Overcoming Policy Collapse in Deep Reinforcement Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract

A long-awaited characteristic of reinforcement learning agents is scalable performance, that is, to continue to learn and improve performance with a never-ending stream of experience. However, current deep reinforcement learning algorithms are known to be brittle and difficult to train, which limits their scalability. For example, the learned policy can dramatically worsen after some initial training as the agent continues to interact with the environment. We call this phenomenon *policy collapse*. We first establish that policy collapse can occur in both policy gradient and value-based methods. Policy collapse happens in these algorithms in typical benchmarks such as Mujoco environments when trained with their commonly used hyper-parameters. In a simple 2-state MDP, we show that the standard use of the Adam optimizer with its default hyper-parameters is a root cause of policy collapse. Specifically, the standard use of Adam can lead to sudden large weight changes even when the gradient is small whenever there is non-stationarity in the data stream. We find that policy collapse can be successfully mitigated by using the same hyper-parameters for the running averages of the first and second moments of the gradient. Additionally, we find that aggressive L2 regularization also mitigates policy collapse in many cases. Our work establishes that a minimal change in the existing usage of deep reinforcement learning can mitigate policy collapse and enable more stable and scalable deep reinforcement learning.

1 Introduction

AI systems that take advantage of available data and computation tend to outperform systems that do not (Sutton, 2019). Historically, in games like Chess and Go, systems that utilize the available data and computation have defeated all other systems (Campbell et al., 2002; Silver et al., 2016). Most recently, large language models like GPT-4 (OpenAI, 2023) have dramatically outperformed previous natural language processing systems, primarily due to the amount of data and computation used by them. The need to improve performance with data is particularly relevant for reinforcement learning systems (Sutton and Barto, 2018) as they experience a potentially unending data stream.

Unfortunately, the performance of many current deep reinforcement learning algorithms does not always improve with more experience. The policy learned by these algorithms can dramatically worsen as the agent continues interacting with the environment, a phenomenon we call *policy collapse*. The evidence of policy collapse is scattered throughout the reinforcement learning literature. For instance, policy collapse can be observed in several reinforcement learning algorithms such as DQN, PPO, and DDPG, as shown in papers by Shaul et al. (2016, Figure 7), Henderson et al. (2018, Figure 2), and Tassa et al. (2018, Figure 4), respectively.

Although we can observe policy collapse in several works, it has not been pointed out and studied in the literature. As a first step, we establish in Section 3 that policy collapse can occur in two

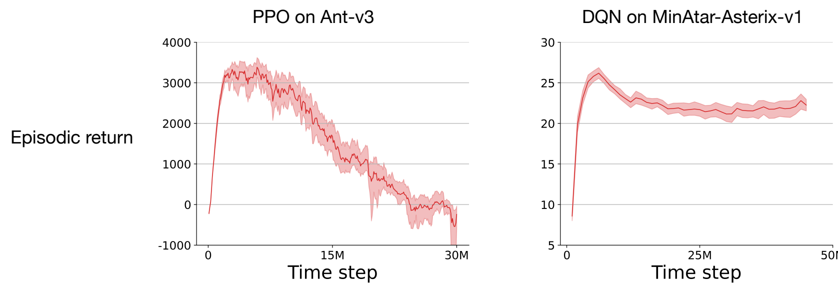

Figure 1: PPO and DQN do not scale with data. As these reinforcement learning agents continue to interact with the environment, instead of improving, their performance degrades. In the case of PPO, the policy completely collapses and performs worse than it did in the beginning. While for DQN, the policy does not fully collapse, but there is still a significant degradation in performance.

widely-used algorithms Proximal Policy Optimization, (PPO), Deep Q-Network (DQN) in typical benchmarks such as Mujoco environments (Todorov et al., 2012). Figure 1 shows a representative performance of PPO in one of the OpenAI gym (Brockman et al., 2016) environments and that of DQN in a MinAtar environment (Young and Tian, 2019). For these experiments, we employed commonly used hyper-parameter settings for PPO and DQN, the details of which are in Appendix A, and the rest of the experimental details are in Section 3. Figure 1 shows that the performance of both algorithms drops as they continue to interact with the environment. This performance degradation in PPO and DQN means that they do not scale with data; their performance worsens with more data. Policy collapse is generally absent in the literature because experiments with PPO on Mujoco environments are usually only run for 3M time steps and DQN on MinAtar for 5M time steps.

Policy collapse is reminiscent of two phenomena in continual learning, catastrophic forgetting and loss of plasticity. Deep networks are known to forget previously acquired information when trained on a non-stationary stream of data (McCloskey and Cohen, 1989; French, 1999; Parisi et al., 2019). Similar to continual learning, reinforcement learning agents have to learn from a non-stationary stream of data due to a changing policy, bootstrapping, etc. When the agent suffers from policy collapse, it has forgotten the good policy it had learned at one point. The second problem deep networks face when learning from a non-stationary stream of data is loss of plasticity (Dohare et al., 2021; Lyle et al., 2022; Nishikin et al., 2022; Abbas et al., 2023), where deep networks lose the ability to learn new things. Once the agent forgets the good policy, it does not re-learn it, which could be because the agent has lost plasticity. We take a deep look at policy collapse in a 2-state MDP and understand its connections to plasticity loss and forgetting, in Section 4. Then in section 5, we show how simple solutions help mitigate policy collapse in DQN and PPO. And finally, in section 6, we discuss the connection between forgetting and plasticity in reinforcement learning.

2 Background

In reinforcement learning (RL), an agent learns to achieve its goal by trial and error. The problem of reinforcement learning can be mathematically formalized as a Markov Decision Process (MDP). Formally, let $M = (\mathcal{S}, \mathcal{A}, \mathrm{P}, r, \gamma)$ be an MDP which includes a state space $\mathcal{S}$, an action space $\mathcal{A}$, a state transition probability function $\mathrm{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$, a reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, and a discount factor $\gamma \in [0, 1)$. For a given MDP, the agent interacts with the MDP according to its policy π, which maps a state to a distribution over the action space. At each time-step t, the agent observes a state $S_t \in \mathcal{S}$ and samples an action A_t from $\pi(\cdot|S_t)$. Then it observes the next state $S_{t+1} \in \mathcal{S}$ according to the transition function P and receives a scalar reward $R_{t+1} = r(S_t, A_t) \in \mathbb{R}$. Considering an episodic task with horizon T, we define the return G_t as the sum of discounted rewards, that is, $G_t = \sum_{k=t}^{T} \gamma^{k-t} R_{k+1}$. The action-value function of policy π is defined as $q_\pi(s, a) = \mathbb{E}[G_t|S_t = s, A_t = a], \forall (s, a) \in \mathcal{S} \times \mathcal{A}$. Similarly, the state-value function v_π maps states to expected returns, $v_\pi(s) = \mathbb{E}[G_t|S_t = s], \forall s \in \mathcal{S}$.

The agent aims to find an optimal policy π^* that maximizes the expected return starting from some initial states. Generally, there are two approaches: value-based methods and policy gradient methods. For example, Q-learning (Watkins and Dayan, 1992) is a value-based method designed to learn an action-value function estimate $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ of an optimal policy. Mnih et al., (2015) generalized Q-learning to DQN, which uses a neural network to approximate the action-value function. Furthermore, to improve training stability, Mnih et al., (2015) introduced a target neural network $\hat{Q}$, a copy of the Q-network and it is updated less frequently than the Q-network. In contrast to value-based methods, policy gradient methods directly learn a policy. Among policy gradient methods, PPO (Schulman et al., 2107) is one of the most widely used. The key idea of PPO is to constrain the policy update by using a clipped surrogate objective to prevent the policy from changing too much. In practice, to achieve good performance and improve training stability, a practical implementation of PPO usually employs a lot of tricks (Huang et al., 2022), such as advantage normalization, loss clipping, gradient clipping, entropy regularization, etc.

Gradient descent algorithms like Stochastic Gradient Descent (SGD) are widely recognized as one of the most prevalent algorithms for training neural networks. However, the performance of SGD is very sensitive to the learning rate. And choosing a suitable learning rate can be difficult and time-consuming. To address this challenge, researchers have developed many adaptive optimizers that are less sensitive to learning rates, significantly improving optimization performance (Duchi et al., 2011; Tieleman and Hinton, 2012; Kingma and Ba, 2015). Among them, Adam (Kingma and Ba, 2015) is one of the most popular adaptive optimizers and is the optimizer of choice in many applications of deep reinforcement learning. It keeps an exponentially moving average of past gradients and squared gradients, updating parameter θ as the following:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\, g_t \text{ and } \hat{m}_t = m_t/(1 - \beta_1^t)$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)\, g_t^2 \text{ and } \hat{v}_t = v_t/(1 - \beta_2^t)$$
$$\theta_t = \theta_{t-1} - \alpha\, \hat{m}_t/(\sqrt{\hat{v}_t} + \epsilon)$$

where g_t is the gradient, m and v are initialized as 0s, $\beta_1, \beta_2 \in [0, 1)$ are two hyper-parameters, and α is the learning rate. In practice, the default βs ($\beta_1 = 0.9, \beta_2 = 0.999$) are usually used (Abadi et al., 2015, Paszke et al., 2019; Babuschkin et al., 2020) without further adjustment in both supervised learning and reinforcement learning.

3 Policy Collapse in Deep Reinforcement Learning

Here, we test the stability and scalability of Proximal Policy Optimization (PPO) on Mujoco environments and Deep-Q Networks (DQN) on MinAtar environments. PPO is a policy gradient method, while DQN is a value-based method, so these algorithms cover two types of commonly used algorithms. We chose PPO as a representative for policy gradient methods as it is computationally cheap, which allows us to perform thorough and reproducible experiments. Additionally, it has been used in many successful applications, from Dota-2 (OpenAI et al., 2019) to reinforcement learning from human feedback in ChatGPT. We chose DQN as it can perform human-level control in Atari games (Mnih et al., 2015). To test the scalability of these methods, we need to run the experiments longer than the common practice in the literature. Typically, DQN is tested on the Arcade learning environment (ALE) (Bellemare et al., 2019) for 200M frames, which takes almost a week to run on modern GPUs. Running these experiments for longer makes it computationally infeasible to perform thorough and reproducible experiments. Young and Tian (2019) developed the MinAtar environments, which capture many critical behavioural aspects of the ALE while allowing good learning with DQN in order of magnitude fewer time steps than ALE. DQN can achieve good performance on MinAtar in just 5M time steps, making it feasible to run long experiments. As a natural choice for our long experiments, we chose to test DQN on MinAtar instead of the ALE.

In the first experiment, we tested the scalability of PPO. Usually, PPO is trained for 1-3 million time steps on the Mujoco environments. However, to test the stability and scalability of PPO we trained it for up to 100M time steps. We used the standard setting of hyper-parameters for PPO, and their values are given in Appendix A. We used the undiscounted episodic return as the measure of performance. The results of the experiments are shown in Figure 2. The x-axis in the plots is the time step, and the y-axis is the undiscounted episodic return in bins of 100k times steps. The first points in the plots are the average return for the episodes in the first 100k time steps, and the next point is the average return for the episodes in the next 100k time steps and so on. We performed 30 independent runs for each environment. In all our experiments we report the 95% bootstrapped confidence interval as suggested by Patterson et al. (2023). In Figure 2, the shaded region shows the 95% bootstrapped confidence interval.

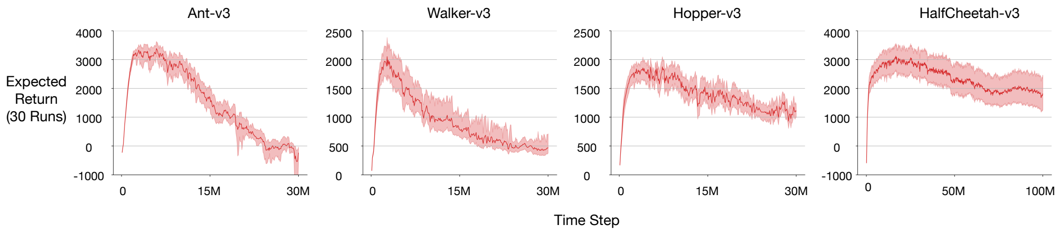

Figure 2: PPO on Mujoco environments. After initial learning, the policy learned by PPO kept degrading. And, in many cases, its performance dropped as low as it was in the beginning. PPO did not scale with data. Instead of improving, its performance decreased with more experience.

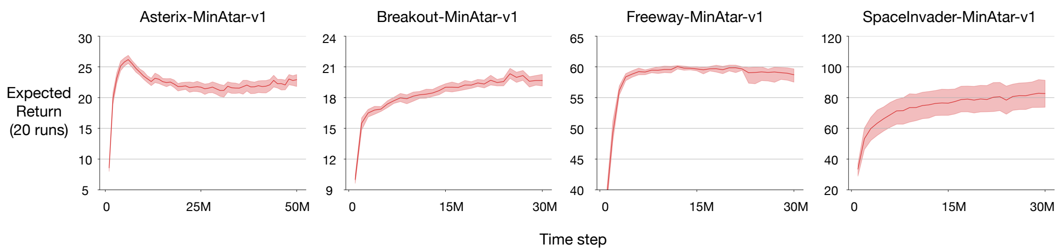

Figure 3: DQN on MinAtar environments. Similar to PPO, after initial learning, the performance of DQN decreased. However, the performance did not degrade in all environments, and the degradation was not as severe as that of PPO. DQN did not always scale with data, and its performance did not improve with more experience.

The performance of PPO had a similar trend in all the environments. Performance improved for the first few million time steps, then it hit a plateau, and finally, it dropped, usually to a level below what it had in the first 100k time steps. Another thing to note is that once the performance dropped, it did not improve, suggesting that the agent might have lost plasticity, which is the ability to learn new things. These results show that PPO does not scale with data as it displays consistent policy collapse when used with standard hyper-parameters.

In the next experiment, we tested the scalability of DQN in MinAtar environments. We used the DQN implementation from the Tianshou library (Weng et al., 2022). DQN can get to a level of good performance on MinAtar in 5M time steps. To test the scalability of DQN we trained it for 50M time steps. The results of experiments with DQN are shown in Figure 3. In Figure 3, the x-axis is the time step, and the y-axis is the undiscounted episodic return in bins of 1M times steps. We performed 20 independent runs for each environment.

Figure 3 shows that the performance of DQN degraded some environments, Asterix and Freeway. In these environments, the performance of DQN first improved, then plateaued, and finally, it started to drop. The drop in the performance of DQN was gradual and not as common as that of PPO. DQN did not always scale with more data, its policy got worse in some environments with more experience, although the policy did not fully collapse.

The experiments in this section showed that policy collapse severely affects PPO. Similarly, DQN is also affected by policy collapse, although the collapse is not as severe as that in PPO. These experiments point out a major problem with current deep reinforcement learning algorithms: they are not stable during training, and their scalability is limited due to policy collapse.

4 Understanding Policy Collapse in a Small MDP

To overcome policy collapse, we first need to understand what happens to the learning agent when the policy collapses. However, fully understanding policy collapse in modern deep reinforcement learning algorithms in standard environments is difficult because deep reinforcement learning algorithms have many interacting parts, such as bootstrapping, off-policy learning, function approximation, exploration, and changing policy. Policy collapse could be due to the deadly triad (Baird, 1995; Sutton, 1995), where reinforcement learning algorithms can diverge if they face off-policy learning,

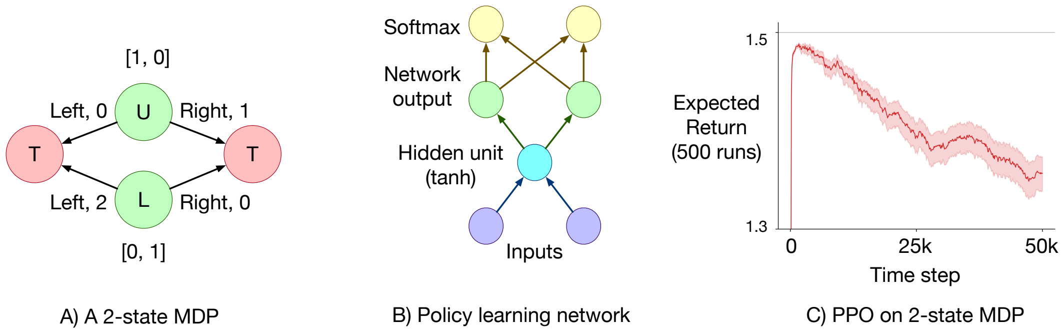

Figure 4: Policy collapse in a 2-state MDP. The figure on the right shows the 2-state MDP, the figure in the middle shows the network used by the learning agent, and the plot on the right shows the performance of PPO on this 2-state MDP. A 2-dimensional vector represents the states of the MDP. When PPO learns using a function approximator, it lacks stability, and its performance degrades as the agent continues interacting in the MDP. PPO is unstable even in this simple MDP.

bootstrapping, and function approximation. Additionally, modern deep reinforcement learning algorithms have dozens of hyper-parameters, and a wrong setting of any one of them could be causing policy collapse. To make matters worse, the environments where deep reinforcement learning algorithms are tested are extremely complicated and computationally expensive, which makes it impossible to do a full grid search over the hyperparameter space.

We used a simple MDP to gain insights into policy collapse in modern deep reinforcement learning algorithms. The MDP consists of two states, the agent can take two actions, left and right, in both states. Both actions take the agent to the terminal state. However, the reward associated with each action is different. The MDP is shown in Figure 4A. After termination, the agent starts with equal probability in both states. There are four deterministic policies for this MDP, and the best one is to choose the right action in state U and the left action in state L. The expected return for the optimal policy is 1.5, while for other deterministic policies, it is 1.0, 0.5, and 0.0.

We trained an agent using PPO on this MDP. The agent used a neural network to learn a policy. The neural network had one hidden layer with one unit and tanh activation. The input to the network is a two-dimensional vector, which is $[1, 0]$ when the agent is in state U and $[0, 1]$ when the agent is in state L. The action probabilities are obtained by passing the network outputs into a softmax operator. Figure 4B shows the policy learning network. The agent used a different network with one hidden layer and one hidden unit to learn a value function for the current policy. The hyper-parameter values for PPO for this experiment are shown in Appendix A.

We use the expected return as the performance measure in this experiment. As we have access to the entire state space, we can calculate the exact expected return instead of approximating it using the actual return. We performed 500 independent runs for this experiment. The results are shown in Figure 4C. In Figure 5A, we plot the performance of a single run.

The performance of PPO in Figure 4C follows the same trend as its performance in the Mujoco environments. The performance of PPO improves at first, then it plateaus at a high level and finally drops. We find two interesting things to note if we look at the individual runs of PPO in Figure 5A. First, the agent gets stuck at different sub-optimal policies. And second, the learned policy is very unstable. It fluctuates rapidly between different policies.

The agent uses a neural network and a softmax policy parameterization. As the optimal policy is deterministic, the optimal values of the weights have infinite magnitude. This means that the gradient will force the outgoing weights in the network to grow. The probability of taking a different action will decrease exponentially as the weights increase. Once the weights become large enough, the agent almost never takes an exploratory action. In Figure 5B, we plot the outgoing weights of the policy network, and as we suspected, the weights kept increasing over time. The large magnitude of outgoing weights explains why the learned policy gets stuck at different deterministic policies.

The sudden jumps in the policy in Figure 5A suggest that there might be sudden large changes in the representation provided by the single hidden unit. Figure 5C shows the absolute difference in the output of the hidden unit for the two states. Let's call the output of the hidden unit for a given

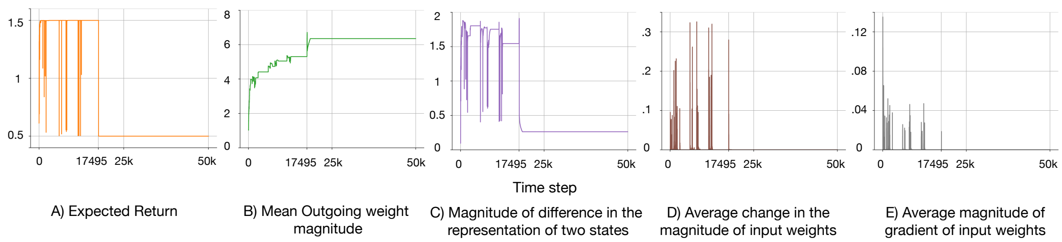

Figure 5: A deep look at one specific run of PPO on the 2-state MDP. Different figures plot the evolution of different quantities as the agent interacted with the MDP. The magnitude of the output weights of the network kept increasing (Figure B) because the optima lie at infinity, making it difficult to try exploratory actions. Once the representation of the two states became sufficiently similar, at time step 17495 (Figure C), the agent kept taking the same action in both states. This resulted in the agent getting stuck at a sub-optimal policy (Figure A). The sudden large changes in input weights (Figure D) caused sudden large changes in the representation and the learned policy. These large changes were caused by the standard use of the Adam optimizer, which caused large weight changes even when the gradient was small (Figure E).

state the representation of the state. Note that when the difference in the representation of the two states is small, the states will look similar to the final layer. Figure 5 shows that sudden jumps in the policy happen whenever the difference in the state representation changes dramatically. The sudden changes in the representation should follow large changes in the input weights, which we observe in Figure 5D. In Figure 5E, we plot the average magnitude of the gradients of the input layer. Perhaps surprisingly, there are large changes in the input weights even when the gradient is small, such as at time step 17495.

The large updates, even when the gradient is small, are due to the Adam optimizer. Adam keeps running averages of the first and second moments of the gradient. The averages use β_1 and β_2 to control the importance of recent gradients in the average. In this experiment, we set $\beta_1 = 0.9$ and $\beta_2 = 0.999$, which are the default βs in popular deep learning frameworks like Pytorch. In the individual run, when the policy changes at time step 17495, there is a sudden non-zero gradient. The sudden non-zero gradient happened because the agent took an exploratory action. Assuming that the gradient before and after time step 17495 is exactly zero, then during the next ten updates, these values of β_1 and β_2 would lead to an update that is 20 times larger than the gradient (see Appendix B for details). This large weight change can lead to a large change in the policy, which explains why the learned policy fluctuates so abruptly.

Policy collapse in the 2-state MDP gave us various insights into the phenomenon. We learned that the agent gets stuck at sub-optimal policies when the representation does not separate the two states and outgoing weights become too large. And we found that the instability in PPO is due to the standard use of Adam($\beta_1 = 0.9$, $\beta_2 = 0.999$), which caused sudden changes in the policy even when the gradient was small. These sudden changes caused the agent to forget the previously learned policy. This instability caused by the standard use of Adam could be the reason why Adam has been observed to cause more forgetting than SGD (Ashley et al., 2021). Similar to our analysis, Lyle et al. (2023) found that the standard use of Adam causes issues learning from a non-stationary stream of data. They showed that standard use of Adam worsens the loss of plasticity. In complement to that, we showed that the standard use of Adam also causes forgetting.

5 Overcoming Policy Collapse

Insights from policy collapse in the 2-state MDP guide us towards potential solutions to policy collapse. The first idea is to use L2 regularization (Goodfellow et al., 2016, pp. 227-230), it shrinks all the weights in the network by a constant rate. L2 regularization has also been shown to reduce the loss of plasticity (Dohare et al., 2021). It can reduce the effect of large updates. And it will reduce feature saturation by reducing weight magnitudes, enabling the network to have a good representation. Another idea is to tackle the problem of large updates even when the gradients are small. Lyle et al. (2023) suggested in their plasticity work to use equal values for β_1 and β_2. Recall that these parameters control the rate of the running averages for the first and second moments of the gradient.

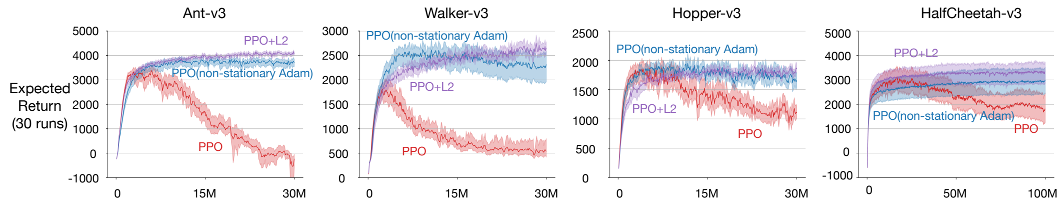

Figure 7: PPO with non-stationary Adam and L2 on Mujoco environments. Both methods successfully mitigate policy collapse in all environments, improving the scalability of PPO.

Adam, with equal values of β_1 and β_2, is also intuitively more reasonable in non-stationary problems like reinforcement learning, where the optimization landscape and the scale of the gradient can change after each update. We call Adam with equal values for β_1 and β_2 *non-stationary Adam*.

First, we tested if PPO with L2 regularization or non-stationary Adam can overcome policy collapse in our MDP. The experiment design and hyper-parameters were the same as in Section 4. The additional setting for the hyper-parameters for L2 and non-stationary-Adam is present in Appendix A. The results are plotted in Figure 6.

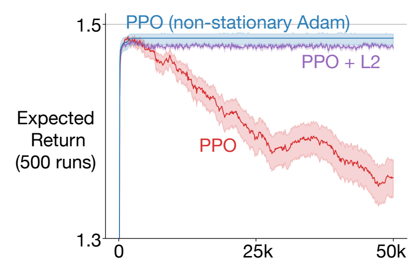

Figure 6: PPO with L2 regularization and non-stationary Adam on the 2-state MDP. Both L2 and non-stationary Adam successfully mitigate policy collapse. PPO+L2 is unstable as there are sudden jumps in the policy, but the constant weight shrinking by L2 enables the agent to find the optimal policy again. While, the policy learned by PPO + non-stationary Adam is more stable.

The data in Figure 6 shows that both L2 regularization and non-stationary Adam can maintain a good level of performance. Although both non-stationary Adam and L2 get to a high level of performance, they do not get to the optimal policy, a return of 1.5, in all runs. Another thing to note is that the performance of non-stationary Adam was more stable than L2. This is not surprising as non-stationary Adam fixes the source of instability while L2 only reduces the harm caused by the instability of standard Adam. Both L2 regularization and non-stationary Adam mitigated policy collapse with PPO in our MDP.

In the next experiment, we test if PPO using L2 regularization or non-stationary Adam can overcome policy collapse in Mujoco environments. First, we compare these methods when using standard parameters for PPO. The experiment design is the same as in Section 3. For L2 regularization, we tuned the weight-decay parameter, while for non-stationary Adam, we tuned β_1, and had $\beta_2 = \beta_1$. We chose the hyper-parameter that had the highest performance after 30M time steps. The results of the experiment are shown in Figure 7.

Figure 7 shows that in all the environments, the performance of non-stationary Adam and L2 does not get worse over time. Their performance kept improving with more experience. Both methods overcome policy collapse for the standard setting of PPO hyper-parameters in the Mujoco environments. The performance of PPO with non-stationary Adam and L2 scaled with experience.

Deep reinforcement learning methods like PPO are sensitive to their hyper-parameter settings. In the next experiment, we tested if PPO with non-stationary Adam or L2 can avoid policy collapse for different parameter settings. We tested these methods in Ant-v3 for three different hyper-parameters for PPO. The first is ReLU activation (Nair and Hinton, 2010) instead of tanh, as ReLU has been shown to perform a lot worse than tanh(Andrychowicz et al., 2020). Second, we use a larger value of the clipping parameter in PPO, this would cause larger changes in the policy. And third, was to use more updates after each buffer is collected, good performance in this case will improve the sample efficiency of PPO. And fourth, we use a smaller replay buffer. For this experiment, we tuned the weight-decay and step-size for L2, and beta and step-size for non-stationary Adam. The details of the hyper-parameters are shown in Appendix A. Again, we chose the hyper-parameter with the highest performance after 30M time steps. The results of these experiments are shown in Figure 8.

The data in Figure 8 shows that PPO with L2 and non-stationary Adam had a different performance for some parameter settings. For a larger value of the clipping parameter, both L2 and non-stationary

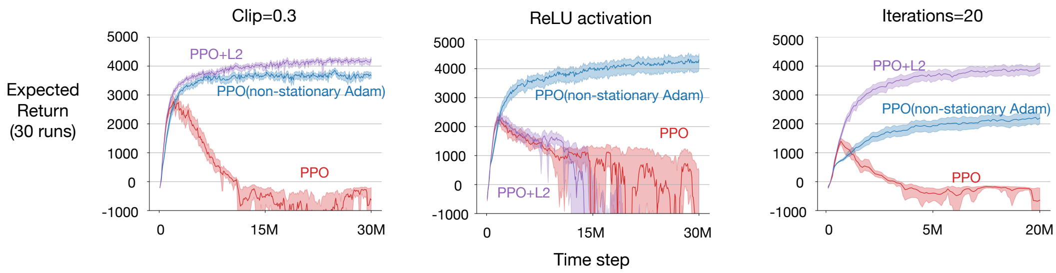

Figure 8: PPO, PPO with L2, and PPO with non-stationary Adam with different hyper-parameter settings of PPO. For all these hyper-parameters, PPO performs worse than its standard hyper-parameter setting. PPO+L2 only mitigated policy collapse in all cases except when we used the ReLU activation. Both methods reduced the parameter sensitivity of PPO and non-stationary Adam successfully mitigated policy collapse in all cases.

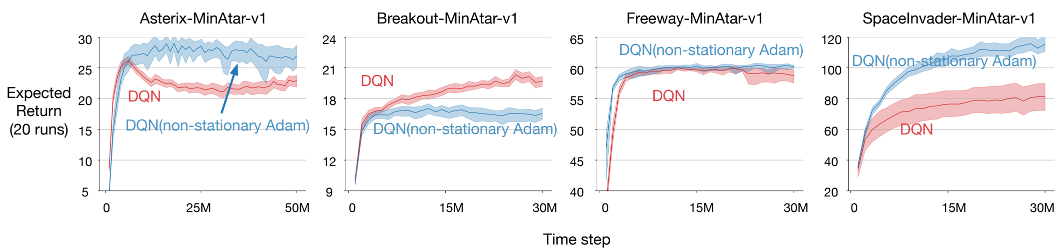

Figure 9: DQN with non-stationary Adam on MinAtar environments. Non-stationary Adam successfully mitigates policy degradation in Asterix and Freeway. Non-stationary Adam improves the stability of DQN.

Adam mitigated policy collapse as well as had a good performance (Figure 8A). However, when used with the ReLU activation, L2 did not achieve good performance (Figure 8B). And, when using a large number of iterations, L2 significantly outperformed non-stationary Adam (Figure 8C). An important thing to note is that L2 only performed well for a large value of weight decay (0.001). For smaller values of weight decay, we observed policy collapse. While, for larger values of weight decay, the performance was never good. L2 regularization overcame policy collapse in all cases except when using ReLU activation. While Non-stationary Adam mitigated policy collapse in all cases, although it did not perform well when using a large number of iterations. Both L2 regularization and non-stationary Adam reduced the sensitivity of PPO to its hyper-parameters.

Finally, we tested if DQN with non-stationary Adam can overcome policy degradation in MinAtar environments. We tuned β_1 for non-stationary Adam; other parameters were the same as in Section 3. We performed 20 independent runs and performed the experiments in all environments. The results of the experiment are shown in Figure 9.

Figure 9 shows that the performance of DQN with non-stationary Adam does not get worse over time in any environment. It performed better than regular DQN on SpaceInvader but worse on Breakout. DQN with non-stationary Adam scales with data as its performance improves with more data.

In conclusion, we tested whether deep reinforcement learning algorithms can overcome policy collapse using L2 regularization or non-stationary Adam. We found that L2 regularization helped overcome policy collapse in all cases except when using ReLU activation. On the other hand, non-stationary Adam overcame policy collapse in all cases with both PPO and DQN. The performance of deep reinforcement learning algorithms with non-stationary Adam kept improving with data. Non-stationary Adam enabled stable and scalable deep reinforcement learning.

6 Discussion

Policy collapse is related to the forgetting problem faced by neural networks in non-stationary problems. When the policy collapses, the agent has forgotten the previously learned policy. This form of forgetting differs from what is commonly studied in continual learning, where forgetting is usually studied in a sequence of stationary supervised learning problems. Although some prior works (Fedus et al., 2020) have studied forgetting in deep reinforcement learning algorithms, the effect of forgetting in reinforcement learning remains obscure. In our work, we found that the standard use of Adam ($\beta_1 = 0.9$, $\beta_2 = 0.999$) is a cause of forgetting. The standard use of Adam might be the reason why Adam has been observed to forget more severely than SGD (Ashley et al., 2021). And it might be the reason why most algorithms addressing forgetting do not use Adam or RMSprop; rather, they use algorithms that do not have Adam-type normalization (Mirzadeh et al., 2020).

Our work adds to evidence that methods that help mitigate plasticity loss also improve reinforcement learning algorithms' performance. Recent works have shown that plasticity injection in different forms improves the performance of reinforcement learning algorithms (D'Oro et al., 2023; Sokar et al., 2023; Nikishin et al., 2023; Lyle et al., 2023). The work of Lyle et al. (2023) is particularly relevant to ours. They found that the standard use of the Adam optimizer significantly worsens the loss of plasticity. In contrast, we found that the standard use of Adam also causes policy collapse and forgetting. Interestingly, the original DQN (Mnih et al., 2015) used the same rate for averaging the first and second moments of the gradient. However, recent implementations of DQN, like DQN Zoo (Quan and Ostrovski, 2020), do not have the same rate for averaging the first and second moments of the gradient. Suggesting that good parameters for averaging the first and second moments of the gradient were found, but the importance of having equal rates for the first and second moments of the gradient was overlooked, and the community moved to the now default parameters of Adam.

We showed that non-stationary Adam significantly improved the stability of deep reinforcement learning methods. A long line of work has been trying to reduce instability in deep reinforcement learning. Methods like trust region policy optimization (Schulman et al., 2015) and PPO were proposed to reduce the change in the policy, as the policy learned by vanilla actor-critic methods (Barto et al., 1983; Konda and Tsitsiklis, 1999; Mnih et al., 2016) varied a lot. Our work complements these works as non-stationary Adam and L2 regularization can be applied to all these algorithms. Various previous works have pointed out that deep reinforcement learning methods are brittle and sensitive to their hyper-parameters (Islam et al., 2017; Henderson et al., 2019). We made progress in reducing the brittleness and hyper-parameter sensitivity of deep reinforcement learning algorithms by showing that non-stationary Adam also reduced the hyper-parameter sensitivity of PPO.

After seeing the severe policy collapse in PPO, one might wonder how PPO was applied successfully to Dota-2 and ChatGPT. We found that the learning system in Dota-2 explicitly tried to counteract the problem with standard Adam. They clipped the gradient per parameter to be less than $5\sqrt{v_t}$ in magnitude (OpenAI et al., 2019, page 5). Recall that v_t is the running estimate of the second moment of the gradient, and it is used in the denominator of Adam's update. We suspect that this clipping was critical to the success of Dota-2 as it would also reduce the instability in PPO.

7 Conclusion, Limitations, and Future Work

We performed an empirical analysis of the phenomenon of policy collapse and found that the standard use of the Adam optimizer is a cause of policy collapse. This finding suggests that directly picking up tools from supervised learning can harm reinforcement learning algorithms. We need to be more careful when borrowing tools developed in other domains. A similar observation, but for recurrent networks, was made by Schlegel et al. (2022), where they pointed out that the best design choices for recurrent networks in supervised learning do not work well in reinforcement learning problems. The second conclusion is that a minimal change in the use of the Adam optimizer, by using the same rate for running averages of the first and second moment of the gradient, significantly improves the stability and scalability of reinforcement learning algorithms.

The main limitation of our work is that we did not study optimizers like SGD using momentum. These optimizers do not have the normalization like Adam. That choice was because Adam and RMSprop are the two commonly used optimizers in the literature. But it would be an exciting direction for future work to thoroughly study optimizers like SGD with momentum in reinforcement learning problems. Although we did not observe policy collapse when using non-stationary Adam, it remains to be seen if this simple solution will also fix policy collapse in larger and more complicated reinforcement learning systems. In light of this, another important direction for future work is to develop algorithms for non-stationary problems that do not suffer from forgetting and loss of plasticity, maybe something along the lines of Elsayed and Mahmood (2023).

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... & Zheng, X. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*.

Abbas, Z., Zhao, R., Modayil, J., White, A., & Machado, M. C. (2023). Loss of Plasticity in Continual Deep Reinforcement Learning. *arXiv preprint arXiv:2303.07507*.

Andrychowicz, M., Raichuk, A., Stańczyk, P., Orsini, M., Girgin, S., Marinier, R., ... & Bachem, O. (2020). What matters in on-policy reinforcement learning? a large-scale empirical study. *arXiv preprint arXiv:2006.05990*.

Ashley, D. R., Ghiassian, S., & Sutton, R. S. (2021). Does the Adam Optimizer Exacerbate Catastrophic Forgetting?. *arXiv preprint arXiv:2102.07686*.

Babuschkin, I., Baumli, K., Bell, A., Bhupatiraju, S., Bruce, J., Buchlovsky, P., ... & Viola, F. (2020). *The DeepMind JAX ecosystem*. https://github.com/deepmind

Barto, A. G., Sutton, R. S., & Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, (5), 834-846.

Bellemare, M. G., Naddaf, Y., Veness, J., & Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. Journal of Artificial Intelligence Research, 47, 253-279.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). Openai gym. *arXiv preprint arXiv:1606.01540*.

Campbell, M., Hoane Jr, A. J., & Hsu, F. H. (2002). Deep blue. *Artificial intelligence*, 134(1-2), 57-83.

Dohare, S., Sutton, R. S., & Mahmood, A. R. (2021). Continual backprop: Stochastic gradient descent with persistent randomness. *arXiv preprint arXiv:2108.06325*.

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*.

D'Oro, P., Schwarzer, M., Nikishin, E., Bacon, P. L., Bellemare, M. G., & Courville, A (2023). Sample-Efficient Reinforcement Learning by Breaking the Replay Ratio Barrier. In *International Conference on Machine Learning* 2023.

Elsayed, M., & Mahmood, A. R. (2023). Utility-based Perturbed Gradient Descent: An Optimizer for Continual Learning. *arXiv preprint arXiv:2302.03281*.

Fedus, W., Ghosh, D., Martin, J. D., Bellemare, M. G., Bengio, Y., & Larochelle, H. (2020). On catastrophic interference in atari 2600 games. *arXiv preprint arXiv:2002.12499*.

French, R. M. (1999). Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences, 3*(4), 128-135.

Goodfellow, I., Bengio, Y., & Courville, A., 2016. *Deep Learning*. MIT Press

Hinton, G., Srivastava, N., & Swersky, K (2012). Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. page 14, 2012.

Huang, S., Dossa, R. F. J., Raffin, A., Kanervisto, A., & Wang, W. (2022). The 37 implementation details of proximal policy optimization. *The ICLR Blog Track 2023*.

Islam, R., Henderson, P., Gomrokchi, M., & Precup, D. (2017). Reproducibility of benchmarked deep reinforcement learning tasks for continuous control. *arXiv preprint arXiv:1708.04133*.

Kingma, D.P., & Ba, J., 2015. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2015.

Konda, V., & Tsitsiklis, J. (1999). Actor-critic algorithms. *Advances in neural information processing systems*, 12.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., ... & Wierstra, D. (2015). Continuous control with deep reinforcement learning. In *International Conference on Learning Representations*, 2016.

Lyle, C., Rowland, M., & Dabney, W. (2022). Understanding and preventing capacity loss in reinforcement learning. In *International Conference on Learning Representations*, 2022.

Lyle, C., Zheng, Z., Nikishin, E., Pires, B. A., Pascanu, R., & Dabney, W. (2023). Understanding plasticity in neural networks. *arXiv preprint arXiv:2303.01486*.

McCloskey, M., & Cohen, N. J. (1989). Catastrophic interference in connectionist networks: The sequential learning problem. *In Psychology of learning and motivation* (Vol. 24, pp. 109-165). Academic Press.

Mirzadeh, S.I., Farajtabar, M., Pascanu, R., & Ghasemzadeh, H. (2020). Understanding the role of training regimes in continual learning. *Advances in Neural Information Processing Systems*, 33.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., ... & Kavukcuoglu, K. (2016, June). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning* (pp. 1928-1937). PMLR.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *nature, 518*(7540), 529-533.

Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. *In Proceedings of the 27th international conference on machine learning (ICML-10)* (pp. 807-814).

Nikishin, E., Oh, J., Ostrovski, G., Lyle, C., Pascanu, R. Dabney, W., & Barreto, A. (2023). Deep Reinforcement Learning with Plasticity Injection, *Workshop on Reincarnating Reinforcement Learning at ICLR 2023*.

Nikishin, E., Schwarzer, M., D'Oro, P., Bacon, P. L., & Courville, A. (2022, June). The primacy bias in deep reinforcement learning. In *International Conference on Machine Learning* (pp. 16828-16847). PMLR.

OpenAI. (2023). GPT-4 technical report. *arXiv*.

OpenAI, Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., ... & Zhang, S. (2019). Dota 2 with large scale deep reinforcement learning. arXiv preprint arXiv:1912.06680.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... & Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.

Parisi, G. I., Kemker, R., Part, J. L., Kanan, C., & Wermter, S. (2019). Continual lifelong learning with neural networks: A review. *Neural networks*, 113, 54-71.

Patterson, A., Neumann, S., White, M., & White, A. (2023). Empirical Design in Reinforcement Learning. *arXiv preprint arXiv:2304.01315*.

Quan, J. & Ostrovski, G. (2020). DQN Zoo: Reference implementations of DQN-based agents. http://github.com/deepmind/dqn_zoo

Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2015). Prioritized experience replay. In *International Conference on Learning Representations*, 2016.

Schlegel, M. K., Tkachuk, V., White, A. M., & White, M (2022). Investigating Action Encodings in Recurrent Neural Networks in Reinforcement Learning. *Transactions on Machine Learning Research*.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., & Moritz, P. (2015, June). Trust region policy optimization. In *International conference on machine learning* (pp. 1889-1897). PMLR.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., ... & Hassabis, D. (2017). Mastering the game of go without human knowledge. *nature, 550*(7676), 354-359.

Sokar, G., Agarwal, R., Castro, P. S., & Evci, U. (2023). The Dormant Neuron Phenomenon in Deep Reinforcement Learning. In *International Conference on Machine Learning* 2023.

Sutton, R. (2019). *The bitter lesson. Incomplete Ideas (blog)*, 13(1).

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Tassa, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., Casas, D. D. L., ... & Riedmiller, M. (2018). Deepmind control suite. *arXiv preprint arXiv:1801.00690*.

Tieleman, T. & Hinton, G. (2012). Lecture 6.5-RMSProp: Divide the gradient by a running average of its recent magnitude. *COURSERA Neural Networks Neural Networks for Machine Learning*.

Todorov, E., Erez, T., & Tassa, Y. (2012, October). Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems* (pp. 5026-5033). IEEE.

Watkins, C. J., & Dayan, P. (1992). Q-learning. *Machine learning*, 8, 279-292.

Weng, J., Chen, H., Yan, D., You, K., Duburcq, A., Zhang, M., ... & Zhu, J. (2022). Tianshou: A highly modularized deep reinforcement learning library. *Journal of Machine Learning Research*, 23(267), 1-6.

Young, K., & Tian, T. (2019). Minatar: An atari-inspired testbed for thorough and reproducible reinforcement learning experiments. *arXiv preprint arXiv:1903.03176*.

A Hyper-Parameter Settings

This section presents the hyper-parameters used in different experiments throughout the paper. First, Table 1 contains the default hyper-parameters for PPO for all experiments in the Mujoco environments. The hyper-parameters described in Table 1 are used to generate the plots in Figures 1 and 2. Next, Table 2 contains the default hyper-parameters for PPO for experiments in the 2-state MDP. These hyper-parameters are used to generate the plots in Figures 4, 5, and 6. For experiments with non-stationary Adam using PPO, we used the best β out of $\{.99, .997, .999\}$. Figures 6, 7, and 8 used these values for betas for non-stationary Adam. For PPO+L2, we swept over weight decay of $1e-2, 1e-3, 1e-4, 1e-5$. In all Mujoco environments, we found that weight decay of $1e-3$ performed best. These values for weight decay were used in Figures 7 and 8. For PPO+L2 in the 2-state MDP, we swept over weight decay of $1e-2, 1e-3, 1e-4, 1e-5, 1e-6, 1e-7, 1e-8$. We found that PPO with weight decay of $1e-6$ had the best performance after 50k time steps in the 2-state MDP. This value of weight decay was used in Figure 6.

Table 1: Default hyper-parameters for PPO in Mujoco environments

Name	Default Value
Policy Network	(64, tanh, 64, tanh, Linear) + Standard deviation variable
Value Network	(64, tanh, 64, tanh, Linear)
Buffer size	2048
Num epochs	10
Mini-batch size	256
GAE, λ	0.95
Discount factor, γ	0.99
clip parameter	0.2
Input Normalization	False
Advantage Normalization	True
Value function loss clipping	False
Gradient clipping	False
Optimizer	Adam
Optimizer step size	0.0003
Optimizer (β_1, β_2)	(0.9, 0.999)
Optimizer ϵ	$1e-8$

Table 2: Default hyper-parameters for PPO in the 2-state MDP

Name	Default Value
Policy Network	(1, tanh, Linear) + Standard deviation variable
Value Network	(1, tanh, Linear)
Buffer size	8
Num epochs	10
Mini-batch size	2
GAE, λ	0.95
Discount factor, γ	0.99
clip parameter	0.2
Input Normalization	False
Advantage Normalization	True
Value function loss clipping	False
Gradient clipping	False
Optimizer	Adam
Optimizer step size	0.01
Optimizer (β_1, β_2)	(0.9, 0.999)
Optimizer ϵ	$1e-8$

For DQN experiments in MinAtar games, we followed most of the hyper-parameter settings in Young and Tian (2019). Specifically, we used smooth L1 loss and the same neural network settings as Young and Tian (2019). For the first $5,000$ exploration steps, we only collected transitions without learning. ϵ-greedy was applied as the exploration strategy with ϵ decreasing linearly from 1.0 to 0.1

in $5,000$ steps. After $5,000$ steps, ϵ was fixed to 0.1. To choose the step size, we run DQN with step sizes of $\{3e-3, 1e-3, 3e-4, 1e-4, 3e-5, 1e-5\}$ for 5M time steps. The step size with the highest expected return after 5M time steps was chosen to generate the plots in Figure 3. For non-stationary Adam, we used the step size that was chosen for regular DQN. Additionally, for non-stationary Adam, we chose the best performing β out of $\{.99, .999, .9997, .9999, .99997, .99999\}$. Other default hyper-parameters are listed in Table 3.

Table 3: Default hyper-parameters for DQN in MinAtar games

Name	Default Value
Q network	(conv2d (out_channels=16), ReLU, 128, ReLU, linear)
Buffer size	1e5
Mini-batch size	32
Discount factor, γ	0.99
Gradient clipping	False
Optimizer	Adam
Optimizer (β_1, β_2)	(0.9, 0.999)
Optimizer ϵ	$1e-8$
Target Q network update steps	1000

B Large Update with Default Parameters of Adam

In an idealized case, where the gradient for a sequence of samples is zero at every update, except t, the Adam optimizer with its default parameters leads to an update that is 30 times larger than the gradient. The t^{th} update is given by,

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\, g_t \text{ and } \hat{m}_t = m_t/(1-\beta_1^t) \tag{1}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)\, g_t^2 \text{ and } \hat{v}_t = v_t/(1-\beta_2^t) \tag{2}$$

$$\theta_t = \theta_{t-1} - \alpha\, \hat{m}_t/(\sqrt{\hat{v}_t} + \epsilon) \tag{3}$$

For $\beta_1 = 0.9$, $\beta_1 = 0.999$, $\epsilon = 1e-8$, and large t, the updates at t, $t+1$ ... are $3.16 * \alpha$, $2.85 * \alpha$ and so on. In 40 updates, this corresponds to a total change of $31 * \alpha$. This explains how a single non-zero gradient can lead to a sudden large update when using Adam with its default parameters. Of course, this idealized case does not directly apply to PPO as PPO collects data for a few time steps and then makes multiple updates at a single time step using the collected data. But, it gives us an intuition of how a single non-zero gradient can lead to sudden large changes in the learned function.

C Computation Usage

A single run of our PPO implementation took about 12 hours for 30M time steps for Mujoco environments. This corresponds to around 3 CPU years of compute for all experiments on Mujoco environments. And a single run on the 2-state MDP takes at most 5 minutes on a CPU, and this corresponds to the total compute usage of around 15 CPU days for all experiments on the 2-state MDP. Finally, a single run on DQN on MinAtar took around 3.5 days. This meant a total compute usage of 4 CPU years. In total, all experiments used 7 CPU years of compute.

D Broader Impact

This work is a fundamental study in scalable online reinforcement learning. Our work may help to develop reinforcement learning systems that scale with experience. Recent advances with big models like GPT-4 are leading to a rapid centralization of power. Only a handful of massive corporations and governments can afford to store and process the data required to train such models. In contrast to these massive models, online learning models only require a small amount of data at a time to improve their performance. Developing online learning models that continually improve performance with small memory requirements can make learning systems accessible to the general population. We do not see any immediate potential negative impacts of our work.