THE FORM OF THE WEIGHTS IN DEEP NETWORKS TRAINED ON MODULAR ADDITION

Anonymous authors
Paper under double-blind review

ABSTRACT

Our work presents the form of the weights between neurons (and thus between layers) that deep neural networks (DNNs) learn on the popular task of modular addition. This task has been studied by many prior works, but only the activation geometry, which are the manifolds corresponding to the learned neural representations, are understood in the case of depth. The weights that are learned are only understood in 1-hidden layer networks and thus, this work serves to extend understanding of the weights to multi-layer networks. To our knowledge, this is the first work to describe and validate the form that learned weights in deep networks take between layers across random seeds on a group multiplication task.

1 INTRODUCTION

Inspired by how exactly solved models contributed to the evolution of modern physics (see Baxter (1985)), deep learning (DL) scientists have been utilizing mathematical objects known as *groups* to construct environments (McCracken, 2021) and create datasets (Power et al., 2022) with the hope of gaining better controlled insight into what deep neural networks (DNNs) learn. The hope is that by uncovering *how* and *what* DNNs learn on group tasks, the community can learn principles that may guide intuitions for harder datasets, *i.e.* natural data where there are no mathematical "truths" that can be derived from known rules about how the dataset was created.

The dataset that started this movement was modular addition, when the result of delayed generalization, called grokking was disseminated by Power et al. (2022). This resulted in a stream of interpretability papers, originally centered around explaining the grokking phenomenon, but later evolving into their own subfield, now treating group multiplication tasks like modular addition as testing grounds for scientific hypotheses. So far, a lot about *what* is learned by networks learning modular addition is known, and to briefly summarize it here, we'll split the prior work into work on activations, and work on weights.

Prior work on activations. The activations of individual neurons are sinusoidal Nanda et al. (2023); Chughtai et al. (2023a); Gromov (2023), the activation manifolds of learned neural representations (a group of neurons collaborating together simultaneously within the network to compute the answer to the same subproblem) are 2-tori embedded in $\mathbb{R}^4$, or linear projections of these tori (Moisescu-Pareja et al., 2026) and the activation-geometry was found to globally implement a universal abstract template for an algorithm, being the Chinese Remainder Theorem (McCracken et al., 2025).

Prior work on weights. As far weights are concerned, only 1-hidden layer models are understood and they are understood well. Nanda et al. (2023) found that the learnable embedding matrix contained sinusoidal entries that could be read off the embedding vectors and avoided the permutation invariance of the weights in transformers by replacing the neurons and their activations (replacing input and output weights together at once) with fitted second order sinusoidal functions $\cos(a+b)$ and $\sin(a-b)$. This work inspired a field seeking to explain why gradient descent methods learn sinusoids in 1-layer networks, starting with Gromov (2023), who gave analytical equations for a toy model of the weights of 1-hidden layer multilayer perceptrons (MLPs). Later, work proving sinusoids would be learned came, with Morwani et al. (2024) and Mohamadi et al. (2023) using the assumption of margin maximization to show that neurons must be sinusoids to maximize the margin. Then heuristic descriptions and proofs of stochastic gradient descent (SGD) converging to sinusoidal weights in 1-hidden layer networks began with Ding et al. (2024); Tian (2025; 2026) and recently the alternating gradient flow from small weight initialization of Kunin et al. (2026).

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Part of the difficulty stems from neurons in hidden layers having permutation symmetry: in a layer with ℓ neurons there are $\ell!$ ways to permute neurons with each permutation resulting in the exact same network, with neurons simply located in different indices of the layer across different permutations (and these reorderings can be done in all layers). When considering a 1-hidden layer network trained on $a + b \equiv c \pmod{n}$, this permutation invariance within layers doesn't yet add difficulty because there is only 1 layer: the neurons can be permuted, but since they immediately connect to output logits, and the output logits have an ordering from 0 to $n - 1$, it is easy to plot the output weights from a neuron and see they are sinusoidal, as found by prior work (Gromov, 2023; Morwani et al., 2024; Yip et al., 2024; Wu et al., 2025). Moisescu-Pareja et al. (2026) indirectly inspected the weights via principal component analyses and diffusion maps, finding that almost all of the computation about the answer $c = (a + b) \bmod n$ was finished by the preactivations of the second layer, and that deeper layers just seemed to rotate the neural representation manifolds in the latent weight space of the network. With all of this said, the equations describing *what* the weights are in deeper networks have remained unknown, even with the forms of the activations of neurons and the forms of the manifolds corresponding to neural representation activations being known (Moisescu-Pareja et al., 2026).

This is where this paper builds on prior work. We uncover the equations that model weights between hidden layers in multi-layer networks. Achieving this creates, to our knowledge, the first description of what is learned between layers in DNNs trained on a task requiring non-linear activations to generalize to the test data (the modular addition dataset is not linearly separable). This work will be of significant interest to theoreticians attempting to create a scientific theory of deep learning. After this, future work required to yield an exact model of modular addition would be a descriptive proof that SGD converges to the form of the weights we discover below, as it must explain the gradient dynamics. Our commentary on this can be found in the Discussion (Section 5).

A summary of our contributions to literature interpreting modular addition follows:

1. We discover the expressions that describe weights in trained networks with different activation functions (ReLU and GELU), depths, and architectures (both transformers and multi-layer perceptrons) over a diverse range of training settings (learning rates and weight decays). We believe we are the first work to provide such a description of learned weights with non-linearities and depth. Please note, prior work has achieved this in the case of deep **linear** networks (Saxe et al., 2013) and did this for the training dynamics of these deep linear networks (no non-linear activations). While our work only gives the form of the weights found after training, it is a critical step for future work investigating the training dynamics of deep nonlinear networks learning modular addition.

2. The key to this result is beating the permutation invariance between neurons within layers. We do this via uncovering a novel insight: *the 2d discrete Fourier transform (DFT) of neuron activations contains phase information that can be used to canonically order weights between hidden layers. This information allows for neurons to be ordered in a human-interpretable way.* In essence, we use 2d DFTs of neuron activations to acquire a human-interpretable coordinate system, yielding the ability to fully describe the inter-layer weights in deep networks with sinusoids, which require an ordering (since their domain is totally ordered from 0 to 2pi).

    - With the 2d DFT, we define a *relative phase* that gets around the problems caused by permutation symmetry within layers by giving a canonical coordinate system based on the activations of neurons. Thus, our work is the natural sequel to both McCracken et al. (2025) and Moisescu-Pareja et al. (2026), who together fully described the activations of neurons, the activation-geometry of neural representations, and even the activation-geometry of a global algorithm networks implement across neural representations to solve modular addition in 1-4 hidden layer networks.
    - Resultantly, our work supplements this story and serves as the logical next step: we present the form of the equations for the weights between hidden layers, and validate them exhaustively over different training conditions and architectures. As we have a full understanding of how to order neurons, we can even study the weights of an entire neural representation simultaneously.

2 RELATED WORK

Group multiplication datasets have become a standard benchmark for researchers studying various phenomena in deep learning (Nanda et al., 2023; Chughtai et al., 2023b; Zhong et al., 2023; Gromov, 2023; Stander et al., 2024; McCracken et al., 2025; Moisescu-Pareja et al., 2026; Kvinge et al., 2026; McCracken et al., 2026; Wei et al.; Hilgert et al.). On just modular addition alone, there is an ample array of work trying to explain precisely how networks learn the task. From *how* networks learn, being the training dynamics during stochastic gradient updates (Power et al., 2022; Morwani et al., 2024; Gromov, 2023; He et al., 2024; Liu et al., 2022; Mohamadi et al., 2023; Mallinar et al., 2024; Ding et al., 2024) and also *what* networks learn, from individual neurons, to neural representations, and even the global algorithm being learned, there is an abundance of literature continually progressing the understanding of networks solving the task (Nanda et al., 2023; Chughtai et al., 2023b; Zhong et al., 2023; Stander et al., 2024; Yip et al., 2024; Doshi et al., 2023; Wu et al., 2025; Tao et al., 2025; Moisescu-Pareja et al., 2026).

Naturally, our work has been inspired and influenced by all of the aforementioned work, but most related to it is work studying the weights of networks learning modular addition. So far, the field has multiple papers studying the 1-hidden layer case, but the multi-layer case remained unknown until this work. Starting with Nanda et al. (2023), the weights of the embedding matrix were uncovered to be sinusoidal, but the weights after the embeddings weren't discussed until Gromov (2023); Morwani et al. (2024); Yip et al. (2024) and Wu et al. (2025). These papers collectively uncovered that the weights of 1-hidden layer networks are sinusoidal, but could only make inferences about what weights could be in deeper layers (see *e.g.* Zhong et al. (2023)'s gradient symmetricity and distance irrelevance metrics). Additionally of note is Doshi et al. (2024), which provided constructions for the weights of 1 hidden layer networks.

This is where our work builds on preceding literature. Via utilization of the 2-dimensional discrete Fourier transform (2d DFT), we find a way to normalize (or equivalently order neurons in layers in a human-interpretable way such that the permutation invariance of neurons isn't an issue). Thus, we present the form of the weights in DNNs on modular addition.

3 BACKGROUND

Two-dimensional Fourier transform. For a function $g(a, b)$ defined on $\mathbb{Z}_n^2$, the two-dimensional discrete Fourier transform decomposes g into Fourier modes indexed by pairs $(k_1, k_2) \in \mathbb{Z}_n^2$. Each pair (k_1, k_2) specifies a two-dimensional frequency, or Fourier mode.

The Fourier coefficient associated with mode (k_1, k_2) is

$$\hat{g}(k_1, k_2) = \frac{1}{n^2} \sum_{a,b \in \mathbb{Z}_n} g(a, b) e^{-\frac{2\pi i}{n}(k_1 a + k_2 b)}.$$

Each Fourier coefficient can be written as

$$\hat{g}(k_1, k_2) = A_{(k_1,k_2)} e^{i\phi_{(k_1,k_2)}},$$

where $A_{(k_1,k_2)} \geq 0$ is the amplitude of the Fourier mode (k_1, k_2), and $\phi_{(k_1,k_2)} \in (-\pi, \pi]$ is its phase. The amplitude measures the strength of the mode in g, while phase indicates the shift.

We treat conjugate Fourier modes as belonging to the same Fourier component. Neurons whose activations are dominated by $(0, k)$, $(k, 0)$, or (k, k) for the same k are assigned the same label as a dominant Fourier component k_{dom}. Let all neurons with the same k_{dom} be a cluster in layer ℓ called $S_{k,\ell}$; a neural representation is all neurons in both $S_{k,\ell}$ and $S_{k,\ell+1}$ with the same k_{dom}.

We study DNNs trained on modular addition over the finite cyclic group ($\mathbb{Z}_n$). Given an input pair $(a, b) \in \mathbb{Z}_n^2$, the model predicts $c = a + b \pmod{n}$. Unless otherwise specified we use $n = 59$, giving $n^2 = 3481$ possible input pairs and $n = 59$ output classes. We train MLPs and transformers with varying depths and nonlinearities, including ReLU and GELU, using a 90%/10% train/test split. All models use 512 neurons per layer and a shared embedding matrix for input tokens $a, b \in \mathbb{Z}_n$, and achieve 100% train and test accuracy. We prune the trained networks using a binary search-like process to remove "dead" neurons with tiny activations (see Appendix A.2), constraining the relative increase in cross-entropy loss to at most 5%.

We evaluate our claims using adjusted R^2 and task accuracy after replacing the weights with our model's best fit. Adjusted R^2 is computed separately for each fixed target (or source) neuron and then averaged across neurons, while replacement accuracy measures whether the fitted weights preserve the network computation (maintain full dataset accuracy at 100%).

We define a source neurons to be neurons in a layer preceding a target neuron. Alternatively worded, a target neuron has many source neurons with weights from the previous layer coming into the target neuron. This gives two different perspectives: 1. *Forward view:* What is the expression for the weights leaving a neuron and going forward to many neurons in the next layer? 2. *Backward view:* What is the expression for the weights coming into a target neuron from the previous layer?

For a hidden-layer connection from layer ℓ to layer $\ell+1$, let $W_{ij}^{(\ell)}$ denote the weight connecting source neuron i in layer ℓ to target neuron j in layer $\ell+1$. For each source neuron, we denote its post-activation over all pairs (a,b) in the dataset as $h_i^{(\ell)}(a,b)$, with $(a,b) \in \mathbb{Z}_n^2$, and we denote a target neurons preactivation by $z_j^{(\ell+1)}(a,b)$. We characterize both $h_i^{(\ell)}$ and $z_j^{(\ell+1)}$ in the Fourier basis of $\mathbb{Z}_n^2$. For source neuron i, we denote the amplitude and phase at Fourier component k by $A_i^{(\ell)}(k)$ and $\phi_i^{(\ell)}(k)$. For target neuron j, we similarly denote the amplitude and phase of its pre-activation by $B_j^{(\ell+1)}(k)$ and $\phi_j^{(\ell+1)}(k)$.

4 Results

Our major result is that we have the form of the weights for the neural representations (define above) that are learned in DNNs trained on modular addition. We present two main results. The first is the *simple form*, which fits most DNNs well, but fails to fit weights in deeper GELU layers. Thus, we later present the *general form*, which is a generalization of the simple form.

Simple form. In the backward view, we inspect the set $S_{k,\ell}$ of all source neurons in layer ℓ that have the same dominant Fourier component k as the target neuron in layer $\ell+1$. We model the weight W_{ij} from source neuron i to target neuron j as

$$W_{ij} \approx \beta_j + C_j \widetilde{A}_i \cos(\phi_i - \phi_j),$$

where ϕ_i and ϕ_j are respectively the phases of the source and target neurons at mode k, as measured from their two-dimensional DFT coefficients. Throughout, the phase difference $\phi_i - \phi_j$ is understood circularly: it is taken modulo 2π and represented in the interval $(-\pi, \pi]$. Note both the y-intercept β_j and the scale C_j are fitted parameters (whereas the phases come from the 2-d DFT), and the normalized amplitude $\widetilde{A}_i = \frac{A_i}{\sqrt{\frac{1}{|S_{k,\ell}|}\sum_{i' \in S_{k,\ell}} A_{i'}^2}}$, is the source amplitude normalized by its RMS within the cluster $S_{k,\ell}$. For the last hidden layer-to-logit weights, the target phase is determined analytically from the Fourier representation of each output class. For class c, the modular-addition indicator $g_c(a,b) = \mathbf{1}[a+b \bmod n = c]$ has phase $\phi_c^{\text{logit}}(k) = -\frac{2\pi kc}{n}$ at Fourier component (k,k). Its amplitude is constant across classes and is therefore taken to be one. Similarly, in the forward view, we examine the outgoing weights from a single source neuron i to a bunch of target neurons j in the next layer. Within the corresponding target set $S_{k,\ell+1}$, we analogously fit $W_{ij} \approx \alpha_i + D_i \widetilde{B}_j \cos(\phi_j - \phi_i)$, where $\widetilde{B}_j = \frac{B_j}{\sqrt{\frac{1}{|S_{k,\ell+1}|}\sum_{j' \in S_{k,\ell+1}} B_{j'}^2}}$, and the y-intercept α_i and scale D_i are fit independently for each source neuron.

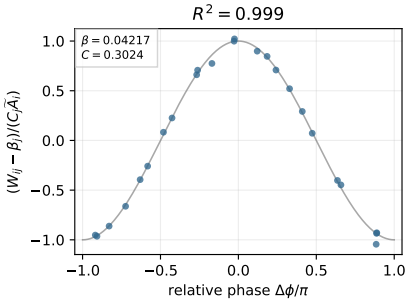

Figure 1: One second-layer target neuron over its source weights in a 2-layer Transformer; simple form: $R^2 = 0.999$.

Normalizing phases. A central difficulty in studying the learned weights with more than 1-hidden layer is that hidden neurons between layers have no canonical ordering: permuting the neurons within a layer together with their corresponding rows and columns of weight matrices, leaves the DNN functionally unchanged.

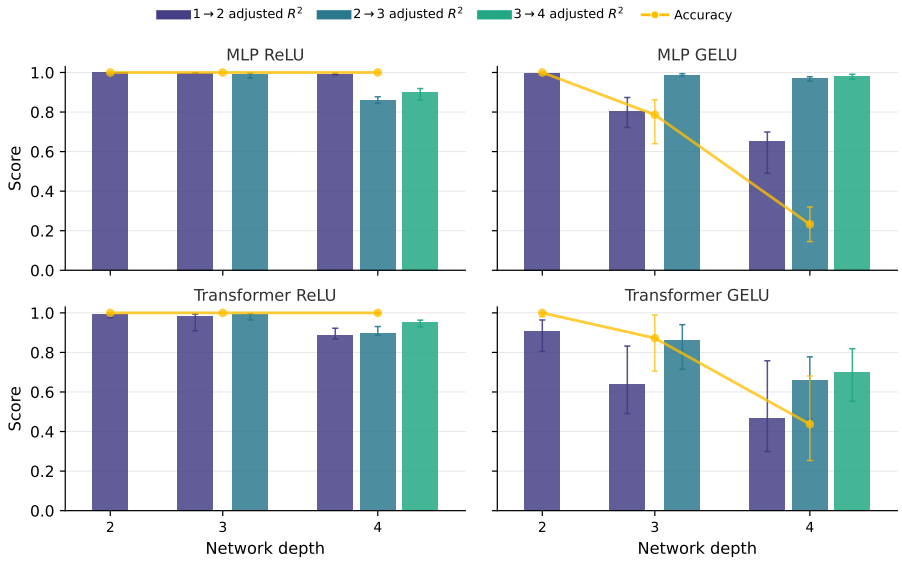

Figure 2: Simple-form fit quality and accuracy after fitted-weight replacement across network depth. Bars show the median adjusted R^2 of the simple-form fit for each interlayer weights, with error bars denoting the interquartile range across seeds. Bar colors indicate the corresponding connection ($1 \to 2$, $2 \to 3$, or $3 \to 4$). The yellow dot line shows the median accuracy after replacing the fitted hidden-layer weights with their simple-form approximations, with error bars denoting the interquartile range across seeds. Deeper networks show greater variation in simple-form fit quality across layers and, in several settings, lower replacement accuracy.

Our discovery that solves this issue comes from the two-dimensional discrete Fourier transform (DFT). The phase information in the postactivation of a source neuron of Fourier mode k in the $2D$ DFT can be combined with the phase information in the preactivations of a target neuron to define the *relative phase*. With $\phi_i(k)$ being the source phase and $\phi_j(k)$ being the target phase, the relative phase $\phi_r(k)$ is the circularly wrapped value of $\phi_i(k) - \phi_j(k)$, taken modulo 2π and represented in the interval $(-\pi, \pi]$. Intuitively, this measures how much the source neuron aligns with the computation the target neuron is responsible for, and a relative phase of 0 means the neuron is maximally aligned. Indeed, we can normalize this wrapped relative phase to be in the range $(-1, 1]$ by dividing by π. We now present Figure 1, showing normalized weights vs relative normalized phase give sinusoidal weights for a second layer neuron. Figure 2 serves to show that the simple form works well except for in GELU DNNs. Note both the R^2 in GELU DNNs and accuracy drops when replacing weights with the simple-form fit's weights.

The reason for the decrease of task performance seen in Figure 2 is that deeper networks and especially GELU networks, like to linearly combine different frequencies of cosines into weights. We quantify this relationship in Appendix A.8 by measuring the fraction of each neuron's Fourier energy outside its dominant component. Representation mixture is negatively correlated with simple-form fit quality across all architecture–activation settings (Pearson correlations: -0.51 to -0.97), supporting the interpretation that increased frequency mixing drives the degradation of the simple form.

General form. The general form models weights using multiple Fourier components, without grouping neurons by a dominant component; in other words, we fit every neuron with a sum of its first few Fourier components. In the backward view, for target neuron j, we fit incoming weights:

$$W_{ij} \approx \beta_j + \sum_{k \in \mathcal{Q}_j} C_{jk}\, \widetilde{A}_i(k) \cos\big(\phi_i(k) - \delta_{jk}\big),$$

where $k = (k_a, k_b)$ indexes a two-dimensional Fourier component and $\mathcal{Q}_j$ is the set of highest scoring Fourier components from the 2d DFT; $\widetilde{A}_i(k)$ and $\phi_i(k)$ are the normalized amplitude and phase of source neuron i at component k; for each term in the sum (Fourier component), C_{jk} is a fitted scale parameter and δ_{jk} is a fitted target-specific phase offset, and the term β_j is a fitted

y-intercept. For clarity note: δ_{jk} and C_{jk} are fitted over all the incoming weights (from source neurons) for each $k \in \mathcal{Q}_j$. To relate this parameterization to the relative-phase structure of the simple form, we may write $\delta_{jk} = \phi_j(k) + \theta_{jk}$, so that the cosine depends equivalently on $\Delta\phi_{ij}(k) - \theta_{jk}$, where $\Delta\phi_{ij}(k) = \phi_i(k) - \phi_j(k)$. Because θ_{jk} is fitted separately for each target and Fourier component, this is only a reparameterization of the fitted phase offset and does not change the expressive form of the fit. The general form extends the single-frequency cosine relation of the simple form by allowing contributions from multiple Fourier components.

As the Fourier transform proves a sum of sinusoids can arbitrarily approximate a function, we believe it's essential to quantitatively demonstrate the general form is being learned. We demonstrate this as follows. First, we show that fitting the weights when they're ordered by their relative phase requires a small number of sinusoids to be summed in order to get a good fit. Then, we compare this to the number of sinusoids required to fit a random permutation of the weights, in order to maintain a network with 100% accuracy. We find that substantially more terms are required to fit random permutations of the weights. Furthermore, we can formalize this, and do so by defining the *fit complexity*, which measures how many fitted parameters (terms in the sum of cosines) are being used to fit the weights compared to the number of weights being fit. This measures overfitting.

Fit complexity. The raw number of selected Fourier components does not by itself determine fit complexity, because different target neurons may be fit over different numbers of source weights and because selected components can contribute linearly dependent directions. For each target neuron, we define $\rho = \frac{r-1}{N-1}$, where r is the dimensionality required by the fitted model (the number of linearly independent fitted parameter directions) and N is the number of source weights being fit. Thus, ρ measures how many independent degrees of freedom are needed to fit the weights relative to the number of weights being fitted. A value near 1.0 means that nearly one independent degree of freedom is needed per weight.

Control experiment. For each target neuron j, let $\mathcal{Q}_j$ denote the set Fourier modes. Our goal is to determine how many terms from Q_j are required to fit the source weights such that the DNN maintains 100% accuracy on the dataset. For the control, we randomly permute the source neurons in the layer before the target neuron j. For a permutation π_j, we fit

$$W_{\pi_j(i),j} \approx \beta_j + \sum_{k \in \mathcal{Q}_j} C_{jk}\, \widetilde{A}_i(k) \cos\big(\Delta\phi_{ijk} - \theta_{jk}\big).$$

After fitting, the fitted weights are inverse-permuted back to their original weight positions before evaluating the network. We use 10 independent permutations per target neuron and seed. Finally, we report the results in Figure 3 and see that the permuted weights require substantially more terms to be fit in order to maintain 100% accuracy on the dataset. In particular, the mean fit complexity of permuted weights, $\bar{\rho}^*$, is close to 1, implying that almost 1 independent degree of freedom is required per weight. Therefore, we have demonstrated that the general form is a good low-dimensional fit for weights in both MLPs and transformers trained on modular addition.

Now we report the general form's fit quality over depths, architectures and different activation functions in Table 1. Finally, we generalize this across training hyperparameters, and show in Figure 4, that over the majority of the volume where training parameters are conducive to learning, our claimed general form is learned, and replacing all weights by it, results in a network that maintains 100% task accuracy (denoted via a green checkmark).

The neural representations. Finally, we introduce the tool required to let us study all weights of neural representations at once simultaneously. A neural representation between two layers is a collection of source and target neurons that collaborate together. Prior work inspected the activations of the neural representations; McCracken et al. (2025) indirectly found DNNs learn $\mathcal{O}(\log(n))$ different neural representations, each composed of all neurons with the same dominant Fourier frequency k in their preactivations. Moisescu-Pareja et al. (2026) found directly that the activation manifold of all neurons with the same frequency in their preactivations was either a universal manifold, being a torus, or a linear projection of the torus. We now present an understanding of the weights of the neurons composing these neural representations in DNNs.

Consider the backwards view: we have every target neuron in layer $\ell + 1$ with the same dominant frequency k, call this set $S_{k,\ell+1}$, and all the source neurons in layer ℓ with the same dominant component k, call this set $S_{k,\ell}$. Now, we normalize the weights from every neuron i in $S_{k,\ell}$ going to

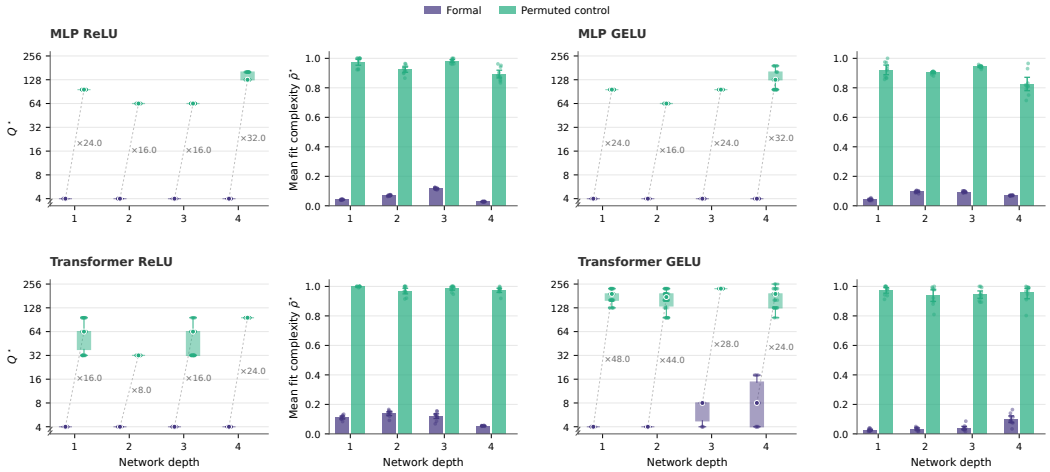

Figure 3: For each architecture and activation setting, results are aggregated over 10 seeds. Column 1 shows, for ReLU MLPs and transformers, the distribution of the minimum number of modes in the general form that must be fit, called $Q^\star$, to preserve 100% full dataset accuracy. Column 2 shows the mean fit complexity $\bar{\rho}^\star$ evaluated at $Q^\star$. Columns 3 and 4 repeat this, but for GELU networks. Center markers denote medians and annotations report the ratio of the permuted-control median $Q^\star$ to the formal median $Q^\star$. Overall, this is strong evidence our general form fits DNN weights well, with few fitted parameters being used, *i.e.* few modes suffice for the fit vs the number of weights.

Table 1: Performance of the general form, fitted across different depths, architectures, and activation functions, each averaged over 50 seeds. At most 16 terms are used in the fit, and this is only necessary in transformer GELU models of depth 4 as shown in Fig. 3.

		ReLU		GELU	
Depth	Architecture	Adj. R^2	Acc.	Adj. R^2	Acc.
1-layer	MLP	0.9832 ± 0.0029	1.000	0.9921 ± 0.0026	1.000
	Transformer	0.9981 ± 0.0020	1.000	0.9998 ± 0.0004	1.000
2-layer	MLP	0.9988 ± 0.0004	1.000	0.9971 ± 0.0072	1.000
	Transformer	0.9997 ± 0.0010	1.000	0.9967 ± 0.0056	1.000
3-layer	MLP	0.9956 ± 0.0031	1.000	0.9930 ± 0.0162	1.000
	Transformer	0.9977 ± 0.0019	1.000	0.9928 ± 0.0098	1.000
4-layer	MLP	0.9929 ± 0.0029	1.000	0.9988 ± 0.0018	1.000
	Transformer	0.9923 ± 0.0043	1.000	0.9840 ± 0.0204	1.000

each next layer target neuron j with: $y_{ij} = \frac{W_{ij} - \widehat{\beta}_{S_{k,\ell+1}}}{\widehat{C}_{S_{k,\ell+1}} \widetilde{A}_i}$, with $\widehat{\beta}_{S_{k,\ell+1}}$ and $\widehat{C}_{S_{k,\ell+1}}$ being the fitted C_j and β_j for each target neuron j in $S_{k,\ell+1}$. We do this for all target neurons in $S_{k,\ell+1}$, and thus normalize every weight y_{ij} going from a neuron in $S_{k,\ell}$ to a target neuron in $S_{k,\ell+1}$.

Similarly, for the forwards view, we normalize the weights leaving a neuron i in $S_{k,\ell}$ and going into any neuron j in $S_{k,\ell+1}$ with: $y_{ij} = \frac{W_{ij} - \widehat{\alpha}_i}{\widehat{D}_i \widetilde{B}_j}$. We do this for all possible target neurons in $S_{k,\ell}$, normalizing all weights of the neural representation.

Now we present the result, using only the simple fit on the neurons in a 2-hidden layer transformer in Figure 5 and we see that the learned neural representation is trying to be as "sinusoidal" as possible (when measured by R^2). This isn't shocking as the activation manifolds found by Moisescu-Pareja et al. (2026) can be inferred to have sinusoidal basis functions, but we now explicitly show it.

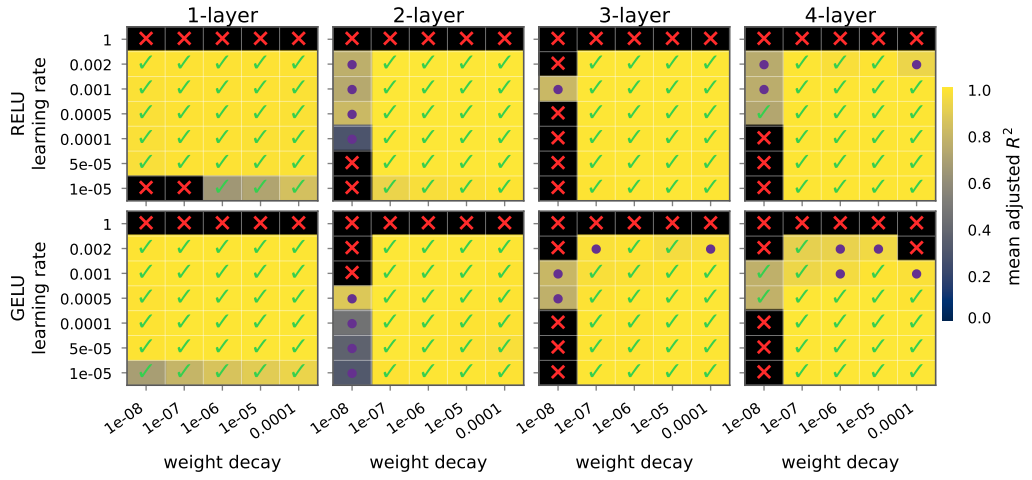

Figure 4: Hyperparameter training grid for MLPs with 10 seeds per training setting. Cell color denotes the mean adjusted R^2 of the general form fitted across seeds. Red crosses indicate settings for which at least one seed fails to reach 100% training accuracy (hyperparameters are not conducive to learning). Purple dots indicate at least one seed failed to maintain 100% task accuracy after weight replacement. Green check marks indicate all seeds maintain 100% dataset accuracy after replacement. Thus, our general form is learned over the majority of the volume where training parameters are conducive to learning, and only fails (purple dot) on some seeds at the boundary of where hyperparameters are not conducive to learning.

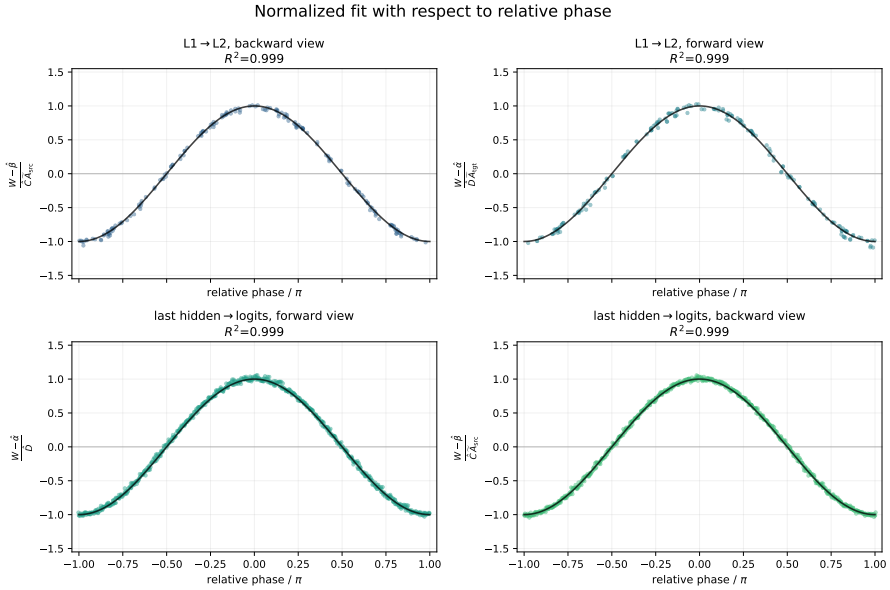

Figure 5: Normalized weights as a function of relative Fourier phase for a two-layer transformer with ReLU activation. The black curve shows the reference $\cos(\Delta\phi)$. The close collapse onto the cosine profile across both interlayer and output-layer views indicates a shared phase-dependent weights distribution structure. Reported R^2 values are averaged across the corresponding neuron-wise fits.

5 DISCUSSION AND CONCLUSION

The main discovery that made this work possible was that the phase information located in the 2d-DFT on the largest Fourier component (k, k) allows for neurons to be "sorted" into a permutation that allows for human-interpretable interpretation of the weights. We speculate that the reason this was missed before, is that the dominant Fourier components in the first layers are $(0, k)$ or $(k, 0)$ and not (k, k) and thus it wasn't inspected by prior work. In fact, the strength of the (k, k) Fourier component in layer 1 preactivations is basically 0 (because neurons in the first layer haven't mixed a and b yet, unless they follow an attention layer) and, thus (k, k) begins to show up weakly only in the first ReLU or GELU non-linear postactivations, which is where we perform our 2d DFT.

With this insight, we present a simple form, later generalized to a general form of the weights in DNNs. Both forms are robust to the permutation invariance between neurons within layers. The general form is required in DNNs, as the feed-forward structure results in more cross-neural-representational mixing, where neurons with other dominant frequencies start to mix into neurons with different frequency in deeper layers. This is particularly bad in GELU networks, and it's worth remarking that GELU networks superimpose neural representations significantly more than ReLU networks, resulting in less-clean interpretations (see Fig. 3).

Overall, this paper serves as a later chapter in the literature on "*what*" is being learned by deep MLPs and transformers trained on modular addition, and should serve to inform the community attempting to research "*why*" stochastic gradient training methods converge to these solutions (training dynamics research). Naturally, it is easier to prove *why* something happens after you know exactly *what* happens.

Fitting into the literature. Our work interfaces nicely with the discussions of the activations in McCracken et al. (2025) and Moisescu-Pareja et al. (2026). The former claimed that DNNs learning modular addition implement the Chinese Remainder Theorem as a global algorithm in their activations (not the group composition by representations algorithm of Chughtai et al. (2023b)). McCracken et al. (2025) showed individual neurons activate on linear combinations of approximate cosets (which result from how sinusoidal functions cut up the cyclic group) and our results explain how these activations arise from the weights preceding neurons. Furthermore, Moisescu-Pareja et al. (2026) inspected the neural representations, being the activation-geometry manifolds that are composed of neurons between layers. They explained that the computation is effectively over by the preactivations of the second layer, and that afterwards, the activation manifolds just rotate as they pass from layer to layer, indeed, it can be inferred from this observation that sinusoidal weights connect neurons between layers, but we directly show this is the case.

5.1 LIMITATIONS AND FUTURE WORK

Limitations. This work only applies to DNNs trained on modular addition.

Future work. We expect this work to serve as proof of concept for discovering how DNNs model non-commutative (non-abelian) group multiplications. For example, the work studying the learned neural representations and learned algorithms in DNNs trained on groups via activation-based interpretability have not yet discovered how DNNs model non-commutativity in non-abelian group multiplications (*e.g.* dihedral (Wei et al.) and alternating groups (Hilgert et al.)).

We expect this work will provide insights to the community studying the training dynamics of modular addition with *deep* neural networks. Additionally, it will be of interest to those studying hidden symmetries in neural networks beyond the obvious permutation symmetry of neurons (Grigsby et al., 2023), as it is to our knowledge, the first description of the form of weights in DNNs, and thus now they can be directly manipulated and observed.

It's interesting that GELU networks require substantially more fitted terms than ReLU (Fig. 3 shows ReLU need only a max of 4, compared to the GELU 4-layer transformer with 16). We speculate it's due to GELU being a smooth, continuous, non-piecewise function, so it does not cleanly isolate frequencies in a fully-connected structure thus encouraging more representational mixing.

AI USE STATEMENT

Generative AI tools were used to polish individual sentences, assist with implementing author-designed methods in code, as well as for debugging and code editing. The research questions, methodological ideas, experimental design, and interpretation of results were developed by the authors. All AI-assisted code was reviewed and validated by the authors.

ETHICS AND REPRODUCIBILITY STATEMENT

This work adheres to the ICLR Code of Ethics. In this study, no human subjects or animal experimentation was involved. All datasets used, including *the modular addition dataset*, were sourced in compliance with relevant usage guidelines, ensuring no violation of privacy. We have taken care to avoid any biases or discriminatory outcomes in our research process. No personally identifiable information was used, and no experiments were conducted that could raise privacy or security concerns. We are committed to maintaining transparency and integrity throughout the research process.

We have made every effort to ensure that the results presented in this paper are reproducible. Code and datasets of the main training and analysis pipeline are included as anonymized supplementary materials to facilitate replication and verification. The experimental setup, including training steps, model configurations, and hardware details, is described in detail in the paper in section A.1. We believe these measures will enable other researchers to reproduce our work and further advance the field.

REFERENCES

Rodney J Baxter. Exactly solved models in statistical mechanics. In *Integrable systems in statistical mechanics*, pp. 5–63. World Scientific, 1985.

Bilal Chughtai, Lawrence Chan, and Neel Nanda. Neural networks learn representation theory: Reverse engineering how networks perform group operations. In *ICLR 2023 Workshop on Physics for Machine Learning*, 2023a.

Bilal Chughtai, Lawrence Chan, and Neel Nanda. A toy model of universality: Reverse engineering how networks learn group operations. In *International Conference on Machine Learning*, pp. 6243–6267. PMLR, 2023b.

Xiaoman Delores Ding, Zifan Carl Guo, Eric J Michaud, Ziming Liu, and Max Tegmark. Survival of the fittest representation: A case study with modular addition. *arXiv preprint arXiv:2405.17420*, 2024.

Darshil Doshi, Aritra Das, Tianyu He, and Andrey Gromov. To grok or not to grok: Disentangling generalization and memorization on corrupted algorithmic datasets. *arXiv preprint arXiv:2310.13061*, 2023.

Darshil Doshi, Tianyu He, Aritra Das, and Andrey Gromov. Grokking modular polynomials. *arXiv preprint arXiv:2406.03495*, 2024.

Elisenda Grigsby, Kathryn Lindsey, and David Rolnick. Hidden symmetries of relu networks. In *International Conference on Machine Learning*, pp. 11734–11760. PMLR, 2023.

Andrey Gromov. Grokking modular arithmetic. *arXiv preprint arXiv:2301.02679*, 2023.

Tianyu He, Darshil Doshi, Aritra Das, and Andrey Gromov. Learning to grok: Emergence of in-context learning and skill composition in modular arithmetic tasks. *arXiv preprint arXiv:2406.02550*, 2024.

Arthur Ayestas Hilgert, Sihui Wei, Doina Precup, Gabriela Moisescu-Pareja, and Gavin McCracken. Toward a general understanding of neural representations learned by deep neural networks on group multiplications. In *Topology, Algebra, and Geometry in Data Science*.

Daniel Kunin, Giovanni Luca Marchetti, Feng Chen, Dhruva Karkada, James Simon, Michael Deweese, Surya Ganguli, and Nina Miolane. Alternating gradient flows: A theory of feature learning in two-layer neural networks. *Advances in Neural Information Processing Systems*, 38:4377–4424, 2026.

Henry Kvinge, Andrew Aguilar, Nayda Farnsworth, Grace O'Brien, Robert Jasper, Sarah Scullen, and Helen Jenne. Can neural networks learn small algebraic worlds? an investigation into the group-theoretic structures learned by narrow models trained to predict group operations. *arXiv preprint arXiv:2601.21150*, 2026.

Ziming Liu, Ouail Kitouni, Niklas S Nolte, Eric Michaud, Max Tegmark, and Mike Williams. Towards understanding grokking: An effective theory of representation learning. *Advances in Neural Information Processing Systems*, 35:34651–34663, 2022.

Neil Mallinar, Daniel Beaglehole, Libin Zhu, Adityanarayanan Radhakrishnan, Parthe Pandit, and Mikhail Belkin. Emergence in non-neural models: grokking modular arithmetic via average gradient outer product. *arXiv preprint arXiv:2407.20199*, 2024.

Gavin McCracken. *Using Exact Models to Analyze Policy Gradient Algorithms*. McGill University (Canada), 2021.

Gavin McCracken, Gabriela Moisescu-Pareja, Vincent Letourneau, Doina Precup, and Jonathan Love. Uncovering a universal abstract algorithm for modular addition in neural networks, 2025. URL https://arxiv.org/abs/2505.18266.

Gavin McCracken, Arthur Ayestas Hilgert, Sihui Wei, Gabriela Moisescu-Pareja, Zhaoyue Wang, and Jonathan Love. Interpreting deep neural networks trained on elementary p groups reveals algorithmic structure. In *Topology, Algebra, and Geometry in Data Science*, 2026.

Mohamad Amin Mohamadi, Zhiyuan Li, Lei Wu, and Danica Sutherland. Grokking modular arithmetic can be explained by margin maximization. In *NeurIPS 2023 Workshop on Mathematics of Modern Machine Learning*, 2023. URL https://openreview.net/forum?id=QPMfCLnIqf.

Gabriela Moisescu-Pareja, Gavin McCracken, Harley Wiltzer, Colin Daniels, Vincent Létourneau, Doina Precup, and Jonathan Love. On the geometry and topology of representations: the manifolds of modular addition. In *International Conference on Learning Representations*, volume 2026, pp. 402–422, 2026.

Depen Morwani, Benjamin L. Edelman, Costin-Andrei Oncescu, Rosie Zhao, and Sham M. Kakade. Feature emergence via margin maximization: case studies in algebraic tasks. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=i9wDX850jR.

Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. Progress measures for grokking via mechanistic interpretability. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=9XFSbDPmdW.

Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. Grokking: Generalization beyond overfitting on small algorithmic datasets, 2022. URL https://arxiv.org/abs/2201.02177.

Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

Dashiell Stander, Qinan Yu, Honglu Fan, and Stella Biderman. Grokking group multiplication with cosets. In *Forty-first International Conference on Machine Learning*, 2024.

Tao Tao, Darshil Doshi, Dayal Singh Kalra, Tianyu He, and Maissam Barkeshli. (how) can transformers predict pseudo-random numbers? In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=asDx9sPAUN.

Yuandong Tian. Provable scaling laws of feature emergence from learning dynamics of grokking. *arXiv preprint arXiv:2509.21519*, 2025.

Yuandong Tian. Composing global solutions to reasoning tasks via algebraic objects in neural nets. *Advances in Neural Information Processing Systems*, 38:70553–70586, 2026.

Sihui Wei, Gavin McCracken, Gabriela Moisescu-Pareja, Harley Wiltzer, Doina Precup, Irina Rish, and Jonathan Love. Deep neural networks divide and conquer dihedral multiplication. In *Forty-third International Conference on Machine Learning*.

Wilson Wu, Louis Jaburi, Jason Gross, et al. Towards a unified and verified understanding of group-operation networks. In *International Conference on Learning Representations*, volume 2025, pp. 61683–61714, 2025.

Chun Hei Yip, Rajashree Agrawal, Lawrence Chan, and Jason Gross. Modular addition without black-boxes: Compressing explanations of mlps that compute numerical integration, 2024. URL https://arxiv.org/abs/2412.03773.

Ziqian Zhong, Ziming Liu, Max Tegmark, and Jacob Andreas. The clock and the pizza: Two stories in mechanistic explanation of neural networks. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=S5wmbQc1We.

A Appendix

A.1 Experimental settings

Table 2: Training hyperparameters for results in the main paper.

Model	Depth	ReLU (lr / wd)	GELU (lr / wd)
MLP	1-layer	$5 \times 10^{-4}/10^{-6}$	$5 \times 10^{-4}/10^{-6}$
MLP	2-layer	$5 \times 10^{-4}/10^{-6}$	$5 \times 10^{-4}/10^{-6}$
MLP	3-layer	$5 \times 10^{-4}/10^{-5}$	$5 \times 10^{-4}/10^{-5}$
MLP	4-layer	$1 \times 10^{-4}/10^{-5}$	$1 \times 10^{-4}/10^{-5}$
Transformer	1-layer	$5 \times 10^{-4}/10^{-6}$	$5 \times 10^{-4}/10^{-6}$
Transformer	2-layer	$5 \times 10^{-4}/10^{-5}$	$5 \times 10^{-4}/10^{-5}$
Transformer	3-layer	$5 \times 10^{-4}/10^{-5}$	$5 \times 10^{-4}/10^{-5}$
Transformer	4-layer	$5 \times 10^{-4}/10^{-6}$	$5 \times 10^{-4}/10^{-4}$

The Transformer consists of an attention block followed by a multi-layer MLP, with residual connections around both components. The two modular inputs are represented as a short token sequence with shared token embeddings and learned positional embeddings.

All experiments are trained for 5000 epochs using Adam. All experiments were done on CPUs and the specific hyperparameters used while training are shown in Table 2.

The set $\mathcal{Q}_j$ is selected separately for each target neuron and is determined before fitting C_{jk}, θ_{jk}, or β_j. Selection is greedy: at each step, each remaining Fourier component is scored by the target energy it explains and by how much nonredundant source-side information it adds beyond the components already selected. Importantly, the observed weights W_{ij} are not used to rank or select modes; they are used only afterward to fit β_j, C_{jk}, and θ_{jk}

A.2 Binary search pruning

We prune neurons iteratively while requiring the pruned model to remain 100% accurate on the full n^2 input grid. Within each layer, neurons are ranked by an importance score that combines their maximum activation with the magnitude of their associated weights: $s_i = \max_x |h_i(x)| \cdot \max_\ell |w_{i\ell}|$.

Lower-scoring neurons are considered for removal first. We use binary search over this ranking to find the largest set of low-scoring neurons that can be removed while preserving full-grid accuracy and keeping the increase in cross-entropy within a fixed tolerance relative to the original model.

The pruned neurons are masked by zeroing their biases and all outgoing and incoming weights. We repeat this procedure across layers in both forward and backward order until a complete forward-backward cycle removes no additional neurons.

A.3 Forward/backward view accordance

Curve concordance. For each common edge (i, j), we normalize its weight in the backward and forward views:

$$z_{ij}^{(B)} = \frac{W_{ij} - \hat{\beta}_j}{\hat{C}_j \widetilde{A}_i}, \qquad z_{ij}^{(F)} = \frac{W_{ij} - \hat{\beta}_i}{\hat{C}_i \widetilde{B}_j}. \tag{1}$$

Here $\widetilde{A}_i$ and $\widetilde{B}_j$ are the source post-activation and target pre-activation carrier amplitudes, respectively; each view has its own fitted intercept and scale.

After aligning the phase conventions of the two views, we divide $\Delta\phi \in [-\pi, \pi]$ into 24 bins. For each bin $\mathcal{B}_k$, we compute the median normalized weight in the backward and forward views:

$$m_k^{(B)} = \text{median}\left\{ z_{ij}^{(B)} : \Delta\phi_{ij} \in \mathcal{B}_k \right\}, \qquad m_k^{(F)} = \text{median}\left\{ z_{ij}^{(F)} : \Delta\phi_{ij} \in \mathcal{B}_k \right\}. \tag{2}$$

These values define the two binned curves $m^{(B)} = (m_1^{(B)}, \ldots, m_{24}^{(B)})$ and $m^{(F)} = (m_1^{(F)}, \ldots, m_{24}^{(F)})$. We measure their agreement using the concordance correlation coefficient (CCC):

$$\text{Curve CCC} = \frac{2\,\text{Cov}(m^{(B)}, m^{(F)})}{\text{Var}(m^{(B)}) + \text{Var}(m^{(F)}) + (\bar{m}^{(B)} - \bar{m}^{(F)})^2}, \tag{3}$$

where $\bar{m}^{(B)}$ and $\bar{m}^{(F)}$ denote the means of the 24 binned curve values. A CCC of 1 indicates that the two binned curves coincide.

We use cross-fitting to avoid evaluating the normalization on the same weights used to estimate it. We randomly split the weights shared by the two views into two halves. For each view, we estimate the normalization parameters on the first half and apply them to the second half, then reverse the roles of the two halves. We pool the resulting held-out normalized weights, construct the two binned curves, and compute a single cross-fitted CCC. We repeat this procedure over 20 random splits and report the mean CCC.

Table 3: Curve CCC across 1- and 2-layer models (mean $\pm$ standard deviation across 10 seeds). "Output" denotes the mapping between the final hidden layer and the logits.

Model	Activation	Depth	Layer	Curve CCC
MLP	ReLU	1	Output	1.000 ± 0.000
MLP	ReLU	2	Interlayer	0.998 ± 0.011
MLP	ReLU	2	Output	0.997 ± 0.008
MLP	GELU	1	Output	1.000 ± 0.000
MLP	GELU	2	Interlayer	0.987 ± 0.052
MLP	GELU	2	Output	0.997 ± 0.008
Transformer	ReLU	1	Output	0.998 ± 0.004
Transformer	ReLU	2	Interlayer	0.998 ± 0.004
Transformer	ReLU	2	Output	0.993 ± 0.016
Transformer	GELU	1	Output	0.965 ± 0.075
Transformer	GELU	2	Interlayer	0.969 ± 0.072
Transformer	GELU	2	Output	0.973 ± 0.063

From Table 3, across architectures and activations, the forward and backward views yield highly concordant phase-dependent weight curves, with CCC values close to one in nearly all settings. The agreement remains strong for both interlayer and output connections. Overall, these results indicate that the two directional views recover largely consistent structure in the learned weights.

A.4 General form fit on modulus 71

Table 4: Performance of the general form across different depths, architectures, and activation functions, averaged across 10 seeds per setting. At most 18 terms are used in the fit, and this is only necessary in transformer GELU models of depth 4, which suffer from representational mixing as previously discussed in the main text.

		ReLU		GELU	
Depth	Architecture	Adj. R^2	Acc.	Adj. R^2	Acc.
1-layer	MLP	0.9862 ± 0.0027	1.000	0.9932 ± 0.0018	1.000
	Transformer	0.9983 ± 0.0018	1.000	0.9998 ± 0.0004	1.000
2-layer	MLP	0.9991 ± 0.0003	1.000	0.9993 ± 0.0004	1.000
	Transformer	0.9999 ± 0.0001	1.000	0.9986 ± 0.0016	1.000
3-layer	MLP	0.9938 ± 0.0047	1.000	0.9831 ± 0.0252	1.000
	Transformer	0.9983 ± 0.0010	1.000	0.9982 ± 0.0028	1.000
4-layer	MLP	0.9906 ± 0.0031	1.000	0.9996 ± 0.0005	1.000
	Transformer	0.9913 ± 0.0028	1.000	0.9971 ± 0.0035	1.000

A.5 HYPERPARAMETER TUNING HEATMAPS FOR MLPS AND TRANSFORMERS

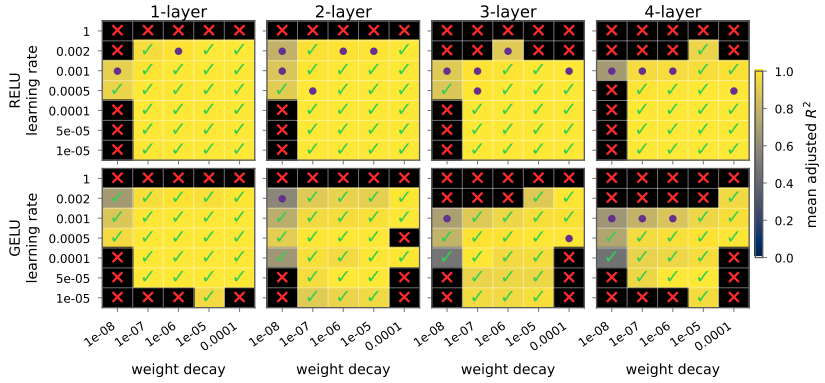

Figure 6: Hyperparameter sweep over learning rate and weight decay for transformers of different depths and activation functions, using 10 seeds per setting and constraining $\rho < 0.2$. Cell color denotes the mean adjusted R^2 across seeds. The markers (red "×", purple "○", and green "✓") follow the same conventions as Fig. 4.

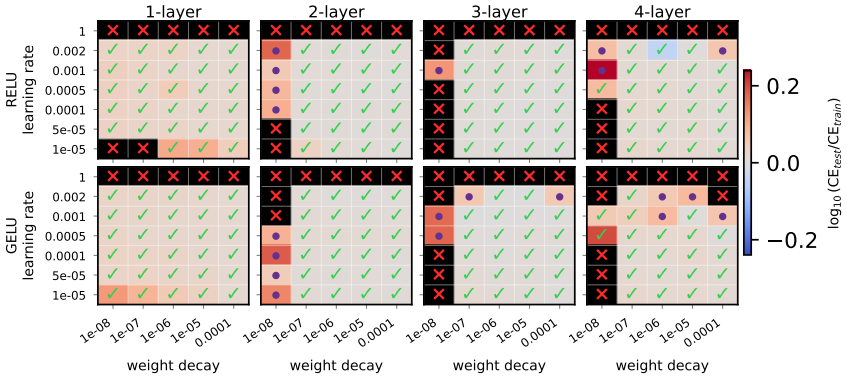

Figure 7: Hyperparameter sweep over learning rate and weight decay for MLPs of different depths and activation functions, using 10 seeds per setting. Cell color denotes mean $\log_{10}\left(\frac{\mathrm{CE}_{\mathrm{test}}}{\mathrm{CE}_{\mathrm{train}}}\right)$. The markers (red "×", purple "○", and green "✓") follow the same conventions as Fig. 4.

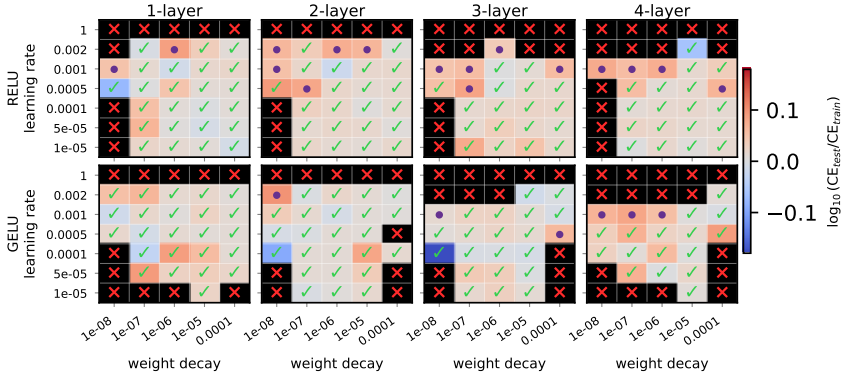

Figure 8: Hyperparameter sweep over learning rate and weight decay for transformers of different depths and activation functions. The color and markers (red "×", purple "○", and green "✓") follow the same conventions as Fig. 7.

The general form is learned over the majority of the volume where training parameters are conducive to learning, and only fails (purple dot) on some seeds at the boundary of where hyperparameters are not conducive to learning.

A.6 Additional control experiment

Table 5: Control experiment using randomly selected Fourier modes. For each target neuron, we randomly select the same number of modes as in the main analysis. Results are averaged across depths and 10 seeds per setting. For the logit reconstruction error, we first center the original and fitted logits by subtracting the mean across output classes for each input, and then compute the normalized squared difference between the two.

Model	Activation	Accuracy	Logit Error	Adjusted R^2
MLP	GELU	0.124	1.0863	0.5666
MLP	ReLU	0.114	0.9029	0.3557
Transformer	GELU	0.524	0.1760	0.4085
Transformer	ReLU	0.091	0.8825	0.3824

Compared with the results from the selected modes in the main analysis, randomly selecting the same number of Fourier modes substantially degrades fit quality and replacement performance across all model and activation settings. This control shows that the results do not arise simply from using a fixed number of Fourier components.

A.6.1 Empirical proof the 2d DFT doesn't guarantee the general fit works with few terms (Fourier modes)

The point of this experiment is to quickly alleviate concerns a reader may have that our utilization of the 2d DFT on activations to get phase information could end up guaranteeing the general form is easily fit with only a few terms.

The preactivations of a target neuron in the next layer are:

$$z_j(a,b) = \sum_i W_{ij} h_i(a,b) + b_j.$$

Taking the Fourier coefficient at mode k

$$\widehat{z}_j(k) = \sum_i W_{ij}\, \widehat{h}_i(k).$$

Writing the source Fourier coefficients in amplitude-phase form:

$$\widehat{h}_i(k) = A_i(k) e^{i\phi_i(k)},$$

gives

$$\widehat{z}_j(k) = \sum_i W_{ij} A_i(k) e^{i\phi_i(k)}.$$

Therefore the target phase is

$$\phi_j(k) = \arg\left(\sum_i W_{ij} A_i(k) e^{i\phi_i(k)} \right).$$

This makes the dependence on the weights explicit:

$$\phi_j(k) = \phi_j(k; \{W_{ij}\}_i)\,.$$

The paper then models the weights using this target phase, e.g.

$$W_{ij} \approx \beta_j + C_j \widetilde{A}_i \cos(\phi_i - \phi_j)\,.$$

So concerns of potential circularity can be summarized compactly as

$$W_{ij} \longrightarrow z_j \longrightarrow \phi_j \longrightarrow \widehat{W}_{ij}.$$

Thus, to show this isn't the case, we take 2-hidden layer networks (MLP and transformer with GELU and ReLU), and do the normal general form fit allowing a max of 4 Fourier modes in the sum. Then, as a control, we hold the postactivations after layer 1 fixed, and replaced the learned weights from the first to the second hidden layer with random weight vectors matched to the mean and scale of the learned weights, and fit them with the general form, using the 2d DFT on the source postactivations and target preactivations as usual. Resultantly, the R^2 of the random-weight fits is poor, as shown in Table 6, demonstrating that the downstream dependence of the target Fourier coordinates on the weights does not by itself produce a high quality general-form fit. Instead, the learned weights admit a substantially more compact description in terms of the Fourier structure of the activations.

Table 6: Adjusted R^2 of the Fourier weight fit for learned weights and a random-weight null in 2-layer networks, using at most 4 Fourier modes per target neuron. For the random-weight control, source activations are held fixed, random weight vectors are generated, the resulting target preactivations are recomputed, and the same Fourier analysis pipeline is applied. Results are averaged across 5 seeds per setting.

Architecture	Activation	Learned adj. R^2	Random adj. R^2
MLP	GELU	0.9950 ± 0.0067	0.1362 ± 0.0212
MLP	ReLU	0.9982 ± 0.0002	0.0623 ± 0.0045
Transformer	GELU	0.9958 ± 0.0075	0.0054 ± 0.0059
Transformer	ReLU	0.9993 ± 0.0043	0.0533 ± 0.0140

A.7 Additional figures showing the permuted control require substantially more fitted terms to maintain accuracy

Fig.9 shows the recovery curves for the GELU 4-layer control experiment, where substantially more modes are required to recover high accuracy.

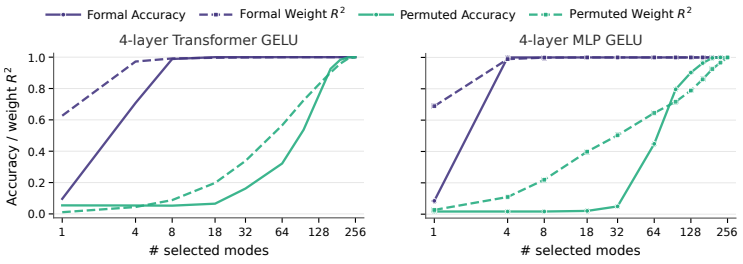

Figure 9: Recovery curves for the GELU depth-4 control experiment. Solid lines show replacement accuracy and dashed lines show weight R^2 as the number of selected Fourier modes increases. Across all four model settings, the permuted control requires substantially more modes to recover high replacement accuracy, despite eventually reaching similar performance at sufficiently large mode counts.

A.8 Pearson correlation showing mixtures of representations (different frequency cosines) negatively correlate with simple fit performance

We show this quantitatively below: for neuron i, we define representation mixture as $m_i = 1 - \frac{E_i^{\text{dom}}}{E_i^{\text{total}}}$, where $E_i^{\text{dom}} = \max_k E_i(k)$, $E_i^{\text{total}} = \sum_k E_i(k)$, and $E_i(k)$ denotes the Fourier energy of neuron i at component k. Thus, m_i measures the fraction of Fourier energy lying outside the dominant component. Values near zero indicate that the representation is concentrated in a single Fourier component, while larger values indicate greater mixing across components.

Table 7: Association between representation mixture and simple-form fit. The goodness of fit of the simple form decreases with increasing representation mixture. Pearson correlations are computed across layer-level means.

	ReLU	GELU
MLP	-0.51	-0.97
Transformer	-0.79	-0.81

We now compute the Pearson correlation between representation mixture and simple-form fit R^2 in Table 7, and see that it is negatively correlated. This motivates our general form of the weights.

A.9 Logit construction error

Table 8: Logit reconstruction error for the general-form fits on modulus 59, across depths, architectures, and activation functions, averaged over 50 seeds per setting. For each input, we first subtract the mean across output classes from both the original and reconstructed logits, and then compute their normalized squared difference over the full input grid.

Depth	Architecture	ReLU	GELU
1-layer	MLP	1.78×10^{-3}	5.34×10^{-4}
	Transformer	5.14×10^{-4}	1.00×10^{-6}
2-layer	MLP	2.16×10^{-4}	6.94×10^{-4}
	Transformer	4.33×10^{-4}	3.70×10^{-5}
3-layer	MLP	2.65×10^{-3}	2.57×10^{-3}
	Transformer	3.78×10^{-3}	3.40×10^{-5}
4-layer	MLP	1.74×10^{-2}	2.86×10^{-4}
	Transformer	1.79×10^{-2}	2.21×10^{-4}

Table 9: Logit reconstruction error for the general-form fits on modulus 71, averaged across 10 seeds per setting. For each input, we subtract the mean across output classes from both the original and fitted logits, and then compute their normalized squared difference over the full input grid.

Depth	Architecture	ReLU	GELU
1-layer	MLP	1.38×10^{-3}	3.25×10^{-4}
	Transformer	6.16×10^{-4}	6.00×10^{-6}
2-layer	MLP	1.81×10^{-4}	2.30×10^{-5}
	Transformer	7.40×10^{-5}	2.00×10^{-6}
3-layer	MLP	3.87×10^{-3}	4.27×10^{-3}
	Transformer	2.95×10^{-3}	6.00×10^{-6}
4-layer	MLP	2.19×10^{-2}	8.30×10^{-5}
	Transformer	2.25×10^{-2}	2.44×10^{-5}