicrap et al., 2015) is a deep RL algorithm based on the Deterministic Policy Gradient theorem (Silver et al., 2014). It borrows the use of a replay buffer and target networks from DQN (Mnih et al., 2015). DDPG is an instance of the Actor-Critic model. It learns both an actor function π_ψ (also called policy) and a critic function Q_θ, represented as neural networks whose parameters are respectively noted ψ and θ.

The deterministic actor takes a state $s \in S$ as input and outputs an action $a \in A$. The critic maps each state-action pair (s, a) to a value in $\mathbb{R}$. The reward $r : S \times A \to \mathbb{R}$, the termination function $t : S \times A \to \{0, 1\}$ and the discount factor $\gamma < 1$ are also specified as part of the environment.

The actor and critic are updated using stochastic gradient descent on two losses L_ψ and L_θ. These losses are computed from mini-batches of samples $(s_i, a_i, r_i, t_i, s_{i+1})$, where each sample corresponds to a transition $s_i \to s_{i+1}$ resulting from performing action a_i in state s_i, with subsequent reward $r_i = r(s_i, a_i)$ and termination index $t_i = t(s_i, a_i)$.

Two target networks $\pi_{\psi'}$ and $Q_{\theta'}$ are also used in DDPG. Their parameters ψ' and θ' respectively track ψ and θ using exponential smoothing. They are mostly useful to stabilize function approximation when learning the critic and actor networks. Since they do not play a significant role in the phenomena studied in this paper, we ignore them in the formal proofs given in appendices.

Equations (1) and (2) define L_ψ and L_θ:

$$L_\psi = -\sum_i Q_\theta\left(s_i, \pi_\psi\left(s_i\right)\right) \tag{1}$$

$$\begin{cases} \forall i, y_i = r_i + \gamma(1 - t_i) Q_{\theta'}\left(s_{i+1}, \pi_{\psi'}\left(s_{i+1}\right)\right) \\ L_\theta = \sum_i \Big[Q_\theta\left(s_i, a_i\right) - y_i \Big]^2 . \end{cases} \tag{2}$$

Training for the loss given in (1) yields the parameter update in (3), with α the learning rate:

$$\psi \leftarrow \psi + \alpha \sum_i \frac{\partial \pi_\psi(s_i)}{\partial \psi}^T \left. \nabla_a Q_\theta(s_i, a) \right|_{a=\pi_\psi(s_i)} . \tag{3}$$

As DDPG uses a replay buffer, the mini-batch samples are acquired using a behaviour policy β which may be different from the actor π. Usually, β is defined as π plus a noise distribution, which in the case of DDPG is either a Gaussian function or the more sophisticated Ornstein-Uhlenbeck noise.

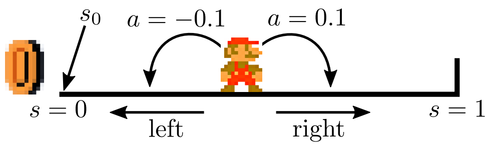
Importantly for this paper, the behaviour of DDPG can be characterized as an intermediate between two extreme regimes:

- When the actor is updated much faster than the critic, the policy becomes greedy with respect to this critic, resulting into a behaviour closely resembling that of the Q-LEARNING algorithm. When it is close to this regime, DDPG can be characterized as off-policy.
- When the critic is updated much faster than the actor, the critic tends towards $Q^{\pi}(s,a)$. The problems studied in this paper directly come from this second regime.

A more detailed characterization of these two regimes in given in Appendix A.

4 Study of a simple environment

In this section, we introduce a simplistic environment which we call 1D-TOY. It is a one-dimensional, discrete-time, continuous state and action problem, depicted in Figure 1.



$$S = [0, 1] \tag{4a}$$
$$A = [-0.1, 0.1] \tag{4b}$$
$$s_0 = 0 \tag{4c}$$
$$s_{t+1} = \min\left(1, \max\left(0, s_t + a_t\right)\right) \tag{4d}$$
$$r_t = t_t = \mathbb{1}_{s_t + a_t < 0} \tag{4e}$$

Figure 1: The 1D-TOY environment

Despite its simplicity, DDPG can fail on 1D-TOY. We first show that DDPG fails to reach 100% success. We then show that if learning a policy does not succeed soon enough, the learning process can get stuck. Besides, we show that the initial actor can be significantly modified in the initial stages before finding the first reward. Finally, we explain how the combination of these phenomena and an undesirable behaviour of DDPG can result into a deadlock situation.

4.1 Empirical study

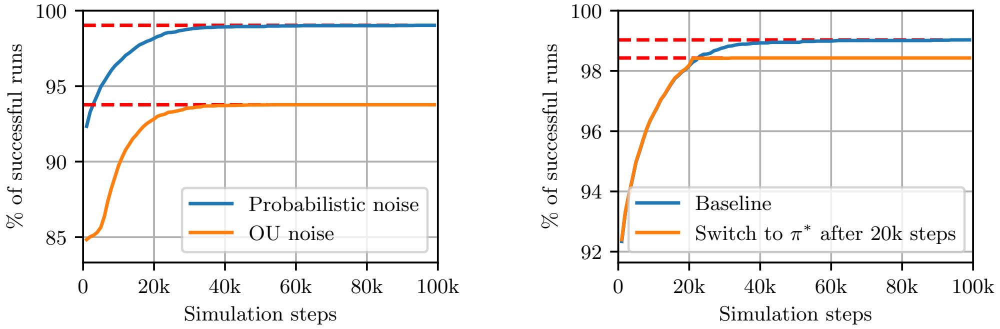
In all experiments, we set the maximum episode length N to 50, but the observed phenomena persist with other values.

Residual failure to converge using different noise processes We start by running DDPG on the 1D-TOY environment. This environment is trivial as one infinitesimal step to the left is enough to obtain the reward, end the episode and succeed, thus we might expect a quick 100% success. However, the first attempt using an Ornstein-Uhlenbeck (OU) noise process shows that DDPG succeeds in only 94% of cases, see Figure 2a.

These failures might come from an exploration problem. Indeed, at the start of each episode the OU noise process is reset to zero and gives little noise in the first steps of the episode. In order to remove this potential source of failure, we replace the OU noise process with an exploration strategy similar to ϵ-greedy which we call ”probabilistic noise”. For some $0 < p < 1$, with probability p, the action is randomly sampled (and the actor is ignored), and with probability $1 - p$ no noise is used and the raw action is returned. In our tests, we used $p = 0.1$. This guarantees at least a 5% chance of success at the first step of each episode, for any policy.

Nevertheless, Figure 2a shows that even with probabilistic noise, about 1% of seeds still fail to converge to a successful policy in 1D-TOY, even after 100k training steps. All the following tests are performed using probabilistic noise.

We now focus on these failures. On all failing seeds, we observe that the actor has converged to a saturated policy that always goes to the right ($\forall s, \pi(s) = 0.1$). However, some mini-batch samples have non-zero rewards because the agent still occasionally moves to the left, due to the probabilistic



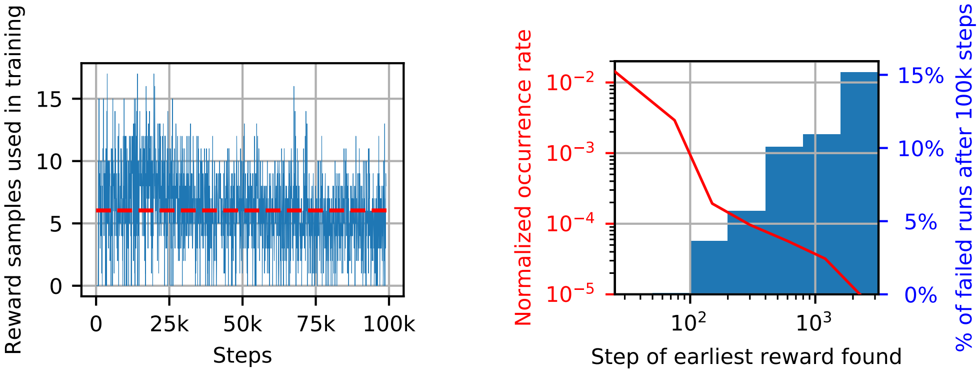
(a) Success rate of DDPG with Ornstein-Uhlenbeck (OU) and probabilistic noise. Even with probabilistic noise, DDPG fails on about 1% of the seeds.

(b) Comparison between DDPG with probabilistic noise and a variant in which the behaviour policy is set to the optimal policy π^* after 20k steps.

Figure 2: Success rate of variants of DDPG on 1D-TOY over learning steps, averaged over 10k seeds. More details on learning algorithm and success evaluation are given in Appendix E.

noise applied during rollouts. The expected fraction of non-zero rewards is slightly more than 0.1%[1]. Figure 3a shows the occurrence of rewards in minibatches taken from the replay buffer when training DDPG on 1D-TOY. After each rollout (episode) of n steps, the critic and actor networks are trained n times on minibatches of size 100. So for instance, a failed episode of size 50 is followed by a training on a total of 5000 samples, out of which we expect more than 5 in average are rewarded transitions. More details about the implementation are available in Appendix E.

The constant presence of rewarded transitions in the minibatches suggests that the failures of DDPG on this environment are not due to insufficient exploration by the behaviour policy.



(a)

(b)

Figure 3: (a) Number of rewards found in mini-batches during training. After a rollout of n steps, the actor and critic are both trained on n minibatches of size 100. The red dotted line indicates an average of 6.03 rewarded transitions present in these n minibatches. (b) In red, normalized probability of finding the earliest reward at this step. In blue, for each earliest reward bin, fraction of these episodes that fail to converge to a good actor after 100k steps. Note that when the reward is found after one or two episodes, the convergence to a successful actor is certain.

Correlation between finding the reward early and finding the optimal policy We have shown that DDPG can get stuck in 1D-TOY despite finding the reward regularly. Now we show that when

[1] 10% of steps are governed by probabilistic noise, of which at least 2% are the first episode step, of which 50% are steps going to the left and leading to the reward.

DDPG finds the reward early in the training session, it is also more successful in converging to the optimal policy. On the other hand, when the first reward is found late, the learning process more often gets stuck with a sub-optimal policy.

From Figure 3b, the early steps appear to have a high influence on whether the training will be successful or not. For instance, if the reward is found in the first 50 steps by the actor noise (which happens in 63% of cases), then the success rate of DDPG is 100%. However, if the reward is first found after more than 50 steps, then the success rate drops to 96%. Figure 3b shows that finding the reward later results in lower success rates, down to 87% for runs in which the reward was not found in the first 1600 steps. Therefore, we claim that there exists a critical time frame for finding the reward in the very early stages of training.

Spontaneous actor drift At the beginning of each training session, the actor and critic of DDPG are initialized to represent respectively close-to-zero state-action values and close-to-zero actions. Besides, as long as the agent does not find a reward, it does not benefit from any utility gradient. Thus we might expect that the actor and critic remain constant until the first reward is found. Actually, we show that even in the absence of reward, training the actor and critic triggers non-negligible updates that cause the actor to reach a saturated state very quickly.

To investigate this, we use a variant of 1D-TOY called DRIFT where the only difference is that no rewarded or terminal transitions are present in the environment. We also use a stripped-down version of DDPG, removing rollouts and using random sampling of states and actions as minibatches for training.

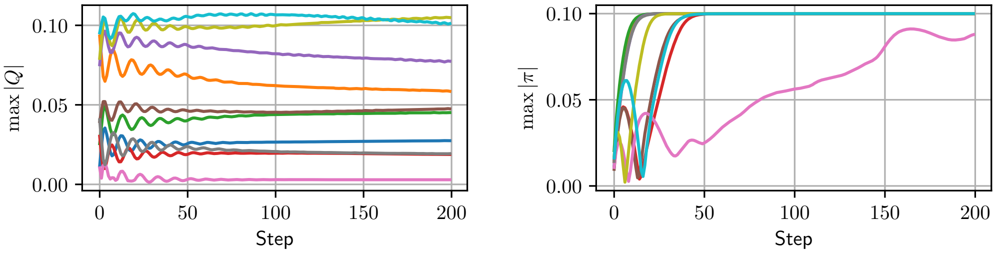


Figure 4: Drift of $\max |Q|$ (left) and $\max |\pi|$ (right) in the DRIFT environment, for 10 different seeds. In the absence of reward, the critic oscillates briefly before stabilizing. However, the actor very quickly reaches a saturated state, at either $\forall s, \pi(s) = 0.1$ or -0.1.

Figure 4 shows that even in the absence of reward, the actor function drifts rapidly (notice the horizontal scale in steps) to a saturated policy, in a number of steps comparable to the ”critical time frame” identified above. The critic also has a transitive phase before stabilizing.

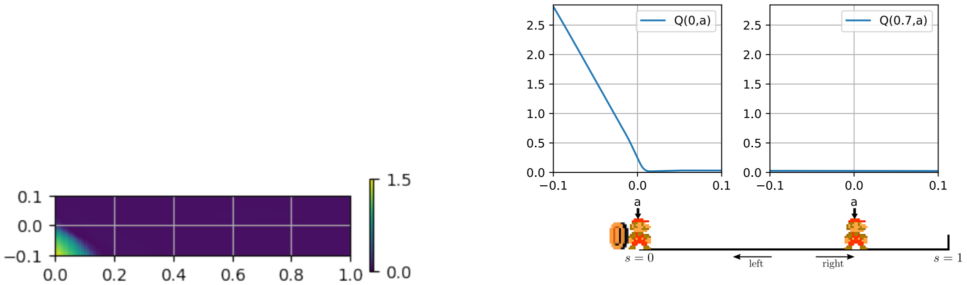
The fact that $\max_{s,a} |Q(s,a)|$ can increase in the absence of reward can seem counter-intuitive, since in the loss function presented in Equation (2), $|y_i|$ can never be greater than $\max_{s,a} |Q(s,a)|$. However, it should be noted that the changes made to Q are not local to the minibatch points, and increasing the value of Q for one input (s,a) may cause its value to increase for other inputs too, which may cause an increase in the global maximum of Q.

4.2 EXPLAINING THE DEADLOCK SITUATION FOR DDPG ON 1D-TOY

Up to now, we have shown that DDPG fails about 1% of times on 1D-TOY, despite the simplicity of this environment. We have now collected the necessary elements to explain the mechanisms of this deadlock in 1D-TOY.

Figure 5 shows the value of the critic in a failed run of DDPG on 1D-TOY. We see that the value of the reward is not propagated correctly outside of the region in which the reward is found in a single step $\{(s,a) \mid s + a < 0\}$.

The key of the deadlock is that once the actor has drifted to $\forall s, \pi(s) = 0.1$, it is updated according to $\nabla_a Q_\theta(s,a)|_{a=\pi_\psi(s)}$ (Equation (3)). Figure 5b shows that for $a = \pi(s) = 0.1$, this gradient is zero



(a) Critic values in the deadlock configuration. The critic is non-zero only in the region that immediately leads to a reward ($s + a < 0$)

(b) Two snapshots of the critic for different states in a failed run. The high Q values in the $s + a < 0$ region are not propagated.

Figure 5: Visualization of the critic in a failing run, in which the actor is stuck to $\forall s, \pi(s) = 0.1$.

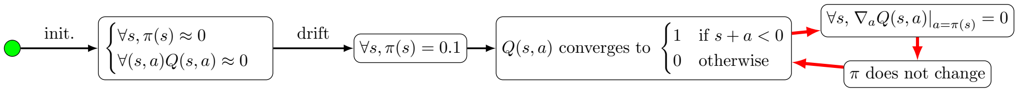

init.
$\forall s, \pi(s) \approx 0$
$\forall (s,a) Q(s,a) \approx 0$
drift
$\forall s, \pi(s) = 0.1$
$Q(s,a)$ converges to $\begin{cases} 1 & \text{if } s+a<0 \\ 0 & \text{otherwise} \end{cases}$
$\forall s, \nabla_a Q(s,a)|_{a=\pi(s)} = 0$
π does not change

Figure 6: Deadlock observed in 1D-TOY, represented as the cycle of red arrows.

therefore the actor is not updated. Besides, the critic is updated using $y_i = r(s_i, a_i) + \gamma Q(s'_i, \pi(s'_i))$ as a target. Since $Q(s'_i, 0.1)$ is zero, the critic only needs to be non-zero for directly rewarded actions, and for all other samples the target value remains zero. In this state the critic loss given in Equation (2) is minimal, so there is no further update of the critic and no further propagation of the state-action values. The combination of the above two facts clearly results in a deadlock.

Importantly, the constitutive elements of this deadlock do not depend on the batches used to perform the update, and therefore do not depend on the experience selection method. We tested this experimentally by substituting the behaviour policy for the optimal policy after 20k training steps. Results are presented in Figure 2b and show that, once stuck, even when it is given ideal samples, DDPG stays stuck in the deadlock configuration.

This also explains why finding the reward early results in better performance. When the reward is found early enough, $\pi(s_0)$ has not drifted too far, and the gradient of $Q(s_0, a)$ at $a = \pi(s_0)$ drives the actor back into the correct direction.

Note however that even when the actor drifts to the right, DDPG does not always fail. Indeed, because of function approximators the shape of the critic when finding the reward for the first time varies, and sometimes converges slowly enough for the actor to be updated before the convergence of the critic.

Figure 6 summarizes the process explained above. The entry point is represented using a green dot. First, the actor drifts to $\forall s, \pi(s) = 0.1$, then the critic converges to Q^{π} which is a piecewise-constant function (Experiment in Figure 5, proof in Theorem 1 in Appendix B), which in turn means that the critic provides no gradient, therefore the actor is not updated (as seen in Equation 3, more details in Theorem 2) [2].

5 Generalization

Our study of 1D-TOY revealed how DDPG can get stuck in this simplistic environment. We now generalize to the broader context of more general continuous action actor critic algorithms, including at least DDPG and TD3, and acting in any deterministic and sparse reward environment. The generalized deadlock mechanism is illustrated in Figure 7 and explained hereafter in the idealized context of perfect approximators, with formal proofs rejected in appendices. We then explore the consequences of getting into this cyclic process.

[2] Note that Figure 5 shows a critic state which is slightly different from the one presented in Figure 6, due to the limitations of function approximators.

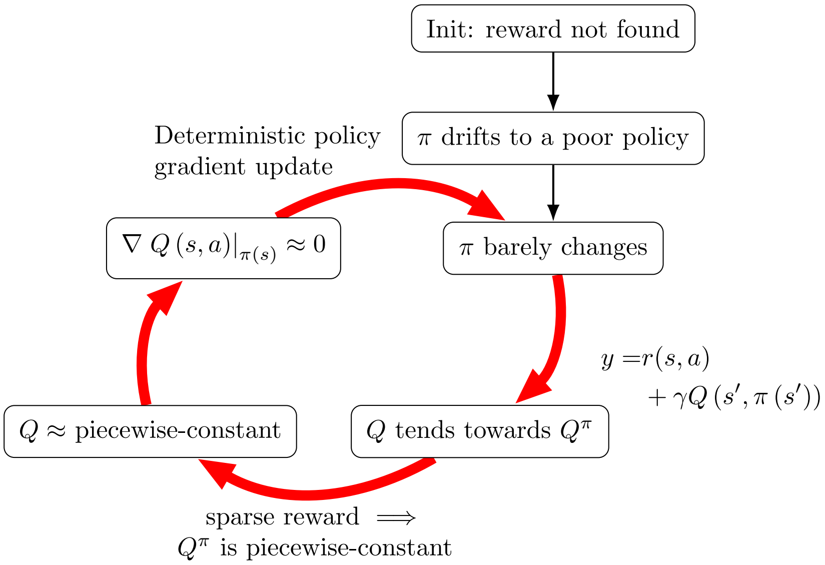

Init: reward not found
π drifts to a poor policy
Deterministic policy gradient update
$\nabla Q(s,a)|_{\pi(s)} \approx 0$
π barely changes
$y = r(s,a) + \gamma Q(s', \pi(s'))$
$Q \approx$ piecewise-constant
Q tends towards Q^π
sparse reward $\Longrightarrow$ Q^π is piecewise-constant

Figure 7: A cyclic view of the undesirable convergence process in continuous action actor-critic algorithms, in the deterministic and sparse reward case.

5.1 CYCLIC CONVERGENCE PROCESS

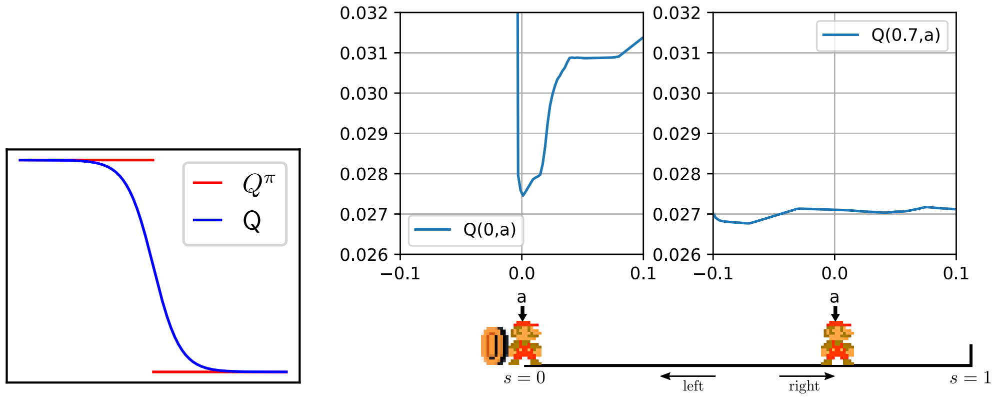
Entry point: As shown in the previous section, before the behaviour policy finds any reward, training the actor and critic can still trigger non-negligible updates that may cause the actor to quickly reach a poor state and stabilize. This defines our entry point in the process.

Q tends towards $\mathbf{Q}^\pi$: A first step into the cycle is that, if the critic is updated faster than the policy, the update rule of the critic Q given in Equation (2) makes Q converge to Q^π. This is presented in detail in Appendix C.

$\mathbf{Q}^\pi$ is piecewise-constant: In Appendix D, we then show that, in a deterministic environment with sparse terminal rewards, Q^π is piecewise-constant because $V^\pi(s')$ only depends on two things: the (integer) number of steps required to reach a rewarded state from s', and the value of this reward state, which is itself piecewise-constant. Note that we can reach the same conclusion with non-terminal rewards, by making the stronger hypothesis on the actor that $\forall s, r(s, \pi(s)) = 0$. Notably, this is the case for the actor $\forall s, \pi(s) = 0.1$ on 1D-TOY.

Q is approximately piecewise-constant and $\nabla_\mathbf{a} \mathbf{Q}(\mathbf{s}, \mathbf{a})|_{\mathbf{a}=\pi(\mathbf{s})} \approx \mathbf{0}$: Quite obviously, from Q^π is piecewise-constant and Q tends towards Q^π, we can infer that Q progressively becomes almost piecewise-constant as the cyclic process unfolds. Actually, the Q function is estimated by a function approximator which is never truly discontinuous. The impact of this fact is studied in Section 6. However, we can expect Q to have mostly flat gradients since it is trained to match a piecewise-constant function. We can thus infer that, globally, $\nabla_a Q(s, a)|_{a=\pi(s)} \approx 0$. And critically, the gradients in the flat regions far from the discontinuities give little information as to how to reach regions of higher values.

π barely changes: DDPG uses the deterministic policy gradient update, as seen in Equation (3). This is an analytical gradient that does not incorporate any stochasticity, because Q is always differentiated exactly at $(s, \pi(s))$. Thus the actor update is stalled, even when the reward is regularly found by the behaviour policy. This closes the loop of our process.



(a)

(b)

Figure 8: (a) Example of a monotonous function approximator. (b) Simply changing the vertical scale of the graphs presented in Figure 5b reveals that the function approximator is not perfectly flat, and has many unwanted local extrema. Specifically, continuously changing $\pi(0)$ from $\pi(0) = 0.1$ to $\pi(0) < 0$ requires traversing a significant valley in $Q(0, a)$, while $\pi(0) = 0.1$ is a strong local maximum.

5.2 Consequences of the convergence cycle

As illustrated with the red arrows in Figure 7, the more loops performed in the convergence process, the more the critic tends to be piecewise-constant and the less the actor tends to change. Importantly, this cyclic convergence process is triggered as soon as the changes on the policy drastically slow down or stop. What matters for the final performance is the quality of the policy reached before this convergence loop is triggered. Quite obviously, if the loop is triggered before the policy gets consistently rewarded, the final performance is deemed to be poor.

The key of this undesirable convergence cycle lies in the use of the deterministic policy gradient update given in Equation (3). Actually, rewarded samples found by the exploratory behaviour policy β tend to be ignored by the conjunction of two reasons. First, the critic is updated using $Q(s', \pi(s'))$ and not $Q(s, \beta(s))$, thus if π differs too much from β, the values brought by β are not properly propagated. Second, the actor being updated through (3), i.e. using the analytical gradient of the critic with respect to the actions of π, there is no room for considering other actions than that of π. Besides, the actor update involves only the state s of the sample taken from the replay buffer, and not the reward found from this sample $r(s, a)$ or the action performed. For each sample state s, the actor update is intended to make $\pi(s)$ converge to $\operatorname{argmax}_a \pi(s, a)$ but the experience of different actions performed for identical or similar states is only available through $Q(s, \cdot)$, and in DDPG it is only exploited through the gradient of $Q(s, \cdot)$ at $\pi(s)$, so the process can easily get stuck in a local optimum, especially if the critic tends towards a piecewise-constant function, which as we have shown happens when the reward is sparse.

Note that since TD3 also updates the actor according to (3) and the critic according to (2), it is susceptible to the same failures as DDPG.

6 Impact of function approximation

We have just explained that when the actor has drifted to an incorrect policy before finding the reward, an undesirable convergence process should result in DDPG getting stuck to this policy. However, in 1D-TOY, we measured that the actor drifts to a policy moving to the right in 50% of cases, but the learning process only fails 1% of times. More generally, despite the issues discussed in this paper, DDPG has been shown to be efficient in many problems. This better-than-predicted success can be attributed to the impact of function approximation.

Figure 8a shows a case in which the critic approximates Q^π while keeping a monotonous slope between the current policy value and the reward. In this case, the actor is correctly updated towards the reward (if it is close enough to the discontinuity). This is the most often observed case, and naturally we expect approximators to smooth out discontinuities in target functions in a monotonous way, which facilitates gradient ascent. However, the critic is updated not only in state-action pairs where $Q^\pi(s,a)$ is positive, but also at points where $Q^\pi(s,a) = 0$, which means that the bottom part of the curve also tends to flatten. As this happens, we can imagine phenomena that are common when trying to approximate discontinuous functions, such as the overshoot observed in Figure 8b. In this case, the gradient prevents the actor from improving.

7 Potential solutions

In the previous section, we have shown that actor-critic algorithms such as DDPG and TD3 could not recover from early convergence to a poor policy due to the combination of three factors whose dependence is highlighted in Figure 7: the use of the deterministic policy gradient update, the use of $Q(s', \pi(s'))$ in the critic update, and the attempt to address sparse reward in deterministic environments. In this section, we categorize existing or potential solutions to the above issue in terms of which of the above factor they remove.

Avoiding sparse rewards: Transforming a sparse reward problem into a dense one can solve the above issue as the critic should not converge to a piecewise-constant function anymore. This can be achieved for instance by using various forms of shaping (Konidaris & Barto, 2006) or by adding auxiliary tasks (Jaderberg et al., 2016; Riedmiller et al., 2018). We do not further investigate these solutions here, as they are mainly problem-dependent and may introduce bias when the reward transformation results in deceptive gradient or modifies the corresponding optimal policy.

Replacing the policy-based critic update: As explained above, if some transition (s, a, s') leading to a reward is found in the replay buffer, the critic update corresponding to this transition uses $Q(s', \pi(s'))$, therefore not propagating the next state value that the behaviour policy may have found. Of course, when using the gradient from the critic, the actor update should tend to update π to reflect the better policy such that $\pi(s') \to a'$, but the critic does not always provide an adequate gradient as shown before.

If performing a maximum over a continuous action space was possible, using $\max_a Q(s', a)$ instead of $Q(s', \pi(s'))$ would solve the issue. Several works start from this insight.

Some methods directly sample the action space and look for such an approximate maximum (Kalashnikov et al., 2018; Simmons-Edler et al., 2019). Quite obviously, even if sampling can provide a good enough baseline, these methods do not scale well to large actions spaces. Many improvements to this can be imagined by changing the way the action space is sampled, such as including $\pi(s)$ in the samples, to prevent picking a worse action than the one provided by the actor, sampling preferentially around $\pi(s)$, or around $\pi(s + \epsilon)$, or just using actions taken from the replay buffer. Interestingly, using a stochastic actor such as in the Soft Actor Critic (SAC) algorithm (Haarnoja et al., 2018a;b) can be considered as an option under this perspective.

Another approach relies on representing the critic as the V function rather than the Q function. The same way $\pi(s)$ tends to approximate $\operatorname{argmax}_a Q(s, a)$, V tends to approximate $\max_a Q(s, a)$, and is updated when finding a transition that raises the value of a state. Using V, performing a maximum in the critic update is not necessary anymore. The prototypical actor-critic algorithm using a model of V as a critic is CACLA (Van Hasselt & Wiering, 2007). However, approximating V with neural networks can prove more unstable than approximating Q, as function approximation can be sensitive to the discontinuities resulting form the implicit maximization over Q values.

Replacing the deterministic policy gradient update: Instead of relying on the deterministic policy gradient update, one can rely on a stochastic policy to perform a different actor update. This is the case of SAC, as mentioned just above. Because SAC does not use $Q(s', \pi(s'))$ in its update rule, it does not suffer from the undesirable convergence process described here.

Another solution consists in completely replacing the actor update mechanism, using regression to update $\pi(s)$ towards any action better than the current one. This could be achieved by updating the actor and the critic simultaneously: when sampling a higher-than-expected critic value $y_i > Q(s_i, a_i)$, one may update $\pi(s_i)$ towards a_i using:

$$L_\psi = \sum_i \delta_{y_i > Q(s_i, \pi(s_i))} \left(\pi(s_i) - a_i\right). \tag{5}$$

This is similar to the behaviour of CACLA, as analyzed in Zimmer & Weng (2019).

8 Conclusion

In RL, continuous action and sparse reward environments are challenging. In these environments, the fact that a good policy cannot be learned if exploration is not efficient enough to find the reward is well-known and trivial. In this paper, we have established the less trivial fact that, if exploration does find the reward consistently but not early enough, an actor-critic algorithm can get stuck into a configuration from which rewarded samples are just ignored. We have formally characterized the reasons for this situation and we have outlined potential solutions. Beyond this, we believe our work sheds new light on the convergence regime of actor-critic algorithms, which may be the starting point of further formal studies.

9 Acknowledgements

Anonymized for submission.

References

Joshua Achiam, Ethan Knight, and Pieter Abbeel. Towards Characterizing Divergence in Deep Q-Learning. *arXiv:1903.08894 [cs]*, March 2019. URL http://arxiv.org/abs/1903.08894. arXiv: 1903.08894.

L. C. Baird and A. H. Klopf. Reinforcement learning with high-dimensional, continuous actions. Technical report, Wright-Patterson Air Force Base Ohio: Wright Laboratory. (Available from the Defense Technical Information Center, Cameron Station, Alexandria, VA 22304-6145), 1993.

Justin A. Boyan and Andrew W. Moore. Generalization in reinforcement learning: Safely approximating the value function. In *Advances in neural information processing systems*, pp. 369–376, 1995.

Cédric Colas, Olivier Sigaud, and Pierre-Yves Oudeyer. GEP-PG: Decoupling Exploration and Exploitation in Deep Reinforcement Learning Algorithms. In *International Conference in Machine Learning (ICML)*, February 2018. URL https://arxiv.org/abs/1802.05054.

Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, Charles Blundell, and Shane Legg. Noisy Networks for Exploration. *arXiv:1706.10295 [cs, stat]*, June 2017. URL http://arxiv.org/abs/1706.10295. arXiv: 1706.10295.

Scott Fujimoto, David Meger, and Doina Precup. Off-Policy Deep Reinforcement Learning without Exploration. *arXiv:1812.02900 [cs, stat]*, December 2018a. URL http://arxiv.org/abs/1812.02900. arXiv: 1812.02900.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing Function Approximation Error in Actor-Critic Methods. *https://arxiv.org/abs/1802.09477*, February 2018b.

M. Geist and O. Pietquin. Parametric value function approximation: A unified view. In *2011 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, pp. 9–16, April 2011. doi: 10.1109/ADPRL.2011.5967355.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. *arXiv:1801.01290 [cs, stat]*, January 2018a. URL http://arxiv.org/abs/1801.01290. arXiv: 1801.01290.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018b.

Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.

Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. Qt-Opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arXiv preprint arXiv:1806.10293*, 2018.

George Konidaris and Andrew Barto. Autonomous shaping: Knowledge transfer in reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pp. 489–496. ACM, 2006.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv:1509.02971 [cs, stat]*, September 2015. URL `http://arxiv.org/abs/1509.02971`. arXiv: 1509.02971.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning. *arXiv:1312.5602 [cs]*, December 2013. URL `http://arxiv.org/abs/1312.5602`. arXiv: 1312.5602.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015. ISSN 1476-4687. doi: 10.1038/nature14236. URL `https://www.nature.com/articles/nature14236`.

Matthias Plappert, Rein Houthooft, Prafulla Dhariwal, Szymon Sidor, Richard Y. Chen, Xi Chen, Tamim Asfour, Pieter Abbeel, and Marcin Andrychowicz. Parameter space noise for exploration. *arXiv preprint arXiv:1706.01905*, 2017.

Martin Riedmiller, Roland Hafner, Thomas Lampe, Michael Neunert, Jonas Degrave, Tom Van de Wiele, Volodymyr Mnih, Nicolas Heess, and Jost Tobias Springenberg. Learning by Playing - Solving Sparse Reward Tasks from Scratch. *arXiv:1802.10567 [cs, stat]*, February 2018. URL `http://arxiv.org/abs/1802.10567`. arXiv: 1802.10567.

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic Policy Gradient Algorithms. In *International Conference on Machine Learning*, pp. 387–395, January 2014. URL `http://proceedings.mlr.press/v32/silver14.html`.

Riley Simmons-Edler, Ben Eisner, Eric Mitchell, Sebastian Seung, and Daniel Lee. Q-learning for continuous actions with cross-entropy guided policies. *arXiv preprint arXiv:1903.10605*, 2019.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, November 2018. ISBN 978-0-262-03924-6. Google-Books-ID: 6DKPtQEACAAJ.

John N. Tsitsiklis and Benjamin Van Roy. Analysis of temporal-diffference learning with function approximation. In *Advances in neural information processing systems*, pp. 1075–1081, 1997.

Hado Van Hasselt and Marco A. Wiering. Reinforcement learning in continuous action spaces. In *IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL)*, pp. 272–279, 2007.

Hado van Hasselt, Yotam Doron, Florian Strub, Matteo Hessel, Nicolas Sonnerat, and Joseph Modayil. Deep Reinforcement Learning and the Deadly Triad. *arXiv:1812.02648 [cs]*, December 2018. URL `http://arxiv.org/abs/1812.02648`. arXiv: 1812.02648.

Christopher J. C. H. Watkins. *Learning with Delayed Rewards*. PhD thesis, Psychology Department, University of Cambridge, England, 1989.

Matthieu Zimmer and Paul Weng. Exploiting the sign of the advantage function to learn deterministic policies in continuous domains. *arXiv preprint arXiv:1906.04556*, 2019.

A Two regimes of DDPG

In this section, we characterize the behavior of DDPG as an intermediate between two extremes, that we respectively call the *critic-centric view*, where the actor is updated faster, resulting in an algorithm close to Q-LEARNING, and the *actor-centric view*, where the critic is updated faster, resulting in a behaviour more similar to Policy Gradient.

A.1 Actor update: the critic-centric view

The Q-LEARNING algorithm (Watkins, 1989) and its continuous state counterpart DQN (Mnih et al., 2013) rely on the computation of a policy which is greedy with respect to the current critic at every time step, as they simply take the maximum of the Q-values over a set of discrete actions. In continuous action settings, this amounts to an intractable optimization problem if the action space is large and nontrivial.

We get a simplified vision of DDPG by considering an extreme regime where the actor updates are both fast enough and good enough so that $\forall s, \pi(s) \approx \mathrm{argmax}_a\, Q(s,a)$. We call this the *critic-centric* vision of DDPG, since the actor updates are assumed to be ideal and the only remaining training is performed on the critic.

In this regime, by replacing $\pi(s)$ with $\mathrm{argmax}_a\, Q(s,a)$ in Equation (2), we get $y_i = r_i + \gamma(1 - t_i)\max_a Q(s_i, a)$, which corresponds to the update of the critic in Q-LEARNING and DQN. A key property of this regime is that, since the update in based on a maximum over actions, the resulting algorithm is truly off-policy. We can thus infer that most of the off-policiness property of DDPG comes from keeping it close to this regime.

A.2 Critic update: the actor-centric view

Symmetrically to the previous case, if the critic is updated well and faster than the actor, it tends to represent the critic of the current policy, Q^{π}. Furthermore, if the actor tends to change slowly enough, critic updates can be both fast and good enough so that it reaches the fixed point of the Bellman equation, that is $\forall(s,a,r,t,s'), Q(s,a) = r + \gamma(1-t)Q(s', \pi(s'))$.

In this case, the optimization performed in (1) mostly consists in updating the actor so that it exploits the corresponding critic by applying the deterministic policy gradient on the actor. This gives rise to a *actor-centric* vision of DDPG.

B Deadlock in 1D-toy

In this section, we prove that there exists a state of DDPG that is a deadlock in the 1D-TOY environment. This proof directly references Figure 6. Let us define two functions Q and π_ψ such that:

$$\forall(s,a), Q(s,a) = \mathbb{1}_{s+a<0} \tag{6}$$

$$\forall s \in S, \pi_\psi(s) = 0.1 \tag{7}$$

From now on, we will use notation $\pi := \pi_\psi$.

Theorem 1. $(Q$,$\pi_\psi)$ is a fixed point for the critic update.

Proof. The critic update is governed by Equation 2.

Let $(s_i, a_i, r_i, t_i, s'_i)$ be a sample from the replay buffer. The environment dictates that $r_i = t_i = \mathbb{1}_{s_i+a_i<0}$.

$$\begin{aligned}
y_i &= r_i + \gamma(1-t_i)Q\left(s'_i, \pi_\psi(s'_i)\right) \\
y_i &= r_i + \gamma(1-t_i)Q\left(s'_i, 0.1\right) && \text{by (7)} \\
y_i &= r_i && \text{by (6)} \\
y_i &= \mathbb{1}_{s_i+a_i<0} \\
y_i &= Q(s_i, a_i) && \text{by (6)}
\end{aligned}$$

Therefore, for each sample $y_i = Q(s_i, a_i)$, and L is null and minimal. Therefore θ will not be updated during the critic update. □

Theorem 2. $(Q$,$\pi_\psi)$ is a fixed point for the actor update.

Proof. The actor update is governed by Equation 3.

Let $\{(s_i, a_i, r_i, t_i, s_i')\}$ be a set of samples from the replay buffer. The environment dictates that $\forall i, r_i = t_i = \mathbb{1}_{s_i+a_i<0}$.

$$\psi \leftarrow \psi + \alpha \sum_i \frac{\partial \pi_\psi(s_i)}{\partial \psi}^T \left.\nabla_a Q(s_i, a)\right|_{a=\pi_\psi(s_i)}$$

Since $Q(s_i, a) = \mathbb{1}_{s_i+a<0}$, $\nabla_a Q(s_i, a) = 0$, so ψ will not be updated during the actor update. □

In this section, we assume that Q is any function, however in implementations Q is often a parametric neural network Q_θ, which cannot be discontinuous. Effects of this approximation are discussed in Section 6.

C Convergence of the critic to Q^π

Notation For a state-action pair s, a, we define s_1 as the result of applying action a at state s in the deterministic environment. For a given policy π, we define a_1 as $\pi(s_1)$. Recursively, for any $i \geq 1$, we define s_{i+1} as the result of applying action a_i to state s_i, and a_i as $\pi(s_i)$.

Definition 1. Let $(s, a) \in S \times A$. If (s, a) is terminal, then we set $N = 0$. Otherwise, we set N to the number of subsequent transitions with policy π to reach a terminal state. Therefore, the transition (s_N, a_N) is always terminal. We generalize by setting $N = \infty$ when no terminal transition is ever reached.

We define the state-action value function of policy π as:

$$Q^\pi(s, a) \coloneqq r(s, a) + \sum_{i=1}^{N} \gamma^i r(s_i, a_i)$$

Note that when $N = \infty$, the sum converges under the hypothesis that rewards are bounded and $\gamma < 1$.

If π is fixed, Q is updated regularly via approximate dynamic programming with the Bellman operator for the policy π. Under strong assumptions, or assuming exact dynamic programming, it is possible to prove (Geist & Pietquin (2011)) that the iterated application of this operator converges towards a unique function Q^π, which corresponds to the state-action value function of π as defined above. It is usually done by proving that the Bellman operator is a contraction mapping, and also applies in deterministic cases.

However, when using approximators such as neural nets, no theroretical results of convergence exists, to the best of our knowledge. In this paper we assume that this convergence is true, and in the experimental results we did not observe any failures to converge towards Q^π. On the contrary, we observe that this converge occurs, and can be what starts the deadlock cycle studied in Section 5.

D Proof that Q^π is piecewise-constant

In this section, we show that in deterministic environments with terminal sparse rewards, Q^π is piecewise-constant.

Definition 2. In this article, for $I \subset \mathbb{R}^n$, we say a function $f : I \to \mathbb{R}$ is piecewise-constant if $\forall x_0 \in I$, either $\left.\nabla_x f(u)\right|_{x=x_0} = 0$, or f has no gradient at x_0.

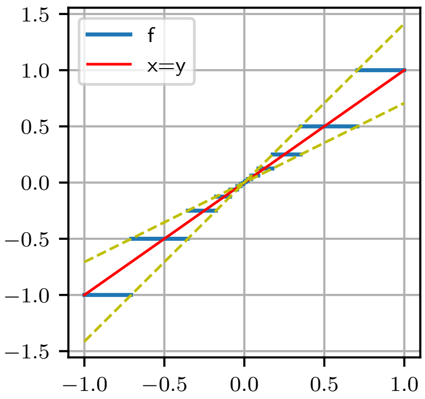


Figure 9: Example of a non-continuous function f with values in $\left\{\left(\frac{1}{2}\right)^n \mid n \in \mathbb{N}\right\}$, approximating the identity function. However, this function is not differentiable because the difference quotient does not converge but instead oscillates between two values.

Theorem 3. In a deterministic environment with terminal sparse rewards, for any π, Q^π is piecewise-constant.

Proof. Note that this proof can be trivialized by assuming that around any point where the gradient is defined, there exists a neighbourhood in which the function is continuous. In this case, the intermediate value theorem yields an uncountable set of values of the function in this neighbourhood, which contradicts the countable number of possible discounted rewards.

The crux of the following proof is that even when no such neighbourhood exists, the gradient is either null of non-existent. This behavior is shown in Figure 9.

Using the notations of Definition 1 and the theorem hypothesis that rewarded transitions are also terminal, we can write $Q^\pi(s,a)$ as:

$$Q^\pi(s,a) = \begin{cases} r(s,a) & \text{if } N = 0 \\ \gamma^N r(s_N, a_N) & \text{if } N \text{ is finite} \\ 0 & \text{otherwise.} \end{cases}$$

We promote N to a function $S \times A \to \mathbb{N} \cup \{+\infty\}$, and we define a function $u : S \times A \to \mathbb{R}$ as given in Equation (8).

$$u(s,a) = \begin{cases} r(s,a) & \text{if } N = 0 \\ r\left(s_{N(s,a)}, a_{N(s,a)}\right) & \text{if } N > 0 \text{ finite} \\ 0 & \text{otherwise.} \end{cases} \tag{8}$$

Now we have $\forall (s,a) \in S \times A, Q^\pi(s,a) = \gamma^{N(s,a)} u(s,a)$.

Let R be the finite set of possible reward values.

Therefore values of Q^π are in a set $M = \{\gamma^n r \mid n \in \mathbb{N}, r \in R\}$. Let $M^+ = M \cap \mathbb{R}^{+*}$ be the set of positive values of M. Since $R \subset \mathbb{R}$ is finite, we order all non-zero positive possible rewards in increasing order $r_1, r_2, \cdots r_k$.

Let $M_k^+ = \{\gamma^n r \mid n \in \mathbb{N}, r \in R_k\}$ where $R_k = \{r_1, \cdots, r_k\}$.

We prove the following by recurrence over the number of possible non-zero rewards:

$$H(k) : \exists \nu_k > 0, \forall \delta > 0, \exists \text{ consecutive } b, a \in M_k^+, \delta \nu_k < a - b \text{ and } b < a < \delta$$

Initialization When $k = 1$, $M^+ = \{r_1\gamma^n \mid n \in \mathbb{N}\}$. Let $\nu = \frac{\gamma^2}{1-\gamma}$. Let $\delta > 0$. Let $n = \lfloor \log_\gamma \frac{\delta}{r_1} \rfloor + 1$. We have:

$$\begin{aligned}
\log_\gamma \frac{\delta}{r_1} - 1 < n - 1 \leq \log_\gamma \frac{\delta}{r_1} \\
\log_\gamma \frac{\delta}{r_1} < n < \log_\gamma \frac{\delta}{r_1} + \log_\gamma(1-\gamma) + 2 - \log_\gamma(1-\gamma) \\
\log_\gamma \frac{\delta}{r_1} < n < \log_\gamma \frac{\delta(1-\gamma)}{r_1} + \log_\gamma \nu \\
\log_\gamma \frac{\delta}{r_1} < n < \log_\gamma \frac{\delta\nu(1-\gamma)}{r_1} \\
\frac{\delta\nu(1-\gamma)}{r_1} < \gamma^n < \frac{\delta}{r_1} \\
\delta\nu(1-\gamma)^2 < r_1\gamma^n(1-\gamma) < \delta(1-\gamma) \\
\delta\nu < r_1\gamma^n - r_1\gamma^{n+1} \text{ and } r_1\gamma^n < \delta
\end{aligned}$$

Let $a = r_1\gamma^n \in M^+$ and $b = r_1\gamma^{n+1} \in M^+$. $\delta\nu < a - b$ and $b < a < \delta$ therefore $H(1)$ is verified.

Recurrence Let $k \geq 1$, and assume $H(k)$ is true. Let ν_k be the ν chosen for $H(k)$. Let $\nu_{k+1} = \frac{\nu_k}{2}$. Let $\delta > 0$. Let b_k, a_k a consecutive pair chosen in M_k^+ such that $\delta\nu_k < a_k - b_k$ and $b_k < a_k < \delta$.

Since R_{k+1} contains only one more element than R_k, which is larger than all elements in R_k, we know that there is either one or zero elements $c \in M_{k+1}^+$ that are strictly between a_k and b_k. If $a_k - c < c - b_k$ then let $a_{k+1} = c$ and $b_{k+1} = b_k$, otherwise $a_{k+1} = a_k$ and $b_{k+1} = c$. If a_k and b_k are still consecutive in M_{k+1}^+, then $a_{k+1} = a_k$ and $b_{k+1} = b_k$.

This guarantees that $[b_{k+1}, a_{k+1}]$ as at least half as big as $[b_k, a_k]$. Therefore, $\frac{1}{2}(a_k - b_k) < a_{k+1} - b_{k+1}$, which means that $\delta\nu_{k+1} < a_{k+1} - b_{k+1}$ and $b_k < a_k < \delta$.

Therefore $H(k+1)$ is verified.

This also gives the general expression of ν, valid for all k: $\nu = \left(\frac{\gamma^2}{1-\gamma}\right)^{|R|}$.

Main proof Using the result above, we prove that $Q^\pi(s,a)$ cannot have any non-null derivatives.

Trivially, Q^π cannot have a non-null derivative at a point (s,a) where $Q^\pi(s,a) = q_0 \neq 0$. Indeed, there exists a neighbourhood of $q_0 \in M$ in which there is a single value.

Let $x_0 = (s,a)$ such that $Q(s,a) = 0$. Let v be a vector of the space $S \times A$. Let $f : \mathbb{R} \to \mathbb{R}$ be defined as $f(h) = Q^\pi(x_0 + hv)$. In the following, we show that $\frac{f(h)}{|h|}$ cannot converge to a non-null value when $h \to 0$.

We use the (ϵ, δ) definition of limit. If f had a non-null derivative l at 0, we owould have $\forall \epsilon > 0, \exists \delta > 0, \forall h, |h| < \delta \implies \left|\frac{f(h)}{|h|} - l\right| < \epsilon$.

Instead, we will show the opposite: $\exists \epsilon > 0, \forall \delta > 0, \exists h, |h| < \delta$ and $\left|\frac{f(h)}{|h|} - l\right| \geq \epsilon$.

Using the candidate derivative l and the ν value computed above that only depends on γ and $|R|$, we set $\epsilon = \frac{l\nu}{2}$.

Let $\delta > 0$.

There exists consecutive b, a in M such that $\delta l\nu \leq a - b$ and $b < a < \delta l$.

We set $h = \frac{a+b}{2l}$. Note that $\frac{a+b}{2} < \delta l$ therefore $h < \delta$.

$f(h)$ is in M, but hl is the center of the segment $[b, a]$ of consecutive points of M. Therefore, the distance between $f(h)$ and hl is at least $\frac{a-b}{2}$.

$$|f(h) - hl| \geq \frac{a-b}{2} \geq \frac{\delta l \nu}{2}$$

Since $h < \delta$, $\frac{1}{h} > \delta$.

$$\left| \frac{f(h)}{h} - l \right| \geq \frac{l\nu}{2} = \epsilon.$$

□

E Implementation details

Here is the complete rollout and training algorithm, taken from the Spinup implementation of DDPG.

```
Result: Policy $\pi_\psi$, number of steps before success
$\pi_\psi, Q_\theta \leftarrow$ Xavier uniform initializer
env_steps $\leftarrow 0$
for $t \leftarrow 1$ to $10000$ do
    $a \leftarrow \pi_\psi(s)$
    if $rand() < 0.1$ then
        $a \leftarrow$ rand$(-0.1, 0.1)$
    end
    Step the environment using action $a$, get a transition $(s, a, r, t, s')$
    Store $(s, a, r, t, s')$ in the replay buffer
    env_steps $\leftarrow$ env_steps $+ 1$
    if $t = 1$ or env_steps $> N$ then
        Reset the environment
        for $k \leftarrow 1$ to env_steps do
            Sample a mini-batch of size 100 from the replay buffer
            Train $\pi_\psi$ and $Q_\theta$ on this replay buffer with losses (1) and (2).
        end
        env_steps $\leftarrow 0$
    end
    if $t$ mod $1000 = 0$ then
        if last 20 episodes were successes then
            Terminate the algorithm, and return success_after_steps= $t$.
        end
    end
end
```