t-Stochastic Neighbor Embedding

Complete 80-Slide Presentation

Prof.Asc. Endri Raco

Polytechnic University of Tirane

October 2025

What is Dimensionality Reduction?

Definition

Transforming high-dimensional data into lower-dimensional representations while preserving meaningful structure

Why We Need It:

- Visualization: Human perception limited to 3D
- Curse of dimensionality: Distance becomes meaningless in high-D
- Computational efficiency: Reduce processing requirements
- Feature extraction: Identify essential patterns

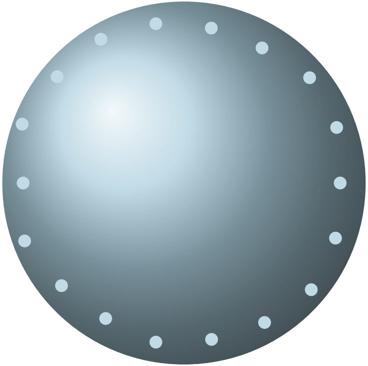
The Central Challenge:

How do we decide what to preserve when we must lose information?

Traditional answer: Preserve distances

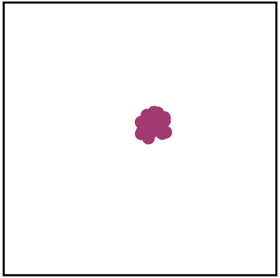
t-SNE answer: Preserve neighborhoods

The Fundamental Challenge of Dimensionality Reduction



784 Dimensions

MNIST digit



2 Dimensions

Your screen

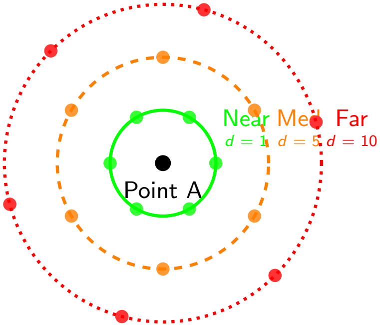
How do we preserve neighborhood relationships

The Crowding Problem: Why Linear Methods Fail

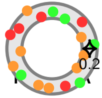
Definition

Crowding Problem: The geometric impossibility of preserving moderate-range distances when projecting from high to low dimensions, causing distinct distance scales to collapse.

High-D Space (10D)



After Linear Projection to 2D



Ratio: 1 : 1.1 : 1.2

The Paradigm Shift: From Geometry to Information

Traditional Methods

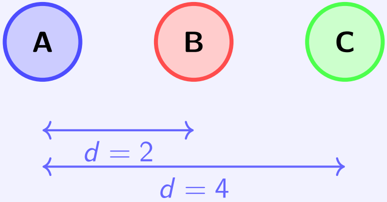
Preserve distances or variance

t-SNE

Preserve probability distributions

The Paradigm Shift: Concrete Example

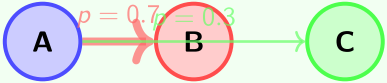
Traditional: Preserve Distances



Problem: All distances treated equally

No context about local density

t-SNE: Preserve Probabilities



Solution: Likelihood encodes context

Adapts to local density automatically

Key Insight: Same distance $\rightarrow$ different probabilities based on neighborhood density

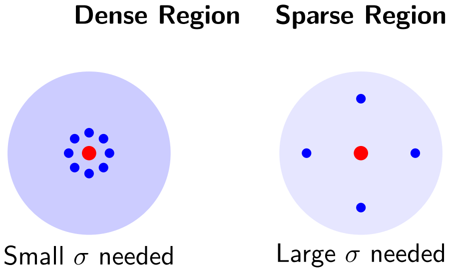
Building Intuition: From Distances to Neighborhoods

The Problem with Raw Distances:

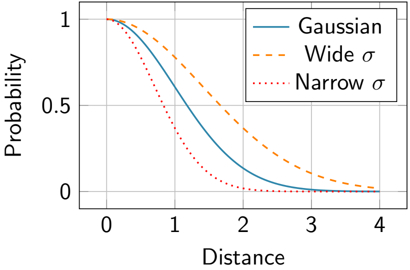
- Point A: 1 unit from B, 10 units from C
- But what if A is in dense region?
- And C is in sparse region?
- Raw distance loses context!

The Solution - Relative Similarity:

- Convert distances to probabilities
- "How likely is B to be A's neighbor?"
- Adapt to local density automatically
- Use Gaussian decay (smooth, differentiable)



Key Idea: Each point gets its own "neighborhood size" (σ_i) based on local density



Key Transformation:

$$p_{j|i} = \frac{\exp(-\|x_i - x_j\|^2/2\sigma_i^2)}{\sum_{k\neq i}\exp(-\|x_i - x_k\|^2/2\sigma_i^2)}$$

Insight: σ_i adapts to local density automatic

Why Gaussian? The Natural Choice for Neighborhoods

What We're Building

Our Goal:

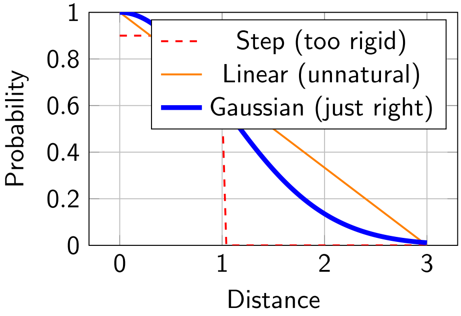
- Convert distances to probabilities
- "How likely is j to be i's neighbor?"
- Must adapt to local density

Three Key Requirements:

1. Smooth decay - no sudden cutoffs
2. Local focus - neighbors matter most
3. Unbiased - don't assume patterns

The Winner: Gaussian $p_{j|i} \propto e^{-d_{ij}^2/2\sigma_i^2}$

Visual Comparison



Analogy: Friendship strength strongest nearby, fading smoothly

The Mathematics Behind Gaussian: Maximum Entropy Principle

The Core Principle

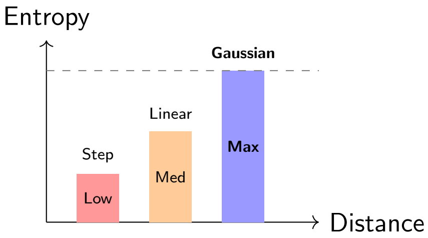
The Question:
Which distribution makes the *fewest assumptions* while matching our constraints?

Answer: Maximum Entropy
The distribution with highest uncertainty (entropy) given the constraints

Why This Matters:

- Most "honest" - no hidden bias
- Adds no assumptions
- Principled approach

Entropy Comparison



Gaussian = Maximum Entropy
Most uncertain = Least biased

The Mathematical Derivation: Problem Setup

Optimization Problem

Maximize Entropy:

$$H(P_i) = -\sum_j p_{j|i} \log p_{j|i}$$

Subject to Constraints:

1. Normalization: $\sum_j p_{j|i} = 1$
2. Fixed Variance: $\sum_j p_{j|i} d_{ij}^2 = \sigma_i^2$

The goal is to find the most unbiased probability distribution ($p_{j|i}$) that meets our constraints.

The Mathematical Derivation: Solution

Solution via Lagrange Multipliers

1. **The Lagrangian:**

$$\mathcal{L} = H(P_i) + \lambda\left(\sum p_{j|i} - 1\right) + \mu\left(\sum p_{j|i} d_{ij}^2 - \sigma_i^2\right)$$

2. **Taking derivatives and solving for $\frac{\partial \mathcal{L}}{\partial p_{j|i}} = 0$ yields the result.**
3. **Result (The Gaussian Distribution):**

$$p_{j|i} = \frac{e^{-\frac{d_{ij}^2}{2\sigma_i^2}}}{\sum_k e^{-\frac{d_{ik}^2}{2\sigma_i^2}}}$$

Perplexity: Setting the Neighborhood Size

The Problem We're Solving

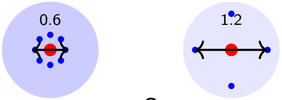
Question: How many neighbors should each point consider?

Challenge: Different regions have different densities!

- Dense areas: Small σ needed
- Sparse areas: Large σ needed

Solution: Perplexity - a user parameter that sets "effective" number of neighbors

Adaptive Neighborhoods



Dense: $\sigma = 0.1$ Sparse: $\sigma = 0.5$

Both have same perplexity $= 5$
Different σ values!

Key Insight: Perplexity is constant across all points, but σ_i adapts to achieve it

Perplexity: The Mathematics and Algorithm

Mathematical Definition

From Shannon Entropy:

$$H(P_i) = -\sum_j p_{j|i} \log_2 p_{j|i}$$

Perplexity:

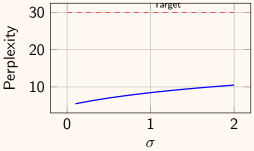
$$\text{Perp}(P_i) = 2^{H(P_i)}$$

Interpretation:

- Perp $= 5 \rightarrow$ "acts like" 5 neighbors
- Perp $= 30 \rightarrow$ "acts like" 30 neighbors

Finding σ_i: Binary Search

Why Binary Search?
Perplexity increases with σ monotonically



Algorithm:

1. Start with $\sigma = 1$
2. Compute current perplexity
3. Too high? $\rightarrow$ Decrease σ

Measuring Information Loss: KL Divergence

What is KL Divergence?

$$\mathrm{KL}(P||Q) = \sum_j p_j \log \frac{p_j}{q_j}$$

Extra bits needed when using Q instead of true P

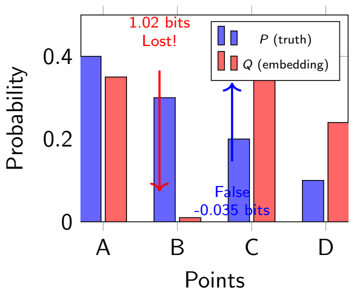
Critical Asymmetry Example

Consider point B with true probability 0.3:

Missing a true neighbor:
True: $p = 0.3$, Embedded: $q = 0.01$
Penalty: $0.3 \times \log(30) \approx$ **1.02 bits**

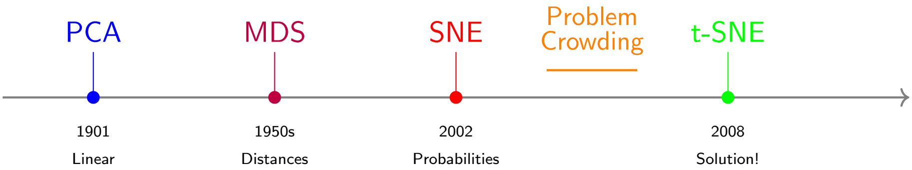
Creating a false neighbor:
True: $p = 0.01$, Embedded: $q = 0.3$
Penalty: $0.01 \times \log(0.033) \approx$ **-0.035 bits**

Visual Example



Original SNE: The Precursor to t-SNE

A Brief History of Dimension Reduction



SNE's Innovation

- First to use probabilities
- Adaptive neighborhoods (σ_i)
- Information-theoretic approach
- KL divergence for optimization

SNE's Fatal Flaw

- Used Gaussian in low-D space
- Cannot represent moderate distances
- Led to "crowding problem"
- All points collapse together

SNE's Mathematics: Where It Went Wrong

The Formulation

- **High-D Similarity (P):**
  Gaussian with adaptive variance σ_i

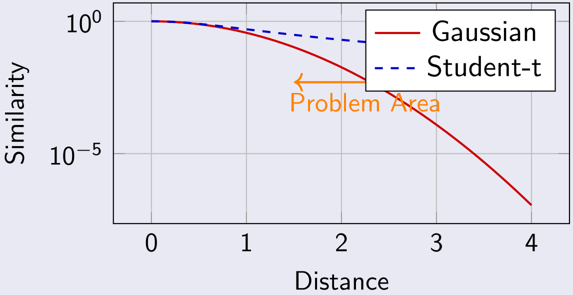
$$p_{j|i} = \frac{\exp(-d_{ij}^2/2\sigma_i^2)}{\sum_k \exp(-d_{ik}^2/2\sigma_i^2)}$$

- **Low-D Similarity (Q):**
  Gaussian with fixed variance

$$q_{j|i} = \frac{\exp(-d_{ij}^2)}{\sum_k \exp(-d_{ik}^2)}$$

- **Cost Function:** $C = \sum_i \mathrm{KL}(P_i || Q_i)$

Why Gaussian Fails in 2D



Problem: Moderate distances in high-D get exponentially tiny similarities in low-D, causing crowding.

The Curse: Why High-D Breaks Our Intuition

The Volume Problem

Question: In a D-dimensional sphere, what fraction of volume is in the outer shell (radius 0.9 to 1.0)?

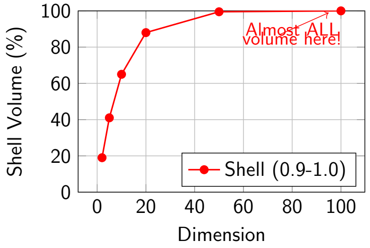
Your intuition (2D):

$$\frac{\pi \cdot 1^2 - \pi \cdot 0.9^2}{\pi \cdot 1^2} = 19\%$$

Reality in high-D:

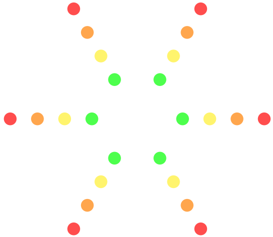
- 5D: 41%
- 10D: 65%
- 50D: 99.5%
- **100D: 99.997%**

Volume Distribution by Dimension



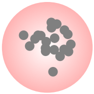
SNE's Fatal Flaw Visualized

High-D: Room for all



Distinct distances

2D with Gaussian: Crushed!



Cannot represent

moderate distances

Solution: Use distribution with heavier tails!

The t-SNE Innovation: Student-t Distribution

The Key Change

SNE (Gaussian in 2D):

$$q_{ij} = \frac{e^{-d_{ij}^2}}{\sum_{k \neq l} e^{-d_{kl}^2}}$$

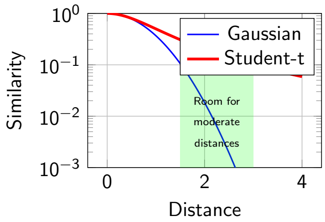
t-SNE (Student-t in 2D):

$$q_{ij} = \frac{(1 + d_{ij}^2)^{-1}}{\sum_{k \neq l}(1 + d_{kl}^2)^{-1}}$$

Mathematical Properties:

- Polynomial decay: $O(d^{-2})$ vs exponential
- Heavy tails preserve moderate distances
- Cauchy distribution (df $= 1$)

Decay Comparison



Quantifying the Solution

Similarity Ratio Analysis

For distances $d_1 = 1$ and $d_2 = 3$:

Gaussian:

$$\frac{q(d_1)}{q(d_2)} = \frac{e^{-1}}{e^{-9}} = e^8 \approx 2981$$

Moderate distance becomes "infinite"

Student-t:

$$\frac{q(d_1)}{q(d_2)} = \frac{1/(1+1)}{1/(1+9)} = 5$$

Moderate distance preserved

600× difference in representation capacity!

The Evolution

Modification 1: Symmetrization

SNE: Asymmetric $p_{j|i} \neq p_{i|j}$

t-SNE: Symmetric $p_{ij} = p_{ji} = \frac{p_{j|i}+p_{i|j}}{2n}$

Why?

- Simplifies gradient (one term instead of two)
- Ensures outliers get fair representation
- More elegant optimization

Modification 2: Student-t in Low-D

SNE: $q_{ij} = \frac{\exp(-d_{ij}^2)}{\sum_{k\neq l}\exp(-d_{kl}^2)}$ (Gaussian)

t-SNE: $q_{ij} = \frac{(1+d_{ij}^2)^{-1}}{\sum_{k\neq l}(1+d_{kl}^2)^{-1}}$ (Student-t)

The Complete t-SNE Algorithm

Input → Probabilities

1. Compute pairwise affinities:

$$p_{j|i} = \frac{\exp(-\|x_i - x_j\|^2/2\sigma_i^2)}{\sum_k \exp(-\|x_i - x_k\|^2/2\sigma_i^2)}$$

2. Symmetrize:

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$$

3. Early exaggeration:

$$p_{ij} \leftarrow 4 \cdot p_{ij} \text{ (first 250 iter)}$$

Optimization

4. Initialize: $y_i \sim \mathcal{N}(0, 10^{-4})$

5. Compute low-D similarities:

$$q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k \neq l}(1 + \|y_k - y_l\|^2)^{-1}}$$

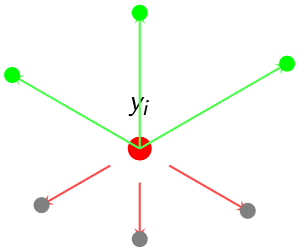
6. Update via gradient:

$$\frac{\partial C}{\partial y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)(1 + d_{ij}^2)^{-1}$$

7. Iterate until convergence

Result: An elegant algorithm that preserves local structure while solving crowding

Understanding the Gradient: Force Interpretation



$$\nabla C = 4 \sum_j \underbrace{(p_{ij} - q_{ij})}_{\text{error}} \underbrace{(y_i - y_j)}_{\text{direction}} \underbrace{(1 + d_{ij}^2)^{-1}}_{\text{adaptive weight}}$$

Insight: Weight term prevents distant clusters from merging

Optimization Trick 1: Early Exaggeration

The Technique

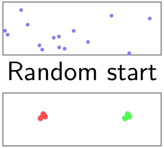
What: Multiply P by 4 for first 250 iterations

$$p_{ij}^{\text{early}} = 4 \cdot p_{ij}$$

Effect on forces:

- True neighbors pull $4\times$ harder
- Clusters form quickly
- Global structure emerges first

Visual Effect



Random start

After 250 iter

Strong initial forces prevent poor local arrangements

Optimization Trick 2: Momentum

The Technique

Update equation:

$$\Delta y_i^{(t)} = -\eta \nabla_i + \alpha(t) \Delta y_i^{(t-1)}$$

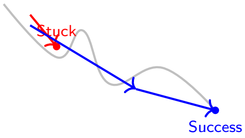
Schedule:

$$\alpha(t) = \begin{cases} 0.5 & t < 250 \\ 0.8 & t \geq 250 \end{cases}$$

Benefits:

- Escapes local minima
- Smooths optimization
- Reduces oscillations

Effect on Optimization



Analogy: Ball rolling downhill - momentum carries it over bumps

Optimization Trick 3: Adaptive Learning Rate

The Technique

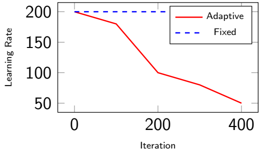
Adaptation rule:

- Same direction: $\eta \times 1.2$
- Direction change: $\eta \times 0.8$
- Min: $\eta_{\text{min}} = 0.01$
- Max: $\eta_{\text{max}} = 1000$

Benefits:

- Fast in flat regions
- Careful near minima
- Automatic adjustment

Learning Rate Evolution



Combined: 5× speedup (5000 → 1000 iterations)

Barnes-Hut: Scaling to Large Datasets

The Algorithm

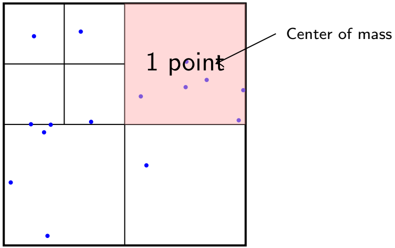
Key Idea: Treat distant clusters as single points

Steps:

1. Build quadtree (2D) or octree (3D)
2. For each point, traverse tree
3. Apply criterion: $\frac{r_{\text{cell}}}{d_{\text{to cell}}} < \theta$
4. If satisfied, use center of mass

Parameter: $\theta \in [0.5, 0.8]$
(higher = faster but less accurate)

Tree Approximation



Points	Exact	Barnes-Hut
1,000	1 sec	0.1 sec
10,000	100 sec	2 sec
100,000	10,000 sec	50 sec

Debugging t-SNE: Visual Diagnosis Guide

Common Problems and Their Fixes

Ball
LR too low
Fix: LR ¿ 100

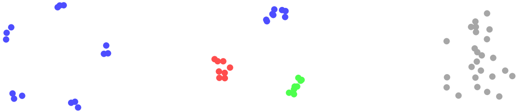
Scattered
LR too high
Fix: LR ¡ 500

Fragmented
Perp too low
Fix: Perp ¿ 20

Good!
Balanced
LR=200, Perp=30

Golden Rule: Run multiple times with different seeds. Trust what's consistent.

Perplexity: Your Main Control Parameter



Perplexity $= 5$ Many fragments | Perplexity $= 30$ Clear clusters | Perplexity $= 100$ One blob

How to Choose?

Rule of thumb:

Perp $= \sqrt{N}/10$ to $\sqrt{N}/2$

(N = number of points)

Examples:

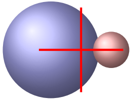
- 1,000 points: 5-15
- 10,000 points: 20-50
- 100,000 points: 50-150

Strategy:

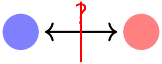
1. Try 3 values (low, mid, high)
2. Look for consistency
3. Trust stable structures

Critical: What You CANNOT Interpret

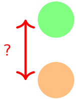
The Three Deadly Sins of t-SNE



Sin #1
Size $\neq$ Count
Visual area meaningless



Sin #2
Distance meaningless
Between clusters



Sin #3
Position arbitrary
No axes meaning

Remember: Only local neighborhoods are meaningful. Everything else is artifact.