Towards Deeper Deep Reinforcement Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract

In computer vision and natural language processing, innovations in model architecture that lead to increases in 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 only 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 soft actor-critic (SAC) algorithms. We verify, empirically, 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 intrinsic instability from the actor in SAC taking gradients through the critic is the culprit. We demonstrate that a simple smoothing method can mitigate this issue, which enables stable training with large modern architectures. After smoothing, larger models yield dramatic performance improvements for state-of-the-art agents — 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, 35]. The earliest neural networks were often shallow [39] with performance dropping for excessively deep models [28]. However, ever since the introduction of batch normalization [30] and residual connections [28], performance has improved more or less monotonically with model scale [62]. As a result, competitive models in computer vision and NLP are growing ever larger [10, 55], 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 [34, 36, 38] and performance gains typically originate from algorithmic innovations such as novel loss functions [57, 60, 66, 69] rather than increasing model capacity. 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 and more redundant environment samples. Additionally, a recent large-scale study has shown that large networks can harm performance in RL [2]. 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, 30] and skip connections [28]. We thus depart from the trend of treating networks as black-box function approximators. To limit the scope and compute requirements we focus on a single setting, continuous

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

control from pixels [63] with the soft actor-critic (SAC) algorithm [24]. This popular agent forms the basis of many SOTA algorithms [34, 37, 38]. As can be expected, we demonstrate that naïvely adopting modern architectures leads to poor performance, supporting the idea that RL might overfit with models that are too large. However, by dissecting the learning dynamics we show that the issue is not necessarily overfitting, but instead, that training becomes unstable with deeper modern networks. We hypothesize that SAC taking the gradient of the actor through the critic network creates exploding gradients [52] for deeper networks, and motivate this argument mathematically. We connect this setup with generative adversarial networks (GAN) [21], and propose to use a simple smoothing technique from the GAN literature named *spectral normalization* [46] to stabilize training.

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 for RL from pixels. Our work suggests that network scaling is complementary with algorithmic innovations and that this simple strategy should not be overlooked. Further, one may argue that resource consumption should become a part of the evaluation protocol in RL, as it already is within computer vision and NLP. We summarize our contributions as follows:

- We verify empirically that large modern networks fail for SAC. 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, and provide mathematical motivation for this. To combat this problem we propose to adopt spectral normalization [46] from the GAN literature.
- We demonstrate that this simple smoothing method enables the use of large modern networks and leads to large improvements on SOTA methods for 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 $(\mathcal{S}, \mathcal{A}, P, r)$ [61]. 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 Soft Actor-Critic (SAC)

Soft actor-critic (SAC) [24] is an algorithm for continuous control with roots in q-learning [67], forming the backbone of many recent state-of-the-art algorithms [34, 37, 38, 69]. Due to its popularity and performance, we focus on this algorithm throughout the paper. Given some features $\phi(\mathbf{s})$ that can be obtained from every 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 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 $\pi_\theta(\mathbf{s})$. To encourage exploration, SAC adds the entropy of the policy distribution $H(\pi_\theta(\mathbf{s}_t))$ times a parameter α to the rewards. α can be a fixed hyperparameter or be updated dynamically to encourage the entropy to

match some predefined value. 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:

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

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. The loss (2) is computed by sampling transitions from a replay buffer [47]. 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, SAC samples an action $\mathbf{a}_\theta$ from $\pi_\theta(\mathbf{s}_t)$ and then takes derivatives of $Q_\psi(\mathbf{a}_\theta, \mathbf{s})$ with respect to θ. 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).

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 [32, 33]. Specifically we evaluate on the DeepMind control suite [63], which has been used in [25, 26, 34, 38]. We use the 15 tasks considered in Kostrikov et al. [34] and evaluate after 500,000 samples, which has become a common benchmark [34, 38, 40]. If learning crashes before 500,000 steps (this occasionally happens for modern networks), we report the performance before the crash. As is now common we use image augmentations on the raw images; this idea has been proposed independently by Laskin et al. [36] and Kostrikov et al. [34]. We start from the SAC from pixels implementation of Kostrikov et al. [34] (also known as DRQ) and adopt the default hyperparameters which are 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 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. In the experiments, we only vary the head architecture. This reduces the computational burden and suffices to illustrate our arguments. Changing encoder architecture is deferred to future work. Evaluation is done over ten seeds, with images showing the mean and standard error.

3.2 Testing Modern networks

Two crucial architectural innovations which have enabled deep supervised models are residual connections [28] and normalization [30]. We test if adopting such modern techniques allows us

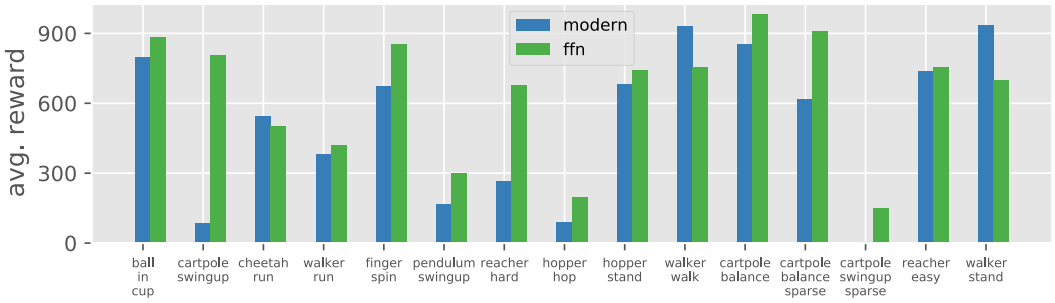

Figure 1: Performance of SAC when using a modern network architecture with skip connections [28] and normalization [6], averaged across ten seeds. Using modern architectures does not enable deeper networks for SAC; instead, performance often decreases — sometimes catastrophically — while increasing for a handful of tasks. This is in stark contrast with supervised learning where performance typically improves monotonically with model capacity.

to successfully scale networks in SAC. Specifically, we use the feedforward network (and not the attention module) found in a Transformer block [65], and will refer to this architecture as *modern*. Each unit 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 strongly favored in practice [49, 68]. We use four transformer blocks for the critic, which is responsible for learning the environment rewards, and two for the actor. Results from these experiments are shown in Figure 1, and we see that these modifications typically 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 feedforward networks deeper is a competitive strategy. To do this, we increase the number of 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. 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. As we shall see, it is possible to improve the performance of these deep networks by just stabilizing training, which suggests that there indeed is some instability.

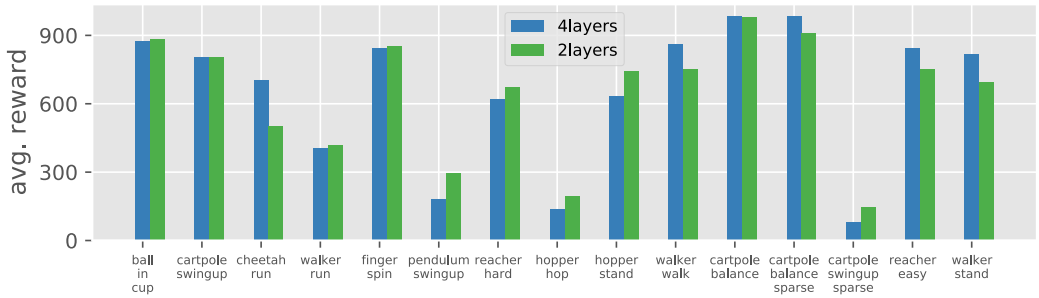

Figure 2: The performance of SAC when making the networks deeper. Performance is averaged across ten seeds and plotted for 15 tasks from the DeepMind control suite [63]. Making the network deeper does not monotonically improve performance as is typically the case in supervised learning. Instead, some tasks benefit, and some tasks suffer. As we shall see later, it is possible to improve the performance by stabilizing training, suggesting that there indeed is 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 sample sizes simply lead to overfitting, which is known to be harmful in RL [74]. This 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 now 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 scale of the gradients for the critic and actor networks and see that they indeed are much 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.

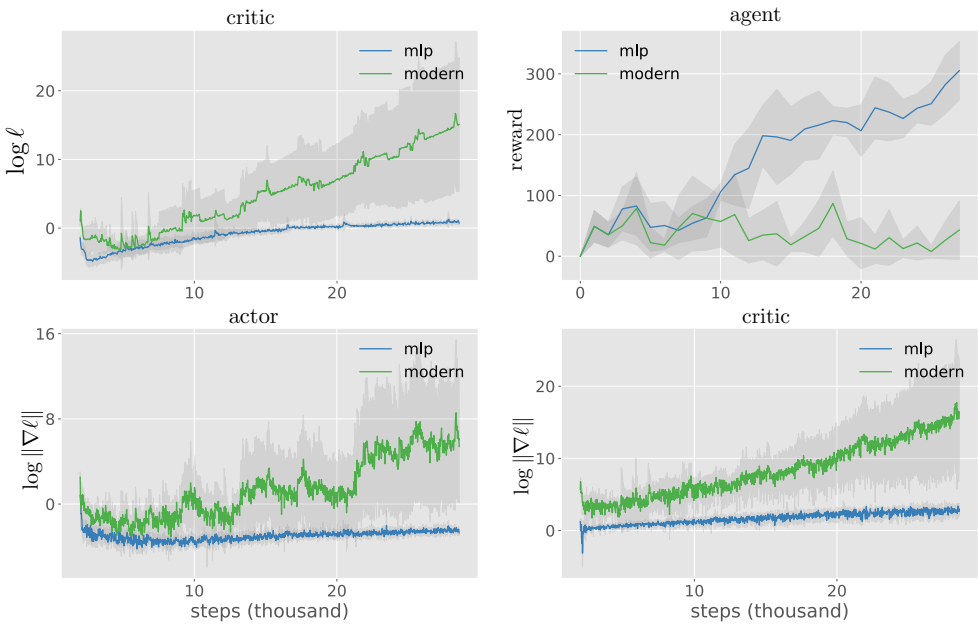

Figure 3: Training dynamics when using a small MLP or a deeper network with normalization and residual connections. **Top left.** The loss, as per eq. (2), of the critic during training. The loss does not decrease when making the network larger, suggesting that overfitting is not the issue. **Top right.** 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 The Importance of Smoothness

4.1 Taking Gradients Through the Critic

To understand why the actor might be more unstable for high-capacity networks, let us consider a simple model. We will assume that the action space has only 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 each using a non-linearity f (e.g. ReLu) elementwise after each layer. The weights are given as matrices w_i^a or w_i^c for layer i and the actor and critic respectively, and we thus have $y_i = f(w_i x_i)$ for layer i. The case with bias is similar and discussed in Appendix D. If we let the functions representing the actor and critic be denoted by A and Q, and let $\|$ denote vector concatenation, we have:

$$A(\mathbf{s}) = \Big(\prod_i f \circ w_i^a\Big) \circ \phi(\mathbf{s}) \qquad Q(\mathbf{s}) = Q(A(\mathbf{s}), \phi(\mathbf{s})) = \Big(\prod_i f \circ w_i^c\Big) \circ \big(A(\mathbf{s}) \| \phi(\mathbf{s})\big) \tag{3}$$

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 function transformations that individually have bounded Lipschitz constants $\|w_i\|$, where $\|w_i\|$ is the operator norm of matrix w_i, equal to its largest singular value $\sigma_{\max}$. We thus have:

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

Here $\|f\|$ is the Lipschitz constant of the function f. For bounding the gradients of the actor, one can use an inductive strategy to bound the gradients of a layer with respect to its output. This gives us the following bound for $\|\nabla_{w^a} Q\|$, the Frobenius norm of the gradient of the actor weights:

Proposition 1 *Under the model given above, one can bound the actor gradients $\|\nabla_{w^a} Q\|$ as follows*

$$\|\nabla_{w^a} Q\| \leq \frac{\sqrt{N}}{(\max_i \|w_i^a\|)} \|\phi(\mathbf{s})\| \|f\|^{2N} \prod_{j=1}^{N} \|w_j^a\| \prod_{k=1}^{N} \|w_k^c\| \tag{4}$$

(proof in Appendix D). In practice, the non-linear function f is typically a Relu function for which $\|f\| = 1$, so that the above expression simplifies to essentially be the product of the spectral norms of the layers of the critic and actor, divided by the norm of one layer. If all layers have approximately the same norm, we would expect that gradient magnitude to grow larger with deeper models, as we empirically observe in Figure 3.

4.2 Enforcing Smoothness

Motivated by eq. (4), we would like to enforce smoothness in both the critic and actor. This should then ensure that the gradient of the actor could not explode. Fortunately, the problem of ensuring network smoothness has been studied extensively in the literature of generative adversarial networks (GANs) [21]. When training 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. Similarly, in SAC, the agent outputs an action $\mathbf{a}$ aiming to maximize $Q(\mathbf{a}, \mathbf{s})$, whereas in GANs, the generator outputs an image $\mathbf{i}$ aiming to maximize the confidence $P(\mathbf{i})$ of the discriminator. Within the GAN literature, many ideas for ensuring network smoothness have been proposed, e.g., constraining weight norms [56], clipping weights [5], biasing networks towards orthogonal weights [9], and penalizing sharp gradients [23]. One technique especially relevant here is spectral normalization [46], where the weight W for each layer is divided by its largest singular value $\sigma_{\max}$. This ensures that all layers have operator norm 1 and should directly constrain the gradient of the actor as per eq. (4). The singular value can be expensive to compute in general, and to this end, one can obtain two vectors u and v – approximately equal to the right and left vectors of the largest singular value, and then approximate $\sigma_{\max} \approx u^T W v$. This gives us the following forward pass:

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

For this to work properly, the vectors u and v should be close to the vectors corresponding to the largest singular value. This can easily be achieved via the power method [45, 46] by taking:

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

If we assume that we can apply this method directly to each layer of our theoretical model above, and that $\|f\| = 1$ as is the case for ReLu, we obtain from eq. (4)

$$\|\nabla_w A(x)\| \leq \frac{\sqrt{N}}{(\max_i \|w_i^a\|)} \|\phi(\mathbf{s})\| \|f\|^{2N} \prod_{k=1}^{N} \|w_k^a\| \prod_{k=1}^{N} \|w_k^c\| \leq \sqrt{N} \|\phi(\mathbf{s})\|$$

This bounds the gradient update of the actor-network, which we already suspect might drive the poor performance of the deeper networks. In the following sections, we evaluate experimentally whether this idea enables deeper networks to be used on RL.

5 Experiments

5.1 Smoothness Enables Larger Networks

We first want to investigate whether smoothing with spectral normalization [46] 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 applied to each layer. 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. These results are in line with the simple model of Section 4. 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-layer MLPs. Thus, we conclude that enforcing smoothness allows SAC to utilize deeper networks.

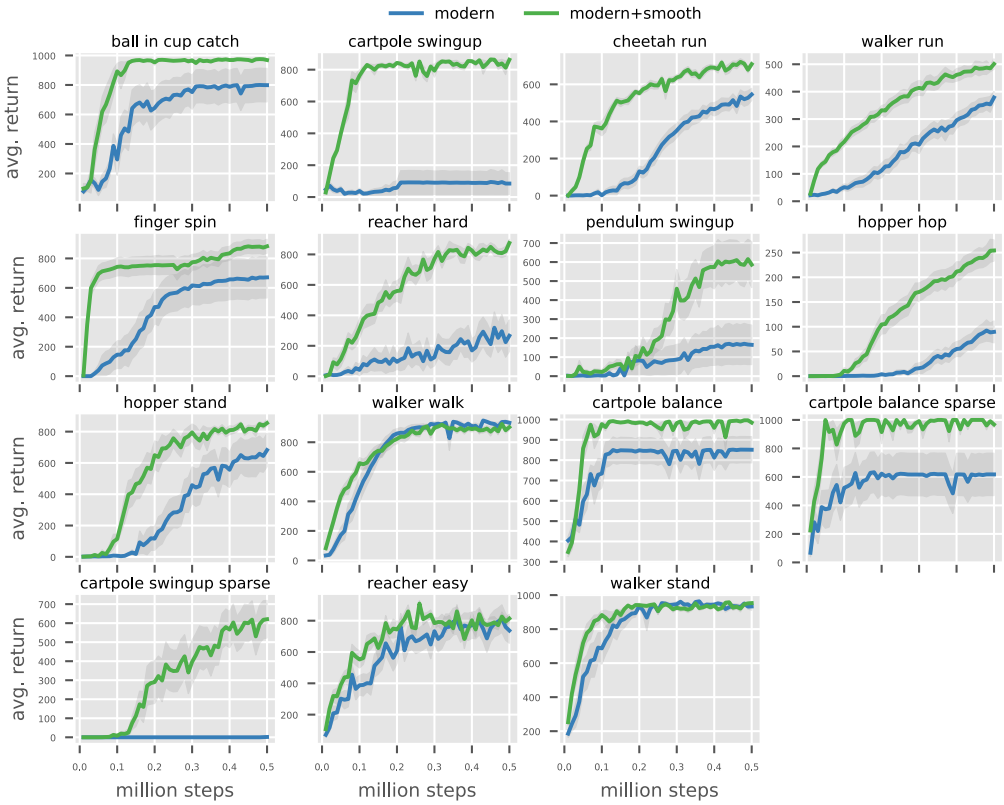

Figure 4: Learning curves for SAC using deep modern networks, with and without smoothing by spectral normalization [46]. We see that smoothing enables the deep network to learn effectively across all tasks, whereas naively using such a deep network often leads to slow learning and sometimes no learning at all.

5.2 Comparing to Other Agents

After demonstrating that smoothing with spectral normalization [46] allows SAC to use larger modern networks and improve its performance, we now see if these gains allow us to beat other agents. We compare against two state-of-the-art methods for continuous control: Dreamer [25] and SAC with augmentations and the original 2-layer MLP architecture, corresponding to the DRQ agent in [34]. Scores at step 500,000, averaged across 10 seeds, are shown in Table 1. 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 [53, 59], 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, see Appendix E for details. Thus, 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 Performance Cost

We have shown how scaling networks can improve performance without algorithmic innovations, but larger networks can substantially increase GPU memory footprint and compute time. Thus, we now 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 layers. Recall that we only modify the head networks, which share features processed by a common convolutional network that is not modified. We consider both training with batch size 512 and interacting with the environment, the

Table 1: Comparison of algorithms across tasks from the DeepMind control suite. We compare SAC with a modern architecture and spectral normalization [46] against two state-of-the-art agents: SAC with two layers and image augmentations [34] (DRQ), and Dreamer [25]. 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 ± 262.0	805.7 ± 64.9	502.8 ± 315.5	419.2 ± 170.5
Modern+smooth	$\mathbf{968.9} \pm 13.3$	$\mathbf{862.3} \pm 15.2$	$\mathbf{708.9} \pm 46.2$	$\mathbf{501.0} \pm 57.0$
Dreamer	767.1 ± 63.3	592.4 ± 31.3	630.7 ± 23.3	466.2 ± 21.1
	finger spin	reacher hard	pendulum swingup	hopper hop
DRQ	851.7 ± 156.8	674.7 ± 332.0	297.8 ± 352.6	196.1 ± 72.0
Modern+smooth	$\mathbf{882.5} \pm 149.0$	$\mathbf{875.1} \pm 75.5$	$\mathbf{586.4} \pm 381.3$	$\mathbf{254.2} \pm 67.7$
Dreamer	466.8 ± 42.1	65.7 ± 16.5	494.3 ± 98.9	136.0 ± 28.5
	hopper stand	walker walk	balance	balance sparse
DRQ	743.1 ± 280.5	754.3 ± 356.7	981.5 ± 39.6	910.5 ± 268.6
Modern+smooth	$\mathbf{854.7} \pm 63.5$	$\mathbf{902.2} \pm 67.4$	$\mathbf{984.8} \pm 29.4$	$\mathbf{968.1} \pm 72.3$
Dreamer	663.7 ± 70.3	$863. \pm 27.2$	938.2 ± 13.0	935.2 ± 22.5
	swingup sparse	reacher easy	walker stand	
DRQ	148.7 ± 299.3	753.9 ± 205.0	696.9 ± 329.4	
Modern+smooth	$\mathbf{620.8} \pm 311.0$	814.0 ± 85.8	953.0 ± 19.8	
Dreamer	267.8 ± 25.4	$\mathbf{830.4} \pm 31.4$	$\mathbf{955.3} \pm 8.7$	

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 2. 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 compute. This demonstrates that simply scaling the head networks can be relatively cheap for end-to-end RL from pixels while dramatically improving performance.

6 Related Work

Early work on deep RL primarily used simple feedforward networks, possibly processing images with a convolutional network [47, 48]. This is still a common strategy [34, 36, 38], 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 [31] or Transformers [50, 51]. There is also work on how to utilize, but not necessarily scale, neural networks in RL, e.g. dueling heads [66] or double q-learning [64]. The proposition that too-large networks perform poorly has been verified for policy gradient methods in the systematic study of Andrychowicz et al. [2]. There is also ample systematic work on generalization in RL. Cobbe et al. [11] performs a large-scale study on generalization in RL and finds that agents overfit to surprisingly large training sets, perhaps as the training set and test set overlap. Zhang et al. [72] studies procedurally generated mazes and find that RL agents can overfit by memorizing levels and that common techniques to avoid overfitting (e.g., sticky actions [18]) fail to do so. Other relevant large-scale studies include [15, 29, 74]. Spectral normalization has been used to regularize model-based reinforcement learning by Yu et al. [70], however, the effects of this seem to not be studied by the authors. There are many proposed ideas for enforcing smoothness in neural

Table 2: 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 1 come at a relatively cheap price. 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

networks, especially for GANs [5, 9, 21, 23, 56]. The idea of directly constraining the singular value has been investigated by multiple authors [19, 22, 46].

7 Discussion

The Importance of Architecture. Research on neural network architecture is arguably less popular nowadays as compared to the late 2010s. 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 such low-level design 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 [49, 62], and similar studies could be fruitful in RL. Such large-scale studies on network architecture in RL could be as impactful as theoretical innovations, and we encourage more focus on network architecture. 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, 44, 54].

Evaluation in RL. Supervised learning has already reached the state where performance seems to monotonically improve with network capacity [62]. 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 should require listing the amount of compute resources used for novel methods. This is especially important for compute-intensive unsupervised methods [1, 38, 43, 58, 71, 73] or model-based learning [4, 12, 20, 27, 41]. There are two common approaches for ensuring fair comparisons: using a standard architecture across algorithms or listing the amount of compute/memory consumption and compare methods on this basis. The second one seems most suitable for RL as the objectives can differ between agents, which necessitates architectural changes.

Limitations. For compute reasons we have only studied the effects of networks on the SAC algorithm — which is a state-of-the-art algorithm for continuous control underlying many popular algorithms – nonetheless, there are plenty of other algorithms. The idea of obtaining gradients through subnetworks is common and algorithms such as DDPG [42] and Dreamer [25] might also benefit from smoothing. Spectral normalization [46] 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 [24]. Naively implementing such changes does not necessarily improve performance, and they can lead to unstable training. To resolve this issue, we have proposed to enforce smoothing via spectral normalization [46], and provided theoretical justification for this. 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 changing network architecture can be competitive with algorithmic innovations in RL.

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] 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.

[22] 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.

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

[24] 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.

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

[26] 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.

[27] 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.

[28] 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.

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

[30] 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.

[31] 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.

[32] 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.

[33] 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.

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

[35] 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.

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

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

[38] 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.

[39] 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.

[40] 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.

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

[42] 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.

[43] 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.

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

[45] 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.

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

[47] 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.

[48] 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, 2016.

[49] 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.

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

[51] 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.

[52] 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.

[53] 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.

[54] 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.

[55] 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.

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

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

[58] 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.

[59] 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.

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

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

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

[63] 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.

[64] 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.

[65] 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.

[66] 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.

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

[68] 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.

[69] D. Yarats, R. Fergus, A. Lazaric, and L. Pinto. Reinforcement learning with prototypical representations. *arXiv preprint arXiv:2102.11271*, 2021.

[70] 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.

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

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

[73] 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.

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

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]

(c) Did you discuss any potential negative societal impacts of your work? [Yes]

(d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]

2. If you are including theoretical results...

(a) Did you state the full set of assumptions of all theoretical results? [Yes]

(b) Did you include complete proofs of all theoretical results? [Yes]

3. If you ran experiments...

(a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]

(b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes]

(b) Did you mention the license of the assets? [Yes] Licenses are attached with submitted code.

(c) Did you include any new assets either in the supplemental material or as a URL? [N/A]

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]