Spectral Collapse Drives Loss of Plasticity in Deep Continual Learning

Arjun Prakash [*1] **Naicheng He** [*1] **Kaicheng Guo** [*1] **Saket Tiwari** [1] **Tyrone Serapio** [1] **Ruo Yu Tao** [1] **Amy Greenwald** [1] **George Konidaris** [1]

Abstract

We investigate why deep neural networks suffer from loss of plasticity in continual learning, and thus fail to learn new tasks without reinitializing parameters. We show that this failure is preceded by Hessian spectral collapse at new-task initialization, where meaningful curvature directions vanish and gradient descent becomes ineffective. Analyzing a linearized ReLU network, we derive explicit ϵ-rank conditions for successful training and prove that the loss-weighted Gram matrix is spectrally equivalent to the Generalized Gauss-Newton approximation, thereby relating NTK dynamics to Hessian curvature. Targeting spectral collapse directly, we then discuss the Kronecker factored approximation of the Hessian, which motivates two regularization enhancements: maintaining high effective feature rank and applying L2 penalties. Experiments on continual supervised and reinforcement learning tasks confirm that combining these two regularizers effectively preserves plasticity.

1. Introduction

Human intelligence is marked not by mastering a single task, but by the ability to continually adapt to new challenges, through the accumulation and refinement of skills and knowledge. Likewise, a central aspiration of artificial intelligence is to create systems that can learn throughout their lifetimes. Endowing artificial systems with this ability would ensure that they remain useful, relevant, and robust in an open world (Javed & Sutton, 2024).

Research on deep learning and deep reinforcement learning has demonstrated remarkable progress, reaching human and even superhuman performance in image recognition (Siméoni et al., 2026), game playing (Silver et al., 2017), and reasoning (Bakhtin et al., 2022). But these advances have largely been realized under stationary data distributions. *Continual learning* concerns a system's ability to perform well through a sequence of evolving tasks. Within this framework, two complementary objectives are often distinguished: (i) *maintaining plasticity*, meaning the capacity to acquire new knowledge (Dohare et al., 2024), and (ii) *preventing catastrophic forgetting*, where the learner forgets how to solve tasks it previously mastered (Kirkpatrick et al., 2017; Parisi et al., 2018; Baker et al., 2023).

Our work explores plasticity loss from an optimization perspective. Despite significant empirical progress in the area (Lyle et al., 2023; 2024; Abbas et al., 2023), there remains no unifying consensus on what drives loss of plasticity (Klein et al., 2024). Several disparate explanations have been investigated, including inactive neurons which output constants regardless of the inputs (Bjorck et al., 2022; Sokar et al., 2023), the reduction in feature expressiveness associated with large parameter norms (Lyle et al., 2024), and overfitting (Igl et al., 2021). Recent work has begun to connect plasticity loss to second-order properties, such as the stable rank of the Hessian or layer-wise spectral conditioning (Lewandowski et al., 2024b;a). Building on these insights, we argue that these seemingly disparate observations are in fact different facets of the same phenomenon, which we call *spectral collapse*: a degeneration of the Hessian eigenspectrum, indicating loss of curvature in most directions. An example comparison of Hessian eigenspectrum with and without the spectral collapse is shown in Figure 1: in the BP run, the task-50 spectrum has a broad positive bulk, indicating many usable curvature directions; but by task 300, most spectral mass has collapsed, leaving only a few isolated non-zero eigenvalue directions. In contrast, L2-ER preserves a broader spectral bulk across tasks.

Our main contributions can be summarized as follows:

1. We define Hessian spectral collapse, and identify it as a central mechanism underlying plasticity loss. To support this claim, we provide extensive empirical evidence across diverse continual learning benchmarks, demonstrating that existing plasticity-preserving inter-

[*]Equal contribution , order drawn from random [1]Department of Computer Science, Brown University, Providence, RI, USA. Correspondence to: Arjun Prakash <arjun_prakash.edu>, Naicheng He <naicheng_he@brown.edu>.

Proceedings of the 43^{rd} International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026.

ventions designed to target seemingly different phenomena all prevent spectral collapse.

2. We analyze a linearized ReLU network and derive explicit ϵ-rank conditions for successful training. We establish that the loss-weighted Gram matrix is spectrally equivalent to the Generalized Gauss-Newton (GGN) matrix, thereby relating function-space analysis to parameter-space curvature.

3. Leveraging this theory, we propose $L2$-ER regularization, intended to target Hessian eigenspectrum collapse directly, and show that it matches or exceeds existing methods on various continual supervised learning and reinforcement learning benchmarks.[1]

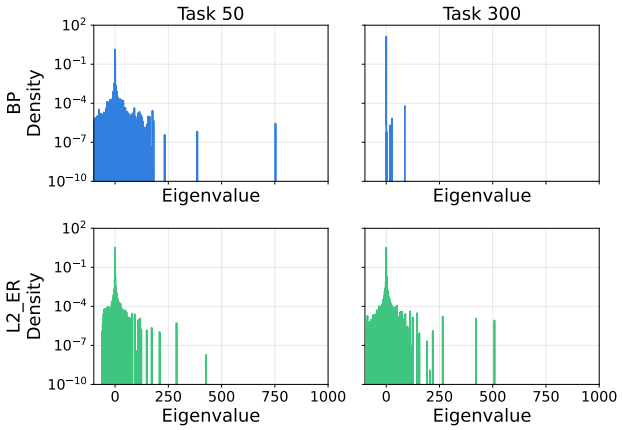


Figure 1. The Hessian eigenspectrum on continual Imagenet at task initialization. The top row shows the eigenspectrum on a network using standard backpropagation (BP). The bottom row shows the eigenspectrum using $L2$-ER regularization (ours).

2. Related Work

Causes of Plasticity Loss Several pathologies have been proposed to explain loss of plasticity in continual learning, both supervised and reinforcement (Lyle et al., 2023; Klein et al., 2024). For example, *dormant neurons*, which output a positive value on all inputs, reduce a network's expressive power, while *dead neurons*, which output zero on all inputs, prevent gradient flow (Montufar et al., 2014; Sokar et al., 2023). Reductions in *effective feature rank*, which measures the intrinsic dimensionality of feature representations and directly relates to the persistence of dead neurons, has also been linked to loss of plasticity (Roy & Vetterli, 2007; Dohare et al., 2024), as has *parameter norm growth*, which leads to ill-conditioned Hessians that cause numerical instability (Obando-Ceron et al., 2024; Lyle et al., 2024). Lyle et al. (2024) propose a "swiss cheese" model that attempts to preserve plasticity by targeting all these pathologies simultaneously.

Mitigating Plasticity Loss Existing methods for mitigating loss of plasticity target one or more of the aforementioned pathologies (Klein et al., 2024). Broadly speaking, these approaches fall into two categories: (i) continuous interventions that apply at every optimization step, and (ii) intermittent interventions that periodically reset or perturb parameters (Juliani & Ash, 2024). For example, Lyle et al. (2024) shows that combining $L2$ regularization to control parameter norm growth with layer normalization to stabilize pre-activation distributions is highly effective. Other approaches regularize the singular values of the layer-wise Jacobians (Lewandowski et al., 2024a) to maintain parameter distributions close to a uniform random initialization (He et al., 2015b; Hinton & Salakhutdinov, 2006). Architectural changes such as PELU (Godfrey, 2019), CRELU (Shang et al., 2016; Abbas et al., 2023), and adaptive rational activations (Delfosse et al., 2024) avoid the saturation of neurons. In contrast, periodic interventions, such as Shrink & Perturb (Ash & Adams, 2020), continual backpropagation (CBP) (Dohare et al., 2024), and ReDo (Sokar et al., 2023), selectively reset dormant neurons. Empirically, hybrid strategies—typically involving $L2$ regularization combined with either architectural or periodic interventions—exhibit the strongest performance (Lyle et al., 2024; Juliani & Ash, 2024), implying that plasticity loss can emerge from multiple mechanisms.

Approximate Second-Order Methods Our work contributes to a recent line of inquiry that connects loss of plasticity to second-order information. Prior work by Lewandowski et al. (2024b) provides strong empirical evidence that loss of plasticity correlates with a decline in the stable rank of a partial Hessian, proposing a Wasserstein penalty to maintain characteristics of the initial parameter distribution. In a complementary approach, Lewandowski et al. (2024a) focus on the spectral properties of the layer-wise weight matrices, rather than the Hessian, using spectral regularization to keep the largest singular value near one, thereby maintaining favorable conditioning and gradient diversity. Similarly, Parseval regularization (Chung et al., 2024) improves weight-matrix conditioning by enforcing orthogonality in the weight matrices to preserve gradient norms.

Neural Tangent Kernel in Continual Learning The neural tangent kernel (NTK) (Jacot et al., 2018) provides a tractable lens on continual learning by approximating training as kernel regression around initialization. Prior work analyzes continual learning in the NTK regime through transfer and forgetting (Doan et al., 2021) or uses empirical-NTK degeneracies as a diagnostic (Lyle et al., 2024). Our

[1]Code is available at here.

analysis isolates a concrete failure mode: learnability is controlled by how much of the task error residual lies in the *fast* eigenspace, meaning the span of those Gram-matrix eigenvectors whose eigenvalues are large enough to contract substantially within the budgeted number of gradient steps. For permuted MNIST, we prove the NTK is invariant to input permutations, which implies that plasticity loss arises from progressive shrinkage of the fast subspace rather than task-induced kernel drift. While Tang et al. (2025) connect rank collapse to output variability, we show that once the fast eigenspace collapses, learning fails even with stable outputs—motivating L2-ER, which targets the underlying geometry directly. Our work is also related to Singular Learning Theory (SLT) (Watanabe, 2009), which is concerned with understanding singularities in the Fisher information matrix of neural networks. The empirical NTK and Fisher information matrices are equivalent up to reversal and scaling (Karakida et al., 2021) and share non-zero eigenvalues. SLT explains why models move toward singular regions, while our work studies the continual-learning consequence of this phenomenon.

3. Preliminaries

Notation We use τ to index tasks, k for optimization iterates, l for layers, n and m for samples, o for output coordinates, and q for eigenvalues. A layer-specific matrix at task τ and iterate k is written $\mathbf{M}_\tau^{[l],(k)}$, with its (a,b)-entry written as $[\mathbf{M}_\tau^{[l],(k)}]_{ab}$. We write $\boldsymbol{x}_{\tau,n}$ and $\boldsymbol{y}_{\tau,n}$ for the nth data sample in dataset $\boldsymbol{X}$. We also write $[a]$ to denote the set of integers $\{1,\dots,a\}$, for some $a \in \mathbb{Z}$.

Deep Neural Networks A multi-layered-perceptron (MLP), also called a feedforward neural network, is constructed by composing L functions, i.e., $F^L \circ \cdots \circ F^1$, one per layer. Given an input (resp. output) dimension of M^{l-1} (resp. M^l), the function at layer l, for $l \in [L]$, is a map $F^l : \mathbb{R}^{M^{l-1}} \to \mathbb{R}^{M^l}$. This function F^l is the composition of an element-wise non-linear activation function $\sigma^l : \mathbb{R} \to \mathbb{R}$ and a weight matrix $\mathbf{W}^{[l]} \in \mathbb{R}^{M^l \times M^{l-1}}$, so that $F^l = \sigma^l \circ \mathbf{W}^{[l]}$. Our activation function of choice is ReLU , defined as $\max\{0, x\}$. The dimension at layer l indicates the number of neurons at that layer. The total number of hidden neurons is $M \doteq \sum_{l=1}^{L-1} M^l$.

We collect all parameters into a single vector using the vec operator, so that the MLP parameterization is represented by $\boldsymbol{\theta} \doteq \mathrm{vec}\,(\mathbf{W}^{[1]},\dots,\mathbf{W}^{[L]}) \in \mathbb{R}^P$, where $P = \sum_{l=1}^{L} M^l M^{l-1}$ is the total parameter count. Denoting the input (resp. output) dimension of the MLP by $I \doteq M^0$ (resp. $O \doteq M^L$), the MLP map is $F_{\boldsymbol{\theta}} : \mathbb{R}^I \to \mathbb{R}^O$.

A task τ is described by a loss function ℓ_τ and a data distribution $\boldsymbol{d}_\tau$. Given such a task τ, a learning algorithm seeks parameter values that minimize the expected loss $\mathcal{L}_\tau(\boldsymbol{\theta}) \doteq \mathbb{E}_{\boldsymbol{d}_\tau}[\ell_\tau(\boldsymbol{z}_\tau, \boldsymbol{y}_\tau)]$ on the task, where $\boldsymbol{z}_\tau = F_{\boldsymbol{\theta}_\tau}(\boldsymbol{x}_\tau)$.

The gradient of the network is the vector of first derivatives of the loss function w.r.t. the network parameters, i.e., $\nabla_{\boldsymbol{\theta}} \ell_\tau \in \mathbb{R}^P$, while the main object of our investigation, the *Hessian*, is the matrix of second derivatives, i.e., $\mathbf{H} \doteq \nabla^2_{\boldsymbol{\theta}} \ell_\tau \in \mathbb{R}^{P\times P}$. Together, the Jacobian and the Hessian capture up to second-order changes in the loss, i.e., when the parameters change by a vector $\delta \in \mathbb{R}^P$, the loss can be approximated as $\mathcal{L}_\tau(\boldsymbol{\theta}+\delta) \approx \mathcal{L}_\tau(\boldsymbol{\theta}) + \nabla_{\boldsymbol{\theta}}\mathcal{L}_\tau^\top \delta + \frac{1}{2}\delta^\top \nabla^2_{\boldsymbol{\theta}} \mathcal{L}_\tau \delta$.

The Hessian The Hessian offers valuable insight into the loss landscape of a neural network (Pouplin et al., 2023). For example, one line of research is concerned with large connected regions in weight space where the error remains approximately constant (Hochreiter & Schmidhuber, 1997); such regions may be less prone to overfitting and result in smaller generalization gaps (Cha et al., 2021). Since the Hessian $\mathbf{H}$ is symmetric, there exists an orthonormal basis of eigenvectors $\{\mathbf{v}_q\}$ and real eigenvalues $\{\lambda_q\}$ satisfying $\mathbf{H}\mathbf{v}_q = \lambda_q \mathbf{v}_q$, for all $q \in \{1,\dots,P\}$. Each eigenvalue λ_q measures the second-order curvature of the loss in the direction of eigenvector $\mathbf{v}_q$. A large eigenvalue indicates a high local curvature in the loss landscape in the direction of the corresponding eigenvector, while an eigenvalue near zero indicates low local curvature in that direction. Given the importance of the eigenvalues of the Hessian, the distribution of all eigenvalues, known as the *eigenspectrum*, provides more information than any single scalar summary, such as the largest eigenvalue or the trace. It is possible decompose the Hessian further using the chain rule, to obtain the Gauss-Newton decomposition (Zhao et al., 2024):

$$\nabla^2_{\boldsymbol{\theta}} \ell = \mathbf{H} = \underbrace{\frac{1}{N}\sum_{n=1}^{N} \mathbf{J}_n^\top \left[\nabla^2_{\boldsymbol{z}} \ell(\boldsymbol{z}_n, \boldsymbol{y}_n)\right] \mathbf{J}_n}_{\mathbf{H}_{\mathrm{GGN}}} + \underbrace{\frac{1}{N}\sum_{n=1}^{N}\sum_{o=1}^{O} \left[\nabla_{\boldsymbol{z}} \ell(\boldsymbol{z}_n, \boldsymbol{y}_n)\right]_o \nabla^2_{\boldsymbol{\theta}} [\boldsymbol{z}_n]_o}_{\mathbf{R}}, \quad (1)$$

where at the nth sample, $\mathbf{J}_n \in \mathbb{R}^{O\times P}$ is the Jacobian of the model w.r.t. its parameters (i.e., $\nabla_{\boldsymbol{\theta}} F_{\boldsymbol{\theta}}(\boldsymbol{x}_n)$), $\nabla^2_{\boldsymbol{z}} \ell \in \mathbb{R}^{O\times O}$ is the Hessian of the loss w.r.t. the model outputs (i.e., the pre-softmax logits in classification or the scalar output in regression), $\left[\nabla_{\boldsymbol{z}} \ell\right]_o \in \mathbb{R}^{O\times 1}$ is the gradient in the o-th output dimension of the loss w.r.t. to the model's output, and $\nabla^2_{\boldsymbol{\theta}}[\boldsymbol{z}_n]_o$ is the Hessian of the o-th output element w.r.t. the model parameters. Simply dropping the residual term $\mathbf{R}$, yields the Generalized Gauss-Newton (GGN) matrix $\mathbf{H}_{\mathrm{GGN}}$, a faithful proxy for the Hessian (Dangel et al., 2025).

Neural Tangent Kernel (NTK) Given the the complexity of deep neural networks with non-linearities, the neural tangent kernel (NTK) has been proposed as an abstraction to characterize the training dynamics of (stochastic) gradient descent in the infinite-width limit (i.e., $M \to \infty$) (Jacot et al., 2018). The main principle behind NTK theory is to approximate a function by linearizing a neural network around its initialization. The seminal work by Jacot et al. shows that if F is an appropriately scaled neural network, with $\boldsymbol{\theta}$ initialized i.i.d. from standard (or isotropic) Gaussian's (i.e. $\boldsymbol{\theta} \sim \mathcal{N}(0, \boldsymbol{I})$), then $F_{\boldsymbol{\theta}}(\boldsymbol{X}) \approx F_{\boldsymbol{\theta}^{(0)}}(\boldsymbol{X}) + \mathbf{J}(\boldsymbol{\theta} - \boldsymbol{\theta}^{(0)})$. The linearized model $F_{\boldsymbol{\theta}}$ can be interpreted as a kernel method, where the Jacobian is the feature map. The kernel induced by the feature map is the NTK and is dependent on the random initialization $\boldsymbol{\theta}^{(0)}$. The neural tangent kernel $\kappa_{\boldsymbol{\theta}} : \mathcal{X} \times \mathcal{X} \to \mathbb{R}^{O \times O}$ is given by $\kappa_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{x}') \doteq \mathbb{E}_{\boldsymbol{\theta} \sim \mathcal{N}(0, \boldsymbol{I})}[\langle \frac{\partial F_{\boldsymbol{\theta}}(\boldsymbol{x})}{\partial \boldsymbol{\theta}}, \frac{\partial F_{\boldsymbol{\theta}}(\boldsymbol{x}')}{\partial \boldsymbol{\theta}} \rangle |_{\boldsymbol{\theta} = \boldsymbol{\theta}^{(0)}}]$. Evaluating this kernel function on a finite dataset of since N (i.e., for all $\boldsymbol{x}_n, \boldsymbol{x}_m \in \boldsymbol{X}$) gives the empirical NTK matrix $\kappa_{\boldsymbol{\theta}}(\boldsymbol{x}_n, \boldsymbol{x}_m) = \mathbf{J}\mathbf{J}^\top$. This matrix, which captures all pairwise inner products among vectors in the set $\boldsymbol{X}$, is called the *Gram matrix* and is denoted by $\mathbf{G} \in \mathbb{R}^{NO \times NO}$. The celebrated result from NTK analysis is that as $M \to \infty$, the NTK $\kappa_{\boldsymbol{\theta}}$ converges in probability to an explicit deterministic limit $\kappa_{\boldsymbol{\theta}}^{\infty} = \mathbb{E}_{\boldsymbol{\theta} \sim \mathcal{N}(0, \boldsymbol{I})}[\mathbf{G}]$ (Jacot et al., 2018, Theorem 1). This deterministic limit only depends on the variance parameter of the Gaussian initialization and the choice of non-linearity. The NTK limit holds for ReLU activations (Arora et al., 2019) and LeCun initialization (Jacot et al., 2018).

4. Continual Learning and Plasticity Loss

Assume an agent facing a sequence of T tasks whose decision-making capabilities are dictated by a neural network with parameters $\boldsymbol{\theta} \in \mathbb{R}^P$. Each task τ is characterized by an evaluation metric $f_\tau : \mathbb{R}^P \to \mathbb{R}$, such as accuracy for supervised learning or expected return for reinforcement learning. In continual learning, we seek a sequence of parameters $\{\boldsymbol{\theta}_\tau^*\}_\tau$ s.t. $\boldsymbol{\theta}_\tau^* \in \arg\max_{\boldsymbol{\theta}_\tau} f_\tau(\boldsymbol{\theta}_\tau)$, for all $\tau \in [T]$.

Since evaluation metrics may be non-differentiable, we further assume a (possibly regularized) surrogate expected loss function $\mathcal{L}_\tau : \mathbb{R}^P \to \mathbb{R}$, for each task τ, which is intended to maximize the evaluation metric, but is generally better behaved. Additionally, we assume the learner has a budget of K learning steps per task. A solution to the continual learning problem is generated by an algorithm $\mathcal{A} : \mathbb{R}^P \to \mathbb{R}^P$, which outputs parameters for the current task τ that optimize τ's surrogate loss function $\mathcal{L}_\tau$, given the parameters learned during the previous task, i.e., the parameters for task $\tau + 1$ are initialized as the learned parameters of task τ: i.e., $\boldsymbol{\theta}_{\tau+1}^{(0)} \doteq \boldsymbol{\theta}_\tau^{(K)}$.

Definition 4.1 (Successful Training)**.** In a continual learning problem, given finite update budget K and a small error tolerance υ, we write $\mathbb{I}(\ell_\tau(\boldsymbol{\theta}_\tau^{(K)}) \leq \upsilon)$ to indicate *successful training* on task τ. We say a neural network is *plastic* if it successfully trains on all tasks $\tau \in [T]$.

In the remainder of this section, we analyze how spectral information of the NTK Gram matrix governs successful training. This analysis motivates the design of our regularizer, which preserves neural network plasticity by controlling the eigenspectrum of the Hessian.

4.1. Linearized ReLU networks

We now instantiate the NTK framework for a linearized ReLU network. Assume input dimension I, width M, and output dimension O. Denote the first-layer weights by $\mathbf{W}^{[1]} = [\mathbf{w}_1^{[1]}, \ldots, \mathbf{w}_M^{[1]}] \in \mathbb{R}^{I \times M}$ where $\mathbf{w}_h^{[1]} = [\mathbf{W}_h^{[1]}]_{:,h} \in \mathbb{R}^I$ is the input weight vector of hidden unit h. We fix the output weights and the ReLU gates, so the only trainable parameters in this linearized model are the first-layer weights. Define the fixed output-layer weights by $\mathbf{W}^{[2]} = [\mathbf{w}_1^{[2]}, \ldots, \mathbf{w}_M^{[2]}]^\top \in \mathbb{R}^{M \times O}$ (i.e., $\mathbf{w}_h^{[2]} = [\mathbf{W}^{[2]}]_{h,:}^\top \in \mathbb{R}^O$) Then, with $\sigma(x) = \max\{0, x\}$, the output of a linearized ReLU network on input $\boldsymbol{x} \in \mathbb{R}^I$ is given by:

$$\hat{F}_{\boldsymbol{\theta}}(\boldsymbol{x}) \doteq \frac{1}{\sqrt{M}} \sum_{h=1}^{M} \mathbf{w}_h^{[2]} \, \sigma((\mathbf{w}_h^{[1]})^\top \boldsymbol{x}) \in \mathbb{R}^O.$$

We fix $\mathbf{W}^{[1](0)}$ to be a frozen reference copy of the initialized first-layer matrix $\mathbf{W}^{[1]}$. With this fixed initialization, we define the fixed ReLU gate function $g_h(\boldsymbol{x}) \doteq \mathbf{1}\{(\mathbf{w}_h^{[1](0)})^\top \boldsymbol{x} > 0\}, h \in 1, \ldots M$, and gate matrix: $\mathbf{D}_g(\boldsymbol{x}) \doteq \mathrm{diag}(g(\boldsymbol{x})) \in \mathbb{R}^{M \times M}$. By construction, the only trainable parameters are $\mathbf{W}^{[1]}$, initialized at $\mathbf{W}^{[1]} = \mathbf{W}^{[1](0)}$, and the model prediction is :

$$\hat{F}_{\boldsymbol{\theta}}(\boldsymbol{x}) = \frac{1}{\sqrt{M}} F_{\mathbf{W}^{[1]}}(\boldsymbol{x})^\top \mathbf{W}^{[2]} \tag{2}$$

$$= \frac{1}{\sqrt{M}} (\mathbf{W}^{[2]})^\top \mathbf{D}_g(\boldsymbol{x}) (\mathbf{W}^{[1]})^\top \boldsymbol{x} \in \mathbb{R}^O. \tag{3}$$

Continual Leaning with NTKs Given task τ with dataset $(\boldsymbol{X}_\tau, \boldsymbol{Y}_\tau) = \{\boldsymbol{x}_{\tau,n}, \boldsymbol{y}_{\tau,n}\}_{n=0}^{N_\tau}$, the model prediction is $\hat{F}_{\boldsymbol{\theta}}(\boldsymbol{X}_\tau) \in \mathbb{R}^{N_\tau \times O}$. We define the squared-error loss as $\ell_\tau(\boldsymbol{\theta}) \doteq \frac{1}{2} \left\| \hat{F}_{\boldsymbol{\theta}}(\boldsymbol{X}_\tau) - \boldsymbol{Y}_\tau \right\|^2$. We define the residual on task τ at iterate k as $r_\tau^{(k)} \doteq \hat{F}_{\boldsymbol{\theta}_\tau^{(k)}}(\boldsymbol{X}_\tau) - \boldsymbol{Y}_\tau \in \mathbb{R}^n$. In continual learning, where each task is trained for a finite number of steps K, $\boldsymbol{\theta}_{\tau+1}^{(0)} \doteq \boldsymbol{\theta}_\tau^{(K)}$. Because the ReLU gates are frozen, the model is linear in $\mathbf{W}^{[1]}$. Thus the linearized expression, $\hat{F}_{\boldsymbol{\theta}} = F_{\boldsymbol{\theta}}$ is exact for all $\mathbf{W}^{[1]}$.

Next, we present two lemmas that enable us to define successful training based on ϵ-rank. The proofs and extended explanations are deferred to Section B.

Lemma 4.2 shows that, under NTK dynamics with squared error loss and learning rate η, gradient descent acts directly on the residual by repeatedly multiplying the residual by $(\boldsymbol{I} - \eta \mathbf{G}_\tau)$. Lemma 4.3 reveals that by diagonalizing $\mathbf{G}_\tau$ each residual component contracts independently at a rate associated with the magnitude of eigenvalue.

Lemma 4.2 (Residual dynamics in the linearized regime)**.** *If $\hat{F}_{\boldsymbol{\theta}}(\boldsymbol{X}_\tau) \approx F_{\boldsymbol{\theta}_\tau^{(0)}}(\boldsymbol{X}_\tau) + \mathbf{J}_\tau(\boldsymbol{\theta} - \boldsymbol{\theta}_\tau^{(0)})$, then gradient descent on $\ell_\tau(\boldsymbol{\theta})$ with learning rate η yields as the $k+1$th iterate $r_\tau^{(k+1)} = (\boldsymbol{I} - \eta \mathbf{G}_\tau) r_\tau^{(k)}$, where $\mathbf{G}_\tau = \mathbf{J}_\tau \mathbf{J}_\tau^\top \in \mathbb{R}^{NO \times NO}$. Hence, the final value of the residual for task τ is $r_\tau^{(K)} = (\boldsymbol{I} - \eta \mathbf{G}_\tau)^K r_\tau^{(0)}$.*

Lemma 4.3 (Eigenvalue Decomposition)**.** *Assume $\mathbf{G}_\tau$ has eigenpairs $\{\lambda_{\tau,q}, \mathbf{v}_{\tau,q}\}_{q=1}^N$ with orthonormal $\{\mathbf{v}_{\tau,q}\}$ and $\lambda_{\tau,q} \geq 0$. Expand the initial residual:*

$$r_\tau^{(0)} = \sum_{q=1}^{n} \alpha_{\tau,q} \mathbf{v}_{\tau,q}, \quad \alpha_{\tau,q} \doteq (\mathbf{v}_{\tau,q})^\top r_\tau^{(0)}. \tag{4}$$

Then, for all iterates k, $r_\tau^{(k)} = \sum_{q=1}^n (1 - \eta\lambda_{\tau,q})^k \alpha_{\tau,q} \mathbf{v}_{\tau,q}$.

4.2. Eigenvalue contraction

We can now formalize a notion of *fast* and *slow eigenspaces* based on an explicit threshold derived from the number of per-task iterates K and the learning rate η.

Definition 4.4. A *fast eigenvalue* must shrink by at least a factor of $\gamma \in (0,1)$ after K steps: i.e., $(1-\eta\lambda)K \leq \gamma$. In particular, $\lambda \geq \epsilon_{K,\gamma}$ where $\epsilon_{K,\gamma} \doteq \frac{1-\gamma^{1/K}}{\eta} \approx -\log \gamma / \eta K$. Given eigenvalues $\lambda_{\tau,q}$, define the fast set $\mathcal{F}_\tau \doteq \{q : \lambda_{\tau,q} \geq \epsilon_{K,\gamma}\}$ and slow set $\mathcal{S}_\tau \doteq \{q : \lambda_{\tau,q} < \epsilon_{K,\gamma}\}$. The *$\epsilon$-rank* can now be defined as $|\mathcal{F}_\tau|$.[2]

Definition 4.5 (Projectors)**.** Given eigenvectors $\mathbf{v}$ of $\mathbf{G}_\tau$, we define the fast (resp. slow) projection matrix $\mathbf{P}_\tau^{\mathcal{F}} \doteq \sum_{q \in \mathcal{F}_\tau} \mathbf{v}_{\tau,q} \mathbf{v}_{\tau,q}^\top$ (resp. $\mathbf{P}_\tau^{\mathcal{S}} \doteq \boldsymbol{I} - \mathbf{P}_\tau^{\mathcal{F}}$).

Lemma 4.6. *For any task τ,*

$$\left\| r_\tau^{(K)} \right\| \geq \left\| \mathbf{P}_\tau^{\mathcal{S}} r_\tau^{(K)} \right\| \geq \gamma \left\| \mathbf{P}_\tau^{\mathcal{S}} r_\tau^{(0)} \right\|. \tag{5}$$

The consequence of Lemma 4.6 is that slow eigenvalues are sticky: If a component of the initial residual lies in the slow eigenspace, then after K gradient steps at least a γ fraction of that component remains in that slow subspace. By Definition 4.1, successful training on task $\tau+1$ requires $\{\ell_\tau(\boldsymbol{\theta}_{\tau+1}^{(K)}) \leq \upsilon\}$, which, by Lemma 4.6, can be equivalently stated as $\{\|r_{\tau+1}^{(K)}\| \leq \sqrt{2\upsilon}\}$.

[2] In our notation, we omit dependence of $\mathcal{F}_\tau, \mathcal{S}_\tau$ on K, γ.

Lemma 4.7 (Necessary fast-projection condition)**.** *Using the fast and slow projections, $\epsilon_{K,\gamma}$, and the initial loss residual $r_\tau^{(0)}$, a necessary condition to achieve successful training on task τ is $\gamma \left\| \mathbf{P}_\tau^{\mathcal{S}} r_\tau^{(0)} \right\| \leq \sqrt{2\upsilon}$. Equivalently,*

$$\left\| \mathbf{P}_\tau^{\mathcal{F}} \frac{r_\tau^{(0)}}{\|r_\tau^{(0)}\|} \right\|^2 \geq 1 - \left(\frac{\sqrt{2\upsilon}}{\left\| \gamma r_\tau^{(0)} \right\|^2} \right) \tag{6}$$

To succeed on a new task, the eigenvalues of the initial Gram matrix must lie overwhelmingly in the fast subspace, unless the initial residual magnitude $\left\| r_\tau^{(0)} \right\|^2$ is already tiny. When too many eigenvalues fall below the threshold $\epsilon_{K,\gamma}$, the fast subspace shrinks and the necessary condition in Lemma 4.7 becomes impossible to satisfy. We refer to this phenomenon as *spectral collapse*. Next, we demonstrate how spectral collapse governs successful training in a linearized ReLU network.

5. Permuted MNIST

We now analyze Permuted MNIST to show that, under the frozen-gate NTK model, fresh pixel permutations do not change the infinite-width kernel geometry. Thus, when spectral collapse is observed in finite-width continual training, it is attributable to the curvature of the learned representation having degenerated across previous tasks, rather than to the permutations themselves.

The MNIST dataset (LeCun, 1998) is a collection of N handwritten digits with input dimension $I = 28 \times 28 = 784$ and labels $\boldsymbol{y} \in \{0, \ldots, 9\}$. Let $\boldsymbol{\Pi} \in \mathbb{R}^{I \times I}$ be a permutation matrix with a single 1 in each row and and a of 1 in each column, but not diagonal. A *permuted MNIST task* is obtained by sampling a fresh permutation $\boldsymbol{\Pi}_\tau$ and transforming every input by $\boldsymbol{x} \mapsto \boldsymbol{\Pi}_\tau \boldsymbol{x}$, leaving labels unchanged. Concretely, from an underlying base dataset $\{(\bar{\boldsymbol{x}}_n, \bar{\boldsymbol{y}}_n)\}_{n=1}^N$ we define the task-τ dataset $\boldsymbol{X}_\tau \doteq \{\boldsymbol{\Pi}_\tau \bar{\boldsymbol{x}}_n\}_{n=1}^N$ and $\boldsymbol{Y}_\tau \doteq \{\bar{\boldsymbol{y}}_n\}_{n=1}^N$. Informally, Lemma C.6 states that $[\kappa^\infty]_{o,o'}(\boldsymbol{\Pi}\boldsymbol{x}_n, \boldsymbol{\Pi}\boldsymbol{x}_m) = [\kappa^\infty]_{o,o'}(\boldsymbol{x}_n, \boldsymbol{x}_m)$ for $m, n \in [N]$. This implies the expected MSE loss and frozen-gates Gram matrix is task-invariant under fresh pixel permutations, up to finite-width fluctuations, which means that loss of plasticity is *not* caused by permutations. Instead, spectral collapse is caused by the accumulation of low-rank features and the shrinking of the fast eigenspace during training of prior tasks, which leaves the network ill-conditioned for subsequent permutations.

Softmax Cross-Entropy Loss For multi-class cross-entropy with logits $\boldsymbol{z} = (\boldsymbol{z}_1, \ldots, \boldsymbol{z}_N) \in \mathbb{R}^{NO}$ and block-wise softmax $\mathrm{softmax}(\boldsymbol{z})$, the empirical risk $\ell(\boldsymbol{z}) = \sum_{n=1}^N \ell(\boldsymbol{z}_n, \boldsymbol{y}_n)$ satisfies $\nabla_{\boldsymbol{z}} \ell(\boldsymbol{z}) = \mathrm{softmax}(\boldsymbol{z}) - \boldsymbol{Y}$. In

the NTK regime, with the Jacobian frozen at initialization, gradient descent induces a closed-form update on logits of the form:

$$\boldsymbol{z}^{(k+1)} = \boldsymbol{z}^{(k)} - \eta \mathbf{G}^{(0)} \nabla_{\boldsymbol{z}} \ell(\boldsymbol{z}^{(k)}), \tag{7}$$

$$\mathbf{G}^{(0)} = \mathbf{J}^{(0)} (\mathbf{J}^{(0)})^{\top}, \tag{8}$$

where $\mathbf{J}^{(0)}$ is the initial feature map induced by the inputs and $\mathbf{G}^{(0)} \in \mathbb{R}^{(NO)\times(NO)}$ is the corresponding multi-output Gram matrix with blocks

$$[\mathbf{G}^{(0)}]_{(n,o),(m,o')} = \langle \nabla_{\boldsymbol{\theta}} [F_{\boldsymbol{\theta}}(\boldsymbol{x}_n)]_o, \nabla_{\boldsymbol{\theta}} [F_{\boldsymbol{\theta}}(\boldsymbol{x}_m)]_{o'} \rangle.$$

This is exactly the discrete-time counterpart of the standard NTK function-space dynamics for general losses (Lee et al., 2019) (i.e., $[\kappa_{\boldsymbol{\theta}}(\boldsymbol{x}_n, \boldsymbol{x}_m)]_{o,o'}$). We show in Section C.3 that the linearized weighted error $\mathbf{e}^{(k)}$ evolves according to

$$\mathbf{e}^{(k+1)} \approx (\boldsymbol{I} - \eta \mathbf{C}^{1/2} \mathbf{G}^{(0)} \mathbf{C}^{1/2}) \mathbf{e}^{(k)}, \tag{9}$$

where $\mathbf{C} = \nabla^2_{\boldsymbol{z}} \ell(\boldsymbol{z})$. Consequently, for cross-entropy loss, the eigenvalues that determine collapse are those of the weighted Gram matrix, $\mathbf{G}_{\mathrm{LW}} = \mathbf{C}^{1/2} \mathbf{G}^{(0)} \mathbf{C}^{1/2}$. Since by Lemma C.6, the infinite-width limit of $\mathbf{G}^{(0)}$ remains task-invariant, the geometry of the optimization landscape is not inherently altered by the permutations themselves. The fast set is therefore defined as $\mathcal{F} = \{q : \lambda_q(\mathbf{G}_{\mathrm{LW}}) > \epsilon_{K,\gamma}\}$. Lemma 4.7 now implies a necessary condition for success on a new task: the initial residual must project negligibly onto the *slow* subspace. However, as the size of the fast set ($|\mathcal{F}|$) shrinks due to spectral collapse, this condition becomes statistically impossible to satisfy. This yields a direct mechanistic interpretation of Figure 2: across tasks, training accuracy is positively correlated with the ϵ-rank ($|\{q : \lambda_q(\mathbf{G}_{\mathrm{LW}}) > \epsilon_{K,\gamma}\}|$). $L2$ regularization acts as a counter-force by shifting $\lambda_q(\mathbf{G}_{\mathrm{LW}})$ by a positive constant, ensuring more eigenvalues remain in $\mathcal{F}$. Conversely, low-rank feature initialization exacerbates the loss of plasticity by starting with fewer effective directions, thereby accelerating the reduction of $|\mathcal{F}|$.

6. Mitigating Spectral Collapse

Our NTK-inspired experiment suggests a practical alternative to directly analyzing the complicated Hessian of a nonlinear neural network. Rather than working with the full Hessian, we analyze training in the linearized regime through the Jacobian of model outputs and a function-space Gram operator. For general losses, the relevant function-space operator is the loss-weighted Gram

$$\mathbf{G}_{\mathrm{LW}} \doteq \mathbf{C}^{1/2} \mathbf{G}^{0} \mathbf{C}^{1/2}, \quad \mathbf{G}^{0} \doteq \mathbf{J}\mathbf{J}^{\top}. \tag{10}$$

This operator has two crucial properties for our purposes: It is positive semidefinite for standard convex-in-logits losses,

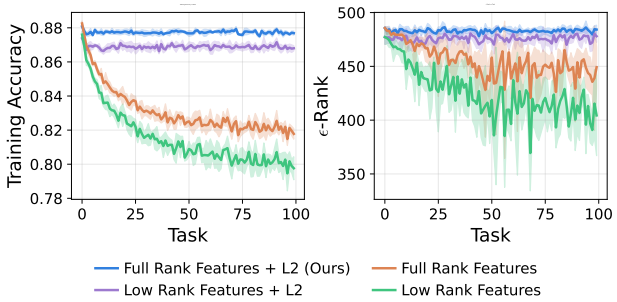


Figure 2. The left plot shows the training accuracy of the width 1000 linearized ReLU network. The right plot shows the $\epsilon_{K,\gamma}$-rank of $\mathbf{G}_{\mathrm{LW}}$ with $K = |\text{train set}|$ and $\gamma = 1e^{-3}$. Accuracy and $\epsilon_{K,\gamma}$-rank are correlated.

and it is spectrally equivalent to the generalized Gauss Newton matrix $\mathbf{H}_{\mathrm{GGN}} \doteq \mathbf{J}^{\top}\mathbf{C}\mathbf{J}$. Indeed, letting $\boldsymbol{A} \doteq \mathbf{C}^{1/2}\mathbf{J}$ yields $\mathbf{G}_{\mathrm{LW}} = \boldsymbol{A}\boldsymbol{A}^{\top}$, while $\mathbf{H}_{\mathrm{GGN}} = \boldsymbol{A}^{\top}\boldsymbol{A}$, so they share the same nonzero eigenvalues. As a result, the ϵ-rank of $\mathbf{G}_{\mathrm{LW}}$ equals that of $\mathbf{H}_{\mathrm{GGN}}$, but now expressed in a parameter space where it connects directly to curvature, optimization stability, and regularization. In Figure 3, we demonstrate the correspondence between successful training and the ϵ-rank of the Hessian matrix in Continual ImageNet. Additional empirical findings appear in Sections H to J.

We therefore treat the $\mathbf{H}_{\mathrm{GGN}}$ as the go-between linking the NTK function-space analysis to a parameter-space curvature target. This choice is also computationally motivated: unlike the true Hessian, $\mathbf{H}_{\mathrm{GGN}}$ can be approximated efficiently using structured second-order methods (e.g., layerwise block and Kronecker-factor approximations), enabling us to define an explicit regularizer that increases the effective rank of the approximate curvature without ever materializing a the full Hessian, a $P \times P$ matrix.

In this section, we operationalize this idea: we build a scalable approximation of $\mathbf{H}_{\mathrm{GGN}}$, identify which factors control its rank in MLPs, and introduce L2-ER as a simple objective that prevents curvature collapse. To do so, we first obtain the Gauss-Newton decomposition of the Hessian from Equation (1).

A neural network has parameters corresponding to each layer. Given a specific parameter, $[\mathbf{W}^{[l]}]_{i,j}$ in layer l and another parameter, $[\mathbf{W}^{[l']}]_{i',j'}$, in $l' \neq l$), in practice $\partial^2 \mathcal{L} / \partial [\mathbf{W}^{[l]}]_{i,j} \partial [\mathbf{W}^{[l']}]_{i',j'}$ is approximately 0 (Becker et al., 1988), where $\mathcal{L}$ is the loss. Therefore, Hessian admits a natural "layer-wise block" structure.

Kronecker-factored approximate curvature (KFAC) is a method to approximate a matrix using only block-diagonal $\{\boldsymbol{M}^l\}_{l \in L}$ elements. In particular, KFAC is frequently used

[3]The x-axis is *normalized ϵ-rank($\mathbf{H}$)*, meaning ϵ-rank($\mathbf{H}$) divided by P, for $\epsilon = 0.03$.

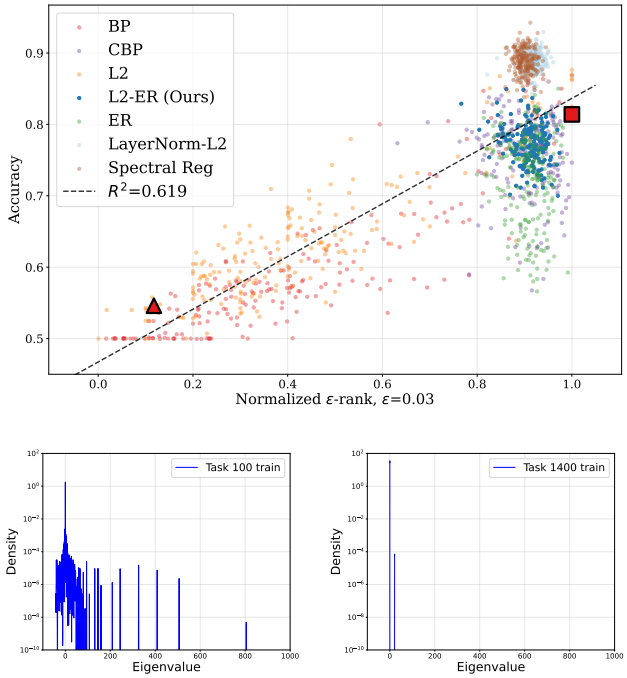


Figure 3. (Top) Final test accuracy vs. Curvature (i.e., normalized ϵ-rank($\mathbf{H}$) at task initialization)[3] on Continual ImageNet. A linear fit (dotted line) highlights the positive association between curvature and accuracy. (Bottom Left) ■: BP Hessian eigenspectrum *before spectral collapse* at task 100. (Bottom Right) ▲: BP Hessian eigenspectrum *after spectral collapse* at task 1400.

for approximating the numerical quantities related to the Hessian like $\mathbf{H}_{\text{GGN}}$ or Fisher Information matrices (Dangel et al., 2025; Martens, 2016). We therefore justify our effective-rank-based regularizer using a KFAC approximation of the Hessian. We then argue that increasing the rank of the KFAC-approximate $\mathbf{H}_{\text{GGN}}$ implies increasing the rank of the true Hessian for mean-square-error (MSE) and softmax-cross-entropy losses (SCE).

As an analogue of a multi-layer MLP, we use a multi-layer linear neural network without bias to justify our regularizer. The layer l in a neural network has an associated weight matrix $\mathbf{W}^{[l]}$. For a dataset of size N, let $\mathbf{a}_n^{[l-1]}$ be the input activations to the layer l for the nth datum, producing pre-activation feature $\mathbf{h}_n^{[l]} = \mathbf{W}^{[l]}\mathbf{a}_n^{[l-1]}$. Let $\boldsymbol{\delta}_{n,o}^{[l]} = \nabla_{\boldsymbol{z}_n^{[l]}}[\ell(\boldsymbol{z}_n, \boldsymbol{y}_n)]_o$ be the gradient of the loss with respect to the layer's output o. We can now define $\mathbf{H}_{\text{GGN}}$ as a sum of Kroneker products :

$$\mathbf{H}_{\text{GGN}} = A \sum_{n=1}^{N} \sum_{o=1}^{O} (\mathbf{a}_n^{[l-1]}\mathbf{a}_n^{[l-1]\top}) \otimes (\boldsymbol{\delta}_{n,o}^{[l]}\boldsymbol{\delta}_{n,o}^{[l]\top}),$$

where A is an accumulation factor[4] and $\otimes$ denotes the Kronecker product operator. The term $\mathbf{a}_n^{[l-1]}\mathbf{a}_n^{[l-1]\top}$ captures the input feature statistics, while $\boldsymbol{\delta}_{n,o}^{[l]}\boldsymbol{\delta}_{n,o}^{[l]\top}$ captures the backpropogated output gradient statistics. We can now apply the computationally tractable KFAC approximation by decoupling the summations over the inputs and gradients (Dangel et al., 2025): $\mathbf{H}_{\text{GGN}} \approx \widehat{\mathbf{H}}_{\text{GGN}}$, where $\widehat{\mathbf{H}}_{\text{GGN}}$ is

$$\left(A \sum_{n=1}^{N} \mathbf{a}_n^{[l-1]}\mathbf{a}_n^{[l-1]\top}\right) \otimes \left(\frac{1}{N} \sum_{n=1}^{N} \sum_{o=1}^{O} \boldsymbol{\delta}_{n,o}^{[l]}\boldsymbol{\delta}_{n,o}^{[l]\top}\right).$$

In our sample-based setting, the approximation error can be expressed as the difference between the average values of the Kronecker products and the Kronecker product of average values (see Section D):

$$\text{KFAC Error} = \left|\mathbf{H}_{\text{GGN}} - \widehat{\mathbf{H}}_{\text{GGN}}\right|.$$

The approximation error is small when the joint distribution over $\mathbf{a}^{[l]}$, $\mathbf{a}^{[l]\top}$, $\boldsymbol{\delta}^{[l]}$, and $\boldsymbol{\delta}^{[l]\top}$ is a multivariate Gaussian, which appears to hold in practice (Martens & Grosse, 2015). Alternatively, the approximation can be viewed as an assumption of statistical independence between $\mathbf{a}_n^{[l-1]}\mathbf{a}_n^{[l-1]\top}$ and $\boldsymbol{\delta}_n^{[l]}\boldsymbol{\delta}_n^{[l]\top}$, which holds for deep linear networks with squared-error loss (Bernacchia et al., 2018). Nonetheless, KFAC has been found to capture useful curvature information for convolutional neural networks (Grosse & Martens, 2016), recurrent neural networks (Martens et al., 2018) and vision-transformers (Eschenhagen et al., 2023).

Maximizing the Rank of the Input Covariance Matrix For each layer l, the KFAC approximation $\widehat{\mathbf{H}}_{\text{GGN}}$ has the block form $\mathbb{E}_n[\mathbf{a}_n^{[l-1]}\mathbf{a}_n^{[l-1]\top}] \otimes \mathbb{E}_n[\boldsymbol{\delta}_n^{[l]}\boldsymbol{\delta}_{n\top}^{[l]}]$. Since $\text{rank}(\boldsymbol{A} \otimes \boldsymbol{B}) = \text{rank}\boldsymbol{A} \cdot \text{rank}\boldsymbol{B}$, the rank of $\widehat{\mathbf{H}}_{\text{GGN}}$ is monotone in the rank of both the input covariance matrix $\mathbb{E}_n[\mathbf{a}_n^{[l-1]}\mathbf{a}_n^{[l-1]\top}]$ and the gradient covariance matrix $\mathbb{E}_n[\boldsymbol{\delta}_n^{[l]}\boldsymbol{\delta}_{n\top}^{[l]}]$. The rank of $\mathbb{E}_n[\mathbf{a}_n^{[l-1]}\mathbf{a}_n^{[l-1]\top}]$ coincides with rank of the input representation of layer l itself. This latter quantity, known as the *effective feature rank (ER)*, has been shown to be a useful indicator of plasticity (Dohare et al., 2024; Lyle et al., 2023), and standard (i.e., single-task) RL (Kumar et al., 2021).

Maximizing the Rank of the Hessian Adding $L2$ regularization is required to ensure the rank of the Hessian is sufficiently high, since $\mathbf{H}_{\text{GGN}}$ does not take into account the contribution of the residual matrix $\mathbf{R}$. Thus, preserving high (effective) rank of the activation covariances (and avoiding degenerate gradient covariances) directly preserves the dimensionality of the non-collapsed curvature subspace captured by $\mathbf{H}_{\text{GGN}}$, motivating an explicit effective-rank penalty on $\mathbb{E}_n[\mathbf{a}_n^{[l-1]}\mathbf{a}_n^{[l-1]\top}]$.

L2-ER Regularization Given task τ, with data distribution $\boldsymbol{d}_\tau$ and loss function ℓ_τ, standard gradient descent mini-

[4]Typical choices are $1/n$, 1, etc. (Dangel et al., 2025).

mizes $\mathbb{E}_{\boldsymbol{d}_\tau}[\ell_\tau(F_{\boldsymbol{\theta}}(\boldsymbol{x}), \boldsymbol{y})]$. We propose the $L2$-ER regularizer, which combines effective rank and $L2$ penalties, leading to a theoretically grounded and practical objective:

$$\min_{\boldsymbol{\theta}} \mathcal{L}_\tau(\boldsymbol{\theta}) = \min_{\boldsymbol{\theta}} \underbrace{\mathop{\mathbb{E}}_{\boldsymbol{d}_\tau}\Big[\ell_\tau(F_{\boldsymbol{\theta}}(\boldsymbol{x}), \boldsymbol{y})\Big]}_{\text{Loss}} + \underbrace{\lambda\|\boldsymbol{\theta}\|_2^2}_{L2 \text{ regularization}} - \underbrace{\beta \text{erank}\left(\sum_{l \in L} \mathbb{E}_n\left[\mathbf{a}_n^{[l-1]}\, \mathbf{a}_n^{[l-1]\top}\right]\right)}_{\text{Effective rank penalty}}$$

In summary, we add an empirically computable effective rank penalty to the objective that increases the ϵ-rank of the approximate GGN Hessian, $\widehat{\mathbf{H}}_{\text{GGN}}$, which in turn affects the bulk of the eigenspectrum of the true Hessian.

7. Experiments

We conduct experiments across a range of network architectures and continual learning tasks adapted from supervised and reinforcement learning environments.

1. **Permuted MNIST** (Goodfellow et al., 2013): a variant of the MNIST dataset (LeCun, 1998) where new tasks are created by permuting the pixels in all images in the same way. Each permutation represents a new task. We model the classifier with a ReLU MLP.

2. **Continual Imagenet** (Dohare et al., 2024): an adaptation of Imagenet (Krizhevsky et al., 2012) where pairs of classes are randomly sampled for binary classification. Distinguishing the two classes in each new class pair is a new task. We use a convolutional neural network.

3. **Incremental CIFAR** (Dohare et al., 2024): an adaptation of the CIFAR-100 dataset (Krizhevsky et al., 2009) where classes are added incrementally. Starting with 5 classes, each task adds 5 new classes until all 100 classes are included. We use a ResNet architecture (He et al., 2015a).

4. **Slippery Ant** (Dohare et al., 2024): a continual reinforcement learning environment where the friction between the ant and the ground changes every T timesteps, forcing the agent to continually adapt its policy.

We compare L2-ER against six baselines: vanilla backpropagation (BP), L2 regularization (L2), effective rank regularization (ER), continual backpropagation (Dohare et al., 2024) (CBP), layer-normalization with L2 (Lyle et al., 2024) (LayerNorm-L2), and spectral regularization (Lewandowski et al., 2024a) (Spectral). For Incremental CIFAR, where task difficulty increases with the number of classes, we include a RESET baseline that reinitializes parameters at each task change; algorithms outperforming RESET are considered to maintain plasticity. We perform extensive hyperparameter sweeps and report accuracy for supervised learning and online returns for RL. Implementation details appear in Sections H to K.

Performance Figure 4 depicts results across all four environments. L2-ER maintains plasticity in all, and provides an early performance boost on Permuted MNIST before plasticity loss would typically occur. L2 preserves plasticity in most cases but fails on Continual ImageNet; CBP succeeds except on Incremental CIFAR. LayerNorm-L2 performs well except on Permuted MNIST. Spectral Regularization succeeds except on Incremental CIFAR but requires roughly twice the training time (Figure 15). ER and BP exhibit plasticity loss across all environments.

8. Discussion

Our goal in this work is to analyze continual learning with a simplified model of a neural network, to find and isolate the mechanisms that contribute to loss of plasticity. Our model of choice is the NTK, which approximates neural network functions by linearizing around initialization. We show that gradient descent applied to the linearized model under the squared error loss progressively reduces the components of the error residual along directions associated with large eigenvalues of the Gram matrix $\mathbf{G} = \mathbf{J}\mathbf{J}^\top$. As a result, components aligned with small eigenvalue directions decay slowly over a finite training horizon. This leads to a simple constraint on learnability: if the new-task residual has substantial mass in these slow directions, then it cannot be reduced sufficiently within the allotted K gradient steps, unless it is already small or happens to align with the fast directions. In this sense, successful learning requires that most of the residuals lie in directions where the model can make rapid progress.

However, in the NTK regime under squared loss, the kernel matrix, $\mathbf{G} = \mathbf{J}\mathbf{J}^\top$ remains invariant over training. As a result, we shift our attention to classification, where the weighted Gram matrix $\mathbf{G}_{\text{LW}} = \mathbf{C}^{1/2}\mathbf{G}^{(0)}\mathbf{C}^{1/2}$ governs training dynamics. This distinction is crucial on Permuted MNIST. We prove that, in the infinite-width frozen-gate model, the raw kernel $\mathbf{G}^{(0)}$ remains invariant to pixel permutations. Thus, the permutations themselves alone are not enough to explain spectral collapse. Empirically, however, the loss-weighted Gram matrix, $\mathbf{G}_{\text{LW}}$ does undergo spectral collapse, and its ϵ-rank tracks the loss of plasticity due to the loss-weighted geometry that governs classification dynamics.

Having built an understanding of the mechanisms behind plasticity loss using the NTK, we apply our findings to

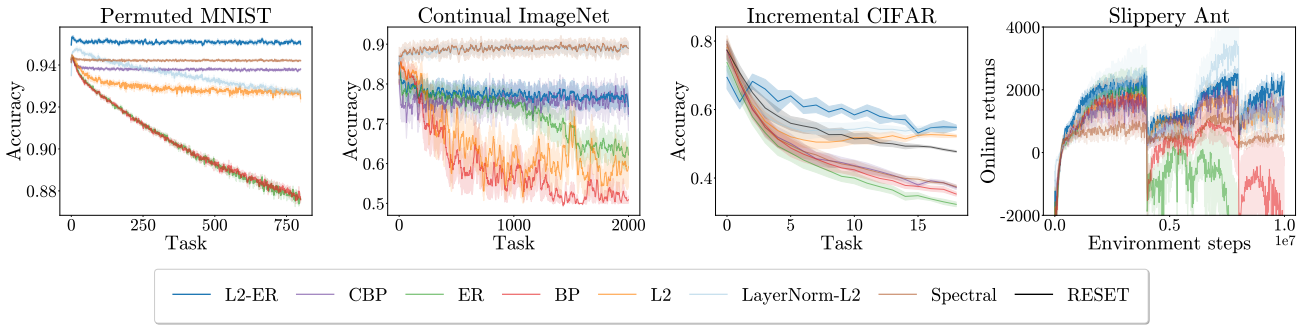


Figure 4. Performance across all four environments. Classification accuracy is reported for Permuted MNIST, Continual ImageNet, and Incremental CIFAR, while online returns are reported for Slippery Ant. Results are averaged across 5 seeds for all environments, except Continual ImageNet, where we used 10 seeds due to higher variance. Shaded regions denote a 95% confidence interval.

conventional deep non-linear networks. In particular, we show that the loss-weighted Gram matrix $\mathbf{G}_{\text{LW}}$, is spectrally equivalent to the generalized Gauss Newton matrix, $\mathbf{H}_{\text{GGN}}$. Consequently, we propose the $L2$-ER regularizer, which aims to force a wide-bulk to the Hessian eigenspectrum to maintain plasticity. We measure the Hessian spectrum at new-task initialization and observe that eigenspectrum collapse precedes failures to learn. When we apply our intervention, the spectrum of the Hessian retains a wide bulk and maintains plasticity. When we remove the intervention, the bulk collapses and so does plasticity. Moreover, we find that other plasticity preserving methods, like CBP also maintain a wide bulk.

$L2$-ER is much cheaper than explicitly estimating the full Hessian spectrum, and the eigenspectrum diagnostics are not used inside the regularizer. However, the effective-rank penalty does introduce a nontrivial SVD cost, which we amortize over U steps. For each layer $l \in L_{ER}$ in the network to be regularized, we collect the activations $\mathbf{a}^{[l]} \in \mathbb{R}^{d^l}$, and stack them row-wise over a batch of data B as $\mathbf{A}^{[l]}_{\text{ER}} = \left[\mathbf{a}^{[l]}_1 \cdots, \mathbf{a}^{[l]}_B\right]^\top \in \mathbb{R}^{B\times d^l}$. The amortized SVD cost of $\mathbf{A}^{[l]}_{\text{ER}}$ is $\sum_{l\in L_{\text{ER}}} O\big(\frac{1}{U}\min\{B(d^l)^2,\ B^2 d^l\}\big)$ with memory $O(Bd^l)$.

In our experiments, small feature buffers were sufficient, and the ER learning rate was the most sensitive ER hyperparameter; the ER batch/window mainly stabilized the singular-value estimate. We recommend tuning the task optimizer first, then the L2 coefficient, and finally the ER learning rate, typically below the task learning rate. The ER buffer should be the smallest one that gives a stable singular-value estimate, and ER updates should be applied as infrequently as possible while retaining their benefit. For larger models, selected-layer regularization, bottleneck-layer regularization, randomized or truncated SVD, and sketching methods may be needed. We do not yet have language-model results; extending $L2$-ER to such models will likely require applying the penalty only to selected representations, adapters, or other low-dimensional modules, and attention layers may require a more specialized curvature approximation.

We believe our work opens several directions for future work. Having identified when a new task becomes unlearnable, our causal claim that spectral collapse drives loss of plasticity is empirical. Future work could use tools from singular learning theory (Watanabe, 2009) or stochastic gradient dynamics (Wu et al., 2018; Wu & Su, 2023; Xu et al., 2026) to understand the evolution of the Hessian eigenspectrum in continual learning.

9. Conclusion

We identify Hessian spectral collapse as the central mechanism for plasticity loss in continual learning. Analyzing a linearized ReLU network, we derive explicit ϵ-rank conditions for successful training and establish that the loss-weighted Gram matrix is spectrally equivalent to the Generalized Gauss-Newton approximation, bridging NTK dynamics to parameter-space curvature. Using the KFAC approximation of the Hessian, we propose $L2$-ER, a simple regularizer that empirically prevents spectral collapse and preserves plasticity across diverse benchmarks. These results position spectral collapse as both a unifying explanation and a practical diagnostic for plasticity loss. We leave adapting our theoretical framework to the unique dynamics of RL, where the Hessian is influenced by non-stationary policy and value functions, for future work. The implications of this work extend to Silver & Sutton's (2025) long-term vision for adaptive AI systems. Consider a large language model or robot acting as an assistant on a multi-year long project. By providing a stable foundation for continual learning, our findings represent a crucial step towards building more capable and continuously adapting AI systems of the future.

Acknowledgments

Arjun Prakash and Ruo Yu Tao were supported by the Office of Naval Research (ONR) grant N00014-22-1-2592 and N00014-24-1-2657. Saket Tiwari was supported by ONR REPRISM MURI N00014-24-1-2603, ONR grant 00014-22-1-2592 and partial funding was also provided by the Robotics and AI Institute.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Abbas, Z., Zhao, R., Modayil, J., White, A., and Machado, M. C. Loss of plasticity in continual deep reinforcement learning. In *Conference on lifelong learning agents*, pp. 620–636. PMLR, 2023.

Arora, S., Du, S. S., Hu, W., Li, Z., Salakhutdinov, R. R., and Wang, R. On exact computation with an infinitely wide neural net. *Advances in neural information processing systems*, 32, 2019.

Ash, J. and Adams, R. P. On warm-starting neural network training. *Advances in neural information processing systems*, 33:3884–3894, 2020.

Baker, M. M., New, A., Aguilar-Simon, M., Al-Halah, Z., Arnold, S. M., Ben-Iwhiwhu, E., Brna, A. P., Brooks, E., Brown, R. C., Daniels, Z., Daram, A., Delattre, F., Dellana, R., Eaton, E., Fu, H., Grauman, K., Hostetler, J., Iqbal, S., Kent, C., Ketz, N., Kolouri, S., Konidaris, G., Kudithipudi, D., Learned-Miller, E., Lee, S., Littman, M. L., Madireddy, S., Mendez, J. A., Nguyen, E. Q., Piatko, C., Pilly, P. K., Raghavan, A., Rahman, A., Ramakrishnan, S. K., Ratzlaff, N., Soltoggio, A., Stone, P., Sur, I., Tang, Z., Tiwari, S., Vedder, K., Wang, F., Xu, Z., Yanguas-Gil, A., Yedidsion, H., Yu, S., and Vallabha, G. K. A domain-agnostic approach for characterization of lifelong learning systems. *Neural Networks*, 160:274–296, 2023. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2023.01.007.

Bakhtin, A., Brown, N., Dinan, E., Farina, G., Flaherty, C., Fried, D., Goff, A., Gray, J., Hu, H., et al. Human-level play in the game of diplomacy by combining language models with strategic reasoning. *Science*, 378(6624): 1067–1074, 2022.

Becker, S., Le Cun, Y., et al. Improving the convergence of back-propagation learning with second order methods. In *Proceedings of the 1988 connectionist models summer school*, volume 2, 1988.

Bernacchia, A., Lengyel, M., and Hennequin, G. Exact natural gradient in deep linear networks and its application to the nonlinear case. *Advances in Neural Information Processing Systems*, 31, 2018.

Bjorck, J., Gomes, C. P., and Weinberger, K. Q. 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.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. JAX: composable transformations of Python+NumPy programs, 2018.

Cha, J., Chun, S., Lee, K., Cho, H.-C., Park, S., Lee, Y., and Park, S. Swad: Domain generalization by seeking flat minima. In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 22405–22418. Curran Associates, Inc., 2021.

Chung, W., Cherif, L., Meger, D., and Precup, D. Parseval regularization for continual reinforcement learning. *Advances in Neural Information Processing Systems*, 37: 127937–127967, 2024.

Dangel, F., Mucsányi, B., Weber, T., and Eschenhagen, R. Kronecker-factored approximate curvature (kfac) from scratch, 2025.

Delfosse, Q., Schramowski, P., Mundt, M., Molina, A., and Kersting, K. Adaptive rational activations to boost deep reinforcement 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=g90ysX1sVs.

Doan, T., Bennani, M. A., Mazoure, B., Rabusseau, G., and Alquier, P. A theoretical analysis of catastrophic forgetting through the ntk overlap matrix. In *International Conference on Artificial Intelligence and Statistics*, pp. 1072–1080. PMLR, 2021.

Dohare, S., Hernandez-Garcia, J. F., Lan, Q., Rahman, P., Mahmood, A. R., and Sutton, R. S. Loss of plasticity in deep continual learning. *Nature*, 632(8026): 768–774, August 2024. ISSN 1476-4687. doi: 10.1038/s41586-024-07711-7. Publisher: Nature Publishing Group.

Dwaraknath, R. V., Ergen, T., and Pilanci, M. Fixing the ntk: From neural network linearizations to exact convex programs. *Advances in Neural Information Processing Systems*, 36:1539–1559, 2023.

Eschenhagen, R., Immer, A., Turner, R., Schneider, F., and Hennig, P. Kronecker-factored approximate curvature for modern neural network architectures. *Advances in Neural Information Processing Systems*, 36:33624–33655, 2023.

Ghorbani, B., Krishnan, S., and Xiao, Y. An investigation into neural net optimization via hessian eigenvalue density. In *International Conference on Machine Learning*, pp. 2232–2241. PMLR, 2019.

Godfrey, L. B. An evaluation of parametric activation functions for deep learning. In *2019 IEEE international conference on systems, man and cybernetics (SMC)*, pp. 3006–3011. IEEE, 2019.

Golub, G. H. and Welsch, J. H. Calculation of gauss quadrature rules. *Mathematics of computation*, 23(106):221–230, 1969.

Goodfellow, I. J., Mirza, M., Xiao, D., Courville, A., and Bengio, Y. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211*, 2013.

Grosse, R. and Martens, J. A kronecker-factored approximate fisher matrix for convolution layers. In *International Conference on Machine Learning*, pp. 573–582. PMLR, 2016.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. corr abs/1512.03385 (2015), 2015a.

He, K., Zhang, X., Ren, S., and Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015b.

Hinton, G. E. and Salakhutdinov, R. R. Reducing the dimensionality of data with neural networks. *science*, 313 (5786):504–507, 2006.

Hochreiter, S. and Schmidhuber, J. Flat minima. *Neural Computation*, 9(1):1–42, 01 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.1.1.

Hosseini, M., Kerridge, S., Allen, L., Kiermer, V., and Holmes, K. Credit roles and example research tasks that could be attributed to them, January 2026. URL https://doi.org/10.5281/zenodo.18421449.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., Chen, W., et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.

Igl, M., Farquhar, G., Luketina, J., Boehmer, W., and Whiteson, S. 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.

Jacot, A., Gabriel, F., and Hongler, C. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems*, 31, 2018.

Javed, K. and Sutton, R. S. The big world hypothesis and its ramifications for artificial intelligence. In *Finding the Frame: An RLC Workshop for Examining Conceptual Frameworks*, 2024.

Juliani, A. and Ash, J. A study of plasticity loss in on-policy deep reinforcement learning. *Advances in Neural Information Processing Systems*, 37:113884–113910, 2024.

Karakida, R., Akaho, S., and Amari, S. Pathological spectra of the fisher information metric and its variants in deep neural networks. *Neural Comput.*, 33(8):2274–2307, 2021. doi: 10.1162/NECO_A_01411. URL https://doi.org/10.1162/neco_a_01411.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017.

Klein, T., Miklautz, L., Sidak, K., Plant, C., and Tschiatschek, S. Plasticity loss in deep reinforcement learning: A survey, 2024.

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

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

Kumar, A., Agarwal, R., Ghosh, D., and Levine, S. 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.

LeCun, Y. The mnist database of handwritten digits. 1998.

Lee, J., Xiao, L., Schoenholz, S., Bahri, Y., Novak, R., Sohl-Dickstein, J., and Pennington, J. Wide neural networks of any depth evolve as linear models under gradient descent. *Advances in neural information processing systems*, 32, 2019.

Lewandowski, A., Bortkiewicz, M., Kumar, S., Schuurmans, D., Ostaszewski, M., Machado, M. C., et al. Learning continually by spectral regularization. *arXiv preprint arXiv:2406.06811*, 2024a.

Lewandowski, A., Tanaka, H., Schuurmans, D., and Machado, M. C. Directions of curvature as an explanation for loss of plasticity, 2024b.

Lu, C., Kuba, J., Letcher, A., Metz, L., Schroeder de Witt, C., and Foerster, J. Discovered policy optimisation. *Advances in Neural Information Processing Systems*, 35: 16455–16468, 2022.

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

Lyle, C., Zheng, Z., Khetarpal, K., van Hasselt, H., Pascanu, R., Martens, J., and Dabney, W. Disentangling the causes of plasticity loss in neural networks. *arXiv preprint arXiv:2402.18762*, 2024.

Martens, J. *Second-order optimization for neural networks*. University of Toronto (Canada), 2016.

Martens, J. and Grosse, R. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pp. 2408–2417. PMLR, 2015.

Martens, J., Ba, J., and Johnson, M. Kronecker-factored curvature approximations for recurrent neural networks. In *International Conference on Learning Representations*, 2018.

Montufar, G. F., Pascanu, R., Cho, K., and Bengio, Y. On the number of linear regions of deep neural networks. *Advances in neural information processing systems*, 27, 2014.

Obando-Ceron, J. S., Courville, A. C., and Castro, P. S. In value-based deep reinforcement learning, a pruned network is a good network. In Salakhutdinov, R., Kolter, Z., Heller, K. A., Weller, A., Oliver, N., Scarlett, J., and Berkenkamp, F. (eds.), *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, Proceedings of Machine Learning Research, pp. 38495–38519. PMLR / OpenReview.net, 2024. URL https://proceedings.mlr.press/v235/obando-ceron24a.html.

Parisi, G. I., Kemker, R., Part, J. L., Kanan, C., and Wermter, S. Continual lifelong learning with neural networks: A review. *CoRR*, abs/1802.07569, 2018.

Pearlmutter, B. A. Fast exact multiplication by the hessian. *Neural computation*, 6(1):147–160, 1994.

Pouplin, A., Roy, H., Singh, S. P., and Arvanitidis, G. On the curvature of the loss landscape, July 2023. arXiv:2307.04719 [cs].

Roy, O. and Vetterli, M. The effective rank: A measure of effective dimensionality. In *2007 15th European signal processing conference*, pp. 606–610. IEEE, 2007.

Savitzky, A. and Golay, M. J. E. Smoothing and differentiation of data by simplified least squares procedures. *Analytical Chemistry*, 36(8):1627–1639, 1964. doi: 10.1021/ac60214a047.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms, 2017.

Shang, W., Sohn, K., Almeida, D., and Lee, H. Understanding and improving convolutional neural networks via concatenated rectified linear units. In *international conference on machine learning*, pp. 2217–2225. PMLR, 2016.

Silver, D. and Sutton, R. S. Welcome to the era of experience. *Google AI*, 1, 2025.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

Siméoni, O., Vo, H. V., Seitzer, M., Baldassarre, F., Oquab, M., Jose, C., Khalidov, V., Szafraniec, M., Yi, S. E., Ramamonjisoa, M., Massa, F., HAZIZA, D., Wehrstedt, L., Wang, J., Darcet, T., Moutakanni, T., Sentana, L., Roberts, C., Vedaldi, A., Tolan, J., Brandt, J., Couprie, C., Mairal, J., Jegou, H., Labatut, P., and Bojanowski, P. DINOv3. *Transactions on Machine Learning Research*, 2026. ISSN 2835-8856. URL https://openreview.net/forum?id=2NlGyqNjns. Featured Certification.

Sokar, G., Agarwal, R., Castro, P. S., and Evci, U. The dormant neuron phenomenon in deep reinforcement learning. In *International Conference on Machine Learning*, pp. 32145–32168. PMLR, 2023.

Tang, H., Obando-Ceron, J., Castro, P. S., Courville, A., and Berseth, G. Mitigating plasticity loss in continual reinforcement learning by reducing churn. In Singh, A., Fazel, M., Hsu, D., Lacoste-Julien, S., Berkenkamp, F., Maharaj, T., Wagstaff, K., and Zhu, J. (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 58883–58904. PMLR, 13–19 Jul 2025.

Watanabe, S. *Algebraic Geometry and Statistical Learning Theory*. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, 2009.

Wu, L. and Su, W. J. The implicit regularization of dynamical stability in stochastic gradient descent. In *International Conference on Machine Learning*, pp. 37656–37684. PMLR, 2023.

Wu, L., Ma, C., et al. How sgd selects the global minima in over-parameterized learning: A dynamical stability perspective. *Advances in Neural Information Processing Systems*, 31, 2018.

Wu, Y., Zhu, X., Wu, C., Wang, A., and Ge, R. Dissecting hessian: Understanding common structure of hessian in neural networks, 2022. URL https://arxiv.org/abs/2010.04261.

Xu, Y., Beneventano, P., Chuang, I., and Ziyin, L. Does sgd seek flatness or sharpness? an exactly solvable model, 2026. URL https://arxiv.org/abs/2602.05065.

Yoshida, Y. and Miyato, T. Spectral norm regularization for improving the generalizability of deep learning. *arXiv preprint arXiv:1705.10941*, 2017.

Zhao, J., Singh, S. P., and Lucchi, A. Theoretical characterisation of the gauss-newton conditioning in neural networks. *arXiv preprint arXiv:2411.02139*, 2024.

A. Summary of Contributions

We describe the contributions of each author following the CRediT Taxonomy (Hosseini et al., 2026).

Arjun contributed to the conceptualization and the investigation of the Hessian. He was the primary contributor to the NTK analysis and the KFAC interpretation of the the $L2$-ER regularizer. He contributed to the original draft, and the final revision by adapting the NTK analysis and responding to reviewers. He also led the general project administration.

Naicheng was the primary contributor to the investigation and conceptualization of spectral collapse. He was the primary contributor to the $L2$-ER regularizer and confirmed its practicality together with Kaicheng. He led the project administration for the NeurIPS ARLET workshop submission and contributed to the original draft.

Kaicheng was a contributor to the $L2$-ER regularizer and to confirming its practical effectiveness across continual learning settings. He led the development of the entire codebase and was responsible for the main experimental pipeline. He led the empirical evaluation of the paper, including the performance experiments across all benchmarks and the Hessian spectrum analyses used to diagnose spectral collapse. His works provide the main experimental evidence supporting the paper's claims.

B. Neural Tangent Kernel Analysis

B.1. Introduction on NTKs

For completeness we provide a short introduction to NTKs. The neural tangent kernel (NTK) has been a popular method to characterize the training dynamics of (stochastic) gradient descent with a infinitesimally small learning rates and the infinite-width limit (i.e., $M \to \infty$). The main principle behind NTK theory is to approximate the neural network function by linearizing it around it's parameters around the initialization. The seminal work (Jacot et al., 2018) show that if F is an appropriately scaled neural network, with $\boldsymbol{\theta}$ initialized i.i.d. from standard (or isotropic) Gaussian's (i.e. $\boldsymbol{\theta} \sim \mathcal{N}(0, \boldsymbol{I})$) then:

$$\hat{F}_{\boldsymbol{\theta}}(\boldsymbol{X}_\tau) \approx F_{\boldsymbol{\theta}_\tau^{(0)}}(\boldsymbol{X}_\tau) + \mathbf{J}_\tau(\boldsymbol{\theta} - \boldsymbol{\theta}_\tau^{(0)}).$$

The linearized model $\hat{F}_{\boldsymbol{\theta}}$ can be interpreted as a kernel method where the Jacobian is the feature map:

$$\boldsymbol{\phi}_{\boldsymbol{\theta}^{(0)}}(\boldsymbol{X}) \doteq \left.\frac{\partial F_{\boldsymbol{\theta}}(\boldsymbol{X})}{\partial \boldsymbol{\theta}}\right|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(0)}} = \mathbb{R}^{NO \times P}. \tag{11}$$

The kernel induced by the feature map is the NTK and is dependent on the random initialization $\boldsymbol{\theta}^{(0)}$. The neural tangent kernel $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}^{O \times O}$ is :

$$\kappa_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{x}') \doteq \mathop{\mathbb{E}}_{\boldsymbol{\theta} \sim \mathcal{N}(0, \boldsymbol{I})} \left[\left\langle \frac{\partial F_{\boldsymbol{\theta}}(\boldsymbol{x})}{\partial \boldsymbol{\theta}}, \frac{\partial F_{\boldsymbol{\theta}}(\boldsymbol{x}')}{\partial \boldsymbol{\theta}} \right\rangle \Bigg|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(0)}} \right] \tag{12}$$

$$= \mathop{\mathbb{E}}_{\boldsymbol{\theta}^{(0)} \sim \mathcal{N}(0, \boldsymbol{I})} [\langle \nabla_{\boldsymbol{\theta}} F_{\boldsymbol{\theta}^{(0)}}(\boldsymbol{x}), \nabla_{\boldsymbol{\theta}} F_{\boldsymbol{\theta}^{(0)}}(\boldsymbol{x}') \rangle]. \tag{13}$$

The evaluating the kernel on a finite dataset (i.e. for all pairs $(\boldsymbol{x}, \boldsymbol{x}') \in \boldsymbol{X}$) gives the empirical NTK matrix:

$$\kappa_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{x}') = \mathbf{J}\mathbf{J}^\top = \mathbf{G} \in \mathbb{R}^{NO \times NO}. \tag{14}$$

The celebrated result from NTK analysis is that as $M \to \infty$, the NTK, $\kappa_{\boldsymbol{\theta}}$ converges in probability to an explicit deterministic limit $\kappa_{\boldsymbol{\theta}}^{\infty}$ (Jacot et al., 2018, Theorem 1) where:

$$\kappa_{\boldsymbol{\theta}}^{\infty} \doteq \mathop{\mathbb{E}}_{\boldsymbol{\theta} \sim \mathcal{N}(0, \boldsymbol{I})} [\mathbf{G}]. \tag{15}$$

This deterministic limit only depends on the variance parameter of the Gaussian initialization distribution and the choice of non-linearity. The NTK limit holds for ReLU activations (Arora et al., 2019) and LeCun initialization (Jacot et al., 2018).

B.2. Linearized ReLU networks

We now instantiate the NTK framework for the exact experimental model used in our continual learning experiments: a linearized ReLU network (Dwaraknath et al., 2023). Let the input dimension be I, width be M, and output dimension be O. Denote the first-layer weights with $\mathbf{W}^{[1]} = [\mathbf{w}_1^{[1]}, \ldots, \mathbf{w}_M^{[1]}] \in \mathbb{R}^{I\times M}$ and the fixed output-layer weights with $\mathbf{W}^{[2]} = [\mathbf{w}_1^{[2]}, \ldots, \mathbf{w}_M^{[2]}]^\top \in \mathbb{R}^{M\times O}$ (so each hidden unit h has output row $(\mathbf{w}_h^{[2]})^\top \in \mathbb{R}^O$). A standard linearized ReLU network is:

$$\hat{F}^{\boldsymbol{\theta}}(\boldsymbol{x}) \doteq \frac{1}{\sqrt{M}} \sum_{h=1}^{M} \mathbf{w}_h^{[2]}\, \sigma((\mathbf{w}_h^{[1]})^\top \boldsymbol{x}), \qquad \sigma(x) = \max\{0, x\}. \tag{16}$$

Linearized ReLU Network Let $\mathbf{W}^{[1](0)} \in \mathbb{R}^{I\times M}$ be a frozen reference copy of the initialized first-layer matrix $\mathbf{W}^{[1]}$. With this fixed initialization, we define the fixed ReLU gate function and gate matrix respectively:

$$g(\boldsymbol{x}) \doteq \mathbf{1}\{(\mathbf{w}_h^{[1](0)})^\top \boldsymbol{x} > 0\} \text{ for } h \in 1, .., M \qquad \mathbf{D}_g(\boldsymbol{x}) \doteq \mathrm{diag}(g(\boldsymbol{x})). \tag{17}$$

Note that the gates $g(\boldsymbol{x})$ (equivalently, $\mathbf{D}_g(\boldsymbol{x})$) and the output matrix $\mathbf{W}^{[2]}$ remain frozen throughout training. The only trainable parameters are $\mathbf{W}^{[1]} \in \mathbb{R}^{d\times M}$, initialized at $\mathbf{W}^{[1]} = \mathbf{W}^{[1](0)}$. Thus, the linearized activations are:

$$F_{\mathbf{W}^{[1]}}(\boldsymbol{x}) \doteq \sigma((\mathbf{W}^{[1](0)})^\top \boldsymbol{x}) \;+\; g(\boldsymbol{x}) \odot ((\mathbf{W}^{[1]})^\top \boldsymbol{x} - (\mathbf{W}^{[1](0)})^\top \boldsymbol{x}) \tag{18}$$

$$= g(\boldsymbol{x}) \odot ((\mathbf{W}^{[1]})^\top \boldsymbol{x}) \tag{19}$$

$$= \mathbf{D}_g(\boldsymbol{x})\, (\mathbf{W}^{[1]})^\top \boldsymbol{x}, \tag{20}$$

where we used $\sigma((\mathbf{W}^{[1](0)})^\top \boldsymbol{x}) = g(\boldsymbol{x}) \odot (\mathbf{W}^{[1](0)})^\top \boldsymbol{x}$. The full linearized network the model prediction is

$$F_{\boldsymbol{\theta}}(\boldsymbol{x}) = F_{\mathbf{W}^{[1]}}(\boldsymbol{x})^\top \mathbf{W}^{[2]} = \mathbf{W}^{[2]} \mathbf{D}_g(\boldsymbol{x}) (\mathbf{W}^{[1]})^\top \boldsymbol{x} \in \mathbb{R}^O. \tag{21}$$

We denote this specific instantiation of the gated-linear tangent kernel (GLTK) with $\chi : \boldsymbol{x} \times \boldsymbol{x} \to \mathbb{R}^{O\times O}$. Specifically, for output coordinates $o, o' \in [O]$:

$$\chi(\boldsymbol{x}, \boldsymbol{x}')_{(o,o')} \;\doteq\; \Big\langle \frac{\partial F_o(\boldsymbol{x})}{\partial \mathbf{W}^{[1]}}, \frac{\partial F_{o'}(\boldsymbol{x}')}{\partial \mathbf{W}^{[1]}} \Big\rangle_{\mathrm{F}}, \tag{22}$$

Given the dataset $\{\boldsymbol{x}^1, .., \boldsymbol{x}^n\}$, using $\partial F_o(\boldsymbol{x})/\partial \mathbf{w}_h^{[1]} = [\mathbf{w}_h^{[2]}]_o\, g_h(\boldsymbol{x})\, \boldsymbol{x}$, yields the Gram matrix:

$$[\chi(\boldsymbol{x}^n, \boldsymbol{x}^m)]_{o,o'} \doteq [\mathbf{G}]_{(n,o),(m,o')} \doteq = \; \langle \boldsymbol{x}^n, \boldsymbol{x}^m \rangle \frac{1}{M} \sum_{h=1}^{M} \Big([\mathbf{w}_h^{[2]}]_o [\mathbf{w}_h^{[2]}]_{o'}\, g_h(\boldsymbol{x}^n)\, g_h(\boldsymbol{x}^m) \Big). \tag{23}$$

Furthermore, the taking $M \to \infty$, the infinite width limit is:

$$[\chi_\infty]_{o,o'}(\boldsymbol{x}_n, \boldsymbol{x}_m) \doteq \mathbf{E}_{\mathbf{W}^{[1](0)}, \mathbf{W}^{[2]}}[\mathbf{G}] = \mathbf{E}_{\mathbf{W}^{[1](0)}, \mathbf{W}^{[2]}} \Big[\langle \boldsymbol{x}^n, \boldsymbol{x}^m \rangle \sum_{h=1}^{M} [\mathbf{w}_h^{[2]}]_o [\mathbf{w}_h^{[2]}]_{o'}\, g_h(\boldsymbol{x}^n)\, g_h(\boldsymbol{x}^m) \Big]. \tag{24}$$

Relation to NTK linearization. Consider the full two-layer network in Equation (16) with frozen V and linearize in W around $W^{(0)}$:

$$\hat{F}_{\boldsymbol{\theta}}(\boldsymbol{X}_\tau) \approx F_{\boldsymbol{\theta}_\tau^{(0)}}(\boldsymbol{X}_\tau) + \mathbf{J}_\tau(\boldsymbol{\theta} - \boldsymbol{\theta}_\tau^{(0)}).$$

In particular, for output coordinate $o \in [O]$.

$$\frac{\partial F_{\boldsymbol{\theta}}(\boldsymbol{x})}{\partial \mathbf{W}^{[1]}} = \frac{1}{\sqrt{M}} \boldsymbol{x} (g \odot [\mathbf{W}^{[2]}]_{:,o})^\top. \tag{25}$$

In our construction Equation (20)–Equation (21), this linearization is *exact globally*: once gates are frozen, $F_{\boldsymbol{\theta}}(\boldsymbol{x})$ is linear in $\mathbf{W}^{[1]}$ for all $\mathbf{W}^{[1]}$, not just locally around $\mathbf{W}^{[1](0)}$. Therefore, Assumption C.1 holds with equality for this experimental model.

C. NTK Mathematical Results

Our detailed mathmatical results rely on the following assumptions:

Assumption C.1 (Local Linearization)**.** For task τ, during the next K gradient descent steps starting at $\boldsymbol{\theta}_\tau^{(0)}$, we approximate:

$$\hat{F}_{\boldsymbol{\theta}}(\boldsymbol{X}_\tau) \approx F_{\boldsymbol{\theta}_\tau^{(0)}}(\boldsymbol{X}_\tau) + \mathbf{J}_\tau(\boldsymbol{\theta} - \boldsymbol{\theta}_\tau^{(0)}), \tag{26}$$

where the Jacobian evaluated on the dataset/batch is:

$$\mathbf{J} \doteq \frac{\partial F_{\boldsymbol{\theta}}(\boldsymbol{X}_\tau)}{\partial \boldsymbol{\theta}}\bigg|_{\boldsymbol{\theta}=\boldsymbol{\theta}_\tau^{(0)}} \in \mathbb{R}^{N\times P}, \tag{27}$$

is treated as constant over the window.

For the continual NTK, $\mathbf{G} = \mathbf{J}\mathbf{J}^\top \in \mathbb{R}^{N\times N}$, we will linearize around $\boldsymbol{\theta}_0 = \boldsymbol{\theta}_{\tau+1}^{(0)}$.

Assumption C.2 (Step size stability)**.** The learning rate η is small enough such that $\boldsymbol{I} - \eta\mathbf{G}_\tau$ is non-expansive (i.e., $\eta\lambda_{\max}(\mathbf{G}_\tau) \leq 1$).

In addition, we also assume that our model linear model is initialized from i.i.d. standard Gaussian distributions as an NTK.

Lemma 4.2 (Residual dynamics in the linearized regime)**.** *If $\hat{F}_{\boldsymbol{\theta}}(\boldsymbol{X}_\tau) \approx F_{\boldsymbol{\theta}_\tau^{(0)}}(\boldsymbol{X}_\tau) + \mathbf{J}_\tau(\boldsymbol{\theta} - \boldsymbol{\theta}_\tau^{(0)})$, then gradient descent on $\ell_\tau(\boldsymbol{\theta})$ with learning rate η yields as the $k+1$th iterate $r_\tau^{(k+1)} = (\boldsymbol{I} - \eta\mathbf{G}_\tau)r_\tau^{(k)}$, where $\mathbf{G}_\tau = \mathbf{J}_\tau\mathbf{J}_\tau^\top \in \mathbb{R}^{NO\times NO}$. Hence, the final value of the residual for task τ is $r_\tau^{(K)} = (\boldsymbol{I} - \eta\mathbf{G}_\tau)^K r_\tau^{(0)}$.*

Proof. Using *Assumption C.1*, $\hat{F}_{\boldsymbol{\theta}}(\boldsymbol{X}_\tau) \approx F_{\boldsymbol{\theta}_\tau^{(0)}}(\boldsymbol{X}_\tau) + \mathbf{J}_\tau(\boldsymbol{\theta} - \boldsymbol{\theta}_\tau^{(0)})$, so we can approximate the residual with the following:

$$\begin{aligned} r_\tau(\boldsymbol{\theta}) &= F_{\boldsymbol{\theta}}(\boldsymbol{X}_\tau) - \boldsymbol{Y}_\tau && (28)\\ &\approx F_{\boldsymbol{\theta}_\tau^{(0)}}(\boldsymbol{X}_\tau) - \boldsymbol{Y}_\tau + \mathbf{J}(\boldsymbol{\theta} - \boldsymbol{\theta}_\tau^{(0)}) && (29)\\ &= r_\tau^{(0)} + \mathbf{J}(\boldsymbol{\theta} - \boldsymbol{\theta}_\tau^{(0)}). && (30)\end{aligned}$$

For the squared error loss $\ell = \frac{1}{2}\|r(\boldsymbol{\theta})\|^2$, with linear model $r(\boldsymbol{\theta}) = r_\tau^{(0)} + \mathbf{J}(\boldsymbol{\theta} - \boldsymbol{\theta}_\tau^{(0)})$ the gradient is:

$$\nabla_{\boldsymbol{\theta}}\ell_\tau(\boldsymbol{\theta}) = \mathbf{J}_\tau^\top r(\boldsymbol{\theta}), \tag{31}$$

which means the gradient descent update is:

$$\boldsymbol{\theta}_\tau^{(k+1)} = \boldsymbol{\theta}_\tau^{(k)} - \eta\mathbf{J}_\tau^\top r_\tau^{(k)}. \tag{32}$$

Now using the fact that $r_\tau^{(k+1)} \approx r_\tau^{(k)} + \mathbf{J}_\tau(\boldsymbol{\theta}^{(k+1)} - \boldsymbol{\theta}^{(k)})$, we obtain :

$$\begin{aligned} r_\tau^{(k+1)} &\approx r_\tau^{(k)} + \mathbf{J}_\tau(\boldsymbol{\theta}^{(k+1)} - \boldsymbol{\theta}^{(k)}) && (33)\\ &= r_\tau^{(k)} + \mathbf{J}_\tau(-\eta\mathbf{J}_\tau^\top r_\tau^{(k)}) && (34)\\ &= (\boldsymbol{I} - \eta\mathbf{J}_\tau\mathbf{J}_\tau^\top)r_\tau^{(k)} && (35)\\ &= (\boldsymbol{I} - \eta\mathbf{G}_\tau)r_\tau^{(k)} && (36)\end{aligned}$$

Which we can repeatedly apply using *assumption C.1* to obtain the final result. □

Lemma 4.3 (Eigenvalue Decomposition)**.** *Assume* $\mathbf{G}_\tau$ *has eigenpairs* $\{\lambda_{\tau,q}, \mathbf{v}_{\tau,q}\}_{q=1}^N$ *with orthonormal* $\{\mathbf{v}_{\tau,q}\}$ *and* $\lambda_{\tau,q} \geq 0$*. Expand the initial residual:*

$$r_\tau^{(0)} = \sum_{q=1}^{n} \alpha_{\tau,q}\mathbf{v}_{\tau,q}, \quad \alpha_{\tau,q} \doteq (\mathbf{v}_{\tau,q})^\top r_\tau^{(0)}. \tag{4}$$

Then, for all iterates k, $r_\tau^{(k)} = \sum_{q=1}^{n}(1-\eta\lambda_{\tau,q})^k \alpha_{\tau,q}\mathbf{v}_{\tau,q}$.

Proof. First, we expand $r_\tau^{(0)}$ in its eigenbasis:

$$r_\tau^{(0)} = \sum_{i=1}^{n} \alpha_{\tau,q}\mathbf{v}_{\tau,q}, \quad \alpha_{\tau,q} \doteq (\mathbf{v}_{\tau,q})^\top r_\tau^{0}, \tag{37}$$

and

$$r_\tau^{(k)} = (\boldsymbol{I} - \eta\mathbf{G}_\tau)^k \left(\sum_{q=1}^{n} \alpha_{\tau,q}\mathbf{v}_{\tau,q} \right) \tag{38}$$

Using the identity of eigenpairs: $\mathbf{G}_\tau \mathbf{v}_{\tau,q} = \lambda_{\tau,q}\mathbf{v}_{\tau,q}$ which implies that:

$$\begin{aligned}(\boldsymbol{I} - \eta\mathbf{G}_\tau)\mathbf{v}_{\tau,q} &= \boldsymbol{I}\mathbf{v}_{\tau,q} - \eta(\mathbf{G}_\tau \mathbf{v}_{\tau,q}) &(39)\\ &= 1 \cdot \mathbf{v}_{\tau,q} - \eta(\lambda_{\tau,q}\mathbf{v}_{\tau,q}) &(40)\\ &= (1-\eta\lambda_{\tau,q})\mathbf{v}_{\tau,q}. &(41)\end{aligned}$$

For any power k

$$(\boldsymbol{I} - \eta\mathbf{G}_\tau)^k \mathbf{v}_{\tau,q} = (1-\eta\lambda_{\tau,q})^k \mathbf{v}_{\tau,q}, \tag{42}$$

since $\lambda_{\tau,q}$ is an eigenvector of $\mathbf{G}_\tau$, it is also an eigenvalue of the polynomial $(\boldsymbol{I} - \eta\mathbf{G}_\tau)$, and by induction of $(\boldsymbol{I} - \eta\mathbf{G}_\tau)^k$ with eigenvalue $(\boldsymbol{I} - \eta\lambda_{\tau,q})^k$.

We can now plugging this back into $eq.$ (38):

$$r_\tau^{(k)} = \sum_{q=1}^{n} (1-\eta\lambda_{\tau,q})^k \alpha_{\tau,q}\mathbf{v}_{\tau,q}. \tag{43}$$

and squaring yields the desired result. □

C.1. Fast and slow modes

We can now formalize *fast* and *slow* with an explicit threshold derived from K and η. Pick $\gamma \in (0,1)$.

Definition C.3. A *fast mode* should shrink by at least a factor of γ after K steps. That is: if a mode is fast $(1-\eta\lambda)K \leq \gamma$.

Solve for λ:

$$(1-\eta\lambda)^K \leq \gamma \iff 1-\eta\lambda \leq \gamma^{1/K} \iff \lambda \geq \epsilon_{K,\gamma}, \tag{44}$$

where

$$\epsilon_{K,\gamma} \doteq \frac{1-\gamma^{1/K}}{\eta} \approx \frac{-\log\gamma}{\eta K}. \tag{45}$$

Given eigenvalues $\lambda_\tau^{\boldsymbol{i}}$, define the fast set:

$$\mathcal{F}_\tau \doteq \{q : \lambda_{\tau,q} \geq \epsilon_{K,\gamma}\}, \tag{46}$$

and slow set:

$$\mathcal{S}_\tau \doteq \{q : \lambda_{\tau,q} < \epsilon_{K,\gamma}\}. \tag{47}$$

The ϵ-rank can now be defined as $|\mathcal{F}|$.

Finally, we define the fast projector:

$$\mathbf{P}_\tau^{\mathcal{F}} \doteq \sum_{i \in \mathcal{F}_\tau} \mathbf{v}_{\tau,q} \mathbf{v}_{\tau,q}^\top, \tag{48}$$

and slow projector:

$$\mathbf{P}_\tau^{\mathcal{S}} \doteq \boldsymbol{I} - \mathbf{P}_\tau^{\mathcal{F}} = \sum_{i \in \mathcal{S}_\tau} \mathbf{v}_{\tau,q} \mathbf{v}_{\tau,q}^\top. \tag{49}$$

Lemma 4.6. *For any task τ,*

$$\left\| r_\tau^{(K)} \right\| \geq \left\| \mathbf{P}_\tau^{\mathcal{S}} r_\tau^{(K)} \right\| \geq \gamma \left\| \mathbf{P}_\tau^{\mathcal{S}} r_\tau^{(0)} \right\|. \tag{5}$$

Proof. From Lemma 4.3,

$$\mathbf{P}^{\mathcal{S}} = \sum_{\boldsymbol{i} \in \mathcal{S}} (1 - \eta\lambda_i)^K \alpha^i \mathbf{v}^i. \tag{50}$$

If $i \in \mathcal{S}_\tau$, then $\lambda^i < \epsilon_{K,\gamma}$, hence

$$1 - \eta\lambda^i > 1 - \eta\epsilon_{K,\gamma} = \gamma^{1/K} \implies (1 - \eta\lambda^i)^K > \gamma. \tag{51}$$

Therefore, each slow coefficient satisfies $|(1 - \eta\lambda^i)^K \alpha^i \geq \gamma |\alpha^i|$. Squaring and summing gives:

$$\|\mathbf{P}_\tau^{\mathcal{S}} r^{(K)}\|^2 = \sum_{i \in \mathcal{S}} (1 - \eta\lambda^i)^{2K} (\alpha^i)^2 \geq \sum_{i \in \mathcal{S}} \gamma^2 (\alpha^i)^2 = \gamma^2 \|\mathbf{P}^{\mathcal{S}} r^{(0)}\|^2. \tag{52}$$

Taking square roots closes the proof. □

Remark C.4. Slow modes are sticky. If the eigenvalues are in the slow-regime, after K steps, the NTK still retains at least γ fraction of the eigenvalues less than $\epsilon_{K,\gamma}$.

C.2. Continual Case

Now we will connect slow-stickiness to the success criterion. For the squared loss on task $\tau + 1$,

$$\ell_{\tau+1}(\boldsymbol{\theta}_{\tau+1}^{(K)}) = \frac{1}{2} \|r_{\tau+1}^{(K)}\|^2. \tag{53}$$

.

Now, we can define a success criterion if the final loss is at most υ:

$$\text{Success}_{\tau+1} \doteq \left\{ \ell_\tau(\boldsymbol{\theta}_{\tau+1}^{(K)}) \leq \upsilon \right\} = \left\{ \|r_{\tau+1}^{(K)}\| \leq \sqrt{2\upsilon} \right\}. \tag{54}$$

Lemma 4.7 (Necessary fast-projection condition)**.** *Using the fast and slow projections, $\epsilon_{K,\gamma}$, and the initial loss residual $r_\tau^{(0)}$, a necessary condition to achieve successful training on task τ is $\gamma \left\| \mathbf{P}_\tau^{\mathcal{S}} r_\tau^{(0)} \right\| \leq \sqrt{2\upsilon}$. Equivalently,*

$$\left\| \mathbf{P}_\tau^{\mathcal{F}} \frac{r_\tau^{(0)}}{\|r_\tau^{(0)}\|} \right\|^2 \geq 1 - \left(\frac{\sqrt{2\upsilon}}{\left\| \gamma r_\tau^{(0)} \right\|^2} \right) \tag{6}$$

Proof. From Lemma 4.6 applied to task $\tau + 1$:

$$\|r_{\tau+1}^{(K)}\| \geq \gamma \|\mathbf{P}_{\tau+1}^{\mathcal{S}} r_{\tau+1}^{(0)}\| \tag{55}$$

If success holds, then $\|\mathbf{P}_{\tau+1}^{\mathcal{S}} r_\tau^{(0)}\| \leq \sqrt{2\upsilon}$, so combining:

$$\sqrt{2\upsilon} \geq \|r_{\tau+1}^{(K)}\| \geq \gamma \|\mathbf{P}_{\tau+1}^{\mathcal{S}} r_{\tau+1}^{(0)}\| \implies \|\mathbf{P}_{\tau+1}^{\mathcal{S}} r_{\tau+1}^{(0)}\| \leq \frac{\sqrt{2\upsilon}}{\gamma}. \tag{56}$$

Now define $V = \frac{r_{\tau+1}^{(0)}}{\|r_{\tau+1}^{(0)}\|_2}$,

$$\|\mathbf{P}_{\tau+1}^{(\mathcal{S})} r_{\tau+1}^{(0)}\|^2 = \|r_{\tau+1}^{(0)}\|^2 \cdot \|\mathbf{P}_{\tau+1}^{(\mathcal{S})} V\|^2 \leq \left(\frac{\sqrt{2\upsilon}}{\gamma} \right)^2 \implies \|\mathbf{P}_{\tau+1}^{(\mathcal{S})} V\|^2 \leq \left(\frac{\sqrt{2\upsilon}}{\gamma \|r_{\tau+1}^{(0)}\|^2} \right)^2. \tag{57}$$

Finally, since $\mathbf{P}_{\tau+1}^{\mathcal{F}} = \boldsymbol{I} - \mathbf{P}_\tau^{\mathcal{S}}$,

$$\|\mathbf{P}_{\tau+1}^{\mathcal{F}} V\|^2 = 1 - \|\mathbf{P}_{\tau+1}^{\mathcal{S}} V\|^2 \geq 1 - \left(\frac{\sqrt{2\upsilon}}{\gamma \|r_{\tau+1}^{(0)}\|^2} \right)^2 \tag{58}$$

□

Remark C.5. To succeed on the new task, the eigenvalues of of the initial Gram matrix must lie overwhelmingly in the fast subspace, unless the initial residual magnitude $\|r_{\tau+1}^{(0)}\|^2$ is already tiny. This means that if the model is successful on the current task, and the new task is similar, we can expect successful training. However, if the model eigenvalues are small and so lie in the slow-subspace, it will not be able to converge to success in K gradient steps.

Lemma C.6 (Permutation invariance of the gated-linear Gram)**.** *Consider a linearized ReLU network with weights from Equation (16) initialized with i.i.d Gaussian entries as NTK. Let the input dimension be I, width be M, and output dimension be O. Given inputs indices n, m, (i.e., $\boldsymbol{x}_n, \boldsymbol{x}_m \in \{\boldsymbol{x}^1, .., \boldsymbol{x}^n\}$), and output indices o, o', the GLTK $\chi_\infty = \mathbb{E}_{\mathbf{W}^{[1](0)}, \mathbf{W}^{[2]}}[\mathbf{G}]$ is given by:*

$$[\chi^\infty]_{o,o'}(\boldsymbol{x}_n, \boldsymbol{x}_m) = \mathbf{E}_{\mathbf{W}^{[1](0)}, \mathbf{W}^{[2]}} \Big[\langle \boldsymbol{x}_n, \boldsymbol{x}_m \rangle \sum_{h=1}^{M} [\mathbf{w}_h^{[2]}]_o [\mathbf{w}_h^{[2]}]_{o'} \, g_h(\boldsymbol{x}_n) \, g_h(\boldsymbol{x}_m) \Big]. \tag{59}$$

Let $\boldsymbol{\Pi} \in \mathbb{R}^{I \times I}$ be a orthogonal permutation matrix [5] (i.e., $\boldsymbol{\Pi}^\top \boldsymbol{\Pi} = \boldsymbol{I}$). Then, the infinite width kernel $[\chi]_{(o,o')}[\infty](\boldsymbol{x}_n, \boldsymbol{x}_m)$ is invariant to the transformation $\mathbf{P}$ applied to it's inputs. Specifically for any pair of data $\boldsymbol{x}_n, \boldsymbol{x}_m$ and output indices o, o':

$$[\kappa^\infty]_{o,o'}(\boldsymbol{\Pi} \boldsymbol{x}_n, \boldsymbol{\Pi} \boldsymbol{x}_m) = [\kappa]_{o,o'}[\infty](\boldsymbol{x}_n, \boldsymbol{x}_m). \tag{60}$$

Proof. First, note that inner products are invariant under orthogonal transforms: $\langle \boldsymbol{\Pi} \boldsymbol{x}, \boldsymbol{\Pi} \boldsymbol{x}' \rangle = \langle \boldsymbol{x}, \boldsymbol{x}' \rangle$. Next, for each hidden unit h,

$$g_h(\boldsymbol{\Pi} \boldsymbol{x}) \;=\; \mathbf{1}\{(\mathbf{w}_h^{[1](0)})^\top \boldsymbol{\Pi} \boldsymbol{x} > 0\} \;=\; \mathbf{1}\{(\boldsymbol{\Pi}^\top \mathbf{w}_h^{[1](0)})^\top \boldsymbol{x} > 0\}. \tag{61}$$

[5] A matrix with exactly one entry of 1 in each row and column, but not diagonal.

Since $\mathbf{w}_h^{[1](0)}$ is initialized from an isotropic distribution and $\mathbf{\Pi}$ is orthogonal, we have $\mathbf{\Pi}^\top \mathbf{w}_h^{[1](0)} \equiv \mathbf{w}_h^{[1](0)}$. Therefore, for any fixed $\boldsymbol{x}, \boldsymbol{x}'$, the pair $(g_h(\mathbf{\Pi}\boldsymbol{x}), g_h(\mathbf{\Pi}\boldsymbol{x}'))$ has the same distribution as $(g_h(\boldsymbol{x}), g_h(\boldsymbol{x}'))$, and in particular $\mathbf{E}[g_h(\mathbf{\Pi}\boldsymbol{x})g_h(\mathbf{\Pi}\boldsymbol{x}')] = \mathbf{E}[g_h(\boldsymbol{x})g_h(\boldsymbol{x}')]$, where the expectation is over $\mathbf{w}_h^{[1](0)}$.

Finally, using the explicit kernel form (23) and independence of V from $\mathbf{W}^{[1](0)}$,

$$\mathbf{E}_{\mathbf{W}^{[1](0)},\mathbf{W}^{[2]}}\Big[[\chi]_{o,o'}(\mathbf{\Pi}\boldsymbol{x}_n, \mathbf{\Pi}\boldsymbol{x}_m)\Big] = \mathbf{E}_{\mathbf{W}^{[1](0)},\mathbf{W}^{[2]}}\Big[\langle \mathbf{\Pi}\boldsymbol{x}_n, \mathbf{\Pi}\boldsymbol{x}_m\rangle \sum_{h=1}^{M} [\mathbf{w}_h^{[2]}]_o [\mathbf{w}_h^{[2]}]_{o'}\, g_h(\mathbf{\Pi}\boldsymbol{x}_n)\, g_h(\mathbf{\Pi}\boldsymbol{x}_m)\Big] \tag{62}$$

$$= \langle \boldsymbol{x}_n, \boldsymbol{x}_m\rangle \sum_{h=1}^{M} \mathbf{E}_{\mathbf{W}^{[2]}}[[\mathbf{w}_h^{[2]}]_o [\mathbf{w}_h^{[2]}]_{o'}]\; \mathbf{E}_{\mathbf{W}^{[1](0)}}[g_h(\mathbf{\Pi}\boldsymbol{x}_n) g_h(\mathbf{\Pi}\boldsymbol{x}_m)] \tag{63}$$

$$= \langle \boldsymbol{x}_n, \boldsymbol{x}_m\rangle \sum_{h=1}^{M} \mathbf{E}_{\mathbf{W}^{[2]}}[[\mathbf{w}_h^{[2]}]_o [\mathbf{w}_h^{[2]}]_{o'}]\; \mathbf{E}_{\mathbf{W}^{[2]}}[g_h(\boldsymbol{x}_n) g_h(\boldsymbol{x}_m)] \tag{64}$$

$$= \mathbf{E}_{\mathbf{W}^{[1](0)},\mathbf{W}^{[2]}}\Big[[\chi]_{(o,o')}(\boldsymbol{x}_n, \boldsymbol{x}_m)\Big], \tag{65}$$

which proves the claim. □

Remark C.7. For permuted MNIST, each task applies some permutation $\mathbf{\Pi}_\tau$ to all inputs. Lemma C.6 implies the *expected* frozen-gates Gram (and hence its expected spectrum / ϵ-rank) is task-invariant under fresh pixel permutations, up to finite-width fluctuations.

C.3. Extension to Multi-class Cross-Entropy

For multi-class cross-entropy with logits $\boldsymbol{z} = (\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n) \in \mathbb{R}^{nO}$ and blockwise softmax $\mathrm{softmax}(\boldsymbol{z})$, the empirical risk $\ell(\boldsymbol{z}) = \sum_{i=1}^n \ell(\boldsymbol{z}_i, \boldsymbol{y}_i)$ satisfies $\nabla_{\boldsymbol{z}} \ell(\boldsymbol{z}) = \mathrm{softmax}(\boldsymbol{z}) - \boldsymbol{Y}$.

In the NTK/linearized regime (Jacobian frozen at initialization), discrete gradient descent on parameters induces a closed-form update on training logits of the form:

$$\boldsymbol{z}^{(k+1)} = \boldsymbol{z}^{(k)} - \eta \mathbf{G}^{(0)} \nabla_{\boldsymbol{z}} \ell(\boldsymbol{z}^{(k)}), \tag{66}$$

$$\mathbf{G}^{(0)} = \mathbf{J}^{(0)} (\mathbf{J}^{(0)})^\top, \tag{67}$$

where $\mathbf{J}_0$ is the initialization-time feature map induced by the training inputs and $\mathbf{G}^{(0)} \in \mathbb{R}^{(nO)\times(nO)}$ is the corresponding multi-output NTK/Gram matrix with blocks $\mathbf{G}^{(0)}_{(i,o),(j,o')} = \langle \nabla_{\boldsymbol{\theta}} F_o(\boldsymbol{x}_i), \nabla_{\boldsymbol{\theta}} F_{o'}(\boldsymbol{x}_j)\rangle$. This is exactly the discrete-time counterpart of the standard NTK function-space dynamics for general losses (Lee et al., 2019). To recover the same fast/slow eigenmode story as in squared loss, we linearize the cross-entropy gradient in logit space around a reference $\boldsymbol{z}^\star$:

$$\nabla_{\boldsymbol{z}} \ell(\boldsymbol{z}^\star + \delta\boldsymbol{z}) = \nabla_{\boldsymbol{z}} \ell(\boldsymbol{z}^\star) + \mathbf{H}^\star \delta\boldsymbol{z} + \text{h.o.t.}, \tag{68}$$

where $\mathbf{C} = \nabla^2_{\boldsymbol{z}} \ell(\boldsymbol{z}^\star) = \mathrm{blockdiag}(\mathbf{C}_1, \ldots, \mathbf{C}_n)$ and $\mathbf{C}_i = \mathrm{diag}(\mathbf{p}_i) - \mathbf{p}_i \mathbf{p}_i^\top \succeq 0$ with $\mathbf{p}_i = \mathrm{softmax}(\boldsymbol{z}^i_\star)$ (Wu et al., 2022). Then,

$$\boldsymbol{z}^{(k+1)} = \boldsymbol{z}^{(*)} + \delta\boldsymbol{z}^{(k)} - \eta \mathbf{G}^{(0)} \nabla_{\boldsymbol{z}} \ell(\boldsymbol{z}^{(*)} + \delta\boldsymbol{z}^{(k)}) \tag{69}$$

$$\approx \boldsymbol{z}^{(*)} + \delta\boldsymbol{z}^{(k)} - \eta \mathbf{G}^{(0)} \nabla_{\boldsymbol{z}} \ell(\boldsymbol{z}^{(*)} + \mathbf{C}^* \delta\boldsymbol{z}^{(k)}), \tag{70}$$

which implies

$$\delta\boldsymbol{z}^{(k+1)} \approx (\boldsymbol{I} - \eta \mathbf{G}^{(0)} \mathbf{C}) \delta\boldsymbol{z}^{(k)} - \eta \mathbf{G}^{(0)} \nabla_{\boldsymbol{z}} \ell(\boldsymbol{z}^*). \tag{71}$$

If $\boldsymbol{z}^*$ is a stationary point, so that $\nabla_{\boldsymbol{z}} \ell(\boldsymbol{z}^*) = 0$, the additive term vanishes; otherwise, the local linearization is affine with respect to the linear same linear part.

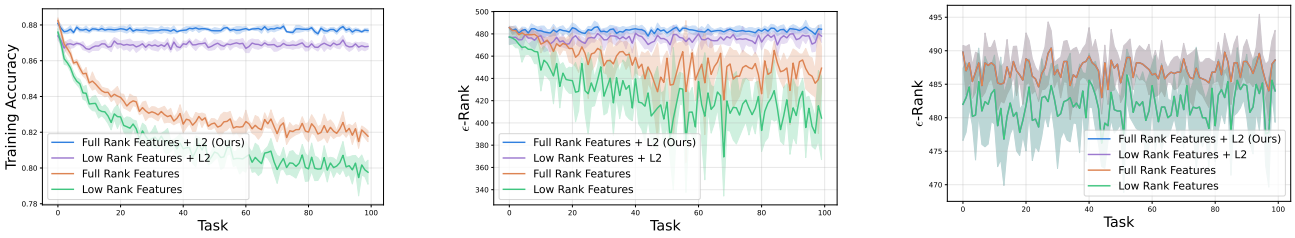
Define the weighted error $\mathbf{e}^{(k)} \doteq \mathbf{C}^{1/2} \delta\boldsymbol{z}^{(k)}$, so that the local cross-entropy quadratic satisfies $\delta\boldsymbol{z}^\top \mathbf{C} \delta\boldsymbol{z} = \|\mathbf{e}\|_2^2$; in these coordinates the linearized update becomes

$$\mathbf{e}^{(k+1)} \approx (\boldsymbol{I} - \eta \mathbf{C}^{1/2} \mathbf{G}^{(0)} \mathbf{C}^{1/2}) \mathbf{e}^{(k)}. \tag{72}$$

Hence, the relevant "fast eigenvalues" for cross-entropy are those of the GGN $\mathbf{G}^{\star}_{\mathrm{LW}} \doteq \mathbf{C}^{1/2}\mathbf{G}^{(0)}\mathbf{C}^{1/2}$, not the raw $\mathbf{G}^{(0)}$. By Lemma C.6 the infinite width limit of $\mathbf{G}^{(0)}$ remains task-invariant.

This matches the generalized Gauss–Newton viewpoint: in the linearized model, the parameter-space curvature is $\mathbf{J}^{\top(0)}\mathbf{C}\mathbf{J}^{(0)}$, whose nonzero spectrum agrees with $\mathbf{C}^{1/2}\mathbf{J}^{(0)}(\mathbf{J}^{(0)})\top\mathbf{C}^{1/2}$ (Martens & Grosse, 2015).

C.4. Empirical Results



(a) Training Accuracy *(b)* $\epsilon_{K,\gamma}$-rank of $\mathbf{G}_{CE}$ *(c)* $\epsilon_{K,\gamma}$-rank of $\mathbf{G}^{(0)}$

Figure 5. Metics tracked in our linearized ReLU network experiment. As expected, the training accuracy correlates with the $\epsilon_{K,\gamma}$-rank of the $\mathbf{G}_{CE}$, while the $\epsilon_{K,\gamma}$-rank of $\mathbf{G}^{(0)}$ remains fairly flat on average over training.

C.5. Proofs For Section 6

Theorem C.8. *Suppose an L-layered MLP $F_{\boldsymbol{\theta}} : \mathbb{R}^I \to \mathbb{R}^O$, and all input data lies in $\mathcal{B}_r(0)$, and either means-squared error or softmax-cross-entropy losses ℓ. Let the network be $\boldsymbol{x} \mapsto F_{\boldsymbol{\theta}}(\boldsymbol{x})$ be L_f-Lipschitz on $\mathcal{B}_r(0)$ and let the logit-gradient, $\boldsymbol{z} \mapsto \nabla_{\boldsymbol{z}}\ell(\boldsymbol{z}, \boldsymbol{y})$ be L_g-Lipschitz. Define: $\Gamma(\boldsymbol{\theta}) := \sup_{\boldsymbol{x}\in\mathcal{B}_r(0), 1\le i\le O} \|\nabla^2_{\boldsymbol{\theta}} \boldsymbol{z}_i(\boldsymbol{x};\boldsymbol{\theta})\|_2$ and $\boldsymbol{z} = F_{\boldsymbol{\theta}}(\boldsymbol{x})$ Then, the residual $\mathbf{R}$ of the Hessian on task $\tau+1$ obeys the following operator-norm bound:*

$$\|\mathbf{R}(\boldsymbol{\theta}^0_{\tau+1})\|_2 \le \sqrt{O}\Gamma(\boldsymbol{\theta}_\tau)\left(\sup_{\boldsymbol{x}\in\mathrm{supp}(\mathcal{X}_\tau)} [\nabla_{\boldsymbol{z}}\ell(\boldsymbol{z}_i, \boldsymbol{y})] + L_f L_g d(\mathcal{X}, \mathcal{X}')\right) \doteq \boldsymbol{B}_\tau. \tag{73}$$

Proof. We start with the definition of the residual term of the Hessian:

$$\mathbf{R}(\boldsymbol{\theta}^0_{\tau+1}) = \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\mathcal{X}_{\tau+1}}\left[\sum_i^O \nabla_{\boldsymbol{z}}\ell(\boldsymbol{z}_i, \boldsymbol{y})\nabla^2_{\boldsymbol{\theta}}\boldsymbol{z}_i(\boldsymbol{x};\boldsymbol{\theta}_\tau)\right] \quad . \tag{74}$$

Next we apply linearity and operator norms:

$$\|\mathbf{R}(\boldsymbol{\theta}^0_{\tau+1})\|_2 = \left\| \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\mathcal{X}_{\tau+1}}\left[\sum_i^O \nabla_{\boldsymbol{z}}\ell(\boldsymbol{z}_i, \boldsymbol{y})\nabla^2_{\boldsymbol{\theta}}\boldsymbol{z}_i(\boldsymbol{x};\boldsymbol{\theta}_\tau)\right]\right\| \tag{75}$$

$$\le \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\mathcal{X}_{\tau+1}} \sum_i^O \|\nabla_{\boldsymbol{z}}\ell(\boldsymbol{z}_i, \boldsymbol{y})\|_2 \|\nabla^2_{\boldsymbol{\theta}}\boldsymbol{z}_i(\boldsymbol{x};\boldsymbol{\theta}_\tau)\|_2 \tag{76}$$

and apply a deterministic worst-case bound followed followed by the Cauchy-Schwartz inequality to obtain:

$$\|\mathbf{R}(\boldsymbol{\theta}^0_{\tau+1})\|_2 \le \sup_{\boldsymbol{x}\in\mathrm{supp}(\mathcal{X}')} \|\nabla_{\boldsymbol{z}}\ell(\boldsymbol{z}_i, \boldsymbol{y})\|_2 \sqrt{O}\Gamma(\boldsymbol{\theta}) \tag{77}$$

Let π be a coupling measure on $\mathcal{X}$ and $\mathcal{X}'$, and denote $h(\boldsymbol{x};\boldsymbol{\theta}) = \nabla_{\boldsymbol{z}}\ell(F_{\boldsymbol{\theta}}(\boldsymbol{x}), \boldsymbol{y})$. First operating inside the supp, we apply the triangle inequality:

$$\|\nabla_{\boldsymbol{z}} h(\boldsymbol{x};\boldsymbol{\theta})\|_2 \le \|h(\boldsymbol{x}';\boldsymbol{\theta})\|_2 + \|h(\boldsymbol{x};\boldsymbol{\theta}) - h(\boldsymbol{x}';\boldsymbol{\theta})\|_2 \tag{78}$$

$$\le \|h(\boldsymbol{x}';\boldsymbol{\theta})\|_2 + L_f L_g d(\mathcal{X}, \mathcal{X}') \quad \text{Applying Lipschitz assumptions.} \tag{79}$$

Now, operating inside the supp w.r.t. π we obtain:

$$\sup_{\boldsymbol{x}\in\mathrm{supp}(\mathcal{X})} \|h(\boldsymbol{x};\boldsymbol{\theta})\|_2 = \sup_{(\boldsymbol{x},\boldsymbol{x}')\in\mathrm{supp}(\pi)} \|h(\boldsymbol{x}';\boldsymbol{\theta})\|_2 \le \sup_{\boldsymbol{x}'\in\mathrm{supp}(\mathcal{X}')} \|h(\boldsymbol{x}';\boldsymbol{\theta})\|_2 + L_f L_g d(\mathcal{X}, \mathcal{X}') \tag{80}$$

Finally, bounding the RHS of the inequality from above by an arbitrary constant $\boldsymbol{B}$ yields the desired result. □

Corollary C.9. *We can use Weyl's inequality to show the existence of an appropriate $L2$ regularizer to guarantee that increasing rank($\mathbf{H}_{GGN}$) implies increasing the rank($\mathbf{H}$). Suppose ordered eigenvalues of $\mathbf{H}_{GGN}$ (i.e., $\lambda_1(\cdot) \ge ...\lambda_n(\cdot)$), then by applying Wely's inequality we obtain*

$$|\lambda_i(\mathbf{H}_{GGN} + \mathbf{R}-)\lambda_i(\mathbf{R})| \le \|\mathbf{R}\|_2 \tag{81}$$

$$\lambda_i(\mathbf{H}_{GGN} + \mathbf{R}) \ge \lambda_i(\mathbf{H}_{GGN}) - \|\mathbf{R}\|_2 \tag{82}$$

$$\lambda_i(\mathbf{H}) \ge \lambda_i(\mathbf{H}_{GGN}) - \|\mathbf{R}\|_2 \tag{83}$$

$$\lambda^+_{\min}(\mathbf{H}) \ge \lambda^+_{\min}(\mathbf{H}_{GGN}) - \|\mathbf{R}\|_2. \tag{84}$$

The $L2$-regularized Hessian becomes:

$$\lambda^+_{\min}(\mathbf{H} + 2\boldsymbol{I}q) \ge \lambda^+_{\min}(\mathbf{H}_{GGN}) - \|\mathbf{R}\|_2 + 2q \tag{85}$$

$$\ge \lambda^+_{\min}(\mathbf{H}_{GGN}) - \|\boldsymbol{B}\|_2 + 2q, \tag{86}$$

so if $2q > \|\mathbf{R}\|_2 - \lambda^+_{\min}(\mathbf{H}_{GGN})$, then every positive eigenvalue of $\mathbf{H}_{GGN}$ remains positive in the q-regularized $\mathbf{H}$.

Remark C.10. Theorem C.8 and Corollary C.9 justify why $L2$ is needed alongside ER regularization. Under softmax-cross-entropy loss and mean-squared-error loss with bounded targets, $h(\boldsymbol{x}; \boldsymbol{\theta})$ is also bounded which ensures the theorem is well-posed. The theorem also requires $\Gamma(\boldsymbol{\theta})$ to be uniformly bounded which can be achieved by techniques like spectral regularization (although we do use it in our experiments) (Yoshida & Miyato, 2017). In practice, we find that small choices of $L2$ regularization are needed which we report in table 2, table 4, and table 6.

D. KFAC Error

For a given layer, $\mathbf{W}^{[l]} \in \mathbb{R}^{d_{in} \times d_{out}}$, with activations into the layer $\mathbf{a}^{[l]} \in \mathbb{R}^{d_{in}}$ and gradient outputs $\boldsymbol{\delta}^{[l]} \in \mathbb{R}^{d_{out}}$. The respective covariances are $\mathbf{a}^{[l]}\mathbf{a}^{[l]\top} \in \mathbb{R}^{d_{in} \times d_{in}}$ and $\boldsymbol{\delta}^{[l]}\boldsymbol{\delta}^{[l]\top} \in \mathbb{R}^{d_{out} \times d_{out}}$. The resulting KFAC block is of size $(d_{out}d_{in}) \times (d_{out}d_{in})$. The per-entry error, on the nth sample, can be obtained by indexing over Kronecker matrix pairs with row-index (i, α) and column index (j, β). We use $\cdot^{[l]}_{(i,*)}$ to denote the ith row of $(\cdot)$ in layer l, and $\cdot^{[l]}_{(*,j)}$ for the jth column. Flattening over the Kronecker product, we obtain the following for a given entry in layer l:

$$(\mathbf{H}_{GGN})^{[l]}_{([i,\alpha],[j,\beta])} = \frac{1}{N}\sum_{n=1}^{N}\sum_{o=1}^{O}[\mathbf{a}^{[l]}_n]_{(i,*)}[\mathbf{a}^{[l]}_n]_{(*,j)}[\boldsymbol{\delta}^{[l]}_{n,o}]_{(\alpha,*)}[\boldsymbol{\delta}^{[l]}_{n,o}]_{(*,\beta)} = \overline{[\mathbf{a}^{[l]}]_{(i,*)}[\mathbf{a}^{[l]}]_{(*,j)}[\boldsymbol{\delta}^{[l]}]_{(\alpha,*)}[\boldsymbol{\delta}^{[l]}]_{(*,\beta)}} \tag{87}$$

$$(\widehat{\mathbf{H}}_{GGN})^{[l]}_{([i,\alpha],[j,\beta])} = \left(\frac{1}{N}\sum_{m=1}^{N}[\mathbf{a}^{[l]}_n]_{(i,*)}[\mathbf{a}^{[l]}_n]_{(*,j)}\right) \cdot \left(\frac{1}{N}\sum_{n=1}^{N}\sum_{o=1}^{O}[\boldsymbol{\delta}^{[l]}_{m,o}]_{(\alpha,*)}[\boldsymbol{\delta}^{[l]}_{m,o}]_{(*,\beta)}\right) = \overline{[\mathbf{a}^{[l]}]_{(i,*)}[\mathbf{a}^{[l]}]_{(*,j)}} \cdot \overline{[\boldsymbol{\delta}^{[l]}]_{(\alpha,*)}[\boldsymbol{\delta}^{[l]}]_{(*,\beta)}}, \tag{88}$$

where we use $\bar{\cdot}$ to denote the empirical sample average. In this way, we can compute the error as the difference between the average of products and the product of averages; for a fixed index quadruple (i, j, α, β), the KFAC error of the $((i, \alpha), (j, \beta))$th-entry is:

$$\Delta_{[i,\alpha],[j,\beta]} = (\mathbf{H}_{GGN} - \widehat{\mathbf{H}}_{GGN})_{[i,\alpha],[j,\beta]} = \overline{[\mathbf{a}^{[l]}]_i[\mathbf{a}^{[l]}]_j[\boldsymbol{\delta}^{[l]}]_\alpha[\boldsymbol{\delta}^{[l]}]_\beta} - \overline{[\mathbf{a}^{[l]}]_i[\mathbf{a}^{[l]}]_j} \cdot \overline{[\boldsymbol{\delta}^{[l]}]_\alpha[\boldsymbol{\delta}^{[l]}]_\beta} \tag{89}$$

E. Algorithm

E.1. Baselines

Backpropagation (BP): We optimize the objective with stochastic gradient descent (SGD) on cross-entropy loss over the current task. Let θ denote all trainable parameters and $\mathcal{D}_t$ the data for task τ. The objective is

$$\min_{\theta} \ \mathbb{E}_{(x,y)\sim\mathcal{D}_t}[L(F_\theta(x), y)].$$

Vanilla Effective Rank (ER): In addition to BP, we collect the output features of each dense layer (excluding convolutional layers) over a fixed number of steps (er-batch in Table 1). From these stacked features, we compute the effective rank (Roy & Vetterli, 2007) for each layer and sum across all layers. Then we maximize the effective rank of the network representations.

$$\mathcal{L}_{\mathrm{ER}} = -\frac{1}{L}\sum_{\ell=1}^{L} \mathrm{ER}(\mathbf{h}^{[l]}), \qquad \mathrm{ER}(\mathbf{h}^{[l]}) = \exp\Big(-\sum_{i=1}^{d_\ell} p_i^{[\ell]} \log p_i^{[\ell]}\Big),$$

where $\mathbf{h}^{[l]} \in \mathbb{R}^{N\times d_\ell}$ is the stacked feature matrix for layer ℓ, $s_i(\mathbf{h}^{[l]})$ are its singular values, and

$$p_i^{[\ell]} = \frac{s_i(\mathbf{h}^{[l]})}{\sum_j s_j(\mathbf{h}^{[l]})}.$$

L2 normalization (L2): We add weight decay to BP. Now the objective becomes:

$$\min_{\theta} \ \mathbb{E}_{(x,y)\sim\mathcal{D}_t}[L(F_\theta(x), y)] \ + \ \lambda_w \|\theta\|_2^2,$$

where λ_w is the weight-decay coefficient (weight-decay in Table 1).
Continual Backpropagation (CBP): We follow the architecture in (Dohare et al., 2024).
Layer-normalization (LayerNorm-L2): We follow the architecture in (Lyle et al., 2024).
Spectral: We follow the architecture in (Lewandowski et al., 2024a).

E.2. Effective Rank with $L2$ normalization ($L2$-ER)

The pseudocode for our implementation of $L2$-ER is shown in Algorithm 1. $L2$-ER augments the standard task objective with two additional regularizers: (1) an effective rank penalty and (2) an $L2$ weight decay term. At each step, the task loss L_{task} together with the weight decay term is minimized via standard backpropagation. Simultaneously, the hidden features $\mathbf{h}^{[l]}$ are collected over a fixed window L of updates. Every L steps, these stacked features are used to compute the effective rank (Roy & Vetterli, 2007) of the representation at each layer. The effective rank losses are averaged across layers. Note that a gradient descent step is taken on this loss only every L steps.

Algorithm 1 Continual Learning with EffectiveRank Regularization ($L2$-ER)

Input: Task datasets $\{\{(\boldsymbol{x}_{\tau,i}, \boldsymbol{y}_{\tau,i})\}_{i=0}^{N-1}\}_{\tau=0}^{T-1}$; model $F_{\boldsymbol{\theta}}$; learning rates α (task), β (ER); weight decay λ; ER update interval U (in steps)
State: Per-layer feature buffers $\{\mathcal{B}^\ell\}_{\ell=1}^{L}$ with capacity U
for $\tau = 0$ **to** $T-1$ **do**
  for $i = 0, \ldots, N-1$ **do**
    $(\hat{\boldsymbol{y}}, \{\mathbf{h}^{[l]}\}_{\ell=1}^{L}) \leftarrow F_{\boldsymbol{\theta}}(\boldsymbol{x}_{\tau,i})$ {$\hat{\boldsymbol{y}}$ is the model prediction}
    $\forall \ell:\ \mathcal{B}^\ell \leftarrow \text{enqueue}(\mathcal{B}^\ell, \mathbf{h}^{[l]});$ if $|\mathcal{B}^\ell| > U$ then drop oldest
    $L_{\text{task}} \leftarrow \text{Loss}(\hat{\boldsymbol{y}}, \boldsymbol{y}_{\tau,i}) + \lambda\|\boldsymbol{\theta}\|_2^2$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha\nabla_{\boldsymbol{\theta}} L_{\text{task}}$
    if $(i+1) \bmod U = 0$ **then**
      $L_{\text{erank}} \leftarrow -\frac{1}{L}\sum_{\ell=1}^{L} \text{ER}(\text{SVD}(\text{Stack}(\mathcal{B}^\ell)))$ {(Roy & Vetterli, 2007)}
      $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \beta\nabla_{\boldsymbol{\theta}} L_{\text{erank}}$
      $\mathcal{B}^\ell \leftarrow \emptyset$
    end if
  end for
end for
Return: $\boldsymbol{\theta}$

F. Dead Neurons and Effective Rank

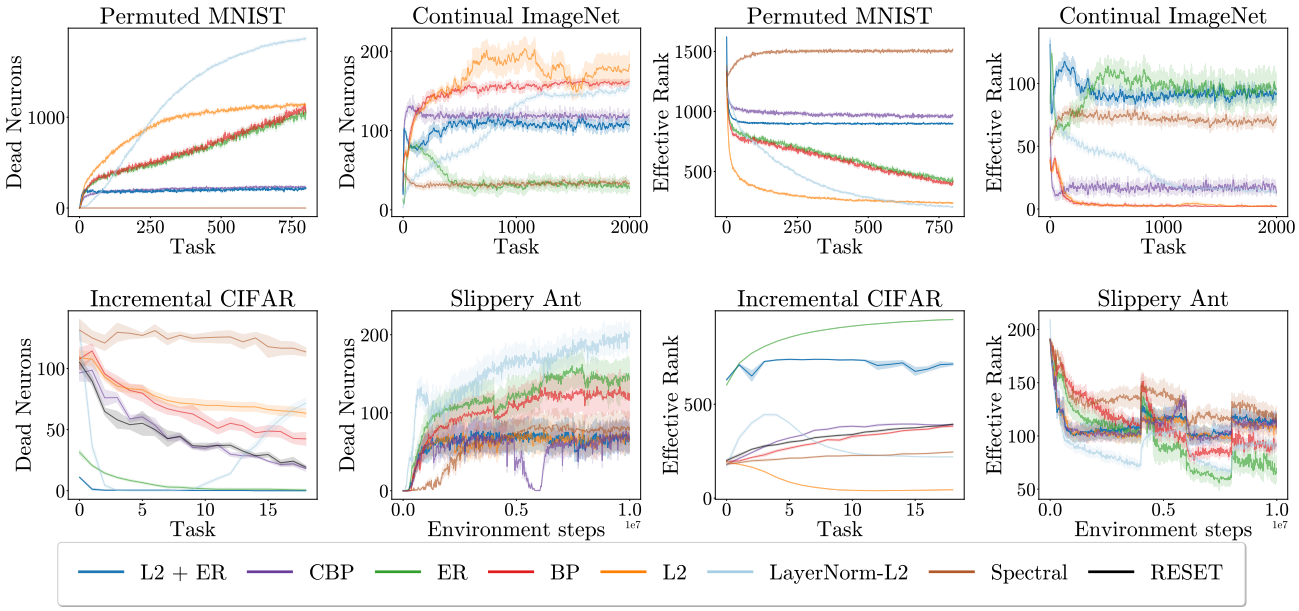


Figure 6. Number of dead neurons (left) and effective rank (right) corresponding to Figure 4.

Here, we present number of dead neurons and effective rank corresponding to Figure 4. An important observation to note here is that dead neurons in Incremental Cifar is decreasing rather than increasing. This effect arises due to the following two reasons: First, at the beginning of training, the environment contains only 5 classes, which gradually increase to 100. As more classes are introduced, the evaluation set becomes much larger, increasing the likelihood of encountering samples that activate a given neuron (i.e., produce nonzero outputs). Second, to accommodate the changing number of classes, we mask the outputs of the final layer to match the number of active classes in each task. In the early tasks, this masking leads to a sharp rise in the number of dead neurons, since the network tends to overfit to the small set of available classes. As more classes are added, this effect diminishes. Therefore, we also measured the number of dead neurons and effective rank in RESET for comparison. Learning curves that lie above RESET indicate an increase in dead neurons due to loss of plasticity, whereas curves below suggest relatively fewer dead neurons.

G. Epsilon Hessian Rank

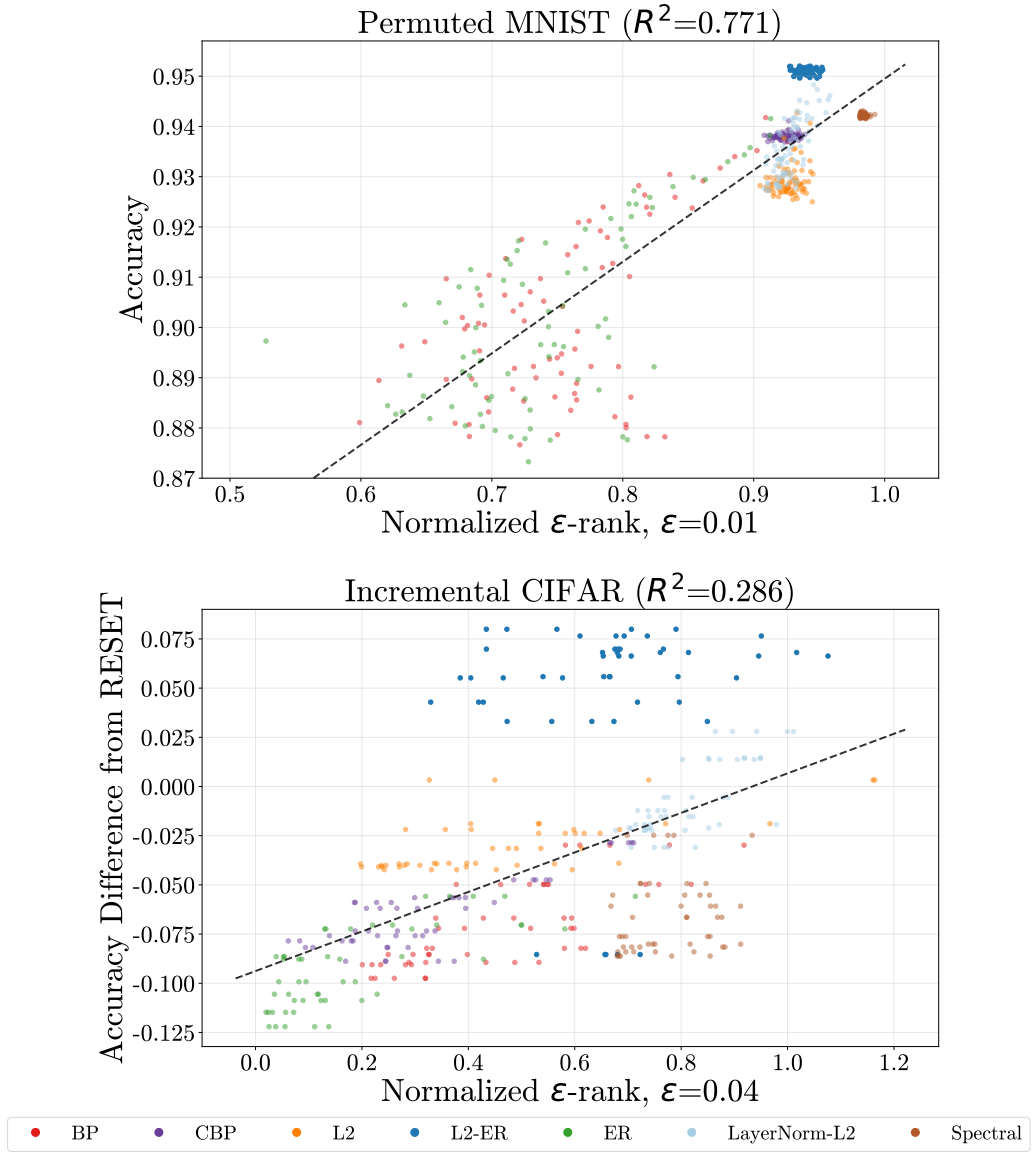


Figure 7. Curvature vs. Accuracy on Permuted MNIST (Left) and First 10 Task Incremental CIFAR (Right). A linear fit (dotted line) highlights the positive association between curvature and accuracy.

G.1. Measuring the Hessian Spectral Density

To empirically demonstrate the relationship between dead ReLUs and the rank of the Hessian, we estimate the Hessian spectrum over continual learning tasks (Ghorbani et al., 2019). The spectral density is defined as $\psi(u) = \frac{1}{P}\sum_{i=1}^{P}\delta(u-\lambda_i)$ where δ is a Dirac delta operator. Since the naive approach to estimating the density would involve calculating every eigenvalue, we turn to a Gaussian relaxation (Ghorbani et al., 2019):

$$\psi_\sigma(u) = \frac{1}{P}\sum_{i=1}^{P} f(\lambda_i; u, \sigma^2)$$

where $f(\lambda_i; u, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}}\exp(-\frac{(u=\lambda)^2}{2\sigma^2})$ and σ^2 is the variance. When σ^2 is small, ψ_σ provides a tight estimate of the spectral density. Since materializing the full Hessian is prohibitively expensive, we estimate the density with the

stochastic Lanczos quadrature algorithm (Golub & Welsch, 1969; Ghorbani et al., 2019), which takes advantage of the fact that accessing Hessian-vector-products (HVP) is a computationally efficient operation (Pearlmutter, 1994). Given $\mathbf{H}$ is diagonalized and f has a closed-form equation, we can define $f(\mathbf{H}) = \mathbf{Q}f(\mathbf{\Lambda})\mathbf{Q}^\top$ where $f(\mathbf{\Lambda})$ acts on the diagonal. Now we estimate the spectrum of Hessian through the HVP with $\mathbf{v} \sim N(0, \frac{1}{P}\boldsymbol{I}_{P\times P})$ which gives:

$$\psi_\sigma = \frac{1}{P}\text{tr}\Big(f(\mathbf{H}, u, \sigma^2)\Big) = \mathbb{E}[\mathbf{v}^\top f(\mathbf{H}, u, \sigma^2)\mathbf{v}]$$

In practice, each $\psi_\sigma^{\mathbf{v}}$ is approximated by m-steps of the Lanczos algorithm resulting in a $m \times m$ tridiagonal matrix with m locations-weight pairs (ℓ_j, ω_j) so that:

$$\psi_\sigma^{\mathbf{v}}(u) \approx \sum_{j=1}^{m} \omega_j f(\ell_j; u, \sigma^2) \doteq \hat{\psi}^{\mathbf{v}}(u)$$

Moreover, $\psi_\sigma^{\mathbf{v}}(\cdot)$ is an unbiased estimator of $\psi_\sigma(\cdot)$ and $\hat{\psi}^{\mathbf{v}}(u)$ converges exponentially fast around its expectation over samples of $\mathbf{v}$ (Ghorbani et al., 2019)[Claim 2.3] resulting in a computationally efficient and accurate estimation of the spectral density, even for large neural networks.

Throughout our experiments, we measure the Hessian eigen-spectrum at the beginning and at the end of each new task. Our results show that the spectrum of standard back-propagation narrows as the task number grows. In fact, a complete loss of learning corresponds to a complete collapse of the spectrum. Furthermore, we show that loss of plasticity mitigation like continual backpropagation and our own $L2$-ER method preserve the spectrum.

In the following sections, we present the details of each environment, their corresponding hyperparameter sweeps and selected best hyperparameters, followed by the analysis of the Hessian spectrum.

G.2. Epsilon Hessian Rank vs Accuracy

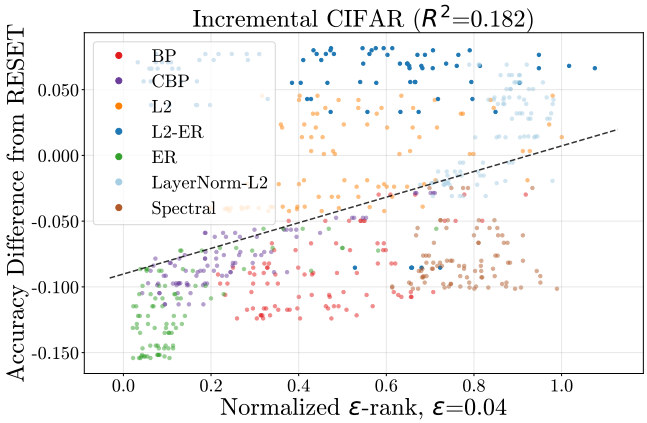


Figure 8. Curvature vs. Accuracy on Full 20 Task Incremental CIFAR. A linear fit (dotted line) highlights the positive association between curvature and accuracy.

In addition to Figure 3, we provide the epsilon rank of the other two supervised learning environments in Figure 7. Continual ImageNet and Permuted MNIST's data points on the graph are calculated by an average across 5 seeds.

Putting all the Incremental Cifar tasks into one plot does not yield as clear a positive relationship as Permuted MNIST or Continual Imagenet. All the tasks in Permuted MNIST or Continual Imagenet are about the same level of difficulties for $\tau \in \{1, \cdots, T\}$, which gives us the ability to easily measure successful training through task accuracies. In Incremental Cifar, each task τ is composed of classifying $5 * \tau$ classes of images using the same computational budget, which makes it a challenge to isolate unsuccessful training due to loss of plasticity from increasing task difficulties. We report the performance difference between algorithms and a freshly initialized network as a measure of successful training. Furthermore,

neural network tends to find a lower-rank solution regardless of initialization (Hu et al., 2022). Since we always report the ϵ-Hessian rank of the Full databatch, the percentage of data the neural network has already been trained on becomes larger and larger as τ grows. For example, at task 2 initialization, we trained on 5 classes and acquired a low rank representation, then 5 classes are added and the ϵ-Hessian rank is high; while at task 20 initialization, we trained on 19 classes and are evaluating on 20, this is basically the same as the eigenspectrum at task 19 convergence, which is a low rank solution. In short, due to the non-uniform property of the tasks, ϵ-Hessian rank is not an accurate measure of Spectral Collapse in Incremental Cifar. When we group the first ten tasks (fig. 7), we can see that the positive relationship is clearer since the task difficulties are more similar.

H. Permuted MNIST

We now detail environments and architectural details in all experiments. All algorithms are written in JAX (Bradbury et al., 2018). Although each environment is independent, our training procedure is designed to be easily generalizable. In the following sections, we provide detailed descriptions of each environment.

Permuted MNIST (Dohare et al., 2024) is a variant of the original MNIST dataset (LeCun, 1998) where the pixels are permuted randomly in the same way for each image in the original dataset. Each permutation defines a new task for the learner. In total, we evaluate on 800 tasks, each of which is a 10-class classification problem.

H.1. Network Architecture

We employ a standard multilayer perceptron (MLP) with three hidden layers of hidden size 1000 each followed by a ReLU activation. All weights are initialized with Kaiming uniform. The architecture can be summarized as follows:

```
MLP(
  Sequential(
    (0): Dense(init=kaiming_uniform, out_dims=1000, bias=True)
    (1): ReLU()
    (2): Dense(init=kaiming_uniform, out_dims=1000, bias=True)
    (3): ReLU()
    (4): Dense(init=kaiming_uniform, out_dims=1000, bias=True)
    (5): ReLU()
    (6): Dense(init=kaiming_uniform, out_dims=10, bias=True)
  )
)
```

H.2. Hyperparameters

In Table 1, we present the default hyperparam of our experiments. Unless otherwise specified, experiments use the default hyperparameters.

H.3. Experiments

Prior work (Dohare et al., 2024), along with our own experiments, shows that the learning rate is a critical factor in continual learning: using a smaller learning rate consistently yields only marginal differences in performance. To better highlight the phenomenon of loss of plasticity, we fix the learning rate to 1×10^{-2} and sweep over the remaining hyperparameters in Table 2. We report the results in Figure 6 and the selected best hyperparameters in Table 2.

H.4. Permuted MNIST Hessian Spectrum

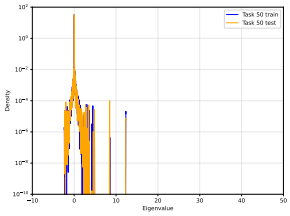
We use the best hyperparameters in Table 2 and rerun it with 5 seeds to plot the hessian spectrum during our training. We calculate the approximated hessian matrix every fixed number of tasks (compute-hessian-interval in Table 1). The corresponding performance is shown in Figure 4 and Hessian spectrum of seed 2025 is shown in Figure 9, where the orange curve corresponds to the Hessian spectrum on the test set and the blue curve corresponds to the training set. Since plotting all tasks is impractical, we present only a subset of representative tasks for BP, L2, ER, CBP, L2-ER.

Comparing Figure 4 with Figure 9, we observe that algorithms which eventually lose plasticity exhibit a sparse Hessian spectrum. In contrast, algorithms that maintain plasticity preserve a rich and dense spectrum, proving our claim that spectral collapse is strongly correlated with the loss of plasticity. Furthermore, even subtle reductions in plasticity are reflected in the spectrum: for instance, $L2$ shows a slight decline of about 1% in accuracy over training. This small change is captured by the eigenspectrum, as shown in Figure 9, where the range of eigenvalues for $L2$ is narrower than that of the other two algorithms that preserve plasticity.

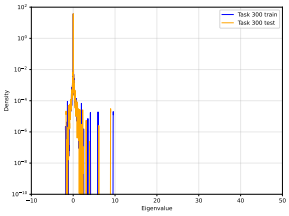
Table 1. permuted MNIST default hyperparameters

Hyperparam Name	Default	Description
study_name	test	experiment name
seed	2024	base random seed
debug	False	true to enable debug mode
platform	gpu	{cpu, gpu}
n_seeds	1	number of seeds
env	permuted_mnist	environment name
agent	l2_er	{er, bp, l2, cbp, l2_er, layernorm_l2, spectral}
activation	relu	activation options: {relu, tanh}
lr	[0.01]	learning rate(s)
optimizer	sgd	{adam, sgd}
weight_decay	0.001	$L2$ weight decay
num_features	1000	hidden size for the mlp
change_after	10×6000	steps between task switches
to_perturb	False	whether to perturb the input data
perturb_scale	1×10^{-5}	magnitude of input perturbation
num_hidden_layers	3	number of hidden layers in the mlp
mini_batch_size	1	minibatch size
no_anneal_lr	True	if true, do not anneal the learning rate
max_grad_norm	0.5	gradient clipping threshold
num_tasks	800	number of tasks used in training/eval
effective rank		
er_lr	[0.01]	ER learning rate
er_batch	100	batch size for er computation
er_step	1	ER update frequency
evaluation		
evaluate	True	evaluate after each task
evaluate_previous	False	evaluate on previous task
eval_size	2000	number of evaluate samples per task
compute_hessian	False	whether to compute hessian spectrum
compute_hessian_size	2000	samples used for hessian computation
compute_hessian_interval	1	interval in tasks between hessian runs
continual backpropagation		
cont_backprop	False	enable CBP
replacement_rate	1×10^{-6}	CBP replacement probability per step
decay_rate	0.99	exponential decay for CBP statistics
maturity_threshold	100	steps before a unit is considered mature
Spectral Regularizer		
k	2	power used in $(\sigma_{\max}^{k} - \texttt{target})^2$
target	2.0	target value for the largest singular value
spectral_strength	0.1	coefficient multiplying the spectral penalty
num_iter	10	number of power-iteration steps to estimate $\sigma_{\max}$

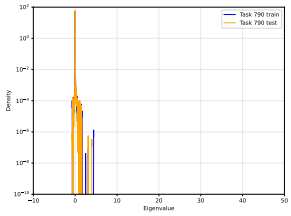
Algorithms that *lose plasticity*



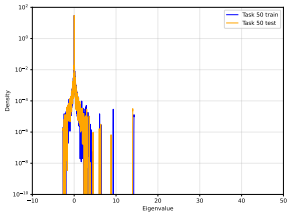
(a) ER: task 50



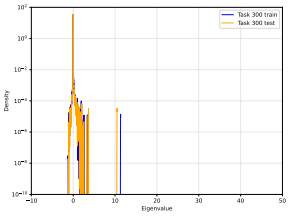
(b) ER: task 300



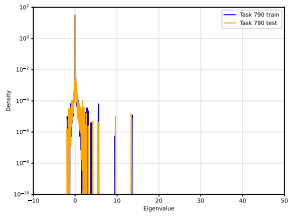
(c) ER: task 790



(d) BP: task 50

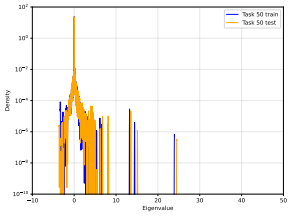


(e) BP: task 300

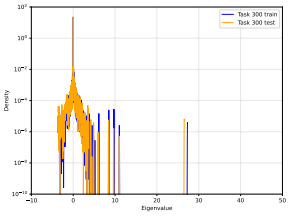


(f) BP: task 790

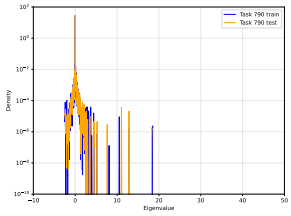
Algorithms that *preserve plasticity*



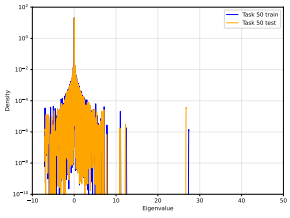
(g) $L2$: task 50



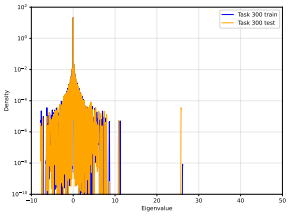
(h) $L2$: task 300



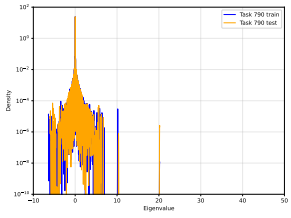
(i) $L2$: task 790



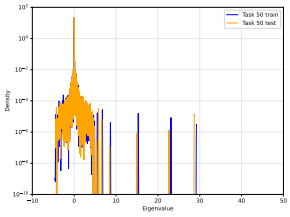
(j) CBP: task 50



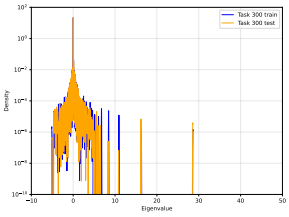
(k) CBP: task 300



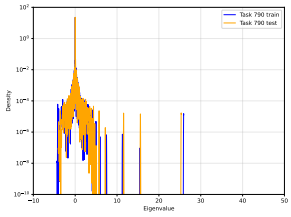
(l) CBP: task 790



(m) $L2$-ER: task 50



(n) $L2$-ER: task 300



(o) $L2$-ER: task 790

Figure 9. Comparison of Hessian eigenspectra at **task init** across permuted MNIST. From top to bottom, the algorithms are ordered by increasing accuracy. **Top:** Algorithms that lose plasticity (ER, BP). **Bottom:** Algorithms that preserve plasticity ($L2$, CBP, $L2$-ER).

Table 2. Hyperparameter sweeps and best values for Permuted MNIST across all algorithms under a fixed learning rate of 1×10^{-2}.

Algorithm	**Hyperparameter**	**Sweep Hyperparameters**	**Best**
BP	Learning rate	$\{1 \times 10^{-2}\}$	1×10^{-2}
ER	Learning rate	$\{1 \times 10^{-2}\}$	1×10^{-2}
	Effective rank lr	$\{1 \times 10^{-2}, 1 \times 10^{-3}, 1 \times 10^{-4}\}$	1×10^{-3}
$L2$-ER	Learning rate	$\{1 \times 10^{-2}\}$	1×10^{-2}
	Effective rank lr	$\{1 \times 10^{-2}, 1 \times 10^{-3}, 1 \times 10^{-4}\}$	1×10^{-3}
	Weight decay	$\{1 \times 10^{-3}, 1 \times 10^{-4}, 1 \times 10^{-5}\}$	1×10^{-3}
CBP	Learning rate	$\{1 \times 10^{-2}\}$	1×10^{-2}
	Replacement rate	$\{1 \times 10^{-4}, 1 \times 10^{-5}, 1 \times 10^{-6}\}$	1×10^{-6}
$L2$	Learning rate	$\{1 \times 10^{-2}\}$	1×10^{-2}
	Weight decay	$\{1 \times 10^{-3}, 1 \times 10^{-4}, 1 \times 10^{-5}\}$	1×10^{-3}
LayerNorm-L2	Learning rate	$\{1 \times 10^{-2}\}$	1×10^{-2}
	Weight decay	$\{1 \times 10^{-2}, 1 \times 10^{-3}, 1 \times 10^{-4}\}$	1×10^{-5}
Spectral Regularizer	Learning rate	$\{1 \times 10^{-2}\}$	1×10^{-2}
	Spectral Strength	$\{1 \times 10^{-2}, 1 \times 10^{-3}, 1 \times 10^{-4}\}$	1×10^{-3}

I. Continual ImageNet

Continual ImageNet (Dohare et al., 2024) is an adaptation of ImageNet (Krizhevsky et al., 2012) in which pairs of classes are randomly sampled to form binary classification tasks. We evaluate performance on a total of 2000 tasks.

I.1. Network Architecture

We adopt the same architecture as (Dohare et al., 2024) with one key modification. In their setup, the final layer of the network is reinitialized at the beginning of every task. To ensure fairness in comparison, we remove this feature and keep the final layer fixed across tasks. We use a threeblock convolutional network followed by two dense layers and a dense classifier. Each convolution is followed by a ReLU nonlinearity and a 2×2 maxpool with stride 2. The architecture is summarized as follows:

```
ConvNet(
  Sequential(
    (0): Conv2d(out_channels=32, kernel_size=5x5, bias=True)
    (1): ReLU()
    (2): MaxPool(window=2x2, stride=2)

    (3): Conv2d(out_channels=64, kernel_size=3x3, bias=True)
    (4): ReLU()
    (5): MaxPool(window=2x2, stride=2)

    (6): Conv2d(out_channels=128, kernel_size=3x3, bias=True)
    (7): ReLU()
    (8): MaxPool(window=2x2, stride=2)
    (9): Flatten()

    (10): Dense(out_dims=128, bias=True)
    (11): ReLU()
    (12): Dense(out_dims=128, bias=True)
    (13): ReLU()
    (14): Dense(out_dims=2, bias=True)
  )
)
```

I.2. Hyperparameters

In Table 3, we present the default hyperparameters for our ImageNet experiments. Unless otherwise specified, experiments use these defaults.

I.3. Experiments

Due to high variance, we conducted full hyperparameter sweeps from Table 4 across 10 seeds and apply Savitzky-Golay filter (Savitzky & Golay, 1964) to the results. The best hyperparameters selected from these sweeps are summarized in Table 4.

I.4. Continual ImageNet Hessian Spectrum

We again use the best hyperparameters from Table 4 to run over 10 seeds to plot the hessian spectrum. We presents our results on BP, ER, L2-ER, CBP, L2 in Figure 11 and its corresponding performance in Figure 4. Note that we categorize ER as preserving plasticity in this case because, in this single run, ER successfully maintained plasticity rather than losing it. We present this single run hessian spectrum accuracy in Figure 10. From Figure 11, we observe that all algorithms that lose plasticity experience spectral collapse, whereas those that preserve plasticity maintain a wide and dense spectrum throughout training.

Table 3. ImageNet default hyperparameters

Hyperparam Name	Default	Description
study_name	test	experiment name
seed	2024	base random seed
debug	False	true to enable debug mode
platform	gpu	{cpu, gpu}
n_seeds	1	number of seeds
env	imagenet	environment name
agent	l2_er	agent options: {er, bp, l2, snp_l2, snp, cbp, l2_er, laynorm_l2, spectral_reg}
alg	ppo	algorithm type: {actor_critic, ppo}
activation	relu	activation options: {relu, tanh}
lr	[0.01]	learning rate(s)
optimizer	sgd	{adam, sgd}
weight_decay	0.001	$L2$ weight decay
to_perturb	False	whether to perturb the input data
perturb_scale	1×10^{-5}	magnitude of input perturbation
mini_batch_size	100	minibatch size
no_anneal_lr	True	if true, do not anneal the learning rate
max_grad_norm	1×10^{9}	gradient clipping threshold
num_tasks	2000	number of tasks used in training/eval
num_epochs	20	number of epochs per task
momentum	0.9	SGD momentum coefficient
effective rank		
er_lr	[0.01]	ER learning rate(s)
er_batch	12	batch size for ER computation
er_step	1	ER update frequency
model architecture		
use_layernorm	False	enable LayerNorm in the model
spectral regularization		
use_spectral_reg	False	enable spectral regularization
spectral_reg_strength	0.1	regularization strength
spectral_k	2	number of singular values/eigs (or rank) used
spectral_target	2.0	target value for spectral objective
spectral_power_iter	10	power-iteration steps (if used)
evaluation		
evaluate	True	evaluate after each task
evaluate_previous	False	also evaluate on previous tasks
eval_size	2000	number of evaluation samples per task
compute_hessian	False	whether to compute Hessian spectrum
compute_hessian_size	2000	samples used for Hessian computation
compute_hessian_interval	1	interval (in tasks) between Hessian runs
continual backpropagation		
cont_backprop	False	enable CBP
replacement_rate	1×10^{-6}	CBP replacement probability per step
decay_rate	0.99	exponential decay for CBP statistics
maturity_threshold	100	steps before a unit is considered mature

Table 4. Hyperparameter sweeps and selected best values for Continual ImageNet across all algorithms.

Algorithm	**Hyperparameter**	**Sweep Hyperparameters**	**Best Hyperparam**
BP	Learning rate	10^{-2}, 10^{-3}, 10^{-4}	10^{-4}
ER	Learning rate	10^{-2}, 10^{-3}, 10^{-4}	10^{-4}
	Effective rank lr	10^{-3}, 10^{-4}, 10^{-5}	10^{-5}
$L2$-ER	Learning rate	10^{-3}	10^{-3}
	Effective rank lr	10^{-3}, 10^{-4}, 10^{-5}	10^{-4}
	Weight decay	10^{-3}, 10^{-4}, 10^{-5}	10^{-3}
LayerNorm-L2	Learning rate	10^{-2}, 10^{-3}, 10^{-4}	10^{-3}
	Weight decay	10^{-3}, 10^{-4}, 10^{-5}	10^{-5}
Spectral	Learning rate	10^{-2}, 10^{-3}, 10^{-4}	10^{-2}
	spectral_strengths	10^{-2}, 10^{-3}, 10^{-4}	10^{-2}
CBP	Learning rate	10^{-2}, 10^{-3}, 10^{-4}	10^{-4}
	Replacement rate	10^{-4}, 10^{-5}, 10^{-6}	10^{-5}
$L2$	Learning rate	10^{-2}, 10^{-3}, 10^{-4}	10^{-4}
	Weight decay	10^{-3}, 10^{-4}, 10^{-5}	10^{-3}

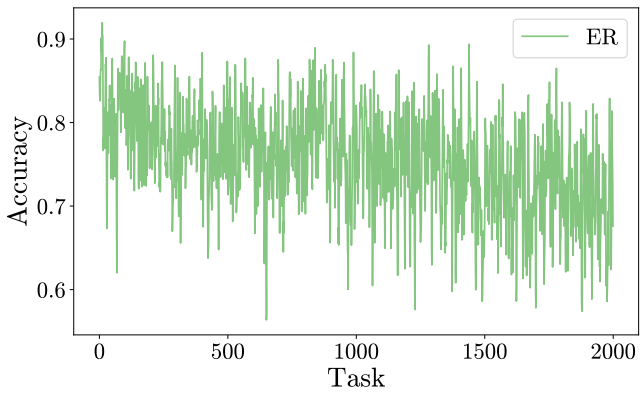
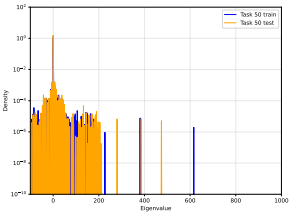
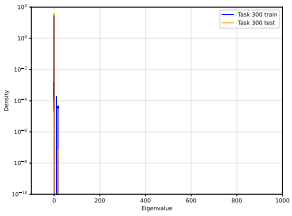


Figure 10. ER Accuracy correspond to Hessian Spectrum on Continual ImageNet.

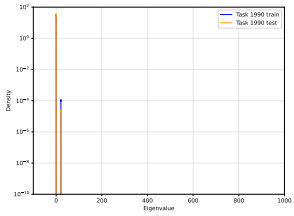
Algorithms that *lose plasticity*



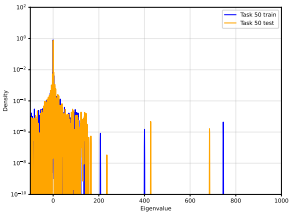
(a) BP: task 50



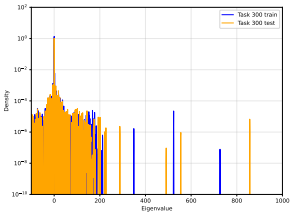
(b) BP: task 300



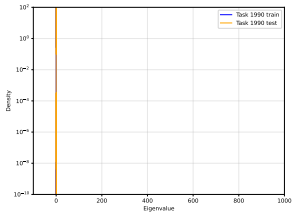
(c) BP: task 1990



(d) $L2$: task 50

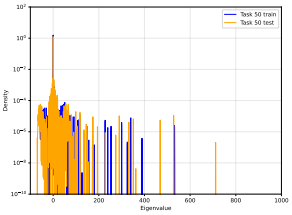


(e) $L2$: task 300

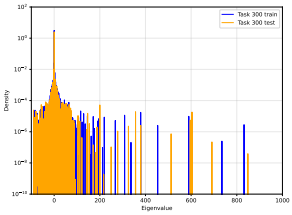


(f) $L2$: task 1990

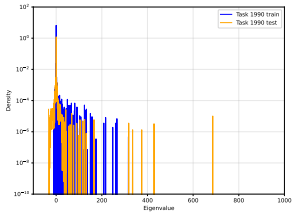
Algorithms that *preserve plasticity*



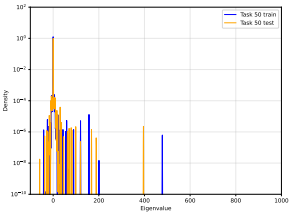
(g) ER: task 50



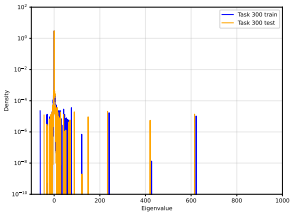
(h) ER: task 300



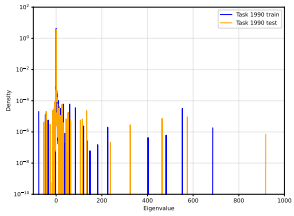
(i) ER: task 1990



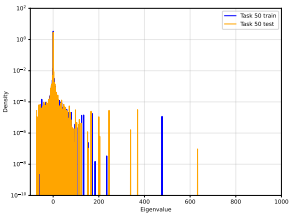
(j) CBP: task 50



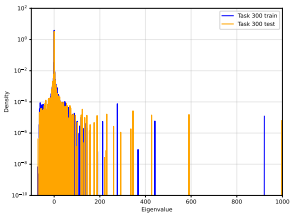
(k) CBP: task 300



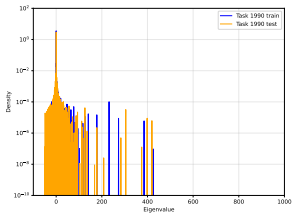
(l) CBP: task 1990



(m) $L2$-ER: task 50



(n) $L2$-ER: task 300



(o) $L2$-ER: task 1990

Figure 11. Comparison of Hessian eigenspectra at task init on Continual ImageNet. From top to bottom, the algorithms are ordered by increasing accuracy. **Top:** Algorithms that lose plasticity (ER, BP). **Bottom:** Algorithms that preserve plasticity ($L2$, CBP, $L2$-ER).

J. Incremental CIFAR

Incremental CIFAR (Dohare et al., 2024) is an adaptation of the original CIFAR-100 dataset (Krizhevsky et al., 2009) where classes are incrementally added. Starting with 5 classes, each task adds 5 new classes for classification until all 100 classes are included. Our setup largely follows (Dohare et al., 2024), but we remove random data transformations and learning rate annealing from their design.

J.1. Network Architecture

We employ a standard ResNet-18 architecture adapted from (Dohare et al., 2024). The network begins with a 3×3 convolutional stem with 64 channels, followed by four sequential residual stages. Each stage contains two BasicBlocks: Layer1 keeps the width at 64 channels, while Layers 24 progressively double the number of channels to 128, 256, and 512, with the first block of each stage performing downsampling via stride-2 convolutions and 1×1 skip connections. After the residual stages, a global average pooling layer reduces the spatial dimension, and then pass the resulting feature vector through two fully connected layers with ReLU activations. Finally, a dense classification head outputs logits for the number of classes in the current task.

```
ResNet18(
  Stem(
    Conv2d(out_channels=64, kernel_size=3x3, stride=1, padding=1, bias=True)
    BatchNorm()
    ReLU()
  )
  Layer1: Sequential(
    BasicBlock(64 -> 64, stride=1)(
      Conv2d(64, 3x3, stride=1, padding=1, bias=True), BatchNorm(), ReLU,
      Conv2d(64, 3x3, stride=1, padding=1, bias=True), BatchNorm(),
      Skip: Identity,
      Add, ReLU
    )
    BasicBlock(64 -> 64, stride=1)(
      Conv2d(64, 3x3, stride=1, padding=1, bias=True), BatchNorm(), ReLU,
      Conv2d(64, 3x3, stride=1, padding=1, bias=True), BatchNorm(),
      Skip: Identity,
      Add, ReLU
    )
  )
  Layer2: Sequential(
    BasicBlock(64 -> 128, stride=2)(
      Conv2d(128, 3x3, stride=2, padding=1, bias=True), BatchNorm(), ReLU,
      Conv2d(128, 3x3, stride=1, padding=1, bias=True), BatchNorm(),
      Skip: Conv2d(128, 1x1, stride=2, bias=True) + BatchNorm(),
      Add, ReLU
    )
    BasicBlock(128 -> 128, stride=1)(
      Conv2d(128, 3x3, stride=1, padding=1, bias=True), BatchNorm(), ReLU,
      Conv2d(128, 3x3, stride=1, padding=1, bias=True), BatchNorm(),
      Skip: Identity,
      Add, ReLU
    )
  )
  Layer3: Sequential(
    BasicBlock(128 -> 256, stride=2)(
      Conv2d(256, 3x3, stride=2, padding=1, bias=True), BatchNorm(), ReLU,
      Conv2d(256, 3x3, stride=1, padding=1, bias=True), BatchNorm(),
```

```
      Skip: Conv2d(256, 1x1, stride=2, bias=True) + BatchNorm(),
      Add, ReLU
    )
    BasicBlock(256 -> 256, stride=1)(
      Conv2d(256, 3x3, stride=1, padding=1, bias=True), BatchNorm(), ReLU,
      Conv2d(256, 3x3, stride=1, padding=1, bias=True), BatchNorm(),
      Skip: Identity,
      Add, ReLU
    )
  )
  Layer4: Sequential(
    BasicBlock(256 -> 512, stride=2)(
      Conv2d(512, 3x3, stride=2, padding=1, bias=True), BatchNorm(), ReLU,
      Conv2d(512, 3x3, stride=1, padding=1, bias=True), BatchNorm(),
      Skip: Conv2d(512, 1x1, stride=2, bias=True) + BatchNorm(),
      Add, ReLU
    )
    BasicBlock(512 -> 512, stride=1)(
      Conv2d(512, 3x3, stride=1, padding=1, bias=True), BatchNorm(), ReLU,
      Conv2d(512, 3x3, stride=1, padding=1, bias=True), BatchNorm(),
      Skip: Identity,
      Add, ReLU
    )
  )
  GlobalAveragePool()
  Dense(out_dims=512, bias=True), ReLU()
  Dense(out_dims=512, bias=True), ReLU()
  Dense(out_dims=num_classes, bias=True)
)
```

J.2. Hyperparameters

In Table 5, we present the default hyperparameters for our ImageNet experiments. Unless otherwise specified, experiments use these defaults.

J.3. Experiments

We conduct our hyperparameter sweep experiments over 5 seeds for all hyperparameters listed in Table 6, and report the selected best hyperparameters in Table 6.

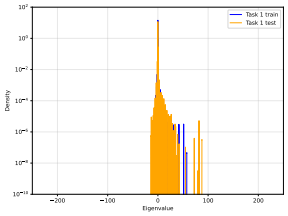
J.4. Incremental CIFAR hessian spectrum

We use the best hyperparameters in Table 6 to plot the Hessian spectrum over 5 seeds, with results shown in Figure 12. Again consistent with our previous experiments, algorithms that preserve plasticity maintain a dense Hessian spectrum, whereas those that lose plasticity exhibit a sparse spectrum. This demonstrates our claim that spectral collapse is a clear indicator of plasticity loss.

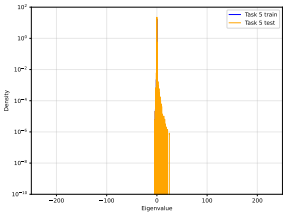
Hyperparam Name	Default	Description
study_name	test	experiment name
seed	2024	base random seed
debug	False	true to enable debug mode
platform	gpu	{cpu, gpu}
n_seeds	1	number of seeds
env	incremental_cifar	environment name
agent	l2	agent options: {er, bp, l2, snp_l2, snp, cbp, l2_er}
alg	ppo	algorithm type: {actor_critic, ppo}
activation	relu	activation options: {relu, tanh}
lr	[0.1]	learning rate(s)
optimizer	sgd	{adam, sgd}
weight_decay	0.0005	$L2$ weight decay
to_perturb	False	whether to perturb the input data
perturb_scale	1×10^{-5}	magnitude of input perturbation
mini_batch_size	100	minibatch size
no_anneal_lr	False	if true, do not anneal the learning rate
max_grad_norm	1×10^{9}	gradient clipping threshold
num_tasks	20	number of tasks used in training/eval
num_epochs	200	number of epochs per task
momentum	0.9	SGD momentum coefficient
effective rank		
er_lr	[0.01]	ER learning rate
er_batch	5	batch size for ER computation
er_step	1	ER update frequency
resetting		
reset	False	reset the network after each task
evaluation		
evaluate	True	evaluate after each task
evaluate_previous	False	evaluate on previous tasks
eval_size	2000	number of evaluation samples per task
compute_hessian	False	whether to compute Hessian spectrum
compute_hessian_size	2000	samples used for Hessian computation
compute_hessian_interval	1	interval in tasks between Hessian runs
continual backpropagation		
cont_backprop	False	enable CBP
replacement_rate	1×10^{-6}	CBP replacement probability per step
decay_rate	0.99	exponential decay for CBP statistics
maturity_threshold	100	steps before a unit is considered mature

Table 5. Incremental CIFAR default hyperparameters

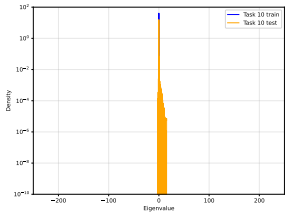
Algorithms that *lose plasticity*



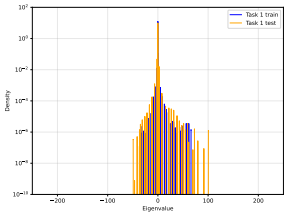
(a) ER: task 1



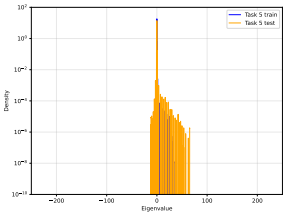
(b) ER: task 5



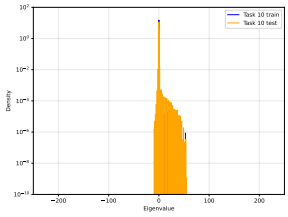
(c) ER: task 10



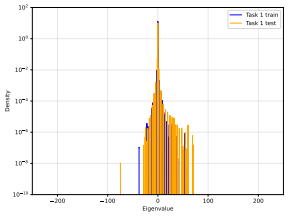
(d) BP: task 1



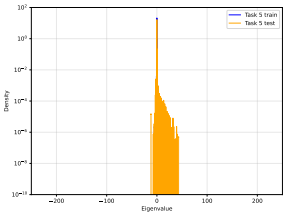
(e) BP: task 5



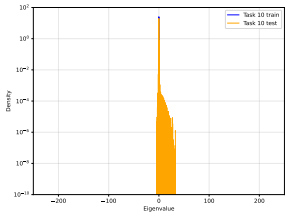
(f) BP: task 10



(g) CBP: task 1

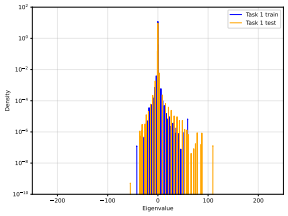


(h) CBP: task 5

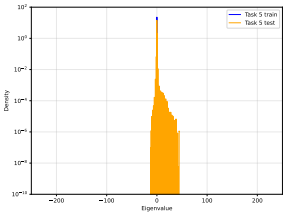


(i) CBP: task 10

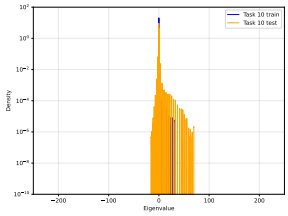
Algorithms that *preserve plasticity*



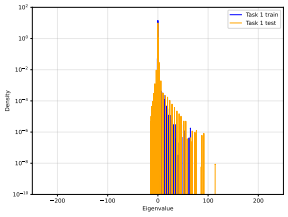
(j) $L2$: task 1



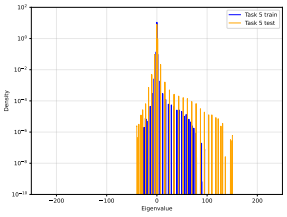
(k) $L2$: task 5



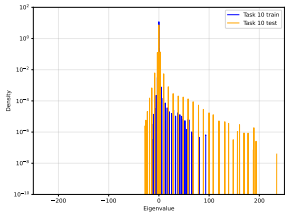
(l) $L2$: task 10



(m) $L2$-ER: task 1



(n) $L2$-ER: task 5



(o) $L2$-ER: task 10

Figure 12. Comparison of Hessian eigenspectra on Incremental CIFAR. Algorithms are ordered by increasing accuracy. **Top:** Algorithms that lose plasticity (ER, BP, CBP). **Bottom:** Algorithms that preserve plasticity ($L2$, $L2$-ER).

Algorithm	Hyperparameter	Sweep Values	Best Value
BP	Learning rate	$\{1 \times 10^{-2}, 1 \times 10^{-3}, 1 \times 10^{-4}\}$	1×10^{-2}
ER	Learning rate	$\{1 \times 10^{-2}, 1 \times 10^{-3}, 1 \times 10^{-4}\}$	1×10^{-2}
	Effective rank lr	$\{1 \times 10^{-3}, 1 \times 10^{-4}, 1 \times 10^{-5}\}$	1×10^{-4}
$L2$-ER	Learning rate	$\{1 \times 10^{-2}\}$	1×10^{-2}
	Effective rank lr	$\{1 \times 10^{-2}, 1 \times 10^{-3}, 1 \times 10^{-4}\}$	1×10^{-3}
	Weight decay	$\{1 \times 10^{-3}, 1 \times 10^{-4}, 1 \times 10^{-5}\}$	1×10^{-3}
CBP	Learning rate	$\{1 \times 10^{-2}, 1 \times 10^{-3}, 1 \times 10^{-4}\}$	1×10^{-2}
	Replacement rate	$\{1 \times 10^{-4}, 1 \times 10^{-5}, 1 \times 10^{-6}\}$	1×10^{-6}
$L2$	Learning rate	$\{1 \times 10^{-2}, 1 \times 10^{-3}, 1 \times 10^{-4}\}$	1×10^{-2}
	Weight decay	$\{1 \times 10^{-3}, 1 \times 10^{-4}, 1 \times 10^{-5}\}$	1×10^{-4}
LayerNorm-L2	Learning rate	$\{1 \times 10^{-2}, 1 \times 10^{-3}, 1 \times 10^{-4}\}$	1×10^{-2}
	Weight decay	$\{1 \times 10^{-3}, 1 \times 10^{-4}, 1 \times 10^{-5}\}$	1×10^{-4}
Spectral	Learning rate	$\{1 \times 10^{-2}, 1 \times 10^{-3}, 1 \times 10^{-4}\}$	1×10^{-2}
	Spectral Strengths	$\{1 \times 10^{-3}, 1 \times 10^{-4}, 1 \times 10^{-5}\}$	1×10^{-2}

Table 6. Hyperparameter sweeps and selected best values for Incremental CIFAR across all algorithms.

K. Slippery Ant

Slippery Ant is a continual reinforcement learning environment where the friction between the ant and the ground changes every 2 million steps. The agent has to adapt its policy to this new friction. In our experiments, we use PPO (Schulman et al., 2017) as our base algorithm, which is an online learning method designed for training on vectorized environments. It is parallelized using the JAX library (Bradbury et al., 2018) based on a batch experimentation library written in JAX (Lu et al., 2022).

K.1. Network Architecture

```
ActorCritic(
  Actor(
    Sequential(
      (0): Dense(in_dims=hidden_size, out_dims=hidden_size, bias=True)
      (1): ReLU()
      (2): Dense(in_dims=hidden_size, out_dims=action_dims, bias=True)
      (3): Categorical() or MultivariateNormalDiag()
    )
  )
  Critic(
    Sequential(
      (0): Dense(in_dims=hidden_size, out_dims=hidden_size, bias=True)
      (1): ReLU()
      (2): Dense(in_dims=hidden_size, out_dims=1, bias=True)
    )
  )
)
```

K.2. Hyperparameters

The default hyperparameters is in Table 8. Unless otherwise specified, experiments use the default hyperparameters.

Algorithm	Hyperparameter	Sweep Values	Best Value
BP	Learning rate	$\{1 \times 10^{-2}, 1 \times 10^{-3}, 1 \times 10^{-4}\}$	1×10^{-4}
ER	Learning rate	$\{1 \times 10^{-2}, 1 \times 10^{-2}, 1 \times 10^{-4}\}$	1×10^{-4}
	Effective rank lr	$\{1 \times 10^{-3}, 1 \times 10^{-4}, 1 \times 10^{-5}\}$	1×10^{-7}
$L2$-ER	Learning rate	$\{1 \times 10^{-4}\}$	1×10^{-4}
	Effective rank lr	$\{1 \times 10^{-5}, 1 \times 10^{-6}, 1 \times 10^{-7}\}$	1×10^{-6}
	Weight decay	$\{1 \times 10^{-3}, 1 \times 10^{-4}, 1 \times 10^{-5}\}$	1×10^{-3}
CBP+$L2$	Learning rate	$\{1 \times 10^{-4}\}$	1×10^{-4}
	Replacement rate	$\{1 \times 10^{-5}, 1 \times 10^{-6}, 1 \times 10^{-7}\}$	1×10^{-7}
	Weight decay	$\{1 \times 10^{-3}, 1 \times 10^{-4}, 1 \times 10^{-5}\}$	–
$L2$	Learning rate	$\{1 \times 10^{-2}, 1 \times 10^{-3}, 1 \times 10^{-4}\}$	1×10^{-4}
	Weight decay	$\{1 \times 10^{-3}, 1 \times 10^{-4}, 1 \times 10^{-5}\}$	1×10^{-3}
LayerNorm-L2	Learning rate	$\{1 \times 10^{-2}, 1 \times 10^{-3}, 1 \times 10^{-4}\}$	1×10^{-4}
	Weight decay	$\{1 \times 10^{-3}, 1 \times 10^{-4}, 1 \times 10^{-5}\}$	1×10^{-3}
Spectral	Learning rate	$\{1 \times 10^{-2}, 1 \times 10^{-3}, 1 \times 10^{-4}\}$	1×10^{-4}
	Weight decay	$\{1 \times 10^{-3}, 1 \times 10^{-4}, 1 \times 10^{-5}\}$	1×10^{-3}

Table 7. Hyperparameter sweeps and selected best values for Slippery Ant across all algorithms.

Hyperparam Name	Default	Description
env	slippery_ant	environment name
num_envs	1	number of parallel environments
gamma	0.99	discount factor
num_steps	2048	steps per rollout
update_epochs	10	number of optimization epochs per update
num_minibatches	128	number of minibatches per epoch
activation	relu	activation function: {relu, tanh}
optimizer	adam	optimizer: {adam, sgd, muon}
lr	[2.5e-4]	learning rate(s)
lambda0	[0.95]	GAE parameter λ
vf_coeff	[1]	value function loss coefficient
weight_decay	0.0	$L2$ weight decay
beta_1	0.9	Adam β_1 coefficient
beta_2	0.999	Adam β_2 coefficient
continual backpropagation		
cont_backprop	False	enable CBP
replacement_rate	1×10^{-4}	CBP replacement probability per step
decay_rate	0.99	exponential decay for CBP statistics
maturity_threshold	1×10^{4}	steps before a unit is considered mature
effective rank		
er	False	enable ER regularization
er_lr	[0.01]	ER learning rate
er_batch	128	batch size for ER computation
er_step	1	ER update frequency
hessian computation		
compute_hessian_init	False	compute Hessian at initialization
compute_hessian_end	False	compute Hessian at the end
compute_hessian_size	2000	number of samples for Hessian computation
compute_hessian_interval	1	interval (in tasks) between Hessian runs
PPO		
hidden_size	256	size of hidden layers
total_steps	5×10^{6}	total number of training steps
entropy_coeff	0.01	entropy regularization coefficient
clip_eps	0.2	PPO clipping parameter
max_grad_norm	1×10^{9}	gradient clipping threshold
anneal_lr	False	anneal learning rate over training
evaluation		
steps_log_freq	4	logging frequency in steps
update_log_freq	8	logging frequency in updates
save_checkpoints	False	save checkpoints during training
save_runner_state	False	save final runner state
seed	2020	random seed
n_seeds	5	number of random seeds
platform	gpu	{cpu, gpu}
debug	False	enable debug mode
show_discounted	False	show discounted returns in logs
study_name	batch_ppo_test	experiment name
change_every	2×10^{6}	steps between environment changes
friction_seed	0	seed for friction schedule

Table 8. Non-stationary policy default hyperparameters

K.3. Experiments

We swept the environment over 5 seeds for all hyperparameters in Table 7. Following the architecture of CBP in (Dohare et al., 2024), we use CBP together with $L2$ normalization instead of just CBP. With the additional sweep on weight decays, we fixed the learning rate of CBP constant. The best hyperparameters are reported in Table 7.

L. Runtime

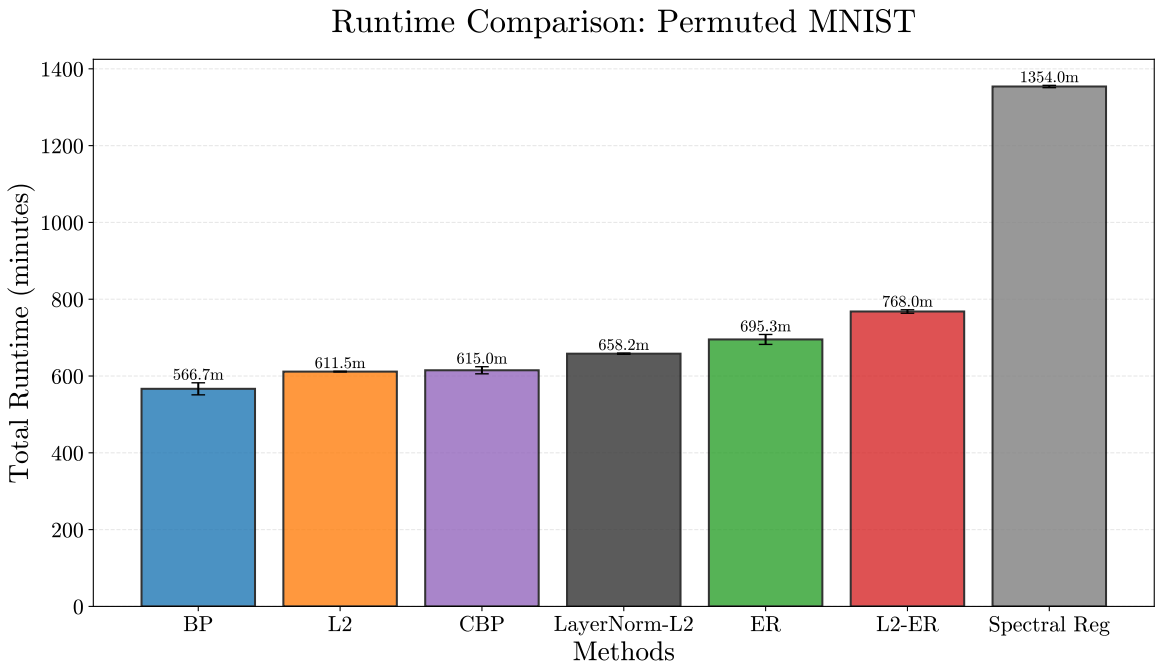


Figure 13. Runtime in Permuted MNIST Across all methods

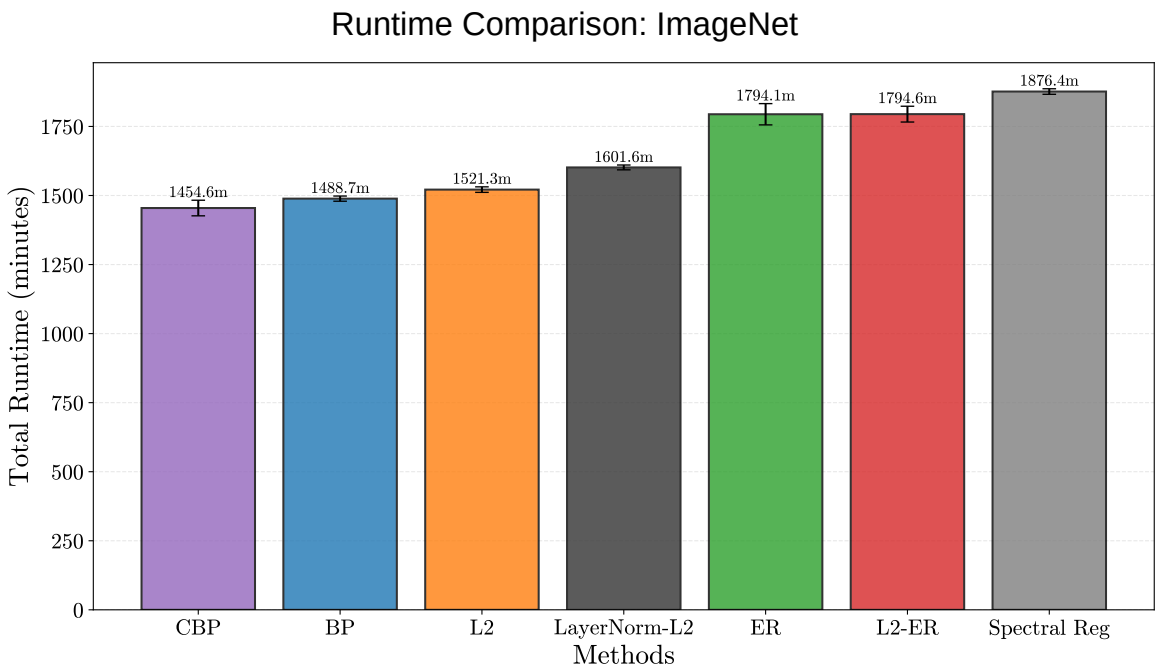


Figure 14. Runtime in ImageNet Across all methods

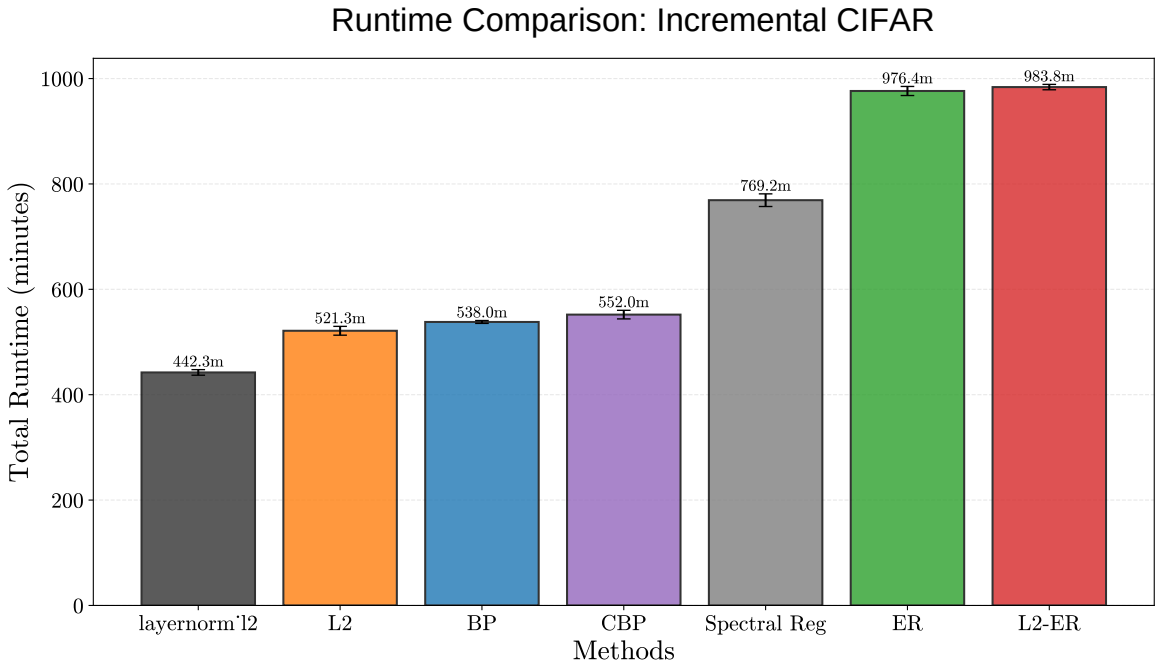


Figure 15. Runtime in Incremental CIFAR Across all methods