ACCIDENTAL EXPLORATION THROUGH VALUE PREDICTORS

Anonymous authors
Paper under double-blind review

ABSTRACT

Infinite length of trajectories is an almost universal assumption in the theoretical foundations of reinforcement learning. Of course, in practice this is never the case. In this paper we examine a specific result of this disparity. We focus on the case where the finiteness of trajectories also makes the underlying process to lose the Markov property. This causes the standard state value estimators to become biased, which in turn manifests as a vastly different learning dynamic when algorithms use value predictors.

We investigate these claims theoretically for a one dimensional random walk and Wiener process, and empirically on a number of simple environments. We use GAE as an algorithm which uses a value predictor and compare it to a plain policy gradient.

In practice all reinforcement learning environments produce only trajectories of finite length. In the case of computer simulations this is sometimes hard-coded in the environment, but even when it is not we have to introduce a cutoff to apply most interesting algorithms. If the information about the limited number of steps is not contained in the observations the environment generates, then the environment usually will not have the Markov property. For many algorithms this property is required for various correctness theorems and its lack introduces very strong biases.

At its most basic, the phenomenon can be seen if we assume that all rewards have the same sign and similar order of magnitude. For a fixed policy the unbiased value estimator for a specific state is

$$V(s) = \sum_{i=t}^{\infty} \gamma^{i-t} r_i \tag{1}$$

where t is the time of occurrence of s, γ is the discount factor and r_i is the reward at time i. In this paper we average the above over multiple occurrences of the same state in a given trajectory, i.e. we use an Every-visit Monte Carlo estimator (Singh & Sutton, 1996). This is known to introduce a bias, but this bias turns out to be significantly weaker than the effects we investigate. What is more, the bias gets weaker as we average over multiple trajectories, while the one we investigate sustains a strong influence.

In the case of finite trajectories of length T we cannot sum to infinity. The usual choice is to assume that all rewards become 0 after time T

$$\forall_{t>T}\ r_t = 0 \tag{2}$$

that is, the sum in equation 1 becomes finite

$$V(s) = \sum_{i=t}^{T} \gamma^{i-t} r_i. \tag{3}$$

If the length of a trajectory is artificially constrained, the assumption in equation 2 is usually not justified. In particular, the estimator from equation 3 often has values with lower absolute value for states encountered closer to the end of a trajectory. Any agent depending on value prediction will therefore prefer certain states, and this preference is correlated with the average time of occurrence of the state in a trajectory. If the average time of occurrence differs considerably between states, then this preference is expressed as various unintended effects. Our core observation is that this situation is very common and the effects often manifest as either encouraging or discouraging exploration.

An especially prevalent situation in which the above biases occur is when we want an agent to achieve or avoid a specific state, which ends its experience. This is usually modelled as an environment in which every step results in constant reward until reaching a terminal state, when the rewards become 0, as in OpenAI's MountainCar and CartPole (Brockman et al., 2016), but also in Sutton et al. (1998).

If we finish an episode early, then the assumption in equation 2 is incorrect, because of all the constant rewards it ignores. As a simple example we will be using the MountainCar environment from the OpenAI gym (Brockman et al., 2016). In this case we will show that the distribution of states close to the end of a trajectory of a random agent trains a value estimator to prefer states which actually encourage exploration. We will also present some modifications of this environment in which the same effect discourages exploration and some even simpler environments which highlight the scope of the effect. Finally, we discuss some partial solutions for removing this bias.

1 Theoretical models

In this section we investigate two simple theoretical models, which provide useful intuition for the effects discussed in this paper.

1.1 Wiener process

We will use the following properties of a Wiener process:

1. A Wiener process W_t is a family of probability distributions on $\mathbb{R}$.
2. $W_0 = 0$ almost surely.
3. $W_t \sim \mathcal{N}(0, t)$.

Since it is a normal distribution, W_t has density function $f_t(x) = \frac{1}{\sqrt{2\pi t}} exp(-\frac{x^2}{2t})$. Constraining ourselves to the time-bounded case translates directly into the assumption that $t \in [0, N]$. If we further assume that the time is distributed uniformly, we can define the density function on $\mathbb{R} \times (0, N]$:

$$f(x,t) = \frac{1}{N} \frac{1}{\sqrt{2\pi t}} exp\left(-\frac{x^2}{2t}\right).$$

We now can define the average time at the state x, denoted by $T(x)$. This value is of interest to us, because in the discrete case with constant reward r in every step, visiting state x at time t results in (undiscounted) cumulative reward equal to $r(N - t)$. In that case the average value of state x is equal to $r\,(N - T(x))$. The average time at the state x is

$$T(x) = \frac{\int_0^N t f(x,t) dt}{\int_0^N f(x,t) dt} = -\frac{x^2}{3} + \frac{N}{3} \frac{1}{1 - \sqrt{\pi} \frac{|x|}{\sqrt{2N}} exp(x^2/2N) \operatorname{erfc}(|x|/\sqrt{2N})},$$

where erfc is the *complementary error function*:

$$\operatorname{erfc}(x) = \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-y^2} \mathrm{d}y. \tag{4}$$

See appendix for detailed calculations.

For clarity, we may write

$$T(-x\sqrt{2N}) = T(x\sqrt{2N}) = \frac{N}{3}\left(-2x^2 + \frac{1}{1 - \sqrt{\pi} x \cdot exp(x^2) \operatorname{erfc}(x)}\right)$$

for $x \geq 0$. This cannot be expressed in terms of elementary functions, but the part $\sqrt{\pi} x \cdot exp(x^2) \operatorname{erfc}(x)$ can be approximated with small relative error, see Oldham (1968).

Even though erfc is not an elementary function we know that $\operatorname{erfc}(0) = 1$ and $(\operatorname{erfc}(x))' = \frac{2}{\pi} exp(-x^2)$. Thus we may calculate the exact value of T and its directional derivatives at zero.

For example $T(0) = \frac{N}{3}$ and $T'(0) = \sqrt{\frac{N\pi}{18}}$. This gives some intuition about the steepness of the function in the proximity of the beginning of the walk. The value at zero grows linearly with N, while the steepness – sublinearly. The expected time for several trajectory lengths can be seen in Figure 1.

Setting $W_t \sim \mathcal{N}(0, t\sigma^2)$ for $\sigma > 0$ entails $f_{\sigma^2}(x,t) = \frac{1}{\sigma} f(\frac{x}{\sigma}, t)$ and, as a consequence, $T_\sigma(x) = T(x/\sigma)$. This should remind us of the discrete N-step random walk, where a single step has mean zero and variance σ^2. For large t the probability distribution of states in such a walk converge in distribution to $\mathcal{N}(0, t\sigma^2)$ due to the central limit theorem.

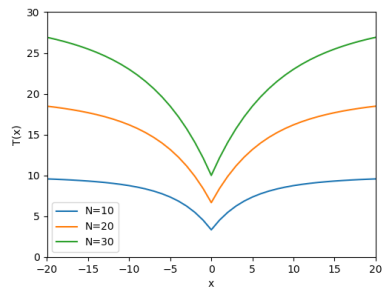


Figure 1: Expected time at a state, for several trajectory lengths.

If we assume that agent receives constant reward $r = -1$ in the discrete N-step random walk, then the (undiscounted) average value at a state x is equal to $T(x) - N$. Of course if the discrete random walk is infinite, then the value function is constant and equal to $\frac{r}{1-\gamma}$, where γ is the discount factor. Even if we included the discount in the finite value calculation, this would still be quite a bias, considering how steep T is as a function of x.

1.2 Random walk

In the next subsection we will further analyse the discrete random walk, with special attention paid to how γ affects the bias. Since we are considering a random walk, the agent's policy is constant and defined as (p_{-1}, p_0, p_{+1}), where p_a is the probability of moving from state s to state $s + a$. Let N is the number of steps taken by the agent, $s \in \mathbb{Z}$ be the state, and $t \in \{0, 1, \ldots, N\}$ – the time. We define the probability distribution $p(s,t)$ to be the probability of the state being s and the time being t. The walk starts at $s = 0$.

Every trajectory has length equal to $N + 1$, thus the marginal distribution $p(t)$ is uniform and equal to $\frac{1}{N+1}$.

$$p(s,t) = p(s \mid t) \cdot p(t) = \frac{1}{N+1} p(s \mid t).$$

It is worth noting that in our experiments this distribution can be interpreted as the distribution of data from the point of view of the value predictor. In this case the marginal distribution on time is also uniform, but only because we use Every-visit Monte Carlo as our value estimator, for other estimators it would be different.

We can calculate $p(s,t)$ using the recurrent formula for $p(s \mid t)$:

$$p(s \mid t = 0) = \begin{cases} 1 & \text{if} s = 0, \\ 0 & \text{otherwise.} \end{cases}$$

$$p(s \mid t = k+1) = p_{+1} \cdot p(s-1 \mid t = k) + p_0 \cdot p(s \mid t = k) + p_{-1} \cdot p(s+1 \mid t = k).$$

Let us assume that the agent receives constant reward r every step. Let $R_{t,\gamma}$ denote the cumulative discounted reward calculated at time t with discount factor γ:

$$R_{t,\gamma} = r + r\gamma + \ldots + r\gamma^{N-t-1} = r\frac{\gamma^{N-t} - 1}{\gamma - 1}.$$

The value of a state s is defined as the average cumulative discounted reward

$$V(s) = \frac{\sum_{t=0}^{N} p(s,t) \cdot R_{t,\gamma}}{\sum_{t=0}^{N} p(s,t)}.$$

Note that every partial trajectory that ends at time t in state s can be bijectively coupled with a mirror partial trajectory, where every step right is replaced with step left and step left replaced with step right. Thus if $p_{-1}, p_{+1} > 0$, then $p(s,t) = (\frac{p_{+1}}{p_{-1}})^s p(-s,t)$.

The value function is an even function if only $p_{-1}, p_{+1} > 0$. Indeed

$$V(-s) = \frac{\sum_{t=0}^{N} p(-s,t) \cdot R_{t,\gamma}}{\sum_{t=0}^{N} p(-s,t)} = \frac{\sum_{t=0}^{N} (\frac{p_{+1}}{p_{-1}})^{-s} p(s,t) \cdot R_{t,\gamma}}{\sum_{t=0}^{N} (\frac{p_{+1}}{p_{-1}})^{-s} p(s,t)} = \frac{\sum_{t=0}^{N} p(s,t) \cdot R_{t,\gamma}}{\sum_{t=0}^{N} p(s,t)} = V(s)$$

For examples of the value function for several different parameters see Figure 2.

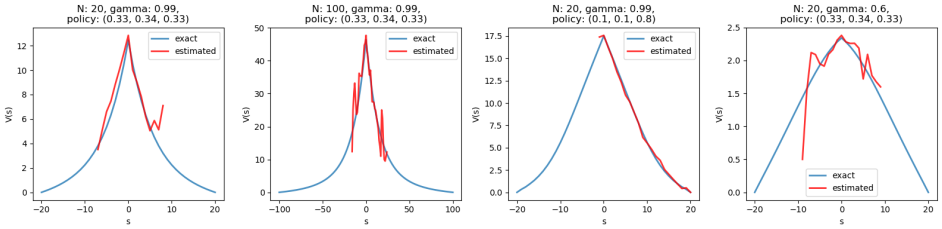


Figure 2: Value functions for a finite discrete random walk. Note that the vertical axis varies between the 4 plots.

The value function is usually decreasing with $|s|$. Small γ values lessen the bias. The value function estimator (here an average over 50 random trajectories) has high variance for states with low marginal probability $p(s)$. Although the value function is even, if p_{+1} is significantly larger than p_{-1}, then the value estimator will usually visit only the right half of the state space. In such a case if we were to try to approximate the value function with linear regression, then it will be a decreasing function instead of constant. The value function is linear with respect to the constant reward r. This means that if $r < 0$, then the value function has sharp minimum at $s = 0$ and if $r > 0$, then it has a sharp maximum.

To relate this theoretical work to the experiments that follow, we examine how the value function we computed affects agent training in our algorithms. Advantage is defined as $r_t - (V(s_t) - \gamma V(s_{t+1}))$. If we decide to train an agent using advantages and r_t is constant, then any agent preferences will stem only from the value function. In such a case, when $r < 0$ then the agent will be strongly *biased* to explore. For $r > 0$ agent will try to stay at $s = 0$. On the other hand, training with policy gradient has no such effect – policy gradient tries to maximize the average value of the whole trajectory, which is always equal to $N \cdot r$, thus the gradient is equal to zero everywhere and all changes in agent are only due to the *variance* of the gradient estimator.

2 Experimental setup

The complete codebase used for this paper can be found in the Github repository [anonymised for review] Here we just describe the basics of our experimental setup.

2.1 Algorithms

We use three algorithms throughout the paper, with various modifications that are described where applicable. Those three are:

policy gradient a plain policy gradient, as described in Williams (1992). This algorithm does not estimate values of states and the effects described in this paper do not occur, so we only use it for comparison.

Generalized Advantage Estimation also referred to as GAE throughout the paper, as described in Schulman et al. (2015). In short this algorithm uses two neural networks, one to estimate the value of a state (referred to as the *value predictor*), and another to actually take the actions (referred to as the *agent*). We train the value predictor to minimize the difference between predicted value and our estimator defined by equation 3, which is one of the approaches discussed in the original paper.

evolution algorithm as described in Salimans et al. (2017). This is a vary basic algorithm, which nonetheless presents excellent exploration in the problems considered. We use it only for

reference as an algorithm that achieves very good performance in the simple environments presented here.

2.2 NETWORKS

All the networks used in the experiments have the same architecture, including the value predictor. They consist of two hidden affine layers of size 64, each followed by an application of a leaky ReLU with leak 0.1. The final layer is again affine, with the output size dependent on the purpose of the net. The agents are all stochastic, so their networks (i.e. all but the value predictor) end with an application of softmax.

The weights in the networks are initialized with values close to 0 using a normal distribution. This is important because, as noted in Sutton et al. (1998), this also introduces some biases in the case of the value predictor. We will show that those biases are, once again, much weaker than the effects we focus on.

2.3 LEARNING PROCESS

In the case of the evolution algorithm and policy gradient the learning process is straightforward, we just run a single episode of the environment under consideration and immediately update the agent using the gathered experience. With GAE we update the agent when the first episode is completed after taking 2048 steps. This lowers the bias due to using the Every-visit Monte Carlo value estimator, while also improving the stability. We use the latter by increasing the learning rate – it is 0.001 for the policy gradient and 0.01 for GAE. By default we use a discount factor of 0.98. In all cases we use the Adam optimizer (Kingma & Ba, 2014).

GAE needs one more parameter called λ, controlling the impact of advantages with different time gaps between states. Our theoretical discussion essentially assumes $\lambda = 0$, that is computing the advantages only using values in consecutive steps. However, the original GAE paper (Schulman et al., 2015) suggests using much bigger values and most implementations follow their advice. In our experiments setting $\lambda = 0.95$ did not add significant effects, so to keep our results practical all the figures use this value.

2.4 NORMALIZATION

To provide greater stability we normalize the cumulative discounted rewards passed to the policy gradient and evolution agents, and also the advantages passed to the GAE agent. We use a running normalize with a discount factor of $\frac{1}{2}$. We do not perform any normalization of target values for the value predictor.

3 ENVIRONMENTS

We consider three environments to show the effects described in section 1. Later we explore various modifications of those environments to pinpoint the exact nature of the effects.

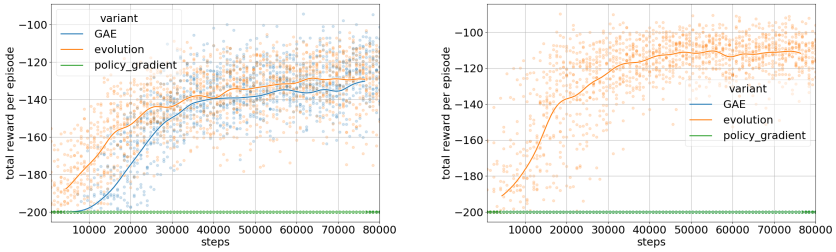
All the graphs presented in this and further sections contain translucent dots, representing the total episode rewards in specific runs of the algorithms, and lines representing the average reward. We ran 16 copies of every algorithm. When interpreting those one should take note that while the lines move, dots usually appear in the same areas. This is due to the challenge in the environments being mostly exploration, so whenever an algorithm finds any solution it often quickly learns how to achieve it consistently.

3.1 MOUNTAINCAR

First we consider the MountainCar environment from the OpenAI gym.

Because of the sparse reward in this environment any algorithm not including a nontrivial exploration component gets stuck close to the starting position. In our case this role is filled by policy gradient. In practice no trajectory will randomly reach the goal, so there simply is no data to learn from. Yet,

when ran with the GAE algorithm, the environment gets solved quite rapidly. The results are shown in Figure 3a.



(a) The standard MountainCar environment (b) The modified DragCar environment

Figure 3: Performance of three algorithms on both the Car problems.

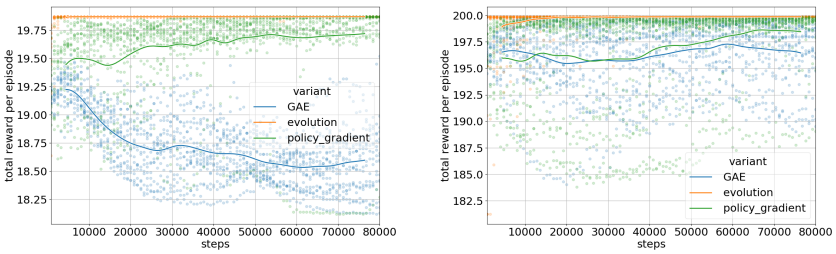
This is precisely the result we should expect. Since all the steps have reward -1 and all random trajectories have length 200, the states that are visited on average later in a trajectory have a higher value estimation. It remains to note that a random trajectory starts close to the centre of the valley, but will gather some momentum with time, thus making the states further from the centre appear more valuable. With a value predictor the agent will be therefore incentivised to seek out states further from the centre, which eventually leads it to the goal.

It is valuable to compare this result with the experiments in Sutton et al. (1998). There the trajectories of the environment are not truncated, so our bias is not present. However, the environment eventually gets solved, partially because of the bias introduced by initializing all the networks to 0, which is also present in our case. As we argue, this bias is weaker than ours.

3.2 Axis walk

To show that the discussed biases can also have obviously negative consequences we consider a very simple environment, in which the agent walks on the axis of integers, starting at 0. In every step it is allowed to move either right or left and the reward at a step ending up in state s (which is an integer) is $1 - \frac{1}{20(1+s^2)}$. Every episode ends in the same number of steps, either 20 or 200, depending on the variant of the environment used. It is easy to see that, since states further from the centre have higher rewards, the optimal strategy is to pick a direction and always walk that way.

This environment is a very close approximation of the random walk described in Section 1, since all rewards are quite close to 1. Therefore we should expect the GAE agent to avoid states far from 0, which is the exact opposite of the optimal strategy. Our expectations are confirmed in Figure 4. Note that policy gradient performs well on this task. The effect is weaker in the variant with more steps and we will discuss this phenomenon in Section 4.



(a) Axis walk with 20 steps per episode (b) Axis walk with 200 steps per episode

Figure 4: Performance of three algorithms on two variants of axis walk.

3.3 DRAGCAR

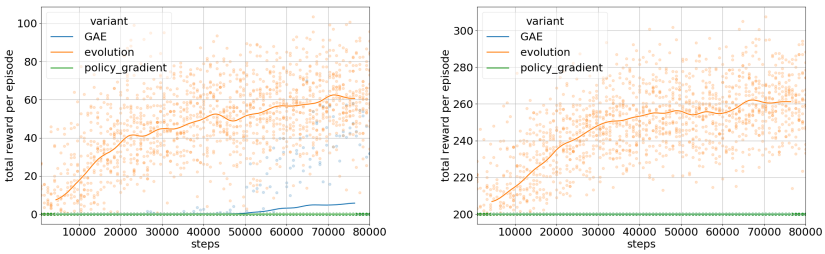
The above examples might give the impression, that just setting rewards in every step to a constant negative value encourages exploration. To dispel this notion we present a slight modification of MountainCar in which standard value predictors discourage exploration. This requires two modifications. First, one needs to introduce drag to the environment. The drag coefficient is not big and decreases the maximal achievable reward insignificantly. It's especially worth pointing out that just adding drag makes the algorithms perform about as good as without it.

Second, one needs to move the starting position. Instead of starting in the middle of the valley, the car randomly starts either halfway up the left or halfway up the right hill. Due to the drag, if the agent does nothing the car will slowly converge to the middle of the valley. So will in fact a random agent. It will obviously still create some swinging, but it's amplitude will be significantly smaller than the difference between the starting points. In this case the value predictor cannot extrapolate good rewards uphill, because the starting positions, which have the worst rewards, are the most uphill. In such a case the agent often learns to stay at the bottom of the valley, never leaving the spot. This is confirmed by the results visible in Figure 3b, which presents a sharp drop in the performance of GAE compared to Figure 3a, as it never reaches the goal. Also note that the evolution strategy performs better on this problem than on the standard MountainCar. This is due to the fact that the car starts with greater potential energy.

3.4 SCOPE OF EFFECTS

To see the scope of the effects described above it is useful to look at slight modifications of the MountainCar environment. We perform the modification in two steps. First, we extend all trajectories to be of length 200, even when the goal is reached earlier. We then treat the goal state as a terminal state and all the steps after reaching it give reward 0. This modification by itself does not in any way change the behaviour of the agents. Second, we produce two new environments by adding a constant equal to either 1 or 2 to all rewards. Since the trajectories have constant length, this modification does not change the goal or optimal strategy of an agent. It does however change the behaviour of GAE agents, precisely because of the bias we are describing.

After adding 1, all the rewards in a trajectory that does not reach the goal state are 0, and therefore the bias is non-existent. Adding 2 on the other hand makes all the rewards 1, so early states are more valuable, which discourages exploration. The results can be seen in Figure 5.



(a) Reward before reaching the goal 0 (b) Reward before reaching the goal 1

Figure 5: Performance of all the algorithms on the modified MountainCar environment, as described in Section 3.4.

In the first case we can still see some exploration. We argue that this is mainly due to the value predictor being initialized randomly and therefore containing some consistent noise, which sometimes encourages exploration. By "consistent noise" we mean that a randomly initialized value predictor will assign higher value to some areas in state space, so the GAE agent will be attracted to those areas. Since the predictor is updated relatively slowly, this attraction will be somewhat consistent between episodes. If the areas happen to include states higher up the slopes, the agent will find the goal. To see that this is a case consider Figure 6, where the standard MountainCar is being solved

by a GAE agent which does not update its value predictor. Note that the results are comparable to Figure 5a.

In the latter case consistent noise effect is completely overwhelmed by the discouraging effect of the constant positive reward, so no exploration occurs at all. Just as with policy gradient the rewards are constantly at the lowest possible value.

The effects of the same change on the DragCar environment are symmetric, so any of the above changes allow it to sometimes solve the environment.

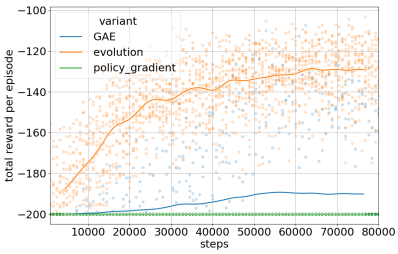


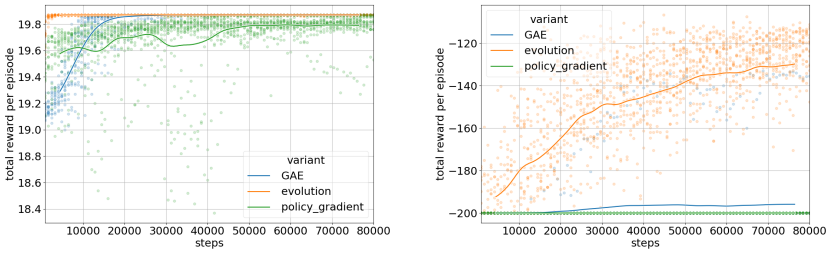
Figure 6: Performance of the GAE algorithm on the MountainCar environment when the value predictor is never updated.

4 Possible solutions

In this section we propose several attempts at removing the bias presented in previous sections. None of the solutions are completely satisfactory, but they can serve as good workarounds in many cases. In particular none of them allow DragCar to be solved, in contrast to changing the rewards as in Section 3.4.

4.1 Enforcing the Markov property

Since cutting off trajectories early mainly results in losing the Markov property, one idea for fixing the bias would be to reintroduce this property. We did it by modifying the environment to return states consisting of the original state and the number of the step. As long as the agent does not reach the goal, the cumulative reward at state (s, t) is completely determined by t, so an ideal predictor would ignore s, thus eliminating the bias. The results of applying this modification can be seen in Figure 7.



(a) Axis walk, 20 steps per episode

(b) MountainCar

Figure 7: Performance when time is part of the state.

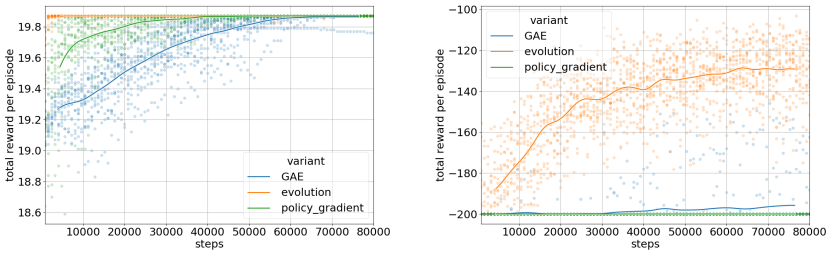
The remaining exploration in the case of MountainCar is mostly caused by the consistent noise effect, with at most slight effects from the value predictor not being able to completely decouple its predictions from the bare state s. In particular compare this to Figure 5a.

Another way of reintroducing the Markov property would be, instead of using a "hard" cutoff for trajectory length, to allow the trajectory to randomly end at any time with constant probability p. The constant p should be picked for the expected end of the trajectories to be sufficiently late as to keep the environment solvable, while still providing a reasonable upper bound on the length of computation. This approach works quite well but is impractical, so we do not explore this possibility further in this paper.

Both these solutions are unsatisfactory. Providing the agent with the time as an additional part of the state makes the policy depend on time, which might not be desirable. If during training we just used times up to T, but deployed the agent in an environment with times greater than T, then the dependency on time would create possibly erratic behaviour for later states. This problem might be solved by only adding time to states passed to the value predictor, and then removing it before passing the states to the agent, but this solution is beside the scope of this paper. As for the second solution, in some cases the cutoff for trajectory length is naturally imposed by the environment, and not just a computational limitation, so changing it might not be feasible.

4.2 Adjusting γ

Instead of attempting to eliminate the bias completely, we might try to limit it. This can be done by lowering the discount factor γ. When we do so, states far away would have a small impact on the value of the current state, so assuming that some of them are 0 (as equation 2 postulates) does not produce big effects. Note that lengthening the episode has a similar effect on the validity of this assumption, as mentioned in Section 3.2. The results of adjusting the discount factor can be seen in Figure 8.



(a) Axis walk, 20 steps per episode (b) MountainCar

Figure 8: Performance when we lower the discount factor in GAE from 0.98 to 0.8.

Of course this does not completely eliminate the biases, they are still visible near the ends of the trajectories. What is perhaps a more important problem with this solution is the obvious trade-off. Lowering the discount factor actually makes the agents less sensitive to rewards further away in the future. In the case of many environments this is not an acceptable sacrifice.

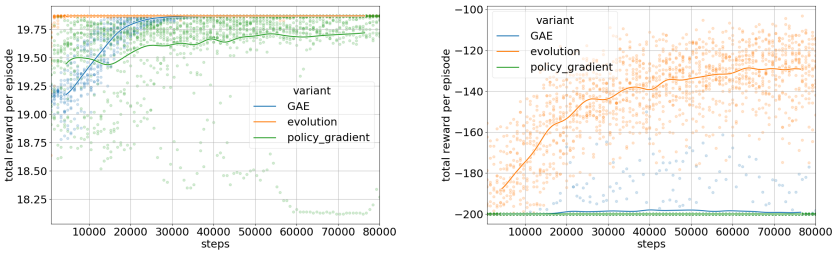
4.3 Better approximation of rewards after episode end

Since the problem stems from the assumption in equation 2, the final class of solutions we propose focuses on finding a better guess for what the rewards after the episode end should be. There are some approaches to this problem, but once again none are quite satisfactory.

A very simple idea would be to assume the value of the last state is the discounted average reward summed to infinity.[1] This obviously introduces new biases, the greater the bigger the changes of the average reward in time. In particular, if the reward in a absorbing state is always zero, then assuming it is equal to the average reward definitely introduces a bias. However, in our environments this works reasonably well in practice, see Figure 9.

Another approach would be to use the value predictor to simply predict the value of the last state and assume this prediction is correct. This approach bears a superficial similarity to n-step Boot-

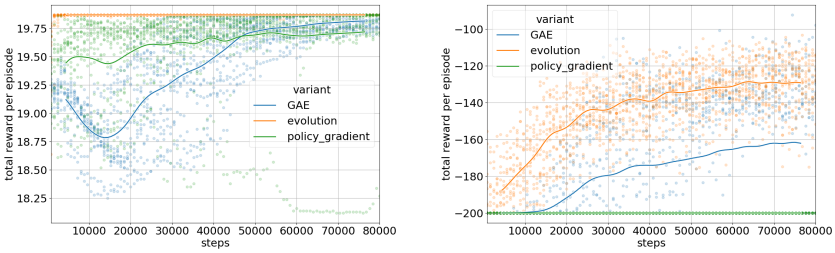
[1] We want to thank [name redacted] for bringing this idea to our attention.



(a) Axis walk, 20 steps per episode (b) MountainCar

Figure 9: Performance where the value of the last state is assumed to be the average reward summed to infinity.

straping, but this is misleading, as the n would have to vary between the states and even trajectories. The assumption introduces some reflective dependency on the learning process, which is of course worrying because of the added complexity. This proves to be a problem also in practice, see Figure 10. In particular note that the biases are all still present, but at least in the case of Axis walk the modification eventually helps. The prediction of the value for an absorbing state is still a problem, since if we cannot distinguish it from other states we cannot set it to 0, and it keeps being determined by the state of the value predictor.



(a) Axis walk, 20 steps per episode (b) MountainCar

Figure 10: Performance where the value of the last state is assumed to be correctly predicted by GAE.

5 Conclusions

In many environments using value predictors might result in unexpected learning patterns. We demonstrated that both discouragement and encouragement of exploration, as well as constraining movement to a small segment of the state space, occur in simple environments. When using GAE or similar algorithms one should therefore exercise caution, keep in mind the possibility of bias, and where applicable attempt some form of mitigation. We proposed a couple partial solutions together with advise when to use or avoid them.

Value predictors are of course very useful and work quite well in many settings, so despite the flaws we demonstrated they should enjoy continued popularity. We hope that further research will discover either better ways of mitigating the biases described in this paper or some algorithms that avoid the problems altogether while keeping the strengths of value predictors.

References

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

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980*, 2014.

KB Oldham. Approximations for the $x \ \exp \ x^2 \ \operatorname{erfc} \ x$ function. *Mathematics of Computation*, 22 (102):454–454, 1968.

Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv:1703.03864*, 2017.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv:1506.02438*, 2015.

Satinder P. Singh and Richard S. Sutton. Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22(1):123–158, Mar 1996. ISSN 1573-0565. doi: 10.1007/BF00114726. URL `https://doi.org/10.1007/BF00114726`.

Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*. MIT press, 1998.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

A THE AVERAGE TIME AT A STATE IN THE WIENER PROCESS

In this appendix we compute the average time at a state in the Wiener process. First some definitions relating to the error function:

$$\begin{aligned}\operatorname{erf}(x) &= \frac{2}{\sqrt{\pi}}\int_0^x e^{-y^2}dy,\\ \operatorname{erfc}(x) &= 1-\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}}\int_x^\infty e^{-y^2}dy,\\ \operatorname{erf}(+\infty) &= 1.\end{aligned}$$

We will also use the following integrals:

$$\int \frac{e^{-x^2}}{x^2}dx = -\frac{e^{-x^2}}{x} - \sqrt{\pi}\operatorname{erf}(x), \tag{5}$$

$$\int \frac{e^{-x^2}}{x^4}dx = \frac{1}{3}(-\frac{e^{-x^2}}{x^3} + 2\frac{e^{-x^2}}{x} + 2\sqrt{\pi}\operatorname{erf}(x)). \tag{6}$$

We want to calculate the expected time at state x

$$T(x) = \frac{\int_0^N tf(x,t)dt}{\int_0^N f(x,t)dt},$$

where

$$f(x,t) = \frac{1}{N}\frac{1}{\sqrt{2\pi t}}exp(-\frac{x^2}{2t}).$$

First we compute the denominator:

$$\begin{aligned}\int_0^N f(x,t)dt &= \int_0^N \frac{1}{N}\frac{1}{\sqrt{2\pi t}}exp(-\frac{x^2}{2t})dt = \dots\\ &\left[w^2 = \frac{x^2}{2t},\ dt = -\frac{x^2}{w^3}dw,\ \sqrt{t} = \frac{|x|}{w\sqrt{2}}\right]\\ \dots &= \int_{+\infty}^{\frac{|x|}{\sqrt{2N}}} -\frac{1}{N}\frac{w}{|x|\sqrt{\pi}}\frac{x^2}{w^3}e^{-w^2}dw\\ &= \int_{+\infty}^{|x|/\sqrt{2N}} -\frac{1}{N}\frac{|x|}{w^2\sqrt{\pi}}e^{-w^2}dw\\ &= \frac{1}{N}\frac{|x|}{\sqrt{\pi}}\int_{\frac{|x|}{\sqrt{2N}}}^{+\infty}\frac{e^{-w^2}}{w^2}dw\\ &\overset{(5)}{=} \frac{1}{N}\frac{|x|}{\sqrt{\pi}}(-\frac{e^{-w^2}}{w} - \sqrt{\pi}\operatorname{erf}(w))\Big|_{\frac{|x|}{\sqrt{2N}}}^{+\infty}\\ &= \frac{1}{N}\frac{|x|}{\sqrt{\pi}}(0 + e^{-x^2/(2N)}\frac{\sqrt{2N}}{|x|} - \sqrt{\pi}(1-\operatorname{erf}(\frac{|x|}{\sqrt{2N}})))\\ &= \frac{1}{N}(\sqrt{\frac{2N}{\pi}}e^{-x^2/(2N)} - |x|\operatorname{erfc}(\frac{|x|}{\sqrt{2N}})).\end{aligned} \tag{7}$$

Now the numerator:

$$
\begin{aligned}
\int_0^N tf(x,t)dt &= \int_0^N t\frac{1}{N}\frac{1}{\sqrt{2\pi t}}exp(-\frac{x^2}{2t})dt = \dots \\
&\left[w^2 = \frac{x^2}{2t},\ dt = -\frac{x^2}{w^3}dw,\ \sqrt{t} = \frac{|x|}{w\sqrt{2}}\right] \\
&\dots = \frac{1}{N}\frac{|x|^3}{2\sqrt{\pi}}\int_{\frac{|x|}{\sqrt{2N}}}^{+\infty}\frac{e^{-w^2}}{w^4}dw \\
&\overset{(6)}{=} \frac{1}{N}\frac{|x|^3}{2\sqrt{\pi}}\frac{1}{3}(-\frac{e^{-x^2}}{x^3} + 2\frac{e^{-x^2}}{x} + 2\sqrt{\pi}\,\mathrm{erf}(x))\Big|_{\frac{|x|}{\sqrt{2N}}}^{+\infty} \\
&= \frac{1}{N}\frac{|x|^3}{2\sqrt{\pi}}\frac{1}{3}(0 + e^{x^2/(2N)}((\frac{\sqrt{2N}}{|x|})^3 - 2\frac{\sqrt{2N}}{|x|}) + 2\sqrt{\pi}(1 - \mathrm{erf}(\frac{|x|}{\sqrt{2N}}))) \\
&= \frac{1}{N}(\frac{N-x^2}{3}\sqrt{\frac{2N}{\pi}}e^{-x^2/(2N)} + \frac{x^2}{3}|x|\,\mathrm{erfc}(\frac{|x|}{\sqrt{2N}}))
\end{aligned}
\tag{8}
$$

Thus, combining the results:

$$
\begin{aligned}
T(x) &= \frac{\int_0^N tf(x,t)dt}{\int_0^N f(x,t)dt} \\
&\overset{(7)(8)}{=} \frac{\frac{1}{N}(\frac{N-x^2}{3}\sqrt{\frac{2N}{\pi}}e^{-x^2/(2N)} + \frac{x^2}{3}|x|\,\mathrm{erfc}(\frac{|x|}{\sqrt{2N}}))}{\frac{1}{N}(\sqrt{\frac{2N}{\pi}}e^{-x^2/(2N)} - |x|\,\mathrm{erfc}(\frac{|x|}{\sqrt{2N}}))} \\
&= -\frac{x^2}{3} + \frac{N}{3}\frac{\sqrt{\frac{2N}{\pi}}e^{-x^2/(2N)}}{\sqrt{\frac{2N}{\pi}}e^{-x^2/(2N)} - |x|\,\mathrm{erfc}(\frac{|x|}{\sqrt{2N}})} \\
&= -\frac{x^2}{3} + \frac{N}{3}\frac{1}{1 - \sqrt{\pi}\frac{|x|}{\sqrt{2N}}e^{x^2/(2N)}\,\mathrm{erfc}(\frac{|x|}{\sqrt{2N}})}.
\end{aligned}
$$