Pattern Injection: UM → RNN

Hutter RNN Project

2026-01-31

1 The Isomorphism

We have two representations:

$$\text{UM} = (E, T, P, f, \omega) \tag{1}$$
$$\text{RNN} = (W_{ih}, W_{hh}, W_{ho}, b_h, b_o) \tag{2}$$

Tock (extraction): Read patterns from trained weights.
Tick (injection): Write patterns into weights, skip training.

2 The Mapping

2.1 Forward: RNN → Predictions

$$h_t = \tanh(W_{ih}x_t + W_{hh}h_{t-1} + b_h) \tag{3}$$
$$o_t = W_{ho}h_t + b_o \tag{4}$$
$$P(e_{t+1}|h_t) = \text{softmax}(o_t) \tag{5}$$

2.2 UM Patterns as Target

We have pattern matrix $P \in \mathbb{R}^{|E|\times|E|}$:

$$P[e_1, e_2] = \log \text{count}(e_1 \to e_2) - \log \text{count}(e_1)$$

This is the log-probability $\log P(e_2|e_1)$.

2.3 The Injection Problem

Goal: Set weights such that:

$$W_{ho}h_{e_1} + b_o \approx P[e_1, :]$$

where h_{e_1} is the hidden state after seeing e_1.

3 Simple Case: 1-Markov

For a 1-Markov model (only previous byte matters):

3.1 One-Hot Encoding

If x_t is one-hot for byte e_1, we want:

$$o_t = P[e_1, :] \quad \text{(the } e_1 \text{ row of pattern matrix)}$$

3.2 Direct Injection

Set:

$$W_{ih} = I \quad \text{(identity, or encoding matrix)} \tag{6}$$
$$W_{hh} = 0 \quad \text{(no recurrence for 1-Markov)} \tag{7}$$
$$W_{ho} = P^T \quad \text{(pattern matrix, transposed)} \tag{8}$$
$$b_o = 0 \tag{9}$$

Then:

$$h_t = \tanh(x_t) \approx x_t \quad \text{(for one-hot)} \tag{10}$$
$$o_t = P^T x_t = P[e_1, :] \tag{11}$$

Result: Network outputs correct bigram probabilities with no training.

4 General Case: Beyond 1-Markov

4.1 The Hidden State Problem

For k-Markov or full context:

- Hidden state h_t encodes context $(e_1, \ldots, e_t)$
- We need: $W_{ho} h_t = P[\text{context}, :]$
- But h_t is a learned representation

4.2 Factored Injection

Factor the pattern matrix through a bottleneck:

$$P \approx UV^T$$

where $U \in \mathbb{R}^{|E| \times H}$ and $V \in \mathbb{R}^{|E| \times H}$.
This is like SVD or matrix factorization.
Set:

$$W_{ih} = U \quad \text{(input embedding)} \tag{12}$$
$$W_{ho} = V^T \quad \text{(output embedding)} \tag{13}$$

4.3 The Recurrence Problem

What about W_{hh}? This encodes how context evolves.

For ES-augmented model:

$$W_{hh} = \text{block structure respecting ES}$$

Inject ES transition patterns into the recurrence.

5 Practical Algorithm

5.1 Step 1: Compute Patterns from Data

```
for each (e1, e2) in corpus:
    count[e1][e2]++
P[e1, e2] = log(count[e1][e2] / sum(count[e1]))
```

5.2 Step 2: Factorize

```
U, S, V = svd(P)
U_h = U[:, :H] * sqrt(S[:H])
V_h = V[:, :H] * sqrt(S[:H])
```

5.3 Step 3: Inject

```
W_ih = U_h      # input projection
W_ho = V_h.T    # output projection
W_hh = ?        # harder - need recurrence structure
b_o = row_means(P)  # bias = marginal
```

5.4 Step 4: Fine-tune

Train normally, but starting from injected weights.

6 What This Buys Us

Benefit	How
Faster convergence	Start near good solution
Interpretable init	Know what patterns are encoded
Transfer learning	Inject patterns from other data
Debugging	Inject known patterns, verify behavior
Hybrid models	Mix learned + injected patterns

7 The ES Advantage

For ES-augmented model:

1. Compute ES-level patterns: $P_{ES}[es_1, es_2]$
2. Compute within-ES patterns: $P_w[e|es]$
3. Inject ES patterns into ES-specific weights
4. Inject within-ES patterns into within-ES weights

The factored structure makes injection cleaner.

8 Open Questions

1. How to inject recurrence patterns (W_{hh})?
2. What's the right factorization rank H?
3. How much does injection help vs. random init?
4. Can we inject higher-order patterns (k-grams)?

9 Summary

```
    TOCK                    TICK
(extraction)            (injection)

RNN weights  >  UM patterns
     W                      P

RNN weights  <  UM patterns
     W                      P
```

The isomorphism goes both ways. We can:

- Read patterns out (interpretability)
- Write patterns in (initialization)