A Note for MJC: Where the Entropy Comes From

Graf
Hutter Archive, 12 March 2026

Abstract

This is an informal note meant to make one point cleanly: when we say a model distribution has entropy, that entropy is the average amount of unbiased randomness a sampler has to spend to realize outcomes from that distribution. In the UM, that is not an alien add-on. It is the count algebra, seen through sampling and code length.

1 The basic picture

A PRNG gives us approximately uniform bits. By itself it has no opinion about what outcome should happen. It just gives a stream of almost-fair coin flips.

A model distribution $P(x)$ is what turns those uniform bits into a non-uniform outcome. If an event has probability $1/2$, it costs about one random bit to choose it. If it has probability $1/8$, it costs about three random bits. In general the cost is

$$-\log_2 P(x).$$

So the entropy

$$H(P) = -\sum_x P(x) \log_2 P(x)$$

is just the *average* number of unbiased bits the sampler has to use.

That same quantity is also the average optimal code length. So there are really two readings of the same number:

- sampling view: how many PRNG bits do I spend on average?
- compression view: how many code bits do I spend on average?

Same quantity, two operational interpretations.

2 How it appears in the UM

In the UM the order of construction is not:

> first probability, then maybe counts.

It is the other way around.

We start with counts, supports, and patterns in event space. From those we get conditional probabilities like

$$P(o \mid i) = \frac{c(i,o)}{c(i,\cdot)}.$$

Once we have a probability, we immediately also have:

$$Q = \frac{1}{P}, \qquad \Lambda = -\log_2 P = \log_2 Q.$$

That means luck, quotient, and code length are all the same structure seen in different coordinates.

Then entropy is just the expectation of that log-luck:

$$H = \mathbb{E}[-\log_2 P] = \mathbb{E}[\log_2 Q].$$

So from the UM point of view, entropy is not mysterious. It is the average log-luck induced by the count-derived distribution.

3 What the PRNG is actually doing

This is the part I wanted to make maximally concrete.

Suppose the UM has produced a next-byte distribution. To *sample* from that distribution, we feed in uniform random bits from the PRNG until the sampling procedure can decide which outcome region those bits landed in.

The PRNG is not contributing semantic structure. The UM contributes the structure. The PRNG contributes fungible randomness. Entropy is the rate at which that fungible randomness gets consumed.

That is why it is misleading to talk as if entropy were a property of the PRNG itself in this context. The PRNG only supplies generic uncertainty. The model distribution determines how much of it is needed, on average, to realize the modeled process.

4 Why quotienting makes the budget split

Once you partition events into classes $[e]$, the random-bit budget splits in the obvious operational way:

1. first use randomness to choose the class;
2. then use randomness to choose the member inside the class.

That is exactly why MCP graf 14 has

$$H(e) = H([e]) + H(e \mid [e]).$$

This is not only the entropy chain rule in abstract notation. It is also a statement about the sampler: the PRNG budget decomposes into a coarse step and a residual step.

The same decomposition is also the code-length split:

1. code which bucket you are in;
2. code which item inside the bucket.

So quotienting, sampling, and compression are all showing the same additive structure.

5 The bridge I think matters

For me the clean bridge is:

> counts → probabilities → luck/quotient → code length/entropy.

Then the PRNG interpretation slots in naturally:

> entropy = average amount of unbiased randomness required to realize the count-derived distribution.

So if we want to say it in one compact line:

> The UM does not start with entropy. It starts with counts. Entropy appears when those counts are read operationally as either sampling cost or coding cost.

6 One caution

There is a temptation to say "the PRNG adds entropy." I would phrase it more carefully.

The PRNG supplies random bits. The *distribution* determines the amount of entropy consumed per sample on average. So what gets "added" is not structure but random fuel. The structure comes from the UM's count algebra.

That phrasing seems less likely to confuse the source of randomness with the source of organization.

7 Sorted outcomes and interval carving

Assume now that we permute the event labels so that the outcomes are in decreasing probability order:

$$P(e_1) \geq P(e_2) \geq \cdots \geq P(e_n).$$

So e_1 is the most common event, e_2 the next most common, and so on. This does not change the entropy. It only makes the geometry of sampling easier to see.

Take a uniform random number $u \in [0, 1)$ from the PRNG. The distribution carves the unit interval into consecutive pieces:

$$[0, P(e_1)), [P(e_1), P(e_1) + P(e_2)), \ldots$$

and we output the unique event whose interval contains u. Because the interval for e_1 is largest, it gets chosen most often.

In that picture, entropy is the average number of unbiased binary decisions needed to locate which interval u fell into. Large early intervals are cheap to identify; tiny late intervals require more bits of refinement. The quantity

$$-\log_2 P(e_i)$$

is exactly the ideal bit budget for isolating event e_i.

The permutation is useful because it separates two things:

- combinatorial order: which event is rank 1, rank 2, rank 3, ...
- metric size: how much interval length each rank receives

The PRNG supplies a point in the unit interval. The UM-derived distribution decides how much territory each ranked event gets.

This also makes clear why entropy is not about names of events but about the shape of the partition. If we rename events but keep the same multiset of probabilities, we get the same interval lengths and hence the same average PRNG-bit budget.

In UM terms, the count algebra determines the sizes of those intervals. Sorting just makes the induced partition easier to reason about: common events occupy large contiguous regions, rare events occupy tiny ones, and the entropy is the average refinement cost of landing inside the correct region.

8 Cumulative mass and rank

With the outcomes sorted this way, define the cumulative masses

$$F_k = \sum_{i=1}^{k} P(e_i), \qquad F_0 = 0.$$

Then event e_k corresponds exactly to the interval

$$[F_{k-1}, F_k).$$

So sampling from the distribution means:

1. draw a uniform $u \in [0, 1)$;
2. output the unique rank k such that $F_{k-1} \leq u < F_k$.

This makes the role of the permutation especially clear. The random source is still uniform. All the non-uniformity is now in the cumulative partition

$$0 = F_0 < F_1 < \cdots < F_n = 1.$$

The sampler is just locating which cumulative threshold the PRNG draw crossed.

If we think in terms of rank instead of event name, then the entropy is the average refinement cost of discovering the sampled rank under this cumulative partition. High ranks are rare because their intervals are tiny. Low ranks are common because they claim most of the unit interval early.

This is another way to say that entropy depends only on the probability shape, not on semantic labels. Once the events are sorted, the probability law is a partition of $[0, 1)$ by cumulative thresholds, and the PRNG merely picks a point to be resolved against that partition.

9 Bottom line

If we sample from a UM-derived conditional distribution, the entropy is the average PRNG-bit budget of that sampling process. If we compress with that same distribution, the entropy is the average code-length budget. In the UM, both budgets come from the same place: count structure pushed through probability into log-luck.

That is the whole note.