UMAP from classic ML

Notes that I wrote for myself in order to keep the chaos of the overwhelming theoretical foundation for that algo . See the UMAP website

By Amrskk

April 10, 2026

Contents

1 Introduction and Overview

UMAP (Uniform Manifold Approximation and Projection) is a dimensionality reduction algorithm introduced by McInnes, Healy, and Melville (2018). It maps high-dimensional data into a low-dimensional embedding while preserving both the local and, to a significant extent, global topological structure of the data.

1.1 Problem Statement

Given a dataset $X = \{x_1, x_2, \ldots, x_N\} \subset \mathbb{R}^D$ with D large, find an embedding $Y = \{y_1, y_2, \ldots, y_N\} \subset \mathbb{R}^d$ with $d \ll D$ (typically $d = 2$ or 3) such that the essential topological structure of X is faithfully represented in Y.

Intuition

Imagine you have photographs of faces, each represented as a vector of 10,000 pixel intensities (so $D = 10{,}000$). Even though these vectors live in a 10,000-dimensional space, the "true" variation (pose, lighting, identity) has far fewer degrees of freedom — maybe 50 or so. UMAP tries to find a 2D or 3D picture of these faces where faces that look similar are placed nearby, and faces that look different are placed far apart.

1.2 High-Level Pipeline

The UMAP algorithm proceeds in three principal stages:

Stage 1: **Approximate the Riemannian manifold.** Assume data lies on or near a Riemannian manifold. Construct local metric spaces around each data point using k-nearest neighbors and a locally adaptive exponential kernel.

Stage 2: **Build a fuzzy topological representation.** Convert the collection of local metric spaces into a unified fuzzy simplicial set (a weighted graph) that captures the topological structure of the data.

Stage 3: **Optimise a low-dimensional layout.** Find a low-dimensional embedding whose fuzzy topological representation is as close as possible to the high-dimensional one, measured by fuzzy set cross-entropy.

2 Math Preliminaries

This section reviews all the mathematical tools required to understand the theoretical foundations of UMAP rigorously.

2.1 Metric Spaces and Extended Pseudo-Metric Spaces

Definition 2.1 (Metric Space)**.** A *metric space* is a pair (M, d) where M is a set and $d : M \times M \to [0, \infty)$ satisfies:

(i) $d(x, y) = 0 \iff x = y$ (identity of indiscernibles),

(ii) $d(x, y) = d(y, x)$ (symmetry),

(iii) $d(x, z) \leq d(x, y) + d(y, z)$ (triangle inequality).

Intuition

A metric space is just a set of objects with a well-behaved notion of "distance." Think of cities on a map with driving distances: the distance from Almighty City to Astanahood equals the distance from Astanahood to Almighty City (symmetry!!), the distance to yourself is zero (identity!!), and driving Almighty City → Cuckmund via Astanahood is at least as far as driving directly (triangle inequality!!).

Example 2.1. $(\mathbb{R}^n, \|\cdot\|_2)$ with the Euclidean distance $d(x,y) = \sqrt{\sum_i (x_i - y_i)^2}$ is the most familiar metric space. Another example: the set of all English words with $d(\text{word}_1, \text{word}_2) =$ edit distance (number of character insertions, deletions, and substitutions needed to transform one into the other).

Definition 2.2 (Extended Pseudo-Metric Space)**.** An *extended pseudo-metric space* (M, d) replaces the codomain with $[0, \infty]$ (allowing $d = \infty$) and relaxes (i) to $d(x,x) = 0$ (but $d(x,y) = 0$ does not imply $x = y$). This is the natural setting for UMAP's local distance functions because:

- Points far outside a local neighborhood can be considered infinitely far away.
- Distinct points can be "identified" when they share membership weight 1.

Intuition

Think of a person standing in a dense fog. They can measure distances to nearby objects accurately, but anything beyond the fog's reach is effectively "infinitely far." That is the *extended* part ($d = \infty$ is allowed). The *pseudo* part means two distinct objects can have distance zero — like two identical-looking twins standing in the same spot from the observer's perspective; the observer cannot distinguish them.

Example 2.2. For UMAP, each data point x_i defines a local extended pseudo-metric: the distance to its k nearest neighbors is finite and computed from the data, while the distance to all other points is set to ∞. If $k = 3$ and x_i has neighbors $\{x_a, x_b, x_c\}$, then $d_i(x_i, x_a) = 0$ (after subtracting ρ_i), $d_i(x_i, x_b), d_i(x_i, x_c)$ are finite, and $d_i(x_i, x_j) = \infty$ for all other x_j.

2.2 Riemannian Manifolds

Definition 2.3 (Smooth Manifold)**.** A *smooth manifold* $\mathcal{M}$ of dimension n is a topological space that is locally homeomorphic to $\mathbb{R}^n$ and equipped with a smooth atlas of charts.

Intuition

A smooth manifold is a shape that, if you zoom in close enough at any point, looks like ordinary flat space. The surface of the Earth is a 2-dimensional manifold embedded in 3D: standing in a field, the ground looks flat (like $\mathbb{R}^2$), even though globally the Earth is curved. The "smooth" part means there are no sharp corners or tears — you can do calculus everywhere.

Example 2.3. The surface of a sphere $S^2 \subset \mathbb{R}^3$ is a 2-dimensional smooth manifold. A flat sheet of paper is a manifold (trivially, since it *is* $\mathbb{R}^2$). A figure-eight curve is *not* a smooth manifold because the crossing point does not have a well-defined single tangent line.

Definition 2.4 (Tangent Space)**.** At each point p of a smooth manifold $\mathcal{M}$, the *tangent space* $T_p\mathcal{M}$ is a real vector space of dimension $n = \dim(\mathcal{M})$ consisting of all tangent vectors at p.

Concretely, if $\varphi = (x^1, \ldots, x^n)$ is a local coordinate chart around p, then $T_p\mathcal{M}$ is spanned by the coordinate basis vectors $\left\{ \frac{\partial}{\partial x^1}\big|_p, \ldots, \frac{\partial}{\partial x^n}\big|_p \right\}$. Any tangent vector $v \in T_p\mathcal{M}$ can be written as:

$$v = \sum_{i=1}^{n} v^i \frac{\partial}{\partial x^i}\bigg|_p.$$

Intuition

Imagine standing on the surface of a globe at some city. The tangent space at that city is the flat plane that just touches the globe at that point — like laying a sheet of paper tangent to the surface. Every direction you could walk (north, east, northwest, etc.) defines a tangent vector. The tangent space is where infinitesimal displacements "live." You can add and scale tangent vectors (walk north at speed 2, add walk east at speed 3), making $T_p\mathcal{M}$ a vector space. Crucially, tangent spaces at different points are *different* planes — the tangent plane at the North Pole is horizontal, while the tangent plane at the equator is vertical.

Example 2.4. On S^2 at the point $p = (1, 0, 0)$ (on the equator), the tangent space is the plane $\{(0, y, z) : y, z \in \mathbb{R}\}$ — the plane perpendicular to the radius at p. A tangent vector $v = (0, 1, 0)$ points "eastward" along the equator, while $v = (0, 0, 1)$ points "northward" toward the pole.

Definition 2.5 (Riemannian Metric)**.** A *Riemannian metric* g on a smooth manifold $\mathcal{M}$ assigns to each point $p \in \mathcal{M}$ an inner product $g_p : T_p\mathcal{M} \times T_p\mathcal{M} \to \mathbb{R}$ on the tangent space $T_p\mathcal{M}$, varying smoothly with p.

Remark 2.1. I will publish notes about tensors and Riemannian metric a little later , sorry

Intuition

A Riemannian metric is a "ruler" that can vary from place to place on the manifold. Imagine walking on a hilly terrain: your sense of distance depends on whether you are on flat ground (easy walking, distances feel "normal") or on a steep slope (a short horizontal step corresponds to a longer actual walk). The Riemannian metric tells you, at every point, how to measure lengths and angles of infinitesimally small vectors.

Example 2.5. On the unit sphere S^2 with spherical coordinates (θ, ϕ), the Riemannian metric is $g = d\theta^2 + \sin^2\theta \; d\phi^2$. Near the equator ($\theta = \pi/2$), the metric looks like the flat metric $d\theta^2 + d\phi^2$. Near the poles ($\theta \approx 0$), the ϕ-direction is "squeezed" by $\sin^2\theta \approx \theta^2$, reflecting the fact that longitudinal lines converge at the poles.

2.2.1 The Metric Tensor in Local Coordinates

In a local coordinate chart $(x^1, \ldots, x^n)$, the Riemannian metric is represented by a symmetric, positive-definite matrix $G(p) = [g_{ij}(p)]$ where:

$$g_{ij}(p) = g_p\left(\frac{\partial}{\partial x^i}\bigg|_p, \; \frac{\partial}{\partial x^j}\bigg|_p \right).$$

The length of a tangent vector $v = \sum_i v^i \frac{\partial}{\partial x^i}$ is then:

$$\|v\|_g = \sqrt{\sum_{i,j} g_{ij} \, v^i \, v^j} = \sqrt{v^T G \, v}. \tag{1}$$

The length of a curve $\gamma(t) = (x^1(t), \ldots, x^n(t))$ for $t \in [a, b]$ is:

$$L(\gamma) = \int_a^b \sqrt{\sum_{i,j} g_{ij}(\gamma(t)) \, \dot{x}^i(t) \, \dot{x}^j(t)} \; dt. \tag{2}$$

Intuition

The matrix $G(p)$ is the metric's "DNA" at point p. On flat Euclidean space, G is the identity matrix everywhere. On a curved manifold, G varies from point to point. When G has large entries, distances are locally "stretched" — a small coordinate change produces a large actual displacement. When G has small entries, distances are "compressed." The key idea is that G encodes all the local geometric information: distances, angles, areas, and volumes.

Example 2.6. In polar coordinates (r, θ) for $\mathbb{R}^2$, the Euclidean metric becomes:

$$G = \begin{pmatrix} 1 & 0 \\ 0 & r^2 \end{pmatrix}, \qquad ds^2 = dr^2 + r^2\, d\theta^2.$$

At $r = 1$, a small angular change $d\theta$ produces displacement $1 \cdot d\theta$. At $r = 5$, the same $d\theta$ produces displacement $5 \cdot d\theta$ — five times as long, because circles at larger radii have larger circumferences. The metric tensor captures this via the r^2 entry.

2.2.2 Curvature: How the Metric Encodes Shape

The Riemannian metric g determines not just distances but also the *curvature* of the manifold. Curvature measures how much the manifold deviates from being flat.

Definition 2.6 (Sectional Curvature (Informal))**.** The *sectional curvature* $K(\sigma)$ of a 2-dimensional plane $\sigma \subset T_p\mathcal{M}$ measures how fast nearby geodesics starting from p in directions within σ converge or diverge, compared to straight lines in flat space. Positive curvature (like a sphere) means geodesics converge; negative curvature (like a saddle) means they diverge; zero curvature (like a plane) means they remain parallel.

Intuition

Walk north from two points on the equator that are 100 m apart. On a flat plane, you remain exactly 100 m apart forever. On the sphere (positive curvature), your paths converge and you meet at the North Pole. On a saddle (negative curvature), your paths diverge and the gap widens. Curvature is this tendency for "parallel" paths to converge or diverge — it is intrinsic to the manifold and can be computed purely from the metric tensor G and its derivatives, without reference to any ambient space.

Example 2.7. The unit sphere has constant positive curvature $K = 1$. Euclidean space has $K = 0$. The hyperbolic plane (Poincaré disk) has constant negative curvature $K = -1$. Real-world data manifolds typically have curvature that varies from point to point.

2.2.3 Connection to UMAP: How the Riemannian Metric is Estimated

UMAP never has direct access to the manifold or its metric. Instead, it *reconstructs* an approximate local Riemannian metric from the data:

1. **Local neighborhood distances:** UMAP computes the ambient distances to k-nearest neighbors. Under the uniformity assumption, these distances are distorted by the (unknown) metric: regions where the metric tensor G has large entries will appear "stretched," making neighbor distances larger.

2. **Normalization as metric estimation:** The parameter σ_i effectively estimates the local "scale" of the metric at x_i. Recall that the local metric is approximated as $g_p \approx \sigma_i^{-2}\, g_{\text{Euclidean}}$. This means UMAP interprets the ambient Euclidean distances as geodesic distances after rescaling by $1/\sigma_i$.

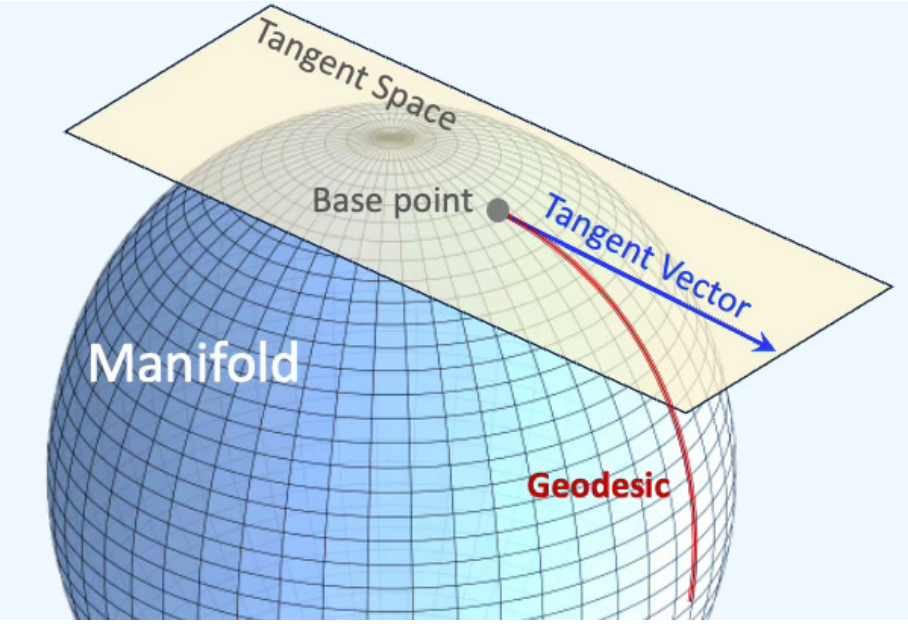

Figure 1: Geodesic Distance

3. **Why this works:** On a Riemannian manifold, in *normal coordinates* centred at p, the metric satisfies $g_{ij}(p) = \delta_{ij}$ (the identity) and the first-order corrections vanish. This means at sufficiently small scales, the metric looks Euclidean. UMAP's σ_i captures the scale at which this Euclidean approximation holds locally.

Intuition

Imagine you are a cartographer and you receive distance measurements between nearby villages, but the terrain is unknown. If villages in the mountains have large inter-distances and villages on the plains have small inter-distances, you can infer the terrain is rough in the mountains (the metric is stretched there) and flat on the plains (the metric is close to Euclidean). UMAP does exactly this: from the pattern of nearest-neighbor distances, it infers where the data manifold is locally stretched or compressed.

Example 2.8. Consider data sampled from a Swiss roll (a 2D sheet rolled up in 3D). Two points near each other in ambient $\mathbb{R}^3$ might be far apart along the roll (think of two points on adjacent layers). UMAP's k-NN graph connects points along the roll surface (following the manifold), not across layers (jumping through ambient space). The metric σ_i at each point reflects how tightly the roll is wound locally: tightly wound regions have small σ_i (many close neighbors), loosely wound regions have large σ_i.

Definition 2.7 (Geodesic Distance)**.** Given a Riemannian manifold $(\mathcal{M}, g)$, the *geodesic distance* between two points $p, q \in \mathcal{M}$ is:

$$d_g(p,q) = \inf_{\gamma} \int_0^1 \sqrt{g_{\gamma(t)}(\dot{\gamma}(t), \dot{\gamma}(t))}\, dt,$$

where the infimum is taken over all smooth curves $\gamma : [0,1] \to \mathcal{M}$ with $\gamma(0) = p$ and $\gamma(1) = q$.

Intuition

The geodesic distance is the length of the shortest path *along the surface* between two points — like the great-circle distance between two cities on Earth, not the straight-line tunnel through the interior. On flat surfaces, geodesics are straight lines; on curved surfaces, they curve along with the surface (great circles on a sphere, for instance).

Remark 2.2. UMAP does not know the manifold or its Riemannian metric directly. Instead, it *estimates* local geodesic distances from the data using nearest-neighbor information and the uniformity assumption described in Section 3.

2.3 Fuzzy Simplicial Sets (Short)

UMAP's theoretical framework uses the language of fuzzy simplicial sets and category theory. For applied engineering, the essential content can be summarised concisely.

Definition 2.8 (Fuzzy Simplicial Set (Working Definition))**.** A *fuzzy simplicial set*, for UMAP's purposes, is a weighted graph where each edge (i, j) has a membership strength $\mu(i, j) \in [0, 1]$. The membership represents the degree to which two points are "connected."

Intuition

Instead of saying two data points are "neighbors" or "not neighbors" (a hard binary decision), UMAP assigns a soft friendship strength. Point x_2 might be a 0.95-strength neighbor of x_1 (very close), while x_3 is a 0.2-strength neighbor (far but not invisible).

Remark 2.3 (TheoreticalLy not a full formal explanation !!)**.** The UMAP paper establishes a correspondence (an *adjunction*) between the category of finite extended pseudo-metric spaces (**FinEPMet**) and the category of finite fuzzy simplicial sets (**FinFuzz**). For practical use of UMAP, the key takeaway is: the exponential kernel $\exp(-d)$ is not an arbitrary choice — it is the *canonical* way to convert distances into fuzzy memberships, justified by the theory of fuzzy metric spaces.

3 The Manifold Assumption and Uniformity

Assumption: Manifold Hypothesis

The data $X = \{x_1, \ldots, x_N\} \subset \mathbb{R}^D$ is sampled from (or lies near) a Riemannian manifold $(\mathcal{M}, g)$ of intrinsic dimension $n \ll D$, possibly with noise.

Intuition

Imagine scattering rice grains on a crumpled bedsheet floating in a room. The room is 3-dimensional ($D = 3$), but the rice lives on the 2-dimensional sheet ($n = 2$). If you only see the rice positions in 3D, they appear to fill some complicated blob — but they actually lie on a low-dimensional surface. The manifold hypothesis says real-world data behaves similarly: images, text, and audio may be represented as D-dimensional vectors, but the meaningful variation occupies a much lower-dimensional "sheet" in that space.

Example 3.1. Consider images of a single object (say, a teapot) under varying rotation angle $\theta \in [0, 2\pi)$ and lighting direction $\phi \in [0, 2\pi)$. Each image is a $64 \times 64 = 4{,}096$-dimensional vector, but the set of all such images forms a 2-dimensional manifold (parametrised by θ and ϕ) embedded in $\mathbb{R}^{4096}$.

Assumption: Local Uniformity

With respect to the Riemannian metric g, the data is *locally uniformly distributed* on $\mathcal{M}$. That is, in a sufficiently small geodesic ball around any point $p \in \mathcal{M}$, the data density is approximately constant.

Intuition

Imagine you are an ant standing on the crumpled bedsheet from before, and you can only see a small circle around your feet. This assumption says: no matter where the ant stands, the rice grains in its visible circle are spread roughly evenly. In reality, some areas may have more rice than others, but UMAP *pretends* the distribution is uniform locally and then corrects for density differences by adjusting the distance scale (σ_i) at each point. Where rice is sparse, the ant stretches its ruler; where rice is dense, the ant shrinks it. After this rescaling, every neighborhood "looks" uniform.

Example 3.2. Suppose points on a 1D manifold (a curve) are densely packed on the left and sparse on the right:

$$x_1 = 0,\ x_2 = 0.1,\ x_3 = 0.2,\ x_4 = 2.0,\ x_5 = 5.0.$$

Without normalization, x_4 and x_5 appear isolated. Under local uniformity, UMAP assigns x_4 a large σ_4 (stretching the ruler) and x_1 a small σ_1 (shrinking the ruler), so that each point "sees" its neighbors at comparable effective distances.

Justification of Local Uniformity

This assumption is strong but serves a precise purpose. If the data is locally uniform, then the geodesic distance between a point and its k-th nearest neighbor encodes information purely about the local geometry (curvature, dimension) and not about density variations. Specifically:

1. In a region of uniform density ρ on an n-dimensional manifold, the expected distance to the k-nearest neighbor is proportional to $(\rho)^{-1/n}$.
2. If data were not locally uniform, density and geometry would be confounded: a large gap to a neighbor could mean high curvature or low density.
3. By *assuming* uniformity, UMAP isolates the geometric signal. Since real data is generally not uniform, UMAP compensates by normalizing the distances locally (via ρ_i and σ_i), effectively re-scaling each local neighborhood to "look" uniform.

Proposition 3.1 (Consequence of Uniformity)**.** *Under Assumption 3, the geodesic distance from x_i to its nearest neighbor x_{i_1} converges to zero as $N \to \infty$, and the Riemannian metric g can be locally approximated as:*

$$g_p \approx \frac{1}{\sigma_i^2} g_{\text{Euclidean}},$$

where σ_i is a local scaling factor determined by the distribution of the k-nearest neighbors of x_i.

Intuition

As you collect more and more data, your nearest neighbor gets closer and closer (you're "filling in" the manifold). At that infinitesimal scale, any smooth manifold looks flat — like the Earth looks flat from your backyard. The factor $1/\sigma_i^2$ tells you how much the manifold is "stretched" relative to flat space at point x_i. Large σ_i (sparse neighborhood)

means the manifold is locally stretched; small σ_i (dense neighborhood) means it is locally compressed.

Example 3.3. If x_i is in a dense cluster and its 15 nearest neighbors are all within distance 0.5, then σ_i will be small (say 0.3). If x_j is in a sparse region with neighbors at distance 10, then σ_j will be large (say 6.0). The local metric at x_i is "zoomed in" by $1/0.3^2 \approx 11\times$ relative to the Euclidean metric, while at x_j it is "zoomed out" by $1/6^2 \approx 0.03\times$.

4 Construction of the High-Dimensional Fuzzy Graph

4.1 Step 1: k-Nearest Neighbor Graph

For each point x_i, compute its k-nearest neighbors $\{x_{i_1}, x_{i_2}, \ldots, x_{i_k}\}$ using the ambient Euclidean distance (or another specified metric). Define:

$$\rho_i = \min_{j : x_j \in \mathrm{kNN}(x_i)} d(x_i, x_j) = d(x_i, x_{i_1}),$$

i.e., ρ_i is the distance to the nearest neighbor.

Key Idea

The parameter ρ_i ensures *local connectivity*: by subtracting ρ_i from all distances, each point has at least one neighbor at effective distance 0, guaranteeing the resulting graph is connected (at least locally). Without this, isolated points with large nearest-neighbor distances would be disconnected from the graph.

Intuition

Think of ρ_i as each point "reaching out" to grab onto its closest friend. No matter how far away that closest friend is, the handshake happens at effective distance zero. This prevents any point from being an island. It is like saying: "At minimum, everyone in this room knows at least one other person."

Example 4.1. With $k = 5$, suppose the distances from x_7 to its 5 nearest neighbors are $[0.3, 0.5, 0.8, 1.2, 2.0]$. Then $\rho_7 = 0.3$, and the adjusted distances become $[0.0, 0.2, 0.5, 0.9, 1.7]$. The nearest neighbor is now at effective distance 0, ensuring full connectivity to at least one point.

4.2 Step 2: Adaptive Exponential Kernel and σ_i

For each point x_i, define the directed edge weight to neighbor x_j as:

$$w_{i \to j} = \exp\left(-\frac{\max(0,\ d(x_i, x_j) - \rho_i)}{\sigma_i} \right). \tag{3}$$

The parameter $\sigma_i > 0$ is found by solving the constraint:

$$\sum_{j=1}^{k} \exp\left(-\frac{\max(0,\ d(x_i, x_{i_j}) - \rho_i)}{\sigma_i} \right) = \log_2(k). \tag{4}$$

Intuition

The constraint says: "Each point must have the same effective number of soft neighbors, namely $\log_2(k)$." If a point is in a dense region, σ_i is small (a narrow kernel suffices to capture $\log_2(k)$ worth of neighbor-mass). If a point is in a sparse region, σ_i is large (the kernel must spread wide to gather enough neighbor-mass). This is how UMAP *implements* the local uniformity assumption: it adaptively rescales each neighborhood so that every point "sees" an equally rich local environment.

Example 4.2. Let $k = 15$, so the target is $\log_2(15) \approx 3.91$. For a point in a dense cluster with adjusted distances $[0, 0.01, 0.02, \ldots]$, a small $\sigma_i \approx 0.008$ would suffice: many neighbors contribute nearly 1.0 each. For a point in a sparse region with adjusted distances $[0, 2.5, 4.0, \ldots]$, σ_i might be ≈ 3.0: the kernel must be very wide for the sum to reach 3.91.

Derivation of the σ_i Constraint (Equation 4)

The constraint arises from a notion of *normalized local connectivity entropy.*

Motivation: We want the effective number of neighbors (measured by the Shannon entropy or a related quantity) to be consistent across all data points, regardless of local density. This implements the uniformity assumption: if the data is locally uniform, each point should "see" the same effective neighborhood size.

Derivation:

1. Define the directed membership weights $p_{j|i} = \exp\left(-\frac{d(x_i, x_{i_j}) - \rho_i}{\sigma_i}\right)$ for $j = 1, \ldots, k$.
2. The sum $S_i = \sum_{j=1}^{k} p_{j|i}$ represents the "effective number of neighbors" in a soft sense.
3. We set $S_i = \log_2(k)$. This choice is motivated by information-theoretic considerations: $\log_2(k)$ is related to the perplexity 2^H at the level of k neighbors, similar to the perplexity constraint used in t-SNE but in a different form.
4. Equation (4) defines σ_i implicitly as a function of the distances $\{d(x_i, x_{i_j})\}_{j=1}^{k}$.
5. Since the left-hand side is continuous and strictly decreasing in σ_i (from k as $\sigma_i \to \infty$ to 0 as $\sigma_i \to 0^+$, after accounting for ρ_i), a unique solution exists by the intermediate value theorem. It is found numerically via binary search.

Connection to Perplexity (t-SNE comparison): In t-SNE, the perplexity parameter directly sets $\sum_j p_{j|i}$ via the Shannon entropy. UMAP's constraint is simpler but achieves a similar normalization. The parameter k in UMAP plays an analogous role to perplexity in t-SNE.

4.3 Step 3: Symmetrization via Fuzzy Set Union

The directed weights $w_{i\to j}$ and $w_{j\to i}$ are generally different because $\sigma_i \neq \sigma_j$ and $\rho_i \neq \rho_j$. To obtain a symmetric (undirected) fuzzy graph, UMAP applies a *probabilistic t-conorm* (fuzzy set union):

$$w_{ij} = w_{i\to j} + w_{j\to i} - w_{i\to j} \cdot w_{j\to i}. \tag{5}$$

Intuition

Imagine two people, Alis and Burger. Alis thinks the friendship strength is 0.8 (she considers Burger a good friend); Burger thinks it is 0.3 (he barely knows Alis). The question is: does a friendship "exist" between them? UMAP takes an inclusive approach: if *either* person considers the other a friend, the edge is strong. Formula (5) is the probability that at least one of them considers the other a friend, assuming independent assessments: $P(\text{at least one}) = 0.8 + 0.3 - 0.8 \times 0.3 = 0.86$. This makes the graph more connected than conservative alternatives (like taking the average 0.55 or the minimum 0.3).

Example 4.3. Three points: $w_{1\to 2} = 0.9$, $w_{2\to 1} = 0.4$, $w_{1\to 3} = 0.1$, $w_{3\to 1} = 0.05$. After symmetrization:

$$w_{12} = 0.9 + 0.4 - 0.9 \times 0.4 = 0.94,$$
$$w_{13} = 0.1 + 0.05 - 0.1 \times 0.05 = 0.145.$$

The strong connection $(1, 2)$ stays strong; the weak connection $(1, 3)$ stays weak but is still present.

Justification of the Symmetrization (Equation 5)

Interpretation via probability: If we interpret $w_{i\to j}$ as the probability that x_j is "connected to" x_i (from x_i's perspective) and $w_{j\to i}$ as the analogous probability from x_j's perspective, then:

$$w_{ij} = P(i \sim j \text{ from } i\text{'s view}) + P(i \sim j \text{ from } j\text{'s view}) - P(\text{both}),$$

assuming independence. This is the inclusion-exclusion formula:

$$P(A \cup B) = P(A) + P(B) - P(A)P(B).$$

Category-theoretic justification: In the category of fuzzy sets, the union of two fuzzy sets A and B with membership functions μ_A and μ_B can be defined via a t-conorm. The *probabilistic sum* t-conorm is:

$$S(a, b) = a + b - ab.$$

This is one of the standard choices in fuzzy set theory and lies between the maximum t-conorm $\max(a, b)$ and the bounded sum $\min(a + b, 1)$.

Practical consequence: This symmetrization is *inclusive*: an edge exists with high weight if *either* endpoint considers the other a close neighbor. This contrasts with taking the minimum or average, which would be more conservative.

Remark 4.1 (Comparison with t-SNE Symmetrization)**.** t-SNE symmetrizes by averaging: $p_{ij} = (p_{j|i} + p_{i|j})/(2N)$. UMAP's probabilistic union tends to produce a more connected graph, which helps preserve more global structure.

5 The Cross-Entropy Cost Function

Let $\mu_h(i, j) = w_{ij}$ denote the high-dimensional edge weights and $\mu_\ell(i, j) = v_{ij}$ denote the low-dimensional edge weights. UMAP seeks an embedding $Y = \{y_1, \ldots, y_N\} \subset \mathbb{R}^d$ minimizing the *fuzzy set cross-entropy*:

$$\mathrm{CE}(\mu_h, \mu_\ell) = \sum_{(i,j)\in E} \left[\mu_h(i,j) \log\left(\frac{\mu_h(i,j)}{\mu_\ell(i,j)}\right) + (1-\mu_h(i,j)) \log\left(\frac{1-\mu_h(i,j)}{1-\mu_\ell(i,j)}\right) \right]. \tag{6}$$

Intuition

For each pair of points, UMAP asks: "In high-D, this edge has strength μ_h. In low-D, this edge has strength μ_ℓ. How *surprised* would I be?" The cross-entropy measures total surprise across all edges. If μ_h is large but μ_ℓ is small (close neighbors mapped far apart), the first term penalizes heavily. If μ_h is small but μ_ℓ is large (distant points mapped close together), the second term penalizes heavily. The optimization finds the embedding that minimizes total surprise — i.e., the low-D layout where the neighborhood structure best matches the high-D one.

Example 5.1. Consider a single edge where $\mu_h = 0.9$ (the points are close in high-D). If $\mu_\ell = 0.9$ (also close in low-D), the cost is $0.9\log(1) + 0.1\log(1) = 0$. If $\mu_\ell = 0.1$ (far apart in low-D), the cost is $0.9\log(9) + 0.1\log(1/9) \approx 1.76$. The large cost penalizes "tearing apart" points that should be close.

5.1 Derivation and Interpretation

Detailed Derivation of the Cost Function

Consider each edge (i, j) as a Bernoulli random variable: the edge "exists" with probability $\mu_h(i, j)$ in the high-dimensional representation. We seek the low-dimensional edge weights $\mu_\ell(i, j)$ that best match this.

Step 1: Binary Cross-Entropy per Edge. For a single edge with true probability $p = \mu_h(i, j)$ and model probability $q = \mu_\ell(i, j)$, the binary cross-entropy is:

$$H(p, q) = -p \log q - (1 - p) \log(1 - q).$$

Step 2: Summing Over All Edges. Summing over all edges E in the fuzzy graph:

$$C = -\sum_{(i,j)\in E} \left[\mu_h \log \mu_\ell + (1-\mu_h) \log(1-\mu_\ell)\right].$$

Step 3: Rewriting as KL-Divergence Form. Adding and subtracting the entropy of μ_h:

$$\begin{aligned} C &= -\sum_{(i,j)} \left[\mu_h \log \mu_\ell + (1-\mu_h)\log(1-\mu_\ell)\right] \\ &= \underbrace{-\sum_{(i,j)} \left[\mu_h \log \mu_h + (1-\mu_h)\log(1-\mu_h)\right]}_{\text{constant (entropy of } \mu_h)} \\ &\quad + \sum_{(i,j)} \left[\mu_h \log \frac{\mu_h}{\mu_\ell} + (1-\mu_h)\log\frac{1-\mu_h}{1-\mu_\ell}\right]. \end{aligned}$$

Since the first term is constant with respect to Y, minimizing C is equivalent to minimizing Equation (6).

Decomposition of the cost:

$$\mathrm{CE} = \underbrace{\sum_{(i,j)} \mu_h \log \frac{\mu_h}{\mu_\ell}}_{\text{Attractive term}} + \underbrace{\sum_{(i,j)} (1-\mu_h) \log \frac{1-\mu_h}{1-\mu_\ell}}_{\text{Repulsive term}}. \tag{7}$$

- **Attractive term:** Large when μ_h is large (points should be close) but μ_ℓ is small (they are far in the embedding). This pulls connected points together.
- **Repulsive term:** Large when μ_h is small (points should be distant) but μ_ℓ is large (they are close in the embedding). This pushes weakly-connected points apart.

Key Idea

The cross-entropy cost simultaneously handles attraction and repulsion, which is essential for producing useful embeddings. Pure attraction (as in a naive force-directed layout) would collapse everything to a point. The repulsive term prevents this and creates meaningful separation between clusters.

6 Low-Dimensional Weight Function

In the low-dimensional space, the edge weights are defined by:

$$\mu_\ell(i,j) = v_{ij} = \left(1 + a\,\|y_i - y_j\|^{2b}\right)^{-1}, \tag{8}$$

where $a > 0$ and $b > 0$ are hyperparameters determined by the `min_dist` parameter.

Intuition

This function converts distances in the low-dimensional embedding into membership strengths. It is a "bell curve" centered at distance 0 with heavy tails: points at distance 0 get membership 1 (fully connected); points at moderate distance get membership that decays smoothly; points very far away get membership near 0. The heavy tails (slower decay than a Gaussian) are crucial — they give the embedding "breathing room" to spread out moderate-distance points, solving the crowding problem (Section 10).

Example 6.1. With default parameters ($a \approx 1.93$, $b \approx 0.79$ for `min_dist`$= 0.1$):

$$\begin{aligned}
\|y_i - y_j\| = 0.05 &\implies v_{ij} \approx 0.997 \quad \text{(very close: nearly full membership)},\\
\|y_i - y_j\| = 1.0 &\implies v_{ij} \approx 0.34 \quad \text{(moderate: partial membership)},\\
\|y_i - y_j\| = 5.0 &\implies v_{ij} \approx 0.03 \quad \text{(far: weak membership)}.
\end{aligned}$$

6.1 Derivation: Fitting a and b from `min_dist`

Deriving a and b from `min_dist`

UMAP defines a target membership function:

$$\psi(r) = \begin{cases} 1 & \text{if } r \le \texttt{min\_dist},\\ \exp(-(r - \texttt{min\_dist})) & \text{if } r > \texttt{min\_dist}. \end{cases}$$

The parameters a and b are obtained by a nonlinear least-squares fit:

$$(a^*, b^*) = \arg\min_{a,b} \int_0^R \left[\psi(r) - \left(1 + a\,r^{2b}\right)^{-1}\right]^2 dr,$$

for a suitable upper bound R.
Rationale:

1. For $r \leq$ `min_dist`, we want $v_{ij} \approx 1$: points within the minimum distance are considered fully connected.
2. For $r >$ `min_dist`, v_{ij} decays smoothly, modelling decreasing affinity.
3. The form $(1 + a\, r^{2b})^{-1}$ generalizes the Cauchy/Student-t distribution kernel ($a = 1, b = 1$ gives the standard Cauchy kernel used by t-SNE).

Remark 6.1 (Connection to the Student-t Distribution)**.** When $a = 1$ and $b = 1$, we recover:

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

which is the Cauchy kernel (Student-t with 1 degree of freedom). t-SNE uses exactly this kernel. The heavy tails of the Cauchy distribution allow moderate distances in high dimensions to be represented by larger distances in low dimensions, alleviating the "crowding problem" where the volume of a low-dimensional ball is insufficient to accommodate all moderately-distant neighbors.

Remark 6.2 (The ordering of tail heaviness for kernels(most common kernels tho)**.**

$$\underbrace{\textbf{Gaussian(RBF)}}_{\text{lightest tails}} \ll \underbrace{\textbf{Laplacian(Exponential)}}_{\text{medium tails}} \ll \underbrace{\textbf{Cauchy/Student- t}}_{\text{heaviest tails}}$$

7 Optimization: Stochastic Gradient Descent with Negative Sampling

7.1 Gradients of the Cost Function

Taking the gradient of CE with respect to y_i:

Gradient Derivation

From the attractive term, for an edge (i, j) with high-dimensional weight μ_h:
Attractive gradient:

$$\frac{\partial}{\partial y_i}\left[\mu_h \log \frac{\mu_h}{\mu_\ell}\right] = -\mu_h \cdot \frac{1}{\mu_\ell} \cdot \frac{\partial \mu_\ell}{\partial y_i}.$$

With $\mu_\ell = (1 + a\|y_i - y_j\|^{2b})^{-1}$, let $u = \|y_i - y_j\|^2$. Then:

$$\begin{aligned}\frac{\partial \mu_\ell}{\partial y_i} &= -\frac{a \cdot 2b \cdot u^{b-1}}{(1 + a\, u^b)^2} \cdot (y_i - y_j) \\ &= -\frac{2ab \cdot \|y_i - y_j\|^{2(b-1)}}{(1 + a\, \|y_i - y_j\|^{2b})^2} \cdot (y_i - y_j).\end{aligned}$$

So the attractive force on y_i due to y_j is:

$$F_{ij}^{\text{attr}} = \frac{2ab\, \mu_h\, \|y_i - y_j\|^{2(b-1)}}{(1 + a\, \|y_i - y_j\|^{2b})} \cdot (y_j - y_i). \tag{9}$$

Note the direction is $y_j - y_i$, attracting y_i toward y_j.
Repulsive gradient: Similarly, the repulsive term yields a force:

$$F_{ij}^{\text{rep}} = \frac{2b\,(1 - \mu_h)}{(\epsilon + \|y_i - y_j\|^2)(1 + a\, \|y_i - y_j\|^{2b})} \cdot (y_i - y_j), \tag{10}$$

where $\epsilon > 0$ is a small constant for numerical stability. The direction is $y_i - y_j$, pushing y_i away from y_j.

Intuition: Forces as Springs and Magnets

The optimization is a physical simulation. Each high-weight edge acts like a *spring*: it pulls connected points together (attractive force). Meanwhile, randomly sampled unconnected pairs act like *same-pole magnets*: they push each other apart (repulsive force). The embedding converges when these forces balance — clusters of tightly-connected points huddle together, while different clusters push away from each other.

7.2 Negative Sampling

Evaluating the repulsive term over all $\binom{N}{2}$ pairs is $O(N^2)$. UMAP uses **negative sampling** to reduce this to $O(N)$.

For each positive edge (i, j) sampled, UMAP samples n_{neg} (typically $n_{\text{neg}} = 5$) random "negative" points $y_{n_1}, \ldots, y_{n_{n_{\text{neg}}}}$ and applies the repulsive force only against these. This is analogous to negative sampling in Word2Vec.

Assumption: Negative Sampling Approximation

The repulsive term can be approximated by:

$$\sum_{j \neq i} (1 - \mu_h(i,j)) \log \frac{1 - \mu_h(i,j)}{1 - \mu_\ell(i,j)} \approx n_{\text{neg}} \cdot \mathbb{E}_{j \sim P_{\text{neg}}} \left[\log(1 - \mu_\ell(i,j)) \right],$$

where P_{neg} is a uniform distribution over all points. This is justified because most edges have $\mu_h \approx 0$, so $(1 - \mu_h) \approx 1$ for the vast majority of pairs.

Intuition

Imagine a party with 1,000 guests. Each person has 15 friends there ($k = 15$). To compute the full repulsive force, each person would need to push away from all 985 non-friends — expensive! Negative sampling says: instead of pushing against all 985, just pick 5 random strangers and push against them. Since almost everyone is a stranger (985 out of 999), a random sample gives a good estimate of the total repulsive effect. The approximation is excellent because the "signal" is spread uniformly across the vast majority of non-neighbor pairs.

Example 7.1. With $N = 10{,}000$ points and $k = 15$, each point has only 15 neighbors, so $\mu_h > 0$ for only 15 pairs, and $\mu_h \approx 0$ for the remaining 9,985 pairs. The repulsive sum over those 9,985 pairs is well-approximated by sampling just 5 random points and scaling, because the $(1 - \mu_h)$ factor is essentially 1 for all of them.

7.3 SGD with Edge Sampling

Algorithm 1 UMAP Optimization (Simplified)

Require: Fuzzy graph edges E with weights $\{w_{ij}\}$, initial embedding $Y^{(0)}$
Require: Learning rate α, number of epochs T, negative samples n_{neg}
for $t = 1, \ldots, T$ **do**
  $\alpha_t \leftarrow \alpha \cdot (1 - t/T)$ ▷ Linear learning rate decay
  for each edge $(i, j) \in E$ (sampled with probability $\propto w_{ij}$) **do**
    $y_i \leftarrow y_i + \alpha_t \cdot F_{ij}^{\text{attr}}$ ▷ Attraction
    $y_j \leftarrow y_j - \alpha_t \cdot F_{ij}^{\text{attr}}$
    for $m = 1, \ldots, n_{\text{neg}}$ **do**
      Sample $n \neq i$ uniformly at random
      $y_i \leftarrow y_i + \alpha_t \cdot F_{in}^{\text{rep}}$ ▷ Repulsion
    end for
  end for
end for
return $Y^{(T)}$

Remark 7.1 (Initialization)**.** UMAP uses spectral embedding (eigenvectors of the normalized graph Laplacian) for initialization, which provides a globally coherent starting configuration. This is important because SGD can only refine a layout, not recover from a completely random starting point in reasonable time. The full details of spectral initialization are given in Section 8.

8 Spectral Initialization

Spectral initialization is a critical component of UMAP that provides the starting embedding $Y^{(0)}$ for the SGD optimization. Without a good initialization, the SGD would need to discover the global layout from scratch, which is both slow and prone to poor local optima. This section provides a complete treatment of the mathematical machinery behind spectral embedding.

8.1 The Graph Laplacian

Definition 8.1 (Degree Matrix)**.** Given the symmetric fuzzy graph with weight matrix $W = [w_{ij}]$ (obtained after the symmetrization step, Equation 5), the *degree matrix* D is the $N \times N$ diagonal matrix:

$$D_{ii} = \sum_{j=1}^{N} w_{ij}, \qquad D_{ij} = 0 \text{ for } i \neq j.$$

Intuition

The degree D_{ii} of a node i is the total "connection strength" of that node. In an unweighted graph, this counts the number of edges. In UMAP's fuzzy graph, it sums the membership strengths of all edges incident to i. A node in a dense cluster has high degree (many strong connections); an isolated node has low degree (few or weak connections).

Example 8.1. For a 4-point graph with weights $w_{12} = 0.9$, $w_{13} = 0.3$, $w_{14} = 0$, $w_{23} = 0.5$, $w_{24} = 0.1$, $w_{34} = 0.7$:

$$D = \text{diag}(1.2,\ 1.5,\ 1.5,\ 0.8).$$

Node 2 has the highest degree (1.5) because it is well-connected to nodes 1, 3, and 4.

Definition 8.2 (Unnormalized Graph Laplacian)**.** The *unnormalized graph Laplacian* is:

$$L = D - W.$$

Definition 8.3 (Normalized Graph Laplacian)**.** The *symmetric normalized graph Laplacian* is:

$$L_{\text{sym}} = D^{-1/2} L\, D^{-1/2} = I - D^{-1/2} W\, D^{-1/2}.$$

The *random walk normalized Laplacian* is:

$$L_{\text{rw}} = D^{-1} L = I - D^{-1} W.$$

UMAP typically uses L_{sym} or equivalently works with the normalized adjacency matrix $D^{-1/2} W D^{-1/2}$.

Intuition

The graph Laplacian is the "discrete analogue of the continuous Laplace operator" from calculus ($\nabla^2 f$). Just as the Laplacian on a surface measures how much a function's value at a point deviates from the average of its neighbors, the graph Laplacian Lf at node i computes a weighted difference between the function value $f(i)$ and the values at its neighbors. The normalized version accounts for degree differences, so high-degree nodes are not unfairly weighted.
Think of the graph as a network of springs (edges) connecting masses (nodes). The Laplacian encodes the stiffness of these springs. The eigenvectors of L describe the natural modes of vibration: the lowest-frequency modes capture the large-scale "shape" of the network, which is exactly what we need for initialization.

Example 8.2. For the path graph $1 - 2 - 3$ (all weights $= 1$):

$$W = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}, \quad D = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad L = D - W = \begin{pmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{pmatrix}.$$

8.2 Key Properties of the Graph Laplacian

Proposition 8.1 (Properties of L)**.** *The unnormalized graph Laplacian $L = D - W$ satisfies:*

(i) L is symmetric and positive semi-definite: all eigenvalues $\lambda_i \geq 0$.

(ii) The smallest eigenvalue is always $\lambda_0 = 0$, with eigenvector $\mathbf{1} = (1, 1, \ldots, 1)^T$.

(iii) The multiplicity of eigenvalue 0 equals the number of connected components of the graph.

(iv) For any vector $f \in \mathbb{R}^N$:

$$f^T L f = \sum_{(i,j) \in E} w_{ij} (f_i - f_j)^2. \tag{11}$$

Intuition

Property (iv) is the most important for understanding spectral embedding. It says that $f^T L f$ measures the "smoothness" of the function f on the graph: if f assigns similar values to connected nodes, the sum is small; if f assigns very different values to connected nodes, the sum is large. The eigenvectors of L are functions that are maximally smooth on the graph (for a given "frequency").

Property (iii) explains why spectral methods detect clusters: if the graph has k well-separated components, there are k eigenvectors with eigenvalue 0, each constant on one component and zero elsewhere. For nearly-separated clusters, the small eigenvalues and their eigenvectors reveal the cluster structure.

Proof of the Quadratic Form (Property iv)

$$\begin{aligned} f^T L f &= f^T (D - W) f = f^T D f - f^T W f \\ &= \sum_i D_{ii} f_i^2 - \sum_{i,j} w_{ij} f_i f_j \\ &= \sum_i \left(\sum_j w_{ij} \right) f_i^2 - \sum_{i,j} w_{ij} f_i f_j \\ &= \frac{1}{2} \left(\sum_i \sum_j w_{ij} f_i^2 + \sum_j \sum_i w_{ij} f_j^2 - 2 \sum_{i,j} w_{ij} f_i f_j \right) \\ &= \frac{1}{2} \sum_{i,j} w_{ij} (f_i - f_j)^2 = \sum_{(i,j) \in E} w_{ij} (f_i - f_j)^2. \end{aligned}$$ □

Example 8.3. On the path graph $1-2-3$, consider $f = (0, 0.5, 1)^T$ (smooth) and $g = (0, 1, 0)^T$ (oscillating):

$$\begin{aligned} f^T L f &= 1 \cdot (0 - 0.5)^2 + 1 \cdot (0.5 - 1)^2 = 0.25 + 0.25 = 0.5, \\ g^T L g &= 1 \cdot (0 - 1)^2 + 1 \cdot (1 - 0)^2 = 1 + 1 = 2. \end{aligned}$$

The smooth function f has a much lower Laplacian energy (0.5) than the oscillating function g (2.0), confirming that eigenvectors with small eigenvalues are "smooth" on the graph.

8.3 The Spectral Embedding Procedure

UMAP's spectral initialization computes a d-dimensional embedding using the eigenvectors of the graph Laplacian. The procedure, based on the Laplacian Eigenmaps algorithm of Belkin and Niyogi (2003), is as follows:

Step 1: Construct the normalized Laplacian. From the symmetric weight matrix W and degree matrix D, compute $L_{\text{sym}} = I - D^{-1/2} W D^{-1/2}$.

Step 2: Compute the bottom eigenvectors. Find the $d+1$ smallest eigenvalues $0 = \lambda_0 \leq \lambda_1 \leq \cdots \leq \lambda_d$ and their corresponding eigenvectors $v_0, v_1, \ldots, v_d$ of L_{sym}.

Step 3: Discard the trivial eigenvector. The eigenvector v_0 corresponding to $\lambda_0 = 0$ is constant and carries no structural information. Discard it.

Step 4: Form the embedding. The initial embedding is:

$$Y^{(0)} = \begin{pmatrix} | & | & & | \\ v_1 & v_2 & \cdots & v_d \\ | & | & & | \end{pmatrix} \in \mathbb{R}^{N \times d}, \tag{12}$$

where each row $y_i^{(0)} = (v_1(i), v_2(i), \ldots, v_d(i))$ gives the initial coordinates of point i.

Step 5: Rescale. Normalize the embedding to have a standard scale appropriate for the subsequent SGD optimization.

Intuition

The spectral embedding solves the optimization problem: "Place N points in $\mathbb{R}^d$ such that strongly-connected points are close together and the total spring energy $\sum_{ij} w_{ij}\|y_i - y_j\|^2$ is minimized, subject to the points not collapsing to a single point." The eigenvectors of the Laplacian are the exact solution to this constrained optimization.
Think of it as finding the "most natural" low-dimensional shape of the graph. The first non-trivial eigenvector v_1 captures the dominant axis of variation (e.g., for a long, thin cluster, it separates the two ends). The second eigenvector v_2 captures the next most important axis (perpendicular to the first). Together, they provide a globally coherent "sketch" of the data that the SGD can then refine.

Why Eigenvectors Minimize Spring Energy

The spectral embedding minimizes:

$$\min_{Y \in \mathbb{R}^{N \times d}} \sum_{i,j} w_{ij}\|y_i - y_j\|^2 = \min_{Y} \sum_{k=1}^{d} y_k^T L\, y_k = \min_{Y} \operatorname{tr}(Y^T L Y),$$

subject to $Y^T D Y = I$ (preventing collapse to a point and enforcing orthogonality). This is a generalized eigenvalue problem:

$$Lv = \lambda D v,$$

or equivalently, the standard eigenvalue problem for $L_{\text{sym}} = D^{-1/2} L D^{-1/2}$:

$$L_{\text{sym}} u = \lambda u, \qquad \text{where } u = D^{1/2} v.$$

The minimizing coordinates are the eigenvectors corresponding to the d smallest non-zero eigenvalues. Each coordinate direction v_k independently minimizes $v_k^T L v_k$ (the spring energy in that direction) subject to being orthogonal to all previous directions.

Example 8.4. Consider 6 points arranged as two triangles connected by a weak edge:

$$\underbrace{1 - 2 - 3}_{\text{cluster A}} \;\text{---}\; \underbrace{4 - 5 - 6}_{\text{cluster B}}$$

with strong weights ($w = 1.0$) within each triangle and a weak weight ($w = 0.1$) on edge $(3, 4)$.

The first non-trivial eigenvector v_1 will assign approximately the same value to nodes $\{1, 2, 3\}$ (say, around -0.4) and a different common value to $\{4, 5, 6\}$ (say, around $+0.4$). This eigenvector separates the two clusters along the first embedding axis.

The second eigenvector v_2 might separate within each cluster (e.g., distinguishing node 1 from 3 within cluster A). Together, (v_1, v_2) gives a 2D initialization where the two clusters are well-separated, and the SGD can refine the internal layout.

8.4 Why Spectral Initialization Matters for UMAP

Key Idea

The spectral embedding captures the *global* structure of the data (connected components, relative cluster positions, overall geometry), while the subsequent SGD optimization refines the *local* structure (within-cluster arrangements, fine-grained distances). This two-phase approach is why UMAP preserves both local and global structure better than

methods that rely solely on local optimization from random starts.

Intuition

Imagine organizing books in a new library. The spectral embedding is like first deciding which genre goes on which shelf (global layout). The SGD is like then arranging individual books within each shelf by author and title (local refinement). If you started by randomly placing books everywhere, even hours of rearrangement might not discover that all the mysteries should go together — but if you start with genres roughly grouped, the fine-tuning is fast and effective.

Remark 8.1 (Fallback to Random Initialization)**.** When the graph Laplacian has numerical issues (e.g., for very large or disconnected datasets), UMAP falls back to a random initialization, typically sampling from a scaled Gaussian distribution. The SGD can still produce good results, but convergence is slower and global structure may be less well-preserved.

Remark 8.2 (Connection to Laplacian Eigenmaps and Diffusion Maps)**.** UMAP's spectral initialization is closely related to Laplacian Eigenmaps (Belkin & Niyogi, 2003) and Diffusion Maps (Coifman & Lafon, 2006). In fact, if one stopped after the spectral embedding step without running SGD, the result would be essentially a Laplacian Eigenmap. What UMAP adds is the subsequent non-linear optimization with the cross-entropy cost, which allows the embedding to better capture non-linear structure and fine-grained local geometry.

9 Theoretical Foundations

9.1 Fuzzy Set Cross-Entropy vs. KL Divergence

Proposition 9.1 (Relationship to KL Divergence)**.** *The fuzzy set cross-entropy in Equation* (6) *is a sum of* pointwise *KL divergences between Bernoulli distributions. Specifically, for each edge* (i, j)*:*

$$\mathrm{CE}_{ij} = \mathrm{KL}(\mathrm{Bernoulli}(\mu_h(i,j)) \parallel \mathrm{Bernoulli}(\mu_\ell(i,j))) \,.$$

Intuition

Imagine each edge has a biased coin. In the high-D world, the coin for edge (i, j) lands heads with probability $\mu_h(i, j)$. In the low-D world, a different coin lands heads with probability $\mu_\ell(i, j)$. The KL divergence measures how "different" these two coins are. The total cost sums up the coin-mismatch across all edges. Minimizing this cost means making each low-D coin match its high-D counterpart as closely as possible.

Example 9.1. For an edge with $\mu_h = 0.8$:

$$\begin{aligned}
&\text{If } \mu_\ell = 0.8: \quad \mathrm{KL} = 0.8 \log \tfrac{0.8}{0.8} + 0.2 \log \tfrac{0.2}{0.2} = 0 \quad \text{(perfect match)},\\
&\text{If } \mu_\ell = 0.5: \quad \mathrm{KL} = 0.8 \log \tfrac{0.8}{0.5} + 0.2 \log \tfrac{0.2}{0.5} \approx 0.194,\\
&\text{If } \mu_\ell = 0.1: \quad \mathrm{KL} = 0.8 \log \tfrac{0.8}{0.1} + 0.2 \log \tfrac{0.2}{0.9} \approx 1.361 \quad \text{(bad mismatch)}.
\end{aligned}$$

Proof. The KL divergence between two Bernoulli distributions with parameters p and q is:

$$\mathrm{KL}(p \| q) = p \log \frac{p}{q} + (1-p) \log \frac{1-p}{1-q},$$

which is exactly the per-edge term in Equation (6). □ □

Remark 9.1 (Comparison with t-SNE's Cost)**.** t-SNE minimizes $\mathrm{KL}(P\|Q)$ where P and Q are full probability distributions over all pairs. UMAP's cost is a sum of *binary* KL divergences, one per edge, which has several advantages:

1. No normalization over all pairs is needed (t-SNE requires computing $\sum_{k\neq l} q_{kl}$).
2. The repulsive forces are naturally local (via negative sampling), not global.
3. The cost decompozes cleanly into attractive and repulsive components.

10 The Crowding Problem and Heavy-Tailed Kernels

Definition 10.1 (Crowding Problem)**.** When embedding high-dimensional data into low dimensions, points at moderate distances in high dimensions must be placed at relatively larger distances in low dimensions because low-dimensional space has less "room." Formally, the volume of a ball of radius r in $\mathbb{R}^d$ grows as r^d, so for $d \ll D$, the available volume for moderately-distant points shrinks dramatically.

Intuition

Imagine you have 100 friends arranged in a circle around you in a large room (3D space). Now you must arrange them on a line (1D). On a line, there are only 2 directions (left and right), so the friends who were at moderate distances in the room must be placed much farther away on the line to make room for the close friends. If you used a Gaussian kernel (which drops off quickly), these moderate-distance friends would all be squished into the same small region of the line. A heavy-tailed kernel says: "It's OK if points that were moderately far in high-D end up quite far in low-D" — it relaxes the penalty for large low-D distances.

Solution: Use a heavy-tailed kernel in the low-dimensional space (Equation 8). The heavy tails of the Student-t/Cauchy-like kernel allow moderate high-dimensional distances to map to larger low-dimensional distances, providing the extra "room" needed and preventing clusters from collapsing together.

11 Practical Guide

11.1 When to Use UMAP

- **Exploratory data analysis:** Visualising high-dimensional sensor data, embeddings, or feature spaces to identify clusters, outliers, and structure.
- **Preprocessing for downstream tasks:** Reducing dimensionality before clustering (HDBSCAN on UMAP embeddings is a common pipeline).
- **Monitoring:** Tracking drift in robot sensor data, model embeddings, or manufacturing quality over time.
- **Feature extraction:** Using UMAP embeddings as features for lightweight classifiers when computational budget is limited.

11.2 When NOT to Use UMAP

- **When you need exact distances:** UMAP preserves *topology* (neighborhood relationships), not exact distances. If you need distance-preserving embeddings, use MDS or Isomap.

- **When interpretability of axes matters:** UMAP axes have no inherent meaning (unlike PCA components). If strong interpretability is critical we therefore it's better to use linear techniques such as PCA, NMF or pLSA.
- **On very small datasets ($N < 100$):** UMAP needs enough data to estimate local geometry. With too few points, the k-NN graph is noisy and the embedding unreliable.
- **For real-time applications:** UMAP's initial computation (k-NN + spectral + SGD) is not real-time. For streaming data, consider incremental PCA or use UMAP's `transform` method on a pre-fitted model.

11.3 Tuning Guide

Common Mistake: Over-Interpreting Cluster Sizes and Distances

In a UMAP plot, the *relative sizes* of clusters and the *distances between* clusters are NOT reliable. Only the *existence* of clusters and their *neighborhood relationships* are meaningful. Two clusters that appear far apart might not be more different than two clusters that appear close. Always verify quantitative claims with the original high-dimensional data.

Tuning Strategy

1. **Start with defaults** ($k = 15$, `min_dist`$= 0.1$) and look at the result.
2. **If clusters are too fragmented:** Increase k (try $30, 50, 100$). This connects more of the graph and merges spurious sub-clusters.
3. **If everything looks like one blob:** Decrease k (try $5, 10$). This makes the graph more local and reveals finer structure.
4. **If clusters are too tight/overlapping:** Increase `min_dist` (try $0.3, 0.5, 0.8$).
5. **If you need to see internal cluster structure:** Decrease `min_dist` (try $0.0, 0.01$).
6. **Always run UMAP multiple times** with different random seeds. If the structure changes dramatically, it may not be real.

11.4 Debugging Checklist

Debugging UMAP Outputs

1. **All points collapsed to a ball:** Check if your data has very high dimensionality with mostly noise. Try PCA first to reduce to 50–100 dimensions, then UMAP.
2. **Spurious clusters:** May be because of too-small k. Increase k and see if they merge.
3. **Known groups not separated:** The chosen metric may not capture the relevant differences. Try cosine distance for text/embeddings, or preprocess features.
4. **Slow performance:** For $N > 100{,}000$, ensure you're using the approximate k-NN (default). Consider subsampling + `transform`.
5. **Reproducibility:** Set `random_state` for deterministic results.

11.5 UMAP in Robotics and ML Pipelines

Example 11.1 (SLAM and Point Cloud Visualisation)**.** In 3D mapping, LiDAR scans produce high-dimensional feature descriptors for each point (e.g., FPFH features in $\mathbb{R}^{33}$). UMAP can embed these into 2D to visualise which points have similar local geometry (planar surfaces, edges, corners), aiding in loop closure detection and scene understanding.

Example 11.2 (Monitoring Learned Representations)**.** During training of a perception model, periodically compute UMAP on the penultimate layer activations. Healthy training shows class-specific clusters becoming more separated over epochs. If clusters fragment or merge unexpectedly, it signals overfitting or data issues.

Example 11.3 (Anomaly Detection in Sensor Data)**.** Embed multi-sensor time-series windows (e.g., IMU + force/torque) into 2D with UMAP. Normal operation forms a dense cluster; anomalous events (collisions, failures) appear as outliers. This is more interpretable than raw high-dimensional anomaly scores.

11.6 Implementation in .py

```
import umap

umap_model = umap.UMAP(n_components=2, n_neighbors=15)
X_umap = umap_model.fit_transform(X)
```

12 Hyperparameters and Their Effects

Parameter	Default	Effect
`n_neighbors` (k)	15	Controls the balance between local and global structure. Small k emphasizes local structure (tight clusters); large k captures more global topology.
`min_dist`	0.1	Minimum distance between embedded points. Small values create tighter clumps; large values spread points more evenly. Controls a and b in Equation (8).
`n_components` (d)	2	Target dimensionality of the embedding.
`metric`	Euclidean	Distance function in input space. Can be any metric supported by the implementation.
`n_epochs`	Auto	Number of SGD iterations. More epochs yield better convergence.
`negative_sample_rate`	5	Number of negative samples per positive edge in SGD.
`learning_rate`	1.0	Initial SGD learning rate, decayed linearly.

13 Comparison with t-SNE

Aspect	UMAP	t-SNE
Theoretical basis	Riemannian geometry, algebraic topology, fuzzy sets	Probability divergence (KL)
Cost function	Fuzzy set cross-entropy (binary KL per edge)	KL divergence over joint distribution
Symmetrization	Probabilistic union	Averaging
Low-dim kernel	Generalized: $(1 + a\,r^{2b})^{-1}$	Cauchy: $(1 + r^2)^{-1}$
Optimization	SGD with edge sampling + negative sampling	Gradient descent with Barnes–Hut approx.
Global structure	Better preserved (connected components, relative cluster distances)	Primarily local structure

14 Summary of the Complete UMAP Pipeline

Step 1: Input: Data $X \subset \mathbb{R}^D$, parameters k, `min_dist`, d.

Step 2: Compute k-NN graph using approximate nearest neighbor search.

Step 3: Compute ρ_i (nearest-neighbor distance) for each point.

Step 4: Solve for σ_i via binary search on Equation (4).

Step 5: **Compute directed edge weights** $w_{i \to j}$ via Equation (3).

Step 6: **Symmetrize** using probabilistic union (Equation (5)).

Step 7: **Fit** a, b from `min_dist` for the low-dimensional kernel.

Step 8: **Initialize embedding** $Y^{(0)}$ via spectral embedding of the graph Laplacian.

Step 9: **Optimize** via SGD with negative sampling (Algorithm 1).

Step 10: **Output:** Embedding $Y \subset \mathbb{R}^d$.