Likelihood as Vanishing Mass

A short note on log-likelihood, information loss, and finite physical computation

J. R. Landers

July 9, 2026

Main framing and innovation

The familiar computational reason for using log-likelihood is that products of many small probabilities can underflow, while sums of log-probabilities usually remain stable. This note reframes that fact as an information-retention statement. Probability-space represents accumulated evidence as exponentially vanishing mass,

$$L_n = 2^{-S_n},$$

while log-space represents the same evidence as additive surprisal,

$$S_n = \sum_{i=1}^{n} -\log_2 p_i.$$

In exact mathematics these are equivalent. Under finite physical computation, however, probability-space becomes a lossy encoding: once the likelihood falls below the smallest representable positive number, distinct information states collapse into the same machine state, namely zero. The main invariant here is a retained information-mass identity: log-space preserves total normalized information mass equal to one, while likelihood-space preserves only the prefix that fits before underflow.

1 Likelihood and log-likelihood

Let

$$x_1, \ldots, x_n$$

be independent observations, and let

$$p_i = f(x_i \mid \theta), \qquad 0 < p_i < 1,$$

be the likelihood contribution of observation i. The ordinary likelihood is

$$L_n(\theta) = \prod_{i=1}^{n} p_i.$$

The log-likelihood is

$$\ell_n(\theta) = \sum_{i=1}^{n} \log p_i.$$

The usual algebraic point is simple:

$$\log L_n(\theta) = \log \prod_{i=1}^{n} p_i = \sum_{i=1}^{n} \log p_i.$$

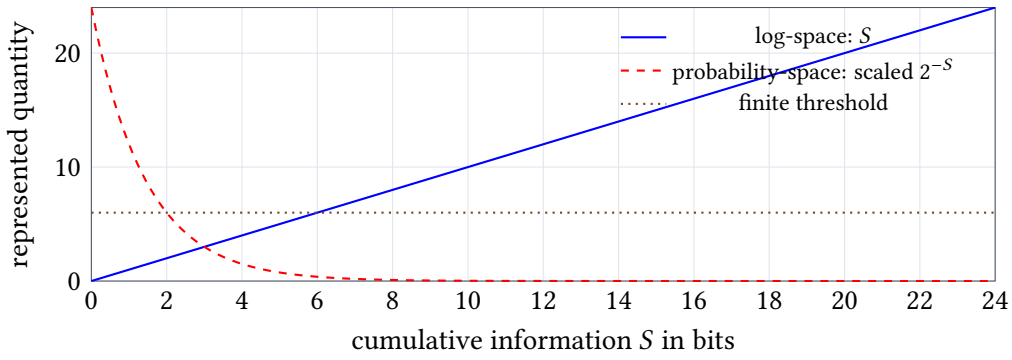

Figure 1: Log-space keeps cumulative information in additive coordinates. Probability-space sends the same information through an exponential collapse. The dashed curve is scaled only to make both curves visible on one axis.

Products become sums. This is often introduced as a convenience. But from the point of view of finite computation, it is deeper than convenience. The two formulas place the same evidence in very different coordinates.

Likelihood-space expresses evidence as shrinking probability mass. Log-space expresses evidence as accumulating information. One coordinate system collapses exponentially; the other grows additively.

2 Evidence as surprisal

Use base-2 logarithms so that information is measured in bits. Define the pointwise surprisal of observation i by

$$s_i = -\log_2 p_i.$$

Equivalently,

$$p_i = 2^{-s_i}.$$

The cumulative information burden of the sample is

$$S_n = \sum_{i=1}^{n} s_i.$$

Therefore

$$L_n(\theta) = \prod_{i=1}^{n} p_i = \prod_{i=1}^{n} 2^{-s_i} = 2^{-\sum_{i=1}^{n} s_i} = 2^{-S_n}.$$

This is the central geometric distinction:

information accumulates linearly $\quad \boxed{S_n = \sum_{i=1}^{n} s_i,}$

while

probability mass collapses exponentially $\quad \boxed{L_n = 2^{-S_n}.}$

The figure is not meant to be mysterious. It says exactly what the formulas say. If S grows by one bit, probability mass is cut in half. If S grows by one hundred bits, probability mass is divided by 2^{100}. The information grows steadily; the probability representation falls off a cliff.

3 Finite representation and underflow

Let $\alpha > 0$ denote the smallest positive representable floating-point number in a chosen finite-precision system. Define the corresponding underflow threshold in bits by

$$\tau_2 = -\log_2 \alpha.$$

A raw likelihood underflows when

$$L_n(\theta) < \alpha.$$

Since $L_n(\theta) = 2^{-S_n}$, this is equivalent to

$$2^{-S_n} < \alpha,$$

or

$$S_n > \tau_2.$$

Thus a probability-space likelihood computation has a finite information budget. It can represent at most about τ_2 bits of cumulative surprisal before the likelihood collapses to zero.

To isolate the mechanism, define a simplified underflow projection

$$Q_\alpha(z) = \begin{cases} z, & z \geq \alpha, \\ 0, & 0 < z < \alpha. \end{cases}$$

Then finite likelihood-space computation follows the recursion

$$\widetilde{L}_i = Q_\alpha(\widetilde{L}_{i-1} p_i), \qquad \widetilde{L}_0 = 1.$$

Once $\widetilde{L}_i = 0$, zero becomes absorbing:

$$\widetilde{L}_{i+1} = Q_\alpha(0 \cdot p_{i+1}) = 0.$$

Log-space follows a different recursion:

$$S_i = S_{i-1} + s_i, \qquad S_0 = 0.$$

There is no absorbing zero state. The evidence continues to accumulate in its natural additive coordinate.

Conceptual point

Probability theory itself is not lossy. In exact real arithmetic, $L_n = 2^{-S_n}$ and $S_n = -\log_2 L_n$ carry the same information. The loss appears when exact probability mass is forced into a finite physical representation. Then sufficiently small positive values are no longer distinct values. They are all the same machine state: zero.

4 Probability-space as a lossy encoding

The exact map

$$S \mapsto 2^{-S}$$

is one-to-one over exact positive real arithmetic. But the finite-computation map

$$S \mapsto Q_\alpha(2^{-S})$$

is not one-to-one. If

$$S_a > \tau_2 \qquad \text{and} \qquad S_b > \tau_2,$$

then

$$Q_\alpha(2^{-S_a}) = Q_\alpha(2^{-S_b}) = 0,$$

even when

$$S_a \neq S_b.$$

Thus distinct information states collapse into one machine state.

Proposition 1: Finite probability-space is lossy

Let $\alpha > 0$ be the smallest positive representable probability value and let $\tau_2 = -\log_2 \alpha$. The finite probability-space encoding

$$E_\alpha(S) = Q_\alpha(2^{-S})$$

is many-to-one on the region $S > \tau_2$. Therefore it loses all distinctions among cumulative information burdens beyond the underflow threshold.

Proof. For any $S > \tau_2$, we have

$$2^{-S} < 2^{-\tau_2} = \alpha.$$

Therefore $Q_\alpha(2^{-S}) = 0$. Hence for any two distinct $S_a, S_b > \tau_2$,

$$E_\alpha(S_a) = E_\alpha(S_b) = 0.$$

So E_α is many-to-one on this region. A many-to-one encoding cannot preserve all distinctions among its inputs. □

This is the formal version of the intuition that raw probability-space is a bottleneck. It asks the computer to preserve exponentially tiny differences in mass. Once the mass falls beneath the representable floor, the tail of the evidence is no longer small-but-distinct. It is gone.

5 The retained information-mass invariant

Define the normalized information-mass contribution of observation i by

$$\phi_i = \frac{s_i}{S_n}, \qquad S_n = \sum_{j=1}^{n} s_j.$$

Then

$$0 \leq \phi_i \leq 1$$

and

$$\sum_{i=1}^{n} \phi_i = 1.$$

Thus ϕ_i measures the fraction of the total empirical information burden carried by observation i.

Define cumulative surprisal up to index i:

$$C_i = \sum_{j=1}^{i} s_j.$$

In log-space, every observation's contribution is retained:

$$\phi_i^{\log} = \phi_i.$$

In likelihood-space, observation i's contribution is retained only if the cumulative product has not yet underflowed:

$$\phi_i^{L} = \phi_i \mathbf{1}\{C_i \le \tau_2\}.$$

Proposition 2: Retained information-mass invariant

The normalized information-mass loss at observation i is

$$\boxed{\Delta_i = \phi_i \left(1 - \mathbf{1}\{C_i \le \tau_2\}\right).}$$

Consequently,

$$\boxed{\sum_{i=1}^{n} \phi_i^{\log} = 1}$$

while

$$\boxed{\sum_{i=1}^{n} \phi_i^{L} = \sum_{i:C_i \le \tau_2} \phi_i \le 1.}$$

Proof. By definition,

$$\Delta_i = \phi_i^{\log} - \phi_i^{L}.$$

Substituting $\phi_i^{\log} = \phi_i$ and $\phi_i^{L} = \phi_i \mathbf{1}\{C_i \le \tau_2\}$ gives

$$\Delta_i = \phi_i - \phi_i \mathbf{1}\{C_i \le \tau_2\} = \phi_i \left(1 - \mathbf{1}\{C_i \le \tau_2\}\right).$$

Summing the log-space contributions gives

$$\sum_i \phi_i^{\log} = \sum_i \phi_i = 1.$$

Summing the likelihood-space contributions gives

$$\sum_i \phi_i^{L} = \sum_i \phi_i \mathbf{1}\{C_i \le \tau_2\} = \sum_{i:C_i \le \tau_2} \phi_i \le 1.$$

□

This is the main invariant. Log-space conserves normalized information mass. Likelihood-space may truncate it. The lost mass is not a statistical property of the data-generating process; it is a computational property of the finite representation.

6 Bits lost per observation

Because ϕ_i is normalized, actual bits lost are recovered by multiplying by total surprisal S_n. Define

$$\Delta_i^{\text{bits}} = S_n \Delta_i.$$

Since $\phi_i = s_i/S_n$, Proposition 2 gives

$$\Delta_i^{\text{bits}} = S_n \frac{s_i}{S_n} \left(1 - \mathbf{1}\{C_i \le \tau_2\}\right).$$

Therefore

$$\boxed{\Delta_i^{\text{bits}} = s_i \left(1 - \mathbf{1}\{C_i \le \tau_2\}\right).}$$

Before underflow, observation i loses zero bits. After underflow, the entire surprisal contribution s_i is lost in raw likelihood-space.

The total bit loss is

$$\boxed{\Delta_{\text{total}}^{\text{bits}} = \sum_{i=1}^{n} s_i \left(1 - \mathbf{1}\{C_i \le \tau_2\}\right).}$$

Equivalently,

$$\boxed{\Delta_{\text{total}}^{\text{bits}} = S_n - \sum_{i:C_i \le \tau_2} s_i.}$$

This identity makes the loss literal: it counts the information bits erased by the probability-space computation after the underflow horizon.

7 The underflow horizon and effective sample size

Define the underflow horizon

$$k_\tau = \max\{k : C_k \le \tau_2\}.$$

This is the last observation index before the cumulative likelihood product underflows. The retained likelihood-space information mass is

$$R_L = \sum_{i=1}^{k_\tau} \phi_i,$$

and the lost information mass is

$$M_L = 1 - R_L.$$

Therefore

$$\boxed{M_L = \sum_{i=k_\tau+1}^{n} \phi_i.}$$

In log-space,

$$R_{\log} = 1, \qquad M_{\log} = 0.$$

The gap is

$$\boxed{R_{\log} - R_L = 1 - \sum_{i=1}^{k_\tau} \phi_i.}$$

Corollary: equal-surprisal case

If every observation has the same likelihood contribution $p_i = p$, then

$$s_i = -\log_2 p = s, \qquad S_n = ns, \qquad \phi_i = \frac{1}{n}.$$

The underflow horizon is

$$\boxed{k_\tau = \left\lfloor \frac{\tau_2}{-\log_2 p} \right\rfloor.}$$

Thus the retained likelihood-space mass is

$$\boxed{R_L = \frac{1}{n} \left\lfloor \frac{\tau_2}{-\log_2 p} \right\rfloor,}$$

and the effective likelihood-space sample size is

$$\boxed{n_{\text{eff}}^L = k_\tau.}$$

In this special case, the statement becomes unusually concrete. The observations after k_τ still exist mathematically. But if one insists on raw likelihood-space, their contributions have been projected into zero.

8 Cross-entropy capacity

The average surprisal is

$$\bar{s}_n = \frac{1}{n} \sum_{i=1}^{n} s_i.$$

Since $s_i = -\log_2 f(x_i \mid \theta)$, this is the empirical cross-entropy of the observed sample under the model:

$$\bar{s}_n = -\frac{1}{n} \sum_{i=1}^{n} \log_2 f(x_i \mid \theta).$$

Because

$$S_n = n\bar{s}_n,$$

the underflow condition

$$S_n > \tau_2$$

becomes

$$n\bar{s}_n > \tau_2.$$

Thus raw likelihood-space underflows when

$$\boxed{n > \text{rac}\tau_2 \bar{s}_n.}$$

This gives a capacity interpretation:

$$\boxed{C_L = \tau_2 \text{ bits.}}$$

Probability-space can carry only τ_2 bits of cumulative surprisal before underflow. The excess burden is

$$\boxed{E_n^{\text{excess}} = (S_n - \tau_2)_+ .}$$

High-surprisal observations consume the probability-space budget faster. Low-surprisal observations consume it more slowly. Either way, once the budget is spent, probability-space collapses to zero.

9 Why the loss is physical

The phrase "finite computation" is not merely a software detail. A computer is a physical system. To preserve a distinction between two mathematical values, the machine must instantiate that distinction as distinguishable physical states. Finite memory, finite energy, thermal noise, and finite precision impose a finite number of reliable distinctions.

This is the sense in which the loss is physical. Probability-space asks the machine to distinguish exponentially tiny masses:

$$2^{-800}, \quad 2^{-1000}, \quad 2^{-100000}.$$

In exact arithmetic, these are different numbers. In finite floating-point probability-space, they may all become

$$0.$$

That collapse is a many-to-one map. Many-to-one maps erase distinctions. In the thermodynamics of computation, irreversible erasure is not free: Landauer's principle relates the erasure of one bit to a minimum heat cost of $k_B T \ln 2$ under idealized thermal conditions [1, 2].

The point is not that likelihood underflow directly computes a heat bill. The point is more basic: information distinctions are physical distinctions. A finite physical machine cannot preserve arbitrary distinctions among vanishing probability masses. When it maps a continuum of tiny positive values to zero, it has traded mathematical distinction for finite physical feasibility.

Energy-conservation framing

Finite energy means finite distinguishability. To keep two values distinct, a physical computer must realize two distinguishable states. Probability-space spends those distinctions exponentially fast, because cumulative information S is encoded as 2^{-S}. Log-space spends them linearly, because it represents S directly.

This is why log-likelihood feels like the natural coordinate system. It does not fight the physics by asking a finite register to preserve exponentially vanishing mass. It keeps the computation in the additive geometry of information.

10 A small numerical example

Suppose

$$p_i = 10^{-3}$$

for each observation. Then

$$s_i = -\log_2(10^{-3}) \approx 9.966 \text{ bits}.$$

For $n = 1000$, total surprisal is

$$S_n \approx 9966 \text{ bits}.$$

In IEEE double precision, the smallest positive normal number is approximately 2^{-1022}, while subnormal values extend the floor further; the exact threshold depends on which underflow convention one uses [5, 6].

Using the normal threshold as a simple benchmark gives

$$\tau_2 \approx 1022 \text{ bits}.$$

The underflow horizon is approximately

$$k_\tau \approx \left\lfloor \frac{1022}{9.966} \right\rfloor = 102.$$

Thus likelihood-space retains roughly

$$R_L \approx \frac{102}{1000} = 0.102$$

of the normalized information mass, while log-space retains

$$R_{\log} = 1.$$

So roughly 89.8% of the normalized information mass lies beyond the raw probability-space underflow horizon. The mathematical likelihood still has meaning. The finite probability-space representation has lost the tail.

11 Interpretation

The classical numerical statement is

$$\prod_i p_i \quad \text{can underflow, while} \quad \sum_i \log p_i \quad \text{is stable.}$$

The information-theoretic statement is stronger:

probability-space represents evidence as vanishing mass,

while

log-space represents evidence as accumulating information.

The finite-physics statement is stronger still:

a finite physical computer cannot preserve arbitrary distinctions among vanishing masses.

Therefore, under finite computation, raw likelihood is a lossy encoding of cumulative information. Log-likelihood is not merely a numerical trick. It is the coordinate system that preserves the information geometry of likelihood under bounded physical representation.

What has been learned

Likelihood and log-likelihood are equivalent in exact mathematics, but not equivalent as finite physical computations. Likelihood-space compresses additive evidence into exponentially shrinking mass, eventually forcing distinct information states through the same representational state: zero. Log-space avoids the collapse by representing evidence in bits, where the sample's information burden accumulates linearly. The retained information-mass invariant makes the difference explicit: log-space conserves the normalized information mass, while probability-space may preserve only the prefix that fits inside the machine's finite range.

Acknowledgment of scope

The numerical-stability principle behind log-likelihood is classical. The contribution of this note is the framing: expressing underflow as a retained information-mass invariant, recovering per-observation bit loss, and interpreting finite likelihood-space as a lossy physical encoding of cumulative surprisal. The thermodynamic discussion is intended as a physical interpretation of finite-state erasure, not as a claim that probability theory itself is lossy.

References

[1] R. Landauer, "Irreversibility and heat generation in the computing process," *IBM Journal of Research and Development*, vol. 5, no. 3, pp. 183–191, 1961.
[2] C. H. Bennett, "The thermodynamics of computation: a review," *International Journal of Theoretical Physics*, vol. 21, pp. 905–940, 1982.
[3] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, 2nd ed., Wiley, 2006.
[4] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, 2nd ed., SIAM, Philadelphia, 2002.
[5] D. Goldberg, "What every computer scientist should know about floating-point arithmetic," *ACM Computing Surveys*, vol. 23, no. 1, pp. 5–48, 1991.
[6] IEEE Computer Society, *IEEE Standard for Floating-Point Arithmetic*, IEEE Std 754-2019, 2019.
[7] D. E. Knuth, *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*, 3rd ed., Addison-Wesley, 1998.