

The Emergent Symbolic Structure of Artificial Neural Networks

R. Thomas McCoy, Paul Soulos, Tal Linzen, and Paul Smolensky

Lossfunk Reading Group

Srinitish Srinivasan 09/09/2026

Why vectors seem to miss symbolic structure

- **Symbolic systems:** discrete units combine in structured arrangements, such as syntax trees and logical formulas.
- **Neural networks:** information appears as continuous vectors that look qualitatively different from symbol structures.
- **The problem:** a sum of concept vectors can record which concepts are present, but addition does not record how they are arranged.
- **The puzzle:** vector-based networks still excel in language, mathematics, and coding.

TLDR; symbolic structure inside vectors

- **Proposal:** standard networks may implicitly realize Tensor Product Representation structure even when they were not designed to do so.
- **Method:** DISCOVER replaces a network's representation-generating process with a closed-form TPR approximation.
- **Evidence:** the approximation preserves behavior across letter-sequence models, seven LLMs, and GPT-OSS on symbolic tasks.
- **Stronger tests:** targeted causal edits and generalization to unseen role-filler pairs indicate that the structure affects behavior and is systematic.

The binding problem

cats chase dogs

$$e(\text{cats}) + e(\text{chase}) + e(\text{dogs})$$

dogs chase cats

$$e(\text{dogs}) + e(\text{chase}) + e(\text{cats})$$

Both sums are identical because addition ignores order, but the sentences mean different things.

The binding problem asks how a neural network pairs each piece of information with its position, such as cats with subject and dogs with object.

Tensor Product Representations

fillers

cats
chase
dogs

**{ subject: cats, verb: chase, object:
dogs }**

roles

subject
verb
object

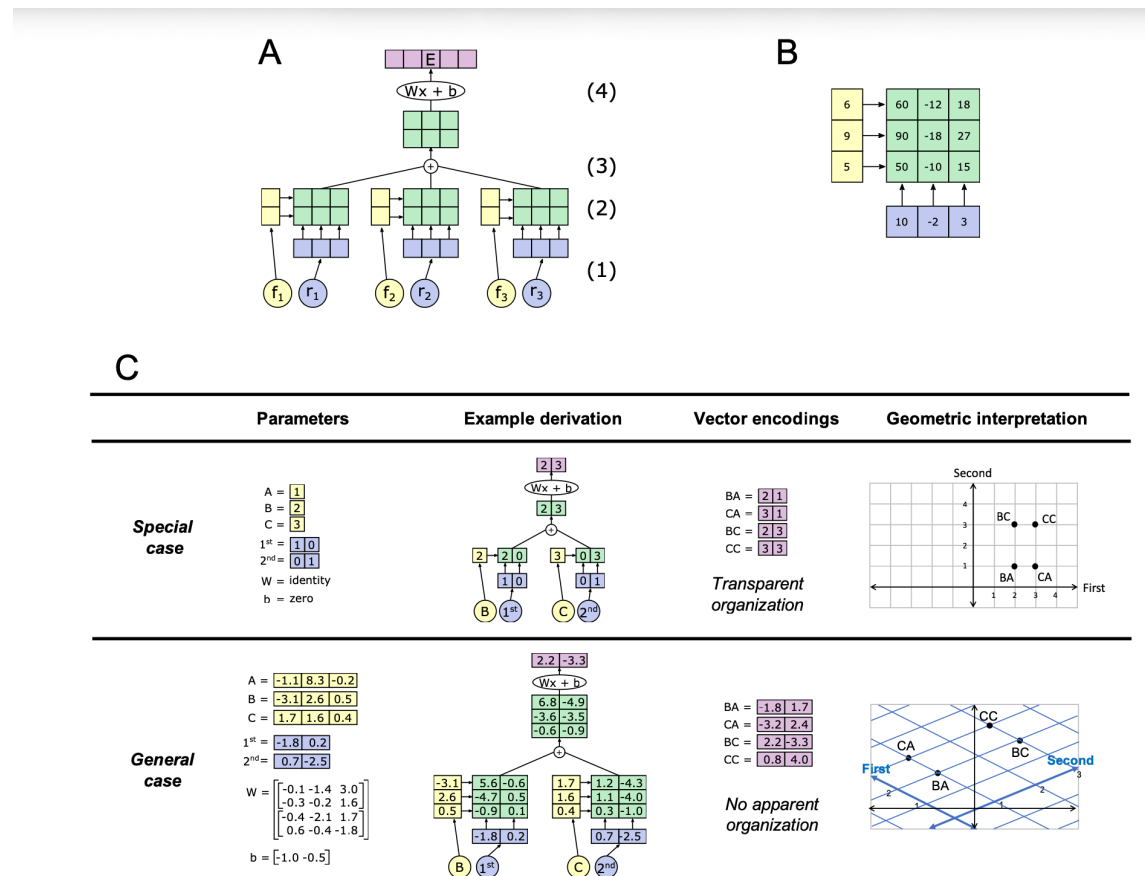
Each filler vector is bound to its role vector with a tensor product.
The bound pairs are summed to encode the complete structure.

Unbinding can recover which filler occupies a role.

Linearly transformed TPRs

$$\mathbf{E} = \mathbf{W}(\sum_i \mathbf{f}_i \otimes \mathbf{r}_i) + \mathbf{b}$$

- Each tensor product encodes one role-filler pair.
- The sum encodes the complete structure.
- $\mathbf{W}$ and $\mathbf{b}$ resize and reshape the sum into the network's vector space.
- Stretching, rotation, and translation can hide the organization from human inspection while preserving it for the network.



DISCOVER running example

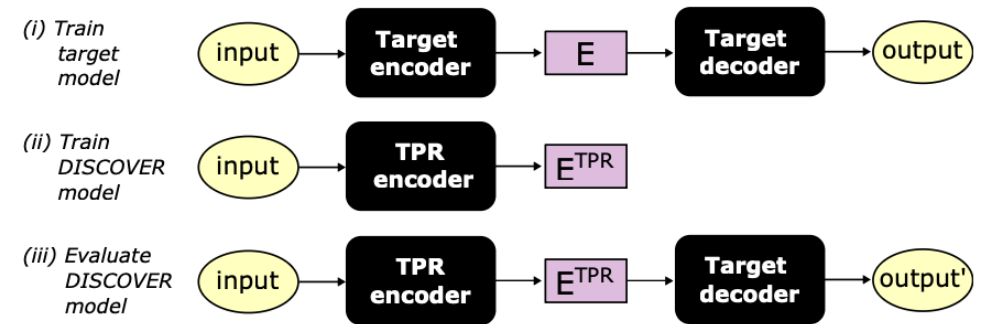
Input: Q M Z

Output: Z M Q

- A sequence-to-sequence GRU encoder maps the input list to one vector E. The GRU decoder uses E to produce the reversed list.
- All 10 trained copies reach 100% accuracy on unseen test examples.
- Therefore E contains the letters and their positions, which are both required for reversal.
- DISCOVER tests whether E captures this information through TPR structure.

The DISCOVER procedure

- **1. Target model:** train the network and collect the internal encoding E .
- **2. TPR approximation:** train filler vectors, role vectors, W , and b to minimize mean squared error between E^{TPR} and E .
- **3. Functional test:** replace E with E^{TPR} and run the target model's original decoder
- **Approximation accuracy:** the proportion of test examples for which the decoder produces the entire correct output.



Which roles does the reversal model use?

Role scheme	Q	M	Z	C
Left-to-right	first	second	third	fourth
Right-to-left	fourth-to-last	third-to-last	second-to-last	last
Bidirectional	(0,3)	(1,2)	(2,1)	(3,0)
Wickelroles	#_M	Q_Z	M_C	Z_#
Bag-of-words	present	present	present	present

Result: bidirectional TPRs are perfect or nearly perfect in all 10 runs. The lowest approximation accuracy is 99.98%.

Right-to-left nearly matches it. Left-to-right and bag-of-words perform poorly.

Letter-sequence tasks and architectures

- **Tasks:** copy Q M Z V R, reverse it as R V Z M Q, or interleave it as Q R M V Z.
- **Why these tasks:** each requires the model to represent which letters occur and where they occur.
- **Architectures:** MLP, GRU, Transformer, and the paper's bottleneck Transformer.
- **Shared setup:** an encoder creates vector representations and a decoder uses them to produce the output sequence.

Transformer and bottleneck Transformer

Standard Transformer

- The encoder produces one vector per input token.
- The decoder can access all input vectors.
- DISCOVER approximates every vector and hypothesizes one role-filler pair per vector: the local letter paired with its role.

Bottleneck Transformer

- The decoder can access only the first input-position vector.
- Because the encoder is bidirectional, that vector can receive information from the entire sequence.
- The bottleneck vector must encode the whole list, so its TPR contains multiple role-filler pairs in one vector.

Letter-sequence results

- The study trains 600 DISCOVER models across four architectures, three tasks, 10 target-model runs, and five role schemes.
- Bidirectional roles give strong approximations for all 12 architecture-task combinations.
- The lowest mean accuracy is 0.973 for the reversing bottleneck Transformer. Every other mean exceeds 0.99.
- Copying favors left-to-right roles, reversing favors right-to-left roles, and interleaving usually needs both directions.
- Bag-of-words fails when order matters. It succeeds when the target task is alphabetic sorting and input order is irrelevant.

Why analyze the period token?

Dogs jump .

- Each LLM layer produces one vector per token, so DISCOVER needs a vector that may encode a complete structure.
- In a causal LLM, the period is the earliest position whose representation can use the entire preceding sentence.
- The authors test periods after noun lists, subject-verb-object sentences, and complex sentences.
- For each of seven open-weight LLMs, they sample five layers: the first non-embedding layer, about 25%, 50%, and 75% depth, and the final layer.
- A period-unpacking Transformer first checks whether the period vector can reconstruct the preceding sequence exactly.

Period encodings have TPR structure

- **Information check:** period unpacking reaches 100% on subject-verb-object sentences, typically above 85% on lists, and above 50% on complex sentences under exact match.
- **DISCOVER:** a separate TPR approximation is trained for each sampled layer. Bidirectional roles approximate list periods well, and subject-verb-object roles give perfect approximation on simple sentences.
- **Complex sentences:** bidirectional roles consistently outperform syntactic roles, suggesting that these period vectors mainly capture linear order.
- **Seven LLMs:** period representations capture most preceding structure and occupy spaces with TPR structure. On complex sentences, clean bidirectional approximations can outperform the original encodings.

Arithmetic, logic, code, and language

Domain	Task structure	Paper example
Arithmetic	$a \times b + c$ or $a + b \times c$. Multiplication precedes addition	$7 + -6 \times 3 = -11$
Logic	Two syllogistic premises and four candidate conclusions	Return the first conclusion that follows
Computer code	Execute a Python list function. Bind function and variable names to values	<code>change(y) = [Q, S, M, V]</code>
Language	Passivization, tense reinflection, and yes/no question formation	Transform syntax while preserving the intended content

How DISCOVER encodes GPT-OSS

One DISCOVER model per layer. All non-templatic input-token vectors are replaced by TPR approximations.

- **Bag-of-words:** the token and all earlier tokens share one role.
- **Bidirectional (self):** each token encodes only itself, paired with its position.
- **Bidirectional (all):** each token encodes itself and the preceding tokens, each with a positional role.
- **Task-specific (self):** each token encodes itself with an arithmetic, logic, code, or syntax role.
- **Task-specific (all):** each token encodes itself and preceding tokens with roles that combine the source and current task positions.

GPT-OSS results

- For all six tasks, task-specific (all) outperforms the other role schemes.
- Its approximation accuracy stays extremely close to GPT-OSS's own task accuracy.
- The largest gap is 2.36 percentage points, on arithmetic.
- The result supports two claims: each token representation combines multiple role-filler pairs, and those roles carry task-specific meaning.
- The paper's interpretation is a TPR at each input token that represents that token and the preceding context.

Why TPRs support precise causal edits

$$\text{TPR}(\mathbf{Q}, \mathbf{M}, \mathbf{Z}) = \text{TPR}(\text{1st}:\mathbf{Q}) + \text{TPR}(\text{2nd}:\mathbf{M}) + \text{TPR}(\text{3rd}:\mathbf{Z})$$

$$\text{target}(\mathbf{Q}, \mathbf{M}, \mathbf{Z}) - \text{TPR}(\text{3rd}:\mathbf{Z}) + \text{TPR}(\text{3rd}:\mathbf{U}) \approx \text{target}(\mathbf{Q}, \mathbf{M}, \mathbf{U})$$

- Tensor-product binding is bilinear, and aggregation across constituents is additive.
- Constituent surgery subtracts the old role-filler contribution and adds the new one.
- The edit changes every coordinate by a precise amount. It does not assume one concept per neuron or coordinate.
- The edited vector is fed back into the original model to test whether behavior changes as predicted.

Causal interventions on sequence and period encodings

Letter-sequence models

Use bidirectional DISCOVER role-filler representations.

Edit the single sequence vector in MLPs, GRUs, and bottleneck Transformers. Standard Transformers are excluded because each vector holds one pair. Subtract one pair, add another, and decode the edited vector.

Result: near-perfect accuracy across tasks and architectures.

LLM period encodings

Edit the period vector that represents the preceding list or sentence.

For lists of length five, replace one word. For coordinated sentences, replace one noun.

Feed the edited period representation to the trained period-unpacking model.

Result: high intervention accuracy, close to the accuracy obtained from the original period encodings.

Causal interventions in GPT-OSS

- **Method:** edit every representation across tokens and layers that DISCOVER predicts must change, because each token encodes itself and earlier tokens.
- **Fillers:** replace values such as a number in arithmetic, an element in code, a called function, or a variable. The arithmetic filler intervention reaches 0.978 accuracy.
- **Roles:** move an element to a new structural position by replacing its role, such as object adjective with subject adjective. Reported averages include 0.978 for swapping arithmetic numbers and 0.950 for moving a code-list element.
- **Results:** 31 intervention types across six tasks average 0.903. Local role edits fall well below complete edits, and structure-respecting edits outperform edits that preserve only linear order.

Discussion

Symbolic Structure: Neural networks without explicit symbols still develop implicit symbolic structure. Their success therefore does not support eliminating symbols from theories of intelligence. The networks approximate rather than exactly implement symbol systems.

Extensions: study how networks produce and process this structure, how it emerges in training, and how DISCOVER can cover continuous or recursive roles and fillers with less supervision.

Why compositional generalization remains hard: a network may develop compositional representations only for role-filler combinations seen during training, so compositional encoding does not guarantee behavioral generalization to unseen combinations.