The Surprise Mechanism: Undersupport and Oversupport in the UM

Claude and MJC

18 February 2026

Abstract

We formalize surprise in the Universal Model as arising from event spaces. Two complementary forms—*undersupport* (no context event fires) and *oversupport* (the wrong context event fires)—exhaust all cases. Both are learning signals for the general learning function ω, which mutates the model in response to surprise. We describe how surprise propagates through the connectome and propose a detection mechanism for the UM runner.

1 Introduction

The UM framework has a forward pass f (max-min update), a pattern set P (the connectome), and a learning function ω that updates (E, P) in response to experience. The forward pass and the connectome are well-developed in our P-programming practice. The learning function is not.

The missing ingredient is *surprise*: the signal that tells ω what to change. Without surprise detection, ω is blind—it cannot know whether the model is performing well or poorly, or where the problems lie.

We argue that all surprise in the UM arises from event spaces, and takes exactly two forms.

2 Two Kinds of Surprise

Definition 1 (Undersupport)**.** *An event space* ES_a *exhibits* undersupport *at time* t *if, after the forward pass, no event in* ES_a *has support above threshold* τ*:*

$$\max_{e \in \mathrm{ES}_a} s_t(e) < \tau$$

The model has no confident hypothesis about the state of ES_a*.*

Definition 2 (Oversupport)**.** *An event space* ES_a *exhibits* oversupport *at time* t *if the event with highest support in* ES_a *does not match the observed outcome:*

$$\arg\max_{e \in \mathrm{ES}_a} s_t(e) \neq e_{\text{observed}}$$

The model has a confident but wrong *hypothesis.*

These two cases are complementary:

- **Undersupport**: the model says "I don't know." This is honest uncertainty. The cost is that the model cannot use context-dependent distributions, falling back to unconditional predictions. In compression terms: the model uses more bits than necessary because it can't narrow the distribution.

- **Oversupport**: the model says "I know" but is wrong. This is actively harmful. The induced prior pushes probability *away* from the true outcome. In compression terms: the model uses *many more* bits because it assigned high probability to the wrong event.

Proposition 1 (Exhaustiveness)**.** *For any event space* ES_a *with a defined observation at time* t*, exactly one of three conditions holds:*

1. Correct support*: the highest-supported event matches the observation.*
2. Undersupport*: no event has sufficient support.*
3. Oversupport*: the highest-supported event doesn't match the observation.*

Case 1 requires no action. Cases 2 and 3 are the learning signals.

3 Surprise at the Output

The simplest and most directly measurable surprise is at the output ES (byte prediction):

$$\text{surprise}_t = -\log_2 P(b_t \mid \text{model})$$

This is exactly the bits-per-byte at position t. High surprise means the model assigned low probability to the observed byte.

For KN-6 on enwik9, the mean surprise is 1.784 bits per byte. But the variance is enormous: some positions have surprise < 0.1 (highly predictable) and others have surprise > 8 (essentially random).

The positions with high surprise are where context events are most needed. They cluster at:

- Tag boundaries ($<$ after text, $>$ before text)
- First characters of unknown words
- Transitions between XML structure levels (indentation changes)
- Rare characters (digits, punctuation after long runs of letters)

4 Internal Surprise

Output surprise tells us *where* the model is failing but not *why*. Internal surprise—measured at intermediate event spaces—localizes the problem.

Definition 3 (Internal Surprise)**.** *For an intermediate event space* ES_a *(not the output), the* internal surprise *is the KL divergence between the predicted and observed support patterns:*

$$S_a(t) = \sum_{e \in \mathrm{ES}_a} s_t^{\text{predicted}}(e) \log \frac{s_t^{\text{predicted}}(e)}{s_t^{\text{observed}}(e) + \epsilon}$$

where $s^{\text{predicted}}$ *comes from the forward pass and* s^{observed} *comes from the actual data.*

For context event spaces, internal surprise has a clear interpretation:

- If $\mathrm{ES}_{\text{word\_len}}$ is undersupported, the model doesn't know the current word length. This means the word-length P-program isn't running or isn't connected.
- If $\mathrm{ES}_{\text{in\_tag}}$ has oversupport (predicts "in tag" but we're actually in text), the tag-detection P-program has a bug or the data has unusual structure.

5 Surprise Propagation

Surprise at the output can be *attributed* to surprise at internal ESs by tracing the connectome backward:

Proposition 2 (Backward Attribution)**.** *If output event e_o has low support and e_o receives input from patterns connected to* ES_a*, then undersupport at* ES_a *is a potential cause of the output surprise.*

This is not backpropagation (no gradients). It is *structural attribution*: following the connectome edges backward from the surprising output event to find which intermediate ESs failed to provide support.

The algorithm:

1. Identify the surprising output event e_o (observed but low predicted support).
2. Find all patterns with e_o as target: (e_i, e_o, w).
3. Check the support of each e_i. If $s(e_i) < \tau$, ES of e_i is a candidate cause.
4. Recurse: check what should have supported e_i.

This traces the "explanation path" for the surprise—which context events failed to fire or fired incorrectly.

6 The Learning Response

Given a surprise signal and its attribution, ω should:

6.1 For Undersupport

1. **If the ES has no context events**: the model lacks the structural feature entirely. Response: add a new context event (the "missing prior = missing context event" conjecture from the context-events paper).
2. **If the ES has context events but none fired**: the context event set is incomplete. Response: add a new event to the ES that covers the unrecognized case.
3. **If the ES has context events but their support is too low**: the detection P-program needs calibration. Response: increase the weight of the relevant pattern.

6.2 For Oversupport

1. **If one context event is always wrong in the same way**: the event's connectivity is too broad. Response: split the event into sub-events (e.g., split "in_tag" into "in_opening_tag" and "in_closing_tag").
2. **If oversupport is rare**: the event is mostly correct but has edge cases. Response: add exception events that override the main context event in specific situations.
3. **If oversupport is systematic**: the event's detection P-program has a bug. Response: fix the P-program.

7 Measuring Surprise in the UM Runner

To implement surprise detection in the UM runner:

1. **Per-position output surprise**: already computed as $-\log_2 P(b_t)$. Store as an array.
2. **Surprise histogram**: bin positions by surprise level. The high-surprise tail is where improvement is possible.
3. **Conditional surprise**: for each context event, compute mean surprise when the event is active vs. inactive. If surprise is *higher* when active, the event is causing oversupport.
4. **Attribution traces**: for the top-k most surprising positions, trace backward through the connectome to identify which intermediate ESs failed.

Implementing items 1–3 in the current UM runner is straightforward (add counters to the scoring loop). Item 4 requires the connectome-derived execution model from the connectome-layers paper.

8 Connection to Compression

Surprise is directly linked to compression:

$$\text{compressed size} = \sum_{t=1}^{N} \text{surprise}_t = \sum_{t=1}^{N} -\log_2 P(b_t)$$

Reducing total surprise by 0.01 bpc across 10^9 bytes saves 10^7 bits ≈ 1.2 MB.

The surprise mechanism tells us *where* those bits can be saved. The context events paper tells us *how* (add context events). The connectome paper tells us *what changes* architecturally when we add them. Together, these three papers form a complete framework for systematic model improvement.

9 What's Missing: ω Itself

We have described the *signal* (surprise) and the *response* (add/split/refine events), but not the *mechanism* by which ω automatically performs these mutations.

In the current UM runner, ω_0 (online counting) updates LPP weights but does not mutate the architecture. The surprise mechanism calls for ω_1 and beyond: learning functions that can:

- Add new events to existing ESs
- Create new ESs
- Add patterns connecting new events to existing ESs
- Adjust pattern weights
- Remove patterns (thinning)

This is the "general-form ω" from the commentary, operating on $u \times u$ (mutating both the data and the model). Building ω_1 is the next major milestone for P-programming.

Conjecture 1 (Surprise Sufficiency)**.** *The surprise signal (undersupport/oversupport at each ES) contains sufficient information for ω to determine the correct model mutation. No additional gradient or second-order information is needed.*

This conjecture, if true, means the UM can learn without backpropagation—using only forward-pass surprise and connectome structure.

10 Conclusion

All surprise in the UM arises from event spaces. Undersupport means the model is uncertain; oversupport means it's wrong. Both are learning signals. Together with context events (which provide the remedy) and the connectome (which provides the attribution path), the surprise mechanism completes the theoretical framework for automated P-programming.

The practical next step is to add surprise measurement to the UM runner and use it to identify where context events should be added.