Emergent Fibration Symmetry Reveals the Geometry of Learning in Deep Neural Networks

Osvaldo M. Velarde[1], Alireza Hashemi[1], Lucas C. Parra[2], Hernán A. Makse[1*]

[1]Levich Institute and Physics Department, City College of New York, New York, NY, 10031, USA.

[2]Biomedical Engineering Department, City College of New York, New York, NY, 10031.

[*]Corresponding author. Email: hmakse@ccny.cuny.edu

Deep neural networks are often regarded as powerful yet opaque black boxes. In the absence of theoretical insights, performance has been driven by empirical scaling laws. Here, we demonstrate that learning generates local symmetries in the structure of deep neural networks, characterized by isomorphic input and output trees, called fibrations and coverings. These graph symmetries are more flexible than global symmetries used in theoretical physics and geometric deep learning. We prove that covers are stable attractors of stochastic gradient descent. Balanced coloring algorithms readily identify these emergent symmetries, and fibrations enable us to drastically compress the size of networks for most modern architectures without compromising performance. In Transformers, we improve the prevailing scaling law in the number of parameters. Breaking local symmetries also yields state-of-the-art performance in continual learning. These results suggest a new foundation for AI, not based on brute-force scaling up, but on emergent geometry that converts black boxes into interpretable colored graphs.

Despite the development of increasingly powerful neural network architectures, our understanding of how these networks learn and make inferences remains limited. They are often regarded as black boxes, unable to provide explanations for their decision-making processes. In the absence of a theoretically grounded understanding of learning, the recent surge in artificial intelligence (AI) has been driven predominantly by empirical scaling laws identified by Kaplan *et al.* (*1, 2*). This paradigm promotes the notion that “bigger is better": more parameters, more layers, more data, and more computation.

The reliance on scaling as a substitute for understanding creates significant challenges. Training with single large runs has huge cost and energy demands, and when trained sequentially, networks often suffer from a loss of plasticity over time (*3, 4, 5*). Additionally, training large models from scratch does not make efficient use of the data, which requires exceedingly large datasets. The processing in these large models has been notoriously difficult to interpret, making it difficult to build trust in AI for critical applications, such as in the medical space. Without principled guidelines, architecture design becomes a costly trial-and-error process. Finally, large models are excessively over-parameterized, yet paradoxically, they perform well in practice (*6*), a subject of ongoing theoretical debate (*7, 8*).

To address these challenges, we propose a theoretical formalism based on symmetries in the internal structure of the computational graph of deep neural networks. We find that learning via stochastic gradient descent (SGD) leads to the emergence of local symmetries in the computational graph, which are less rigid than better-known global symmetry groups, such as shift equivariance in convolutional neural networks (CNNs) (*9, 10*) or permutation equivariance (*11, 12, 13*) in graph neural networks (GNNs). Although symmetry groups are fundamental to geometric deep learning (GDL) (*14*), and our understanding of theoretical physics (*14, 15, 16*), they are too restrictive to capture the diversity observed not only in artificial, but also in biological neural systems (*17, 18, 19, 20*). We instead identify *local* symmetries that naturally emerge from the symmetries in the input trees of network nodes, relevant during inference, and in the output trees, relevant during learning.

These local symmetries are represented using mathematical maps known as fibrations, opfibrations, and coverings, and are found with "balance coloring" algorithms from graph theory (*17, 21*). These maps were originally developed by Grothendieck in category theory (*22*) and later adapted for graphs (*23, 24*). Here, we prove mathematically that covering symmetries emerge in stochastic gradient descent due to "synchronized learning" of the weight updates. This, in turn, induces fibration symmetries, which explains the emergence of neural activity synchronization often reported in deep networks (*20, 25, 26*), and allows substantial compression of over-parameterized models without compromising performance. We validate this phenomenon across a wide range of architectures, including multilayer perceptrons (MLP), CNNs, recurrent networks (e.g., long short-term memory, LSTM), and Transformers in both supervised and reinforcement learning (RL) settings. This geometric compression outperforms existing scaling laws (*1*) to achieve the same performance with fewer parameters. Furthermore, by carefully breaking these symmetries, we expand the network capacity to overcome loss of plasticity (*4, 5*), demonstrating state-of-the-art performance in continual learning (*3*). This principled approach, grounded in systematic symmetry identification, mitigates the inefficiencies of trial-and-error methods.

1 Hierarchy of symmetries

The symmetries we will discuss form a hierarchy and emerge in a variety of network structures. For an easy visualization of this hierarchy, consider a layered feedforward network with binary weights (where the connections are 1 or 0), as shown in Fig. 1. In the figure, nodes are colored to indicate their class according to various symmetries. We will generalize these concepts to continuous-valued weights in the next section.

Nodes have a *fibration symmetry* and are said to belong to the same *fiber* if they have isomorphic *input trees* (*17, 23, 24*). This means that the entire structure of the connections from the graph's input to the nodes is isomorphic (see Methods 6.1). In practice, fibers are identified by balanced coloring algorithms from graph theory (*17, 21*). This algorithm partitions the network into balanced coloring classes of nodes (the fiber partition) by iteratively assigning the same color to nodes that receive the same set of input colors (*27*), hence the term *balanced coloring*. An example is shown in Fig. 1a, where nodes in the same fiber are colored the same. The input tree for a red node is shown on the left.

Fibration symmetries allow the graph G to be compressed into a smaller *base* graph B_{fib}. This compression $\varphi_{\text{fib}} : G \to B_{\text{fib}}$ works by merging all nodes that share the same color (i.e., belong to the same fiber) while conserving the input trees (Fig. 1a). We will show that the resulting base

graph B_{fib} performs the same "forward" computation (see Eq. 1) as the original graph G. In deep neural networks (DNNs), it means that we can compress the network without losing performance. The inverse of this compression is called *lifting* (*23*), explained in more detail in Methods 6.1.1.

A similar principle applies when learning the parameters of the network. For the canonical learning algorithm, backpropagation, the error at the output is propagated backward through the *output tree* (see Eq. 2). An *opfiber symmetry* occurs when nodes share isomorphic output trees — the structure of connections leading from them to the output layer (Fig. 1b right). This allows for a similar compression $\varphi_{\text{op}} : G \to B_{\text{op}}$ of *opfibers* which are out-balanced colorings of the graph as shown in Fig. 1b. We will show that nodes with this symmetry receive the same error signal.

When multiple nodes have isomorphic input and output trees, they form a *covering symmetry* and belong to a *cover*(Fig. 1c). The corresponding coloring is the intersection of the fibration and the opfibration coloring partitions; i.e., a finer partition of the graph. Clearly, a covering has a more stringent symmetry than fibers or opfibers.

The most strict symmetry is the *automorphism*, which forms symmetry groups. This is a global permutation of the network's nodes that leaves the entire graph's connectivity unchanged, i.e., it is a global symmetry, which contrasts with the local preservation of inputs (outputs) in (op)fibration. An example of nodes that are in the same automorphism symmetry is shown in Fig. 1d along with the permutation of node labels. Although automorphisms are the cornerstone of GDL (*14*) and theoretical physics (*15, 16, 28*), this symmetry is so restrictive that we have never observed it in our trained networks.

In summary, these symmetries form a hierarchy of increasing strictness: from fibrations and opfibrations, to coverings, and finally to automorphisms. As conditions become more stringent, there are more distinct color classes and fewer nodes within each class (meaning fewer symmetries). Less strict local symmetries, such as fibrations, are more common and allow greater compression into a more compact base. This compression results in a model with fewer effective degrees of freedom, which corresponds to a stronger inductive bias. For more theoretical details about symmetries of a graph, see Methods 6.1.

2 Theoretical results

2.1 Computational graphs in deep learning

The hierarchical local symmetries explained above emerge in DNN structures with trainable parameters, like MLPs, CNNs, RNNs and Transformers (see Methods 6.2), but they are most easily understood in the canonical MLP. In this architecture, nodes are organized into multiple layers ($\ell = 1, ..., N$), with weight $W_{ik}^{(\ell)}$ connecting node k to i from layer $\ell - 1$ to ℓ. The weighted edges $W_{ik}^{(\ell)}$ create the computational graph that is the basis of our fibration analysis (see Fig. S6). Each node i in the network performs a weighted sum of its inputs k, followed by a point-wise nonlinear activation function σ,

$$h_i^{(\ell)} = \sigma\left(\sum_k W_{ik}^{(\ell)} h_k^{(\ell-1)}\right), \tag{1}$$

where $h^{(0)} = x \in \mathbb{R}^{d_0}$ is the input of the network and $h^{(N)} = y \in \mathbb{R}^{d_N}$ is the output of the network.

The activity propagates "forwards" across layers from the input x to the output y. During learning, the gradient of a loss function $\mathcal{L}$ with respect to weights $W_{ik}^{(\ell)}$ can be calculated with the error backpropagation algorithm, with the error signal $\delta_i^{(\ell)}$ flowing "backwards" through the computational graph (*29*),

$$\delta_i^{(\ell)} = \sigma'\left(h_i^{(\ell)}\right) \sum_k W_{ki}^{(\ell+1)} \delta_k^{(\ell+1)} , \tag{2}$$

where $\delta_i^{(N)} = \frac{\partial \mathcal{L}}{\partial h_i^{(N)}}$ is the error evaluated from the output layer. Note that the error signal depends on the activity of the node via the derivative of the activation function.

2.2 Fibration symmetry implies activity synchronization

We can now formally generalize the concept of fibration symmetry from binary weights to continuous weights. Two nodes i and j in layer ℓ belong to the same fiber (denoted $i \underset{\text{fib}}{\sim} j$) if and only if (iff):

- **Fibration symmetry:**

$$i \underset{\text{fib}}{\sim} j \iff \forall c \in C_{\ell-1}^{\text{fib}} : \quad \sum_{k\in c} W_{ik}^{(\ell)} = \sum_{k\in c} W_{jk}^{(\ell)} \tag{3}$$

This criterion partitions the nodes in the layer ℓ into a coloring C_ℓ^{fib} based on the input weights $W_{ik}^{(\ell)}$ and the color partitioning of the previous layer $C_{\ell-1}^{\text{fib}}$. In words, this definition states that two nodes are in the same fiber iff for all colors from the previous layer the sums of weights from these colors are the same. Or, more simply, the total input from each color is the same, which also ensures that their activity is the same. This recursive definition starts in the input layer, $\ell = 0$, with all nodes (d_0 nodes) having distinct colors $C_0^{\text{fib}} = \{1, 2, ..., d_0\}$, i.e., trivial fibers. The definition of fibers in Eq. (3) based on the sum of weights is the correct generalization from binary graphs (Fig. 1) to weighted graphs, as we show in Methods 6.3.

Similarly to symmetry-induced invariance theorems in physics (*15*), fibration symmetry induces activity synchronization during forward inference. This is the subject of the following theorem:

- **Activity synchronization:** Given two nodes i and j in layer ℓ, the following hold in networks with inputs x:

$$i \underset{\text{fib}}{\sim} j \implies h_i^{(\ell)} = h_j^{(\ell)} \qquad \text{for any network input } x \tag{4}$$

The proof of Eq. (4) is in Methods 6.4. The relation emerges naturally from the recursive dependence of neuronal activity $h_i^{(\ell)}$ on activity in the previous layer through input weights $W_{ik}^{(\ell)}$. Similar proofs exist for dynamical systems in (*17, 30, 31*).

Equation (1) establishes the relationship between the network structure captured by its weights and the network function captured by its activity. Therefore, the input tree structure governs the synchronization patterns.

2.3 Flavored opfibration implies error synchronization

The backpropagation of errors in Eq. (2) exhibits greater complexity due to the intrinsic coupling between error gradients and forward activations mediated by the slope of the nonlinear activation $\sigma'(h_i^\ell)$. Specifically, the error $\delta_i^{(\ell)}$ for node i depends not only on the error signals from the next layer (mediated by the output weights $W_{ik}^{(\ell)}$) but also on the activation value $h_i^{(\ell)}$ at that node, which introduces a nonlinear coupling between $\delta_i^{(\ell)}$ and $h_i^{(\ell)}$, absent in forward propagation.

This coupling can be systematically absorbed into an output tree that governs error propagation analogously to how the input tree governs activity propagation. The key observation is that the slope — which is a function of $h_i^{(\ell)}$ — couples the in-balanced color assignments from the forward fiber structure (see Eq. (3)) to the edges that define the error flow in Eq. (2).

We formalize this by introducing *edge flavors* into the output tree, where the flavors are inherited from the node colors of the forward computation. We take advantage of the concept of *flavor-preserving* opfibration symmetry (*32*), which preserves flavor assignments of edges. Therefore, two nodes belong to the same flavored opfiber if:

- **Flavor-preserving opfibration symmetry:**

$$i \underset{\text{op}}{\sim} j \implies \quad \forall \hat{c} \in C_{\ell+1}^{\text{op}} : \quad \sum_{k\in\hat{c}} W_{ki}^{(\ell+1)} = \sum_{k\in\hat{c}} W_{kj}^{(\ell+1)} \quad \& \quad q(i) = q(j) \tag{5}$$

with flavors given by

$$q(i) = \begin{cases} 1 & \text{linear activation} \\ c(i) \in C_\ell^{\text{fib}} & \text{non-linear activation} \end{cases}$$

Equation (5) defines the opfiber partition C_ℓ^{op}. In the linear activation case ($\sigma' = 1$), all weights have the same flavor ($q = 1$), and the forward and backward coloring decouple and can be treated independently. In the non-linear case, the flavors must first be calculated using the fiber coloring C_ℓ^{fib} computed forward through the entire network as above. Then, the sum-of-weights condition of Eq. (5) is evaluated for each flavor separately, recursively starting at the output of the network, with all nodes in different opfibers $C_N^{\text{op}} = \{1, 2, ..., d_N\}$.

Using this definition, we arrive at the following theorem. The proof is given in Methods 6.4.

- **Errors synchronization:** Given two nodes i and j in layer ℓ, the following hold in a network with inputs x and targets y:

$$i \underset{\text{op}}{\sim} j \implies \delta_i^{(\ell)} = \delta_j^{(\ell)} \qquad \text{for any network input } x, \text{ target } y\,. \tag{6}$$

We note that for linear activations ($\sigma' = 1$), the relationship between synchronization and (op)fibration becomes bidirectional ($\iff$) in both Eqs. (4) and (6, as we prove in Methods 6.5. This means that synchronized nodes necessarily form (op)fibers, and conversely, nodes in a (op)fiber will synchronize. This represents a strict one-to-one structure-function equivalence that is difficult to demonstrate for more general non-linear dynamical systems (*33*).

Alternative definitions for opfibration that capture the coupling with the forward pass are treated in Supplementary text 7.1. In the theory of gases, the linear case is analogous to non-interacting particles in an ideal gas, whereas the nonlinear case corresponds to particle interactions in a real gas, and flavor plays the role of a compression factor.

2.4 Fibers and opfibers synchronize learning

The weight update of the gradient descent algorithm locally depends on node activity and error signals (*29*):

$$\Delta W_{mi}^{(\ell)} = -\alpha \frac{\partial \mathcal{L}}{\partial W_{mi}^{(\ell)}} = -\alpha \delta_m^{(\ell)} \cdot h_i^{(\ell-1)} , \tag{7}$$

where α is a learning rate (see the derivation in Methods 6.6).

According to Eqs. (4) and (6) nodes i, j of layer $\ell - 1$ have the same activity, $h_i^{(\ell-1)} = h_j^{(\ell-1)}$ if they are in a fiber; and nodes m, n of layer ℓ have the same error signal $\delta_n^{(\ell)} = \delta_m^{(\ell)}$ if they are in an opfiber. From this and Eq. (7) follows:

- **Learning synchronization:** Given nodes $i \underset{\text{fib}}{\sim} j$ in layer $\ell - 1$ and $m \underset{\text{op}}{\sim} n$ in layer ℓ, the weight updates according to gradient descent are synchronized:

$$\Delta W_{mi}^{(\ell)} = \Delta W_{mj}^{(\ell)} = \Delta W_{ni}^{(\ell)} = \Delta W_{nj}^{(\ell)} . \tag{8}$$

The proof is given in Methods 6.7. Note that, unlike the synchronization of activations and errors in Eqs. (4) and (6), this synchronization occurs in the parameter space. Fig. 2a shows an example of this synchronization.

2.5 Covering symmetries emerge during stochastic gradient descent

The final symmetry that we introduce in DNNs is the covering symmetry. Two nodes belong to the same cover iff they are in the same fiber and the same flavored opfiber:

- **Covering symmetry:**

$$i \underset{\text{cov}}{\sim} j \iff i \underset{\text{fib}}{\sim} j \quad \& \quad i \underset{\text{op}}{\sim} j . \tag{9}$$

This condition defines a covering partition of the network C_ℓ^{cov}, which, as we shall see, is preserved under GD.

We now turn to the question of how these symmetries emerge during learning. When a neural network is initialized, its weights are set to random values. Large random graphs of this kind rarely have significant global automorphism symmetries as they are forbidden by Erdős-Rényi asymmetry theorem (*34*). However, the training process itself creates local redundancies. For example, recent studies have observed that SGD causes different nodes to develop similar input and output weights (*35*). We have also previously observed the emergence of *synchronized* nodes – nodes that have similar activity across all inputs during training (*20*).

Our results provide a theoretical proof linking these empirical observations to the emergence of covering symmetry. Namely, as a consequence of *synchronized learning*, once two nodes are in a single covering at learning time step t, the weight update with the gradient descent algorithm keeps the nodes in the covering at time $t + 1$. This is formalized in the following theorem:

Theorem 2.1 ***Cover Coarse-Graining Theorem.*** *Under the dynamic of GD, the covering partition* $C_\ell^{\text{cov}}(t+1)$ *is a coarsening of* $C_\ell^{\text{cov}}(t)$.

We proof this theoretical result in Methods 6.8. The theorem implies that once multiple nodes are in the same cover, they cannot exit that cover, and thus constitute an invariant set. However, distinct covers can merge to form larger (coarser) covers. The proof of this Cover Coarse-Graining Theorem is based on stability rules (see Methods 6.8). These provide an intuition for how fibration and opfibration symmetries develop during learning. The stability rules state that the fibers in the layer ℓ preserve the fibers in the layer $\ell + 1$, while the opfibers preserve the opfibers in the layer $\ell - 1$, as shown in Fig. 2b. Thus, opfiber formation tends to propagate backward from the output layer, while fiber formation tends to propagate forward from the input as learning progresses. Note that for the non-linear case, the opfibers have a flavoring of the edges to capture their dependence on the forward pass. This introduces an asymmetry in the stability rules, reflected in Fig 2b, with edge colors defined at the output, but not at the input.

Chen *et al.* (*35*) have shown that any invariant parameter set of GD is a stable attractor in the SGD algorithm, provided that there is a sufficiently large learning constant. Another interpretation of Theorem 2.1 is: *If network parameters θ_t at time t are such that $i \underset{\text{cov}}{\sim} j$; then $i \underset{\text{cov}}{\sim} j$ still holds for θ_{t+1}.* That is, $i \underset{\text{cov}}{\sim} j$ is invariant under the weight update in GD. The intuition is that stochastic fluctuations during learning, akin to noise-induced stabilization in dynamical systems (*36*), occasionally bring nodes into an invariant set; once there, our theorem says the gradient descent algorithm can no longer break the symmetry due to synchronized learning. Using the same result, we conclude that cover formation is a stable attractor of the SGD dynamics. In Section 3.1, we provide empirical evidence for this claim.

The theoretical results presented here readily generalize to other computational graphs, including hypergraphs, as we discuss in Methods 6.2, which includes operations such as convolutions (in CNN), multiplicative gating (in most RNN and LSTM in particular), residual connections, and attention layers, which is the core innovation of Transformers.

2.6 Fibration compression preserves activity and loss

Once fibers are identified, one can produce a more compact base graph by merging all nodes in a fiber. The forward computation of the graph G is preserved in this base graph B_{fib} if the new weights $\hat{W}$ are specified as follows (see the proof in Methods 6.9.1):

$$\hat{W}^{(\ell)}_{c'c} = \frac{1}{|c'|} \sum_{i \in c', k \in c} W^{(\ell)}_{ik}. \tag{10}$$

where $c \in C^{\text{fib}}_{\ell-1}$, $c' \in C^{\text{fib}}_{\ell}$ and $|c'|$ is the number of nodes in the fiber c'. In other words, we sum up all output weights with the same colors and average over the input weights. This equation defines fiber compression in the weighted graph $\varphi_{\text{fib}} : G \to B_{\text{fib}}$, and is exemplified in Fig. 2c.

For convolutional networks, the same compression rule applies to feature channels rather than nodes (Fig. 2d, Methods 6.9.2). For gates in recurrent networks (e.g. LSTM) and attention layers (Transformers), there are no weighted interactions and compression only needs to connect nodes in the base as exemplified in Fig. 2e-f (see Methods 6.9.3).

We will demonstrate that these compression methods allow substantial compression of networks without changing their performance, and that this targeted fibration compression outperforms existing pruning methods (Section 3.2). We then show that more aggressive compression followed by retuning results in a new scaling law that outperforms the existing parameter scaling law for

Transformers (*1*) (Section 3.3). Finally, we argue that the emergence of covering symmetry results in the loss of plasticity in deep networks (*5*) and show empirically that breaking these symmetries achieves a new state-of-the-art in continual learning (Section 3.4). But first, we empirically confirm how fiber symmetries emerge during learning (Section 3.1).

3 Empirical results

3.1 Emergence of fibers and opfibers during learning

We empirically validate the emergence of covers during SGD for an MLP trained in MNIST and a CNN trained in ImageNet (see Fig. 3a, c). At the start of learning, all nodes have unique random connectivity, constituting trivial (single-node) fibers and opfibers. As learning progresses, the nodes start to group into covers, forming fibers and opfibers. Hence, the number of unique fibers decreases. When the fibers stabilize and the accuracy no longer increases, the network has learned the training set. Details about the MLP and CNN used here are provided in Methods 6.10.1 and 6.10.2. The formation of fibers across layers is summarized in Fig. S7 - Methods 6.11. The emergence of symmetries is illustrated in Supplementary Material Movie S1.

3.2 Fibration compression outperforms existing pruning methods

To identify fibers, we have to check if weights satisfy Eq. (3). In practice, this can only be determined with a precision ε: $|\sum_{k\in c}[W_{ik}^{(\ell)} - W_{jk}^{(\ell)}]| \leq \varepsilon$. The coloring algorithm that identifies isomorphic input trees (detailed in Methods 6.12) therefore has this fiber precision ε as a parameter. With increasing ϵ, a larger fraction of nodes can be compressed into (quasi-) fibers (*37*), but the forward computation is no longer exactly preserved and we observe a loss of performance.

For the MLP trained on MNIST, the fibration compression reduces the model to 17% of its original parameter count without an appreciable loss of classification accuracy (black point in Fig. 3b). We observe comparable compression for a CNN trained on ImageNet, where fibration compression achieves reductions of similar magnitude without degradation in test performance (Fig. 3d). For both MLP and CNN, we find that alternative compression methods are less effective. In Fig. 3b-d, fibration compression (red curve) outperforms random pruning nodes (blue curve) and pruning nodes with small weights (L2-norm, pink curve). Details about pruning are shown in Methods 6.10.3.

Next, we evaluate lossless compression in a reinforcement learning context using an LSTM-based agent trained to play the Atari game Beam Rider (details in Methods 6.10.4, see Fig. 3e, green curve). At training epoch 50, we compress the network using $\varepsilon = 0.3$ (black), achieving a network with 40% of the original parameters. The compressed network did not exhibit degraded performance after continued training. The latter continues to create fibers, allowing for further compression to 20% of the original model size in episode 300.

3.3 Fibration compression with retuning reveals efficient scaling law for Transformers

For larger compression factors (achieved with larger ε) the loss in performance can be compensated for by retuning the network after compression, a standard approach in existing pruning methods (*38, 39*). We explore this in the context of a sequence-to-sequence Transformer trained for German-English translation (Methods 6.13). For this purpose, we develop a fibration compression and retuning method with adaptive fiber precision (Methods 6.14 - Fig. S8) resulting in retuning curves (Fig. 3g, orange and purple curves). For a model with 78M parameters, we compress the transformer by a large factor of 31x of its original size with 4.1% loss in performance (black arrow on the orange retuning curve in Fig. 3g). In Fig. 3g, we compare this method with other approaches: standard pruning methods (blue and pink curves), and fibration compression without retuning while searching for the optimal ε value for each layer individually (red curve).

Next, we train different Transformers with varying network sizes (e.g. 50M and 78M parameters - purple and orange curves in Fig.3g and orange retuning curves in Fig.3h) and then compress each model size with fibration retuning. The baseline model (without compression) follows the established scaling law of Kaplan *et al.* (*1*) between the loss function and the number of parameters (black dashed line) with a power-law behavior:

$$\mathcal{L} = L_0^k \; P^{-\alpha_k} \,, \tag{11}$$

where P is the number of parameters of the baseline model, L_0^k is a constant and $\alpha_k = 0.028$ is the Kaplan exponent (Methods 6.15). By Kaplan scaling, we mean increasing the model's parameter count by increasing all layers' dimensions by the same amount. For each baseline model, we then apply the fibration compression-retuning method and find a more efficient scaling law for the compressed network (orange dashed line):

$$\mathcal{L} = L_0^f \; (P_{min|L})^{-\alpha_f} \,. \tag{12}$$

Here, $P_{min|L}$ is the minimum number of parameters of the compressed model at a given loss value and baseline model (see Methods 6.15). The power-law exponent obtained with the fibration compression protocol ($\alpha_f = 0.049$) is larger than the value obtained without compression implying a faster-decaying scaling curve.

The model at $P_{min|L}$ represents a minimal fibration base of the transformer computational graph with baseline P, effectively capturing the data's symmetries using a minimal representation. The faster decay of the fibration scaling suggests the existence of an alternative, more efficient approach to modeling scaling than the brute-force method, in which all layers are uniformly scaled up, as exemplified by the Kaplan scaling laws.

3.4 Fibration symmetry breaking for continual learning

Cover symmetries that emerge during SGD reduce the network's degrees of freedom and manifest as redundant, synchronized activity that limits the repertoire of learnable features. Several heuristics have been proposed to promote diversity and avoid redundancy. For instance, "dropout" (*40*) breaks symmetries by updating only a fraction of nodes selected at random. Residual connections (*41*) break symmetries that have emerged in the skipped layers, as we show in Methods 6.2. Techniques

that explicitly avoid redundancy also prevent fiber formation. For example, (*42*) proposes to identify and remove redundancy in convolutional layers (see Supplementary Text 7.2). SlimConv recombines features in convolutional layers (*43*) and Barlow Twins (*44*), aiming to reduce correlations between learned features (see Supplementary Text 7.3). However, these ad hoc mechanisms operate indiscriminately by preventing or breaking the fibers that emerge from the data with SGD.

In contrast, we propose a principled surgical approach to symmetry breaking (*45*) by first compression covers and then randomizing the remaining weights as follows (Fig. 1e):

1. Identify covers and reduce the graph to the covering base by applying fibration symmetry compression via Eq. (10) to guarantee that the function of the network and its performance are preserved.

2. Add nodes to restore the original size. Each new node is initialized with random input weights and zero outgoing weights. This restores degrees of freedom for further learning.

Deep networks have been found to lose their ability to learn when trained sequentially on multiple new tasks (*4, 5*). Current heuristics for training new tasks in continual learning, such as resetting the final layers with random weights or resetting specific nodes, do not fully overcome this loss of plasticity (*5*).

We propose using the symmetry-breaking approach above to overcome this loss of plasticity in continual learning. We evaluate the breaking process on the continuous ImageNet task (*46*), which comprises 5,000 sequential binary classification tasks drawn from the 1,000-class data set (details in Methods 6.16). First, we observe that conventional SGD stops learning after approximately 2,000 tasks (Fig. 3f, blue curve), as shown by (*3*). We find that this is accompanied by cover formations (Methods 6.16 - Fig. S9). The emergence of symmetry induces parameter tying, which reduces the degrees of freedom in the parameter space. This constraint directly explains the observed loss of network plasticity: with fewer independent directions for optimization, learning stagnates. We compared performance to the state-of-the-art "Continual Backpropagation" from (*3*), which outperforms heuristics like shrink-and-perturb (*47*). Continual backpropagation resets nodes that have become inactive. In Supplementary text 7.4, we show that this technique is a special case of our symmetry-breaking mechanism. Up to task 1,500, the performance of continual backpropagation and our method "Fibration Symmetry Breaking" is comparable. However, beyond that point, our method outperforms continual backpropagation, cutting the remaining deficit in accuracy by half (from 90% to 95% by task 5,000). This indicates that the network continues to capture useful features that generalize between classes.

3.5 Emergence of synchronization clusters

The activity synchronization theorem, Eq. (4), establishes that the nodes of a fiber have the same activity for all inputs. We ask how node synchronization changes when we limit the input space and, in particular, if we draw inputs from individual classes in a classification task. We find that the correlation matrices for individual classes form large clusters of synchronized nodes (Fig. 4a). These class-conditional clusters are a coarsening of the fibers (see Methods 6.17 - Fig. S11). Thus, each class-conditional synchrony cluster can be decomposed into multiple fibers, forming hierarchical clusters of nodes. One can interpret the class-conditional synchronization clusters in Fig. 4a in terms of features of the input: It is well accepted that node activity in hidden layers

captures a hierarchy of features in the stimulus, that are shared across classes; e.g. simple examples for 2D CNN are edges, junctions, and corners (*48*). The intersection of all class-conditional clusters is a refinement that is captured by the fibers. Therefore, fibers capture the hierarchical regularity present in all training data.

4 Hierarchical organization of the loss landscape under SGD dynamics

Cover formation is a hierarchical coarse-graining under the SGD dynamic in the parameter and node space. This is exemplified in Fig. 4b-c, where we show the state of the network in three different training stages (T_1, T_2, T_3), their transitions $T_{1\to2}$ and $T_{2\to3}$, and the evolution of three nodes (i, j, k) as covers merge.

Covers are a refinement of fibers. Fibers are a refinement of synchronization clusters, and these in turn are a refinement of class-conditional synchrony clusters. This is exemplified in Fig. 4b, along with the development of covers and class-conditional clusters during learning. The red box groups the class-conditional clusters for the red class (and similarly for the blue class) at time T_1. Covers in green are a refinement of both red and blue clusters. As learning progresses, the covers merge at transitions $T_{1\to2}$ and $T_{2\to3}$ according to the coarse-graining theorem.

Figure 4c is a symbolic representation of a parameter space (weights). The potential learning trajectories (black curves) merge the covers containing nodes i, j and k. During the learning phase T_1, the three nodes i, j, and k belong to distinct covers. At a later point in the training, i and j merge into the same cover. According to the coarse-graining theorem, once this occurs, they cannot separate again. In other words, the first time i and j share a cover defines an irreversible transition (green dashed line $T_{1\to2}$). A similar transition occurs when the cover containing node k merges into the cover containing i and j, defining a second transition line $T_{2\to3}$. The colored zones in this space are regions where the network has the same cover base B (with identical nodes and weights).

The loss landscape $\mathcal{L}(w)$, with $w \in \mathbb{R}^d$, is characterized by a hierarchy that can be formally represented as a tree (Fig. 4d). All network configurations in this d-dimensional parameter space that have the same minimal base B'' will have the same loss value. The root of the tree is this base in $\mathbb{R}^{d''}$; while the leaf nodes are all the possible lifting of the base in $\mathbb{R}^d$. The tree contains multiple leaves because φ_{fib} can reduce different graphs G into the same minimal base B'' (This is evident in Eq. (10), as a linear subset of all possible $W^{(\ell)}$ can generate the same $\hat{W}^{(\ell)}$). The inverse lifting operation can transform a representation from $\mathbb{R}^{d''}$ (minimal base B'') to $\mathbb{R}^d$ (original network G) through an intermediate lifting to $\mathbb{R}^{d'}$ where $d'' < d' < d$. These intermediate steps create the hierarchy of the trees (see Methods 6.18 - Fig. S12).

The learning trajectory navigates downhill in this loss landscape (solid black line, in Figs. 4c-d) following the hierarchical coarsening of the covers under the SGD learning dynamics where the system progressively explores increasingly coarse-grained symmetry quotients during optimization. This structure has analogies in the organization of associative memory in neural networks (*49*) and the energy landscape in spin glasses (*50*).

5 From black box to colored graphs through geometric interpretability

Our geometric interpretation transforms the interpretability problem in deep learning. Rather than treating neural networks as opaque black boxes, the fibration framework reveals them as colored geometric structures where each color represents a distinct fiber. The training process becomes interpretable as a geometric flow through hierarchical parameter space, where SGD converges on regions of constant-loss manifolds and crosses boundaries at irreversible symmetry transitions. The interpretability emerges directly from the geometry: we can trace which nodes belong to which fibers (colors), understand when and why they merge during training (geometric transitions) and predict their collective behavior (synchronization within fibers).

This geometric lens transforms the fundamental question from "what is the network computing?" to "what geometric structure has the network discovered?" The colored fiber decomposition provides an atlas of the network's internal organization, where the geometry itself constitutes the interpretable representation. Unlike post-hoc interpretability methods that attempt to explain black boxes after training, our framework reveals that networks are inherently geometric objects whose training dynamics naturally expose their interpretable colored structure. The black box was never truly opaque; it simply waited to be viewed through the right geometric coordinates, which are the colored fibers that capture its fundamental symmetries.

We have shown how learning aligns the network's internal structure with the structure of the data, finding a compact, minimal set of features sufficient to describe the dataset, and thus controlling for overtraining. This perspective resolves the paradox of over-parameterization in deep networks: gradient descent finds structured symmetric solutions, implying that the emergent model is much simpler than the raw parameter count suggests. It also explains why deep networks outperform shallow networks despite the Universal Approximation Theorem; depth facilitates the gradual layer-by-layer formation of coverings, a process that is statistically improbable in a wide, shallow network.

Our framework also provides a unified theoretical foundation for many ad-hoc heuristics. Hard-coded inductive biases such as weight-tying in CNNs are simply a predefined symmetry; our work shows how such symmetries can emerge from learning. Our symmetry-driven pruning provides an exact, data-independent theory for network compression, validated across diverse architectures including MLPs, CNNs, transformers, and in reinforcement learning settings. The fibration allows for drastic compression of symmetric nodes without loss of performance. Furthermore, we introduce a novel symmetry-breaking protocol based on the lifting property that outperforms state-of-the-art continual learning. This provides a principled mechanism to "stay young" and generalizes practices such as continual backpropagation.

In essence, our work frames learning not as parameter-tuning, but as structure formation. The process transforms the opaque black box into an interpretable colored graph, where emergent symmetries reveal the data's learned regularities. This theory-driven perspective offers a path to designing future AIs in which mathematical principles drive performance, generalization, and interpretability.

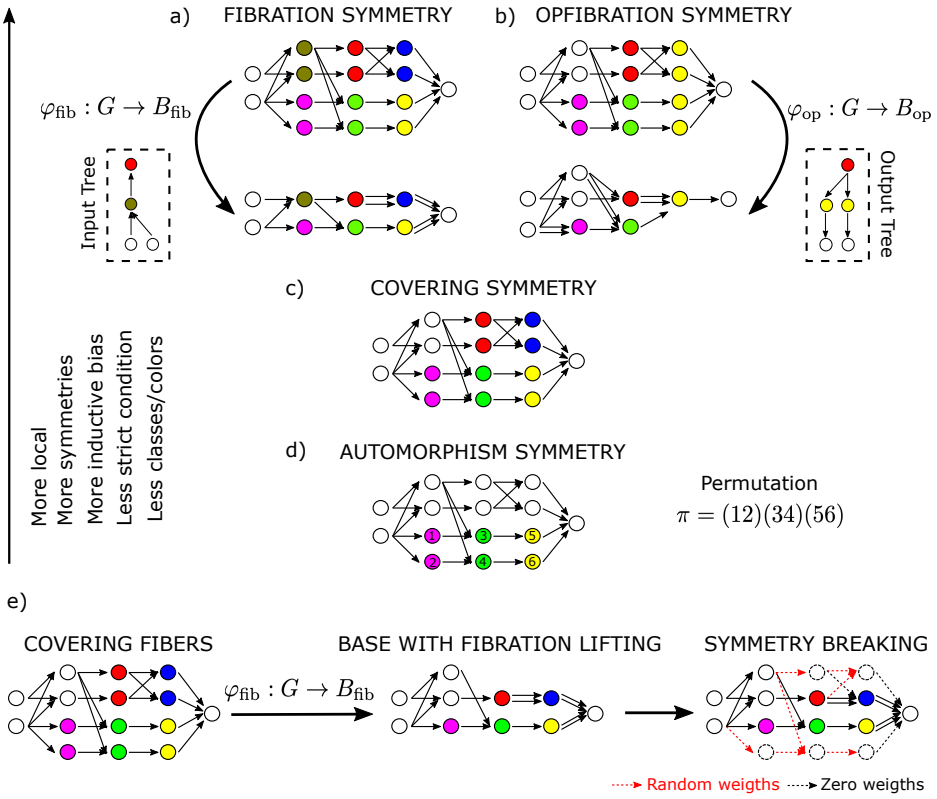

Figure 1: **Hierarchy of symmetries in a feedforward network.** For simplicity, weights are binary, and edges with zero weight are not plotted. Conventional automorphism permutation is exemplified in the case of a graph in panel **(d)**. It identifies three pairs of symmetric nodes by a permutation π. The proposed framework is based on the generalization to local symmetries, such as (op)fibrations and coverings (**a-c**); see text for detail. These symmetries have less strict conditions that can be satisfied by a larger number of nodes resulting in fewer symmetry classes (colors) (blank nodes should be interpreted as distinct colors). The reduced degrees of freedom increases the inductive bias. **(e)** Breaking of symmetry for continual learning consists of two steps. Compress to the covering base via fibration-lifting (middle) to preserve the learned task. Randomize or zero out redundant weights (right) to provide new degrees of freedom to continue learning.

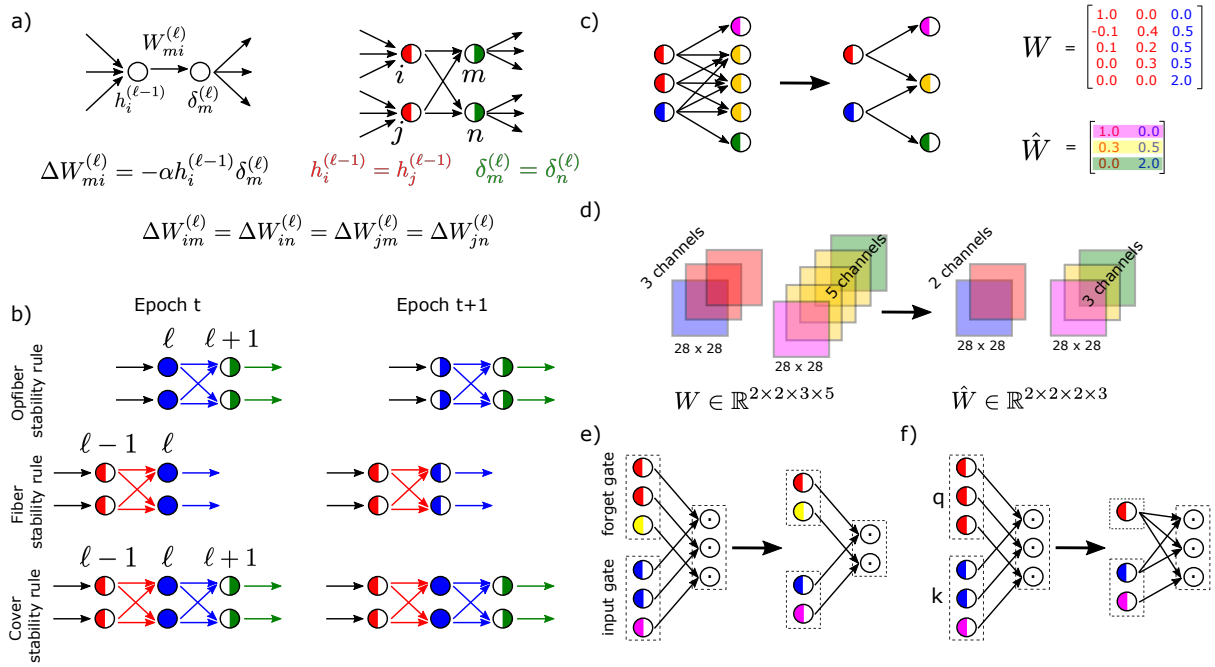

Figure 2: **Theoretical results**. **(a)** The gradient descent learning rule updates a connection's weight $W_{mi}^{(\ell)}$ based on the product of the activity $h_i^{(\ell-1)}$ and the error signal $\delta_m^{(\ell)}$. Nodes in the same fiber (red left sides) synchronize their activities - Eq. (4). Nodes in the same opfibers (green right sides) synchronize their error signals - Eq. (6). Weights connecting nodes in a fiber with nodes in an opfiber will have the same updates $\Delta W_{mi}^{(\ell)}$ - Eq. (8). **(b)** Stability rules used to proof the cover coarse-graining theorem (see Methods 6.8). Connection colors indicate "flavor" for the opfibers. Black connections for fibrations are not flavored. **(c)** Example of fibration compression of a dense layer. Edges with zero weight are not plotted. $\hat{W}$ is calculated based on W using Eq. (10). **(d)** Example of fibration compression of a convolution layer. It reduces the number of channels (3 to 2 and 5 to 3). **(e)** Example of fibration compression of gates in LSTM. The base has an identical number of nodes across all gates (here 2 nodes per gate). **(f)** The fibration compression for attention modules preserves the count of interactions (3 multiplications) even when the number of fibers of the k and q differ.

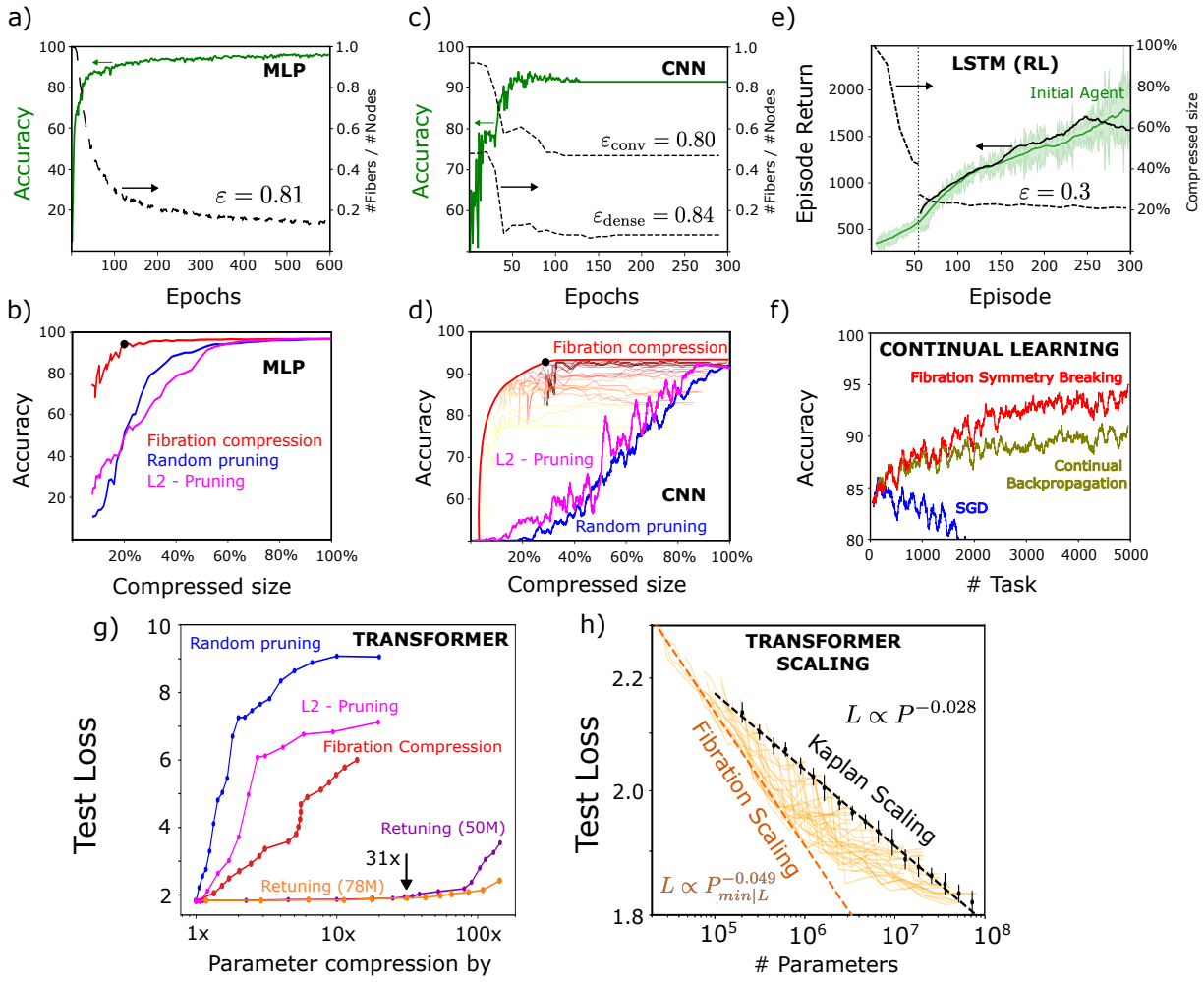

Figure 3: **Empirical results. (a)** MLP trained on MNIST dataset. Classification accuracy increases across learning epochs (green), while the number of trivial (single-node) fibers decreases in favor of larger multi-node fibers (black dashed curves). **(b)** Accuracy of the compressed network depends on the level of compression (red). Alternative pruning methods at different levels of compression (blue and pink curves). **(c)** Same as (a) but for CNN trained on ImageNet. **(d)** Same as (b) for CNN. Yellow to red curves correspond to different values of $\varepsilon_{\text{conv}}$. Bold red curve represents maximum performance for different combinations of ($\varepsilon_{\text{conv}}$,$\varepsilon_{\text{dense}}$). **(e)** Agent with LSTM trained via reinforcement learning to play the Atari game Beam Rider. Returns increase with learning episode (solid curves). Learning with no compression (green). At episode 50 (vertical line), the network is compressed (black) and training continues. Size of fibration base (dashed curves). **(f)** Continual learning with sequential ImageNet binary classification tasks, using conventional SGD (blue), Continual Backpropagation (brown), and the proposed Fibration Symmetry Breaking (red). **(g)** Transformers trained on the Multi30k dataset. Fibration compression using layer-specific optimal ε (red). Fibration compression followed by retuning for a model with 50M parameters (purple) and 78M (orange). Blue and pink as in (b). **(h)** Power-law scaling of the loss value as a function of the number of parameters in the original transformer (black) and after fibration compression and retuning (orange curves). Parameter count excludes the embedding layers. Dashed scaling lines are fit to the original networks and to the most compressed network at a fixed loss. More details in the text.

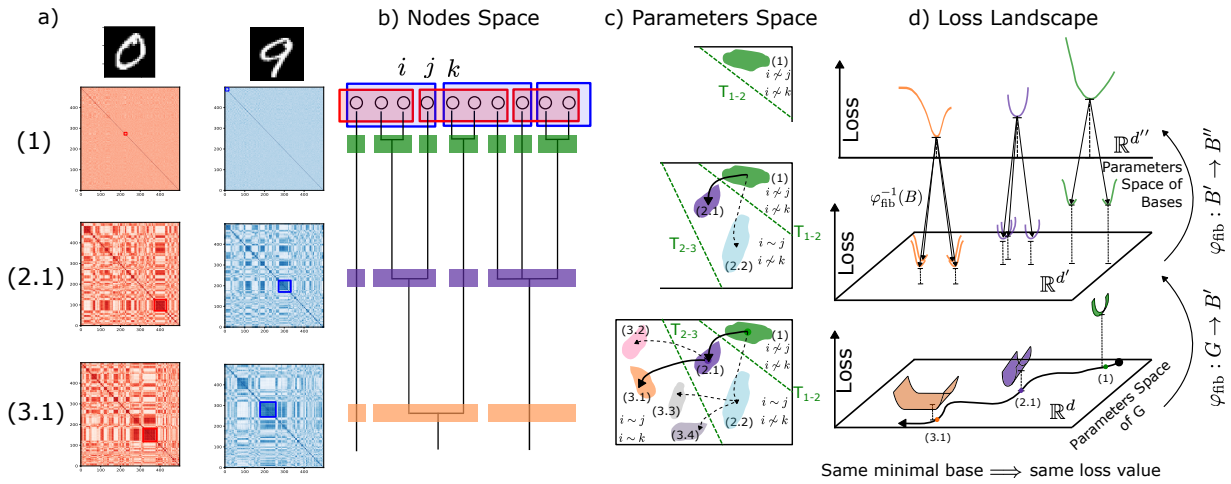

Figure 4: **Synchronization, hierarchical clustering and loss landscape. (a)** Class-conditional correlation matrix of activity with network inputs drawn from individual classes. Here they are shown for the last layer of an MLP trained on MNIST in early epochs (1), in middle of the training (2.1) and after training (3.1). Clusters in this matrix are referred to as synchrony clusters. **(b)** Temporal evolution of covers in one layer. In one of the early epochs, (1), we identify synchrony clusters (red and blue) and covers (green). Covers are located within cluster intersections. In later epochs, (2.1) and (3.1), covers merged to form coarser partitions (purple and orange). Nodes i, j and k start in different covers (green), but merge into the same cover (purple and orange). **(c)** A network is a point in parameter space, e.g. at time point (1). Multiple networks can have the same basis B (green region). Learning trajectories (black curves) merge nodes i and j into the same cover at an irreversible transition boundary T_{1-2}. Multiple B may have i and j in the same cover (e.g., network (2.1) and (2.2)). SGD selects one of these possibilities. Once k merges into the same cover, another irreversible transition occurs T_{2-3}. **(d)** Hierarchical loss landscape (see text for details).

References and Notes

1. J. Kaplan, et al., Scaling laws for neural language models. arXiv (2020), https://arxiv.org/abs/2001.08361.

2. J. Hoffmann, et al., Training compute-optimal large language models. arXiv (2022), https://arxiv.org/abs/2203.15556.

3. S. Dohare, et al., Loss of plasticity in deep continual learning. Nature **632**, 768–774 (2024).

4. C. Lyle, et al., Understanding plasticity in neural networks, in International Conference on Machine Learning (PMLR) (2023), pp. 23190–23211.

5. C. Lyle, et al., Disentangling the causes of plasticity loss in neural networks. arXiv (2024), https://arxiv.org/abs/2402.18762.

6. T. J. Sejnowski, The unreasonable effectiveness of deep learning in artificial intelligence. Proc. Nat. Acad. Sci. USA **117**, 30033–30038 (2020).

7. M. Belkin, D. Hsu, S. Ma, S. Mandal, Reconciling modern machine-learning practice and the classical bias–variance trade-off. Proc. Nat. Acad. Sci. USA **116**, 15849–15854 (2019).

8. P. Nakkiran, et al., Deep double descent: Where bigger models and more data hurt, in International Conference on Learning Representations (ICLR) (2020).

9. Y. Lecun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition. Proceedings of the IEEE **86** (11), 2278–2324 (1998).

10. T. Cohen, M. Welling, Group equivariant convolutional networks, in Proceedings of The 33rd International Conference on Machine Learning, M. F. Balcan, K. Q. Weinberger, Eds. (PMLR, New York, New York, USA), vol. 48 of Proceedings of Machine Learning Research (2016), pp. 2990–2999.

11. T. N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks. arXiv (2016), https://arxiv.org/abs/1609.02907.

12. V. G. Satorras, E. Hoogeboom, M. Welling, E(n) Equivariant Graph Neural Networks, in International Conference on Machine Learning (2021).

13. H. A. Makse, R. P. J. Perazzo, The thermodynamics of dyslexic learning. Inter. J. Neural Sys. **3**, 351–360 (1992).

14. M. M. Bronstein, J. Bruna, T. Cohen, P. Veličković, Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. arXiv (2021), https://arxiv.org/abs/2104.13478.

15. F. Wilczek, A Beautiful Question: Finding Nature's Deep Design (Penguin Press) (2016).

16. S. Weinberg, The Quantum Theory of Fields, vol. 2 (Cambridge University Press) (1995).

17. H. A. Makse, P. Boldi, F. Sorrentino, I. Stewart, Symmetries of Living and Artificial Intelligence Systems: Fibrations and Synchronization in Networks (Cambridge University Press, forthcoming) (2026), https://arxiv.org/pdf/2502.18713.

18. T. Gili, et al., Fibration symmetry-breaking supports functional transitions in a brain network engaged in language. arXiv (2025), https://arxiv.org/abs/2409.02674.

19. B. Avila, et al., Symmetries and synchronization from whole-neural activity in *C. elegans* connectome: Integration of functional and structural networks. Proc. Natl. Acad. Sci. USA **122**, e2417850122 (2025).

20. O. Velarde, L. C. Parra, P. Boldi, H. A. Makse, The role of fibration symmetries in Geometric Deep Learning. Proc. Nat. Acad. Sci. USA, in press (2026), https://arxiv.org/abs/2408.15894.

21. H. Kamei, P. J. A. Cock, Computation of balanced equivalence relations and their lattice for a coupled cell network. SIAM J. Appl. Dyn. Syst. **12**, 352–382 (2013).

22. A. Grothendieck, Technique de descente et théorémes d'existence en géométrie algébrique, I. Généralités. Descente par morphismes fidélement plats. Seminaire Bourbaki **190** (1959-1960).

23. P. Boldi, S. Vigna, Fibrations of graphs. Discrete Math. **243**, 21–66 (2002).

24. F. Morone, I. Leifer, H. A. Makse, Fibration symmetries uncover the building blocks of biological networks. Proc. Nat. Acad. Sci. USA **117**, 8306–8314 (2020).

25. V. Papyan, X. Han, D. L. Donoho, Prevalence of neural collapse during the terminal phase of deep learning training. Proc. Natl. Acad. Sci. USA **117**, 24652–24663 (2020).

26. D. Doimo, A. Glielmo, S. Goldt, A. Laio, Redundant representations help generalization in wide neural networks, in Advances in Neural Information Processing Systems, S. Koyejo, et al., Eds. (Curran Associates, Inc.), vol. 35 (2022), pp. 19659–19672.

27. M. Golubitsky, I. Stewart, Nonlinear dynamics of networks: the groupoid formalism. Bulletin of the American Meteorological Society **43**, 305–364 (2006).

28. H. Georgi, Lie Algebras in Particle Physics: From Isospin to Unified Theories (Taylor & Francis) (2000).

29. D. E. Rumelhart, G. E. Hinton, R. J. Williams, Learning representations by back-propagating errors. Nature **323**, 533–536 (1986).

30. L. DeVille, E. Lerman, Modular dynamical systems on networks. J. Eur. Math. Soc. **17**, 2977–3013 (2015).

31. E. Nijholt, B. Rink, J. Sanders, Graph fibrations and symmetries of network dynamics. Diff. Equ. **261**, 4861–4896 (2016).

32. P. Boldi, V. Lonati, M. Santini, S. Vigna, Graph fibrations, graph isomorphism, and PageRank. RAIRO - Theoretical Informatics and Applications **40**, 227–253 (2006).

33. H. Park, K. Friston, Structural and functional brain networks: from connections to cognition. Science **342**, 1238411 (2013).

34. P. Erdős, A. Rényi, Asymmetric graphs. Acta Mathematica Academiae Scientiarum Hungaricae **14**, 295–315 (1963).

35. F. Chen, D. Kunin, A. Yamamura, S. Ganguli, Stochastic collapse: How gradient noise attracts SGD dynamics towards simpler subnetworks. Advances in Neural Information Processing Systems **36**, 35027–35063 (2023).

36. D. Herzog, J. Mattingly, Noise-induced stabilization of planar flows I. Electronic Journal of Probability **20**, 1–43 (2015).

37. P. Boldi, I. Leifer, H. A. Makse, Quasifibrations of graphs to find symmetries and reconstruct biological networks. J. Stat. Mech.: Theor. Exp. **2022**, 113401 (2022).

38. F. Meng, et al., Pruning filter in filter, in Advances in Neural Information Processing Systems (Curran Associates, Inc.), vol. 33 (2020), pp. 17629–17640.

39. S. Vadera, S. Ameen, Methods for pruning deep neural networks. IEEE Access (2020).

40. N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: A simple way to prevent neural Networks from Overfitting. Journal of Machine Learning Research **15**, 1929–1958 (2014).

41. K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in Proceedings of the IEEE conference on computer vision and pattern recognition (2016).

42. Z. Wang, C. Li, X. Wang, Convolutional neural network pruning with structural redundancy reduction, in Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (2021), pp. 14913–14922.

43. J. Qiu, C. Chen, S. Liu, H. Y. Zhang, B. Zeng, Slimconv: Reducing channel redundancy in convolutional neural networks by features recombining. IEEE Transactions on Image Processing **30**, 6434–6445 (2021).

44. J. Zbontar, L. Jing, I. Misra, Y. LeCun, S. Deny, Barlow twins: Self-supervised learning via redundancy reduction, in International conference on machine learning (PMLR) (2021), pp. 12310–12320.

45. P. W. Anderson, More is different: Broken symmetry and the nature of the hierarchical structure of science. Science **177**, 393–396 (1972).

46. G. M. van de Ven, T. Tuytelaars, A. S. Tolias, Three types of incremental learning. Nature Machine Intelligence **4**, 1185–1197 (2022).

47. A. Chebykin, A. Dushatskiy, T. Alderliesten, P. A. N. Bosman, Shrink-Perturb improves architecture mixing during population based training for neural Architecture Search. arXiv (2023), `https://arxiv.org/abs/2307.15621`.

48. M. D. Zeiler, R. Fergus, Visualizing and understanding convolutional networks, in European conference on computer vision (Springer) (2014), pp. 818–833.

49. N. Parga, M. A. Virasoro, The ultrametric organization of memories in a neural network. J. Phys. (Paris) **47**, 1857–1864 (1986).

50. M. Mézard, G. Parisi, M. A. Virasoro, Spin Glass Theory and Beyond, vol. 9 (World Scientific Lecture Notes in Physics) (1987).

51. M. A. Aguiar, A. P. S. Dias, F. Ferreira, Patterns of synchrony for feed-forward and auto-regulation feed-forward neural networks. Chaos: An Interdisciplinary Journal of Nonlinear Science **27** (2017).

52. I. Leifer, D. Phillips, F. Sorrentino, H. A. Makse, Symmetry-driven network reconstruction through pseudobalanced coloring optimization. J. Stat. Mech.: Theor. Exp. **2022**, 073403 (2022).

53. J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms. arXiv (2017), https://arxiv.org/abs/1707.06347.

54. N. Norris, Universal covers of graphs: Isomorphism to depth n-1 implies isomorphism to all depths. Discrete Applied Mathematics **56**, 61–74 (1995).

55. D. Elliott, S. Frank, K. Simaan, L. Specia, Multi30K: Multilingual English-German image descriptions, in Proceedings of the 5th Workshop on Vision and Language (Association for Computational Linguistics) (2016), pp. 70–74.

56. A. Vaswani, et al., Attention is all you need. arXiv (2023), https://arxiv.org/abs/1706.03762.

57. D. P. Kingma, J. Ba, Adam: A method for stochastic optimization. arXiv (2017), https://arxiv.org/abs/1412.6980.

58. B. O. Ayinde, T. Inanc, J. M. Zurada, Redundant feature pruning for accelerated inference in deep neural networks. Neural Networks **118**, 148–158 (2019).

59. K. Liu, M. Suganuma, T. Okatani, Bridging the gap from asymmetry tricks to decorrelation principles in non-contrastive self-supervised learning. Advances in neural information processing systems **35**, 19824–19835 (2022).

Acknowledgments

Funding and acknowledgments: Partial support for this work was provided by the National Institutes of Health through grants R01CA247910 (OMV, LCP, HAM) and R01EB028157 (HAM), as well as the Army Research Office with grant WF911-NF-24-1-0031 (OMV, LCP). We thank Matteo Serafino for help and discussions.

Competing interests: There are no competing interests to declare.

Data and materials availability: The code is available at Github/MakseLab/fibrations_in_dnns

Supplementary materials

Methods
Supplementary Text
Figs. S1 to S3
Tables S1 to S4
References *(7-59)*
Movie S1
Data S1

Supplementary Materials for Emergent Fibration Symmetry Reveals the Geometry of Learning in Deep Neural Networks

Osvaldo M. Velarde[1], Alireza Hashemi[1], Lucas C. Parra[2], Hernán A. Makse[1*]
[1]Levich Institute and Physics Department, City College of New York, New York, NY, 10031, USA.
[2]Biomedical Engineering Department, City College of New York, New York, NY, 10031.
[*]Corresponding author. Email: hmakse@ccny.cuny.edu

This PDF file includes:

Methods
Supplementary Text
Figures S1 to S7
Captions for Movies S1

Other Supplementary Materials for this manuscript:

Movies S1

6 Methods

This supplementary section first reviews the formal definitions of fibration symmetries in graphs. We then link them to the most common computational graphs used for deep neural networks. Then a number of formal proofs are provided for the main theoretical results. In addition, we include details of the empirical evaluations reported in the main text.

6.1 Global and local symmetries in graphs: coverings, fibrations, and opfibrations

We first discuss symmetries in a binary (unweighted) graph where edges are restricted to binary values (0 or 1) to introduce the main concepts didactically, following Fig. 1. Graph fibrations were first introduced for binary graphs by Boldi and Vigna (*23*) based on the categorical definition of Grothendieck fibration from (*22*). Section 6.3 generalizes the definition to weighted graphs and hypergraphs for applications to DNNs.

A directed graph $G = (N_G, A_G)$ consists of a set N_G of nodes and a set A_G of edges. Each edge is associated with a source and a target node, and one way to represent the full structure of the graph is its adjacency matrix. For every node $u \in N_G$, there is a corresponding input tree T_u that represents the set of all paths of G ending in u. Similarly, there exists a corresponding output tree $\hat{T}_u$ that represents the set of all paths of G starting from u. We say that two input trees T_u and T_v are isomorphic ($T_u \sim T_v$) when there is a bijective map $\tau_{u\to v} : T_u \to T_v$, which maps the nodes and edges of T_u one-to-one to the nodes and edges of T_v. The same definition and notation apply to output trees with a bijective map denoted by $\hat{\tau}_{u\to v} : \hat{T}_u \to \hat{T}_v$.

For graph G,

1. A *fibration symmetry* of G is a surjective homomorphism $\varphi_{\text{fib}} : G \to B$ that preserves the input tree:
$$\forall u, v \in N_G : \varphi_{\text{fib}}(u) = \varphi_{\text{fib}}(v) \in N_B \iff T_u \sim T_v. \tag{S1}$$

2. An *opfibration symmetry* of G is a surjective homomorphism $\varphi_{\text{op}} : G \to B$ that preserves the output tree:
$$\forall u, v \in N_G : \varphi_{\text{op}}(u) = \varphi_{\text{op}}(v) \in N_B \iff \hat{T}_u \sim \hat{T}_v. \tag{S2}$$

3. A *covering symmetry* of G is a surjective homomorphism $\varphi_{\text{cov}} : G \to B$ that preserves the input and output trees:
$$\forall u, v \in N_G : \varphi_{\text{cov}}(u) = \varphi_{\text{cov}}(v) \in N_B \iff T_u \sim T_v \quad \& \quad \hat{T}_u \sim \hat{T}_v. \tag{S3}$$

4. An *automorphism* of a graph G is a bijective map $\pi_{auto} : G \to G$, such that the pair of nodes u and v forms an edge (u, v) if and only if $(\pi_{auto}(u), \pi_{auto}(v))$ also forms an edge. Automorphisms are also called permutation symmetries.

Each symmetry, fibration, opfibration, covering, and automorphism, gives rise to partitions called (respectively): fiber C^{fib}, opfiber C^{op}, cover C^{cov}, and orbit partitions C^{auto}, which are hierarchical in the sense of coarsening: $C^{\text{fib}} \geq C^{\text{cov}} \geq C^{\text{auto}}$ and $C^{\text{op}} \geq C^{\text{cov}} \geq C^{\text{auto}}$.

A partition C of a graph G is *finer* than (also called *a refinement of*) a partition C' if every element of C is a subset of some element of C'. That is, C is a further fragmentation of C'. We denote this case as $C \leq C'$. In this example, C' is *coarser* than C ($C' \geq C$) and C' is a *c*oarsening (merger) of elements of C. The coarsest possible partition of a graph is one in which all nodes belong to a single partition. This is a graph with maximal symmetry. The finest possible partition is the (trivial) partition into singletons, which is the partition of a graph with no symmetry at all (or with only trivial symmetry).

As seen in Fig. 1, for a directed graph, the fiber and the opfiber partitions are two different coarsenings of the cover partition, and the cover partition is a coarsening of the orbit partition. In principle, there is no relation (finer or coarser) between the fiber and the opfiber partition. However, when the graph is undirected, the fiber, opfiber, and cover partitions coincide. Yet, they are still a coarsening of the orbital partition. Furthermore, as shown in Fig. 4b, the cluster synchronization partition (obtained dynamically by clustering activity synchronization in the forward pass, averaging over samples) is a coarsening of the fiber partition (but not of the opfiber partition).

All four partitions, fibers, opfibers, covers, and orbits, are different balanced colorings of the graph (*27*): fibers are in-balanced colorings, opfibers are out-balanced colorings, and covers are in- and out-balanced colorings. Orbits are also in- and out-balanced colorings. However, orbits have an extra condition compared to covers: two nodes in an orbit must be obtainable from one another through the application of a permutation symmetry (automorphism) of the graph.

For any partition, we will use the notation c to indicate some color and $|c|$ the cardinality of the color, i.e., the number of nodes with the same color c.

Due to the refinement relations, the number of balanced colors increases from the (op)fiber partition to cover to orbit partition. This is reflected in Fig. 1 with 9 colors for fibers and 9 colors for opfibers as compared to 10 colors for covers and 12 colors for orbits (consider that the blank nodes in the figure have all different colors). This also reflects the increase in symmetry from automorphisms to covers to (op)fibers.

Automorphisms form symmetry groups. That is, the set of automorphisms of a graph satisfies the composition law, associativity, and has an inverse and identity. They are global symmetry transformations of the graph since they apply a global permutation of all nodes (some nodes may be trivially permuted to themselves, Eq. (S4) that preserves the global adjacency matrix, that is, the edges between all nodes are the same before and after the permutation symmetry.

In particular, we show an automorphism that maps 1 to 2, 2 to 1, 3 to 4, 4 to 3, 5 to 6, 6 to 5, and all other nodes map to themselves (see Fig. 1d). It is denoted in cycle notation by:

$$\pi = (1\,2)(3\,4)(5\,6). \tag{S4}$$

The other nodes are trivially permuted.

Fibrations, opfibrations and coverings strictly generalize automorphisms by relaxing the global constraints to impose only a local preservation of structure such as the input and output trees. Examples of these symmetries in a graph are shown in Fig. 1. These symmetries do not form groups, but categories. The color-preserving isomorphisms between (output)input trees form groupoids (i.e., groups without composition law) (*27*). The fibration framework is therefore also known as 'the groupoid formalism' in the work of Golubitsky and Stewart (*27*).

Remark 1. One might wonder why coverings are not automorphisms, since both preserve inputs and outputs. The key distinction is scope: coverings preserve the input and output structure of the

nodes within each cover, while automorphisms preserve the input and output structure globally across the entire graph.

To exemplify the meaning of input and output trees and its relation with automorphism, we consider the graph G in Fig. 1c and call the remaining nodes as: first hidden layer: 7 (top blank) and 8 (bottom blank), second hidden layer: 9 (top red), 10 (bottom red), third hidden layer: 11 (top blue), 12 (bottom blue). We also call 13 and 14 the top and bottom inputs, and 15 the graph's output. Red nodes 9 and 10 form a cover since they have isomorphic input and output trees. That is, there is an input-isomorphism: $\tau_{9\to10} = (9, 7, 13, 14) \to (10, 8, 13, 14)$ and an output-isomorphism: $\hat{\tau}_{9\to10} = (9, 11, 12, 15) \to (10, 11, 12, 15)$ from the input and output tree of node 9 to 10. There is also an isomorphism between edges, but since all edges are the same (binary graph), there is no need to specify it.

One may wonder if nodes 9 and 10 might form an orbit, i.e., if they are also symmetric under a permutation symmetry. If we permute 9 and 10, then 7 and 8 need to be permuted as well to preserve the inputs of 9 and 10 (11 and 12 do not need to be permuted, as the outputs of 9 and 10 are preserved by the permutation). Thus, the permutation $\pi = (9\,10)(7\,8)$, in principle, preserves the adjacency of 9 and 10. However, node 8 is now connected to 3 and 4, while node 7 is not, and there is no other permutation that can restore the original adjacency (for instance, permuting 3 and 4 will not fix it). Thus, while π is a valid permutation of the graph, it is not a permutation symmetry (automorphism) of the graph. The violation occurs not in the input or output trees of 9 and 10. In fact, one can check that the input and output trees of nodes 9 and 10 are still isomorphic in the permuted graph $\pi(G)$. The violation occurs in other parts of the graph: the inputs of nodes 3 and 4. This simple example illustrates the 'local' versus 'global' preservation condition for (op)fibrations/coverings versus automorphisms, an important ingredient of the framework presented here.

Remark 2. Note that 'local' gauge symmetries in physics have a different meaning (*28*). They refer to spacetime locality captured by the fiber bundle, not the fibration-theoretic locality. Fibrations are more general than fiber bundles since they only require relaxed fibers with local symmetry, rather than the symmetry groups required by the fibers in a fiber bundle. Fiber bundles are widespread in theoretical physics but not in biology or AI (see (*17*) for further details).

Remark 3. In our formulation, nodes with no inputs (for instance, all nodes in the input layer of an MLP) are assigned different balanced colors, ie, each node belongs to a different fiber. Same for the output layer and opfibers. This differs from the original definition of graph fibrations in (*23*).

Remark 4. When G is a graph with flavored edges (*32*), the previous symmetry definitions remain valid using colored input and output trees. Two flavored trees (T_u with flavor q_u and T_v with flavor q_v) are considered isomorphic if there exists a bijection $\tau : T_u \to T_v$ that is a one-to-one mapping of both nodes and edges from T_u to T_v and preserves flavors of the edges, i.e. $\forall a \in A_{T_u} : q_u(a) = q_v \circ \tau(a)$. When all edges have the same flavor, the initial definitions are recovered as a special case.

6.1.1 Lifting property by fibration symmetry: Construction of fibration base

So far, we have not provided a complete definition of a base B for any symmetry because Eqs. (S1, S2, S3) only give us information about N_B but not about the edges A_B. The edges for base fibrations and opfibrations are defined using *lifting property*. In the case of fibration symmetry, this *lifting property* is based on the structure of the in-neighborhoods of the graph G. The edges in the case of binary connections are selected as follows:

1. Calculate the partition C^{fib} (see colors in Fig. 1a).

2. Identify the in-neighborhood for all nodes of G.

3. Replace the node IDs in the in-neighborhoods with its color to obtain the in-neighborhoods of the nodes in the fibration base B_{fib}.

4. Connect the nodes of B_{fib} using the information from the in-neighborhoods. Note that each node in the base B can receive more than one edge from another (see base in Fig. 1a).

Multiple edges indicate that the input is now a multiple of a single-edge input. For networks with weighted edges, the weights are to be selected as explained in Section 2.6 of the main text. The addition of binary edges in step 4 above then takes the form of Eq. (10) derived in Section 6.9.1

For opfibration symmetry, the lifting property is analogous but differs in that it is based on the structure of the out-neighborhoods of the graph G. The lifting property ensures that the input (resp., output) tree of a node in the fibration (opfibration) base is isomorphic to the trees of the nodes in that fiber (opfiber).

For a covering symmetry – since it is simultaneously a fibration and an opfibration symmetry – the base can be constructed using either the fibration or the opfibration lifting property after calculating the partition C^{cov}. Both cases are shown in Fig. S5. In general, the results will differ. Therefore, one has to select either to follow a fibration compression (preserving the inputs, as we have done in Fig. 1e) or an opfibration compression.

In this work, we will use the construction of the cover base via the fibration lifting property, as it preserves the forward computation of the deep networks. This should not be confused with the proper fibration symmetry that compresses the fibers using the fibration map shown in Fig. 1a.

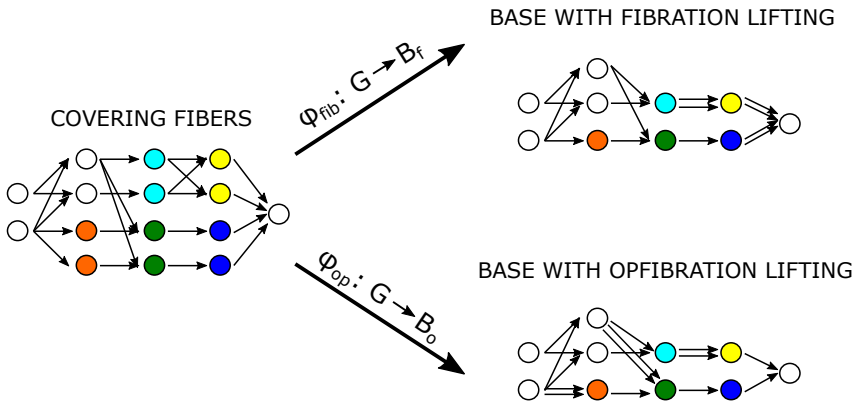

Figure S5: **Compression to the cover base:** One can preserve the input tree or output tree, but generally not both unless all covers have the same cardinality which is not satisfied in this case, since 5 covers have cardinality 2 and 5 covers have cardinality 1 (trivial). Note that our notation uses white for trivial covers. A more strict representation should assign a different color to every one of the trivial covers, but this representation would be harder to grasp with so many colors.

6.2 Deep neural networks and computational graphs

Here, we define the main network architectures considered in this study. The structure of MLP and CNN can be captured by a simple *graph*, which connects one node to the next with an edge (Fig. S6a and b). More generally, modern network architectures such as LSTM and Transformers can be described as *hypergraph* (Fig. S6c and d). In a hypergraph, two or more nodes interact before providing input to the next node. This interaction – often a product or a sum – can be represented as a "factor" (the circles in Fig. S6c and d).

Multilayer Perceptron (MLP)

A MLP with $N-1$ hidden layers is described by the following equations.

$$\begin{aligned} z^{(\ell)} &= W^{(\ell)} h^{(\ell-1)} + b^{(\ell)}, \\ h^{(\ell)} &= \sigma(z^{(\ell-1)}), \\ \hat{y} &= h^{(N)}, \end{aligned}$$

where $z_i^{(\ell)}$ and $h_i^{(\ell)}$ are the input and activity of the node $i = 1, ..., d_\ell$ in the layer $\ell = 1, ..., N$, respectively. The vector $b^{(\ell)} \in \mathbb{R}^{d_\ell}$ is called the bias of the layer ℓ and the matrix $W^{(\ell)} \in \mathbb{R}^{d_\ell \times d_{\ell-1}}$ is the weight matrix from layer $\ell - 1$ to layer ℓ. σ is a non-linear activation and $\hat{y}$ is the prediction of the network for input $x = h^(0)$.

Convolutional Neural Network (CNN)

A CNN with N convolutions (symbol $*$) and average pooling is described by,

$$\begin{aligned} \mathbf{H}^{(\ell)} &= \text{AvgPool}[(\sigma(\mathbf{H}^{(\ell-1)} * \mathbf{W}^{(\ell)} + \mathbf{b}^{(\ell)})], \\ \hat{y} &= \text{AvgPool}(\mathbf{H}^{(N)}) \in \mathbb{R}^{d_N} \end{aligned}$$

where $\mathbf{W}^{(\ell)} \in \mathbb{R}^{D \times D \times d_{\ell-1} \times d_\ell}$ is called kernel (size $D \times D$), $\mathbf{b}^{(\ell)} \in \mathbb{R}^{d_\ell}$ is the bias and $\mathbf{H}^{(\ell)} \in \mathbb{R}^{L \times L \times d_\ell}$ is the activity for all layers $\ell = 1, ..., N$. The dimension d_ℓ is the number of channels in each layer ℓ.

Recurrent Neural Networks (RNN) - LSTM

The LSTM is a particular form of a recurrent neural network (RNN). Although LSTMs have been replaced by Transformers, recurrence remains important because it can implement multiple processing stages in a more compact network, and because recurrence is the dominant architecture in biological neural network. The LSTM is shown as a hypergraph in Fig. S6c. The dynamic in time t is defined as:

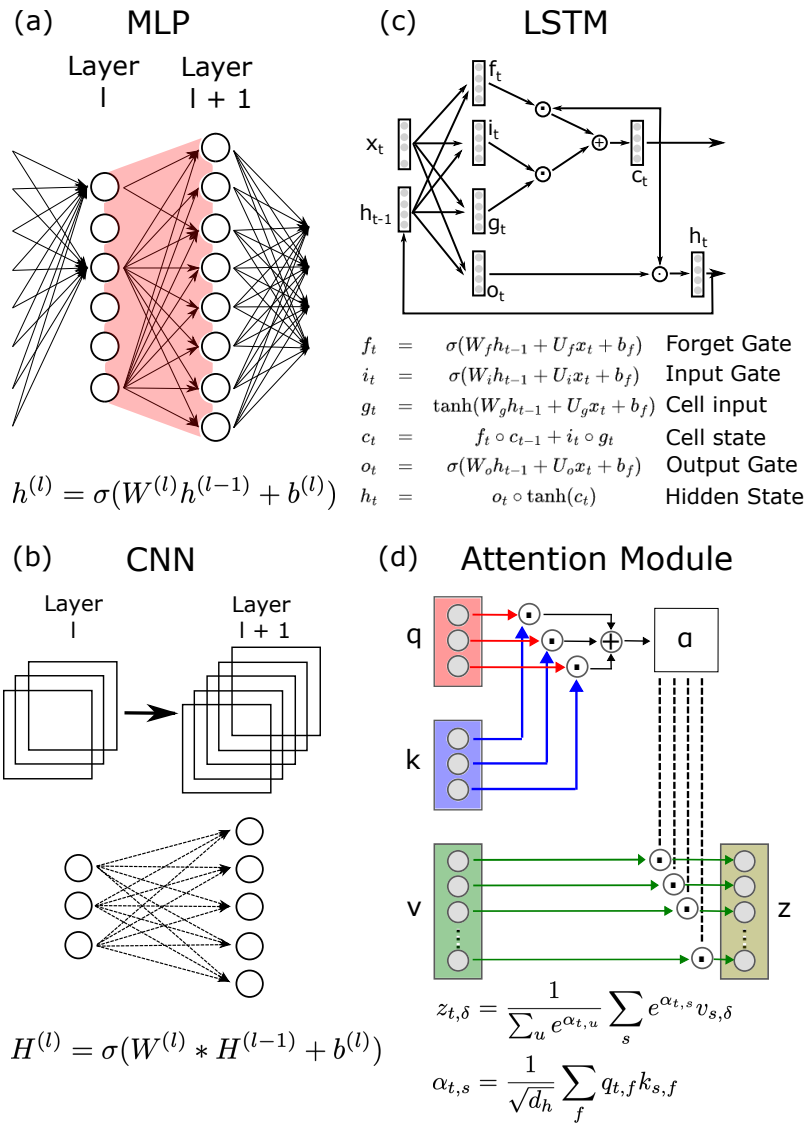

Figure S6: **Schematic representation of deep learning architectures as computational graphs and hypergraphs.** The figure illustrates the core topologies of: (a) MLP (Multilayer Perceptron), (b) CNN (Convolutional Neural Network), (c) LSTM (Long Short-Term Memory), and (d) Attention Module, along with their characteristic equations. In MLP and CNN, f is a non-linear activation. For LSTM, activations are represented with sigmoid and hyperbolic tangent. In all the cases, W, U and b are trainable parameters. The symbol $\circ$ and + represents the operations product and sum element-wise, resp. In Attention Module, α is called attention score calculated based on **q**uery and **k**ey. The output z is proportional to **v**alue.

$$
\begin{aligned}
\text{Input gate:} \quad & i_t = \sigma(W_{hi} h_{t-1} + W_{ii} x_t + b_i) \\
\text{Forget gate:} \quad & f_t = \sigma(W_{hf} h_{t-1} + W_{if} x_t + b_f) \\
\text{Output gate:} \quad & o_t = \sigma(W_{ho} h_{t-1} + W_{io} x_t + b_o) \\
\text{Cell gate:} \quad & g_t = \tanh(W_{hg} h_{t-1} + W_{ig} x_t + b_g) \\
\text{Cell state:} \quad & C_t = f_t \odot C_{t-1} + i_t \odot g_t \\
\text{Hidden state:} \quad & h_t = o_t \odot \tanh(C_t)
\end{aligned}
$$

where x_t is the input at time t, and $\odot$ element-wise multiplication. In an LSTM and Transformer the input is a sequence of values, and time t selects the elements in that sequence. Otherwise, in the rest of the text, t refers to a learning time step.

As with most RNN, the LSTM contains "gates" that multiply with the state of a node. For example, Cell State C_t takes as input the Cell gate g_t and input i_t combining them with $\odot$ — an element-wise multiplication: $C_t = f_t \odot C_{t-1} + i_t \odot g_t$. Let us consider the second term in this sum. It represents a multiplicative two-node interaction. For a fiber to emerge in this product, two nodes must be a fiber in the corresponding nodes of i_t **and** in a fiber of nodes of g_t, as exemplified with the colors in Fig. 2e. Two nodes in the product have the same colors, if they had the same colors in both factors. The same is true for the element-wise product of f_t and C_{t-1} — this color combination rule is the same *And operation* as we used for covers, but here it applies to the factors of the product. The sum can be treated as a concatenation of nodes $[f_t \odot C_{t-1}; i_t \odot g_t]$ with a weight matrix $[Id|Id]$ and the same conditions as in Eq. (3). This definition constitutes a general algorithm to identify fibers in any feedforward hypergraph composed of sums and products.

Attention modules and Transformers

Transformers are architectures that use the "attention" mechanism (see Fig. S6d) to process sequential data. The attention mechanism is defined by the following equations

$$\begin{aligned} \text{Query, Key, Value:} \quad & \mathbf{Q} = \mathbf{X}\mathbf{W}^Q, \quad \mathbf{K} = \mathbf{X}\mathbf{W}^K, \quad \mathbf{V} = \mathbf{X}\mathbf{W}^V \\ \text{Score:} \quad & \alpha = \frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}} \\ \text{Context vector:} \quad & \mathbf{z} = \text{softmax}(\alpha)\mathbf{V} \end{aligned}$$

where $\mathbf{X} \in \mathbf{R}^{T\times N}$ is the embedding of tokens (items in a sequence), $\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V$ are the trainable parameters, and d_k is the dimension of $\mathbf{K}$. It can be described as a sequence of weighted sums and multiplications similar to the gating mechanism in LSTMs. In total, all operations are a product or a weighted sum of nodes, which can be treated with the same two rules for weighted sums and element-wise products described above to identify fibers. The attention mechanism is typically followed by dense layers that operate over features for each time point (token) independently.

Note that the gating in the LSTM and the multiplication in the transformer can be treated the same for the coloring algorithms, but the two structures do differ for the fiber compression rule as shown in Fig. 2e-f, and explained in Section 6.9.3

The attention mechanism has both a feature dimension and a time (token) dimension. Here we have implemented a coloring algorithm to find fibration symmetries for the matrices $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ in the feature dimension, which can be propagated across all layers as before. The computation of symmetries for attention scores α requires defining symmetries across the token dimension while also incorporating the quadratic operation inherent in the feature dot product and will be part of future work.

Finally, Transformers and many other deep networks have "residual connections", which simply add the input to the output of a block of multiple layers. These "skip connections" combine entirely different colors (from different layers), in which case the addition operation simplifies to the same *And operation* as for the element-wise multiplication. If the input does not have large fibers, this *And operation* will tend to break up the fibers that have formed in the block.

6.3 Symmetries in weighted feedforward networks

In weighted feedforward networks (e.g. Multi-Layer Perceptrons), an automorphism symmetry between two nodes would require that they are in the same layer and that they have identical weight vectors at the input and output, so that permuting the two nodes keeps the entire network unchanged. Fibration and opfibration symmetries are less restrictive than automorphisms. As we shall see, they specify a weaker constraint on the weights.

In order to define fibers in weighted layered graphs, we first discuss isomorphic input trees. The input tree T_i of a node i in layer ℓ consists of all paths from the input layer to node i. For example, for a node i in the first layer ($\ell = 1$), its input tree is the vector

$$T_i^{(1)} = [W_{i1}^{(1)}, W_{i2}^{(1)}, ..., W_{id_0}^{(1)}].$$

Then, two nodes have isomorphic input trees if $T_i^{(1)} = T_j^{(1)}$. Using this criterion, we can define fibers in the first layer. For layer $\ell = 2$, the input tree $T_i^{(2)}$ of node i can be expressed as the concatenation of the input trees $T_m^{(1)}$ of nodes m in layer $\ell = 1$, and their corresponding connection weights $W_{im}^{(2)}$, i.e.

$$T_i^{(2)} = [T_1^{(1)}, W_{i1}^{(1)}, T_2^{(1)}, W_{i2}^{(1)}, ..., T_{d_1}^{(1)}, W_{id_1}^{(1)}].$$

If the fibers c in the first layer are already known, $T_i^{(2)}$ can be represented more compactly as the concatenation of the input trees of the fibers $T_c^{(1)}$ (because they are all the same within a fiber), but we do have to aggregate the weights as $\sum_{k\in c} W_{ic}$ to replicate their sum of effects in layer $\ell = 1$. Therefore, two nodes i and j of the layer $\ell = 2$ have isomorphic input trees, $T_i^{(2)} = T_j^{(2)}$, iff $\forall c : \sum_{k\in c} W_{jk} = \sum_{k\in c} W_{ik}$. This criterion therefore defines the fiber for layer $\ell = 2$. This argument can be extended to the next layers $\ell = 3, ..., N$ thus recursively defining fibers. This leads to the equal-sum criterion in the definition (3) of fiber in weighted graphs. Similar formulations have been used in the context of dynamical system synchronization (*51, 52*).

In order to define flavored opfibers, we will now discuss isomorphic flavored output trees. (They are referred to as colored weights in (*32*), but here we use the word “flavor" to not confuse with node colors). For a node i in layer $\ell = N - 1$, its output tree is the concatenation of the output weights w of i with its corresponding flavors $q(w)$; i.e.,

$$\hat{T}_i^{(N-1)} = [(W_{1i}^{(N)}, q(W_{1i}^{(N)})), (W_{2i}^{(N)}, q(W_{2i}^{(N)})), ..., (W_{d_N i}^{(N)}, q(W_{d_N i}^{(N)}))] \,.$$

We are interested in the case where the flavors of the edges W_{ki} depend only on i (i.e. $q(W_{ki}) = q(i)$). That means

$$\begin{aligned}\hat{T}_i^{(N-1)} &= [(W_{1i}^{(N)}, q(i)), (W_{2i}^{(N)}, q(i)), ..., (W_{d_N i}^{(N)}, q(i))] = \\ &= ([W_{1i}^{(N)}, W^{(N)}, ..., W_{d_N i}^{(N)}], q(i)) \ .\end{aligned}$$

Then, two nodes have isomorphic flavored output trees if $q(i) = q(j)$ and $\forall k : W_{ki}^{(N)} = W_{kj}^{(N)}$. Using this criterion, we can define flavored opfibers in layer $\ell = N - 1$. For layer $\ell = N - 2$, the output tree $\hat{T}_i^{(N-2)}$ of node i can be expressed as the concatenation of the output trees $\hat{T}_m^{(N-1)}$ of nodes m in layer $\ell = N - 1$, and their corresponding connection weights $W_{mi}^{(N-1)}$, i.e.,

$$\hat{T}_i^{(N-2)} = ([\hat{T}_1^{(N-1)}, W_{1i}^{(N-1)}, \hat{T}_2^{(N-1)}, W_{2i}^{(N-1)}, ..., \hat{T}_{d_{N-1}}^{(N-1)}, W_{d_{N-1}i}^{(N-1)}], q(i)).$$

If the opfibers $\hat{c}$ in layer $\ell = N-1$ are already known, $\hat{T}_i^{(N-2)}$ can be more compactly represented as the concatenation of the output trees of the opfibers $\hat{T}_{\hat{c}}^{(N-1)}$ (because they are all the same within an opfiber), but we do have to aggregate the weights $\sum_{k\in\hat{c}} W_{ki}$ to replicate their summed effect from layer $\ell = N-1$. Then, two nodes i and j of the layer $\ell = N-2$ have isomorphic output trees iff $\forall \hat{c} : \sum_{k\in\hat{c}} W_{kj} = \sum_{k\in\hat{c}} W_{ki}$ and $q(i) = q(j)$ to preserve flavors. This argument can be extended to the next layers $\ell = N-3, ..., 1$. This leads to the equal-sum criterion in definition (5) of opfiber in weighted graphs along with the flavor-preserving condition.

To capture the coupling of forward pass and backpropagation of Eq. 2, we flavor the weights with the coloring of the fibration symmetries (i.e., when $q(i) = c(i) \in C^{\text{fib}}$). This ensures that nodes in a flavored opfiber have the same σ', not just in the current layer but for the entire tree propagating the error from the output to each node of the network. When the network is linear, $q = 1$ for all edges and flavored opfibers simplify to regular opfibers, uncoupling forward from backward coloring.

6.4 Proof of synchronization in Eqs. (4) and (6)

We will prove both equations by induction. Suppose that the synchronization of activity holds for the layer $\ell - 1$. Let i, j be two nodes in layer ℓ such as $i \underset{\text{fib}}{\sim} j$, then:

$$
\begin{aligned}
h_i^{(\ell)} &= \sigma\left(\sum_k W_{ik}^{(\ell)} h_k^{(\ell-1)}\right) \\
&= \sigma\left(\sum_{c\in C_{\ell-1}^{\text{fib}}} \sum_{k\in c} W_{ik}^{(\ell)} h_k^{(\ell-1)}\right) \\
&= \sigma\left(\sum_{c\in C_{\ell-1}^{\text{fib}}} h_c^{(\ell-1)} \sum_{k\in c} W_{ik}^{(\ell)}\right) && \text{activity synchronization in } \ell-1 \\
&= \sigma\left(\sum_{c\in C_{\ell-1}^{\text{fib}}} h_c^{(\ell-1)} \sum_{k\in c} W_{jk}^{(\ell)}\right) && i \underset{\text{fib}}{\sim} j \\
&= h_j^{(\ell)}.
\end{aligned}
$$

Therefore, the synchronization holds for the layer ℓ. Now, let us prove the base case of the induction ($\ell = 1$).

$$
i \underset{\text{fib}}{\sim} j \implies W_{ik}^{(1)} = W_{jk}^{(1)} \implies \sigma\left(\sum_k W_{ik}^{(1)} x_k\right) = \sigma\left(\sum_k W_{jk}^{(1)} x_k\right) \implies h_i^{(1)} = h_j^{(1)}.
$$

With these two steps, we have proven Eq. (4). In Supplementary text 6.5, we show the *if and only if* for the linear case.

Now, let's prove Eq. (6), which is based on the backpropagation rule–Eq. (2). First, note that the non-linear case: $i \underset{\text{op}}{\sim} j \implies i \underset{\text{fib}}{\sim} j \implies \sigma'(h_i) = \sigma'(h_j)$. In the linear case, $\forall i : \sigma'(h_i) = 1$, but i, j are not necessarily in the same fiber. Suppose that the error is synchronized in layer $\ell + 1$. Let i, j be two nodes in layer ℓ such as $i \underset{\text{op}}{\sim} j$.

$$
\begin{aligned}
\delta_i^{(\ell)} &= \sigma'\left(h_i^{(\ell)}\right) \sum_k W_{ki}^{(\ell+1)} \delta_k^{(\ell+1)} \\
&= \sigma'\left(h_i^{(\ell)}\right) \sum_{c \in C_{\ell+1}^{\text{op}}} \sum_{k \in c} W_{ki}^{(\ell+1)} \delta_k^{(\ell+1)} \\
&= \sigma'\left(h_i^{(\ell)}\right) \sum_{c \in C_{\ell+1}^{\text{op}}} \delta_c^{(\ell+1)} \sum_{k \in c} W_{ki}^{(\ell+1)} \qquad \text{error synchronization in } \ell + 1 \\
&= \sigma'\left(h_j^{(\ell)}\right) \sum_{c \in C_{\ell+1}^{\text{op}}} \delta_c^{(\ell+1)} \sum_{k \in c} W_{kj}^{(\ell+1)} \qquad i \underset{\text{op}}{\sim} j \\
&= \delta_j^{(\ell)}
\end{aligned}
$$

Then, the error synchronization holds for layer ℓ. Now, let's prove the base case of the induction ($\ell = N - 1$).

$$
\begin{aligned}
i \underset{\text{op}}{\sim} j &\implies W_{ki}^{(N)} = W_{kj}^{(N)} \quad \& \quad \sigma'(h_i^{(N-1)}) = \sigma'(h_j^{(N-1)}) \\
&\implies \sum_k W_{ki}^{(N)} \delta_k = \sum_k W_{kj}^{(N)} \delta_k \quad \& \quad \sigma'(h_i^{(N-1)}) = \sigma'(h_j^{(N-1)}) \\
&\implies \delta_i^{(N-1)} = \delta_j^{(N-1)}.
\end{aligned}
$$

We have proven Eq. 6.

6.5 Equivalence between fibration and synchronization for linear activation

Here we show that synchronization of activity for all inputs implies fibration symmetry, provided node activations are linear, i.e. two nodes that are synchronized are in the same fiber. Because in the linear case, the error backpropagation is entirely analogous to the forward propagation of activity, we also conclude that error synchronization implies opfibration: two nodes with synchronized errors are in the same opfiber. Therefore, structure not only defines function, but conversely, function defines structure. This is an instance of the structure-function relation, which here appears only in the linear case where the forward pass is decoupled from the backward pass.

We will prove this by induction. First, for $\ell = 1$:

$$
\begin{aligned}
\forall x : h_i^{(1)} = h_j^{(1)} &\implies \forall x : [W^{(1)} x]_i = [W^{(1)} x]_j \\
&\implies W_{ik}^{(1)} = W_{jk}^{(1)} \\
&\implies i \underset{\text{fib}}{\sim} j.
\end{aligned}
$$

Now, suppose that this is valid for $\ell - 1$, then:

$$\begin{aligned}
\forall x : h_i^{(\ell)} = h_j^{(\ell)} &\implies \forall x : \sum_k W_{ik}^{(\ell)} h_k^{(\ell-1)} = \sum_k W_{jk}^{(\ell)} h_k^{(\ell-1)} \\
&\implies \forall x : \sum_{c \in C_\ell^{\text{fib}}} \hat{h}_c^{(\ell-1)} \sum_{k \in c} W_{ik}^{(\ell)} = \sum_{c \in C_\ell^{\text{fib}}} \hat{h}_c^{(\ell-1)} \sum_{k \in c} W_{jk}^{(\ell)} \qquad \text{Hyp of induction} \\
&\implies \sum_{k \in c} W_{ik}^{(\ell)} = \sum_{k \in c} W_{jk}^{(\ell)} \qquad \text{because is valid for all } \hat{h}_c \\
&\implies i \underset{\text{fib}}{\sim} j.
\end{aligned}$$

For opfibration symmetry and error synchronization, we have an analogous proof.

6.6 Gradient Descent Rule - Eq. (7)

For a dataset $\mathcal{D} = \{(x, y)\}$, we denote the network's input as $h^{(0)} = x$ or $\mathbf{H}^{(0)} = x$. The output of the network is a prediction $\hat{y}$ to the desired y. The error of this prediction is $L = \mathcal{L}(\hat{y}, y)$, where $\mathcal{L}$ is called *loss function*. The weights $W^{(\ell)}$ and $\mathbf{W}^{(\ell)}$ can then be adjusted based on corrections that minimize the error L. Using gradient descent, the change in the weight matrix/tensor is

$$W^{(\ell)}(t) = W^{(\ell)}(t-1) - \alpha \frac{\partial L}{\partial W^{(\ell)}}(t-1)$$

where α is called the learning rate. Calculating the derivative we find:

$$\begin{aligned}
\text{MLP} : &\frac{\partial L}{\partial W^{(\ell)}} = \delta^{(\ell)} \cdot \left(h^{(\ell-1)}\right)^T, \\
\text{CNN} : &\frac{\partial L}{\partial \mathbf{W}^{(\ell)}} = \delta^{(\ell)} * \mathbf{H}^{(\ell-1)}
\end{aligned}$$

where δ^ℓ is the error signal for the nodes in layer ℓ.

6.7 Proof of Synchronized Learning - Eq. (8)

Consider two nodes i and j in the layer $\ell - 1$ in the same fiber and two nodes m and n in the layer ℓ in the same opfiber (see red and green nodes in Fig. 2a). Using Eqs. (4)-(6), we obtain $h_i^{(\ell-1)} = h_j^{(\ell-1)}$ and $\delta_m^{(\ell)} = \delta_n^{(\ell)}$.

Then,

$$\begin{aligned}
\Delta W_{mi}^{(\ell)} = -\alpha \frac{\partial L}{\partial W_{mi}^{(\ell)}} &= -\alpha \delta_m^{(\ell)} h_i^{(\ell-1)} \\
&= -\alpha \delta_m^{(\ell)} h_j^{(\ell-1)} = \Delta W_{mj}^{(\ell)} \\
&= -\alpha \delta_n^{(\ell)} h_i^{(\ell-1)} = \Delta W_{ni}^{(\ell)} \\
&= -\alpha \delta_n^{(\ell)} h_j^{(\ell-1)} = \Delta W_{nj}^{(\ell)}
\end{aligned}$$

6.8 Proof of Theorem 2.1 Cover Coarse Graining Theorem

Here we will proof three stability rules used to proof the cover coarsening theorem. The opfiber stability rule states that nodes in the same cover (blue nodes in Fig. 2b) at learning time t will belong to the same opfiber after one GD learning step $t+1$, if their output nodes are on the same opfiber. The fiber stability rule states that nodes in the same cover at time t will belong to the same fiber the next time $t+1$, if their input nodes are on the same fiber. Both rules are combined in the cover stability rule, which states that nodes in the same covering at time t remain in the same covering in the subsequent time $t+1$.

Opfiber, Fiber and Cover Stability Rules.

During the training process, the network weights W, the symmetries, and partitions of the nodes C will depend on the time t.

During gradient descent training of an FNN, *learning synchronization* ensures that:

1. Nodes in the same cover $i \underset{cov,\ell,t}{\sim} j$ will belong to the same opfiber at the next learning times step $i \underset{op,\ell,t+1}{\sim} j$ if $C^{\text{op}}_{\ell+1}(t+1)$ is coarser than $C^{\text{op}}_{\ell+1}(t)$.

2. Nodes in the same cover $i \underset{cov,\ell,t}{\sim} j$ will belong to the same fiber at the next learning time step $i \underset{fib,\ell,t+1}{\sim} j$ if $C^{\text{fib}}_{\ell-1}(t+1)$ is coarser than $C^{\text{fib}}_{\ell-1}(t)$.

3. Nodes in the same cover $i \underset{cov,\ell,t}{\sim} j$ will belong to the same cover at the next learning time step $i \underset{cov,\ell,t+1}{\sim} j$ if $C^{\text{fib}}_{\ell-1}(t+1)$ is coarser than $C^{\text{fib}}_{\ell-1}(t)$ and $C^{\text{op}}_{\ell+1}(t+1)$ is coarser than $C^{\text{op}}_{\ell+1}(t)$.

Proof (1) Let $i \underset{cov,\ell,t}{\sim} j$ be nodes in layer ℓ. For learning time step $t+1$, for $\forall \hat{c} \in C^{\text{op}}_{\ell+1}(t+1)$:

$$\begin{aligned}\sum_{k\in\hat{c}} W^{(\ell+1)}_{ki}(t+1) &= \sum_{k\in\hat{c}} W^{(\ell+1)}_{ki}(t) + \Delta W^{(\ell+1)}_{ki}(t) \\ &= \sum_{k\in\hat{c}} W^{(\ell+1)}_{ki}(t) + \Delta W^{(\ell+1)}_{kj}(t).\end{aligned}$$

The last equality is valid due to $i \underset{fib,\ell,t}{\sim} j$ and activity synchronization. Now, note that

$$\begin{aligned}\sum_{k\in\hat{c}} W^{(\ell+1)}_{ki}(t) &= \sum_{r\in C^{\text{op}}_{\ell+1}(t)} \sum_{k\in\hat{c}\cap r} W^{(\ell+1)}_{ki}(t) \\ &= \sum_{r\in C^{\text{op}}_{\ell+1}(t)} \Theta(r\subset\hat{c}) \sum_{k\in r} W^{(\ell+1)}_{ki}(t) \qquad \textit{Hyp) } C^{\text{op}}_{\ell+1}(t+1) \textit{ is coarser than } C^{\text{op}}_{\ell+1}(t) \\ &= \sum_{r\in C^{\text{op}}_{\ell+1}(t)} \Theta(r\subset\hat{c}) \sum_{k\in r} W^{(\ell+1)}_{kj}(t) \qquad \textit{Hyp) } i \underset{op,\ell,t}{\sim} j \\ &= \sum_{k\in\hat{c}} W^{(\ell+1)}_{kj}(t).\end{aligned}$$

Then, we obtain $\sum_{k\in\hat{c}} W^{(\ell+1)}_{ki}(t+1) = \sum_{k\in\hat{c}} W^{(\ell+1)}_{kj}(t+1)$. That means $i \underset{op,\ell,t+1}{\sim} j$.

The proof of (2) is similar. The proof of (3) is the consequence of (1) and (2).

Proof of cover coarsening theorem

The cover coarsening theorem indicates nodes in the same cover $i \underset{cov,\ell,t}{\sim} j$ will belong to the same cover at the next time $i \underset{cov,\ell,t+1}{\sim} j$. The theorem relaxes the conditions of the cover stability rule.

For $\ell = 0$ and $\ell = N$, $C_0^{\text{fib}}(t) = \{[1], [2], ..., [d_0]\}$, and $C_N^{\text{op}}(t) = \{[1], [2], ..., [d_N]\}$ that remain constant over time t.

We apply fiber stability rule for layer $\ell = 1$) Nodes in the same cover $i \underset{cov,\ell=1,t}{\sim} j$ will belong to the same fiber at the next time $i \underset{fib,\ell=1,t+1}{\sim} j$ because $C_0^{\text{fib}}(t+1)$ is coarser than $C_0^{\text{fib}}(t)$. This means $C_1^{\text{fib}}(t+1)$ is coarser than $C_1^{\text{fib}}(t)$.

With the same argument, we apply the fiber stability rule for layer $\ell = 2$, and we obtain $C_2^{\text{fib}}(t+1)$ is coarser than $C_2^{\text{fib}}(t)$.

By iterating over all layers, we obtain $\forall \ell : i \underset{cov,\ell,t}{\sim} j \implies i \underset{fib,\ell,t+1}{\sim} j$.

Applying the same reasoning with the opfiber stability rule from layer $\ell = N$, then $\forall \ell : i \underset{cov,\ell,t}{\sim} j \implies i \underset{op,\ell,t+1}{\sim} j$.

Using both results, we obtain: $\forall \ell : i \underset{cov,\ell,t}{\sim} j \implies i \underset{cov,\ell,t+1}{\sim} j$.

The nodes in the same cover at learning time step t, will belong to the same cover in the next time step. Equivalently, $C_\ell^{\text{cov}}(t+1)$ is coarser than $C_\ell^{\text{cov}}(t)$.

6.9 Fibration compression rules

Now we want to derive the fibration compression rule for the parameters of a base network B_{fib}, so that the activity of the nodes remains unchanged, i.e., the base network maintains forward computation of the original network G. We will do this first for the MLP, and then discuss compression for other architectures.

6.9.1 Fibration compression for a feedforward network

Let $\hat{h}_{c'}$ be the activity of a fiber $c' \in C_\ell^{\text{fib}}$, then it follows that:

$$\begin{aligned}
\hat{h}_{c'}^{(\ell)} &= h_i^{(\ell)} \qquad \forall i \in c' \qquad \text{(activity synchronization)} \\
&= \frac{1}{|c'|} \sum_{i \in c'} h_i^{(\ell)} \\
&= \frac{1}{|c'|} \sum_{i \in c'} \left[W_{ik}^{(\ell)} h_k^{(\ell)} + b_i^{(\ell)} \right] \\
&= \frac{1}{|c'|} \sum_{i \in c'} \sum_{c \in C_{\ell-1}^{\text{fib}}} \sum_{k \in c} W_{ik}^{(\ell)} h_k^{(\ell)} + \sum_{i \in c'} b_i^{(\ell)} \\
&= \sum_{c \in C_{\ell-1}^{\text{fib}}} \left[\frac{1}{|c'|} \sum_{i \in c'} \sum_{k \in c} W_{ik}^{(\ell)} \right] \hat{h}_c^{(\ell)} + \left[\sum_{i \in c'} b_i^{(\ell)} \right] \\
&= \sum_{c \in C_{\ell-1}^{\text{fib}}} \hat{W}_{c'c} \hat{h}_c^{(\ell)} + \hat{b}_c^{(\ell)} ,
\end{aligned} \tag{S5}$$

where $\hat{W}_{c'c} = \frac{1}{|c'|} \sum_{i \in c'} \sum_{k \in c} W_{ik}^{(\ell)}$ and $\hat{b}_{c'}^{(\ell)} = \sum_{i \in c'} b_i^{(\ell)}$. These matrices $\hat{W}_{c'c}$ and vectors $\hat{b}_{c'}^{(\ell)}$ are the parameters of the base B_{fib}.

A similar equation can be derived for a base network B_{op} whose nodes are the opfibers of the original network G and the backward computation of the error is preserved.

Then compression to a covering base can be chosen to preserve the input or output trees. This is represented pictorially for the same covering base in a network with binary weights in Fig S5. Compressing a graph into its covering base where both forward and backward computations are preserved is only possible when all covers have the same cardinality $|c'|$. This is too strong of a requirement and has not been explored in more detail here.

6.9.2 Fibration compression for a convolutional layer in an CNN

A similar line of reasoning works for CNNs where the nodes of the computational graph are the channels in the convolutions. The definition of fibers and lifting for MLPs (Eqs. 3, S5) can be generalized to CNNs. These equations are applied to the channels i, k in a layer ℓ; where $W_{ik}^{(\ell)}$ is a D^2-dimensional vector that represents the kernel between the channels i and j.

More precisely,

$$\begin{aligned}
\hat{\mathbf{W}}[:, :, c, c'] &= \frac{1}{|c'|} \sum_{i \in c'} \sum_{k \in c} \mathbf{W}^{(\ell)}[:, :, k, i], \\
\hat{\mathbf{b}}_{c'}^{(\ell)} &= \sum_{i \in c'} \mathbf{b}_i^{(\ell)}.
\end{aligned} \tag{S6}$$

6.9.3 Fibration compression for an Hypergraph

Let us generalize the concepts of fibrations and lifting to any computational hypergraphs, where two or more nodes interact before acting on the subsequent node. The most common interaction is

a multiplicative modulation or "gating" of activity.

Suppose that there are two triplets of nodes (p, i, j) and (p', i', j') such as $h_p = h_i.h_n$ and $h_{p'} = h_{i'}.h_{n'}$ where h is the activity of the node. We will say $p \sim p'$ if and only if $i \sim i'$ and $j \sim j'$. Note $p \sim p' \implies h_p = h_{p'}$.

Many times, this modulation appears between node layers $z = x \odot y$, where $x, y, z \in \mathbb{R}^d$ (product between forget and input gate in LSTM or product between key and query in the attention mechanism). The activity in z is $z_i = x_i y_i = \hat{x}_c \hat{y}_{c'}$ when i is in the fiber c of the layer x and i is in the fiber c' of the layer y.

- **LSTM**. An important property of fibration in the gates in an LSTM is that they have the same fiber distribution (see the forget gate and the input gate in Fig. 2e). This implies that all layers ($x, y, z \in \mathbb{R}^d$) will be compressed to the same number ($\hat{x}, \hat{y}, \hat{z} \in \mathbb{R}^{d'}$) and the dynamics is $\hat{z}_c = \hat{x}_c \hat{y}_c$ where c is a fiber. Therefore, if nodes are in the same fiber for the forget gate (see red nodes in Fig. 2e), their corresponding nodes in the input gate (see blue nodes), output gate, cell input and hidden state, will also be in the same fiber. Consequently, the base has an identical number of nodes across all gates (2 nodes per gate in Fig. 2e).

- **Attention Mechanism**. Within an attention module of Transformers, the number of fibers may vary for the query (q), key (k), and value (V) projections (see Fig. 2f). Then we can reduce the number of nodes in q and k from d to d'_q and d'_k, respectively; but we cannot reduce the number of nodes in $z = q \odot k$, i.e. fibration compression preserves the number of interactions (see Fig. 2f).

6.10 Datasets and tasks used in empirical tests

6.10.1 MNIST digit classification using a multilayer perceptron

We trained a MLP with a 784-dimensional input layer, three hidden layers of 500 nodes each, and a 10-node output layer. The model was trained to classify the 10 digit classes of the MNIST dataset (28 × 28 binary images). The MNIST dataset contains 10 digit classes (0–9), with a nearly uniform distribution across categories. It is split into a training set of 60,000 images and a test set of 10,000 images. Training was performed using mini-batch gradient descent on Cross Entropy Loss with a batch size of 100 and the Adam optimizer with a learning rate of 0.001. Ten independent training runs were conducted. The average training accuracy over 600 epochs is shown in Fig. 3a, where an epoch is a full pass through the entire training data.

6.10.2 ImageNet classification using a CNN

We trained a CNN with two convolutional layers (32 5x5-kernels and 64 3x3-filters) and two dense layers of 128 nodes each. The model was trained to classify images in the ImageNet dataset (32 x 32 RGB images). ImageNet dataset contains 1,000 classes, each of 700 images (600 for the training set, 100 for the test set). Training was performed using mini-batch gradient descent with the Cross Entropy Loss with a batch size of 100 and the SGD optimizer with a learning rate of 0.001 and momentum 0.9. Ten independent training runs were conducted. The average training accuracy over 300 epochs is shown in Fig. 3c.

6.10.3 Pruning null models

Pruning is a widely used model compression technique aimed at reducing a neural network's size and computational footprint by removing redundant or less important parameters (*38*). Traditionally, pruning algorithms often rely on parameter magnitude as a proxy for importance, under the assumption that smaller weights contribute less to the network's output.

To contextualize the efficiency of our fibration compression, we compare it against two standard pruning baselines: random pruning and L2-norm-based pruning. For a fair comparison, all methods are constrained to produce a compressed network with the same number of nodes as the fibration base.

1. Random pruning serves as a naive baseline, where nodes are removed uniformly at random, independent of their structural or functional role in the network.

2. L2-norm pruning employs a more structured criterion: it removes the nodes with the smallest input-weight L2 norm, based on the heuristic that nodes with weaker incoming connections are less critical.

In contrast, our fibration compression identifies and merges nodes that play identical functional roles, resulting in a compressed network that preserves the forward computation.

6.10.4 Training Atari - Beam Rider with RL

We investigated a typical use of LSTMs in Reinforcement Learning (RL) by training an agent on the Atari Beam Rider game using the Proximal Policy Optimization (PPO) algorithm (*53*). The agent is a deep neural network consisting of a sequential stack of three Conv2d layers, a linear layer, and a LSTM. From the LSTM output, the network branches into two heads: an Actor head, which transforms the output into a probability distribution over actions using a linear projection and Softmax activation; and a Critic head, which estimates the state value via a simple linear projection.

The agent interacts with the environment in a continuous cycle. The actor samples and executes an action; the environment responds with a reward and a new state; and the critic assesses the outcome to calculate a temporal difference error. This error guides the update of both networks, pushing the Actor to refine its policy toward more rewarding actions and helping the Critic improve the accuracy of its value predictions.

For training, we use the hyperparameters shown in Table S1. More details are presented in our Github.

Total Agent Time Steps	9216 K
Batch size	30720
Minibatch size	1024
Environment	Atari-Beam Rider
Num of Environment	65
BPTT Horizon	2
Learning rate	4e-4

Table S1: Training hyperparameters on RL.

6.11 Formation of fibers during training

The formation of fibers (as a result of cover formation) is shown in Fig. S7. This shows results for the same CNN structure and MNIST training data as in Fig. 3a and c. The prevalence of large fibers in later layers of the network is explained in our theory by the increasingly more relaxed conditions for the fibers: Each layer removes an additional degree of freedom whenever two nodes satisfy the fiber equation (3). As we move across layers, the subspace of solutions increases and the fibers grow in size. This is paralleled by the observed node synchronization (Fig. S10), which is more prevalent in later layers of the network. This observation is consistent with the phenomenon of "node collapse" previously reported in the literature (*25, 26*). Our theory explains this as the result of the cover symmetry formation during SGD together with the fiber synchronization theorem.

6.12 Refinement algorithm for balanced colorings in feedforward networks

A *coloring* c of a graph G is a map $c : N_G \to C$ where $c(u)$ is called the color of the node u and C is called the set of *colors*. The coloring c is *in-balanced* if $c(u) = c(v)$ implies that T_u and T_v are color isomorphic. The minimal in-balanced coloring is a in-balanced coloring of a graph with the minimal number of colors. In terms of synchronization, nodes inside the same subset of the in-balanced coloring partition (i.e. nodes with the same color) can synchronize its activities, since they receive the same color inputs from the same synchronized nodes.

A refinement algorithm of G starts with an initial coloring c_0 and produces a new coloring c_t in each iteration t, based on some criteria, until a desired property is achieved (e.g. in-balanced coloring).

In a graph without weighted edges and without labeled nodes, the method can be summarized as follows.

1. c_0 assigns a trivial color to each vertex v (e.g. $c_0(v)$=1).
2. $c_{i+1}(v) = (c_i(v), \{\{c_i(w) \mid w \text{ is a neighbor of } v\}\})$

At some point, it stabilizes $c_{t+1}(u) = c_t(u)\ \forall u \in N_G, t > T$. We designed a new method for a layered feedforward network. First, let us reformulate Eq. (3)

$$
\begin{aligned}
i \underset{\text{fib}}{\sim} j \iff \forall r : \sum_{k|c(k)=r} W^{(\ell)}_{jk} = \sum_{k|c(k)=r} W^{(\ell)}_{ik} &\iff \forall r : \hat{W}^{(\ell)}_{ir} = \hat{W}^{(\ell)}_{jr} \\
&\iff \hat{W}^{(\ell)}_{i,:} = \hat{W}^{(\ell)}_{j,:} \\
&\iff d(\hat{W}^{(\ell)}_{i,:}, \hat{W}^{(\ell)}_{j,:}) = 0 \\
&\iff 1 - \hat{W}^{(\ell)}_{i,:}.\hat{W}^{(\ell)}_{j,:} = 0 \qquad \text{if } \hat{W} \text{ is normalized}
\end{aligned}
\tag{S7}
$$

where $\hat{W}^{(\ell)}_{ir} = \sum_{k|c(k)=r} W^{(\ell)}_{ik}$.

Given that these conditions are impossible to satisfy exactly for continuous weights, we transform Eq. S7 into a set of inequality constraints parameterized by a tolerance ε. More precisely, we define the distance matrix $D^{(\ell)}_W = 1 - \hat{W}^{(\ell)}\hat{W}^{(\ell)T}$ and we use it to calculate the colors/clusters ($i \underset{\text{fib}}{\sim} j$) via agglomerative clustering with distance threshold ε:

1. $c^{(\ell=0)} \in C_0^{\text{fib}}$ assigns unique colors to each of the input features ($\ell = 0$).

2. For $\ell = 1, ..., N$:

   2.1. Calculation of $\hat{W}_{ir}^{(\ell)} = \sum_{k|c(k)=r} W_{ik}^{(\ell)} \; \forall r \in C^{(\ell)}$

   2.2. Normalization of $\hat{W}^{(\ell)}$.

   2.3. Calculation of the matrix $D_W^{(\ell)} = I - \hat{W}^{(\ell)} \hat{W}^{(\ell)T}$

   2.4. Agglomerative Clustering of the nodes in ℓ based on the distance matrix $D_W^{(\ell)}$ (see Alg. 1). We use a distance threshold $\varepsilon \in (0, 2)$.

   2.5. Fibers $c^{(\ell)} \in C_\ell^{\text{fib}}$ are defined as the clusters.

Algorithm 1 Agglomerative Clustering

Initialize clusters $C = 1, 2, ..., n$
while $|C| > 1$ **do**
    $\forall c, c' \in C : d_{\text{complete}}(c, c') = \max_{i \in c, j \in c'} D_W(i, j)$
    $d_{min} = \min_{c,c' \in C} d_{\text{complete}}(c, c')$
    If $d_{min} > \varepsilon$: stop
    $C \leftarrow$ Merge the clusters whose distance is less than d_{min}.
end while
return C

Note that agglomerative clustering employs a distance threshold ε to determine cluster merging (see Alg. 1). According to Eq. (S7), this is equivalent to treating the equalities $\sum_{k \in c} W_{ik}^{(\ell)} = \sum_{k \in c} W_{jk}^{(\ell)}$ with a tolerance parameter ε, which defines the tolerance to a weak fibration breaking (see Step 2.4). In other words, all approximate fibration symmetries should be regarded as quasi-fibrations (*37*) or pseudo-balanced colorings (*52*).

The coloring generated in the feedforward pass generates C_ℓ^{fib} for all layers. The same algorithm run backwards starting at the output generates C_ℓ^{op}. The number of operations to execute this coloring algorithm in a layered feed-forward graph scales linearly with the parameter size in each layer and linearly with the number of layers N.

For recurrent networks, the fibration and opfibration definitions apply similarly, except that partitioning C_ℓ^{fib} and C_ℓ^{op} must be iterated recursively forward and backward, respectively, until convergence. Convergence requires, at most, the times of the longest loop in the recurrence (*54*).

The code is available at `github.com/MakseLab/fibrations_in_dnns`.

6.13 Sequence-to-Sequence Transformers

In order to study the application of our fibration symmetry in Transformer models with attention mechanism, we turn to a sequence-to-sequence translation model for a German-to-English translation task using the Multi30k dataset (*55*). This dataset consists of 31,014 sentences split into 29k

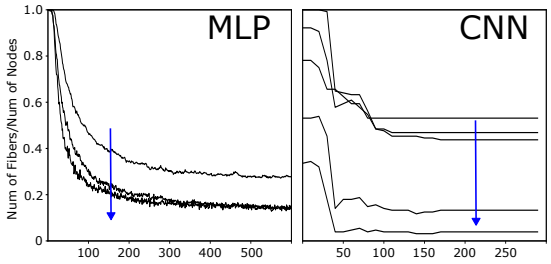

Figure S7: **Number of fibers in distinct layers across epochs:** As nodes group into fibers the number of distinct fibers decreases and they grow in size (individual nodes count as trivial fiber). This is the same as in Fig. 3a and c, but here each curve represents a separate layers. Higher layers in the networks (blue arrow) cluster into fibers more quickly and have a coarser partition at the end of learning (fewer larger fibers).

Number of encoder layers	3
Number of decoder layers	3
Number of attention heads	8
Source vocabulary size	10837
Target vocabulary size	19214
Batch size	128
Learning rate	1e-4
Adam Optimizer β_1	0.9
Adam Optimizer β_2	0.98
Adam Optimizer ϵ	1e-9
Early stopping patience	20

Table S2: Sequence-to-Sequence transformer model and training hyperparameters

training, 1k test, and 1k validation sets. For the model, we use a vanilla sequence-to-sequence transformer model as introduced in (*56*) with the Adam optimizer (*57*) and early stopping regularization. The detailed specifications of the model and the training process are given in Table S2.

We keep the dimension of the token embeddings (d_{emb}) and the size of the feedforward layer equal and use it to control the number of parameters in the model, similar to (*1*). Then, for each model size, we follow the procedure explained in Algorithm 2 to compress the trained model and iteratively retune. For each layer, we select a compression threshold $\varepsilon \in (0, 2)$ (as defined in 6.12) using a greedy algorithm explained in Section 6.14.

This Algorithm starts training with a vanilla Transformer model with early stopping. We then find a set of fiber precision $\{\varepsilon\}$ with the greedy Algorithm 3 described below. We then attempt to overcompress with somewhat larger thresholds (quenching rate $q > 1$) and regain performance by retuning with early stopping. If this regained performance is within tolerance ρ we attempt further over-compression. Otherwise, we compress with the last effective attempt and return this compressed network. We attempted q between 1.01-1.1 and found good performance with a fixed $q = 1.05$. Figure S8 shows this iterative process of compressing and retuning the model for a model with 50 million initial non-embedding parameters.

Algorithm 2 Compress-Retune Procedure for Transformer Model

Initialize model $\mathcal{M}$ and weights
Train $\mathcal{M}$ until convergence with early stopping
$L \leftarrow \mathcal{L}(\mathcal{M}, \text{test set})$ ▷ $\mathcal{L}$ is the loss function evaluated on the test set
$L_c \leftarrow L$
while $L_c \leq (1+\rho) \times L$ **do** ▷ Attempt over-compression with loss tolerance ρ
    $\mathcal{M}_o \leftarrow \mathcal{M}$
    $\{\varepsilon\} \leftarrow$ The set of compression thresholds for $\mathcal{M}$ using the greedy Algorithm 3
    $\mathcal{M} \leftarrow$ Compressed $\mathcal{M}$ using $\{\varepsilon \times q\}$ ▷ over-compress with quenching rate $q > 1$
    $\mathcal{M} \leftarrow$ Retuned $\mathcal{M}$ with with early stopping
    $L_c \leftarrow \mathcal{L}(\mathcal{M}, \text{test set})$
end while
$\{\varepsilon\} \leftarrow$ The set of compression thresholds for $\mathcal{M}_o$ using the greedy Algorithm 3
$\mathcal{M}_o \leftarrow$ Compressed $\mathcal{M}_o$ using $\{\varepsilon\}$
return $\mathcal{M}_o$

6.14 The greedy algorithm for calculating fiber precision

To find a set of compression thresholds that minimizes the number of parameters while preserving the loss, the algorithm selects a different fibration tolerance threshold ε for the weight matrix of each layer in the Transformer model, forming the set $\{\varepsilon\}$. For this, we employ a greedy sequential strategy in which the compression threshold ϵ is maximized using binary search for each layer individually, so as not to increase the loss by more than a fraction τ (Here we used $\tau = 0.001$). The complete algorithm is detailed in Algorithm 3.

Algorithm 3 The greedy algorithm for optimal compression thresholds

Require: Transformer Model M, Test Dataset $\mathcal{D}$, Tolerance τ
Ensure: Set of compression thresholds $\{\varepsilon\}$
$L_{base} \leftarrow \mathcal{L}(\mathcal{M}, \mathcal{D})$
$L_{limit} \leftarrow L_{base} \cdot (1+\tau)$
$\{\varepsilon\} \leftarrow \emptyset$
for each layer ℓ in $\mathcal{M}$ **do**
    $W \leftarrow \text{GetWeights}(\ell)$
    via Binary Search, find $\epsilon^* = \max \epsilon \ (\epsilon \in (0, 2))$ **subject to**:
        Condition 1: $W' = \text{Compress}(W, \epsilon)$
        Condition 2: $\mathcal{L}(\mathcal{M}|_{W \leftarrow W'}, \mathcal{D}) \leq L_{limit}$
    $\text{SetWeights}(\mathcal{M}, l, \text{Compress}(W, \epsilon^*))$ ▷ Permanently update $\mathcal{M}$
    $\{\varepsilon\} \leftarrow \{\varepsilon\} \cup \epsilon^*$
end for
return $\{\varepsilon\}$

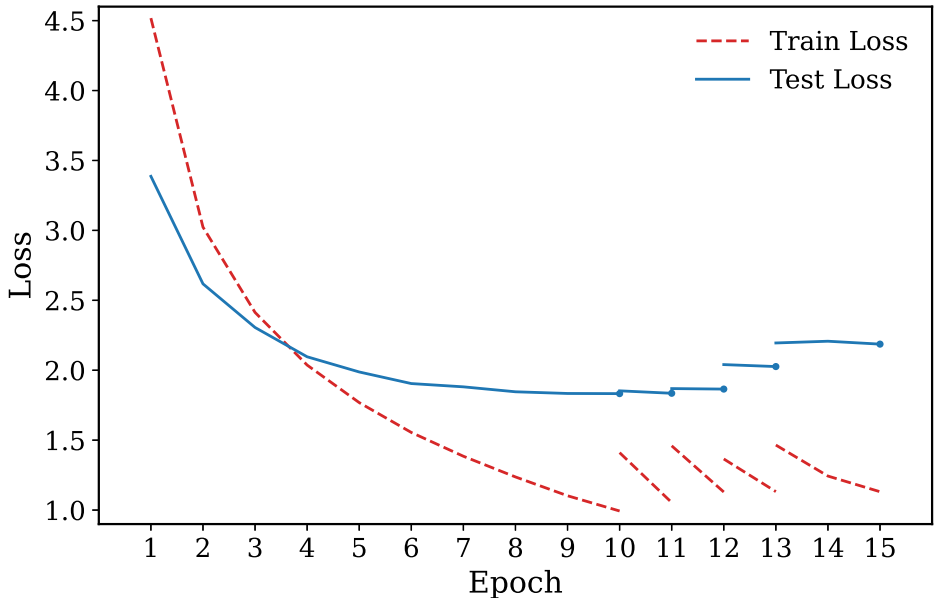

Figure S8: **Compress-retune procedure for a Transformer model.** Starting with a model with 50 million non-embedding parameters, on every iteration, we train the model with early stopping, then at each endpoint (shown with circles), we over-compress the model using larger compression thresholds, and attempt retuning to recover performance. Each blue point marks the model after retuning (line 6 in Algorithm 2. The numbers above the point, indicate the number of parameters of this model after one additional loss-preserving compression with thresholds $\{\varepsilon\}$ (from line 7 of Algorithm 2). These loss values and number of parameters are reported in the main text and Fig. 3g, so as to document the entire compression-retuning history.

6.15 Scaling law fit

The power law following Kaplan is established by training the baseline network with varying parameter sizes P. The loss $\mathcal{L}$ is recorded at the end of training. The scaling law Eq. (11) is fitted with a linear function between $\log(\mathcal{L})$ and $\log(P)$ (dashed black line in Fig. 3h), resulting in coefficients $L_0^k = 3.0$ (from the offset) and $\alpha_k = 0.028$ (from the slope).

The power law for the compressed network is measured by starting with a given P-sized network, then compressing and retuning as described above. For each P this gives a tuning curve (orange curves on Fig. 3h). Then, for a given loss value L, we find the smallest network (at all points on the orange curves and at all curves). This gives $P_{min|L}$. For this set of loss values in the range of 1.9-2.2 we fit a line as before (orange dashed line, in Fig. 3h) resulting in $L_0^f = 3.8$ (from the offset) and $\alpha_f = 0.049$.

6.16 Continual learning

We explore the fibration symmetry breaking algorithm to resolve the problem of loss of plasticity encountered in sequential learning scenarios. Our investigation focuses on the Continual ImageNet

task, comprising 5,000 binary classification tasks formed by pairing distinct classes from ImageNet. A deep neural network is trained on images from two classes per task and evaluated on a corresponding test set. ImageNet, with 1,000 classes and approximately 700 images per class, provides the dataset. Each class is divided into 600 training images and 100 test images. Tasks are presented sequentially, with a new task sampling a new pair of classes. Training occurs over 250 epochs per task. Performance, assessed by the accuracy of the test set, highlights a loss of plasticity if the accuracy decreases.

We utilize a CNN with three convolutional layers and three fully connected layers, with the final layer consisting of two nodes for the current task's classes. The output layer is reinitialized for each new task, following common practice. We evaluate standard gradient descent, deep continual learning, and symmetry-breaking gradient descent, all minimizing cross-entropy loss. Each method undergoes 10 independent runs with the same sequence of class pairs, and the network weights are initialized once before the first task.

The results, illustrated in Fig. 3f, are representative of a feed-forward convolutional network using unmodified backpropagation. Initially, networks achieved up to 88% accuracy on early tasks but lost plasticity by the 2,000th task. This pattern was consistent across architectures, parameters, and optimizers, indicating standard deep-learning methods struggle with continual-learning tasks.

The emergence of covers during learning is exemplified in Fig. S9 for the sequential ImageNet task. At the start (Epoch 0) all 512 nodes in the layer constitute trivial fibers (of size 1). After some training epochs, large covers emerge. Continual backpropagation and even more so symmetry breaking keep the cover size in check and therefore have a larger number of them, across the 250 epochs of the first task (first row) and even more clearly after 900 tasks (second row).

To decide when and how many nodes to reinitialize, we have to select a particular schedule. For easier comparison, we use the same re-initialization schedule as in (*3*). Namely, nodes are re-initialized at a fixed replacement rate (which is the number of node per gradient step), taken from covers only if they reach a certain size maturity threshold. We trained the network with a momentum term. The results in Fig. 3f are for a learning rate of 0.01, a replacement rate of 3e-4, a maturity threshold of 100 and a decay rate of 0.99. Minibatch sizes of 100, a momentum of 0.9.

6.17 Fibers are a refinement of clustering partitions

For each class c, we define $P_c^{(\ell)}$ as the partition obtained by activity synchronization due to class c. Each set $A \in P_c^{(\ell)}$ is a cluster of nodes in layer ℓ where the activity is synchronized for all inputs in class c (i.e. $h_i(x) = h_j(x) \quad \forall x \in c$). Examples are shown in Fig. S10.

Assume $N_{classes}$ number of classes and compute the intersection of clusters from different classes

$$I = A_1 \cap A_2 \cap \ldots \cap A_{N_{classes}}$$

Two nodes in I are synchronized for all inputs in the data set; however, they may not be synchronized for an input that is not part of the data set. That is, a fiber is always contained within a set like I. In other words, the partition formed by the fibers is a refinement of the partition formed by sets of the form I (i.e. intersection of clusters across classes in Fig. 4b).

To test this theory numerically, we threshold the correlation coefficients for the synchrony clusters as well as the fibration condition - Eq. (3), and then measure a refinement score between these two partitions of nodes (Fig. S11b). The **refinement score**, counts the number of fibers

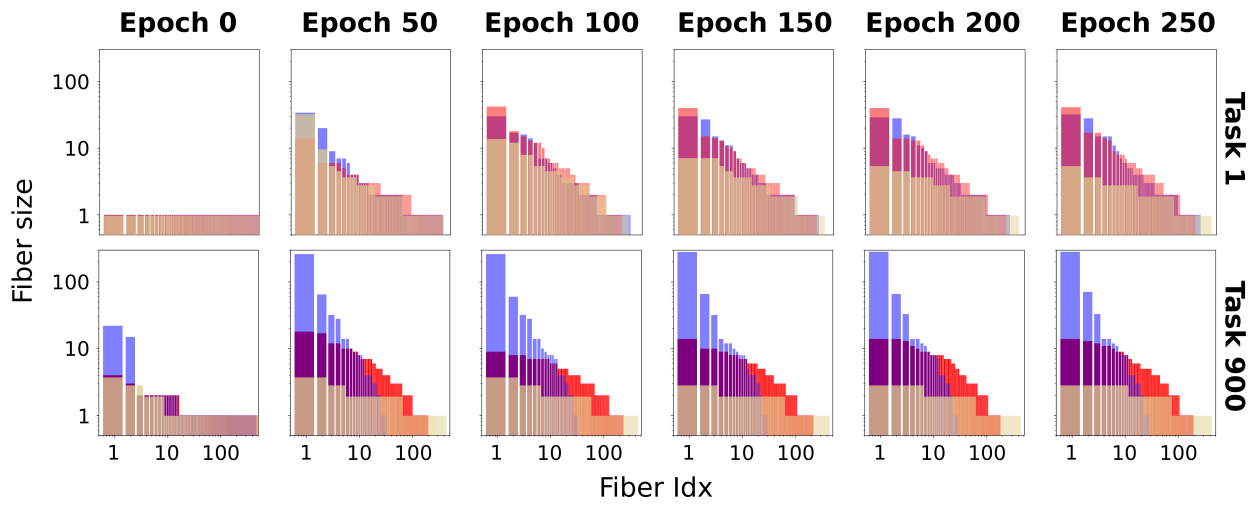

Figure S9: **Distribution of covers, fibers and opfibers during continual learning on the Sequential ImageNet Task.** Evolution of the size and number of covers across leaning epochs (columns) and task number (rows) for different training methods: (blue) SGD (red) continual backpropagation (yellow) fibration symmetry breaking. This distribution of sizes for the second dense layer.

that have a unique cluster label normalized by the total number of fibers. For small thresholds, the theoretical result is validated (refinement score=1) while for larger cluster thresholds the relationship breaks down (refinement score=0), as expected. This observation is more generally true for any subset of all possible inputs, including inputs generated at random.

The synchronization theorem states that fibers imply synchronization for any arbitrary input, but the reverse is not necessarily true. We test the reverse of this statement empirically by comparing the fiber partition with synchronization clusters for random inputs. We measure how well the two partitions match using as **matching score**. For this we use the "clustering accuracy", which is the maximized sum of the diagonal of the confusion matrix divided by the total number of items. The maximum assignment of partition labels is found with the linear sum assignment algorithm. For small thresholds, the fiber partition and synchronization clusters are a perfect match (matching score=1, Fig. S11a), showing that fibers can also be identified simply by correlating nodes under random inputs.

6.18 Lifting operation for weighted network

The origin of the name "lifting" is the inverse operation of compression in Fig.1, where the base is "lifted" to a full graph. Since compression is not an injective function, the lifting is not unique. For example, the following weights W (full graph) are transformed into the same weights $\tilde{W}$ (base),

$$W_{ij}^{(\ell)} = \frac{1}{|c|}\hat{W}_{c'c}^{(\ell)} + \Delta_{ij} \tag{S8}$$

where $\sum_{\substack{k\in c'\\ m\in c}} \Delta_{km} = 0$ and $i \in c'$, $j \in c$.

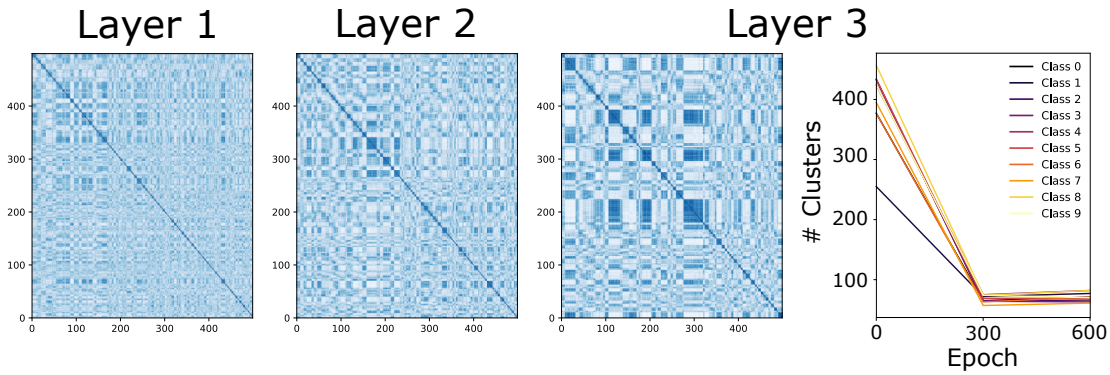

Figure S10: Synchrony clusters for class 9 in multiples layers of the MLP trained on MNIST. On the right, we show the number of clusters for layer 3 as function of learning epochs for all 10 classes.

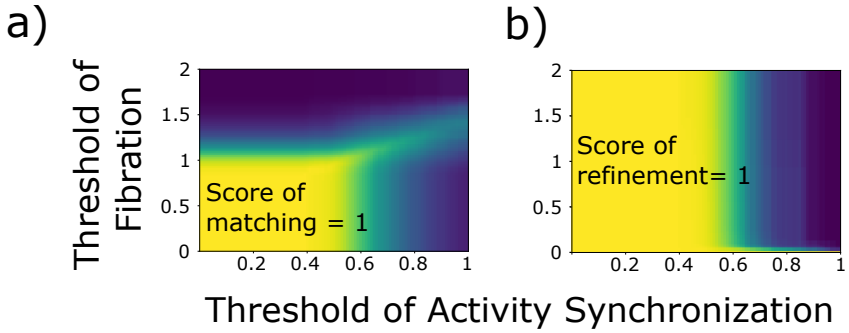

Figure S11: Matching score and refinement score compare the these clusters with fibers for different thresholds of identifying the clusters (horizontal) and fibers (vertical)

In Fig. S12, we compress using Eq. (10) from G_1 to the base B_1'; then, B_1' to the minimal base B. B is a base of G_1 that has two fibers (blue and red). In B_1', two nodes of the blue fiber are compressed; while in the minimal base B, the three nodes of the blue fiber are compressed. Two networks in different configurations can have the same base. For example, G_1 and G_2 can be compressed to B_1', while G_3 and G_4 can be compressed to B_2'. In the end, these four graphs have the same minimal base B. In the present work, we address the case of weighted graphs. With binary edge weights (representing simply the connection or its absence), the scenario is reduced to that described in (*23*).

All networks G_1, G_2, G_3, G_4, bases B_1, B_2, and the minimal base B share the same forward computation. That is, for every input x, the output $\hat{y}$ is identical across all networks. Moreover, for every pair (x, y) in the dataset, the loss values are the same for all networks. This implies that the loss function $\mathcal{L}(w)$ is the same for all aforementioned networks (a degeneracy of the loss function). More precisely, if two networks G and G' have the same minimal base B, then $\mathcal{L}(w_G) = \mathcal{L}(w_{G'})$ where w_G and $w_{G'}$ denote the parameters of G y G', resp. This result has important implications for the loss landscape as discussed in Section 4 and Fig. 4d.

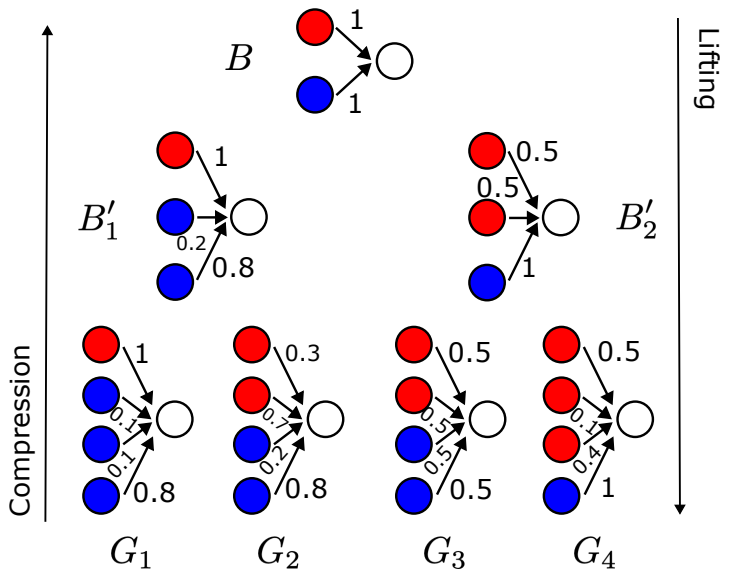

Figure S12: Examples of compression and lifting operations as reduction and expansion of dimensions, respectively. Colors represent fibers.

7 Supplementary text

7.1 Alternative definitions to capture forward-backward coupling in nonlinear Feedforward Networks

There are a few alternatives to our approach of using flavor-preserving opfibrations. The proof of the synchrony theorems and stability rules can be rewritten for these alternative definitions of opfibers and covers, with minor modifications. Instead of requiring flavors, one can directly require that the sum equality hold, including the slopes for all inputs x

- **Option 1: Hybrid opfiber definition**

$$i \underset{\text{op}}{\sim} j \iff \forall x, \forall \hat{c} \in C_{\ell+1}^{\text{op}} : \quad \sigma'(h_i^{(\ell)}) \sum_{k \in \hat{c}} W_{ki}^{(\ell+1)} = \sigma'(h_j^{(\ell)}) \sum_{k \in \hat{c}} W_{kj}^{(\ell+1)} \,.$$

These conditions are satisfied if the sum of the weight conditions is met and the slopes are synchronized between all possible inputs x. In the case of linear activations $\sigma' = 1$, the definition simplifies to a sum constraint on the output weight alone.

The proofs of the stability rules and the theorem actually only require error synchronization, so we can alternatively make that explicit in the definition of a cover:

- **Option 2: Hybrid cover definition**

$$i \underset{\text{cov}}{\sim} j \iff i \underset{\text{fib}}{\sim} j \quad \& \quad \forall x : \delta_i^{(\ell)} = \delta_j^{(\ell)} \,,$$

and require a covering symmetry, where we currently require a flavor-preserving opfibration. This alternative formulation of the stability rules and the cover coarsening theorem holds for all cover

definitions in this work. Indeed, note that our definition of a flavored opfibration implies a fiber in the current layer. Therefore, it automatically satisfies the cover condition 9. In other words, the flavor-preserving opfibration as we have defined it happens to be a cover. In the linear case, the error synchronization condition can be relaxed to the (unflavored) opfibration condition.

The two alternatives above swap the structural condition based on the weights, with the explicit but weaker requirement of activity/error synchronization (hence the label "Hybrid"). Indeed, the proofs can be derived entirely from activity and error synchronization. However, we can also use a purely structural definition based solely on weights, without requiring flavored trees or synchronization, as follows.

This definition also explains the phenomenon that we observed in the formation of covers. As we see in the accompanying video. Covers and opfibers first appear in the output layers, and their emergence propagates backwards across layers. However, while fibers emerge in the first layer, they do not appear to propagate forward until the covers have reached the input.

- **Option 3: Recursive cover definition**

$$i \underset{\text{cov}}{\sim} j \iff \quad i \underset{\text{fib}}{\sim} j \quad \& \quad \forall \hat{c} \in C^{\text{cov}}_{\ell+1} : \quad \sum_{k\in\hat{c}} W^{(\ell+1)}_{ki} = \sum_{k\in\hat{c}} W^{(\ell+1)}_{kj} \, .$$

This last definition leverages the knowledge that the structural requirement of fibers ensures activity synchronization and, therefore, slope synchronization $\sigma'(h^\ell_i)$. It essentially couples the definition of an opfiber with the definition of the cover, which is decoupled in the linear case.

The first two alternatives above do not have an obvious coloring algorithm based solely on weights, since the definitions depend on the activity. The last definition has the same coloring algorithm as for the flavored opfibration: One has to perform a forward coloring based on fibers, and use this to establish the cover coloring starting at the output, where in each layer, the cover is an intersection of the fibers partition with the partition due to the weight constraints.

7.2 Redundant feature pruning is a special case of fibration compression

A previous work (*58*) proposed to identify nodes that have the same input and output weights, and compress networks accordingly to accelerate forward inference. Here we show that this is a special case of covering symmetry, namely, trivial permutation of nodes in a single layer, i.e., Eq. (3) and (5) without the sums. Their method organizes weights as follows:

$$\mathbf{W}^{(\ell)} \mapsto \begin{bmatrix} \text{vect}(W[:,:,1,1]) & \dots & \text{vect}(W[:,:,1,d_L]) \\ \text{vect}(W[:,:,2,1]) & \dots & \text{vect}(W[:,:,2,d_L]) \\ \dots & \dots & \dots \\ \text{vect}(W[:,:,d_{\ell-1},1]) & \dots & \text{vect}(W[:,:,d_{L-1},d_L]) \end{bmatrix} = \begin{bmatrix} w_1 & \dots & w_{d_L} \end{bmatrix}$$

and uses the following similarity metric for agglomerative clustering

$$\text{Sim}(C_a, C_b) = \frac{1}{|C_a||C_b|} \sum_{i\in C_a, j\in C_b} w_i \cdot w_j \tag{S9}$$

In the strictest case (sim = 1), i, j are in the same cluster if $w_i = w_j$. That means, $\forall n = 1, ..., d_{\ell-1} : W_{in} = W_{jn}$, which corresponds to trivial permutation symmetry.

7.3 Orthogonal regularization is a particular case of symmetry breaking.

"Node collapse" sometimes refers to nodes in an artificial neural network that become correlated in their activity, representing the same features of the data. Note that this phenomenon can signal the emergence of fibration symmetry due to the synchronization theorem. It was reported in (*25, 26*). Several ad hoc ways to avoid correlations are mentioned in Section 3.4. For example, orthogonal regularization prevents node collapse, forcing weight matrices to be orthogonal (*59*). Networks are initialized with random connections that are generally orthogonal, and this method ensures that the weight updates maintain this orthogonality.

Assume a linear network $y = Wx \in \mathbb{R}^{d_1}$ with $x \in \mathbb{R}^{d_0}$ y $W \in \mathbb{R}^{d_1 \times d_0}$. For simplicity, assume that each column of W is normalized. Orthogonal regularization aims to reduce $||I - W^T W|| = ||I - 1 + 1 - W^T W|| = ||I - 1 + D_W||$ (see the definition of D_W in 6.12). That is, orthogonal regularization expects D_W to have zeros on the diagonal and ones everywhere else. This implies that if a clustering algorithm is applied to D_W, all dimensions will be in distinct clusters. In other words, the fibers in the output layer will all be trivial. Consequently, if the weights for each output node are orthogonal, no fibers exist. Then, orthogonality is a way to prevent fiber formation and avoid node synchronization

7.4 Deep Continual Learning is a particular case of Fibration Symmetry Breaking

In Deep Continual Learning (DCL) algorithm proposed in (*3*), the contribution utility is

$$u_i^{(\ell)}(t+1) = \eta . u_i^{(\ell)}(t) + (1-\eta) F_i^{(\ell)}(t),$$
$$F_i^{(\ell)}(t) = h_i^{(\ell)}(t) . \sum_{k=1}^{d_{\ell+1}} W_{ki}^{(\ell+1)}(t)$$

with $u_i^{(\ell)}(0) = 0$.

If there are two nodes $i \underset{cov,\ell,t}{\sim} j$, then $F_i^{(\ell)}(t) = F_j^{(\ell)}(t)$ because $h_i^{(\ell)} = h_j^{(\ell)}$ for fibration symmetry and $\sum_{k=1}^{d_{\ell+1}} W_{ki}^{(\ell+1)}(t) = \sum_{k=1}^{d_{\ell+1}} W_{kj}^{(\ell+1)}(t)$ for opfibration symmetry.

If there are two nodes $i \underset{cov,\ell,t}{\sim} j$ with identical utilities $u_i^{(\ell)}(t) = u_j^{(\ell)}(t)$, then $u_i^{(\ell)}(t+1) = u_j^{(\ell)}(t+1)$ (for definition of u) and $i \underset{cov,\ell,t+1}{\sim} j$. Therefore, we obtain $u_i(s) = u_j(s) \quad \forall t \leq s$. That means that *covering symmetry implies utility synchronization*.

Deep Continual Learning resets nodes belonging to the same cover when their utility reaches zero. For these nodes, input weights are reinitialized randomly, and output weights are fixed at zero. This modifies the node's activity h while avoiding immediate disruption of the learned network behavior. Altering both input and output weights necessarily modifies the input and output trees of the nodes, thus breaking their associated covers. In particular, zero-utility nodes typically exhibit null activity ($h = 0$) and belong to the same fiber. Thus, DCL inherently targets the breaking of inactive covers. The proposed fibration symmetry-breaking protocol, on the other hand, targets all covers, not only the inactive ones. This explains why the fibration breaking approach overcomes DCL in continual learning in Fig. 3f.

Caption for Movie S1. Formation of symmetries in MLPs during training with SGD. Top left. Non-trivial fibers emerge from the input layer toward the output layer. Top right. Opfibers propagate from the output layer toward the input layer. Bottom left. Covers propagation across layers. Bottom right. Accuracy as a function of learning epochs follows the formation of fibers.