Eight Experiments: The English Context Neuron

Claude and MJC — February 24, 2026

These eight experiments trace a single thread: discovering the English context neuron from the data, defining subtraction in the UM, and measuring whether it compresses.

1. Experiment 1: Output ES Factoring

1.1. Question

All ESs in our model cluster around the output ES. In what sense do they "factor" through it, and what are the number frequencies?

1.2. Method

Take the trained bigram model (wm-bigram on 1B). For each LPP $L : E_s \to E_o$ (where E_o is the output ES), examine the per-source-event entry counts: how many entries does each source event $e \in E_s$ have in L?

Concretely, for the I→O unigram LPP: every input byte maps to some subset of output bytes. The cardinality of that subset (how many output bytes have been observed after input byte a) is a number $n_a \in \{0, \ldots, 256\}$.

Collect the distribution of these n_a values across all source events and all LPPs. This is the "factoring number" for each source event. Do the same for the bigram LPP (prev→O): each prev-byte event maps to some number of output events.

1.3. Expected outcome

A bimodal or structured distribution: some source events (common letters in English contexts) map to many output events (high fan-out, low information per pattern), while others (rare bytes, XML-specific characters) map to few. The factoring numbers should cluster, revealing natural groupings of source events by their output fan-out. These clusters are the precursors of context neurons.

1.4. What we build

`umr factor-analysis model.sn` — prints per-LPP, per-source-event entry counts, sorted. No new UM structure; this is pure inspection of the trained model.

2. Experiment 2: English Orthography Failure Map

2.1. Question

Where does the current byte-level model (bigram/trigram) fail to predict the output, and do these failures cluster by context?

2.2. Method

Run the trained bigram on full enwik9 in frozen mode. At each position, record the surprise $-\log_2 p(x_t \mid \text{model})$. We already have `umr surprise` for this.

Classify each position by a simple external label: (a) inside an XML tag (`<...>`), (b) inside an XML entity (`&`), (c) at a word boundary (after space/newline), (d) word-internal lowercase, (e) word-internal uppercase, (f) digit, (g) punctuation, (h) other.

Compute the mean surprise for each class. The gap between classes is the signal: if word-internal lowercase has 2.5 bpc and tag-internal has 5.0 bpc, then the model "knows English" but doesn't know XML.

2.3. Expected outcome

Word-internal lowercase: low surprise (∼2–3 bpc). The bigram model captures English letter sequences well. XML tags, entities, digits: high surprise (∼5–7 bpc). The model has no context neuron to switch distributions.

2.4. What we build

`umr surprise-by-class model.sn data [max_bytes]` — extends the existing surprise command with per-class breakdown. The classification is done on the raw bytes (detecting `<`, `>`, `&`, case, etc.), not by adding UM structure. This is observation of where the model fails, which motivates the next experiments.

3. Experiment 3: The English Context Neuron

3.1. Question

Can we define a single context neuron — "English lowercase text" — that, when active, explains the lowercase ASCII frequency table via an LPP to the output ES?

3.2. Method

Create a new ES: `ES ''context'' 2` with events "English" (index 0) and "Other" (index 1). This is a *manually defined* context ES, not discovered — we are testing the concept before automating discovery.

Define activation rules:

- After seeing a lowercase letter followed by another lowercase letter: set `context[English] = 255`.
- After seeing `<`: set `context[Other] = 255`.
- At all other positions: inherit from previous position (shift).

Add an LPP: `context` → `byte_output`. Train on enwik9. When "English" is active, this LPP learns the lowercase letter distribution; when "Other" is active, it learns whatever distribution applies there.

Measure: how much does adding this context neuron improve bpc over the baseline bigram? The improvement comes from the context neuron providing a *better marginal* for positions where it is active, rather than using the unconditional byte marginal.

3.3. Expected outcome

The "English" context covers ∼60–70% of positions (lowercase English text). Its LPP to output learns the English letter frequency table (∼4.2 bpc for unigram English vs 5.0 bpc unconditional). Expected improvement: 0.3–0.5 bpc overall, because the context-conditioned marginal is sharper than the unconditional marginal.

3.4. What we build

A new SN model with the context ES and its LPP, expressed as a standard UM. The activation rules are absolute patterns (context events set by detecting byte patterns). `umr setup english-context` generates this model.

4. Experiment 4: Log-Stochastic Subtraction

4.1. Question

What does "subtracting English from the output" mean concretely in the UM, and does log-stochastic subtraction produce a meaningful residual?

4.2. Method

Subtraction in the UM is not arithmetic subtraction of support values. It is: given a context neuron C with LPP $C \to O$ that explains some of the output distribution, the *residual* is the information in the output that C does *not* explain.

Formally: the context LPP produces a distribution $q(b) = 2^{s_C(b)} / \sum 2^{s_C(b')}$ over output bytes. The actual empirical distribution is $p(b)$. The residual information per position is:

$$r(b) = -\log_2 p(b) - (-\log_2 q(b)) = \log_2 \frac{q(b)}{p(b)}$$

Positions where $r > 0$: the context *over-predicts* byte b (it's less common than the context expects). Positions where $r < 0$: the context *under-predicts* (byte is more common than expected — the context doesn't explain it).

Log-stochastic subtraction: in the UM, this maps to having two LPPs targeting the same output ES: the context LPP and the "everything else" LPP. The sharpest-LPP mechanism selects which one wins at each position. When the context LPP wins, it has "subtracted" the generic marginal and replaced it with the context-specific distribution.

Run Experiment 3's model. At each position, record which LPP won (context vs. baseline) and the surprise under each. The gap = the value of subtraction.

4.3. Expected outcome

At English-context positions, the context LPP wins with lower surprise. At non-English positions, the baseline wins. The per-position "subtraction value" should be positive on average, confirming that context-conditioned prediction is better than unconditional.

4.4. What we build

`umr subtraction-map model.sn data [max_bytes]` — per-position output showing which LPP won, surprise under winner vs. loser, and running subtraction value. This validates the subtraction concept before building more contexts.

5. Experiment 5: Lowercase Frequency Table via Word Events

5.1. Question

Can the English context neuron explain the lowercase ASCII frequency table specifically *through word events*, rather than directly?

5.2. Method

This is the architecture MJC describes: English context → word events → byte output.

The chain is: the English context neuron fires → it activates word events ("the", "and", "of", etc.) → each word event has an LPP to the output ES giving its spelling distribution → the mixture of active word events produces the lowercase frequency table.

Concretely, add:

- `ES ‘‘word’’ N` where N is the vocabulary size (start with top 100 words).
- `LPP ‘‘context’’ ‘‘word’’` — English context activates word events with their frequency-based support.
- `LPP ‘‘word’’ ‘‘byte_output’’` — each word event predicts its next-byte distribution.

Train on enwik9 using the existing 3-pass pipeline. Measure: does the word-mediated chain produce a distribution over output bytes that matches the empirical lowercase frequency table? And does it compress better than the direct context→output LPP from Experiment 3?

5.3. Expected outcome

The word-mediated chain should recover the lowercase frequency table as a mixture of word-specific distributions. It may not compress *better* than the direct LPP at this stage (the direct LPP already captures the marginal). But it provides the foundation for word-level prediction: once word→word LPPs exist, the gain comes from predicting which word follows which.

5.4. What we build

A new SN model extending Experiment 3 with a word ES and two LPPs. The word ES is populated during training via threshold creation at word boundaries (existing mechanism from #lexicon_path_paper). This is the first model with three levels: context → word → byte.

6. Experiment 6: Uppercase as Separate Context

6.1. Question

Does adding a second context neuron for uppercase text (sentence starts, proper nouns, acronyms) provide independent compression gain?

6.2. Method

Extend the context ES from Experiment 3: `ES ''context'' 3` with events "English_lower", "English_upper", "Other".

Activation:

- After space + uppercase letter: `context[English_upper] = 255`.
- After lowercase letter in a lowercase run: `context[English_lower] = 255`.
- After `<` or other non-English byte: `context[Other] = 255`.

The uppercase context has its own LPP to output, learning a different distribution. In English Wikipedia, uppercase appears at: sentence starts (∼equal to lowercase frequency but shifted), proper nouns (names, places — different letter frequencies, e.g., more capital letters), acronyms (all-caps, very different distribution).

Measure: (a) bpc improvement over Experiment 3 (which has only English/Other). (b) The learned uppercase distribution vs. the lowercase distribution — how different are they?

6.3. Expected outcome

Small but measurable gain (∼0.05–0.1 bpc) from separating uppercase. The uppercase frequency table differs from lowercase: more 'T' ("The"), more 'A' ("A", articles), different bigram structure. The gain is modest because uppercase is only ∼5% of positions, but it validates that multiple context neurons each contribute independently.

6.4. What we build

Extended SN model with 3-event context ES. Same pipeline as Experiment 3 but with refined activation rules. `umr setup english-context-2` generates this model.

7. Experiment 7: XML Tag Open as Context Activation

7.1. Question

Does "<" (tag open) work as the activation signal for switching away from English context, and what distribution does the non-English context learn?

7.2. Method

This is the explicit test of MJC's note: "activated by things like xml tag open." The context neuron is activated (English) or deactivated (Other) by specific byte patterns.

Using Experiment 3's model, instrument the context switches:

- Record every position where context switches from English to Other (the "deactivation" signal, triggered by <).
- Record every position where context switches from Other to English (the "reactivation" signal — what byte pattern triggers this? Likely > followed by text).
- Measure the surprise at switch points vs. non-switch points.

Additionally, examine what the "Other" context LPP learns. Inside XML tags, the byte distribution is very different from English: tag names (`title`, `text`, `id`), attributes, `=`, `"`, `/`, digits. This is a discoverable structure.

7.3. Expected outcome

The < signal cleanly separates English from XML. Context switch points have elevated surprise (the model hasn't yet incorporated the switch). The "Other" LPP learns a distribution heavy on lowercase tag-name letters plus /, >, =, " — visibly different from the English distribution. This confirms that a single bit of context (English vs. not) is highly informative.

7.4. What we build

`umr context-switches model.sn data [max_bytes]` — reports context switch locations, surrounding bytes, surprise at switches. Validates that the activation mechanism (tag open/close) is the right signal.

8. Experiment 8: One-Context-at-a-Time Learning

8.1. Question

Can we discover context neurons sequentially — learn one context, subtract it, find the next from the residual — without an external clustering algorithm?

8.2. Method

This is the learning protocol MJC describes: "we will cluster probably not by using an algorithm but just by learning one context at a time."

Round 1: Start with a blank model (unigram only). Train on enwik9. The unigram learns the unconditional byte frequency table. Write a trace. Measure bpc.

Round 2: Examine the residual (high-surprise positions). Hypothesize a context: "English lowercase" (the most obvious signal). Add the context neuron and its LPP (as in Experiment 3). Retrain. Write a new trace. Measure improvement.

Round 3: Examine the *new* residual. The English context has been "subtracted." What remains? Likely: XML structure, digits, uppercase patterns. Hypothesize the next context: "XML tag." Add it. Retrain. Measure.

Round 4: Continue. Each round adds one context, subtracts it, and examines what's left.

The key constraint: **no algorithmic clustering**. Each context is hypothesized from inspecting the residual (via `umr surprise` and manual examination of high-surprise positions), then added as a single neuron with manually defined activation. The UM discovers frequencies; the human discovers contexts.

8.3. Expected outcome

A sequence of contexts discovered in order of information value: (1) English lowercase (∼0.3–0.5 bpc gain), (2) XML tags (∼0.1–0.2 bpc), (3) digits/numbers, (4) uppercase English, (5) XML entities (`&` etc.), ... Each round's gain is smaller as the easy contexts are subtracted first. The cumulative gain should approach the gap between unigram and bigram (∼1.3 bpc), validating that context neurons explain most of the bigram's advantage.

8.4. What we build

A documented protocol, not a program. Each round produces: (a) an SN model, (b) a trace file, (c) a surprise map, (d) a one-paragraph hypothesis for the next context. The protocol is the Tock step applied manually: inspect residual → hypothesize structure → add it → Tick (retrain + re-encode).

Dependencies: Experiments 1–2 are pure observation (can run now on the trained bigram). Experiment 3 defines the core concept. Experiments 4–7 each extend Experiment 3 in a different direction. Experiment 8 is the meta-protocol that subsumes all others.