Data, Auxiliary Losses, or Normalization Layers for Plasticity? A case study with PPO on Atari

Daniil Pyatko[*] CLAIRE, EPFL
Andrea Miele[*] CLAIRE, EPFL
Skander Moalla[†] CLAIRE, EPFL
Caglar Gulcehre[†] CLAIRE, EPFL

Abstract

Deep reinforcement learning agents often suffer from plasticity loss, in which their neural networks gradually lose the ability to incorporate new information during extended training. To understand how to mitigate this issue, we compare the impact of data interventions, auxiliary losses, and normalization layers—forming input, output, and architecture perspectives, respectively). We conduct a case study using Proximal Policy Optimization (PPO) on the Atari Learning Environment (ALE), a widely used on-policy algorithm and benchmark suite for vision-based discrete control tasks. Although many interventions have been proposed to address the inability a deep networks to continue learning due to plasticity loss, no single solution has emerged. We find that neither unpreprocessed input information nor reduced gradient noise from larger batch sizes prevents collapse. Additionally, we categorize auxiliary loss interventions based on the component being regularized and the target of the regularization. Thanks to this taxonomy, we identify unexplored solutions in the current literature and, as an illustration, derive an unstudied intervention: CHAIN-SP. We find that the best performance and training stability among the loss interventions that require tuning is achieved with churn-reduction auxiliary losses. Finally, we find that LayerNorm is best at mitigating plasticity loss among the normalization layers.

1 Introduction

Non-stationary objectives appear in various contexts, such as continual learning, where the target distribution shifts over time, but also in reinforcement learning, where the policy of the agent constantly evolves. Thus, one of the major challenges in the practical application of deep reinforcement learning algorithms is the adaptation of the underlying neural networks to non-stationary targets resulting from changes in states and rewards during learning. When a network can keep adapting to these shifts, we say it retains *plasticity*. We use *plasticity loss* to denote the regime in which adaptation stalls, observed as a drop in episodic return in RL or a rise in task loss in continual learning (Lyle et al., 2023; Dohare et al., 2024). Although recent case studies link plasticity loss to abrupt collapses in feature rank (Moalla et al., 2024; Juliani & Ash, 2024), capacity loss (Lyle et al., 2022), increase in weight norm and rank (Chung et al., 2024), and dead neurons (Sokar et al., 2023), the community has not identified any single metric to predict the collapse (Lyle et al., 2023; Lewandowski et al., 2023).

Different interventions have been proposed to mitigate plasticity loss. **Data interventions** modify the training signals—either by modifying the observations fed to the network or by transforming the reward—e.g., batch size, frame resolution (RGB vs. down-sampled grayscale), or reward clipping (Mayor et al., 2025). **Auxiliary-losses** use regularization losses in addition to the PPO surrogate loss, such as PFO, penalizing feature drift (Moalla et al., 2024), Parseval, maintaining weight orthogonality, (Chung et al., 2024), and CHAIN, penalizing large output churn (Tang & Berseth, 2024). These

[*]Shared first authorship. [†]Joint supervision. Correspondence to `daniil.pyatko@epfl.ch`.

Preprint. Under review.

Table 1: Auxiliary loss functions addressing plasticity can be organized by (1) what component they apply the auxiliary loss on, and (2) the target to which this component is regularized. This taxonomy is illustrative rather than exhaustive: it organizes the losses already explored in previous work and reveals a large region of unexplored losses. A novel intervention, which we call CHAIN-SP, is derived as a modification of CHAIN (Tang & Berseth, 2024) thanks to this taxonomy. We encourage future work to investigate the remaining opportunities.

Towards \ Loss on	Weights	Features	Outputs
Null vector (0)	Weight Decay (AdamW)		
Initial model distribution (μ_{θ_0})	Parseval		
Initial model (θ_0)	L2 init	InFeR	
Model used in rollout (θ_t)		PFO	CHAIN-SP
Model after the grad step (θ_t^i)			CHAIN

extra losses aim to constrain updates so that weights, features, or outputs neither collapse nor grow unbounded. **Normalizing interventions** insert layers like LayerNorm, BatchNorm (Lyle et al., 2022), or Unit-Ball Normalization (UBN) to stabilize activations or weight distributions.

Each work, however, has its own setup: environment, deep RL algorithm, metrics, and a set of interventions used for comparison. Thus, it is difficult for a practitioner to get a clear overview of which metrics correlate with the loss of plasticity, which interventions provide significant benefits, and how they compare when they are all tested in the same setting. Previous plasticity studies have largely ignored diverse vision-based environments; for example, all of the vision-based environments considered by Juliani & Ash (2024) were sparse and their analysis mostly focused on plasticity under training distribution shifts. In contrast, we evaluate interventions across Atari environments with varying levels of reward sparsity and unchanging environments, allowing for a broader and more representative understanding of plasticity loss in vision-based RL. We use Proximal Policy Optimization (PPO) (Schulman et al., 2017), one of the most popular policy gradient methods used by a lot of practitioners, which re-uses each rollout for mini-batch updates over several epochs. This reuse of stale data makes it a particularly suitable candidate when analyzing plasticity loss. Hence, we aim to fill the gap by conducting a comprehensive study of interventions under a fixed training protocol with PPO, addressing how all these methods and metrics compare against each other and track plasticity loss. With this setup, we empirically investigate the following questions about plasticity:

- Do common plasticity-loss metrics reliably capture plasticity loss across different interventions?
- Which intervention type—architecture level (normalization), output level (auxiliary-loss), or input level (data)—best preserve plasticity?
- Do these findings hold across environments with varying reward sparsity?

Our contributions

1. **Taxonomy and novel intervention.** We classify 18 candidate methods into data (input), auxiliary losses (output), normalization layers (architecture), combinations of those, and rigorously tune and evaluate each within a single PPO framework on the ALE benchmark. Our auxiliary loss taxonomy allows us to identify gaps and introduce CHAIN-SP, a variant of CHAIN based on the model used in the rollout (Tang & Berseth, 2024).
2. **Empirical insights on data interventions.** We show that neither unpreprocessed inputs (full-resolution RGB) nor reduced gradient noise with larger batches prevent loss of plasticity.
3. **Evaluation of auxiliary losses and normalization.** We find that among the tuned auxiliary losses, churn losses result in the best performance and training stability, and that among normalization layers, LayerNorm provides the best stability without any adjustment.
4. **Assessment of combination methods.** We demonstrate how pairing LayerNorm with various auxiliary losses (e.g., L2 Init, PFO, CHAIN-SP) affects plasticity, revealing that certain combinations outperform their standalone counterparts.

5. **Feature-representation finding.** We show that a low PCA policy rank, near the number of discrete actions, does not hinder learning a strong policy, challenging the idea that preserving high PCA feature-rank alone is sufficient.
6. **Reproducibility.** We open-source our codebase, training logs, and hyperparameter sweeps to facilitate full reproducibility.

2 Background

Reinforcement Learning We work with a finite-horizon Markov decision process (MDP) $(\mathcal{S}, \mathcal{A}, r, P, H)$, where H is the horizon. At each time $t = 0, \ldots, H-1$, the agent observes $S_t \in \mathcal{S}$, selects $A_t \in \mathcal{A}$, then transitions to S_{t+1}, via P, and receives reward $r(S_t, A_t, S_{t+1})$. The return is $G_0 = \sum_{t=0}^{H-1} r(S_t, A_t, S_{t+1})$. We seek a policy that maximizes $\mathbb{E}[G_0]$.

On-policy Actor–Critic. In each iteration, the policy network π_θ collects N-step rollouts $\{(S_t, A_t, R_{t+1})\}$ by sampling actions from the current policy π_θ and storing the resulting transitions. The actor π_θ and the critic V_φ are then updated concurrently via gradient descent on that same batch of data. The critic minimizes the squared error between its prediction $V_\varphi(s_t)$ and a return target $\hat{G}_t$, where $\hat{G}_t$ is the λ-return computed using the Generalized Advantage Estimator (Schulman et al., 2015). The actor is trained with Proximal Policy Optimization (PPO) (Schulman et al., 2017).

Non-stationarity in deep RL and PPO. Each gradient step in policy gradient methods updates the policy, which in turn shifts the state-visitation distribution (occupancy measure) (Kang et al., 2018), alters the reward landscape, and changes the critic targets, introducing non-stationarity throughout training. Proximal Policy Optimization (PPO) limits how far each update can move away from the policy that generated the data by clipping the likelihood ratio between new and old policies to avoid large parameter jumps (Schulman et al., 2017). In practice, PPO collects a batch of transitions under the old policy and then performs multiple epochs of minibatch optimization on that same batch. Thus, PPO amplifies the impact of this non-stationarity by reusing the same rollout for multiple optimization epochs. (Nikishin et al., 2022) This repeated training on stale data increases the risk of overfitting to outdated trajectories and accentuates the distribution shift between the policy used for data collection and the one being updated. As a result, higher epoch counts can increase the effects of non-stationarity and contribute to plasticity loss, a central focus of this study.

Plasticity Loss Plasticity loss occurs when the capacity of a network to learn new data deteriorates despite ongoing training, often indicated by sharp decreases in feature representation metrics (e.g., feature-rank collapse (Moalla et al., 2024; Lyle et al., 2022), exploding capacity loss (Lyle et al., 2022; Nikishin et al., 2023) and by signs such as increased weight norms or increasing number of inactive ("dead") neurons (Gulcehre et al., 2022). This representation collapse is reflected in degrading performance, such as a decline in episodic return, even though optimization continues. In PPO, one can regard a consistent drop in episodic return across random seeds as evidence of plasticity loss. Tracking metrics such as capacity loss (error in fitting random targets), feature rank, weight/gradient norms, and dead neurons offers early warnings of incoming collapse. Understanding these indicators is important for judging how different interventions, such as adjusting inputs, adding auxiliary losses, or using normalization, keep representation capacity and avoid performance degradation.

3 Study design and methods

Experimental setup We train PPO on *Phoenix*, *NameThisGame*, and *Gravitar* from ALE. We chose these to cover both dense and sparse rewards and because together they achieve a high predictive correlation (Aitchison et al., 2023). We use 25% sticky actions. To study non-stationarity, we run each setting with $E \in \{4, 6, 8\}$ epochs; more epochs usually lower returns but make collapse happen sooner. Actor and critic use separate trunks (Cobbe et al., 2021). We use three seeds for each hyperparameter configuration. More details are in Appendix D.1.

3.1 Interventions

We organize interventions according to the three categories (1) data (input), (2) auxiliary losses (output), and (3) normalization layers (architecture), and include a fourth category for combinations of normalization layers with auxiliary losses. By developing the auxiliary-loss taxonomy in Table 1, we reveal a large region of unstudied loss functions and introduce a new CHAIN-based intervention,

Table 2: **Catalogue of interventions, grouped by how they modify (1) input or data (baseline & data-devel), (2) loss function (loss-level), (3) architecture via normalization (normalization layers), or (4) combinations (combinations).** For all interventions, we keep the *total* number of environment interactions fixed (100 million steps). Within each epoch, we also hold the number of gradient-update steps constant across interventions.

Category	Intervention	Main idea / target
Baseline & data-level	BASELINE	PPO with default hyper-parameters (sign reward, 84×84 grayscale, minibatch $= 256$ samples).
	BATCH$\times$8	8× more parallel envs, 8x bigger mini-batch; same number of environment steps.
	RGB–RAW	Full-resolution RGB frames; no grayscaling.
	NOCLIP	Raw rewards instead of $\mathrm{sign}(r)$.
Loss-level	L2-INIT	Initial weights regularization (Kumar et al., 2023).
	ADAMW	Adam w. decoupled weight decay (Loshchilov, 2017).
	PARSEVAL	Parseval weight orthogonality (Chung et al., 2024).
	CHAIN	Churn Reduction (Tang & Berseth, 2024).
	CHAIN-SP **(Ours)**	Sampling-Policy variant (ours). Towards the model used in rollout instead of the model after the grad. step.
	PFO	Proximal Feature Optimization (regularizes feature drift from sampling policy) (Moalla et al., 2024).
	INFER	Initial-Feature Regularisation (auxiliary-head drift penalty) (Lyle et al., 2022).
Normalization Layers	LN	LayerNorm with learnable (γ, β) (Ba et al., 2016).
	LN-NS	LayerNorm, γ=1, β=0.
	BN	BatchNorm with affine parameters.
	BN-NS	BatchNorm, γ=1, β=0.
	UBN	Unit-Ball Normalisation (Hussing et al., 2024).
Combinations	LN+PFO	LayerNorm + PFO.
	LN+BN-NS	LayerNorm + BatchNorm (no scale).
	LN+CHAIN-SP	LayerNorm + CHAIN-SP.

CHAIN-SP. We provide a comprehensive experimental design and the complete set of individual result plots in Appendix F, G. Table 2 presents a summary of the methods we test, with detailed experimental descriptions available in Appendix F. Figure 1 presents the summary boxplots, showing the effect of each intervention on episode return, feature rank (PCA), capacity loss, and the number of dead neurons in the policy. The full catalogue and the hyperparameter tuning protocol for the auxiliary loss intervention are in Appendix F and D.3.

3.2 Performance metrics

To analyze plasticity loss, we evaluate several metrics proposed in the literature, including the feature rank of the policy and value networks, the number of dead neurons, capacity loss, the penultimate layer pre-activation norm, and the norm of the policy weights; full definitions and details can be found in Appendix E.

- **Feature rank (policy/value).** PCA-based rank of the last hidden features. *Lower rank* means features collapse and plasticity collapsed.
- **Dead neurons (policy/value).** Count of ReLU units that are zero on the whole minibatch (Gulcehre et al., 2022). *More dead neurons* means less usable capacity.
- **Capacity loss.** After training, we briefly fine-tune a checkpoint to match actions/values from a fresh rollout and measure the remaining error. *Lower is better* (more plasticity kept); see Appendix for the loss terms and setup (Appendix E).

4 Experiment Results

4.1 Modifying the input data or rewards does not mitigate plasticity loss

Bigger batches boost returns but do not prevent collapse in the long run Increasing the minibatch size leads to improved performance across all environments and epoch counts. We

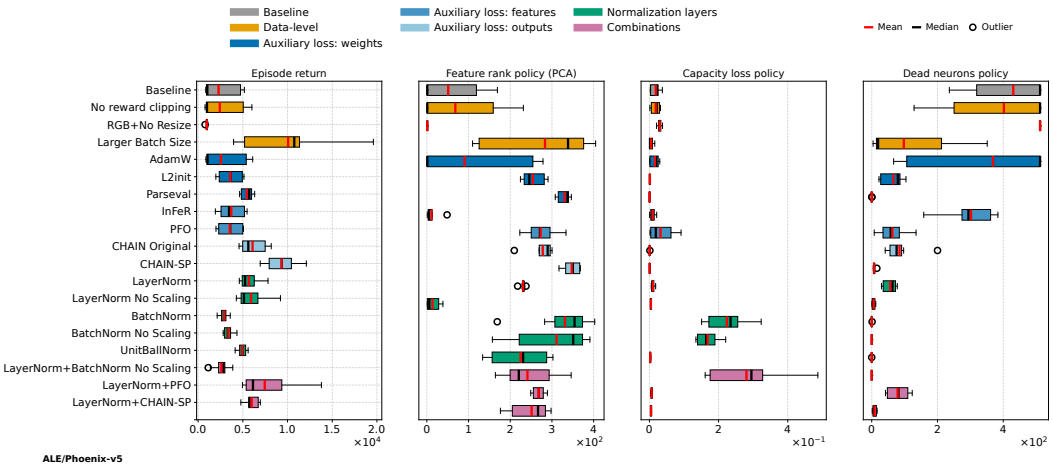


Figure 1: **Summary of all the studied interventions' effects on performance and plasticity in ALE/Phoenix** Most of the interventions improve on the baseline in plasticity loss metrics. We organize interventions into data-level, auxiliary-loss (weights, features and outputs), and normalization categories (plus their combinations), introduce CHAIN-SP as a novel loss-based method, and demonstrate that churn-reduction losses and LayerNorm most effectively prevent plasticity collapse. Auxiliary-loss approaches (especially churn-based) have few dead neurons and strong representations, while LayerNorm (alone or paired with losses) stabilizes training. By contrast, weight-based (e.g., AdamW) and data-based (e.g., reward-clipping removal) methods, although they can drastically improve absolute performance, fail to mitigate plasticity loss. A boxplot includes 9 runs with different epochs. More details in Appendix G.1

suspect this larger batch size is not commonly used because its benefits only become apparent after approximately 10 million training steps, which is already the typical total timestep limit for Atari (Huang et al., 2022; Schulman et al., 2017). This batch size and sample budget are more commonly observed in distributed settings (Huang et al., 2024). However, extended runs up to 200 million steps show that the trend of performance degradation persists, and the actor still collapses, similar to what occurs with a smaller batch size: despite the improved performance, the agent collapses after 8 epochs (Fig. 50). This suggests that the reduced gradient noise from the larger batch only delays the collapse, rather than preventing it. Even when the agent reaches a reward level of 6000 on Phoenix, a level that is not achieved on other collapsing runs, it still eventually fails. This provides evidence that there is no specific reward threshold (i.e., an Atari map stage) beyond which an agent becomes immune to collapse. Instead, some form of network regularization is necessary to prevent it. Quite interestingly, if we consider 4 epochs of training, increased batch size doesn't lead to collapse, but has typical values of plasticity-associated metrics. So, for short runs, a bigger batch can improve performance without hurting the network's ability to keep learning, just as Smith et al. (2017) observed when they scaled batch size in supervised tasks.

Enhancing the richness of the environment signal doesn't prevent the collapse, instead, it can speed it up When we train PPO on unscaled RGB frames, policy feature rank and episodic return collapse faster than the baseline, and the norm of policy preactivations increases rapidly. Unpreprocessed inputs, like full-resolution RGB frames or unclipped rewards, increase the number of available features and targets, which in turn gives the network more opportunities to overfit. Instead of preserving plasticity, this added complexity accelerates its collapse. By contrast, Ma et al. (2023) show that data augmentation (another technique to increase input variability) is critical in visual RL: introducing controlled perturbations prevents overfitting and helps maintain good feature representation metrics. Consequently, it makes sense that simply feeding unpreprocessed signals causes the network to overfit more quickly. Hence, we show that increasing input and reward complexity amplifies plasticity loss, underscoring the importance of resorting to visual preprocessing such as gray-scaling or reward clipping or using techniques like data augmentation.

4.2 Loss-level interventions impact different plasticity metrics differently

Weight based: AdamW, L2init and Parseval Each of the three weight–based regularization interventions displays different dynamics that illustrate how penalizing weights influences plasticity in deep RL. AdamW loses plasticity when the number of epochs is increased. When this happens, unlike with Adam, the weights of the network decrease significantly because of the weight decay.

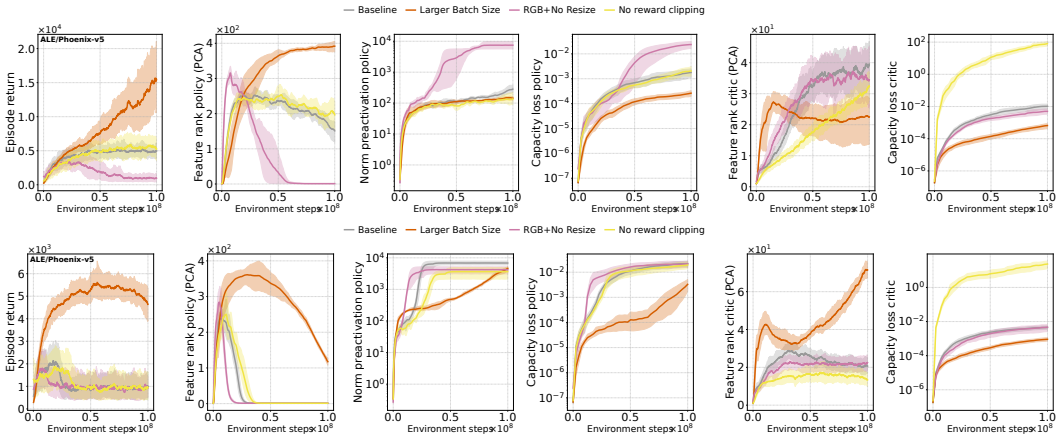


Figure 2: **Data-Level interventions on Phoenix (4 vs. 8 epochs)** Larger batches improve performance up to 100 M steps and delay collapse—maintaining high pre-activation norms and delaying rank loss—as shown by an 8-epoch feature-rank decline or a late-training capacity-loss rise. Giving raw full-RGB inputs leads to faster representation collapse, suggesting stronger overfitting between distribution shifts.

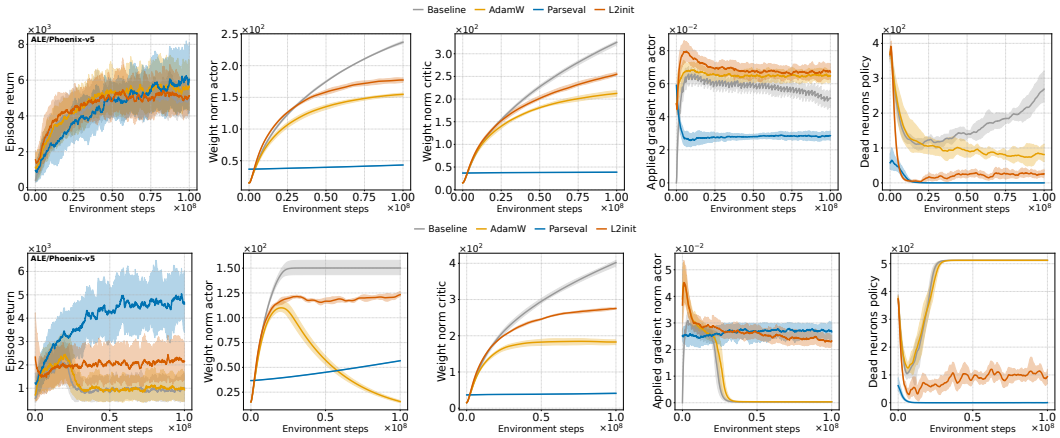


Figure 3: **Weight-level auxiliary loss interventions on Phoenix for 4 and 8 epochs.** Parseval shows the tightest control over weight norms, maintains nearly zero dead neurons, and keeps the actor's gradient norm stable—effectively preventing any plasticity collapse. L2Init also avoids collapse: it keeps high weight norms in both the actor and critic, while keeping the actor's gradient norm steady. In contrast, AdamW acts like the baseline under high-epoch training: the actor's weight norm drops and the number of dead neurons rises sharply.

Interestingly, even though the weights decrease continuously until only unregularized biases are left, the network is not able to find useful parameters during this decrease and return to non-trivial gradients and recover from the collapsed state. In fact, we can describe the dynamics of weight magnitude when training with AdamW precisely. We know that all of the neurons are dead, so the gradient in all layers before the classification layer will be 0. As a result, parameter updates come only from Adam's exponential moving average and the L2 weight penalty. After many gradient iterations with 0 gradient, the exponential moving average will be close to 0 and the only change will be from the L2 penalty. See Figure 3, actor and critic weight plots. In contrast, L2init pulls weights toward their initial nonzero values, and Parseval uses orthogonal initialization, starting with a higher initial weight norm, and is considered a good plasticity-preserving regularization, looking at plasticity indicators such as high and stable policy feature rank (Moalla et al., 2024), low weight norm (Kumar et al., 2023), and low number of dead neurons (Juliani & Ash, 2024). However, these metrics alone don't always give decisive conclusions (Lyle et al., 2023). We observe good metric results when using adapted Parseval regularization to convolutional layers, even though it was originally designed for linear layers (Chung et al., 2024); in the case of convolutions, we cannot claim that the function is being regularized toward Lipschitzness as we can do with linear layers (Chung et al., 2024).

Feature based: PFO and InFeR In this study, we apply PFO (Moalla et al., 2024) to both the actor and critic networks—unlike Moalla et al. (2024), who only regularize actor. While their results

suggest that targeting the policy alone can help with plasticity loss, we intentionally extend PFO to the critic to match our study design. We observe more plasticity loss and an eventual collapse in one environment (Fig. 29) by applying PFO on both actor and critic networks than just applying it to the actor. We hypothesize that over-constraining the critic might actually be disadvantageous, which echoes with the work of Liu et al. (2019). InFeR (Lyle et al., 2022) was originally designed for value-based methods, but here we apply it to both actor and critic. Even when used outside its original setting, InFeR reduces plasticity loss, showing that keeping features close to the initial model can also help the actor.

Output-based methods: CHAIN and CHAIN-SP The main goal of CHAIN is to reduce output churn—the undesired change in a model's outputs on data outside of the training set which emerges after the parameter update.

To mitigate the churn of the models, Chung et al. (2024) introduce an auxiliary loss that penalizes changes in the model's predictions on a held-out batch $\bar{B}_t$ not used for training. Specifically, the CHAIN loss is defined as the average distance between the outputs of the reference models (($V_{\varphi_t^{i-1}}(s)$, $\pi_{\theta_t^{i-1}}(s)$) for CHAIN or ($V_{\varphi_t}(s)$, $\pi_{\theta_t}(s)$) for CHAIN-SP) and current (($V_\varphi(s)$, $\pi_\theta(s)$)) models.

Both CHAIN and CHAIN-SP show systematically lower B_{simple} at early epochs and higher values later, in contrast to the baseline, whose B_{simple} drops right around collapse (Fig. **??**).

CHAIN Reference models in the original CHAIN (Tang & Berseth, 2024) are the last minibatch update models, giving the loss

$$L^{\pi}_{\text{CHAIN}} = \frac{1}{|\bar{B}_t|} \sum_{(s,a)\in\bar{B}_t} \text{KL}(\pi_\theta(s)||\pi_{\theta_t^{i-1}}(s)), \quad L^{V}_{\text{CHAIN}} = \frac{1}{|\bar{B}_t|} \sum_{(s,a)\in\bar{B}_t} (V_{\varphi_t^{i-1}}(s) - V_\varphi(s))^2$$

In Chung et al. (2024), CHAIN regularization in PPO experiments was only applied to actor and the loss scaling was in most cases dynamic to keep approximately the same ratio to the PPO loss during training. In our setup, we regularize both actor and critic and fix the loss scaling coefficient to have a fair comparison with other interventions.

CHAIN-SP Building on this work, we introduce CHAIN-SP (CHAIN Sampling Policy), whom reference models are the sampling policy models, yielding the loss:

$$L^{\pi}_{\text{CHAIN-SP}} = \frac{1}{|\bar{B}_t|} \sum_{(s,a)\in\bar{B}_t} \text{KL}(\pi_\theta(s)||\pi_{\theta_t}(s)), \quad L^{V}_{\text{CHAIN-SP}} = \frac{1}{|\bar{B}_t|} \sum_{(s,a)\in\bar{B}_t} (V_{\varphi_t}(s) - V_\varphi(s))^2$$

CHAIN-SP regularizes more strongly than CHAIN, as it regularizes towards an older version of the network. This likely explains its better plasticity metrics across environments.

On NameThisGame, however, we observe that CHAIN-SP collapses. We note that it happens in a very specific case. The collapse in CHAIN-SP is likely due to the strong regularization of *critic* towards the *sampling* parameters. To prove that, we separately ablate critic regularization and change the objective of regularization. When using CHAIN — which regularizes towards the previous minibatch's model outputs — and keeping critic regularization, no collapse occurs. Similarly, the collapse doesn't happen when disabling the critic's auxiliary loss by setting its weight to zero in CHAIN-SP. The collapse of PFO on NameThisGame, which also regularizes *critic* features towards the *sampling* policy, supports our hypothesis.

A closer analysis shows that the distinct reward structure of NameThisGame leads to different training dynamics. The value loss is an order of magnitude higher and the PCA rank of value features is an order of magnitude lower than in Phoenix during training (Fig 39). This suggests that CHAIN-SP's strong regularization has irreversibly damaged the already weak value function representations.

We also note that within the 100M environment steps range, CHAIN-SP only collapses on low epochs. This type of collapse is different from the baseline collapse which happens only on high epochs and it can't be associated with overfitting (Nikishin et al., 2022).

A collapse occurring with the use of universal hyperparameters, which could have been avoided by setting the critic's loss coefficient to zero or by thoroughly tuning specifically for NameThisGame, highlights the importance of careful hyperparameter tuning for auxiliary-loss interventions.

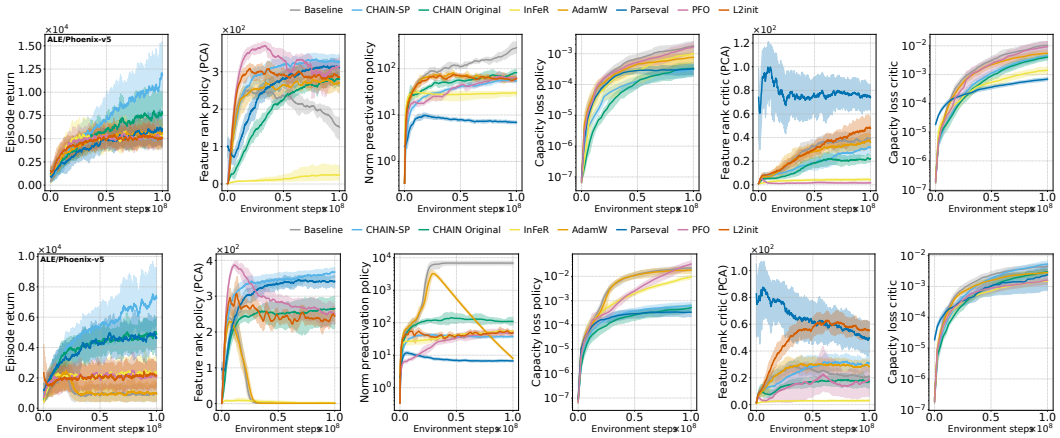


Figure 4: **Auxiliary loss interventions on Phoenix, 4 and 8 epochs** Compared to the baseline, CHAIN-SP, CHAIN, and Parseval all lead to improved representations and prevent performance collapse when we increase the number of epochs to 8. InFeR prevents collapse while having a relatively low policy feature rank and a high capacity loss policy. L2init and PFO on actor and critic produce stronger representations, though without performance improvement.

4.3 Normalization layers

LayerNorm, a simple but effective intervention LayerNorm (Ba et al., 2016) standardizes each layer's pre-activations before the nonlinearity and applies learnable scale γ and bias β. LayerNorm (i) reduces gradient covariance and prevents large gradient norms (Lyle et al., 2023), (ii) mitigates "unit linearization" by stabilizing the pre-activation distribution (Lyle et al., 2024b), and (iii) reactivates dead ReLU units by guaranteeing nonzero normalized gradients even when pre-activations would be negative (Lyle et al., 2024a). Klein et al. (2024) provide a short summary of these results. In our experiments, LN maintains feature rank and prevents representational collapse, even as the norm of the pre-activation features of policy grow over time (Fig.15). Removing the learnable scale (LN-NS) still enforces zero-mean, unit-variance but doesn't necessarily preserve rank and has a lower capacity loss than LN with learnable scale. Without the scale parameter, LayerNorm outputs are normalised to mean 0 and standard deviation 1. There isn't a learned multiplier to make some features bigger or smaller. This can reduce differences across features (so rank isn't preserved), but it makes gradient magnitudes more consistent (i.e., more stable), but it also makes the model easier to fit to random targets than with LN-S — hence lower capacity loss. Overall, inserting LN before each activation gives a robust, high-rank representation throughout deep RL training.

Other normalization layers BatchNorm helps training at first but it hurts model's plasticity over time. Studies show that normalizing across each mini-batch smooths the loss surface and makes gradients smaller, which keeps weights from changing enough to fit new data (Santurkar et al., 2018). Previous work have shown that in continual learning, models with BatchNorm lose accuracy on later tasks faster than models without it or with LayerNorm (Dohare et al., 2023a). In our PPO-Atari tasks, policies using BatchNorm always show higher capacity loss and have larger preactivation norms than both baseline and LayerNorm policies, even when returns stay stable on NameThisGame. Overall, BatchNorm stabilizes training early but reduces long-term plasticity, and methods like LayerNorm work better when the network needs to keep adapting over time.

LayerNorm with auxiliary interventions: combining different strengths Combining LN with additional loss-level or regularization techniques can further enhance plasticity, though not all pairings succeed. Since the precise mechanisms behind plasticity loss remain uncertain and likely involve multiple factors (Lyle et al., 2024b), it makes sense to combine methods that each address particular aspects of this degradation. Accordingly, various interventions, like Normalize-and-Project (Lyle et al., 2024a)—couple pre-activation normalization with a *projection* step that rescales each weight matrix to unit Frobenius norm. Klein et al. (2024, Section 6.9) provides an exhaustive list of combined methods. We note that many of these approaches use LN as a foundation because of its benefits, and we follow the same strategy. When integrating LN with CHAIN-SP, we get higher returns, more stable feature rank, and lower capacity loss than LN alone. We hypothesize that this is due to different

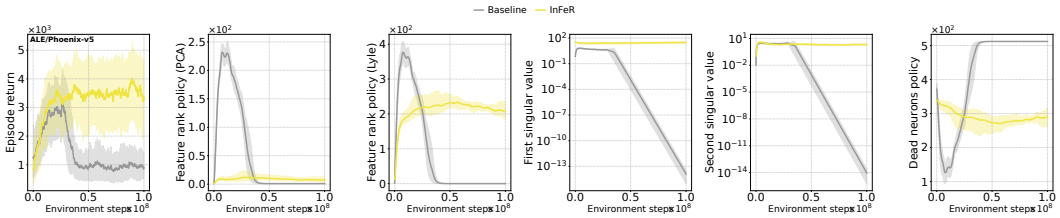


Figure 5: **Reward and SVD statistic for Phoenix on 6 epochs.** Using InFeR, only a small subspace remains that keeps almost all of the PCA information about the policy feature matrix. We suspect that the policy PCA rank is hovering above 10 because 10 is the dimension in which we try to keep the output features in InFeR. See Appendix G.3 for the plots on other environments.

ways these modifications affect training: LayerNorm smoothes out the loss function and PFO and CHAIN-SP don't allow actor and critic drift too much during training.

4.4 Dynamics over magnitudes

We observe that an increase in the policy network's pre-activation norm with a decrease in its feature rank correlates with future reward collapse, indicating that it seems to be the *dynamics* of these metrics during training, not their values that matter. Different papers use different rank definitions (e.g., singular-value–based stable-rank vs. PCA-based rank; see Moalla et al., 2024, App. D), and this choice strongly affects how one interprets collapse. For example, using InFeR we aim to preserve a low-dimensional subspace from the initialization, yet training never collapses, showing that high PCA feature-rank is *not* necessary for stability (Fig. 5). We suspect that the PCA rank hovers around 10 because InFeR regularizes policy features towards a 10-dimensional subspace (10 auxiliary heads). At the same time, the normalized SVD-based rank (which highlights that most singular values are relatively small but separated from zero; see Lyle et al., 2022) seems more appropriate to monitor the collapse (Fig. 5). In a similar way, LN-NS has a low policy PCA feature rank for several settings (Fig. 14, 15, 20, 21) without any reward collapse or capacity-loss increase. LayerNorm, too, has a steadily increasing norm preactivation policy without collapse as shown by Lyle et al. (2024a). In short, it seems to be more about *how which* rank and norm evolve across epochs (rapid rise or fall rather than plateaus or slow changes) that seems to correlate with plasticity loss, rather than their static values. We can draw this conclusion thanks to our comprehensive list of interventions and plasticity metrics.

5 Related Work

Large batch sizes with distributed training High-throughput agents such as IMPALA (Espeholt et al., 2018) and SEED RL (Espeholt et al., 2019) collect millions of frames per second using hundreds to thousands of actors, and large-scale projects like OpenAI Five and AlphaStar trained for months using massive batches across distributed workers (Berner et al., 2019; Vinyals et al., 2019). These systems show that large batches can support sustained learning in practice without apparent loss of plasticity. However, since none of them report plasticity-related metrics, such as feature rank or dead neurons, it remains unclear whether such training avoids plasticity loss. Building on the idea that higher throughput might prevent plasticity collapse, Mayor et al. (2025) show that, with a budget of 100 million steps, collecting 2048 transitions per update (16 environments $\times$ 128 steps) and using minibatches of 512 samples preserves feature rank and prevents weight-norm spikes. In our setting, with a doubled budget of 200 million environment steps, collecting 8192 transitions per update (64 envs $\times$ 128 steps) and using minibatches of 2048 samples, we still observe feature-rank collapse when we increase data reuse by raising the number of epochs (see Fig 50). Hence, increasing the batch size seems to only help up to some limits by delaying the collapse.

Normalization-centric architectures and impacts across observation and action spaces Recent works suggest that architectural choices, especially normalization layers, impact training stability in various domains, which can be related to preserving plasticity. In continuous control with vector state representations, Nauman et al. (2024b) show that adding LayerNorm after every dense layer plus lightweight decay lets SAC critics grow to 26M parameters without gradient spikes or value overestimation, and Nauman et al. (2024a) also that applying LayerNorm boosts performance without introducing value overestimation, making pessimistic Q-learning unnecessary. Without such intervention, Dohare et al. (2023b); Moalla et al. (2024) report performance collapse due to plasticity loss when training PPO agents on MuJoCo. Lyle et al. (2024b) sheds light on the connection between

stability and plasticity, showing that training instabilities such as abrupt weight updates create dead neurons and lead to plasticity loss. Hence, keeping those updates stable prevents the increase in dead neurons and thus preserves plasticity. Neither of these works studying normalization, however, evaluate pixel-based or discrete-action environments, leaving it unclear whether normalization offers the same stability and plasticity benefits in those environments. In this work, we apply the same LayerNorm recipe to pixel-based, discrete-action agents and find that it prevents the loss of plasticity in those environments. Our pixel–only results complement these vector-state findings and provide baselines for future cross-domain studies.

Reset-style interventions Several approaches have been proposed to recover representation capacity once neurons become inactive. They fit in a different family than the three families of interventions we study in this work and are termed *reset methods* (Farias & Jozefiak, 2024). Continual Backpropagation (CBP) (Dohare et al., 2021, 2024) computes a running *utility score*—a moving average of how often a neuron fires times the strength of its forward connections—for every neuron and, at *every* training step, resets the few neurons with the lowest scores to fresh random weights;, without changing the network's size. ReDo (Sokar et al., 2023) periodically scans the network; at each pass it identifies any neuron whose current normalized mean activation on the mini-batch falls below a threshold τ as τ-dormant, then simultaneously re-initializes the incoming weights (and zeroes the outgoing ones) of all such neurons. Capacity is therefore restored only after dormancy is observed, not in advance. Self-Normalised Resets (Farias & Jozefiak, 2024) monitor neuron firing rates (fraction of recent inputs for which that ReLU's output is positives) and reset a neuron's weights when that activity drops to zero, preventing plasticity loss in continual learning tasks. Neuroplastic Expansion (Liu et al., 2024) adds new neurons instead of resetting old ones, and the authors show it helps in MuJoCo. Although these interventions are popular in the literature and provide effective mitigation strategies, we do not investigate them in the scope of this work which is focused on studying continuous stable changes during training to keep neurons active rather than ad hoc resets. We believe continuous interventions are closer to biological neural networks which rely on gradual processes such as homeostatic plasticity, a slow feedback that scales synapses to keep firing rates near a target rate - average firing level a neuron tries to stay at, (Tononi & Cirelli, 2003; Surget & Belzung, 2022; Turrigiano, 2008).

6 Conclusion

We empirically evaluate eighteen data-level, auxiliary-loss, and normalization interventions on plasticity loss for PPO in ALE, including our new CHAIN-SP loss, and introduce a framework for classifying auxiliary losses. We find that simple churn-reduction losses and LayerNorm yield the strongest performance and show that standard plasticity metrics—dead-neuron number and policy PCA rank—become misleading. Based on these results, we recommend LayerNorm, a carefully tuned churn-based loss, and larger batches to improve stability and preserve plasticity. Finally, our classification framework lays the groundwork for future methods.

Limitations We show that applying considered interventions—LayerNorm, churn-based loss, and larger batch size—consistently improves PPO performance on the pixel-based ALE benchmark. But whether these gains carry over to value-based algorithms such as DQN, off-policy actor-critic methods like SAC, continuous-control domains, or non-vision tasks remains an open question. Furthermore, we establish a baseline by selecting all hyperparameters from results on the Phoenix game (see Appendix D.3), but per-game or joint tuning across multiple environments could uncover configurations that further enhance sensitive methods such as PFO or CHAIN-SP. We apply all interventions to the actor and critic jointly, however testing them on only one component could change their effects and clarify how each intervention separately influences policy learning and value estimation. We investigate data strategies, auxiliary losses, and normalization techniques; yet evaluating alternative architectures, off-policy data, broader algorithm and task suites, additional metrics, and other intervention types is still required to generalize our conclusions beyond our case study.

Acknowledgments

We thank the reviewers for their valuable insights, which greatly improved the clarity and rigour of this work. We thank the EPFL SCITAS team for the access to the beta testing phase of their new cluster. We also thank Karin Gétaz for the administrative support provided within EPFL.

References

Matthew Aitchison, Penny Sweetser, and Marcus Hutter. Atari-5: Distilling the arcade learning environment down to five games. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 421–438. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/aitchison23a.html. 3, 20

Maksym Andriushchenko, Dara Bahri, Hossein Mobahi, and Nicolas Flammarion. Sharpness-aware minimization leads to low-rank features. In A. Oh, T. Neumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 47032–47051. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/92dd1adab39f362046f99dfe3c39d90f-Paper-Conference.pdf. 22

Jordan Ash and Ryan P Adams. On warm-starting neural network training. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 3884–3894. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/288cd2567953f06e460a33951f55daaf-Paper.pdf. 26

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. 4, 8, 25

Yoav Benjamini and Yosef Hochberg. Controlling the false discovery rate: A practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society: Series B (Methodological)*, 57(1):289–300, 1995. doi: https://doi.org/10.1111/j.2517-6161.1995.tb02031.x. URL https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/j.2517-6161.1995.tb02031.x. 18

Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysλaw Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019. 9

Wesley Chung, Lynn Cherif, Doina Precup, and David Meger. Parseval regularization for continual reinforcement learning. *Advances in Neural Information Processing Systems*, 37:127937–127967, 2024. 1, 4, 6, 7, 25

Karl W Cobbe, Jacob Hilton, Oleg Klimov, and John Schulman. Phasic policy gradient. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 2020–2027. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/cobbe21a.html. 3, 20

Shibhansh Dohare, Richard S Sutton, and A Rupam Mahmood. Continual backprop: Stochastic gradient descent with persistent randomness. *arXiv preprint arXiv:2108.06325*, 2021. 10, 26

Shibhansh Dohare, J Fernando Hernandez-Garcia, Parash Rahman, A Rupam Mahmood, and Richard S Sutton. Maintaining plasticity in deep continual learning. *arXiv preprint arXiv:2306.13812*, 2023a. 8

Shibhansh Dohare, Qingfeng Lan, and A. Rupam Mahmood. Overcoming policy collapse in deep reinforcement learning. In *Sixteenth European Workshop on Reinforcement Learning*, 2023b. URL https://openreview.net/forum?id=m9Jfdz4ym0. 9

Shibhansh Dohare, J. Fernando Hernandez-Garcia, Qingfeng Lan, Parash Rahman, A. Rupam Mahmood, and Richard S. Sutton. Loss of plasticity in deep continual learning. *Nature*, 632 (8026):768–774, August 2024. ISSN 1476-4687. doi: 10.1038/s41586-024-07711-7. URL https://www.nature.com/articles/s41586-024-07711-7. Publisher: Nature Publishing Group. 1, 10

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International conference on machine learning*, pp. 1407–1416. PMLR, 2018. 9

Lasse Espeholt, Raphaël Marinier, Piotr Stanczyk, Ke Wang, and Marcin Michalski. Seed rl: Scalable and efficient deep-rl with accelerated central inference. *arXiv preprint arXiv:1910.06591*, 2019. 9

Massimiliano Falzari and Matthia Sabatelli. Fisher-guided selective forgetting: Mitigating the primacy bias in deep reinforcement learning. *arXiv preprint arXiv:2502.00802*, 2025. 24

Zho Fan, 2016. URL `https://web.stanford.edu/class/archive/stats/stats200/stats200.1172/Lecture11.pdf`. 18

Vivek F Farias and Adam D Jozefiak. Self-normalized resets for plasticity in continual learning. *arXiv preprint arXiv:2410.20098*, 2024. 10

Caglar Gulcehre, Srivatsan Srinivasan, Jakub Sygnowski, Georg Ostrovski, Mehrdad Farajtabar, Matthew Hoffman, Razvan Pascanu, and Arnaud Doucet. An empirical study of implicit regularization in deep offline RL. *Transactions on Machine Learning Research*, 2022. ISSN 2835-8856. URL `https://openreview.net/forum?id=HFfJWx6OIT`. 3, 4, 22

Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G.M. Araújo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022. URL `http://jmlr.org/papers/v23/21-1342.html`. 5, 20

Shengyi Huang, Jiayi Weng, Rujikorn Charakorn, Min Lin, Zhongwen Xu, and Santiago Ontanon. Cleanba: A reproducible and efficient distributed reinforcement learning platform. In *The Twelfth International Conference on Learning Representations*, 2024. 5

Marcel Hussing, Claas Voelcker, Igor Gilitschenski, Amir-massoud Farahmand, and Eric Eaton. Dissecting deep rl with high update ratios: Combatting value overestimation and divergence. *arXiv preprint arXiv:2403.05996*, 2024. 4, 25

Sergey Ioffe. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015. 25

Ilse CF Ipsen and Arvind K Saibaba. Stable rank and intrinsic dimension of real and complex matrices. *arXiv preprint arXiv:2407.21594*, 2024. 24

Arthur Juliani and Jordan Ash. A study of plasticity loss in on-policy deep reinforcement learning. *Advances in Neural Information Processing Systems*, 37:113884–113910, 2024. 1, 2, 6, 17, 25, 26

Bingyi Kang, Zequn Jie, and Jiashi Feng. Policy optimization with demonstrations. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 2469–2478. PMLR, 10–15 Jul 2018. URL `https://proceedings.mlr.press/v80/kang18a.html`. 3

Timo Klein, Lukas Miklautz, Kevin Sidak, Claudia Plant, and Sebastian Tschiatschek. Plasticity loss in deep reinforcement learning: A survey. *arXiv preprint arXiv:2411.04832*, 2024. 8

Aviral Kumar, Rishabh Agarwal, Dibya Ghosh, and Sergey Levine. Implicit under-parameterization inhibits data-efficient deep reinforcement learning. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=O9bnihsFfXU`. 22

Saurabh Kumar, Henrik Marklund, and Benjamin Van Roy. Maintaining plasticity in continual learning via regenerative regularization, 2023. 4, 6, 25

Alex Lewandowski, Haruto Tanaka, Dale Schuurmans, and Marlos C Machado. Directions of curvature as an explanation for loss of plasticity. *arXiv preprint arXiv:2312.00246*, 2023. 1

Jiashun Liu, Johan Obando-Ceron, Aaron Courville, and Ling Pan. Neuroplastic expansion in deep reinforcement learning. *arXiv preprint arXiv:2410.07994*, 2024. 10

Zhuang Liu, Xuanlin Li, Bingyi Kang, and Trevor Darrell. Regularization matters in policy optimization. *arXiv preprint arXiv:1910.09191*, 2019. 7

J. Scott Long and Laurie H. Ervin. Using heteroscedasticity consistent standard errors in the linear regression model. *The American Statistician*, 54(3):217–224, 2000. ISSN 00031305, 15372731. URL http://www.jstor.org/stable/2685594. 17

I Loshchilov. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017. 4, 25

Clare Lyle, Mark Rowland, and Will Dabney. Understanding and preventing capacity loss in reinforcement learning. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=ZkC8wKoLbQ7. 1, 2, 3, 4, 7, 9, 22, 25

Clare Lyle, Zeyu Zheng, Evgenii Nikishin, Bernardo Avila Pires, Razvan Pascanu, and Will Dabney. Understanding plasticity in neural networks. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 23190–23211. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/lyle23b.html. 1, 6, 8

Clare Lyle, Zeyu Zheng, Khimya Khetarpal, James Martens, Hado P van Hasselt, Razvan Pascanu, and Will Dabney. Normalization and effective learning rates in reinforcement learning. *Advances in Neural Information Processing Systems*, 37:106440–106473, 2024a. 8, 9

Clare Lyle, Zeyu Zheng, Khimya Khetarpal, Hado van Hasselt, Razvan Pascanu, James Martens, and Will Dabney. Disentangling the causes of plasticity loss in neural networks. *arXiv preprint arXiv:2402.18762*, 2024b. 8, 9

Guozheng Ma, Lu Li, Sen Zhang, Zixuan Liu, Zhen Wang, Yixin Chen, Li Shen, Xueqian Wang, and Dacheng Tao. Revisiting plasticity in visual reinforcement learning: Data, modules and training stages. *arXiv preprint arXiv:2310.07418*, 2023. 5

Walter Mayor, Johan Obando-Ceron, Aaron Courville, and Pablo Samuel Castro. The impact of on-policy parallelized data collection on deep reinforcement learning networks. *arXiv preprint arXiv:2506.03404*, 2025. 1, 9

Skander Moalla, Andrea Miele, Razvan Pascanu, and Caglar Gulcehre. No representation, no trust: Connecting representation, collapse, and trust issues in ppo. *arXiv preprint arXiv:2405.00662*, 2024. 1, 3, 4, 6, 9, 20, 22, 24, 25, 26

Michal Nauman, Michaλ Bortkiewicz, Piotr Miλoś, Tomasz Trzciński, Mateusz Ostaszewski, and Marek Cygan. Overestimation, overfitting, and plasticity in actor-critic: the bitter lesson of reinforcement learning. *arXiv preprint arXiv:2403.00514*, 2024a. 9

Michal Nauman, Mateusz Ostaszewski, Krzysztof Jankowski, Piotr Miλoś, and Marek Cygan. Bigger, regularized, optimistic: scaling for compute and sample-efficient continuous control. *arXiv preprint arXiv:2405.16158*, 2024b. 9

Evgenii Nikishin, Max Schwarzer, Pierluca D'Oro, Pierre-Luc Bacon, and Aaron Courville. The primacy bias in deep reinforcement learning. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 16828–16847. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/nikishin22a.html. 3, 7, 20

Evgenii Nikishin, Junhyuk Oh, Georg Ostrovski, Clare Lyle, Razvan Pascanu, Will Dabney, and Andre Barreto. Deep reinforcement learning with plasticity injection. In A. Oh, T. Neumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 37142–37159. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/75101364dc3aa7772d27528ea504472b-Paper-Conference.pdf. 3, 22

Olivier Roy and Martin Vetterli. The effective rank: A measure of effective dimensionality. In *2007 15th European Signal Processing Conference*, pp. 606–610, 2007. 24

Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? *Advances in neural information processing systems*, 31, 2018. 8

Amartya Sanyal, Philip HS Torr, and Puneet K Dokania. Stable rank normalization for improved generalization in neural networks and gans. *arXiv preprint arXiv:1906.04659*, 2019. 24

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015. 3

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 2, 3, 5

Samuel L Smith, Pieter-Jan Kindermans, Chris Ying, and Quoc V Le. Don't decay the learning rate, increase the batch size. *arXiv preprint arXiv:1711.00489*, 2017. 5

Ghada Sokar, Rishabh Agarwal, Pablo Samuel Castro, and Utku Evci. The dormant neuron phenomenon in deep reinforcement learning. In *International Conference on Machine Learning*, pp. 32145–32168. PMLR, 2023. 1, 10, 26

A Surget and C Belzung. Adult hippocampal neurogenesis shapes adaptation and improves stress response: a mechanistic and integrative perspective. *Mol. Psychiatry*, 27(1):403–421, January 2022. 10

Hongyao Tang and Glen Berseth. Improving deep reinforcement learning by reducing the chain effect of value and policy churn. *Advances in Neural Information Processing Systems*, 37:15320–15355, 2024. 1, 2, 4, 7, 26

Giulio Tononi and Chiara Cirelli. Sleep and synaptic homeostasis: a hypothesis. *Brain Res. Bull.*, 62 (2):143–150, December 2003. 10

Gina G Turrigiano. The self-tuning neuron: synaptic scaling of excitatory synapses. *Cell*, 135(3): 422–435, October 2008. 10

Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, L. Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander Sasha Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom Le Paine, Caglar Gulcehre, Ziyun Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy P. Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575:350 – 354, 2019. URL `https://api.semanticscholar.org/CorpusID:204972004`. 9

Yuzhe Yang, Guo Zhang, Zhi Xu, and Dina Katabi. Harnessing structures for value-based planning and reinforcement learning. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=rklHqRVKvH`. 22

Appendix contents

A The role of the reward structure in plasticity collapse **16**

B Which studied metric best captures plasticity? **17**

C Breaking Down Batch Size Improvements **18**

D Study and methods details **20**

D.1 Experimental setup 20

D.2 Interventions 20

D.3 Hyperparameter Tuning 21

D.4 Additional details on our experimental setup 22

E Metrics details **22**

E.1 Dead neurons 22

E.2 Feature rank 22

E.3 Capacity loss 22

E.4 Discussion about other possible metrics. 24

F Interventions **24**

F.1 Data-level modifications 25

F.2 Loss function modifications 25

F.3 Network architecture modifications 25

F.4 Churn based modifications 26

F.5 Combinations of modifications 26

F.6 Where do reset-style methods fit in our taxonomy? 26

G Main paper figures on all environments. **26**

G.1 Figure 1 on all environments 26

G.2 Figure 2 and 4 with other interventions and number of epochs 28

G.3 Figure 5 on other environments 35

H Extra figures. **35**

H.1 Extended training runs 35

H.2 Plasticity figures 35

A The role of the reward structure in plasticity collapse

Baseline: dense vs. sparse. We study how *plasticity-related metrics* relate to *reward structure* using a **baseline PPO** agent, grouping tasks as dense (*Phoenix*, *NameThisGame*) and sparse (*Gravitar*).

Scope note. Our dense/sparse split uses two dense games (Phoenix, NameThisGame) and one sparse game (Gravitar); conclusions for sparse rewards are therefore illustrative and may vary with other sparse environments.

What we see (baseline). Across epochs, dense and sparse behave differently (Fig. 6):

1. **Policy feature rank (PCA)** drops much faster in dense. In *Gravitar* it decays more slowly even at high epoch.
2. **Policy preactivation norm** grows quickly in dense but only slowly in sparse and thus also across all epochs.
3. **Dead policy neurons** rise faster in dense than in sparse across all epochs.

Other monitored metrics do not show a consistent baseline pattern.

Why this pattern is expected. Dense tasks provide frequent, informative rewards. Under PPO this means more (and larger) updates per epoch, so representations move faster: feature rank collapses sooner, activations grow, and more units go inactive. Increasing the epoch budget compounds these effects in dense tasks, hence the visibly faster collapse. In sparse tasks, useful updates are rarer and noisier, so the same increase in epochs leads to fewer effective updates; metrics still drift in the same direction but with a much smaller slope.

Main takeaways (beyond baseline). Within each reward regime (Tab. 3), higher **policy PCA rank** and **policy variance** track higher reward, while **dead neurons** and large pre-activation norms track lower reward; the pattern is clear in **dense** and weaker (sometimes flipped for dead neurons) in **sparse**. By environment (Tab. 4), **Phoenix** shows faster drift toward collapse as the number of epochs E increases, **Gravitar** trends the other way, and **NameThisGame** is mixed. Interventions that stabilize features and scales (larger batches, normalization, CHAIN–SP) help more in **dense**, and they shift plasticity in the desired direction (higher policy rank, fewer dead neurons), producing larger dense–sparse gaps (Tabs. 5, 6). *Why does reward structure matter here?* Beyond update frequency (baseline), the key is **credit assignment strength**: dense rewards give consistent, fine-grained signals that *select* which features stay active and useful. Runs that keep many distinct policy features (high rank) and maintain action diversity get reinforced and earn higher reward; collapse (dead neurons, blown-up pre-activations) is penalized. In sparse rewards, feedback is *rare and noisy*, so many updates carry too little information to point to the helpful features; links to rank and variance are weak, and the number of dead neurons can still go up even though we never prune: some ReLU units just stop firing because their inputs stay non-positive. Methods that clean up gradients or stabilize scales raise the effective signal-to-noise of credit assignment, so they yield bigger gains where feedback is already informative—**dense**.

Summary. Dense tasks collapse faster as epochs increase. Rank gap and policy PCA rank track reward in both regimes (stronger in dense). Dead units are regime-dependent. Interventions that stabilize features (larger batch size, normalization layers, CHAIN–SP) improve reward and reduce collapse markers, with larger effects in dense.

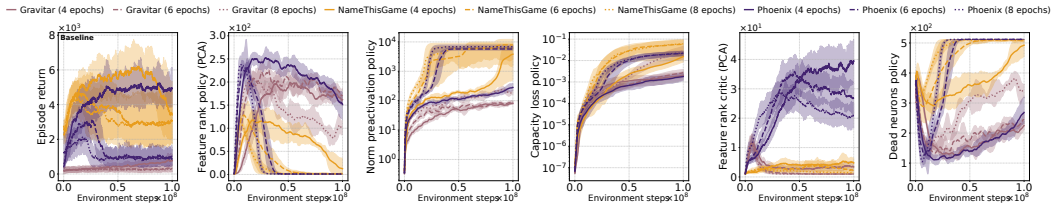


Figure 6: The 3 environments without any interventions with 4, 6 and 8 epochs.

Note on comparability. Absolute PCA ranks are not directly comparable across environments because input statistics and action spaces differ. We therefore report *within-environment* changes (deltas vs. that env's baseline).

Metrics	Dense	Sparse
Rank Gap	0.583***	0.299***
PCA Rank (Policy)	0.515***	0.328***
PCA Rank (Value)	-0.051	0.033
Policy Variance	0.423***	0.146
Dead Neurons (Policy)	-0.398***	0.257***
Norm Preac. Policy	-0.176**	-0.238**

Table 3: Spearman correlation between metric and normalized reward within dense/sparse groups. Stars: $p<.01$**, $p<.001$***.

Env	PCA Rank (Policy)	Policy Variance	Dead Neurons (Policy)	Norm Preac. Policy	Rank Gap
Gravitar	21.824	-0.001	-2.85	-186.757	22.68
NameThisGame	11.684	-0.002	5.102	672.438	12.414
Phoenix	-4.916	-0.005	22.576	390.948	-4.144

Table 4: Slope per epoch by environment (positive means the metric increases with more epochs).

Group	Rank	@4 epochs	@6 epochs	@8 epochs
Dense	1	Larger Batch Size (+1.228)	Larger Batch Size (+1.069)	CHAIN–SP (+0.655)
	2	CHAIN–SP (+0.497)	LayerNorm+CHAIN–SP (+0.764)	LayerNorm+CHAIN–SP (+0.562)
	3	LayerNorm (+0.497)	LayerNorm No Scaling (+0.631)	Larger Batch Size (+0.529)
Sparse	1	Larger Batch Size (+0.153)	Larger Batch Size (+0.165)	CHAIN–SP (+0.123)
	2	PFO (+0.109)	LayerNorm (+0.128)	Larger Batch Size (+0.096)
	3	No reward clipping (+0.007)	CHAIN–SP (+0.116)	LayerNorm+CHAIN–SP (+0.090)

Table 5: Top-3 interventions by Δ normalized reward *within* each reward structure (median vs. baseline).

Intervention	@4 epochs	@6 epochs	@8 epochs
Larger Batch Size	+1.075	+0.905	+0.433
LayerNorm	+0.653	+0.492	+0.420
CHAIN–SP	+0.562	+0.454	+0.531
RGB+No Resize	-0.156	+0.004	+0.001

Table 6: Dense $-$ Sparse gap in Δ normalized reward (median vs. baseline). Positive $\Rightarrow$ helps dense more.

B Which studied metric best captures plasticity?

Setup. We fit two generalized linear models (GLMs) over all environments (Phoenix, NameThisGame, Gravitar), epochs (4/6/8), and interventions. The first predicts *final human-normalized reward*; the second follows the "Juliani-style" target (Juliani & Ash, 2024), i.e., change from round 1 to end of training. We define round 1 as the subset of each run with $\texttt{global_step} \leq 0.10\ \max(\texttt{global_step})$, and we report its score as the mean episodic return over the final 5% of that subset. For each run we aggregate metric values over the last 5% of steps, standardize predictors within-environment, and include environment/epoch controls for the final-reward model (dropped for the Juliani target).

We report HC3 heteroskedasticity-consistent standard errors, which are recommended when residual variance can differ across runs and leverage may vary (Long & Ervin, 2000).[2]

[2]We aggregate to one observation per run (env$\times$seed$\times$intervention), so within-run dependence is removed.

To address multiple comparisons across metric coefficients, we control the false discovery rate using the Benjamini–Hochberg (BH) procedure at q=0.05, which provides powerful inference while bounding the expected proportion of false discoveries (Benjamini & Hochberg, 1995; Fan, 2016).

What we report. Below we explain each column shown for the GLM table so it's clear how to read the results.

- $\beta_{\textbf{std}}$ — Standardized GLM coefficient (features are z-scored, i.e. $x \mapsto (x - \mu_x)/\sigma_x$ so each predictor has mean 0 and standard deviation 1). Bigger $|\beta_{\text{std}}|$ means a stronger effect. Positive means higher metric $\Rightarrow$ higher normalized reward (all else equal).
- q — FDR–adjusted p-value (Benjamini–Hochberg). Smaller is better. We treat $q < 0.05$ as statistically significant.
- $\Delta R^2_{\textbf{adj}}$ — Drop in adjusted R^2 if we remove this metric from the model:
$$\Delta R^2_{\text{adj}} = R^2_{\text{adj}}(\text{full}) - R^2_{\text{adj}}(\text{minus this metric}).$$
Larger values mean the metric adds more unique explanatory power.
- **Rank** — Importance rank (lower is better), combining $|\beta_{\text{std}}|$ and ΔR^2_{adj}; ties are broken by smaller q.

Main findings Across both targets, higher **policy PCA rank** and non-trivial **policy variance** are the strongest signals: they have the largest standardized coefficients in our GLMs. Signals about spectrum shape line up too—**lower top eigenvalue** (λ_N) helps, and **stronger second eigenvalue** (λ_2) is modestly good—while simple **scale** terms (e.g., critic weight norm) only add a little once rank and variance are in. Capacity/instability markers like **capacity loss** and **dead units** hurt or add little after the main signals. *Why?* The visuals suggest a simple mechanism: higher policy rank $\approx$ richer features; higher action variance $\approx$ healthier exploration; together they lift reward. Large λ_N and capacity loss look like instability or collapse that the GLM learns to down-weight. In practice, if you can only track a few things, log **policy PCA rank**, **policy variance**, and a **simple stability proxy** (λ_2/λ_N or capacity loss); the rest contribute marginal gains once these are present.

Table 7: Final–reward GLM: top predictors (standardized design). Lower q and higher ΔR^2_{adj} indicate stronger signal.

Metric (short)	β_{std}	q	ΔR^2_{adj}
Policy PCA rank	0.183	1.0×10^{-10}	0.0465
Critic weight norm	0.084	1.9×10^{-11}	0.0201
Policy variance	0.077	1.7×10^{-9}	0.0178
λ_2 (policy)	0.083	1.2×10^{-4}	0.0151
λ_N (policy)	-0.065	1.4×10^{-7}	0.0145
Capacity loss (policy)	-0.061	6.0×10^{-9}	0.0139
Value PCA rank	-0.062	3.2×10^{-5}	0.0138

Table 8: Juliani-style GLM (final $-$ round 1): top predictors. Coefficient scales differ from the final-reward model; compare ΔR^2_{adj}.

Metric (short)	β_{std}	q	ΔR^2_{adj}
Policy PCA rank	942.19	2.0×10^{-6}	0.0374
Policy variance	596.30	7.0×10^{-6}	0.0328
Critic weight norm	506.04	2.3×10^{-4}	0.0221
λ_N (policy)	-441.17	5.5×10^{-4}	0.0190
λ_2 (policy)	406.52	1.42×10^{-2}	0.0095
Capacity loss (policy)	-317.59	1.00×10^{-2}	0.0109
Dead neurons (policy)	-330.44	5.95×10^{-2}	0.0056

C Breaking Down Batch Size Improvements

Bigger batches can help for two main reasons: more *parallel environments* (#Envs) give more diverse data per update, and larger *minibatches* (MB) reduce gradient noise. Longer *rollouts* (R) and extra

epochs (E) can also change the outcome. To separate these factors, we run the matched settings in Table 9. The **Baseline** uses 8 envs, MB=256, R=128 with $E \in \{4, 6, 8\}$. **Larger Batch Size** scales both diversity and averaging (64 envs, MB=2048, same R); **Larger Batch Size, 4** is the half step (32 envs, MB=1024). To isolate the minibatch effect at fixed diversity, **Larger Batch Size, MB4** keeps 8 envs and R=128 but sets MB=1024; **MB4, R4** lengthens the rollout to R=512; and **MB8, R8** pushes MB and R further (MB=2048, R=1024) with 8 envs. We compare both reward and plasticity metrics across these matched runs, holding the overall training budget (env steps) fixed, to see whether gains track MB (noise reduction), #Envs/R (data diversity), or E (data reuse).

Experiment	Epochs (E)	Envs	Minibatch (MB)	Rollout per Env (R)	Gradient Steps
Baseline	4, 6, 8	8	256	128	4
Larger Batch Size	4, 6, 8	64	2048	128	4
Larger Batch Size, 4	4, 6, 8	32	1024	128	4
Larger Batch Size, MB4	4, 6, 8	8	1024	128	1
Larger Batch Size, MB4, E4	16, 24, 32	8	1024	128	1
Larger Batch Size, MB4, R4	4, 6, 8	8	1024	512	4
Larger Batch Size, MB8, R8	4, 6, 8	8	2048	1024	4

Table 9: Batch-size ablations configuration summary

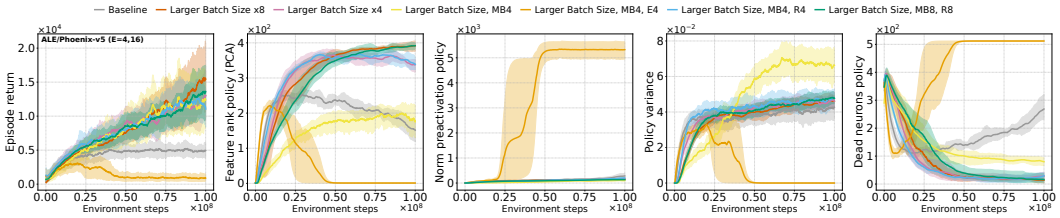


Figure 7: Plasticity metrics tracked for the ablation on bigger batch size, Phoenix, 4 and 16 epochs.

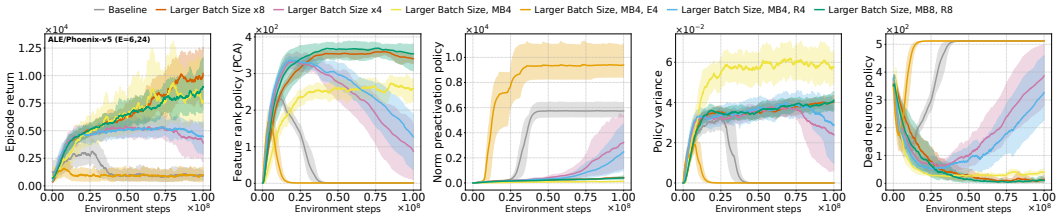


Figure 8: Plasticity metrics tracked for the ablation on bigger batch size, Phoenix, 6 and 24 epochs.

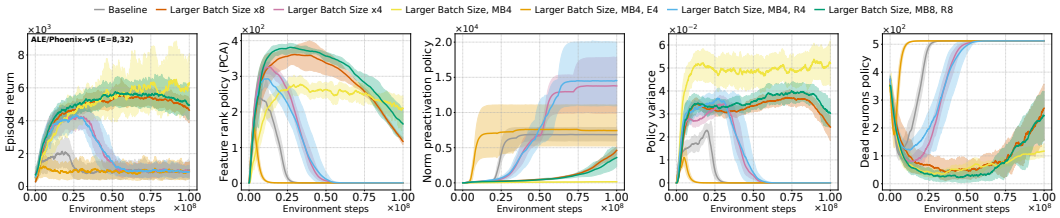


Figure 9: Plasticity metrics tracked for the ablation on bigger batch size, Phoenix, 8 and 32 epochs.

What changes with each factor?

1. **Minibatch size positively affects training.** Increasing the minibatch size to 1024 from the default value of 256 leads to better perfomance in low epoch runs and slower collapse in

higher epoch ones (Figures 7 and 9). The same trend holds when comparing runs with mini-batch size 1024 and 2048.

2. **The way batch size is increased doesn't matter** Batch size can be either increased by adding new environments or extending the rollout lengths. To test the way each of these increasing methods works, we can consider pairs (Larger Batch Size, 4; Larger Batch Size, MB4, R) and (Larger Batch Size; Larger Batch Size, MB8, R8) (see Table 9). In each of them, the runs have the same minibatch size and number of gradient steps per batch, but they differ in the way the batch size is increased. From our results, we see that no matter the increasing method, the performance and collapse trends are similar (Figures 7, 8 and 9).

Higher data-reuse setting (E=8). At the same 10^8 env steps but with more reuse per update (E=8), the differences sharpen. **MB4** remains stable when looking at plasticity metrics, whereas both **MB4, R4** and the **half Larger Batch Size** collapse. Even the full **larger batch** and **MB8, R8** partially degrade at the end of the curve and when looking at the feature rank policy (PCA). These effects of high epochs are even sharper when looking at the MB4-E4 curves when none of them really converge and where the collapse happens at the really beginning of the training. This points to a simple story: longer rollouts and heavier reuse can hurt plasticity under the same interaction budget, and the most reliable stabiliser is the larger minibatch (better gradient averaging), not just more diverse or longer trajectories.

Takeaway. Bigger batches improve plasticity *mainly* because the **minibatch is larger** (better gradient averaging). In our matched-budget runs, MB-only (#Envs fixed) reproduces most of the gains seen with the full larger-batch setting and stays robust at the end of training.

D Study and methods details

D.1 Experimental setup

We follow the experimental protocol in Moalla et al. (2024) and train PPO on Phoenix, NameThisGame and Gravitar from the Arcade Learning Environment (ALE). Following Aitchison et al. (2023), we selected NameThisGame (Atari-1) and Phoenix (the next one to form Atari-3) because together they achieve a high predictive correlation to the full 57-game suite, and include Gravitar to form the third environment, as a sparse-reward environment which among sparse ALE games has the highest correlation with full-suite performance, making it a representative sparse task. Furthermore, we want to see if the claims of Moalla et al. (2024) regarding the critic collapsing before the actor in this environment hold for our interventions, which we apply to both the actor and the critic, in contrast to Moalla et al. (2024), who only apply the interventions to the actor. In this protocol, we add stochasticity to transitions, in an otherwise deterministic ALE, by repeating the previous action independently of the action that the agent played with probability 25% (sticky actions). We use three seeds for each hyperparameter configuration and show the average result with a shaded area bounded by the minimum and maximum values. Actor and critic have separate trunks, because shared trunks lead to interference between policy and value objectives, which can hurt performance (Cobbe et al., 2021). Moreover, we observe that the baseline converges in all environments, so our training budget is sufficient to train two trunks. The default number of epochs for the ALE benchmark is 4 (Huang et al., 2022), and to amplify the effect of non-stationarity, we run each experiment with 4, 6, and 8 epochs. Runs with 6 and 8 epochs tend to have a worse performance, but the collapse in them happens faster, allowing us to analyze the degradation of metrics within our sampling budget. A more rapid collapse with the increased number of epochs can be explained by overfitting, so we expect to see similar dynamics as in the works that studied training in an overfitting regime (Moalla et al., 2024; Nikishin et al., 2022).

D.2 Interventions

We organize interventions according to the three categories (1) data (input), (2) auxiliary losses (output), and (3) normalization layers (architecture), and include a fourth category for combinations of normalization layers with auxiliary losses. Each intervention is evaluated against our plasticity metrics under a unified PPO framework. In contrast to Moalla et al. (2024), we apply the interventions to both the actor and the critic, which have separate networks. By developing the auxiliary-loss

taxonomy in Figure 1, we reveal a large region of unstudied loss functions and introduce a new CHAIN-based intervention, CHAIN-SP. We provide a comprehensive experimental design and the complete set of individual result plots in Appendix F, G. Table 2 presents a summary of the methods we test, with detailed experimental descriptions available in Appendix F. Figure 1 presents the summary boxplots, showing the effect of each intervention on episode return, feature rank (PCA), capacity loss, and the number of dead neurons in the policy.

D.3 Hyperparameter Tuning

We use a rigorous, multi-step protocol to tune the actor and critic networks' hyperparameters for each auxiliary-loss intervention (Table 2). First, we optimize the actor hyperparameters over a logarithmically spaced range (8–10 values) on the Phoenix environment for 6 epochs (3×10^7 steps), while keeping the critic hyperparameters fixed. After identifying the optimal actor hyperparameters and verifying its stability over an extended training period (1×10^8 steps), we tune the critic hyperparameters using a similar approach. To ensure generality, we refine the actor settings by testing values right above and below the optimum across three distinct environments, and confirmed robustness using multiple random seeds. Full details of the tuning procedure are provided below:

D.3.1 Hyperparameter tuning protocol

In order to robustly tune the hyperparameters of **each auxiliary-loss** interventions, we followed a multi-step protocol. First, we tuned the actor network hyperparameters while keeping the critic hyperparameters fixed. Then, we performed a complementary tuning of the critic hyperparameters using the optimal actor settings. The detailed protocol is as follows:

1. **Initial Actor Hyperparameter Selection.** We selected a broad range of possible values for the actor network hyperparameters. They are 8-10 consecutive powers of 10. In this initial phase, the critic hyperparameter was fixed to 0.

2. **Actor Hyperparameter Tuning on Phoenix.** The actor hyperparameters were evaluated on the Phoenix environment over 6 epochs, corresponding to approximately 3×10^7 steps. The best-performing actor hyperparameter was identified based on our performance metric. To ensure that the optimal value was not an artifact of the chosen range, if the optimum was observed at the boundary of the range, the range was extended and the new values were also evaluated and thus up until finding an optimal value not on the boundary of the range.

3. **Extended Stability Verification.** Using the best actor hyperparameter identified in Step 2, we run a longer training for 1×10^8 steps with a single seed. This is to ensure that the performance did not collapse over longer training horizons.

4. **Critic Hyperparameter Tuning.** With the actor hyperparameter fixed to the optimal value from Step 2, we then selected a broad range of candidate values for the critic hyperparameter, again on a logarithmic scale. Similar to Step 2, experiments were run on the Phoenix environment for 6 epochs (up to 3×10^7 steps), and the best critic hyperparameter was determined based on the performance metric. As before, if the optimum was located at the range boundary, we then extended the range and tested the new values.

5. **Multi-Environment Evaluation.** In order to really optimize the actor hyperparameter, we consider 1–2 values immediately below and above the optimal value identified in Step 2 (again in logarithmic scale). The new actor hyperparameters were tested with the optimal critic hyperparameter (from Step 4) across 3 distinct environments. In each environment, training was conducted for 6 epochs (up to 3×10^7 steps) to test generality.

6. **Final Hyperparameter Selection.** The hyperparameter configuration that consistently gave superior performance across all 3 environments was selected. In the ideal case, a single hyperparameter value is optimal for all environments.

7. **Robustness and Generalization Test via Multiple Seeds.** Finally, complete runs were conducted using multiple random seeds to evaluate the reproducibility, robustness and generalization of the selected hyperparameter configuration.

D.4 Additional details on our experimental setup

We use the hyperparameters detailed in Table 10 when training the baseline PPO agent in ALE.

E Metrics details

E.1 Dead neurons

We call a neuron dead if it does not activate any sample from the batch. Because in all our modifications we use ReLU activations, that is equivalent to all of the pre-activations of the neuron in the batch being non-positive, as proposed by Gulcehre et al. (2022).

E.2 Feature rank

In deep reinforcement learning, the activations of the penultimate layer are treated as the network's *features*, which on a batch of N states form a matrix $\Phi \in \mathbb{R}^{N \times D}$ with $D < N$ Kumar et al. (2021); Lyle et al. (2022); Gulcehre et al. (2022); Andriushchenko et al. (2023). Various *feature rank* metrics quantify the "quality" of Φ by examining its singular values $\{\sigma_i(\Phi)\}_{i=1}^{D}$, including both *relative* measures (e.g., approximate rank via PCA) and *absolute* measures (e.g., counting singular values above a threshold) Lyle et al. (2022). The *approximate rank* (PCA) is defined as

$$\mathrm{rank}_{\mathrm{PCA}}(\Phi) \;=\; \min_k\Big\{\, k : \sum_{i=1}^{k} \sigma_i(\Phi)^2 \;>\; (1-\delta) \sum_{j=1}^{D} \sigma_j(\Phi)^2 \Big\},$$

with $\delta = 0.01$ to retain 99% of the variance Andriushchenko et al. (2023); Yang et al. (2020). An example of an *absolute* rank metric is the *Feature Rank* of Lyle et al. (2022), defined as

$$\big|\{\, i : \sigma_i(\Phi)/\sqrt{N} > \delta\}\big|,$$

while the *PyTorch rank* counts indices i for which $\sigma_i(\Phi)/\big(\sigma_1(\Phi)\sqrt{N}\big) > \varepsilon$ (e.g., $\varepsilon = 1.19 \times 10^{-7}$ Lyle et al. (2022). Absolute and relative rank measures, while yielding different numerical values, exhibit highly correlated temporal trajectories that fall into two distinct clusters, as demonstrated by Moalla et al. (2024) in Appendix E.

Furthermore, in the same study, Moalla et al. (2024) reveals that PPO agents also experience *feature rank deterioration*—a decline in $\mathrm{rank}_{\mathrm{PCA}}(\Phi)$ over time—driven by non-stationarity in policy optimization Moalla et al. (2024). They show that as $\mathrm{rank}_{\mathrm{PCA}}(\Phi)$ decreases, PPO's heuristic trust region degrades, ultimately causing *performance collapse* even when the critic remains strong Moalla et al. (2024).

E.3 Capacity loss

Capacity loss—often called target-fitting capacity (see Lyle et al. (2022))—is evaluated on intermediate checkpoints of a network during training to track how its ability to match a fixed, externally defined target changes over time. In other words, it provides a concrete measure of the model's plasticity. Concretely, given a pre-specified target distribution (over inputs and outputs) and a fixed number of optimization steps, the capacity loss at a particular checkpoint is simply the loss incurred when that checkpoint is trained (within the allotted budget) to reproduce the target.

In deep RL, one typically measures an agent's capacity by asking it to fit the outputs of another model whose parameters were sampled from the same initialization distribution as the agent, using data collected from a rollout generated by this "random" model ((Lyle et al., 2022); (Nikishin et al., 2023)). We adopt this approach here, so that the data used for fitting comes—on average—from the same distribution as the agent's initial checkpoint. When performing the fit, the critic is trained by minimizing an L^2 (mean-squared) loss between its outputs and the target's outputs, whereas the actor is trained by minimizing the forward Kullback–Leibler divergence between the target policy and the checkpoint's policy.

Table 10: Hyperparameters used when training the baseline for ALE.

Environment	
Repeat action probability (Sticky actions)	0.25
Frameskip	3
Max environment steps per episode	108,000
Noop reset steps	0
Observation transforms	
Grayscale	True
Resize width (‘resize_w‘)	84
Resize height (‘resize_h‘)	84
Frame stack	4
Normalize observations	False
Reward transforms	
Sign	True
Collector	
Total environment steps	100,000,000
Num envs in parallel	8
Num envs in parallel capacity	1
Agent steps per batch	1.024 (128 per env)
Total agent steps capacity	36,000 (at least one full episode)
Models (actor and critic)	
Activation	ReLU
Convolutional Layers	
Filters	[32, 64, 64]
Kernel sizes	[8, 4, 3]
Strides	[4, 2, 1]
Linear Layers	
Number of layers	1
Layer size	512
Optimization	
Advantage estimator	
Advantage estimator	GAE
Gamma	0.99
Lambda	0.95
Value loss	
Value loss coefficient	0.5
Loss type	L2
Policy loss	
Normalize advantages	minibatch normalization
Clipping epsilon	0.1
Entropy coefficient	0.01
Optimizer (actor and critic)	
Optimizer	Adam
Learning rate	0.00025
Betas	(0.9, 0,999)
Max grad norm	0.5
Annealing linearly	False
Number of epochs	4, 6, 8
Number of epochs capacity fit	1
Minibatch size	256
Logging (% of the total number of batches)	
Training	every 0.1% (~100,000 env steps)
Capacity	every 2.5% (41 times in total)

$$L_{\theta_{\mathrm{cp}}} = \mathbb{E}_{(s,a)\sim\pi_{\theta_0}}\big[\mathrm{KL}(\pi_{\theta_{\mathrm{cp}}}(\cdot \mid s) \,\|\, \pi_{\theta_0}(\cdot \mid s))\big], \quad L_{\phi_{\mathrm{cp}}} = \mathbb{E}_{(s,a)\sim\pi_{\theta_0}}\big[(V_{\varphi_{\mathrm{cp}}}(s) - V_{\varphi_0}(s))^2\big],$$

$$L^{\mathrm{cp}} = L_{\theta_{\mathrm{cp}}} + L_{\phi_{\mathrm{cp}}}$$

Capacity loss for the checkpoint cp, L^{cp}, is optimized the same way as PPO using minibatches. For each rollout we do 1 epoch of optmization.

Smaller values of L_{cp} indicate that the checkpoint cp has preserved more plasticity.

E.4 Discussion about other possible metrics.

Fisher Information Matrix (FIM) A useful extra metric is the Fisher Information Matrix (FIM), which measures how sensitive the network is to parameter changes through the covariance of the score (the gradient of the log-likelihood). Prior work reports a simple signal in the *trace* of the empirical FIM during training: an early sharp rise ("memorization") followed by a decline ("reorganization"). The switch between the two can be picked up by looking at a smoothed time-derivative of the trace. (Falzari & Sabatelli, 2025) We did not track FIM here, but adding it is practical: an EKFAC-style approximation makes it cheap enough to log per layer or globally. In PPO, we would compute the *policy FIM* as the on-policy empirical Fisher, i.e., the batch average of $\nabla_\theta \log \pi_\theta(a \mid s)\, \nabla_\theta \log \pi_\theta(a \mid s)^\top$; compute the *critic FIM* by modeling $V_\phi(s)$ as the mean of a Gaussian with fixed variance σ^2 and using its Gauss–Newton/Fisher $\mathbb{E}\big[\sigma^{-2}\nabla_\phi V_\phi(s)\nabla_\phi V_\phi(s)^\top\big]$ (or the empirical form $((y - V_\phi(s))/\sigma^2)^2\, \nabla_\phi V_\phi(s)\, \nabla_\phi V_\phi(s)^\top$), averaged over the batch. This would complement our current rank, dead neurons and norm metrics by adding a curvature view. Practically, the expected overhead is about 10–20% of training time, and prior results suggest consistent actor–critic differences (critic traces often about an order of magnitude larger) and sensitivity to the amount of data reuse (replay/epoch ratio). (Falzari & Sabatelli, 2025).

Different types of rank. We track several rank metrics (PCA rank, `torch` algebraic rank, stable/s-rank, feature rank, and effective/Vetterli rank), following Moalla et al. (2024); see their Appendix E for definitions and details. They often differ because each summarises the singular-value spectrum differently. For example, the *stable rank* $\mathrm{srank}(A) = \|A\|_F^2/\|A\|_2^2$ is steady under tiny singular values and changes smoothly as the spectrum decrease, so it works well as a noise-robust size measure. By contrast, the algebraic rank can jump when one singular value passes a threshold. The *effective rank* $\mathrm{erank}(A) = \exp(H(p))$ with $p_i = \sigma_i/\sum_j \sigma_j$ tells how evenly the singular values are spread; it usually lies between 1 and the algebraic rank (Roy & Vetterli, 2007). Our case study already covers a broad set of rank metrics. As optional additions—useful if one wants extra nuance—it could help to log two simple stable-rank variants that *complement* (not replace) what we have:

1. A **normalised** stable rank, $\mathrm{srank}(A)/\min\{m,n\}$, to put layers of different sizes on the same scale. (Sanyal et al., 2019)
2. A p**-stable rank**, $\mathrm{sr}_p(A) = \sum_i (\sigma_i/\sigma_{\max})^p$. Here p=2 is the usual stable rank; p=1 is more sensitive to the tail. (Ipsen & Saibaba, 2024)

Normalising stable rank would put layers on a common scale for clearer cross-layer and cross-run plots; logging both $\mathrm{sr_1}$ (tail-sensitive) and srank (p=2, more top-eigenvalue-oriented) would help separate "spike growth" from "tail filling," and both are cheap to add since they only require $|A| * F$ and $\sigma_{\max}$ (via a short power iteration). We do not include them by default because our current set already captures the spectrum trends we study; these variants add finer detail but bring diminishing returns, so we list them as optional complements for future ablations.

F Interventions

In this section, we provide some details on the interventions tested. We apply all modifications to both the actor and the critic.

F.1 Data-level modifications

Larger batch size We examine the impact of batch size on training dynamics. We increase it by a factor of 8, but keep the number of gradient updates the same by increasing the minibatch size by 8 as well. The only change for BATCH×8 is that we collect eight times more transitions before each optimization round by using eight times as many parallel environments and keep the number of gradient steps constant by correspondingly using an eight-times larger minibatch, while maintaining the same learning rate. The result is therefore more data per policy update round, the same number of gradient updates per update round, but a lower number of policy update rounds overall. We hypothesize that the noise in gradient updates would decrease, and thus the training will be more stable.
RGB inputs unscaled To further investigate the richness of feature representations, we use colored pixels inputs without image resizing compared to the baseline. Other image preprocessing parameters are left the same.
No reward clipping. By default, we replace the reward by its sign. In this regularization, however, we use the raw reward for training. We do that to test how an unpreprocessed signal affects training.

F.2 Loss function modifications

Regenerative Regularization We experiment with regenerative regularization (L2 Init) as proposed in Kumar et al. (2023). We test regenerative regularization alone to evaluate its baseline effectiveness.
PFO. Proximal Feature Optimization (PFO) (Moalla et al., 2024) is a regularization technique for stabilizing representation learning in PPO by constraining pre-activation drift during the policy optimizations stage of the training. This approach mitigates plasticity loss, reduces the number of dead neurons, and enhances feature diversity, effectively addressing performance degradation during training.
InFeR. Initial Feature Regularization (InFeR) (Lyle et al., 2022) mitigates capacity loss by penalizing the distance between auxiliary outputs of current weights and initial weights. This stabilizes feature dynamics, preserving plasticity and improving performance, particularly in sparse-reward environments.
AdamW. AdamW (Loshchilov, 2017) is tested as a regularization technique analogous to regenerative regularization, but instead, it encourages weights to remain close to zero.

F.3 Network architecture modifications

LayerNorm. LayerNorm, introduced by (Ba et al., 2016), is a widely used regularization technique in deep learning. As shown by Lyle et al. (2022) and Juliani & Ash (2024), LayerNorm is effective in mitigating plasticity loss during training for both off- and on-policy reinforcement learning. We also test Layer Normalization without scaling (γ fixed to 1, β fixed to 0).

BatchNorm. BatchNorm (Ioffe, 2015) is tested to assess its comparative performance against LayerNorm. BatchNorm normalizes each layer's pre-activations across the minibatch, reducing internal covariate shift and smoothing the loss landscape. To evaluate its regularizing effect in PPO, we insert BatchNorm layers immediately after each linear transform in both actor and critic networks, keeping the learnable scale (γ) and shift (β) parameters active so that the model can re-adapt normalized activations as needed during training. We also test a variant of BatchNorm with its affine parameters frozen ($\gamma = 1$, $\beta = 0$) to isolate the pure normalization effect—this "no-scale" BatchNorm lets us quantify how much of its benefit comes from variance stabilization alone versus the added flexibility of learnable rescaling.

Unit Ball Normalization. Unit Ball Normalization (UBN) (Hussing et al., 2024) is proposed as a method to mitigate Q-value divergence, providing stable gradients and enhancing performance in high UTD scenarios. To evaluate its effectiveness, we implement UBN for actor models in PPO to investigate its potential in preventing plasticity loss caused by large gradients.

Parseval. Parseval regularization is proposed as a method to mitigate plasticity loss, trainability degradation, and primacy bias by enforcing orthogonality constraints on weight matrices—thereby preserving useful optimization properties and stabilizing training dynamics in continual reinforcement learning settings (Chung et al., 2024). Parseval regularization was initially proposed for linear layer regularization. We extend it to convolutional layers by flattening each convolutional kernel into a matrix. As in the original paper, we do not apply regularization to the last layer of the network.

F.4 Churn based modifications

CHAIN Churn Approximated ReductIoN (CHAIN) is proposed as a method to mitigate the chain effect of value and policy churn, providing more stable predictions and enhancing learning performance across online and offline, value-based and policy-based RL settings (Tang & Berseth, 2024). To evaluate its effectiveness, we implement CHAIN within the actor and critic network of PPO to investigate its potential in preventing plasticity loss caused by large churn-induced parameter updates.

CHAIN-SP (Sampling-Policy) extends CHAIN by applying the churn-approximated reduction not to the post-gradient-update model, but to the policy used during rollouts. In practice, rather than stabilizing the actor network after each gradient step, CHAIN-SP constrains the sampling model—i.e., the policy that interacts with the environment—to minimize output churn relative to its previous rollout version. By enforcing this stability at the rollout stage, CHAIN-SP aims to reduce large shifts in action distributions that can accelerate plasticity loss. This modification is incorporated into the PPO framework by maintaining a separate "rollout" copy of the actor network, applying the CHAIN objective to penalize deviations between the new rollout policy and its immediate predecessor, and using this stabilized policy for environment interactions.

F.5 Combinations of modifications

Recent results show that LayerNorm helps models retain its plasticity. Moreover, combining LayerNorm with other regularizations results in an increase in model effectiveness and mitigation in plasticity loss. For example, in Juliani & Ash (2024), LayerNorm with Regenerative Regularization or Shrink+Perturb were the most successful interventions, retaining plasticity and having good generalization ability Ash & Adams (2020).

We continue this direction of study, by combining LayerNorm with three types of regularizations: **LayerNorm + PFO**, **LayerNorm + BatchNorm No Scale**, **LayerNorm + CHAIN-SP**.

F.6 Where do reset-style methods fit in our taxonomy?

Our taxonomy in Table 1 is only for *auxiliary losses* (component-on-which-the-loss-is-applied $\times$ target). Reset-style methods like **CBP** (Dohare et al., 2021) and **ReDo** (Sokar et al., 2023) are not auxiliary losses: they *re-initialise parameters during training*. Therefore they sit *outside* this taxonomy. They are orthogonal to our three families (data, losses, normalisation): you can combine a reset schedule with any loss-based method in the grid, but resets act via discontinuous parameter changes rather than a loss term. We do not benchmark resets here; our focus is on continuous changes that keep units active.

G Main paper figures on all environments.

G.1 Figure 1 on all environments

Figures 10, 11 and 12 aggregate every run: (1 baseline + 18 interventions) × 3 games × 3 seeds x 3 epochs, in a single set of boxplots.

We present all the modifications together. We follow the details of Moalla et al. (2024), Appendix B.3 to compute the boxplots.

Each box summarises 9 independent runs (3 epoch budgets×3 random seeds). The box spans the inter-quartile range (Q1–Q3); the centre black tick marks the median and the red tick marks the mean. Whiskers extend to the largest (smallest) value within Q3+1.5IQR (Q1–1.5IQR) (which is the default Matplotlib rule). Points beyond the whiskers are plotted as outliers.

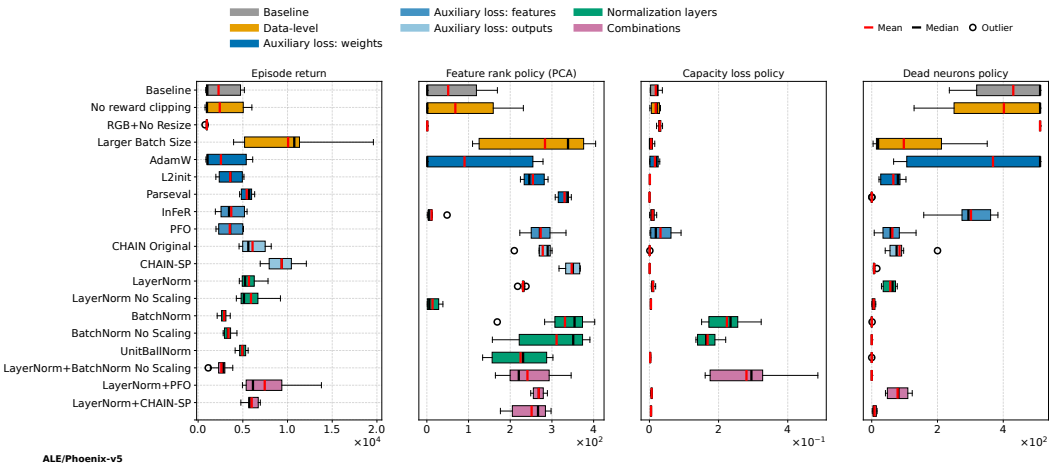


Figure 10: Figure 1 on Phoenix.

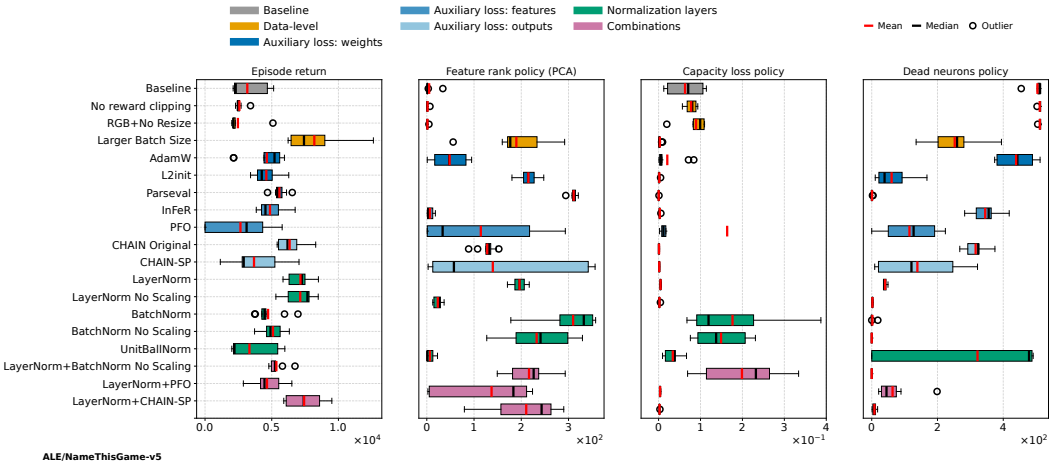


Figure 11: **Figure 1 on NameThisGame.** We observe that LayerNorm and LayerNorm combined with loss based methods mitigate the performance collapse and results in a good policy representation as in Phoenix. Weight based methods perform better than on Phoenix.

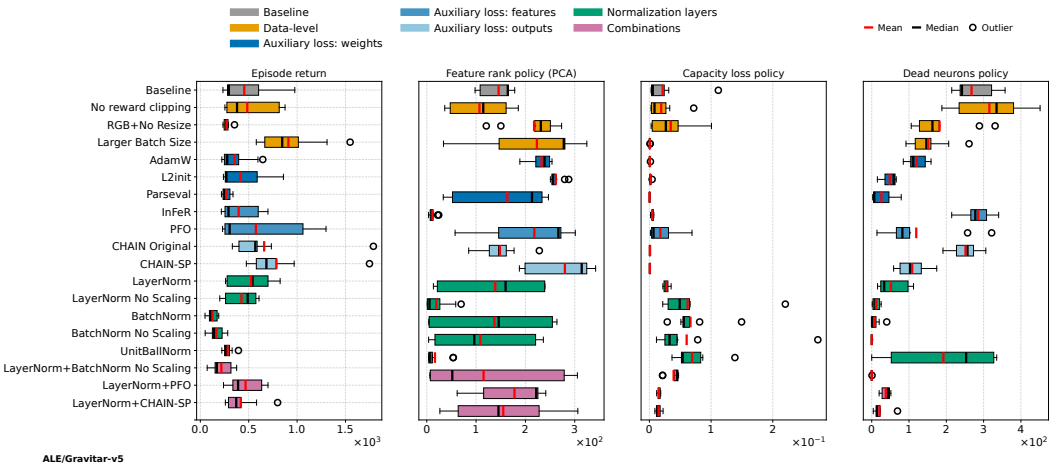


Figure 12: **Figure 1 on Gravitar.** On a sparse environment, we observe that the combinations don't perform (in terms of feature representation metrics) as well as on Phoenix and NameThisGame. Furthermore, CHAIN and CHAIN-SP inverts themselves compared to NTG. PFO has a high variance in terms of episodic return and policy PCA feature rank for sparser environment.

G.2 Figure 2 and 4 with other interventions and number of epochs

G.2.1 Architecture interventions

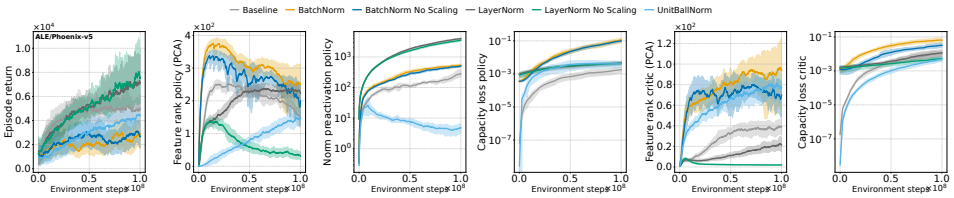


Figure 13: Figure 2, 4 with architecture interventions on Phoenix, 4 epochs

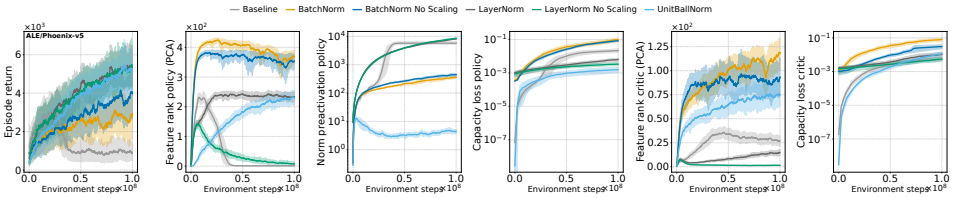


Figure 14: Figure 2, 4 with architecture interventions on Phoenix, 6 epochs

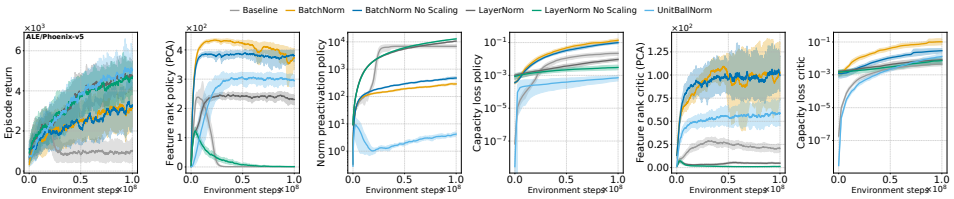


Figure 15: Figure 2, 4 with architecture interventions on Phoenix, 8 epochs

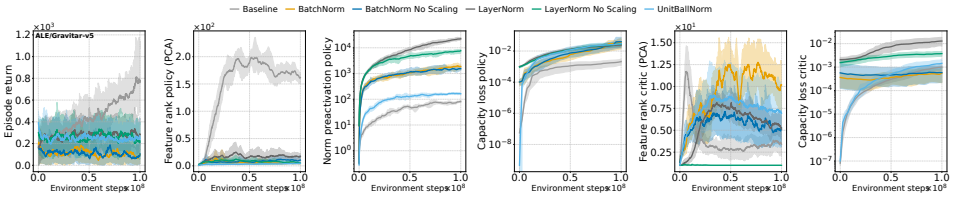


Figure 16: Figure 2, 4 with architecture interventions on Gravitar, 4 epochs

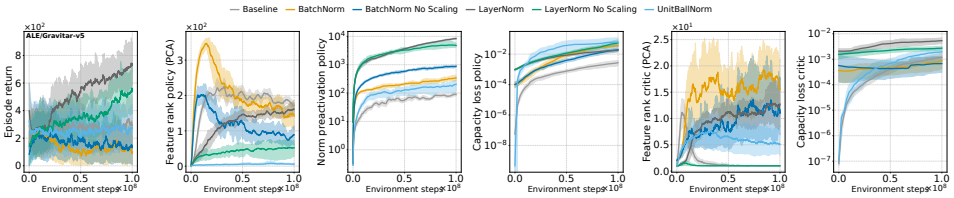


Figure 17: Figure 2, 4 with architecture interventions on Gravitar, 6 epochs

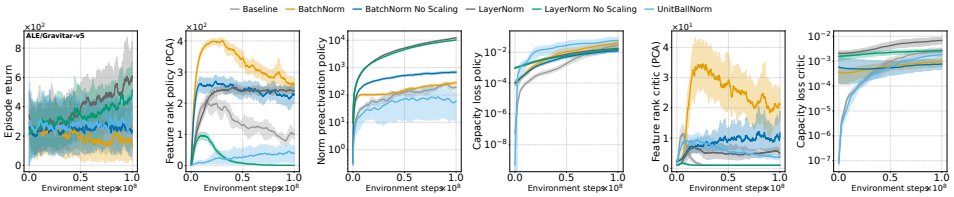


Figure 18: Figure 2, 4 with architecture interventions on Gravitar, 8 epochs

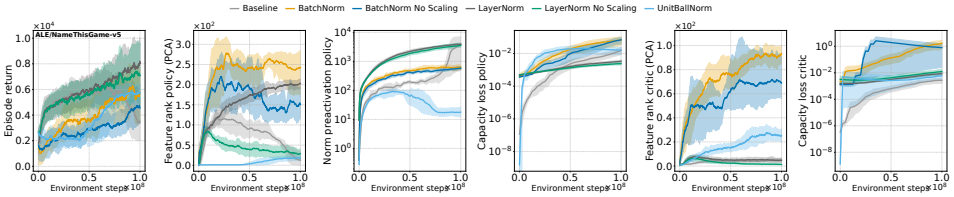


Figure 19: Figure 2, 4 with architecture interventions on Name This Game, 4 epochs

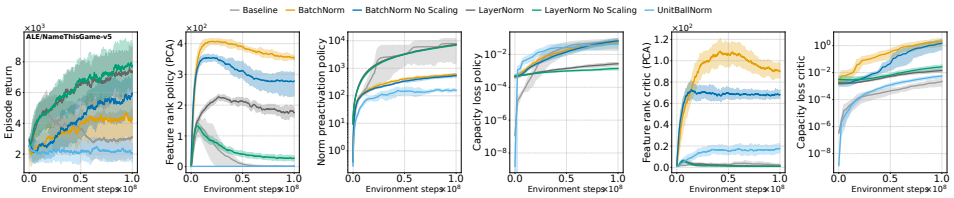


Figure 20: Figure 2, 4 with architecture interventions on Name This Game, 6 epochs

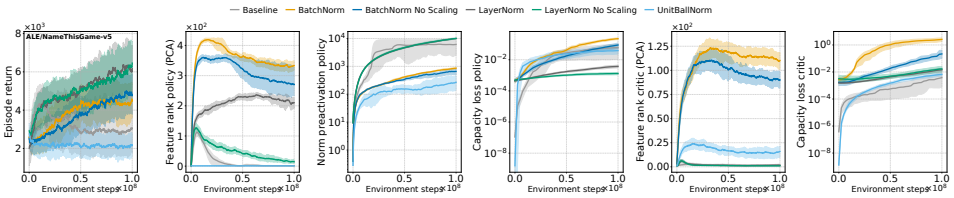


Figure 21: Figure 2, 4 with architecture interventions on Name This Game, 8 epochs

G.2.2 Combination interventions

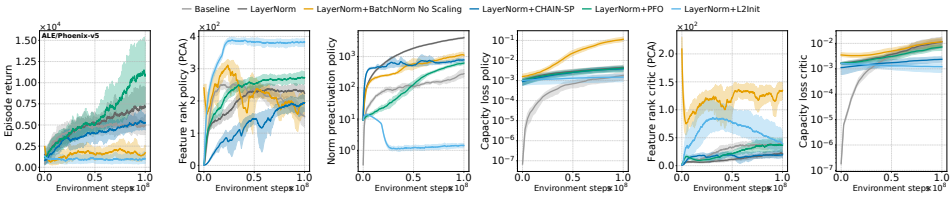


Figure 22: Figure 2, 4 with combination interventions on Phoenix, 4 epochs

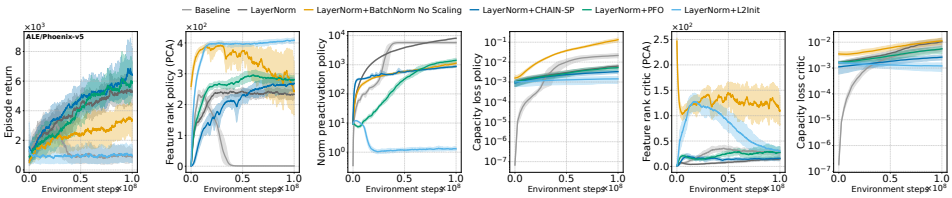


Figure 23: Figure 2, 4 with combination interventions on Phoenix, 6 epochs

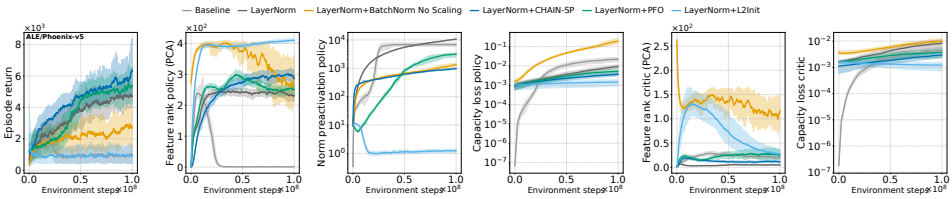


Figure 24: Figure 2, 4 with combination interventions on Phoenix, 8 epochs

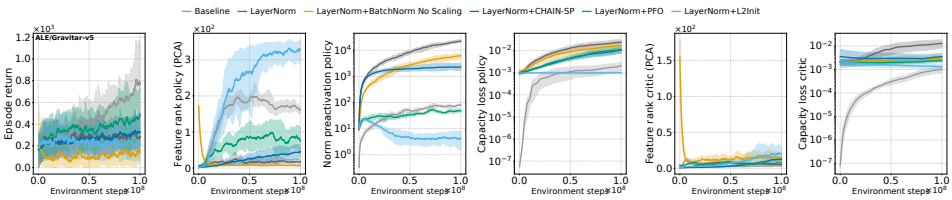


Figure 25: Figure 2, 4 with combination interventions on Gravitar, 4 epochs

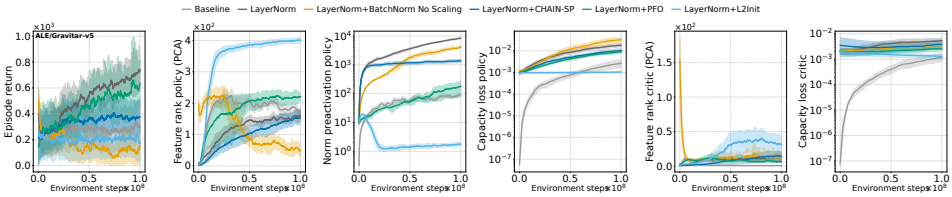


Figure 26: Figure 2, 4 with combination interventions on Gravitar, 6 epochs

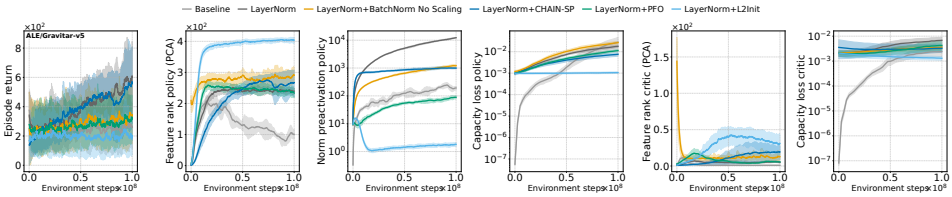


Figure 27: Figure 2, 4 with combination interventions on Gravitar, 8 epochs

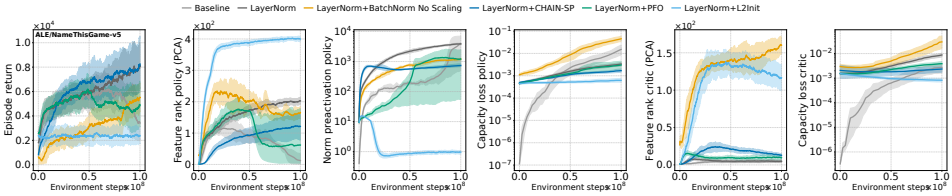


Figure 28: Figure 2, 4 with combination interventions on Name This Game, 4 epochs

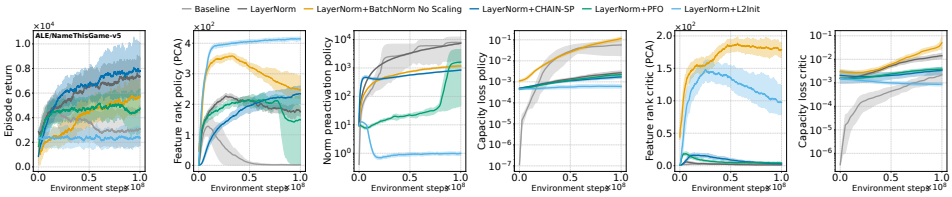


Figure 29: Figure 2, 4 with combination interventions on Name This Game, 6 epochs

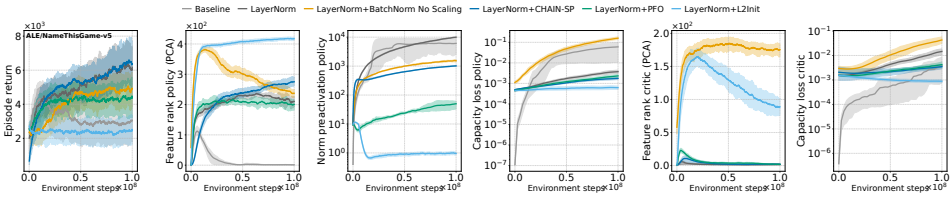


Figure 30: Figure 2, 4 with combination interventions on Name This Game, 8 epochs

G.2.3 Auxiliary-loss interventions

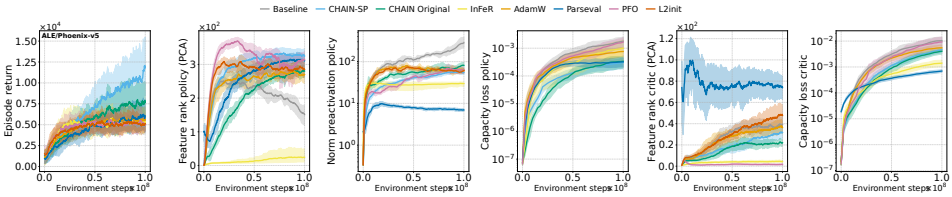


Figure 31: Figure 2, reffig:3-auxiliary with auxiliary loss interventions on Phoenix, 4 epochs

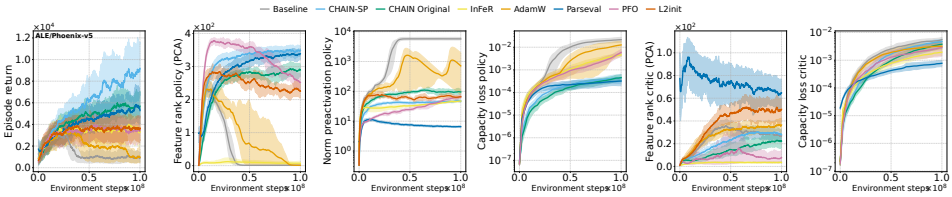


Figure 32: Figure 2, 4 with auxiliary loss interventions on Phoenix, 6 epochs

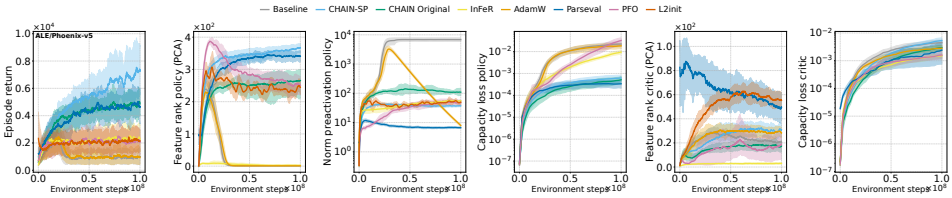


Figure 33: Figure 2, 4 with auxiliary loss interventions on Phoenix, 8 epochs

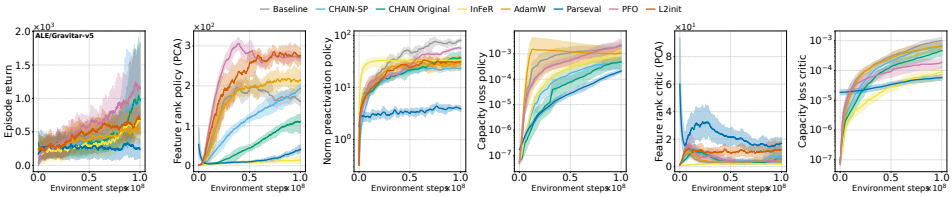


Figure 34: Figure 2, 4 with auxiliary loss interventions on Gravitar, 4 epochs

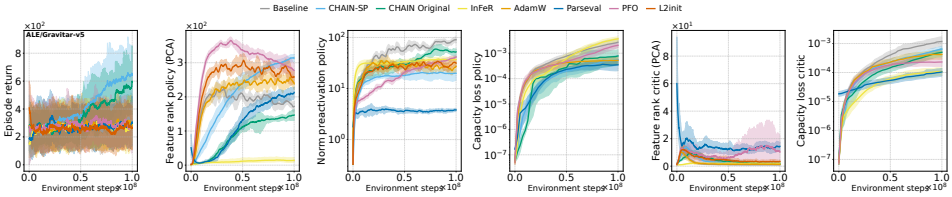


Figure 35: Figure 2, 4 with auxiliary loss interventions on Gravitar, 6 epochs

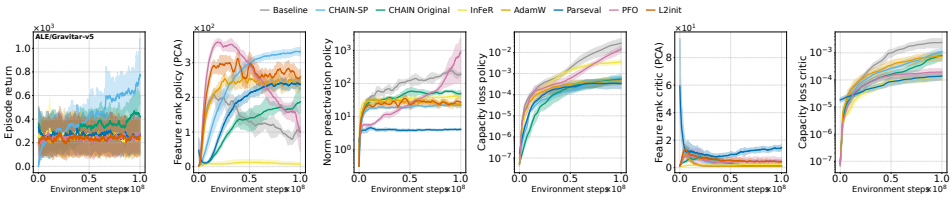


Figure 36: Figure 2, 4 with auxiliary loss interventions on Gravitar, 8 epochs

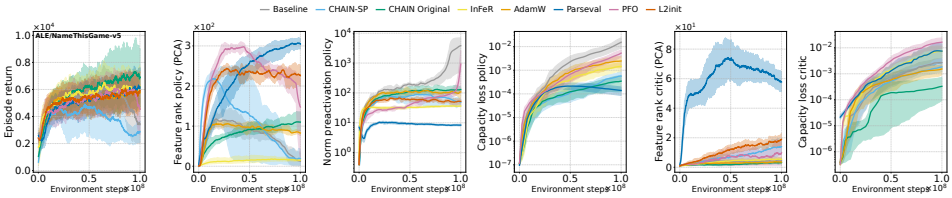


Figure 37: Figure 2, 4 with auxiliary loss interventions on Name This Game, 4 epochs

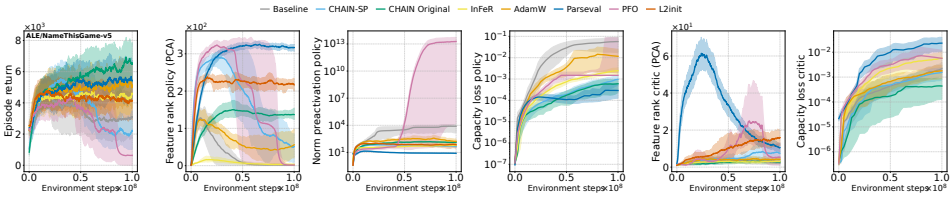


Figure 38: Figure 2, 4 with auxiliary loss interventions on Name This Game, 6 epochs

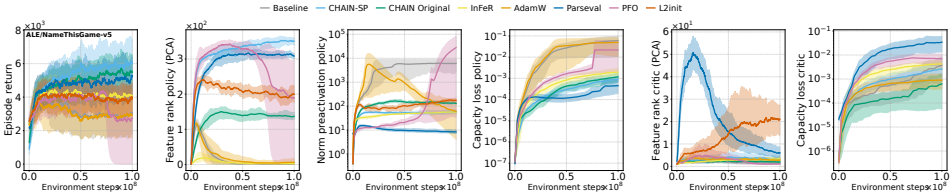


Figure 39: Figure 2, 4 with auxiliary loss interventions on Name This Game, 8 epochs

G.2.4 Data-Level interventions

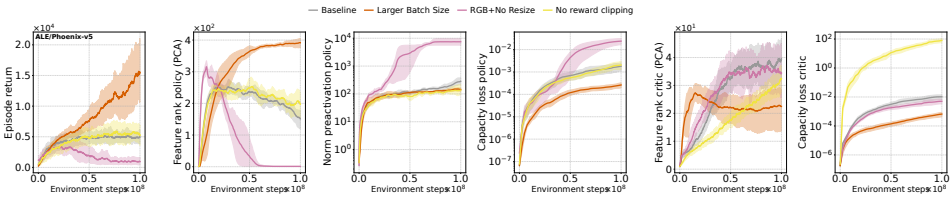


Figure 40: Figure 2, 4 with data-level interventions on Phoenix, 4 epochs

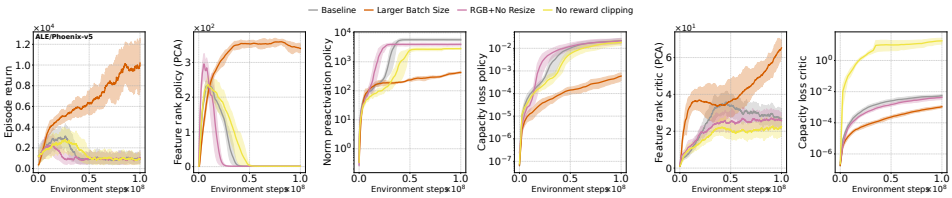


Figure 41: Figure 2, 4 with data-level interventions on Phoenix, 6 epochs

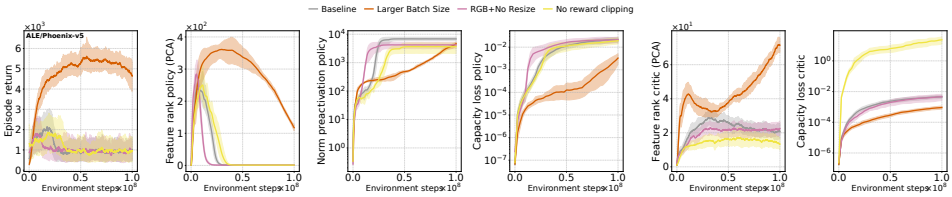


Figure 42: Figure 2, 4 with data-level interventions on Phoenix, 8 epochs

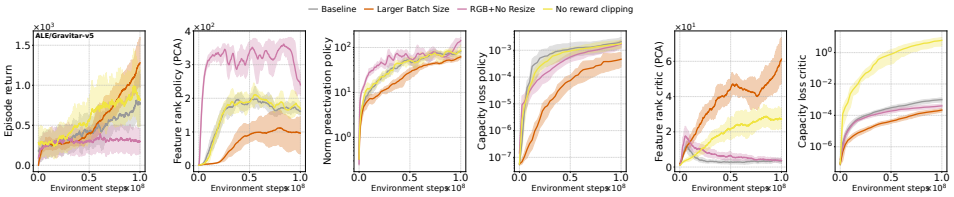


Figure 43: Figure 2, 4 with data-level interventions on Gravitar, 4 epochs

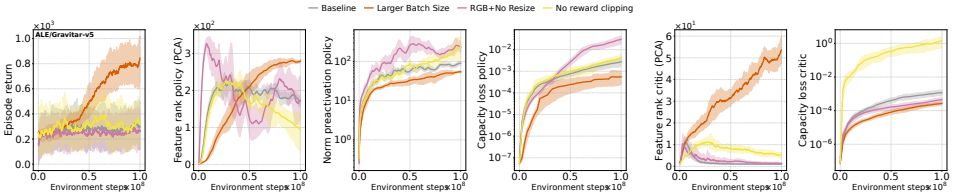


Figure 44: Figure 2, 4 with data-level interventions on Gravitar, 6 epochs

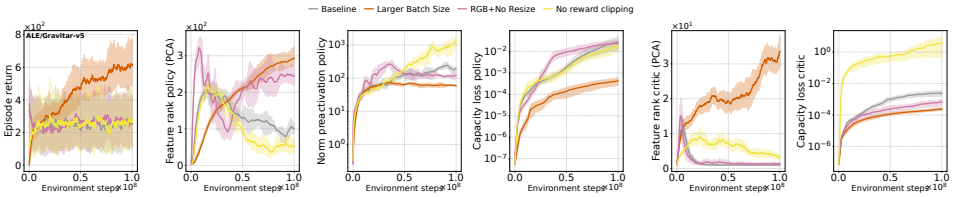


Figure 45: Figure 2, 4 with data-level interventions on Gravitar, 8 epochs

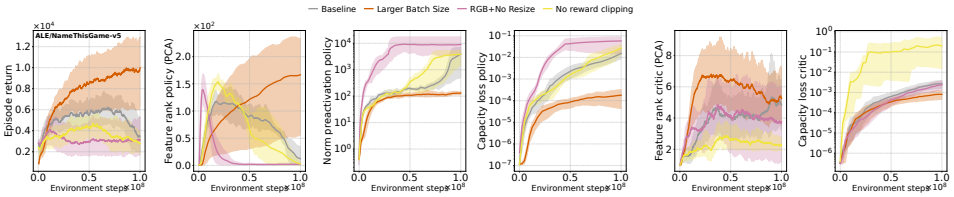


Figure 46: Figure 2, 4 with data-level interventions on Name This Game, 4 epochs

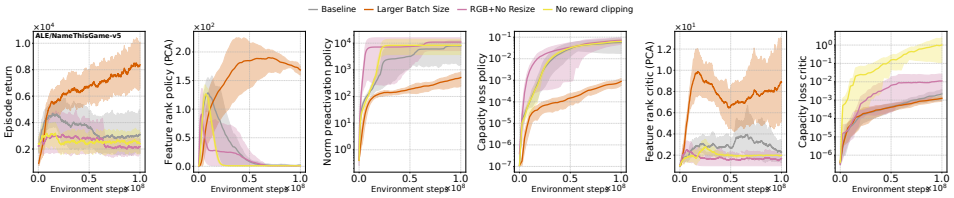


Figure 47: Figure 2, 4 with data-level interventions on Name This Game, 6 epochs

G.3 Figure 5 on other environments

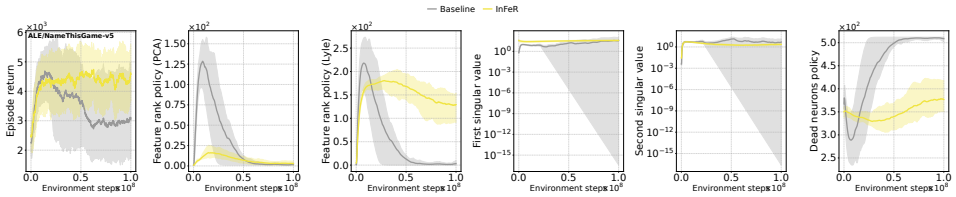


Figure 48: Figure 2, 5 with data-level interventions on Name This Game, 6 epochs

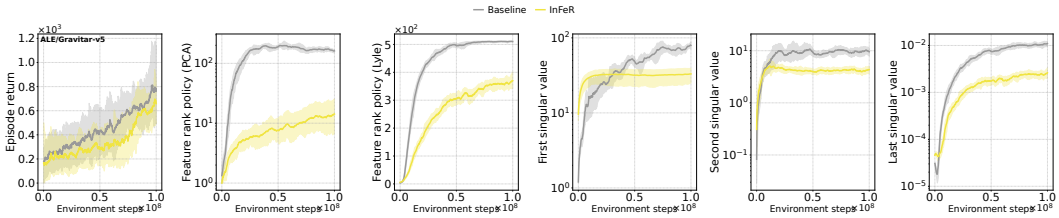


Figure 49: Figure 2, 5 with data-level interventions on Gravitar, 6 epochs

H Extra figures.

H.1 Extended training runs

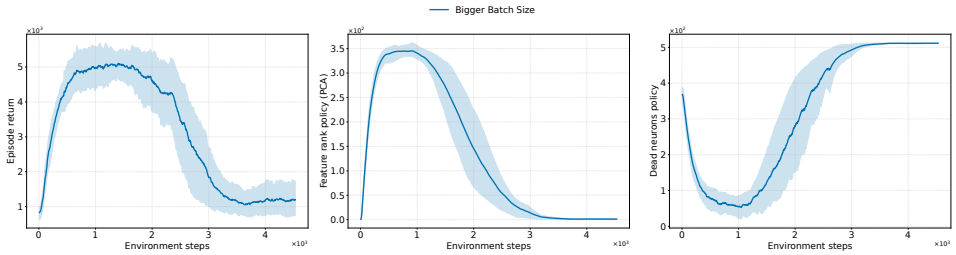
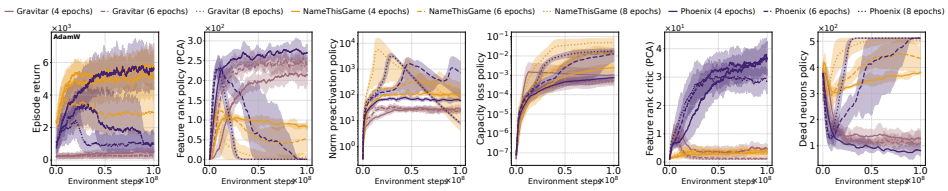
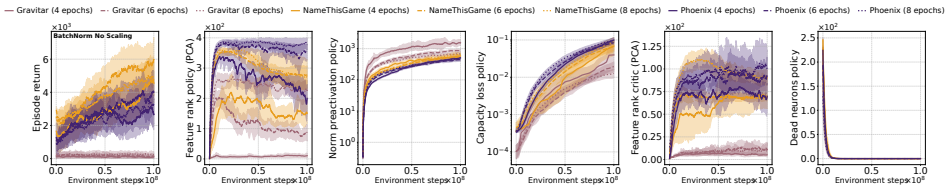
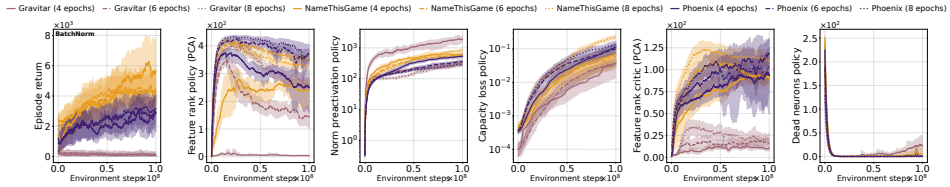


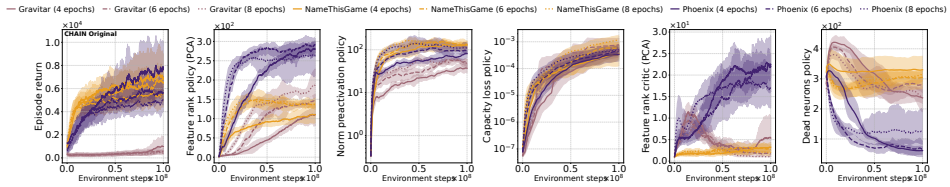
Figure 50: With an 8-epoch training budget and 200m steps, the agent's performance peaks early but then collapses, showing the delayed collapse observed with smaller batch sizes.

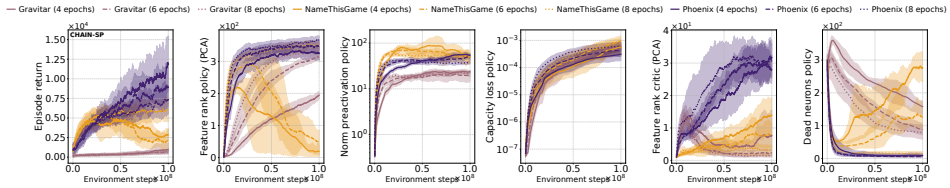
H.2 Plasticity figures

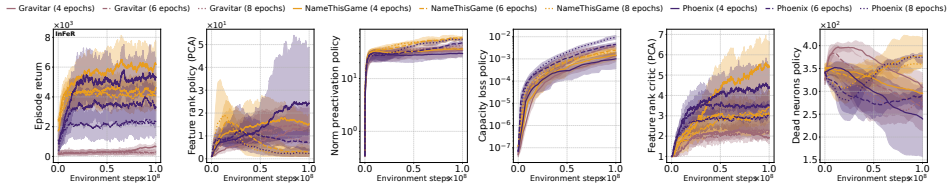


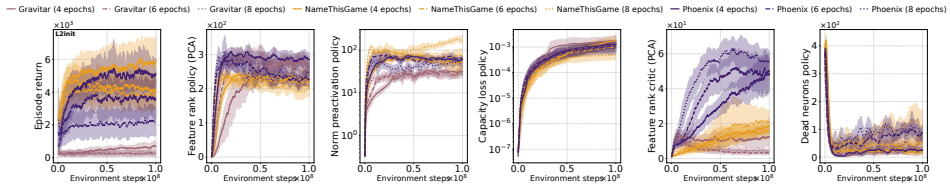


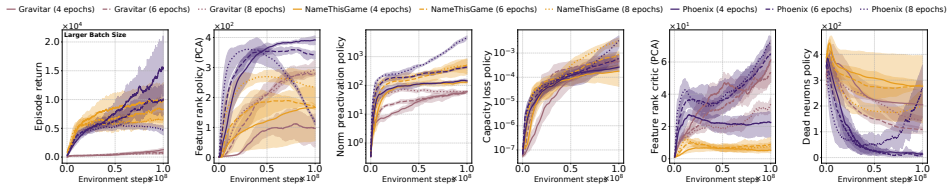

Gravitar (4 epochs)
Gravitar (6 epochs)
Gravitar (8 epochs)
NameThisGame (4 epochs)
NameThisGame (6 epochs)
NameThisGame (8 epochs)
Phoenix (4 epochs)
Phoenix (6 epochs)
Phoenix (8 epochs)
BatchNorm
Episode return
Feature rank policy (PCA)
Norm preactivation policy
Capacity loss policy
Feature rank critic (PCA)
Dead neurons policy
Environment steps

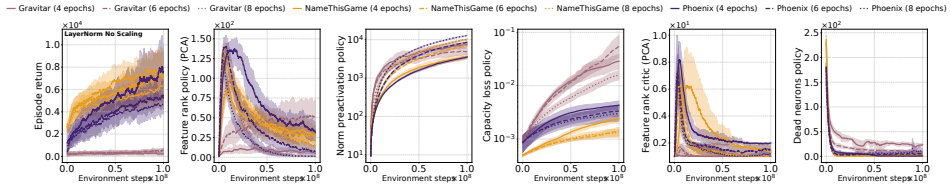

Gravitar (4 epochs)
Gravitar (6 epochs)
Gravitar (8 epochs)
NameThisGame (4 epochs)
NameThisGame (6 epochs)
NameThisGame (8 epochs)
Phoenix (4 epochs)
Phoenix (6 epochs)
Phoenix (8 epochs)
CHAIN Original
Episode return
Feature rank policy (PCA)
Norm preactivation policy
Capacity loss policy
Feature rank critic (PCA)
Dead neurons policy
Environment steps

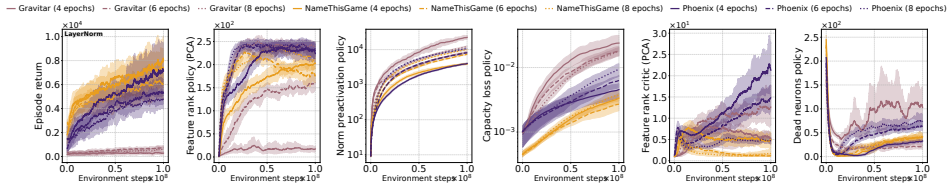

Gravitar (4 epochs)
Gravitar (6 epochs)
Gravitar (8 epochs)
NameThisGame (4 epochs)
NameThisGame (6 epochs)
NameThisGame (8 epochs)
Phoenix (4 epochs)
Phoenix (6 epochs)
Phoenix (8 epochs)
CHAIN-SP
Episode return
Feature rank policy (PCA)
Norm preactivation policy
Capacity loss policy
Feature rank critic (PCA)
Dead neurons policy
Environment steps

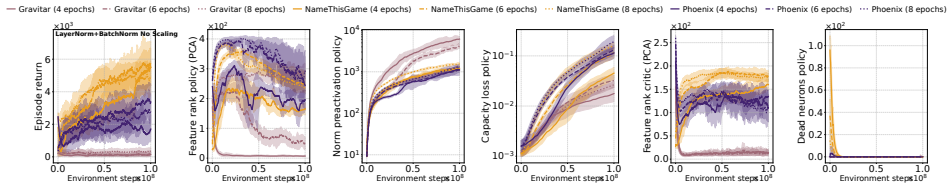

Gravitar (4 epochs)
Gravitar (6 epochs)
Gravitar (8 epochs)
NameThisGame (4 epochs)
NameThisGame (6 epochs)
NameThisGame (8 epochs)
Phoenix (4 epochs)
Phoenix (6 epochs)
Phoenix (8 epochs)
InPeR
Episode return
Feature rank policy (PCA)
Norm preactivation policy
Capacity loss policy
Feature rank critic (PCA)
Dead neurons policy
Environment steps

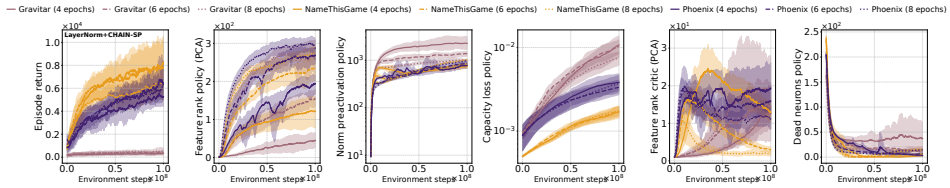

Gravitar (4 epochs)
Gravitar (6 epochs)
Gravitar (8 epochs)
NameThisGame (4 epochs)
NameThisGame (6 epochs)
NameThisGame (8 epochs)
Phoenix (4 epochs)
Phoenix (6 epochs)
Phoenix (8 epochs)
L2init
Episode return
Feature rank policy (PCA)
Norm preactivation policy
Capacity loss policy
Feature rank critic (PCA)
Dead neurons policy
Environment steps

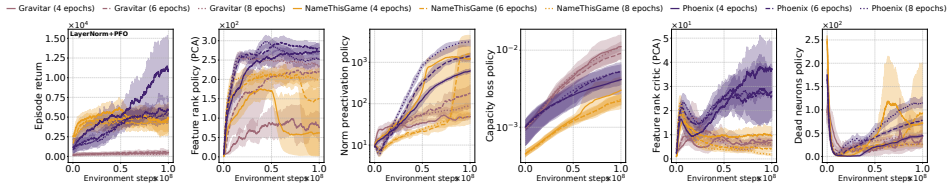

Gravitar (4 epochs)
Gravitar (6 epochs)
Gravitar (8 epochs)
NameThisGame (4 epochs)
NameThisGame (6 epochs)
NameThisGame (8 epochs)
Phoenix (4 epochs)
Phoenix (6 epochs)
Phoenix (8 epochs)
Larger Batch Size
Episode return
Feature rank policy (PCA)
Norm preactivation policy
Capacity loss policy
Feature rank critic (PCA)
Dead neurons policy
Environment steps

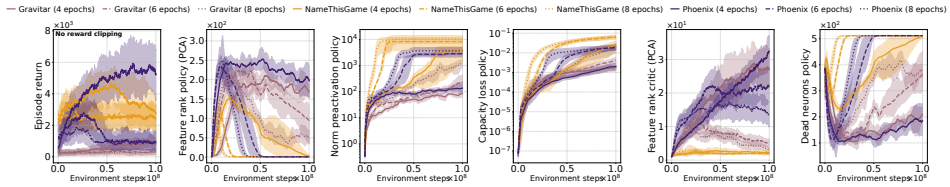

Gravitar (4 epochs)
Gravitar (6 epochs)
Gravitar (8 epochs)
NameThisGame (4 epochs)
NameThisGame (6 epochs)
NameThisGame (8 epochs)
Phoenix (4 epochs)
Phoenix (6 epochs)
Phoenix (8 epochs)
LayerNorm No Scaling
Episode return
Feature rank policy (PCA)
Norm preactivation policy
Capacity loss policy
Feature rank critic (PCA)
Dead neurons policy
Environment steps

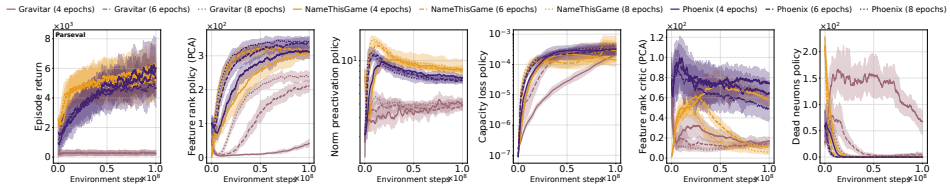

Gravitar (4 epochs)
Gravitar (6 epochs)
Gravitar (8 epochs)
NameThisGame (4 epochs)
NameThisGame (6 epochs)
NameThisGame (8 epochs)
Phoenix (4 epochs)
Phoenix (6 epochs)
Phoenix (8 epochs)
LayerNorm
Episode return
Feature rank policy (PCA)
Norm preactivation policy
Capacity loss policy
Feature rank critic (PCA)
Dead neurons policy
Environment steps

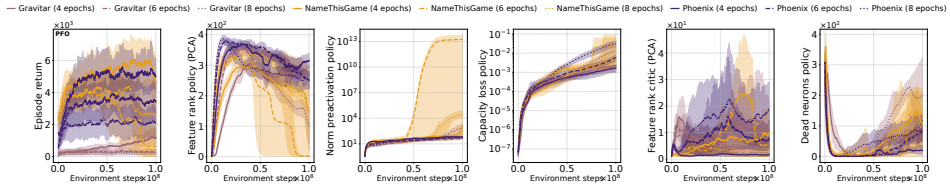

Gravitar (4 epochs)
Gravitar (6 epochs)
Gravitar (8 epochs)
NameThisGame (4 epochs)
NameThisGame (6 epochs)
NameThisGame (8 epochs)
Phoenix (4 epochs)
Phoenix (6 epochs)
Phoenix (8 epochs)
LayerNorm+BatchNorm No Scaling
Episode return
Feature rank policy (PCA)
Norm preactivation policy
Capacity loss policy
Feature rank critic (PCA)
Dead neurons policy
Environment steps

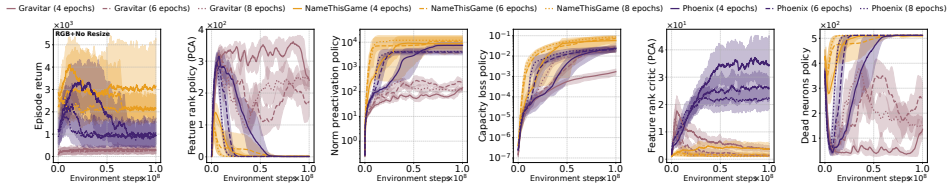

Gravitar (4 epochs)
Gravitar (6 epochs)
Gravitar (8 epochs)
NameThisGame (4 epochs)
NameThisGame (6 epochs)
NameThisGame (8 epochs)
Phoenix (4 epochs)
Phoenix (6 epochs)
Phoenix (8 epochs)
LayerNorm+CHAIN-SP
Episode return
Feature rank policy (PCA)
Norm preactivation policy
Capacity loss policy
Feature rank critic (PCA)
Dead neurons policy
Environment steps

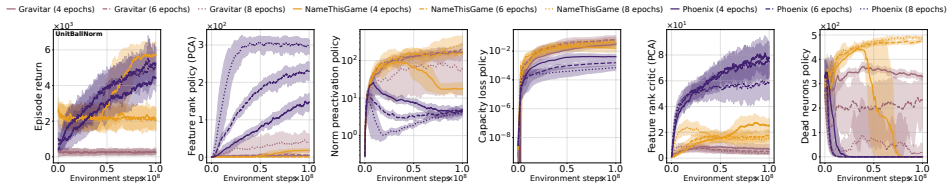

Gravitar (4 epochs)
Gravitar (6 epochs)
Gravitar (8 epochs)
NameThisGame (4 epochs)
NameThisGame (6 epochs)
NameThisGame (8 epochs)
Phoenix (4 epochs)
Phoenix (6 epochs)
Phoenix (8 epochs)
LayerNorm+PFO
Episode return
Feature rank policy (PCA)
Norm preactivation policy
Capacity loss policy
Feature rank critic (PCA)
Dead neurons policy
Environment steps


Gravitar (4 epochs)
Gravitar (6 epochs)
Gravitar (8 epochs)
NameThisGame (4 epochs)
NameThisGame (6 epochs)
NameThisGame (8 epochs)
Phoenix (4 epochs)
Phoenix (6 epochs)
Phoenix (8 epochs)
No reward clipping
Episode return
Feature rank policy (PCA)
Norm preactivation policy
Capacity loss policy
Feature rank critic (PCA)
Dead neurons policy
Environment steps


Gravitar (4 epochs)
Gravitar (6 epochs)
Gravitar (8 epochs)
NameThisGame (4 epochs)
NameThisGame (6 epochs)
NameThisGame (8 epochs)
Phoenix (4 epochs)
Phoenix (6 epochs)
Phoenix (8 epochs)
Parseval
Episode return
Feature rank policy (PCA)
Norm preactivation policy
Capacity loss policy
Feature rank critic (PCA)
Dead neurons policy
Environment steps


Gravitar (4 epochs)
Gravitar (6 epochs)
Gravitar (8 epochs)
NameThisGame (4 epochs)
NameThisGame (6 epochs)
NameThisGame (8 epochs)
Phoenix (4 epochs)
Phoenix (6 epochs)
Phoenix (8 epochs)
PFO
Episode return
Feature rank policy (PCA)
Norm preactivation policy
Capacity loss policy
Feature rank critic (PCA)
Dead neurons policy
Environment steps


Gravitar (4 epochs)
Gravitar (6 epochs)
Gravitar (8 epochs)
NameThisGame (4 epochs)
NameThisGame (6 epochs)
NameThisGame (8 epochs)
Phoenix (4 epochs)
Phoenix (6 epochs)
Phoenix (8 epochs)
RGB+No Resize
Episode return
Feature rank policy (PCA)
Norm preactivation policy
Capacity loss policy
Feature rank critic (PCA)
Dead neurons policy
Environment steps


Gravitar (4 epochs)
Gravitar (6 epochs)
Gravitar (8 epochs)
NameThisGame (4 epochs)
NameThisGame (6 epochs)
NameThisGame (8 epochs)
Phoenix (4 epochs)
Phoenix (6 epochs)
Phoenix (8 epochs)
UnitBallNorm
Episode return
Feature rank policy (PCA)
Norm preactivation policy
Capacity loss policy
Feature rank critic (PCA)
Dead neurons policy
Environment steps