You have all seen this plot

Two-dimensional maps of high-dimensional data are everywhere — single-cell biology, embedding atlases, every "here is our latent space" figure in a paper.

Most of them are UMAP or t-SNE.

Today: what is it actually doing, and *which parts of the picture are you allowed to believe?*

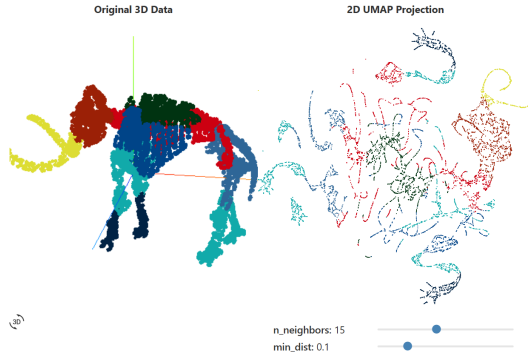


Figure 5: UMAP projections of a 3D woolly mammoth skeleton (50k points, 10k shown) into 2 dimensions, with various settings for the `n_neighbors` *and* `min_dist` *parameters.*

A 3-D elephant and its 2-D UMAP projection. The trunk, legs and ears survive — from the neighbor graph alone.

Source: *Understanding UMAP*, Coenen & Pearce (Google PAIR).

Where we left off

From the last lecture: **Uniform Manifold Approximation and Projection** (**UMAP**) builds a **fuzzy nearest-neighbor graph**, then finds a 2-D layout whose graph matches. Faster than t-SNE, keeps more global structure. That was *what it gives you*.

One thing we did not say, and it is the reason UMAP exists:

t-SNE minimizes $\mathrm{KL}(P\|Q) = \sum p_{ij} \log \frac{p_{ij}}{q_{ij}}$. Look at what that punishes. If p_{ij} is **large** and q_{ij} small — true neighbors torn apart — the term blows up. But if p_{ij} is **near zero**, the whole term is near zero *whatever* q_{ij} *does*. Collapsing two distant points on top of each other is **free**.

So t-SNE guards local structure and leaves global structure unprotected. UMAP's loss pays for both. Everything else today is machinery in service of that.

Outline

The manifold assumption

What UMAP believes about your data before it looks at any of it.

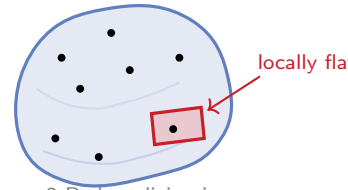
What is a manifold?

A **manifold** is a space that *locally* looks like ordinary flat space, even when its *global* shape is curved.

Earth's surface. Globally a sphere. But the room you are in is flat, and a street map of Yerevan needs no curvature correction.

Face photos. A 64×64 photo is 4096 numbers, but real faces vary along a few knobs — pose, lighting, expression. The photos live on a thin, curved sheet inside that 4096-dimensional space.

The assumption: the data is sampled *uniformly* from a manifold — with respect to a **locally varying** metric.



Why "locally varying" is the whole trick

Predict-first: your data is dense in one region and sparse in another. If you insist the sampling is *uniform* everywhere, what must you conclude about distance?

Why "locally varying" is the whole trick

Predict-first: your data is dense in one region and sparse in another. If you insist the sampling is *uniform* everywhere, what must you conclude about distance?

That your ruler is wrong — and wrong **differently in different places**.

In a **sparse** region, points are far apart in raw distance. If the sampling is really uniform, those gaps must be *shrunk*: the neighbors are closer than they look.

In a **dense** region, points are packed together. Those gaps must be *stretched*: the neighbors are farther than they look.

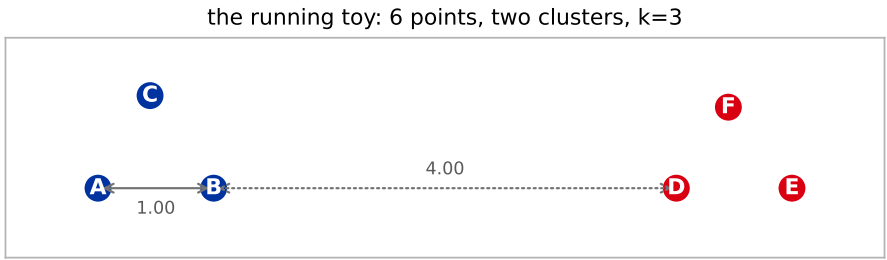
So each point gets its **own** distance scale. That is not a hack bolted on for robustness — it falls straight out of the uniformity assumption, and it is exactly what ρ_i and σ_i compute in the next section. It is also why UMAP copes with varying density where a fixed-radius neighborhood does not.

Step 1 of 2

Turn the data into a weighted graph, using a different ruler at every point.

The running toy

Six points, two clusters, $k = 3$ neighbors. Every number on the next five frames is computed from these coordinates.



Within a cluster, distances are around 0.83 to 1.00. Between clusters, at least 4.0. The apexes (C, F) sit slightly off-centre on purpose — a perfectly symmetric triangle ties two neighbors at the same distance and breaks the σ search on the next frames.

ρ_i: nobody is left alone

$$\rho_i = \min_{j \in \mathcal{N}_i} d(x_i, x_j)$$

the distance to your **nearest** neighbor.

Every distance is then measured *from* ρ_i *outward*: your closest neighbor sits at shifted distance 0, and so gets weight exactly **1**.

Why do this at all? It guarantees every point is connected to something with full strength — even a point stranded in an empty region. Without it, isolated points would detach from the graph entirely and the layout would have nothing to hold them.

This is the "shrink the sparse regions" half of the previous frame.

point	3 nearest	ρ_i
A	C, B, D	0.918
B	C, A, D	0.971
C	A, B, D	0.918
D	F, E, B	0.832
E	F, D, B	0.890
F	D, E, B	0.832

Note every point's third neighbor is *across* the gap — with $k = 3$ in a 3-point cluster it has to be. Watch what the weights do to it.

σ_i: how fast the weight falls off

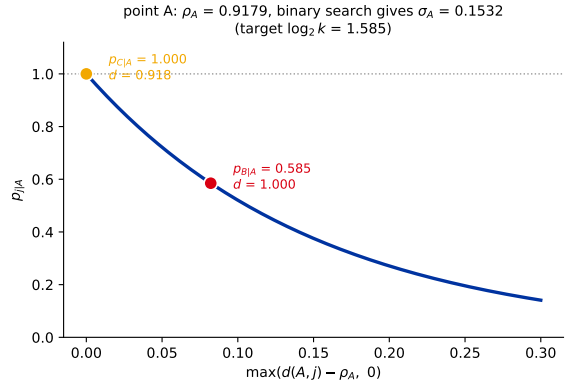
$$p_{j|i} = \exp\left(-\frac{\max(d(x_i, x_j) - \rho_i,\, 0)}{\sigma_i}\right)$$

σ_i is the **local scale**: large σ means the weight decays slowly (a wide neighborhood), small σ means it drops off a cliff.

It is not a hyperparameter. It is solved for, per point, by **binary search** on

$$\sum_{j \in \mathcal{N}_i} p_{j|i} = \log_2 k.$$

Every point ends up with the same "amount of neighborhood" ($\log_2 3 = 1.585$), whatever the local density — the **stretch the dense regions** half



By hand: point A's three weights

With $\rho_A = 0.9179$ and the search returning $\sigma_A = 0.1532$:

neighbor	$d(A,j)$	$d - \rho_A$	$\mathbf{p_{j\|A}}$	
C	0.9179	0	$e^0 = \mathbf{1.000}$	nearest, by construction
B	1.0000	0.0821	$e^{-0.0821/0.1532} = \mathbf{0.585}$	clearly a neighbor
D	5.0000	4.0821	$e^{-4.0821/0.1532} = \mathbf{2.6 \times 10^{-12}}$	a neighbor on paper only

Check the constraint: $1.000 + 0.585 + 0.000 = \mathbf{1.585} = \log_2 3$. ✓

This is how the two clusters separate *without anyone telling UMAP there are two*. D is on A's neighbor list, because with $k = 3$ it has to be. But the exponential gives it a weight of 10^{-12}, so the edge exists on paper and carries nothing.

Symmetrization: whose opinion counts?

$p_{j|i}$ is **directional**. A can think B is a close neighbor while B, sitting in a denser patch, does not return the compliment. UMAP resolves it with a **fuzzy union** (a probabilistic OR):

$$p_{ij} = p_{j|i} + p_{i|j} - p_{j|i}\, p_{i|j}$$

Read it as: the edge is strong if *either* endpoint considers the other a neighbor.

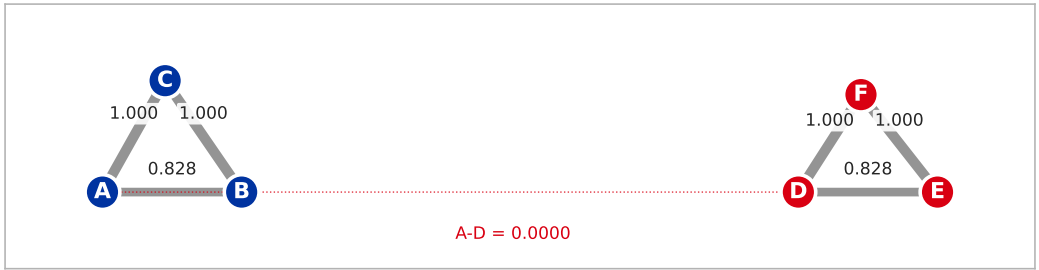
On the toy $p_{B|A} = p_{A|B} = 0.585$, so

$$p_{AB} = 0.585 + 0.585 - 0.585^2 = \mathbf{0.828},$$

stronger than either direction alone. And $p_{AD} = \mathbf{0.0000}$.

The finished graph

the symmetrised graph p_{ij} -- edge width and label = weight



This weighted graph is the entire high-dimensional input from here on. UMAP never looks at the original coordinates again. Whatever the graph failed to capture is gone — so if k was badly chosen, no amount of optimizing can recover it.

Step 2 of 2

Lay the graph out in 2-D, and let attraction and repulsion fight to a draw.

The low-dimensional kernel

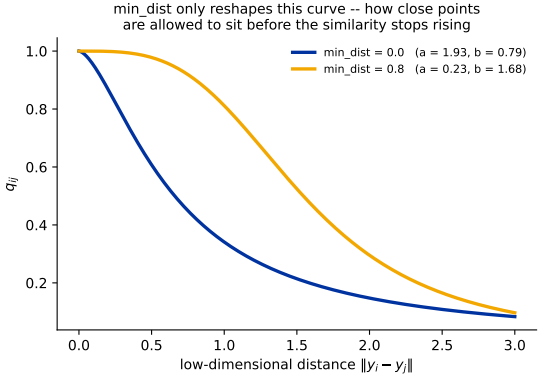
Put the points at positions $y_i \in \mathbb{R}^2$ and define the low-dimensional similarity

$$q_{ij} = \frac{1}{1 + a \, \|y_i - y_j\|^{2b}}$$

a and b are **not** yours to set. UMAP fits them so the curve matches the shape you asked for with `min_dist`:

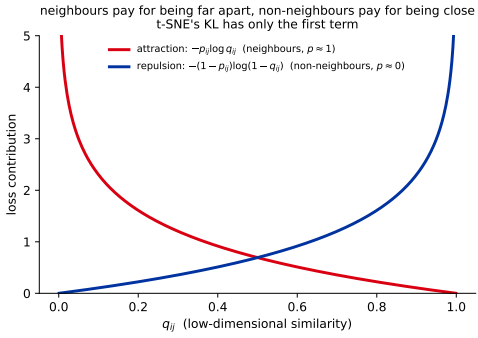
`min_dist`	a	b
0.0	1.933	0.791
0.8	0.232	1.681

A heavy tail, like t-SNE's Student-t: distant pairs are allowed to be genuinely distant without the loss exploding.



The loss: cross-entropy, and its two halves

$$\mathcal{L} = -\sum_{i<j} \Big[\underbrace{p_{ij} \log q_{ij}}_{\text{attraction}} + \underbrace{(1-p_{ij}) \log(1-q_{ij})}_{\text{repulsion}} \Big]$$



Attraction bites when p_{ij} is large and q_{ij} small: true neighbors placed far apart. Pulls them together.

Repulsion bites when p_{ij} is small and q_{ij} large: strangers placed on top of each other. Pushes them apart.

This is the payoff from frame 2. t-SNE's KL has only the first term. Its second term is missing, so nothing in t-SNE's objective objects to two unrelated clusters landing next to each other. UMAP's does — and that is *part* of where the extra global structure comes from. The honest rest of the story is two frames ahead.

Intuition: springs and magnets

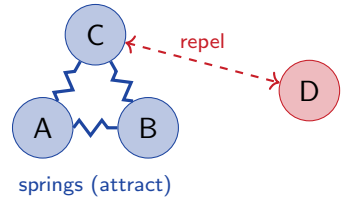
Springs. Every strong edge is a spring pulling its two points together. Strength $\propto p_{ij}$.

Magnets. Every pair repels, gently. Without this the whole layout collapses to a point, which would make the loss's first term perfectly happy.

Solved by SGD, and here is the trick that makes it fast:

- sample **positive** edges in proportion to p_{ij} — so weak edges are mostly skipped;
- for repulsion, do not sum over all pairs. Sample a handful of **random** pairs and push those apart (**negative sampling**).

That is what buys UMAP its scaling. The exact loss is $O(n^2)$ in pairs; negative sampling estimates it in roughly $O(n)$ work per epoch.



Forces balance, and the layout stops moving. That equilibrium *is* the embedding.

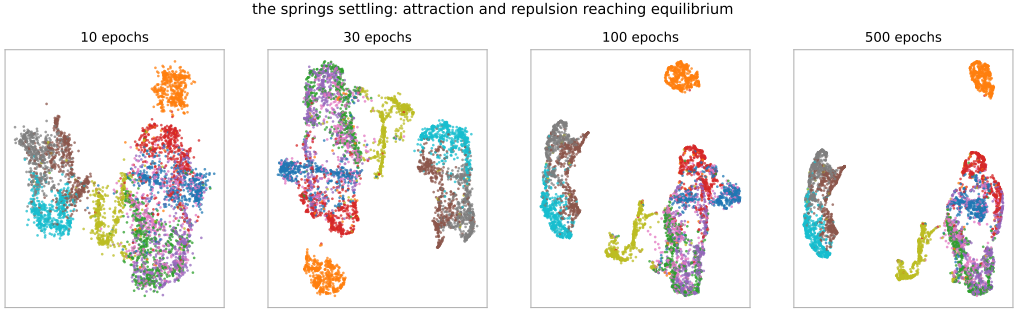
Where does the layout start?

Not at random. UMAP's initial positions are a **spectral embedding** of the fuzzy graph (Laplacian eigenmaps; `init="spectral"`, the default): fast, deterministic, and already placing connected regions near each other. SGD then *refines* that start rather than inventing a layout from noise.

An honest footnote to frame 2. We credited UMAP's global structure to the repulsion term — that is the paper's story. Kobak & Linderman (2021) showed that much of the measured advantage over t-SNE disappears once *both* methods get an informative start: t-SNE initialized with PCA preserves global structure about as well. The loss difference is real; the practical gap is largely this starting point. (sklearn's t-SNE has defaulted to PCA initialization since v1.2 for exactly this reason.)

Moral: **defaults matter as much as objectives** — "which algorithm is better" can hinge on a setting the abstract never mentions.

Watching it settle



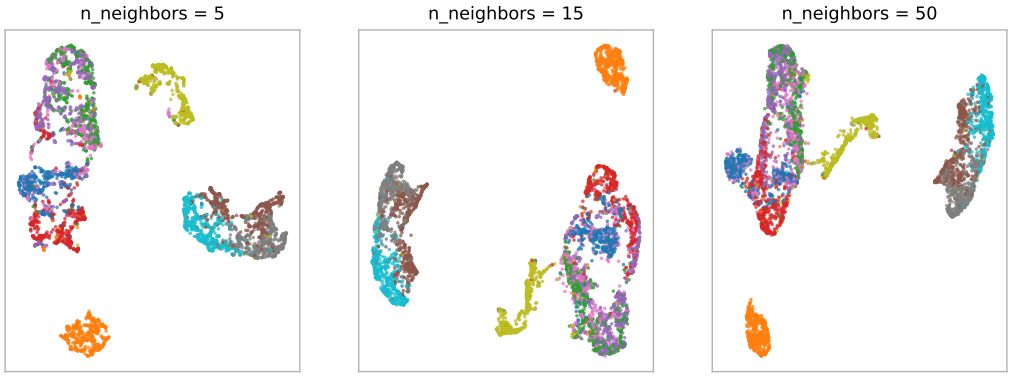
Fashion-MNIST, same data and settings, stopped at increasing epoch counts. The structure appears early — partly because the spectral start already contains it — and then mostly *sharpens*, which is why UMAP gets away with far fewer epochs than you would expect, and why the default (200 for large data, 500 for small) is rarely worth raising.

Using it

Two knobs that matter, and a short list of what the picture is not telling you.

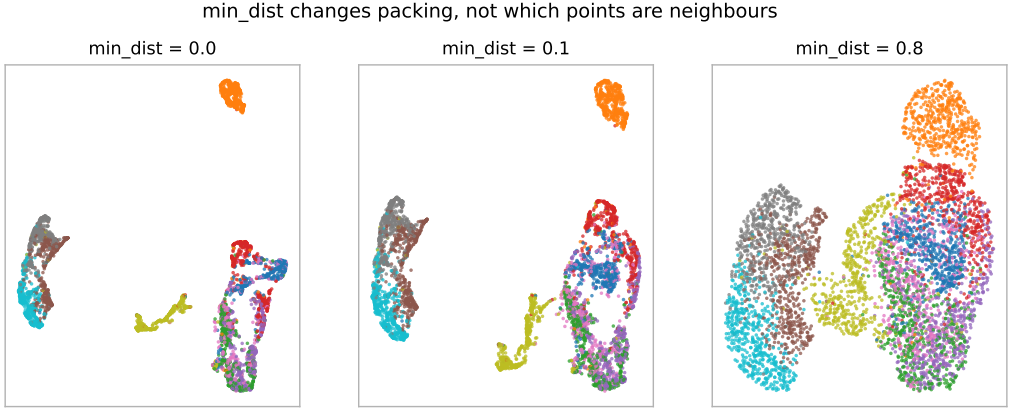
n_neighbors: how far you look

small k sees local detail, large k sees the global shape



It sets k — and therefore how much of the data each point's ruler is calibrated against. **Small** ($\sim$5): fine local detail, many small islands, and a real risk of fragmenting one true cluster into several. **Large** ($\sim$50): each point sees more context, so the layout preserves more of the global arrangement and smooths detail away. Default 15.

`min_dist`: how tightly points may pack



Watch out: `min_dist` is *purely cosmetic in the topological sense*. It changes a and b, so it changes how tightly a cluster is allowed to compress — but it does not change the graph, so it does not change who is whose neighbor. Small values look more "clustered" without the data being any more clustered. Do not read the tightness of a blob as evidence.

Predict-first: same data, same settings, new seed

Nothing changes except `random_state`. How much of the picture survives?

Predict-first: same data, same settings, new seed

Nothing changes except `random_state`. How much of the picture survives?

same data, same settings, three seeds:
the clusters survive, their positions and orientation do not

The **groups** survive. Their **positions, orientation and mirroring** do not. If you describe your result as "cluster 3 sits between cluster 1 and cluster 5", you have described this seed and nothing more.

How to read a UMAP plot

You can trust:

- **Local neighborhoods** — points near each other in high-D usually stay near each other.
- **Cluster membership** — which points group with which.
- **Connected components** — genuinely separated groups stay separated.

You cannot trust:

- **The axes** — they mean nothing; rotation and reflection are arbitrary.
- **Cluster sizes** — visual area is not sample count, and `min_dist` sets it anyway.
- **Between-cluster distances** — "far apart" is not "very different".
- **Small clusters at low `n_neighbors`** — often fragments, not findings.

The test that costs you nothing: rerun with a few seeds and a few `n_neighbors`. If the pattern you are about to write down survives, it is real. If it does not, you were reading the optimizer.

In practice

```python
import umap

reducer = umap.UMAP(
    n_neighbors=15,       # local <-> global
    min_dist=0.1,         # packing only
    metric="cosine",      # cosine for embeddings
    random_state=509,     # fixes the seed (and disables parallelism)
)
Z = reducer.fit_transform(X)
```

- **Standardize first** if your features have different units — UMAP inherits PCA's scale sensitivity through the distance metric.
- **Pick the metric deliberately.** `cosine` for text or CLIP embeddings (an embedding's *direction* carries the meaning; its length picks up frequency and other artifacts), `euclidean` for physical measurements, `hamming` for binary. This matters more than either knob above.
- On wide data, **PCA to $\sim$50 first** — faster, and it strips noise the graph would otherwise take seriously.

The four mistakes people actually make

1. **Wrong metric** — Euclidean on text embeddings. The graph is then built from distances that do not mean what you think, and everything downstream inherits it.
2. **Unstandardized features** — one large-range column dominates every distance, exactly as it hijacks PC1.
3. **Trusting a single run** — see the previous frames.
4. **Feeding the 2-D embedding to a model.** This is the worst one. The embedding threw away almost everything to make a picture. Cluster or classify on the *high-dimensional* data (or on PCA components); use UMAP for your eyes.

Recap

- **The assumption:** data is uniformly sampled from a manifold under a *locally varying* metric. Everything else follows from it.
- ρ_i connects every point to its nearest neighbor at full strength; σ_i is solved per point so all neighborhoods carry the same fuzzy mass $\log_2 k$. Together they give each point its own ruler.
- **Fuzzy union** turns two directed opinions into one undirected edge. The result is a weighted graph, and from there UMAP never sees your data again.
- **Cross-entropy** has an attraction term *and* a repulsion term. t-SNE's KL has only the first — though in practice UMAP's spectral *initialization* does much of the global-structure work (Kobak & Linderman, 2021).
- **Read it carefully:** neighborhoods and membership are real; axes, sizes and between-cluster distances are not.

That closes unsupervised learning (clustering + dimensionality reduction). Next chapter: a new family of models.

References

- McInnes, Healy & Melville (2018). *UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction.* arXiv:1802.03426.
- Kobak & Linderman (2021). *Initialization is critical for preserving global data structure in both t-SNE and UMAP.* Nature Biotechnology.
- Coenen & Pearce, *Understanding UMAP* (Google PAIR) — `pair-code.github.io/understanding-umap/`
- Wattenberg, Viegas & Johnson (2016). *How to Use t-SNE Effectively.* distill.pub — the same lesson for t-SNE.
- van der Maaten & Hinton (2008). *Visualizing Data using t-SNE.* JMLR.