The Dormant Neuron Phenomenon in Deep Reinforcement Learning

Ghada Sokar [1 2] **Rishabh Agarwal** [3 4] **Pablo Samuel Castro** [3 *] **Utku Evci** [3 *]

Abstract

In this work we identify the *dormant neuron phenomenon* in deep reinforcement learning, where an agent's network suffers from an increasing number of inactive neurons, thereby affecting network expressivity. We demonstrate the presence of this phenomenon across a variety of algorithms and environments, and highlight its effect on learning. To address this issue, we propose a simple and effective method (*ReDo*) that *Re*cycles *Do*rmant neurons throughout training. Our experiments demonstrate that *ReDo* maintains the expressive power of networks by reducing the number of dormant neurons and results in improved performance.

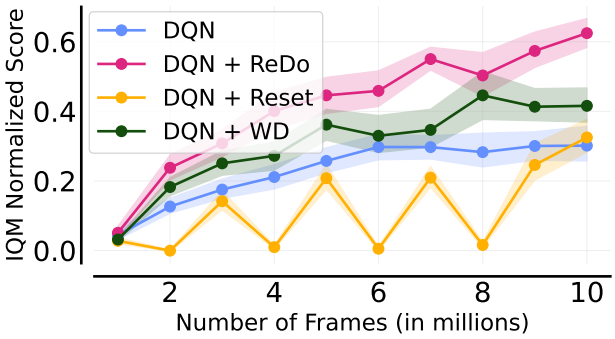

Figure 1. **Sample efficiency curves** for DQN, with a replay ratio of 1, when using network resets (Nikishin et al., 2022), weight decay (WD), and our proposed *ReDo*. Shaded regions show 95% CIs. The figure shows interquartile mean (IQM) human-normalized scores over the course of training, aggregated across 17 Atari games and 5 runs per game. Among all algorithms, *DQN+ReDo* performs the best.

1. Introduction

The use of deep neural networks as function approximators for value-based reinforcement learning (RL) has been one of the core elements that has enabled scaling RL to complex decision-making problems (Mnih et al., 2015; Silver et al., 2016; Bellemare et al., 2020). However, their use can lead to training difficulties that are not present in traditional RL settings. Numerous improvements have been integrated with RL methods to address training instability, such as the use of target networks, prioritized experience replay, multi-step targets, among others (Hessel et al., 2018). In parallel, there have been recent efforts devoted to better understanding the behavior of deep neural networks under the learning dynamics of RL (van Hasselt et al., 2018; Fu et al., 2019; Kumar et al., 2021a; Bengio et al., 2020; Lyle et al., 2021; Araújo et al., 2021).

Recent work in so-called "scaling laws" for supervised learning problems suggest that, in these settings, there is a positive correlation between performance and the number of parameters (Hestness et al., 2017; Kaplan et al., 2020; Zhai et al., 2022). In RL, however, there is evidence that the networks *lose* their expressivity and ability to fit new targets over time, despite being over-parameterized (Kumar et al., 2021a; Lyle et al., 2021); this issue has been partly mitigated by perturbing the learned parameters. Igl et al. (2020) and Nikishin et al. (2022) periodically *reset* some, or all, of the layers of an agent's neural networks, leading to improved performance. These approaches, however, are somewhat drastic: reinitializing the weights can cause the network to "forget" previously learned knowledge and require many gradient updates to recover.

In this work, we seek to understand the underlying reasons behind the loss of expressivity during the training of RL agents. The observed decrease in the learning ability over time raises the following question: *Do RL agents use neural network parameters to their full potential?* To answer this, we analyze neuron activity throughout training and track *dormant* neurons: neurons that have become practically inactive through low activations. Our analyses reveal that the number of dormant neurons increases as training progresses, an effect we coin the "*dormant neuron phenomenon*". Specifically, we find that while agents start the training with a

[*]Equal advising [1]Eindhoven University of Technology, The Netherlands [2]Work done while the author was intern at Google DeepMind [3]Google DeepMind [4]Mila. Correspondence to: Ghada Sokar <g.a.z.n.sokar@tue.nl>, Rishabh Agarwal < rishabhagarwal@google.com>, Pablo Samuel Castro <psc@google.com>, Utku Evci <evci@google.com>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

small number of dormant neurons, this number increases as training progresses. The effect is exacerbated by the number of gradient updates taken per data collection step. This is in contrast with supervised learning, where the number of dormant neurons remains low throughout training.

We demonstrate the presence of the dormant neuron phenomenon across different algorithms and domains: in two value-based algorithms on the Arcade Learning Environment (Bellemare et al., 2013) (DQN (Mnih et al., 2015) and DrQ(ϵ) (Yarats et al., 2021; Agarwal et al., 2021)), and with an actor-critic method (SAC (Haarnoja et al., 2018)) evaluated on the MuJoCo suite (Todorov et al., 2012). To address this issue, we propose *Re*cycling *Do*rmant neurons (*ReDo*), a simple and effective method to avoid network under-utilization during training without sacrificing previously learned knowledge: we explicitly limit the spread of dormant neurons by "recycling" them to an active state. *ReDo* consistently maintains the capacity of the network throughout training and improves the agent's performance (see Figure 1). Our contributions in this work can be summarized as follows:

- We demonstrate the existence of the dormant neuron phenomenon in deep RL.
- We investigate the underlying causes of this phenomenon and show its negative effect on the learning ability of deep RL agents.
- We propose *Re*cycling *Do*rmant neurons (*ReDo*), a simple method to reduce the number of dormant neurons and maintain network expressivity during training.
- We demonstrate the effectiveness of *ReDo* in maximizing network utilization and improving performance.

2. Background

We consider a Markov decision process (Puterman, 2014), $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma \rangle$, defined by a state space $\mathcal{S}$, an action space $\mathcal{A}$, a reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, a transition probability distribution $\mathcal{P}(s'|s, a)$ indicating the probability of transitioning to state s' after taking action a from state s, and a discounting factor $\gamma \in [0, 1)$. An agent's behaviour is formalized as a policy $\pi : \mathcal{S} \to Dist(\mathcal{A})$; given any state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$, the value of choosing a from s and following π afterwards is given by $Q^\pi(s, a) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, a_t)]$. The goal in RL is to find a policy π^* that maximizes this value: for any π, $Q^{\pi^*} := Q^* \geq Q^\pi$.

In deep reinforcement learning, the Q-function is represented using a neural network Q_θ with parameters θ. During training, an agent interacts with the environment and collects trajectories of the form $(s, a, r, s') \in \mathcal{S} \times \mathcal{A} \times \mathbb{R} \times \mathcal{S}$. These samples are typically stored in a *replay buffer* (Lin, 1992), from which batches are sampled to update the parameters of Q_θ using gradient descent. The optimization performed aims to minimize the temporal difference loss (Sutton, 1988): $\mathcal{L} = Q_\theta(s, a) - Q^{\mathcal{T}}_\theta(s, a)$; here, $Q^{\mathcal{T}}_\theta(s, a)$ is the bootstrap target $[\mathcal{R}(s, a) + \gamma \max_{a' \in \mathcal{A}} Q_{\tilde{\theta}}(s', a')]$ and $Q_{\tilde{\theta}}$ is a delayed version of Q_θ that is known as the *target* network.

The number of gradient updates performed per environment step is known as the *replay ratio*. This is a key design choice that has a substantial impact on performance (Van Hasselt et al., 2019; Fedus et al., 2020; Kumar et al., 2021b; Nikishin et al., 2022). Increasing the replay ratio can increase the sample-efficiency of RL agents as more parameter updates per sampled trajectory are performed. However, prior works have shown that training agents with a high replay ratio can cause training instabilities, ultimately resulting in decreased agent performance (Nikishin et al., 2022).

One important aspect of reinforcement learning, when contrasted with supervised learning, is that RL agents train on highly non-stationary data, where the non-stationarity is coming in a few forms (Igl et al., 2020), but we focus on two of the most salient ones.

Input data non-stationarity: The data the agent trains on is collected in an online manner by interacting with the environment using its current policy π; this data is then used to update the policy, which affects the distribution of future samples.

Target non-stationarity: The learning target used by RL agents is based on its own estimate $Q_{\tilde{\theta}}$, which is changing as learning progresses.

3. The Dormant Neuron Phenomenon

Prior work has highlighted the fact that networks used in online RL tend to *lose* their expressive ability; in this section we demonstrate that *dormant neurons* play an important role in this finding.

Definition 3.1. Given an input distribution D, let $h_i^\ell(x)$ denote the activation of neuron i in layer ℓ under input $x \in D$ and H^ℓ be the number of neurons in layer ℓ. We define the score of a neuron i (in layer ℓ) via the normalized average of its activation as follows:

$$s_i^\ell = \frac{\mathbb{E}_{x \in D} |h_i^\ell(x)|}{\frac{1}{H^\ell} \sum_{k \in h} \mathbb{E}_{x \in D} |h_k^\ell(x)|} \tag{1}$$

We say a neuron i in layer ℓ is τ**-dormant** if $s_i^\ell \leq \tau$.

We normalize the scores such that they sum to 1 within a layer. This makes the comparison of neurons in different layers possible. The threshold τ allows us to detect neurons with low activations. Even though these low activation neurons could, in theory, impact the learned functions when

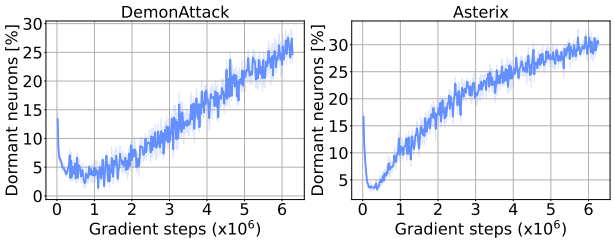

Figure 2. The percentage of dormant neurons increases throughout training for DQN agents.

recycled, their impact is expected to be less than the neurons with high activations.

Definition 3.2. An algorithm exhibits the **dormant neuron phenomenon** if the number of τ-dormant neurons in its neural network increases steadily throughout training.

An algorithm exhibiting the dormant neuron phenomenon is not using its network's capacity to its full potential, and this under-utilization worsens over time.

The remainder of this section focuses first on demonstrating that RL agents suffer from the dormant neuron phenomenon, and then on understanding the underlying causes for it. Specifically, we analyze DQN (Mnih et al., 2015), a foundational agent on which most modern value-based agents are based. To do so, we run our evaluations on the Arcade Learning Environment (Bellemare et al., 2013) using 5 independent seeds for each experiment, and reporting 95% confidence intervals. For clarity, we focus our analyses on two representative games (DemonAttack and Asterix), but include others in the appendix. In these initial analyses we focus solely on $\tau = 0$ dormancy, but loosen this threshold when benchmarking our algorithm in sections 4 and 5. Additionally, we present analyses on an actor-critic method (SAC (Haarnoja et al., 2018)) and a modern sample-efficient agent (DrQ(ϵ) (Yarats et al., 2021)) in Appendix B.

The dormant neuron phenomenon is present in deep RL agents. We begin our analyses by tracking the number of dormant neurons during DQN training. In Figure 2, we observe that the percentage of dormant neurons steadily increases throughout training. This observation is consistent across different algorithms and environments, as can be seen in Appendix B.

Target non-stationarity exacerbates dormant neurons. We hypothesize that the non-stationarity of training deep RL agents is one of the causes for the dormant neuron phenomenon. To evaluate this hypothesis, we consider two supervised learning scenarios using the standard CIFAR-10 dataset (Krizhevsky et al., 2009): (1) training a network with *fixed targets*, and (2) training a network with *non-stationary targets*, where the labels are shuffled throughout training (see Appendix A for details). As Figure 3 shows, the number of dormant neurons *decreases* over time with fixed targets, but *increases* over time with non-stationary targets. Indeed, the sharp increases in the figure correspond to the points in training when the labels are shuffled. These findings suggest that the continuously changing targets in deep RL are a significant factor for the presence of the phenomenon.

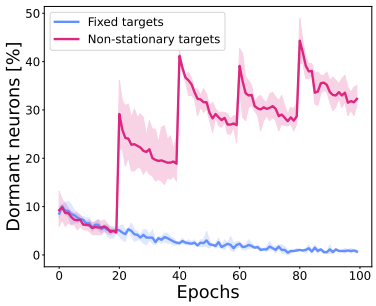

Figure 3. Percentage of dormant neurons when training on CIFAR-10 with fixed and non-stationary targets. Averaged over 3 independent seeds with shaded areas reporting 95% confidence intervals. The percentage of dormant neurons increases with non-stationary targets.

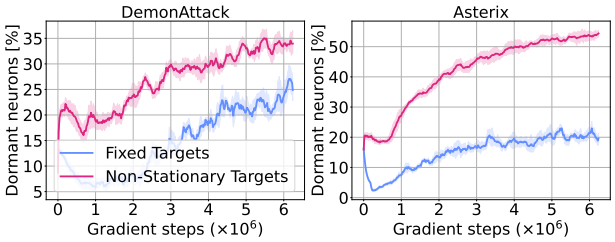

Figure 4. Offline RL. Dormant neurons throughout training with standard moving targets and fixed (random) targets. The phenomenon is still present in offline RL, where the training data is fixed.

Input non-stationarity does not appear to be a major factor. To investigate whether the non-stationarity due to online data collection plays a role in exacerbating the phenomenon, we measure the number of dormant neurons in the *offline* RL setting, where an agent is trained on a fixed dataset (we used the dataset provided by Agarwal et al. (2020)). In Figure 4 we can see that the phenomenon remains in this setting, suggesting that input non-stationary is not one of the primary contributing factors. To further analyze the source of dormant neurons in this setting, we train RL agents with *fixed* random targets (ablating the non-stationarity in inputs and targets). The decrease in the num-

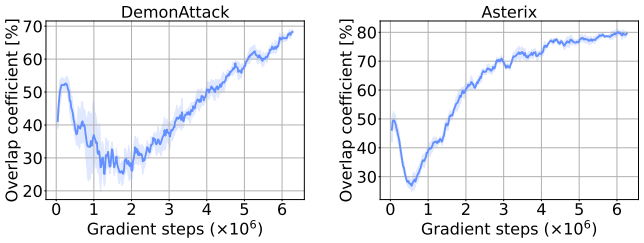

Figure 5. The overlap coefficient of dormant neurons throughout training. There is an increase in the number of dormant neurons that remain dormant.

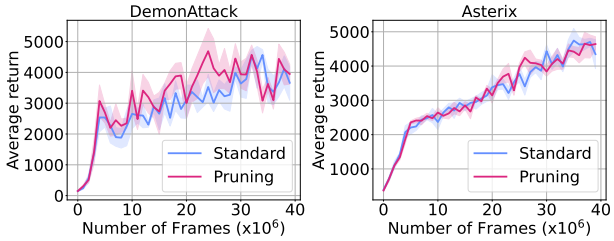

Figure 6. Pruning dormant neurons during training does not affect the performance of an agent.

ber of dormant neurons observed in this case (Figure 4) supports our hypothesis that target non-stationarity in RL training is the primary source of the dormant neuron phenomenon.

Dormant neurons remain dormant. To investigate whether dormant neurons "reactivate" as training progresses, we track the overlap in the set of dormant neurons. Figure 5 plots the overlap coefficient between the set of dormant neurons in the penultimate layer at the current iteration, and the historical set of dormant neurons.[1] The increase shown in the figure strongly suggests that once a neuron becomes dormant, it remains that way for the rest of training. To further investigate this, we explicitly *prune* any neuron found dormant throughout training, to check whether their removal affects the agent's overall performance. As Figure 6 shows, their removal does *not* affect the agent's performance, further confirming that dormant neurons remain dormant.

More gradient updates leads to more dormant neurons. Although an increase in replay ratio can seem appealing from a data-efficiency point of view (as more gradient updates per environment step are taken), it has been shown to cause overfitting and performance collapse (Kumar et al., 2021a; Nikishin et al., 2022). In Figure 7 we measure neuron dormancy while varying the replay ratio, and observe a strong correlation between replay ratio and the fraction of neurons turning dormant. Although difficult to assert conclusively, this finding could account for the difficulty in training RL agents with higher replay ratios; indeed, we will demonstrate in Section 5 that recycling dormant neurons and activating them can mitigate this instability, leading to better results.

[1]The overlap coefficient between two sets X and Y is defined as $overlap(X, Y) = \frac{|X \cap Y|}{\min(|X|, |Y|)}$.

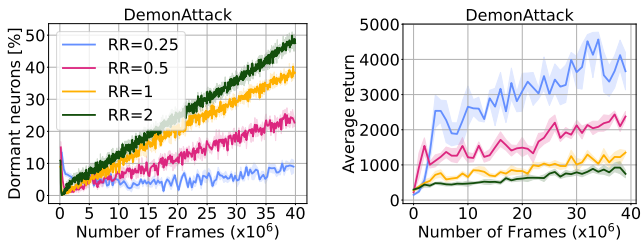

Figure 7. The rate of increase in dormant neurons with varying replay ratio (RR) (left). As the replay ratio increases, the number of dormant neurons also increases. The higher percentage of dormant neurons correlates with the performance drop that occurs when the replay ratio is increased (right).

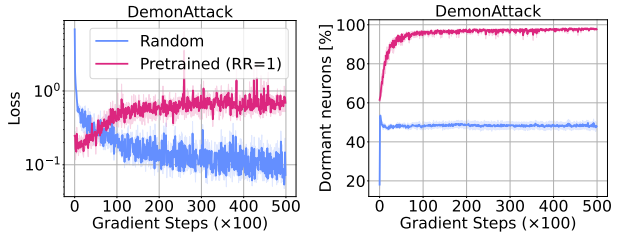

Figure 8. A pretrained network that exhibits dormant neurons has less ability than a randomly initialized network to fit a fixed target. Results are averaged over 5 seeds.

Dormant neurons make learning new tasks more difficult. We directly examine the effect of dormant neurons on an RL network's ability to learn new tasks. To do so, we train a DQN agent with a replay ratio of 1 (this agent exhibits a high level of dormant neurons as observed in Figure 7). Next we fine-tune this network by distilling it towards a well performing DQN agent's network, using a traditional regression loss and compare this with a randomly initialized agent trained using the same loss. In Figure 8 we see that the pre-trained network, which starts with a high level of dormant neurons, shows degrading performance throughout training; in contrast, the randomly initialized baseline is able to continuously improve. Further, while the baseline network maintains a stable level of dormant neurons, the number of dormant neurons in the pre-trained

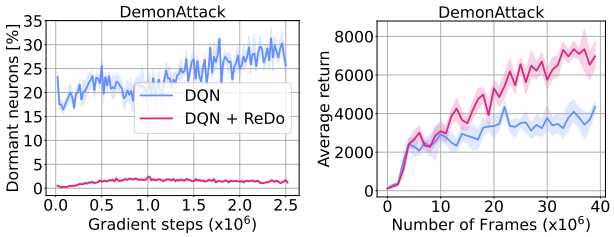

Figure 9. Evaluation of *ReDo*'s effectiveness (with $\tau = 0.025$) in reducing dormant neurons (left) and improving performance (right) on DQN (with $RR = 0.25$).

network continues to increase throughout training.

4. *Re*cycling *Do*rmant Neurons (*ReDo*)

Our analyses in Section 3, which demonstrates the existence of the dormant neuron phenomenon in online RL, suggests these dormant neurons may have a role to play in the diminished expressivity highlighted by Kumar et al. (2021a) and Lyle et al. (2021). To account for this, we propose to **re**cycle **do**rmant neurons periodically during training (*ReDo*).

The main idea of *ReDo*, outlined in Algorithm 1, is rather simple: during regular training, periodically check in all layers whether any neurons are τ-dormant; for these, reinitialize their incoming weights and zero out the outgoing weights. The incoming weights are initialized using the original weight distribution. Note that if τ is 0, we are effectively leaving the network's output unchanged; if τ is small, the output of the network is only slightly changed.

Figure 9 showcases the effectiveness of *ReDo* in dramatically reducing the number of dormant neurons, which also results in improved agent performance. Before diving into a deeper empirical evaluation of our method in Section 5, we discuss some algorithmic alternatives we considered when designing *ReDo*.

Alternate recycling strategies. We considered other recycling strategies, such as scaling the incoming connections using the mean of the norm of non-dormant neurons. However, this strategy performed similarly to using initial weight distribution. Similarly, alternative initialization strategies like initializing outgoing connections randomly resulted in similar or worse returns. Results of these investigations are shared in Appendix C.2.

Are ReLUs to blame? RL networks typically use ReLU activations, which can saturate at zero outputs, and hence zero gradients. To investigate whether the issue is specific to the use of ReLUs, in Appendix C.1 we measured the number of dormant neurons and resulting performance when using a different activation function. We observed that there is a mild decrease in the number of dormant neurons, but the phenomenon is still present.

Algorithm 1 *ReDo*

Input: Network parameters θ, threshold τ, training steps T, frequency F
for $t = 1$ to **to** T **do**
  Update θ with regular RL loss
  if $t \mod F == 0$ **then**
    for each neuron i **do**
      if $s_i^\ell \leq \tau$ **then**
        Reinitialize input weights of neuron i
        Set outgoing weights of neuron i to 0
      end if
    end for
  end if
end for

5. Empirical Evaluations

Agents, architectures, and environments. We evaluate DQN on 17 games from the Arcade Learning Environment (Bellemare et al., 2013) (as used in (Kumar et al., 2021a;b) to study the loss of network expressivity). We study two different architectures: the default CNN used by Mnih et al. (2015), and the ResNet architecture used by the IMPALA agent (Espeholt et al., 2018).

Additionally, we evaluate DrQ(ϵ) (Yarats et al., 2021; Agarwal et al., 2021) on the 26 games used in the Atari 100K benchmark (Kaiser et al., 2019), and SAC (Haarnoja et al., 2018) on four MuJoCo environments (Todorov et al., 2012).

Implementation details. All our experiments and implementations were conducted using the Dopamine framework (Castro et al., 2018)[2]. For agents trained with *ReDo*, we use a threshold of $\tau = 0.1$, unless otherwise noted, as we found this gave a better performance than using a threshold of 0 or 0.025. When aggregating results across multiple games, we report the Interquantile Mean (IQM), recommended by Agarwal et al. (2021) as a more statistically reliable alternative to median or mean, using 5 independent seeds for each DQN experiment, 10 for the DrQ and SAC experiments, and reporting 95% stratified bootstrap confidence intervals.

5.1. Consequences for Sample Efficiency

Motivated by our finding that higher replay ratios exacerbate dormant neurons and lead to poor performance (Figure 7), we investigate whether *ReDo* can help mitigate these. To

[2]Code is available at https://github.com/google/dopamine/tree/master/dopamine/labs/redo

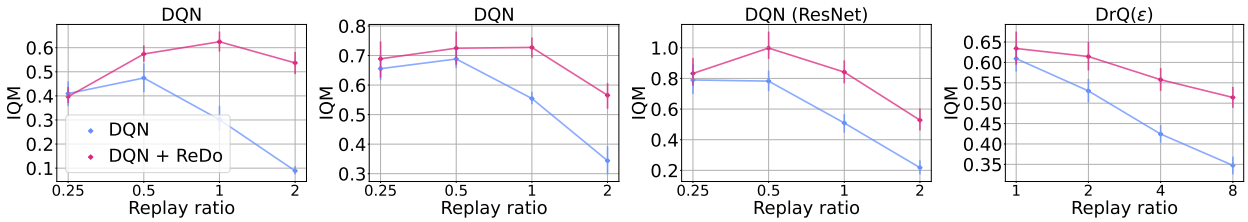

Figure 10. Evaluating the effect of increased replay ratio with and without *ReDo*. From left to right: DQN with default settings, DQN with n-step of 3, DQN with the ResNet architecture, and DrQ(ϵ). We report results using 5 seeds, while DrQ(ϵ) use 10 seeds; error bars report 95% confidence intervals.

do so, we report the IQM for four replay ratio values: 0.25 (default for DQN), 0.5, 1, and 2 when training with and without *ReDo*. Since increasing the replay ratio increases the training time and cost, we train DQN for 10M frames, as opposed to the regular 200M. As the leftmost plot in Figure 10 demonstrates, *ReDo* is able to avoid the performance collapse when increasing replay ratios, and even to benefit from the higher replay ratios when trained with *ReDo*.

Impact on multi-step learning. In the center-left plot of Figure 10 we added n-step returns with a value of $n = 3$ (Sutton & Barto, 2018). While this change results in a general improvement in DQN's performance, it still suffers from performance collapse with higher replay ratios; *ReDo* mitigates this and improves performance across all values.

Varying architectures. To evaluate *ReDo*'s impact on different network architectures, in the center-right plot of Figure 10 we replace the default CNN architecture used by DQN with the ResNet architecture used by the IMPALA agent (Espeholt et al., 2018). We see a similar trend: *ReDo* enables the agent to make better use of higher replay ratios, resulting in improved performance.

Varying agents. We evaluate on a sample-efficient value-based agent DrQ(ϵ) (Yarats et al., 2021; Agarwal et al., 2021)) on the Atari 100K benchmark in the rightmost plot of Figure 10. In this setting, we train for 400K steps, where we can see the effect of dormant neurons on performance, and study the following replay ratio values: 1 (default), 2, 4, 8. Once again, we observe *ReDo*'s effectiveness in improving performance at higher replay ratios.

In the rest of this section, we do further analyses to understand the improved performance of *ReDo* and how it fares against related methods. We perform this study on a DQN agent trained with a replay ratio of 1 using the default CNN architecture.

5.2. Learning Rate Scaling

An important point to consider is that the default learning rate may not be optimal for higher replay ratios. Intuitively,

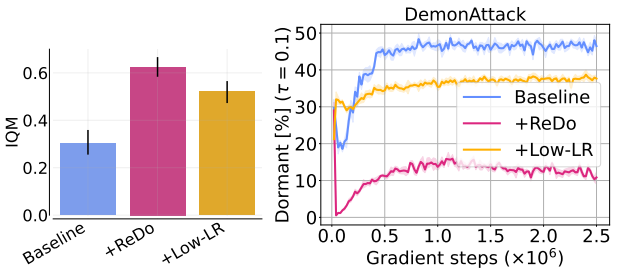

Figure 11. Effect of reduced learning rate in high replay ratio setting. Scaling learning rate helps, but does not solve the dormant neuron problem. Aggregated results across 17 games (left) and the percentage of dormant neurons during training on DemonAttack (right).

performing more gradient updates would suggest a *reduced* learning rate would be more beneficial. To evaluate this, we decrease the learning rate by a factor of four when using a replay ratio of 1 (four times the default value). Figure 11 confirms that a lower learning rate reduces the number of dormant neurons and improves performance. However, percentage of dormant neurons is still high and using *ReDo* with a high replay ratio and the default learning rate obtains the best performance.

5.3. Is Over-parameterization Enough?

Lyle et al. (2021) and Fu et al. (2019) suggest sufficiently over-parameterized networks can fit new targets over time; this raises the question of whether over-parameterization can help address the dormant neuron phenomenon. To investigate this, we increase the size of the DQN network by doubling and quadrupling the width of its layers (both the convolutional and fully connected). The left plot in Figure 12 shows that larger networks have at most a mild positive effect on the performance of DQN, and the resulting performance is still far inferior to that obtained when using *ReDo* with the default width. Furthermore, training with *ReDo* seems to improve as the network size increases, suggesting that the agent is able to better exploit network parameters, compared to when training without *ReDo*.

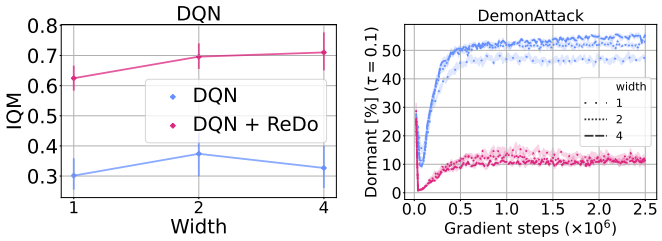

Figure 12. Performance of DQN trained with $RR = 1$ using different network width. Increasing the width of the network slightly improves the performance. Yet, the performance gain does not reach the gain obtained by *ReDo*. *ReDo* improves the performance across different network sizes.

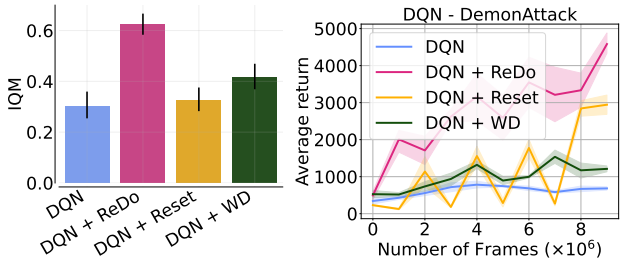

Figure 13. Comparison of the performance for *ReDo* and two different regularization methods (Reset (Nikishin et al., 2022) and weight decay (WD)) when integrated with training DQN agents. Aggregated results across 17 games (left) and the learning curve on DemonAttack (right).

An interesting finding in the right plot in Figure 12 is that the percentage of dormant neurons is similar across the varying widths. As expected, the use of *ReDo* dramatically reduces this number for all values. This finding is somewhat at odds with that from Sankararaman et al. (2020). They demonstrated that, in supervised learning settings, increasing the width decreases the gradient confusion and leads to faster training. If this observation would also hold in RL, we would expect to see the percentage of dormant neurons decrease in larger models.

5.4. Comparison with Related Methods

Nikishin et al. (2022) also observed performance collapse when increasing the replay ratio, but attributed this to overfitting to early samples (an effect they refer to as the "primacy bias"). To mitigate this, they proposed periodically resetting the network, which can be seen as a form of regularization. We compare the performance of *ReDo* against theirs, which periodically resets only the penultimate layer for Atari environments. Additionally, we compare to adding weight decay, as this is a simpler, but related, form of regularization. It is worth highlighting that Nikishin et al. (2022) also found high values of replay ratio to be more amenable to their method. As Figure 13 illustrates, weight decay is comparable to periodic resets, but *ReDo* is superior to both.

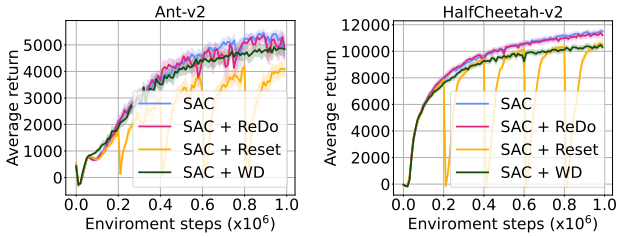

Figure 14. Comparison of the performance of SAC agents with *ReDo* and two different regularization methods (Reset (Nikishin et al., 2022) and weight decay (WD)). See Figure 20 for other environments.

We continue our comparison with resets and weight decay on two MuJoCo environments with the SAC agent (Haarnoja et al., 2018). As Figure 14 shows, *ReDo* is the only method that does not suffer a performance degradation. The results on other environments can be seen in Appendix B.

5.5. Neuron Selection Strategies

Finally, we compare our strategy for selecting the neurons that will be recycled (Section 3) against two alternatives: (1) **Random:** neurons are selected randomly, and (2) **Inverse *ReDo*:** neurons with the *highest* scores according to Equation 1 are selected. To ensure a fair comparison, the number of recycled neurons is a fixed percentage for all methods, occurring every 1000 steps. The percentage of neurons to recycle follows a cosine schedule starting at 0.1 and ending at 0. As Figure 15 shows, recycling active or random neurons hinders learning and causes performance collapse.

6. Related Work

Function approximators in RL. The use of over-parameterized neural networks as function approximators was instrumental to some of the successes in RL, such as achieving superhuman performance on Atari 2600 games (Mnih et al., 2015) and continuous control (Lillicrap et al., 2016). Recent works observe a change in the network's capacity over the course of training, which affects the agent's performance. Kumar et al. (2021a;b) show that the expressivity of the network decreases gradually due to bootstrapping. Gulcehre et al. (2022) investigate the sources of expressivity loss in offline RL and observe that under-paramterization emerges with prolonged training. Lyle et al. (2021) demonstrate that RL agents lose their ability to fit new target functions over time, due to the non-stationary in the targets. Similar observations have been found, referred to as plasticity loss, in the continual learning setting

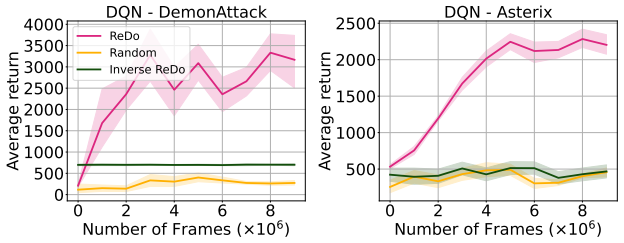

Figure 15. Comparison of different strategies for selecting the neurons that will be recycled. Recycling neurons with the highest score (Inverse *ReDo*) or random neurons causes performance collapse.

where the data distribution is changing over time (Berariu et al., 2021; Dohare et al., 2021). These observations call for better understanding how RL learning dynamics affect the capacity of their neural networks.

There is a recent line of work investigating network topologies by using sparse neural networks in online (Graesser et al., 2022; Sokar et al., 2022; Tan et al., 2022) and offline RL (Arnob et al., 2021). They show up to 90% of the network's weights can be removed with minimal loss in performance. This suggests that RL agents are not using the capacity of the network to its full potential.

Generalization in RL. RL agents are prone to overfitting, whether it is to training environments, reducing their ability to generalize to unseen environments (Kirk et al., 2021), or to early training samples, which degrades later training performance (Fu et al., 2019; Nikishin et al., 2022). Techniques such as regularization (Hiraoka et al., 2021; Wang et al., 2020), ensembles (Chen et al., 2020), or data augmentation (Fan et al., 2021; Janner et al., 2019; Hansen et al., 2021) have been adopted to account for overfitting.

Another line of work addresses generalization via *re-initializing* a subset or all of the weights of a neural network during training. This technique is mainly explored in supervised learning (Taha et al., 2021; Zhou et al., 2021; Alabdulmohsin et al., 2021; Zaidi et al., 2022), transfer learning (Li et al., 2020), and online learning (Ash & Adams, 2020). A few recent works have explored this for RL: Igl et al. (2020) periodically reset an agent's full network and then performs distillation from the pre-reset network. Nikishin et al. (2022) (already discussed in Figure 13) periodically resets the last layers of an agent's network. Despite its performance gains, fully resetting some or all layers can lead to the agent "forgetting" prior learned knowledge. The authors account for this by using a sufficiently large replay buffer, so as to never discard any observed experience; this, however, makes it difficult to scale to environments with more environment interactions. Further, recovering performance after each reset requires many gradient updates. Similar to our approach, Dohare et al. (2021) adapt the stochastic gradient descent by resetting the smallest utility features for *continual learning*. We compare their utility metric to the one used by *ReDo* in Appendix C.4 and observe similar or worse performance.

Neural network growing. A related research direction is to *prune* and *grow* the architecture of a neural network. On the growing front, Evci et al. (2021) and Dai et al. (2019) proposed gradient-based strategies to grow new neurons in dense and sparse networks, respectively. Yoon et al. (2018) and Wu et al. (2019) proposed methods to split existing neurons. Zhou et al. (2012) adds new neurons and merges similar features for online learning.

7. Discussion and Conclusion

In this work we identified the *dormant neuron phenomenon* whereby, during training, an RL agent's neural network exhibits an increase in the number of neurons with little-or-no activation. We demonstrated that this phenomenon is present across a variety of algorithms and domains, and provided evidence that it does result in reduced expressivity and inability to adapt to new tasks.

Interestingly, studies in neuroscience have found similar types of dormant neurons (precursors) in the adult brain of several mammalian species, including humans (Benedetti & Couillard-Despres, 2022), albeit with different dynamics. Certain brain neurons *start off* as dormant during embryonic development, and progressively awaken with age, eventually becoming mature and functionally integrated as excitatory neurons (Rotheneichner et al., 2018; Benedetti et al., 2020; Benedetti & Couillard-Despres, 2022). Contrastingly, the dormant neurons we investigate here emerge over time and exacerbate with more gradient updates.

To overcome this issue, we proposed a simple method (*ReDo*) to maintain network utilization throughout training by periodic recycling of dormant neurons. The simplicity of *ReDo* allows for easy integration with existing RL algorithms. Our experiments suggest that this can lead to improved performance. Indeed, the results in Figure 10 and 12 suggest that *ReDo* can be an important component in being able to successfully scale RL networks in a sample-efficient manner.

Limitations and future work. Although the simple approach of recycling neurons we introduced yielded good results, it is possible that better approaches exist. For example, *ReDo* reduces dormant neurons significantly but it doesn't completely eliminate them. Further research on initialization and optimization of the recycled capacity can address this and lead to improved performance. Additionally, the dormancy threshold is a hyperparameter that requires tuning; having an adaptive threshold over the course

of training could improve performance even further. Finally, further investigation into the relationship between the task's complexity, network capacity, and the dormant neuron phenomenon would provide a more comprehensive understanding.

Similarly to the findings of Graesser et al. (2022), this work suggests there are important gains to be had by investigating the network architectures and topologies used for deep reinforcement learning. Moreover, the observed network's behavior during training (i.e. the change in the network capacity utilization), which differs from supervised learning, indicates a need to explore optimization techniques specific to reinforcement learning due to its unique learning dynamics.

Societal impact. Although the work presented here is mostly of an academic nature, it aids in the development of more capable autonomous agents. While our contributions do not directly contribute to any negative societal impacts, we urge the community to consider these when building on our research.

Acknowledgements

We would like to thank Max Schwarzer, Karolina Dziugaite, Marc G. Bellemare, Johan S. Obando-Ceron, Laura Graesser, Sara Hooker and Evgenii Nikishin, as well as the rest of the Brain Montreal team for their feedback on this work. We would also like to thank the Python community (Van Rossum & Drake Jr, 1995; Oliphant, 2007) for developing tools that enabled this work, including NumPy (Harris et al., 2020), Matplotlib (Hunter, 2007) and JAX (Bradbury et al., 2018).

References

Agarwal, R., Schuurmans, D., and Norouzi, M. An optimistic perspective on offline reinforcement learning. In *International Conference on Machine Learning*, pp. 104–114. PMLR, 2020.

Agarwal, R., Schwarzer, M., Castro, P. S., Courville, A. C., and Bellemare, M. Deep reinforcement learning at the edge of the statistical precipice. *Advances in neural information processing systems*, 34:29304–29320, 2021.

Alabdulmohsin, I., Maennel, H., and Keysers, D. The impact of reinitialization on generalization in convolutional neural networks. *arXiv preprint arXiv:2109.00267*, 2021.

Araújo, J. G. M., Ceron, J. S. O., and Castro, P. S. Lifting the veil on hyper-parameters for value-based deep reinforcement learning. In *Deep RL Workshop NeurIPS 2021*, 2021. URL https://openreview.net/forum?id=Ws4v7nSqqb.

Arnob, S. Y., Ohib, R., Plis, S., and Precup, D. Single-shot pruning for offline reinforcement learning. *arXiv preprint arXiv:2112.15579*, 2021.

Ash, J. and Adams, R. P. On warm-starting neural network training. *Advances in Neural Information Processing Systems*, 33:3884–3894, 2020.

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

Bellemare, M. G., Candido, S., Castro, P. S., Gong, J., Machado, M. C., Moitra, S., Ponda, S. S., and Wang, Z. Autonomous navigation of stratospheric balloons using reinforcement learning. *Nature*, 588(7836):77–82, 2020.

Benedetti, B. and Couillard-Despres, S. Why would the brain need dormant neuronal precursors? *Frontiers in Neuroscience*, 16, 2022.

Benedetti, B., Dannehl, D., König, R., Coviello, S., Kreutzer, C., Zaunmair, P., Jakubecova, D., Weiger, T. M., Aigner, L., Nacher, J., et al. Functional integration of neuronal precursors in the adult murine piriform cortex. *Cerebral cortex*, 30(3):1499–1515, 2020.

Bengio, E., Pineau, J., and Precup, D. Interference and generalization in temporal difference learning. In *International Conference on Machine Learning*, pp. 767–777. PMLR, 2020.

Berariu, T., Czarnecki, W., De, S., Bornschein, J., Smith, S. L., Pascanu, R., and Clopath, C. A study on the plasticity of neural networks. *CoRR*, abs/2106.00042, 2021. URL https://arxiv.org/abs/2106.00042.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., et al. Jax: composable transformations of python+ numpy programs. 2018.

Castro, P. S., Moitra, S., Gelada, C., Kumar, S., and Bellemare, M. G. Dopamine: A Research Framework for Deep Reinforcement Learning. 2018. URL http://arxiv.org/abs/1812.06110.

Chen, X., Wang, C., Zhou, Z., and Ross, K. W. Randomized ensembled double q-learning: Learning fast without a model. In *International Conference on Learning Representations*, 2020.

Dai, X., Yin, H., and Jha, N. K. Nest: A neural network synthesis tool based on a grow-and-prune paradigm. *IEEE Transactions on Computers*, 68(10):1487–1497, 2019.

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

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

Evci, U., van Merrienboer, B., Unterthiner, T., Pedregosa, F., and Vladymyrov, M. Gradmax: Growing neural networks using gradient information. In *International Conference on Learning Representations*, 2021.

Fan, L., Wang, G., Huang, D.-A., Yu, Z., Fei-Fei, L., Zhu, Y., and Anandkumar, A. Secant: Self-expert cloning for zero-shot generalization of visual policies. In *International Conference on Machine Learning*, pp. 3088–3099. PMLR, 2021.

Fedus, W., Ramachandran, P., Agarwal, R., Bengio, Y., Larochelle, H., Rowland, M., and Dabney, W. Revisiting fundamentals of experience replay. In *International Conference on Machine Learning*, pp. 3061–3071. PMLR, 2020.

Fu, J., Kumar, A., Soh, M., and Levine, S. Diagnosing bottlenecks in deep q-learning algorithms. In *International Conference on Machine Learning*, pp. 2021–2030. PMLR, 2019.

Graesser, L., Evci, U., Elsen, E., and Castro, P. S. The state of sparse training in deep reinforcement learning. In *International Conference on Machine Learning*, pp. 7766–7792. PMLR, 2022.

Guadarrama, S., Korattikara, A., Ramirez, O., Castro, P., Holly, E., Fishman, S., Wang, K., Gonina, E., Wu, N., Kokiopoulou, E., Sbaiz, L., Smith, J., Bartók, G., Berent, J., Harris, C., Vanhoucke, V., and Brevdo, E. TF-Agents: A library for reinforcement learning in tensorflow. https://github.com/tensorflow/agents, 2018. URL https://github.com/tensorflow/agents. [Online; accessed 25-June-2019].

Gulcehre, C., Srinivasan, S., Sygnowski, J., Ostrovski, G., Farajtabar, M., Hoffman, M., Pascanu, R., and Doucet, A. An empirical study of implicit regularization in deep offline rl. *arXiv preprint arXiv:2207.02099*, 2022.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pp. 1861–1870. PMLR, 2018.

Hansen, N., Su, H., and Wang, X. Stabilizing deep q-learning with convnets and vision transformers under data augmentation. In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 3680–3693. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper/2021/file/1e0f65eb20acbfb27ee05ddc000b50ec-Paper.pdf.

Harris, C. R., Millman, K. J., Van Der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., et al. Array programming with numpy. *Nature*, 585(7825):357–362, 2020.

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

Hestness, J., Narang, S., Ardalani, N., Diamos, G., Jun, H., Kianinejad, H., Patwary, M., Ali, M., Yang, Y., and Zhou, Y. Deep learning scaling is predictable, empirically. *arXiv preprint arXiv:1712.00409*, 2017.

Hiraoka, T., Imagawa, T., Hashimoto, T., Onishi, T., and Tsuruoka, Y. Dropout q-functions for doubly efficient reinforcement learning. In *International Conference on Learning Representations*, 2021.

Hunter, J. D. Matplotlib: A 2d graphics environment. *Computing in science & engineering*, 9(03):90–95, 2007.

Igl, M., Farquhar, G., Luketina, J., Boehmer, W., and Whiteson, S. Transient non-stationarity and generalisation in deep reinforcement learning. In *International Conference on Learning Representations*, 2020.

Janner, M., Fu, J., Zhang, M., and Levine, S. When to trust your model: Model-based policy optimization. *Advances in Neural Information Processing Systems*, 32, 2019.

Kaiser, Ł., Babaeizadeh, M., Miłos, P., Osiński, B., Campbell, R. H., Czechowski, K., Erhan, D., Finn, C., Kozakowski, P., Levine, S., et al. Model based reinforcement learning for atari. In *International Conference on Learning Representations*, 2019.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In Bengio, Y. and LeCun, Y. (eds.),

3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings, 2015. URL http://arxiv.org/abs/1412.6980.

Kirk, R., Zhang, A., Grefenstette, E., and Rocktäschel, T. A survey of generalisation in deep reinforcement learning. *arXiv preprint arXiv:2111.09794*, 2021.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. 2009.

Kumar, A., Agarwal, R., Ghosh, D., and Levine, S. Implicit under-parameterization inhibits data-efficient deep reinforcement learning. In *International Conference on Learning Representations*, 2021a.

Kumar, A., Agarwal, R., Ma, T., Courville, A., Tucker, G., and Levine, S. Dr3: Value-based deep reinforcement learning requires explicit regularization. In *International Conference on Learning Representations*, 2021b.

Li, X., Xiong, H., An, H., Xu, C.-Z., and Dou, D. Rifle: Backpropagation in depth for deep transfer learning through re-initializing the fully-connected layer. In *International Conference on Machine Learning*, pp. 6010–6019. PMLR, 2020.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. In *ICLR (Poster)*, 2016.

Lin, L.-J. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3):293–321, 1992.

Lyle, C., Rowland, M., and Dabney, W. Understanding and preventing capacity loss in reinforcement learning. In *International Conference on Learning Representations*, 2021.

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

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

Oliphant, T. E. Python for scientific computing. *Computing in Science & Engineering*, 9(3):10–20, 2007. doi: 10.1109/MCSE.2007.58.

Puterman, M. L. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

Rotheneichner, P., Belles, M., Benedetti, B., König, R., Dannehl, D., Kreutzer, C., Zaunmair, P., Engelhardt, M., Aigner, L., Nacher, J., et al. Cellular plasticity in the adult murine piriform cortex: continuous maturation of dormant precursors into excitatory neurons. *Cerebral Cortex*, 28(7):2610–2621, 2018.

Sankararaman, K. A., De, S., Xu, Z., Huang, W. R., and Goldstein, T. The impact of neural network overparameterization on gradient confusion and stochastic gradient descent. In *International conference on machine learning*, pp. 8469–8479. PMLR, 2020.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

Sokar, G., Mocanu, E., Mocanu, D. C., Pechenizkiy, M., and Stone, P. Dynamic sparse training for deep reinforcement learning. In *International Joint Conference on Artificial Intelligence*, 2022.

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

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

Taha, A., Shrivastava, A., and Davis, L. S. Knowledge evolution in neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12843–12852, 2021.

Tan, Y., Hu, P., Pan, L., and Huang, L. Rlx2: Training a sparse deep reinforcement learning model from scratch. *arXiv preprint arXiv:2205.15043*, 2022.

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

van Hasselt, H., Doron, Y., Strub, F., Hessel, M., Sonnerat, N., and Modayil, J. Deep reinforcement learning and the deadly triad. *CoRR*, abs/1812.02648, 2018. URL http://arxiv.org/abs/1812.02648.

Van Hasselt, H. P., Hessel, M., and Aslanides, J. When to use parametric models in reinforcement learning? *Advances in Neural Information Processing Systems*, 32, 2019.

Van Rossum, G. and Drake Jr, F. L. *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

Wang, K., Kang, B., Shao, J., and Feng, J. Improving generalization in reinforcement learning with mixture regularization. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 7968–7978. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/5a751d6a0b6ef05cfe51b86e5d1458e6-Paper.pdf.

Wu, L., Wang, D., and Liu, Q. Splitting steepest descent for growing neural architectures. *Advances in neural information processing systems*, 32, 2019.

Yarats, D., Kostrikov, I., and Fergus, R. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=GY6-6sTvGaf.

Yoon, J., Yang, E., Lee, J., and Hwang, S. J. Lifelong learning with dynamically expandable networks. In *International Conference on Learning Representations*, 2018.

Zaidi, S., Berariu, T., Kim, H., Bornschein, J., Clopath, C., Teh, Y. W., and Pascanu, R. When does re-initialization work? *arXiv preprint arXiv:2206.10011*, 2022.

Zhai, X., Kolesnikov, A., Houlsby, N., and Beyer, L. Scaling vision transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12104–12113, 2022.

Zhou, G., Sohn, K., and Lee, H. Online incremental feature learning with denoising autoencoders. In *Artificial intelligence and statistics*, pp. 1453–1461. PMLR, 2012.

Zhou, H., Vani, A., Larochelle, H., and Courville, A. Fortuitous forgetting in connectionist networks. In *International Conference on Learning Representations*, 2021.

Table 1. Common Hyper-parameters for DQN and DrQ(ϵ).

Parameter	Value
Optimizer	Adam (Kingma & Ba, 2015)
Optimizer: ϵ	1.5×10^{-4}
Training ϵ	0.01
Evaluation ϵ	0.001
Discount factor	0.99
Replay buffer size	10^6
Minibatch size	32
Q network: channels	32, 64, 64
Q-network: filter size	8×8, 4×4, 3×3
Q-network: stride	4, 2, 1
Q-network: hidden units	512
Recycling period	1000
τ-Dormant	0.025 for default setting, 0.1 otherwise
Minibatch size for estimating neurons score	64

Table 2. Hyper-parameters for DQN.

Parameter	Value
Optimizer: Learning rate	6.25×10^{-5}
Initial collect steps	20000
n-step	1
Training iterations	Default setting: 40, otherwise: 10
Training environment steps per iteration	250K
(Updates per environment step, Target network update period)	(0.25, 8000)
	(0.5, 4000)
	(1, 2000)
	(2, 1000)

Author Contributions

- Ghada: Led the work, worked on project direction and plan, participated in discussions, wrote most of the code, ran most of the experiments, led the writing, and wrote the draft of the paper.
- Rishabh: Advised on project direction and participated in project discussions, ran an offline RL experiment, worked on the plots and helped with paper writing.
- Pablo: Worked on project direction and plan, participated in discussions throughout the project, helped with reviewing code, ran some experiments, worked substantially on paper writing, supervised Ghada.
- Utku: Proposed project direction and the initial project plan. Reviewed and open-sourced the code. Ran part of the experiments, worked on the plots and helped with paper writing, supervised Ghada.

A. Experimental Details

Discrete control tasks. We evaluate DQN (Mnih et al., 2015) on 17 games from the Arcade Learning Environment (Bellemare et al., 2013): Asterix, Demon Attack, Seaquest, Wizard of Wor, Bream Reader, Road Runner, James Bond, Qbert, Breakout, Enduro, Space Invaders, Pong, Zaxxon, Yars' Revenge, Ms. Pacman, Double Dunk, Ice Hockey. This set is used by previous works (Kumar et al., 2021a;b) to study the *implicit under-parameterization* phenomenon in offline RL. For hyper-parameter tuning, we used five games (Asterix, Demon Attack, Seaquest, Breakout, Beam Rider). We evaluate DrQ(ϵ) on the 26 games of Atari 100K (Kaiser et al., 2019). We used the best hyper-parameters found for DQN in training DrQ(ϵ).

Table 3. Hyper-parameters for DrQ(ϵ).

Parameter	Value
Optimizer: Learning rate	1×10^{-4}
Initial collect steps	1600
n-step	10
Training iterations	40
Training environment steps per iteration	10K
Updates per environment step	1, 2, 4, 8

Table 4. Hyper-parameters for SAC.

Parameter	Value
Initial collect steps	10000
Discount factor	0.99
Training environment steps	10^6
Replay buffer size	10^6
Updates per environment step (Replay Ratio)	1, 2, 4, 8
Target network update period	1
target smoothing coefficient τ	0.005
Optimizer	Adam (Kingma & Ba, 2015)
Optimizer: Learning rate	3×10^{-4}
Minibatch size	256
Actor/Critic: Hidden layers	2
Actor/Critic: Hidden units	256
Recycling period	200000
τ-Dormant	0
Minibatch size for estimating neurons score	256

Continuous control tasks. We evaluate SAC (Haarnoja et al., 2018) on four environments from MuJoCo suite (Todorov et al., 2012): HalfCheetah-v2, Hopper-v2, Walker2d-v2, Ant-v2.

Code. For discrete control tasks, we build on the implementation of DQN and DrQ provided in Dopamine (Castro et al., 2018), including the architectures used for agents. The hyper-parameters are provided in Tables 1, 2, and 3. For continuous control, we build on the SAC implementation in TF-Agents (Guadarrama et al., 2018) and the codebase of (Graesser et al., 2022). The hyper-parameters are provided in Table 4.

Evaluation. We follow the recommendation from (Agarwal et al., 2021) to report reliable aggregated results across games using the interquartile mean (IQM). IQM is the calculated mean after discarding the bottom and top 25% of normalized scores aggregated from multiple runs and games.

Baselines. For weight decay, we searched over the grid $[10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}]$. The best found value is 10^{-5}. For reset (Nikishin et al., 2022), we consider re-initializing the last layer for Atari games (same as the original paper). They use a reset period of 2×10^4 in for Atari 100k (Kaiser et al., 2019), which corresponds to having 5 restarts in a training run. Since we run longer experiments, we searched over the grid $[5 \times 10^4, 1 \times 10^5, 2.5 \times 10^5, 5 \times 10^5]$ gradient steps for the reset period which corresponds to having 50, 25, 10 and 5 restarts per training (10M frames, replay ratio 1). The best found period is 1×10^5. For SAC, we reset agent's networks entirely every 2×10^5 environment steps, following the original paper.

Replay ratio. For DQN, we evaluate replay ratio values: $\{0.25$ (default), 0.5, 1, $2\}$. Following (Van Hasselt et al., 2019), we scale the target update period based on the value of the replay ratio as shown in Table 2. For DrQ(ϵ), we evaluate the values: $\{1$ (default), 2, 4, $8\}$.

Table 5. Hyperparameters for CIFAR-10.

Parameter	Value
Optimizer	SGD
Minibatch size	256
Learning rate	0.01
Momentum	0.9
Architecture:	
Layer	(channels, kernel size, stride)
Convolution	(32, 3, 1)
Convolution	(64, 3, 1)
MaxPool	(-, 2, 2)
Convolution	(64, 3, 1)
MaxPool	(-, 2, 2)
Dense	(128, -, -)

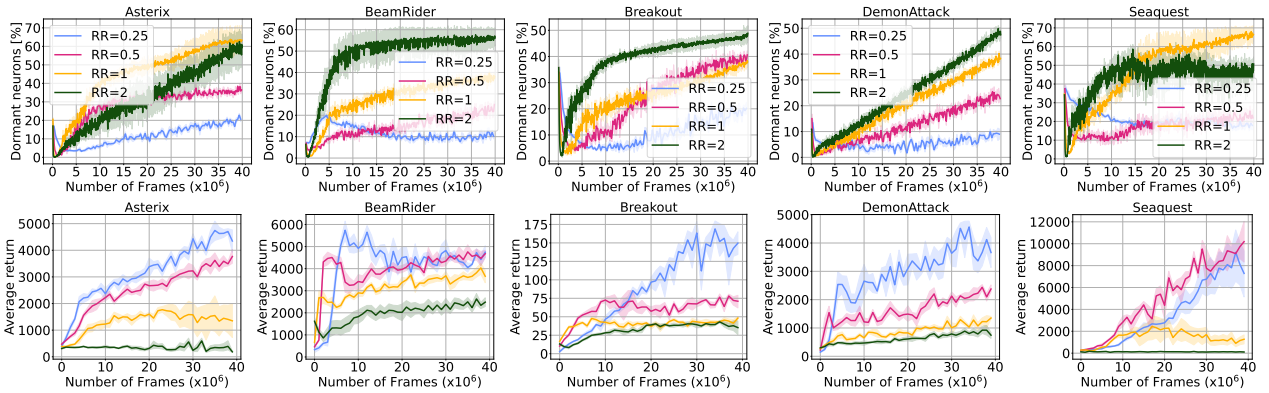

Figure 16. Effect of replay ratio in the number of dormant neurons for DQN on Atari environments (experiments presented in Figure 7).

***ReDo* hyper-parameters.** We did the hyper-parameter search for DQN trained with $RR = 1$ using the nature CNN architecture. We searched over the grids [1000, 10000, 100000] and [0, 0.01, 0.1] for the recycling period and τ-dormant, respectively. We apply the best values found to all other settings of DQN, including the ResNet architecture and DrQ(ϵ), as reported in Table 1.

Dormant neurons in supervised learning. Here we provide the experimental details of the supervised learning analysis illustrated in Section 3. We train a convolutional neural network on CIFAR-10 (Krizhevsky et al., 2009) using stochastic gradient descent and cross-entropy loss. We select 10000 samples from the dataset to reduce the computational cost. We analyze the dormant neurons in two supervised learning settings: (1) training a network with *fixed targets*, the standard single-task supervised learning, where we train a network using the inputs and labels of CIFAR-10 for 100 epochs, and (2) training a network with *non-stationary targets*, where we shuffle the labels every 20 epochs to generate new targets. Table 5 provides the details of the network architecture and training hyper-parameters.

Learning ability of networks with dormant neurons. Here we present the details of the regression experiment provided in Section 3. Inputs and targets for regression come from a DQN agent trained on DemonAttack for 40M frames with the default hyper-parameters. The pre-trained network was trained for 40M frames using a replay ratio of 1.

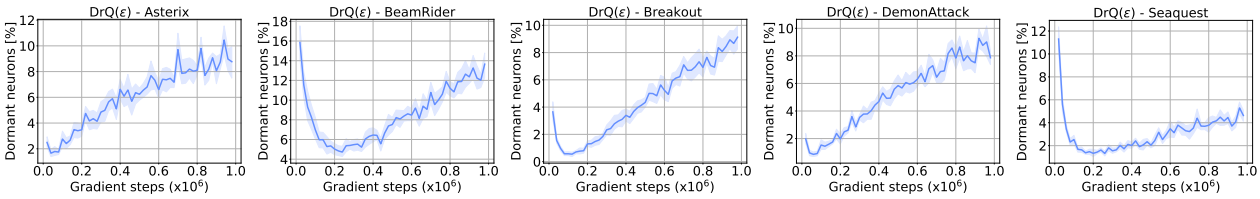

Figure 17. The dormant neuron phenomenon becomes apparent as the number of training steps increases during the training of DrQ(ϵ) with the default replay ratio on Atrai 100K.

B. The Dormant Neuron Phenomenon in Different Domains

In this appendix, we demonstrate the dormant neuron phenomenon on DrQ(ϵ) (Yarats et al., 2021) on the Atari 100K benchmark (Kaiser et al., 2019) as well as on additional games from the Arcade Learning Environment on DQN. Additionally, we show the phenomenon on continuous control tasks and analyze the role of dormant neurons in performance. We consider SAC (Haarnoja et al., 2018) trained on MuJoCo environments (Todorov et al., 2012). Same as our analyses in Section 3, we consider $\tau = 0$ to illustrate the phenomenon.

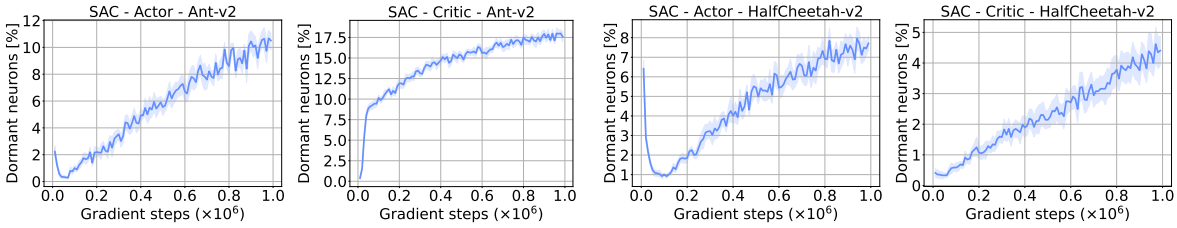

Figure 18. The number of dormant neurons increases over time during the training of SAC on MuJoCo environments.

Figure 16 shows that across games, the number of dormant neurons consistently increases with higher values for the replay ratio on DQN. The increase in dormant neurons correlates with the performance drop observed in this regime. We then investigate the phenomenon on a modern valued-based algorithm DrQ(ϵ). As we see in Figure 17, the phenomenon emerges as the number of training steps increases.

Figure 18 shows that the phenomenon is also present in continuous control tasks. An agent exhibits an increasing number of dormant neurons in the actor and critic networks during the training of SAC on MuJoco environments. To analyze the effect of these neurons on performance, we prune dormant neurons every 200K steps. Figure 19 shows that the performance is not affected by pruning these neurons; indicating their little contribution to the learning process. Next, we investigate the effect of *ReDo* and the studied baselines (Reset (Nikishin et al., 2022) and weight decay (WD)) in this domain. Figure 20 shows that *ReDo* maintains the performance of the agents while other methods cause a performance drop in most cases. We hypothesize that *ReDo* does not provide gains here as the state space is considerably low and the typically used network is sufficiently over-parameterized.

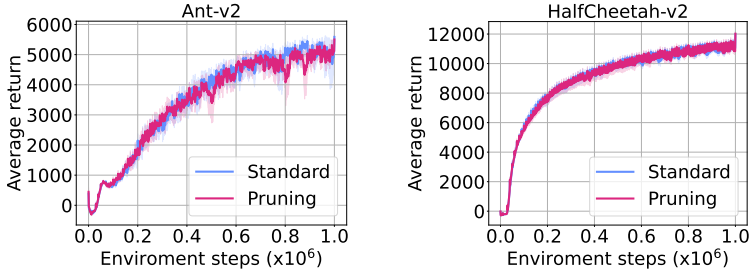

Figure 19. Pruning dormant neurons during the training of SAC on MuJoCo environments does not affect the performance.

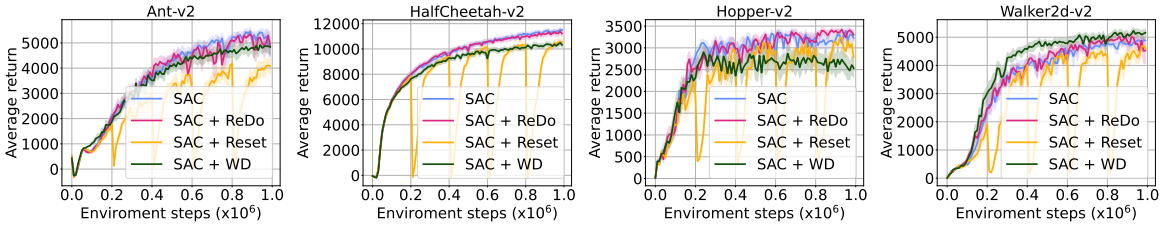

Figure 20. Comparison of the performance of SAC agents with *ReDo* and two different regularization methods.

Table 6. Performance of SAC on Ant-v2 using using half and a quarter of the width of the actor and critic networks.

Width	SAC	SAC+ReDo
0.25	2016.18 $\pm$ 102	**2114.52** $\pm$ 212
0.5	3964.04 $\pm$ 953	**4471.61** $\pm$ 648

To investigate this, we decrease the size of the actor and critic networks by halving or quartering the width of their layers. We performed these experiments on the complex environment Ant-v2 using 5 seeds. Table 6 shows the final average return in each case. We observe that when the network size is smaller, there are some gains from recycling the dormant capacity. Further analyses of the relation between task complexity and network capacity would provide a more comprehensive understanding.

C. Recycling Dormant Neurons

Here we study different strategies for recycling dormant neurons and analyze the design choices of *ReDo*. We performed these analyses on DQN agents trained with $RR = 1$ and $\tau = 0.1$ on Atari games. Furthermore, we provide some additional insights into the effect of recycling the dormant capacity on improving the sample efficiency and the expressivity of the network.

C.1. Effect of Activation Function

In this section, we attempt to understand the effect of the activation function (ReLU) used in our experiments. The ReLU activation function consists of a linear part (positive domain) with unit gradients and a constant zero part (negative domain) with zero gradients. Once the distribution of pre-activations falls completely into the negative part, it would stay there since the weights of the neuron would get zero gradients. This could be the explanation for the increased number of dormant neurons in our neural networks. If this is the case, one might expect activations with non-zero gradients on the negative side, such as leaky ReLU, to have significantly fewer dormant neurons.

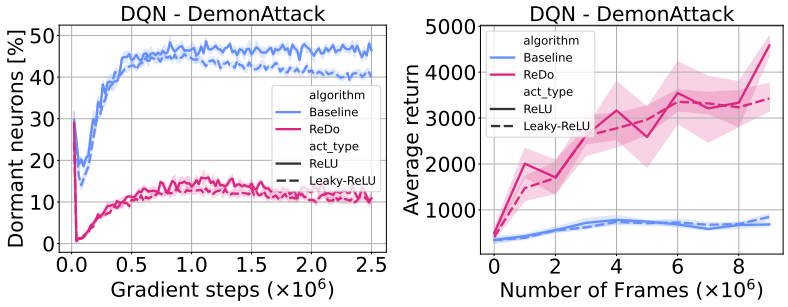

Figure 21. Training performance and dormant neuron characteristics of networks using leaky ReLU with a negative slope of 0.01 (default value) compared to original networks with ReLU.

In Figure 21, we compare networks with leaky ReLU to original networks with ReLU activation. As we can see, using leaky

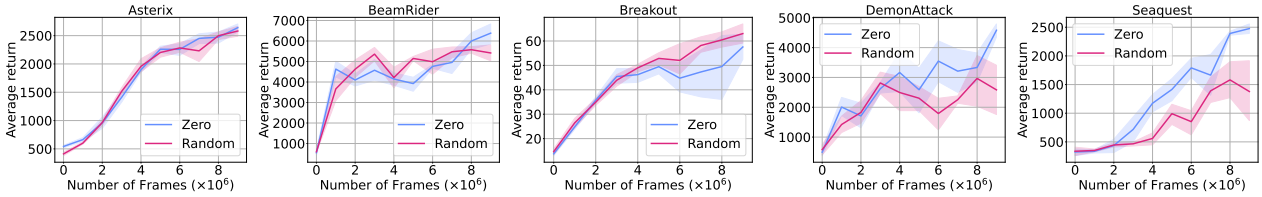

Figure 22. Comparison of performance with different strategies of reinitializing the outgoing connections of dormant neurons.

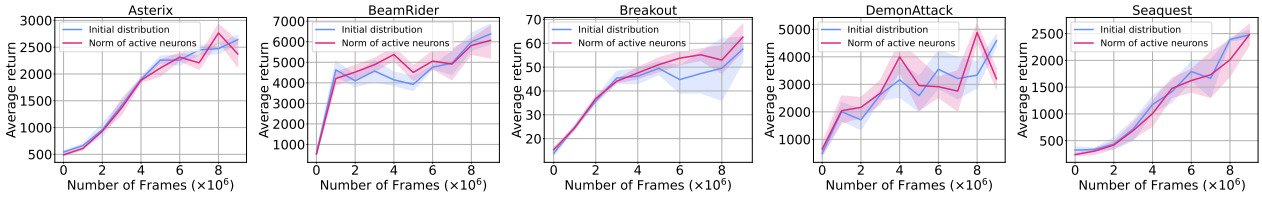

Figure 23. Comparison of performance with different strategies of reinitializing the incoming connections of dormant neurons.

ReLU slightly decreases the number of dormant neurons but does not mitigate the issue. *ReDo* overcomes the performance drop that occurs during training in the two cases.

C.2. Recycling Strategies

Outgoing connections. We investigate the effect of using random weights to reinitialize the outgoing connections of dormant neurons. We compare this strategy against the reinitialization strategy of *ReDo* (*zero weights*). Figure 22 shows the performance of DQN on five Atari games. The random initialization of the outgoing connections leads to a lower performance than the zero initialization. This is because the newly added random weights change the output of the network.

Incoming connections. Another possible strategy to reinitialize the incoming connections of dormant neurons is to scale their weights with the average norm of non-dormant neurons in the same layer. We observe that this strategy has a similar performance to the random weight initialization strategy, as shown in Figure 23.

C.3. Effect of Batch Size

The score of a neuron is calculated based on a given batch $\mathcal{D}$ of data (Section 3). Here we study the effect of the batch size in determining the percentage of dormant neurons. We study four different values: $\{32, 64, 256, 1024\}$. Figure 24 shows that the identified percentage of dormant neurons is approximately the same using different batch sizes.

C.4. Comparison with Continual Backprop

Similar to the experiments in Figure 15, we use a fixed recycling schedule to compare the activation-based metric used by *ReDo* and the utility metric proposed by Continual Backprop (Dohare et al., 2021). Results shown in Figure 25 show that both metrics achieve similar results. Note that the original Continual Backprop algorithm calculates neuron scores at every

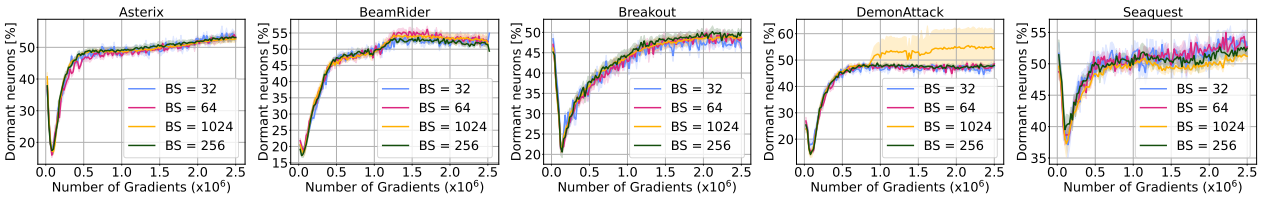

Figure 24. Effect of the batch size used to detect dormant neurons.

iteration and uses a running average to obtain a better estimate of the neuron saliency. This approach requires additional storage and computing compared to the fixed schedule used by our algorithm. Given the high dormancy threshold preferred by our method (i.e., more neurons are recycled), we expect better saliency estimates to have a limited impact on the results presented here. However, a more thorough analysis is needed to make general conclusions.

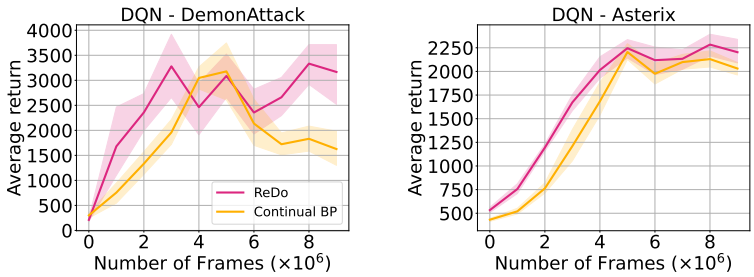

Figure 25. Comparison of different strategies for selecting the recycled neurons.

C.5. Effect of Recycling the Dormant Capacity

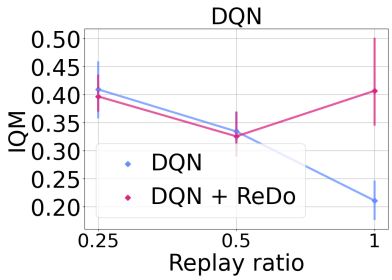

Figure 26. Comparison of agents with varying replay ratios, while keeping the number of gradient updates constant.

Improving Sample Efficiency. To examine the impact of recycling dormant neurons on enhancing the agents' sample efficiency, an alternative approach is to compare agents with varying replay ratios, while keeping the number of gradient updates constant during training. Consequently, agents with a higher replay ratio will perform fewer interactions with the environment.

We performed this analysis on DQN and the 17 Atari games. Agents with a replay ratio of 0.25 run for 10M frames, a replay ratio of 0.5 run for 5M frames, and a replay ratio of 1 run for 2.5M frames. The number of gradient steps are fixed across all agents. Figure 26 shows the aggregated results across all games. Interestingly the performance of *ReDo* with $RR = 1$ is very close to $RR = 0.25$, while significantly reducing the number of environment steps by four. On the other hand, DQN with $RR = 1$ suffers from a performance drop.

Improving Networks' expressivity. Our results in the main paper show that recycling dormant neurons improves the learning ability of agents measured by their performance. Here, we did some preliminary experiments to measure the effect of neuron recycling on the learned representations. Following (Kumar et al., 2021a), we calculate the effective rank, a measure of expressivity, of the feature learned in the penultimate layer of networks trained with and without *ReDo*. We performed this analysis on agents trained for 10M frames on DemonAttack using DQN. The results are averaged over 5 seeds. The results in Table 7 suggest recycling dormant neurons improves the expressivity, shown by the increased rank of the learned representations. Further investigation of expressivity metrics and analyses on other domains would be an exciting future direction.

Table 7. Effective rank (Kumar et al., 2021a) of the learned representations of agents trained on DemonAttack.

Agent	Effective rank
DQN	449.2 ± 5.77
DQN + *ReDo*	$\mathbf{470.8} \pm 1.16$

D. Performance Per Game

Here we share the training curves of DQN using the CNN architecture for each game in the high replay ratio regime ($RR = 1$) (Figure 27) and the default setting ($RR = 0.25$) (Figure 28). Similarly, Figure 29 and 30 show the training curves of DrQ(ϵ) for each game in the high replay ratio regime ($RR = 4$) and the default setting ($RR = 1$), respectively.

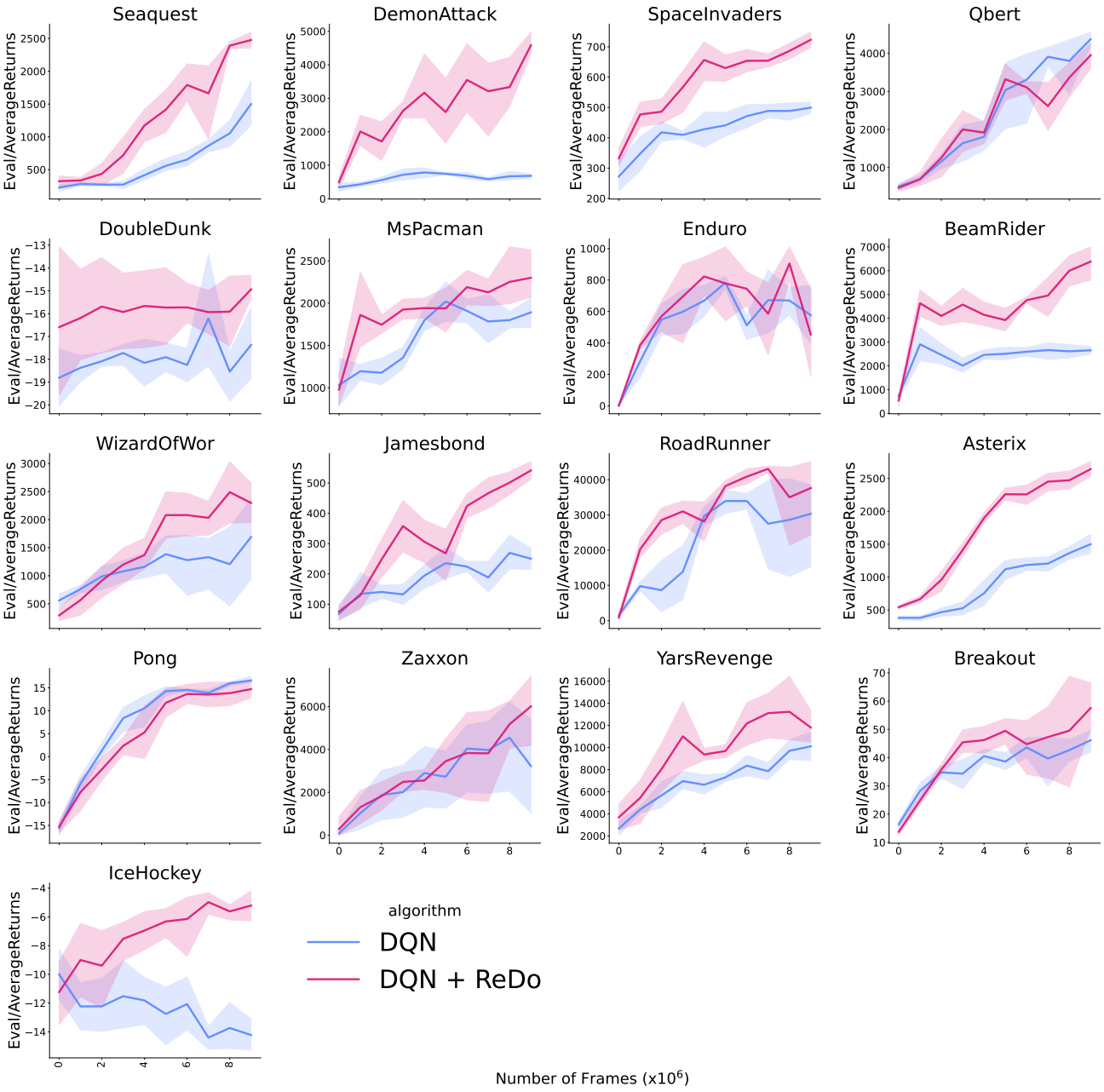

Figure 27. Training curves for DQN with the nature CNN architecture ($RR = 1$).

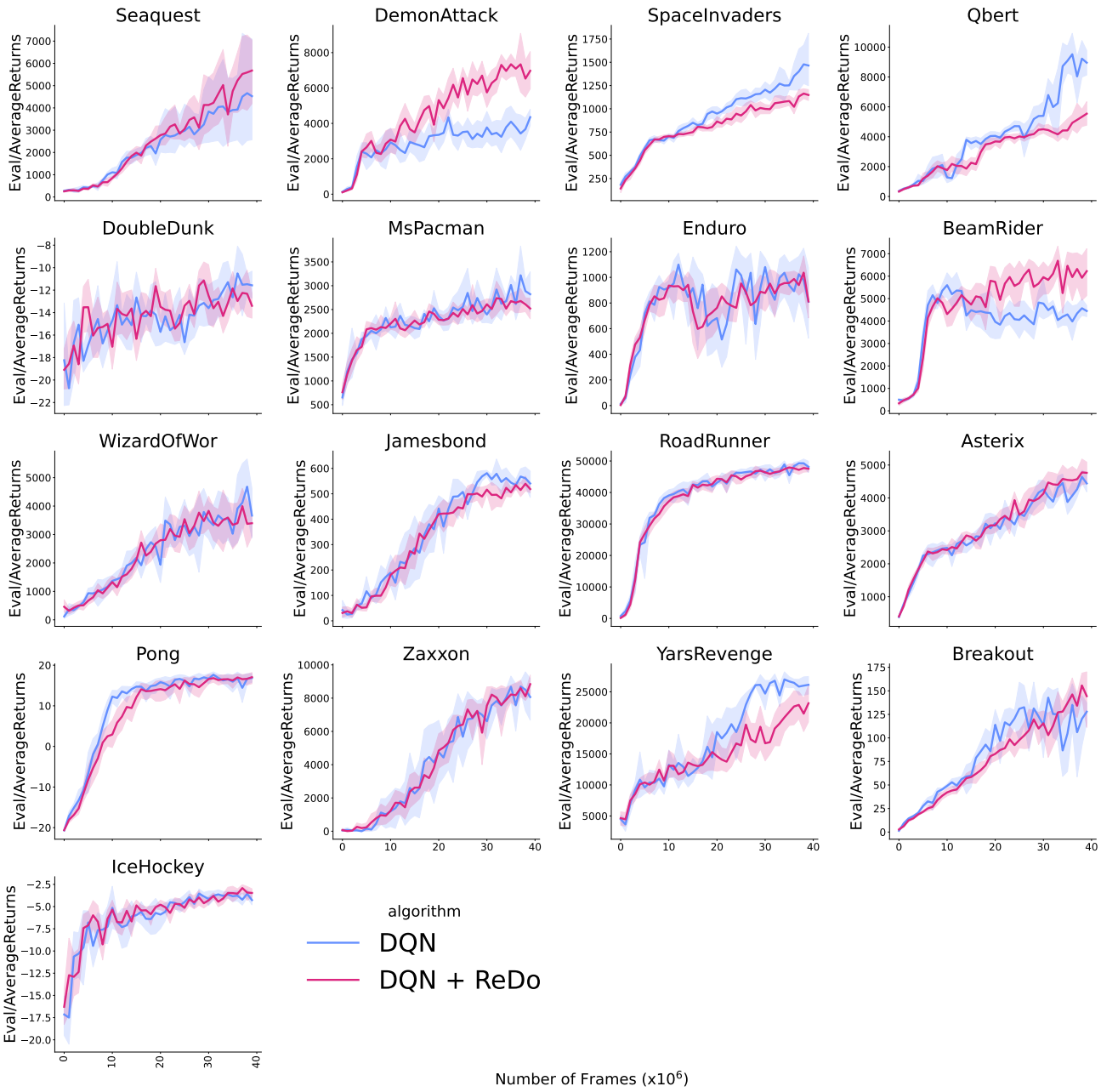

Figure 28. Training curves for DQN with the nature CNN architecture ($RR = 0.25$).

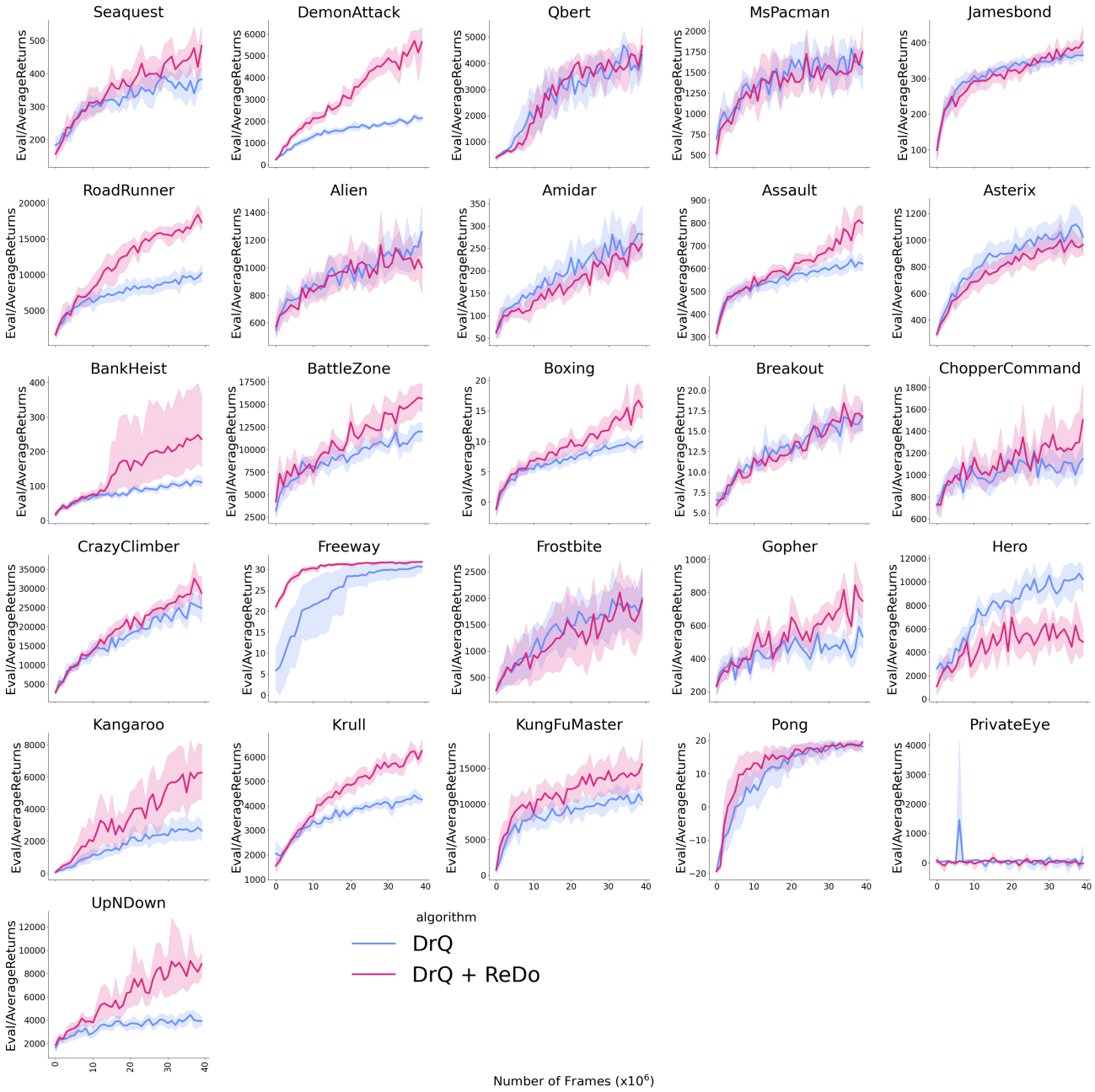

Figure 29. Training curves for DrQ(ϵ) with the nature CNN architecture ($RR = 4$).

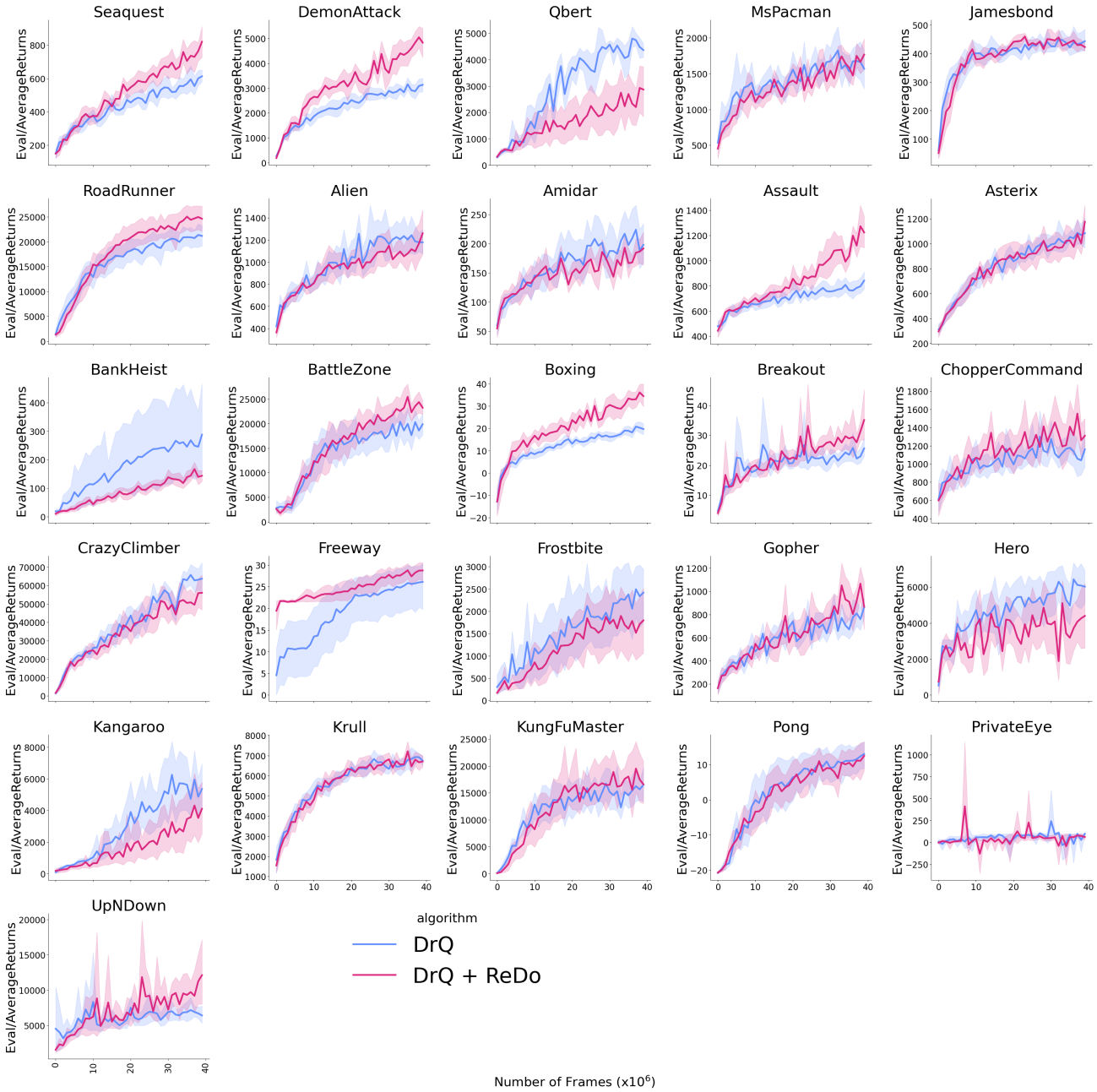

Figure 30. Training curves for DrQ(ϵ) with the nature CNN architecture ($RR = 1$).