IS HIGH VARIANCE UNAVOIDABLE IN RL? A CASE STUDY IN CONTINUOUS CONTROL

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement learning (RL) experiments have notoriously high variance, and minor details can have disproportionately large effects on measured outcomes. This is problematic for creating reproducible research and also serves as an obstacle for real-world applications, where safety and predictability are paramount. In this paper, we investigate causes for this perceived instability. To allow for an in-depth analysis, we focus on a specifically popular setup with high variance – continuous control from pixels with an actor-critic agent. In this setting, we demonstrate that poor outlier runs which completely fail to learn are an important source of variance, but that weight initialization and initial exploration are not at fault. We show that one cause for outliers is numerical instability which leads to saturating nonlinearities. We investigate several fixes to this issue and find that one particular method is surprisingly effective and simple – normalizing penultimate features. Addressing the instability significantly decreases the variance of outcomes, lowering the average standard deviation across 21 tasks by a factor > 3 for a state-of-the-art agent. This demonstrates that the perceived variance in RL is not necessarily inherent to the problem definition and may be addressed via simple modifications.

1 INTRODUCTION

With the advent of deep learning, reinforcement learning (RL) with function approximators has become a promising strategy for learning complex behavior (Silver et al., 2017; Mnih et al., 2015). While RL can work remarkably well, deep RL experiments have been observed to exhibit high variance (Chan et al., 2019; Clary et al., 2019; Lynnerup et al., 2020). Specifically, the rewards obtained by RL agents commonly vary significantly depending on the random seeds used for training, even if no other modifications are made. Furthermore, implementation level details often have an outsized effect on performance (Engstrom et al., 2020) and even hardware scheduling can significantly affect outcomes (Henderson et al., 2017). In comparison to RL, experiments in computer vision and natural language processing (NLP) often have lower variance (Bouthillier et al., 2021).

High variance is problematic for creating reproducible RL research, and achieving statistically significant results over large RL benchmarks is often prohibitively time-consuming (Colas et al., 2018; Agarwal et al., 2021). Furthermore, this variance provides an obstacle for real-world applications of RL, where safety and predictability can be critical (Garcıa & Fernández, 2015; Srinivasan et al., 2020). RL is already being evaluated for medical testing (Bastani et al., 2021), autonomous vehicles (Bellemare et al., 2020) and communications infrastructure (Liu et al., 2020). RL agents are typically evaluated on the average performance, but for these sensitive real-world applications, the drastic performance drop for the worst-performing training runs could yield disastrous outcomes.

In this paper, we aim to investigate causes for high variance in RL. To limit the paper's scope and computational footprint we focus on continuous control from pixels with a state-of-the-art (SOTA) actor-critic algorithm (Lillicrap et al., 2015; Yarats et al., 2021b). This is a popular setup that exhibits high variance, and where actor-critic algorithms often give SOTA results (Laskin et al., 2020b;a; Kostrikov et al., 2020). Additionally, this class of tasks is relatively close to real-world robotics (Zhan et al., 2020).

In this setting, we first show that an important cause of variance is outlier runs that fail early and never recover. It is natural to hypothesize that the typical sources of randomness, network initializa-

tion and initial exploration, are to blame. However, we demonstrate that these do in fact have little impact on the final performance. We similarly provide experiments that suggest that neither poor feature representations nor excessive sensitivity of the environment is at fault. We instead demonstrate that numerical instability can drive the variance and show that excessively large activations that saturate action distributions can often yield poor performance. We propose several methods to avoid such issues and find that they decrease variance. The most effective method is simply normalizing the penultimate features which avoids exploding activations by design. By combining multiple approaches we increase average rewards while decreasing variance as measured of 21 tasks. The average standard deviation decreases by a factor > 3 and for some tasks by more than an order of magnitude. Our experiments suggest that high variance is not necessarily inherent to RL and that addressing numerical issues is a promising avenue for stabilizing RL agents. We summarize our contributions:

- We provide extensive experiments on variance in a popular continuous control setup. We show that outlier runs that completely fail to learn is an important driver of variance, but that poor initialization, sensitive environments, and feature learning are likely not at fault.
- We demonstrate that excessively large activations that saturate action distributions can cause poor outlier runs and propose fixes to stabilize learning and avoid this issue.
- Over 21 continuous control tasks and with a SOTA agent, we show that our modifications decrease the average standard deviation by a factor > 3 while improving average reward. This shows that directly optimizing for lower variance can be both feasible and fruitful.

2 BACKGROUND

Reinforcement Learning. Reinforcement learning tasks can be formulated as Markov decision processes (MDPs) defined by a state-space $\mathcal{S}$, an action space $\mathcal{A}$, transition probabilities P, and a reward distribution r (Sutton & Barto, 2018). For continuous control tasks, the action space $\mathcal{A}$ and the state space $\mathcal{S}$ are typically continuous. Each dimension of the action space $\mathcal{A}$ might for example correspond to a joint of a robotic arm, and the space is often bounded to account for physical limitations, e.g., how much force the robotic arm can exert. A common setting is simply $\mathcal{A} = [-1, 1]^{dim(\mathcal{A})}$. At every time step t, the agent is in some state $\mathbf{s}_t \in \mathcal{S}$ and takes an action $\mathbf{a}_t \in \mathcal{A}$ to arrive at a new state $\mathbf{s}_{t+1} \in \mathcal{S}$. The transition between states given an action is random with transition probability $P : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, \infty)$. Given some action $\mathbf{a}_t$ at timestep t, the agent also receives a reward r_t from the reward distribution r. When benchmarking RL agents, it is common to measure performance by the cumulative rewards $\sum_t r_t$.

DDPG. Deep Deterministic Policy Gradients (DDPG) (Lillicrap et al., 2015) is an algorithm for continuous control with roots in Q-learning (Watkins, 1989). Together with its offshoot soft actor-critic (SAC) (Haarnoja et al., 2018), DDPG forms the backbone of many state-of-the-art algorithms for continuous control (Laskin et al., 2020b;a; Kostrikov et al., 2020; Yarats et al., 2021b). DDPG uses a neural network (the *critic*) to predict the Q-value $Q_\phi(\mathbf{a}_t, \mathbf{s}_t)$ for each state-action pair. The parameters ϕ of the critic are trained by minimizing the soft Bellman residual:

$$\mathbb{E}\Big(Q_\phi(\mathbf{a}_t, \mathbf{s}_t) - \big[r_t + \gamma \mathbb{E}[\hat{Q}(\mathbf{a}_{t+1}, \mathbf{s}_{t+1})]\big]\Big)^2. \tag{1}$$

Here, r_t is the reward and $\mathbb{E}[\hat{Q}(\mathbf{a}_{t+1}, \mathbf{s}_{t+1})]$ is the Q-value estimated by the *target* critic – a network whose weights can e.g. be the exponentially averaged weights of the critic. One can use multi-step returns for defining the critic target (Sutton, 1988) and double clipped q-learning (i.e. use two Q-networks Q_1, Q_2 and take $Q = \min(Q_1, Q_2)$) to decreased Q-value overestimation (Fujimoto et al., 2018). The policy $\pi(\mathbf{s}) = \arg\max_{\mathbf{a}} Q_\phi(\mathbf{s}, \mathbf{a})$ is intractable since it requires finding $\arg\max$ over a continuous action space. Instead the policy is given by an *actor*-network with parameters θ, which maps each state $\mathbf{s}_t$ to a distribution $\pi_\theta(\mathbf{s}_t)$ over actions. The action is obtained by calculating $\mathbf{a}_\theta^{pre}(\mathbf{s})$, the vector output of the actor network. Thereafter a $\tanh$ non-linearity is applied elementwise to ensure $\mathbf{a} \in [-1, 1]^n$ for $n = dim(\mathcal{A})$. The policy is then obtained as:

$$\mathbf{a}_\theta(\mathbf{s}) = \tanh(\mathbf{a}_\theta^{pre}(\mathbf{s})) + \boldsymbol{\epsilon} \tag{2}$$

$\boldsymbol{\epsilon}$ is some additive noise which encourages exploration, e.g. a Gaussian variable with time-dependent variance σ_t^2. Various noise types have been employed (Achiam, 2018; Lillicrap et al., 2015), Yarats

et al. (2021b) for example use Gaussian noise which is truncated outside of acting. Since the Q-values correspond to the expected discounted rewards, the actions $\mathbf{a}_\theta$ should maximize $Q_\phi(\mathbf{a}_\theta, \mathbf{s})$. Thus, to obtain a gradient update for the actor one uses the gradient $\nabla_\theta Q_\phi(\mathbf{a}_\theta, \mathbf{s})$, effectively calculating gradients *through* the critic network. In control from pixels, the state $\mathbf{s}$ is typically an image (possibly data augmented) processed by a convolutional network (Kostrikov et al., 2020; Laskin et al., 2020a).

3 Benchmarking Variance

3.1 Experimental Setup

There is a large number of available RL agents and environments to consider. To limit the paper's scope and computational footprint, we consider continuous control from pixels with an actor-critic algorithm. Continuous control from pixels is an environment type close to real-world applications such as autonomous driving (Kiran et al., 2021) and robotics (Zhu et al., 2020). Investigating causes of high variance in this domain could therefore have a large potential impact on downstream applications. We consider the standard continuous control benchmark deepmind control (dm-control) (Tassa et al., 2020). Within dm-control, a long line of state-of-the-art models relies on the actor-critic setup (Laskin et al., 2020b;a; Kostrikov et al., 2020; Yarats et al., 2021b). We consider DDPG (Lillicrap et al., 2015), using the recent data-augmentation based DDPG agent of Yarats et al. (2021b) which provides state-of-the-art performance and runs fast. We use the default hyperparameters that Yarats et al. (2021b) uses on the *medium* benchmark (listed in Appendix A) throughout the paper. We will refer to this agent as the *baseline* agent. We are interested in using a large number of random seeds to obtain accurate statistics and thus need to limit the number of environments we consider. To select environments we simply rank the dm-control tasks based upon their relative variance as measured by Yarats et al. (2021a). We select the five tasks with the highest relative variance – finger turn hard, walker walk, hopper hop, acrobot swingup, and reacher hard (see Table 5 in Appendix A for details). In Section 4.2 we consider a larger set of tasks. We will refer to these as *environments* or *tasks* interchangeably. For each run, we train the agent for one million frames, or equivalently 1,000 episodes, and evaluate over ten episodes. We use 10 or 40 seeds for each method we consider, these are not shared between experiments, but we reuse the runs for the baseline agent across figures.

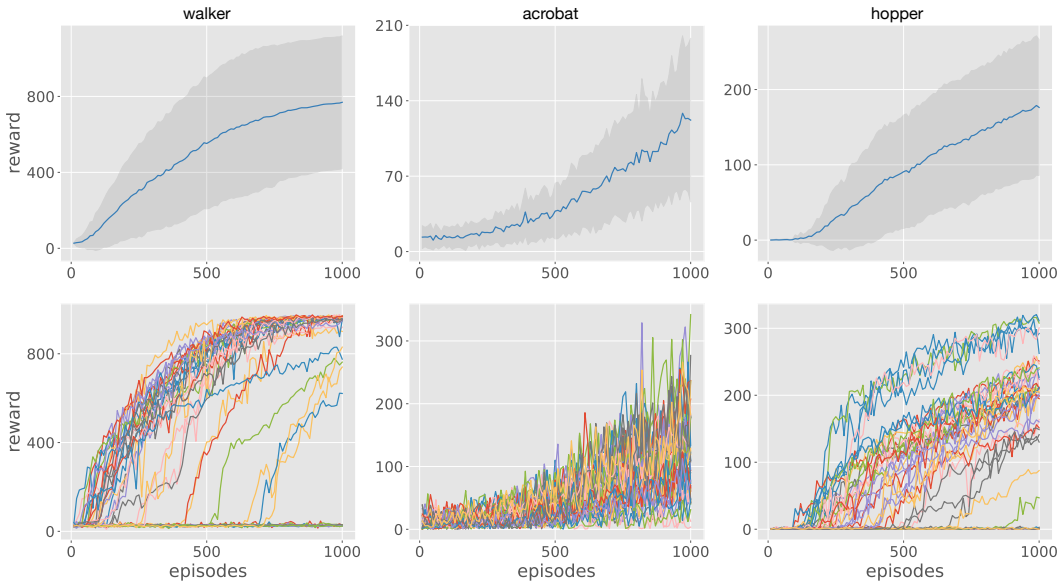

Figure 1: Learning curves over 40 seeds for the baseline agent. The upper row shows the mean reward with one standard deviation indicated by shading. The bottom row shows learning curves for individual runs. For some tasks, the variance is dominated by a few outlier runs that completely fail to learn. This is especially clear in the walker task. Removing such outlier runs would decrease the variance.

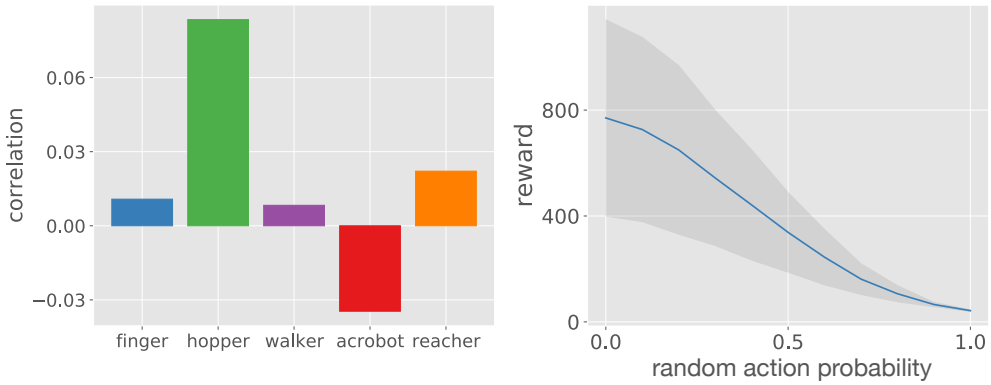

Figure 2: (**Left**) Correlation between runs that share network initialization and starting experience, but otherwise have independent other sources of randomness (e.g. from mini-batch sampling). Across all tasks, the correlation is small. This suggests that initialization and starting experience have limited effect on the outcomes. (**Right**) The average reward for the walker task when interpolating between a learned policy (random action probability = 0) and random policy (random action probability = 1). The rewards obtained decrease gracefully as we adopt a more random policy. This suggests that the environment is relatively forgiving with respect to actions.

3.2 Verifying Variance

We first aim to verify that the environments indeed do exhibit high variance. To this end, we run the baseline algorithm with 40 random seeds on each task. Figure 1 shows the result of this experiment for three tasks, see Appendix A for the two other tasks. We show both individual runs as well as mean and standard deviations. For some tasks, the variance is dominated by *outlier runs* that fail to learn early in the training and never recover. E.g. for the walker task, we see several runs which have not improved during the first 500 episodes, and most of these do not improve in the following 500 episodes either. Except for these outliers, the variance of the walker task is relatively small. We also note that these outliers separate themselves relatively early on, suggesting that there are some issues in early training that cause variance. If it was possible to eliminate poor outlier runs, we would expect the variance to go down. Motivated by these observations, and the walker task in particular, we will investigate various potential causes of outlier runs.

3.3 Does initialization matter?

There are multiple sources of randomness specific to the early training iterations. The agent networks are randomly initialized and the agent is given some initial experience, collected by random actions, to start training on. Could these two initial factors be the cause of the high variance? To study this hypothesis we measure the correlation between two runs that share the same network initialization and initial experience, but otherwise have independent randomness (from e.g. the environment, action sampling, and mini-batch sampling). Across all tasks, we repeat this process for ten combinations of initializations and initial experience, from each such combination we start two runs A, B which do not share other sources of randomness. For each time-step t we measure the Pearson correlation between $score(A)$ and $score(B)$. This correlation, averaged across all timesteps, is shown in Figure 2 (left). We see that the correlation is very small, suggesting that the initialization and initial experience does not have a large effect on the outcome. Specifically, the hypothesis that there are good and bad initializations that leads to good or bad performance is inconsistent with the results of Figure 2 (left). Correlation across time is noisy but centers around zero, see Appendix A.

3.4 Are the Environments too Sensitive?

Another cause of variance is the environments. Perhaps the environments require very specific policies to yield rewards, and unless the agent stumbles upon a good policy early on, learning fails. To study the sensitivity of the environments we consider gradually exchanging a good learned policy

Table 1: Performance when using self-supervised learning or pretrained weights for the concolutional encoder. Improvements are small and variance remain high. This suggests that poor features might not cause the high variance. Indeed, Table 2 shows that performance can be improved significantly without better representation learning. Metrics are calculated over 10 seeds.

metric	method	acrobot	finger turn	hopper hop	reacher	walker	avg.
μ	baseline	122.0	278.6	176.2	678.0	769.1	404.8
μ	self-supervised 10k	141.4	456.2	176.5	671.6	677.1	424.6
μ	self-supervised 20k	159.4	325.3	209.5	659.6	773.8	**425.5**
μ	pretrain	160.0	303.7	153.3	620.8	581.4	363.8
μ	pretrain + small lr	157.3	358.8	124.8	679.0	674.2	398.8
σ	baseline	76.0	221.2	90.3	167.3	350.1	181.0
σ	self-supervised 10k	68.4	276.1	104.6	222.4	425.9	219.5
σ	self-supervised 20k	67.7	191.9	71.6	164.8	374.4	**174.1**
σ	pretrain	39.8	146.7	86.8	168.7	429.1	174.2
σ	pretrain + small lr	71.8	204.6	102.7	165.4	393.5	187.6

with a random policy. Specifically, at every timestep t, with probability p, we exchange the policy's action with a random action. We use the final policy of a performant run with the baseline agent as the learned policy. The effect of this, for ten final policies learned for the walker environment, is shown in Figure 2 (right). Performance degrades gracefully as we gradually move to a random policy. This suggests that the environment is relatively stable with regard to actions. Results for other environments are similar and are shown in Appendix A. Furthermore, in Appendix B we investigate how much variance comes from evaluating the agent on a finite number of episodes. When using ten seeds and ten episodes (the lowest numbers we use), the evaluation variance is dominated by the variance of the random training process. Thus, environment sensitivity and evaluation methodology do not appear to explain the excessive variance.

3.5 Is feature learning to blame?

In the early stages of training, the image encoder will extract poor features. Without good representations, it is hard to learn a good policy, and without a policy generating interesting behavior it is hard to learn good representations. If such a vicious cycle was causing runs to fail, jump-starting the encoder to extract good features should help to decrease the variance. To test this hypothesis, we consider initializing the weights of the convolutional encoder to some good pretrained values before training as normal. For each task, we pick a run with an above-average final performance and extract the convolutional weights at episode 500. We refer to this strategy as pretrain. We also consider the same strategy but decrease the learning rate of the pre-trained weights by a factor of 10x. We refer to this strategy as pretrain + small lr. In addition to this fix, we also consider improving the features with self-supervised learning during early training (Chen et al., 2020). We use the loss function proposed in Grill et al. (2020) which is useful for smaller batch sizes (we use 256) and has already successfully been applied in RL (Schwarzer et al., 2020). We use standard methods in self-supervised learning: a target network with exponentially moving weights and a *head* network. See Appendix A for details. We consider adding the self-supervised loss to the first 10,000 and 20,000 steps and refer to these modifications as self-supervised 10k and self-supervised 20k. The results for this experiment are shown in Table 1. For these fixes, the large variance persists even though the performance improves marginally with self-supervised learning. We thus hypothesize that representation learning is not the main bottleneck. Indeed, experiments in later sections of the paper show that one can significantly decrease variance without improving representation learning.

3.6 Stability of Network Parametrization

We have considered the effects of initialization, feature learning, and environment sensitivity and found that none of these factors satisfactorily explain the observed variance. Another factor to consider is the stability of the learning algorithms. Such instability could make the learning process "chaotic", which would cause high variance. To study this hypothesis we investigate the failure

modes of individual runs. In Figure 3 we consider some runs that fail as well as some successful runs for the walker task. Plots for all runs are available in Appendix A. We plot four quantities during training: rewards, actor gradient norm $\|\nabla \ell\|$, actor weight norm $\|w\|$, and average absolute value of actions $|\mathbf{a}|$. All agents initially have poor rewards, but once the rewards start to improve, they improve steadily. During this initial period of no or little learning, the gradients of the actor-network (row two in Figure 3) are small. Similarly, the weight norm of the actor seems to be constant (row three in Figure 3). This suggests that the actor isn't changing its weight much (technically the weights could be "rotating", but this seems unlikely). Why would the actor weights not change? Recall that the actor is optimized via gradients:

$$\nabla_\theta Q_\phi(\mathbf{s}, \mathbf{a}_\theta(\mathbf{s})) = \sum_i \frac{\partial Q_\phi}{\partial \mathbf{a}_i} \frac{\partial \mathbf{a}_i}{\partial \theta} \tag{3}$$

If this gradient is close to zero, we might suspect that $\frac{\partial \mathbf{a}_i}{\partial \theta} \approx 0$. This could easily happen if the $\tanh$ non-linearity in equation 2 is saturated. In row four of Figure 3 we show the average absolute value of the actions, which are all bound to lie in $[-1, 1]$. We see that for the runs that fail the average absolute action stays around 1 throughout training, which implies that the gradients for the actor are small since $|\tanh(x)| \approx 1$ implies $\frac{\partial}{\partial x}\tanh(x) \approx 0$. The cause of saturating tanh must be excessively large activations, i.e. $\mathbf{a}_\theta^{pre}(\mathbf{s})$ in equation 2 becoming too large, which saturates the tanh non-linearity. In fact, all runs seem to initially suffer from saturating tanh, but most runs escape from this poor policy. The Q-values could of course also suffer from exploding activations, but since there are no saturating nonlinearities for the Q-values, such issues are harder to pinpoint. These experiments suggest that improving numerical stability and avoiding excessively large activations could reduce the number of poor outlier runs, and thus decrease variance.

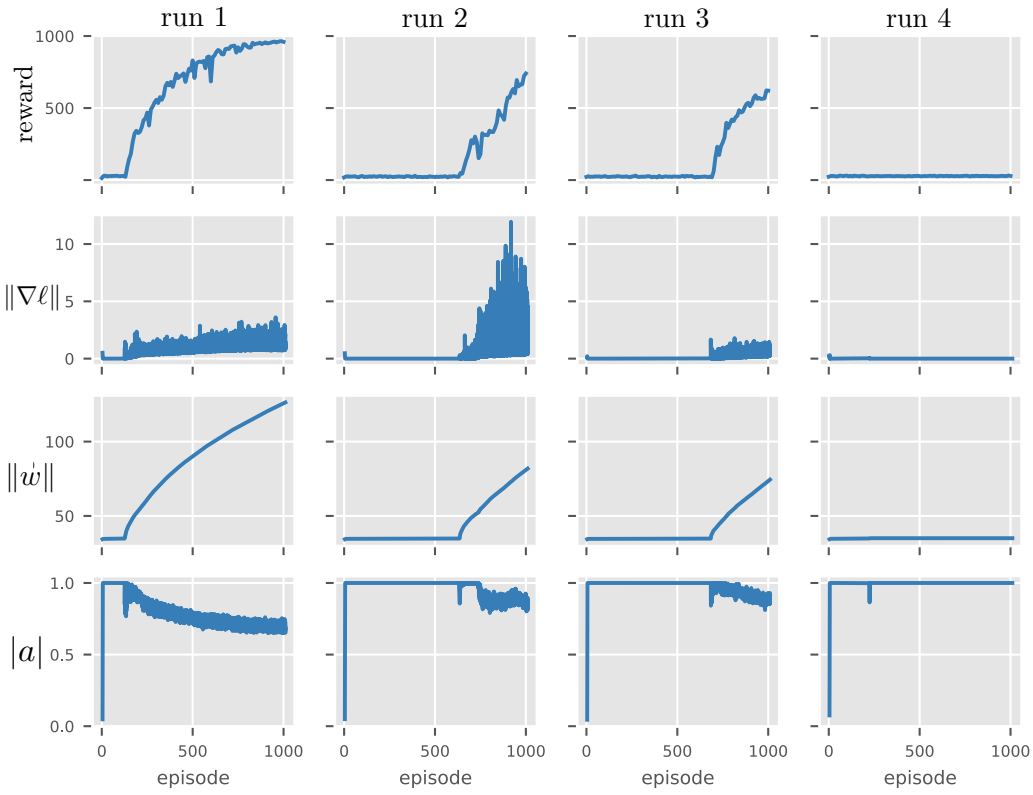

Figure 3: Each row shows a different quantity during training for the baseline agent. Row 1 shows rewards, row 2 shows actor gradient norm $\|\nabla \ell\|$, row 3 shows actor weight norm $\|w\|$ and row 4 shows average absolute action $|\mathbf{a}|$. Each column corresponds to an independent run of the walker task: run 1 is successful, runs 2 and 3 eventually learn, while run 4 fails to learn. Before learning starts, all runs suffer from small gradients and weights which barely change. During this period, the average absolute value of actions is close to 1. This causes small gradients since the gradient of $\tanh(x)$ is small when $|\tanh(x)| \approx 1$. This suggests that numerical issues may hamper learning.

4 IMPROVING STABILITY

Motivated by our qualitative observations we now consider a few ideas for improving the stability of learning, hoping that these will decrease the variance. We consider several methods:

Normalizing activations. Recall that the actor selects actions using $\mathbf{a}_\theta^{pre}(\mathbf{s})$, where $\mathbf{a}_\theta^{pre}(\mathbf{s})$ is a deep neural network. Figure 3 shows that learning can fail due to excessively large activations. To avoid this we propose to parametrize $\mathbf{a}_\theta^{pre}(\mathbf{s})$ as a linear function of features with a fixed magnitude. Specifically, let $\Lambda_\theta(\mathbf{s})$ be the penultimate features of the actor-network. We propose to simply normalize $\Lambda_\theta^n(\mathbf{s}) = \Lambda_\theta(\mathbf{s})/\|\Lambda_\theta(\mathbf{s})\|$ and let $\mathbf{a}^{pre}$ be a linear function of Λ_θ^n, i.e. $\mathbf{a}_\theta^{pre}(\mathbf{s}) = L\Lambda_\theta^n(\mathbf{s})$ for a linear function L. We refer to this straightforward modification as *actor norm*. Our motivation has been to stabilize the actor output, but rapidly changing critic output might also be harmful and the critic network could similarly benefit from normalizing penultimate features. We refer to the strategy of normalizing both the actor and the critic as *both norm*. We also consider *layer norm* (Ba et al., 2016) applied similarly to both the actor and critic. At last, following Gogianu et al. (2021), we consider applying *spectral* normalization (Miyato et al., 2018) to the second to last layer.

Penalizing saturating actions. Another strategy for avoiding actions with large magnitude is to add a term to the loss function which directly penalizes large actions. One can imagine many such loss functions, we here consider the simple and natural choice of squared loss:

$$\ell(\theta, \mathbf{s}) = \lambda \|\mathbf{a}_\theta^{pre}(\mathbf{s})\|^2 \tag{4}$$

We can simply add this term to the loss function times some hyperparameter λ. By tuning λ, one should be able to remove the saturating actions illustrated in Figure 3. We heuristically find that $\lambda = 0.000001$ works well in practice (see Appendix A) and use this value. We refer to this modification as *penalty*. Since the Adam optimizer is invariant under re-scaling the loss, the specific value of λ has a small effect as long as $\nabla_\theta Q_\phi(\mathbf{s}, \mathbf{a}_\theta(\mathbf{s})) \approx 0$ (which happen as per Figure 3).

Learning rate warmup. One common strategy to improve stability is to start the learning rate at zero and increase it linearly to its top value during early training (Goyal et al., 2017). This can improve numerical stability (Ma & Yarats, 2019). We consider using this strategy and linearly increasing the learning rate from zero over the first 10,000 steps. We refer to this strategy as *lr warmup*.

Table 2: Performance for various stability improvements. These methods often decrease variance, especially for the walker task where variance decreases by orders of magnitude. Larger stability often leads to higher reward, likely as poor outlier runs are eliminated. Normalizing features gives consistent improvements. Metrics are calculated over 10 seeds.

metric	method	acrobot	finger turn	hopper hop	reacher	walker	avg.
μ	baseline	122.0	278.6	176.2	678.0	769.1	404.8
μ	penalty	128.0	431.6	241.7	584.6	956.3	468.4
μ	actor norm	117.8	372.7	240.5	701.7	956.7	477.9
μ	both norm	192.2	605.8	188.7	892.7	961.5	**568.2**
μ	layer norm	43.5	661.2	218.6	659.7	952.4	507.1
μ	lr warmup	150.3	265.4	173.7	660.9	670.4	384.1
μ	grad clip 1	138.2	214.0	206.0	520.0	675.6	350.7
μ	grad clip 10	79.3	166.2	157.7	548.4	678.7	326.0
μ	spectral	104.0	101.0	201.0	213.6	775.6	279.1
μ	scale down	127.2	258.4	111.7	508.7	386.4	278.5
σ	baseline	76.0	221.2	90.3	167.3	350.1	181.0
σ	penalty	42.1	160.6	46.5	236.8	15.4	100.3
σ	actor norm	39.9	227.9	42.7	101.9	8.0	**84.1**
σ	both norm	60.1	193.4	78.4	122.5	7.0	92.3
σ	layer norm	57.2	228.4	112.5	297.1	13.5	141.7
σ	lr warmup	53.8	133.6	98.2	133.4	423.4	168.5
σ	grad clip 1	60.6	127.1	92.4	231.9	425.3	187.5
σ	grad clip 10	56.1	143.9	113.6	243.2	426.3	196.6
σ	spectral	37.0	61.4	75.0	224.8	128.0	105.2
σ	scale down	27.8	186.9	96.6	241.3	444.0	199.3

Small weights final layer. Following Andrychowicz et al. (2020), we consider using small weights for the final layers of the actor and the critic. Specifically, we downscale these weights by a factor of 100 at initialization time. We refer to this strategy as *scale down*.

Gradient clipping. Another stabilizing strategy is gradient clipping (Zhang et al., 2019). To avoid occasional exploding gradients, one simply clips the norm of gradients that are above some threshold. This strategy is popular in NLP (Gehring et al., 2017; Peters et al., 2018) and is sometimes used in RL (Raffin et al., 2019). We consider clipping the gradient when the norm, calculated independently for the actor, critic, and convolutional encoder, is larger than 1 or 10. We refer to these strategies as *grad clip 1* and *grad clip 10*.

Results. In Table 2 we evaluate the effects of these ideas. Normalizing activations has the best performance and lowest variability on average, whereas layer norm can make performance worse. The performance improvement coming from normalizing the critic suggests that the critic network also suffers from instabilities. The penalty method improves the performance relative to the baseline. Other methods decrease average reward, and lowering variance by using a poor policy is not hard to achieve and not particularly desirable. These experiments show that it is possible to significantly decrease the variance of the baseline agent by applying stabilization methods. Additionally, decreasing the variance often increases the average reward – likely as bad outlier runs are eliminated.

4.1 Combining Fixes

We have found that many of the proposed methods can decrease variance and improve average reward. For optimal performance, we now consider combining three orthogonal methods which improve performance individually – actor and critic penultimate normalization, pre-tanh penalty, and self-supervised learning (we apply it to the first 10,000 steps). In Table 3 we compare the performance between the baseline and an agent using these three modifications (which we refer to as *combined*) over 40 seeds. We see that the variances go down while the average score goes up. In fact, for the walker task, the variance decreases by orders of magnitude. Eliminating poor outlier runs seems to benefit the average performance in addition to decreasing variance. While our methods have mostly been motivated by outlier runs that fail catastrophically, they likely help more normal runs too. Learning curves for the combined agent are given in Appendix A.

Table 3: Performance when combining three improvements: penultimate feature normalization, action penalty and early self-supervised learning. The variance decreases and the performance improves across all tasks. This demonstrates that one can substantially improve variance in RL experiments by simply improving numerical stability. Metrics are calculated over 40 seeds.

metric	method	acrobot	finger turn	hopper hop	reacher	walker	avg.
μ	baseline	122.0	278.6	176.2	678.0	769.1	404.8
μ	combined	227.7	653.5	231.1	893.4	964.2	**594.0**
σ	baseline	76.0	221.2	90.3	167.3	350.1	181.0
σ	combined	59.0	183.1	40.8	120.0	6.2	**81.8**

4.2 Additional Tasks

We now apply the combined agent of Table 3 to additional tasks. Specifically, we consider all 21 tasks identified as easy or medium by Yarats et al. (2021b). Note that the combined agent has been constructed using only 5 of these tasks, so this experiment measures generalization. For computational reasons, the additional 16 tasks only use 10 seeds each. We use the baseline hyperparameters across all tasks without tuning (Yarats et al. (2021b) tunes parameters on a per-task basis). Results are shown in Figure 4. Our methods consistently improve the average reward while the average standard deviation decreases by a factor > 2. Furthermore, a few additional tasks (finger spin, hopper stand, walker run, walker stand) have their variance decreased by an order of magnitude or more. The tasks cheetah run and quadruped walk perform worse than the baseline, although this could potentially be alleviated by tuning hyperparameters. Just improving learning rate on these tasks improve performance, see Figure 12 in Appendix A. These results demonstrate that our modifications significantly decrease variance and that directly optimizing variance can be feasible and fruitful.

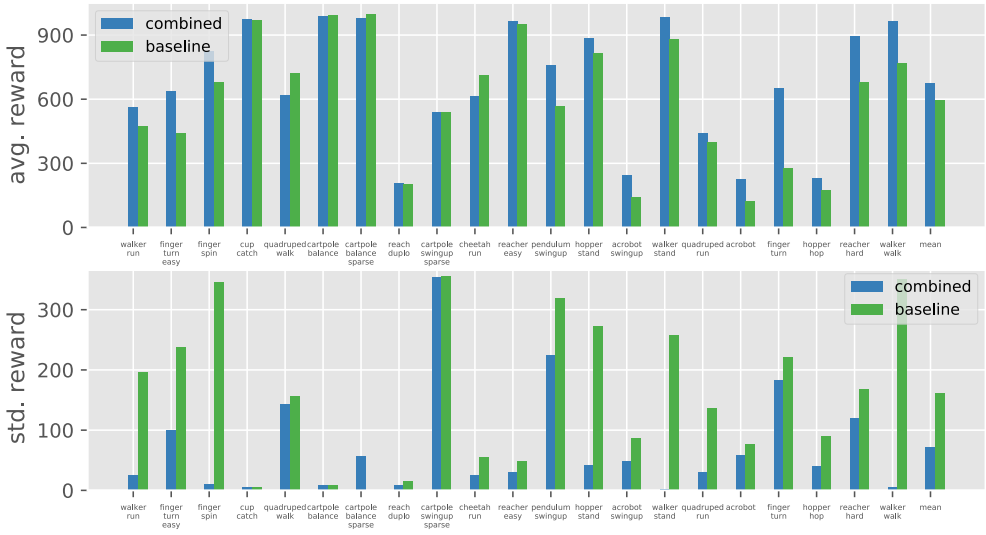

Figure 4: Performance for 21 tasks. (**Top**) Average reward per task. The combined agent improve the average reward, and most tasks benefit from our modifications. (**Bottom**) Standard deviation of reward, broken up by task. The combined agent have less variability, and for a few tasks the variance decreases by orders of magnitude. Except for measurements in Table 3, we use 10 seeds.

Consider the five tasks in Figure 4 where combined has the highest standard deviation. Four out of these have sparse rewards – finger turn hard, reacher hard, pendulum swingup, and cartpole swingup sparse. Once stability has been addressed, sparsity seems to be a major driver of residual variance. In Appendix C we consider further modification to improve performance specifically for sparse tasks. We find that removing the double clipped Q-learning for the critic loss and extending the self-supervision helps for these tasks. With these changes, the agent achieves even better performance and an average standard deviation which is > 3 times lower than the baseline.

4.3 Learning rate stability

To verify that our fixes indeed increase the stability of learning we now consider changing the learning rate. A high learning rate can cause divergence (Li et al., 2019), and can thus serve as a proxy for learning stability. We consider learning rates in the set $\{1\mathrm{e}{-4}, 3\mathrm{e}{-4}, 1\mathrm{e}{-3}, 3\mathrm{e}{-3}\}$, where $1\mathrm{e}{-4}$ is the default learning rate used by Yarats et al. (2021b). Results for the walker task are shown in Table 4. We see that the baseline algorithm fails for large learning rates, the average rewards drop quickly while variance decreases when no effective policy is learned. With our modifications, the agent can tolerate much higher learning rates while retaining good average rewards and small standard deviation. Finally, to verify that the actions do not saturate when using our fixes, we repeat the experiments of Figure 3 with the combined agent. The actions are shown in Figure 5, where we see that the action magnitude indeed goes up quickly early in training but never saturates. This supports our hypothesis that normalizing activations can stabilize training.

Table 4: Performance when changing learning rates. The baseline algorithm fails with larger learning rates whereas the combined agent succeeds. This suggests that our fixes improve the stability. Metrics are calculated over 10 seeds on the walker task.

metric	method	$1\mathrm{e}{-4}$	$3\mathrm{e}{-4}$	$1\mathrm{e}{-3}$	$3\mathrm{e}{-3}$
μ	baseline	769.1	330.2	26.8	25.5
μ	combined	964.2	962.2	954.9	960.1
σ	baseline	350.1	415.3	3.9	6.4
σ	combined	6.2	4.7	5.6	5.1

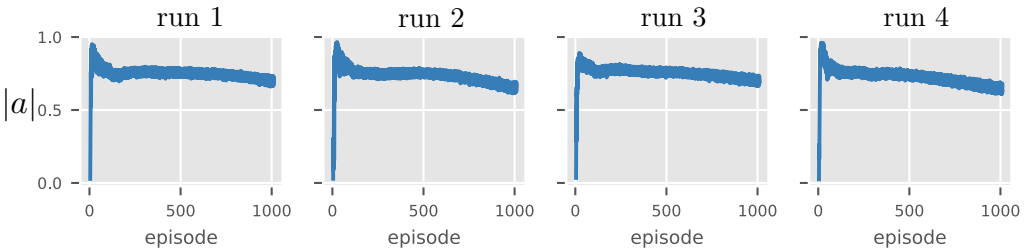

Figure 5: The average absolute value of the actions for the combined agent. We show four representative runs of walker task. Compared to the baseline algorithm in Figure 3, the actions do not saturate early in training despite increasing rapidly, indicating improved stability.

5 Related work

Variance in RL. Many researchers have studied variance in RL: Engstrom et al. (2020) demonstrates variability from small implementation details; Henderson et al. (2017) shows how even hardware non-determinism can cause variability; Islam et al. (2017) studies reproducibility of benchmarking in RL; Mania et al. (2018) demonstrates variability for random search strategies; Clary et al. (2019) studies variability in the Atari suite; Lynnerup et al. (2020) studies variance in real-world robots. Given this high variance, there is ample work on how to accurately measure performance in RL (Colas et al., 2019). Chan et al. (2019) proposes several metrics that can measure different notations of reliability in reinforcement learning. Jordan et al. (2020) proposes an evaluation methodology that takes into account the effects of hyperparameter tuning. Recently Agarwal et al. (2021) proposes new robust estimates of performance. Given this plethora of proposed metrics, we simply report average mean and standard deviation which is commonly reported in practice. Our strategy of simply reducing the variance of RL algorithms is complementary to developing robust metrics.

Stabilizing RL. Within Q-learning, it is known that high variance Q-estimates can impede learning (Fujimoto et al., 2018). Hence, there is a large amount of work on improving the Q-estimates for DQN: averaging weights (Nikishin et al., 2018), averaging Q-estimators (Anschel et al., 2017), using classical variance reduction methods (Jia et al., 2020) and regretted reward (Chen et al., 2018). These papers focus on Q-value variance instead of performance variance and apply to the discrete DQN setting, distinct from continuous control. In policy learning there are several methods for decreasing variance of gradient estimates (Greensmith et al., 2004; Wang et al., 2017), however, these do not directly apply to DDPG. On the architectural side, Parisotto et al. (2020) stabilizes transformers for RL. On the task-specific side, Kumar et al. (2019) stabilizes learning for offline tasks while Mao et al. (2018) considers reducing gradient variance for externally driven environments. Normalization strategies in RL have been investigated by Salimans & Kingma (2016) and Gogianu et al. (2021).

6 Discussion

Limitations. We have considered a single algorithm on a single suite of tasks. Our setting uses a state-of-the-art method on a popular benchmark relatively close to real-world applications. Nonetheless, many important settings remain, e.g. model-based algorithms (Hafner et al., 2020) and discrete environments (Bellemare et al., 2013). This paper shows that *it is possible* to decrease variance with simple modifications, although which methods are effective likely depends upon the base algorithm. We leave evaluating how well our modifications transfer to other settings for future work.

Conclusion. We have investigated causes of variance in a popular continuous control setup. Whereas feature learning and network initialization do not appear to be at fault, we observe that learning instability causes significant variance. We propose several fixes to this issue and demonstrate across 21 tasks that these dramatically decrease variance while improving average rewards for a state-of-the-art agent. Our experiments demonstrate that the perception that RL has intrinsically high variance might need to be revisited – in some cases, numerical issues might instead be at fault.

Ethics Statement. There are many positive RL applications – examples include safe autonomous cars, accelerated scientific discovery, and medical testing. Studying causes of high variance, and proposing fixes to this issue, can benefit real-world adoption of RL techniques. Additionally, reducing the variance of RL experiments would aid the research community as reproducibility can be problematic in the field. However, there are also many potentially harmful military applications. Our study is academic in nature and only considers simulated toy environments, but insights from our work could nonetheless be useful in harmful applications. We do not believe that our work differs from other RL work significantly concerning ethics.

Reproducability Statement. To aid the reproducibility of our work, anonymized code for the experiments is available [1]. Furthermore, in Appendix A we detail the infrastructure and software used. The perhaps most important component of generating reproducible findings in RL is to use many seeds. To this end, we only consider five environments but evaluated these thoroughly. The main experiments of the paper (Table 3) use 40 seeds whereas all others use 10 seeds.

REFERENCES

Joshua Achiam. Spinning Up in Deep Reinforcement Learning. 2018.

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron Courville, and Marc G Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *arXiv preprint arXiv:2108.13264*, 2021.

Marcin Andrychowicz, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphael Marinier, Léonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, et al. What matters in on-policy reinforcement learning? a large-scale empirical study. *arXiv preprint arXiv:2006.05990*, 2020.

Oron Anschel, Nir Baram, and Nahum Shimkin. Averaged-dqn: Variance reduction and stabilization for deep reinforcement learning. In *International conference on machine learning*, pp. 176–185. PMLR, 2017.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Philip Bachman, R Devon Hjelm, and William Buchwalter. Learning representations by maximizing mutual information across views. *arXiv preprint arXiv:1906.00910*, 2019.

Hamsa Bastani, Kimon Drakopoulos, Vishal Gupta, Jon Vlachogiannis, Christos Hadjicristodoulou, Pagona Lagiou, Gkikas Magiorkinis, Dimitrios Paraskevis, and Sotirios Tsiodras. Efficient and targeted covid-19 border testing via reinforcement learning, 2021.

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47: 253–279, 2013.

Marc G Bellemare, Salvatore Candido, Pablo Samuel Castro, Jun Gong, Marlos C Machado, Subhodeep Moitra, Sameera S Ponda, and Ziyu Wang. Autonomous navigation of stratospheric balloons using reinforcement learning. *Nature*, 588(7836):77–82, 2020.

Xavier Bouthillier, Pierre Delaunay, Mirko Bronzi, Assya Trofimov, Brennan Nichyporuk, Justin Szeto, Nazanin Mohammadi Sepahvand, Edward Raff, Kanika Madan, Vikram Voleti, et al. Accounting for variance in machine learning benchmarks. *Proceedings of Machine Learning and Systems*, 3, 2021.

Stephanie CY Chan, Samuel Fishman, John Canny, Anoop Korattikara, and Sergio Guadarrama. Measuring the reliability of reinforcement learning algorithms. *arXiv preprint arXiv:1912.05663*, 2019.

[1] https://drive.google.com/drive/folders/1SvTjN6I8Q8MA8fMy1RyuBX005npemupw?usp=sharing

Shi-Yong Chen, Yang Yu, Qing Da, Jun Tan, Hai-Kuan Huang, and Hai-Hong Tang. Stabilizing reinforcement learning in dynamic environment with application to online recommendation. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1187–1196, 2018.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pp. 1597–1607. PMLR, 2020.

Kaleigh Clary, Emma Tosch, John Foley, and David Jensen. Let's play again: Variability of deep reinforcement learning agents in atari environments. *arXiv preprint arXiv:1904.06312*, 2019.

Cédric Colas, Olivier Sigaud, and Pierre-Yves Oudeyer. How many random seeds? statistical power analysis in deep reinforcement learning experiments. *arXiv preprint arXiv:1806.08295*, 2018.

Cédric Colas, Olivier Sigaud, and Pierre-Yves Oudeyer. A hitchhiker's guide to statistical comparisons of reinforcement learning algorithms. *arXiv preprint arXiv:1904.06979*, 2019.

Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. Implementation matters in deep policy gradients: A case study on ppo and trpo. *arXiv preprint arXiv:2005.12729*, 2020.

Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pp. 1587–1596. PMLR, 2018.

Javier Garcıa and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. In *International Conference on Machine Learning*, pp. 1243–1252. PMLR, 2017.

Florin Gogianu, Tudor Berariu, Mihaela Rosca, Claudia Clopath, Lucian Busoniu, and Razvan Pascanu. Spectral normalisation for deep reinforcement learning: an optimisation perspective. *arXiv preprint arXiv:2105.05246*, 2021.

Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

Evan Greensmith, Peter L Bartlett, and Jonathan Baxter. Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research*, 5(9), 2004.

Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, et al. Bootstrap your own latent: A new approach to self-supervised learning. *arXiv preprint arXiv:2006.07733*, 2020.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*, 2020.

Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. *arXiv preprint arXiv:1911.05722*, 2019.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. *arXiv preprint arXiv:1709.06560*, 2017.

Riashat Islam, Peter Henderson, Maziar Gomrokchi, and Doina Precup. Reproducibility of benchmarked deep reinforcement learning tasks for continuous control. *arXiv preprint arXiv:1708.04133*, 2017.

Haonan Jia, Xiao Zhang, Jun Xu, Wei Zeng, Hao Jiang, Xiaohui Yan, and Ji-Rong Wen. Variance reduction for deep q-learning using stochastic recursive gradient. *arXiv preprint arXiv:2007.12817*, 2020.

Scott Jordan, Yash Chandak, Daniel Cohen, Mengxue Zhang, and Philip Thomas. Evaluating the performance of reinforcement learning algorithms. In *International Conference on Machine Learning*, pp. 4962–4973. PMLR, 2020.

B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A Al Sallab, Senthil Yogamani, and Patrick Pérez. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 2021.

Ilya Kostrikov, Denis Yarats, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. 2020.

Aviral Kumar, Justin Fu, George Tucker, and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. *arXiv preprint arXiv:1906.00949*, 2019.

Michael Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. Reinforcement learning with augmented data. *arXiv preprint arXiv:2004.14990*, 2020a.

Michael Laskin, Aravind Srinivas, and Pieter Abbeel. Curl: Contrastive unsupervised representations for reinforcement learning. In *Proceedings of the 37th Annual International Conference on Machine Learning (ICML)*, 2020b.

Yuanzhi Li, Colin Wei, and Tengyu Ma. Towards explaining the regularization effect of initial large learning rate in training neural networks. *arXiv preprint arXiv:1907.04595*, 2019.

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 preprint arXiv:1509.02971*, 2015.

Yan Liu, Zhiyuan Jiang, Shunqing Zhang, and Shugong Xu. Deep reinforcement learning-based beam tracking for low-latency services in vehicular networks. In *ICC 2020-2020 IEEE International Conference on Communications (ICC)*, pp. 1–7. IEEE, 2020.

Nicolai A Lynnerup, Laura Nolling, Rasmus Hasle, and John Hallam. A survey on reproducibility by evaluating deep reinforcement learning algorithms on real-world robots. In *Conference on Robot Learning*, pp. 466–489. PMLR, 2020.

Jerry Ma and Denis Yarats. On the adequacy of untuned warmup for adaptive optimization. *arXiv preprint arXiv:1910.04209*, 7, 2019.

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

Hongzi Mao, Shaileshh Bojja Venkatakrishnan, Malte Schwarzkopf, and Mohammad Alizadeh. Variance reduction for reinforcement learning in input-driven environments. *arXiv preprint arXiv:1807.02264*, 2018.

Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*, 2018.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Evgenii Nikishin, Pavel Izmailov, Ben Athiwaratkun, Dmitrii Podoprikhin, Timur Garipov, Pavel Shvechikov, Dmitry Vetrov, and Andrew Gordon Wilson. Improving stability in deep reinforcement learning with weight averaging. In *Uncertainty in artificial intelligence workshop on uncertainty in Deep learning*, 2018.

Emilio Parisotto, Francis Song, Jack Rae, Razvan Pascanu, Caglar Gulcehre, Siddhant Jayakumar, Max Jaderberg, Raphael Lopez Kaufman, Aidan Clark, Seb Noury, et al. Stabilizing transformers for reinforcement learning. In *International Conference on Machine Learning*, pp. 7487–7498. PMLR, 2020.

Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

Antonin Raffin, Ashley Hill, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, and Noah Dormann. Stable baselines3. `https://github.com/DLR-RM/stable-baselines3`, 2019.

Tim Salimans and Diederik P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *arXiv preprint arXiv:1602.07868*, 2016.

Max Schwarzer, Ankesh Anand, Rishab Goel, R Devon Hjelm, Aaron Courville, and Philip Bachman. Data-efficient reinforcement learning with momentum predictive representations. *arXiv preprint arXiv:2007.05929*, 2020.

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.

Krishnan Srinivasan, Benjamin Eysenbach, Sehoon Ha, Jie Tan, and Chelsea Finn. Learning to be safe: Deep rl with a safety critic. *arXiv preprint arXiv:2010.14603*, 2020.

Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3 (1):9–44, 1988.

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

Yuval Tassa, Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, and Nicolas Heess. dm control: Software and tasks for continuous control, 2020.

Tongzhou Wang, Yi Wu, David A Moore, and Stuart J Russell. Meta-learning mcmc proposals. *arXiv preprint arXiv:1708.06040*, 2017.

Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. 1989.

Denis Yarats, Rob Fergus, Alessandro Lazaric, and Lerrel Pinto. Mastering visual continuous control: Improved data-augmented reinforcement learning. *arXiv preprint arXiv:2107.09645*, 2021a.

Denis Yarats, Rob Fergus, Alessandro Lazaric, and Lerrel Pinto. Mastering visual continuous control: Improved data-augmented reinforcement learning. *arXiv preprint arXiv:2107.09645*, 2021b.

Albert Zhan, Philip Zhao, Lerrel Pinto, Pieter Abbeel, and Michael Laskin. A framework for efficient robotic manipulation. *arXiv preprint arXiv:2012.07975*, 2020.

Jingzhao Zhang, Tianxing He, Suvrit Sra, and Ali Jadbabaie. Why gradient clipping accelerates training: A theoretical justification for adaptivity. *arXiv preprint arXiv:1905.11881*, 2019.

Henry Zhu, Justin Yu, Abhishek Gupta, Dhruv Shah, Kristian Hartikainen, Avi Singh, Vikash Kumar, and Sergey Levine. The ingredients of real-world robotic reinforcement learning. *arXiv preprint arXiv:2004.12570*, 2020.

A APPENDIX

Tuning λ. For tuning the hyperparameters λ defined in equation 4 we search over the set $\{1\text{e}{-1}, 1\text{e}{-2}, 1\text{e}{-3}, 1\text{e}{-4}, 1\text{e}{-5}, 1\text{e}{-6}\}$. For each value, we run the hopper task for a short amount of time with five different seeds. We found that too large λ penalized actions too much, and that even very small values like $1\text{e}{-6}$ sufficed to give improvement.

Details on self-supervised Learning. We extract features from the convolutional network and the first feedforward layer of the critic, which projects the feature map to 50-dimensions. Feature vectors are normalized to unit norm before being fed into the self-supervised loss (Bachman et al., 2019). We use exponentially moving weights for the target network (He et al., 2019), using the same decay parameters as for the target critic. We use a *head* for the online network which is simply a two-layer MLP with hidden dimension 512 and final dimension 50 (Chen et al., 2020). The two augmentations we consider are simply two adjacent frames with augmentations that the DDPG agent of Yarats et al. (2021b) considers naturally. We use the square loss as proposed in Grill et al. (2020), which has been shown to be useful in smaller batch sizes. The self-supervised loss is then simply added to the normal loss function for the critic.

Infrastructure. We run our experiments on Nvidia Tesla V100 GPUs and Intel Xeon CPUs. The GPUs use CUDA 11.1 and CUDNN 8.0.0.5. We use PyTorch 1.9.0 and python 3.8.10. Our experiments are all based upon the open-source implementation of Yarats et al. (2021b).

Table 5: Variance across tasks as reported by Yarats et al. (2021b). Mean reward and task variance often increase in tandem – to avoid bias towards easy tasks with high mean and thus high variance, we rank tasks based upon relative variance σ/μ. We use the five tasks with the highest relative variance, removing one duplicate: finger turn easy/hard. Used tasks are highlighted. Scores are measured just before 1 million frames (i.e. frame 980000) since not all runs have scores at the 1 million mark. All numbers are rounded to two decimal digits.

task	σ/μ	μ	σ
finger turn hard	0.74	268.90	198.36
walker walk	0.48	769.66	371.48
hopper hop	0.36	221.13	80.63
finger turn easy	0.31	558.52	174.19
acrobot swingup	0.31	177.65	54.77
reacher hard	0.22	628.29	135.57
walker run	0.20	538.59	110.02
cup catch	0.13	909.95	118.06
finger spin	0.12	860.30	104.18
quadruped run	0.12	446.49	53.51
quadruped walk	0.12	732.52	87.19
cartpole balance sparse	0.12	962.30	113.10
reach duplo	0.08	202.51	16.99
cartpole swingup sparse	0.08	760.11	60.11
cheetah run	0.07	710.24	52.43
reacher easy	0.05	931.54	50.86
pendulum swingup	0.04	838.49	36.38
hopper stand	0.02	917.85	20.45
cartpole swingup	0.02	864.68	15.47
walker stand	0.01	980.72	5.21
cartpole balance	0.01	993.45	5.00

Figure 6: Learning curves for the two tasks not shown in Figure 1. The upper row shows the mean reward with one standard deviation indicated by shading. The bottom row show learning curves for individual runs. Although the signal is noisy, several outlier runs fail to learn.

Table 6: Hyperparameters used throughout the paper. These follow Yarats et al. (2021b) for the *medium* tasks.

parameter	value
learning rate	1e−4
soft-update τ	1e−2
update frequency	2
stddev. schedule	linear(1.0, 0.1, 500000)
stddev. clip	0.3
action repeat	2
seed frames	4e3
batch size	256
n-step returns	3
discount γ	0.99

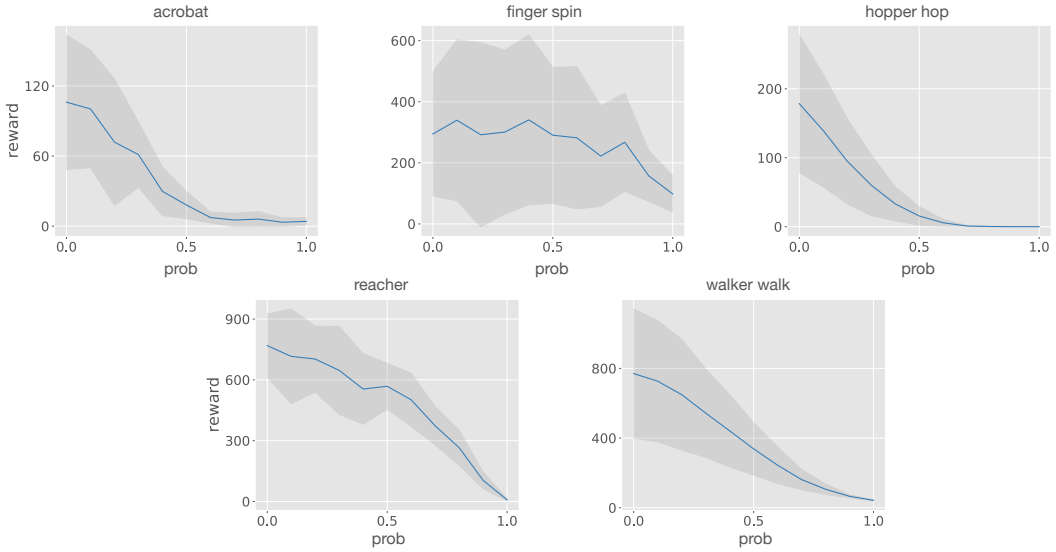

Figure 7: Rewards obtained when gradually replacing a learned policy with a random policy. At every time-step t, the action suggested by the learned policy is replaced with a random action with probability p. We start from the final policy learned by the baseline algorithm and average over ten starting policies. Across all tasks, the rewards decrease gracefully with a more random policy, suggesting that the environments are robust to perturbed policies.

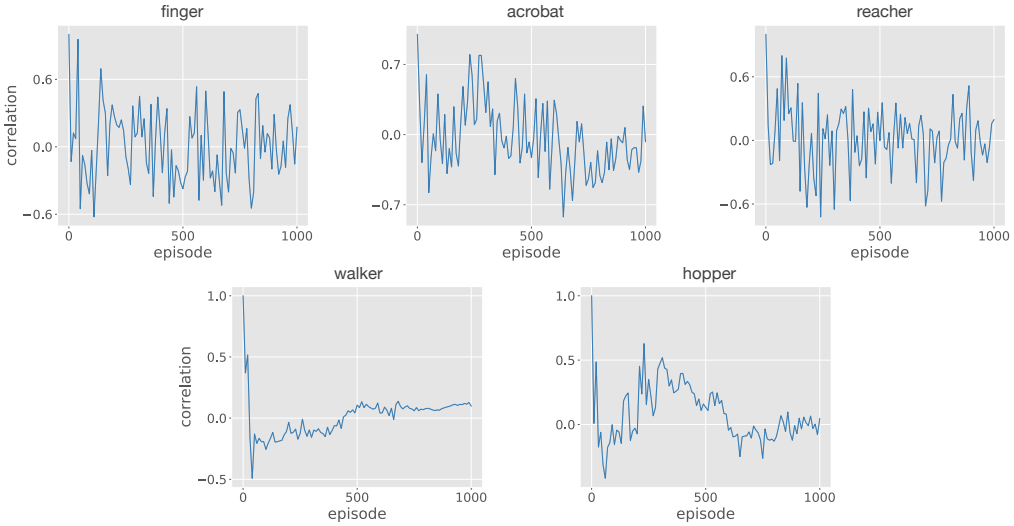

Figure 8: Correlation between the evaluation scores of two runs that share network initialization and seed experience, but otherwise have different randomness. Correlation is calculated across ten runs and plotted against time. The correlation is noisy for many environments but oscillates around zero, suggesting that there is little actual correlation and that network initialization and seed experience have a small effect.

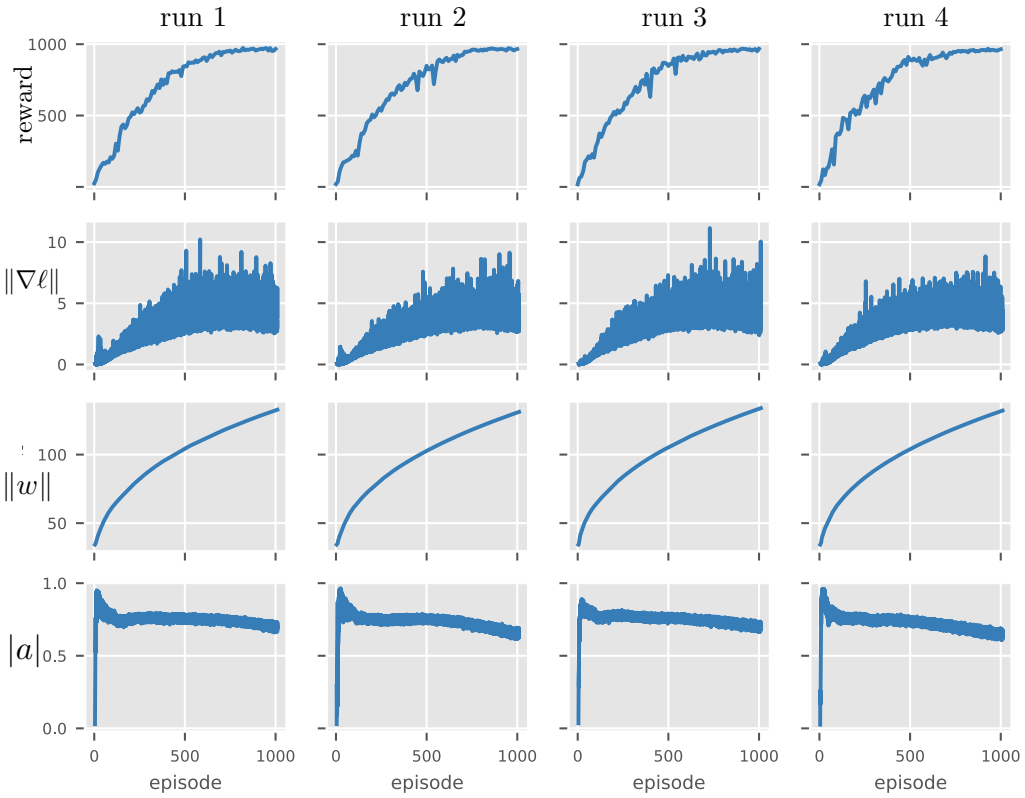

Figure 9: Four quantities shown during training for the combined agent. We show reward, norm of actor gradient $\|\nabla\ell\|$, norm of actor weight $\|w\|$ and average absolute action $|\mathbf{a}|$. The actions never saturate and learning starts quickly. Compare with Figure 3, which shows the same quantities for the baseline algorithm.

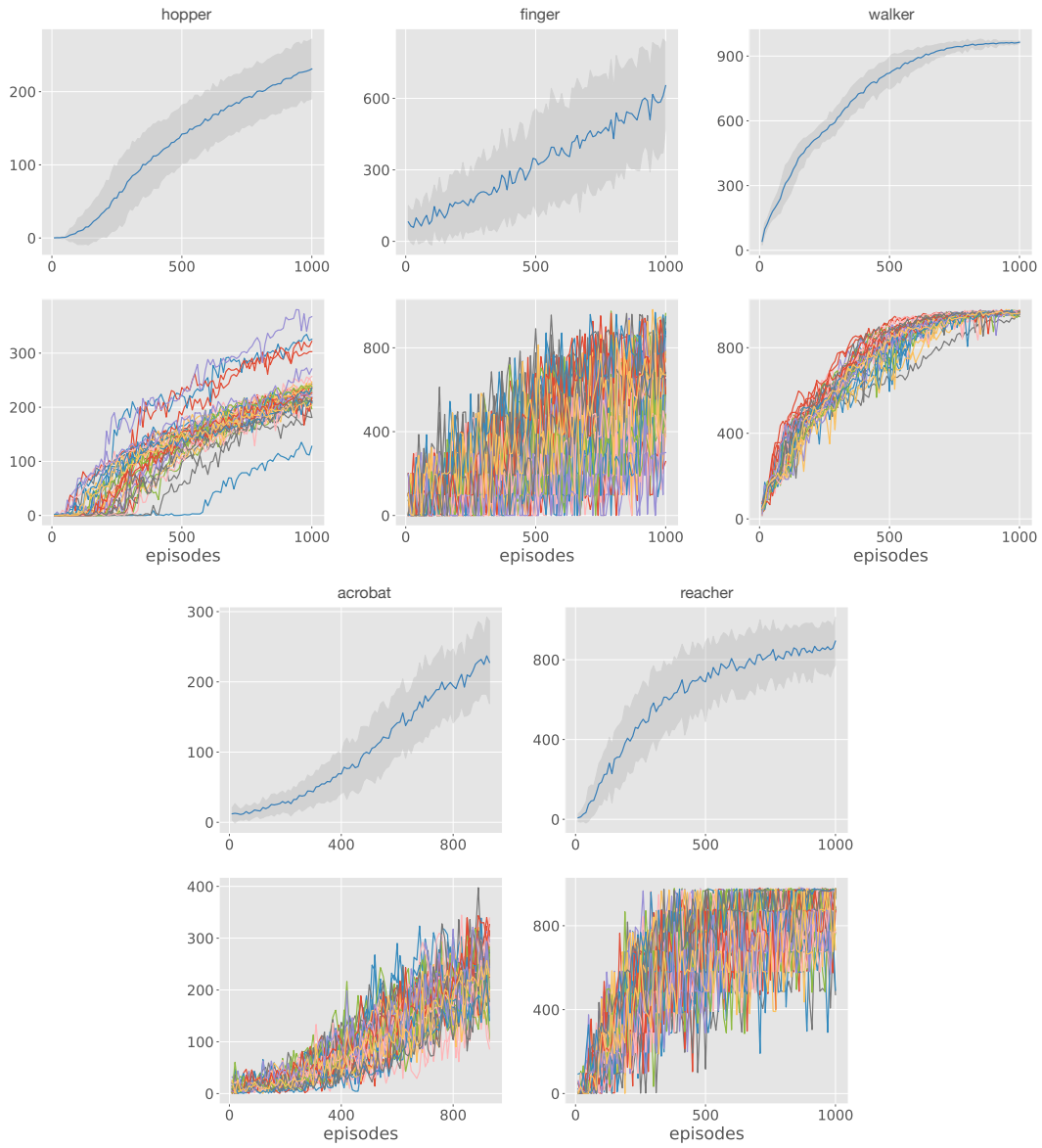

Figure 10: Learning curves when using our combined agent. We show mean reward with one standard deviation indicated by shading and learning curves for individual runs.

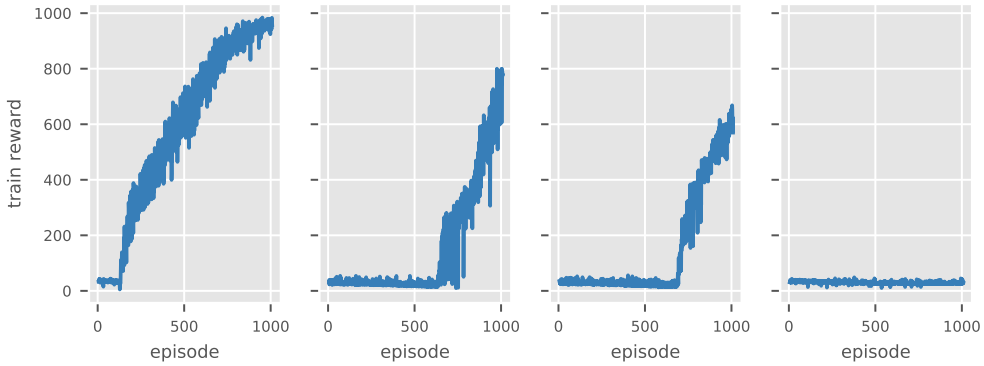

Figure 11: Rewards obtained during training for the runs illustrated in Figure 3. Note that the rewards are nonzero from the start of training. This implies that the agent observes rewards and can train on these. Thus, lack of rewards is not the cause of poor performance.

Figure 12: The performance of the combined agents on two tasks when increasing the learning rate by a factor of 10. Performance improves markedly on these tasks. This highlights how hyperparameter tuning could further improve the results of Figure 4.

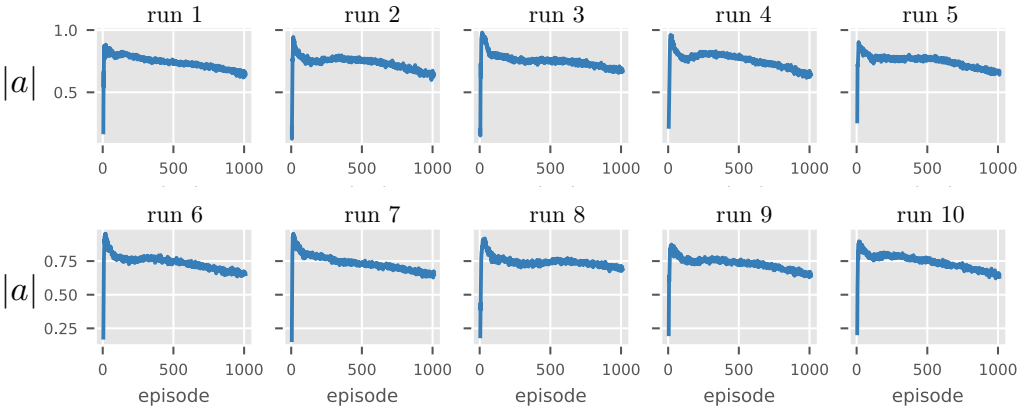

Figure 13: We repeat Figure 5 for all 10 seeds of the norm agent of Table 2. Note that the action distribution does not saturate.

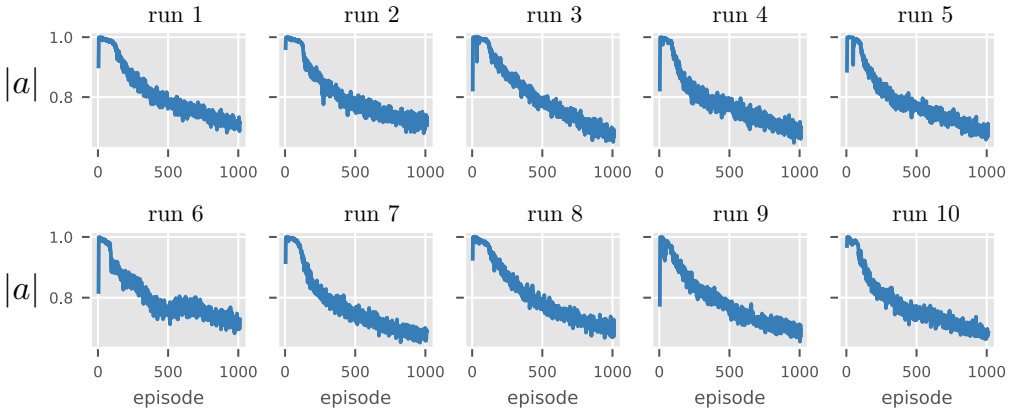

Figure 14: We repeat Figure 5 for all 10 seeds of the penalty agent of Table 2. Note that the action distribution does not saturate.

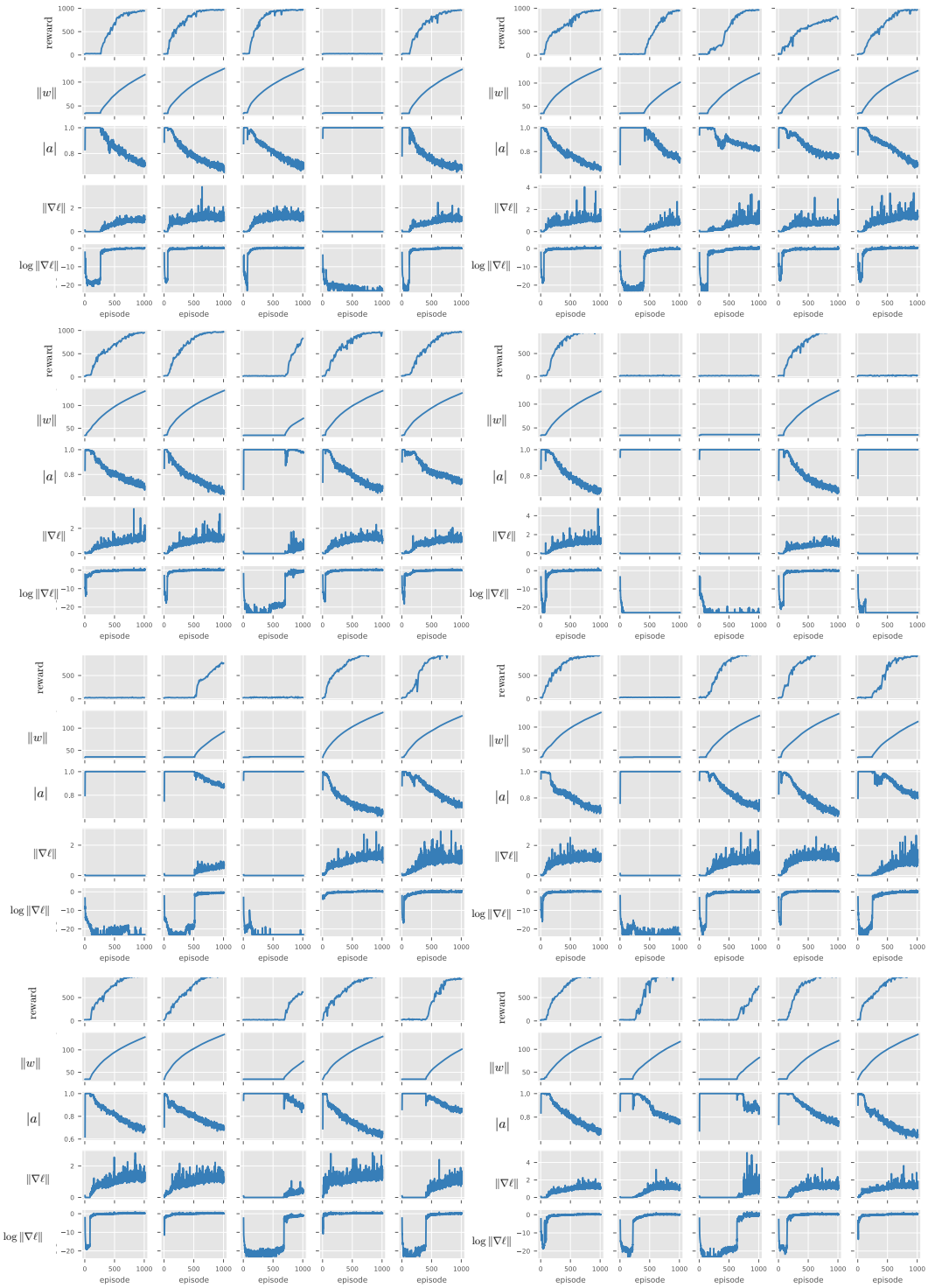

Figure 15: We repeat Figure 3 for all 40 seeds. The plots show five quantities for the baseline agent of the walker task: reward, absolute average action $|a|$, actor weight norm $\|w\|$, actor gradient norm $\|\nabla \ell\|$ and the logarithm of the actor gradient norm $\log \|\nabla \ell\|$. We add $1\mathrm{e}{-10}$ to the logarithm for numerical stability. Results are similar to Figure 3.

B EVALUATION VARIANCE

We will now consider the variance arising from evaluating RL agents on a finite set of episodes. Similar to the bias-variance decomposition, we will see that the variance here can be divided into one part that depends upon the randomness of the algorithm, and one term that depends on the randomness from the finite set of episodes we evaluate on. We let R be a set of training runs and S be a set of evaluation episodes sampled. The performance is measured by

$$\text{perf} = \frac{1}{|R||S|}\sum_{i\in R}\sum_{j\in S} r_{ij}$$

Let us calculate the variance of this quantity. We will assume that rewards from different runs are independent, and that rewards from the same training run are independent conditioned on that run. We now have

$$\text{Var}\Big(\frac{1}{|R||S|}\sum_{i\in R}\sum_{j\in S} r_{ij}\Big)$$

The variance of a sum of independent variables is simply the sum of their variances. Thus we have

$$\text{Var}\Big(\frac{1}{|R||S|}\sum_{i\in R}\sum_{j\in S} r_{ij}\Big) = \frac{1}{|R|}\text{Var}\Big(\frac{1}{|S|}\sum_{j\in S} r_{ij}\Big)$$

Let $\mu = \mathbb{E}[r_{ij}]$. We then have

$$\text{Var}\Big(\frac{1}{|S|}\sum_{j\in S} r_{ij}\Big) = \mathbb{E}_{alg,samples}\Big[\Big(\frac{1}{|S|}\sum_{j\in S} r_{ij} - \mu\Big)^2\Big]$$

Here we explicitly calculate the expectation over random training runs (*alg*) and the random episodes sampled (*samples*). We can condition the expectation upon the random training run and then integrate over this expectation. Furthermore, let $\mu_i = \mathbb{E}[r_{ij}|i]$, i.e. the expected value an episode sampled from training run i. We then have

$$\text{Var}\Big(\frac{1}{|S|}\sum_{j\in S} r_{ij}\Big) = \mathbb{E}_{alg}\Big[\mathbb{E}_{samples}\Big[\Big(\frac{1}{|S|}\sum_{j\in S} r_{ij} - \mu\Big)^2\Big|i\Big]\Big]$$

$$= \mathbb{E}_{alg}\Big[\mathbb{E}_{samples}\Big[\Big(\frac{1}{|S|}\sum_{j\in S} r_{ij} - \mu_i + \mu_i - \mu\Big)^2\Big|i\Big]\Big]$$

$$= \mathbb{E}_{alg}\Big[\mathbb{E}_{samples}\Big[\Big(\frac{1}{|S|}\sum_{j\in S} r_{ij} - \mu_i\Big)^2\Big|i\Big]$$

$$+2\mathbb{E}_{samples}\Big[\Big(\frac{1}{|S|}\sum_{j\in S} r_{ij} - \mu_i\Big)\Big(\mu_i - \mu\Big)\Big|i\Big]$$

$$+\mathbb{E}_{samples}\Big[\Big(\mu_i - \mu\Big)^2\Big|i\Big]\Big]$$

The middle part vanishes since $\mu_i = \mathbb{E}[r_{ij}|i]$. For the first term, given a fixed training run the terms r_{ij} are independent. This gives us

Table 7: Sample variance and algorithm variance as defined in eq. (5). Some tasks have high variance and others have low. However, sampling more episodes does not decrease variance considerable in the 10 training seeds and 10 episodes setting, see eq. (6).

method	walker	hopper hop	finger turn	acrobot	reacher
sample	1922.1	450.6	143763.0	11182.9	176211.9
alg	80002.6	9774.0	55215.6	5710.9	35231.7

$$\mathbb{E}_{alg}\left[\mathbb{E}_{samples}\left[\left(\frac{1}{|S|}\sum_{j\in S} r_{ij} - \mu_i\right)^2 \middle| i\right]\right] = \frac{1}{|S|^2}\mathbb{E}_{alg}\left[\mathbb{E}_{samples}\left[\left(\sum_{j\in S} r_{ij} - |S|\mu_i\right)^2 \middle| i\right]\right]$$

$$= \frac{1}{|S|^2}\mathbb{E}_{alg}\left[\mathbb{E}_{samples}\left[\sum_{j\in S}\left(r_{ij} - \mu_i\right)^2 \middle| i\right]\right] = \frac{1}{|S|}\mathbb{E}_{alg}\left[\mathbb{E}_{samples}\left[\left(r_{ij} - \mu_i\right)^2 \middle| i\right]\right]$$

We then have

$$\text{Var(perf)} = \frac{1}{|R|}\underbrace{\mathbb{E}_{alg}[(\mu_{alg} - \mu)^2]}_{\text{algorithm variance}} + \frac{1}{|R||S|}\underbrace{\mathbb{E}_{alg}\left[\mathbb{E}_{samples}[(r - \mu_{alg})^2|i]\right]}_{\text{sample variance}} \tag{5}$$

Here $\mu_{alg} = \mu_i$. The first term is what we are interested in measuring. The second term can be large if the environment is noisy and we sample too few episodes. The second term is straightforward to estimate: run the algorithm N times and measure the performance over M samples. For each run, calculate the variance over the samples. Then average this variance over all runs.

In Table 7 we estimate these quantities over ten training runs and ten episodes for the baseline algorithm. As we can see, the environments have very different characteristics. Some have high sample variance and some have very low sample variance. It is common to evaluate with ten training seeds and ten episodes. Can we improve the variance by evaluating on more episodes, say 100? Even for the task with the highest sample noise (reacher), the relative improvement in the standard deviation of the sample variance, i.e.,

$$\sqrt{\left(\sigma^2_{alg} + \frac{1}{10}\sigma^2_{sample}\right)/\left(\sigma^2_{alg} + \frac{1}{100}\sigma^2_{sample}\right)} \approx 1.195 \tag{6}$$

is no larger than 20 %. Thus, we conclude that only minor gains can come from evaluating on more episodes. Other environments outside the five we consider, however, might benefit more from using more samples.

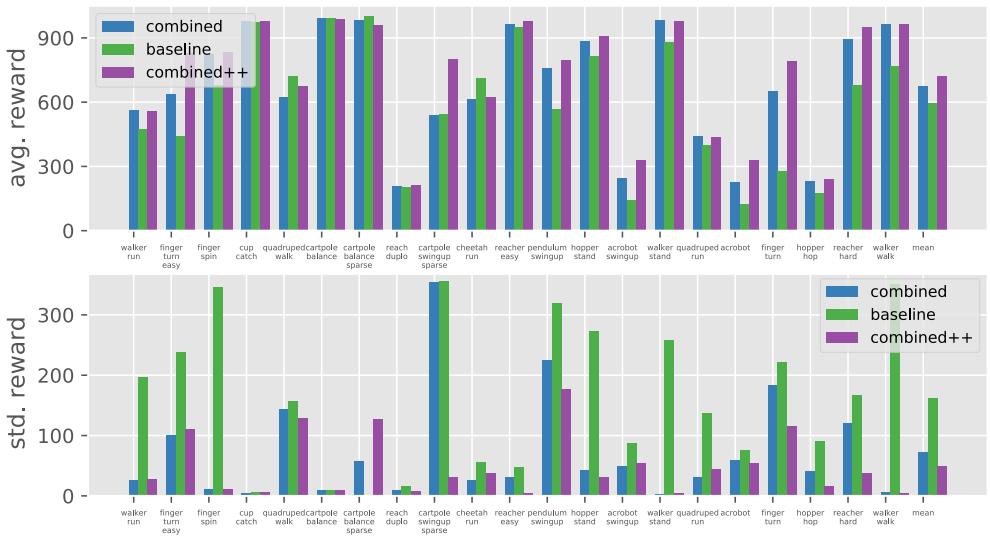

Figure 16: We repeat Figure 4, adding the combined++ agent. It further improves performance and decreases the average standard deviation by a factor > 3 compared to the baseline.

C Improvements for Sparse Tasks

In Figure 4, consider the five tasks where the combined agent suffers from the highest standard deviation. Four out of these have sparse rewards. This suggests that sparsity is a major driver of variance. To further decrease the variance, we now investigate fixes for four sparse tasks – cartpole balance sparse, pendulum swingup, reacher hard, and finger turn hard. We will consider three additional modifications which aim to specifically combat reward sparsity:

Avoiding training without rewards. For the sparse tasks, an agent might not see any rewards during the first 100 episodes. Perhaps training in the absence of any rewards can ruin the network weights. The fix we consider is simple, do not apply any gradient updates before at least one non-zero reward has been observed.

Increasing self-supervised learning. Since the agent might potentially not observe any rewards for a hundred episodes, we consider extending the self-supervised learning. This could still support a supervisory signal to the network. We simply extend the self-supervised time from 10k steps to the entire training process.

Removing clipped double q-learning. The agent of Yarats et al. (2021a) employs clipped double q-learning (Fujimoto et al., 2018) – i.e. the agent uses two critic networks Q_1, Q_2 and takes $Q = \min(Q_1, Q_2)$ to avoid overestimation. We hypothesise that the min-operator causes excessive pessimism in sparse environments where reward often are zero. For example, if $Q_1 = 0$ and $Q_2 > 0$ we have $Q = \min(Q_1, Q_2) = 0$. Thus, if one Q-network outputs only zeros and the rewards are almost only zeros, the critic target of eq. (1) will be mostly zeros and the other Q-network will be encouraged to output mostly zeros too. Thus, the Q-network could potentially get stuck with mostly outputting zeros, an issue shown in Figure 17. We thus propose to use $Q_a = \text{avg}(Q_1, Q_2)$ for defining the targets in Equation (1). We still use double clipping for the actor, i.e. the actor gradients are $\nabla_\theta \min(Q_{1\phi}(\mathbf{a}_\theta, \mathbf{s}), Q_{2\phi}(\mathbf{a}_\theta, \mathbf{s},)$ as before.

Results for these experiments are shown in Table 8. We see that decreasing pessimism and increasing self-supervised learning help the most. Increasing self-supervised learning helps the most for cartpole swingup sparse and pendulum swingup. Note that these two tasks were not used in the ablation experiments of Table 2. Additionally, whereas feature learning was not the bottleneck for the baseline agent, it might be the bottleneck for the combined agent. Avoiding training without rewards does not seem to improve performance significantly. We conclude that increased self-supervised learning and decreasing pessimism improve performance for tasks with sparse rewards. We add these two modifications to the combined agent and call the resulting agent combined++. Its performance across all 21 tasks is shown in Figure 16. It achieves even better results, and its average standard deviation is smaller than the baseline by a factor > 3.

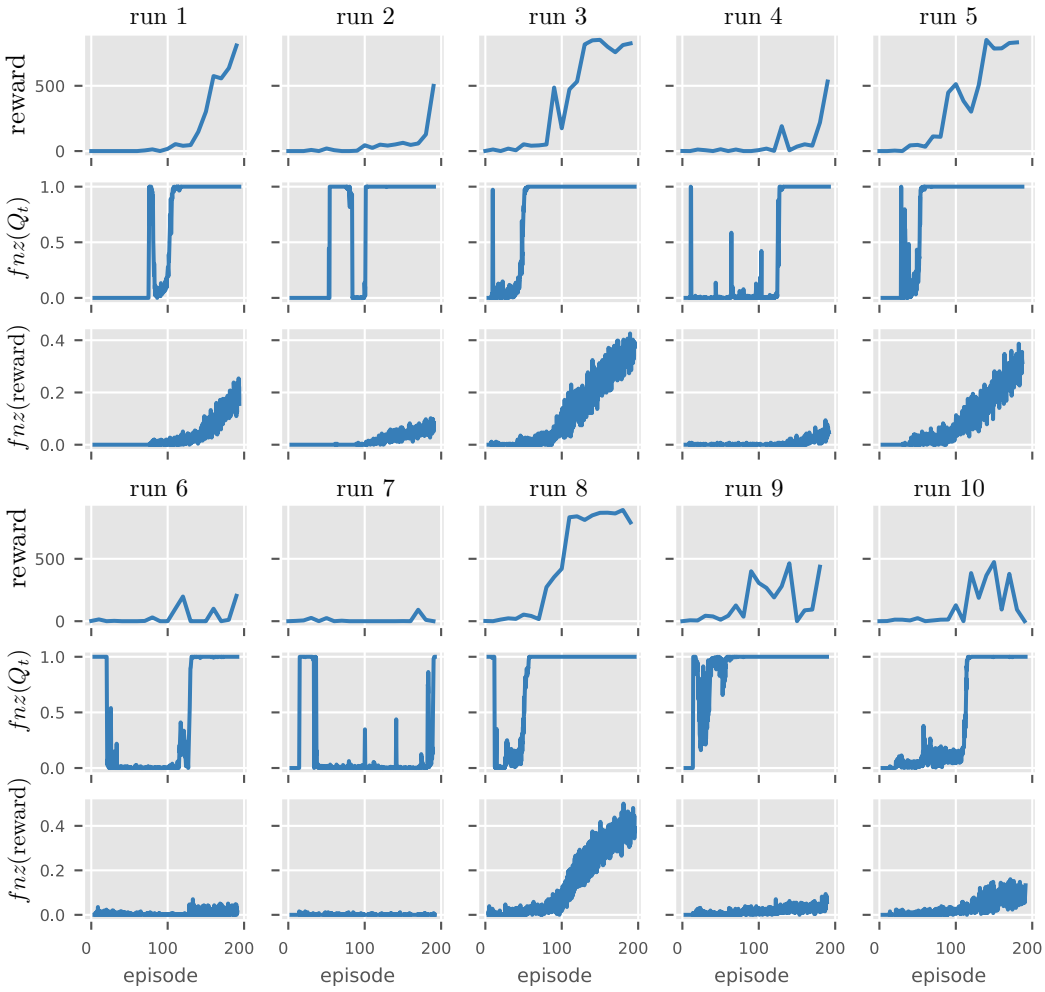

Figure 17: We illustrate three quantities for 10 runs of the pendulum swingup task using the combined agent: the reward, $fnz(Q_t)$ the fraction of non-zero Q-target values obtained over the training batches, and $fnz(\text{reward})$ the fraction of non-zero rewards obtained over the training batches. The Q-values frequently "get stuck" at outputting almost only zeros.

Table 8: Performance for various fixes for sparse tasks. We show mean reward and standard deviation of rewards, using ten seeds. Using average q-values and increasing the constrastive learning are the best fixes. Increasing the constrastive learning helps the most for cartpole swingup sparse and pendulum swingup, two tasks which were not part of the five tasks we consider first.

metric	method	cartpole sparse	finger turn hard	pendulum swingup	reacher hard
μ	baseline	641.8	596.5	648.0	872.6
μ	cpc+avg	**802.5**	788.0	**828.0**	930.2
μ	cpc+rnd	607.6	659.4	813.5	869.3
μ	avg+rnd	561.4	770.6	624.6	860.9
μ	rnd	618.2	663.4	657.2	882.9
μ	avg	761.5	**811.3**	639.0	**939.2**
μ	cpc	235.0	574.6	686.7	882.7
μ	all	742.7	729.8	758.0	922.7
σ	baseline	321.5	292.4	283.5	95.5
σ	cpc+avg	**29.4**	135.6	**24.2**	47.6
σ	cpc+rnd	286.1	173.4	29.8	78.4
σ	avg+rnd	306.2	150.4	330.0	80.3
σ	rnd	267.5	192.1	252.5	117.7
σ	avg	127.5	**89.4**	318.6	**37.3**
σ	cpc	359.7	236.7	272.5	80.1
σ	all	151.2	130.9	220.8	45.3