Towards Deeper Deep Reinforcement Learning

Johan Bjorck[*], **Carla P. Gomes**, **Kilian Q. Weinberger**
Cornell University

Abstract

In computer vision and natural language processing, innovations in model architecture that increase model capacity have reliably translated into gains in performance. In stark contrast with this trend, state-of-the-art reinforcement learning (RL) algorithms often use small MLPs, and gains in performance typically originate from algorithmic innovations. It is natural to hypothesize that small datasets in RL necessitate simple models to avoid overfitting; however, this hypothesis is untested. In this paper we investigate how RL agents are affected by exchanging the small MLPs with larger modern networks with skip connections and normalization, focusing specifically on actor-critic algorithms. We empirically verify that naïvely adopting such architectures leads to instabilities and poor performance, likely contributing to the popularity of simple models in practice. However, we show that dataset size is not the limiting factor, and instead argue that instability from taking gradients through the critic is the culprit. We demonstrate that spectral normalization (SN) can mitigate this issue and enable stable training with large modern architectures. After smoothing with SN, larger models yield significant performance improvements — suggesting that more "easy" gains may be had by focusing on model architectures in addition to algorithmic innovations.

1 Introduction

In computer vision and natural language processing (NLP), competitive models are growing increasingly large, and researchers now train billion-parameter models [13, 36]. The earliest neural networks were often shallow [40] with performance dropping for excessively deep models [29]. However, ever since the introduction of batch normalization [31] and residual connections [29], performance has improved more or less monotonically with model scale [64]. As a result, competitive models in computer vision and NLP are growing ever larger [10, 56], and further architectural innovations are continuously researched [14].

In stark contrast with this trend, state-of-the-art (SOTA) reinforcement learning (RL) agents often rely on small feedforward networks [35, 37, 39] and performance gains typically originate from algorithmic innovations such as novel loss functions [58, 62, 68] rather than increasing model capacity. Indeed, a recent large-scale study has shown that large networks can harm performance in RL [2]. It is natural to suspect that high-capacity models might overfit in the low-sample regime common in RL evaluation. Imagenet contains over a million unique images whereas RL is often evaluated in contexts with fewer environment samples [37, 39]. However, to date, this overfitting hypothesis remains largely untested.

To address this, we study the effects of using larger *modern* architectures in RL. By modern architectures we mean networks with high capacity, facilitated by normalization layers [6, 31] and skip connections [29]. We thus depart from the trend in RL of treating networks as black-box function approximators. To limit the scope and compute requirements we focus on continuous control from pixels [65] with two actor-critic algorithm: Soft Actor-Critic (SAC) [25] and Deep Deterministic

[*]Correspondence to: Johan Bjorck <njb225@cornell.edu>

35th Conference on Neural Information Processing Systems (NeurIPS 2021).

Policy Gradients (DDPG) [43]. Actor-critic methods form the basis of many SOTA algorithms for continuous control [35, 38, 39]. As can be expected, we demonstrate that naïvely adopting modern architectures leads to poor performance, supporting the idea that RL does not benefit from larger models. However, we show that the issue is not necessarily overfitting, but instead, that training becomes unstable with deeper modern networks. We hypothesize that taking the gradient of the actor through the critic network creates exploding gradients [53] for deeper networks. We connect this setup with generative adversarial networks (GANs) [22], and propose to use a simple smoothing technique from the GAN literature to stabilize training: *spectral normalization* [47].

We demonstrate that this simple strategy allows the training of larger modern networks in RL without instability. With these fixes, we can improve upon state-of-the-art RL agents on competitive continuous control benchmarks, demonstrating that improvements in network architecture can dramatically affect performance in RL. We also provide performance experiments showing that such scaling can be relatively cheap in terms of memory and compute time in RL from pixels. Our work suggests that model scaling is complementary to algorithmic innovations and that this simple strategy should not be overlooked. We summarize our contributions as follows:

- We verify empirically that large modern networks fail for SAC and DDPG. We demonstrate dramatic instabilities during training, which casts doubt on overfitting being responsible.
- We argue that taking the gradients through the critic is the cause of this instability. To combat this problem we propose to adopt spectral normalization [47] from the GAN literature.
- We demonstrate that this simple smoothing method enables the use of large modern networks and leads to significant improvements for SOTA methods on hard continuous control tasks. We further provide evidence that this strategy is computationally cheap in RL from pixels.

2 Background

2.1 Reinforcement Learning

Reinforcement learning tasks are often formulated as Markov decision processes (MDPs), which can be defined by a tuple $(\mathcal{S}, \mathcal{A}, P, r)$ [63]. For continuous control tasks the action space $\mathcal{A}$ and the state space $\mathcal{S}$ are continuous and can also be bounded. Each dimension of the action space might for example correspond to one joint of a humanoid walker. At time step t, the agent is in a state $\mathbf{s}_t \in \mathcal{S}$ and takes an action $\mathbf{a} \in \mathcal{A}$ 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 $P : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, \infty)$. The agent receives a reward r_t from the reward distribution r at each timestep t. The goal is typically to find a policy $\pi : \mathcal{S} \to \mathcal{A}$ that maximizes expected discounted reward $\mathbb{E}[\sum_t \gamma^t r_t]$, where $0 < \gamma < 1$ is a pre-determined constant. In practice, it is common to measure performance by the cumulative rewards $\sum_t r_t$.

2.2 Actor-Critic Agents

Actor-critic methods have roots in Q-learning [69] and form the backbone of many recent state-of-the-art algorithms [35, 38, 39]. We focus on two popular and performant actor critic methods: SAC [25] and DDPG [43]. Given some features $\phi(\mathbf{s})$ that can be obtained from any state $\mathbf{s} \in \mathcal{S}$, the actor-network maps each state-feature $\phi(\mathbf{s})$ to a distribution $\pi_\theta(\mathbf{s})$ over actions. The action distribution is known as the *policy*. For each dimension, the actor-network outputs an independent normal distribution, where the mean $\boldsymbol{\mu}_\theta(\mathbf{s})$ and possibly the standard deviation $\boldsymbol{\sigma}_\theta(\mathbf{s})$ come from the actor-network. The action distribution is then obtained as

$$\pi_\theta(\mathbf{s}) = \tanh(\boldsymbol{\mu}_\theta(\mathbf{s}) + \boldsymbol{\epsilon} \odot \boldsymbol{\sigma}_\theta(\mathbf{s})), \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{1}). \tag{1}$$

The $\tanh$ non-linearity is applied elementwise, which bounds the action space to $[-1, 1]^n$ for $n = dim(\mathcal{A})$. Sampling $\boldsymbol{\epsilon}$ allows us to sample actions $\mathbf{a}_\theta$ from the policy $\pi_\theta(\mathbf{s})$. The biggest difference between DDPG and SAC is the entropy bonus the latter uses. To encourage exploration, SAC adds the entropy of the policy distribution $H(\pi_\theta(\mathbf{s}_t))$ times a parameter α to the rewards. The *critic* network outputs a Q-value $Q_\psi(\mathbf{a}, \mathbf{s})$ for each state $\mathbf{s}$ and action $\mathbf{a}$. In practice, one can simply concatenate the feature $\phi(\mathbf{s})$ with the action $\mathbf{a}$ and feed the result to the critic. The critic network is trained to minimize the soft Bellman residual:

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

Here, r_t is the obtained 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 are the exponentially averaged weights of the critic. One can also use multi-step targets for the rewards [62]. The loss (2) is computed by sampling transitions from a replay buffer [48]. Since the Q-values that the critic outputs measure expected discounted rewards, the actor should simply take actions that maximize the output of the critic network. Thus, to obtain a gradient step for the actor, both SAC and DDPG sample an action $\mathbf{a}_\theta$ from $\pi_\theta(\mathbf{s}_t)$ and then take derivatives of $Q_\psi(\mathbf{a}_\theta, \mathbf{s})$ with respect to θ. That is

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

As the critic is a differentiable neural network, the derivatives are taken *through* the Q-values $Q_\psi(\mathbf{a}_\theta, \mathbf{s})$. In practice, the features $\phi(\mathbf{s})$ can be obtained from a convolutional network trained by backpropagating from (2). For further details, see Haarnoja et al. [25], Lillicrap et al. [43].

3 Motivating Experiments

3.1 Experimental Setup

For experimental evaluation, we focus on continuous control from pixels, a popular setting relevant to real-world applications [33, 34]. Specifically we evaluate on the DeepMind control suite [65], which has been used in [26, 27, 35, 39]. We use the 15 tasks considered in Kostrikov et al. [35] and evaluate after 500,000 samples, which has become a common benchmark [35, 39, 41]. If learning crashes before 500,000 steps (this occasionally happens for modern networks), we report the performance before the crash. We will primarily focus on the image augmentation based SAC implementation of Kostrikov et al. [35], known as DRQ, and adopt their hyperparameters (listed in Appendix B). The network consists of a common convolutional encoder that processes the image into a fixed-length feature vector. Such feature vectors are then processed by the critic and actor-network, which are just feedforward networks. We will refer to these feedforward networks as the *heads*. The convolutional encoder consists of four convolutional layers followed by a linear layer and layer normalization [6] and the heads consist of MLPs with two hidden layers that concatenate the actions and image features; see Appendix C for details. We will also consider the augmentation-based DDPG agent of Yarats et al. [71], using the default hyperparameters proposed therein. This implementation uses the same network design as SAC. In all experiments, we only vary the head architecture. We evaluation over ten seeds.

3.2 Testing Modern networks

Two crucial architectural innovations which have enabled deep supervised models are residual connections [29] and normalization [31]. We test if adopting such modern techniques allows us to successfully scale networks in SAC. Specifically, we use the feedforward network (and not the attention module) found in a Transformer block [67] and will refer to this architecture as *modern*. Each

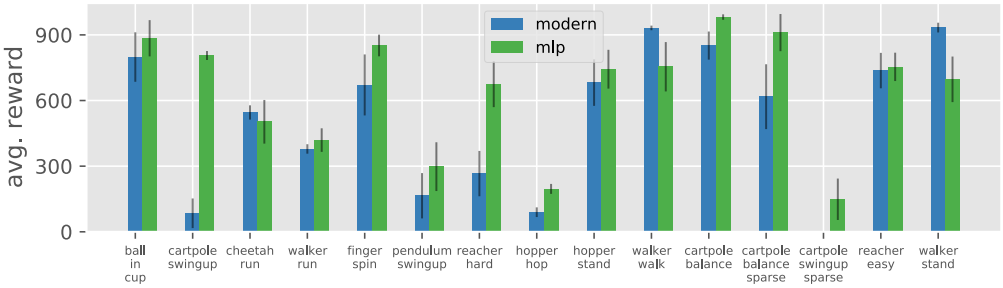

Figure 1: Performance of SAC when using a modern network architecture with skip connections [29] and normalization [6], averaged across ten seeds. Using modern architectures does not enable deeper networks for SAC; instead, performance often decreases — sometimes catastrophically — while increasing for a handful of tasks. This is in stark contrast with supervised learning where performance typically improves monotonically with model capacity.

block consists of two linear transformations w_1, w_2 with a ReLu between and layer normalization [6] that is added to a residual branch. I.e. the output of a layer is $x + w_2\text{Relu}\big(w_1\text{norm}(x)\big)$. Compared to the original architecture, normalization is applied before the feedforward blocks instead of after, which is now favored in practice [50, 70]. We use four transformer blocks for the critic and two for the actor. Results from these experiments are shown in Figure 1, with one standard error given. We see that these modifications often decrease performance. Thus, SAC has seemingly not reached the level where model capacity improves performance monotonically, and tricks used in supervised learning do not seem to suffice. This is perhaps not very surprising as state-of-the-art RL agents typically use simple feedforward networks.

3.3 Testing Deeper Networks

Whereas naively adopting modern networks failed, we now investigate if simply making the MLP networks deeper is a competitive strategy. To do this, we increase the number of MLP layers in the actor and critic — from two to four hidden layers — while keeping the width constant. Results for the two configurations are shown in Figure 2, with one standard error given. We see that for some tasks, the performance improves, but for many others, it decreases. Simply scaling the network does not monotonically improve performance, as typically is the case in supervised learning. For practitioners, the extra computational burden might not make it worthwhile to increase network scale for such dubious gains.

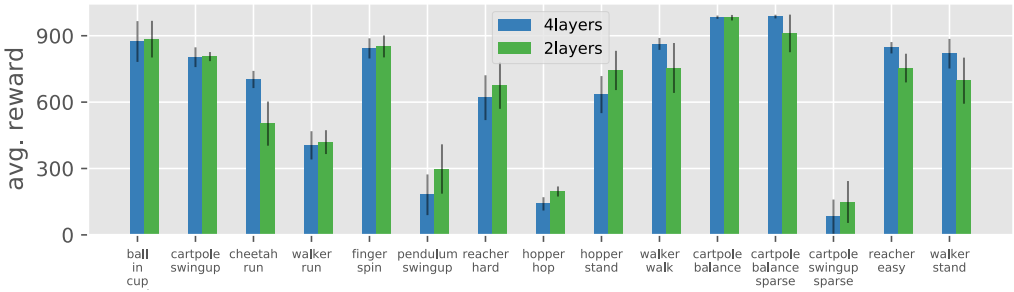

Figure 2: The performance of SAC when making the networks deeper. Performance is averaged across ten seeds and plotted for 15 tasks from the DeepMind control suite [65]. Making the network deeper does not monotonically improve performance as is typically the case in supervised learning. Instead, some tasks benefit, and some tasks suffer. As we shall see later, it is possible to improve the performance by stabilizing training, suggesting that deeper networks cause some instability.

3.4 Is it overfitting?

Both making the networks deeper and adopting modern methods such as normalization and residual connections do not monotonically improve performance, and can occasionally lead to catastrophic drops in performance. It is natural to think that the small number of environment transitions available leads to overfitting for large models, which is known to be harmful in RL [76]. Indeed, since training and collecting samples are interleaved in RL, RL agents need to generate useful behavior when only 10 % of the training has elapsed, at which point only 10 % of the sample budget is available. Such an overfitting hypothesis is relatively easy to probe. As per eq. (2), the critic is trained to minimize the soft Bellman residual. If overfitting truly was the issue, we would expect the loss to decrease when increasing the network capacity. We investigate this by comparing a modern network against a smaller MLP in an environment where the former performs poorly — cartpole-swing-up. First, we study the critic loss as measured over the replay buffer — which plays the role of training loss in supervised learning. Results averaged over five runs are shown in Figure 3. The losses initially start on the same scale, but the modern network later has losses that increase dramatically — suggesting that overfitting is not the issue. Instead, training stability could be the culprit, as stable training should enable the network to reach lower training loss. To probe this, we simply plot the norm of the gradients for the critic and actor networks and see that they indeed are larger for the large modern networks. Furthermore, the gradient magnitude increases during training and shows large spikes, especially for the actor. This suggests that training stability, rather than overfitting, is the issue.

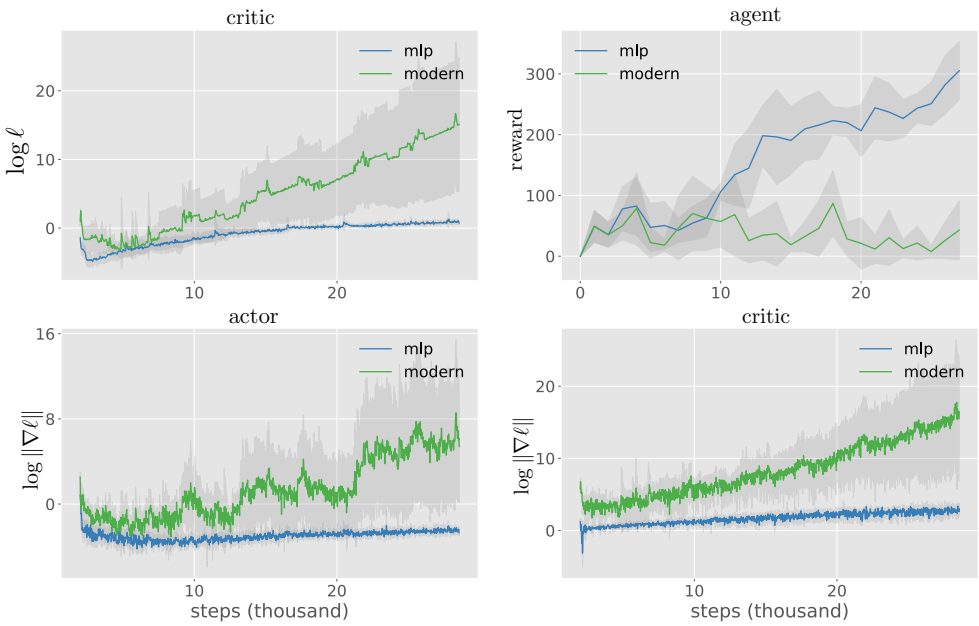

Figure 3: Training dynamics when using a small MLP or a deeper network with normalization and residual connections. **Top left.** The loss, as per eq. (2), of the critic during training. The loss does not decrease when making the network larger, suggesting that overfitting is not the issue. **Top right.** The rewards obtained. The deeper network fails to learn and improve its performance. **Bottom left & right.** The gradient of the actor and critic during training. For the deeper network, training becomes unstable with occasional spikes and large and growing gradients. Notice the logarithmic scale.

4 Stabilizing with Spectral Normalization

To improve the stability of actor-critic algorithms, let us consider a simple model. We will assume that the action space has one degree of freedom, i.e. $\mathcal{A} = \mathbb{R}$. We also assume that we have some fixed features ϕ which maps states to n-dimensional vectors, i.e. $\phi(\mathcal{S}) = \mathbb{R}^n$. The critic and actor are both modeled as MLPs with N layers and a nonlinearity f (e.g. ReLu) applied elementwise after each layer. The weights are given as matrices w_i^a or w_i^c for layer i in the actor and critic respectively. We then have $y_i = f(w_i x_i)$ for layer i. If we let the functions representing the actor and critic be denoted by A and Q, and let $\|$ denote vector concatenation, we have:

$$A(\mathbf{s}) = \Big(\prod_i f \circ w_i^a\Big) \circ \phi(\mathbf{s}) \qquad Q(\mathbf{s}) = Q(A(\mathbf{s}), \phi(\mathbf{s})) = \Big(\prod_i f \circ w_i^c\Big) \circ \big(A(\mathbf{s}) \| \phi(\mathbf{s})\big) \tag{4}$$

Recall that the Lipschitz constant of a function f is the smallest constant C such that $\|f(x) - f(y)\| \leq C\|x - y\|$ for all x, y. It is straightforward to bound the Lipshitz constant of the critic, as it is made up of transformations with bounded constants themselves. The Lipschitz constant of the linear layer w_i is the operator norm $\|w_i\|$, equal to the largest singular value $\sigma_{\max}$ of w_i. We have:

$$\|Q(\mathbf{s})\| \leq \big(\|A(\mathbf{s})\| + \|\phi(\mathbf{s})\|\big) \prod_i \|f\| \|w_i^c\| \tag{5}$$

Here $\|f\|$ is the Lipschitz constant of the function f, which for Relu is 1. Equation (5) bounds the smoothness of the critic in the forward pass. A function that is L-Lipschitz smooth also has a gradient bounded by L. Thus, if we could ensure that the critic is L-smooth, that would also imply that gradients $\frac{\partial Q_\psi}{\partial \mathbf{a}_\theta}$ being propagated into the actor, as per (3), are bounded. Equation (5) suggest that the critic could be made smooth if the spectral norms of all layers are bounded. Fortunately, there is a method from the GAN literature which achieves this: spectral normalization [47]. Spectral normalization divides the weight W for each layer by its largest singular value $\sigma_{\max}$ which ensures that all layers have operator norm 1. The singular value can be expensive to compute in general, and to this end, spectral normalization uses two vectors u and v – approximately equal to the right and left

vectors of the largest singular value, and approximate $\sigma_{\max} \approx u^T W v$. The forward pass becomes:

$$y = \frac{Wx}{\sigma_{\max}(W)} \approx \frac{Wx}{u^T W v}$$

For this to work properly, the vectors u and v should be close to the vectors corresponding to the largest singular value. This is achieved via the power method [46, 47] by taking:

$$u \leftarrow W^T u / \|W^T u\| \qquad v \leftarrow W v / \|W v\|$$

By repeating this procedure for all layers, we ensure that the spectral norms of all layers are no larger than one. If that is the case, eq. (5) suggests that the critic should be stable in the forward pass. This would then bound the gradients being propagated into the actor as per eq. (3). Figure 3 has shown that exploding gradients are associated with poor performance, and we thus hypothesize that applying spectral normalization should stabilize learning and enable deeper networks.

5 Experiments

5.1 Smoothness Enables Larger Networks

We first investigate whether smoothing with spectral normalization allows us to use deeper modern networks in SAC. To do this, we simply compare SAC using the modern network defined in Section 3.2 without and with spectral normalization. Specifically, for both the actor and the critic, we apply spectral normalization to each linear layer except the first and last. Otherwise, the setup follows Section 3.1. As before, when learning crashes, we simply use the performance recorded before crashes for future time steps. Learning curves for individual environments are given in Figure 4, again over 10 seeds. We see that after smoothing with spectral normalization, SAC is relatively stable across tasks, even when using a deep network with normalization and skip connections. On the other hand, without smoothing, learning is slow and sometimes fails. Note that the improvements differ by tasks, but performance essentially improves monotonically when making the network deeper — just as in supervised learning. The only exception is walker walk, where the smoothed strategy is narrowly beat, however, this might be a statistical outlier. In Appendix A, we also show that smoothing with spectral normalization improves performance when using the 4 hidden layer MLPs. Thus, we conclude that enforcing smoothness allows SAC to utilize deeper networks.

5.2 Comparing to Other Agents

After demonstrating that smoothing with spectral normalization allows SAC to use larger modern networks and improve its performance, we now see if these gains allow us to improve upon other agents. We compare against two state-of-the-art methods for continuous control: Dreamer [26] and the SAC with augmentations agent of Kostrikov et al. [35] (DRQ) from which we start from. To ensure fair and identical evaluation across different algorithms, we run DRQ and Dreamer with the author-provided implementations across 10 seeds. We show scores at step 500,000, averaged across 10 seeds in Table 2. For most tasks, the agent using deep modern networks with smoothing outperforms the other agents, although for a handful of tasks it is narrowly beaten. Note that the improvements that are obtained from scaling the network differ significantly between tasks, likely as some tasks are more complex and thus more amenable to high-capacity models. It is also interesting to note that performance on some sparse tasks improves, where artificial curiosity is often employed [54, 61], suggesting that such tasks might not always require specialized solutions. We note that these gains are comparable to those of algorithmic innovations, e.g. the improvement for changing architecture is larger than the difference between Dreamer and DRQ showed here, see Appendix D for details. Appendix D also compare against learning curves reported in [26, 35] which does not necessarily use more than 5 seeds. Here, deeper modern networks with smoothing again outperform the two alternatives. We conclude that by enforcing smoothness and simply scaling the network, it is possible to improve upon state-of-the-art methods without any algorithmic innovations.

5.3 Experiments with DDPG

Our motivation for using spectral normalization is not specific for SAC [25] and extends to actor-critic methods in general. To ensure that our methods work well beyond SAC, we now consider the DDPG

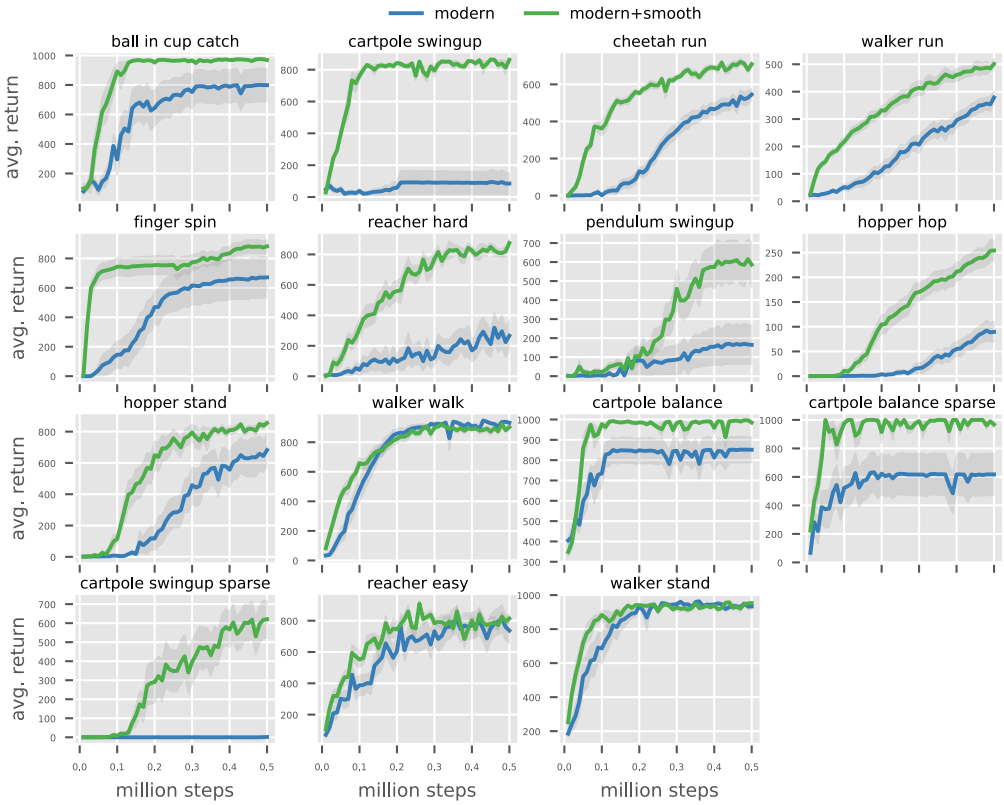

Figure 4: Learning curves for SAC using deep modern networks, with and without smoothing by spectral normalization. Smoothing enables the deep network to learn effectively across tasks, whereas naively using such a deep network often leads to slow learning and sometimes no learning at all.

Table 1: Results for the DDPG agent of Yarats et al. [71] using different networks. Using a larger modern network results in poor performance compared to a traditional MLP, but once SN is used larger networks are beneficial. Metrics are given after 0.5 million frames and averaged over 10 seeds.

task	**MLP**	**modern**	**modern + SN**
cup catch	966.67 ± 2.32	107.46 ± 21.43	$\mathbf{975.32} \pm 1.3$
walker walk	477.40 ± 117.96	6.10 ± 9.08	$\mathbf{739.94} \pm 20.88$
cartpole sparse	$\mathbf{1000.00} \pm 0.00$	11.27 ± 0.16	$\mathbf{1000.00} \pm 0.00$
hopper stand	403.82 ± 123.85	2.95 ± 0.99	$\mathbf{796.60} \pm 85.04$
reacher easy	756.15 ± 70.48	77.22 ± 19.95	$\mathbf{801.83} \pm 66.31$

algorithm [43]. We evaluate how larger networks and spectral normalization performs for DDPG using the image-augmentation-based implementation of Yarats et al. [71], dubbed DRQv2, and adopt hyperparameters therein. We use the same modern and base networks as for the SAC agent. For computational reasons we only focus on a smaller set of tasks, these include tasks with both dense and sparse rewards, and both locomotion and classical control tasks. Results over 10 seeds and after 500,000 frames are reported in Table 1. We see a similar result for DDPG as for SAC – using a larger network is detrimental to performance, but if spectral normalization is applied it can often be beneficial. These results suggest that our intuition for using spectral normalization benefits beyond the SAC setup. Results after 1 million frames are similar and are given in Appendix A. We conclude that spectral normalization can enable larger networks in DDPG too and that this agent also benefits from high-capacity networks.

Table 2: Comparison of algorithms across tasks from the DeepMind control suite. We compare SAC with a modern architecture and spectral normalization [47] against two state-of-the-art agents: data augmented SAC with two hidden layers [35] (DRQ), and Dreamer [26]. By using spectral normalization the modern network outperforms the other methods. This demonstrates that simply stabilizing larger network can achieve gains comparable to those of algorithmic innovations.

	ball in cup catch	swingup	cheetah run	walker run
DRQ	884.6 $\pm$ 262.0	805.7 $\pm$ 64.9	502.8 $\pm$ 315.5	419.2 $\pm$ 170.5
Modern+smooth	**968.9** $\pm$ 13.3	**862.3** $\pm$ 15.2	**708.9** $\pm$ 46.2	**501.0** $\pm$ 57.0
Dreamer	767.1 $\pm$ 63.3	592.4 $\pm$ 31.3	630.7 $\pm$ 23.3	466.2 $\pm$ 21.1
	finger spin	reacher hard	pendulum swingup	hopper hop
DRQ	851.7 $\pm$ 156.8	674.7 $\pm$ 332.0	297.8 $\pm$ 352.6	196.1 $\pm$ 72.0
Modern+smooth	**882.5** $\pm$ 149.0	**875.1** $\pm$ 75.5	**586.4** $\pm$ 381.3	**254.2** $\pm$ 67.7
Dreamer	466.8 $\pm$ 42.1	65.7 $\pm$ 16.5	494.3 $\pm$ 98.9	136.0 $\pm$ 28.5
	hopper stand	walker walk	balance	balance sparse
DRQ	743.1 $\pm$ 280.5	754.3 $\pm$ 356.7	981.5 $\pm$ 39.6	910.5 $\pm$ 268.6
Modern+smooth	**854.7** $\pm$ 63.5	**902.2** $\pm$ 67.4	**984.8** $\pm$ 29.4	**968.1** $\pm$ 72.3
Dreamer	663.7 $\pm$ 70.3	863. $\pm$ 27.2	938.2 $\pm$ 13.0	935.2 $\pm$ 22.5
	swingup sparse	reacher easy	walker stand	
DRQ	148.7 $\pm$ 299.3	753.9 $\pm$ 205.0	696.9 $\pm$ 329.4	
Modern+smooth	**620.8** $\pm$ 311.0	814.0 $\pm$ 85.8	953.0 $\pm$ 19.8	
Dreamer	267.8 $\pm$ 25.4	**830.4** $\pm$ 31.4	**955.3** $\pm$ 8.7	

5.4 Performance Cost

Scaling networks can improve performance, but larger networks can also increase memory footprint and compute time. Thus, we here measure the memory consumption and compute time for SAC across four architectures; MLPs and modern networks (specified in Section 3.2) with two or four hidden layers. Recall that we only modify the head networks, which share features processed by a common convolutional network that is not modified. We consider training with batch size 512 and interacting with the environment, the latter setup models deployment after training. We use Tesla V100 GPUs and measure time with CUDA events and measure memory with PyTorch native tools. For a single run, we average time across 500 iterations with warm-start. The results averaged across 5 runs are given in Table 3. The memory consumption during training changes relatively little, increasing roughly 17%. The reason for this is of course that memory footprint comes both from the convolutional layers, which we do not modify, and the head networks, which we increase in size. As the convolutional activations are spread across spatial dimensions, these dominate the memory consumption, and increasing the scale of the head networks becomes relatively cheap. The memory consumption during acting, which is dominated by storing the model weights, changes dramatically in relative terms but is less than 200 MB in absolute terms. For compute time, we see similar trends. Increasing the scale of the head network is relatively cheap as processing the images dominates computation time. During acting, however, the increase in time is smaller, likely as the time is dominated by Python computation overhead on the CPU rather than GPU computation. This demonstrates that simply scaling the head networks can be relatively cheap for RL from pixels while significantly improving performance.

6 Related Work

Early work on deep RL primarily used simple feedforward networks, possibly processing images with a convolutional network [48, 49]. This is still a common strategy [35, 37, 39], and the setting we have focused on. For environments with sequential information, it can be fruitful to incorporate memory into the network architecture via RNNs [32] or Transformers [51, 52]. There is also work on how to utilize, but not necessarily scale, neural networks in RL, e.g. dueling heads [68] or double Q-learning [66]. The proposition that too-large networks perform poorly has been verified

Table 3: We compare memory and compute requirements for using SAC on Tesla V100 GPUs with four different architectures for the heads. For training, increasing the capacity of the head incurs a relatively small cost in both memory and compute since convolutional processing is the bottleneck. Thus, the improvements of Table 2 are relatively cheap. During acting, memory cost increases a lot in relative terms, but less in absolute terms. Compute cost during acting changes little, likely as it is dominated by CPU rather than GPU computations.

	mlp2	mlp4	modern 2	modern4
train mem (GB)	4.05 ± 0.0	4.19 ± 0.0	4.27 ± 0.0	4.73 ± 0.0
act mem (MB)	50.16 ± 0.0	93.26 ± 0.0	112.66 ± 0.0	247.04 ± 0.0
train time (ms)	98.14 ± 0.15	104.1 ± 0.14	105.76 ± 0.23	125.21 ± 0.1
act time (ms)	1.19 ± 0.09	1.26 ± 0.02	1.4 ± 0.1	1.35 ± 0.02

for policy gradient methods in the systematic study of Andrychowicz et al. [2]. There is also ample systematic work on generalization in RL [11, 18, 74] and many other relevant large-scale studies include [15, 30, 76]. Spectral normalization has been used to regularize model-based reinforcement learning by Yu et al. [72], however, the effects of SN seem to not be studied through ablations or controlled experiments here. Concurrently with our work, Gogianu et al. [21] has investigated the use of spectral normalization for DQN. In contrast with Gogianu et al. [21], we show how SN can be used to enable larger networks, and use it in RL algorithms where the actor takes gradients through the critic. This setting is fundamentally different from the discrete DQN setting, and one much closer to the GAN setup where SN has been developed. Sinha et al. [60] consider using dense connections to enable larger networks in RL. In contrast, we consider even larger models and also provide a solution that is agnostic to the underlying network architecture.

There are many proposed ideas for enforcing smoothness in neural networks, especially for GANs [5, 9, 22, 24, 57]. The idea of directly constraining the singular value has been investigated by multiple authors [19, 23, 47]. For GANs, one updates the weight of a generator network by taking the gradient *through* the discriminator network, similar to how the actor is updated in SAC by taking gradients *through* the output of the critic network. Due to this similarity, perhaps more techniques from the GAN literature apply to actor-critic methods in RL.

7 Discussion

The Importance of Architecture. Within RL, it is common to abstract the neural network as function approximators and instead focus on algorithmic questions such as loss functions. As we have shown, scaling up the network can have a dramatic effect on performance, which can be larger than the effects of algorithmic innovations. Our results highlight how low-level network architecture decisions should not be overlooked. There is likely further progress to be made by optimizing the architectures. Within supervised learning, there is ample systematic work on the effects of architectures [50, 64], and similar studies could be fruitful in RL. Whereas larger architecture requires more compute resources, this can potentially be offset by asynchronous training or other methods designed to accelerate RL [8, 16, 17, 45, 55].

Evaluation in RL. Supervised learning has already reached the state where performance seems to monotonically improve with network capacity [64]. As a consequence, it is common the compare methods with similar compute resources. We have demonstrated how architectural modifications can enable much larger networks to be used successfully in RL. If such benefits can be extended across agents, fair comparisons in RL might require listing the amount of compute resources used for novel methods. This is especially important for compute-intensive unsupervised methods [1, 39, 44, 59, 73, 75] or model-based learning [4, 12, 20, 28, 42].

Limitations. For compute reasons we have only studied the effects of networks on SAC and DDPG — these are state-of-the-art algorithms for continuous control underlying many popular algorithms – nonetheless, there are plenty of other algorithms. While our study gives some hints, an outstanding question is why large architectures are not popular in RL in general, beyond the algorithms we consider. The idea of obtaining gradients through subnetworks is common and algorithms such as Dreamer [26] might also benefit from smoothing. Spectral normalization [47] is a relatively

straightforward smoothing strategy and many alternatives which might perform better are known. We emphasize that the goal of our paper is to demonstrate that enabling large networks is an important and feasible problem, to not provide a solution for every conceivable RL algorithm. Finally, there are also further environments to try out. We have focused on continuous control from pixels as it is a setting relevant for real-world applications, but other common benchmarks such as Atari games [7] and board games [3] are also important.

Conclusion. We have investigated the effects of using modern networks with normalization and residual connections on SAC [25] and DDPG [43]. Naively implementing such changes does not necessarily improve performance, and can lead to unstable training. To resolve this issue, we have proposed to enforce smoothing via spectral normalization [47]. We show that this fix enables stable training of modern networks, which can outperform state-of-the-art methods on a large set of continuous control tasks. This demonstrates that changing network architecture can be competitive with algorithmic innovations in RL.

Acknowledgement and Funding Transparency

This research is supported in part by the grants from the National Science Foundation (III-1618134, III-1526012, IIS1149882, IIS-1724282, and TRIPODS- 1740822), the Office of Naval Research DOD (N00014- 17-1-2175), Bill and Melinda Gates Foundation. We are thankful for generous support by Zillow and SAP America Inc. This material is based upon work supported by the National Science Foundation under Grant Number CCF-1522054. We are also grateful for support through grant AFOSR-MURI (FA9550-18-1-0136). We also acknowledges support from the TTS foundation. Any opinions, findings, conclusions, or recommendations expressed here are those of the authors and do not necessarily reflect the views of the sponsors. We thank Rich Bernstein, Joshua Fan, Ziwei Liu, and the anonymous reviewers for their help with the manuscript.

References

[1] R. Agarwal, M. C. Machado, P. S. Castro, and M. G. Bellemare. Contrastive behavioral similarity embeddings for generalization in reinforcement learning. *arXiv preprint arXiv:2101.05265*, 2021.

[2] M. Andrychowicz, A. Raichuk, P. Stańczyk, M. Orsini, S. Girgin, R. Marinier, L. Hussenot, M. Geist, O. Pietquin, M. Michalski, et al. What matters in on-policy reinforcement learning? a large-scale empirical study. *arXiv preprint arXiv:2006.05990*, 2020.

[3] T. Anthony, T. Eccles, A. Tacchetti, J. Kramár, I. Gemp, T. C. Hudson, N. Porcel, M. Lanctot, J. Pérolat, R. Everett, et al. Learning to play no-press diplomacy with best response policy iteration. *arXiv preprint arXiv:2006.04635*, 2020.

[4] A. Argenson and G. Dulac-Arnold. Model-based offline planning. *arXiv preprint arXiv:2008.05556*, 2020.

[5] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR, 2017.

[6] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[7] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, jun 2013.

[8] J. Bjorck, X. Chen, C. De Sa, C. P. Gomes, and K. Q. Weinberger. Low-precision reinforcement learning. *arXiv preprint arXiv:2102.13565*, 2021.

[9] A. Brock, T. Lim, J. M. Ritchie, and N. Weston. Neural photo editing with introspective adversarial networks. *arXiv preprint arXiv:1609.07093*, 2016.

[10] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[11] K. Cobbe, O. Klimov, C. Hesse, T. Kim, and J. Schulman. Quantifying generalization in reinforcement learning. In *International Conference on Machine Learning*, pages 1282–1289. PMLR, 2019.

[12] B. Cui, Y. Chow, and M. Ghavamzadeh. Control-aware representations for model-based reinforcement learning. *arXiv preprint arXiv:2006.13408*, 2020.

[13] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[14] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[15] L. Engstrom, A. Ilyas, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph, and A. Madry. Implementation matters in deep policy gradients: A case study on ppo and trpo. *arXiv preprint arXiv:2005.12729*, 2020.

[16] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International Conference on Machine Learning*, pages 1407–1416. PMLR, 2018.

[17] L. Espeholt, R. Marinier, P. Stanczyk, K. Wang, and M. Michalski. Seed rl: Scalable and efficient deep-rl with accelerated central inference. *arXiv preprint arXiv:1910.06591*, 2019.

[18] J. Farebrother, M. C. Machado, and M. Bowling. Generalization and regularization in dqn. *arXiv preprint arXiv:1810.00123*, 2018.

[19] F. Farnia, J. M. Zhang, and D. Tse. Generalizable adversarial training via spectral normalization. *arXiv preprint arXiv:1811.07457*, 2018.

[20] J. Fu and S. Levine. Offline model-based optimization via normalized maximum likelihood estimation. *arXiv preprint arXiv:2102.07970*, 2021.

[21] F. Gogianu, T. Berariu, M. Rosca, C. Clopath, L. Busoniu, and R. Pascanu. Spectral normalisation for deep reinforcement learning: an optimisation perspective. *arXiv preprint arXiv:2105.05246*, 2021.

[22] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks. *arXiv preprint arXiv:1406.2661*, 2014.

[23] H. Gouk, E. Frank, B. Pfahringer, and M. J. Cree. Regularisation of neural networks by enforcing lipschitz continuity. *Machine Learning*, 110(2):393–416, 2021.

[24] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville. Improved training of wasserstein gans. *arXiv preprint arXiv:1704.00028*, 2017.

[25] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

[26] D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019.

[27] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, pages 2555–2565. PMLR, 2019.

[28] J. B. Hamrick, A. L. Friesen, F. Behbahani, A. Guez, F. Viola, S. Witherspoon, T. Anthony, L. Buesing, P. Veličković, and T. Weber. On the role of planning in model-based deep reinforcement learning. *arXiv preprint arXiv:2011.04021*, 2020.

[29] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[30] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger. Deep reinforcement learning that matters. *arXiv preprint arXiv:1709.06560*, 2017.

[31] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[32] S. Kapturowski, G. Ostrovski, J. Quan, R. Munos, and W. Dabney. Recurrent experience replay in distributed reinforcement learning. In *International conference on learning representations*, 2018.

[33] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. Al Sallab, S. Yogamani, and P. Pérez. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[34] J. Kober, J. A. Bagnell, and J. Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[35] I. Kostrikov, D. Yarats, and R. Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. 2020.

[36] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

[37] M. Laskin, K. Lee, A. Stooke, L. Pinto, P. Abbeel, and A. Srinivas. Reinforcement learning with augmented data. arXiv:2004.14990.

[38] M. Laskin, K. Lee, A. Stooke, L. Pinto, P. Abbeel, and A. Srinivas. Reinforcement learning with augmented data. *arXiv preprint arXiv:2004.14990*, 2020.

[39] M. Laskin, A. Srinivas, and P. Abbeel. Curl: Contrastive unsupervised representations for reinforcement learning. In *Proceedings of the 37th Annual International Conference on Machine Learning (ICML)*, 2020.

[40] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4): 541–551, 1989.

[41] A. X. Lee, A. Nagabandi, P. Abbeel, and S. Levine. Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model. *arXiv preprint arXiv:1907.00953*, 2019.

[42] B.-J. Lee, J. Lee, and K.-E. Kim. Representation balancing offline model-based reinforcement learning. In *International Conference on Learning Representations*, 2021.

[43] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[44] G. Liu, C. Zhang, L. Zhao, T. Qin, J. Zhu, J. Li, N. Yu, and T.-Y. Liu. Return-based contrastive representation learning for reinforcement learning. *arXiv preprint arXiv:2102.10960*, 2021.

[45] I.-J. Liu, R. Yeh, and A. Schwing. High-throughput synchronous deep rl. *Advances in Neural Information Processing Systems*, 33, 2020.

[46] R. Mises and H. Pollaczek-Geiringer. Praktische verfahren der gleichungsauflösung. *ZAMM-Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik*, 9(1):58–77, 1929.

[47] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*, 2018.

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

[49] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[50] S. Narang, H. W. Chung, Y. Tay, W. Fedus, T. Fevry, M. Matena, K. Malkan, N. Fiedel, N. Shazeer, Z. Lan, et al. Do transformer modifications transfer across implementations and applications? *arXiv preprint arXiv:2102.11972*, 2021.

[51] E. Parisotto and R. Salakhutdinov. Efficient transformers in reinforcement learning using actor-learner distillation. *arXiv preprint arXiv:2104.01655*, 2021.

[52] E. Parisotto, F. Song, J. Rae, R. Pascanu, C. Gulcehre, S. Jayakumar, M. Jaderberg, R. L. Kaufman, A. Clark, S. Noury, et al. Stabilizing transformers for reinforcement learning. In *International Conference on Machine Learning*, pages 7487–7498. PMLR, 2020.

[53] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318. PMLR, 2013.

[54] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell. Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning*, pages 2778–2787. PMLR, 2017.

[55] A. Petrenko, Z. Huang, T. Kumar, G. Sukhatme, and V. Koltun. Sample factory: Egocentric 3d control from pixels at 100000 fps with asynchronous reinforcement learning. In *International Conference on Machine Learning*, pages 7652–7662. PMLR, 2020.

[56] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789, 2019.

[57] T. Salimans and D. P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *arXiv preprint arXiv:1602.07868*, 2016.

[58] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[59] M. Schwarzer, A. Anand, R. Goel, R. D. Hjelm, A. Courville, and P. Bachman. Data-efficient reinforcement learning with momentum predictive representations. *arXiv preprint arXiv:2007.05929*, 2020.

[60] S. Sinha, H. Bharadhwaj, A. Srinivas, and A. Garg. D2rl: Deep dense architectures in reinforcement learning. *arXiv preprint arXiv:2010.09163*, 2020.

[61] S. A. Sontakke, A. Mehrjou, L. Itti, and B. Schölkopf. Causal curiosity: Rl agents discovering self-supervised experiments for causal representation learning. *arXiv preprint arXiv:2010.03110*, 2020.

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

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

[64] M. Tan and Q. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019.

[65] Y. Tassa, S. Tunyasuvunakool, A. Muldal, Y. Doron, S. Liu, S. Bohez, J. Merel, T. Erez, T. Lillicrap, and N. Heess. dm control: Software and tasks for continuous control, 2020.

[66] H. van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. corr abs/1509.06461 (2015). *arXiv preprint arXiv:1509.06461*, 2015.

[67] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

[68] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR, 2016.

[69] C. J. C. H. Watkins. Learning from delayed rewards. 1989.

[70] R. Xiong, Y. Yang, D. He, K. Zheng, S. Zheng, C. Xing, H. Zhang, Y. Lan, L. Wang, and T. Liu. On layer normalization in the transformer architecture. In *International Conference on Machine Learning*, pages 10524–10533. PMLR, 2020.

[71] D. Yarats, R. Fergus, A. Lazaric, and L. Pinto. Mastering visual continuous control: Improved data-augmented reinforcement learning. *arXiv preprint arXiv:2107.09645*, 2021.

[72] T. Yu, G. Thomas, L. Yu, S. Ermon, J. Zou, S. Levine, C. Finn, and T. Ma. Mopo: Model-based offline policy optimization. *arXiv preprint arXiv:2005.13239*, 2020.

[73] A. Zadaianchuk, M. Seitzer, and G. Martius. Self-supervised visual reinforcement learning with object-centric representations. *arXiv preprint arXiv:2011.14381*, 2020.

[74] A. Zhang, N. Ballas, and J. Pineau. A dissection of overfitting and generalization in continuous reinforcement learning. *arXiv preprint arXiv:1806.07937*, 2018.

[75] A. Zhang, R. McAllister, R. Calandra, Y. Gal, and S. Levine. Learning invariant representations for reinforcement learning without reconstruction. *arXiv preprint arXiv:2006.10742*, 2020.

[76] C. Zhang, O. Vinyals, R. Munos, and S. Bengio. A study on overfitting in deep reinforcement learning. *arXiv preprint arXiv:1804.06893*, 2018.

A Appendix

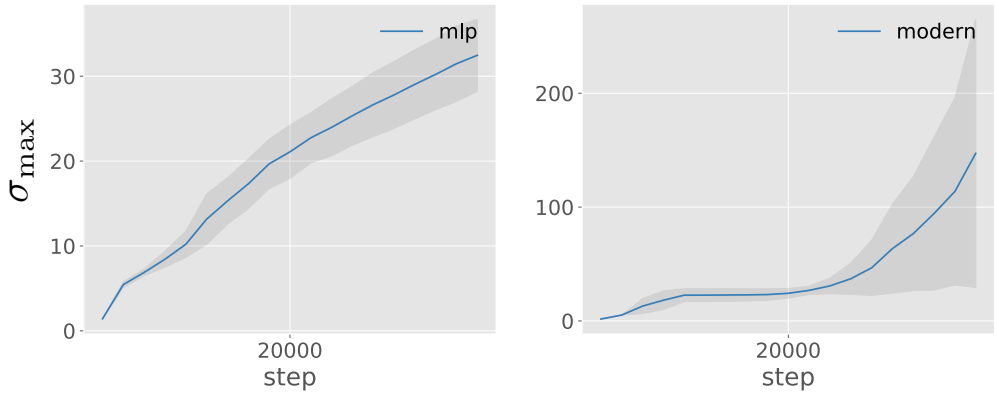

Figure 5: The largest singular value of the weight matrix at layer 2 when using the MLP or modern architecture (without smoothing). Results are shown for the cartpole swingup environment and averaged across 5 runs. For the modern network with skip connections and normalization, the largest singular value grows dramatically.

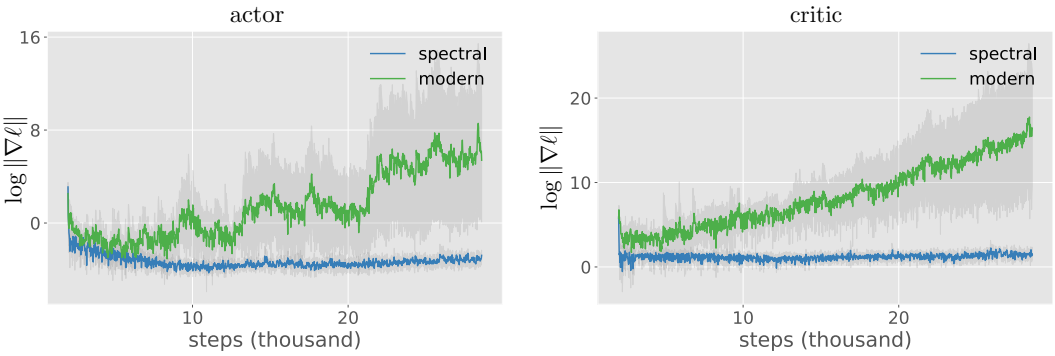

Figure 6: We here compare the gradients when using a modern network with and without smoothing with spectral normalization. When using spectral normalization, the gradients are stable during training even for the larger modern networks. Compare with Figure 3.

B Hyperparameters list

Hyperparameters are listed in Table 4.

C Network Details

The network details follow Kostrikov et al. [35] except for the heads. The image is resized into an 84-by-84 tensor with frame-stacking from the last 3 steps. This tensor is then processed by the convolutional encoder which consists of four convolutional layers with 3-by-3 kernels and ReLu between them. The encoder uses 32 filters and stride 1 except for the first layer, which uses stride 2. This spatial map is then fed into a linear layer which outputs a 50-dimensional vector, which is processed by a layer norm [6] unit. The MLPs for the critic and actor are feedforward networks that alternate between linear transformations and Relu, using a width of 1024 and two hidden layers.

For the modern architecture we use the transformer feedforward architecture [67] with layer norm applied before the input; i.e., we have $x + w_2 \text{Relu}\big(w_1 \text{norm}(x)\big)$. Dropout is not used. The width of the residual branch is 1024 whereas the linear transformations upscale and downscale the width to 2048. The network will need to increase the initial feature dimension to 1024 and in the final layer

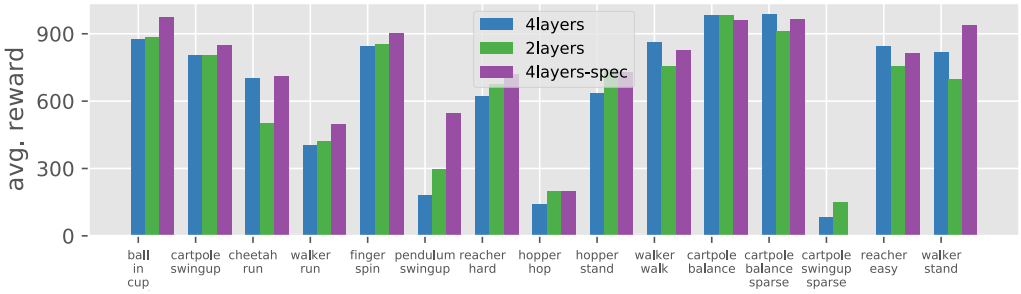

Figure 7: The performance of SAC when making the networks deeper. Performance is averaged across ten seeds and plotted for 15 tasks from the DeepMind control suite [65]. When smoothing with spectral normalization [47], the performance improves for the 4-hidden-layer network for most tasks.

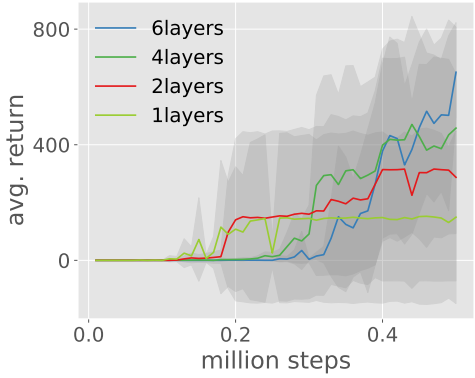

Figure 8: Small ablation experiment on cartpole swingup sparse where we vary the number of *modern* network blocks in both the critic and the actor while smoothing with spectral normalization. Performance is average across 5 seeds. Performance seems to increase, or at least not decrease, when using deeper networks.

decrease it to 1 or twice the number of actions. To this end, one linear layer is used at either end of the network. For the purpose of calculating depth, we count these two linear layers as one transformer layer (since they both use two linear layers). When using spectral normalization, the normalization is added to all *intermediate* layers — i.e. not the first and last layer. The reason for this is that we do not want the network to be 1-Lipschitz throughout, as we might need them to represent very large q-values.

D Improvements

As per Table 2, the average improvement when changing architecture from the 2 hidden layer MLP to the modern network is slightly more than 141. We can compare this to improvements gained from algorithmic innovations between the Dreamer [26] and DRQ agent [35] which Table 2 puts at less than 40. We can also compare this to the improvements gained for CURL [39], which represented a new SOTA at the time of publication. Specifically, we consult Table 1 of the CURL paper [39]. There, performance is also measured after 500,000 steps on DeepMind control suite [65], however a smaller set of environments is used. CURL reaches an average score of 846 whereas the previously published Dreamer agent [26] scores 783. This results in an improvement of 63. In follow-up work of [37], the average scores increase to 898, which is an even smaller improvement.

In Table 5 we compare using deeper networks smoothed with spectral normalization against Dreamer [26] and DRQ [35], using the curves from [26, 35]. Dreamer is there run with only 5 seeds [26]. For the sake of using 10 seeds uniformly and controlling for changes in the DRQ setup, we run the baselines ourselves with code released by the authors [26, 35] in Table 2. Since we modify the

Table 4: Hyper-parameters for the experiments, following [35]. We use action repeat of 2 for the dreamer benchmark, following [26, 35].

Parameter	Value
γ	0.99
T_0	0.1
τ	0.01
α_{adam}	1e-3
ϵ_{adam}	1e-8
$\beta_1$adam	0.9
$\beta_2$adam	0.999
batch size	512
target update freq	2
seed steps	5000
log σ bounds	[-10, 2]
actor update frequency	2
seed steps	1000
action repeat	2

Table 5: Comparison of algorithms across tasks from the DeepMind control suite. We compare SAC with a modern architecture and spectral normalization [47] against two state-of-the-art agents: SAC with two layers and image augmentations [35] (DRQ), and Dreamer [26]. Here we use numbers for [26, 35] for DRQ and Dreamer obtained through communications with the authors. Note that these do not use 10 seeds consistently, and some tasks appear to have aan artificially low standard deviation. Using modern networks and smoothing still comes out ahead, beating dreamer at 11 out of 15 games and beating MLP at 13 out of 15 games.

	ball in cup catch	swingup	cheetah run	walker run
DRQ	945.9 $\pm$ 20.7	825.5 $\pm$ 18.2	701.7 $\pm$ 22.1	465.1 $\pm$ 27.2
Modern+smooth	**968.9** $\pm$ 13.3	**862.3** $\pm$ 15.2	**708.9** $\pm$ 46.2	**501.0** $\pm$ 57.0
Dreamer	961.7 $\pm$ 2.2	678.4 $\pm$ 108.2	630.0 $\pm$ 16.1	463.5 $\pm$ 37.0
	finger spin	reacher hard	pendulum swingup	hopper hop
DRQ	**905.7** $\pm$ 43.2	824.0 $\pm$ 35.7	584.3 $\pm$ 114.5	208.3 $\pm$ 11.5
Modern+smooth	882.5 $\pm$ 149.0	**875.1** $\pm$ 75.5	586.4 $\pm$ 381.3	**254.2** $\pm$ 67.7
Dreamer	338.6 $\pm$ 24.3	95.2 $\pm$ 26.1	**789.2** $\pm$ 25.7	120.0 $\pm$ 18.5
	hopper stand	walker walk	balance	balance sparse
DRQ	755.9 $\pm$ 39.7	897.5 $\pm$ 24.3	**993.5** $\pm$ 2.7	903.2 $\pm$ 63.4
Modern+smooth	**854.7** $\pm$ 63.5	**902.2** $\pm$ 67.4	984.8 $\pm$ 29.4	968.1 $\pm$ 72.3
Dreamer	609.7 $\pm$ 76.2	885.8 $\pm$ 12.6	981.8 $\pm$ 3.1	**997.7** $\pm$ **0.9**
	swingup sparse	reacher easy	walker stand	
DRQ	520.4 $\pm$ 114.1	804.8 $\pm$ 31.0	944.4 $\pm$ 17.8	
Modern+smooth	620.8 $\pm$ 311.0	**814.0** $\pm$ 85.8	953.0 $\pm$ 19.8	
Dreamer	**799.7** $\pm$ 7.4	429.5 $\pm$ 82.3	**960.0** $\pm$ 7.0	

architecture and source code of DRQ, running experiments with 2 hidden layer MLPs ourselves allows us to make sure that the setup is identical – even details such as CUDA versions, minor code changes, GPUs, and seeds can have a large effect in RL [30]. Our setup of modern networks and smoothing beats Dreamer for 11 out of 15 games and beats the 2-hidden-layer MLP of DRQ for 13 out of 15 games in Table 5. At any rate, we do not make any specific claims about what algorithms might be the best — only that one can improve SAC from pixels/DRQ by using deeper networks with smoothing.

Table 6: Results for the DDPG agent after 1 million frames. Using a larger modern network results in poor performance compared to a traditional MLP, but once SN is used larger networks are beneficial. Metrics are given over 10 seeds.

task	**MLP**	**modern**	**modern + SN**
cup catch	972.76 ± 2.03	105.58 ± 27.65	978.89 ± 1.73
walker walk	581.57 ± 143.06	57.29 ± 25.95	958.07 ± 2.32
cartpole sparse	999.54 ± 0.44	11.31 ± 0.13	1000.00 ± 0.00
hopper stand	567.84 ± 137.13	4.39 ± 1.28	832.01 ± 86.13
reacher easy	929.52 ± 15.71	86.60 ± 15.56	957.29 ± 11.7

E Compute Resources

Experiments were conducted with Nvidia Tesla V100 GPUs using CUDA 11.0 and CUDNN 8.0.0.5, running on nodes with Intel Xeon CPUs. Experiments contained in the paper represent approximately a year or two of GPU time.