Journal of Machine Learning Research 1337 (2024) 1-48 Submitted 4/00; Published 10/00

A Survey on Recent Advances in Plasticity Loss

Timo Klein MMP@STAT.WASHINGTON.EDU
Faculty of Computer Science
University of Vienna
UniVie Doctoral School Computer Science

Kevin Sidak JORDAN@CS.BERKELEY.EDU
Faculty of Computer Science
University of Vienna
UniVie Doctoral School Computer Science

Lukas Miklautz JORDAN@CS.BERKELEY.EDU
Faculty of Computer Science
University of Vienna

Sebastian Tschiatschek JORDAN@CS.BERKELEY.EDU
Faculty of Computer Science
University of Vienna

Editor:

Abstract

Akin to neuroplasticity in humans, plasticity in deep neural networks is an attribute of the network that enables adapting quickly to new data. This makes it particularly crucial for deep Reinforcement Learning (RL) agents: Once plasticity is lost, an agent's performance will inevitably plateau because it cannot improve its policy to account for a changing data distribution. Thus, developing sample-efficient agents hinges on their ability to remain plastic during training. Loss of plasticity can be connected to many other issues plaguing deep RL, such as training instability, scaling failures, overestimation bias, and insufficient exploration, making it even more compelling. With this survey, we aim to provide an overview of this important field. First, we will present different definitions of plasticity loss and related metrics. Then, we will categorize and discuss numerous possible causes of plasticity loss from the literature, including possible explanatory causal mechanisms. We then review and categorize mitigation strategies currently deployed before looking at future directions in the field. For example, we recommend more research on the links between classical RL issues such as overestimation bias, modern deep RL regularizers, and plasticity loss, and we suggest more deeply exploring the connection between an agent's neural activity and its behavior.

Keywords: Reinforcement Learning, Plasticity Loss, Continual Learning, Review, Survey

1. Introduction

Deep Reinforcement Learning (RL) has recently seen many successes and breakthroughs: It has beaten the best human players in Go [82] and Dota [8], discovered new matrix multiplication algorithms [25], makes the difference between a dreary language model giving highly artificial replies and a chatbot breaking the Turing test [10], and has allowed for substantial progress in robotic control [73]. Its ability to react to changes in an environment and subsequently make near-optimal decisions is likely a crucial component for any generally capable agent while learning through interaction purely from trial-and-error mimics human learning [84].

Despite all these successes, deep RL still fails to deliver on the promises made by supervised learning. For example, scaling has been an integral component in ever-better deep learning models, whereas most RL research does not go past the small network used in the seminal DQN paper [63]. Deep RL needs a plethora of hacks and stabilization tricks to reach high-level performance that are notoriously difficult to get right: From Replay buffers and target networks [63] to noise de-correlation [87] and pessimistic value functions [28], and finally to hacky optimizer settings [2, 56] and bespoke hyperparameter schedules [80].

There are many reasons why this is the case: First and foremost, Deep RL is inherently non-stationary; this makes it a substantially harder learning problem than supervised learning. Additionally, it suffers from its own optimization issues, such as under-exploration, sample correlation, and overestimation bias. Much recent work has been devoted to tackling these problems with ever-more elaborate algorithms, aiming to transfer insights from tabular RL to the deep RL setting [15, 75].

But what if the problems in deep RL are not (only) rooted in classical issues but instead can be attributed to a significant extent to optimization pathologies arising from applying deep neural networks to non-stationary tasks [9, 29, 67]? Recently, this view has gained traction under the umbrella term *plasticity loss*. It summarizes a line of research that can be broadly described as trying to find answers to the following three main research questions:

- *Why do RL agents lose their learning ability over time?*
- *What causes neural networks to lose their learning ability?*
- *How can learning ability be maintained?*

These questions are not just RL specific issues that are of no interest to the rest of machine learning: They address fundamental problems relevant to anybody who wants to deploy machine learning agents that can adapt to their circumstances. As recent work has shown, making progress on these questions may likely also alleviate some of the issues afflicting deep RL, such as overestimation bias [67] and its inability to scale [24].

Scope This survey focuses on the phenomenon of plasticity loss and its applications in deep RL. We want to highlight that deep RL is not the only field where plasticity loss occurs: For example, it is also a relevant issue in continual learning [22] and known to occur in the ubiquitous pre-train/fine-tune setup in supervised learning [7, 52]. While this survey touches on these scenarios, they are not our focus. Within deep RL, we concentrate on the single-agent setting, as the understanding of plasticity loss is most advanced here. Lastly, we will not discuss theoretical work.

Our survey starts with an overview of the RL formalism and definitions relevant to plasticity loss. As we will see, the phenomenon is intuitively easy to define as networks losing their ability to learn but without a single accepted definition in the literature yet.

Next, we categorize and present hypothetical causes for plasticity loss from the literature, followed by an overview of currently deployed remedies. We then discuss some of the factors that deep RL researchers and practitioners should consider when using deep RL algorithms from the perspective of plasticity loss. Finally, our survey concludes with a discussion of the current state of the field and an outlook on future directions.

To summarize our contributions:

- We categorize the different causes of plasticity loss proposed in the literature in a cohesive overview. These findings are synthesized into a *coherent model of how plasticity loss occurs.*

- We propose a classification of the current mitigation strategies in the space of plasticity loss, providing researchers and practitioners with a toolkit from which to choose for their specific use case.

- We critically discuss the status quo in plasticity research and outline several directions for future research.

2. Notation and Preliminaries

2.1 General Notation

We adopt the following notation: We use lower-case bold symbols for vectors, e.g., $\mathbf{x}$ to denote an input sample. Upper-case bold symbols denote matrices, e.g., $\mathbf{X} \subset \mathcal{X}$ denotes the design matrix, which is a subset of the data domain $\mathcal{X}$. Expectations with respect to a distribution P are denoted as $\mathbb{E}_{\mathbf{a} \sim P}[\cdot]$. If it is clear from the context, we skip the subscript for brevity. Singular matrices of a matrix are denoted as σ. We use $\mathrm{SVD}(\mathbf{A})$ to denote the multiset of all singular values of $\mathbf{A}$, $\sigma_i(\mathbf{A})$ to denote the ith largest singular value of matrix $\mathbf{A}$ and $\sigma_{\min}$ and $\sigma_{\max}$ to denote the smallest and largest singular values, respectively. Many works [35, 54] decompose a deep RL agent into a learned representation ϕ, covering all layers up to and including the penultimate layer, and a linear transformation $\mathbf{W}$. Using the value function as example, Our notation for this decomposition is $V(\mathbf{s}) = \langle \phi(\mathbf{s}), \mathbf{W} \rangle$.

2.2 Reinforcement Learning

Reinforcement learning is concerned with optimizing intelligent agents' actions via trial-and-error to maximize a so-called cumulative reward (defined below). The agents' interactions with an environment are formalized via Markov Decision Processes (MDPs) that can be described as tuples $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \rho_0, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, i.e., set of possible actions, $\mathcal{P} \colon \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$ is the transition kernel specifying the probability of transitioning from one state to another state upon taking a specific action, $\mathcal{R} \colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function specifying the reward the agent obtains for taking an action in a state, ρ_o is the initial state distribution, and γ is the so-called discount factor. The behavior of an agent is defined by its possibly stochastic policy $\pi \colon \mathcal{S} \to [0, 1]^{|\mathcal{A}|}$ specifying for each state a distribution over the actions the agent can take. The agent aims to maximize the (discounted) cumulative reward

$$J(\pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, a_t) \mid \pi\right], \tag{1}$$

where the expectation is over the randomness of the transitions and the agent's policy, the initial state s_0 is sampled according to ρ_0, and the agent's actions are taken according to the policy π. An optimal policy π^* maximizes $J(\pi)$.

Key quantities for RL algorithms are their *state-value*,

$$V^\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, a_t) \mid s_0 = s\right], \tag{2}$$

i.e., the expected return when starting from state s and following policy $\pi(s)$ from there, and the *action-value*,

$$Q^\pi(s,a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, a_t) \mid s_0 = s, a_0 = a\right], \tag{3}$$

i.e., the expected return starting from state s, taking actions a, and following policy $\pi(s)$ afterwards. An optimal policy can be found by maximizing the expected value of the initial state, i.e.,

$$\pi^* \in \arg\max_{\pi} \mathbb{E}_{s\sim\rho_0}[V^\pi(s)] \tag{4}$$

Note that state-values and action-values can also be defined recursively:

$$V^\pi(s) = \mathbb{E}_{a\sim\pi(s))}[r(s,a) + \gamma \sum_{s'} \mathcal{P}(s,s',a) V^\pi(s')], \tag{5}$$

$$Q^\pi(s,a) = \mathbb{E}_{a\sim\pi(s))}[r(s,a) + \gamma \sum_{s'} \mathcal{P}(s,s',a) \mathbb{E}_{a'\sim\pi(s')} Q^\pi(s',a')], \tag{6}$$

Inspired by these recursive definitions are so-called *temporal-difference* learning approaches, e.g., based on iteratively updating state-value estimates as

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]. \tag{7}$$

In these approaches, estimates of state-values (or action-values) are updated based on estimates of those values for successor states (and actions) — that is why such methods are also called *bootstrapping methods*.

2.3 Definitions of Effective Rank

The effective rank definition below is based on the *energy ratio* (also called *cumulative explained variance*) commonly used when looking into principal component analysis and defined as the minimum k such that a rank-k approximation of the feature matrix explains a $(1-\delta)$ fraction of its total variance.

Definition 1 (Effective Rank [46]) *Let $\Phi = \phi(\mathbf{X})$ be a feature matrix, e.g., the embeddings of a neural network, for a collection of samples $\mathbf{X}$. Let $\delta \in [0,1]$ and $\sigma_1 \geq \cdots \geq \sigma_d \geq 0$ be the singular values of Φ in increasing order. The* effective rank *is defined as*

$$\operatorname{srank}_\delta(\Phi) = \min\left\{k : \frac{\sum_{i=1}^{k}\sigma_i(\Phi)}{\sum_{i=1}^{d}\sigma_i(\Phi)} \geq 1-\delta\right\}. \tag{8}$$

The *feature rank* defined below can be understood as quantifying "how easily states can be distinguished by updating only the final layer of the network" [55], given by the feature mapping ϕ. The feature rank was introduced to approximate the target fitting capacity while being cheaper to compute.

Definition 2 (Feature rank [55]) *Let $\phi : \mathcal{X} \to \mathbb{R}^d$ be a feature mapping. Let $\mathbf{X}_n \subset \mathcal{X}$ be a set of n states in X sampled from some fixed distribution P. Fix $\varepsilon \geq 0$, and let $\phi(\mathbf{X}_n) \in \mathbb{R}^{n\times d}$ denote the matrix whose rows are the feature embeddings of states $x \in \mathbf{X}_n$. The* feature rank *of ϕ for distribution P is defined as*

$$\rho(\phi, P, \epsilon) = \lim_{n\to\infty} \mathbb{E}_{\mathbf{X}_n\sim P}\left[\left|\left\{\sigma \in \operatorname{SVD}\left(\frac{1}{\sqrt{n}}\phi(\mathbf{X}_n)\right) \,\middle|\, \sigma > \varepsilon\right\}\right|\right] \tag{9}$$

for which a consistent estimator can be constructed as follows, letting $\mathbf{X} \subseteq \mathcal{X}, |\mathbf{X}| = n$

$$\hat{\rho}_n(\phi, \mathbf{X}, \epsilon) = \left|\left\{\sigma \in \operatorname{SVD}\left(\frac{1}{\sqrt{n}}\phi(\mathbf{X})\right) \,\middle|\, \sigma > \varepsilon\right\}\right|. \tag{10}$$

Feature Rank Definitions

- Kumar et al. [46] provide the most widely used definition of feature rank, using the sorted singular values $\sigma_1(\phi) \geq \sigma_2(\phi) \geq \ldots$ of an agent's penultimate layer representation ϕ:

$$\operatorname{srank}_\delta(\phi, \mathbf{X}) := \min\left\{k : \frac{\sum_{i=1}^{k}\sigma_i\big(\phi(\mathbf{X})\big)}{\sum_{i=1}^{d}\sigma_i\big(\phi(\mathbf{X})\big)} \geq 1-\delta\right\},$$

  where δ is a threshold parameter usually set to $\delta = 0.01$.

- Lyle et al. [55] normalize the representation by the number of samples in the batch before computing the feature rank:

$$\operatorname{srank}_\epsilon(\phi, \mathbf{X}) = \left|\left\{\sigma \in \operatorname{SVD}\left(\frac{1}{\sqrt{n}}\phi(\mathbf{X})\right) \,\middle|\, \sigma > \varepsilon\right\}\right|,$$

  with ϵ being a threshold parameter set to $\epsilon = 0.01$.

- Huh et al. [37] define feature rank as the entropy of the distribution of normalized singular values. This has the advantage of being non-parametric and continuous:

$$\operatorname{srank}_{\text{continuous}}(\phi, \mathbf{X}) = -\sum_{i=1}^{\min(n,m)} \bar{\sigma}_i\big(\phi(\mathbf{X})\big)\,\log\big(\bar{\sigma}_i\big(\phi(\mathbf{X})\big)\big),$$

  where $\sum_i \bar{\sigma}_i = 1$.

2.4 Gradient Covariance Matrix and Empirical Neural Tangent Kernel

The gradient covariance matrix and empirical neural tangent kernel (eNTK) are closely related quantities, giving insights into both the optimization behavior and the generalization abilities of a neural network [56]. In the following, we will use the gradient covariance matrix $\mathbf{C}_k$, noting that the eNTK is just the unnormalized gradient covariance matrix [58]. Given sample indices i and j, Equation (11) defines $\mathbf{C}_k$ as:

$$\mathbf{C}_k[i,j] = \frac{\langle \nabla_\theta L(\theta, \mathbf{x}_i), \nabla_\theta L(\theta, \mathbf{x}_j) \rangle}{\|\nabla_\theta L(\theta, \mathbf{x}_i)\| \|\nabla_\theta L(\theta, \mathbf{x}_j)\|} \; . \tag{11}$$

In practice, it is impossible to compute this matrix over the complete state space of a deep RL environment, which means it has to be approximated with k samples. From a generalization perspective, $\mathbf{C}_k$ gives insights into whether updates between a pair of inputs $\mathbf{x}_i, \mathbf{x}_j$ *generalize.* This is the case when the dot product of their gradients is positive. When $\langle \nabla_\theta L(\theta, \mathbf{x}_i), \nabla_\theta L(\theta, \mathbf{x}_j) \rangle$ is negative, it means that updates between samples don't generalize, but instead *interfere* [56]. From an optimization perspective, a pronounced block structure of the gradient covariance matrix indicates a sharp and unstable loss landscape. Additionally, when $\mathbf{C}_k$ is low-rank, i.e., when most values are positive or negative values of similar magnitude, then the network has learned a degenerate function where updates on single samples generalize to the whole input space [58]. The leftmost plot in Figure 1 provides an example of the gradient covariance matrix for highly collinear gradients.

3. Definitions of Plasticity Loss and Related Phenomena

In this section, we first provide and generalize definitions of plasticity loss from the literature and then relate it to aspects of related phenomena in the field of continual learning.

3.1 Definitions of Plasticity Loss

In the literature, there is no single accepted definition of plasticity loss. In this section, we attempt a unification of some of the existing definitions and illustrate that many definitions from the literature are special cases of it. Intuitively, all these definitions try to capture the loss of ability to fit new targets but formalize it to different extents or relate it to properties of the features a model is trained on.

Definition 3 (Loss of plasticity) *Let $P_{\mathcal{X}}^{(t)}$ be a distribution over inputs $\mathcal{X}$, and $P_L^{(t)}$ a distribution over a family of real-valued loss functions L with domain $\mathcal{X}$. Let g_θ represent a neural network with parameters θ, $\mathcal{O}$ correspond to an optimization algorithm for supervised learning, and $\mathcal{I}$ represent an* intervention *on the parameters θ. Using*

$$c^{(t)}(\theta) = \mathbb{E}_{L \sim P_L^{(t)}} \left[\mathbb{E}_{\mathbf{x} \sim P_{\mathcal{X}}^{(t)}} \left[L(g_\theta, \mathbf{x}) \right] \right] \tag{12}$$

to denote the loss of the neural network at time t using parameters θ, we define the loss of plasticity as

$$\mathcal{C}(\{P_{\mathcal{X}}^{(t)}\}_{t=1}^T, \{P_L^{(t)}\}_{t=1}^T, \mathcal{O}, \mathcal{I}) = c^{(T)}(\theta^{(T)}) - c^{(T)}(\theta^{(0)}) \tag{13}$$

where

$$\theta^{(t+1)} = \mathcal{O}\left(\theta^{(t)\prime}, P_{\mathcal{X}}^{(t)}, P_L^{(t)}\right) \quad \textit{and} \quad \theta^{(t)\prime} = \mathcal{I}(\theta^{(t)}, P_{\mathcal{X}}^{(t)}, P_L^{(t)}). \tag{14}$$

This definition generalizes many existing definitions in that it enables different losses at different time steps which is, e.g., relevant for multi-task learning, and in that it allows for explicit manipulations of the parameters, outside of the behavior of the optimization algorithm. Clearly, interventions on the parameters, e.g., resetting of parameters to revive dead neurons, could also be considered as part of the optimizer but making the interventions explicit can help to explicate the considered way of measuring the loss of plasticity.

Many definitions of *loss of plasticity* — sometimes referred to by a different term — in the literature are special cases of the above definition:

- Berariu et al. [7] defined the *generalization gap* as "the difference in performance between a pretrained model — e.g. one that has learned a few tasks already — versus a freshly initialized one". The notion of the already-learned tasks corresponds to different tasks given by $P_{\mathcal{X}}^{(t)}, P_L^{(t)}$ for $t = 1, \ldots, T-1$ while the performance is evaluated with respect to a final task characterized by $P_{\mathcal{X}}^{(t)}, P_L^{(t)}$. The freshly initialized model is given by $g_{\theta(0)}$ with $\theta^{(0)}$ being random initial parameters.

- Lyle et al. [55] defined the *target-fitting capacity* as a measure of how well a neural network can fit a distribution of targets given by a family of labeling functions (real-valued functions mapping inputs X to targets). This definition arises from Definition 3 by considering $T = 1$ and selecting $P_{\mathcal{X}}^{(t)}, P_L^{(t)}$ accordingly.[1]

- Lyle et al. [56] also define *loss of plasticity* but don't explicitly account for time-dependent distributions $P_{\mathcal{X}}^{(t)}, P_L^{(t)}$ and interventions. Their definition is thus a special case without applying any intervention and constant distributions for the input and the loss functions.

- Elsayed and Mahmood [22] provide a sample-based notion of plasticity loss corresponding to a baseline normalized version of the plasticity loss defined in Lyle et al. [56]. Their definition arises as a special case from ours by fixing the loss function when making the loss function dependent on the optimizer and the intervention.

3.2 Aspects of Plasticity loss

In this section, we will briefly discuss the aspects of plasticity loss and its connection to other fields. For example, network plasticity is also a crucial property required to facilitate *continual learning*, where a system learns from a sequence of data distributions, each defining a new task [65]. To be successful in continual learning, the learner must also mitigate another well-known pathology of deep learning, namely *catastrophic forgetting* [60]. It refers to a network suddenly being unable to solve a previously learned task after learning a new task. In continual learning, the system must find a balance between tackling plasticity

1. There is still a slight difference between the definition in Lyle et al. [55] and our definition: our definition subtracts $c^{(T)}(\theta^{(0)})$ as a baseline.

loss and catastrophic forgetting: If the network is very plastic and immediately adapts to data from a different distribution, it is also more prone to quickly forgetting previously learned knowledge, resulting in catastrophic forgetting. The other extreme is a network with frozen weights: It certainly won't forget previous tasks but also suffers from maximal loss of plasticity.

It is reasonable to assume that task shifts exacerbate plasticity loss [3, 7]. However, these are not the only scenarios in which it occurs. Plasticity can also be lost catastrophically in single reinforcement learning environments such as the Atari game Phoenix [70]. Whether an agent in an RL environment faces a single task is a peculiar question: In value-based RL with target networks, it can indeed be argued that with each new target network update and change in the policy's state-action distribution, the agent solves a new task [2]. However, completely stationary learning problems may also result in plasticity loss. For example, Lyle et al. [58] show that using large-mean targets in stationary regression is enough for networks to lose their plasticity.

Lastly, the phenomenon dubbed plasticity loss is a broad categorization, likely subsuming several individual phenomena and causes. As we will discuss in Section 4.5, plasticity loss entails different sub-categories, such as adaptability to input distribution shifts and to target shifts [51]. Similarly, there is a *generalizability* component to plasticity loss, for example, when a fine-tuned network trained under non-stationarity has significantly lower test accuracy than one not facing non-stationarity. What is commonly referred to as plasticity loss and used as the basis for the definitions in Section 3.1, however, is *trainability*: It means that a network is unable to update its parameters, even given a high loss [1, 52].

4. Causes of Plasticity Loss

In this section, we present and categorize possible causes of plasticity loss from the literature. Note that these causes are on different conceptual levels: Non-stationarity is a common high-level property of a learning problem that seems to occur in almost all settings where plasticity loss occurs. In contrast, the curvature of a network's optimization landscape is a very specific, measurable network property. With this in mind, we will also fuse the current understanding within the literature into a model relating different network pathologies and possible causes of plasticity loss.

4.1 Reduced Capacity due to Saturated Units

Saturated [12, 58] or dormant [83] units are one of the most prominent pathologies associated with reduced agent performance and plasticity loss. They are an obvious and objectively measurable sign indicating that the network fails to fully utilize its full capacity. Therefore, it is easy to draw a connection between neurons ceasing to activate in response to inputs and reduced network expressivity and slow learning [83]. However, it is unclear as to whether dormant neurons are the main causal mechanism driving plasticity loss or are just a pathology associated with networks that have lost their plasticity. For example, Sokar et al. [83] discuss target non-stationarity as a contributing factor to an increase in dormant neurons over the course of training. Other aspects of modern deep RL algorithms, such as training with high replay ratios [21, 80] might also exacerbate the phenomenon.

How can units with reduced capacity be formally defined? The literature provides a plethora of options to do this, which we group into two categories: *Saturated units* for which a shift in the pre-activation distribution reduces the capacity of the neuron to produce meaningfully different outputs given inputs from its input distribution. *Dormant units* are instead characterized by low post-nonlinearity activations. When considering a tanh unit, one can easily see how these categories differ: For large pre-activations, the unit's output will be close to one for all inputs. Importantly, the output will be close to one (show small numerical differences) even considering significant differences in the pre-activation. On the other hand, such a saturated tanh unit is clearly not dormant because its post-nonlinearity activation is high and likely still high, even when normalizing by all activations for a particular layer.

Saturated Neurons have first been discussed in Bjorck et al. [12] in the context of pixel-based continuous control. The authors take a closer look at runs of the then-state-of-the-art agent DrQ-v2 [92] and observed that most of them exhibit saturated tanh policies, with runs failing to learn and not being able to move out of this regime[2]. This results in actions at the extreme end of the $[-1, 1]$ interval from which actions are usually selected in continuous control combined with vanishing gradients, as $\forall \mathbf{x} \colon |\tanh(g_\theta(\mathbf{x}))| \approx 1 \implies \frac{\partial}{\partial\theta}\tanh(g_\theta(\mathbf{x})) = [1 - \tanh^2(g_\theta(\mathbf{x}))]\frac{\partial}{\partial\theta}g_\theta(\mathbf{x}) \approx 0$. Interestingly, their proposed solution to normalize the features of the penultimate layer increases performance when applied to both actor and critic [12].

More recently, Lyle et al. [58] partition non-stationary learning problems into an erasing or unlearning phase and a disentanglement phase, finding that the first causes a form of saturation in ReLU units. These *Linearized units* are characterized by their pre-activation distribution having only positive support. To be more precise, the unlearning phase after a task shift causes a distribution shift in the neuron's pre-activation distribution through gradients that either increase or decrease the preactivation values for all training samples. If all pre-activations for a neuron are now positive, the unit becomes *linearized*, removing its nonlinear component. If a neuron's pre-activations are negative for the training dataset, it moves into the dead neuron regime. Both of the aforementioned pathologies effectively reduce the expressive capacity of the network. As a remedy, the authors propose to apply LayerNorm [4] before a unit's nonlinearity [58].

Dormant Neurons have been introduced by Sokar et al. [83] as a measure for the reduced expressivity of a neural network. In experiments, they have shown that dormant neurons are associated with performance plateaus and reduced performance for DQN [63] agents trained on different Atari games and with different hyperparameters. To determine the level of dormancy, one has to first calculate normalized activation scores s_i^l for each neuron i in all non-final layers l [83]:

$$s_i^l = \frac{\mathbb{E}_{\mathbf{x}\in\mathcal{D}}\left[|h_i^l(\mathbf{x})|\right]}{\frac{1}{H^l}\sum_{k\in h}\mathbb{E}_{\mathbf{x}\in\mathcal{D}}\left[|h_k^l(\mathbf{x})|\right]} \ . \tag{15}$$

2. DrQ-v2 builds on top of TD3 [28], which is an agent based on deterministic policy gradients that uses the tanh function to clip actions in $[-1, 1]$.

Here $\mathbf{x} \in \mathcal{D}$ are samples drawn from an input distribution, e.g., the replay buffer in off-policy deep RL, $h_i^l(\mathbf{x})$ denotes the post-nonlinearity activation of a neuron in layer l, and H^l the number of neurons in layer l.

Neuron Dormancy

If the score in Equation (15) is below a threshold $s_i^l \leq \tau$, then neuron i in layer l is τ-*dormant*. The *dormancy ratio* is the fraction of dormant neurons divided by the number of total neurons in all layers except the final layer [89]:

$$\beta_\tau = \sum_{l \in \theta} H_\tau^l / \sum_{l \in \theta} N^l \tag{16}$$

An agent exhibits the *dormant neuron phenomenon* if its fraction of dormant neurons increases over the course of the training.

The definition above is certainly not the only way of defining a measure for neurons with reduced activity, but it is currently widely accepted within the literature [59, 89] due to its simplicity and intuitive understanding. Dohare et al. [19] define a more complex measure of neuron utility. A more straightforward option for ReLU networks is to just count the number of zero activations [1, 22].

To summarize: While there are different options for defining inactive neurons [19, 22, 83], all of the works cited above agree that they are a symptom of reduced expressivity caused by some form of non-stationarity [1, 59, 83]. Where they differ is in the proposed solution to address inactive units: Sections 5.1 and 5.2 discuss a plethora of reset strategies as solution algorithms [19, 69, 83, 89], Section 5.5 considers activation functions for non-stationary problems such as CReLU [1], and Section 5.3 examines parameter regularization [53].

4.2 Effective Rank Collapse

Multiple works observe a correlation between an agent's performance degradation and its representation becoming low-rank. This phenomenon is dubbed "rank collapse" and has been observed both in online [35, 46, 55] and offline [32, 46] RL. Kumar et al. [46] establish a connection between the effective rank of an agent's representation and its ability to learn: A decrease in the network's rank leads to increased TD error. In theoretical and empirical analysis, Kumar et al. [46] show that a drop in rank is caused by the largest singular values of the representation outgrowing the smaller ones, most likely caused by bootstrapping in value-based deep RL. This effect may be exacerbated in sparse-reward environments such as Montezuma's revenge on Atari [56]. Gülçehre et al. [32] presents a thorough empirical study and find that the association between effective rank and agent performance is not as straightforward as previously assumed in offline RL. In particular, it depends on potentially confounding hyperparameter and architecture choices, such as the activation function and the learning rate. However, they show that the collapse of the effective rank to a very small value is reliable and allows for identifying underfitting agents with suboptimal performance at the end of training.

The phenomenon of rank collapse is tightly intertwined with other phenomena pertaining to loss of plasticity. In offline RL, there is a very strong correlation between the effective rank and the number of dead units in the agent's network [32]. Similarly, resetting dead or dormant neurons increases feature rank at the end of online RL training [83]. As of now, the exact causal relationship between dead neurons and rank collapse is unclear.

4.3 First-order Optimization Effects

Deep neural networks are most commonly trained using stochastic gradient descent (SGD); looking at a neural network's gradients is, therefore, the most natural direction to look for causes of plasticity loss. *Vanishing gradients* are a well-known issue in training recurrent neural networks and may also play a role in the loss of plasticity. In particular, a collapse in the ℓ_0 and ℓ_1 gradient norms [3] despite high loss values is an indication that the agent isn't able to adapt despite a strong update signal [1]. A collapse in the ℓ_0 norms indicates increased gradient sparsity, which may contribute to more sparse ReLU activations and dead neurons [1]. A more nuanced notion of sparsity is *gradient dormancy*, which is closely related to neuron dormancy discussed in Section 4.1, but using the gradient ℓ_2 norms instead of neuron activations in its calculations [42]. Ji et al. [42] show that gradient dormancy is an issue faced by proprioceptive control agents in sparse-reward tasks, indicating a connection between sparse feedback, learning ability, and representation quality of an agent [54].

On the other end of the spectrum, *exploding gradients* are a clear sign of divergence and have been shown to occur when trying to scale deep RL networks [11] or in high-complexity environments, such as DM control's DOG [67]. Both vanishing and exploding gradients are easy to detect, although the latter usually has more severe performance implications than the former.

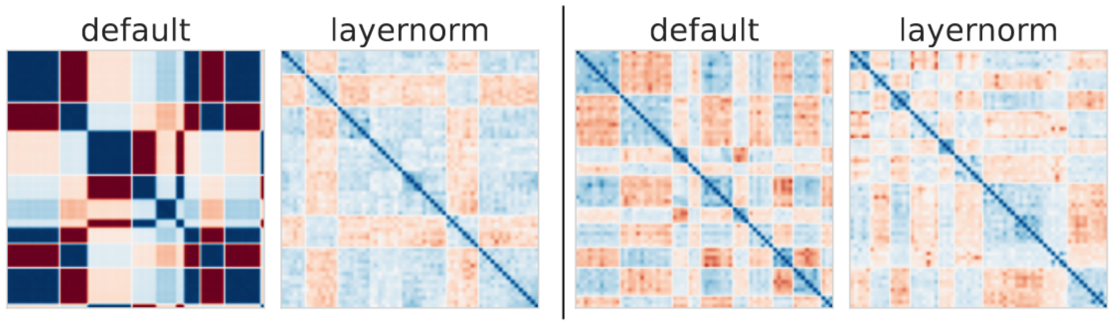

Figure 1: **Gradient covariance with and without LayerNorm on Atari**. *Left)* On Freeway, the gradient covariance matrix displays a very severe structure. Applying LayerNorm [4] weakens the correlation, leading to a 40.7% increase in human-normalized score [56]. *Right)* When LayerNorm yields no improvement, as on Kangaroo, gradients are also less collinear [56]. Figure taken from Lyle et al. [56].

3. Abbas et al. [1] note that the ℓ_2 gradient norm can be misleading due to the potentially disproportional weight of a few outliers.

A more subtle gradient-related optimization issue is gradient collinearity [56], which refers to the structure of the gradient covariance matrix or empirical neural tangent kernel (eNTK) [58], measuring how well gradients for one sample generalize to other samples. If this matrix has a diagonal structure, updates don't generalize, leading to overfitting. If all entries have the same value, then gradients from one sample generalize to all other samples, indicating that the network has learned a degenerate function [58]. For value-based deep RL, the eNTK often has a very severe structure, as shown in Figure 1. Many methods mitigating plasticity loss weaken this structure and reduce the correlation between gradients, such as LayerNorm, weight clipping, and pruning [23, 56, 58, 72]. This hints at highly correlated gradients being a sign of an ill-conditioned optimization landscape, potentially causing plasticity loss [56].

4.4 Second-order Optimization Effects

It is by now well-known from supervised learning that flat minima generalize better [26]. Connecting to this insight, there is a growing body of work linking plasticity loss with the sharpness of a network's optimization landscape: Networks that have lost plasticity usually also display high curvature [53, 56]. Lyle et al. [56] find that increased curvature as measured by the largest eigenvalue of a network's Hessian makes optimization more difficult and leads to plasticity loss. In a similar vein, Lewandowski et al. [53] argue that a collapse of the Hessian's effective rank [46] (cf. Definition 1) is causing plasticity loss. To measure curvature, they find that the empirical Fisher information matrix outperforms other approximations, such as the Gauss-Newton approximation, in terms of accuracy [53]. If a sharp optimization landscape were to cause plasticity loss, then methods explicitly reducing sharpness such as the SAM optimizer [26] should also mitigate it. Lee et al. [51] evaluate SAM in an Atari-100k agent and indeed find it very effective at reducing sharpness as measured by the Hessian's largest eigenvalue. However, when applied in isolation, head resets [69] still outperform SAM despite significantly higher curvature, indicating that sharpness cannot be the sole cause behind plasticity loss. Lee et al. [51] attributes this to smooth minima only reducing sensitivity to changes in the input distribution, whereas the orthogonal issue of changes in the target distribution must be tackled by complementary methods.

4.5 Non-stationarity

Non-stationarity is simultaneously the most likely culprit for causing plasticity loss and one of the most elusive. What type of non-stationarity causes networks to lose plasticity? Which capabilities of a network are affected? How can non-stationarity even be quantified? Intuitively, non-stationarity makes sense as a driving mechanism of plasticity loss. Practically, it is difficult to verify whether it is the sole underlying cause or to what extent non-stationarity causes networks to lose their ability to learn. In the following, we will give an overview of different aspects of non-stationarity, which may overlap in some parts.

Input Non-stationarity is defined through changes in the data distribution $p(x)$ [51]. In deep RL, this usually occurs due to an improving policy that generates data from a slightly different distribution with each update. Elsayed and Mahmood [22] use *input-permuted* MNIST to evaluate different methods with respect to plasticity loss, where the network has to solve a sequence of classification tasks with the pixels of MNIST images being shuffled new for each task. The idea behind input-permutation as a benchmark for plasticity is that for a network to be successful, it has to be able to re-learn its representation from scratch for each new task. This argument supports the hypothesis that input non-stationarity may cause plasticity loss. Other works simulate input distribution shifts by training networks on progressively expanding subsets of data, commonly based on CIFAR-10 or MINST classification [38, 51, 52, 58]. Depending on the size of the initial subset, this type of non-stationarity can have dramatic consequences: While the network is often able to reach similar levels of training accuracy, *test accuracy* suffers massively for smaller initial subset ratios [38, 52], even when later training on the full dataset (warm-starting). An early hypothesis for explaining this phenomenon is that the network initially overfits to the smaller subsets, learning suboptimal features that are insufficient for high-performance classification on the full dataset [38]. Lee et al. [52] decompose plasticity into *trainability* and *generalizability*, indicating that current methods [1, 83] focus on the former, neglecting the latter. Instead, generalizability can be somewhat maintained by common regularization techniques such as data augmentation or weight decay in the warm-starting setting. These findings (a) verify the existence of the generalization gap from Definition **??**, and (b) they indicate that enhancing trainability without addressing generalizability leads to overfitting.

Target Non-stationarity refers to changes in a learning problem's labels over the course of the training, that is, the distribution $p(y|x)$ changes [51]. In deep RL, this typically occurs due to bootstrapping: Assuming the use of target networks, each update of the target network changes the optimization problem, such that, effectively, an RL algorithm solves a sequence of optimization problems [2]. There is mounting evidence in the literature suggesting that target non-stationarity is a main driver of plasticity loss, but the exact mechanism behind it is still unclear. Igl et al. [38] hypothesize that non-stationarity leads to the network learning suboptimal features, which are then reused and cause reduced performance over time. Similarly, Kumar et al. [46] have found that bootstrapping targets in deep RL produce low-rank representations that are associated with performance collapse. Bootstrapping may even cause early performance collapses that are hard to recover from when it leads to large fluctuations early in training [59]. Another study has determined whether input or target non-stationarity is causing plasticity loss by performing experiments with offline and online RL agents. They find that plasticity loss, as measured by the fraction of dormant neurons, is still present in offline RL. The authors conclude that target non-stationarity is the key factor [83]. This has been verified in toy experiments, where a network has to memorize labels on a sequence of MNIST tasks or regress on targets with increasing magnitude. Both types of non-stationarity lead to growing parameter norms, which is one of the pathologies commonly associated with plasticity loss and suspected of inhibiting learning [58]. Interestingly, Elsayed and Mahmood [22] argue that maintaining performance in the presence of target distribution shifts is more of a *catastrophic forgetting* problem. This is because the main challenge with moving targets is the agent maintaining its representation in the face of distribution shifts. In theory, the agent should be able to cope with changing labels simply by adapting its last layer [22], whereas input distribution shifts require full plasticity because the representation has to be relearned with each new task.

In summary, the evidence that non-stationarity plays a role in plasticity loss seems overwhelming. While research into the mechanisms as to how exactly non-stationarity causes plasticity loss is still in rather early stages, some recent works have started to provide plausible hypotheses for the phenomenon [57, 58]. Similarly, splitting up the hand-wavy argument that ”non-stationarity causes plasticity loss” into more fine-grained hypotheses pertaining to individual aspects such as input and label plasticity [51] will help to advance our understanding. As of now, it is still unclear whether the currently known facets of plasticity loss capture all relevant angles or whether more is to be discovered, e.g., trainability and generalizability [52]. From a deep RL perspective, it will also be useful to develop toy scenarios for analysis that more closely mimic the peculiarities of common RL benchmarks. For example, most of the works listed above use classification on MNIST or CIFAR to verify their initial hypotheses, despite most deep RL approaches using regression losses, which have different characteristics [24, 58].

4.6 Regression Loss

Classification tasks have been easier for deep neural networks than regression ever since the AlexNet breakthrough in image classification. Because value-based deep RL methods use a regression loss, recent work has hypothesized that regressing on non-stationary targets might be one of the leading causes of deep RL’s optimization issues [24, 58]. For example, the two-hot trick [79] has been successfully applied to train value networks and subsequently been linked to plasticity loss, albeit at a performance cost [56]. Farebrother et al. [24] find that reformulating regression as classification using a specific method called HL-Gauss [40] improves RL agents in many ways such as representational capacity, robustness to noise and uncertainty, and sample efficiency. However, they do not provide deeper insights into the exact mechanisms behind regression optimization issues.

A very recent investigation into the mechanisms driving plasticity loss poses the theory that *regression with large-mean targets* causes networks to lose plasticity even in stationary learning problems [58]. This issue is particularly prevalent in value-based deep RL, where an agent ideally improves over the course of training, resulting in temporal difference targets with increasing magnitude. Regressing on large-mean targets has two negative side effects: First, deep networks are prone to encoding the target offset into their weights instead of the bias. This leads to the singular values of the corresponding dimensions in parameter space exploding, resulting in an ill-conditioned feature matrix [58]. Second, when the network predicts the large mean targets insufficiently well, the squaring in the MSE yields large error terms. As gradients are proportional to errors in regression tasks, these large errors blow up the parameter norms of the network [24, 58]. This growth of the parameter norms is associated with a wide range of pathologies such as poor generalization [23], loss landscape sharpness [58], and finally plasticity loss [58].

4.7 Parameter Norm Growth

Lyle et al. [58] found that parameter norm growth is a common concomitant of networks that have lost plasticity and is associated with reduced task performance. In the following, we present four possible mechanisms that could explain this effect. First, Lyle et al. [58] link growing parameter norms and the sharpness of the optimization landscape: As the norms of the parameters grow, so does the maximum eigenvalue of the Hessian. This connects to the line of works described in Section 4.4, hypothesizing that curvature may explain plasticity loss. The same authors also hypothesize that large parameter norms may cause nonlinearities within the network such as activation functions or softmax heads to saturate. The effects of saturated units are described in detail in Section 4.1, but among them are gradient propagation issues and a loss in effective network capacity, both of which are associated with plasticity loss. Third, it has been found that growing parameter norms may lead to gradients with sparse and/or collinear gradient covariance matrices. As a result, updates may over-generalize or under-generalize in the sample space. In an extreme case, an over-generalizing network may learn a degenerate function assigning similar outputs to all inputs. On the other hand, an under-generalizing network can only memorize task labels [58]. Both states are linked to plasticity loss. Lastly, a recent study finds that large parameter norms affect the effective learning rate of a network. In particular, as the parameter norms of a network grow, so does the norm of its gradient, leading to instability and potential plasticity loss during training [57]. The implicit effect of parameter norms on the learning rate can be mitigated with a technique called Normalize-and-Project, which we discuss in Section 5.9.

4.8 High Replay Ratio Training

4.9 Early Overfitting

When training networks under non-stationarity, some early works have hypothesized that residual effects of overfitting to early training data might hinder late training progress. The earliest work describing this phenomenon attributes it to the network trying to re-use suboptimal features acquired early during training [38]. Their results on toy datasets have been confirmed in deep RL, named the ”primacy bias” [69]. The primacy bias refers to a psychological phenomenon in human learning, where early experiences can have long-lasting effects on later tasks. In the context of deep RL, it refers to agents overfitting to early interactions and subsequently being unable to update their networks in the face of new data.

While these early findings regarding plasticity loss have been compelling, more recent work has moved away from the hypothesis of early overfitting towards mechanistic and measurable explanations such as dead neurons, parameter norm growth, or feature rank collapse.

4.10 Discussion of Causes

When looking at the causes mentioned in the previous sections, it bears repeating that they are on different semantic levels: Non-stationarity is broad, hard to measure, and hard to observe, whereas the number of dead neurons is a specific, measurable property of a neural network at a given time step. As of now, the research is still inconclusive about (a) how exactly the individual mechanisms work, and (b) what the causal connections between them are. Additionally, it might well be possible that unknown confounding factors exist that might provide a more abstract understanding of plasticity loss.

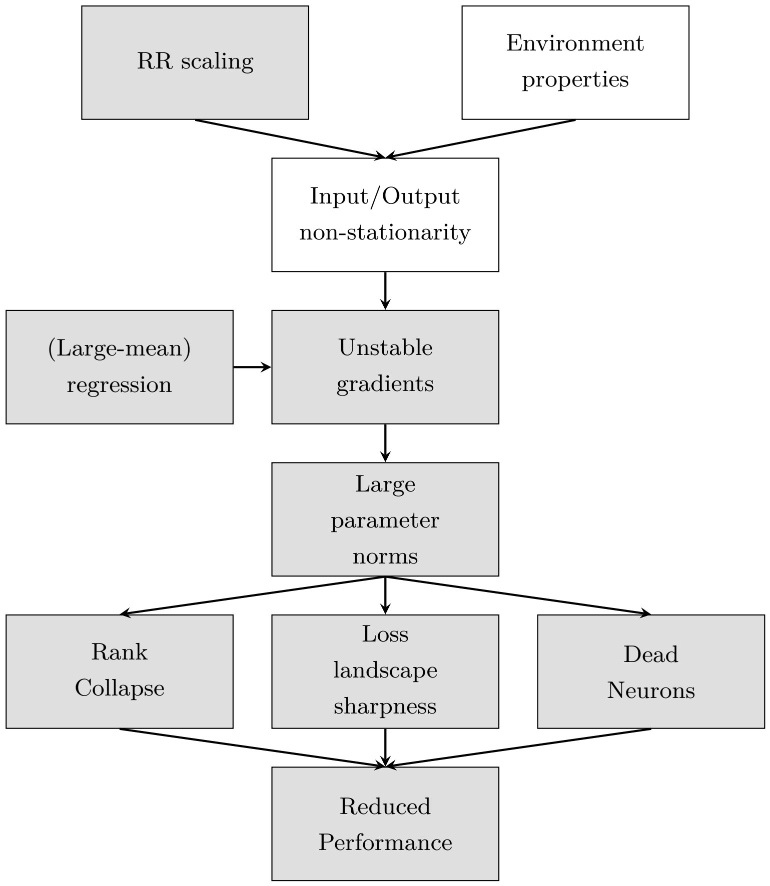

Figure 2: **Possible connections between causes of plasticity loss in value-based RL**. Large-mean regression targets combined with non-stationarity of deep RL training cause large and unstable gradients, leading to an increase in parameter norms. Large parameter norms are known to increase loss sharpness and cause other pathologies, together leading to reduced agent performance.

Figure 2 synthesizes the current understanding of plasticity loss in value-based deep RL agents into a model. It seems clear that non-stationarity, both for inputs and for targets, is one of the driving mechanisms behind networks losing their ability to learn [51, 52]. Recently, the issues in optimizing value-based deep RL have also been tied to the usage of regression losses, which are both less amenable to the training of deep networks in general [24] and are prone to lead to large parameter norms [58]. The latter factor is likely caused by the fact that in regression, gradients are proportional to the error, causing instability for growing and/or non-stationary regression targets [24, 58]. Growing parameter norms and unstable gradients may then cause numerous other pathologies commonly associated with plasticity loss, for example, dead neurons or a sharp optimization landscape [58, 83]. Together, all of these issues lead to reduced performance and the network's failure to adapt to new targets.

While Figure 2 tries to present a coherent model of plasticity loss, there is, as of now, no abstract theory of the phenomenon available. Instead, current research notes different optimization issues and that plasticity loss is likely caused by an amalgamation of these different factors. Whether these can be merged into a unified high-level understanding of the phenomenon is not yet clear.

5. Mitigating Loss of Plasticity

5.1 Non-targeted Weight Resets

Non-targeted weight resets are the most simple method of tackling plasticity loss and have initially been used to tackle early overfitting [69]. They come in two different flavors: Hard resets, where network layers are fully reset, and soft resets, where new network weights are generated from a linear combination of the current weights and freshly initialized ones.

Hard Resets build on the insight that plasticity loss has empirically been shown to be concentrated in the last layers of a network [7, 21], although it may affect all layers of the network to some extent. The simplest possible strategy for mitigating plasticity loss is, therefore, a periodic, full reset of the last few layers of the agent. For small agents, e.g., the ones typically used for non-pixel continuous control tasks, this amounts to resetting the complete network. In pixel-based control environments such as Atari, the convolutional encoder is not reset to retain the knowledge encoded in the agent's representation.

Soft Resets [3] aka Shrink and Perturb (S & P) have been introduced to mitigate the so-called "*generalization gap*", which is the difference in performance observed when training a freshly initialized model compared to one using the pretrain-finetune paradigm [7]. S & P uses a linear combination of the current weights θ_{t-1} and randomly initialized parameters ψ from the initialization distribution to perform the reset [21] [4]:

$$\theta_t = \alpha\theta_{t-1} + (1-\alpha)\psi,\ \psi \sim \text{initializer}. \tag{17}$$

With a well-tuned hyperparameter α, S & P is able to restore the network's plasticity while retaining the knowledge. In the context of deep RL, the SP-SPR agent uses soft resets

4. In the original work by Ash and Adams [3], the reset is defined as $\theta_t = \lambda\theta_{t-1} + \gamma\psi$. We choose to use the definition that is more common in deep RL.

for the convolutional encoder combined with hard resets for the last layers to increase the number of gradient steps per environment steps on the Atari 100k benchmark [21, 43]. Using the same combination of hard and soft resets as SP-SPR, BBF incorporates a number of other improvements, such as deeper networks and discount annealing to set the current state-of-the-art on Atari 100k [80].

Looking at the advantages and disadvantages of each reset type, it stands out that a requirement for hard resets is the presence of a replay buffer. The experience stored in the buffer allows the agent to quickly recover lost knowledge after a reset [21, 69]. The application of hard resets is, therefore, confined to off-policy deep RL algorithms, whereas well-tuned soft resets are, in theory, also applicable to on-policy algorithms. In terms of mechanisms of action, Shrink and Perturb has been shown to improve a variety of metrics associated with loss of plasticity: It decreases the number of dead neurons, prevents vanishing gradients, and prevents weight norms from exploding [22]. However, it may be possible that resets have other positive effects on deep RL training orthogonal to mitigating loss of plasticity. For example, Xu et al. [89] hypothesize that S & P may improve exploration, presumably via inducing more rapid policy change [78].

5.2 Targeted Weight Resets

Instead of indiscriminately resetting the network or parts of it, algorithms in this family track some measure of utility for each neuron/layer and perform resets based on it. This allows resetting only the features that are likely affected by plasticity loss at the cost of adding additional computational [19, 22, 83] or memory burden [19]. A special case is the plasticity injection [70], which does not track any measure of utility but resets the last layers of a network in a very specific way to not change its outputs.

Continual Backprop (CBP) is a reset algorithm building on top of a heuristic measure of neuron utility [19]. After each gradient step, this measure is updated for each neuron and used to reset the $x\%$ least useful neurons in each layer. As a reset strategy, CBP replaces the ingoing weights of a neuron with fresh weights sampled from the initialization distribution and sets the outgoing weights to zero. Adam moments are also reset by CBP for the selected neurons. Lastly, the algorithm tracks a count for each neuron, measuring the number of steps since the last reset, preventing the same neurons from being reset over and over during training. The authors show promising results in small toy problems such as input-permuted MNIST or in proprioceptive RL environments. However, the necessity to track multiple metrics for each neuron and perform updates and resets at every step makes CBP expensive both in terms of memory and compute. This likely prohibits scaling it to larger networks and agents.

ReDo has been developed concurrently with CBP and uses a more lightweight notion of neuron utility that is easier to track. Instead of updating a utility measure at every gradient step, ReDo checks every x time step to see whether the per-layer normalized activations of a neuron are below a threshold τ. If they are, it resets all of these *dormant neurons* with the same strategy as CBP, namely resampling ingoing weights from the initialization distribution and setting outgoing weights to zero. On Atari, ReDo has improved the performance of DQN [63] and DrQ [91] agents substantially, particularly in games like SpaceInvaders or Seaquest, where plasticity loss is known to occur. For a SAC [33] agent, performance improvements have been more muted, likely due to proprioceptive continuous control agents exhibiting different dynamics in terms of dormant neurons than pixel-based Atari games [42].

UPGD is a noisy SGD variant focusing on both plasticity loss and catastrophic forgetting in continual learning [22]. It consists of two components. First, it adds Gaussian noise to the gradients. Second, it scales the learning rate by neuron utility U. Let θdenote the set of all weights of a neural network. Formally, a UPGD step can be written as:

$$\theta_{l,i,j} \leftarrow \theta_{l,i,j} - \alpha \left(1 - \bar{U}_{l,i,j}\right) \left(\frac{\partial L}{\partial \theta_{l,i,j}} + \xi\right), \tag{18}$$

where $\xi \sim \mathcal{N}(\mu, \sigma^2)$ and $\bar{U}_{l,i,j} \in [0,1]$ is the exponential moving average of the utility of parameter scalar i,j in the weight matrix of layer l. The term $\left(1 - \bar{U}_{l,i,j}\right)$ leads to important weights not being changed due to their utility being close to 1, whereas unimportant weights will be both updated through SGD and perturbed via Gaussian noise. Intuitively, this can be seen as a targeted reset weighted by a continuous measure of parameter utility. The utility approximates a causal measure of weight utility based on the intervention of "*How would the loss change if a parameter scalar would be set to zero $\theta_{\neg[l,i,j]}$?*". As this is infeasible to compute exactly for deep networks, the authors instead approximate it with a Taylor expansion around zero and current parameters θ[22]:

$$\begin{aligned} U_{l,i,j}(Z) &= L\left(\theta_{\neg[l,i,j]}, Z\right) - L(\theta, Z) \\ &\approx L(\theta, Z) + \frac{\partial L(\theta, Z)}{\partial \theta_{l,i,j}} (0 - \theta_{l,i,j}) + \frac{1}{2} \frac{\partial^2 L}{\partial \theta_{l,ij}^2} (0 - \theta_{l,i,j})^2 - L(\theta, Z) \\ &= \underbrace{-\frac{\partial L(\theta, Z)}{\partial \theta_{l,i,j}} \theta_{l,i,j}}_{\text{first-order utility}} + \underbrace{\frac{1}{2} \frac{\partial^2 L(\theta, Z)}{\partial \theta_{l,i,j}^2} \theta_{l,i,j}^2}_{\text{second-order utility}}. \end{aligned} \tag{19}$$

Here Z denotes a sample, e.g., a state-action pair in RL. In theory, the Taylor expansion can be made arbitrarily accurate. Practically, even second-order expansions are already very expensive to compute, which results in the first-order approximation having the best compute vs accuracy trade-off [22]. Even though Equation (18) states the update for SGD, the UPGD algorithm can also be extended to momentum-based optimizers like Adam [22, 44].

Plasticity Injection [70] builds on the idea that plasticity loss is usually concentrated in the later layers of a neural network [21] and resets last layers of an agent in a very specific way: At a step T, it freshly initializes the last two layers of the agent's such that its outputs and the number of learnable parameters stay the same. Denoting $\theta$to be the current parameters of an agent's head, θ'_1 a set of freshly initialized parameters, and θ'_2 a *frozen copy* of θ'_1. If we let h represent the agent's head as a function parameterized by a certain set of parameters and specify the Q-function in the following way

$$Q(\mathbf{s}) = \underbrace{h_\theta(\mathbf{s})}_{\text{frozen}} + \underbrace{h_{\theta'_1}(\mathbf{s})}_{\text{learnable}} - \underbrace{h_{\theta'_2}(\mathbf{s})}_{\text{frozen}} \tag{20}$$

plasticity injection produces *unaffected predictions* and *preserves the number of trainable parameters* [70]. Fulfilling these two desiderata allows the injection to be a diagnostic tool that can be used to verify the existence of plasticity loss.

5.3 Parameter Regularization

The idea of regularizing parameter norms originates from linear regression, where it is used to cope with overfitting [64]. Plasticity loss differs from overfitting in that the goal is not to learn a simpler model but rather to preserve the general ability of the model to learn. Parameter norm regularizers developed to tackle plasticity loss build on the fact that the initial weights allow for rapid adaptation to new targets. As such, they aim to preserve certain properties of the initial weights via regularization. Such properties include the rank of the learned representation, the rank of the network Hessian matrix, or the low magnitude of the learned parameters. For example, low weight magnitudes are deemed to be beneficial for training, likely by inducing a smoother optimization landscape through smaller gradient norms [58].

L2 Regularization or Weight Decay is the most prominent weight regularization method. When applied to linear regression, the resulting method is called ridge regression. Seen from a probabilistic perspective, L2 regularization assumes a standard Normal distribution as prior for the weights of layer l $\mathbf{W}_l$. Formally, the prior is of the form $p(\mathbf{W}_l) = \mathcal{N}(\mathbf{W}_l \mid \mathbf{0}, C\mathbf{I})$. From this assumption about the prior, one can derive a penalty term using the squared ℓ_2-norm of the weights for all n layers:

$$L_{\text{L2reg}}(\theta) = L(\theta) + \lambda \sum_{l=1}^{n} \|\mathbf{W}_l - \mathbf{0}\|_2^2, \tag{21}$$

where $\mathbf{W}_l^t$ are layer l's weight matrices, and λ is a hyperparameter that determines the strength of the regularization. While well-tuned L2 regularization is able to keep parameters magnitudes small, in the context of RL it often interferes with training, particularly for value-based agents [58].

Spectral Normalization is a technique for discriminator regularization in GANs [62]. It directly controls the ℓ_2 matrix norm of layer l's weight matrix by dividing that matrix by the largest singular value $\sigma_{\max}$:

$$\mathbf{W}_l \leftarrow \frac{\mathbf{W}_l}{\|\mathbf{W}_l\|_2} = \sigma_{\max}^{-1}\mathbf{W}_l \ . \tag{22}$$

By applying spectral normalization to all layers of the GAN discriminator, it is possible to constrain the network to be K-Lipschitz, where K is a hyperparameter to be tuned. In deep RL, it has been observed that spectral normalization has two desirable effects for gradient-based optimization. First, it prevents exploding gradients, which is an issue when scaling RL agent networks [11, 67]. Second, spectral normalization implicitly schedules the learning rate of the Adam optimizers, preventing performance plateaus [31]. This effect is akin to LayerNorm's implicit learning rate scheduling described in the previous paragraph [57]. Furthermore, it is more effective at mitigating overestimation bias than techniques specifically targeted at mitigating overestimation bias, such as clipped double Q learning [67]. Nauman et al. [67] also show that spectral normalization substantially reduces the number of dormant neurons [83] even compared to methods like ReDO that are designed to prevent dormant units [67]. Lastly, applying it to the penultimate layer of value-based RL agents decreases their hyperparameter sensitivity [31]. In summary, spectral normalization positively affects a wide variety of factors commonly associated with plasticity and is thus an easy-to-use method for reducing plasticity loss.

L2 Init builds on L2 regularization by using the initial weights as regularization targets instead of the origin [20]. As the initial weights are capable of quickly fitting targets, keeping the weights closer to the initial weights instead of zero is more desirable from the perspective of plasticity loss. Consequently, the resulting prior distribution for the weights of layer l at step t is $p(\mathbf{W}_l^t) = \mathcal{N}(\mathbf{W}_l^t \mid \mathbf{W}_l^0, C\mathbf{I})$, with mean $\mathbf{W}_l^0$ instead of $\mathbf{0}$. This leads to the following regularized objective:

$$L_{\text{L2Init}}(\theta^t, \theta^0) = L(\theta^t) + \lambda \sum_{l=1}^{n} \|\mathbf{W}_l^t - \mathbf{W}_l^0\|_2^2. \tag{23}$$

Drawing a connection to targeted reset methods from Section 5.2, the **L2 Init** regularizer can be seen as a method for resetting weights that are of low importance. Compared to Continual Backprop [19] or ReDo [83], L2 Init does not explicitly calculate utilities for neurons. Instead, it implicitly determines the regularization strength for a weight based on the loss: If a gradient update for a particular weight results in a large change in loss value $L(\theta_t)$, the relative importance of the regularizing term $\|\mathbf{W}_l^t - \mathbf{W}_l^0\|_2^2$ decreases. Conversely, if changing a weight has little effect on the loss, then the regularizing term moves this particular parameter closer to its initialization value.

2-Wasserstein regularization One problem of L2 Init is that it regularizes weights to their uninformative initial values θ^0. Lewandowski et al. [53] propose to regularize weights to their initial distribution instead of initial values. Equation (24) uses the squared 2-Wasserstein distance to enforce a small distance between the initial weight distribution p_0 and the current distribution p_t:

$$L_{\text{2-Wass}}(\theta^t, \theta^0) = L(\theta_t) + \mathcal{W}_2^2(p_t \parallel p_0), \tag{24}$$

$$\text{where} \qquad W_2^2(p_t \parallel p_0) = \sum_{l=1}^{n} \sum_{i=1}^{d} \left(\bar{w}_{l,t}^i - \bar{w}_{l,0}^i \right),$$

with $\bar{w}_{l,t}^i$ denoting the i-th largest parameter of layer l at step t. Equation (24) highlights a particular advantage of the squared 2-Wasserstein distance: The existence of a closed-form solution makes computations tractable and cheap. We can compare the 2-Wasserstein regularizer with L2 Init by writing out the L2 Init penalty in terms of individual weights:

$$R_{\text{L2Init}}(\theta_t, \theta_0) = \sum_{l=1}^{n} \sum_{i=1}^{d} \left(w_{l,t}^i - w_{l,0}^i \right).$$

As we can see, the Wasserstein regularizer is just L2 Init with *parameters sorted by their magnitude*. This subtle change allows for larger deviations of individual weights from their initial value while still keeping the current distribution over weights close to their initial distribution.

Weight Clipping is another technique for preventing parameter norm growth, which may be one of the driving factors of plasticity loss [23]. Please refer to Section 4.7 for more information on how this occurs. The idea behind weight clipping is simple: After each gradient update, ensure that weights are in a predefined range $[-b, b]$. Elsayed et al. [23] define $b = \kappa s_l$, where κ is a scaling hyperparameter and s_l are the bounds of the uniform distribution used to initialize the weight matrix $\mathbf{W}_l$ of layer l. The big advantage of weight clipping compared to weight decay or L2 Init [20] is that it does not bias the weights towards a particular point in parameter space; instead, they can freely move within the bounds $[\kappa s_l, \kappa s_l]$. Its big disadvantage is that it only works for uniform initialization like Kaiming Uniform [23, 34].

Parameter Regularization

- Weight decay or **L2 regularization** mitigates overfitting in regression models by regularizing weights towards a standard Normal prior [64]:

$$R_{\text{L2reg}}(\theta) = \lambda \sum_{l=1}^{n} \|\mathbf{W}_l - \mathbf{0}\|_2^2.$$

  Note that the term $\|\mathbf{W}_l - \mathbf{0}\|_2^2$ only regularizes the weight matrices; the bias terms don't contribute to overfitting.

- **SpectralNorm** enforces a Lipschitz-constraint on all weight matrices $\mathbf{W}_l$ of a network by dividing through the largest singular value $\sigma_{\max}$ of each weight matrix [62]:

$$\mathbf{W}_l \leftarrow \frac{\mathbf{W}_l}{\|\mathbf{W}_l\|_2} = \sigma_{\max}^{-1} \mathbf{W}_l \ .$$

  $\sigma_{\max}$ can be approximated efficiently through the power iteration algorithm.

- Dohare et al. [20] observe that the initial weights of a network possess high levels of plasticity and propose to *regularize towards the initial weights instead of zero weights* in their method **L2 Init**:

$$R_{\text{L2Init}}(\theta^t, \theta^0) = \lambda \sum_{l=1}^{n} \|\mathbf{W}_l - \mathbf{W}_0\|_2^2.$$

- Lewandowski et al. [53] extend the idea of parameter regularization by *regularizing towards the initial weight distribution* via the **squared 2-Wasserstein regularizer**:

$$R_{\text{2-Wass}}(\theta^t, \theta^0) = \sum_{l=1}^{n} \sum_{i=1}^{d} \left(\bar{w}_{l,t}^i - \bar{w}_{l,0}^i\right)^2 .$$

  Here $\bar{w}_{l,t}^i$ denotes the i-th largest parameter of layer l at step t.

- **Weight clipping** [23] truncates the weight matrices $\mathbf{W}_l$ for each layer l with the bounds $[\kappa s_l, \kappa s_l]$, where s_l are the bounds of the uniform distribution used to initialize the layer and κ is a hyperparameter.

5.4 Feature Rank Regularization

While it is unclear that feature rank collapse is the underlying cause of plasticity loss [56], it is a highly associated phenomenon for networks that have lost their ability to learn [19, 20]. Since the first observation that under-parameterized or low-rank representations are strongly associated with low performance in value-based deep RL [46], many algorithms have been designed to either explicitly or implicitly encourage high-rank representations.

Direct Singular Value Regularization Kumar et al. [46] note that the effective rank from Definition 1 is non-differentiable. To circumvent this issue, they propose a regularizer that minimizes the largest singular value $\sigma_{\max}^2(\phi(\mathbf{s}, \mathbf{a}))$ while jointly maximizing the smallest singular value $\sigma^2_{\min}(\phi(\mathbf{s},\mathbf{a}))$ of a representation $\phi(\mathbf{s},\mathbf{a})$ with parameters θ:

$$L_{\text{sing}}(\theta) = L(\theta) + \alpha \left(\sigma^2_{\max}(\phi(\mathbf{s},\mathbf{a})) - \sigma^2_{\min}(\phi(\mathbf{s},\mathbf{a})) \right). \tag{25}$$

L_{sing} is conceptually simple and applicable to any type of deep RL algorithm but suffers from issues with stability. To see how potential instabilities arise, note that the regularizer is globally minimized when all singular values are exactly the same, which is achieved by a constant representation: $\forall \mathbf{s},\mathbf{a} : \phi(\mathbf{s},\mathbf{a}) = \mathbf{0}$. This makes practical applications difficult as performance and stability are highly sensitive to the choice of the parameter α.

InFeR takes a more indirect approach towards maintaining a representation with high effective rank by utilizing a set of k auxiliary regression tasks [55]. First, the algorithm adds k auxiliary prediction heads g_i on top of the agent's encoder ϕ_{θ}. Before starting the training, InFeR creates a copy of the initialization weights for both the encoder and the auxiliary task head. These weights are used to generate the regression targets for the auxiliary task using training batches from the agent's replay buffer $\mathbf{x} \sim \mathcal{D}$:

$$L_{\text{InFeR}}\left(\theta^t, \theta^0\right) = \mathcal{L}(\theta) + \alpha \underset{\mathbf{x}\sim\mathcal{D}}{\mathbb{E}} \left[\sum_{i=1}^{k} \left(g_i(\mathbf{x};\theta) - \beta g_i\left(\mathbf{x};\theta_0\right)\right)^2 \right]. \tag{26}$$

Here β is a scaling hyperparameter for the random regression targets, and α is a coefficient for the regularization strength. Together with ensemble size k for the number of auxiliary tasks this introduces a number of different hyperparameters that must be tuned, which requires lots of compute. InFeR improves performance in games where plasticity loss is known to occur, such as Phoenix [70] as well as sparse-reward hard exploration games such as Montezuma's Revenge. The performance in sparse-reward settings may indicate the presence of confounding effects of InFeR due to improved exploration from the randomized regression task.

DR3 is a method developed to prevent feature co-adaptation between successive state-action pairs in value-based deep RL [47]. This co-adaption implicitly occurs due to an implicit regularization effect of SGD updates, leading to large feature dot products and eventually diverging Q-values. By penalizing large feature dot products, the implicit regularization effect of SGD can be mitigated:

$$L_{\text{DR3}}(\theta) = L(\theta) + c_0 \sum_{i\in\mathcal{D}} \phi\left(\mathbf{s}_i, \mathbf{a}_i\right)^{\top} \phi\left(\mathbf{s}'_i, \mathbf{a}'_i\right). \tag{27}$$

DR3 is not designed to explicitly maximize feature rank. However, experimental results show that it is effective at preventing rank collapse. At the same time, DR3 does not explicitly maximize rank and consequently has lower rank than a DQN agent in games such as Breakout. Despite its lower rank, DR3 still achieves superior performance, hinting at the relationship between effective rank and agent performance not being straightforward [47].

BEER adaptively regularizes the rank of a representation based on an upper bound of the cosine similarity between the representations of consecutive state-action pairs [35]. It builds on the insight that naively maximizing feature rank may lead to overly complex models and overfitting. The upper bound derived by He et al. [35] is a necessary condition for the optimal value function:

$$\cos\left(\phi(\mathbf{s},\mathbf{a}),\overline{\phi\left(\mathbf{s}',\mathbf{a}'\right)}\right) \leq \left(\|\phi(\mathbf{s},\mathbf{a})\|^2+\gamma^2\|\overline{\phi\left(\mathbf{s}',\mathbf{a}'\right)}\|^2-\frac{\|r\|^2}{\|\mathbf{w}\|^2}\right)\cdot \frac{1}{2\gamma\|\phi(\mathbf{s},\mathbf{a})\|\|\overline{\phi\left(\mathbf{s}',\mathbf{a}'\right)}\|}. \tag{28}$$

Here $\phi(\mathbf{s},\mathbf{a})$ is a state-action representation obtained from the penultimate layer of the agent, $\overline{\phi\left(\mathbf{s}',\mathbf{a}'\right)}=\mathbb{E}_{s',a'}\,\phi\left(\mathbf{s}',\mathbf{a}'\right)$ is the expected representation under the distribution for the next state-action pair, $\mathbf{w}$ are the weights for the final layer and $\|r\|^2$ is the squared reward for the transition. As minimizing the RL loss under this constraint is a non-convex optimization problem, the authors propose to use the constraint as a penalty term with weight β:

$$\begin{aligned} L_{\text{BEER}}(\theta) =&L(\theta)\\ &+\beta\,\text{ReLU}\Bigg(\cos\left(\phi(\mathbf{s},\mathbf{a}),\mathbb{SG}\overline{\phi\left(\mathbf{s}',\mathbf{a}'\right)}\right)\\ &-\mathbb{SG}\Bigg(\left(\|\phi(\mathbf{s},\mathbf{a})\|^2+\gamma^2\|\overline{\phi\left(\mathbf{s}',\mathbf{a}'\right)}\|^2-\frac{\|r\|^2}{\|\mathbf{w}\|^2}\right)\frac{1}{2\gamma\|\phi(\mathbf{s},\mathbf{a})\|\|\overline{\phi\left(\mathbf{s}',\mathbf{a}'\right)}\|}\Bigg)\Bigg). \end{aligned} \tag{29}$$

Compared to the conceptually similar DR regularizer from Equation (27), BEER is able to adaptively adjust the rank depending on the complexity of the task. In turn, this leads to better performance and more accurate Q-values when compared to InFeR [55].

The relationship between effective rank, feature rank collapse, and plasticity loss is, as of now, still murky. It seems clear that rank collapse does have a negative effect on a network's ability to learn [20, 32, 46], particularly for value-based deep RL agents. What's not so clear is whether maximizing effective rank is beneficial for learning. Here, He et al. [35] provide the first negative results. For offline RL, Gülçehre et al. [32] study in-depth how effective rank relates to a host of other training variables. Key findings include that rank collapse and performance are more strongly associated with ReLU networks than with TanH ones and that using high learning rates increases the chance of rank collapse. Another interesting finding of Gülçehre et al. [32] is the strong negative correlation between the number of dead units and effective rank. However, their work does not establish a causal relationship between the two quantities. To summarize: While a drop in effective rank seems to be a concomitant for networks that have lost their ability to learn, the nature and direction of a causal relationship between rank and plasticity has not been established as of now.

Feature Rank Regularization

- Given a state-action pair $(\mathbf{s}, \mathbf{a})$, Kumar et al. [46] propose a **singular value regularizer** to directly prevent rank collapse of a representation ϕ with parameters θ:

  $$R_{\text{sing}}(\theta) = \sigma^2_{\max}(\phi\left(\mathbf{s}, \mathbf{a}\right)) - \sigma^2_{\min}(\phi\left(\mathbf{s}, \mathbf{a}\right)).$$

  While the above penalty is computationally efficient, it is prone to representation collapse as $\theta = \mathbf{0}$ is a global minimizer of R_{sing}.

- **DR3** [47] penalizes large feature dot products between consecutive state-action pairs:

  $$R_{\text{DR3}}(\theta) = \phi\left(\mathbf{s}, \mathbf{a}\right)^\top \phi\left(\mathbf{s}', \mathbf{a}'\right).$$

  Despite not explicitly aimed at increasing feature rank, DR3 prevents rank collapse [47] and enhances performance [48].

- **InFeR** [55] preserves the rank of a representation by regressing $i = 1, \ldots, k$ auxiliary task heads $g_i(\cdot, \theta)$ on counterparts $g_i(\cdot, \theta_0)$ with frozen initial parameters θ_0:

  $$R_{\text{InFeR}}\ (\theta, \theta_0; \beta) = \sum_{i=1}^{k} \left(g_i(\mathbf{x}; \theta) - \beta g_i\left(\mathbf{x}; \theta_0\right)\right)^2.$$

  The frozen target heads $g_i(\cdot, \theta_0)$ utilize a shared encoder ϕ_0 with frozen initialization weights.

- **BEER** [35] enforces an upper bound on the cosine similarity between successive state-action representations derived from the Bellman equation:

  $$R_{\text{BEER}}(\theta) = \text{ReLU}\left(\cos\left(\phi(\mathbf{s}, \mathbf{a}), \mathbb{SG}\overline{\phi\left(\mathbf{s}, \mathbf{a}\right)}\right) - \right.$$
  $$\left. \mathbb{SG}\left(\left(\|\phi(\mathbf{s}, \mathbf{a})\|^2 + \gamma^2 \|\overline{\phi\left(\mathbf{s}', \mathbf{a}'\right)}^2\| - \frac{\|r\|^2}{\|\mathbf{W}\|^2}\right) \frac{1}{2\gamma \|\phi(\mathbf{s}, \mathbf{a})\| \|\overline{\phi\left(\mathbf{s}', \mathbf{a}'\right)}\|}\right)\right),$$

  where $\mathbb{SG}$ denotes stopped gradients, $\mathbf{w}$ are the final layer weights, and $\|r\|^2$ is the squared reward for DQN-style sample backups.

5.5 Activation Functions

When looking at loss of plasticity through the perspective of dead/dormant neurons, we can observe that some activation functions lead to more dead neurons than others. This is particularly relevant for ReLU, which is the most commonly used activation function in deep RL. Therefore, it is reasonable to assume that a different activation function can improve a network's plasticity. Below, we will present three activation functions used in the literature to prevent plasticity loss, with the last two being specifically designed to maintain plasticity. We will use a scalar input $x \in \mathbb{R}$ for all our definitions as these functions are applied element-wise.

Parameterized Exponential Linear Units (PELU) are a generalization of the ELU [16] activation function with learnable parameters [30]. This allows PELU to adjust the activation slopes, yielding an ability to respond to distribution shifts akin to rational activations described below. Using α and β to denote the learnable parameters, PELU is defined as

$$\text{PELU}(x) = \begin{cases} \frac{\alpha}{\beta} h & x \geq 0 \\ \alpha \left(e^{\frac{h}{\beta}} - 1 \right) & x < 0. \end{cases} \tag{30}$$

It has been shown to improve performance over ReLU and CReLU agents, particularly in environments with heavy input distribution shifts [18].

Concatenated ReLU (CReLU) Originally developed in the context of image classification, the CReLU activation function in Equation (31) concatenates the ReLU output with its negation [81]:

$$\text{CReLU}(x) = \Big[\text{ReLU}(x), \text{ReLU}(x) \Big] \ . \tag{31}$$

The obvious effect of applying this activation function is that subsequent layers now must have twice the number of input features, which requires care when comparing CReLU networks to regular ReLU networks. Regarding plasticity loss, Abbas et al. [1] apply CReLU to deep RL agents and observe three benefits. First, Rainbow + CReLU is able to change its parameters even late in training and after multiple task shifts. Second, CReLU improves gradient flow through the network and prevents gradient collapse. Third, CReLU effectively mitigates dead neurons because CReLU$(x) = 0$ if and only if $x = 0$ compared to ReLU, where ReLU$(x = 0)$ if $x \in (-\infty, 0]$.

Adaptive Rational Activations build on the fact that ratios of polynomials can approximate any continuous function. Denoting P and Q as two polynomials with input x, an adaptive rational activation is defined as

$$\text{R}(x) = \frac{\text{P}(x)}{\text{Q}(x)} = \frac{\sum_{j=0}^{m} a_j x^j}{1 + \sum_{k=1}^{n} b_k x^k}, \tag{32}$$

where $\{a_j\}_{j=0}^m$ and $\{b_k\}_{k=1}^n$ are $m+1$ and n learnable parameters, respectively. Rational activations have two desirable properties from the perspective of plasticity loss, namely that they can adjust their parameters depending on input distribution shifts and that they can approximate residual connections. To stabilize them in practice, one uses the absolute sum in the denominator and sets $m = n + 1$ [18] [5].

It seems clear that activation functions can help mitigate plasticity loss to some extent by reducing the number of dead units and improving gradient flow through the network. However, multiple experiments have shown that plasticity loss still occurs to varying levels with different activation functions [19, 83]. Therefore, we conclude that while choosing specific activation functions may alleviate the symptoms of plasticity loss, the activation function itself is not the root cause of the phenomenon.

5. Delfosse et al. [18] use $m = 5$ and $n = 4$ throughout their paper.

Activation functions

All activation functions below are defined for a scalar input $x \in \mathbb{R}$:

- **PELU** [30] is a variant of ELU with learnable slope parameters α and β parameters:

$$\mathrm{PELU}(x) = \begin{cases} \frac{\alpha}{\beta} h & x \geq 0 \\ \alpha \left(e^{\frac{h}{\beta}} - 1 \right) & x < 0. \end{cases}$$

- **CReLU** [1, 81] concatenates the ReLU activation with its negation, which leads to twice the number of inputs at the subsequent layer but prevents dead neurons except for exactly zero pre-activations:

$$\mathrm{CReLU}(x) = \Big[\mathrm{ReLU}(x), \mathrm{ReLU}(-x) \Big] .$$

- **Adaptive rational activations** [18] use a ratio of polynomials with learnable coefficients a_j and β_k as activation function, allowing the activations to adapt to input distribution shifts:

$$\mathrm{R}(x) = \frac{\mathrm{P}(x)}{\mathrm{Q}(x)} = \frac{\sum_{j=0}^{5} a_j x^j}{\left| 1 + \sum_{k=1}^{4} b_k x^k \right|}.$$

Compared to Equation (32), we directly use the values used by Delfosse et al. [18] here.

5.6 Categorical Losses

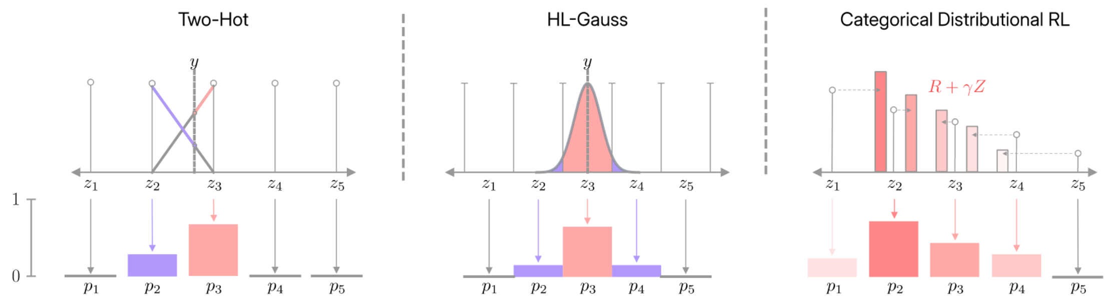

Figure 3: **Visualization of categorical losses for deep RL**. The two-hot representation [79] proportionally assigns probability mass to the two neighboring bins of a scalar target y. HL-Gauss [40] constructs a Gaussian with fixed standard deviation and integrates over each bin to obtain the corresponding probability mass. Distribution RL algorithms such as C51 [5] model the full return distribution. Figure taken from Farebrother et al. [24].

It is common knowledge among deep learning practitioners that it is easier to scale network sizes for classification tasks when compared to regression tasks. Even when regression is the actual task, re-formulating the learning problem using a cross-entropy loss is often beneficial [24, 40]. As explained in Sections 4.6 and 4.7, regression gradients are proportional to the loss, which in turn may lead to parameter norm growth. All categorical losses for deep RL have in common that they (a) require the rewards to be bounded to avoid distributions with infinite support, and (b) that the finite reward range is binned into M bins $(z_1, ..., z_M)$. Bounded rewards can usually be obtained by just clipping rewards within a range of $(-10, 10)$ [5]. One can then obtain a Categorical target distribution by either directly projecting a scalar target value y onto the bins [79] or by first constructing an auxiliary distribution $p(y|x)$ around y before the projection step [24]. In the following, we will present the three most popular approaches for converting regression into classification tasks used in deep RL.

Distributional RL (C-51) tries to estimate the full reward distribution instead of only its expected value [6]. This paragraph focuses on C-51 [5], an early representative of distributional RL that projects the return distribution directly onto a categorical distribution with $M = 51$ bins, giving rise to its name. The authors justify this approach compared to using a continuous distribution with the advantages of the categorical distribution in terms of expressivity and computational efficiency. C-51 then minimizes the forward KL-divergence between the network's distribution $Z(\mathbf{s}, \mathbf{a}; \theta)$ and the target distribution $Z(\mathbf{s}, \mathbf{a}; \tilde{\theta})$ by using a cross-entropy loss:

$$L(\theta) = \mathrm{D}_{\mathrm{KL}}(Z(\mathbf{s}, \mathbf{a}; \theta) \parallel Z(\mathbf{s}, \mathbf{a}; \tilde{\theta})). \tag{33}$$

Subsequently, more modern distributional RL algorithms such as QR-DQN or IQN have refined the idea of modeling the return distribution and yielded large gains in sample efficiency [6]. While it is still elusive as to what exactly causes the performance benefits of distributional RL, recent work suggests that using a cross-entropy loss instead of regression might be one of the driving forces [24]. However, modeling return distributions has other benefits apart from improved sample efficiency: For example, it allows quantifying risk in risk-sensitive applications.

Two-hot Representation The two-hot representation projects a scalar-valued regression target $y \in \mathbb{R}$ onto a categorical distribution by proportionally assigning probability mass to the two closest bins around y. The probability of the lower bin is $P(\lfloor c \rfloor) = \lceil c \rceil - c$, whereas the probability of the higher bin is the complementary probability $P(\lceil c \rceil) = 1 - P(\lfloor c \rfloor)$. All other bins are assigned zero probability mass. Using the binning strategy above, one can construct a categorical distribution that is usable as a target in a cross-entropy loss function. The scalar target y can then be recovered by taking the expectation of the corresponding categorical distribution. First popularized in the MuZero paper [79], the two-hot representation has since been noted to alleviate plasticity loss in multiple follow-up works, albeit at the cost of training stability [56, 58].

HL-Gauss uses two steps to project a scalar valued regression target $y \in \mathbb{R}$ onto a categorical distribution $Z(\mathbf{s}, \mathbf{a}; \tilde{\theta})$ with m bins $(z_1, ..., z_m)$ of width ζ. The target can be recovered as the expected value of the categorical distribution $y = \mathbb{E}[Z(\mathbf{s}, \mathbf{a}; \tilde{\theta})]$. In the first step of the projection, we construct a Gaussian distribution with mean y and a fixed variance σ^2: $\mathcal{N}(y, \sigma^2)$. Subsequently, we integrate over the Gaussian's density function in each bin z_i to obtain the corresponding probability mass p_i of the categorical distribution:

$$\begin{aligned} p_i(\mathbf{s}, \mathbf{a}; \tilde{\theta}) &= \int_{z_i - \varsigma/2}^{z_i + \varsigma/2} f_{Y|\mathbf{s},\mathbf{a}}(y \mid \mathbf{s}, \mathbf{a}) dy \\ &= F_{Y|\mathbf{s},\mathbf{a}}\left(z_i + \varsigma/2 \mid \mathbf{s}, \mathbf{a}\right) - F_{Y|\mathbf{s},\mathbf{a}}\left(z_i - \varsigma/2 \mid \mathbf{s}, \mathbf{a}\right), \end{aligned} \tag{34}$$

where $\tilde{\theta}$are the target network parameters. HL-Gauss then uses a cross-entropy loss to optimize the agent's value function [24]. Compared to the two-hot representation, HL-Gauss improves performance in several settings and yields more expressive representations. The authors hypothesize that two effects are at play here: First, spreading the probability mass leads to reduced overfitting due to an effect akin to label smoothing in supervised learning. Second, HL-Gauss generalizes across ranges of target values by exploiting the ordinal structure of a regression problem [24].

5.7 Distillation

Distillation is a technique for transferring the knowledge of one network architecture to a different architecture by training a "*student network*" on the outputs of a "*teacher network*" [36]. However, it can also be used to reduce negative side effects occurring during the training of the teacher network.

ITER is a method for mitigating "*transient non-stationarity*" — another term for plasticity loss — occurring during network training [38]. It periodically distills the actor and critic networks of a PPO agent, showing that this mitigates plasticity loss in ProcGen and supervised toy experiments. To improve the computational efficiency of ITER, the authors propose to update the student networks in parallel with the teacher's RL training. This has the advantage of not requiring a replay buffer for on-policy algorithms such as PPO and enables parallel training of the teacher and student.

Hare & Tortoise Networks combine the intuition behind periodic weight resets with distillation by building on top of target networks that have become ubiquitous in modern deep RL. Specifically, they train a "*hare*" network to rapidly learn a new task via SGD. The "*tortoise*" network is a copy of the hare network that is updated using an exponential moving average of the hare's parameters [52]. This dual architecture is analogous to the hippocampus and neocortex in the brain, allowing the tortoise network to slowly integrate useful information from the hare network over the course of training. Instead of periodic hard resets or soft resets with uninformative initial parameters [3], the hare is reset to the tortoise's parameters. This enables frequent resetting without losing knowledge accumulated via SGD and even provides a mechanism for escaping suboptimal local minima via a hard reset of the hare [52].

5.8 Other Methods

This section serves as a general receptacle for methods that haven't fit one of the previous categories. Many of them are well-known regularization techniques such as LayerNorm [4] or data augmentations that weren't designed to mitigate plasticity loss but also affect it.

Layer Normalization [4] is a normalization scheme developed to address some limitations of Batch Normalization [41], namely removing its dependence on the batch size and enabling an application to recurrent neural networks. It does so by standardizing a network's features using the per-layer l activation statistics:

$$\mathbf{a}_l \leftarrow \frac{\mathbf{a}_l - \mathbb{E}[\mathbf{a}_l]}{\sqrt{\mathrm{Var}[\mathbf{a}_l] + \epsilon}} * \boldsymbol{\gamma} + \boldsymbol{\beta}, \tag{35}$$

where $\mathbf{a}_l$ is the parameter matrix of layer l, $\boldsymbol{\gamma}$ a learnable scale parameter, $\boldsymbol{\beta}$ a learnable bias parameter, and ϵ a constant for numerical stability. Layer Normalization first surfaced as a mitigation strategy for plasticity loss in the study of Lyle et al. [56], which found that it maintains network plasticity while only minimally affecting task performance. Many subsequent works have by now corroborated these results [57, 58, 67, 68].

But what exactly causes the performance gains of LayerNorm? First and most obviously, it does what it was supposed to do at conception and maintains zero-main and unit-variance pre-activation statistics [4]. This is supposed to help with network training [58]. Lyle et al. [56] hypothesize that this is due to Layer Normalization's ability to reduce gradient covariance. A study by Nauman et al. [67] finds that Layer Normalization is very effective at mitigating overestimation bias, reducing dormant neurons [83], and preventing large gradient norms. Another answer is that pre-nonlinearity LayerNorm prevents unit linearization by stabilizing the pre-activation distribution [58]. This prevents divergence in value-based RL, which can also be achieved by using bounded activation functions [9]. Recently, Lyle et al. [57] study the effects of LayerNorm on deep RL agents in-depth and examine two mechanisms behind its performance improvements in detail. First, LayerNorm is able to reactivate dead ReLU neurons by providing them with non-zero gradients from the normalization statistics. This is an effect that has also been observed by previous work [67]. In fact, studies with transformer networks have shown that the gradients from normalization statistics are the driving force behind LayerNorm's benefits [90] and not the zero-mean, unit-variance activations that were previously hypothesized [4, 57]. Second, LayerNorm induces an implicit, parameter-norm-dependent learning rate schedule that might be required for deep RL agents to learn certain behaviors [57].

Data Augmentations have been the driving force behind impressive sample efficiency gains in pixel-based control. Conventional wisdom assumes that these gains are due to improved representation learning [50, 91, 92]. Ma et al. [59] challenge this assumption and provide evidence that data augmentations actually help prevent plasticity loss in the critic of a DrQ-v2 [92] agent. They support their hypothesis by measuring the fraction of active units, finding that including data augmentations prevents early plasticity loss as measured by a drop in the fraction of active units. Building on this insight, Ma et al. [59] propose to increase the number of gradient steps per environment step once the algorithm is past the early stage of plasticity loss.

Moment Resets Asadi et al. [2] examine the effect of Adam moment accumulation in the context of value-based deep RL. As many of these methods use target networks, they are effectively solving a sequence of optimization problems instead of a single problem with stationary targets. When examining the Adam moments directly after a target network update, Asadi et al. [2] find that the gradients for the new targets are orthogonal to the moments stored by Adam. As the moments update only slowly due to high exponential smoothing coefficients β_1 and β_2, a target network update is followed by a lengthy phase of unlearning. Resetting the moments provides a simple solution to this issue [2].

Activation Normalization p-norm is a normalization technique aiming to prevent saturated tanh units in continuous control actor networks [12]. It does so by dividing the features of the penultimate layer of the network $\phi(\mathbf{s})$ by their norm, i.e., $\phi_{\text{norm}}(\mathbf{s}) = \frac{\phi(\mathbf{s})}{\|\phi(\mathbf{s})\|}$. Afterward, the actions are generated by a linear transformation of the normalized features $\phi_{\text{norm}}(\mathbf{s})$, followed by a tanh activation function. While being targeted specifically developed specifically for the actor of DrQ-v2 [92], Bjorck et al. [12] also apply p-norm to the critic and find that regularizing both networks leads to substantially increased performance and training stability.

Discrete Representations build on vector quantization techniques and have shown impressive results and a number of advantages when applied to generative models [86]. Meyer et al. [61] examine their effect in continual deep RL and discover they also have beneficial effects there. In particular, one-hot representations seem to learn more general representations that are able to adapt faster to non-stationarities in the environment. In their work, they hypothesize that the sparse and binary nature of the learned embeddings drives the performance benefits. This is verified by showing that one-hot representations substantially outperform quantized ones despite their identical information content [61].

Sharpness-Aware Minimization (SAM) is an optimization technique for finding local minima with low curvature. Such flat minima are desirable because they are commonly associated with better generalization performance [26]. In the context of deep RL, Lyle et al. [56] hypothesize that loss curvature as measured by the largest eigenvalue of the network's Hessian matrix might be an underlying driver of plasticity loss. SAM formulates finding a flat minimum as a minimax optimization problem, where it first has to find a set of "*adversarial*" weights θ_{adv} that maximally increase the loss $L(\theta_{\text{adv}})$ within an ϵ-ball of radius ρ around the current parameters θ_t. As solving this problem exactly is costly, SAM uses a first-order Taylor expansion to find θ_{adv} by taking an ascending step along $\nabla_\theta L(\theta_t)$, with the step-size scaled to put θ_{adv} on the border of the ϵ-ball. We can then compute a gradient $\nabla_\theta L(\theta_{\text{adv}})$ that moves in the direction of a loss region with uniformly flat value. SAM applies proceeds to apply $\nabla_\theta L(\theta_{\text{adv}})$ *at the current parameters* θ_t for its update. Together, these two steps can be seen as a perturbation step followed by an update step, forming an alternating optimization algorithm. An advantage of SAM can increase plasticity even in pre-trained networks as it doesn't change the network architecture compared to other interventions such as LayerNorm. Its main disadvantage is that each update now requires two gradient steps instead of just one, making it computationally more expensive than SGD or momentum-based optimizers such as Adam [51].

5.9 Combined Methods

As the exact causes for plasticity loss are currently unclear and potentially multi-faceted [58], combining different regularizers targeting specific symptoms of plasticity loss is a reasonable strategy. As all of these regularizers except Normalize-and-Project [57] have been introduced and discussed in previous sections, the box below gives a brief overview of particularly well-performing combinations.

Normalize-and-Project (NaP) combines LayerNorm with a regularization step similar to SpectralNorm. Concretely, given a desired target norm ρ, NaP regularizes a layer l's parameters $\mathbf{W}$ to be within ρ [57]:

$$\mathbf{W}_l \leftarrow \rho \frac{\mathbf{W}_l}{\|\mathbf{W}_l\|} \tag{36}$$

Comparing Equation (36) with SpectralNorm (Equation (22), we can see that setting $\rho = 1$ and using the ℓ_2 recovers a variant of SpectralNorm.

Lyle et al. [57] proceed to note several important implementation details: First, linear layers now do not need a bias term, as LayerNorm already adds a learnable offset parameter. Second, an important question is how to handle these learnable parameters of LayerNorm in the projection step. The easiest solution provided by Lyle et al. [57] aligns with previous research in supervised learning: Simply remove them, they are not relevant for LayerNorm's performance gains [90]. Nevertheless, the authors of NaP discuss several other options in their work. Lastly, adding a projection step to networks using LayerNorm removes its implicit, parameter-norm-dependent learning rate schedule. In the case of deep RL, this may require a custom learning rate schedule mimicking the one induced by LayerNorm. Lyle et al. [57] find that a simple linear decay proportional to the otherwise occurring parameter norm growth is sufficient.

Regularization Combinations

- **LayerNorm + L2 regularization** has been proposed by Lyle et al. [58], who were also the first to use LayerNorm to mitigate plasticity loss [56]. In this setup, LayerNorm prevents preactivation distribution shifts, and L2 regularization prevents parameter norm growth and gradient explosion.
- **Resets + L2 regularization** is a synergistic combination of regularization techniques found in the broad study by [67], which performs particularly well for SAC [33] on the Meta-World benchmark [93].
- **LayerNorm + Resets** has been shown to yield a large performance boost for SAC [33] on DeepMind control suite by the same authors [67].
- **PLASTIC** combines the CReLU activation function with sharpness-aware optimization, Layer Normalization, and head resets for strong performance gains on Atari-100k and DeepMind control suite [51].
- **BRO** [68] is a state-of-the-art algorithm for proprioceptive continuous control tasks. From a plasticity perspective, it combines full network resets with LayerNorm and weight decay. Additionally, BRO uses a bespoke residual network, optimistic exploration, and a quantile network for the critic.
- **Normalize-and-Project (NaP)** combines LayerNorm with a version of Spectral-Norm, projecting the weights onto a ball with fixed radius ρ instead of the unit ball [57]:

  $$\textit{Normalize}: \mathbf{a}_l \leftarrow \frac{\mathbf{a}_l - \mathbb{E}[\mathbf{a}_l]}{\sqrt{\mathrm{Var}[\mathbf{a}_l] + \epsilon}} * \boldsymbol{\gamma} + \boldsymbol{\beta}\ ; \quad \textit{Project}: \mathbf{W}_l \leftarrow \rho \frac{\mathbf{W}_l}{\|\mathbf{W}_l\|}$$

  When implementing NaP, the scale and offset parameters of LayerNorm can often be removed [57, 90].

5.10 Discussion of Mitigation Strategies

One thing stands out when looking at the strategies currently proposed to mitigate plasticity loss: Many of them ”treat” a particular pathology of plasticity loss, similarly to how medication against a headache treats that particular symptom of an underlying disease. Recalling the findings of Section 4, this is rather unsurprising, as a deeper understanding of what causes plasticity loss is just now being developed.

Similarly, the mechanisms behind the performance gains are just now beginning to be better understood. For example, LayerNorm’s performance benefits have been attributed to its ability to maintain zero-mean and unit-variance pre-activations [4], when in reality, it seems to be the case that it effectively prevents dead units [90] and induces an advantageous learning rate schedule [57]. On the same front, recent insights into BatchNorm [41] have shown that it can effectively regularize a Q-function such that target networks are superfluous [9], but at the same time, BatchNorm also makes agents myopic [29]. With a better understanding of the causal mechanisms behind plasticity loss, such discoveries will be made more likely.

With the two previous paragraphs in mind, it is, therefore, no surprise that current SOTA methods often combine multiple regularizers, each treating an individual aspect of plasticity loss. For example, PLASTIC employs the SAM optimizer and LayerNorm to deal with the loss of input plasticity and uses resets combined with the CReLU activation function to improve target plasticity [51]. BRO [68] and BBF [80] are agents in a similar vein, as they both use multiple regularizers to successfully mitigate plasticity loss and improve sample efficiency.

6. Factors Influencing Plasticity Loss

Non-stationarity Section 4.5 discusses different types of non-stationarity as possible causes of plasticity loss. Clearly, the presence, type, and extent of non-stationarity have an influence on networks losing plasticity. How this can be quantified and used to strategically deploy specific remedies is currently not yet determined. The extent of target non-stationarity seems to play a clear role in plasticity loss [38, 83], but whether input non-stationarity plays a similarly important role or not is not yet clear [51, 83]. For example, offline RL seems to be less affected than online RL [83]. In supervised learning, task shifts are causing plasticity loss to some extent [3, 7], but this depends on how different the tasks are, which is hard to quantify. Lastly, a change in the dynamics of an MDP is associated with plasticity loss [1]. Given that agents employing TD learning implicitly learn about an MDP's transition function, it seems likely that their learning ability suffers more from dynamics changes compared to other types of RL algorithms [54].

Activation Function The choice of activation function can substantially affect loss of plasticity, mainly due to its effects on gradient propagation. Clearly, ReLU's propensity to incur dead neurons over the course of the training may contribute to plasticity loss. Using specific activation functions that have been shown well to work under non-stationarity such as CReLU [1] or rational activations [18] is likely to improve performance. However, the choice of activation function is no panacea: LeakyReLU should be less prone to dead neurons but still suffers from plasticity loss [83], maybe due to linearized units [58] or unbounded activations [9]. And while TanH nonlinearities guarantee bounded activations, they can still saturate, reducing performance [12]. Above all, not all activation functions may lead to the same task performance, requiring trade-offs to be made [19].

Optimizer The choice of optimizer, particularly when it uses moments for its updates, can impact plasticity loss. Equation (37) shows the update rule of Adam [44]:

$$u_t = \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}, \tag{37}$$

where $m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla_\theta f(\theta_{\theta_t})$ is the bias-correct first moment, $v_t = \beta_2 v_{t-1} + (1 - \beta_2) (\nabla_\theta f(\theta_{\theta_t}))^2$ the bias-corrected second moment, and ϵ a constant added for numerical stability.

In deep RL, the updates can diverge because of $\hat{m}_t$ being updated more aggressively than $\hat{v}_t$ due to $\beta_1 > \beta_2$. This leads to larger values being divided by small values due to gradients scaling proportionally with errors in regression problems such as value-based deep RL [2, 56]. Additionally, the ubiquitous use of target networks in value-based deep RL means that after a target update, the moments and the current optimization objective are not aligned. This manifests itself in new gradients being orthogonal to the current moment

Table 1: **Overview over causes and mitigation strategies for plasticity loss**. For the **causes**, we use the following abbreviations: Saturated Units (SU), Rank Collapse (RC), First-Order Effects (GOE), Second-Order Effects (SOE), Non-stationarity (NS), Regression loss (RE), Parameter Norm Growth (PNG), and Early Overfitting. For the **mitigation strategies**, we use Non-targeted Weight Resets (NW), Targeted Weight Resets (TW), Parameter Regularization (PR), Feature Rank Regularization (FR), Activation Functions (AF), Categorical Loss (CL), Distillation (DI), LayerNorm (LN), Moment Resets (MR). Methods that just occur once, e.g., data augmentation, are spelled out.

Name	**Cause**	**Mitigation Strategies**
p-norm [12]	SU	p-norm
Continual Backprop [19]	SU, NS	TW, MR
UPGD [22]	SU, NS	TW
ReDo [83]	SU, NS	TW, MR
DrM [89]		NW
”Revisiting plasticity” [59]	NS	Data augmentation, RR scaling
Singular value regularizer [46]	RC, NS	FR
InFeR [55]	RC, NS	FR
BEER [35]	RC	FR
Regularization in offline RL [32]	RC, SU	FR
ITER [38]	NS	DI
LayerNorm + L2 reg [56]	NS, FOE, SOE, RE	LN, PR, CL
Wasserstein regularizer [53]	SOE	PR
CReLU [1]	NS, FOE	AF
Resets [69]	EO	NW
SR-SAC [21]	EO	NW
BBF [80]	-	NW, PR
L2 Init [20]	NS	PR
LayerNorm + L2 reg [58]	NS, RE, FOE, SU, PNG	LN, PR
SpectralNorm + RL[11]	FOE	PR
HL-Gauss[24]	RE, NS	CL
Normalize-and-Project [57]	PNG, SU, FOE, learning rate schedule	LN, PR
”Bitter lesson” [67]	SU, FOE, PNG	LN, PR, NW
BRO [68]	-	LN, PR, NW, CL

estimates and requires several updates to wear off. A remedy is resetting the moments after each target network update [2].

An especially interesting case in Adam is the hyperparameter ϵ. In deep RL, it has become best practice to deviate from the low default values used in current frameworks [6], opting for higher values instead. For example, Google's dopamine framework uses a default of $1.5e-4$, more than a 1000× higher value [14]. This helps prevent Adam moments from diverging in the presence of task shifts, e.g., when the target networks are being updated.

Optimization Hyperparameters In this place, we want to provide a brief overview of some optimization hyperparameters that likely have an effect on plasticity loss. These are the learning rate, the total number of training steps, and the batch size. In supervised learning, larger learning rates add perturbations that may help escape suboptimal local minima in the pretrain-finetune paradigm [7]. In offline deep RL, on the other hand, using high learning rates is prone to rank collapse and an increase in dead neurons [32]. Additionally, Lyle et al. [58] have shown recently that implicit learning rate scheduling may play a more prominent role in deep RL than previously thought. Much work is left in understanding the interplay between learning rate and plasticity loss, but it seems clear that properly tuning the learning rate can catastrophic loss of plasticity, mainly when using adaptive optimizers such as Adam [32, 56].

Another parameter affecting plasticity loss is the total update budget. Intuitively, the longer a network is trained on one task before shifting to the next task, the more likely it is that it has lost plasticity.

Lastly, the batch size may affect plasticity loss as well. Here, there are conflicting reports: Obando-Ceron et al. [71] report improved results when using smaller batch sizes, likely due to more noisy gradients and less gradient collinearity. However, in offline deep RL, using larger batch sizes allows for maintaining strong performance even with larger learning rates, whereas smaller batch sizes are associated with dead units [32]. On the contrary, Nauman et al. [68] find that lower learning rates work better to scale online continuous control. Therefore, we advise manually tuning the batch size for the specific environment.

Regularization Many commonly used regularizers from supervised deep learning affect plasticity [67]. LayerNorm and SpectralNorm may be very effective at preventing gradient explosion, particularly when training with high replay ratios [67]. Recently, Lyle et al. [57] have shed light on how LayerNorm can even prevent dead or dormant units by providing gradients from its normalization statistic [90]s. Both SpectralNorm and LayerNorm also significantly reduce overestimation bias in SAC agents [67]. L2 regularization prevents parameter norm growth, which is hypothesized to be one of the main drivers of plasticity loss [23, 58]. Unsurprisingly, it is therefore used in the current SOTA agents for Atari-100k [80] and proprioceptive continuous control [68]. The usage of BatchNorm [41] in deep RL agents is less common despite generalization benefits [17], likely because integration into target networks is non-trivial [9]. An advantage of the well-established regularizers mentioned above over domain-specific methods [23, 53] is that they are battle-tested and efficiently implemented in common deep learning frameworks such as Pytorch [77]. As of now, it is still an open question whether there are other regularization techniques that may

6. For example, Pytorch uses a default value of 10^{-8}.

help with plasticity loss and what other RL-specific issues they may be able to alleviate apart from overestimation bias.

Objective Function As regression losses are less amenable to neural network optimization than classification losses [24], it may seem prudent to use cross-entropy losses as default. Having gradients that are proportional to error magnitude seems to clearly contribute to several pathologies associated with plasticity loss, such as large gradients and growing parameter norms. However, it is not entirely clear which type of categorical loss is best for which kind of problem. For example, HL-Gauss yields great results when applied to DQN [63] or CQL agents [45] in discrete-control environments. For continuous control, Nauman et al. [68] found implicit quantile networks [6] to be superior to alternative objectives for SAC agents [33]. While it is, therefore, sensible to try out different loss reformulations, there is no conclusive evidence of one being consistently better than the others.

Network Architecture Compared with supervised learning, most deep RL agents have a somewhat peculiar architecture: For proprioceptive control tasks, their networks are usually small MLPs, especially when compared with modern deep networks [24, 68]. For pixel-based control tasks, many algorithms still build on top of the Nature-CNN architecture presented in the original DQN paper [63]. For example, SR-SPR still uses the same CNN trunk as the one used in DQN [21]. Recently, more modern algorithms have used ResNet-style architectures [68, 80] to scale up the agent networks. However, the field of deep RL is still only at the beginning of utilizing large-scale networks, with the number of parameters peaking at $\approx 80M$ [24] due to optimization issues. This is despite findings that bigger agents may be somewhat more robust to plasticity loss [55, 56]. Network width, in particular, seems to help alleviate plasticity loss to some extent, although even very wide networks cannot completely mitigate it [56]. Deeper networks, on the other hand, don't seem to be able to mitigate plasticity loss. Another successful strategy for increasing the network sizes in deep RL might be to use residual architectures in conjunction with commonly used deep learning regularization such as LayerNorm, SpectralNorm, or L2 regularization [68, 80].

Environment Properties Deep RL environments come in various ways, and it seems almost certain that some of the properties of specific environments exacerbate plasticity loss. For Atari games, different games possess different levels of input non-stationarity. Delfosse et al. [18] accordingly classify games into the categories *stationary*, *dynamic*, and *progressive* environments. In stationary environments, the input distribution doesn't change at all. Dynamic environments have shifts in their input distribution, occurring independently of the current policy. An example of this is the game Asterix, where the agent receives gear such as a helmet as the game progresses. Progressive environments differ from dynamic environments because the input distribution shifts depend on the current policy. A typical environment in this category is JamesBond, where the agent progresses to levels with different characteristics [18]. In continuous control, and particularly in the DeepMind control suite Tassa et al. [85], the action dimension also plays a crucial role in plasticity loss. DOG and Humanoid are notoriously difficult benchmarks due to the agent having to control many different joints simultaneously. When training agents in these environments, the large number of dimensions in the action space leads to diverging gradients for the critic, and networks quickly lose plasticity [67]. These different environment properties between, e.g. DM Control and MetaWorld, lead to fluctuating performance for different regularizers [67].

7. Current State and Future Directions

In this section, we will first describe and discuss current trends concerning research on plasticity loss. Then, we will suggest some directions for future research.

7.1 Discussion of Current Trends

From the perspective of causes, the field has come a long way in the last few years. Compared to simply noting training instabilities [63] or gradient issues [11], we now have a much deeper understanding of which factors of plasticity loss might cause which pathologies. An example is large-mean regression causing exploding parameter norms causing sharpness in the optimization landscape [58].

Stepping back a bit and trying to distill trends in the field, we notice that well-established regularizers such as LayerNorm, BatchNorm, or weight decay are becoming more and more common to deal with plasticity loss in particular, but also other deep RL optimization issues [68, 80]. The reason for this is that they're easy to implement, efficient to compute, and usually have few hyperparameters. In contrast, customized variants of SGD that require expensive tracking of various metrics on top of a myriad of hyperparameters are rarely used in SOTA agents.

Drawing a conclusion, there seems to be a "bitter lesson" of plasticity loss: It seems that general network regularization will prevail over domain-specific methods purely targeting plasticity loss. A similar observation has recently been made regarding overestimation bias, where it has been found that general network regularizers such as LayerNorm are often more effective at preventing overestimation than domain-specific pessimistic learning algorithms [67].

Given the difficulty of theoretical research concerning everything related to deep learning, it seems likely that future progress in the field will be driven by empirical research. Developing theory is already highly difficult for RL in general and even more so related to deep RL optimization issues. Nevertheless, the community is progressing towards a better theoretical understanding of plasticity loss and its potential remedies [29]. Regarding the empirical research in the field, recent advances in just-in-time compilation [13] and GPU-accelerated environments [27, 49] have allowed the community to massively scale its experiments. This democratization of computational resources will likely accelerate research on plasticity loss, just as it will generally accelerate research in deep RL.

7.2 Directions for Future Research

7.2.1 What are the Causal Mechanisms behind Plasticity Loss?

The causal relationships between different aspects and pathologies in plasticity loss are not yet fully understood. Non-stationarity likely plays a role [38, 52, 58], but how different types of non-stationarity interact with different optimization problems is shrouded in mystery. Dividing non-stationarity into different aspects and examining them separately has yielded some progress [51, 52]. However, it is still challenging to quantify the extent of non-stationarity caused by, for example, a deep RL environment. Making progress in this area would make it possible to determine the strength of regularization based on the level of non-stationarity of the optimization problem. This is an exciting direction for future work due to many of the methods tackling plasticity loss interfering with solving the task [56].

Looking closer at the learning problem itself, a growing body of evidence hints at regression losses causing parameter norm growth, which in turn results in many of the commonly observed pathologies of networks that lost their plasticity [24, 58]. Large-mean regression can exacerbate this issue [58]. However, plasticity loss is also an issue for non-stationary classification, if maybe not to the same extent as for regression problems [38, 58]. Trying to develop causal relationships between different factors and symptoms of plasticity loss would allow a more fundamental understanding of the problem and make developing regularizers that interfere less with task optimization easier.

Lastly, it is clear that the initialization weights are more amenable to optimization and do not suffer from plasticity loss. This fact is used by some methods to regularize the parameters towards their initial values or initial distribution during training [20, 53]. But what exactly makes the initial weights so good for learning? It seems that zero-mean, unit-variance pre-activations are helpful [57], but are there other factors at play? By borrowing from the extensive literature on neural network initialization, it might be possible to arrive at definitive properties that are necessary to maintain the learning ability of an agent.

To summarize, arriving at a more abstract conceptual understanding that connects different facets of plasticity loss is the most pressing area for future research. While recent studies have made substantial progress in this respect [56, 58, 67], a "grand theory" of plasticity loss is not yet in sight.

7.2.2 A call for broader evaluation

Section 6 mentions environment properties as one of the possible causes of plasticity loss. Nauman et al. [67] confirm this hypothesis to the extent that different environments seem to suffer from lost plasticity to a different extent and that different pathologies occur in different levels of severity. For example, in DM Control DOG, large gradient norms pose a substantial optimization challenge to SAC agents. On other environments in DM Control, these large-norm gradients don't occur [67, 85]. Similarly, large parameter norms seem to harm agent performance on Metaworld [93] but don't affect the DM Control environments [67]. These effects result in fluctuating performance between different regularizers on different benchmarks. For example, while LayerNorm is highly useful for DM Control DOG, it doesn't improve performance for small networks on MetaWorld [67].

Here, the GPU-accelerated environment suites mentioned previously provide a unique opportunity for broadening our understanding of plasticity loss. What types of agents suffer the most from it? In which environments does it occur most severely? Developing abstract, measurable properties of environments that are associated with plasticity loss would dramatically help to build a more coherent understanding of the phenomenon. Additionally, it would allow deploying specific regularizers based on how strong the target environment causes plasticity loss, enabling more targeted mitigation of the phenomenon for practitioners. Lastly, it would enable testing whether some regularizers have unintended side effects and how harmful these are. For example, it is well-known that LayerNorm makes a network input scale-invariant, mapping $\alpha\mathbf{x}$ and $\mathbf{x}$ to the same output [57]. This could lead to potentially harmful state aliasing, e.g., by mixing Euclidean coordinates.

7.2.3 How do well-known regularizers actually work?

State-of-the-art agents currently deploy a range of well-known regularization techniques from deep learning, such as LayerNorm [68], L2 regularization ¸[68, 80], or BatchNorm [9]. But how exactly these interact with deep RL objectives and what the mechanisms driving their improvements are is not very well understood, both from an empirical as well as from a theoretical perspective.

On the empirical side, there is growing consensus that applying LayerNorm to deep RL agents is beneficial [58, 67]. Whether these benefits are due to LayerNorm maintaining zero-mean, unit-variance pre-activations [57], or due to it preventing dead neurons through gradients from normalization statistics [57, 90], or due to it centering and scaling gradients, or whether it is all of the above [90]: We currently don't yet know. Similar analyses can likely be made for other standard regularization techniques such as SpectralNorm.

Theoretically, these regularizers might have implicit effects on the RL formalism. For example, Bhatt et al. [9] use batch normalization to remove the need for a target network. This yields substantial performance gains for the standard, dense-reward continuous control tasks from DM Control. However, later work has shown that under mild assumptions, this provably leads to myopic behavior in linear agents with growing batch sizes [29]. Whether this is a problem in practice that outweighs the potential benefits of BatchNorm is still very much unclear.

7.2.4 What are the Links between Plasticity Loss and Established Deep RL Issues?

Overestimation bias is an issue plaguing any algorithm of the Q-Learning family: Because the argmax over Q-values uses the same network to select the target action and obtain the corresponding Q-value, the noise in both processes is correlated, leading to overestimation bias [84]. This issue has spawned many domain-specific solutions, ranging from classical double Q-Learning [87] to more recent approaches specifically targeting deep RL such as Generalized Pessimism Learning [15]. Are these domain-specific approaches really the best way to tackle overestimation bias? Recent work in plasticity loss suggests, in fact, no as an answer. In a broad study, Nauman et al. [67] find that well-known network regularization techniques such as LayerNorm [4] or SpectralNorm [62] are far superior at mitigating overestimation bias than GPL or commonly used clipped double Q-learning (CDQ). Removing CDQ is actually one of the most performance-critical components of the current SOTA algorithm for proprioceptive control [68]. Other works also connect plasticity loss and overestimation bias [18].

Similar stories can be found for other algorithmic components. ϵ-greedy may make sense in tabular Q-learning, but in deep RL, an optimization artifact called policy churn is actually driving exploration [78]. Baird's counterexample [39] can easily be solved by modern, regularized Q-Learning even though function approximation should provably diverge [29]. Target networks as a stabilization technique can be made obsolete by proper regularization [9, 29], vastly improving the computational efficiency of Q-learning algorithms. These examples highlight a crucial insight: **Intuitions from tabular RL often don't transfer directly to deep RL and may even be harmful to understanding**.

What other mysteries lie at the intersection between RL, deep network optimization, and regularization? Section 7.2.5 provides an outlook from an exploration perspective. Many stabilizing hacks used in current algorithms, such as replay buffers, pessimism, or target networks, may prove unnecessary once we correctly understand the interplay between deep network training and RL [29, 68]. Looking at issues like the deadly triad [88] or the phenomenon of passive learning [76] from an up-to-date network optimization and regularization perspective is an intriguing direction for future work. Deep RL is learning its bitter lesson [67]: Over-engineered, domain-specific approaches are being replaced by general neural network regularizers.

7.2.5 Plasticity, Agent Behavior, and Exploration

Modern neuroscience has made great strides in linking patterns of neuronal activity to certain behaviors, moods, and feelings. In contrast, deep learning and deep RL, in particular, are only at the beginning of establishing links between agent behavior and neuron activity, making this an exciting direction for future research. In a seminal work, Xu et al. [89] have connected the phenomenon of neuron dormancy [83] with reduced activity in pixel-based continuous control agents. Resetting these neurons causes immobile agents to "*activate*" and become. In the future, one could envision exploration algorithms that modulate specific neurons in the network, causing the agent to explore its environment in a more directed fashion compared to current approaches. In exploration terms, this would be called *deep* exploration, which refers to coherent long-term exploratory behaviors. In contrast, *dithering* algorithms such as ϵ-greedy cannot create long-term strategies and therefore struggle with hard exploration problems [74].

Another avenue is to investigate the success of current algorithms through a lens of plasticity. Schaul et al. [78] have recently shown that exploratory behavior of DQN agents is not due to ϵ-greedy as previously assumed, but instead depends on the agent's frequent action changes due to neural network optimization called *policy churn*. While they did not consider plasticity in their work, it is reasonable to assume that when networks lose their ability to learn, this also affects policy churn and, in turn, an agent's ability to explore.

Lastly, regularizing and mitigating plasticity loss has side effects that may allow the design of more efficient algorithms in general. One such example is clipped double Q-Learning, which has been an integral component in many foundational continuous control algorithms [28, 33]. Recently, it has been shown that this pessimistic bias is actually harmful to performance, hurting the sample efficiency of these agents [66]. By applying insights from plasticity loss and proper regularization to these agents, it is possible to improve exploration without suffering from overestimation bias, vastly improving the performance of these agents [68].

8. Conclusion

References

[1] Z. Abbas, R. Zhao, J. Modayil, A. White, and M. C. Machado. Loss of plasticity in continual deep reinforcement learning. In S. Chandar, R. Pascanu, H. Sedghi, and D. Precup, editors, *Conference on Lifelong Learning Agents, 22-25 August 2023, McGill University, Montréal, Québec, Canada*, volume 232 of *Proceedings of Machine Learning Research*, pages 620–636. PMLR, 2023. URL `https://proceedings.mlr.press/v232/abbas23a.html`.

[2] K. Asadi, R. Fakoor, and S. Sabach. Resetting the optimizer in deep RL: an empirical study. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL `http://papers.nips.cc/paper_files/paper/2023/hash/e4bf5c3245fd92a4554a16af9803b757-Abstract-Conference.html`.

[3] J. T. Ash and R. P. Adams. On warm-starting neural network training. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL `https://proceedings.neurips.cc/paper/2020/hash/288cd2567953f06e460a33951f55daaf-Abstract.html`.

[4] L. J. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *CoRR*, abs/1607.06450, 2016. URL `http://arxiv.org/abs/1607.06450`.

[5] M. G. Bellemare, W. Dabney, and R. Munos. A distributional perspective on reinforcement learning. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 449–458. PMLR, 2017. URL `http://proceedings.mlr.press/v70/bellemare17a.html`.

[6] M. G. Bellemare, W. Dabney, and M. Rowland. *Distributional Reinforcement Learning*. MIT Press, 2023. `http://www.distributional-rl.org`.

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

[8] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. de Oliveira Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang. Dota 2 with large scale deep reinforcement learning. *CoRR*, abs/1912.06680, 2019. URL `http://arxiv.org/abs/1912.06680`.

[9] A. Bhatt, D. Palenicek, B. Belousov, M. Argus, A. Amiranashvili, T. Brox, and J. Peters. Crossq: Batch normalization in deep reinforcement learning for greater sample efficiency and simplicity. In *The Twelfth International Conference on Learning Representations*, 2024.

[10] C. Biever. Chatgpt broke the turing test-the race is on for new ways to assess ai. *Nature*, 619(7971):686–689, 2023.

[11] J. Bjorck, C. P. Gomes, and K. Q. Weinberger. Towards deeper deep reinforcement learning with spectral normalization, 2021. URL `https://proceedings.neurips.cc/paper/2021/hash/4588e674d3f0faf985047d4c3f13ed0d-Abstract.html`.

[12] J. Bjorck, C. P. Gomes, and K. Q. Weinberger. Is high variance unavoidable in rl? A case study in continuous control. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL `https://openreview.net/forum?id=9xhgmsNVHu`.

[13] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL `http://github.com/google/jax`.

[14] P. S. Castro, S. Moitra, C. Gelada, S. Kumar, and M. G. Bellemare. Dopamine: A research framework for deep reinforcement learning. *CoRR*, abs/1812.06110, 2018. URL `http://arxiv.org/abs/1812.06110`.

[15] E. Cetin and O. Çeliktutan. Learning pessimism for reinforcement learning. In B. Williams, Y. Chen, and J. Neville, editors, *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7-14, 2023*, pages 6971–6979. AAAI Press, 2023. doi: 10.1609/AAAI.V37I6.25852. URL `https://doi.org/10.1609/aaai.v37i6.25852`.

[16] D. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). In Y. Bengio and Y. LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL `http://arxiv.org/abs/1511.07289`.

[17] K. Cobbe, O. Klimov, C. Hesse, T. Kim, and J. Schulman. Quantifying generalization in reinforcement learning. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 1282–1289. PMLR, 2019. URL `http://proceedings.mlr.press/v97/cobbe19a.html`.

[18] Q. Delfosse, P. Schramowski, M. Mundt, A. Molina, and K. Kersting. Adaptive rational activations to boost deep reinforcement learning. In *The Twelfth International Con-*

ference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024. OpenReview.net, 2024. URL `https://openreview.net/forum?id=g90ysX1sVs`.

[19] S. Dohare, J. F. Hernandez-Garcia, P. Rahman, R. S. Sutton, and A. R. Mahmood. Maintaining plasticity in deep continual learning. *CoRR*, abs/2306.13812, 2023. doi: 10.48550/ARXIV.2306.13812. URL `https://doi.org/10.48550/arXiv.2306.13812`.

[20] S. Dohare, J. F. Hernandez-Garcia, P. Rahman, R. S. Sutton, and A. R. Mahmood. Maintaining plasticity in deep continual learning. *CoRR*, abs/2306.13812, 2023. doi: 10.48550/ARXIV.2306.13812. URL `https://doi.org/10.48550/arXiv.2306.13812`.

[21] P. D'Oro, M. Schwarzer, E. Nikishin, P. Bacon, M. G. Bellemare, and A. C. Courville. Sample-efficient reinforcement learning by breaking the replay ratio barrier. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL `https://openreview.net/pdf?id=OpC-9aBBVJe`.

[22] M. Elsayed and A. R. Mahmood. Addressing loss of plasticity and catastrophic forgetting in continual learning. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL `https://openreview.net/forum?id=sKPzAXoylB`.

[23] M. Elsayed, Q. Lan, C. Lyle, and A. R. Mahmood. Weight clipping for deep continual and reinforcement learning. *CoRR*, abs/2407.01704, 2024. doi: 10.48550/ARXIV.2407.01704. URL `https://doi.org/10.48550/arXiv.2407.01704`.

[24] J. Farebrother, J. Orbay, Q. Vuong, A. A. Taïga, Y. Chebotar, T. Xiao, A. Irpan, S. Levine, P. S. Castro, A. Faust, A. Kumar, and R. Agarwal. Stop regressing: Training value functions via classification for scalable deep RL. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. URL `https://openreview.net/forum?id=dVpFKfqF3R`.

[25] A. Fawzi, M. Balog, A. Huang, T. Hubert, B. Romera-Paredes, M. Barekatain, A. Novikov, F. J. R. Ruiz, J. Schrittwieser, G. Swirszcz, D. Silver, D. Hassabis, and P. Kohli. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nat.*, 610(7930):47–53, 2022. doi: 10.1038/S41586-022-05172-4. URL `https://doi.org/10.1038/s41586-022-05172-4`.

[26] P. Foret, A. Kleiner, H. Mobahi, and B. Neyshabur. Sharpness-aware minimization for efficiently improving generalization. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL `https://openreview.net/forum?id=6Tm1mposlrM`.

[27] C. D. Freeman, E. Frey, A. Raichuk, S. Girgin, I. Mordatch, and O. Bachem. Brax - a differentiable physics engine for large scale rigid body simulation, 2021. URL `http://github.com/google/brax`.

[28] S. Fujimoto, H. van Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In J. G. Dy and A. Krause, editors, *Proceedings of the 35th*

International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018, volume 80 of *Proceedings of Machine Learning Research*, pages 1582–1591. PMLR, 2018. URL `http://proceedings.mlr.press/v80/fujimoto18a.html`.

[29] M. Gallici, M. Fellows, B. Ellis, B. Pou, I. Masmitja, J. N. Foerster, and M. Martin. Simplifying deep temporal difference learning. *CoRR*, abs/2407.04811, 2024. doi: 10.48550/ARXIV.2407.04811. URL `https://doi.org/10.48550/arXiv.2407.04811`.

[30] L. B. Godfrey. An evaluation of parametric activation functions for deep learning. In *2019 IEEE International Conference on Systems, Man and Cybernetics, SMC 2019, Bari, Italy, October 6-9, 2019*, pages 3006–3011. IEEE, 2019. doi: 10.1109/SMC.2019.8913972. URL `https://doi.org/10.1109/SMC.2019.8913972`.

[31] F. Gogianu, T. Berariu, M. Rosca, C. Clopath, L. Busoniu, and R. Pascanu. Spectral normalisation for deep reinforcement learning: An optimisation perspective. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 3734–3744. PMLR, 2021. URL `http://proceedings.mlr.press/v139/gogianu21a.html`.

[32] Ç. Gülçehre, S. Srinivasan, J. Sygnowski, G. Ostrovski, M. Farajtabar, M. Hoffman, R. Pascanu, and A. Doucet. An empirical study of implicit regularization in deep offline RL. *Trans. Mach. Learn. Res.*, 2022, 2022. URL `https://openreview.net/forum?id=HFfJWx60IT`.

[33] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In J. G. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 1856–1865. PMLR, 2018. URL `http://proceedings.mlr.press/v80/haarnoja18b.html`.

[34] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 1026–1034. IEEE Computer Society, 2015. doi: 10.1109/ICCV.2015.123. URL `https://doi.org/10.1109/ICCV.2015.123`.

[35] Q. He, T. Zhou, M. Fang, and S. Maghsudi. Adaptive regularization of representation rank as an implicit constraint of bellman equation. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL `https://openreview.net/forum?id=apXtolxDaJ`.

[36] G. E. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015. URL `http://arxiv.org/abs/1503.02531`.

[37] M. Huh, H. Mobahi, R. Zhang, B. Cheung, P. Agrawal, and P. Isola. The low-rank simplicity bias in deep networks. *Trans. Mach. Learn. Res.*, 2023, 2023. URL `https://openreview.net/forum?id=bCiNWDmlY2`.

[38] M. Igl, G. Farquhar, J. Luketina, W. Boehmer, and S. Whiteson. Transient non-stationarity and generalisation in deep reinforcement learning. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL `https://openreview.net/forum?id=Qun8fv4qSby`.

[39] L. C. B. III. Residual algorithms: Reinforcement learning with function approximation. In A. Prieditis and S. Russell, editors, *Machine Learning, Proceedings of the Twelfth International Conference on Machine Learning, Tahoe City, California, USA, July 9-12, 1995*, pages 30–37. Morgan Kaufmann, 1995. doi: 10.1016/B978-1-55860-377-6.50013-X. URL `https://doi.org/10.1016/b978-1-55860-377-6.50013-x`.

[40] E. Imani and M. White. Improving regression performance with distributional losses. In J. G. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 2162–2171. PMLR, 2018. URL `http://proceedings.mlr.press/v80/imani18a.html`.

[41] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In F. R. Bach and D. M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 448–456. JMLR.org, 2015. URL `http://proceedings.mlr.press/v37/ioffe15.html`.

[42] T. Ji, Y. Liang, Y. Zeng, Y. Luo, G. Xu, J. Guo, R. Zheng, F. Huang, F. Sun, and H. Xu. ACE: off-policy actor-critic with causality-aware entropy regularization. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. URL `https://openreview.net/forum?id=1puvYh729M`.

[43] L. Kaiser, M. Babaeizadeh, P. Milos, B. Osinski, R. H. Campbell, K. Czechowski, D. Erhan, C. Finn, P. Kozakowski, S. Levine, A. Mohiuddin, R. Sepassi, G. Tucker, and H. Michalewski. Model based reinforcement learning for atari. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL `https://openreview.net/forum?id=S1xCPJHtDB`.

[44] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL `http://arxiv.org/abs/1412.6980`.

[45] A. Kumar, A. Zhou, G. Tucker, and S. Levine. Conservative q-learning for offline reinforcement learning. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and

H. Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL `https://proceedings.neurips.cc/paper/2020/hash/0d2b2061826a5df3221116a5085a6052-Abstract.html`.

[46] A. Kumar, R. Agarwal, D. Ghosh, and S. Levine. Implicit under-parameterization inhibits data-efficient deep reinforcement learning. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL `https://openreview.net/forum?id=O9bnihsFfXU`.

[47] A. Kumar, R. Agarwal, T. Ma, A. C. Courville, G. Tucker, and S. Levine. DR3: value-based deep reinforcement learning requires explicit regularization. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL `https://openreview.net/forum?id=POvMvLi91f`.

[48] A. Kumar, R. Agarwal, X. Geng, G. Tucker, and S. Levine. Offline q-learning on diverse multi-task data both scales and generalizes. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL `https://openreview.net/pdf?id=4-k7kUavAj`.

[49] R. T. Lange. gymnax: A JAX-based reinforcement learning environment library, 2022. URL `http://github.com/RobertTLange/gymnax`.

[50] M. Laskin, K. Lee, A. Stooke, L. Pinto, P. Abbeel, and A. Srinivas. Reinforcement learning with augmented data. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL `https://proceedings.neurips.cc/paper/2020/hash/e615c82aba461681ade82da2da38004a-Abstract.html`.

[51] H. Lee, H. Cho, H. Kim, D. Gwak, J. Kim, J. Choo, S. Yun, and C. Yun. PLASTIC: improving input and label plasticity for sample efficient reinforcement learning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL `http://papers.nips.cc/paper_files/paper/2023/hash/c464fc4516aca4e68f2a14e67c6f0402-Abstract-Conference.html`.

[52] H. Lee, H. Cho, H. Kim, D. Kim, D. Min, J. Choo, and C. Lyle. Slow and steady wins the race: Maintaining plasticity with hare and tortoise networks. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. URL `https://openreview.net/forum?id=VF177x7Syw`.

[53] A. Lewandowski, H. Tanaka, D. Schuurmans, and M. C. Machado. Curvature explains loss of plasticity. *CoRR*, abs/2312.00246, 2023. doi: 10.48550/ARXIV.2312.00246. URL `https://doi.org/10.48550/arXiv.2312.00246`.

[54] C. Lyle, M. Rowland, G. Ostrovski, and W. Dabney. On the effect of auxiliary tasks on representation dynamics. In A. Banerjee and K. Fukumizu, editors, *The 24th International Conference on Artificial Intelligence and Statistics, AISTATS 2021, April 13-15, 2021, Virtual Event*, volume 130 of *Proceedings of Machine Learning Research*, pages 1–9. PMLR, 2021. URL `http://proceedings.mlr.press/v130/lyle21a.html`.

[55] C. Lyle, M. Rowland, and W. Dabney. Understanding and preventing capacity loss in reinforcement learning. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL `https://openreview.net/forum?id=ZkC8wKoLbQ7`.

[56] C. Lyle, Z. Zheng, E. Nikishin, B. Á. Pires, R. Pascanu, and W. Dabney. Understanding plasticity in neural networks. In A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, editors, *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 23190–23211. PMLR, 2023. URL `https://proceedings.mlr.press/v202/lyle23b.html`.

[57] C. Lyle, Z. Zheng, K. Khetarpal, J. Martens, H. van Hasselt, R. Pascanu, and W. Dabney. Normalization and effective learning rates in reinforcement learning. *CoRR*, abs/2407.01800, 2024. doi: 10.48550/ARXIV.2407.01800. URL `https://doi.org/10.48550/arXiv.2407.01800`.

[58] C. Lyle, Z. Zheng, K. Khetarpal, H. van Hasselt, R. Pascanu, J. Martens, and W. Dabney. Disentangling the causes of plasticity loss in neural networks. *CoRR*, abs/2402.18762, 2024. doi: 10.48550/ARXIV.2402.18762. URL `https://doi.org/10.48550/arXiv.2402.18762`.

[59] G. Ma, L. Li, S. Zhang, Z. Liu, Z. Wang, Y. Chen, L. Shen, X. Wang, and D. Tao. Revisiting plasticity in visual reinforcement learning: Data, modules and training stages. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL `https://openreview.net/forum?id=0aR1s9YxoL`.

[60] M. McCloskey and N. J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier, 1989.

[61] E. Meyer, A. White, and M. C. Machado. Harnessing discrete representations for continual reinforcement learning. *CoRR*, abs/2312.01203, 2023. doi: 10.48550/ARXIV.2312.01203. URL `https://doi.org/10.48550/arXiv.2312.01203`.

[62] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida. Spectral normalization for generative adversarial networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL `https://openreview.net/forum?id=B1QRgziT-`.

[63] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nat.*, 518(7540):529–533, 2015. doi: 10.1038/NATURE14236. URL `https://doi.org/10.1038/nature14236`.

[64] K. P. Murphy. *Probabilistic Machine Learning: An Introduction*. Adaptive Computation and Machine Learning Series. The MIT Press, Cambridge, Massachusetts, 2022. ISBN 978-0-262-04682-4.

[65] K. P. Murphy. *Probabilistic Machine Learning: Advanced Topics*. Adaptive Computation and Machine Learning Series. The MIT Press, Cambridge, Massachusetts, 2023. ISBN 978-0-262-04843-9. URL `http://probml.github.io/book2`.

[66] M. Nauman and M. Cygan. On the theory of risk-aware agents: Bridging actor-critic and economics. In *ICML 2024 Workshop: Aligning Reinforcement Learning Experimentalists and Theorists*, 2023.

[67] M. Nauman, M. Bortkiewicz, P. Milos, T. Trzcinski, M. Ostaszewski, and M. Cygan. Overestimation, overfitting, and plasticity in actor-critic: the bitter lesson of reinforcement learning. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. URL `https://openreview.net/forum?id=5vZzmCeTYu`.

[68] M. Nauman, M. Ostaszewski, K. Jankowski, P. Milos, and M. Cygan. Bigger, regularized, optimistic: scaling for compute and sample-efficient continuous control. *CoRR*, abs/2405.16158, 2024. doi: 10.48550/ARXIV.2405.16158. URL `https://doi.org/10.48550/arXiv.2405.16158`.

[69] E. Nikishin, M. Schwarzer, P. D'Oro, P. Bacon, and A. C. Courville. The primacy bias in deep reinforcement learning. In K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvári, G. Niu, and S. Sabato, editors, *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 16828–16847. PMLR, 2022. URL `https://proceedings.mlr.press/v162/nikishin22a.html`.

[70] E. Nikishin, J. Oh, G. Ostrovski, C. Lyle, R. Pascanu, W. Dabney, and A. Barreto. Deep reinforcement learning with plasticity injection. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL `http://papers.nips.cc/paper_files/paper/2023/hash/75101364dc3aa7772d27528ea504472b-Abstract-Conference.html`.

[71] J. S. Obando-Ceron, M. G. Bellemare, and P. S. Castro. Small batch deep reinforcement learning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA,*

December 10 - 16, 2023, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/528388f1ad3a481249a97cbb698d2fe6-Abstract-Conference.html.

[72] J. S. Obando-Ceron, A. C. Courville, and P. S. Castro. In value-based deep reinforcement learning, a pruned network is a good network. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=seo9V9QRZp.

[73] OpenAI, I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, J. Schneider, N. Tezak, J. Tworek, P. Welinder, L. Weng, Q. Yuan, W. Zaremba, and L. Zhang. Solving rubik's cube with a robot hand. *CoRR*, abs/1910.07113, 2019. URL http://arxiv.org/abs/1910.07113.

[74] I. Osband, C. Blundell, A. Pritzel, and B. V. Roy. Deep exploration via bootstrapped DQN. In D. D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 4026–4034, 2016. URL https://proceedings.neurips.cc/paper/2016/hash/8d8818c8e140c64c743113f563cf750f-Abstract.html.

[75] G. Ostrovski, M. G. Bellemare, A. van den Oord, and R. Munos. Count-based exploration with neural density models. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 2721–2730. PMLR, 2017. URL http://proceedings.mlr.press/v70/ostrovski17a.html.

[76] G. Ostrovski, P. S. Castro, and W. Dabney. The difficulty of passive learning in deep reinforcement learning. In M. Ranzato, A. Beygelzimer, Y. N. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 23283–23295, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/c3e0c62ee91db8dc7382bde7419bb573-Abstract.html.

[77] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Z. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 8024–8035, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html.

[78] T. Schaul, A. Barreto, J. Quan, and G. Ostrovski. The phenomenon of policy churn. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances*

in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022, 2022. URL `http://papers.nips.cc/paper_files/paper/2022/hash/114292cf3f930ba157ed33f66997fee2-Abstract-Conference.html`.

[79] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. P. Lillicrap, and D. Silver. Mastering atari, go, chess and shogi by planning with a learned model. *Nat.*, 588(7839):604–609, 2020. doi: 10.1038/S41586-020-03051-4. URL `https://doi.org/10.1038/s41586-020-03051-4`.

[80] M. Schwarzer, J. S. Obando-Ceron, A. C. Courville, M. G. Bellemare, R. Agarwal, and P. S. Castro. Bigger, better, faster: Human-level atari with human-level efficiency. In A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, editors, *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 30365–30380. PMLR, 2023. URL `https://proceedings.mlr.press/v202/schwarzer23a.html`.

[81] W. Shang, K. Sohn, D. Almeida, and H. Lee. Understanding and improving convolutional neural networks via concatenated rectified linear units. In M. Balcan and K. Q. Weinberger, editors, *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 2217–2225. JMLR.org, 2016. URL `http://proceedings.mlr.press/v48/shang16.html`.

[82] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. P. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of go with deep neural networks and tree search. *Nat.*, 529(7587):484–489, 2016. doi: 10.1038/NATURE16961. URL `https://doi.org/10.1038/nature16961`.

[83] G. Sokar, R. Agarwal, P. S. Castro, and U. Evci. The dormant neuron phenomenon in deep reinforcement learning. In A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, editors, *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 32145–32168. PMLR, 2023. URL `https://proceedings.mlr.press/v202/sokar23a.html`.

[84] R. S. Sutton and A. G. Barto. *Reinforcement learning - an introduction.* Adaptive computation and machine learning. MIT Press, 1998. ISBN 978-0-262-19398-6. URL `https://www.worldcat.org/oclc/37293240`.

[85] Y. Tassa, Y. Doron, A. Muldal, T. Erez, Y. Li, D. de Las Casas, D. Budden, A. Abdolmaleki, J. Merel, A. Lefrancq, T. P. Lillicrap, and M. A. Riedmiller. Deepmind control suite. *CoRR*, abs/1801.00690, 2018. URL `http://arxiv.org/abs/1801.00690`.

[86] A. van den Oord, O. Vinyals, and K. Kavukcuoglu. Neural discrete representation learning. In I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 6306–6315, 2017. URL `https://proceedings.neurips.cc/paper/2017/hash/7a98af17e63a0ac09ce2e96d03992fbc-Abstract.html`.

[87] H. van Hasselt. Double q-learning. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23: 24th Annual Conference on Neural Information Processing Systems 2010. Proceedings of a meeting held 6-9 December 2010, Vancouver, British Columbia, Canada*, pages 2613–2621. Curran Associates, Inc., 2010. URL `https://proceedings.neurips.cc/paper/2010/hash/091d584fced301b442654dd8c23b3fc9-Abstract.html`.

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

[89] G. Xu, R. Zheng, Y. Liang, X. Wang, Z. Yuan, T. Ji, Y. Luo, X. Liu, J. Yuan, P. Hua, S. Li, Y. Ze, H. D. III, F. Huang, and H. Xu. Drm: Mastering visual reinforcement learning through dormant ratio minimization. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL `https://openreview.net/forum?id=MSe8YFbhUE`.

[90] J. Xu, X. Sun, Z. Zhang, G. Zhao, and J. Lin. Understanding and improving layer normalization. In H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 4383–4393, 2019. URL `https://proceedings.neurips.cc/paper/2019/hash/2f4fe03d77724a7217006e5d16728874-Abstract.html`.

[91] D. Yarats, I. Kostrikov, and R. Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL `https://openreview.net/forum?id=GY6-6sTvGaf`.

[92] D. Yarats, R. Fergus, A. Lazaric, and L. Pinto. Mastering visual continuous control: Improved data-augmented reinforcement learning. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL `https://openreview.net/forum?id=_SJ-_yyes8`.

[93] T. Yu, D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, and S. Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In L. P. Kaelbling, D. Kragic, and K. Sugiura, editors, *3rd Annual Conference on Robot Learning, CoRL 2019, Osaka, Japan, October 30 - November 1, 2019, Proceedings*,

volume 100 of *Proceedings of Machine Learning Research*, pages 1094–1100. PMLR, 2019. URL `http://proceedings.mlr.press/v100/yu20a.html`.