Emergent Symbol-like Number Variables in Artificial Neural Networks

Anonymous authors
Paper under double-blind review

Abstract

Symbolic programs, defined by discrete variables with explicit rules and relations, often have the benefit of interpretability, ease of communication, and generalization. This is contrasted against neural systems, consisting of distributed representations with rules and relations defined by learned parameters, which often have opaque inner mechanisms. There is an interest in finding unity between these two types of systems for cognitive and computer scientists alike. There is no guarantee, however, that these two types of systems are reconcilable. To what degree do neural networks induce abstract, mutable, slot-like variables in order to achieve next-token prediction (NTP) goals? Can neural functions be thought of analogously to a computer program? In this work, we train neural systems using NTP on numeric cognitive tasks and then seek to understand them at the level of symbolic programs. We use a combination of causal interventions and visualization methods in pursuit of this goal. We find that models of sufficient dimensionality do indeed develop strong analogs of symbolic algorithms purely from the NTP objective. We then ask how variations on the tasks and model architectures affect the models' learned solutions to find that numeric symbols are not formed for every variant of the task, and transformers solve the problem in a different fashion than their recurrent counterparts. Lastly, we show that in all cases, some degree of gradience exists in the neural symbols, highlighting the difficulty of finding simple, interpretable symbolic stories of how neural networks perform their tasks. Taken together, our results are consistent with the view that neural networks can approximate interpretable symbolic programs of number cognition, but the particular program they approximate and the extent to which they approximate it can vary widely, depending on the network architecture, training data, extent of training, and network size.

1 Introduction

Both biological and artificial Neural Networks (NNs) have powerful modeling abilities. We can see this in biological NNs from the impressive capabilities of human cognition, and in artificial NNs (ANNs) from the recent advances that have had such great success that they have been crowned the "gold standard" in many machine learning communities (Alzubaidi et al., 2021). The inner workings of NNs, however, remain largely opaque to humans, in part due to their representations being highly distributed. Individual neurons can play multiple roles within a network (Rumelhart et al., 1986; McClelland et al., 1986; Smolensky, 1988; Olah et al., 2017; 2020; Elhage et al., 2022; Scherlis et al., 2023; Olah, 2023).

Symbolic algorithms/programs, in contrast, defined as processes that manipulate distinct, typed entities according to explicit rules and relations, can have the benefit of consistency, transparency, and generalization when compared to their neural counterparts. A concrete example of a symbolic algorithm is a computer program, where the variables are abstract, mutable entities, able to represent many different values, and these variables are processed by well defined functions. Human designed symbolic systems, however, can lack the expressivity and performance of NNs. This is apparent in the field of natural language processing where neural architectures trained on vast amounts of data (Vaswani et al., 2017; Brown et al., 2020; Kaplan et al., 2020) have swept the field, surpassing the pre-existing symbolic approaches. Can we find symbolic interpretations of these powerful neural systems to grant us the benefits of symbolic programs in neural systems? Furthermore, there are

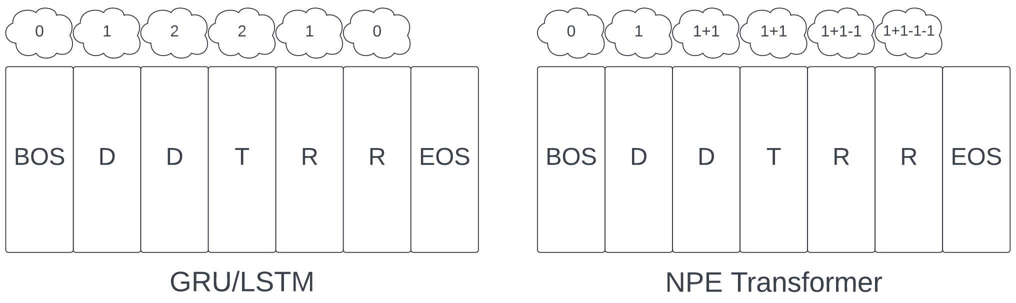

Figure 1: Visual depiction of different architecture's solutions achieving the same accuracy on the same numeric equivalence task. The rectangles represent token types for a task in which the T token indicates that the model must produce the same number of Rs as it witnessed Ds, and must end the sequence with an EOS (see Methods for more details). The thought bubbles represent causally discovered symbolic variable values encoded within subspaces of the models' representations. The recurrent models encode a mutable count that increments up and then back down to indicate the end of the task. The transformers learn a solution in which they recompute the necessary information to solve the task from the context at each step in the sequence. In the panel, we show the specific solution used by transformers with no positional encodings (NPE), where they assign opposite numeric values to the D and R tokens and then recompute their sum at each step in the sequence, knowing to stop when the difference equals 0.

many existing theories that posit the necessity of algorithmic, symbolic, processing for higher level cognition (Do & Hasselmo, 2021; Fodor & Pylyshyn, 1988; Fodor, 1975; 1987; Newell, 1980; 1982; Pylyshyn, 1980; Marcus, 2018; Lake et al., 2017). While the aforementioned successes of neural systems may call such cognitive claims into question, it might be argued that neural systems actually implement such symbolic algorithms, or they may approximate them well enough that seeking to find the most aligned symbolic algorithm could be a powerful step toward making explicit and accessible that which seems to be implicit and opaque in NNs.

This approach of seeking to characterize the performance of a NN-based system in terms of the most aligned symbolic algorithm has been applied within Large Language Models (LLMs) in and outside of in-context learning (ICL) (Geiger et al., 2023; Feng & Steinhardt, 2024) and in simpler models that can perform copy operations on Gaussian vectors (Geiger et al., 2022). In both of these cases, however, the variables of interest were explicit in the data, taking the form of directly specified attributes such as color or shape, or labels in the training data. Can NNs invent representations of symbolic variables that would be useful for solving a task? Can these representations emerge purely as a function of a next-token prediction (NTP) objective applied to valid examples of correct task action sequences?

To address these questions we turn to the domain of number cognition and focus on the numeric equivalence tasks used to test the numeric abilities of members of an Amazonian culture whose language lacks explicit number words (Gordon, 2004). In these tasks, human participants were asked to output the same quantity of items that were provided by an experimenter in a demonstration phase at the beginning of each instance of the task. Numbers are interesting in that the entities we enumerate using numbers can be of many different kinds, as long as they are discretely enumerable. Furthermore, the numeric equivalence tasks from Gordon (2004) allow for experimental procedures in which the neural network models have no explicit numeric instruction, exploring whether they can invent abstract, symbol-like representations as a tool to predict the next token. Can ANNs solve such a task? And if so, what do their representations and computations look like? Does the system employ different computations (or use disparate pathways) for different inputs? Or, does the network share the same computations across different values of an abstract variable, of the kind we might allocate to a distinct storage location in a computer program? Numeric reasoning also has the advantage of being well studied in humans of different ages and with different numeric experience, providing a powerful domain for comparisons between biological and artificial cognitive systems (Di Nuovo & Jay, 2019).

Beyond the intrigue of finding a deeper understanding of abstract, neural reasoning and of exploring connections between biological and artificial systems, numbers remain an important aspect of modern artificial systems. At the time of writing this paper, the best AI systems have still not achieved 100% accuracy on GSM8k (Cobbe et al., 2021; Zhong et al., 2024), despite the word problems being only grade-school level—although we recognize that this may change very soon. Outside of mathematical benchmarks, numbers might naturally arise in many autoregressive modeling tasks such as tracking parentheses and punctuation, numeric symbol processing, tracking objects (in both text and vision), and many spatial reasoning tasks (also in both text and vision).

In this work, we pursue these interests by training recurrent and attention based ANNs on next token prediction tasks inspired by one of the numerical equivalence tasks from Gordon (2004). Our contributions are as follows:

1. We causally demonstrate alignment of recurrent neural representations with a symbolic counting program that increments and decrements a count variable represented analogously to a memory storage location in the network's internal state.
2. We demonstrate that these symbol-like variables that represent latent, task-relevant information can emerge purely from a next-token prediction objective.
3. We show that transformer based architectures solve the task by referencing and recomputing information from the context at each time step without using the abstract equivalent of a designated storage location in memory.
4. We show that the emergence of the symbol-like numeric variables in recurrent models can be influenced by task details that are unrelated to the underlying numeric principles.
5. We show that these symbol-like numeric variables are somewhat graded, with greater interchangeability between smaller numbers and between numbers that have a smaller difference in magnitude.
6. Lastly, we show an effect of model size, where models of minimal size induce these symbolic variables with more gradience, while larger models induce them more precisely.

We use these results to inform our thinking on what types of representations are learnable from NTP, to encourage use of multiple causal interpretability tools for any representational analysis, to highlight functional differences that might emerge from architectural constraints like Markovian states vs attention based structures, and to highlight the varying degrees of gradience in neural representations—adding to discussions on mechanistic interpretability.

2 Background/Related Work

2.1 Causal Interpretability

In this work, we wish to highlight the importance of using causal manipulations for interpreting neural functions as opposed to using mere statistical associations. Causal inference broadly refers to methods that isolate the particular effects of individual components within a larger system (Pearl, 2010). An abundance of causal interpretability variants exist that have been used to determine what functions are being performed by the models' activations (or circuits) (Olah et al., 2018; 2020; Wang et al., 2022; Geva et al., 2023; Merrill et al., 2023; Bhaskar et al., 2024; Wu et al., 2024). Vig et al. (2020) is a recent review that provides an integrative review of the rationale for and utility of causal mediation in neural model analyses.

In our work, we rely heavily on a particular causal analysis method known as Distributed Alignment Search (DAS) (Geiger et al., 2021; 2023). This method can be thought of as a specific type of activation patching (also referred to as causal tracing). Activation patching is a method that substitutes a portion of the model activations under a given input into the activations under a different input (Meng et al., 2023; Vig et al., 2020). DAS mainly differs in that it uses a learned rotation matrix to target a specific subspace for the substitution. A closely related technique is Attribution Patching (Nanda, 2022) and, more recently, AtP* (Kramár et al., 2024) which first uses an attribution method on the model activations with respect to some metric and then uses this attribution to find an appropriate patch.

2.2 Models of Symbolic and Numeric Reasoning

A closely related recent work to ours is Csordás et al. (2024) who examine RNNs in a recall task. Another related work is Geiger et al. (2022) who show evidence of ANNs learning symbolic processes from gradient descent on a copying task. Other works have shown that LLMs can store references to specific information in cases of in-context-learning (Feng & Steinhardt, 2024). Further explorations on the idea of interchangable variables in ANNs exist in Elhage et al. (2022).

Many publications that ANNs' abilities to perform counting tasks (Di Nuovo & McClelland, 2019; Fang et al., 2018; Sabathiel et al., 2020; Kondapaneni & Perona, 2020; Nasr et al., 2019; Zhang et al., 2018; Trott et al., 2018). Our tasks and modeling paradigms differ from these publications in that numbers are only latent in the structure of our tasks without explicit teaching of distinct symbols for distinct numeric values, and our tasks allow for open-ended responses from the models.

3 Methods

In this work, we train models on numeric equivalence tasks and then use causal interpretability methods such as Distributed Alignment Search (DAS) (Geiger et al., 2021; 2023) to understand the manner in which the models solve the task.

3.1 Numeric Equivalence Tasks

We examine three variants of a numeric equivalence task in this work. Each task is defined by variable length sequences of tokens. Each sequence starts with a Beginning of Sequence (BOS) token and ends with an End of Sequence (EOS) token. The contents of each sequence are determined by first uniformly sampling a target quantity from the inclusive range of 1 to 20. The sequence is then constructed as the combination of two phases. The first phase, called the demonstration phase (**demo phase**), starts with the BOS token and continues with a series of demo tokens equal in quantity to the sampled target quantity. Following the demo tokens is the Trigger token (T), indicating the end of the demo phase and the beginning of the response phase (**resp phase**). The resp phase starts with a series of resp tokens equal in number to target quantity and therefor the number of demo phase tokens. The EOS token follows the resp tokens, denoting the end of the sequence.

During the initial model training, we include all token types in the autoregressive loss. During model evaluation and DAS trainings, we only consider tokens in the resp phase—which are fully determined by the demo phase. Also during model trainings, we hold out target quantities of 4, 9, 14, and 17. A trial is considered correct when all resp tokens and the EOS token are correctly predicted by the model following the trigger. See the supplement for more training details. We include three variants of this task differing only in their demo and resp token types.

Multi-Object Task: there are 3 demo token types $\{D_1, D_2, D_3\}$ with a single response token type, R. The demo tokens are uniformly sampled from the 3 possible token types. An example sequence with a target quantity of 2 could be: "BOS D_3 D_1 T R R EOS"

Single-Object Task: there is a single demo token type, D, and a single response token type, R. An example with a target quantity of 2 is: "BOS D D T R R EOS"

Same-Object Task: there is a single token type, C, used by both the demo and resp phases. An example with a target quantity of 2 would be: "BOS C C T C C EOS".

3.2 Model Architectures

The recurrent models in this paper consist of Gated Recurrent Units (GRUs) (Cho et al., 2014), and Long Short-Term Memory networks (LSTMs) (Hochreiter & Schmidhuber, 1997). These architectures both have a Markovian, hidden state vector that bottlenecks all predictive computations following the structure:

$$h_{t+1} = f(h_t, x_t) \tag{1}$$

$$\hat{x}_{t+1} = g(h_{t+1}) \tag{2}$$

Where h_t is the hidden state vector at step t, x_t is the input token at step t, f is the recurrent function (either a GRU or LSTM cell), and g is a multi-layer perceptron (MLP) used to make a prediction, denoted $\hat{x}_{t+1}$, of the token at step $t+1$. We contrast the recurrent architectures against transformer architectures (Vaswani et al., 2017; Touvron et al., 2023; Su et al., 2023) in that the transformers use a history of input tokens, $X_t = [x_1, x_2, ..., x_t]$, at each time step, t, to make a prediction:

$$\hat{x}_{t+1} = f(X_t) \tag{3}$$

Where f now represents the transformer architecture. We show results from 2 layer, single attention head transformers that use RoPE positional encodings (Su et al., 2023). Refer to Supplement A.4 and Figure 5 for more model and architectural details. Except for in the training curves in Figure 3, we first train the models to >99.99% accuracy on their respective tasks before performing analyses. The models are evaluated on 15 sampled sequences of each of the 16 trained and 4 held out target quantities. We train 6 model seeds for each training condition. Model seeds that failed to achieve this standard were dropped from the analyses, including 3 model seeds from the LSTM models in the Same-Object task and one seed from the transformer models in each of the Single-Object and Same-Object tasks.

3.3 Symbolic Programs

In this work, we examine the alignment of 3 different symbolic programs to the models' distributed representations.

1. **Up-Down Program:** uses a single numeric variable, called the **Count**, to track the difference between the number of demo tokens and resp tokens at each step in the sequence. It also contains a **Phase** variable to determine whether it is in the demo or resp phase. The program ends when the Count is equal to 0 during the resp phase.
2. **Up-Up Program:** uses two numeric variables—the **Demo Count** and **Resp Count**—in addition to a Phase variable to track quantity at each step in the sequence. This program increments the Demo Count during the demo phase and increments the Resp Count during the resp phase. It ends when the Demo Count is equal to the Resp Count during the resp phase.
3. **Context Distributed (Ctx-Distr) Program:** queries a history of inputs at each step in the sequence to determine when to stop rather than encoding a cumulative quantity variable. A more specific version of this program (that appears to emerge under some conditions) is is one in which the program assigns a value of 1 to each demo token and a -1 to each resp token (or *visa-versa*) and computes their combined sum at each step in the sequence to determine the count. This program knows to stop when the sum is 0.

We include Algorithms 1- 2 in the supplement representing the pseudocode used to implement the Up-Down and Up-Up programs in simulations. Refer to Figure 1 for an illustration of the Up-Down strategy and a specific instance of the Ctx-Distr strategy observed in some transformers.

It is important to note that there are an infinite number of equivalent implementations of these programs. For example, the Up-Down program could immediately add and subtract 1 from the Count at every step in addition to carrying out the rest of the program as previously described. We do not discriminate between programs that are causally indistinct from one another.

3.4 Distributed Alignment Search (DAS)

The majority of our model analyses use DAS to find alignments between symbolic programs and the neural network representations. DAS is a hypothesis testing framework for finding alignments between distributed systems such as ANNs, and symbolic programs/algorithms (also referred to as causal abstractions) Geiger et al. (2021; 2023) by performing interchange interventions (equivalently referred to as causal interventions).

For all DAS experiments, we freeze the model weights when performing the analysis. In general, DAS measures the degree of alignment between the representations at a specific processing layer of a distributed model and a hypothesized, symbolic program. In our RNN experiments, we focus on the recurrent hidden state, and in the transformer experiments, we focus on the hidden states between the first and second transformer layers. See Figure 5 for more detail into the selected activation layer. For a given variable from the symbolic program, DAS learns an orthogonal rotation matrix,

$\mathcal{R} \in R^{m \times m}$, that is used to find the subspace of the distributed representation that is most aligned with the specified symbolic variable.

Concretely, we uniformly sample a time point from two separate sequences. These time points are t and u. We then run the model independently on the sequences until time point t and u respectively. We then take the latent representations from the prespecified processing layer from the model from the two separate sequences respectively. We refer to these representations as the original and source vectors, $h_t^o \in R^m$ and $h_u^s \in R^m$ (referred to as base and source in previous work), where m is the number of neurons in each distributed representation. We then rotate h_t^o and h_u^s using $\mathcal{R}$ into r_t^o and r_u^s respectively, and then we replace a pre-specified number of dimensions in r_t^o with the same dimensions from r_u^s. Lastly we apply the inverse of the rotation to r_u^s resulting in a new vector, denoted h_t^v. This can be written formally as:

$$h_t^v = \mathcal{R}^{-1}(D\mathcal{R}h_t^o + (1 - D)\mathcal{R}h_u^s) \tag{4}$$

Where $D \in R^{m \times m}$ is a diagonal, binary matrix used to isolate the desired set of dimensions to replace. The binary values of D can be learned or pre-specified. In this work, we pre-specify the number of dimensions to be half of m. It is only important to specify the number of non-zero dimensions in D—rather than the indices of the non-zero dimensions—due to the ability of the orthogonal matrix to equivalently learn each basis in any row order. As such, we default the non-zero dimensions to be contiguous along the diagonal and do not explore non-contiguous configurations. Finally, we allow the model to continue making token predictions from point t in the original sequence starting with h_t^v. We discard h_u^s.

For a given symbolic program, we know what the outputs should be under the assumption that the DAS interchange intervention was successful. If we make the assumption that the model is performing the task in the same way as the symbolic program, we can use the true outputs of the symbolic program as the target tokens for an NTP objective. This allows us to train the orthogonal rotation matrix—backpropagating the NTP signal through the frozen model weights into $\mathcal{R}$. We train the rotation matrix to convergence.

Once we have a trained rotation matrix, we can evaluate the quality of the alignment using the accuracy of the model's predictions on the hypothesized target outputs. This accuracy has been referred to as the Interchange Intervention Accuracy (IIA) in previous work (Geiger et al., 2023). Higher IIAs indicate stronger alignments between the model and the symbolic program.

For the LSTM architecture, we perform DAS on a concatenation of the h and c recurrent state vectors. Unless otherwise stated, we use 10000 intervention samples for training and 1000 samples for validation and testing. We uniformly sample target quantities and intervention time points, t and u, for both the original and source sequences in the training, validation, and testing sets. We orthogonalize the rotation matrix using PyTorch's orthogonal parameterization with default settings. We train the rotation matrix for 1000, with a batch size of 512, selecting the checkpoint with the best validation performance for analysis. We use a learning rate of 0.003 and an Adam optimizer.

3.5 Additional Interventions

The DAS rotation matrices were unnecessary to perform causal interventions for the Ctx-Distr solution. Instead, a sufficient experiment to demonstrate the lack of use of a cumulative count variable is to look for unchanged behavior after performing a full activation vector intervention on relevant hidden representations—in which we replace a full activation vector at time step t under one set of inputs with the full activation vector at time step u under a different set of inputs. We provide further detail in Supplement A.5 as to why this experiment is sufficient for the claim. We apply these full activation interventions on the recurrent hidden states in the RNNs just after the recurrent processing, and unless otherwise stated, we apply these interventions to the hidden states from Layer 1 in the transformer architectures (as labeled in Figure 5).

More specifically, we pause the network's processing at a non-terminal step t, and then we replace its hidden state at step t with a representation from a different sequence also at a non-terminal step u of the same phase as t. If the model is using the Ctx-Distr program, we would expect the models' subsequent token predictions to be unaffected by this intervention. Thus the IIAs shown in Figure 2 for the Ctx-Distr intervention indicate that, as expected, performance is disrupted by this intervention in the RNNs but not in the transformers.

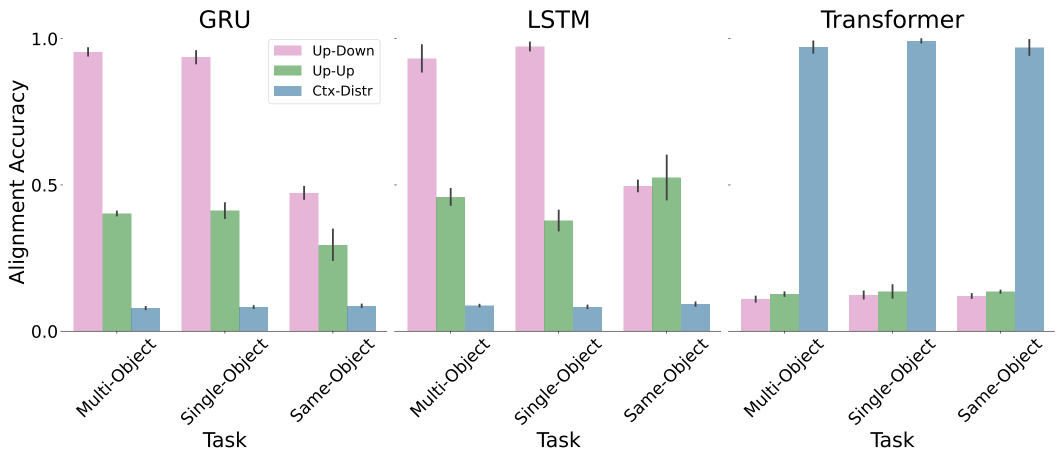

Figure 2: Interchange intervention accuracy (IIA) on variables from different symbolic programs for different tasks faceted by architecture type. The displayed IIA in the Up-Down program is taken from the Count variable. The IIA in the Up-Up program is the better performing of the two possible count variables. See the methods for more details into the IIA for the Ctx-Distr program. Within each architecture type, bars are arranged horizontally by task. All IIA measurements show the proportion of trials in which the model predicts all R and EOS tokens correctly after a causal intervention.

4 Results

4.1 Causal Abstractions

We first turn our attention to Figure 2 where we can see DAS performance as a function of the causal abstraction used in the alignment. In the recurrent models (GRUs and LSTMs), we see that the most aligned causal abstraction is the Up-Down program. The results are compared against the Up-Up program and the Ctx-Distr program which have significantly lower, albeit non-zero IIAs. We use this as evidence in favor of the interpretation that the recurrent models develop a count up, count down strategy to track quantities within the task.

To determine how the transformer architectures were performing the task, we first looked at the attention weights for both of the two transformer layers (see Figures 8 and 5). The transformers with positional encodings gave surprisingly little attention to the resp tokens when producing resp and EOS tokens. This pattern of attention is supportive of the idea that they are not encoding a cumulative state variable of the count within each time step. As predicted, swapping two non-terminal hidden states within the same phase did not appreciably change the position of the models' EOS token predictions. We can see the results of these interventions in Figure 2.

The low attention weight placed on the resp tokens is consistent with the possibility that the transformers have using the corresponding positional information of the EOS token relative to the trigger token's position. An alternative distributed solution, however, could be one in which the transformer assigns a positive numeric value to the demo tokens and a negative numeric value to the resp tokens and performs a summing operation at each step in the sequence, knowing to stop when it has reached zero. We include a treatment of a minimalist transformer without positional encodings in Supplement A.4 to support the notion that this is the transformers' solution in the absence of positional encodings.

4.2 Model Dimensionality

We can see from Figure 3 that although many model sizes can solve the Multi-Object task, increasing the number of dimensions in the hidden states of the GRUs improves IIA in alignments with the Up-Down program. We can also see in Figure 4 that the larger models tend to have less graded alignments. This is perhaps to be expected from the stronger IIA.

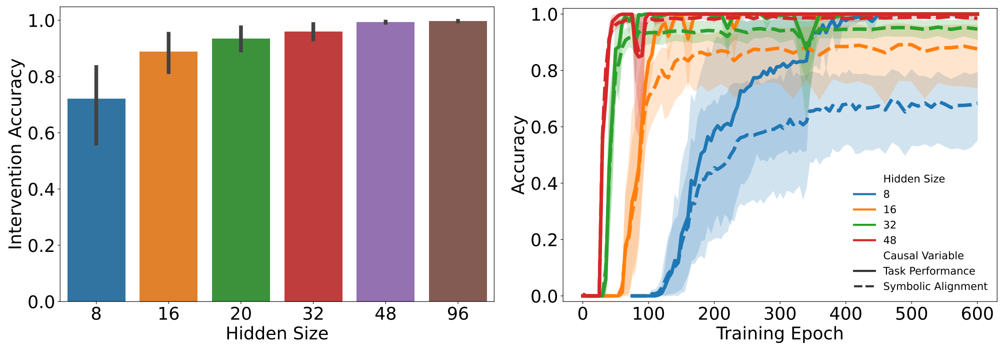

Figure 3: Left: Interchange intervention accuracy (IIA) on the Up-Down program in the GRU models on the Multi-Object task as a function of increasing hidden state dimensionality (model size). We see that the mean of the IIA increases as the number of dimensions increases while the variability decreases with increasing model size. Right: Both task accuracy and IIA (symbolic alignment) on held out data over the course of training for different model sizes (also GRUs on the Multi-Object task). We see a correlation between the first epoch with non-zero in both the IIA and task accuracy; the first non-zero epoch occurs more consistently, earlier with greater model size; and we see greater variability in the IIA with smaller model sizes.

4.3 Task

An interesting result is the impact of demonstration token type on the resulting alignment of the recurrent models with the Up-Down program. We can see from Figure 2 that recurrent models trained on the Same-Object task, in which the model responds with the same token type as it initially counts, have poor alignment with any of the proposed symbolic programs. Although we do not know exactly why the Same-Object task causes this poor alignment compared to both the Single-Object and Multi-Object tasks, we use this result as a baseline to highlight the unified, interchangeable numeric representations found in the Multi-Object and Single-Object tasks.

4.4 Symbolic Gradience

We now shift towards a more nuanced perspective of symbolic alignments with neural systems, where we highlight the graded nature of the symbols within the neural representations. We can see from Figure 4 that the GRU models trained on the Multi-Object task have worse IIA when the quantities involved in the intervention are larger, and when the intervention quantities have a larger difference in magnitude. We point out that the task training data forces the models to have more experience with smaller numbers, as they necessarily interact with smaller numbers every time they interact with larger numbers. This is perhaps a causal factor for the more graded representations at larger numbers. The DAS training data suffers from a similar issue, where we use a uniform sampling of the target quantities that define the training sequences and then we uniformly sample the intervention indices from these sequences. This results in a disproportionately large number of training interventions containing smaller values.

5 Discussion/Conclusion

In this work we used causal interpretability methods to demonstrate the existence of symbol-like variables within NN solutions to numeric equivalence tasks. We showed that these neural variables emerged purely from an NTP objective and represent abstract information that is only latent in the task structure. This serves as a proof of principle for the types of representations and solutions that can emerge from simple learning objectives such as NTP. These findings are also a proof of principle that neural systems do not need explicit exposure to discrete numeric symbols for symbol-like representations of number to emerge. Nor do neural systems need built-in counting principles to inform their numeric learning. These findings have potential relevance to the types of representations that might be formed from NTP on large scale internet training data, and they have potential relevance for the types of learning objectives that could be used in the human brain.

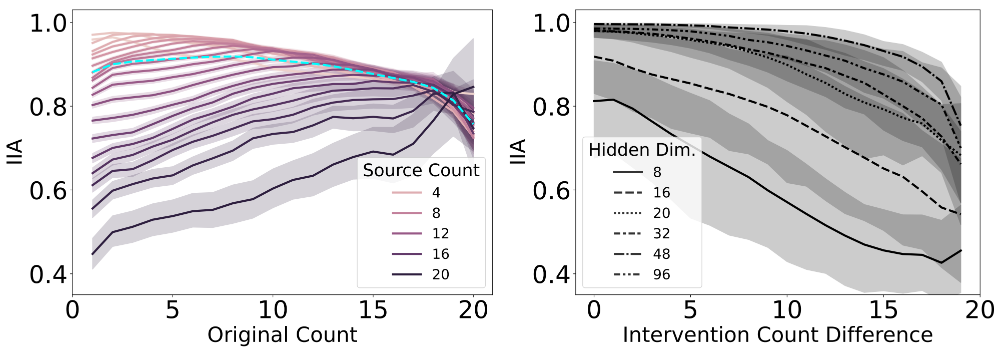

Figure 4: *Left*: DAS IIA where the x-axis shows the existing count in the original sequence before the interchange intervention. The colors denote the counts that replace the original count after the intervention. The data is from GRUs trained on the Multi-Object task of all reported sizes that are listed in in the right panel. The cyan, dashed line represents the mean IIA, highlighting the greater number of samples for the lower number interventions. *Right:* DAS IIA where the x-axis shows the absolute difference in magnitude between the original, pre-existing count and the source count (the count used to replace the original). The different dashes indicate different model sizes. We can see from both panels that the contents of the interventions affect the IIA in a relatively smooth fashion.

We also demonstrated differences in the high-level solutions used by different model architectures to solve the task. Namely, we showed that increasing the dimensionality of the recurrent models improved their symbolic alignment, and we showed that transformers solved the tasks by recomputing the relevant information at each step in the sequence—contrasted against the cumulative count variables discovered in the recurrent models. An interesting phenomenon in the LLM literature is the effect of model scale on performance (Brown et al., 2020; Kaplan et al., 2020). Although our scaling results are for GRUs on toy tasks, they are provocative for understanding why size might improve autoregressive results. Perhaps increased dimensionality allows the models to find more symbol-like, disentangled solutions when solving their next-token prediction tasks. In future work we hope to explore the possibility that this result can be explained by the lottery ticket hypothesis (Frankle & Carbin, 2019) combined with lazy learning dynamics (Jacot et al., 2020). Perhaps the majority of what these models learn are linear functions of their initial features, and increasing the dimensionality of the model increases the number of potential pathways/features that the model can use to solve the task.

We are unsure if the "stateless", time-distributed solution exhibited by the transformers generalizes beyond the counting tasks presented in this work. It is possible that this finding is representative of a more general principle: that transformers avoid solutions that use cumulative, Markovian state variables. We provide an analysis in Supplement A.4 of a one-layer transformer without positional encodings trained on a variant of the Single-Object task without a BOS token, and without a T token. We experimentally and mathematically support the idea that this minimal model solves the task by assigning opposite numeric values to the demo and resp tokens and averaging their values at each step in the sequence. We include additional transformer experiments in the supplement using variants of each of the numeric equivalence tasks that include Void tokens in the demo phase that should not be included when determining the target count (see Supplement A.6). As before, we use IIA to show that these transformers are not relying on a mutable count variable. The results are consistent with the possibility that these models have learned the same program as the transformers trained without positional embeddings. In ongoing work we will pursue further analyses to test this possibility further. We find it worth noting that the Ctx-Distr solution exhibited by the transformers lends itself to the type of solutions that might be predicted by RASP-L (Zhou et al., 2023).

An interesting result can be found in the learning trajectories displayed in Figure 3. We can see from the performance curves that both the models' task performance and alignment performance begin a transition away from 0% at similar epochs and plateau at similar epochs depending on model size. This result can be contrasted with a hypothetical result in which the alignment curves significantly lag behind the task performance of the models. In this hypothetical case, a possible interpretation could have been that the network first develops unique solutions for many different input-output pairs and progressively unifies them. The pattern we observe instead raises the possibility that the network is using a unified strategy throughout, improving by gradually refining its representations so that they are more and more accurate at representing exact quantity. Perhaps our result is to be expected in light of works like Saxe et al. (2019) and Saxe et al. (2022) which show an inherent tendency for NNs trained via gradient descent to find solutions that share network pathways.

Lastly, we demonstrated that the symbol-like, neural subspaces illuminated by DAS are not always perfectly symbolic, often exhibiting a smooth, graded influence from the content of the variables being intervened upon. We interpret these results as a reminder that representations in distributed systems exist on a continuum despite seemingly discrete, symbolic performance on tasks. These results have an analogy to children's number cognition in which children may appear to possess a symbol-like understanding of exact numbers and their associated principles, but when probed deeper, the symbol-like picture falls apart (Wynn, 1992; Davidson et al., 2012). Perhaps the graded nature of the neural representations reinforces the utility of thinking about network solutions as trajectories in a dynamical system, where the values along a set of dimensions are analogous to the values of high-level, causal variables. We use these findings as a reminder that although NNs may discover approximations to interpretable, symbol-like solutions, their representations are still ultimately graded, adding nuance to the effort to find in them exact implementations of any symbolic computer program.

5.1 Limitations and Future Directions

A limitation of our work is the small-scale of the models we use. On the one hand, the use of such small scale models can be a powerful tool, revealing details of mechanisms that would be much harder to fully understand in the massive language models used in performant AI systems (Chan et al., 2022; Reddy). On the other hand, such work leaves open the possibility that quite different mechanisms may be in play in larger systems. In future we hope to explore extending the kinds of analyses used here to probing larger models, recognizing that computations might be more distributed and therefore more difficult to intervene on in such systems.

Another limitation is the minimal nature of the tasks we have used. Once again the simplicity is in some ways a virtue, allowing initial success that future work can build upon. In such work, we plan to consider larger, more complex tasks. We also plan to do an in-depth analysis on why there is a relationship between model size and DAS performance, ideally providing an additional theoretical understanding.

References

Laith Alzubaidi, Jinglan Zhang, Amjad J Humaidi, Ayad Al-Dujaili, Ye Duan, Omran Al-Shamma, J Santamaría, Mohammed A Fadhel, Muthana Al-Amidie, and Laith Farhan. Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. *Journal of Big Data*, 8(1):53, 2021. ISSN 2196-1115. doi: 10.1186/s40537-021-00444-8. URL `https://doi.org/10.1186/s40537-021-00444-8`.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016. URL `https://arxiv.org/abs/1607.06450`.

Adithya Bhaskar, Dan Friedman, and Danqi Chen. The heuristic core: Understanding subnetwork generalization in pretrained language models, 2024. URL `https://arxiv.org/abs/2403.03942`.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

Stephanie Chan, Adam Santoro, Andrew Lampinen, Jane Wang, Aaditya Singh, Pierre Richemond, James McClelland, and Felix Hill. Data distributional properties drive emergent in-context learning in transformers. *Advances in Neural Information Processing Systems*, 35:18878–18891, 2022.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014. URL `http://arxiv.org/abs/1406.1078`.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL `https://arxiv.org/abs/2110.14168`.

Róbert Csordás, Christopher Potts, Christopher D. Manning, and Atticus Geiger. Recurrent neural networks learn to store and generate sequences using non-linear representations, 2024. URL `https://arxiv.org/abs/2408.10920`.

Kathryn Davidson, Kortney Eng, and David Barner. Does learning to count involve a semantic induction? *Cognition*, 123(1):162–173, 2012. ISSN 0010-0277. doi: https://doi.org/10.1016/j.cognition.2011.12.013.

Alessandro Di Nuovo and Tim Jay. Development of numerical cognition in children and artificial systems: a review of the current knowledge and proposals for multi-disciplinary research. *Cognitive Computation and Systems*, 1(1):2–11, 2019. doi: https://doi.org/10.1049/ccs.2018.0004. URL `https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/ccs.2018.0004`.

Alessandro Di Nuovo and James L. McClelland. Developing the knowledge of number digits in a child-like robot. *Nature Machine Intelligence*, 1(12):594–605, 2019. ISSN 2522-5839. doi: 10.1038/s42256-019-0123-3. URL `http://dx.doi.org/10.1038/s42256-019-0123-3`.

Quan Do and Michael E. Hasselmo. Neural Circuits and Symbolic Processing. *Neurobiology of learning and memory*, 186:107552, December 2021. ISSN 1074-7427. doi: 10.1016/j.nlm.2021.107552. URL `https://www.ncbi.nlm.nih.gov/pmc/articles/PMC10121157/`.

Nelson Elhage, Tristan Hume, Catherine Olsson, Nicholas Schiefer, Tom Henighan, Shauna Kravec, Zac Hatfield-Dodds, Robert Lasenby, Dawn Drain, Carol Chen, Roger Grosse, Sam McCandlish, Jared Kaplan, Dario Amodei, Martin Wattenberg, and Christopher Olah. Toy models of superposition. *Transformer Circuits Thread*, 2022. https://transformer-circuits.pub/2022/toy_model/index.html.

M. Fang, Z. Zhou, S. Chen, and J. L. McClelland. Can a recurrent neural network learn to count things? *Proceedings of the 40th Annual Conference of the Cognitive Science Society*, pp. 360–365, 2018.

Jiahai Feng and Jacob Steinhardt. How do language models bind entities in context?, 2024. URL `https://arxiv.org/abs/2310.17191`.

Jerry A. Fodor. *The Language of Thought*. Harvard University Press, 1975. ISBN 978-0-674-51030-2. Google-Books-ID: XZwGLBYLbg4C.

Jerry A. Fodor. *Psychosemantics: The Problem of Meaning in the Philosophy of Mind*. MIT Press, 1987.

Jerry A. Fodor and Zenon W. Pylyshyn. Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28(1):3–71, March 1988. ISSN 0010-0277. doi: 10.1016/0010-0277(88)90031-5. URL `https://www.sciencedirect.com/science/article/pii/0010027788900315`.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks, 2019. URL `https://arxiv.org/abs/1803.03635`.

Atticus Geiger, Hanson Lu, Thomas Icard, and Christopher Potts. Causal abstractions of neural networks. *CoRR*, abs/2106.02997, 2021. URL `https://arxiv.org/abs/2106.02997`.

Atticus Geiger, Alexandra Carstensen, Michael C. Frank, and Christopher Potts. Relational reasoning and generalization using non-symbolic neural networks, 2022. URL https://arxiv.org/abs/2006.07968.

Atticus Geiger, Zhengxuan Wu, Christopher Potts, Thomas Icard, and Noah D. Goodman. Finding alignments between interpretable causal variables and distributed neural representations, 2023.

Mor Geva, Jasmijn Bastings, Katja Filippova, and Amir Globerson. Dissecting recall of factual associations in auto-regressive language models, 2023. URL https://arxiv.org/abs/2304.14767.

Peter Gordon. Numerical cognition without words: Evidence from Amazonia. *Science*, 306(5695): 496–499, 2004. ISSN 00368075. doi: 10.1126/science.1094492.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8): 1735–1780, nov 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL https://doi.org/10.1162/neco.1997.9.8.1735.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks, 2020. URL https://arxiv.org/abs/1806.07572.

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

Neehar Kondapaneni and Pietro Perona. A Number Sense as an Emergent Property of the Manipulating Brain. *arXiv*, pp. 1–23, 2020. URL http://arxiv.org/abs/2012.04132.

János Kramár, Tom Lieberum, Rohin Shah, and Neel Nanda. Atp*: An efficient and scalable method for localizing llm behaviour to components, 2024. URL https://arxiv.org/abs/2403.00745.

Brenden M. Lake, Tomer D. Ullman, Joshua B. Tenenbaum, and Samuel J. Gershman. Building machines that learn and think like people. *Behavioral and Brain Sciences*, 40:e253, January 2017. ISSN 0140-525X, 1469-1825. doi: 10.1017/S0140525X16001837. URL https://www.cambridge.org/core/journals/behavioral-and-brain-sciences/article/building-machines-that-learn-and-think-like-people/A9535B1D745A0377E16C590E14B94993.

Gary Marcus. Deep learning: A critical appraisal, 2018. URL https://arxiv.org/abs/1801.00631.

J. L. McClelland, D. E. Rumelhart, and PDP Research Group (eds.). *Parallel Distributed Processing. Volume 2: Psychological and Biological Models*. MIT Press, Cambridge, MA, 1986.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in gpt, 2023. URL https://arxiv.org/abs/2202.05262.

William Merrill, Nikolaos Tsilivis, and Aman Shukla. A tale of two circuits: Grokking as competition of sparse and dense subnetworks, 2023. URL https://arxiv.org/abs/2303.11873.

Neel Nanda. Attribution patching: Activation patching at industrial scale. https://www.neelnanda.io/mechanistic-interpretability/attribution-patching, 2022.

Khaled Nasr, Pooja Viswanathan, and Andreas Nieder. Number detectors spontaneously emerge in a deep neural network designed for visual object recognition. *Science Advances*, 5(5):1–11, 2019. ISSN 23752548. doi: 10.1126/sciadv.aav7903.

Allen Newell. Physical symbol systems. *Cognitive Science*, 4(2):135–183, April 1980. ISSN 0364-0213. doi: 10.1016/S0364-0213(80)80015-2. URL https://www.sciencedirect.com/science/article/pii/S0364021380800152.

Allen Newell. The knowledge level. *Artificial Intelligence*, 18(1):87–127, January 1982. ISSN 0004-3702. doi: 10.1016/0004-3702(82)90012-1. URL `https://www.sciencedirect.com/science/article/pii/0004370282900121`.

Chris Olah. Distributed representations: Composition superposition. `https://transformer-circuits.pub/2023/superposition-composition`, 2023.

Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. *Distill*, 2017. doi: 10.23915/distill.00007. https://distill.pub/2017/feature-visualization.

Chris Olah, Arvind Satyanarayan, Ian Johnson, Shan Carter, Ludwig Schubert, Katherine Ye, and Alexander Mordvintsev. The building blocks of interpretability. *Distill*, 2018. doi: 10.23915/distill.00010. https://distill.pub/2018/building-blocks.

Chris Olah, Nick Cammarata, Ludwig Schubert, Gabriel Goh, Michael Petrov, and Shan Carter. Zoom in: An introduction to circuits. *Distill*, 2020. doi: 10.23915/distill.00024.001. https://distill.pub/2020/circuits/zoom-in.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. *CoRR*, abs/1912.01703, 2019. URL `http://arxiv.org/abs/1912.01703`.

Judea Pearl. An Introduction to Causal Inference. *The International Journal of Biostatistics*, 6(2):7, February 2010. ISSN 1557-4679. doi: 10.2202/1557-4679.1203. URL `https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2836213/`.

Zenon W. Pylyshyn. Computation and cognition: Issues in the foundations of cognitive science. *Behavioral and Brain Sciences*, 3(1):111–169, 1980. ISSN 1469-1825. doi: 10.1017/S0140525X00002053. Place: United Kingdom Publisher: Cambridge University Press.

Gautam Reddy. The mechanistic basis of data dependence and abrupt learning in an in-context classification task. In *The Twelfth International Conference on Learning Representations*.

D. E. Rumelhart, J. L. McClelland, and PDP Research Group (eds.). *Parallel Distributed Processing. Volume 1: Foundations*. MIT Press, Cambridge, MA, 1986.

Silvester Sabathiel, James L. McClelland, and Trygve Solstad. Emerging Representations for Counting in a Neural Network Agent Interacting with a Multimodal Environment. *Artificial Life Conference Proceedings*, ALIFE 2020: The 2020 Conference on Artificial Life:736–743, 07 2020. doi: 10.1162/isal_a_00333. URL `https://doi.org/10.1162/isal_a_00333`.

Andrew M. Saxe, James L. McClelland, and Surya Ganguli. A mathematical theory of semantic development in deep neural networks. *Proceedings of the National Academy of Sciences*, 116 (23):11537–11546, May 2019. ISSN 1091-6490. doi: 10.1073/pnas.1820226116. URL `http://dx.doi.org/10.1073/pnas.1820226116`.

Andrew M. Saxe, Shagun Sodhani, and Sam Lewallen. The neural race reduction: Dynamics of abstraction in gated networks. 2022.

Adam Scherlis, Kshitij Sachan, Adam S. Jermyn, Joe Benton, and Buck Shlegeris. Polysemanticity and capacity in neural networks, 2023. URL `https://arxiv.org/abs/2210.01892`.

Paul Smolensky. On the proper treatment of connectionism. 1988.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2023.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023.

Alexander Trott, Caiming Xiong, and Richard Socher. Interpretable counting for visual question answering. *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, pp. 1–18, 2018.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL `http://arxiv.org/abs/1706.03762`.

Jesse Vig, Sebastian Gehrmann, Yonatan Belinkov, Sharon Qian, Daniel Nevo, Simas Sakenis, Jason Huang, Yaron Singer, and Stuart Shieber. Causal mediation analysis for interpreting neural nlp: The case of gender bias, 2020. URL `https://arxiv.org/abs/2004.12265`.

Kevin Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt. Interpretability in the wild: a circuit for indirect object identification in gpt-2 small, 2022. URL `https://arxiv.org/abs/2211.00593`.

Zhengxuan Wu, Atticus Geiger, Thomas Icard, Christopher Potts, and Noah D. Goodman. Interpretability at scale: Identifying causal mechanisms in alpaca, 2024. URL `https://arxiv.org/abs/2305.08809`.

Karen Wynn. Children's acquisition of the number words and the counting system. *Cognitive psychology*, 24(2):220–251, 1992.

Yan Zhang, Jonathon Hare, and Adam Prügel-Bennett. Learning to count objects in natural images for visual question answering. *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, pp. 1–17, 2018.

Qihuang Zhong, Kang Wang, Ziyang Xu, Juhua Liu, Liang Ding, Bo Du, and Dacheng Tao. Achieving >97 URL `https://arxiv.org/abs/2404.14963`.

Hattie Zhou, Arwen Bradley, Etai Littwin, Noam Razin, Omid Saremi, Josh Susskind, Samy Bengio, and Preetum Nakkiran. What algorithms can transformers learn? a study in length generalization. In *ICLR, NeurIPS Workshop*, 2023. URL `https://arxiv.org/abs/2310.16028`.

A Appendix / supplemental material

A.1 Additional Figures

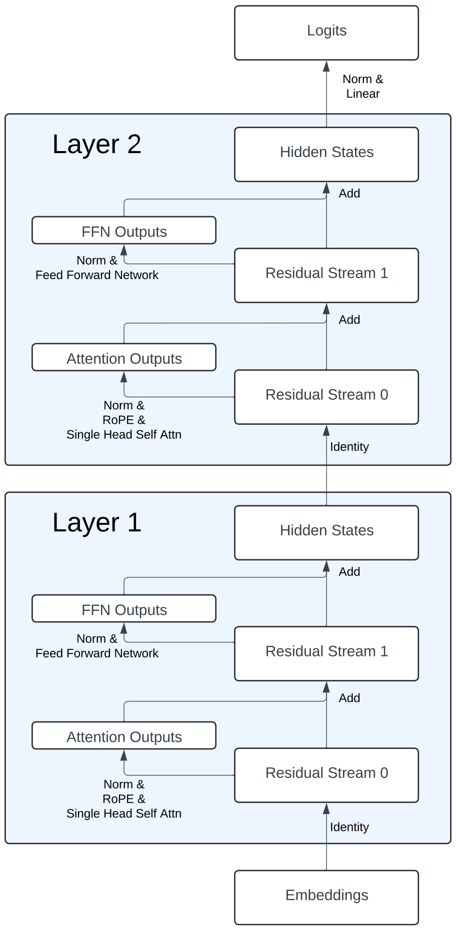

Figure 5: Diagram of the main transformer architecture used in this work. The white rectangles represent activation vectors. The arrows represent model operations. Unless otherwise stated, all interchange interventions were performed on the Hidden State activations from Layer 1. All normalizations were Layer Norms (Ba et al., 2016).

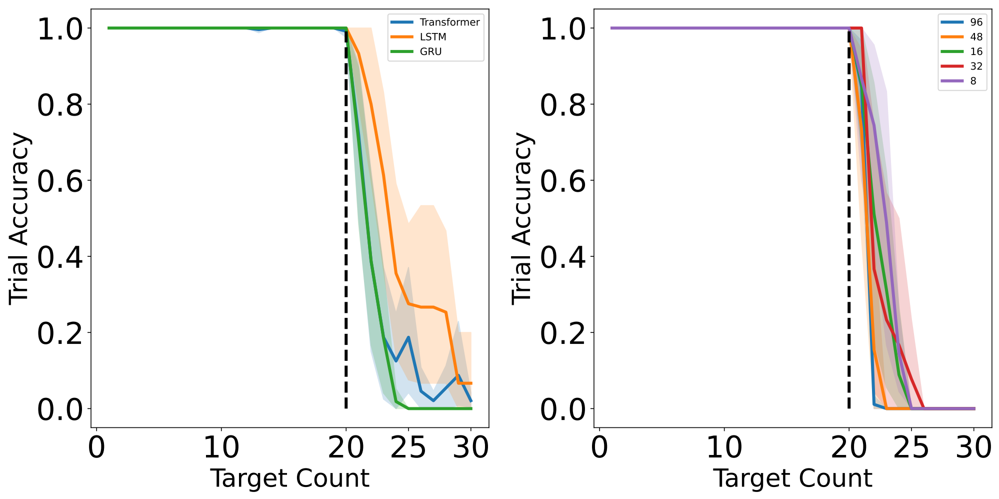

Figure 6: *Left:* The model performance on the tasks. This result includes the Multi-Object, Single-Object, and Same-Object tasks. Each target quantity includes 15 sampled sequences (even when only one configuration exists for that target quantity). 3 model seeds were dropped from the LSTM models in the Same-Object task due to lower than 99% accuracy. One seed was dropped from the transformer models in each the Single-Object and Same-Object tasks for the same reason. *Right:* The GRU performance on the tasks facetted by model size (hidden dimensionality). This result is only for GRUs train on the Multi-Object task.

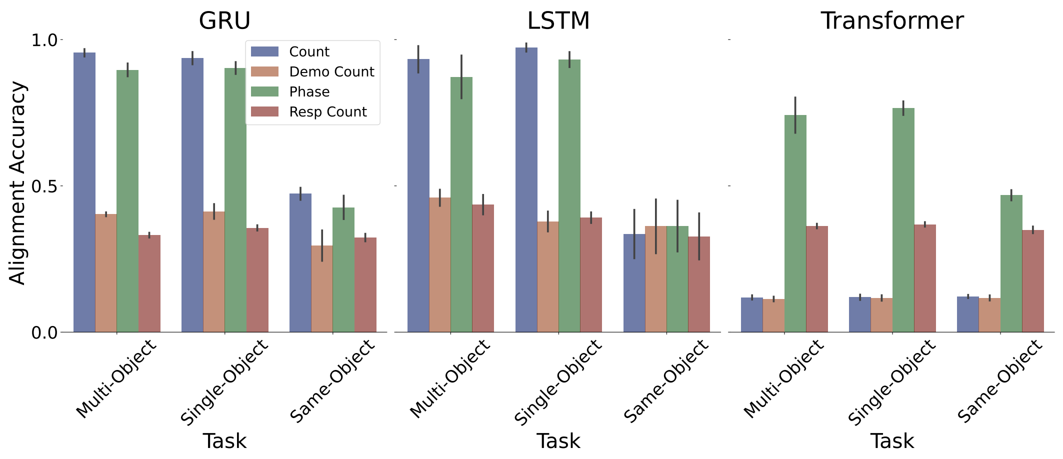

Figure 7: Interchange intervention accuracy (IIA) on variables from different symbolic programs for different tasks faceted by architecture type. The y-axis shows the proportion of trials in which the model predicts all counterfactual tokens correctly after a causal intervention for the corresponding variable on held out data.

A.2 Model Details

All artificial neural network models were implemented and trained using PyTorch (Paszke et al., 2019) on Nvidia Titan X GPUs. Unless otherwise stated, all models used an embedding and hidden state size of 20 dimensions. To make the token predictions, each model used a two layer multi-layer perceptron (MLP) with GELU nonlinearities, with a hidden layer size of 4 times the hidden state dimensionality with 50% dropout on the hidden layer. The GRU and LSTM model variants each consisted of a single recurrent cell followed by the output MLP. Unless otherwise stated, the transformer architecture consisted of two layers using Rotary positional encodings (Su et al., 2023). Each model variant used the same learning rate scheduler, which consisted of the original transformer (Vaswani et al., 2017) scheduling of warmup followed by decay. We used 100 warmup steps, a maximum learning rate of

0.001 , a minimum of 1e-7, and a decay rate of 0.5. We used a batch size of 128, which caused each epoch to consist of 8 gradient update steps.

A.3 DAS Training Details

A.3.1 Rotation Matrix Training

To train the DAS rotation matrices, we applied PyTorch's default orthogonal parametrization to a square matrix of the same size as the model's state dimensionality. PyTorch creates the orthogonal matrix as the exponential of a skew symmetric matrix. In all experiments, we selected the number of dimensions to intervene upon as half of the dimensionality of the state. We chose this value after an initial hyperparameter search that showed the number of dimensions had little impact on performance between 5-15 dimensions. We sampled 10000 sequence pairs and for each of these pairs, we uniformly sampled corresponding indices to perform the interventions. We excluded the BOS, and EOS tokens from possible intervention sample indices. When intervening upon a state in the demo phase, we uniformly sampled 0-3 steps to continue the demo phase before changing the phase by inserting the trigger token. We used a learning rate of 0.003 and a batch size of 512.

A.3.2 Symbolic Program Algorithms

Algorithm 1 One sequence step of the Up-Down Program

```
$q \leftarrow$ Count
$p \leftarrow$ Phase
$y \leftarrow$ input token
**if** $y ==$ BOS **then**                          ▷ BOS is beginning of sequence token
    $q \leftarrow 0, p \leftarrow 0$
    return sample(D)                                ▷ sample a demo token
**else if** $y \in$ D **then**                      ▷ D is set of demo tokens
    $q \leftarrow q + 1$
    return sample(D)
**else if** $y ==$ T **then**                       ▷ T is trigger token
    $p \leftarrow 1$
**else if** $y ==$ R **then**                       ▷ R is response token
    $q \leftarrow q - 1$
**end if**
**if** $(q == 0)$ & $(p == 1)$ **then**
    return EOS                                      ▷ EOS is end of sequence token
**end if**
return R
```

A.4 Simplified Transformer

The self-attention calculation for a single query $q_r \in R^d$ from a response token, denoted by the subscript r, is as follows:

$$\text{Attention}(q_r, K, V) = V\big(\text{softmax}(\frac{K^\top q_r}{\sqrt{d}})\big) = \sum_{i=1}^{n} \frac{e^{\frac{q_r^\top k_i}{\sqrt{d}}}}{\sum_{j=1}^{n} e^{\frac{q_r^\top k_j}{\sqrt{d}}}} v_i = \sum_{i=1}^{n} \frac{s_i^r}{\sum_{j=1}^{n} s_j^r} v_i \tag{5}$$

Where d is the dimensionality of the model, n is the sequence length, $K \in R^{d\times n}$ is a matrix of column vector keys, $V \in R^{d\times n}$ is a matrix of column vector values, and $s_i^r = e^{\frac{q_r^\top k_i}{\sqrt{d}}}$, using r to denote the token type that produced q. We refer to $s_i^r v_i$ as the strength value of the i^{th} token for the query q_r.

In a transformer without positional encodings, each of the queries for the response tokens will produce equal strength values to one another for a given key-value pair. Thus, under the assumption that the attention mechanism is performing a sum of the count contributions from each token in the sequence,

Algorithm 2 One sequence step of the Up-Up Program

```
$d \leftarrow$ Demo Count
$r \leftarrow$ Resp Count
$p \leftarrow$ Phase
$y \leftarrow$ input token
**if** $y ==$ BOS **then**                          ▷ BOS is beginning of sequence token
    $d \leftarrow 0, r \leftarrow 0, p \leftarrow 0$
    return sample(D)                                ▷ sample a demo token
**else if** $y \in$ D **then**                      ▷ D is set of demo tokens
    $d \leftarrow d + 1$
    return sample(D)
**else if** $y ==$ T **then**                       ▷ T is trigger token
    $p \leftarrow 1$
**else if** $y ==$ R **then**                       ▷ R is response token
    $r \leftarrow r + 1$
**end if**
**if** $(d == r)$ & $(p == 1)$ **then**
    return EOS                                      ▷ EOS is end of sequence token
**end if**
return R
```

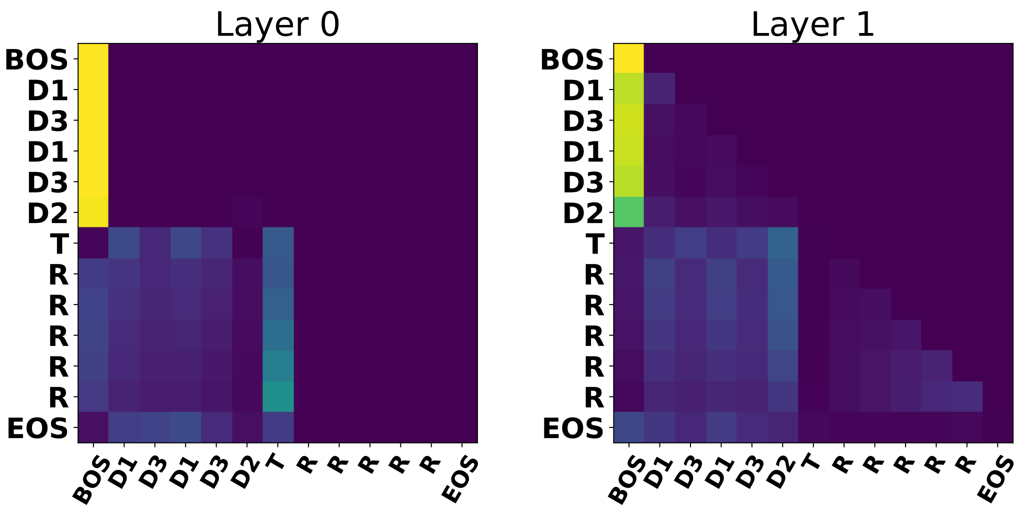

Figure 8: Attention weights for a single transformer model with two layers and using rotary positional encodings. Queries are displayed on the vertical axis in order of their appearance starting at the top. Keys are displayed on the horizontal axis starting from the left. Queries are only able to attend to themselves and preceding keys.

we should be able to use the $s_i^r v_i$ to increment and decrement the number of tokens the model will produce for a given sequence in the following way:

$$\text{IncrementedAttention}(q_r, K, V) = \frac{1}{s_r^r + \sum_{j=1}^{n} s_j^r}\left(s_r^r v_r + \sum_{i=1}^{n} s_i v_i\right) \tag{6}$$

Where the subscript r denotes the strength s_r and value v_r were calculated from a response key-value pair. Similarly, we can decrement the count using a key-value pair from a demonstration token, D, in the following way.

$$\text{DecrementedAttention}(q_r, K, V) = \frac{1}{s_D^r + \sum_{j=1}^{n} s_j^r}\left(s_D^r v_D + \sum_{i=1}^{n} s_i v_i\right) \tag{7}$$

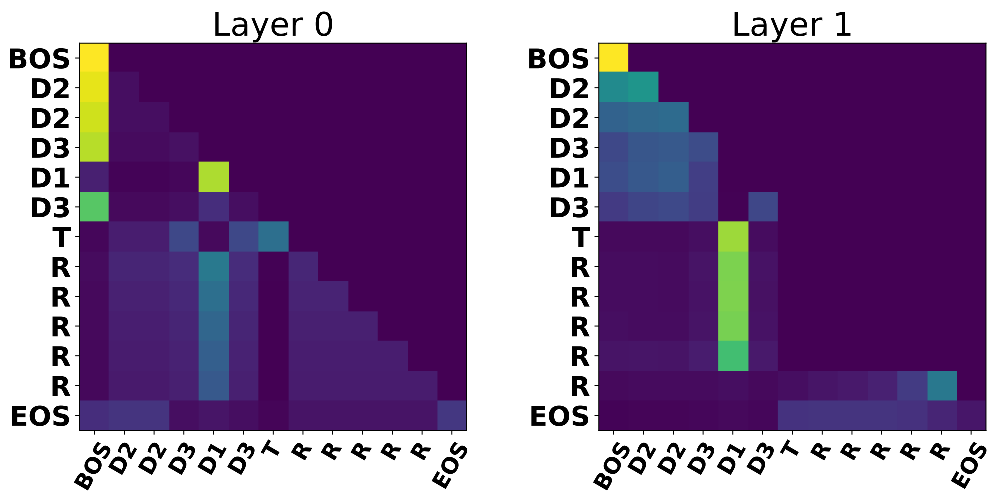

Figure 9: Attention weights for a single transformer model seed with two layers and no positional encodings (NPE). Queries are displayed on the vertical axis in order of their appearance starting at the top. Keys are displayed on the horizontal axis starting from the left. Queries are only able to attend to themselves and preceding keys.

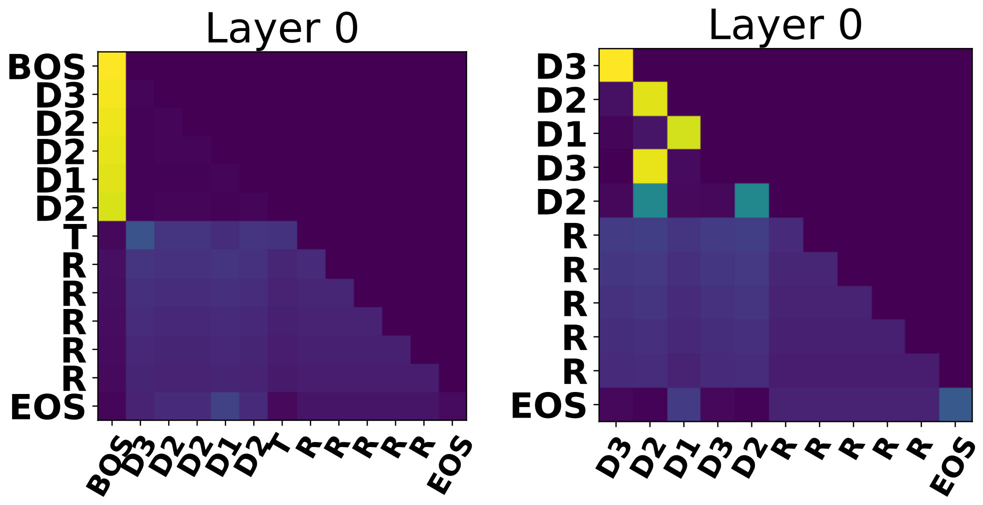

Figure 10: Left: Attention weights for a single transformer model seed with one layer and no positional encodings. Right: Attention weights for a single transformer seed with one layer and no positional encodings trained without the BOS and trigger token types. In both figures, queries are displayed on the vertical axis in order of their appearance in the sequence starting at the top. Keys are displayed on the horizontal axis starting from the left. Queries are only able to attend to themselves and preceding keys.

As a sanity check we use single layer transformers without positional encodings and add and subtract from the transformer's count using the strength values as described in this section. We are able to change the position at which it produces the EOS token with 100% accuracy.

A.5 Additional Interventions Continued

We detail in this section why our activation transfers are sufficient to demonstrate that the transformers use a solution that re-references/recomputes the relevant information to solve the tasks at each step in the sequence. The hidden states in Layer 1 are a bottleneck at which a cumulative counting variable must exist if it were to use a strategy like the Up-Down or Up-Up programs. This is because the Attention Outputs of Layer 1 are the first activations that have had an opportunity to cross communicate between token positions. This means that the representations between the Residual Stream 1 of Layer 1 up to the Residual Stream 0 of Layer 2 cannot have read off a cumulative state from the previous token position other than reading off the positional information from the previous positional encodings. The 2-layer architecture is then limited in that it has only one more opportunity to transfer information between positions—the attention mechanism in Layer 2. Thus, if a hidden state at time t were to have encoded a cumulative representation of the count that will be used by the model at time $t + 1$, that cumulative representation must exist in the activation vectors between the Residual Stream 1 in Layer 1 and the Residual Stream 0 of Layer 2. If it is using such a cumulative representation, then when we perform a full activation swap in the Layer 1 hidden states then the resulting predictions should be influenced by the swap. As Figures 2 and 11 indicate, the resulting transformer predictions are mostly unchanged by the intervention, demonstrating a recomputing of information at each step in the task.

A.6 VARIABLE-LENGTH TASK VARIANTS

Here we include additional tasks to prevent the transformers with positional encodings from learning a solution that relies on reading out positional information. We introduce Variable-Length variants of each of the Multi-Object, Single-Object, and Same-Object tasks. In the Variable-Length versions, each token in the demo phase has a 0.2 probability of being sampled as a unique "void" token type, V, that should be ignored when determining the target count of the sequence. The number of demo tokens will still be equal to the target count when the trigger token is presented. We include these void tokens as a way to vary the length of the demo phase for a given target count, thus breaking correlations between positional information and target quantities. As an example, consider the possible sequence with a target count of 2: "BOS V D V V D T R R EOS".

We show the transformer performance and the IIA for the Ctx-Distr interventions in Figure 11. Although we do not make strong claims about the manner in which these transformers solve these new tasks, we do highlight the fact that the transformers can no longer use a direct positional encoding readout to achieve 100% accuracy. These results are consistent with the hypothesis that the transformers are using the more specific, summing version of the Ctx-Distr strategy to solve these tasks, much as the no-positional encoding transformers do.

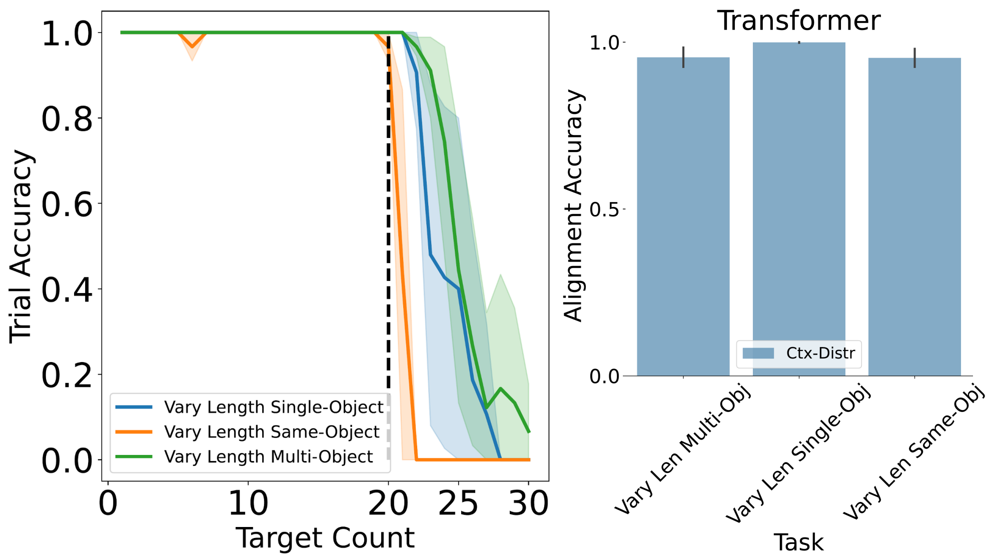

Figure 11: *Left:* The transformer performance on variable length variants of the 3 tasks. *Right:* The interchange intervention accuracy using the Ctx-Distr program for the transformer models on the variable length tasks. In both panels, 4 model seeds were dropped from the models in the variable length Same-Object task due to lower than 99% accuracy, and one seed was dropped from the variable length Single-Object task for the same reason.

A.7 Principle Components Analysis

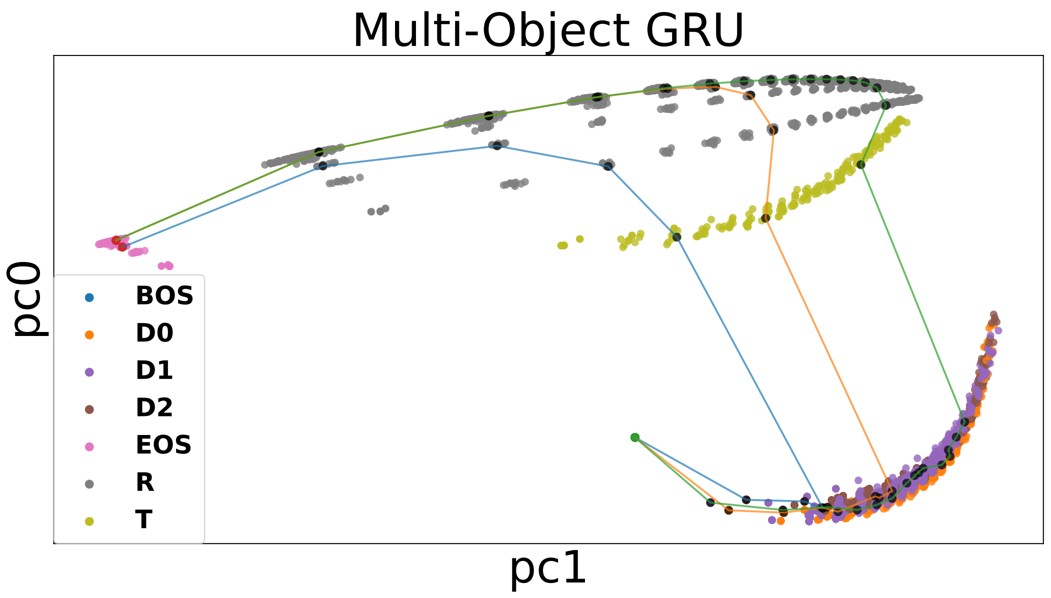

Figure 12: Principal Components Analysis of a single GRU model seed including hidden state representations over 10 trials for each target quantity from 1 to 20 in the Multi-Object task variant. Green points indicate the start of a plotted trajectory, black points indicate an intermediate step, and red points indicate the end of a plotted trajectory. The blue line plots a single trajectory from start to finish with a target quantity of 3. Similarly, the orange and green lines follow single trajectories of 7 and 15 respectively.

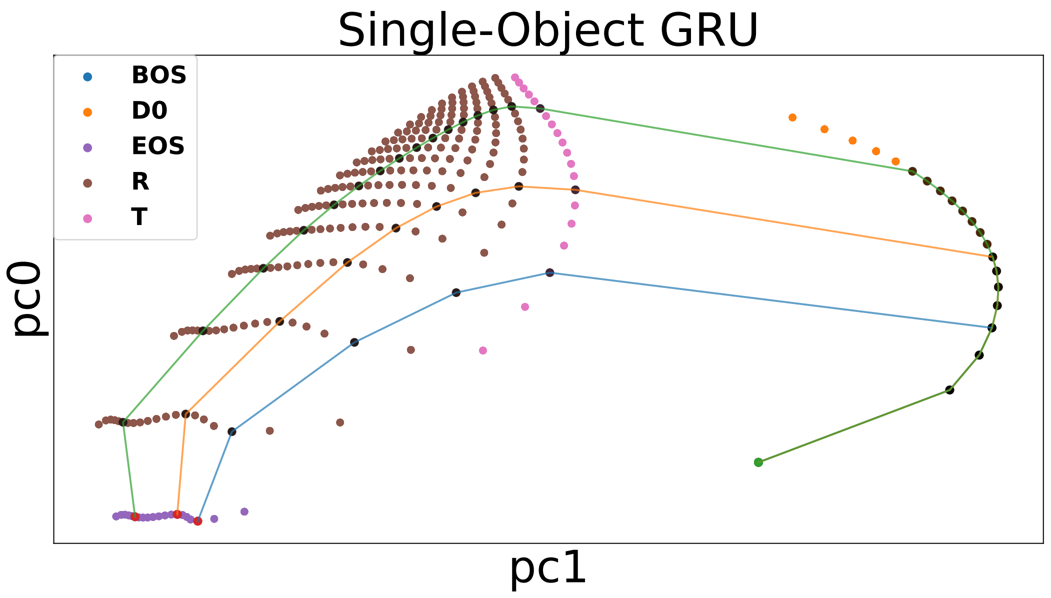

Figure 13: Principal Components Analysis of a single GRU model seed including hidden state representations over 10 trials for each target quantity from 1 to 20 in the Single Object task variant. Green points indicate the start of a plotted trajectory, black points indicate an intermediate step, and red points indicate the end of a plotted trajectory. The blue line plots a single trajectory from start to finish with a target quantity of 3. Similarly, the orange and green lines follow single trajectories of 7 and 15 respectively.

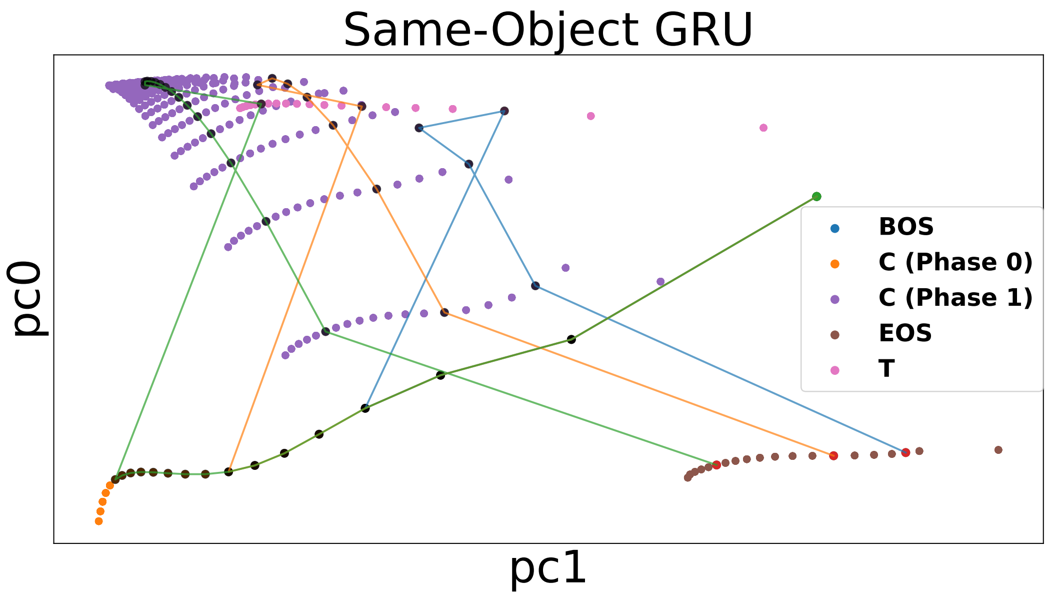

Figure 14: Principal Components Analysis of a single GRU model seed including hidden state representations over 10 trials for each target quantity from 1 to 20 in the Same-Object task variant. Green points indicate the start of a plotted trajectory, black points indicate an intermediate step, and red points indicate the end of a plotted trajectory. The blue line plots a single trajectory from start to finish with a target quantity of 3. Similarly, the orange and green lines follow single trajectories of 7 and 15 respectively.