Sorbonne Université

Master ANDROIDE 1ST Year

P-Androide : Study of Swimmer environment

Visualizing objective function geometry using random perturbations

Authors :

Coline LACOUX
Ryan OHOUENS
Maël FRANCESCHETTI

Professor:

Olivier SIGAUD

April 2020

Sorbonne Université

Abstract

Reinforcement learning algorithms generally gives better performance than evolutionary algorithms on MuJoCo environments. The Swimmer benchmark is a special case: gradient-based reinforcement learning algorithms like TD3 fails, where evolutionary algorithms like the Cross-Entropy Method (CEM) performs very well.

The hypothesis of the deceptive gradient is often made on this subject. In fact, several mechanisms are behind TD3's failure, and the goal of this article is to implement tools to understand the disappointing performance of TD3 on SWIMMER.

To understand the reason for the failure of TD3, one must understand the SWIMMER environment and the geometry of its objective function, but also be able to study the behavior of TD3. The tools proposed make it possible to respond to these needs, and ultimately to understand several mechanisms responsible for the failure of TD3.

Table of content

1 Introduction

Despite recent outstanding successes [13, 6, 14, 2], Reinforcement Learning (RL) algorithms are still far from being immediately applicable to any sequential decision problem. Instead, it is generally necessary to spend a lot of time hand tuning many hyper-parameters before eventually getting satisfactory performance. Even worse, there are problems for which the best performance one can get with deep RL algorithms is far from what other optimization techniques like evolutionary methods can bring. This is the case for instance of the SWIMMER benchmark [3], where several authors have reported better performance with simple evolution strategies like the Cross-Entropy Method (CEM) [9, 12, 4] than with deep RL algorithms [11, 15]. The reasons for such failures remain elusive.

A source of difficulty in understanding failures or in tuning hyper-parameters is the lack of adequate visualization tools. The gradient descent performed by deep RL methods is a local process taking place in a very high dimensional parameter space and which is inherently prone to convergence to local optima, hence visualizing what is happening is difficult. Tools like T-SNE [7] have been shown to be helpful in discrete action domains [16], but it still requires effort intensive design choices to be used properly and the demonstrated use cases do not seem to generalize easily to the continuous action domain.

In this paper, we provide a visualization tool which can be use straightforwardly and helps understanding the gradient landscape in the parameter space and the trajectory of a deep RL algorithm in that space. We build on previous work dedicated to understanding the impact of entropy regularization [1], but the visualizations we provide are more intuitive.

After presenting the tool itself, we illustrate its use by providing an explanation of why a standard deep RL algorithm, TD3, fails on SWIMMER. Based on this understanding, we fix the corresponding issues and obtain for the first time competitive results on SWIMMER with TD3.

2 Approach

The general idea of our method consists in visualizing in a two dimensional picture the performance of various policy parameters in the neighborhood of the current policy parameters. The number of dimensions of the policy parameter space being much higher than two, we need to draw a large set of directions in the parameter space and visualize the performance of policies in these directions as a line in our 2D plot. The way to draw a set of random directions is explained in Section 2.1.

2.1 Random perturbations

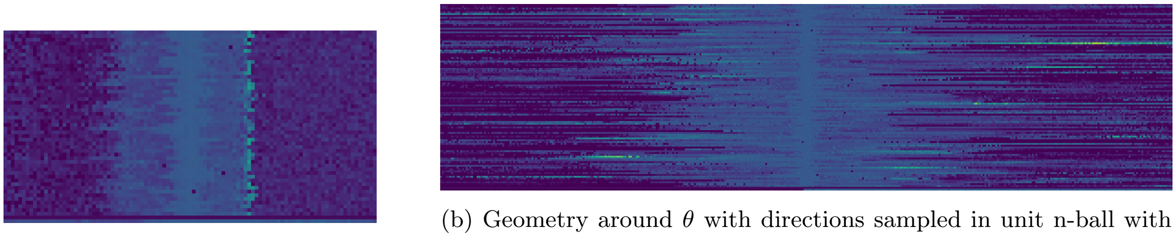
Let us call $J(\theta)$ the performance of objective function of a policy of parameters θ. Using random perturbation makes it possible to characterize the geometry of $J(\theta)$ around θ. Indeed, it gives a global indication about the geometry (flat region, slope, local maximum, local minimum). From this geometry one can even measure the curvature of $J(\theta)$ along the optimization trajectory, as shown in [1]. In order to visualize more precisely the geometry of $J(\theta)$ around θ, in contrast to [1], we chose to represent it visually.

Visualization along a direction: The principle is to sample directions, and for each direction d to evaluate $J(\theta + d \times \alpha)$ and $J(\theta - d \times \alpha)$ by varying α. Each sampled direction d is associated with a horizontal line of the visualization, the central point is the initial actor θ. Each point on the right of the central point represents an actor $\theta + d \times \alpha$, and each point on the left represents an actor $\theta - d \times \alpha$.

The points are ordered like α values: relative to the center, the first point on the right corresponds to $\theta + d \times \alpha_1$, the second to $\theta + d \times \alpha_2$. Symmetrically, the first point on the left corresponds to $\theta - d \times \alpha_1$, the second to $\theta - d \times \alpha_2$.

Sampling directions: Uniform sampling in (0,1) is not suitable for sampling varied and unitary directions, see Figure 1. Direction are sampled randomly in the unit ball of n dimensions, where n is the number of dimensions of the parameter space. For that, we use Muller's method, introduced in [10] and described in [5]. The method is as follows:

1. sample U following normal n-dimensional distribution: $U \sim \mathcal{N}_n(0_n, 1_n)$
2. take the uniform distribution on the unit n-sphere: $S_n = \frac{U}{\|U\|}$
3. sample r following uniform distribution on (0,1) interval: $r \sim \text{random}(0, 1)$
4. get a sample following the uniform distribution on the unit n-ball: $x = r^{\frac{1}{n}} \times S_n$



(a) Geometry around θ with directions sampled uniformly in (0,1). We observe that all the sampled directions give a similar result.

(b) Geometry around θ with directions sampled in unit n-ball with Muller's method. We observe that the directions give very different results than uniform sampled directions in (0,1). This method highlights improving slopes which were not visible with a uniform sampling.

Figure 1: Comparing uniform and Muller sampling around the same θ on Swimmer.

Parameter tuning The α values goes from 0 to max_alpha with a step of $step\_alpha$. In order to explore far enough in each sampled direction, without going too far, it is necessary to configure the maximum value taken by the alpha parameter depending on the environment. We observe for example on the SWIMMER environment a Euclidean distance between two consecutive actors which can go up to 20 but is usually below 10, if we consider the actors every 1000 iterations (cf. Figure 2). We can then choose for this environment a parameter value max_alpha of 10 or 20 to visualize in a possible horizon. We can choose a higher value, for example $max\_alpha = 100$ to study the geometry further.

Depending on the settings, with this display we get a rapid or a more detailed and more precise evaluation of the local geometry of $J(\theta)$, see Figure 19 for the comparison of different sets of parameters.

The value of the $step\_alpha$ parameter can be reduced to have greater precision, or increased for a faster overview. A standard value of 1 gives a good result since we we sample directions on the unit n-ball, smaller values will give better accuracy.

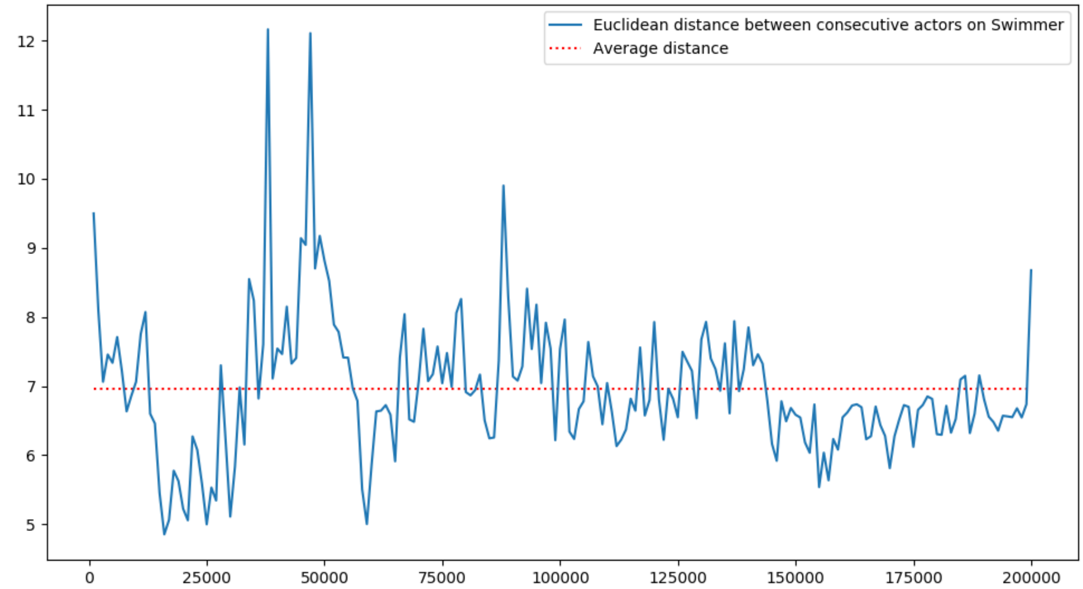


Figure 2: Distance between consecutive actors on SWIMMER, considering actors every 1000 steps (average over 4 runs).

2.2 Gradient direction

To be able to compare several successive actors, we initially sample the directions applied for all the visualizations carried out during an execution.

We display the direction followed by the algorithm between two successive actors. This direction is the difference between the current actor and the previous one: $\theta_t - \theta_{t-1}$. To evaluate the score of the actors on the direction followed in the same way as for the directions drawn in the n dimensional unit ball, the direction followed is reduced to a unit vector by dividing it by its Euclidean distance to the origin. The direction followed is the line at the bottom on the visualization, separated from the other lines by a dark line.

We also used another visualization grouping the directions of the estimated gradient followed by TD3 at each iteration. This allows us to globally view the route followed. The dot product indicator makes it possible to determine whether an abrupt change in direction has been encountered (value very close to 0 or negative).

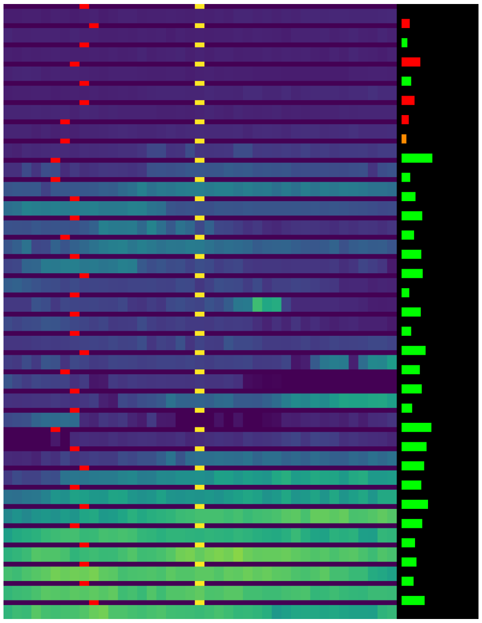


Figure 3: Each line represents the estimated gradient followed by TD3 between two steps of the algorithm. The colored rectangles on the right indicate the value of the dot product between two consecutive followed directions: red for a negative value, orange for a value very close to 0 and green for a positive value. The longer the rectangle, the higher the value. The maximum size represents a dot product of 15. Yellow markers indicates the position of the current parameters, and red markers indicates the position of the precedent parameter.

3 Understanding SWIMMER

Several authors report an unexpected performance gap on the SWIMMER benchmark [3] between using simple evolutionary methods and deep RL algorithms [11, 15].

With evolutionary methods such as the Cross-Entropy Method (CEM) [9, 12, 4], one can easily obtain a performance over 300. The best performance we found in the literature is 362, using Augmented Random Search [8].

By contrast, performance barely reaches 100 with deep RL methods, with a strong local minimum around 40. The goal of this section is to understand the reasons behind this performance gap.

3.1 Deceptive gradient

We see in Figure 4 that CEM, which usually reach the optimum in less than 300000 steps, does not reach the optimum after 500000 steps if it starts from an actor obtained with TD3 after 200000 steps. It seems that TD3 is moving away from the optimum, which is why CEM needs more time to reach the optimum.

Following the gradient would not then lead to the optimum. We could assume that this is therefore a deceptive gradient problem, but we will see later that it is in fact several different mechanisms which are responsible for the failure of TD3 on Swimmer.

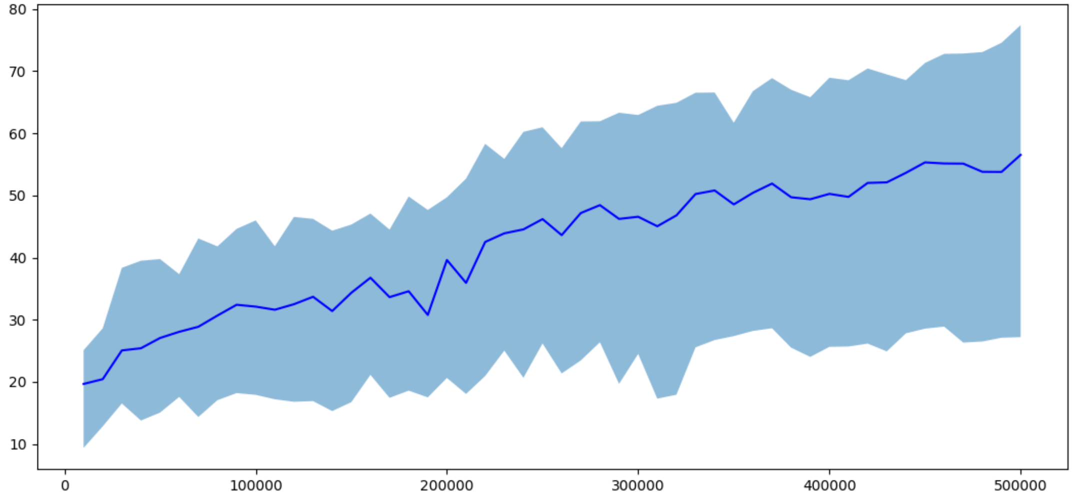


Figure 4: Performance curves of CEM starting from actors obtained with TD3 after 200000 steps.

3.2 Impact of the discount factor

Visualizing the behavior of the SWIMMER environment using the MUJOCO rendering tool shows that agents performing over 300 behave in a snake-like way along the x-axis, whereas agents performing around 40 quickly move their tail towards higher x value into a "U" body shape and then just drift on the ground based on the initial rightward impulse. We call the former the "*snake*" strategy and the latter the "*U-shape*" strategy.

See videos of these actors here: `https://72indexdescartes.wixsite.com/swimmer`.

Looking more closely at the velocities along the x-axis in Figure 5, one can see that that the U-shape strategy produces a higher initial velocity than the snake strategy, but then gets stuck into an inefficient configuration where the agent cannot keep producing high velocity. The snake strategy alternates positive and negative algebraic velocities, but gives a better performance than the other strategies if we consider the whole episode.

This suggests that a deep RL agent finds it more optimal to maximize velocity in the first steps of the evaluation without caring much about the velocity at the final steps, which is not the case of evolutionary agents.

A simple explanation for this phenomenon lies in the role of the discount factor in deep RL agents. In RL, the agents usually use a discount factor to consider more the rewards they immediately get than the rewards they may get after more time steps. This is not the case in evolutionary methods, where the performance along a trajectory is considered as a whole. The standard discount factor in deep RL agents on MUJOCO benchmark is $\gamma = 0.99$, and the performance on SWIMMER is measured over 1000 steps.

It happens that $0.99^{1000} = 0.00004317124$, thus the last steps are discount by a factor 0.00004317124 with respect to the first step. This is enough to explain that deep RL agents maximize their velocity in the first steps and disregard what happens next.

If we take these coefficients into account, we obtain the weighted cumulative speed curves of Figure 6. We note that the behavior of the "*U*" or the "*tryhard*" are preferred to the behavior of the "*snake*", which nevertheless gives a better score by episode. If we then use $\gamma = 0.99999$ in place of 0.99, the "*snake*" is now preferred, see Figure 7.

If we set $\gamma = 1.0$, this phenomenon does not happen anymore.

Using TD3 with $\gamma = 1.0$, we start getting a few agents which perform over 359.

Thus, as a conclusion of this section, the SWIMMER benchmark is characterized by a strong local minimum where maximizing velocity in the first steps and disregarding what happens next may outperform longer term strategies depending on the discount factor.

However, tuning the discount factor is not enough. With this change, we do not get efficient agents performing over 300 at all attempts. A second mechanism is at play, that we investigate below.

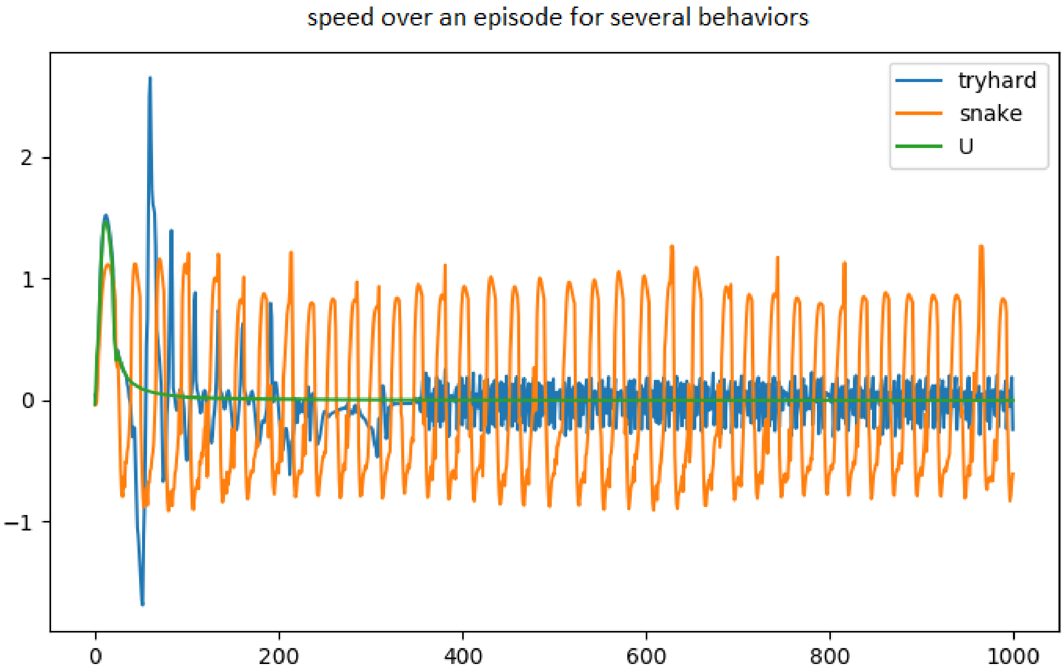


Figure 5: Algebraic speed on x-axis during an episode for different behaviors on SWIMMER.

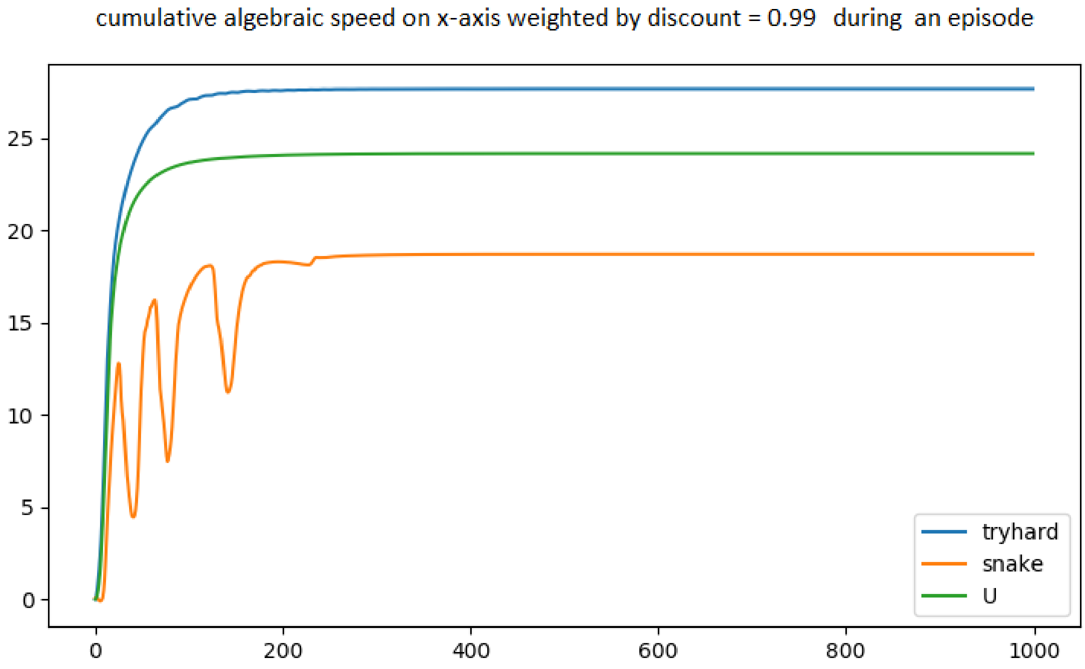


Figure 6: Cumulative algebraic speed on x-axis weighted by $discount^{time}$ with $discount = 0.99$ during an episode for different behaviors on SWIMMER.

3.3 Disturbances

By visualizing the estimated gradient followed by TD3 and using the scalar product as an indicator of changes in direction, we note that TD3 sometimes successively makes radical changes in direction (dot product close to 0 or negative). These sudden changes in direction occur mainly in flat or noisy areas, where the scores are very irregular. See Figure 8a and Figure 8b. It seems that these areas disturbs TD3. We better see these flat and noisy area with

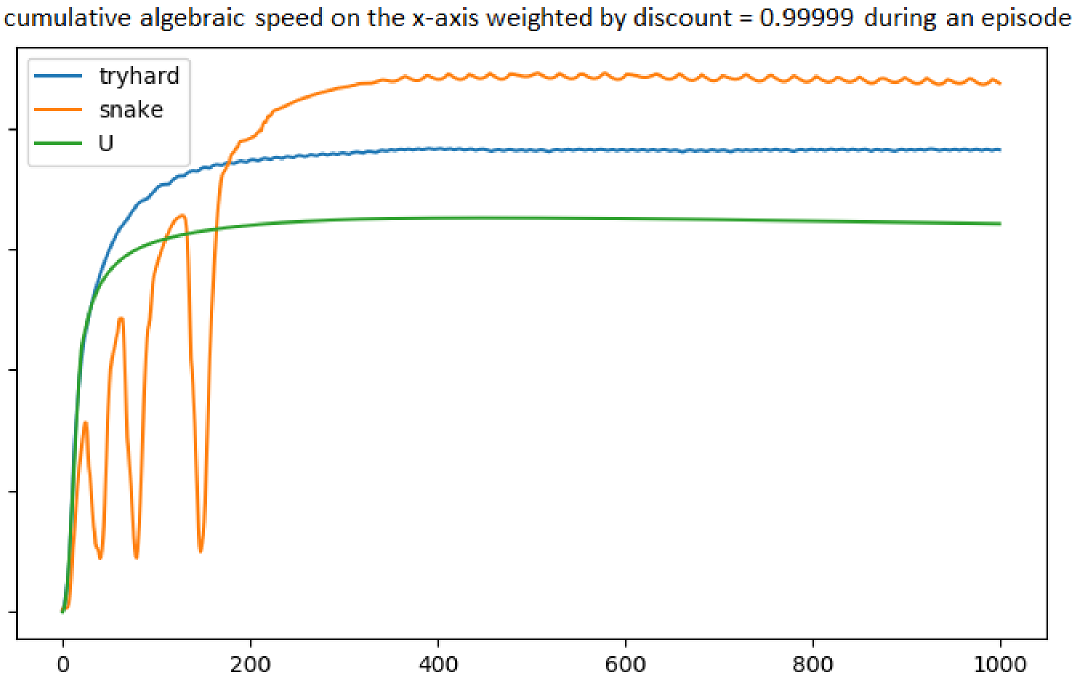
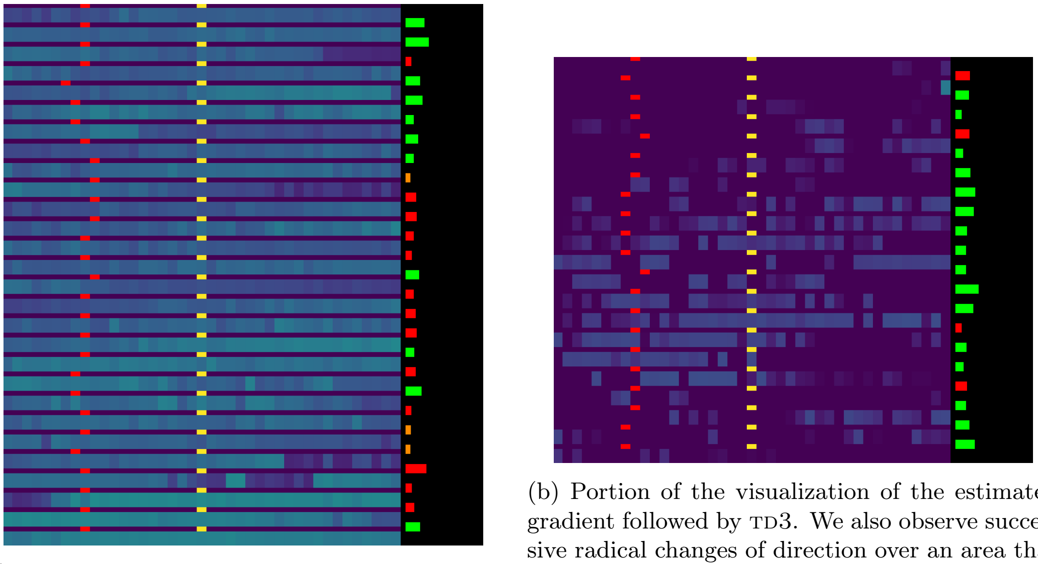


Figure 7: Cumulative algebraic speed on x-axis weighted by $discount^{time}$ with $discount = 0.99999$ during an episode for different behaviors on SWIMMER.

the random perturbation method, see Figure 9.



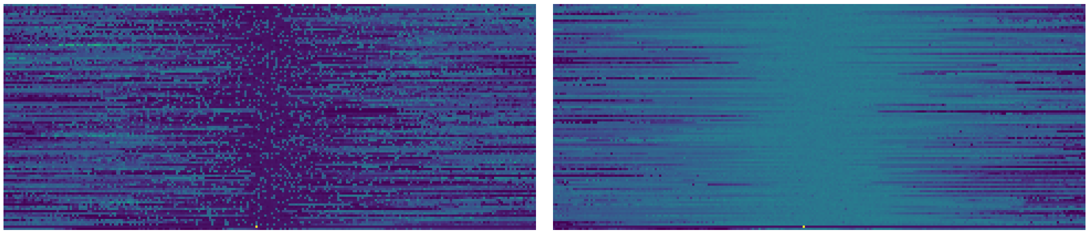
(a) Portion of the visualization of the estimated gradient followed by TD3. We observe successive radical changes of direction over an area that seems fairly flat

(b) Portion of the visualization of the estimated gradient followed by TD3. We also observe successive radical changes of direction over an area that seems noisy and that leads to another fairly flat area.

Figure 8: Visualisation of perturbations on estimated gradient.

More generally, it seems that TD3 passes successively through fairly flat and very noisy areas in places, separated by areas with very poor scores. Sometimes, TD3 is perturbed by these areas and make successive radical changes of direction.

Out of a large number of TD3 executions, the majority of executions did not lead to a satisfactory score.

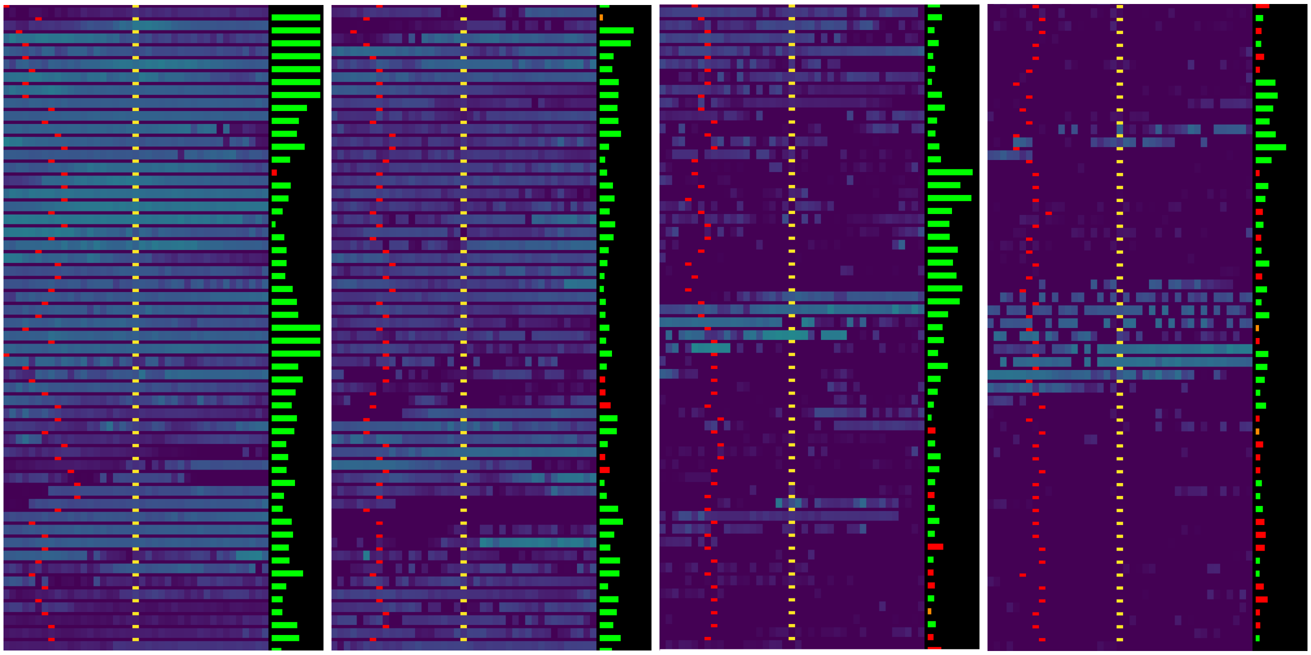


(a) Visualization obtained on SWIMMER environment with *max_alpha* = 120 and *step_alpha* = 1, on 100 sampled directions. There is a local minimum appearance but with a very noisy area around the actor.

(b) Visualization obtained on SWIMMER environment with *max_alpha* = 120 and *step_alpha* = 1, on 100 sampled directions. There is a fairly wide flat area around the actor, beyond which there are mostly descending slopes.

Figure 9: Visualisation of flat and noisy areas on SWIMMER.

However, a few rare executions have produced a correct score. We have studied the estimated gradient followed by a successfully TD3, as can be seen in Figure 11. Comparing it to a failed execution of TD3 on Figure 10, we observe a similar behavior: sudden changes of direction which are more and more numerous as and when. However, the failed execution makes a first radical change of direction from the start of the execution, unlike the successful execution. In addition, the changes of direction on successful execution correspond to a gradual rise towards the optimum, while on failed execution, these radical changes of direction seem to be due to the small variations in score encountered which disturb TD3.



(a) Part 1

(b) Part 2

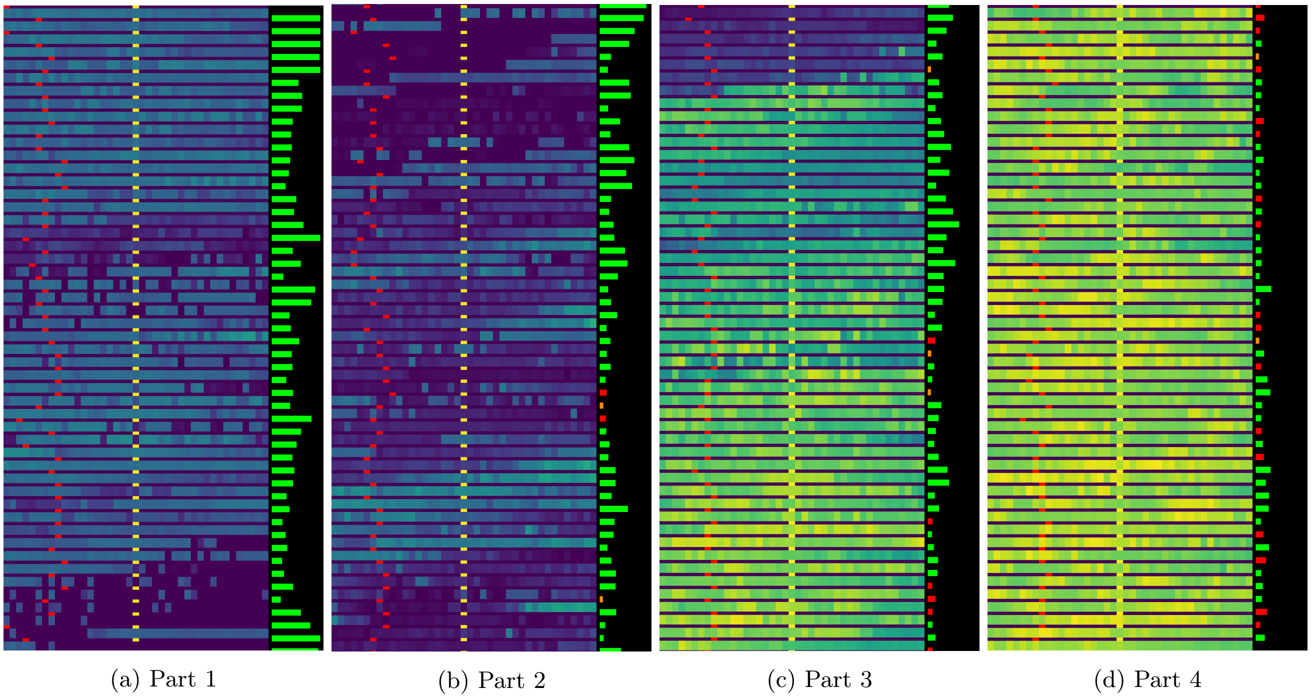
(c) Part 3

(d) Part 4

Figure 10: Execution of TD3 on SWIMMER which failed, we observe multiple radical changes of direction

To conclude this section, in view of the gradient followed, TD3 passes through flat and noisy areas. TD3 seems to be perturbed by these areas but we're not sure if it's the cause or the consequence of its failure. TD3 behaviour in failed executions has only minor differences with successfully executions. The fact that certain executions rarely succeed may be due to initialization (certain starting points would be advantageous), or else to the random actions used to train TD3 the first time steps (up to *start_steps* iterations). We will study those two points below.

The behavior of TD3 is very different on HALF-CHEETAH, see Figure 12.



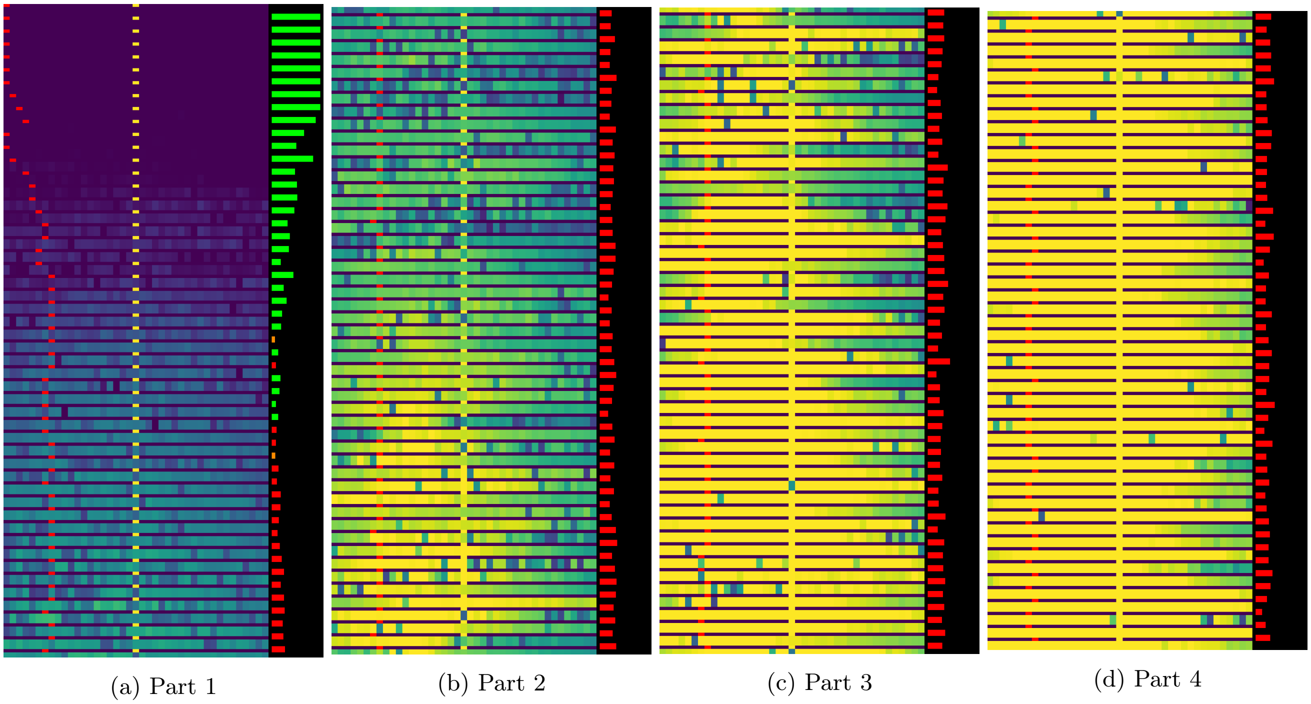
(a) Part 1

(b) Part 2

(c) Part 3

(d) Part 4

Figure 11: Execution of TD3 on SWIMMER which reached a score of more than 200. We notice that the first radical changes of direction are later than on the executions which fail.



(a) Part 1

(b) Part 2

(c) Part 3

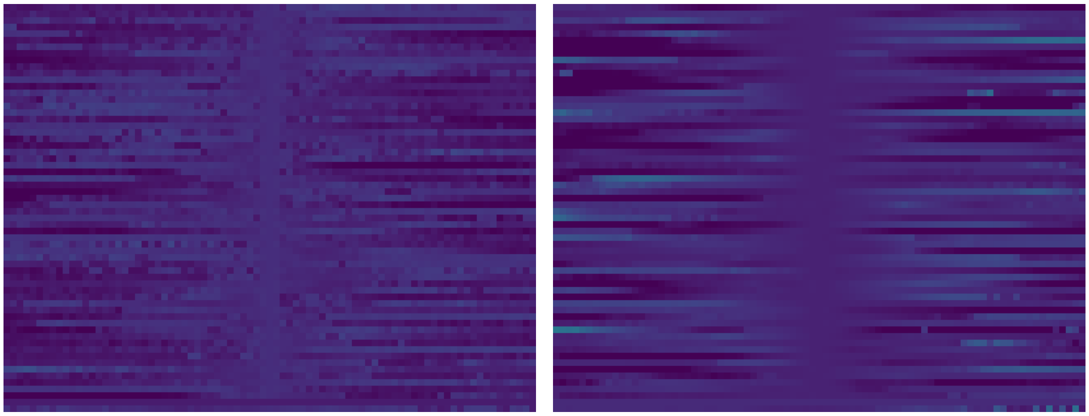
(d) Part 4

Figure 12: TD3 run on HALF-CHEETAH over 200000 steps. It seems after going out initial area by going forward (highly positive dot product values), TD3 advances in zigzag (consecutive negative dot product values).

3.4 Impact of initialization

By visualizing the geometry of the objective function we observe an area with very low scores and some slightly degrading slopes. If we compare with the starting points of Half-Cheetah, we notice that the starting point of SWIMMER has a fairly noisy geometry and does not present a frankly improving slope, see Figure 13. Looking at the gradient followed by TD3 on several executions and by performing a visualization of the geometry by random perturbation on the crossed zones, we note that the starting zone of SWIMMER is surrounded by low score flat zones separated by local minima with very low scores and noisy areas, see Figure 17.

TD3 must therefore explore enough at the start to be able to reach improving slopes.



(a) Swimmer

(b) Half-Cheetah

Figure 13: Geometry around a start point of TD3 using random perturbation method: the geometry around the starting point on SWIMMER is more noisy and has less improving slopes than on Half-Cheetah.

In view of the gradient followed by TD3 on SWIMMER (see Figure 11 and Figure 10), and compared to the gradient followed on Half-Cheetah (see Figure 12), the starting point on SWIMMER seems quite far from the optimum zone: a large zone of low scores punctuated by noisy or flat regions separates the departure zone from the optimum.

This hypothesis seems confirmed by the fact that the modified TD3 executions having reached the optimum make a long way before reaching the slope leading to the optimum, see Figure 18.

3.5 Exploration

In view of the geometry around the starting point (very flat, noisy areas ...), it seems important that TD3 explores enough to leave this area and find a slope leading to the optimum.

In order to improve exploration at the start of learning, TD3 plays actions sampled uniformly from among the valid actions up to *start_steps* iterations. The update of TD3 takes actions randomly from the replay buffer to constitute the mini-batch. At the beginning of a TD3 run, up to *start_steps*, the replay buffer consists only of random actions sampled uniformly on possible valid actions, the critics are therefore gradually updated on the basis of random actions. Increasing the value of *start_steps* did not improved performance. See the performances of TD3 for different values of *start_steps* in Figure 21, Figure 22 and Figure 23 in the appendices.

We have observed that starting an execution while keeping the replay buffer of a previous execution makes it possible to reach optimal scores in the majority of cases, even if the replay buffer comes from a failed execution that did not reach a correct score, see Figure 14.

In this situation, the replay buffer consists for the most part of actions that are not resulting from the random policy. The mini-batch is therefore likely to contain a majority of actions resulting from the TD3 policy.

3.6 Removing training before start_steps

Noting that using a pre-filled replay buffer gives good results but that increasing *start_steps* does not give good results, we hypothesized that learning only on random actions at the beginning of TD3 is going in the wrong direction irreversibly.

By not training TD3 on the random actions of the first iterations but by nevertheless filling the replay buffer with these, TD3 manages to reach correct scores, see Figure 15 and Figure 16.

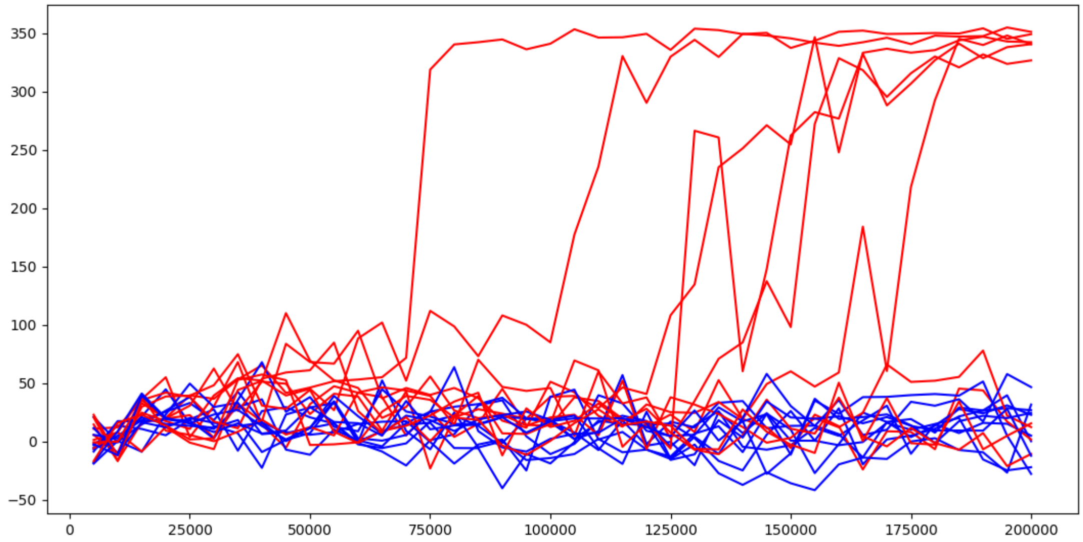


Figure 14: Performance value of TD3 every 1000 iteration. TD3 executions in blue all failed. The executions in red each kept the replay buffer of one of the executions in blue: they achieved good performance for the majority. There is 10 runs of each. We used discount = 1.

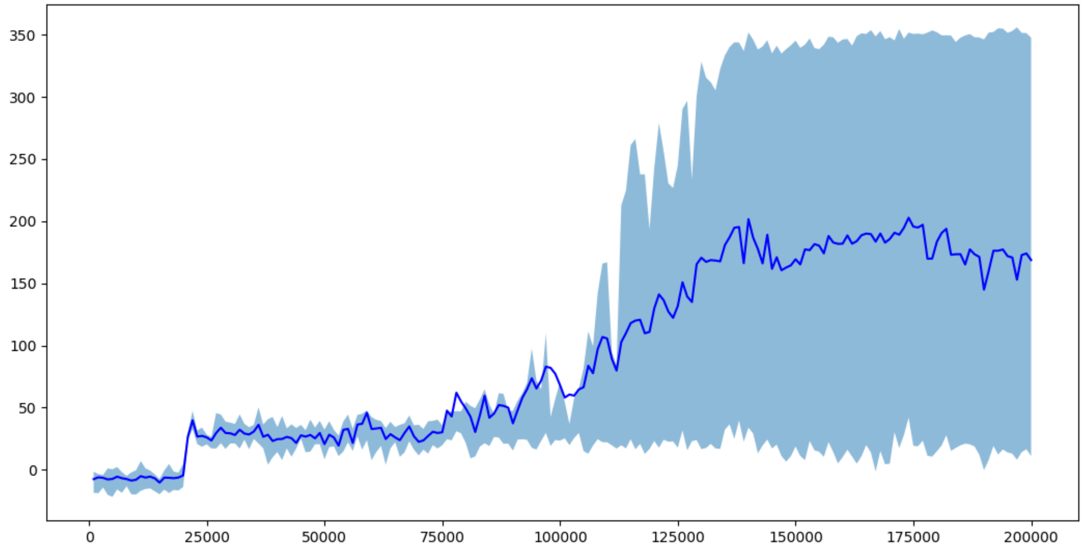


Figure 15: Performance of TD3 with discount = 1, *start_steps* = 20000 and without learning before *start_steps*. Mean and quantiles over 19 runs.

We observe in Figure 17 that at the start of the course of a standard TD3 learning session, the value of the dot product decreases until TD3 makes a first abrupt change of direction, visible to the very small and orange dot product indicator, which corresponds to a value close to zero. TD3 then passes through areas of low and slightly noisy scores. TD3 makes many abrupt changes in the second part of the execution, without reaching a better area.

In comparison, we observe in Figure 18 that the behavior of the modified TD3 is different: it makes its first sudden change of direction much later, and it achieves better scores from the start, although it also passes through noisy or flat areas. The first negative dot product is before reaching the optimum zone. TD3 makes several changes

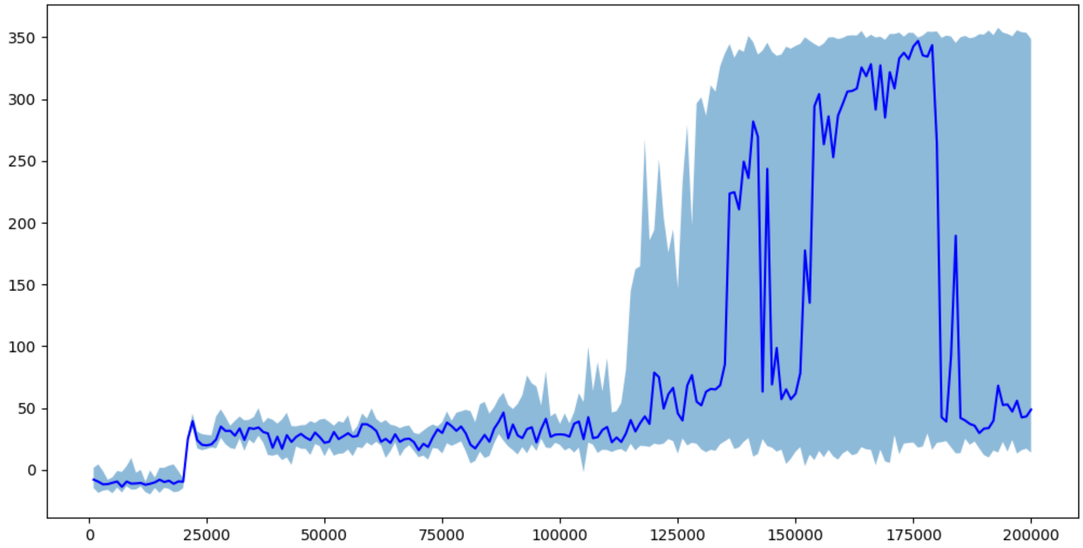
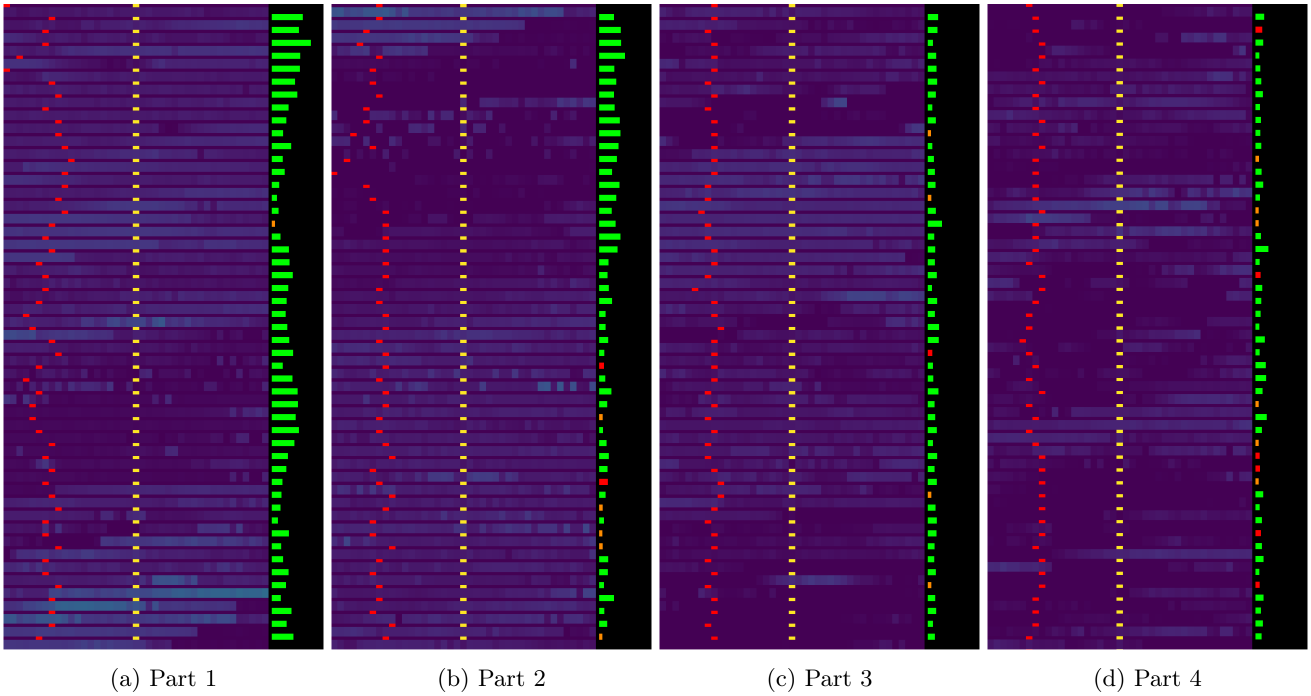


Figure 16: Performance of TD3 with discount = 1, *start_steps* = 20000 and without learning before *start_steps*. Median and quantiles over 19 runs. The majority of executions reach the optimum, but the performance suddenly falls unexplained.

of direction within this maximum.


(a) Part 1

(b) Part 2

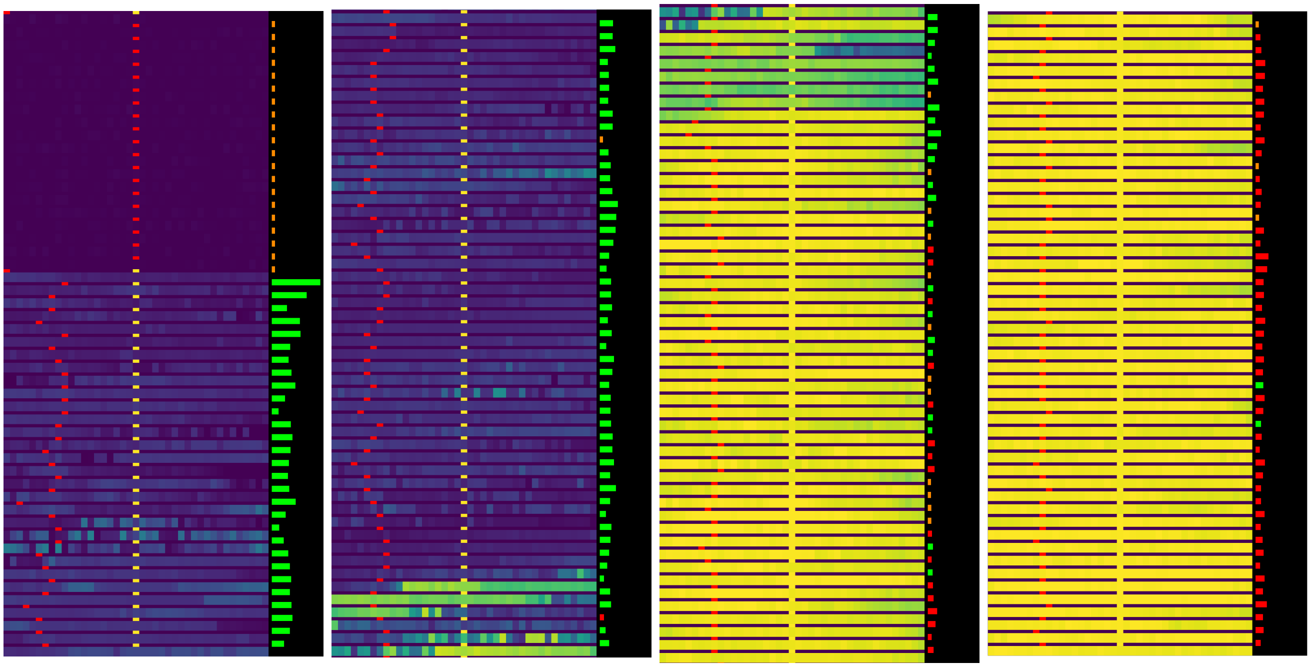
(c) Part 3

(d) Part 4

Figure 17: TD3 followed gradient on SWIMMER.

4 Conclusion

Among the MuJoCo benchmarks, SWIMMER is a special case, being the only one on which reinforcement learning algorithms achieve very poor performance compared to evolutionary approaches.



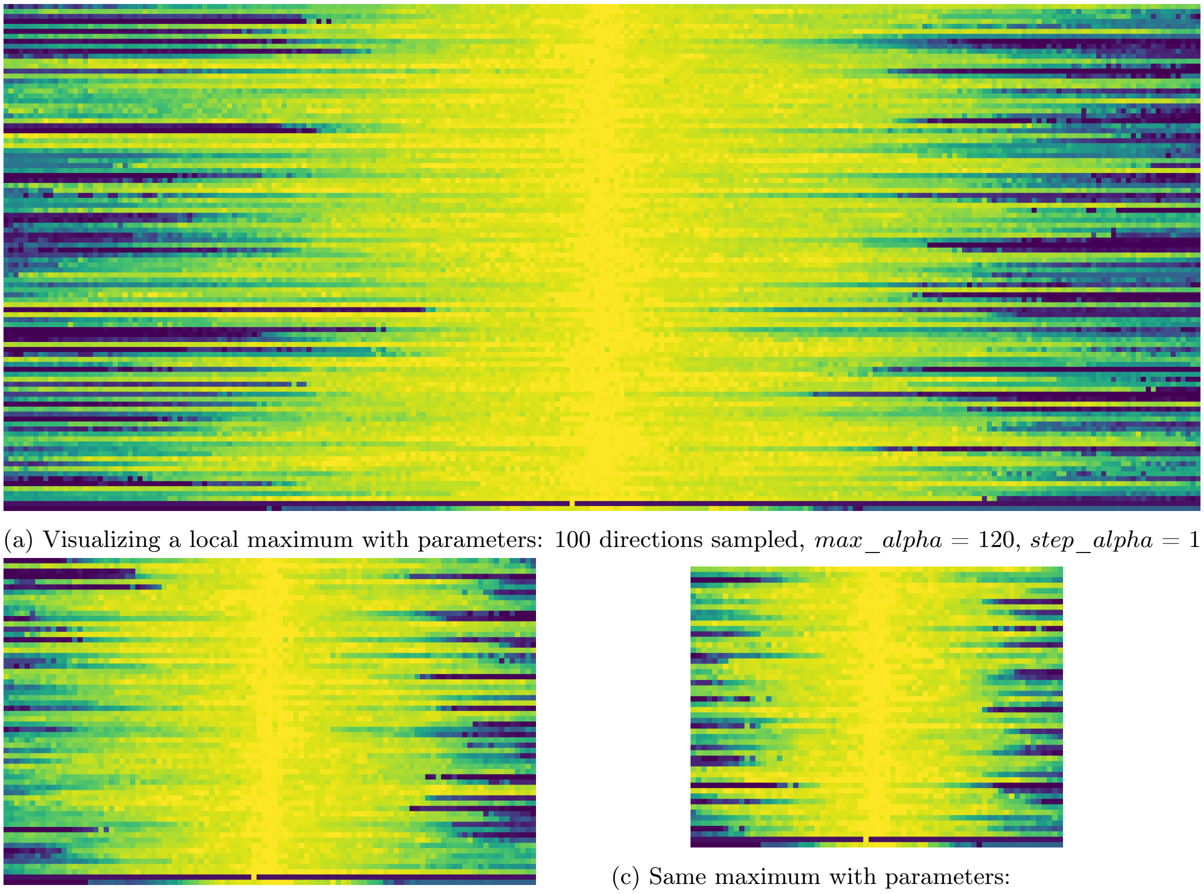
(a) Part 1 (b) Part 2 (c) Part 3 (d) Part 4

Figure 18: TD3 followed gradient without training up to *start_steps* iterations, from the same initial parameters than Figure 17.

The tools developed in this project made it possible to understand certain aspects of this environment and to identify several hypotheses: with an unfavorable starting point and many complex zones to cross to reach the optimum, it is difficult for a standard TD3 agent to achieve satisfactory performance on SWIMMER. The modification of the discount parameter and the withdrawal of the TD3 updates before $start_steps$ now make it possible to achieve satisfactory performance. However, the fall in scores after reaching the optimum remains unexplained and the impact of updates at the start of TD3 from the random policy remains unclear.

More generally, the tools developed here provide important information that can help understand the behavior of gradient-based reinforcement learning algorithms.

5 Appendices



(a) Visualizing a local maximum with parameters: 100 directions sampled, $max\_alpha = 120$, $step\_alpha = 1$

(b) Same maximum with parameters:
60 directions sampled, $max\_alpha = 100$, $step\_alpha = 2$

(c) Same maximum with parameters:
50 directions sampled, $max\_alpha = 100$, $step\_alpha = 3$

Figure 19: Comparing visualization of a local maximum with different parameters

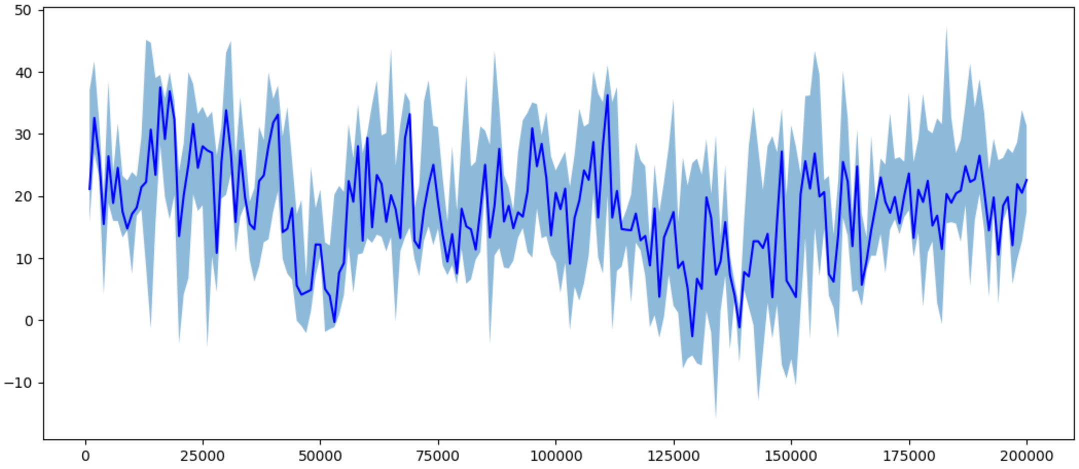


Figure 20: TD3 performance observed on SWIMMER with default parameters, average and quartiles over 10 runs.

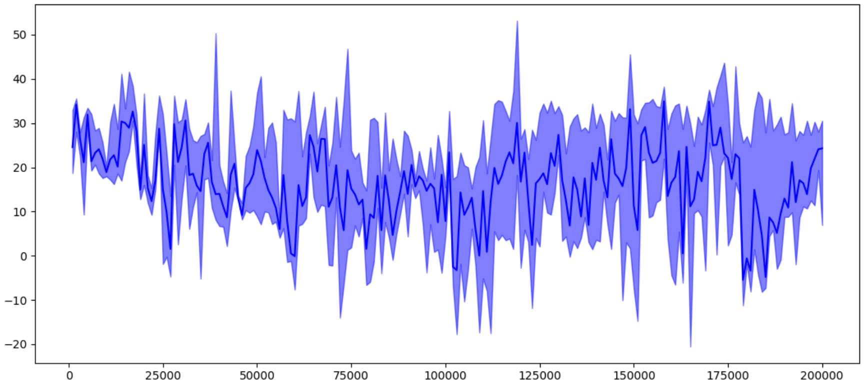


Figure 21: Performance of TD3 with discount = 1, *start_steps* = 10000 over 10 runs.

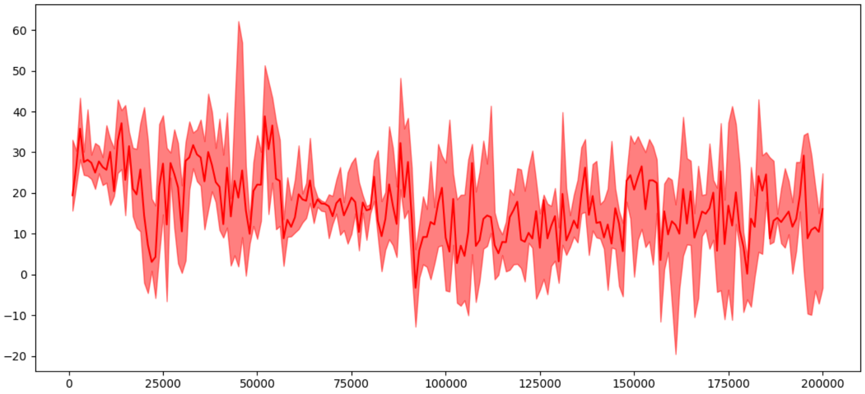


Figure 22: Performance of TD3 with discount = 1, *start_steps* = 20000 over 10 runs.

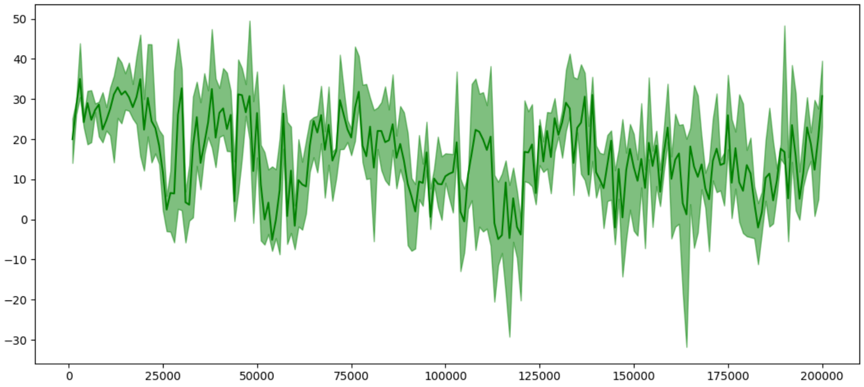


Figure 23: Performance of TD3 with discount = 1, *start_steps* = 50000 over 10 runs.

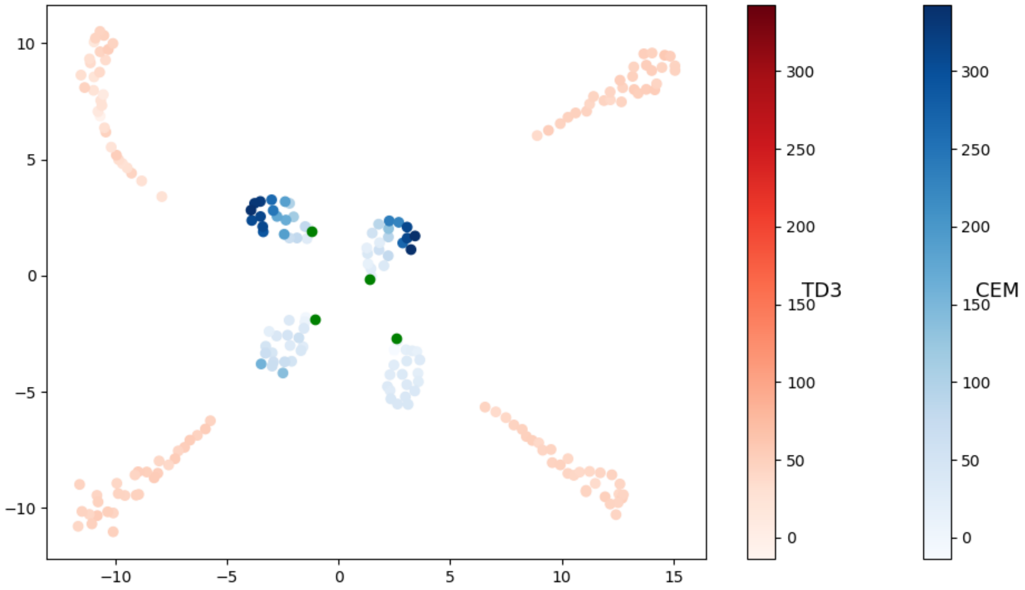


Figure 24: Result obtained with T-SNE on CEM and TD3 actors obtained all along the executions from the same start points (in green). We can't interpret these results due to the way T-SNE works, but TD3 seems to go far away the optimum zone.

References

[1] Zafarali Ahmed, Nicolas Le Roux, Mohammad Norouzi, and Dale Schuurmans. Understanding the impact of entropy on policy optimization. In *International Conference on Machine Learning*, pages 151–160, 2019.

[2] Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

[3] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[4] P-T. De Boer, D.P Kroese, S. Mannor, and R.Y. Rubinstein. A tutorial on the cross-entropy method. *Annals of Operations Research*, 134(1):19–67, 2005.

[5] Radoslav Harman and Vladimír Lacko. On decompositional algorithms for uniform sampling from n-spheres and n-balls. *Journal of Multivariate Analysis*, 101, nov 2010.

[6] Max Jaderberg, Wojciech M Czarnecki, Iain Dunning, Luke Marris, Guy Lever, Antonio Garcia Castaneda, Charles Beattie, Neil C Rabinowitz, Ari S Morcos, Avraham Ruderman, et al. Human-level performance in 3d multiplayer games with population-based reinforcement learning. *Science*, 364(6443):859–865, 2019.

[7] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[8] Horia Mania, Aurelia Guy, and Benjamin Recht. Simple random search provides a competitive approach to reinforcement learning. *arXiv preprint arXiv:1803.07055*, 2018.

[9] S. Mannor, R. Y. Rubinstein, and Y. Gat. The cross-entropy method for fast policy search. In *Proceedings of the 20th International Conference on Machine Learning*, pages 512–519, 2003.

[10] Mervin E. Muller. A note on a method for generating points uniformly on n-dimensional spheres. *Commun. ACM*, 2(4):19–20, April 1959.

[11] Aloïs Pourchot and Olivier Sigaud. Cem-rl: Combining evolutionary and gradient-based methods for policy search. *arXiv preprint arXiv:1810.01222*, 2018.

[12] R. Y. Rubinstein and D. P. Kroese. *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation, and Machine Learning.* Springer-Verlag, 2004.

[13] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

[14] Josh Tobin, Lukas Biewald, Rocky Duan, Marcin Andrychowicz, Ankur Handa, Vikash Kumar, Bob McGrew, Alex Ray, Jonas Schneider, Peter Welinder, et al. Domain randomization and generative models for robotic grasping. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3482–3489. IEEE, 2018.

[15] Youngchul Sung Whiyoung Jung, Giseung Park. Population-guided parallel policy search for reinforcement learning. In *International Conference on Learning Representations*, 2020.

[16] Tom Zahavy, Nir Ben-Zrihem, and Shie Mannor. Graying the black box: Understanding DQNs. In *International Conference on Machine Learning*, pages 1899–1908, 2016.