From Black Boxes to Colored Graphs: 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
2 Biomedical Engineering Department, City College of New York, New York, NY 10031

* hmakse@ccny.cuny.edu

Abstract

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, characterized by isomorphic input and output trees, called fibrations and coverings. These graph theoretic 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, this outperforms the existing scaling law in the number of parameters. Breaking local symmetries also yields new state-of-the-art performance in continual learning. These results suggest a new foundation for AI, not based on ever-increasing network scale, but on emergent geometry that converts black boxes into interpretable colored graphs.

1 Symmetries in learning

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 demonstrated by Kaplan *et al.* [1, 2] which argue that "bigger is better": more parameters, more layers, more data and more compute.

The reliance on scaling as a substitute for understanding creates significant challenges. Training with single large runs has huge energy demands, and when trained sequentially, networks often suffer from a loss of plasticity over time [3]. The processing in these large models has been notoriously difficult to interpret, making it difficult to build trust in AI for critical applications. 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 [4], a subject of ongoing theoretical debate [5, 6].

To address these challenges, we propose a theoretical formalism based on symmetries in the internal structure of deep neural networks. We find that learning via stochastic gradient descent naturally induces the emergence of local symmetries in the computational graph, which are less rigid than the well-known global symmetries, such

as shift invariance in convolutional neural networks or permutation invariance in graph neural networks (GNNs). Although these global symmetries are fundamental to geometric deep learning (GDP) [7] and our understanding of theoretical physics [7–9], they are too restrictive to capture the diversity observed in biological and artificial neural systems [10–13]. We instead identify local symmetries that naturally emerge in the input trees of network nodes, which determine inference, and in the output trees, which determine the backpropagation of learning signals.

These local symmetries can be rigorously described using mathematical structures known as fibrations, opfibrations, and coverings, and are found with "balance coloring" algorithms from graph theory [10, 14]. These structures were originally developed by Grothendieck in category theory [15] and later adapted for graphs [16, 17]. We prove mathematically that covering symmetries emerge from "synchronized learning" within stochastic gradient descent. This, in turn, induces fibration symmetries, which explains the emergence of redundant representations often reported in deep networks [18, 19], 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), convolutional neural networks (CNN), recurrent networks (RNN - LSTM), and Transformers in both supervised and reinforcement learning (RL) settings. This geometric compression enables us to outperform the existing scaling law [1] with a larger exponent, achieving superior performance with fewer parameters through principled symmetry-based optimization rather than the ever-increasing network size. Furthermore, by carefully breaking these symmetries, we can expand the network capacity to overcome loss of plasticity, demonstrating state-of-the-art performance in continuous learning [3].

The symmetries that emerge in these deep networks are intimately linked to synchronized activity, which means that the nodes have correlated activity across varying input exemplars. This synchronous activity reflects shared (redundant) features in the input data, which serve to form these symmetries. Rather than treating neural networks as impenetrable black boxes, this symmetry analysis transforms them into colored graphs whose internal structure reflects the regularities in the data. Reducing the parameter space through symmetry reveals that over-parameterized models trained by gradient descent create structured redundancy, minimizing the model to its minimal computational base.

Our results demonstrate that fibration symmetry provides a unifying geometric framework for understanding and improving deep learning architectures. Identifying and breaking these symmetries offers a novel mechanism for controlling inductive bias and a principled approach to designing efficient neural networks. This opens a path toward scalable, structure-aware, and lifelong-learning AI systems guided by first principles rather than ever-increasing network size.

2 Computational Graphs in Deep Learning

The symmetries we will discuss emerge in a variety of network structures like MLPs, CNNs, RNNs and Transformers, but they are most easily understood in the canonical MLP. In this architecture, nodes are organized into multiple dense layers ($l = 1, ..., N$), with weight $W_{ik}^{(l)}$ connecting node k to i from layer $l-1$ to l. The weighted edges $W_{ik}^{(l)}$ create the computational graph that is the basis of our fibration analysis (Fig. S5a). Each node i in the network performs a weighted sum of its inputs k, followed by a point-wise nonlinear activation function σ,

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

where $h_i^{(0)} = x$ is the input of the network.

The activity propagates "forward" across layers from the input x to the output. During learning, the gradient of a loss function $\mathcal{L}$ with respect to weights $W_{ik}^{(l)}$ can be calculated with the error backpropagation algorithm, with the error signal $\delta_i^{(l)}$ flowing "backward" through the computational graph [20],

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

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.

Hierarchical local symmetries emerge in this graph. For an easy visualization of this hierarchy, consider an MLP with binary weights (where connections are 1 or 0), as shown in Fig. 1. In the figure, nodes are colored to indicate their symmetry class according to various symmetries. We will generalize these concepts to continuous-valued weights in the definition (3).

There is a *fibration symmetry* between the nodes when the nodes have isomorphic "input trees" [10, 16, 17]. This means that the entire structure of the connections and colors from the input to the nodes is the same. An example is shown in Fig. 1a). For example, the two cyan nodes have the identical input tree, as shown to the left. When two nodes have isomorphic input trees, then they belong to the same *fiber* (represented by the same color). Fibration symmetries are identified by balanced coloring algorithms from graph theory (see Ch. 13 in [10]). This algorithm partitions the network into color classes C_{fib} by assigning the same color to nodes that receive the exact same set of input colors [21], hence called *balanced coloring*.

Once identified, these fibration symmetries allow the total network 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 summing up their output weights to preserve their total effect. Crucially, this fibration compression (its inverse is called *lifting* [16]) conserves the inputs. Namely, the resulting base graph B_{fib} performs the exact same "forward" computation as the original graph G, which means that we can compress the network without changing its dynamics and, therefore, conserving its performance.

A similar principle applies to the backward propagation of the error signal during training. An *opfiber symmetry* occurs when nodes share an identical "output tree" — the structure of connections leading from them to the output layer (Fig. 1b). Nodes with this symmetry receive the same error signal. This allows for a similar compression $\varphi_{\text{op}} : G \to B_{\text{op}}$ that preserves the output trees as shown in the upper right of the figure.

When nodes have both isomorphic input and output trees, they form a *covering symmetry* (Fig. 1c). The corresponding coloring C_{cov} is a refinement (or intersection) of C_{fib} and C_{op}, i.e., a finer partition of the graph. Clearly, a covering has a more stringent symmetry than fibers or opfibers.

The most stringent symmetry is the *automorphism*. 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 in contrast to 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 a cornerstone of GDL [7] and theoretical physics [8, 9, 22], 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

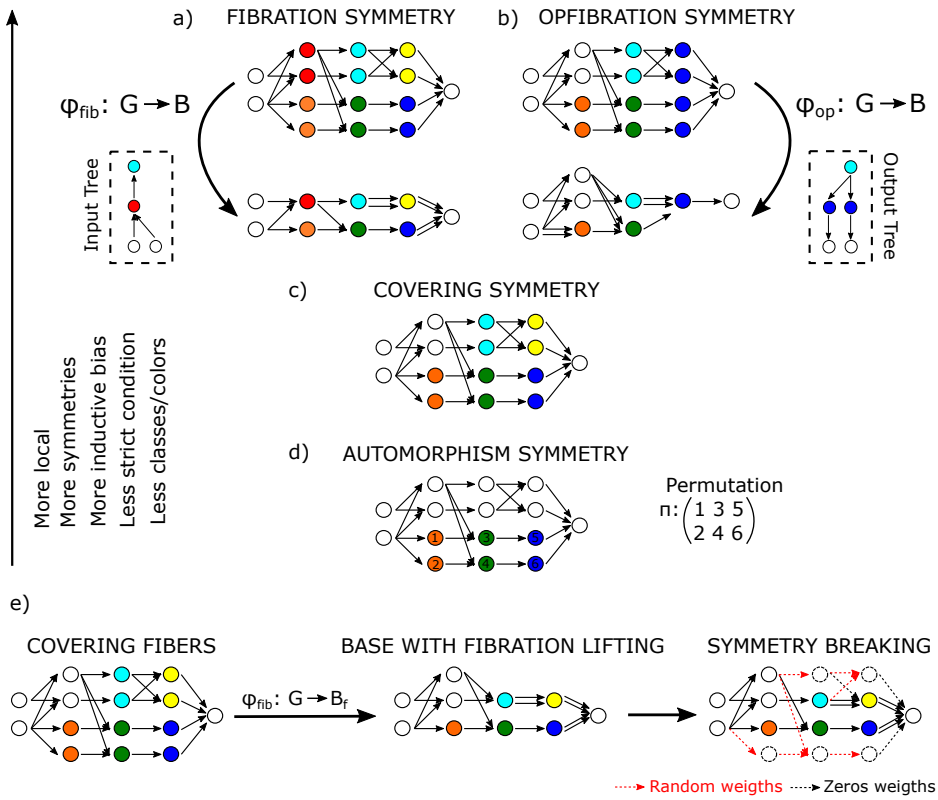

Figure 1. Hierarchy of symmetries in a sample layered feedforward network with binary edges for simplicity. Weights are binary, and edges with zero weight are not plotted. GDL is based on the most restrictive types of global symmetry groups or automorphisms. An automorphism permutation is exemplified in the case of a graph at panel (d). It identifies three pairs of symmetric nodes by a permutation (they are called orbits). The proposed framework is based on the generalization to local symmetries, such as (op)fibrations and coverings (see panels a-c). These symmetries contain the orbits but also identify other symmetric nodes. This increases the inductive bias with less strict conditions and more symmetries, resulting in more compression with less balanced colors (in these examples, blank nodes should be interpreted as distinct colors). (e) Breaking of symmetry consists of two optimized steps. Middle panel: Collapse to the covering base via fiber-lifting to preserve the old task. Left panel: Randomizing redundant nodes by symmetry breaking.

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.

3 Theoretical results

3.1 Fibration symmetry implies activity synchornization

Generalizing to networks with continuous weights, and more formally, two nodes i and j in layer l belong to the same fiber (denoted $i \underset{fib}{\sim} j$) if and only if (iff):

- **Fibration symmetry:**

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

This criterion partitions the nodes in the layer ℓ into a coloring $\mathcal{C}^{\text{fib}}_{\ell}$ based on input weights $W^{(\ell)}_{ik}$ and the color partitioning of the previous layer $\mathcal{C}^{\text{fib}}_{\ell-1}$. 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 having distinct colors $\mathcal{C}^{\text{fib}}_0 = \{1, 2, ..., d_0\}$, i.e., trivial fibers. The definition of fibers in Eq. (3) based on the sum of weights is the correct generalization from binary graphs to weighted graphs, as we show in the Methods Section 8.1.1.

Similarly to symmetry-induced invariance theorems in physics [8], these symmetries imply the synchronization of a specific network variable: 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:

$$i \underset{\text{fib}}{\sim} j \implies \forall x : h^{(l)}_i = h^{(l)}_j \tag{4}$$

The proof of Eq. (4) is in the Methods Section 9.1; similar proofs exist for dynamical systems in [10, 23, 24].

Equation (1) establishes the relationship between the network structure captured by its weights and the network function captured by its activity. This relation emerges naturally from the recursive dependence of neuronal activity h^l_i on activity in the previous layer through input weights W^l_{ik}. Therefore, the input tree structure deterministically governs synchronization patterns.

3.2 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 $\sigma'(h^l_i)$. Specifically, the error $\delta^(l)_i$ for node i depends not only on the error signals from the higher layers (mediated by the output weights $W^{(\ell)}_{ik}$) but also on the activation value $h^(l)_i$ at that node, which introduces a coupling 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, being a function of h^l_i effectively couples the balanced color assignments from the forward fiber structure Eq. (3) onto the edges defining error flow in Eq. (2).

We formalize this by introducing *edge flavors* into the backpropagation 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, which preserves flavor assignments of edges [25]. Therefore, two nodes belong to the same flavored opfiber if:

- **Flavor-preserving opfibration symmetry:**

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

with flavors given by

$$q(i) = \begin{cases} 1 & \text{linear case: } \sigma' = 1 \\ c(i) \in \mathcal{C}_l^{fib} & \text{non-linear case: general } \sigma'. \end{cases}$$

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

Using this definition, we arrive at the following theorem.

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

$$i \underset{\text{op}}{\sim} j \implies \forall x, y : \delta_i^{(l)} = \delta_j^{(l)} . \tag{6}$$

The proof is given in the Methods Section 9.1.

We note that for linear activations ($\sigma' = 1$), the relationship between synchronization and (op)fibration becomes bidirectional ($\iff$ in Eqs. (4) and (6) as we prove in Section 9.2). Synchronized nodes necessarily form fibers, and conversely, nodes in a fiber will synchronize. This represents a strict structure-function equivalence that is difficult to demonstrate for more general dynamical systems [26]. The limiting behavior confirms that nonlinearity is the essential source of forward-backward coupling. Alternative definitions for expressing the coupling of forward and backward computation are treated in Method Section 8.1.2. In the theory of gasses, 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 quantized compression factor.

3.3 Coverings symmetries synchronize learning

The final symmetry we will introduce is the covering symmetry, which we will related to the gradient descent update equations. Two nodes belong to the same covering iff they are in the same fiber and same opfiber:

- **Covering symmetry:**

$$i \underset{cov}{\sim} j \iff i \underset{\text{fib}}{\sim} j \quad \& \quad i \underset{\text{op}}{\sim} j . \tag{7}$$

This condition defines a partition of the network $\mathcal{C}_l^{\text{cov}}$, which, as we shall see, is preserved under gradient descent. The weight update of the gradient descent algorithm locally depends on node activity and error signals [20].

$$\Delta W_{mi}^{(l)} = -\alpha \frac{\partial L}{\partial W_{mi}^{(l)}} = -\alpha \delta_m^{(l)} \cdot h_i^{(l-1)} , \tag{8}$$

where α is a learning rate (see the derivation in Methods 9.3).

According to Equations 4-6 nodes i, j of the layer $l-1$ 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 follows the following:

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

$$\Delta W_{mi}^{(\ell)} = \Delta W_{mj}^{(\ell)} = \Delta W_{ni}^{(\ell)} = \Delta W_{nj}^{(\ell)} . \tag{9}$$

The proof is in the Methods Section 9.4). Note that, unlike the synchronization of activations and errors in Eqs. 4-6, this synchronization occurs in the parameter space.

3.4 Covering symmetries emerge during stochastic gradient descent

We now turn to the question of how there symmetries emerge during learning. When a neural network is initialized, its weights are typically set to random values. Large random graphs of this kind rarely have significant global automorphism symmetries as they are forbidden by Erdos-Renyi asymmetry theorem [27]. However, the training process itself creates symmetry. For example, recent studies have observed that stochastic gradient descent (SGD) causes different nodes to develop similar input and output weights [28]. We have also previously observed the emergence of *synchronized* nodes – nodes that have identical activity across all inputs during training [13].

Our results offer a theoretical proof that links these empirical observations to the emergence of covering symmetry. Namely, as a consequence of *synchronized learning*, once two nodes are in a single covering, the weight update with the gradient descent algorithm keeps the nodes in the covering. This is formalized in the following Theorem:

Theorem 3.1 ***Cover Coarse-Graining Theorem.*** *Under the dynamic GD, the covering partition $\mathcal{C}^{\text{cov}}(t+1)$ is a coarsening of $\mathcal{C}^{\text{cov}}(t)$. If network parameters θ_t are such that $i \underset{cov}{\sim} j$; then $i \underset{cov}{\sim} j$ still holds for θ_{t+1}. That is, $i \underset{cov}{\sim} j$ is invariant under GD dynamics.*

We proof this theoretical result in the Methods Section 9.5. 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. A summary of these theoretical results are presented in Fig. 2a-b along with the stability rules used during the proofs.

Chen *et al.* [28] have shown that any invariant set is a stable attractor in the stochastic gradient descent algorithm, provided that there is a sufficiently large learning constant. The intuition is that stochastic fluctuations during learning occasionally bring nodes into an invariant set; once there, the gradient descent algorithm can no longer break the symmetry due to synchronized learning. Leveraging the same result, we conclude that the covering invariant set is a stable attractor of the SGD dynamic. In Section 4.1 we provide empirical evidence that our *Covering Invariant Set* exhibits the same attracting behavior.

The theoretical results presented here readily generalize to other computational graphs, including hypergraphs, as we discuss in Section 8.4, 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.

3.5 Fibration compression preserves activity and loss

Once covers have formed and fibers are identified, one can produce a more compact base graph by merging all nodes in a fiber. The forward computation and the output of the graph G are preserved in this base graph B_{fib} if we sum up all the output weights

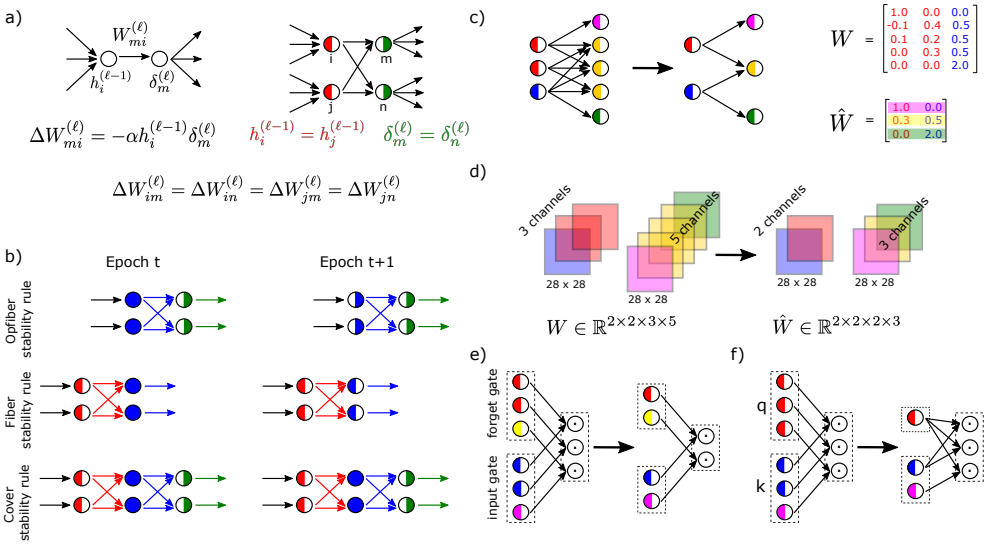

Figure 2. Theoretical results. (a) The gradient descent learning rule updates a connection's weight based on the product of the input signal (activity from the sending node) and the error signal received by the receiving nodes. Nodes in the same fiber (nodes with red left sides) synchronize their activities (i.e. $h_i = h_j$, see Eq. 4). Nodes in the same opfibers (nodes with green right sides) synchronize their error signals (i.e. $\delta_n = \delta_m$, see Eq. 6). The weights of connections between nodes in a fiber and nodes in an opfiber update with the same weight updates ΔW. In Eq. 9, we present this phenomenon called *synchronized learning* for Gradient Descent. (b) The opfiber stability rule (see Section 9.5) states that nodes in the same covering fiber (blue nodes) at epoch t will belong to the same opfiber at the next epoch $t+1$, if their output nodes are on the same opfiber. Fiber stability rule states that nodes in the same covering fiber at epoch t will belong to the same fiber at the next epoch $t+1$, if their input nodes are on the same fiber. Both rules are combined into the cover stability rule, which states that nodes in the same covering at epoch t remain in the same covering in the subsequent epoch $t+1$. Edge colors indicate "flavor" for the opfibers. Black edges for fibrations are no "flavored". (c) Example of fibration compression of a dense layer. $\hat{W}$ 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), not the spatial dimension of the activity/kernel (28 x 28, 2 x 2). (e) Example of fibration compression of "gates" in recurrent networks, e.g. LSTM. If nodes are in the same fiber for the forget gate (see red nodes), 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 (here 2 nodes per gate). (f) Within an attention module of Transformers, the number of fibers may vary for the query (q), key (k), and value (V) projections. The fibration compression preserves the count of interactions (3 multiplications). This is achieved by allowing a single fiber to participate in multiple interactions.

(and average over the input weights) with the same colors (see the proof in Section 8.5). The new weights $\hat{W}$ in the base are then:

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

where $c \in \mathcal{C}^{\text{fib}}_{l-1}$, $c' \in \mathcal{C}^{\text{fib}}_{l}$ and $|c'|$ is the number of nodes in the fiber c'. In particular, this compression rule ensures that the graph and its base have the same loss value. The compression rules are exemplified for an MLP in Fig. 2c.

The function φ_{fib} is not injective in the space of graphs; i.e. multiple graphs can have the same base. In weighted networks, Eq. 14 shows that there exist multiple combinations of the $W^{(l)}$ that generate the same $\hat{W}^{(l)}$. Since φ_{fib} preserves the forward computation, all graphs with the same base will have the same forward computation (see example in Section 8.6). As we will show later, this implies that multiple points in parameter space will share the same loss value (see Fig. 4).

For convolutional networks, the same compression rule applies to feature channels rather than nodes (Fig. 2d). For the "gating" typical in recurrent networks (e.g. LSTM) and attention layers (Transformers) there are no weights in the multiplicative interactions and collapse only needs to connects nodes in the base as exemplified in (Fig. 2c), the proof for this is in Section 8.5.3.

We will demonstrate in Section that this allows substantial compression of the network without substantially changing its function. A special case of compression for the trivial permutation symmetry appeared previously in the literature (see Supplement 12.2).

Compressing fibers during learning will allow us to achieve high performance on a fixed task with much smaller networks. Compression can also be thought of as a way to prune the network, and in Section 4.2 we show that this targeted fiber pruning outperforms existing pruning methods in the literature.

Finally, we will also demonstrate empirically that the collapse of nodes into these symmetries limits the network's ability to learn new data. This will serve as the foundation for a new proposed learning rules based on symmetry breaking in Section 4.3 for continual learning.

4 Empirical results

4.1 Emergence of fibers and opfibers during learning

The main theoretical results indicate that during training, coverings are created that act as stable attractors of the stochastic gradient descent dynamic. At the start of learning, all nodes have unique random connectivity, and thus constitute 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 (Fig. 3a), while the number of nodes in each fiber/opfiber increases (Fig. 10). When the fibers stabilize and the accuracy no longer increases, the network has learned the training set.

The proof of Theorem is based on stability rules (see Section 9.5), which provide an intuition for how fibration and opfibration symmetries develop during learning. The Fiber and Opfiber Stability Rules state that the fibers in the layer ℓ preserve the fibers in the layer $\ell + 1$, while the op fibers preserve the op fibers 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. This empirical finding is visualized for an MLP trained on MNIST data in Fig. 3 and a video provided as supplementary information (Video Link). 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. This, in effect, causes cover symmetries to propagate primarily from the output to the input of the networks, which is clearly observed in the learning dynamic in the video (Video Link). This also causes larger and fewer fibers, i.e. coarser fibers in the later layers of the network (Fig. 8).

4.2 Targeted pruning of fibers outperforms other pruning method

Fibers and opfibers are defined based on an exact equality of the sums of weights in Eqs. (3) and (5). In practice, this equality has to be numerically evaluated, and we test it up to a precision of ϵ. When applying the fiber pruning procedure Eq. (10), the forward computation is preserved exactly, only for $\epsilon = 0$. With increasing ϵ, a larger fraction of nodes can be pruned, but the forward computation is no longer exactly preserved, and we observe a loss of accuracy (Fig. 3right). For the MLP trained on MNIST one can reduce the model to 17% of its original size (measured by parameter count) without appreciable loss of classification accuracy. This is a much larger compression factor than can be obtained by randomly pruning nodes or by removing nodes with small weights (measured as the L2 norm) [29].

We observe comparable compression for a CNN trained on ImageNet, where fibration-guided compression achieves reductions of similar magnitude without degradation in test performance (Fig. 3b).

We explored fiber compression followed by fine tuning to recover performance (Fig. 3d) achieving a compression of 25× the original network with less than 3.8% of performance loss in the 50m model (31×, 4.1% loss in the 78m model) as shown in Fig. 3d. Shows scaling law.

4.3 Mechanisms of symmetry breaking

The cover symmetries that emerge during SGD reduce the degrees of freedom of a network, and thus limit its capacity to learn a task. This manifests itself as synchronized (redundant) activity in a network, which limits the repertoire of features the network can represent. We recognize this as the result of fiber formation.

Several heuristics have been proposed in the literature to promote diversity of representation and avoid redundancy. The "dropout" technique [30], removes nodes at random in a given learning step. This breaks cover symmetries as the weights of the active neurons are updated while the weights of the dropout neurons remain the same. Another effective mechanism are "residual connections" that skip several layers in a deep network [31]. In our formalism, residual connections are seen as a way to break symmetries that have emerged in the skipped layers (see Section 8.4.5). Then there are methods that explicitly aim to avoid the formation of redundant activity [32–34], which here we recognize as a way to prevent fiber formation (see also Section 12.3). However, these and other ad hoc mechanisms operate in an indiscriminate manner: they break or prevent symmetries arbitrarily, often destroying critical fibers that SGD has already consolidated into the learned representation.

We propose a principled approach to symmetry breaking by identifying and targeting cover symmetries with a surgical intervention that preserves performance on existing data while providing new degrees of freedom for further learning. The following symmetry-breaking protocol is designed specifically to preserve current input-output map while re-initializing weights to learn new data. This is implemented in two steps (Fig. 3c):

1. Identify covers and reduce the graph to the covering base, i.e. preserve the unique colors of the cover symmetry, and use the fibration symmetry collapse to set the weights using Eq. (10). The latter guaranties that the input tree is preserved. This step represents a reduction in the dimensionality of the network that maintains the inference process intact.

2. The remaining nodes in the covers are redundant and can be utilized for new data. Training continues by randomizing the input weights of these redundant nodes as shown in Fig. 3c. The out-going weights of these remaining neurons are initialized to zero to ensure that the existing input-output map of the cover base is not disrupted.

Note that we chose to preserve the covering base and not the more compact fibration base. We did this because the covering base will preserve the gradient update directions, and thus learning will have the same learning trajectory as the original graph. Neither base can exactly preserve the output tree, once we decide that we want to preserve the input tree (see Section 8.3).

4.4 Fibration symmetry breaking for continual learning

By carefully breaking symmetries in this manner, we can increase network capacity and overcome the problem of plasticity loss, a phenomenon observed in neural networks in the context of continuous learning [35, 36]. Current heuristics for learning new tasks, such as resetting the final layers with random weights or resetting specific nodes, offer limited scope and fail to provide a general solution [36]. We tested the symmetry breaking approach for the case of class-incremental learning [37] using the ImageNet dataset. The task consists of training and evaluating the model on a distinct subset of two classes out of the total of 1,000 classes. After completion of training and evaluation of two classes, the model progresses sequentially to the next binary classification. The state-of-the-art solution for this problem is "continual backpropagation" [3], outperforming shrink-and-perturb [38] – another heuristic method. Continual backpropagation uses a similar resetting mechanism as we propose here. However, it only resets nodes that have become permanently inactive. In section 11.3.1 we show that this technique is a special case of our symmetry breaking mechanism.

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 (for details, see Section 11.3). As we see in (Fig. 3c) SGC loses its ability to learn new tasks after approximately 2,000 sequential tasks are presented. Our symmetry breaking method continues to improve in performance even after 5,000 sequential tasks, indicating that the network continues to capture useful features that generalize across class classes. Deep Continuous Learning, the current state of the art, also outperforms conventional SGD, but eventually also saturates in performance.

5 Emergence of synchronization of class-conditional clusters as a result of symmetry formation

The main theorem of this paper establishes that learning with SGD forms coverings in deep networks. The emergence of covers during learning implies the formation of fibers. Equation 4 established that the nodes in a fiber have identical activity for all inputs and therefore are perfectly correlated. In other words, node correlation is a consequence of SGD in deep networks. This provides a theoretical explanation for the well-known

phenomenon of node correlation [18, 19]. This phenomenon has remained unexplained until now. Here we see that it is a necessary consequence of the formation of covering symmetries during SGD.

We asked how the correlation of the nodes changes when we limit the input to a particular dataset. Specifically, we analyze how the correlation of activity behaves for inputs selected from particular classes in a classification problem. We find that correlation matrices for individual classes form large clusters of correlated nodes (Fig. 4a, shows layer 1 of an MLP trained on MNIST).

In Section 10 we show that fibers are a refinement of these class-conditioned synchrony clusters (equivalently, class-conditional clusters are a coarsening of fibers). Thus, each class-specific synchrony cluster can be decomposed into multiple fibers, revealing a hierarchical structure. To test this theory numerically, we threshold the correlation coefficients for the synchrony clusters as well as the fibration condition (3), and then measure a refinement score (see Section 10) between these two partitions of nodes (Fig. 4a, bottom). 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.

One can interpret class-conditional synchronization clusters in terms of features of the input: It is well accepted that node activity in hidden layers captures the presence of features in the stimulus (of increasing complexity as you go up in layers of the network), e.g. simple examples for 2D CNN are edges, junctions, corners. Class-conditional clusters mean that those features are redundant to describe a single class but are needed to describe other classes. For the MNIST data, the classes are images of individual digits. The intersection of the clusters for "0" and "9" will capture the features shared between these two digits (blue and red clusters in Fig. 4b, top). The intersection of all class-conditional clusters is a refinement that is captured by the fibers. Therefore, fibers capture the regularity in all training data.

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 input (see Section 10). For small thresholds, they are a perfect match (matching score=1, Fig. 4a, bottom), showing that fibers can also be identified simply by correlating nodes under random input.

As in previous work, we observe node synchronization in particular for the last layers of the network (Fig. 4a, top). A phenomenon that has thus far remained unexplained. Our theory explains this as the result of increasingly more relaxed conditions for the fibers. Each layer removes an additional degree of freedom if one of the fiber equations is satisfied. As we move across layers, the subspace of solutions increases and the fibers grow in size.

6 Hierarchical organization of the loss landscape under SDG dynamics

Cover formation drives parameter reduction, naturally eliminating the massive overparametrization that characterizes modern deep networks. This process is a hierarchical coarsening under the SGD dynamic in the parameter and node space. The is exemplified in Fig. 4b-d, where we show the state of the network in three different training stages (T_1, T_2, T_3), their transitions $T_{1\to 2}$ and $T_{2\to 3}$, and the evolution of three nodes (p, q, z) as they merge by covering coarsening.

Figure 4c shows the node space, with each point in this space indicating a node. Disjoint areas in this space (separated by white or colored lines) indicate nodes in

distinct covers. For a classification problem (red and blue class), the red lines delineate the synchronization clusters when the input belongs to the red class (and similarly for the blue class). By definition, a cover must be contained within a fiber. Furthermore, we know that two nodes belong to the same fiber if and only if they synchronize for any input. Therefore, they must at a minimum be synchronized when the input belongs to a particular class (red and blue). This implies that the nodes within a cover cannot be in different class-specific synchronization clusters. In panel (c), this means that the colored areas (covers) cannot cross the blue or red lines (class-specific clusters). Our theorem establishes that the training progressively merges these colored areas (covers). An example of a layer with 10 nodes is shown in panel (b). The merging of covers is represented as a tree.

Figure 4d is a symbolic representation of a parameter space (weights). The learning trajectory (black curve) merges the nodes p, q and z into covers. During the learning phase T_1, the three nodes p, q, and z belong to distinct covers. At a later point in the training, p and q merge into the same cover. According to the cover coarsening theorem 3.1, once this occurs, they can never separate again, as shown in the learning stage T_2 and T_3. In other words, the first time p and q share a cover defines an irreversible transition (green dashed line $T_{1\to 2}$). A similar transition occurs when node z merges into the cover containing p and q, defining a second transition line $T_{2\to 3}$. The colored zones in this space are regions where the network has the same cover base (with identical nodes and weights). Networks sharing the same base will have identical activity and consequently will have the same loss function as exemplified by the flat areas in the landscape in Fig. 4e. In other words, the loss function is flat in these regions.

The loss landscape possesses an attractor structure under SGD dynamics, which is also characterized by a hierarchy that can be formally represented as a tree (Fig. 4e). In this tree structure, the leaf nodes represent degenerate minima with constant errors associated with the fibers and energy barriers that separate these basins of attraction. This hierarchical coarsening of the covers under the SDG learning dynamics mirrors the energy landscape structure observed in spin glass systems [39], where the system progressively explores increasingly coarse-grained symmetry quotients during optimization.

7 Conclusions: From Black Box to Colored Graphs through Geometric Interpretability

Our geometric interpretation fundamentally transforms the interpretability problem in deep learning. Rather than treating neural networks as opaque black boxes, our 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 neurons 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 is the geometric structure the network has discovered?" The colored fiber decomposition provides a complete 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 necessary 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 that correspond to the flat regions in the loss-parameter landscape in Fig. 4a that generalize well, meaning that the effective model that emerges is far simpler than the raw parameter count suggests. It also explains why deep networks outperform shallow ones 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 where random fluctuations are too numerous.

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, and in reinforcement learning settings. The fibration allows for drastic compression by collapsing 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 like transfer learning.

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 the emergent symmetries reveal the learned regularities of the data. This theory-driven perspective offers a path to designing future AIs where mathematical principles, not ever-increasing scaling, drive performance, generalization, and interpretability.

Acknowledgments

This work was partially supported by National Institutes of Health (NIH) Grants R01 EB028157 and R01 CA247910.

Authors Contribution

OMV developed theory and algorithms, and performed analysis, AH developed algorithms and performed analysis, LCP developed theory and contributed to analysis, and HAM designed and led the project, developed theory and contributed to analysis.

Data and Code

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

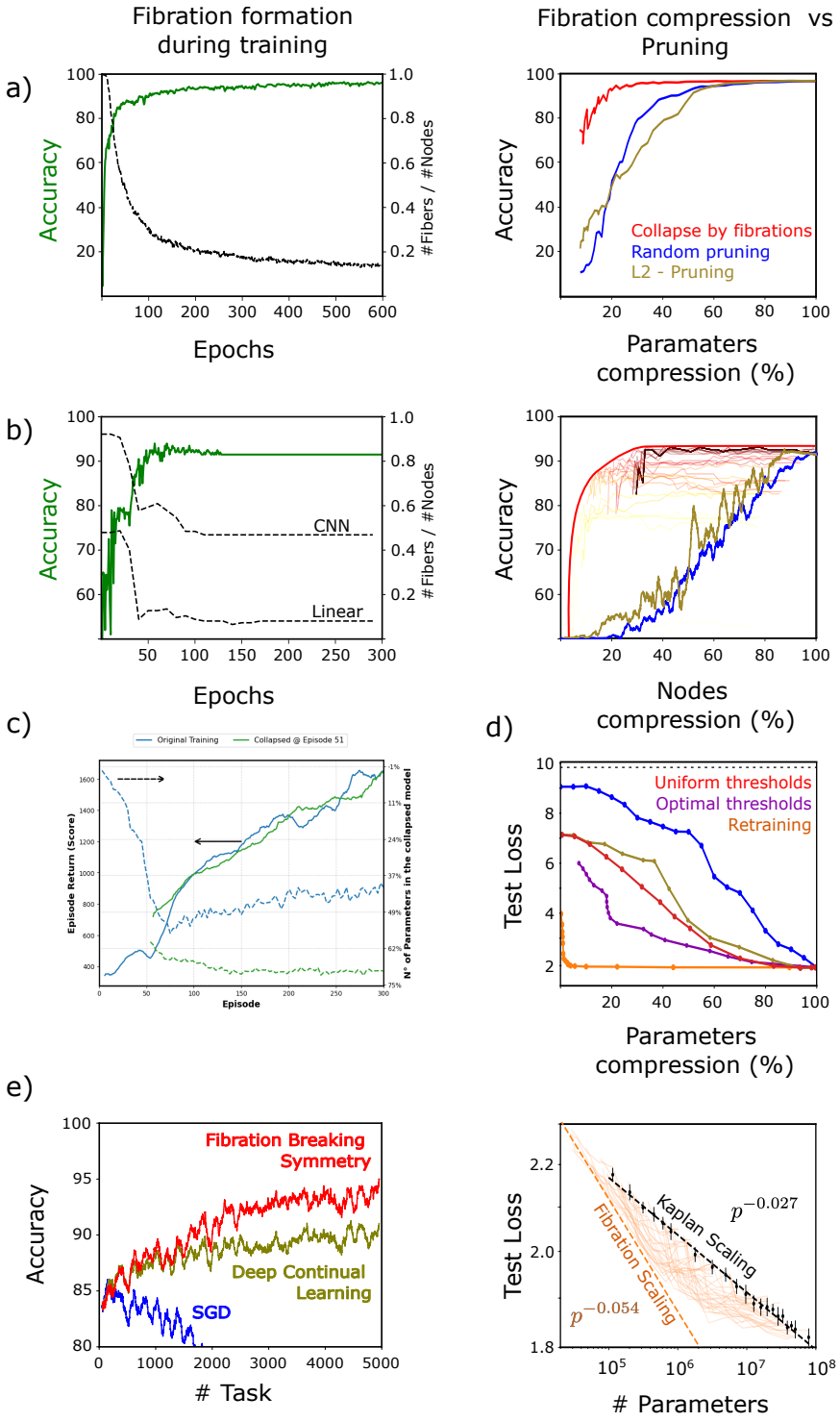

Figure 3. (a-b) Fiber formation and pruning in various network architectures. **Left:** Formation of fibers as performance improves with learning on the validation set. **Right:** Compressing the network to its fibration base graph. As ε is increased more nodes can be compressed as the cost of decreased performance. Gray and blue colors indicate alternative random pruning methods explained in Methods Section 11.2. (a) MLP trained on MNIST with performance measured as classification accuracy. Here we used a fixed $\varepsilon = 0.81$ to determine fibers. (b) CNN trained on ImageNet (3 convolution layers and 2 dense layers, with separate ε thresholds each). Curves with gradually changing colors (red to yellow) indicate varying ε_{conv} threshold for convolution layer fibers at one of 16 different dense-layer thresholds ε_{dense}. The red curve is the maximum performance for different combinations of threshold values. (c) LSTM trained on reinforcement learning. (d) Transformer trained on German-English translation showing compression with and without retraining. d.1 Different transformer compressions. d.2 Scaling law.

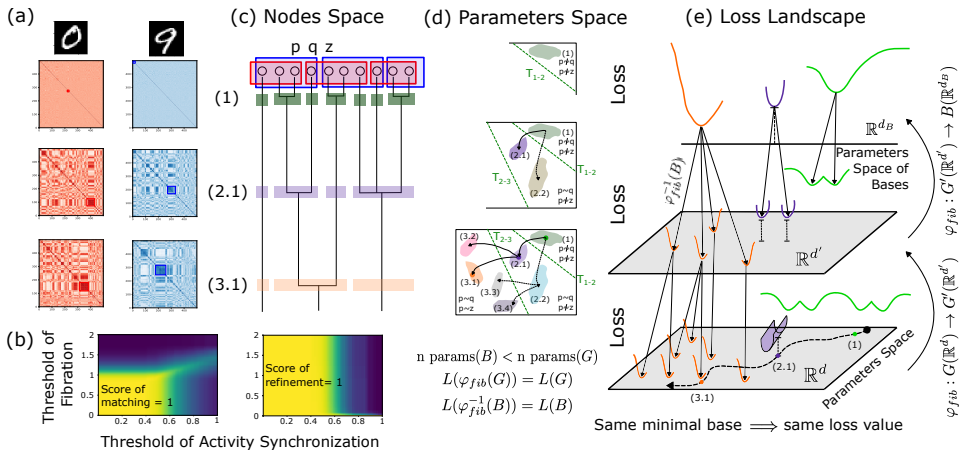

Figure 4. Synchronization, hierarchical clustering and energy landscape. The relation between synchrony clusters per class and fibers. The results provide an interpretation of the energy landscape in terms of an attractor of the SGD dynamics, giving rise to a hierarchical energy landscape that can be conceptually modeled as a tree structure where nodes represent states and internal nodes represent barriers, capturing the coarsening of covers symmetries analogous to spin glasses. (a) Top: Pearson's correlation coefficients of activity in Layers 2 and 3 with inputs drawn from individual classes of the MNIST set after training. Bottom: Measures the matching score (see Section 10) between node partitions obtained with activity synchronization (using random input images) versus clusters obtained with fibration (up to ϵ in Algorithm presented in Section 8.2). Provided that threshold on activity synchronization and fibration equality are small, both method find the same partition. We also quantify the agreement between fibers and clusters by considering if the fiber partition is inside the synchronization partition by using a refinement score (see Section 10). For this, synchronization was measured with class-conditioned inputs from the MNIST data for individual classes, and partitions then intersected to produce a refinement that is synchronous for inputs from for all classes. This is a weaker condition than fibration.

8 Materials and 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.

8.1 Global and Local symmetries in graphs: coverings, fibrations, and opfibrations

The mechanisms of how symmetries emerge and how they are broken provide a deep understanding of the underlying principles that govern the dynamics of networks [40, 41]. Usually, the formation of symmetries arises from optimization processes [42, 43]; conversely, breaking symmetries leads to diversity, complexity, and functionality in systems [44, 45]. The framework for analyzing these symmetries is provided by graph theory.

In [13], we defined the symmetries of a directed graph as transformations that preserve the particular properties of a graph. These transformations can include merging, splitting, or permutation of nodes, and the properties to be preserved can be more or less strict. For example, a *fibration symmetry* is a transformation that preserves the input trees. A *covering symmetry* preserves both the input and output trees; finally, the strictest level of symmetry is the *automorphism symmetry*, which preserves all connections between nodes.

A directed graph 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 $\tau : T_u \to T_v$, 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.

For graph G,

1. A *fibration symmetry* of G is a surjective homomorphism $\Phi_{\text{fib}} : G \to B$ that preserves the input tree:

$$\forall u, v \in N_G : \varphi_{\text{fib}}(u) = \varphi_{\text{fib}}(v) \in N_B \iff T_u \sim T_v. \tag{11}$$

2. An *opfibration symmetry* of G is a surjective homomorphism $\Phi_{opf} : G \to B$ that preserves the output tree:

$$\forall u, v \in N_G : \varphi_{opf}(u) = \varphi_{opf}(v) \in N_B \iff \hat{T}_u \sim \hat{T}_v. \tag{12}$$

3. A *covering symmetry* of G is a surjective homomorphism $\Phi_{cov} : G \to B$ that preserves the input and output trees.

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

Unlike automorphism symmetries, which are global and highly restrictive, fibration symmetry, opfibration symmetry, and covering symmetry are all local symmetries. Examples of these symmetries are shown in Fig. 1.

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 by:

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 1 & 4 & 3 & 6 & 5 \end{pmatrix}, \tag{13}$$

in Cauchy's two-line notation which consists of a first row with the original labels of nodes, and a second row with the permuted labels.

When G is a graph with flavored edges, 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.

8.1.1 Symmetries in Weighted Feedforward Networks

In 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, we will 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 ($l = 1$), 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 $l = 2$, 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 $l = 1$, 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 $l = 1$. Therefore, two nodes i and j of the layer $l = 2$ 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 $l = 2$. This argument can be extended to the next layers $l = 3, ..., N$ thus recursively defining fibers. This leads to the equal-sum criterion in the definition (3) of fiber in weighted graphs in Section 3). Similar formulations have been used in the context of dynamical system synchronization [46, 47].

In order to define flavored opfibers, we will now discuss isomorphic flavored output trees. (They are referred to as colored weights in [25], but here we use the word "flavor" to not confuse with node colors). For a node i in layer $l = N - 1$, 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

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

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 $l = N - 1$. For layer $l = N - 2$, 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 $l = N - 1$, 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 $l = N - 1$ 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 $l = N - 1$. Then, two nodes i and j of the layer $l = N - 2$ 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 $l = N - 3, ..., 1$. This leads to the equal-sum criterion in definition (5) of opfiber in weighted graphs in Section 3) 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 $q(i) = c(i) \in \mathcal{C}^{fib}$). 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.

8.1.2 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**

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

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 $\sigma' = 1$ and the definition simplify 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 7. 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 on the basis of activity and error synchronization. However, we can also use a purely structural definition based on weights alone, 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 (which now we are practically the same) first start appearing at 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**

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

This last definition leverages the knowledge that the structural requirement of fibers ensures activity synchronization and, therefore, slope synchronization $\sigma'(h_i^l)$. 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 on weights alone, as 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.

8.2 Refinement Algorithms for Balanced Colorings in Feedforward Networks

A *coloring* c of a graph G is a map $c : N_G \to \mathcal{C}$ where $c(u)$ is called the color of the node u and $\mathcal{C}$ 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 $c_{t+1}(u) = c_t(u)\ \forall u \in N_G,\ t > T$.
In a layered feedforward network, the method can be expressed as follows:

1. $c^{(l=0)} \in \mathcal{C}_0^{\text{fib}}$ assigns unique colors to each of the input features ($l = 0$).
2. For $l = 1, ..., N$:
   - 2.1. Calculation of $\hat{W}_{ir}^{(l)} = \sum_{k|c(k)=r} W_{ik}^{(l)}\ \forall r \in \mathcal{C}^{(l)}$
   - 2.2. Normalization of $\hat{W}^{(l)}$.
   - 2.3. Calculation of the matrix $D_W^{(l)} = I - \hat{W}^{(l)}\hat{W}^{(l)T}$
   - 2.4. Agglomerative Clustering of the nodes in l based on the distance matrix $D_W^{(l)}$. We use a distance threshold $\epsilon \in (0, 2)$.
   - 2.5. Fibers $c^{(l)} \in \mathcal{C}_l^{\text{fib}}$ are defined as the clusters.

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 [48].

Note that the equalities like $\sum_{k\in c} W_{ik}^{(\ell)} = \sum_{k\in c} W_{jk}^{(\ell)}$ are treated with a threshold ϵ that defines the tolerance to a weak fibration breaking (see Step 2.4) - i.e, all approximate fibration symmetries should be considered as quasi-fibrations [49] or pseudo-balanced colorings [47].

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

8.3 Lifting property by fibration symmetry

Strictly speaking, a *fibration symmetry* is the map $\varphi_{\text{fib}} : G \to B_{\text{fib}}$ that collapses the in-balanced color fibers of the original graph G into a unique node in the base B_{fib} while preserving the inputs of every node. The complete definition of fibration also concerns the mapping of edges. This is done through the *lifting property* of the fibration (see Fig. 1a). The origin of the name "lifting" is the inverse operation of compression in Fig.1, where the base is "lifted" to a full graph. Lifting is not unique, and we will use one of such lifting operations as the basis for targeted symmetry breaking.

A *covering symmetry* is a map that collapses each covering fiber in G into a single node in the base B_{cov}. However, it is generally not possible to preserve the input and output trees in this base. This would require all covers to have the same cardinality, that is, all covers to have the same number of nodes. Since this may never happen in a real system, one may compress a cover by following either a fibration compression (preserving the inputs, as we have done in Fig. 1e) or an opfibration compression (preserving the outputs). Both cases are shown in Fig. 6. In either case, the gradient update directions are the same, and they differ only by a scalar (the cardinality of the fiber versus the cardinality of the opfiber).

Typically, we are interested in preserving the inputs to preserve the learning, so that we follow a fibration compression of the covers. This should not be confused with the proper symmetry fibration that compresses the fibers using the fibration map.

8.4 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. 5a & b). More generally, modern network architectures such as LSTM and Transformers can be described as *hypergraph* (Fig. 5c & 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 squares in (Fig. 5c & d).

8.4.1 Multilayer Perceptron (MLP)

Let an MLP with $N-1$ hidden layers be described by the following equations.

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

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

8.4.2 Convolutional Neural Network (CNN)

Let a deep convolutional network with N convolutions (symbol $*$) and an average pooling,

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

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

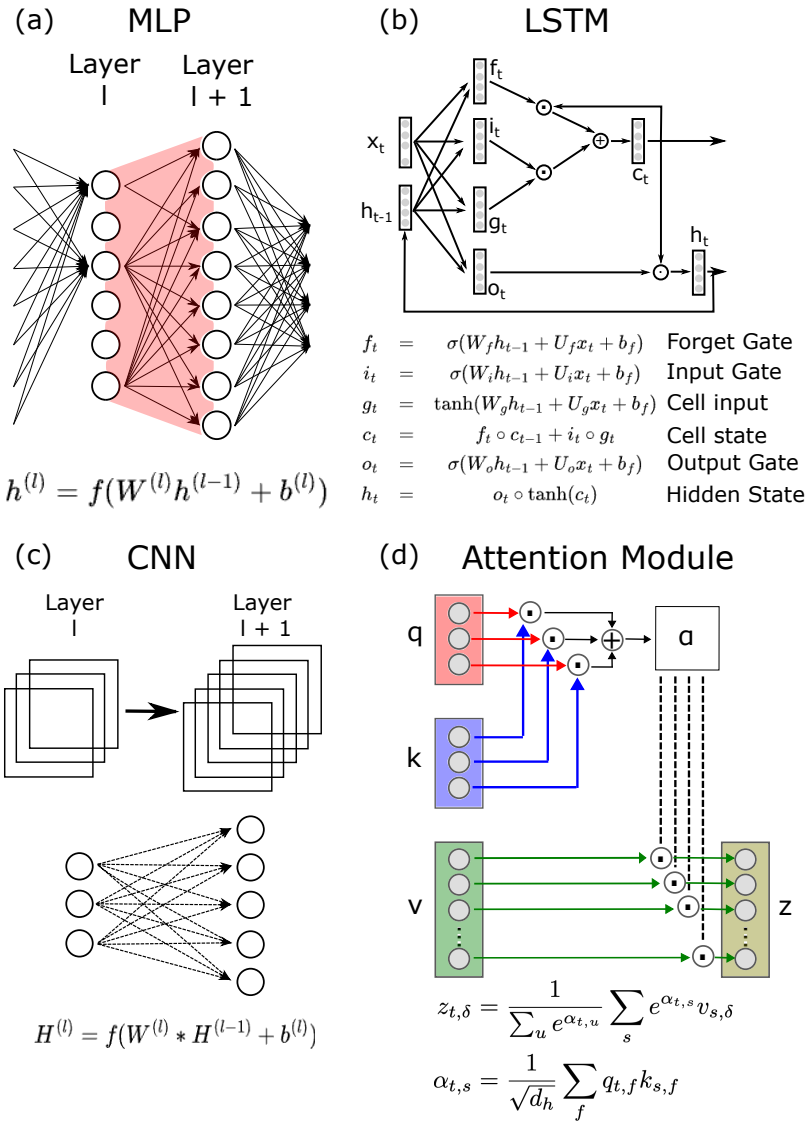

Figure 5. Schematic representation of deep learning architectures as computational graphs and hypergraphs. The figure illustrates the core topologies of: (a) MLP (Multilayer Perceptron), (b) CNN (Convolutional Neural Network), (c) LSTM (Long Short-Term Memory), and (d) Attention Module, along with their 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.

8.4.3 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. 5 that the dynamic is

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

where x_t is the input at time t, and $\odot$ element-wise multiplication.

As with most RNN, it 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. 5. 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 $[Id \quad Id]$ 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. To our knowledge, both the sum and product rules for fibers in hypergraphs are new to the literature.

8.4.4 Attention modules and Transformers

Transformers are architectures that use the attention mechanism (see Fig. 5d) to process sequential data. The attention mechanism consists of

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

where $\mathbf{X} \in \mathbf{R}^{T \times N}$ is the embedding of tokens, $\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V$ are the trainable parameters, and d_k is the dimension of 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 independently.

The attention mechanism has both a feature dimension and a token dimension. Here we have implemented a coloring algorithm to find fibration symmetries for the matrices Q, K, and 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.

8.4.5 Attention modules and Transformers

Finally, Transformers and some CNNs have "residual connections", which simply add the input to the output 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, that is, being skipped with the residual connections.

8.5 Fiber compression rules

Now we want to derive the fiber 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.

8.5.1 Fiber compression for a "dense" layer in an MLP

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

$$\begin{aligned}
\hat{h}_{c'}^{(l)} &= h_i^{(l)} \qquad \forall i \in c' \qquad \text{(activity synchronization)} \\
&= \frac{1}{|c'|} \sum_{i \in c'} h_i^{(l)} \\
&= \frac{1}{|c'|} \sum_{i \in c'} \left[W_{ik}^{(l)} h_k^{(l)} + b_i^{(l)} \right] \\
&= \frac{1}{|c'|} \sum_{i \in c'} \sum_{c \in \mathcal{C}_{l-1}^{\text{fib}}} \sum_{k \in c} W_{ik}^{(l)} h_k^{(l)} + \sum_{i \in c'} b_i^{(l)} \\
&= \sum_{c \in \mathcal{C}_{l-1}^{\text{fib}}} \left[\frac{1}{|c'|} \sum_{i \in c'} \sum_{k \in c} W_{ik}^{(l)} \right] \hat{h}_c^{(l)} + \left[\sum_{i \in c'} b_i^{(l)} \right] \\
&= \sum_{c \in \mathcal{C}_{l-1}^{\text{fib}}} \hat{W}_{c'c} \hat{h}_c^{(l)} + \hat{b}_c^{(l)}
\end{aligned} \tag{14}$$

where $\hat{W}_{c'c} = \frac{1}{|c'|} \sum_{i \in c'} \sum_{k \in c} W_{ik}^{(l)}$ and $\hat{b}_{c'}^{(l)} = \sum_{i \in c'} b_i^{(l)}$. These matrices $\hat{W}_{c'c}$ and vectors $\hat{b}_{c'}^{(l)}$ 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 6. 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. This is too strong of a requirement and has not been explored in more detail here.

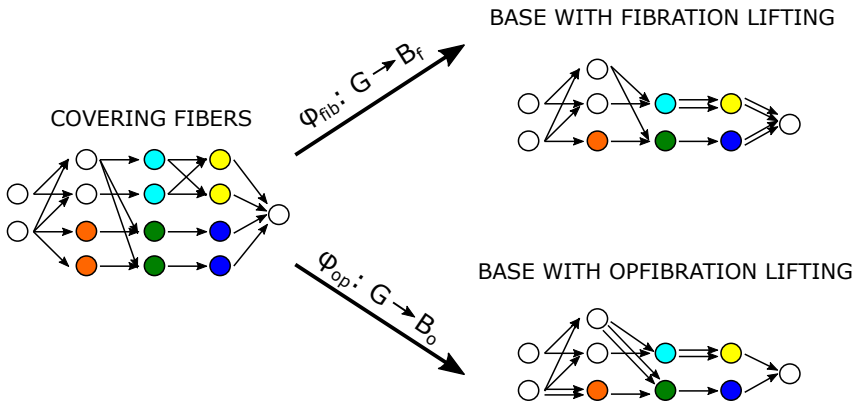

Figure 6. Collapse to the cover base: One can preserve the input tree or output tree, but generally not both unless all covers have the same cardinality which is not satisfied in this case, since 5 covers have cardinality 2 and 5 covers have cardinality 1 (trivial). Note that our notation uses blank for trivial covers. A more strick notation should assign a different color to every one of the trivial covers, but this notation would use too many colors.

8.5.2 Fiber 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, 14) can be generalized to CNNs. These equations are applied to the channels i, k in a layer l; where $W_{ik}^{(l)}$ is a D^2-dimensional vector that represents the kernel between the channels i and j.

More precisely,

$$\begin{aligned} \hat{\mathbf{W}}[:,:,c,c'] &= \frac{1}{|c'|} \sum_{i \in c'} \sum_{k \in c} \mathbf{W}^{(l)}[:,:,k,i], \\ \hat{\mathbf{b}}_{c'}^{(l)} &= \sum_{i \in c'} \mathbf{b}_i^{(l)}. \end{aligned} \tag{15}$$

8.5.3 Fiber 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 forgotten 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.

- **Attention Mechanism**. In this case, q and k can be different distributions of fibers. 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 $z = q \odot k$ (see Fig. 2e).

8.6 Comments about Lifting

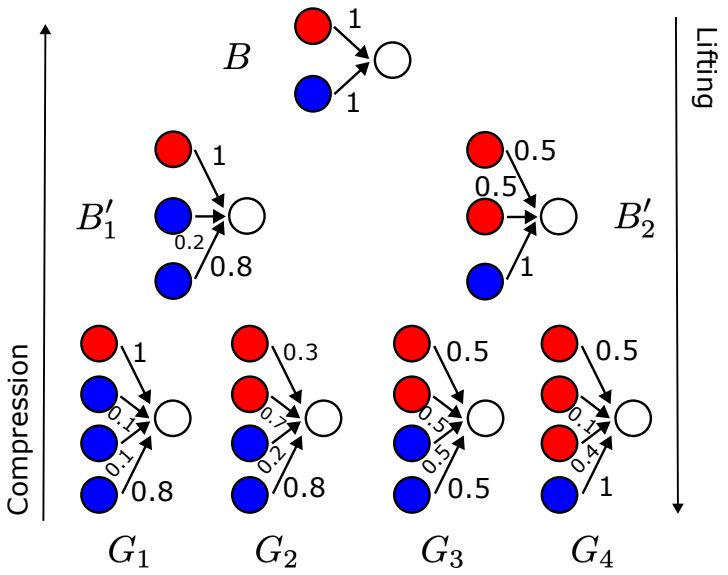

Figure 7. Caption

9 Datasets and tasks used in empirical tests

9.1 Proof of Synchronization Theorem.

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

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

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

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

With these two steps, we have proven Eq. (4).

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 synchronization holds for layer $l+1$. Let i, j be two nodes in layer l such as $i \underset{\text{op}}{\sim} j$.

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

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

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

We have proven Eq. 6.

9.2 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: to nodes with synchronized errors are in the same opfiber. Therefore, structure not only defined functions, but conversely, function defined structure.

We will prove by induction. First, for $l = 1$:

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

.

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

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

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

9.3 Gradient Descent Rule - Eq. (8)

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

$$W^{(l)}(t) = W^{(l)}(t-1) - \alpha \frac{\partial L}{\partial W^{(l)}}(t-1) \tag{16}$$

where α is called the learning rate.
We calculate the derivative

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

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

9.4 Proof of Synchronized Learning

Consider two nodes i and j in the layer $l-1$ 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 $h_i^{(l-1)} = h_j^{(l-1)}$ and $\delta_m^{(l)} = \delta_n^{(l)}$.

Then,

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

9.5 Proof of Theorem 3.1 cover coarsening theorem

During the training process, the network weights W, the symmetries, and partitions of the nodes $\mathcal{C}$ will depend on the epoch t.

Opfiber, Fiber and Cover Stability Rules. During gradient descent training of an FNN, *learning synchronization* ensures that:

1. Nodes in the same covering fiber $i \underset{cov,l,t}{\sim} j$ will belong to the same opfiber at the next epoch $i \underset{op,l,t+1}{\sim} j$ if $\mathcal{C}^{\text{op}}_{l+1}(t+1)$ is coarser than $\mathcal{C}^{\text{op}}_{l+1}(t)$.

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

3. Nodes in the same covering fiber $i \underset{cov,l,t}{\sim} j$ will belong to the same covering fiber at the next epoch $i \underset{cov,l,t+1}{\sim} j$ if $\mathcal{C}^{\text{fib}}_{l-1}(t+1)$ is coarser than $\mathcal{C}^{\text{fib}}_{l-1}(t)$ and $\mathcal{C}^{\text{op}}_{l+1}(t+1)$ is coarser than $\mathcal{C}^{\text{op}}_{l+1}(t)$.

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

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

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

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

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

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 covering fiber $i \underset{cov,l,t}{\sim} j$ will belong to the same covering fiber at the next epoch $i \underset{cov,l,t+1}{\sim} j$. The theorem relaxes the conditions of the cover stability rule.

For $l=0$ and $l=N$, $\mathcal{C}^{\text{fib}}_0(t) = \{[1],[2],...,[d_0]\}$, and $\mathcal{C}^{\text{op}}_N(t) = \{[1],[2],...,[d_N]\}$ that remain constant over time t.

We apply fiber stability rule for layer $\ell = 1$) Nodes in the same covering fiber $i \underset{cov,l=1,t}{\sim} j$ will belong to the same fiber at the next epoch $i \underset{fib,l=1,t+1}{\sim} j$ because $\mathcal{C}^{\text{fib}}_0(t+1)$ is coarser than $\mathcal{C}^{\text{fib}}_0(t)$. This means $\mathcal{C}^{\text{fib}}_1(t+1)$ is coarser than $\mathcal{C}^{\text{fib}}_1(t)$.

With the same argument, we apply the fiber stability rule for layer $\ell = 2$, and we obtain $\mathcal{C}^{\text{fib}}_2(t+1)$ is coarser than $\mathcal{C}^{\text{fib}}_2(t)$.

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

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

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

The nodes in the same cover at epoch t, will belong to the same cover in the next epoch. Equivalently, $\mathcal{C}^{\text{cov}}(t+1)$ is coarser than $\mathcal{C}^{\text{cov}}(t)$.

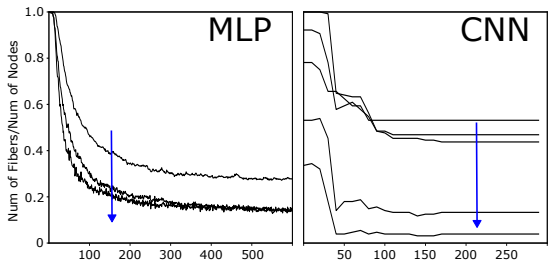

Figure 8. Number of fibers in distinct layers across epochs: As nodes group into fibers the number of separate fibers decreases and they grow in size (individual nodes count as trivial fiber). Each curve represents a layer. Higher layers in the networks (blue arrow) cluster into fibers more quickly and have a coarser partition at the end of learning (fewer larger fibers). This is the same network structure and training data as in Fig. 3.

10 Fibers are a refinement of clustering partitions

For each class c, we define $P_c^{(l)}$ as the partition obtained by activity synchronization due to class c. Each set $A \in P_c^{(l)}$ 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. 9.

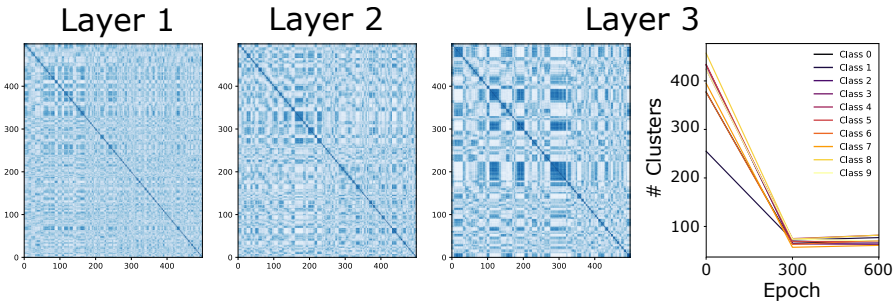

Figure 9. Partitions obtained by activity synchronization due to class 9 for hidden layers. In the right, we show the number of clusters in the partition for layer 3 as function of epoch

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

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

Two nodes in I are synchronized for all inputs in the dataset; however, they may not be synchronized for an input that is not part of the dataset. 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 (intersection of clusters across

classes). This observation is more generally true for any subset of all possible inputs, including inputs generated at random, as we did for Fig 1e.

In Fig 4a how well two partitions match each other and if one is a refinement of the other. As **matching score**, 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 the **refinement score**, we count the number of fibers that have a unique cluster label and divide by the total number of fibers.

11 Datasets and tasks used in empirical tests

11.1 Reinforcement Learning (RL)

In Deep Reinforcement Learning, the Actor-Critic method features two components that often share a common initial structure. The base network processes the environment's state s to extract representative features, which then branch into two specialized heads: the *Actor head*, which outputs a probability distribution $p(a)$ over actions a (deciding "what to do"), and the *Critic head*, which estimates the state value V (judging "how good it is to be here").

This full architecture called *Agent* interacts with the environment in a continuous cycle: the Actor samples and executes an action based on its learned policy $s \mapsto a$; the environment responds with a reward and a new state; the Critic then assesses the outcome using the value $s \mapsto V$ to calculate a temporal difference error; this error, in turn, 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.

In our case, the agent utilizes a network composed of a sequential stack of convolutional layers, dense layers, and an LSTM. The Actor then transforms the LSTM's output into a probability distribution over possible actions using a linear projection followed by a Softmax activation. Simultaneously, the Critic estimates the state value by applying a simple linear projection directly to the same LSTM output.

We have tested a typical use of LSTMs in agent training with reinforcement learning (RL). The agent is a deep neural network that maps states to actions. We have investigated fiber formation in reinforcement learning using the proximal policy optimization (PPO) algorithm in the beam-rider game [50]. The architecture consists of three Conv2d layers followed by a linear layer and LSTM with a linear policy and value head. After training, we identify fibers, collapse the network, and evaluate performance. The results in Fig. 3d show that the fibration collapse of the linear layers provides a reduction of 70% without loss of performance, as given by the score of the game. This study will be extended to a fibration analysis of the LSTM module, convolutional and linear layers in different games in RL training.

The study of symmetries in RL represents a new area of research. Since the environment works as the data source for the RL agent, the emergence of symmetries is inherently dependent on it. We will study how the environment and multi-agent interactions affect the formation of symmetries. On the other hand, LSTM represents one specific case of a recurrent architecture. We will extend our study to other existing recurrent models with different gating mechanisms, memory structures, number of feedback loops, and temporal delays.

11.2 Pruning null models

Pruning is a technique to reduce the size and computational complexity of a network. The removal of parameters or nodes is often based on their magnitude. We compare our fibration-based compression technique with random pruning and L2-pruning. For both pruning methods, the pruned network has the same number of nodes as the fibration base. Random pruning selects random nodes of the network; while L2-pruning selects nodes with the smallest $||w_{in}||_2$.

11.3 Results on continual learning

Here we will explore our approach for symmetry breaking 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, and 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. 3c, 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 for all step-size parameters. Some step sizes performed well initially but worsened later, eventually performing below a linear network's level. Others showed initial improvement but later fell, ending near or below the linear baseline. This pattern was consistent across architectures, parameters, and optimizers, indicating standard deep-learning methods struggle with continual-learning tasks. The choice of architecture, parameters, and optimizers influenced when performance dropped, yet a significant drop was observed across various configurations.

The emergence of covers during learning is exemplified in Fig. 10 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. We used a grid-search to find these parameters on a validation set. The results in Fig. 3c 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.

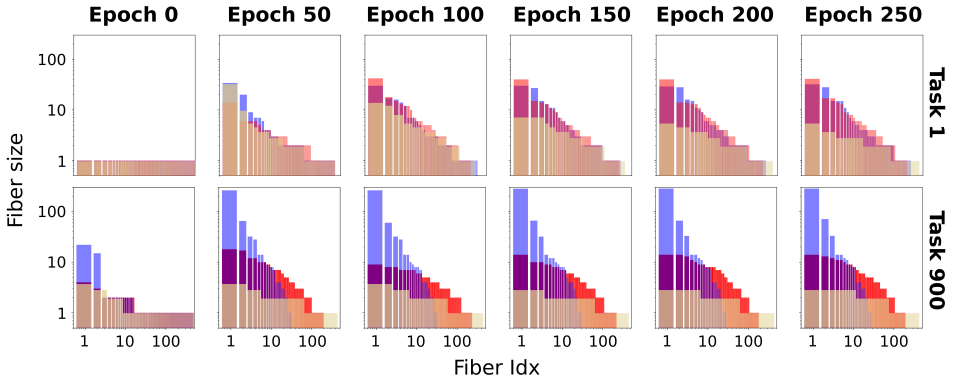

Figure 10. First two rows: Distribution of covers, fibers and opfibers during continual learning on the Sequential ImageNet Task. Evolution of the size and number of covering fibers across leaning epochs (columns) and task number (rows). Color indicates training with conventional SGD (blue), continual backpropagation (red), and symmetry breaking SGD (yellow). This distribution of for the second dense layer containing 128 nodes.

11.3.1 Deep Continual Learning is a particular case of symmetry breaking

In Deep Continual Learning (DCL), the contribution utility [3] is

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

with $u_i^{(l)}(0) = 0$.

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

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

DCL proposes resetting connections for nodes with zero utility during training.

Deep Continual Learning resets nodes belonging to the same covering fiber 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 covering fibers. 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 covering fibers.

11.4 Results on Sequence-to-Sequence Transformer models

In order to study the application of our fibration symmetry in transformer models with attention mechanism, we turn into a sequence-to-sequence translation model for a German-to-English translation task using the Multi30k dataset [51]. This dataset

consists of 31,014 sentences split into 29k training, 1k test, and 1k validation sets. For the model we use a vanilla sequence-to-sequence transformer model as introduced in [52] with Adam [53] optimizer and early stopping regularization for optimal training. The detailed specifications of the model and training process are given in Table 1.

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 1. Sequence-to-Sequence transformer model and training hyperparameters

We keep the dimension of the token embeddings and the feedforward layer size to be equal and use it to control the number of parameters in the model. Then for each of the models of different initial size we follow the procedure explained in Algorithm 1 to collapse the trained model and retrain it iteratively. Here we define the "optimal thresholds" as the optimal distance thresholds $\epsilon \in (0, 2)$ (as defined in 8.2) for each of the weight matrices in the model such that while maximizing ϵ, collapsing does not result in an increase in the test loss of the model. We obtain these sets of values for the model using a greedy algorithm explained in Sec. 11.4.1.

Algorithm 1 Collapse–Retrain 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
$L_c \leftarrow L$
Compute optimal collapse thresholds using greedy algorithm
while $L_c \leq (1 + \epsilon) \times L$ **do** ▷ ϵ is the tolerance for increase in loss after collapse
    Increase thresholds by 5%
    Collapse $\mathcal{M}$ using the adjusted thresholds
    Retrain $\mathcal{M}$ until convergence with early stopping
    $L_c \leftarrow \mathcal{L}(\mathcal{M}, \text{test set})$
    Compute optimal collapse thresholds using greedy algorithm
end while

Following this procedure, we start from a vanilla transformer model and whenever the early stopping signals the saturation of learning, we collapse the fibers in all of the intermediate layers with a slightly higher threshold than the optimal thresholds (we tried values between 1-10% and 5% was the constantly best-performing value). When collapsing, and in order to be able to follow the residual connections, we repeat the outputs of each layer in-place of the collapsed fiber nodes. At the end of each retraining, we can calculate the optimal thresholds and find the number of parameters for a collapse with no loss in performance at that step. Figure 11 shows this iterative process of collapsing and retraining the model for three models of initially different sizes.

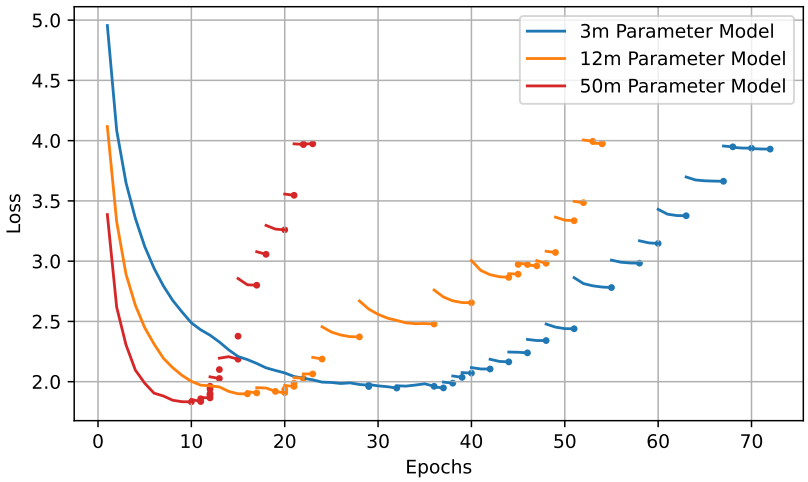

Figure 11. Collapse-Retrain procedure for transformer models Three different model sizes

11.4.1 The greedy algorithm for calculating optimal distance thresholds

12 Additional theoretical observations

12.1 Optimizers

We proved the cover coarsening theorem for conventional Gradient Descent, with stability deriving from stochastic gradient descent. However, in practice SDG is always run with additional modifiers, such as Momentum, Adam optimizer and/or L2 regularization, among others. In Section 12.1 we show that these modifications all still satisfy the cover coarsening theorem, making it relevant for current learning practice.

12.1.1 SGD with L2 regularization

For SGD with L2 regularization, the weight update equation is

$$\begin{aligned} W_{ij}(t+1) &= W_{ij}(t) - \alpha \left[h_j \delta_i + \lambda W_{ij}(t)\right] \\ &= [1 - \alpha\lambda] W_{ij}(t) - \alpha h_j \delta_i \end{aligned}$$

This equation is identical to the original SGD case, only with an extra fixed factor multiplying $W_{ij}(t)$. The proof of the cover coarsening theorem follows exactly the same steps.

12.2 Redundant feature pruning is a special case of fiber compression

A previous work [54] 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, permutation of nodes in a single layer, i.e. Eq. (3) and (5) without the sums. Their method organized weights as follows:

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

and uses the following similarity metric for agglomerative clustering

$$\text{Sim}(C_a, C_b) = \frac{1}{|C_a||C_b|} \sum_{i \in C_a, j \in C_b} w_i \cdot w_j \tag{17}$$

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

12.3 Fibration symmetry as a generalization of orthogonal regularization.

Node collapse sometimes refers to nodes in an artificial neural network taking on correlated activity so that they represent the same features of the data. This phenomenon was reported in [18, 19]. Various ad hoc solutions are mentioned in section 4.3. One in particular involves ensured orthogonal connections [55]. Networks are initialized with random connections that are generally orthogonal. This method ensures that the weight updates maintain this orthogonality.

We now explain how this approach is equivalent to preventing fiber symmetry formation. 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. To find fibers in the output layer, we have to apply a clustering algorithm on the distance matrix $D = 1 - W^T W$ (see Section 8.2) . In other words, if the weights for each output node are orthogonal, then there are no fibers. Obtaining orthogonality is therefore a way to ensure that no fibers form and to present node synchronization. This technique has previously been used to prevent node collapse.

12.4 Power Law in Transformer

The number of parameters p in a Seq2Seq Transformer scales proportionally to d_{emb}^2, while the number of nodes n grows linearly with d_{emb}. In other words, $p \propto n^2$.However, when we fix the embedding dimension and apply methods such as fibration compression or pruning, the number of nodes n becomes independent of d_{emb}, and the parameter count scales linearly with n. This yields $p \propto n$. This is shown in Fig. xxx

In particular, for fibration compression, the following holds:

$$\begin{aligned} L(G) = L(B) &\implies L(p_G) = L(p_B) \\ &\implies L(n_G^2) = L(n_B) \\ &\implies (n_G^2)^{\alpha_G} \propto n_B^{\alpha_B} \\ &\implies \alpha_B = 2\alpha_G \end{aligned}$$

This fact explains the change in the exponent in the scaling laws presented in Fig. 3.

References

1. Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

2. J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. de Las Casas, L. A. Hendricks, J. Welbl, A. Clark, et al. Training compute-optimal large language models. *arXiv:2203.15556*, 2022.

3. S. Dohare, J. F. Hernandez-Garcia, Q. Lan, P. Rahman, A. R. Mahmood, and R. S. Sutton. Loss of plasticity in deep continual learning. *Nature*, 632:768–774, 2024.

4. T. J. Sejnowski. The unreasonable effectiveness of deep learning in artificial intelligence. *Proceedings of the National Academy of Sciences*, 117(48):30033–30038, 2020.

5. M. Belkin, D. Hsu, S. Ma, and S. Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proc. Nat. Acad. Sci. USA*, 116(32):15849–15854, 2019.

6. P. Nakkiran, G. Kaplun, Y. Bansal, T. Yang, B. Barak, and I. Sutskever. Deep double descent: Where bigger models and more data hurt. In *International Conference on Learning Representations (ICLR)*, 2020. arXiv:1912.02292.

7. M. M Bronstein, J. Bruna, T. Cohen, and P. Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*, 2021.

8. F. Wilczek. *A Beautiful Question: Finding Nature's Deep Design*. Penguin, 2016.

9. S. Weinberg. *The Quantum Theory of Fields*, volume 2. Cambridge U.P., 1995.

10. H. A. Makse, P. Boldi, F. Sorrentino, and I. Stewart. *Symmetry of Living and Artificial Intelligence Systems: Fibrations and Synchronization in Networks*. Cambridge University Press, 2026. `https://arxiv.org/abs/2502.18713v1`.

11. T. Gili, B. Avila, L. Pasquini, A. Holodny, D. Phillips, P. Boldi, A. Gabrielli, G. Caldarelli, M. Zimmer, and H. A. Makse. Fibration symmetry-breaking supports functional transitions in a brain network engaged in language. *Nat. Comm., under review, https://arxiv.org/abs/2409.02674*, 2025.

12. B. Avila, P. Augusto, A. Hashemi, D. Phillips, T. Gili, M. Zimmer, and H. A. Makse. 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.

13. O. Velarde, L. C. Parra, P. Boldi, and H. A. Makse. The role of fibration symmetries in Geometric Deep Learning, 2026. Proc. Natl. Acad. Sci. USA.

14. H. Kamei and P. J. A. Cock. Computation of balanced equivalence relations and their lattice for a coupled cell network. *SIAM J. Appl. Dyn. Syst.*, 12(1):352–382, 2013.

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

16. P. Boldi and S. Vigna. Fibrations of graphs. *Discrete Math.*, 243(1-3):21–66, 2002.

17. F. Morone, I. Leifer, and H. A. Makse. Fibration symmetries uncover the building blocks of biological networks. *Proc. Nat. Acad. Sci. USA*, 117(15):8306–8314, 2020.

18. V. Papyan, X. Han, and D. L. Donoho. Prevalence of neural collapse during the terminal phase of deep learning training. *Proceedings of the National Academy of Sciences*, 117(40):24652–24663, 2020.

19. D. Doimo, A. Glielmo, S. Goldt, and A. Laio. Redundant representations help generalization in wide neural networks. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 19659–19672. Curran Associates, Inc., 2022.

20. D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

21. M. Golubitsky and I. Stewart. Nonlinear dynamics of networks: the groupoid formalism. *BAMS*, 43(3):305–364, 2006.

22. H. Georgi. *Lie Algebras in Particle Physics: From Isospin to Unified Theories*. Taylor & Francis, 2000.

23. L. DeVille and E. Lerman. Modular dynamical systems on networks. *J. Eur. Math. Soc.*, 17(12):2977–3013, 2015.

24. E. Nijholt, B. Rink, and J. Sanders. Graph fibrations and symmetries of network dynamics. *Diff. Equ.*, 261(9):4861–4896, 2016.

25. P. Boldi, V. Lonati, M. Santini, and S. Vigna. Graph fibrations, graph isomorphism, and pagerank. *RAIRO - Theoretical Informatics and Applications*, 40:227–253, 2006.

26. H. Park and K. Friston. Structural and functional brain networks: from connections to cognition. *Science*, 342(6158):1238411, 2013.

27. P. Erdős and A. Rényi. Asymmetric graphs. *Acta Mathematica Academiae Scientiarum Hungaricae*, 14:295–315, 1963.

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

29. Fanxu Meng, Hao Cheng, Ke Li, Huixiang Luo, Xiaowei Guo, Guangming Lu, and Xing Sun. Pruning Filter in Filter. In *Advances in Neural Information Processing Systems*, volume 33, pages 17629–17640. Curran Associates, Inc., 2020.

30. N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

31. K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

32. Z. Wang, C. Li, and X. Wang. Convolutional neural network pruning with structural redundancy reduction. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 14913–14922, 2021.

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

34. J. Zbontar, L. Jing, I. Misra, Y. LeCun, and S. Deny. Barlow twins: Self-supervised learning via redundancy reduction. In *International conference on machine learning*, pages 12310–12320. PMLR, 2021.

35. C. Lyle, Zeyu Z., E. Nikishin, B. A. Pires, R. Pascanu, and W. Dabney. Understanding plasticity in neural networks. In *International Conference on Machine Learning*, pages 23190–23211. PMLR, 2023.

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

37. G. M. van de Ven, T. Tuytelaars, and A. S. Tolias. Three types of incremental learning. *Nature Machine Intelligence*, 4(12):1185–1197, 2022.

38. A. Chebykin, A. Dushatskiy, T. Alderliesten, and P. A. N. Bosman. Shrink-Perturb Improves Architecture Mixing during Population Based Training for Neural Architecture Search, July 2023. arXiv:2307.15621 [cs].

39. M. Mézard, G. Parisi, and M. A. Virasoro. *Spin Glass Theory and Beyond*, volume 9. World Scientific Lecture Notes in Physics, 1987.

40. P. W. Anderson. More Is Different. *Science*, 177(4047):393–396, August 1972. Publisher: American Association for the Advancement of Science.

41. P. Ball. *The self-made tapestry: pattern formation in nature*. Oxford Univ. Press, Oxford, repr edition, 2004.

42. D. W. Thompson. *On Growth and Form: The Complete Revised Edition*. Dover Publications, New York, June 1992.

43. P. Ball. *Patterns in Nature: Why the Natural World Looks the Way It Does*. University of Chicago Press, Chicago, 2016.

44. R. Li and B. Bowerman. Symmetry Breaking in Biology. *Cold Spring Harbor Perspectives in Biology*, 2(3):a003475, March 2010.

45. A. García-Bellido. Symmetries throughout organic evolution. *Proceedings of the National Academy of Sciences of the United States of America*, 93(25):14229–14232, December 1996.

46. M. A. Aguiar, A. P. S Dias, and F. Ferreira. Patterns of synchrony for feed-forward and auto-regulation feed-forward neural networks. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 27(1), 2017.

47. I. Leifer, D. Phillips, F. Sorrentino, and H. A Makse. Symmetry-driven network reconstruction through pseudobalanced coloring optimization. *J. Stat. Mech.: Theor. Exp.*, 2022(7):073403, 2022.

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

49. P. Boldi, I. Leifer, and H. A Makse. Quasifibrations of graphs to find symmetries and reconstruct biological networks. *J. Stat. Mech.: Theor. Exp.*, 2022(11):113401, 2022.

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

51. D. Elliott, S. Frank, K. Sima'an, and L. Specia. Multi30k: Multilingual english-german image descriptions. In *Proceedings of the 5th Workshop on Vision and Language*, pages 70–74. Association for Computational Linguistics, 2016.

52. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need, 2023.

53. Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

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

55. K. Liu, M. Suganuma, and 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.