MECHANISTIC INTERPRETABILITY OF IRREDUCIBLE INTEGER IDENTIFIERS

NOAH SYRKIS AND ANDERS SØGAARD

Abstract. This paper investigates how neural networks learn to solve multiple related mathematical tasks simultaneously, through the lens of mechanistic interpretability (MI). A transformer model is trained on 29 parallel tasks, each requiring the prediction of remainders when dividing two-digit base-113 numbers—as has been the domain of previous MI work [1]—by all potential prime factors less than 113. This setup naturally creates a spectrum of task complexity, from binary classification (division by 2) to 109-way classification (division by 109). Analysis of the model's learned representations indicates that after independently solving the first four tasks (mod 2, 3, 5, and 7), the model develops a shared computational strategy that enables rapid generalization to the remaining 25 tasks. Additionally, findings [2] that show amplifying slow-moving gradients significantly accelerates this generalization process, are reproduced. The results provide insights into how neural networks discover and implement mathematical algorithms, particularly when learning multiple related tasks of varying complexity. Specifically, a phase transition *after* generalization in which circuits merge is indicated, encouraging the community to explore this phenomnon. Project repo is `https://github.com/syrkis/miiii`.

1. Introduction

Recent years have seen deep learning (DL) models achieve remarkable proficiency in complex computational tasks, including protein structure prediction [3], strategic reasoning [4], and natural language generation [5]—areas previously thought to be the exclusive domain of human intelligence. Traditional (symbolic) programming allows functions like $f(x, y) = \cos(a \cdot x) + \sin(b \cdot y)$ to be implemented in code with clear typographical isomorphism—meaning the code's structure directly mirrors the mathematical notation. For example, in Haskell: `f x y = cos(a * x) + sin(b * y)`. In contrast, DL models are inherently sub-symbolic, meaning that the models' atomic constituents (often 32-bit floating-point numbers centered around 0) are meaningless when viewed directly. For reference, Appendix D shows a DL-based implementation of the affore mentioned function. Indeed, the increasing prevalence of DL can be understood as a transition from symbolic to sub-symbolic algorithms.

Precursors to modern DL methods learned how to weigh human-designed features [6], with later works learning to create features from data to then weigh [7], [8]—in combination with tree search strategies, in the case of games [9]. Very recent DL work has even eliminated tree search in the case of chess, mapping directly from observation space to action space [10]. Pure DL methods are thus becoming ubiquitous but remain largely inscrutable, with recent works still attempting to define what interpretability even means in the DL context [11]. Given the breadth [12] of tasks that DL models can be (and are) trained to solve—along with their sub-symbolic nature—it is, however, hardly a surprise that their interpretation remains difficult.

Mathematically, DL refers to a set of methods that combine linear maps (matrix multiplications) with non-linearities (activation functions). Formally, all the potential numerical values of a given model's weights W can be thought of as a hypothesis space $\mathcal{H}$. Often, $\mathcal{H}$ is determined by human decisions (number of layers, kinds of layers, sizes of layers, etc.). $\mathcal{H}$ is then navigated using some optimization heuristic, such as gradient descent, in hope of finding a W that "performs well" (i.e., successfully minimizes some loss $\mathcal{L}$ often computed by a differentiable function) on whatever training data is present. This vast, sub-symbolic hypothesis space, while enabling impressive performance and the solving of relatively exotic[1] tasks, makes it challenging to understand how any one particular solution actually works.

The ways in which a given model can minimize $\mathcal{L}$ can be placed on a continuum: on one side, we have overfitting, remembering the training data, (i.e. functioning as an archive akin to lossy and even lossless compression); and on the other, we have generalization, learning the rules that govern the relationship between input and output (i.e. functioning as an algorithm).

When attempting to give a mechanistic of a given DL model's behavior, it entails the exsistence of a mechanism. MI assumed this mechanism to be general, thus mkaing generalization a necessary (though insufficient) condition. Generalization ensures that there *is* an algorithm present to be uncovered (necesity); however, it is possible for that algorithm to be so obscurely implemented that reverse engineering, for all intents and purposes, is impossible (insufficiceny). Various forms of regularization are used to incentivize the emergence of algorithmic (generalized) rather than archiving (overfitted) behavior [13], [14], [15].

As of yet, no MI work has explored the effect of multitask learning, the focus of this paper. Multitask learning also has a regularizing effect [16]. Formally, the set of hyptheses spaces for each task of a set of tasks (often called environment) is denoted by $\mathcal{H} \in \mathbb{H}$. When minising the losses across all tasks in parallel, generalising W's are thus incentivised, as these help lower loss across tasks (in contract to memorizing W's that lower loss for one task). A W derived from a multi-task training process can thus be thought of as the intersection of the high-performing areas of all $\mathcal{H} \in \mathbb{H}$.

In this spirit, the present paper builds on the work of Nanda et al. (2023), which trains a transformer [17] model to perform modular addition, as seen in Eq. 1, as task denoted as $\mathcal{T}_{\text{nanda}}$ throughout the paper.

$$(x_0 + x_1) \bmod p, \quad \forall x_0, x_1 < p, \quad p = 113 \tag{1}$$

The task of this paper focuses on predicting remainders modulo all primes q less than p, where x is interpreted as $x_0 p^0 + x_1 p^1$, formally shown in Eq. 2, and is referred to as $\mathcal{T}_{\text{miiii}}$:

$$(x_0 p^0 + x_1 p^1) \bmod q, \quad \forall x_0, x_1 < p, \quad \forall q < p, \quad p = 113 \tag{2}$$

$\mathcal{T}_{\text{miiii}}$ differentiates itself from $\mathcal{T}_{\text{nanda}}$ in two significant ways: *1)* it is non-commutative, and *2)* it is, as mentioned, multitask. These differences present unique challenges for mechanistic interpretation, as the model must learn to handle both

[1]Try manually writing a function in a language of your choice that classifies dogs and cats from images.

the order-dependent nature of the inputs and develop shared representations across multiple modular arithmetic tasks. Further, as $\mathcal{T}_{\text{miiii}}$ is harder than $\mathcal{T}_{\text{nanda}}$ the model can be expected to generalize slower when trained on the former. Therefore, Lee et al. (2024)'s recent work on speeding up generalization, by positing the model parameters gradients through time can be viewed as the sum a slow varying generalizing component (which is boosted) and a quick varying overfitting component (which is suppressed), was (successfully) replicated to make training tractable.

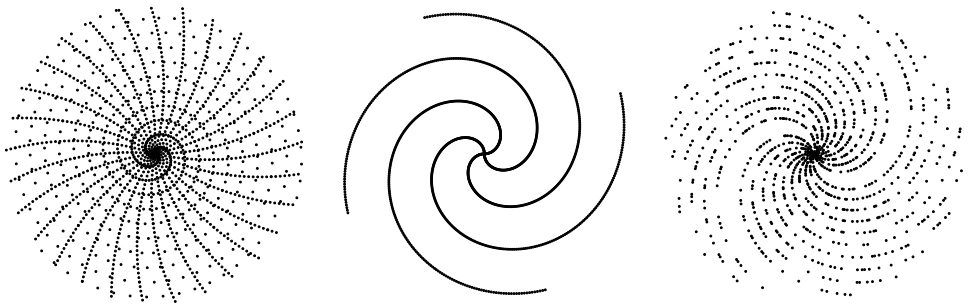

FIGURE 1. Periodic patterns in polar coordinates $(n, n \bmod \tau)$ for numbers less than 12 769. Left: numbers with remainder 0 mod 17 or 23 (see the two spirals). Middle: numbers with remainder 0 mod 11. Right: prime numbers.

More generally, modular arithmetic on primes is a particularly useful task for MI as it ensures uniformity among the output classes, allows for comparison with other MI work [1], and, from a number-theoretic point of view, primes contain mysteries ranging from the trivially solved—are there an infinite number of primes?—to the deceptively difficult—can all even numbers larger than 4 be described as the sum of two primes? The latter, known as Goldbach's Conjecture, remains unsolved after centuries. The choice of using every prime less than the square root of the largest number of the dataset, also serves the following purpose: to test if a given natural number is prime, it suffices to test that it is not a multiple of any prime less than its square root—the set of tasks trained for here, can thus be viewed in conjunction as a single prime detection task (primes are the only samples whose target vector is the zero vector).

To provide insight into the periodic structure of these remainders mod 1 for natual numbers less than 12 769 (and motivate thinking in rotational terms), Figure 1 visualizes various modular patterns in polar coordinates $(n, n \bmod 2\pi)$. One could imagine tightening and loosening the spiral by multiplying τ by a constant, to align multiples of a given number in a straight line (imagining this is encouraged).

2. Background and related work

2.1. **Generalization and grokking.**

Power et al. (2022) shows generalization can happen "[...] well past the point of overfitting", dubbing the phenomenon "grokking". The phenomenon is now well

established [1], [19], [20]. Nanda et al. (2023) shows that a generalized circuit "arises from the gradual amplification of structured mechanisms encoded in the weights," rather than being a relatively sudden and stochastic encounter of an appropriate region of $\mathcal{H}$. The important word of the quote is thus "gradual". Further, by regarding the series of gradients in time as a stochastic signal, Lee et al. proposes decomposing the signal into two components: a fast-varying overfitting component and a slow-varying generalization component. They show that amplification of the slow-varying component accelerates grokking substantially (more than fifty-fold in some cases). This echoes the idea that generalized circuits go through gradual amplification [1]. To the extent that this phenomenon is widespread, it bodes well for generalizable deep learning, in that the generalizing signal that one would want to amplify exists long before the model is fully trained, and might be boosted in a targeted way my the method described by Lee et al..

Conceptually, Lee et al. argues that in the case of gradient descent, the ordered sequence of gradient updates can be viewed as consisting of two components: *1)* a fast varying overfitting component, and *2)* a slow varying generalizing components. The general algorithm explaining the relationship between input and output is the same for all samples, whereas the weights that allow a given model to function are unique for all samples. Though not proven, this intuition bears out in that generalization is sped up fifty-fold in some cases.

2.2. **Mechanistic Interpretability (MI).**

2.2.1. *Foundations and definitions.*

MI is a relatively new field focused on reverse-engineering the internal mechanisms of neural networks. Lipton (2018) contrasts MI with other forms of interpretability, such as feature importance analysis. While feature importance measures correlations between inputs and outputs (e.g., red pixels correlating with "rose" classifications), MI aims to understand how the model actually processes information.

2.2.2. *Current methods and tools.*

Methods and tools used so far in MI include: Activation visualization across large datasets; Singular value decomposition of weight matrices; Ablation studies to identify critical circuits; Circuit discovery automation 21, Weiss, G., Goldberg, Y., Yahav, E. [22]'s RASP language demonstrates how architectural constraints can be made explicit, helping researchers "think like a transformer" by expressing computation in terms of the architecture's native operations (attention as reduction, MLP as mapping). Current research is being done (CITE GROW AI?) into how to grow and merge neural network models, indicating a potential for composition of networks, with "modules" fulfilling certain properties.

2.2.3. *Case study: modular addition.*

Nanda et al. (2023)'s analysis of a transformer trained on modular addition ($\mathcal{T}_{\text{nanda}}$) exemplifies MI methodology. They discovered that:

- The embedding layer learns trigonometric lookup tables
- The feed-forward network combines these through multiplication
- The final layer performs the equivalent of argmax

This implementation exploits the commutative property of modular addition (Eq. 3):

$$(x_0 + x_1) \bmod p = (x_1 + x_0) \bmod p \tag{3}$$

2.2.4. *Theoretical context.*

While MI provides concrete insights into specific models, broader theoretical understanding of deep learning remains elusive. Different frameworks compete to explain the successes and limitations (and underlying theory) of DL.

- Information theory [23]
- Geometric approaches [24]
- Category theory [25]

This theoretical uncertainty leads MI researchers to focus on simple algorithmic tasks with known solutions and architectures using ReLU activation functions, which favor interpretable orthogonal representations [1].

2.2.5. *Open questions.*

The observation that networks can transition from memorization to algorithmic solutions raises several questions:

1. Can we bypass the memorization phase?
2. What determines which algorithms emerge?
3. How does multi-task learning affect algorithm discovery?
4. How can memorization and algorithmic computation coexist?
5. What other tricks than slow gradient boosting [2] can speed up grokking?
6. If a circuit is general, can we make proofs about the model;s function?

These questions are central to both theoretical understanding and practical applications of deep learning.

2.2.6. *Circuits.*

The term circuit is frequently used in MI. It refers to a distinct subset of the network, performing a specific task. As of yet, the notions of circuits in neural networks are relatively informal (i.e., the algebra of circuits from electrical engineering and neuroscience has not yet been rigorously applied).

2.3. **Multi-task learning in DL.**

As stated, multitask learning has been shown to have a regularizing effect [16], [26] as the hypothesis $\mathbf{W}$ that performs well across the of hypothesis spaces $\mathbb{H}$ is more likely to be general. Viewed information theoretically, this concept is reminiscent of Shannon (2001)'s asymptotic equipartition property [28], or even more generally, the law of large numbers, which states that the more samples we have of a distribution, the closer our estimates of its underlying properties will align with the true underlying property.

In the DL context, multitask learning is done by having the last layer output predictions for multiple tasks independently. Thus, whereas $\mathcal{T}_{\text{nanda}}$ outputs a single 1×113 vector for each of the potential remainders, $\mathcal{T}_{\text{miiii}}$, as we shall see, outputs one vector for each prime $f < p$ (29 when $p = 113$), each of which has shape $1 \times f$. The embeddings layer and the transformer block is thus shared for all tasks, meaning that representations that perform well across tasks are incentivized.

2.4. **Loss functions and training dynamics.**

Perhaps the most widespread loss functions used in deep learning are mean cross-entropy `Eq. 4.1` (for classification) and mean squared error `Eq. 4.2` (for regression).

$$L_{\text{MCE}} = \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{k} y_{p_{ij}} \ln \left(\frac{1}{\hat{y}_{p_{ij}}} \right) \tag{4.1}$$

$$L_{\text{MSE}} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{4.2}$$

These have various computational and mathematical properties that make them convenient to use, while they, however, struggle to generalize [29]. Due to its prevalence, however, MCE is chosen in this paper. However, since we have multiple tasks, the MCE is modified as shown in

2.5. **Deep number theory.**

Multiple papers describe the use of deep learning to detect prime numbers [30], [31], [32]. None are particularly promising as prime detection algorithms, as they do not provide speedups, use more memory, or are less accurate than traditional methods. However, in exploring the foundations of deep learning, the task of prime detection is interesting, as it is a simple task that is difficult to learn, and is synthetic, meaning that the arbitrary amounts of data are generated by a simple algorithm.

Prime numbers, in particular, are an interesting domain for deep learning. A frequent feature of number theoretical problems is the ease with which they can be stated. This is true for trivial problems (such as proving there are infinitely many primes) and deceptive problems (such as "all even numbers can be expressed as the sum of two primes"). The latter, known as Goldbach's conjecture, remains unsolved. There are about $\frac{n}{\ln(n)}$ primes less than n. To test if a given number n is prime, it is sufficient to test if it is divisible by any prime less than $\sqrt{n}$ (Sieve of Eratosthenes), of which there are about $\frac{\sqrt{n}}{\ln(\sqrt{n})}$.

2.6. **Transformer architecture.**

Various modifications/simplifications have been made to the transformer block [33], [34]. Transformers combine self-attention (a communication mechanism) with feed-forward layers (a computation mechanism). Importantly, transformers tend to rely on residual streams (I will elaborate). I am currently using the original transformer block, but I want to switch to [33]'s block, as it is simpler and more interpretable—but there is not much research on it yet.

Traditional loss functions like cross-entropy and mean squared error, have been shown to not generalize well to out-of-distribution data [23]. Indeed, additional regularization techniques are a hallmark of many modern architectures, the most extreme example of which is perhaps the original transformer [17]—layer norm [13], dropout, weight decay, residual connections, are all integral components of the original architecture, though recent years have seen simplifications yielding similar performance [33]. Importantly, deep learning architectures can function both as archives—overfitting to training data—and as generalized algorithms [18].

3. Methods

However, how exactly a given model implements an algorithm is a non-trivial question—as we shall see, even modular addition is implemented in an obscure way [1]. This investigation probes the fundamental algorithmic structures internalized by a transformer model trained on a set of basic prime number-related modular arithmetic tasks, with slight variations in complexity. This approach provides insights into how and why specific algorithmic patterns emerge from seemingly straightforward learning processes.

My setup thus differentiates itself from $\mathcal{T}_{\text{nanda}}$ in two crucial ways:

1. Mine is non-commutative.
2. It is multitask.

A model deep learning model, $\mathcal{M}$, consists of a set of model weights $\mathcal{W}$ and a procedure on how to apply these to a given input $\mathcal{X}$. Viewed in the context of the procedure, the set of potential values of $\mathcal{W}$ can be thought of as a hypothesis space $\mathcal{H}$ on the mapping between $\mathcal{X}$ and $\mathcal{Y}$, regarding a loss function $\mathcal{L}$. Algorithms, like gradient descent, are heuristics for finding optimal / optimized values of $\mathcal{W}$ within $\mathcal{H}$. H itself is not modified by optimization algorithms of this level (i.e. $ax + b$ yields optimal a and b values, but we might need an x^2 term to describe the given phenomena).

Baxter (2011) further extends the notion of generalization and training to a multitask paradigm.

3.1. **Tasks.**

Stated plainly: the task predicts the remainder when dividing a two-digit base-p number by each prime factor q less than p. The set of prime factors we construct tasks for is thus $F = \{f \in \mathbb{P} : f < p)$ For $p = 113$, this yields 29 parallel tasks, one for each prime less than p. Each task predicts a remainder in the range $[0, f - 1]$. This means smaller primes like 2 and 3 require binary and ternary classification, respectively, while the largest prime less than p, 109, requires predictions across 109 classes. The tasks thus naturally vary challenged: predicting $\bmod 2$ requires distinguishing odd from even numbers (which in binary amounts to looking at the last bit), while predicting $\bmod 109$ involves making a selection between many relatively similar classes. From an information-theoretical perspective, the expected cross entropy for an n-class problem is $\ln(n)$, which has implications for the construction of the loss function, further discussed in Section 3.4.

3.2. **Data.**

Input Space ($\boldsymbol{X}$) Each input $x \in X$ represents a number in base p using two digits, (x_0, x_1), where the represented number is $x_0p^0 + x_1p^1$. For example, with $p = 11$, the input space consists of all pairs (x_0, x_1) where $x_0, x_1 < 11$, representing numbers up to $11^2 - 1 = 120$. This yields a dataset of 121 samples. Figure 2 visualizes this input space, with each cell representing the value $x_0p^0 + x_1p^1$.

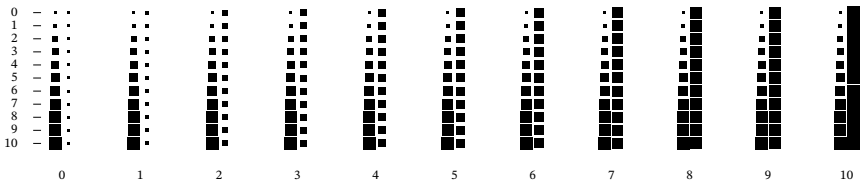

FIGURE 2. Visualization of input space X for $p = 11$. Each cell (x_0, x_1) represents the number $x_0p^0 + x_1p^1$. The top left shows 0 $(0, 0)$, and the bottom right shows 120 $(10, 10)$—both in base-11

Output Space (Y) For each input x, a vector $y \in Y$ contains the remainder when dividing by each prime less than p. For $p = 11$, this means predicting the remainder when dividing by 2, 3, 5, and 7. Each element y_i ranges from 0 to $f_i - 1$ where f_i is the i-th prime. Figure 3 visualizes these remainders, with each subplot showing the remainder pattern for a specific prime divisor. For comparison, the rightmost plot shows the output space of [1]'s modular addition task.

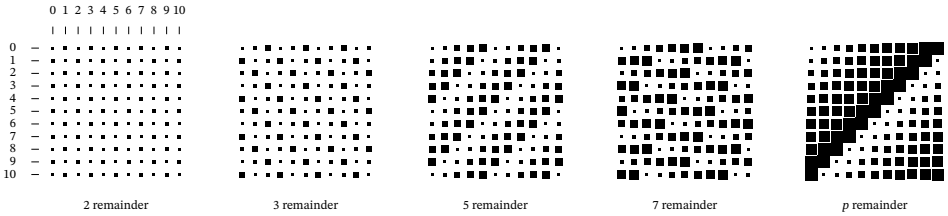

FIGURE 3. Output space Y for $p = 11$. The first four plots show remainders when dividing by 2, 3, 5, and 7 respectively. The rightmost plot shows the output space of the modular addition task for comparison.

3.3. **Model.**

The model follows the original transformer architecture [17] with several key design choices aligned with recent work on mechanistic interpretability [1], [2]: biases are disabled, and layer normalization is not used. The model consists of three main components: an embedding layer, transformer blocks, and an output layer. All weights are initialized following He et al. (2015).

Input tokens are embedded into a d-dimensional space using learned token and positional embeddings:

$$z = \text{TokenEmbed}(x) + \text{PosEmbed}(\text{pos}) \tag{5}$$

Each transformer block comprises multi-head attention:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(Q\frac{K^T}{\sqrt{d_k}}\right)V \tag{6}$$

where Q, K, and V are linear projections of the input. Attention heads are combined through addition rather than concatenation. This is followed by a feed-forward network with ReLU activation:

$$\mathrm{FFN}(x) = \mathrm{ReLU}(xW_{\text{in}})W_{\text{out}} \tag{7}$$

mapping from $d \to 4d \to d$ dimensions. Each component includes residual connections and dropout.

The final representation is projected to predict remainders for each prime factor:

$$\hat{y} = z_{-1}W_{\text{out}} \tag{8}$$

where W_{out} projects to $\sum_{i=1}^{k} f_i$ dimensions for k prime factors, with q_i being the ith prime less than p.

$$\begin{bmatrix} x_0 & x_1 & \hat{y} \end{bmatrix} \tag{9}$$

3.4. **Training.**

Hyper parameter optimization was conducted using Optuna [36], searching over Table 1.

dropout	λ	wd	d	lr	heads
$0, \frac{1}{2}, \frac{1}{5}, \frac{1}{10}$	$0, \frac{1}{2}, 2$	$0, \frac{1}{10}, \frac{1}{2}, 1$	$128, 256$	3e-4, 1e-4	4, 8

TABLE 1. Hyperparameter search space for training.

The model is trained using AdamW [37] with $\beta_1 = 0.9$, $\beta_2 = 0.98$ following [1]. To handle the varying number of classes across tasks (from 2 classes for mod 2 to 109 classes for mod 109), a modified (weighted) mean cross-entropy (`Eq. 4.1`) loss is created, correcting for the difference in expected loss within each task. Note that $\mathbb{E}[L_{\text{MCE}}] = \ln\left(\frac{1}{k}\right)$, where k is the number of classes within the task in question. Correcting for this, the loss function becomes as shown in `Eq. 10.3`.

$$L_{\mathcal{T}_{\text{miiii}}} = -\sum_{q \in Q} \frac{L_{\text{MCE}_f}}{-\ln(f)} \tag{10.1}$$

$$= -\sum_{q \in Q} \frac{\sum_{i=1}^{n} \sum_{j=1}^{f} y_{k_f ij} \ln\left(\hat{y}_{k_f ij}\right)}{-n \ln(f)} \tag{10.2}$$

$$= \sum_{q \in Q} \sum_{i=1}^{n} \sum_{j=1}^{f} \frac{y_{k_f ij} \ln\left(\hat{y}_{k_f ij}\right)}{n \ln(f)} \tag{10.3}$$

To accelerate generalization, gradient filtering as per Lee, J., Kang, B. G., Kim, K., Lee, K. M. [2] is implemented and replicated.

$$g_t = \nabla_\theta L + \lambda(\alpha e_{t-1} + (1-\alpha) g_{t-1}) \tag{11}$$

where e_t is the exponential moving average of gradients with decay rate $\alpha = 0.98$, and $\lambda = 2$ controls the influence of the slow-varying component.

Training uses full batch gradient descent with the entire dataset of p^2 samples (12 769 when $p = 113$). The model is evaluated on a held-out validation set after each epoch, tracking per-task accuracy and loss. As the setup used in $\mathcal{T}_{\text{nanda}}$,

training was done on thirty percent of the total dataset, with the remaining used for validation (1000 samples) and testing (remaining). Further as $\mathcal{T}_{\text{miiii}}$ involves the learning of 29 (when $p = 113$) tasks rather then 1, and due to each task's non-commutativity, a larger hidden dimension of 256 was added to the hyper parameter search space, as well as the potential for 8 heads ($\mathcal{T}_{\text{nanda}}$ was solved with a hidden dimension of 128, and 4 heads). The number of transformer blocks was kept at 1, as this ensures consistency with $\mathcal{T}_{\text{nanda}}$ (and as full generalization was possible, as we shall see in the results).

Training was done on a NVIDIA GeForce RTX 4090 GPU, with Python3.11 and extensive use of "JAX 0.4.35" and it's assocaited ecosystem.

3.5. **Visualization.**

Much of the data worked with here is inherently high dimensional. For training, for example, we have n steps, two splits (train/valid) about $\frac{p}{\ln(p)}$ tasks, and two metrics (accuracy, and loss). This, along with the inherent opaqueness of deep learning models, motivated the developed custom visualization library, `esch`[2] to visualize attention weights, intermediate representations, training metrics, and more. The most important plot type for the reader to keep in mind is seen in TODO ADD. As there are only 12 769 samples when $p = 113$, all samples can be fed at once to the model. Inspecting a specific activation thus yields a $1 \times 12\,796$ vector v, which can be reshaped at a 113×113 matrix, with the two axes varying x_0 and x_1 from 0 to 112, respectively. The top-left corner then shows the given value for the sample $(0 \cdot p^0 + 0 \cdot p^1)$, and so on.

Note that that in `esch` plots, when appropriate, only the top leftmost 37×37 slice is shown, to not overwhelm the reader. Visualizations are available in the Appendix.

3.6. **Mechanistic interpretability.**

A combination of linear products is itself a linear product. As a mechanistic interpretability rule of thumb, one should look at the outputs of the non-linear transformations. In our case, that will be the attention weights and the intermediate representations with each transformer block's MLP (which follows a ReLU activation). Additionally, the embedding layers will be inspected. blah blah.

Our interpretability approach combines visualization techniques with frequency analysis to understand the learned algorithmic patterns. Following [1], we analyze both the attention patterns and the learned representations through several lenses:

3.6.1. *Attention visualization.*

Using `esch`, the custom visualization library, to visualize attention weights and intermediate representations. The library allows for the visualization of attention patterns across different layers, as well as the visualization of intermediate representations at each layer. These visualizations provide insights into the learned patterns and help identify potential areas of improvement.

3.6.2. *The fast Fourier transform.*

As periodicity is established by Nanda et al. (2023) as a fundamental feature of the model trained on $\mathcal{T}_{\text{nanda}}$, the fast Fourier transform (FFT) algorithm is used to

[2] `https://github.com/syrkis/esch`

detect which frequencies are in play. Note that any square image, can be described as a sum of 2d sine and cosine waves varying in frequency from 1 to the size of the image divided by 2 (plus a constant). This is a fundamental tool used in signal processing. The theory is briefly outlined in Appendix B for reference. This analysis helps identify the dominant frequencies in the model's computational patterns.

The default basis of the one-hot encoded representation of the input is thus the identity matrix. This can be projected into a Fourier basis by multiplying with the discrete Fourier transform (DFT) matrix visualized in Appendix C.

4. Results and analysis

4.1. **Hyperparameter Optimization.**

The best-performing hyperparameters for training the model on $\mathcal{T}_{\text{miiii}}$ are listed in Table 2. Notably, the model did not converge when $\lambda = 0$, confirming the utility of the gradient amplification method proposed by Lee et al. (2024) in the context of $\mathcal{T}_{\text{miiii}}$ (see Appendix E).

dropout	λ	wd	d	lr	heads
$\frac{1}{10}$	$\frac{1}{2}$	$\frac{1}{3}$	256	3×10^{-4}	4

Table 2. Reslt of hyperparameter search over $\mathcal{T}_{\text{miiii}}$.

4.2. **Model Performance.**

Figure 4 show the training and validation accuracy on $\mathcal{T}_{\text{miiii}}$ over time. The model achieved perfect accuracy on both the validation set across all 29 tasks. In short —and to use the terminology of Power et al. (2022)—the model "grokked" on all tasks. Interestingly, tasks corresponding to moduli 2, 3, 5, and 7 generalized in succession, while the remaining 25 tasks generalized around epoch 40 000 in no particular order. This might suggests that the model initially learned solutions for the simpler tasks and later developed a more general computational strategy that allowed it to generalize across the remaining, more complex tasks. To understand if this is a form of phase transition facilitated by the reuse of circuitry developed during the initial four tasks, more work is required.

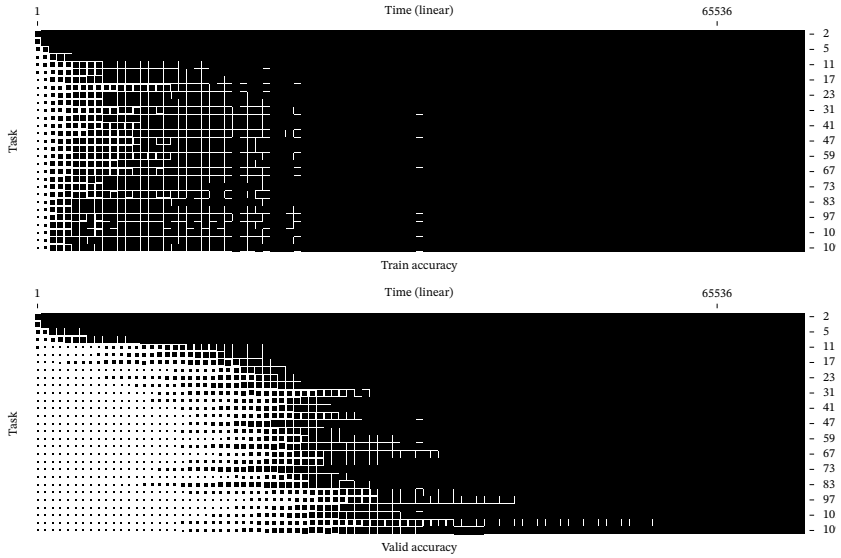

FIGURE 4. Training (top) and validation (bottom) accuracy over time (note the log scale on the x-axis).

As can be seen in Figure 5 when dropout was disabled (i.e., set to zero), the model's performance diverged on the validation set, leading to overfitting, and highlighting the importance of dropout in even with heavy regularization (multi-task, l2, etc.).

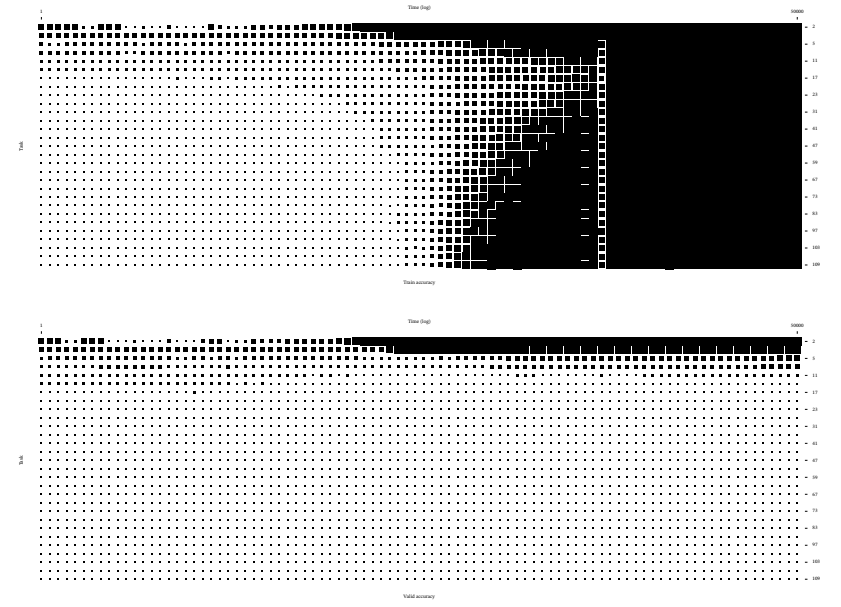

FIGURE 5. Training and validation accuracy with dropout disabled, showing divergence due to overfitting.

4.3. **Analysis of Neuron Activations and Frequencies.**

To understand the internal mechanisms developed by the model, we analyzed the neuron activations after the output weight matrix W_{out} for the model trained on $\mathcal{T}_{\text{miiii}}$. Figure Figure 6 shows that these activations exhibit periodic patterns with respect to (x_0, x_1). This periodicity aligns with the modular arithmetic nature of the tasks, mirrors Nanda et al. (2023) ($\mathcal{T}_{\text{nanda}}$).

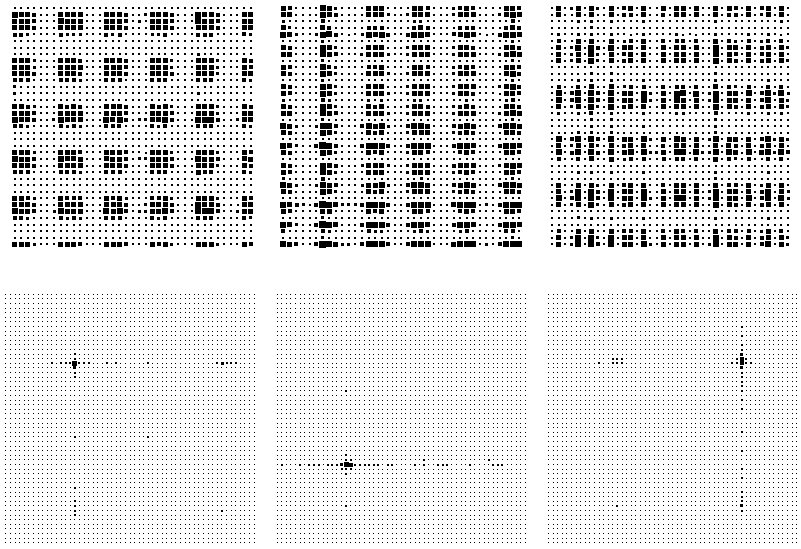

FIGURE 6. Neuron activations after W_{out} for the model trained on $\mathcal{T}_{\text{miiii}}$ (top), with corresponding Fourier transforms below. The activations demonstrate periodicity in (x_0, x_1).

For comparison, Figure Figure 7 shows the neuron activations for a model trained on $\mathcal{T}_{\text{basis}}$. These activations do *not* exhibit periodicity, confirming that the observed periodic patterns in the models trained for $\mathcal{T}_{\text{miiii}}$ and $\mathcal{T}_{\text{nanda}}$ are indeed a result of the moduli operations inherent in the tasks.

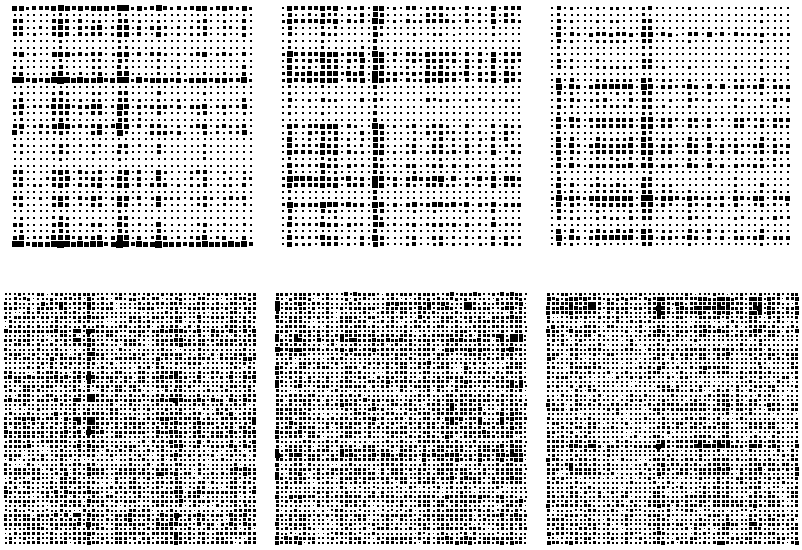

FIGURE 7. Neuron activations after W_{out} for the model trained on $\mathcal{T}_{\text{basis}}$ (top), with corresponding Fourier transforms below. The absence of periodicity contrasts with the activations in Figure Figure 6, emphasizing the influence of the mod operator in $\mathcal{T}_{\text{miiii}}$.

The analysis of active frequencies *through training* using the Fast Fourier Transform (FFT) is illustrated in Figure Figure 8. The top plot shows the different frequencies of the transformer block's MLP neurons evolving as the model learns. The bottom plot displays the variance of frequency activations and the number of frequencies exceeding a significance threshold $\omega > \mu + 2\sigma$ (i.e., which spots like the ones of the bottom row of Figure 6 are active). Initially, a handful of frequencies become dominant as the model generalizes on the first four tasks. As training progresses and the model begins to generalize on the remaining tasks, more frequencies become significant, suggesting that the model is developing more complex internal representations to handle the additional tasks.

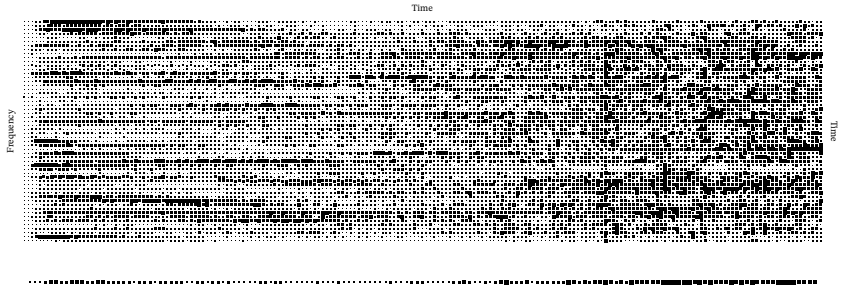

FIGURE 8. Top: Evolution of active frequencies (as per the FFT) of the transformer block neurons during training. Bottom: Variance of frequency activations and the number of frequencies exceeding the threshold $\omega > \mu + 2\sigma$.

However, we observe that significant frequencies appear after generalization has occurred, which may suggest the presence of another phase following grokking in the context of multi-task learning. This could be indicative of circuit merging or the integration of task-specific circuits into a more general solution.

Figure 9 shows the L2 norms of gradients through time for the different weight matrices of the model trained on $\mathcal{T}_{\text{miiii}}$. The gradient norms provide insights into how different parts of the model are being updated during training. Like with Nanda et al. (2023), the attention layer converges quickly, echoing their finding that it does not contribute much to solving their modular eritmetic task.

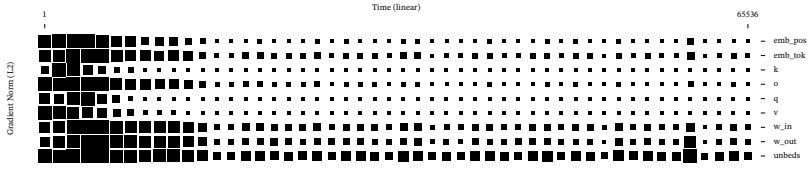

FIGURE 9. L2 norms of gradients over time for the different weight matrices of the model trained on $\mathcal{T}_{\text{miiii}}$.

4.4. **Embeddings.**

Positional embeddings play a crucial role in transformers by encoding the position of tokens in a sequence. Figure Figure 10 compares the positional embeddings of models trained on $\mathcal{T}_{\text{nanda}}$ and $\mathcal{T}_{\text{miiii}}$.

For $\mathcal{T}_{\text{nanda}}$, which involves a commutative task, the positional embeddings are virtually identical, with a Pearson correlation of 0.95, reflecting that the position of input tokens does not significantly alter their contribution to the task. In contrast, for $\mathcal{T}_{\text{miiii}}$, the positional embeddings have a Pearson correlation of -0.64, indicating that the embeddings for the two positions are different. This difference is expected due to the non-commutative nature of the task, where the order of x_0 and x_1

matters ($x_0c \cdot p^0n = x_0c \cdot p^1$). This confirms that the model appropriately encodes position information for solving the tasks.

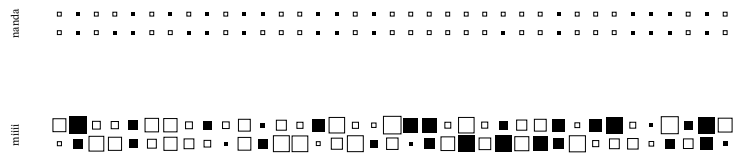

FIGURE 10. Positional embeddings for the network trained on $\mathcal{T}_{\text{nanda}}$ (top) and $\mathcal{T}_{\text{miiii}}$ (bottom).

Recall that a matrix $\mathbf{M}$ of size $m \times n$ can be decomposed to its singular values $\mathbf{M} = \mathbf{U\Sigma V^T}$ (with the transpose being the complex conjugate when $\mathbf{M}$ is complex), where $\mathbf{U}$ is $m \times m$, $\mathbf{\Sigma}$ an $m \times n$ rectangular diagonal matrix (whose diagonal is represented as a flat vector throughout this paper), and $\mathbf{V^T}$ a $n \times n$ matrix. Intuitively, this can be thought of rotating in the input space, then scaling, and then rotating in the output space.

Figure Figure 11 displays the singular values of the token embeddings learned for $\mathcal{T}_{\text{nanda}}$ and $\mathcal{T}_{\text{miiii}}$. The singular values for $\mathcal{T}_{\text{miiii}}$ are more diffuse, indicating that a larger number of components are needed to capture the variance in the embeddings compared to $\mathcal{T}_{\text{nanda}}$. This suggests that the token embeddings for $\mathcal{T}_{\text{miiii}}$ encode more complex information, reflecting the increased complexity of the multi-task learning scenario.

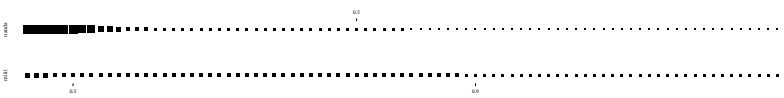

FIGURE 11. First 83 of 113 singular values (truncated for clarity) of U for $\mathcal{T}_{\text{nanda}}$ (top) and $\mathcal{T}_{\text{miiii}}$ (bottom). The ticks indicate the points where 50% and 90% of the variance is accounted for.

Figures Figure 12 and Figure 13 present the most significant singular vectors of U for $\mathcal{T}_{\text{nanda}}$ and $\mathcal{T}_{\text{miiii}}$, respectively. Visual inspection shows periodicity in the top vectors for both models, but the $\mathcal{T}_{\text{miiii}}$ model requires more vectors to capture the same amount of variance, consistent with the diffuse singular values observed.

FIGURE 12. Most significant singular vectors of U for $\mathcal{T}_{\text{nanda}}$.

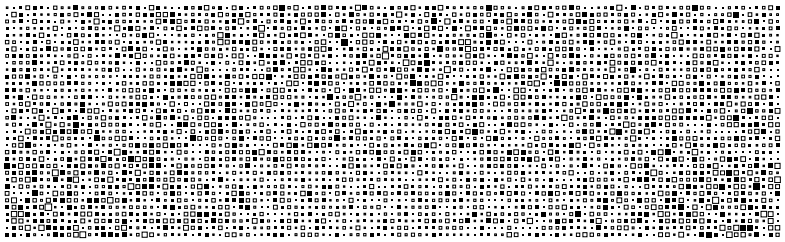

FIGURE 13. Most significant singular vectors of U for $\mathcal{T}_{\text{miiii}}$.

To further understand the structure of the token embeddings, we applied the Fast Fourier Transform (FFT). Figure Figure 14 shows the Fourier spectrum of the token embeddings for the $\mathcal{T}_{\text{nanda}}$ model. Only a few frequencies are particularly active, consistent with the model implementing a cosine-sine lookup table as described in Nanda, N., Chan, L., Lieberum, T., Smith, J., Steinhardt, J. [1].

For the $\mathcal{T}_{\text{miiii}}$ model, we observe a broader spectrum of active frequencies (Figure Figure 15). This is expected due to the model having to represent periodicity corresponding to multiple primes. Comparing with a randomly initialized baseline, the periodicity remains apparent, reinforcing that these patterns result from the learned representations rather than initialization.

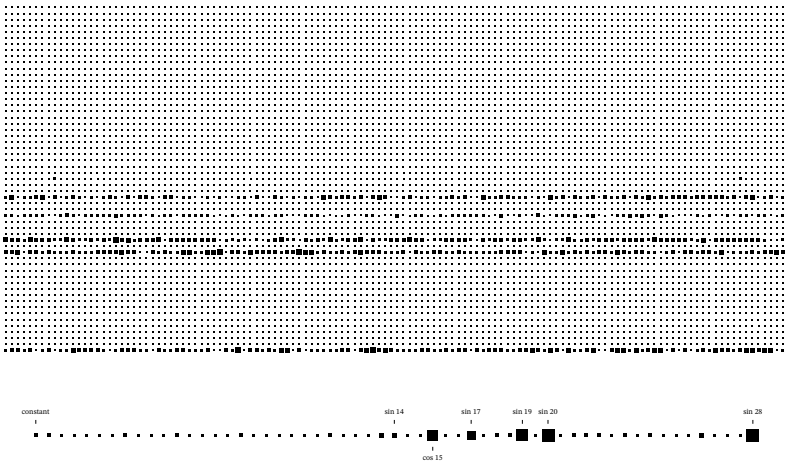

FIGURE 14. Fourier spectrum of the token embeddings $W_{\text{E\_t}}$ for $\mathcal{T}_{\text{nanda}}$, with row norms below.

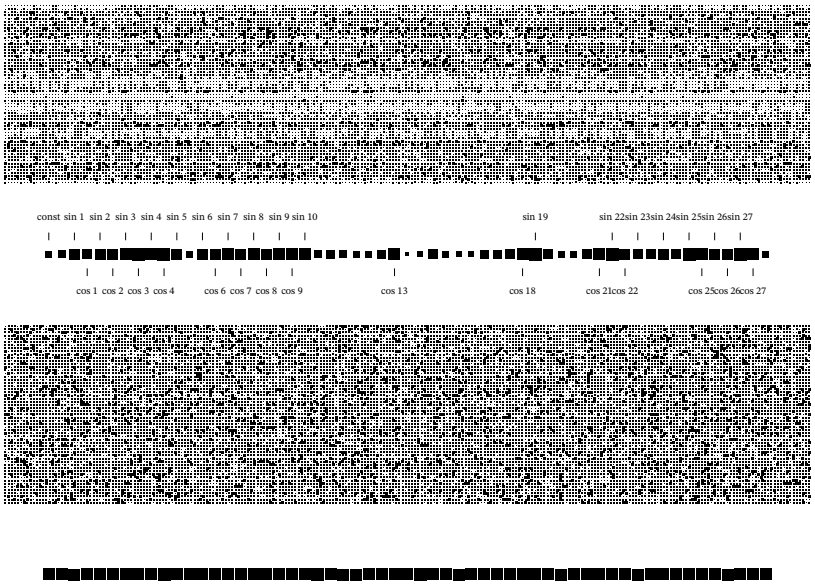

FIGURE 15. Fourier spectrum of the token embeddings $W_{\text{E\_t}}$ for $\mathcal{T}_{\text{miiii}}$ (top two plots) and a random matrix initialized as per [35] of the same shape (bottom two plots), with row norms below each.

These results demonstrate that more frequencies are involved when training on $\mathcal{T}_{\text{miiii}}$ compared to $\mathcal{T}_{\text{nanda}}$. The increased frequency components reflect the need for the model to encode multiple periodic patterns corresponding to the various modular arithmetic tasks.

Combining the anaysis of embeddings and the transformer block neurons, we see that:

1. A lot more frequencies are in play for $\mathcal{T}_{\text{miiii}}$ than in $\mathcal{T}_{\text{nanda}}$.
2. Each individual neuron i still higly reactive to a very small set of frequencies.
3. The reactivity is confirmed to be an artificat of the moduli group by analysis of $\mathcal{T}_{\text{basis}}$

4.5. **Attention Patterns.**

In contrast to the model trained on $\mathcal{T}_{\text{nanda}}$, where attention heads may focus jointly on input tokens due to the commutative nature of the task, our attention heads for the $\mathcal{T}_{\text{miiii}}$ model focus on one digit or the other. This behavior is likely due to the non-commutative nature of the task, where the position of each digit significantly affects the outcome. Figure Figure 16 illustrates this attention pattern.

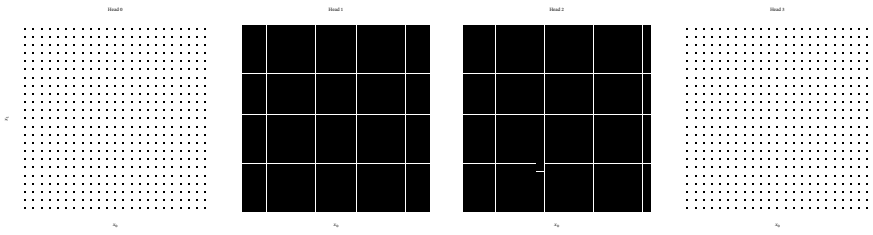

FIGURE 16. Attention from $\hat{y}$ to x_0 for the four attention heads in the $\mathcal{T}_{\text{miiii}}$ model. The attention heads tend to focus on one digit, reflecting the non-commutative nature of the task.

Overall, our results demonstrate that the model effectively learns to solve multiple modular arithmetic tasks by developing internal representations that capture the periodic nature of these tasks. The analysis of embeddings and neuron activations provides insights into how the model generalizes from simpler to more complex tasks, possibly through the reuse and integration of learned circuits. Interestingly, circuits seem to merge only *after* generalization on all tasks, indicating a post grokking phaseshift.

5. Discussion

The evolution of the frequencies ω in the neuron space shown in Figure 8 echoes the generalization phases shown in Figure 4. As the model groks on the primes 2, 3, 5, and 7 in the first fifth of the training run a handful of frequencies become dominant, Nanda, N., Chan, L., Lieberum, T., Smith, J., Steinhardt, J. [1]'s findings. As the model improves training loss with respect to the rest of the tasks, we see a more uniform distribution of active frequencies, with no single frequenci cross the frequently used signal processing threshold of $\mu + 2\omega$.

As the reamining 25 tasks are solved, we do see a significant number of frequencies cross the significance threshold. The gully generalized sollution to all 29 tasks has 9 frequencies above the threshold, while the soultuion to the first four tasks has 4. This is indication of circuit reuse and generality, though more work needs to be done to fully confirm this.

One possible alternative explenation is that the standard $\mu + 2\sigma$ threshold is too high, and that multiple frequencies are present and doing useful work, but there remains too much noise.

Interstingly, the number of active frequencies for the first four tasks is similar to that of a model trained on $\mathcal{T}_{\text{nanda}}$.

However, inspecting when generalization happens, and when significant frequencies appear in the neuron space, we see that frequencies appear AFTER generalization. This could be an indication that there exists another phase after grokking in the case of multi task learning, i.e. task circuit merging. A sort of syzygy of circuits.

A limitation of the work is that inspite of increasing model size and epoch count for $\mathcal{T}_{\text{miiii}}$ compared to that used of Nanda, N., Chan, L., Lieberum, T., Smith, J.,

Steinhardt, J. [1], the late stage circuit merger is only subtley visible (could be solved by increasing epoch count an order of magnitude, but is computationally intractable with the resources used to run the experiemnts of this paper.).

6. Further work

A logical next step would be to further explore post grokking phase transitions, and todo so for bigger models, on tasks where such phase transtions could be expected to exists: i.e. tasks that involve the repeasted use of the same functions, were they to be implemented functionally.

Several promising directions emerge from this work. First, our observation that circuits developed for initial tasks are later reused suggests a novel approach to accelerating learning: by identifying and amplifying these emergent circuits early in training, we might extend Lee et al. (2024)'s gradient-based acceleration techniques. While their method amplifies slow-moving components of the gradient signal uniformly, targeted amplification of specific emerging circuits could provide even greater speedups.

Second, the choice between predicting remainders versus binary divisibility presents an interesting trade-off. While remainder prediction requires more output neurons, it might provide richer gradient signals during training. A systematic comparison of these approaches could yield practical insights for training mathematical neural networks.

Lastly, this work raises questions about the relationship between circuit reuse and task difficulty. Are simpler tasks always learned first, and do their circuits necessarily form building blocks for more complex tasks? More specifically: how related to a simpler task a does a more complex task b for circuits to reusable between the two. Understanding these relationships could inform better training strategies for multi-task learning more broadly.

The code associated with this paper is available as a PyPI package (`pip install miiii`) to facilitate exploration of these questions (as well as replication of the findings at hand).

7. Conclusion

This paper presents several key findings about how neural networks learn multiple related mathematical tasks. First, we observed a striking pattern in the model's learning trajectory: after independently solving four fundamental tasks (remainders modulo 2, 3, 5, and 7), the model rapidly generalized to 25 additional tasks. This suggests the emergence of a general computational strategy, rather than task-specific solutions. Our ablation studies, where masking the initial four tasks prevented generalization within feasible training time, further support this hierarchical learning pattern.

Second, our analysis of the model's internal representations revealed that multi-task learning promotes the development of more robust and general algorithmic solutions. The periodic patterns observed in the model's embeddings and attention mechanisms, while more complex than those found in single-task modular arithmetic, demonstrate how the model learns to handle multiple periodic relationships simultaneously.

These findings contribute to our understanding of mechanistic interpretability in two ways: they demonstrate how multi-task learning can guide networks toward more general solutions, and they show how the complexity of these solutions can be systematically analyzed even as the number of tasks increases. Future work might explore how these insights could be applied to accelerate learning in other domains where multiple related tasks must be solved simultaneously.

References

1. Nanda, N., Chan, L., Lieberum, T., Smith, J., Steinhardt, J.: Progress Measures for Grokking via Mechanistic Interpretability, (2023)
2. Lee, J., Kang, B. G., Kim, K., Lee, K. M.: Grokfast: Accelerated Grokking by Amplifying Slow Gradients, (2024)
3. Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R.,ˇZídek, A., Potapenko, A., Bridgland, A., Meyer, C., Kohl, S. A. A., Ballard, A. J., Cowie, A., Romera-Paredes, B., Nikolov, S., Jain, R., Adler, J., Back, T., Petersen, S., Reiman, D., Clancy, E., Zielinski, M., Steinegger, M., Pacholska, M., Berghammer, T., Bodenstein, S., Silver, D., Vinyals, O., Senior, A. W., Kavukcuoglu, K., Kohli, P., Hassabis, D.: Highly Accurate Protein Structure Prediction with AlphaFold. Nature. 596, 583–589 (2021). https://doi.org/10.1038/s41586-021-03819-2
4. Dinan, E., Farina, G., Flaherty, C., Fried, D., Goff, A., Gray, J., Hu, H., Jacob, A. P., Komeili, M., Konath, K., Kwon, M., Lerer, A., Lewis, M., Miller, A. H., Mitts, S., Renduchintala, A., Roller, S., Rowe, D., Shi, W., Spisak, J., Wei, A., Wu, D., Zhang, H., Zijlstra, M.: Human-Level Play in the Game of \emph{Diplomacy} by Combining Language Models with Strategic Reasoning. Science. 378, 1067–1074 (2022). https://doi.org/10.1126/science.ade9097
5. Radford, A., Narasimhan, K.: Improving Language Understanding by Generative Pre-Training. Presented at the (2018)
6. Shannon, C. E.: Programming a Computer for Playing Chess. Computer Chess Compendium. 2–13 (1950). https://doi.org/10.1007/978-1-4757-1968-0_1
7. Tesauro, G.: TD-Gammon, A Self-Teaching Backgammon Program, Achieves Master-Level Play. (1993)
8. Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., Hassabis, D.: Mastering the Game of Go without Human Knowledge. Nature. 550, 354–359 (2017). https://doi.org/10.1038/nature24270
9. Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S.: A Survey of Monte Carlo Tree Search Methods. IEEE Transactions on Computational Intelligence and AI in Games. 4, 1–43 (2012). https://doi.org/10.1109/TCIAIG.2012.2186810
10. Ruoss, A., Delétang, G., Medapati, S., Grau-Moya, J., Wenliang, L. K., Catt, E., Reid, J., Genewein, T.: Grandmaster-Level Chess Without Search, (2024)

11. Lipton, Z. C.: The Mythos of Model Interpretability: In Machine Learning, the Concept of Interpretability Is Both Important and Slippery. Queue. 16, 31–57 (2018). https://doi.org/10.1145/3236386.3241340
12. Cybenko, G.: Approximation by Superpositions of a Sigmoidal Function. Mathematics of Control, Signals, and Systems. 2, 303–314 (1989). https://doi.org/10.1007/BF02551274
13. Ba, J. L., Kiros, J. R., Hinton, G. E.: Layer Normalization, (2016)
14. Krizhevsky, A., Sutskever, I., Hinton, G. E.: ImageNet Classification with Deep Convolutional Neural Networks. Communications of the ACM. 60, 84–90 (2017). https://doi.org/10.1145/3065386
15. Krogh, A., Hertz, J.: A Simple Weight Decay Can Improve Generalization. In: Advances in Neural Information Processing Systems. Morgan-Kaufmann (1991)
16. Baxter, J.: A Model of Inductive Bias Learning, (2011)
17. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., Polosukhin, I.: Attention Is All You Need, (2017)
18. Power, A., Burda, Y., Edwards, H., Babuschkin, I., Misra, V.: Grokking: Generalization Beyond Overfitting on Small Algorithmic Datasets, (2022)
19. Humayun, A. I., Balestriero, R., Baraniuk, R.: Deep Networks Always Grok and Here Is Why, (2024)
20. Wang, B., Yue, X., Su, Y., Sun, H.: Grokked Transformers Are Implicit Reasoners: A Mechanistic Journey to the Edge of Generalization, (2024)
21. Conmy, A., Mavor-Parker, A. N., Lynch, A., Heimersheim, S., Garriga-Alonso, A.: Towards Automated Circuit Discovery for Mechanistic Interpretability, (2023)
22. Weiss, G., Goldberg, Y., Yahav, E.: Thinking Like Transformers, (2021)
23. Yu, S., Sanchez Giraldo, L., Principe, J.: Information-Theoretic Methods in Deep Neural Networks: Recent Advances and Emerging Opportunities. In: Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence. pp. 4669–4678. International Joint Conferences on Artificial Intelligence Organization, Montreal, Canada (2021)
24. Bronstein, M. M., Bruna, J., Cohen, T., Veličković, P.: Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges, (2021)
25. Gavranović, B., Lessard, P., Dudzik, A., von Glehn, T., Araújo, J. G. M., Veličković, P.: Categorical Deep Learning: An Algebraic Theory of Architectures, (2024)
26. Maurer, A.: Bounds for Linear Multi-Task Learning.
27. Shannon, C. E.: A Mathematical Theory of Communication. ACM SIGMOBILE Mobile Computing and Communications Review. 5, 3–55 (2001). https://doi.org/10.1145/584091.584093
28. Cover, T. M., Thomas, J. A.: Elements of Information Theory. Wiley-Interscience, Hoboken, N.J (2006)
29. Jeon, H. J., Van Roy, B.: An Information-Theoretic Framework for Deep Learning. Advances in Neural Information Processing Systems. 35, 3279–3291 (2022)

30. Egri, L., Shultz, T. R.: A Compositional Neural-network Solution to Prime-number Testing. (2006)
31. Lee, S., Kim, S.: Exploring Prime Number Classification: Achieving High Recall Rate and Rapid Convergence with Sparse Encoding, (2024)
32. Wu, D., Yang, J., Ahsan, M. U., Wang, K.: Classification of Integers Based on Residue Classes via Modern Deep Learning Algorithms, (2023)
33. He, B., Hofmann, T.: Simplifying Transformer Blocks, (2023)
34. Hosseini, M., Hosseini, P.: You Need to Pay Better Attention, (2024)
35. He, K., Zhang, X., Ren, S., Sun, J.: Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification, (2015)
36. Akiba, T., Sano, S., Yanase, T., Ohta, T., Koyama, M.: Optuna: A Next-generation Hyperparameter Optimization Framework. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. pp. 2623–2631. Association for Computing Machinery, New York, NY, USA (2019)
37. Loshchilov, I., Hutter, F.: Decoupled Weight Decay Regularization, (2019)

Appendix

A Training plots

AA **Training loss.**

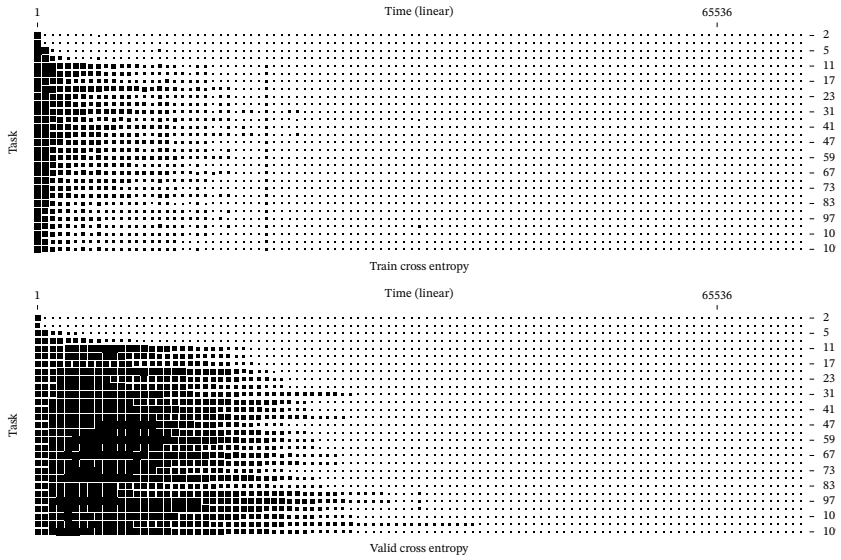

Figure 17. Representation of training and validation loss (x-axis is in log scale).

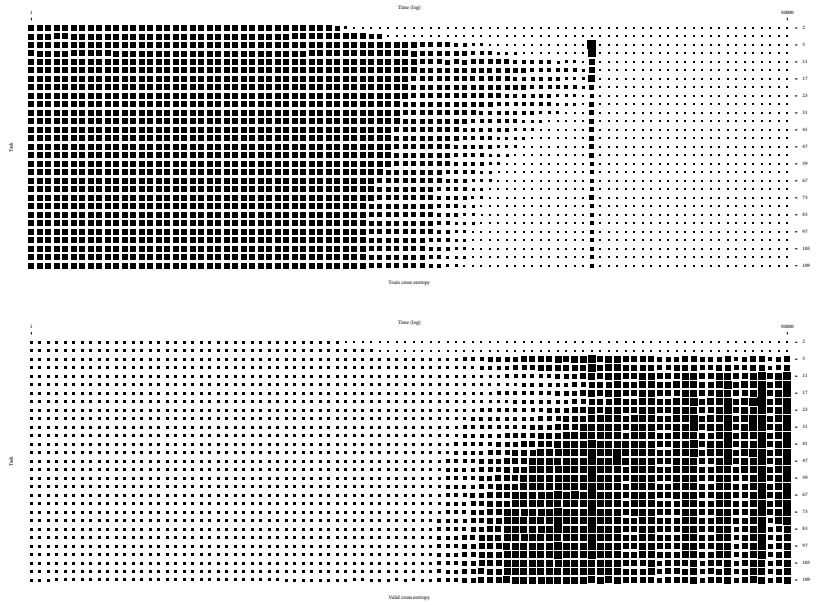

Figure 18. Loss for run with dropout disabled

B Fast Fourier Transform

The inner product between two vectors $\boldsymbol{v}$ and $\boldsymbol{u}$ of length n can be written as per Eq. 12.

$$\sum_{i}^{n} \boldsymbol{v}[i]\boldsymbol{u}[i] \tag{12}$$

We can extend the meaning of inner products to functions f and g over the interval $[a; b]$ with Eq. 13.

$$\int_{a}^{b} f(x)g(x)dx \tag{13}$$

It is a fact that any continuous, differentiable function $f(x)$, can be written as a sum of cosine and sine terms plus a constant as per:

$$f(x) = \frac{A_0}{2} + \sum_{k=1}^{\infty}(A_k \cos(kx) + B_k \sin(kx)) \tag{14}$$

Where A_k and B_k are the normalized inner products $\langle f(x), \cos(kx)\rangle$ and $\langle f(x), \sin(kx)\rangle$ respectively[3]. These are explicitly written out in Eq. 15.

$$A_k = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\cos(kx)dk, \quad B_k = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\sin(kx)dk \tag{15}$$

This can be similarly extended for that grid, which is the basis for the two-dimensional FFT.

[3]Note the pointy brackets denote inner product

C Discrete Fourier transform (DFT) matrix

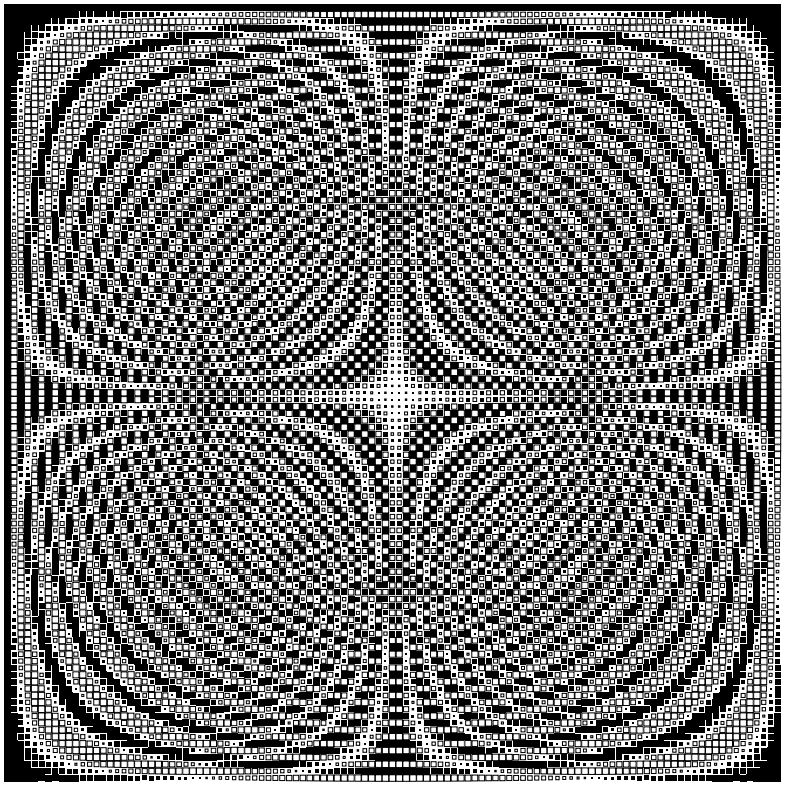

Figure 19. DFT matrix

D Sub-symbolic implementation of $f(x, y)$

Compute $f(x)$ for $\{(a, b) \in \mathbb{N}^2 : 0 \leq a, b < 113)$, by adding the two rows of $W_{E_{\text{pos}}}$ in Figure 20 to a one-hot encoded a and b, and then multiplying by $W_{E_{\text{tok}}}$. Then multiply by W_k, W_q and W_v indecently in performing the operation described in Eq. 6, and then add to the output of the embedding operations. Send that through the feed-forward network with the weights in Figure 22. The reader is asked to confirm visually that the weight in the figures indeed computes $f(x, y) = \cos(ax) + \sin(bx)$ when applied in the order described above.

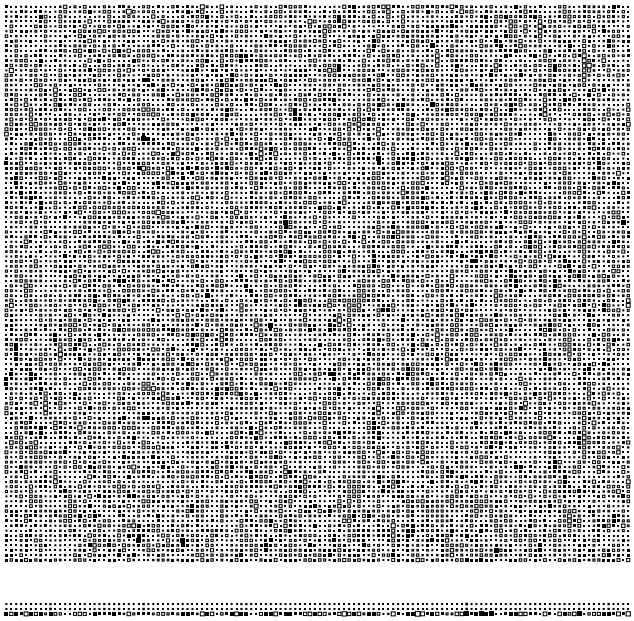

Figure 20. $W_{E_{\text{tok}}}$ and $W_{E_{\text{pos}}}$

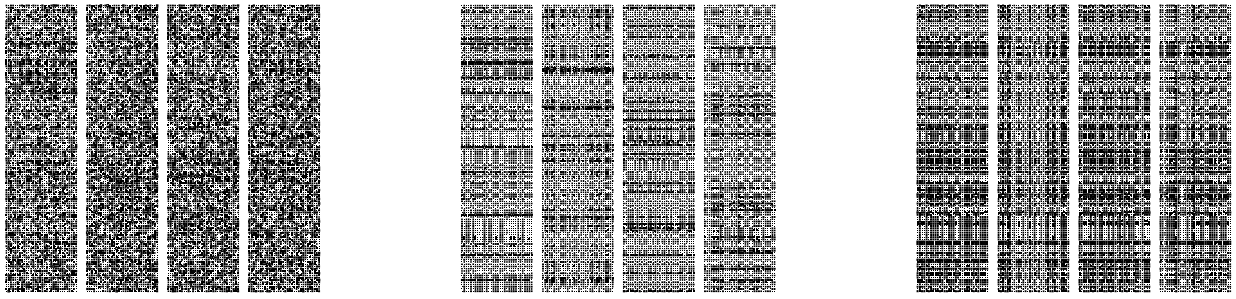

Figure 21. W_k, W_q and W_v

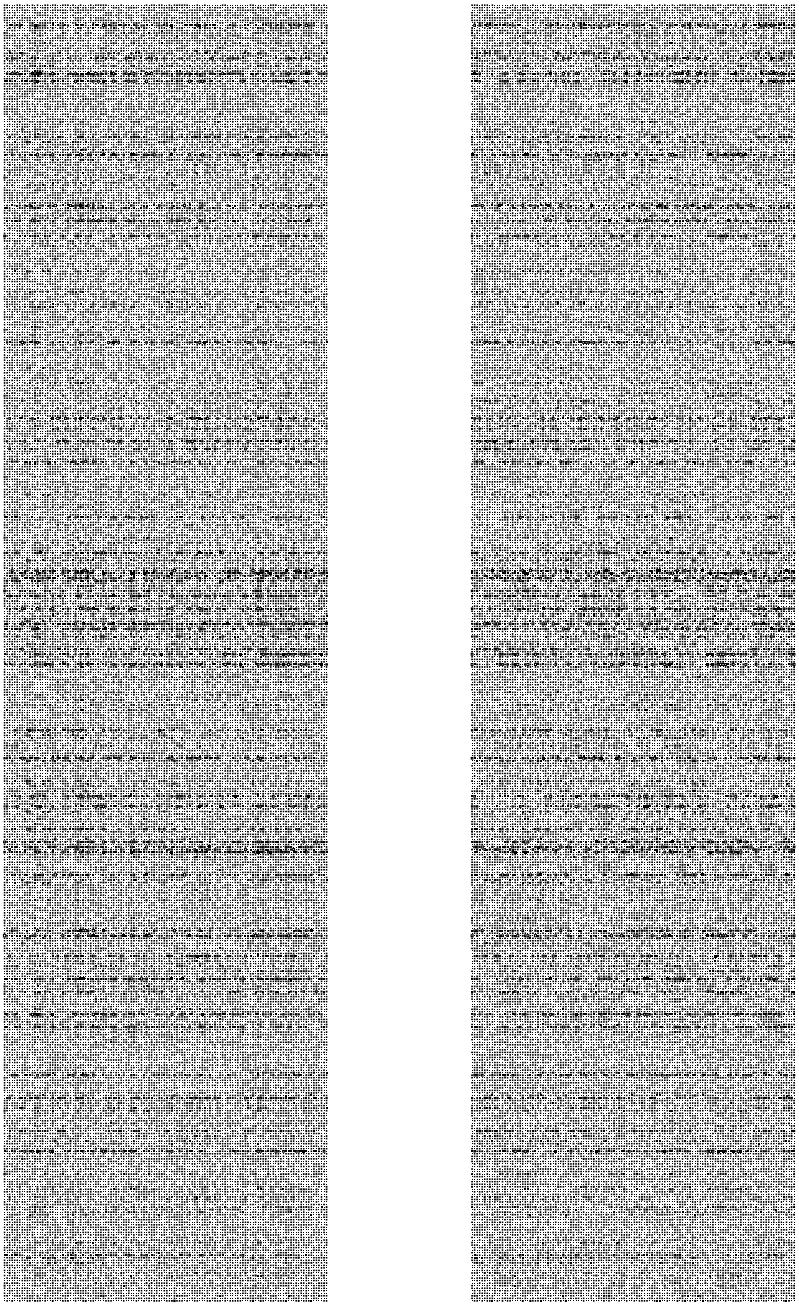

FIGURE 22. W_{in} and W_{out}^T

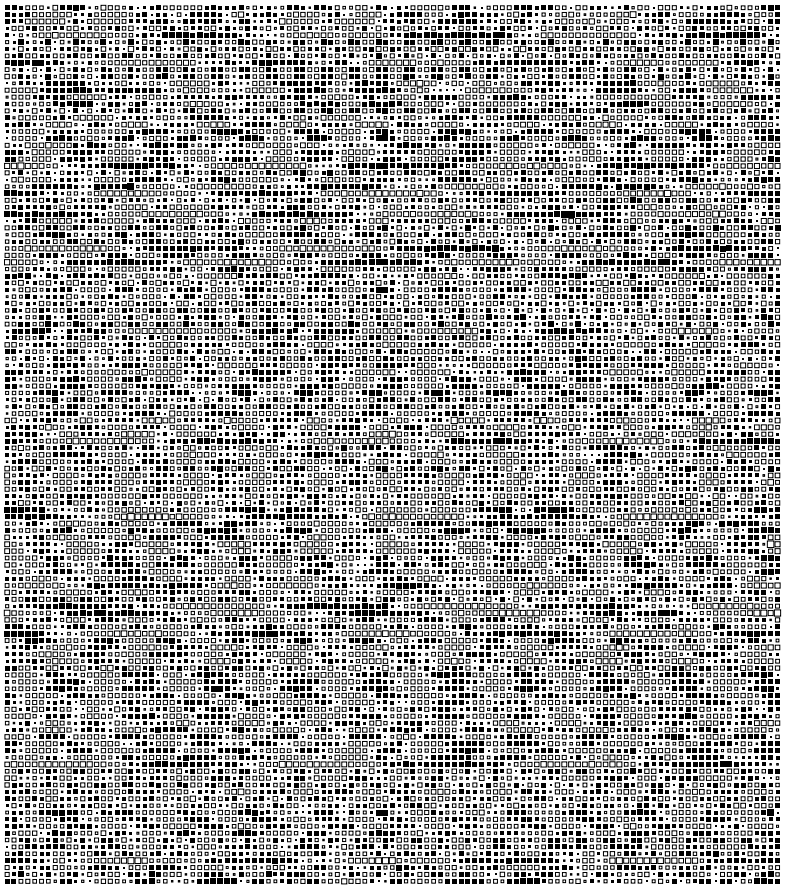

FIGURE 23. W_U

E Training curves

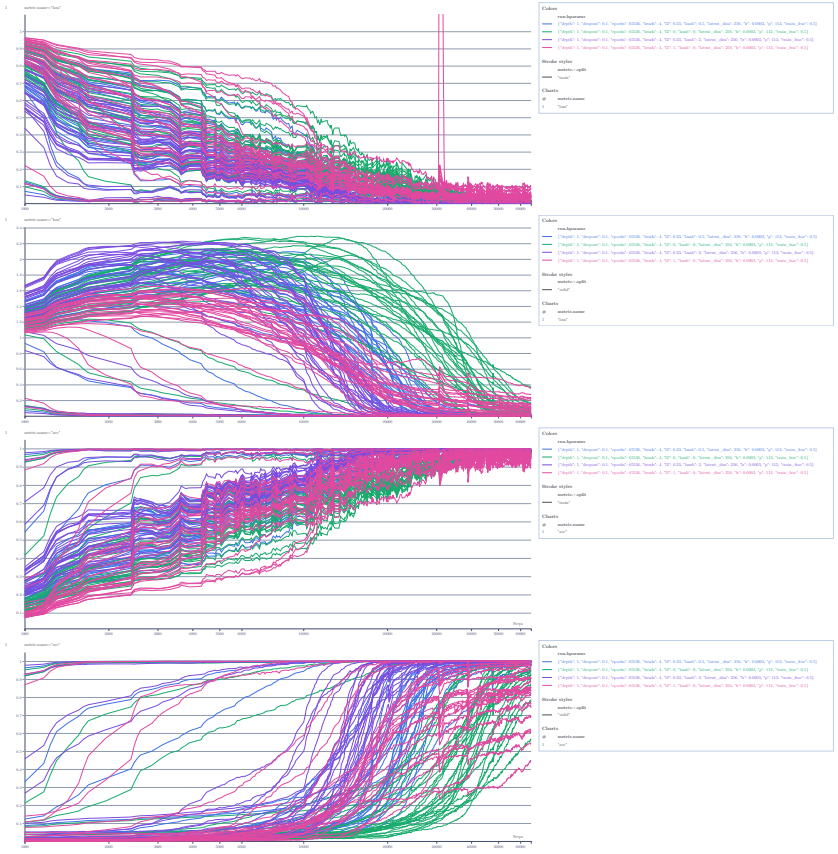

FIGURE 24. Training curves. Green and red have $\lambda = 0$, and do not converge in training time, vindicating [2]. This is most obivous validation accuracy plot (bottom).