t-Stochastic Neighbor Embedding

Complete 80-Slide Presentation

Prof.Asc. Endri Raco

Polytechnic University of Tirane

October 2025

What is High-Dimensional Data? An Intuitive View

The Core Idea

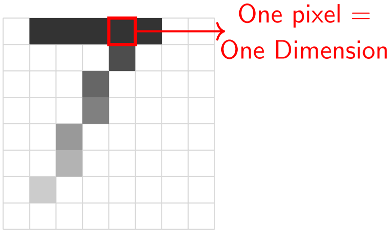
Think of data not as a spreadsheet, but as points in a "feature space." Each feature is a dimension.

Simple Example: Housing Prices (2D)

- Dimension 1: Square Footage
- Dimension 2: Price
- *Easy to plot and see patterns!*

Complex, High-Dimensional Examples:

- **An Image (784-D):** Each pixel's brightness is one dimension.
- **A Customer (100s of D):** Dimensions for age, purchase history, clicks...



Key Idea: We often work with data that has far more dimensions than the 2 or 3 we can perceive. This creates unexpected problems.

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

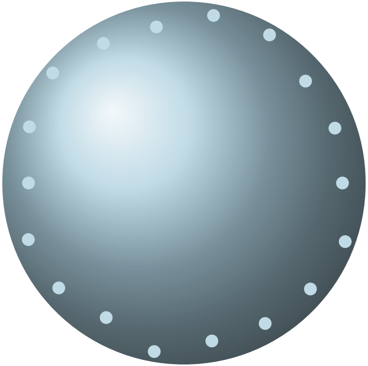
The Central Challenge:

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

Traditional answer: Preserve distances

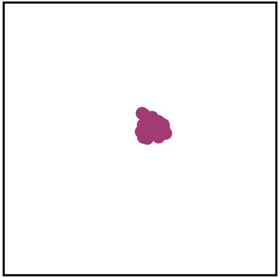
t-SNE answer: Preserve neighborhoods

The Fundamental Challenge of Dimensionality Reduction



784 Dimensions

MNIST digit



2 Dimensions

Your screen

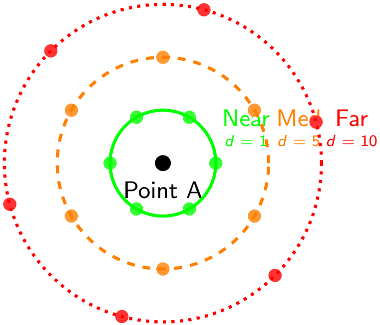
How do we preserve neighborhood relationships

The Crowding Problem: Why Linear Methods Fail

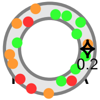
Definition

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

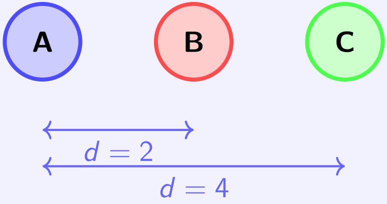
$\downarrow$

t-SNE

Preserve probability distributions

The Paradigm Shift: Concrete Example

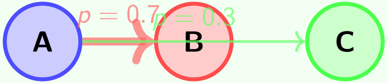
Traditional: Preserve Distances



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 → different probabilities based on neighborhood density

Why Probability Distributions for Neighborhoods?

The Fundamental Question:
How do we mathematically represent "neighborliness" between points?

Option 1: Binary (Yes/No)

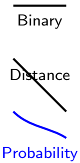
- Neighbor or not neighbor
- Problem: Too rigid!
- Loses gradual relationships

Option 2: Raw Distances

- Use actual measurements
- Problem: No context!
- 5 units means different things

Option 3: Probabilities ✓

- Continuous values [0,1]
- Context-aware (adapts to density)
- Mathematically tractable
- Information theory foundation



Next: How to convert distances to probabilities mathematically

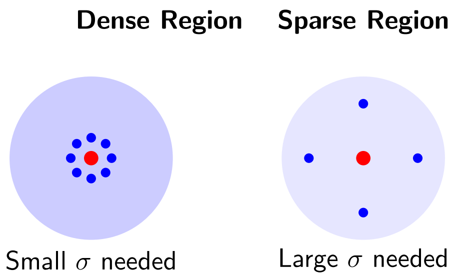
Building Intuition: From Distances to Neighborhoods

The Problem with Raw Distances:

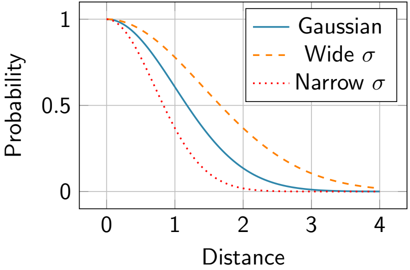
- Point A: 1 unit from B, 10 units from C
- But what if A is in dense region?
- And C is in sparse region?
- Raw distance loses context!

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:

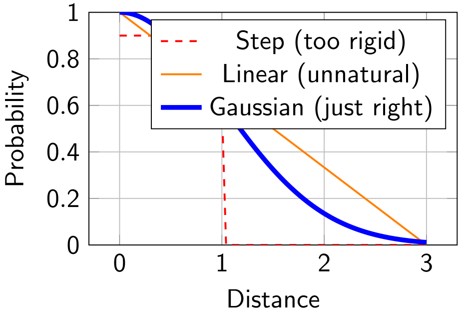
- Convert distances to probabilities
- "How likely is j to be i's neighbor?"
- Must adapt to local density

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

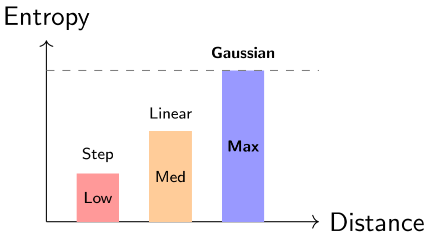
The Question:
Which distribution makes the *fewest assumptions* while matching our constraints?

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

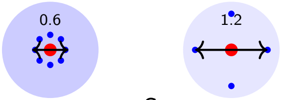
Question: How many neighbors should each point consider?

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:

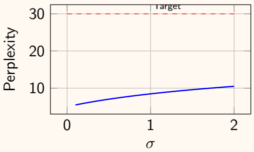
$$\mathrm{Perp}(P_i) = 2^{H(P_i)}$$

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

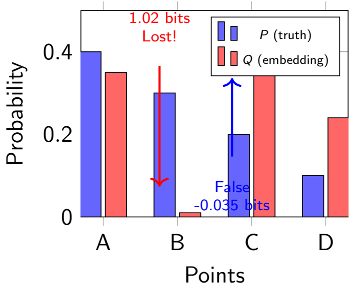
Critical Asymmetry Example

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**

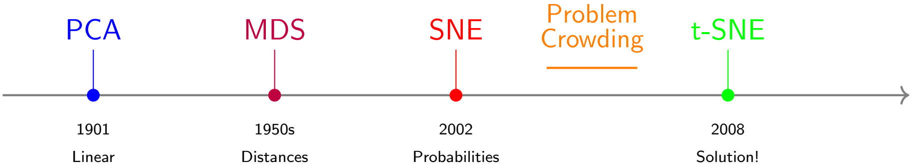
Creating a false neighbor:
True: $p = 0.01$, Embedded: $q = 0.3$
Penalty: $0.01 \times \log(0.033) \approx$ **-0.035 bits**

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

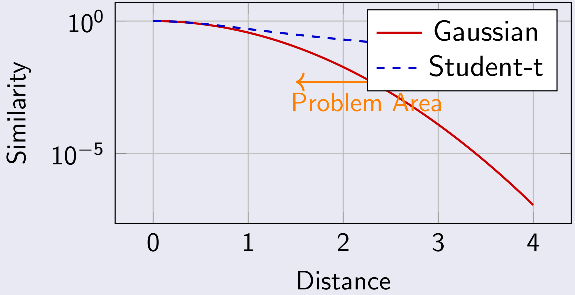
$$p_{j|i} = \frac{\exp(-d_{ij}^2/2\sigma_i^2)}{\sum_k \exp(-d_{ik}^2/2\sigma_i^2)}$$

- **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)?

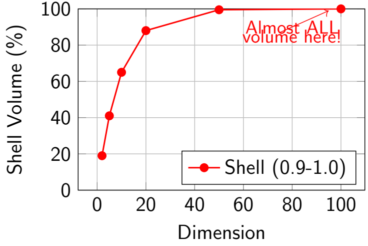
Your intuition (2D):

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

Reality in high-D:

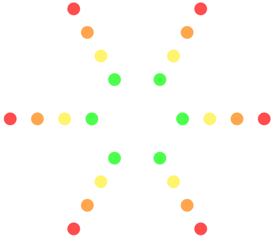
- 5D: 41%
- 10D: 65%
- 50D: 99.5%
- **100D: 99.997%**

Volume Distribution by Dimension



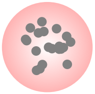
SNE's Fatal Flaw Visualized

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):

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

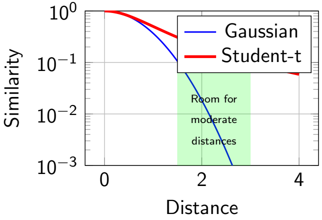
t-SNE (Student-t in 2D):

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

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$\times$ 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 $\rightarrow$ 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:**

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

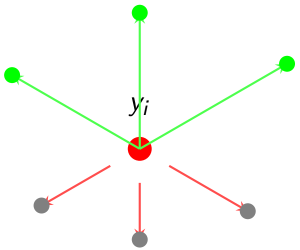
6. **Update via gradient:**

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

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:

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

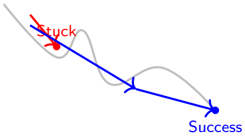
Schedule:

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

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

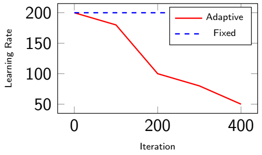
Adaptation rule:

- 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: $5\times$ speedup ($5000 \to 1000$ iterations)

Barnes-Hut: Scaling to Large Datasets

The Algorithm

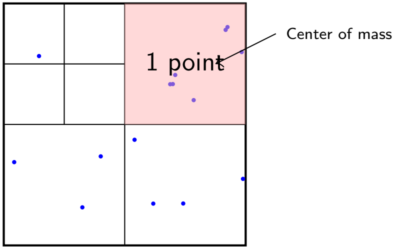
Key Idea: Treat distant clusters as single points

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

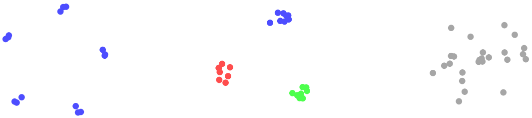
Scattered
LR too high
Fix: LR ¡ 500

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:

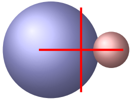
- 1,000 points: 5-15
- 10,000 points: 20-50
- 100,000 points: 50-150

Strategy:

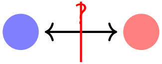
1. Try 3 values (low, mid, high)
2. Look for consistency
3. Trust stable structures

Critical: What You CANNOT Interpret

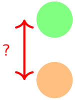
The Three Deadly Sins of t-SNE



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



Sin #2
Distance meaningless
Between clusters



Sin #3
Position arbitrary
No axes meaning

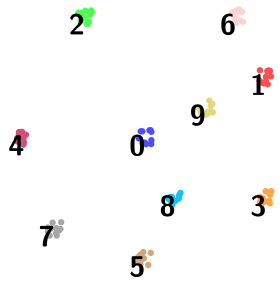
Remember: Only local neighborhoods are meaningful. Everything else is artifact.

MNIST Case Study: Complete Pipeline

Data Preparation

1. **Dataset:**
   - 70,000 handwritten digits
   - 28×28 pixels $= 784$ dimensions
2. **Preprocessing:**
   - Scale pixels to [0,1]
   - PCA to 50D (95% variance)
   - Remove outliers ($> 3\sigma$)
3. **t-SNE Settings:**
   - Perplexity $= 30$
   - Learning rate $= 200$
   - Iterations $= 1000$

Result: Clear Digit Separation



Success indicators:

- Each digit forms cluster
- Similar digits nearby (7 near 1)
- Clear boundaries

Quantitative Validation: Understanding the Metrics

1. Neighborhood Preservation

What it measures:
Do neighbors in high-D stay neighbors in 2D?
Formula:

$$\mathrm{NPr}(k) = \frac{\text{overlap of } k \text{ neighbors}}{k}$$

Good values:

- $\mathrm{NPr}(10) > 0.7$ ✓
- $\mathrm{NPr}(30) > 0.6$ ✓
- $\mathrm{NPr}(50) > 0.5$ ✓

2. Trustworthiness

What it measures:
Are 2D neighbors actually close in high-D?
Range: 0 to 1 (higher = better)
Good values:

- $\mathrm{T}(10) > 0.95$ ✓
- $\mathrm{T}(30) > 0.90$ ✓

Continuity: Similar but checks if high-D neighbors preserved

Rule: If NPr ¿ 0.6 and T ¿ 0.9, your embedding is trustworthy

Stability Analysis: Is Your Embedding Reliable?

Testing Protocol

Steps:

1. Run t-SNE 10 times
2. Different random seeds
3. Same parameters
4. Compare results

Measure correlation:

- Between pairwise distances
- Or cluster assignments

Interpreting Results

$r > 0.9$ Very stable ✓

$r = 0.7 - 0.9$ Acceptable

$r < 0.7$ Unreliable ✗

Example correlation matrix:

Run	1	2	3
1	1.00	0.92	0.89
2	0.92	1.00	0.91
3	0.89	0.91	1.00

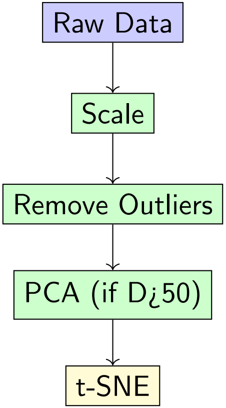
Mean $r = 0.91 \rightarrow$ Very stable!

Critical: Data Preprocessing for t-SNE

Essential Steps

1. **Scaling (CRITICAL!)**
   - Standardize: mean=0, std=1
   - Or normalize to [0,1]
   - Never mix scales!
2. **Dimensionality**
   - If D ¿ 50, use PCA first
   - Keep 95% variance
   - Speeds computation 10×
3. **Missing Data**
   - Impute with median
   - Or remove samples

Preprocessing Pipeline



Modern Alternatives: When to Use What

Aspect	**t-SNE**	**UMAP**	**Choose**
Speed	Slower	5-10× faster	UMAP ✓
Local structure	Excellent	Excellent	Tie
Global structure	Weak	Better	UMAP ✓
Scalability	¡100K	Millions	UMAP ✓
Reproducibility	Random	More stable	UMAP ✓
New points	No	Yes	UMAP ✓
Interpretability	Intuitive	Complex	t-SNE ✓
Fine detail	Better	Good	t-SNE ✓

Use t-SNE when:

- Dataset ¡ 50K points
- Local detail critical
- Publication figures
- Exploring clusters

Use UMAP when:

- Large datasets
- Need speed
- Global structure matters
- Production systems

Mathematical Insight 1: Why Symmetrize?

The Problem

Original SNE uses asymmetric probabilities:

$$p_{j|i} \neq p_{i|j}$$

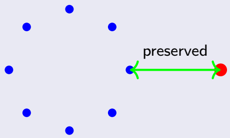
Issue with outliers:

- Outlier $\rightarrow$ others: very small
- Others $\rightarrow$ outlier: normal
- Result: Outliers disappear!

The Solution

Symmetrization:

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



Outlier stays connected

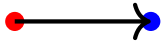
Takeaway: Symmetrization ensures every point gets fair representation

The t-SNE gradient has beautiful structure:

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

Error Term

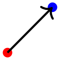
$(p_{ij} - q_{ij})$



How wrong?

Direction

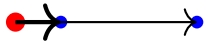
$(y_i - y_j)$



Which way?

Adaptive Weight

$(1 + d_{ij}^2)^{-1}$



How much?

Mathematical Insight 3: Why Exactly df=1?

The Student-t Family

General form with df=ν:

$$q_{ij} \propto (1 + d_{ij}^2/\nu)^{-(\nu+1)/2}$$

Special case $\nu = 1$:

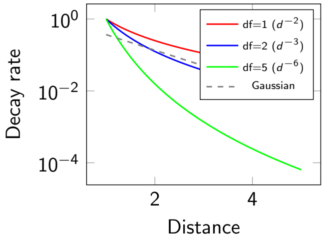
$$q_{ij} \propto (1 + d_{ij}^2)^{-1}$$

This is the Cauchy distribution!

Why df=1 is Perfect

- Heaviest possible tails
- Simplest gradient formula
- Maximum space for moderate distances

Tail Thickness Comparison



df=1 has the slowest decay
= most room for moderate distances

Practical Math: Computational Complexity

Algorithm	**Time**	**Space**	**Practical Limit**
Exact t-SNE	$O(n^2)$	$O(n^2)$	5,000 points
Barnes-Hut	$O(n \log n)$	$O(n)$	100,000 points
FFT-accelerated	$O(n)$	$O(n)$	10 million points

Where Time Goes

- Computing P: Once, $O(n^2)$
- Computing Q: Every iteration
- Gradient: Every iteration
- Total: ~ 1000 iterations

Speed Tips

- PCA to 50D first: $10\times$ speedup
- Barnes-Hut: $50\times$ speedup
- Fewer iterations: $2\times$ speedup
- GPU: 10-200$\times$ speedup

Rule of thumb: 10K points = 1 minute, 100K points = 1 hour (CPU)

When Does the Math Actually Matter?

Math You Can Ignore

For most users:

- Lagrangian derivations
- Exact gradient formulas
- Entropy calculations
- Proof details

Just use the default implementation!

Math You Should Know

For better results:

- Perplexity $\sim$ expected neighbors
- Heavy tails prevent crowding
- Local vs global structure trade-off
- Why preprocessing matters

This helps you debug problems!

The 80/20 Rule:
Understanding 20% of the math gives you 80% of the practical benefit

Real-World Impact: Single-Cell Genomics

The Challenge

Data characteristics:

- 20,000+ genes per cell
- 100,000+ cells per experiment
- 90%+ zeros (sparse data)
- Batch effects between samples

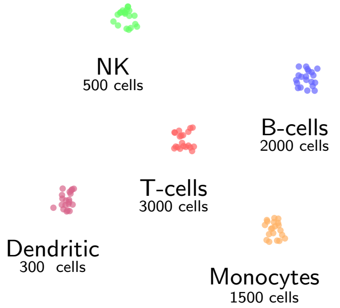
Processing pipeline:

1. Log-normalize counts
2. Select top 2000 variable genes
3. PCA to 50 components
4. t-SNE with perplexity 30-100

Computational requirements:

- Time: 2-4 hours (100K cells)

Discovering Cell Types



Each cluster = cell type

Impact: Found 3 new cell subtypes in 2019

NLP Revolution: Word Embeddings Visualization

The Pipeline

Data preparation:

- Word2Vec/BERT embeddings
- 300-768 dimensions
- Vocabulary: 10K-50K words
- Cosine distance metric

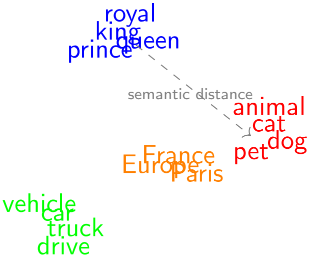
t-SNE parameters:

- Perplexity: 20-50
- Learning rate: 500
- Iterations: 5000
- Metric: cosine (not Euclidean!)

Computational cost:

- 10K words: 5 minutes

Semantic Clusters Revealed



Key insight: Analogies preserved!

king - man + woman $\sim$ queen

Deep Learning: Understanding Neural Networks

Visualizing CNN Features

What to visualize:

- Layer activations (conv5, fc7)
- 512-4096 dimensional vectors
- Per image or per class

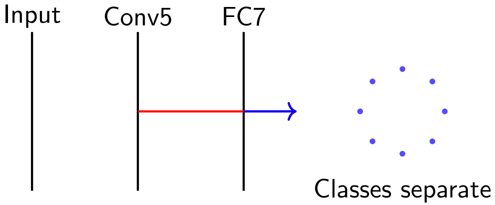
Implementation:

1. Extract features: 1 min/1K images
2. PCA to 50D: 10 seconds
3. t-SNE: 5-15 minutes
4. Total: ¡20 min for 10K images

Hardware requirements:

- GPU for feature extraction
- 16GB RAM minimum

What You Discover



Discoveries made:

- Hierarchy emerges (dogs near wolves)
- Confusion patterns visible
- Dead neurons identified
- Adversarial vulnerabilities found

Beyond Basic t-SNE: Advanced Methods Overview

Choose Your Enhancement Based on Your Need

Your Problem	**Solution**	**Trade-off**
Need to embed new data	Parametric t-SNE	$10\times$ slower training
Data changes over time	Dynamic t-SNE	Complex parameters
Dataset ¿ 100K points	GPU acceleration	Requires CUDA
Need inverse mapping	Parametric t-SNE	Less flexible
Multiple scales important	Multi-scale t-SNE	More parameters
Want global structure	UMAP instead	Different algorithm

Reality check: 90% of users just need standard t-SNE with good parameters

Parametric t-SNE: Learning a Mapping Function

How It Works

Instead of just finding positions, learn a function $f_\theta : \mathbb{R}^d \to \mathbb{R}^2$

Architecture:

- Neural network (3-5 layers)
- Input: high-D data
- Output: 2D coordinates
- Train to minimize t-SNE loss

When to use:

- Streaming data
- Need to embed new points
- Production systems

Practical Details

Advantages:

- Embed new data instantly
- Consistent mapping
- Can learn inverse

Disadvantages:

- $10\times$ slower to train
- Slightly worse quality
- Needs more tuning

Implementation:

- Library: `parametric_tsne`
- Time: 2-5 hours for 50K points
- GPU recommended

GPU Acceleration: Scaling to Millions

Performance Gains

Dataset	CPU	GPU
10K	2 min	10 sec
100K	2 hours	5 min
1M	Days	1 hour

Speedup: 20-200$\times$ typical

Requirements

- NVIDIA GPU ($\geq$ 4GB VRAM)
- CUDA toolkit installed

Implementation Tips

Best practices:

- Batch size = 512-1024
- Use mixed precision
- Monitor GPU memory

Code example:

```
# RAPIDS cuML
from cuml import TSNE
tsne = TSNE(n_components=2,
            perplexity=30,
            method=’fft’)  # Fast!
Y = tsne.fit_transform(X)
```

Cloud options:

- Google Colab (free tier)

Dynamic t-SNE: Visualizing Change Over Time

The Approach

Add temporal coherence to cost:

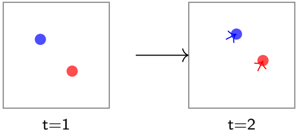
$$C_{\text{total}} = C_{\text{t-SNE}} + \lambda C_{\text{temporal}}$$

Use cases:

- Gene expression over time
- Topic evolution in text
- Learning progression
- Market dynamics

Key parameter:
λ = stability vs accuracy
(0.1-0.5 typical)

Visualization Example

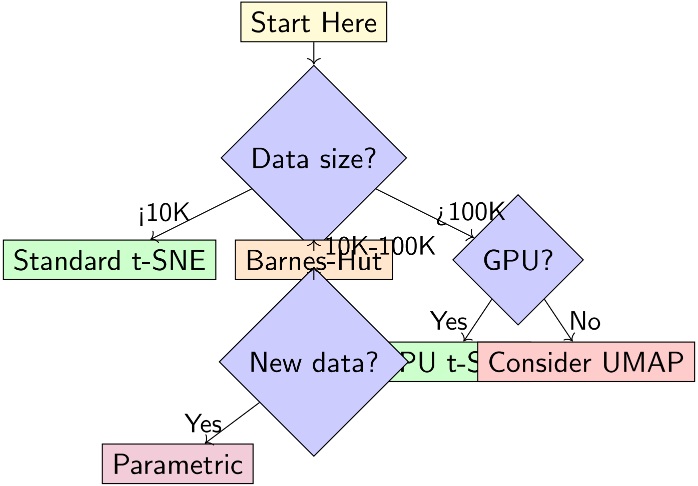


Points move smoothly between frames

Implementation:

- Process all timepoints together
- 2-3× slower than static
- Results: animated visualization

Which Method Should You Actually Use?



Default recipe: Barnes-Hut t-SNE, perplexity=30, iterations=1000

Key Extensions for Specialized Cases

When Standard t-SNE Isn't Enough:
Supervised t-SNE (Fisher Kernel):

- Incorporates class labels
- Modifies: $p_{ij} \times (1 + \lambda)$ if same class
- Better class separation
- Use when: Labels available

Alternative Kernels:

- α-SNE: $(1 + d^2)^{-\alpha}$
- Controls tail heaviness
- Use for specific distance needs

Method	Best For	Cost
Standard	Exploration	$O(n \log n)$
Fisher	Classification	$O(n \log n)$
α-SNE	Custom	$O(n^2)$
Parametric	New data	$O(n) +$ train

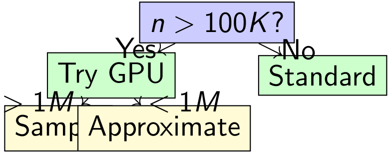
Insight: Most users need only standard t-SN

Scaling to Millions of Points

Three Approaches for Large-Scale t-SNE:

1. **Approximation Methods:**
   - **Random Walk:** Sample neighborhoods stochastically
   - **Landmarks:** Embed subset, interpolate others
   - **FFT:** For $d \leq 3$, use grid interpolation
2. **Computational Optimizations:**
   - **GPU:** 50-100x speedup (RAPIDS, cuda-tsne)
   - **Distributed:** Split across machines
   - **Progressive:** Start with subset, add points
3. **Out-of-Sample Extension:**
   - Train on subset, project new points
   - Use parametric t-SNE or kernel regression

Decision Tree:



Implementation Libraries:

- `openTSNE`: All methods
- `FIt-SNE`: FFT acceleration
- `RAPIDS`: GPU implementation

Numerical Stability and Validation

Critical Implementation Details:

Common Numerical Issues:

- **Overflow:** Use log-sum-exp trick
- **Division by zero:** Add $\epsilon = 10^{-12}$
- **Gradient explosion:** Clip gradients to $[-4, 4]$
- **Poor initialization:** Multiple restarts

Validation Best Practices:

- Run **minimum 5 times** with different seeds
- Report **stability**: std dev of positions
- Check **convergence**: KL divergence plateau
- Measure **trustworthiness** at $k = 10$

Quality Checklist:

- ☐ Perplexity tested: 5-50 range
- ☐ Multiple runs performed
- ☐ Trustworthiness > 0.9
- ☐ No isolated points
- ☐ Convergence achieved
- ☐ Parameters documented

Warning: Never trust a single t-SNE run!

Reporting Requirements:

- State exact parameters used
- Include convergence plots
- Show multiple perplexities
- Provide code/random seeds

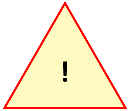
Critical Limitations and Ethical Use

Fundamental Limitations:

- **Non-deterministic:** Different runs $\rightarrow$ different layouts
- **No absolute positions:** Only relative structure matters
- **No cluster sizes:** Visual size $\neq$ actual size
- **Parameter sensitive:** Results change with perplexity

Ethical Responsibilities:

- Report **all** parameters used
- Show **multiple** runs (minimum 3)
- Never cherry-pick best result
- Acknowledge when patterns unclear



Common Misuses:

- Interpreting distances globally
- Comparing cluster sizes
- Single run as "truth"
- Ignoring failed convergence

Ethics: Always provide: data, code, paramet

Key Takeaways: Test Your Understanding

Five Essential Questions:

1. **Why does SNE fail?**
   Hint: Think about available area in 2D for different distance scales
2. **How does Student-t distribution solve the crowding problem?**
   Hint: Compare tail behavior to Gaussian
3. **What does perplexity actually control?**
   Hint: Not just "number of neighbors"
4. **When should you NOT trust a t-SNE visualization?**
   Hint: Multiple warning signs exist
5. **What's the minimum validation needed?**
   Hint: Think runs, parameters, metrics

If you can answer these, you understand t-SNE's core concepts!

Resources and Final Guidance

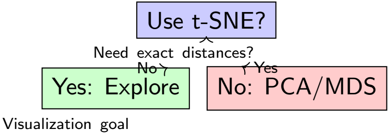
Essential Resources:

- **Original paper:** van der Maaten & Hinton (2008)
- **Interactive guide:** distill.pub/2016/misread-tsne
- **Python:** `scikit-learn`, `openTSNE`
- **R:** `Rtsne`, `tsne`
- **GPU:** `rapids-tsne`, `tsnecuda`

Getting Started Checklist:

1. Start with perplexity $= n/100$
2. Run at least 3 times
3. Try perplexities: 5, 30, 50, 100
4. Check convergence (1000+ iterations)
5. Validate with known structure

Decision Flowchart:



Remember:

t-SNE is a **tool for exploration**, not proof
Trust **consistent patterns** across runs
Always **validate** findings with other methods

Thank you! Questions welcome

Contact: eraco@polytechnic.cat