Deriving Layers from the Connectome

Claude and MJC

18 February 2026

Abstract

In traditional neural networks, layers are specified explicitly in the architecture. In the Universal Model, *layers emerge from the pattern set.* The total pattern P defines a directed graph on the event space E; topological analysis of this graph determines the causal ordering for the forward pass, identifies recurrent cycles, and reveals the model's implicit architecture. We formalize this derivation and show how it unifies feedforward, recurrent, and attention-like computations as special cases of the same graph structure.

1 The Connectome

Definition 1 (Connectome)**.** *Given a UM instance $u = (E, P, t)$ with event space E, pattern set P, and thought vector t, 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 can propagate to e_j via the forward pass.

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 in the UM runner, 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 is the natural execution order.

2 Deriving Layers

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

This is the key insight: **we do not specify layers; we derive them from P**.
In the UM runner, this means:

1. Parse the pattern set P to build $\bar{G}(P)$

2. Topologically sort the ES-level graph
3. Execute the forward pass layer by layer, applying all patterns within each layer simultaneously
4. After all layers, the output ES contains the prediction distribution

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 $\rightarrow$ context ESs $\rightarrow$ output.

2.2 Recurrent Networks

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 whose patterns include self-connections: $\text{ES}_h \rightarrow \text{ES}_h$ via W_h. In the connectome, this is a self-loop.

For the forward pass with cycles:

- Option A: **Fixed-depth iteration**. Apply the cyclic patterns k times (analogous to unrolling an RNN for k steps). This is what BPTT-50 does.
- Option B: **Convergence**. Iterate until the thought vector stabilizes ($\|t^{(k+1)} - t^{(k)}\| < \epsilon$). This is more principled but may not converge if the dynamics are chaotic.
- Option C: **Temporal unrolling**. The cycle is applied once per input byte, with the support vector persisting across bytes. This is the standard RNN interpretation.

Option C is what the UM runner currently does for context ESs: the hash table persists across bytes, accumulating counts. The "recurrence" is temporal, not within a single forward pass.

3 LPPs as Zero-Weight Patterns

A learned probabilistic pattern (LPP) between ES_a and ES_b can be represented in the connectome as:

A set of atomic patterns $(e_i, e_j, 0)$ for all $e_i \in \text{ES}_a$, $e_j \in \text{ES}_b$.

The zero weight means these patterns do not propagate support during the forward pass. Instead, they *instrument* the two ESs: they declare that joint events between ES_a and ES_b should be recorded.

This is the distinction between:

- **Structural patterns** (weight > 0): propagate support, define the forward pass
- **Recording patterns** (weight $= 0$): define what the learning function ω should observe

In the connectome, zero-weight edges are shown differently (dashed, perhaps). They don't affect the forward pass but they define the model's capacity to learn.

Remark 1 (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). This corresponds to the zero-weight patterns: the connection must be present even if its weight is zero.*

Biology's developmental thinning (massive synapse pruning in adolescence) corresponds to removing zero-weight patterns that were never activated. This reduces the model's memory overhead without losing learned patterns.

In the UM runner, LPPs avoid this problem by using hash tables: we don't pre-allocate all possible joint events, we record them on demand. This is more memory-efficient than biology's approach 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. If ES_c was not previously in the graph, it adds a new vertex and potentially a new layer.

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 the topological sort.*

This is the architectural significance of context events: they literally add depth to the model. A KN-6 model with no context events has depth 8 (input + 6 context + output). Adding a word-length context ES makes it depth 9. Adding a tag-structure ES makes it depth 10.

Each added layer gives the model more representational power—the ability to condition its predictions on progressively richer structural features.

5 The Execution Model for the UM Runner

Given the connectome, the UM runner's forward pass should:

1. **Build** $\bar{G}(P)$ from all registered patterns and LPPs.
2. **Detect cycles**. If $\bar{G}(P)$ is a DAG, proceed with topological sort. If cyclic, identify strongly connected components (SCCs) and collapse them.
3. **Sort** the (possibly collapsed) DAG topologically.
4. **Execute** layer by layer:
   - For non-cyclic layers: apply all patterns in parallel (one max-min pass).
   - For cyclic layers (SCCs): iterate the contained patterns to convergence or fixed depth.
5. **Record**: after the forward pass, update all LPPs (recording patterns) based on the final thought vector.

Currently, the UM runner hardcodes the execution order (KN interpolation chain). Implementing connectome-derived ordering would make the runner truly general: any P-program could be loaded and executed without changing the runner's code.

6 Implications for P-Programming

6.1 Composability

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 automatically determines the correct execution order.

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.

6.2 Debugging

The connectome provides a visual map of the model. For debugging P-programs, we can:

- Visualize the connectome (which ESs connect to which)
- Trace support propagation through the graph
- Identify bottlenecks (ESs with many incoming but few outgoing edges)
- Detect unintended connections (patterns that shouldn't be there)

6.3 The Layer Count Question

How deep should a UM model be? The connectome answers this empirically: add context events until adding more doesn't improve compression. Each context event adds at most one layer. The current KN-6 model has 8 layers; the sat-rnn (as a UM) would have 3 (input, hidden, output) but with a self-loop giving temporal depth.

The conjecture from the context events paper—that every missing prior is a missing context event—suggests that the "right" depth is determined by the number of independent structural features in the data.

7 Conclusion

The relationship between P and f in the UM is:

- P defines the connectome: a directed graph on E.
- The connectome's structure determines f: topological sort gives layer ordering, cycles give recurrence, zero weights give recording (learning).
- Adding context events changes the connectome, adding depth and representational power.
- The UM runner should derive its execution order from P, not from hardcoded layer structure.

This is the architectural principle that makes P-programming scalable: we don't design architectures, we design event spaces and their connections. The architecture emerges.