Deriving Layers from the Connectome

v2: f resolves what P leaves ambiguous

Claude and MJC

18 February 2026

Abstract

In the Universal Model $u = (E, P, t, f)$, the pattern set P defines a directed graph on E—the *connectome*—whose structure constrains but does not fully determine the forward pass f. When the connectome is a DAG, f is determined (up to parallelism) by topological sort. When the connectome has cycles, P is *ambiguous*: multiple forward passes are consistent with the same patterns. This ambiguity must be resolved by f as part of the model specification; the UM runner surfaces the ambiguity rather than resolving it. We show that the RNN's solution—synchronization with an input stream—is a specific choice of f for a cyclic connectome, and one observed biologically. We describe the relationship f = execute, ω = record, and sketch a UM explorer that enables interactive querying of models.

1 The Connectome

Definition 1 (Connectome)**.** *Given a UM instance $u = (E, P, t, f)$ with event space E, pattern set P, thought vector t, and forward pass f, the* connectome $G(P)$ *is the directed graph with:*

- *Vertices: the events $e \in E$*
- *Edges: for each atomic pattern $p = (e_i, e_j, w)$ with $w > 0$, an edge $e_i \to e_j$*

The connectome captures the *potential* information flow of the model. An edge $e_i \to e_j$ means that support at e_i *could* propagate to e_j. Whether it does, and in what order, is the business of f, not P.

Definition 2 (ES-level Connectome)**.** *The* coarsened connectome $\bar{G}(P)$ *has event spaces as vertices:*

- *Vertices: the event spaces $\{\mathrm{ES}_1, \ldots, \mathrm{ES}_k\}$*
- *Edges: $\mathrm{ES}_a \to \mathrm{ES}_b$ if any pattern connects an event in ES_a to an event in ES_b*

For KN-6, the coarsened connectome is:

$$\mathrm{ES}_{\mathrm{input}} \to \mathrm{ES}_{\mathrm{ctx}_1} \to \mathrm{ES}_{\mathrm{ctx}_2} \to \cdots \to \mathrm{ES}_{\mathrm{ctx}_6} \to \mathrm{ES}_{\mathrm{output}}$$

plus the LPP connections from each context ES to the output ES. This is a DAG—no cycles—and the topological order uniquely determines f (up to parallelism within layers).

2 Deriving Layers

Proposition 1 (Layer Derivation from DAG)**.** *If $\bar{G}(P)$ is a DAG, then a topological sort gives a valid execution order for f. All patterns within a single layer may be applied in parallel.*

This is the acyclic case: **when P is a DAG, layers are fully determined by P.** No additional specification is needed. This is the easy case, and it is what the UM runner currently implements for KN-6.

2.1 Feedforward Networks

A feedforward network corresponds to a DAG connectome. Each "layer" in the traditional sense is one or more ESs at the same depth in the topological sort. The KN-6 tower is an example: input → context ESs → output.

2.2 Cycles: Where P Becomes Ambiguous

A cycle in $\bar{G}(P)$ means the connectome is not a DAG. The topological sort is no longer unique; in fact, no topological sort exists for the cyclic subgraph.

Definition 3 (Recurrence)**.** *A cycle in $\bar{G}(P)$ defines a* recurrence*. The ESs participating in the cycle form a* recurrent group*.*

The sat-rnn's hidden state corresponds to an ES with a self-loop: $\mathrm{ES}_h \to \mathrm{ES}_h$ via W_h. This is the simplest possible cycle.

The critical point: a cyclic P does not determine f. The same pattern set P (same events, same connections, same weights) is consistent with multiple forward passes:

- Apply the cycle once, twice, or k times per input.
- Iterate to convergence (if it converges).
- Apply asynchronously with different ESs updating at different rates.
- Synchronize with an external clock (input stream frequency).

Each of these is a *different model* u even though E and P are identical. The choice is part of f, which is part of the model specification.

Remark 1 (The Runner Must Not Resolve Ambiguity)**.** *The UM runner's job, when encountering a cyclic connectome, is to* surface the ambiguity*: report the cycles, identify the recurrent groups, and require f to specify how they are resolved. The runner should not silently pick a resolution strategy; that would be making architectural decisions that belong to the model designer.*

2.3 Input Stream Synchronization

The RNN's solution to the cycle ambiguity is elegant: synchronize the recurrent group with the input stream. At each input event (each byte in our case):

1. Read the input into $\mathrm{ES}_{\mathrm{input}}$.
2. Apply f once through the entire connectome, including the cyclic patterns.
3. The thought vector t persists to the next input event.

This resolves the ambiguity by tying the cycle to an external clock. The recurrence is temporal: one application of the cyclic patterns per input byte, with state carried forward in the thought vector.

This pattern is observed biologically: sensory processing synchronizes neural oscillations with the input stream (auditory cortex locks to speech rhythm, visual cortex locks to saccade frequency). The biological analogy is not accidental—it reflects a deep constraint. Cyclic neural circuits *need* an external clock to be well-defined.

Example 1 (Sat-RNN as UM)**.** *The sat-rnn has $u = (E, P, t, f)$ with:*

- $E = \mathrm{ES}_{\mathrm{input}} \cup \mathrm{ES}_h \cup \mathrm{ES}_{\mathrm{output}}$

- *P includes $W_x : \mathrm{ES}_{\mathrm{input}} \to \mathrm{ES}_h$, $W_h : \mathrm{ES}_h \to \mathrm{ES}_h$ (cycle), $W_y : \mathrm{ES}_h \to \mathrm{ES}_{\mathrm{output}}$*
- *f = "at each byte, apply W_x then W_h then W_y, once; carry $t|_{\mathrm{ES}_h}$ forward"*

The cycle in P ($\mathrm{ES}_h \to \mathrm{ES}_h$) is resolved by f's specification of "once per byte." A different f—say, "iterate W_h until convergence"—would give a different model with the same P.

2.4 Other Resolutions

Input stream synchronization is not the only option. Other valid resolutions for cyclic connectomes include:

- **Fixed-depth unrolling**: apply cyclic patterns k times per input (BPTT-k). This is a family of models parameterized by k.
- **Convergence**: iterate until $\|t^{(k+1)} - t^{(k)}\| < \epsilon$. Well-defined only if the dynamics are contractive. Our chaos results (20260207) show the sat-rnn does *not* converge—the Jacobian grows, so this resolution fails.
- **Multi-frequency**: different recurrent groups synchronized to different clocks. E.g., one group updates per byte, another per word, another per sentence. This corresponds to a hierarchy of temporal scales.
- **Asynchronous**: events fire when their inputs change, with no global clock. This is biologically plausible (spike timing) but computationally complex.

Each choice gives a different f and therefore a different model. The connectome $G(P)$ constrains f (the execution must respect the edge directions) but does not determine it when cycles are present.

3 f-Patterns and ω-Patterns

The distinction between structural patterns (weight > 0) and recording patterns (weight $= 0$) corresponds precisely to the two components of the UM update:

- **f-patterns** (weight > 0): executed during the forward pass. These propagate support through the connectome. Their application order is determined by f, which respects the causal structure derived from P.
- **ω-patterns** (weight $= 0$, i.e. LPPs): observed during learning. After f completes, the learning function ω examines the final thought vector t and updates the ω-patterns' recorded counts. These are the LPPs' hash tables.

This gives the two-phase structure of each time step:

1. **Execute** (f): apply f-patterns to propagate support from input through the connectome to output. The connectome's causal structure (with cycle resolution from f's spec) determines the order.
2. **Record** (ω): examine the thought vector after f has completed. For each ω-pattern (LPP) between ES_a and ES_b, record the joint event: which events in ES_a and ES_b have support, and update the counts.

In the UM runner, this is already what happens: the scoring loop runs the KN-6 forward pass (f), then calls `umr_kn6_learn` (ω). The notation just makes explicit what was implicit.

Remark 2 (The Brain Analogy)**.** *In biological neural networks, synapses must exist before joint activity can be detected (Hebbian learning requires co-activation across an existing synapse). An ω-pattern is a synapse with zero weight: the connection exists, enabling recording, but does not propagate signal.*

Biology's developmental thinning (massive synapse pruning in adolescence) corresponds to removing ω-patterns that were never activated. In the UM runner, LPPs use hash tables—demand-allocated recording—which avoids the pre-allocation problem entirely. This is more memory-efficient than biology but less biologically plausible.

4 How Context Events Change the Connectome

Adding a context event $e_c \in \mathrm{ES}_c$ to the model adds new edges to the connectome:

$$e_c \to e_i \quad \text{for each } e_i \in \mathrm{ES}_a \text{ that } e_c \text{ connects to}$$

This changes the coarsened connectome: $\mathrm{ES}_c \to \mathrm{ES}_a$ is a new edge.

Proposition 2 (Context Events Add Depth)**.** *Adding a context ES to a DAG connectome increases the graph's depth by at most 1. Each new context ES adds one new layer to f's execution order.*

Crucially, adding a context ES to a DAG keeps it a DAG (assuming the context ES is causally prior, not part of a cycle). This means no new ambiguity is introduced: f is still determined by topological sort. Context events are "safe" extensions of the connectome.

If a context event *does* introduce a cycle (e.g., a context ES that depends on the output ES), then the model designer must extend f to resolve the new ambiguity. This would correspond to a model that adjusts its context based on its own predictions—a form of self-attention or deliberation.

5 The UM Runner's Execution Model

Given the connectome, the UM runner implements the two-phase update:

1. **Build** $\bar{G}(P)$ from all registered patterns and LPPs.
2. **Detect cycles**. If $\bar{G}(P)$ is a DAG, topological sort determines f. If cyclic, *surface the ambiguity*: report the cycles, identify the recurrent groups, and require f to be specified.
3. **Execute** f: apply f-patterns layer by layer.
   - For non-cyclic layers: apply all patterns in parallel (one max-min pass).
   - For cyclic layers: apply as specified by f (e.g., once per input byte with state carry-forward).
4. **Record** ω: after f completes, update all ω-patterns (LPPs) based on the final thought vector t.

The runner currently hardcodes f (KN interpolation chain). Making it connectome-driven requires:

- A graph builder that analyzes P and constructs $\bar{G}(P)$.
- A cycle detector (Tarjan's SCC algorithm).
- A specification format for f that covers DAG ordering and cycle resolution.
- An ω scheduler that runs after f on each time step.

6 Composability of P-Programs

If P-program A has connectome G_A and P-program B has connectome G_B, their composition has connectome $G_A \cup G_B$ plus any interface edges. The topological sort of the combined graph (if acyclic) determines the execution order automatically.

This is how context events compose: a tag-detection P-program and a word-length P-program can be added independently, and the connectome determines that both run before the main KN-6 prediction.

If composition introduces cycles (two P-programs that depend on each other), the runner surfaces this as an error requiring f to be extended. The model designer must decide: which program runs first, or do they iterate, or are they synchronized to different clocks?

7 The UM Explorer

The connectome perspective enables a powerful interactive tool: the *UM explorer*, a web-based viewer for stepping through a model's execution on data.

7.1 Capabilities

Given a model $u = (E, P, t, f)$ and a dataset:

- **Step through**: advance one byte at a time, watching the thought vector t evolve. See which events have support, which patterns fire, how support propagates through the connectome.
- **Query by Boolean expression**: ask "show me all positions where $e_{\text{in\_tag}}$ AND NOT $e_{\text{after\_eq}}$" and see the matching positions highlighted in the data.
- **Query by truth assignment**: set specific events to specific support values and run f forward to see what the model predicts. This is model interrogation: "if the model believed we were inside a tag and the word length were 3, what would it predict?"
- **Query by support pattern**: "show me all positions where $\text{ES}_{\text{word\_len}}$ has support > 100 for event w_5" — find the positions where the model is confident about a specific context.
- **Oversupport highlighting**: at each position, color-code by surprise level. Red for high oversupport (model confidently wrong), green for correct predictions, gray for undersupport (model uncertain).
- **Connectome visualization**: show the ES-level graph with support values flowing through it in real time as you step through the data.

7.2 Architecture

The UM explorer shares its core with the UM runner—the same f and ω implementations—but adds an interactive layer:

- The runner processes data sequentially, accumulating statistics. The explorer pauses at each step, allowing inspection and querying.
- Boolean queries compile to support-vector masks: for each position in the dataset, check whether the query predicate holds on the thought vector at that position.
- The web viewer presents the data as a character grid (as in the context-events explainer) with overlaid ES support tracks, queryable in real time.

- For practical use, the explorer operates on subsets of the data (first 1M bytes, specific byte ranges, or positions matching a query). Full enwik9 exploration requires the runner's batch statistics; the explorer adds drill-down capability.

7.3 Acceleration of P-Programming

The explorer transforms P-programming from a write-run-analyze cycle into an interactive conversation with the model:

1. Score the data with the current model (runner mode).
2. Identify high-oversupport positions (surprise analysis).
3. Open the explorer at those positions. Step through byte by byte.
4. Query: "what context events are active here? What is the model predicting?"
5. Formulate a hypothesis: "the model needs a context event for [feature X]."
6. Test the hypothesis: set the hypothetical context event's support manually, re-run f, see if the prediction improves.
7. Implement the context event as a P-program.

Step 6—manual intervention in the thought vector—is key. It lets the model designer test context events *before implementing them*, dramatically reducing the cycle time from hypothesis to validation.

8 Deeper Implications

8.1 P Is the Model's Knowledge; f Is Its Reasoning

The connectome $G(P)$ encodes everything the model *knows*: which events exist, how they relate, what has been learned from data. The forward pass f encodes how the model *reasons*: in what order it considers its knowledge, how it resolves ambiguity, how it synchronizes with the world.

Two models with the same P but different f have the same knowledge but different reasoning strategies. This is analogous to two people who know the same facts but think in different orders or at different speeds. The distinction is essential: much of neural architecture search is really a search over f (layer ordering, skip connections, attention patterns) holding P's capacity roughly constant.

8.2 The Connectome as a Theory of the Data

A model's connectome is its implicit theory of the data's causal structure. Each edge in $G(P)$ asserts: "this event is causally related to that event." Each ES asserts: "these events form a meaningful group." The connectome's topology—its depth, its cycles, its connected components—reflects the model's understanding of how the data is generated.

Improving the model means improving this theory. Adding a context event means asserting a new causal relationship. Removing a dead pattern means retracting a false claim. The connectome is not just a computational graph; it is a *scientific theory* about the data, expressed in the language of events and patterns.

8.3 f and the Arrow of Time

In a DAG connectome, f is determined: information flows in one direction. Time is implicit in the causal ordering. But when cycles are present, f must choose how time works: does information circulate instantaneously (convergence), once per input (synchronization), or at some other rate?

The RNN's choice—synchronize with the input stream—ties the model's internal time to the data's external time. This is the simplest and most natural choice for sequential data. But for data with multiple temporal scales (bytes, words, sentences, documents), a single clock may be insufficient. Multi-frequency f—different recurrent groups updating at different rates—would let the model maintain state at multiple temporal scales simultaneously.

This connects to the carrier signal concept from the context-events paper: a persistent event that spans multiple input steps, providing temporal context that a single-step forward pass cannot.

8.4 The Limit: When P Fully Determines f

In the limit of a fully acyclic model (every ES depends only on causally prior ESs), P fully determines f. The model has no ambiguity, no cycles, no choices to make about reasoning order. This is the "pure data" limit: all structure comes from learned patterns, none from architectural decisions.

Whether this limit is achievable or desirable is an open question. Biological neural networks are deeply cyclic; the RNN's success depends on its recurrence. Perhaps cycles—and the f-level choices they require—are essential for temporal modeling. Or perhaps sufficiently rich acyclic connectomes (with enough context events) can capture any temporal structure without recurrence.

The answer may depend on the data. For enwik9, the acyclic KN-6 tower achieves 1.784 bpc. The cyclic sat-rnn (synchronized to input stream) achieves 0.079 bpc on 1M bytes but only via temporal recurrence. The gap between them is the value of cycles—or equivalently, the cost of the context events that would be needed to replace them in an acyclic model.

9 Conclusion

The relationship between P, f, and ω in the UM is:

- P defines the connectome: a directed graph on E, encoding the model's knowledge.
- When P is acyclic, f is determined: topological sort gives the layer ordering. f-patterns propagate support; ω-patterns record joint events.
- When P has cycles, f must resolve the ambiguity. The runner surfaces cycles; the model designer specifies f. Input stream synchronization is the RNN's biologically-observed solution.
- Context events extend the connectome by adding depth (safely, without cycles). Composition of P-programs is graph union.
- The UM explorer enables interactive querying of u: step through data, query Boolean expressions, test hypothetical context events, visualize oversupport. This accelerates the P-programming development cycle.

The connectome is more than a computational graph. It is the model's theory of the data's causal structure. Improving the model means improving the theory. The tools we build—runner, explorer, surprise analysis—are instruments for developing and testing these theories, one context event at a time.