entropy

MDPI

Article

Seeing Is Believing: Brain-Inspired Modular Training for Mechanistic Interpretability

Ziming Liu *, Eric Gan and Max Tegmark

Institute for Artificial Intelligence and Fundamental Interactions, Massachusetts Institute of Technology, Cambridge, MA 02139, USA; ejgan@mit.edu (E.G.); tegmark@mit.edu (M.T.)
* Correspondence: zmliu@mit.edu

Abstract: We introduce Brain-Inspired Modular Training (BIMT), a method for making neural networks more modular and interpretable. Inspired by brains, BIMT embeds neurons in a geometric space and augments the loss function with a cost proportional to the length of each neuron connection. This is inspired by the idea of minimum connection cost in evolutionary biology, but we are the first the combine this idea with training neural networks with gradient descent for interpretability. We demonstrate that BIMT discovers useful modular neural networks for many simple tasks, revealing compositional structures in symbolic formulas, interpretable decision boundaries and features for classification, and mathematical structure in algorithmic datasets. Qualitatively, BIMT-trained networks have modules readily identifiable by the naked eye, but regularly trained networks seem much more complicated. Quantitatively, we use Newman's method to compute the modularity of network graphs; BIMT achieves the highest modularity for all our test problems. A promising and ambitious future direction is to apply the proposed method to understand large models for vision, language, and science.

Keywords: brain-inspired artificial intelligence; mechanistic interpretability; modularity

Citation: Liu, Z.; Gan, E.; Tegmark, M. Seeing Is Believing: Brain-Inspired Modular Training for Mechanistic Interpretability. *Entropy* **2024**, *26*, 41. https://doi.org/10.3390/e26010041

Academic Editor: Fernando Morgado-Dias

Received: 2 November 2023
Revised: 21 December 2023
Accepted: 27 December 2023
Published: 30 December 2023

1. Introduction

Although deep neural networks have achieved great successes, mechanistically interpreting them remains quite challenging [1–5]. If a neural network can be decomposed into smaller modules [1], interpretability may become much easier.

In contrast to artificial neural networks, brains are remarkably modular [6–8]. We conjecture that this is because artificial neural networks (e.g., fully connected neural networks) have a symmetry that brains lack: both the loss function and the most popular regularizers are invariant under permutations of neurons in each layer. In contrast, the cost of connecting two biological neurons depends on how far apart they are because an axon needs to traverse this distance, thereby using energy and brain volume and causing time delay [7,8].

We argue that (current) artificial neural networks are missing a key ingredient, which is the very reason why human brains are modular. From Darwin's evolution theory, modular brains have survival advantages over non-modular ones, since modular brains react faster by processing certain functions within local areas. By contrast, modular neural networks do not necessarily have "survival advantages" (e.g., lower losses) over non-modular ones in standard training.

To facilitate the discovery of more modular and interpretable neural networks, we introduce Brain-Inspired Modular Training (BIMT). Inspired by brains, we embed neurons in a geometric space where distances are defined and augment the loss function with a cost proportional to the length of each neuron connection times the absolute value of the connection weight. This obviously encourages *locality* , i.e., keeping neurons that need to communicate as close together as possible. Any Riemannian manifold can be used;

we explore 2D and 3D Euclidean space for easy visualization (see Figure 1). Our work is inspired by the minimum connection cost idea explored in [9–12]. While their focus is on understanding biological neural networks under evolution, our goal is to enhance interpretability of artificial neural networks trained with gradient descent.

We demonstrate the power of BIMT on a broad range of tasks, finding that it can reveal interesting and sometimes unexpected structures. Qualitatively, BIMT-trained networks have modules readily visible to the naked eye, but standard trained networks seem to have much more complicated connections. Quantitatively, we use Newman's method to compute modularity of the network connection graph; BIMT can achieve the highest modularity in all test cases. On symbolic formula datasets, BIMT is able to discover structures such as independence, compositionality, and feature-sharing, which are useful for scientific applications. For classification tasks, we find that BIMT may produce interpretable decision boundaries and features. For algorithmic tasks, we find BIMT to produce tree-like connectivity graphs, not only supporting the group representation argument in [13], but also revealing a (somewhat unexpected) mechanism where multiple modules vote. Although most of our experiments are conducted on fully connected networks for vector inputs, we also conduct experiments demonstrating that BIMT generalizes to other types of data (e.g., images) and architectures (e.g., transformers).

This paper is organized as follows: Section 2 introduces Brain-Inspired Modular Training (BIMT). Section 3 applies BIMT to various tasks, demonstrating its interpretability power. We describe related work in Section 4 and discuss our conclusions in Section 5.

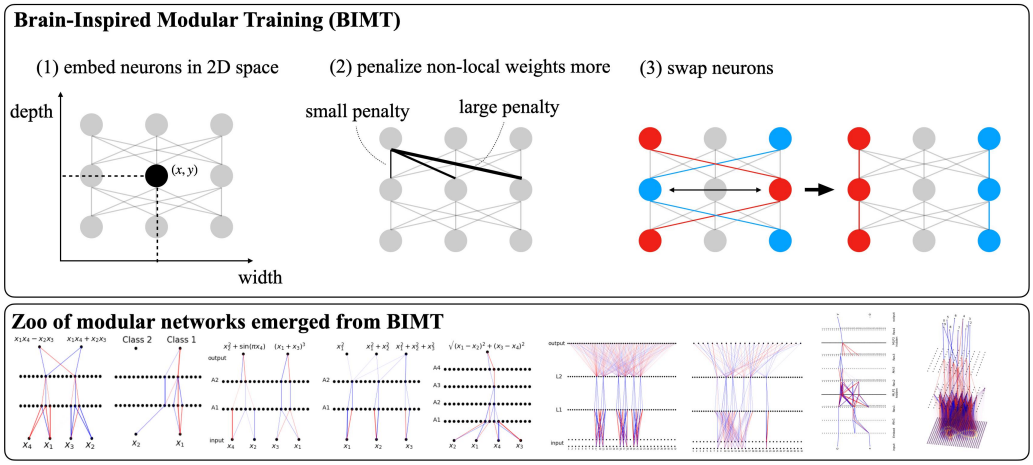


Figure 1. (**Top**): Brain-Inspired Modular Training (BIMT) contains three ingredients: (1) embedding neurons into a geometric space (e.g., 2D Euclidean space); (2) training with regularization, which penalizes non-local weights more; (3) swapping neurons during training to further enhance locality. (**Bottom**): Zoo of modular networks obtained via BIMT (see experiments for details).

2. Brain-Inspired Modular Training (BIMT)

Human brains are modular and sparse, which is arguably the reason why they are so efficient. To make neural networks more efficient, it is desirable to make them modular and sparse, just like our brains. Sparsity is a well-studied topic in neural networks and can be encouraged by including the L_1/L_2 penalty in training or by applying pruning to model weights [14,15]. As for modularity, most research explicitly introduce modules [16,17], but this requires prior knowledge about problem structures. Our motivation question is thus:

Q: What training techniques can induce modularity in otherwise non-modular networks?

In other words, our goal is to let modularity emerge from non-modular networks when possible. In this section, we propose a method called Brain-Inspired Modular Training (BIMT), which explicitly steers neural networks to become more modular and sparse during training. BIMT consists of three key ingredients (see Figure 1): (1) embedding the network

in a geometric space; (2) training to encourage locality and sparsity; and (3) swapping neurons for better locality.

Notation: For simplicity, we describe how to perform BIMT with fully connected networks. Generalization to other architectures is possible. We distinguish between *weight layers* and *neuron layers*. Assuming a fully connected network to have L weight layers, whose ith weight layer $(i = 1, \cdots, L)$ has weights $\mathbf{W}_i \in \mathbb{R}^{n_{i-1} \times n_i}$ and biases $\mathbf{b}_i \in \mathbb{R}^{n_i}$, where n_{i-1} and n_i are the number of neurons incoming to and outgoing from the ith weight layer, the ith $(i = 0, \cdots, L)$ neuron layer has n_i neurons. The input and output dimensions of the whole network are n_0 and n_L, respectively.

Step 1: Embedding the network into a geometric space. We now embed the whole network into a space where the jth neuron in the ith layer is the (i, j) neuron located at $\mathbf{r}_{ij}$. If this is 2D Euclidean space, neurons in the same neuron layer share the same y-coordinate and are uniformly spaced in $x \in [0, A] (A > 0)$. Different neuron layers are vertically separated by a distance $y_* > 0$, so

$$\mathbf{r}_{ij} \equiv (x_{ij}, y_{ij}) = (Aj/n_i, iy_*). \tag{1}$$

The weight that connects the $(i-1, j)$ neuron and the (i, k) neuron has value $w_{ijk} \equiv (\mathbf{W}_i)_{jk}$, the bias at the (i, k) neuron is $b_{ik} \equiv (\mathbf{b}_i)_k$, and its length is defined as

$$d_{ijk} \equiv |\mathbf{r}_{i-1,j} - \mathbf{r}_{ik}|. \tag{2}$$

We will use L_1-norm, giving $d_{ijk} = |x_{i-1,j} - x_{ik}| + y_*$, but other vector norms can also be used. For example, L_2-norm gives $d_{ijk} = \left(|x_{i-1,j} - x_{ik}|^2 + y_*^2\right)^{1/2}$.

Step 2: Imposing regularization that encourages locality. We define the connection cost for weight and bias parameters of the whole network to be

$$\ell^w = \sum_{i=1}^{L} \sum_{j=1}^{n_i} \sum_{k=1}^{n_{i+1}} d_{ijk} |w_{ijk}|, \quad \ell^b = \sum_{i=1}^{L} \sum_{j=1}^{n_i} y_* |b_{ij}|. \tag{3}$$

When training for a particular task, in addition to the prediction loss ℓ^{pred}, we include ℓ^w and ℓ^b as regularizations:

$$\ell = \ell^{\text{pred}} + \lambda(\ell^w + \ell^b), \tag{4}$$

where λ is the strength of the regularization. Without loss of generality, we can set $y_* = 1$, leaving only the two hyper-parameters λ and A. Setting $A = 0$ reduces to standard L_1 regularization, which solely encourages sparsity. $A > 0$ further encourages locality, in addition to sparsity.

Step 3: Swapping neurons for better locality. We encourage locality (reduction of ℓ^w) not only by updating weights via gradient descent, but also by swapping two neurons in the same neuron layer (i.e., swapping corresponding incoming/outgoing weights), when this reduces ℓ^w. Gradient descent (continuous search) can get stuck at bad local minima where non-local connections are still present (see Figure 2c), while swapping (discrete search) can avoid this (see Figure 2e). Such swapping leaves the function implemented by the whole network (hence, ℓ^{pred}) unchanged but improves locality (see Figure 1, right). However, trying every possible permutation is prohibitively expensive. We assign each neuron (i, j) a score s_{ij} to indicate its importance:

$$s_{ij} = \sum_{p=1}^{n_{i-1}} |w_{ipj}| + \sum_{q=1}^{n_{i+1}} |w_{i+1,jq}|, \tag{5}$$

which is the sum of (absolute values) of incoming and outgoing weights. We sort neurons in the same layer based on their scores and define neurons with the top k-scores as "important" neurons. For each important neuron, we swap it with the neuron in the same layer, causing the greatest decrease in l^w if it helps. Since swaps are somewhat expensive, requiring

$O(nkL)$ computations, we implement swaps only every $S \gg 1$ training steps. We allow swaps also of input and output neurons, if not stated otherwise.

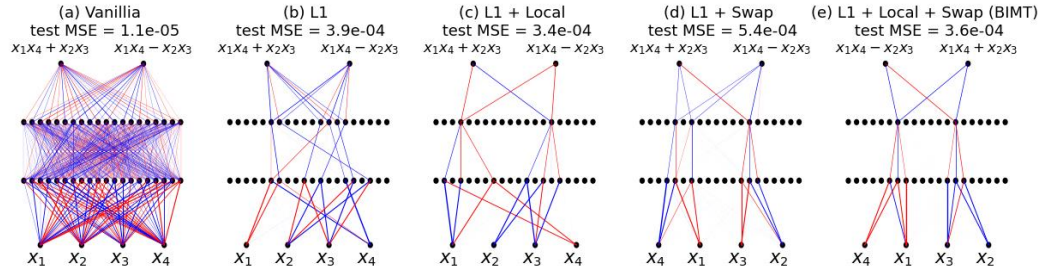


Figure 2. The connectivity graphs of neural networks when trained with different techniques for a regression problem (blue/red denote positive/negative weights). Our proposed BIMT = L_1 regularization (not novel) + local regularization (novel) + swap (novel). BIMT finds the simplest circuit (**e**) which clearly contains two parallel modules, with a moderate sacrifice in test loss compared to vanilla (**a**), but with lower loss than for mere L_1 regularization (**b**). Note that swapping aims to reduce the local connection cost, so all of (**c**–**e**) encourage locality.

BIMT = L_1 + Local + Swap. To summarize, BIMT means local L_1 regularization with swaps. Both "local" and "swap" are novel contributions of this paper, while L_1 regularization is quite standard. If one wants to ablate "local" or "swap", one can set $A = 0$ to remove "local" or set $S \to \infty$ to remove "swap". Our experience is that the joint use of "local" and "swap" usually gives the most interpretable networks. As a simple case, we compare BIMT to baselines on a regression problem, shown in Figure 2. On top of L_1, although using "local" or "swap" alone gives reasonably interpretable networks, the joint use of both produces the most interpretable network (at least visually). Although using L_1 alone leads to a reasonably sparse network, the network is neither modular nor optimally sparse (see Appendix A for pruning results).

Connectivity Graphs. As in Figure 2 and throughout the paper, we use connectivity graphs to visualize neural network structures. For visualization purposes, we normalize weights by the max absolute value in the same layer (so the normalized values lie in range $[-1, 1]$). A weight is displayed as a line connecting two neurons, with its thickness proportional to its normalized value, and its color set to blue (red) if the value is positive (negative). Note that we draw all weights and do not explicitly ignore small weights. The reason connectivity graphs appear sparse is because naked eyes cannot identify very thin lines.

3. Experiments

In this section, we apply BIMT to a wide range of applications. In all cases, BIMT can result in modular and sparse networks, which immediately provide interpretability on the microscopic level and the macroscopic level. At the microscopic level, we can understand which neurons are useful, what each useful neuron is doing, and where/how information of interest is located/computed. At the macroscopic level, we can understand relations between different modules (e.g., in succession or in parallel) and how they cooperate to make the final prediction. From Sections 3.1–3.3, we train fully connected neural networks with BIMT for regression, classification, and algorithmic tasks. In Section 3.4, we show that BIMT can generalize to transformers and demonstrate it through in-context learning. In Section 3.5, we demonstrate that BIMT can easily go beyond vector-type data to tensor-type data (e.g., images). In general, BIMT achieves interpretability with either no drop or a modest drop in performance, summarized in Table 1. We also report the modularity metric in Table 2, computed using Newman's method provided by the NetworkX package, showing that BIMT gives the most modular networks in all examples. Our baseline training methods are training with no penalty or with the L_1 [18] penalty with the Adam

optimizer [19]. Results are averaged over five random seeds. All experiments are runnable on a CPU (M1), usually within minutes (at most two hours).

Table 1. BIMT achieves interpretability with no or a modest performance drop.

Dataset	Symbolic (a)	Symbolic (b)	Symbolic (c)	Two Moon	Modular Addition	Permutation	In-Context	MNIST
metric	loss	loss	loss	accuracy	accuracy	accuracy	loss	accuracy
Vanilla	$(5.2 \pm 1.0) \times 10^{-3}$	$(1.1 \pm 0.4) \times 10^{-5}$	$(1.2 \pm 0.5) \times 10^{-4}$	$(100.0 \pm 0.0)\%$	$(100.0 \pm 0.0)\%$	$(100.0 \pm 0.0)\%$	$(7.8 \pm 1.8) \times 10^{-5}$	$(98.5 \pm 0.2)\%$
L_1 penalty	$(7.9 \pm 0.8) \times 10^{-3}$	$(1.2 \pm 0.3) \times 10^{-5}$	$(1.8 \pm 0.4) \times 10^{-4}$	$(100.0 \pm 0.0)\%$	$(100.0 \pm 0.0)\%$	$(100.0 \pm 0.0)\%$	$(7.2 \pm 1.0) \times 10^{-5}$	$(98.4 \pm 0.3)\%$
BIMT (ours)	$(7.4 \pm 1.0) \times 10^{-3}$	$(8.0 \pm 1.5) \times 10^{-5}$	$(1.3 \pm 0.3) \times 10^{-3}$	$(100.0 \pm 0.0)\%$	$(100.0 \pm 0.0)\%$	$(100.0 \pm 0.0)\%$	$(1.8 \pm 0.4) \times 10^{-4}$	$(98.2 \pm 0.3)\%$

Table 2. BIMT achieves the highest modularity for all tasks.

Dataset	Symbolic (a)	Symbolic (b)	Symbolic (c)	Two Moon	Modular Addition	Permutation	In-Context	MNIST
Vanilla	0.207 ± 0.025	0.151 ± 0.015	0.145 ± 0.023	0.291 ± 0.032	0.194 ± 0.028	0.129 ± 0.012	0.085 ± 0.014	0.073 ± 0.005
L_1 penalty	0.512 ± 0.042	0.456 ± 0.035	0.328 ± 0.027	0.289 ± 0.016	0.435 ± 0.028	0.371 ± 0.031	0.083 ± 0.010	0.099 ± 0.008
BIMT (ours)	0.581 ± 0.042	0.543 ± 0.063	0.392 ± 0.021	0.338 ± 0.039	0.535 ± 0.027	0.634 ± 0.043	0.161 ± 0.013	0.238 ± 0.021

3.1. Symbolic Formulas

Symbolic formulas are prevalent in scientific domains. In recent years, as increasingly more data are collected from experiments, it is desirable to distill symbolic formulas from experimental data, a task called symbolic regression [20]. However, symbolic regression usually faces an all-or-nothing situation, i.e., either succeeds gloriously or fails miserably. Consequently, a tool supplementary to symbolic regression is called for, which can robustly reveal the high-level structure of formulas. We show below that BIMT can discover such structures in formulas.

We consider the task of predicting $\mathbf{y} = (y_1, \cdots, y_{d_o})$ from $\mathbf{x} = (x_1, \cdots, x_{d_i})$, where $y_i = f_i(\mathbf{x})$ are symbolic functions. We randomly sample each $\mathbf{x}_i$ from $U[-1, 1]$ and compute $y_i = f_i(\mathbf{x})$ to generate the dataset. We use fully connected networks with SiLU activations (architectures shown in Figure 3) and training networks using the MES loss with the Adam optimizer with learning rate 10^{-3} for 20,000 steps, while choosing $A = 2$, $y_* = 0.1$, $k = 6$, and $S = 200$. We schedule λ as such: $(10^{-3}, 10^{-2}, 10^{-3})$ for (5000, 10,000, 5000) steps.

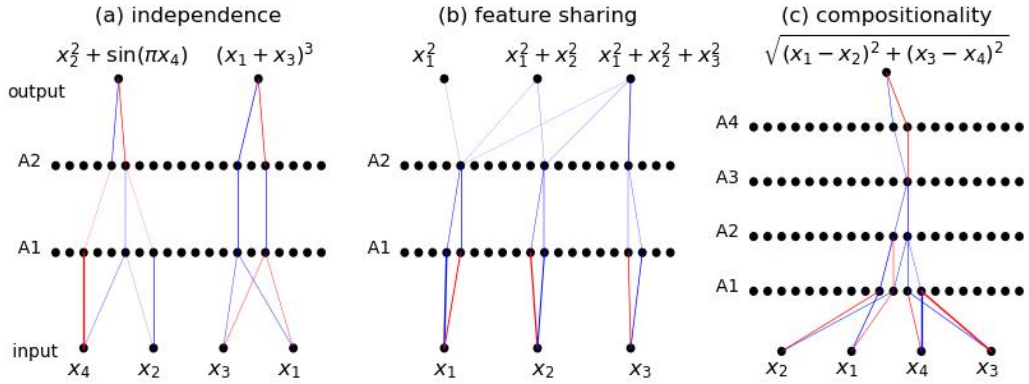


Figure 3. The connectivity graphs of neural networks trained with BIMT to regress symbolic formulas (blue/red lines stand for positive/negative weights). For symbolic formulas with modular properties, e.g., independence, shared features, or compositionality, the connectivity graphs display modular structures revealing these properties.

We apply BIMT to several formulas, each of which has certain modular properties, as shown in Figure 3. (a) **Independence**. $y_1 = x_2^2 + \sin(\pi x_4)$ is independent of x_1 and x_3, while $y_2 = (x_1 + x_3)^3$ is independent of x_2 and x_4. As desired, BIMT results in a network split into two parallel modules independent of each other, one only involving (x_1, x_3), the other only involving (x_2, x_4). (b) **Feature Sharing**. For targets $(y_1, y_2, y_3) = (x_1^2, x_1^2 + x_2^2, x_1^2 + x_2^2 + x_3^2)$,

learning shared features (x_1^2, x_2^2, x_3^2) is beneficial for predicting all targets. Indeed, in the neuron layer A2, the only three active neurons correspond to these shared features (see Appendix B). (c) **Compositionality**. Computing $y = \sqrt{(x_1 - x_2)^2 + (x_3 - x_4)^2} \equiv \sqrt{I}$ requires computing I first, which is an important intermediate quantity. We find that the only active neuron in layer A3 has activations highly correlated with I. Although one might worry that these extremely sparse networks could severely under-fit, we show that fitting is reasonably good in Appendix B.

3.2. Two Moon Classification

Interpretable decision boundaries help to make classification trustworthy. Moreover, decision boundaries with fewer pieces are more likely to generalize better. Therefore, it is desirable that neural networks used for classifications are sparse and interpretable.

We apply BIMT to the toy Two Moon dataset [21]. The architecture is shown in Figure 4 (the final softmax layer is not shown), with the same training details used in Section 3.1, with the only difference being the use of cross-entropy loss. We choose $\lambda = 0.01$ and $A = 2$. The evolution of the neural network is shown in Figure 4. Starting from a (randomly initialized) dense network, the network becomes increasingly sparse and modular, ending up as a network with only six useful hidden neurons. We can roughly split the training process into three phases: (i) in the first phase (steps 0 to 1000), the neural network mainly aims to fit the data while slightly sparsifying the network; (ii) in the second phase (steps 1000 to 3000), the neural network sparsifies the network in a symmetric way (both outputs of Classes 1 and 2 have neurons connecting to them); (iii) in the third phase (steps 3000 to end), the network prunes itself to become asymmetric, with useful neurons only connecting to Class 1 output. In Appendix C, we interpret what each weight is doing by editing them (zeroing) and see how this affects decision boundaries.

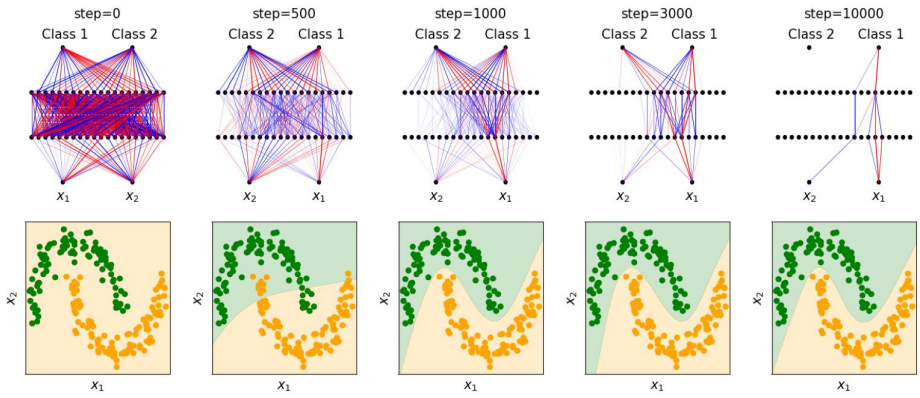
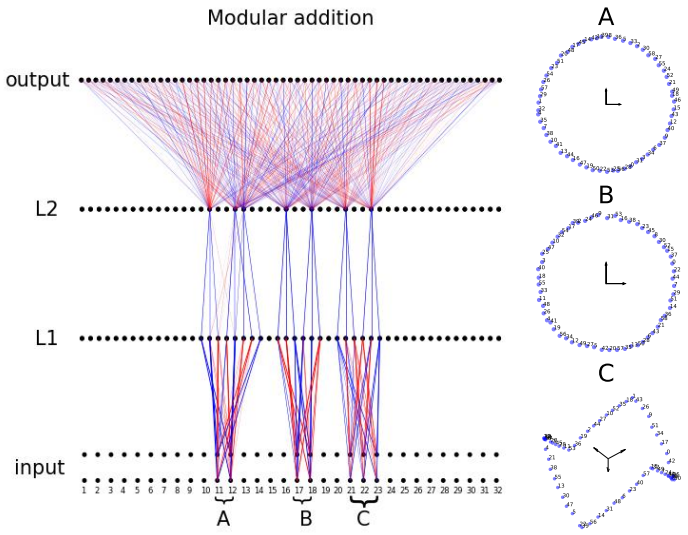


Figure 4. (**Top**): Evolution of network structures trained with BIMT on the Two Moon dataset. Blue and red lines stand for postive and negative weights, respectively. (**Bottom**): Evolution of decision boundaries.

3.3. Algorithmic Datasets

Algorithmic datasets are ideal for benchmarking mechanistic interpretability methods because they are mathematically well understood. Consider a binary operation $a \circ b = c$ (a, b, c are discrete and treated as tokens), in which a neural network is tasked with predicting c from embeddings of a and b. For modular addition, Ref. [22] discovers that ring-like representations emerge in training. Ref. [23] reverse-engineered these networks, finding that the network internally implements trigonometric identities. For more general group operations, Ref. [13] suggests that representation theory is key for neural networks to generalize. However, in these papers, it is usually not obvious which neurons are useful or what the overall modular structure of the network is. Since BIMT explicitly optimizes modularity, it is able to produce networks that self-reveal their structure.

Modular addition. The task is to predict c from (a, b), where $a + b = c \pmod{59}$. Each token a is embedded as a $d = 32$-dimensional vector $\mathbf{E}_a$, initialized as a random normal vector at initialization and trainable later. The concatenation of $\mathbf{E}_a$ and $\mathbf{E}_b$ is fed to a two-hidden-layer MLP, shown in Figure 5. We split train/test 80/20%. We train the network with BIMT with cross-entropy loss using the Adam optimizer (lr $= 10^{-3}$) for 20,000 steps. We choose $A = 2$, $y_* = 0.5$, $k = 30$, and $S = 200$. We schedule λ as such: $(0.1, 1, 0.1)$ for (5000, 10,000, 5000) steps.



Knockout	Accuracy
None	100.00%
A	15.25%
B	29.33%
C	33.67%
A, B	3.39%
A, C	5.08%
B, C	10.28%
A, B, C	1.69%
A11	47.11%
A12	46.51%
B17	50.47%
B18	51.42%
C21	77.10%
C22	73.60%
C23	78.17%
All but A, B, C	100.00%

Figure 5. MLP trained with BIMT for modular addition. (**Left**): the final connectivity graph is tree-like, demonstrating three parallel modules (voters); middle: the representations of each module in the input layer. Blue and red lines stand for postive and negative weights, respectively. (**Right**): ablation results, which imply a voting mechanism. The input layer contains embeddings of two tokens, which overlap each other but are drawn to be vertically separated.

After training, the network looks like a tree with three roots (A, B, C), shown in Figure 5. We visualize embeddings corresponding to these roots (modules), finding that the token embeddings form circles in 2D (A, B) and a bow tie in 3D (C). In contrast to Refs. [22,23], where post-processing (e.g., principal component analysis) is needed to obtain ring-like representations, the ring structures here automatically align to privileged bases, which is probably because embeddings are also regularized with L_1. To evaluate how these parallel modules are important for making predictions, we compute accuracy after knocking out some of them. The result is quite surprising: knocking out one of the modules can severely degrade the performance (from 100% to 15.25%, 29.33%, or 33.67% for knocking out A, B, or C). This means that modules are cooperating together to make predictions correct, similar to majority voting for error correction. To verify the universality of this argument, we include more tree graphs for perturbed initializations and different random seeds in Appendix D.

Permutation group. The task is to predict c from (a, b), where a, b, c are elements in the 24-element group (the permutation of 4 objects) S_4, and $ab = c$. Our training is the same as for modular addition. Figure 6 shows that, after training with BIMT, the network is quite modular. Notice that there are only nine active components in the embedding layer, exactly agreeing with the representation theory argument of Ref. [13] (S_4 has a 3×3 matrix representation). In Figure 6 (right), we show how each embedding neuron is activated by each group element, revealing that BIMT has discovered a crucial group-theoretical structure! Note that we have normalized these embeddings when plotting: denote the value of the ith neuron and the jth token as e_{ij}. The normalized embedding is defined as $\tilde{e}_{ij} = e_{ij} / \left(\max_j |e_{ij}|\right)$. In particular, neuron 22 is the sign neuron ($1/-1$ for even/odd

permutations), and other active neurons correspond to subgroups or cosets (see further analysis in Appendix E).

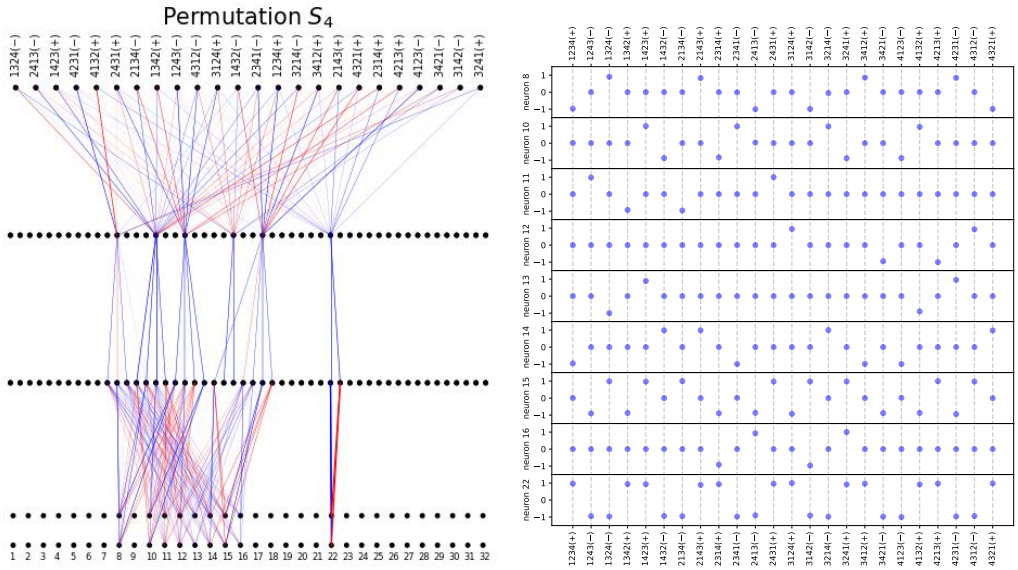


Figure 6. Apply BIMT to MLP on the Permutation S_4 dataset. (**Left**): the final connectivity graph, with only nine active embedding neurons. The input layer contains embeddings of two tokens, which overlap each other but are drawn to be vertically separated. Blue and red lines stand for postive and negative weights, respectively. (**Right**): the nine active neurons correspond to group representations of S_4, whose values are normalized into the range $[-1, 1]$. In particular, neuron 22 is the sign neuron ($1/-1$ for even/odd permutations).

3.4. Extension to Transformers: In-Context Linear Regression

So far, we have demonstrated the effectiveness of BIMT for MLPs. We can generalize BIMT to transformers [24]: we simply apply BIMT to linear layers in transformers (see details in Appendix F). Following the setup of Ref. [25], we now study in-context linear regression. Linear regression aims to predict y from $\mathbf{x} \in \mathbb{R}^d$, assuming that we know training data $(\mathbf{x}_i, \mathbf{y}_i)$ $(i = 1, \cdots, n)$, where $\mathbf{y}_i = \mathbf{w} \cdot \mathbf{x}_i$. In-context linear regression aims to predict y from the sequence $(\mathbf{x}_1, y_1, \cdots, \mathbf{x}_n, y_n, \mathbf{x})$, which is called in-context learning because the unknown weight vector $\mathbf{w}$ needs to be learned in context, i.e., when the transformer runs in test time rather than when it is trained. To make things maximally simple, we choose $d = 1$ (the weight vector degrades to a scalar) and $n = 1$.

The architecture is displayed in Figure 7, where, for clarity, we only show the last block, ignoring its attention dependence on previous blocks. The embedding size is 32, the number of transformer layers is 2 (each layer containing an attention layer and an MLP), and the number of heads is 1. We draw $w \in U[1, 3]$ (Instead, we can investigate $w \sim U[-1, 1]$, which has a singularity issue (please see Appendix F for details).) and $x \in U[-1, 1]$ to create datasets. With MSE loss, we train with the Adam optimizer (lr: 1×10^{-3}) for 4×10^4 steps ($\lambda = 0.001, 0.01, 0.1, 0.3$ each for 10^4 steps). We choose $A = 2$, $y_* = 0.5$, $k = 30$, and $S = 200$.

It is shown in Ref. [25] that w is linearly encoded in neural network latent representations, but it is not easy to track where this information is located. From Figure 7 (left), it is immediately clear which neurons are useful (active). In Figure 7 (top right), we show that the prediction is quite good, even though the network has become extremely sparse. We examine active neurons in the Res2 layer, finding that several neurons are correlated with the weight scalar, although no one neuron alone can determine the weight scalar perfectly. In Figure 7 (right middle and bottom), we show that pairs of neurons (8 and 9, 11 and 19) implicitly encode information about the weight scalar in nonlinear ways.

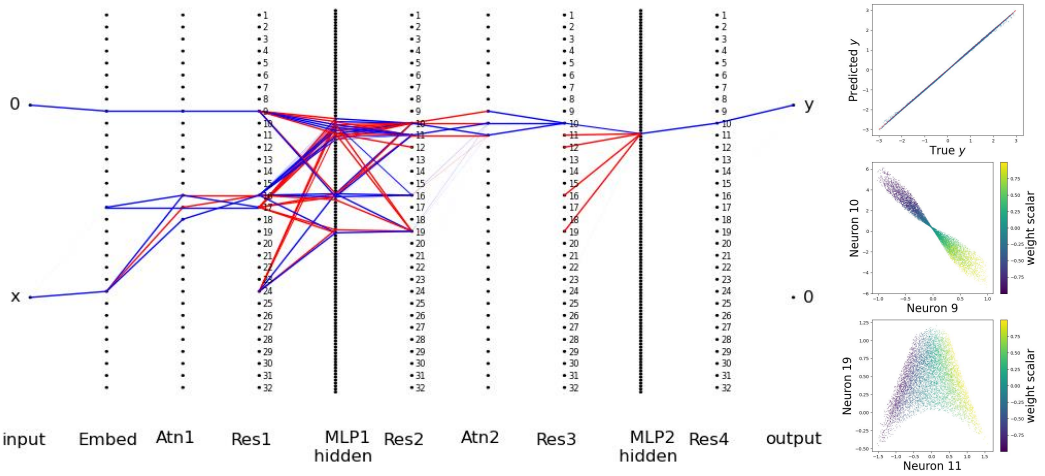


Figure 7. Application of BIMT to transformers during in-context learning linear regression. Left: the connectivity graph of the transformer after training. Only the last block is shown, which takes in $[0, x]$ to predict $[y, 0]$. Blue and red lines stand for postive and negative weights, respectively. Right top: predicted vs true y. Right middle and bottom: neurons in the Res2 layer contain the information about the weight scalar, encoded non-linearly.

3.5. Extension to Tensor Data: Image Classification

So far, we have always embedded neural networks into 2D Euclidean space, but BIMT can be used in any geometric space. We now consider a minimal extension: embedding neural networks into a 3D Euclidean space. For 2D image data, to maintain their local structure, it is better to leave them as 2D rather than flatten them to 1D. As a result, an MLP for 2D image data should be embedded in 3D, as shown in Figure 8. The only modification for BIMT is that, when computing distances, we use 3D rather than 2D vector norms.

We train with MSE loss and use the Adam optimizer (lr = 1×10^{-3}) for 4×10^4 steps ($\lambda = 0.001, 0.01, 0.1, 0.3$ each for 10^4 steps). We choose $A = 2$, $y_* = 0.5$, $k = 30$, and $S = 200$. We disable swaps of input pixels. We show the evolution of the network in Figure 8. Starting from a dense network, the network becomes more modular and sparser over time. Notably, the receptive field shrinks for the input layer, since BIMT learns to prune away peripheral pixels that always equal zero. Another interesting observation is that most of the weights in the middle layer are negative (colored red), while most of the weights in the last layer are positive (colored blue). This suggests that the middle layer is not adopting the strategy of pattern matching, but rather *pattern mismatching*. Pattern matching/mismatching means that, if an image has/does not have these patterns, it is more likely to be, for example, an 8. We visualize features in Appendix G, where we also include the results for MLPs with different depths. Moreover, in the output layer, Classes 1 and 7 are automatically swapped to become neighbors, probably due to their similarity. In future work, we would like to compare our method with convolutional neural networks (CNNs). It might be best to combine CNNs with BIMT, since CNNs guarantee the locality of inputs, while BIMT encourages locality of model internals.

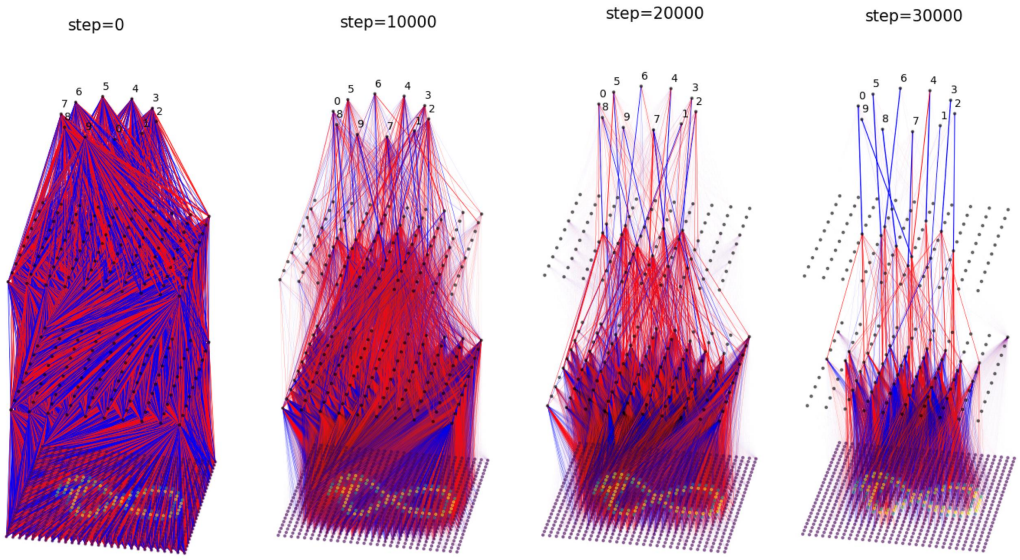


Figure 8. Application of BIMT to 3D MLP on MNIST. From left to right: connectivity graph evolution. Blue and red lines stand for postive and negative weights, respectively.

4. Related Work

Mechanistic Interpretability (MI) is an emerging field that aims to mechanically understand how neural networks work. Various modules/circuits are identified from neural networks via reverse engineering, including image circuits [1], induction heads [2], computational quanta [3], transformer circuits [4], factual associations [26], and heads in the wild [5], although superposition [27] makes interpretability more complicated. A generalization puzzle called grokking [28] has also been understood by reverse-engineering neural networks [13,22,23,29].

Modularity in neural networks can help generalization in transfer learning [16] and can enhance interpretability [1]. Non-modular neural networks trained in standard ways are shown to present an imperfect extent of modularity [30–32]. Modular networks explicitly use trainable modules in constructing neural networks [17,33], but this inductive bias may require prior knowledge about the tasks. The multi-head attention layer in transformers lies in the category of explicitly introducing modularity. By contrast, this work does not explicitly introduce modules, but rather lets modules emerge from non-modular networks with the help of BIMT.

Pruning can lead to sparse and efficient neural networks [14,15,34,35], usually achieved by L_1 or L_2 regularization and thresholding small weights to zero. BIMT borrows the L_1 regularization technique for sparsity but improves modularity by making the L_1 regularization distance-dependent.

Analogy between neuroscience and neural networks has existed for a long time in the literature [36,37]. Although biological and artificial neural networks may not have the same low-level learning mechanisms [38], we can still borrow high-level ideas and concepts from neuroscience to design more interpretable artificial neural networks, which is the goal of this work. The minimal connection cost idea has been explored in [9–12], where an evolutionary algorithm is applied to evolve tiny networks. By contrast, our method is more aligned with modern machine learning, i.e., gradient-based optimization and broader applications.

Neuroscience-inspired learning. Since the literature of neuroscience is vast [39], we will not review it here, but we do want to highlight some progress in neuroscience that can potentially inspire and improve current machine learning systems. The study of neural codes [40], neural information processing and transmission [41–43], and neural spikes [44] may provide tools and language towards the study and design of artificial neural networks.

5. Conclusions and Discussion

We have proposed Brain-Inspired Modular Training (BIMT), which explicitly encourages neural networks to be modular and sparse. BIMT is a principled idea that could be generalized to many types of data and network architectures. Tested on several relatively small-scale tasks, we show its ability to provide interpretable insights for these problems. In future studies, we would like to see if this training strategy remains valid for larger-scale tasks, e.g., large language models (LLMs). In particular, can we fine-tune LLMs with BIMT to make them more interpretable? Moreover, BIMT achieves interpretability at the price of slight performance degradation. In fact, it is known that modularity might improve performance in continual learning when the tasks have modular and compositional structures [12]. To understand the benefits and limitations of BIMT, further studies will require careful disentanglement of network sparsity and modularity and rigorous definition of task modularity. We would like to improve BIMT such that interpretability and performance are achieved at the same time.

Broader Impacts. We believe that building interpretable neural networks will make AI more controllable, more reliable, and safer. However, like other AI interpretability research, the controllability brought by interpretability should be regulated, making sure the technology is not misused.

Limitations. (1) This work deals with small-scale toy problems, where neural networks can be easily visualized. It is still unclear whether this method remains effective for larger-scale problems. (2) BIMT requires users to define the embedding geometric space, which may require some prior knowledge about the task. Arguably, this might be a feature not a bug, especially for neuromorphic computing. (3) The swapping step may incur additional overhead. Additionally, swapping is currently implemented layer by layer, so global topological problems cannot be resolved via swapping. (4) Visualizing the connectivity graph of neural networks is only efficient and useful for small networks. (5) There is generally a trade-off between accuracy and simplicity/interpretability, which is also true for BIMT. (6) The connectivity graphs can be quite sensitive to random seeds, although they share common global structures, which might be enough for interpretation. Please see Appendix D for more details.

Future Directions. Two major concerns of this work are scalability and sensitivity. We would like to mention our plans for how to address these two issues in future work. To improve scalability, we may need to explicitly introduce some notion of hierarchy (beyond modularity), because hierarchy can potentially decompose complicated modules into simpler sub-modules, hence making scaling easier. To reduce sensitivity, some theoretical research is first needed to characterize equivalent classes of neural networks so that we can better understand whether the changes caused by perturbation are superficial (in the same equivalent class) or fundamental (in different equivalent classes).

Author Contributions: Conceptualization, Z.L. and M.T.; Methodology, Z.L. and M.T.; Software, Z.L. and E.G.; Writing—original draft, Z.L.; Writing—review & editing, Z.L. and M.T. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by The Casey Family Foundation, the Foundational Questions Institute, the Rothberg Family Fund for Cognitive Science, the NSF Graduate Research Fellowship (Grant No. 2141064), and IAIFI through NSF grant PHY-2019786.

Institutional Review Board Statement: Not applicable.

Data Availability Statement: The data presented in this study can be reproduced with code openly available at https://github.com/KindXiaoming/BIMT accessed on 1 November 2023.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A. Pruning

Although the original goal of BIMT is to make neural networks modular and interpretable, the fact that it can make networks sparse is also useful for pruning. Here, we

show the benefits of BIMT in terms of pruning on a toy example (in Figure 2). The task is to fit $(y_1, y_2) = (x_1x_4 + x_2x_3, x_1x_4 - x_2x_3)$ from (x_1, x_2, x_3, x_4) with a two-hidden-layer MLP. As in Figure 2, we test five training methods: vanilla, L1, L1 + Local, L1 + Swap, and BIMT (L1 + Local + Swap). For each trained network, we sort their parameters (including weights and biases) from small to large (in magnitudes), defining a threshold below which parameters are set to zero. Given a threshold, we can compute the number of unpruned parameters N_u, as well as test loss ℓ_{test}. By sweeping the threshold, we obtain a trade-off frontier, as shown in Figure A1. Note that, in this plot, the lower left the curve goes, the better the pruning. Therefore, BIMT and L1 + Local achieve the best pruning results, even better than L1 (which is standard in pruning). We leave the full investigation of BIMT as a pruning method for future work.

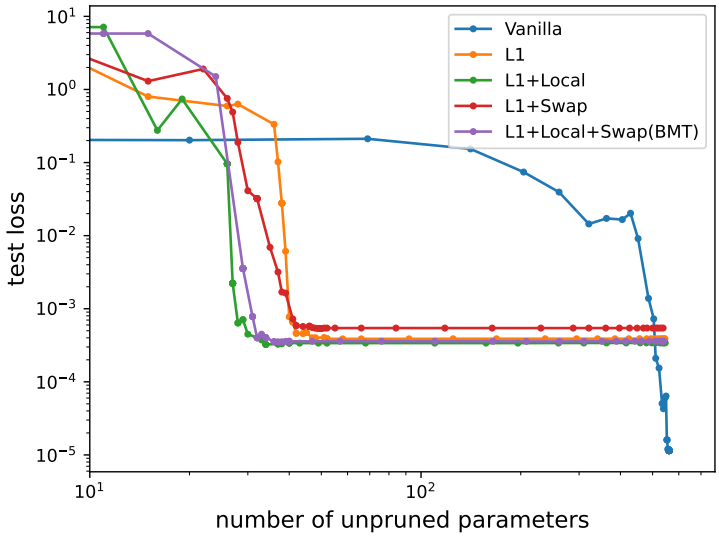


Figure A1. In a toy regression problem (see Figure 2), BIMT and L1 + Local achieve the best pruning results, even better than L1 regularization alone (a standard training method for pruning).

Appendix B. Symbolic Formulas

Appendix B.1. How Good Are the Predictions?

Since the connectivity graphs in Figure 3 are extremely sparse, one may suspect that these sparse networks severely under-fit. We show that this is not the case, since the (test) losses are quite low, and sample-wise prediction errors are quite small, as shown in Figure A2. The explanation is that SiLU activations (and other similar activations) are surprisingly effective. In fact, in the following, we reverse-engineer how these symbolic functions can be approximated with very few parameters.

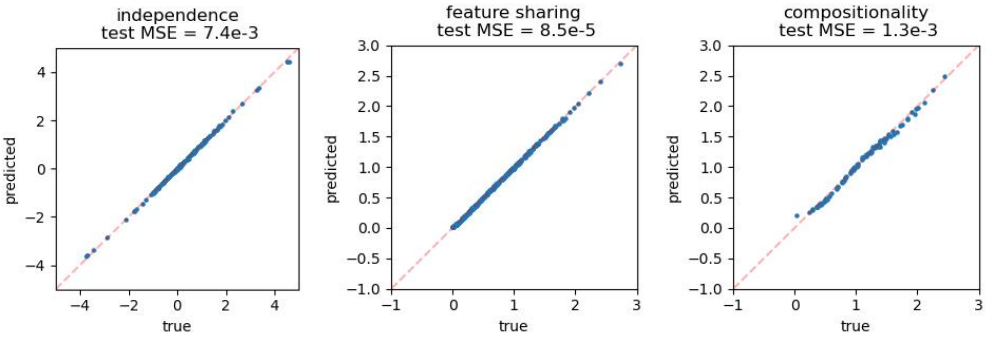


Figure A2. Although the networks are extremely sparse in Figure 3, their predictions are quite good.

Appendix B.2. Reverse-Engineering Implementation of Formulas

Once we obtain the sparse connectivity graphs in Figure 3, we can easily reverse-engineer how neural networks approximate these analytical functions with linear operations and SiLU activation functions $\sigma(x) = x/(1+e^{-x})$. If not stated otherwise, the approximations below hold for $x \in [-1, 1]$.

(a) Independence

$$\begin{aligned} x^2 &\approx -1.33x + 1.84\sigma(1.53x), \\ \sin(x) &\approx -2.27x + 1.72\sigma[-0.91\sigma(-3.24x + 1.54) + 2.63] - 2.10, \\ x^3 &\approx 2.30\sigma[3.34\sigma(0.90x - 0.51) - 0.46] - 2.27\sigma[3.00\sigma(-0.87x - 0.19) - 1.07]. \end{aligned} \tag{A1}$$

(b) Feature Sharing:

$$x^2 \approx 0.35\sigma[1.41\sigma(2.64x) + 1.99\sigma(-1.80x + 0.05)]. \tag{A2}$$

(c) Compositionality:

$$1.60\sqrt{x - 1.24} \approx 0.80\sigma(1.04x) - 1.18\sigma(-2.26x + 2.44) - 0.18, \quad x \in [1.24, 3.66]. \tag{A3}$$

Appendix B.3. Intermediate Quantities

In the compositionality example, $y = \sqrt{(x_1 - x_2)^2 + (x_3 - x_4)^2} \equiv \sqrt{I}$, we argue that there is an intermediate quantity I contained in the network. Figure A3 shows this to be neuron 11. The relationship between this neuron and the NN output is seen to accurately approximate the square root function.

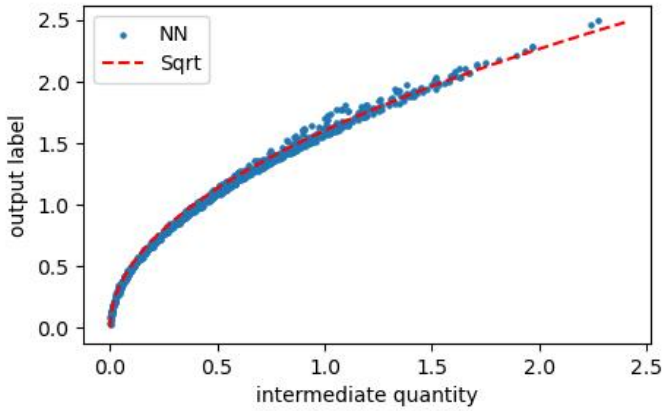


Figure A3. Verifying the existence of an intermediate quantity.

Appendix C. Two Moon Classification

We showed in Section 3.2 that a very sparse network is able to classify the Two Moon dataset. Since the active weights are so few, we are able to interpret each of them by removing them (setting the weight value to be 0). We show how the decision boundaries change under removing one of the weights (marked as a cross) in Figure A4. It is clear that every weight is necessary for prediction, since removing any of them can lead to false classifications. We can also write the symbolic formula for the network ($\sigma(x) = x/(1+e^{-x})$) as:

$$\begin{aligned} p(\text{green}|x_1, x_2) =& \frac{\exp(s(x_1, x_2))}{1 + \exp(s(x_1, x_2))} \\ s(x_1, x_2) =& 5.16\sigma(1.44x_2 + 1.43) - 6.36\sigma(-0.86\sigma(1.44x_2 + 1.43) + 1.72\sigma(1.34x_1) \\ & - 2.47\sigma(-3.29x_1 - 0.17) + 1.99\sigma(2.32x_1 - 2.07)). \end{aligned} \tag{A4}$$

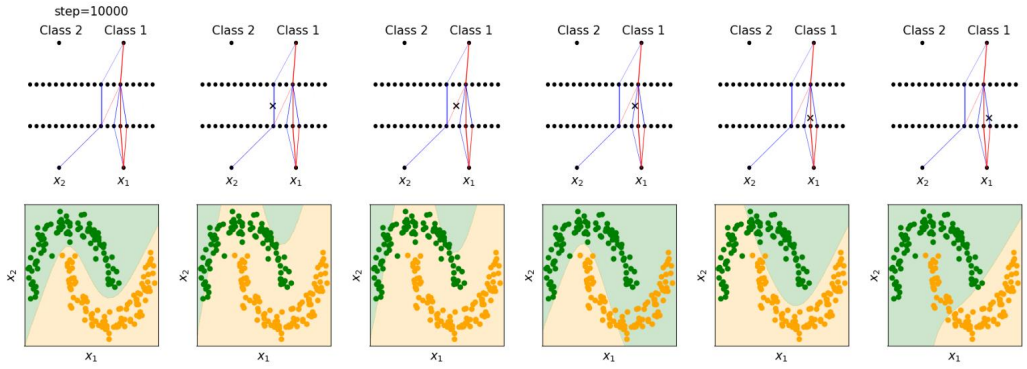


Figure A4. For the Two Moon dataset, we interpret what each weight is doing by setting it to zero (marked with an 'x' in the top panel) and visualizing the resulting decision boundary (bottom).

Appendix D. Modular Addition

Sensitivity (add noise). In the modular addition task in Section 5, we investigate the sensitivity of the module structures to small perturbations during their initializations. To do this, we first initialize a model's parameters using a fixed random seed and then add zero-mean Gaussian noise with varying standard deviations σ. In Figure A5 (top) presented in the graphs, the "noise" values refer to the standard deviation of the Gaussian noise added to the model's parameters during initialization.

We find that small perturbations to initialization have sizable impacts on the final model. Even the least perturbed model ($\sigma = 10^{-6}$) is quite different from the base model going from layer 2 to the output, although the modules are mostly in the same positions. We conjecture that the training dynamics have many branching points, where a small perturbation can lead to quite different basins. Luckily, these basins all have similar tree structures, amenable to be interpreted.

More tree graphs. We also investigate the behavior of the model with different random seeds for initialization and see a diverse pattern of module formation. Although they are different in detail, there are some universal features: (1) the number of modules is odd most times, supporting the argument of (majority) voting; (2) between layer 1 and layer 2, many copies of the same motif emerge, which connects three neurons in L1 to one neuron in L2.

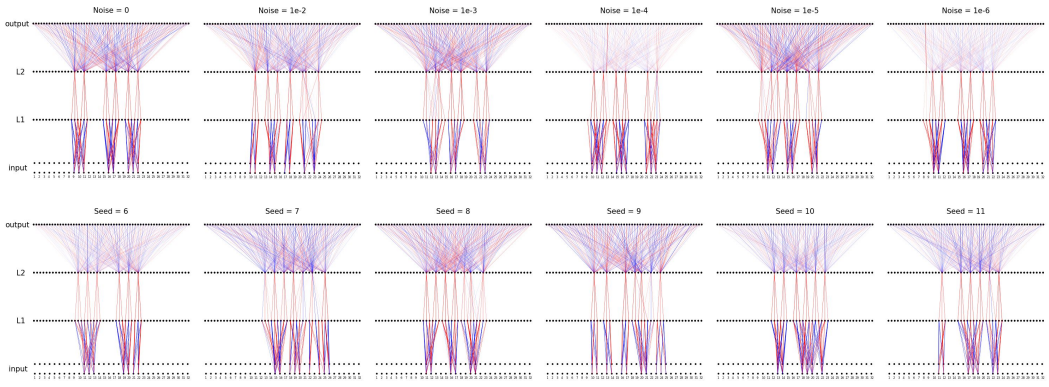


Figure A5. For the modular addition task, we test what happens when we add small perturbations to the model (**top**) and when we initialize the the parameters using different random seeds (**bottom**).

Appendix E. Reverse-Engineering Learned S_4 Embeddings

Appendix E.1. Visualizing Neurons with Cayley Graphs

In Figure 6, we find that there are only nine active embedding neurons. For each of them except for the sign neuron, only a subset of group elements is non-zero and,

interestingly, non-zero elements are close to $+1$ or -1. Therefore, to visualize what each neuron is doing, we can highlight its active group elements on a Cayley graph of the permutation group S_4, as shown in Figure A6, where green/orange/no circle means $+1/-1/0$, respectively. Red and blue arrows represent two generators 4123 and 2314. There are a few interesting observations: (1) Moving circles along blue arrows for neuron 8 gives neuron 10; (2) For neuron 14 (15), the inside square (octagon) activates to $+1$, while the outside square (octagon) activates to -1. Moreover, both of them are closed under red arrows. (3) For neurons 11, 12, 13, 16, they display similar structures, up to translations and rotations.

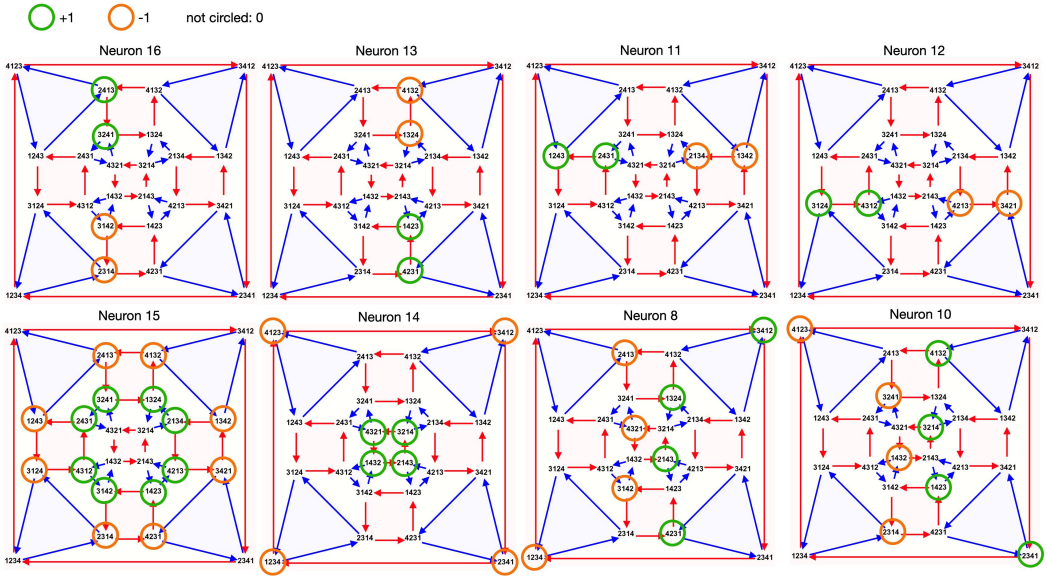


Figure A6. For active neurons in Figure 6, we highlight active group elements on Cayley graphs (green/orange/no circle means $1/-1/0$), revealing interesting structures.

Appendix E.2. The Learned Embedding Is Not a Linear Transformation of the Faithful Group Representation

Although a lot of interesting structure emerges from learned embeddings, we show that the learned embedding is not a linear transformation of the faithful group representation; at least some extent of non-linearity is at play.

S_4 has a 3×3 (truthful) matrix representation, corresponding to 3D rotations and reflection of the tetrahedron. We denote this representation as $\mathbf{E}_i^{\text{true}} \in \mathbb{R}^{3\times 3}$, and we denote the learned embedding $\mathbf{E}_i \in \mathbb{R}^9 (i = 1, \cdots, 24)$. If $\mathbf{E}_i^{\text{true}}$ and $\mathbf{E}_i$ are linearly related, there exists $\mathbf{A} \in \mathbb{R}^{3\times 3}$ and $\mathbf{V} \in \mathbb{R}^{9\times 9}$ such that

$$\mathbf{E}_i = \mathbf{V}\text{vec}(\mathbf{A}\mathbf{E}_i^{\text{true}}\mathbf{A}^{-1}), \tag{A5}$$

where vec flattens a matrix to a vector. We define a loss function:

$$L(\mathbf{V}, \mathbf{A}) = \frac{\sum_{i=1}^{24} |\mathbf{E}_i - \mathbf{V}\text{vec}(\mathbf{A}\mathbf{E}_i^{\text{true}}\mathbf{A}^{-1})|^2}{\sum_{i=1}^{24} |\mathbf{E}_i|^2}. \tag{A6}$$

If $L \approx 0$, this means a linear relation between $\mathbf{E}_i^{\text{true}}$ and $\mathbf{E}_i$; otherwise, nonlinearity is present. We optimize the above loss function with `scipy.optimize.minimize`, consistently finding the minimal value to be 0.56 (and the same for 100 random seeds), implying that learned embedding $\mathbf{E}_i$ is not a linear transformation of a truthful group representation.

Since the learned representation is quite sparse, it is likely that no single truthful representation can reach that sparsity (defined below). We conjecture that the learned representation could be combining multiple (sparse) representations in a clever way such that the combined representation is even more sparse and remains "faithful" to the extent

that prediction accuracy is perfect. The "combination" might not be that surprising, since we saw that, for modular addition (Figure 5), the learned embedding consists of three different faithful group representations.

To measure sparsity of a representation, we define a representation matrix $\mathbf{R}$ and its normalized version $\tilde{\mathbf{R}}$:

$$\mathbf{R} \equiv [\mathbf{E}_1, \cdots, \mathbf{E}_{24}] \in \mathbb{R}^{9\times 24}, \quad \tilde{\mathbf{R}} \equiv \frac{\mathbf{R}}{|\mathrm{vec}(\mathbf{R})|_1}, \tag{A7}$$

and its entropy S and effective dimension D as

$$S \equiv \mathrm{Entropy}(\mathrm{vec}(|\tilde{\mathbf{R}}|)), \quad D \equiv 2^S. \tag{A8}$$

For faithful representations corresponding to tetrahedra

$$\begin{aligned} &A: (1,0,0), (-1,1,0), (0,-1,1), (0,0,-1) \\ &B: (-1,0,0), (0,-1,0), (0,0,-1), (1,1,1) \\ &C: (1,-\frac{1}{\sqrt{3}},-\frac{1}{\sqrt{6}}), (-1,-\frac{1}{\sqrt{3}},-\frac{1}{\sqrt{6}}), (0,\frac{2}{\sqrt{3}},-\frac{1}{\sqrt{6}}), (0,0,\frac{\sqrt{6}}{2}), \end{aligned} \tag{A9}$$

their effective dimensions are $D \approx 120, 108, 153$, respectively, while the learned representation has $D \approx 80$, which is noticeably smaller.

Appendix F. In-Context Learning

In this section, we show how to modify BIMT (presented in Section 2 for MLPs) to use with transformers.

Appendix F.1. Applying BIMT to Transformers

In Section 2, we discussed how to use BIMT with fully connected neural networks. Generalization to transformers is also possible; we simply apply BIMT to "linear layers", which include not only linear layers in MLPs, but also (key, query, value) matrices, embed/unembed layers, as well as projection layers in attention blocks. In summary, we count any matrix as a "linear layer" if the matrix belongs to model parameters and does matrix–vector multiplications.

Attention layers can be seen as a special type of linear layer, involving $[\mathbf{W_Q}, \mathbf{W_K}, \mathbf{W_V}]$ as the weight matrix. We leave softmax and dot product of keys and queries unchanged, since they involve no trainable parameters. The way to calculate regularizations is the same as MLPs. However, care needs to be taken when swapping neurons. We want to swap neurons (with their corresponding weights and biases), such that the whole network remains unchanged as a function. For MLPs, we can therefore swap any two neurons in the same layer (and their corresponding weights and biases). However, for transformers, since each head operates independently, only neurons in the same head can be swapped. In addition, two heads in the same attention layer can be swapped. In summary, swapping choices are more restricted for attention layers.

Residual connections. For MLPs, swapping can be implemented independently for each layer. However, the residual connections couple all the layers in the residual stream. This means that all layers in the residual stream share the same permutations/swapping.

LayerNorm normalizes features, which contradicts the goal of sparsity. Currently, we simply remove LayerNorm layers, which works fine for our two-layer transformers. In the future, we would like to explore principled ways to handle LayerNorm in the framework of BIMT.

Appendix F.2. A Singularity Problem

In Section 3.4, we trained a transformer with BIMT for an in-context learning linear regression problem. Our setup simply has $d = 1$ and $n = 1$, which means that, given

(x_1, y_1, x) and knowing $y_1 = wx_1$, the network aims to predict $y = wx$ based on x_1, x, y_1. The ground truth formula is $y = \frac{y_1}{x_1}x$, which is singular at $x_1 = 0$. In Section 3.4, we explicitly constrained x_1 to be positive and bound it away from zero, to avoid the possible singularity. In this section, we investigate the effect of the singularity. The setup is exactly the same as in Section 3.4, with the only difference being that x_1, x are now drawn from $U[-1, 1]$ instead of $U[1, 3]$, where $U[a, b]$ stands for a uniform distribution on $[a, b]$.

After training with BIMT, the transformer is shown in Figure A7. The top right shows the predicted y versus the true y, where the prediction is good for large $|x_1|$ and bad for small $|x_1|$, which is an indication of the singularity point. Moreover, similar to Section 3.4, we look for neurons that potentially encode the information of the weight scalar in the Res2 layer. We find that neurons 9, 13, and 23 are correlated with the weight scalar, although none of them can predict the weight scalar single-handedly. In Figure A7 (bottom right), the 2D plane, spanned by neuron 9 and neuron 23, is split into four regions, with very abrupt changes at the boundaries, which are also evidence for the singularity.

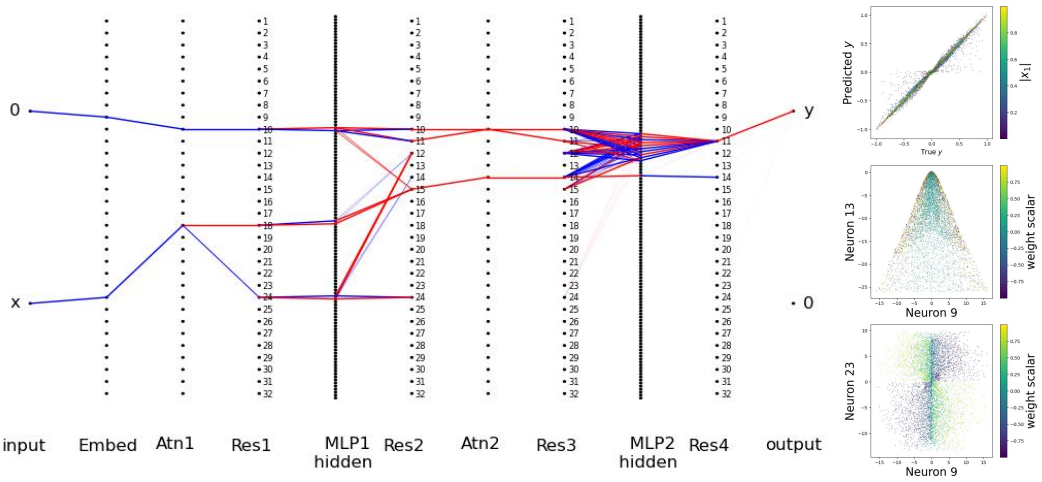


Figure A7. Applying BIMT to transformers using in-context learning linear regression. The setup is almost the same as in Figure 7, except that here, data present some singularities.

Appendix G. MNIST

Applying BIMT to tensor data. For simplicity, we usually embed neural networks in 2D Euclidean space, but it can be any geometric space. For image data, for example, to maintain locality of input images, it is more reasonable to embed neural networks into 3D Euclidean space (two along image axes, one along depth). Now, neurons in the same layer are arranged in a 2D grid instead of a 1D grid. This only affects distances between neurons, with everything else unchanged. In fact, to change MLP embedding from 2D to 3D, the only thing we need to change is to redefine the coordinates of neurons. Similarly, networks can be embedded in higher-dimensional Euclidean space or even Riemannian manifolds by properly redefining coordinates and computing distances based on the manifold metric.

Positive vs. negative weights. It was observed in Figure 8 that, at the end of training, most weights in layer 3 (the last layer before outputs) are positive (blue), while most weights in layer 2 are negative (red). To verify that this is not just a visual artifact, we plot the rank distribution of positive and negative weights in Figure A8. In layer 1, there are more positive weights with large magnitude, while negative weights seem to have a heavier tail. In layer 2 and layer 3, there are clearly more positive and negative weights, respectively. We are still not sure why such symmetry breaking happens because, upon initialization, the number of positive and negative weights is roughly balanced. In Section 3.5, we called this phenomenon "pattern mismatching". It would be interesting to investigate if pattern mismatching is prevalent in neural networks, or if it is specific to some combination of specific architectures, datasets, and/or training techniques.

Learned features. To understand what the neural network has learned, we visualize the features (weight matrices) in layer 1. For each feature, we compute its score as the sum of absolute weights. We rank features from high to low scores, finding there are 38 features with large scores, as shown in Figure A9. The features look like intermediate to high-level feature maps of convolutional filters in trained convolutional neural networks, since they are more than just edge detectors (low-level convolutional filters), containing some extent of global correlations.

MLPs with other depths. In the main text, we showed the results for a three-layer MLP. We also show the results (how the connectivity graphs evolve in training) for a two-layer MLP and a four-layer MLP in Figures A10 and A11, respectively.

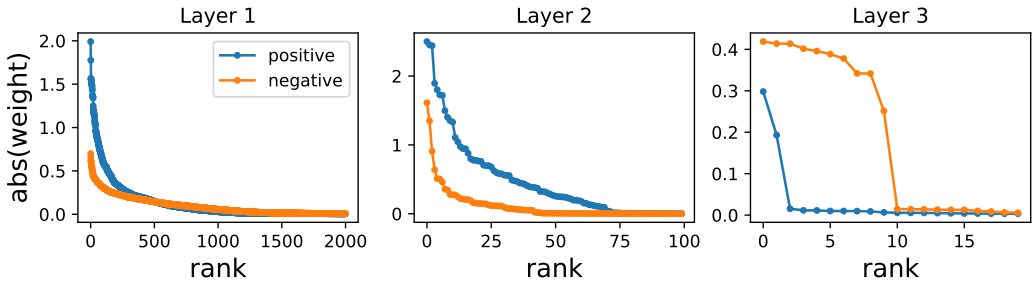


Figure A8. The magnitudes of positive and negative weights in MLP layers after training. Positive weights dominate in layer 2, while negative weights dominate in layer 3.

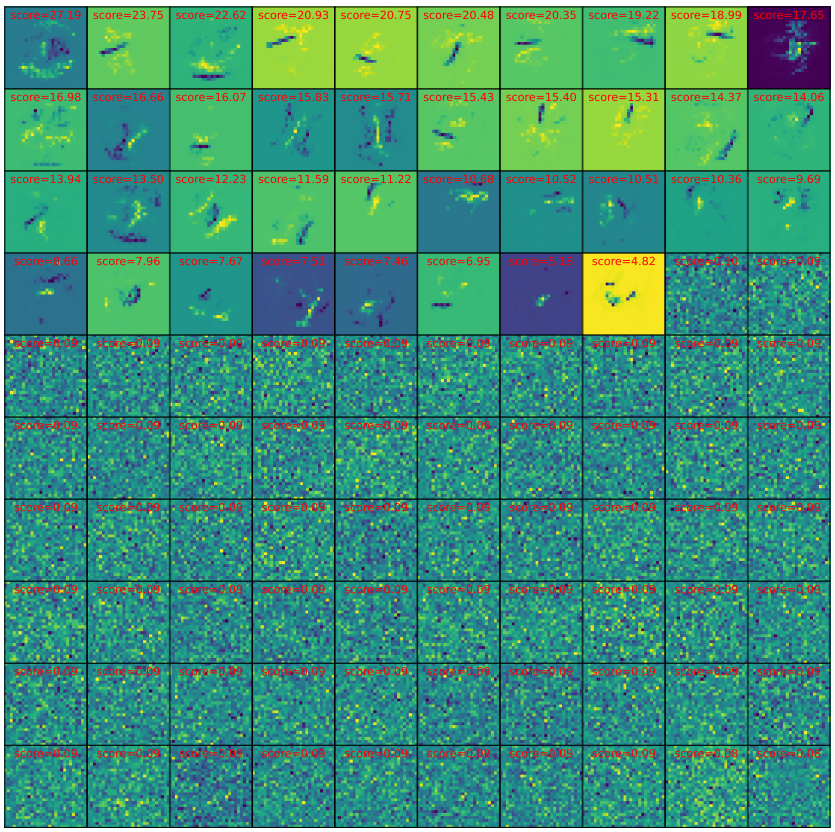


Figure A9. Visualizing MNIST features (layer 1 of Figure 8).

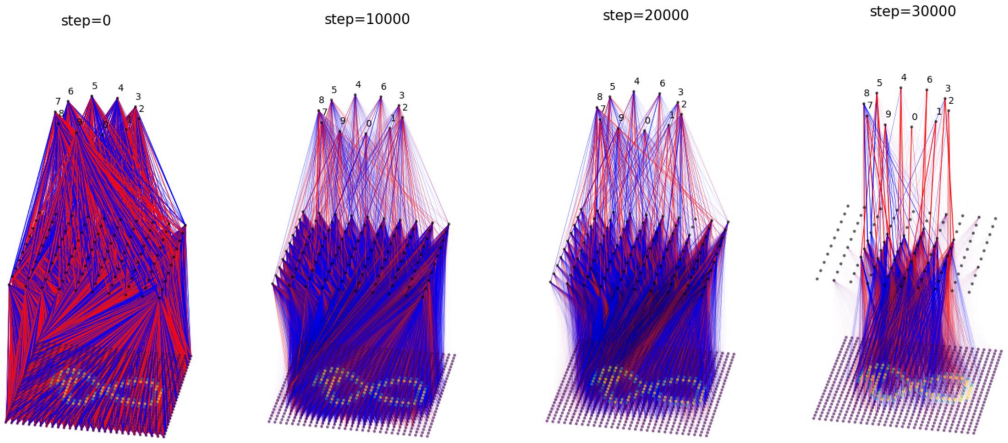


Figure A10. Evolution of a two-layer MLP trained with BIMT.

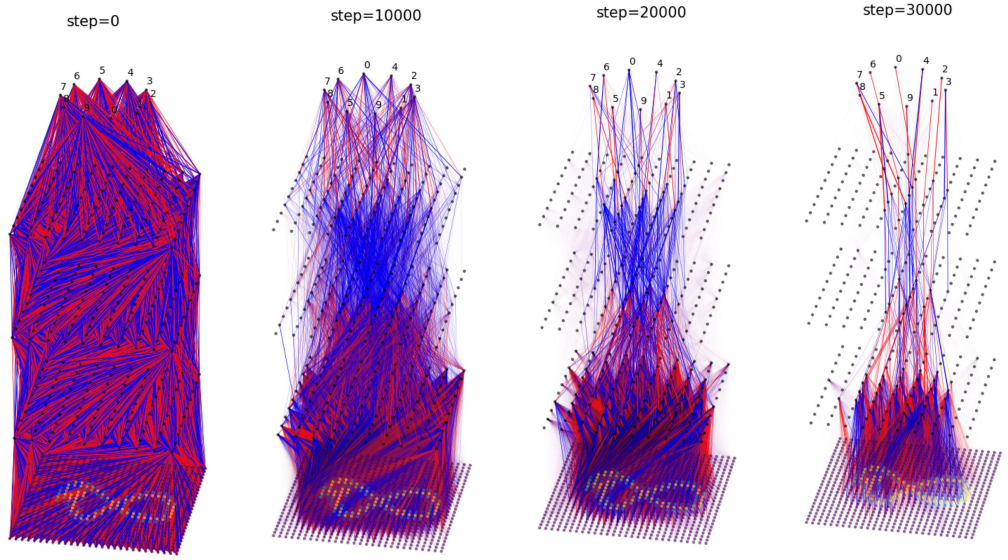


Figure A11. Evolution of a four-layer MLP trained with BIMT.

References

1. Olah, C.; Cammarata, N.; Schubert, L.; Goh, G.; Petrov, M.; Carter, S. Zoom In: An Introduction to Circuits. *Distill* **2020**. Available online: https://distill.pub/2020/circuits/zoom-in (accessed on 1 November 2023).
2. Olsson, C.; Elhage, N.; Nanda, N.; Joseph, N.; DasSarma, N.; Henighan, T.; Mann, B.; Askell, A.; Bai, Y.; Chen, A.; et al. In-Context Learning and Induction Heads. *Transform. Circuits Thread* **2022**. Available online: https://transformer-circuits.pub/2022/in-context-learning-and-induction-heads/index.html (accessed on 1 November 2023).
3. Michaud, E.J.; Liu, Z.; Girit, U.; Tegmark, M. The Quantization Model of Neural Scaling. *arXiv* **2023**, arXiv:2303.13506.
4. Elhage, N.; Nanda, N.; Olsson, C.; Henighan, T.; Joseph, N.; Mann, B.; Askell, A.; Bai, Y.; Chen, A.; Conerly, T.; et al. A Mathematical Framework for Transformer Circuits. *Transform. Circuits Thread* **2021**. Available online: https://transformer-circuits.pub/2021/framework/index.html (accessed on 1 November 2023).
5. Wang, K.R.; Variengien, A.; Conmy, A.; Shlegeris, B.; Steinhardt, J. Interpretability in the Wild: A Circuit for Indirect Object Identification in GPT-2 Small. In Proceedings of the The Eleventh International Conference on Learning Representations, Kigali, Rwanda, 1–5 May 2023.
6. Bear, M.; Connors, B.; Paradiso, M.A. *Neuroscience: Exploring the Brain, Enhanced Edition: Exploring the Brain*; Jones & Bartlett Learning: Burlington, MA, USA, 2020.
7. Meunier, D.; Lambiotte, R.; Bullmore, E.T. Modular and hierarchically modular organization of brain networks. *Front. Neurosci.* **2010**, *4*, 200. [CrossRef] [PubMed]
8. Friston, K. Hierarchical models in the brain. *PLoS Comput. Biol.* **2008**, *4*, e1000211. [CrossRef] [PubMed]
9. Clune, J.; Mouret, J.B.; Lipson, H. The evolutionary origins of modularity. *Proc. R. Soc. Biol. Sci.* **2013**, *280*, 20122863. [CrossRef] [PubMed]

10. Mengistu, H.; Huizinga, J.; Mouret, J.B.; Clune, J. The evolutionary origins of hierarchy. *PLoS Comput. Biol.* **2016**, *12*, e1004829. [CrossRef] [PubMed]
11. Huizinga, J.; Clune, J.; Mouret, J.B. Evolving neural networks that are both modular and regular: Hyperneat plus the connection cost technique. In Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation, Nanchang, China, 18–20 October 2014; pp. 697–704.
12. Ellefsen, K.O.; Mouret, J.B.; Clune, J. Neural modularity helps organisms evolve to learn new skills without forgetting old skills. *PLoS Comput. Biol.* **2015**, *11*, e1004128. [CrossRef] [PubMed]
13. Chughtai, B.; Chan, L.; Nanda, N. A Toy Model of Universality: Reverse Engineering How Networks Learn Group Operations. *arXiv* **2023**, arXiv:2302.03025.
14. Han, S.; Pool, J.; Tran, J.; Dally, W. Learning both weights and connections for efficient neural network. *Adv. Neural Inf. Process. Syst.* **2015**, *28*, 1135–1143.
15. Anwar, S.; Hwang, K.; Sung, W. Structured pruning of deep convolutional neural networks. *Acm J. Emerg. Technol. Comput. Syst. (JETC)* **2017**, *13*, 1–18. [CrossRef]
16. Pfeiffer, J.; Ruder, S.; Vulić, I.; Ponti, E.M. Modular deep learning. *arXiv* **2023**, arXiv:2302.11529.
17. Kirsch, L.; Kunze, J.; Barber, D. Modular networks: Learning to decompose neural computation. *Adv. Neural Inf. Process. Syst.* **2018**, *31*, 2414–2423.
18. Ng, A.Y. Feature selection, L 1 vs. L 2 regularization, and rotational invariance. In Proceedings of the Twenty-First International Conference on Machine Learning, Banff, AB, Canada, 4–8 July 2004; p. 78.
19. Kingma, D.P.; Ba, J. Adam: A method for stochastic optimization. *arXiv* **2014**, arXiv:1412.6980.
20. Cranmer, M.; Sanchez Gonzalez, A.; Battaglia, P.; Xu, R.; Cranmer, K.; Spergel, D.; Ho, S. Discovering symbolic models from deep learning with inductive biases. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 17429–17442.
21. Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; et al. Scikit-learn: Machine Learning in Python. *J. Mach. Learn. Res.* **2011**, *12*, 2825–2830.
22. Liu, Z.; Kitouni, O.; Nolte, N.; Michaud, E.J.; Tegmark, M.; Williams, M. Towards Understanding Grokking: An Effective Theory of Representation Learning. In *Advances in Neural Information Processing Systems*; Oh, A.H., Agarwal, A., Belgrave, D., Cho, K., Eds.; MIT Press: Cambridge, MA, USA, 2022.
23. Nanda, N.; Chan, L.; Liberum, T.; Smith, J.; Steinhardt, J. Progress measures for grokking via mechanistic interpretability. *arXiv* **2023**, arXiv:2301.05217.
24. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I. Attention is all you need. *Adv. Neural Inf. Process. Syst.* **2017**, *30*, 6000–6010.
25. Akyürek, E.; Schuurmans, D.; Andreas, J.; Ma, T.; Zhou, D. What learning algorithm is in-context learning? Investigations with linear models. *arXiv* **2022**, arXiv:2211.15661.
26. Meng, K.; Bau, D.; Andonian, A.; Belinkov, Y. Locating and editing factual knowledge in gpt. *arXiv* **2022**, arXiv:2202.05262.
27. Elhage, N.; Hume, T.; Olsson, C.; Schiefer, N.; Henighan, T.; Kravec, S.; Hatfield-Dodds, Z.; Lasenby, R.; Drain, D.; Chen, C.; et al. Toy Models of Superposition. *arXiv* **2022**, arXiv:2209.10652.
28. Power, A.; Burda, Y.; Edwards, H.; Babuschkin, I.; Misra, V. Grokking: Generalization beyond overfitting on small algorithmic datasets. *arXiv* **2022**, arXiv:2201.02177.
29. Zhong, Z.; Liu, Z.; Tegmark, M.; Andreas, J. The clock and the pizza: Two stories in mechanistic explanation of neural networks. *arXiv* **2023**, arXiv:2306.17844.
30. Filan, D.; Casper, S.; Hod, S.; Wild, C.; Critch, A.; Russell, S. Clusterability in neural networks. *arXiv* **2021**, arXiv:2103.03386.
31. Hod, S.; Casper, S.; Filan, D.; Wild, C.; Critch, A.; Russell, S. Detecting modularity in deep neural networks. *arXiv* **2021**, arXiv:2110.08058.
32. Csordás, R.; van Steenkiste, S.; Schmidhuber, J. Are Neural Nets Modular? Inspecting Functional Modularity through Differentiable Weight Masks. In Proceedings of the International Conference on Learning Representations, Virtual, 3–7 May 2021.
33. Azam, F. Biologically Inspired Modular Neural Networks. Ph.D. Thesis, Virginia Polytechnic Institute and State University, Blacksburg, VA, USA, 2000.
34. Blalock, D.; Gonzalez Ortiz, J.J.; Frankle, J.; Guttag, J. What is the state of neural network pruning? *Proc. Mach. Learn. Syst.* **2020**, *2*, 129–146.
35. Frankle, J.; Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv* **2018**, arXiv:1803.03635.
36. Richards, B.A.; Lillicrap, T.P.; Beaudoin, P.; Bengio, Y.; Bogacz, R.; Christensen, A.; Clopath, C.; Costa, R.P.; de Berker, A.; Ganguli, S.; et al. A deep learning framework for neuroscience. *Nat. Neurosci.* **2019**, *22*, 1761–1770. [CrossRef] [PubMed]
37. Hassabis, D.; Kumaran, D.; Summerfield, C.; Botvinick, M. Neuroscience-inspired artificial intelligence. *Neuron* **2017**, *95*, 245–258. [CrossRef]
38. Lillicrap, T.P.; Santoro, A.; Marris, L.; Akerman, C.J.; Hinton, G. Backpropagation and the brain. *Nat. Rev. Neurosci.* **2020**, *21*, 335–346. [CrossRef]
39. Van Hemmen, J.L.; Sejnowski, T.J. *23 Problems in Systems Neuroscience*; Oxford University Press: New York, NY, USA, 2005.
40. Rieke, F.; Warland, D.; Van Steveninck, R.D.R.; Bialek, W. *Spikes: Exploring the Neural Code*; MIT Press: Cambridge, MA, USA, 1999.
41. Pregowska, A. Signal fluctuations and the Information Transmission Rates in binary communication channels. *Entropy* **2021**, *23*, 92. [CrossRef]

42. Salinas, E.; Sejnowski, T.J. Correlated neuronal activity and the flow of neural information. *Nat. Rev. Neurosci.* **2001**, *2*, 539–550. [CrossRef]
43. Knoblauch, A.; Palm, G. What is signal and what is noise in the brain? *Biosystems* **2005**, *79*, 83–90. [CrossRef] [PubMed]
44. Mainen, Z.F.; Sejnowski, T.J. Reliability of spike timing in neocortical neurons. *Science* **1995**, *268*, 1503–1506. [CrossRef] [PubMed]