A Gentle Introduction to Stochastic Neighbor Embedding (SNE)

Based on the Tutorial by B. Ghojogh, A. Ghodsi, et al.

Data Science University

September 27, 2025

The Big Picture

[cite$_s tart]Wehavehigh-dimensionaldata,$\{x_i\}_{i=1}^n$ where each $x_i \in \mathbb{R}^d$[cite: 46]. Think of data with hundreds of features. It's impossible to "see".

[cite$_s tart]Ourgoalistocreateameaningfullow-dimensional" map" ofthisdata[cite : 46].
We want to find a new set of points, $\{y_i\}_{i=1}^n$ where each $y_i \in \mathbb{R}^p$[cite: 46].
[cite$_s tart$]

Usually, the new dimension p is 2 or 3 so we can make a scatter plot[cite: 47].

Stochastic Neighbor Embedding (SNE)

[cite$_s$*tart*]**SNE***isa* $*$ $*$*manifoldlearning* $*$
$*$*methodfordimensionalityreduction*[*cite* : 27].

The Core Idea

[cite$_s$*tart*]*Insteadofthinkingaboutneighborsinastrict" yes*/*no" way*, *SNEusesprob*
32].[*cite$_s$tart*]*Everypointconsiderseveryotherpointitsneighbor*, *butwithacertainp*
21].[*cite$_s$tart*]*Closerpointsgetahigherprobability*[*cite* : 31].

[cite$_s$*tart*]*Thealgorithm′smainjobistoarrangethepointsinthelow* −
dimensionalmapsothattheseneighborhoodprobabilitiesarepreservedascloselyaspo
33].

Our Journey Today

1. **Part 1: Stochastic Neighbor Embedding (SNE)**
   - How do we define "neighborhood probability"?
   - How do we measure if our map is good?
   - How do we optimize the map?
2. **Part 2: Symmetric SNE** (In the next section!)
3. **Part 3: The Crowding Problem & t-SNE** (Later!)

Let's Dive In!

Step 1: Probabilities in the Original Space

[cite$_s tart]SNEplacesa**Gaussiandistribution**centeredoneverydatapoint$,$\mathrm{x}_i$[cite: 53].

ite$_s tartThisGaussianmeasurestheprobabilitydensityforanyotherpoint$,$\mathrm{x}_j$, to be a neighbor of x_i[cite: 53].

[cite$_s tart]ite_s tartPointsthatarefartherawaywillnaturallyhavealowerprobabil$
53].

[cite$_s tart]Thekeyideaistoconvertthe**Euclideandistances**betweenpointsinto**conditionalprobabilities**[cite: 60]$.

The Math: Calculating $p_{j|i}$

The probability that point x_j is a neighbor of x_i is given by $p_{j|i}$.

Similarity Score

[cite$_s$*tart*]*First*, *wecalculateasimilarityscorebasedontheGaussianfunction* :exp($-$
$x_j||^2/2\sigma_i^2$)[cite: 61, 62].

Softmax Normalization

[cite$_s$*tart*]*Toturnthesescoresintorealprobabilitiesthatsumto*1, *weusea*" *softmax*"
59].[*cite$_s$tart*]*Wedivideeachscorebythesumofallpossiblescores*(*foragiven*i)[*cite* :
61].$p_{j|i} = \frac{\exp(-||x_i - x_j||^2/2\sigma_i^2)}{\sum_{k \neq i} \exp(-||x_i - x_k||^2/2\sigma_i^2)}$

[cite$_s$*tart*]*Thisistheconditionalprobabilityof* j*given*i[*cite* :
60].[*cite$_s$tart*]*Notethat*$p_{i|i}$ is set to 0[cite: 92].

What is that σ_i^2 (Variance)?

[cite$_s tart]Theterm\sigma_i^2$ is the variance of the Gaussian centered on point x_i[cite: 73].

- It controls the "width" of the Gaussian curve.
- A small σ_i means only very close points are considered neighbors.
- A large σ_i means even distant points can be neighbors.

How is it chosen?

[cite$_s tart]Crucially, eachpointx_i$ gets its own σ_i[cite: 73].
[cite$_s tart]It'sfoundusingabinarysearchtomatchauser - definedvaluecalled *$
$*perplexity *$
$*, whichyoucanthinkofasthedesirednumberofeffectiveneighbors[cite : 73]$.

This allows the algorithm to adapt to regions of varying data density.

Step 2: Probabilities in the Map

Now, we do the exact same thing for our low-dimensional points, y_i!

- Our goal is to arrange the y_i points on a 2D or 3D map. [cite$_s tart$]

- We also define a conditional probability, $q_{j|i}$, that y_j is a neighbor of y_i in this new map[cite: 75].

A Simpler Gaussian

We use the same softmax approach, but with a much simpler Gaussian. [cite$_s tart$]$Wefixthevariancetobe \sigma^2 = 1/2$[cite: 81].

$$q_{j|i} = \frac{\exp(-||y_i - y_j||^2)}{\sum_{k \neq i} \exp(-||y_i - y_k||^2)}$$

Why fixed?
[cite$_s tart$]$Becausethescaleofthemapissomethingwecanchoose, sowedon'tneedto$
81].

Summarizing Our Two Worlds

We now have two sets of probabilities.

High-D World (Input)

Probabilities $P_i = \{p_{1|i}, p_{2|i}, \ldots\}$

- Based on original data x_i.
- Variances σ_i^2 are learned.

Low-D World (Map)

Probabilities $Q_i = \{q_{1|i}, q_{2|i}, \ldots\}$

- Based on map points y_i.
- Variance is fixed.

The Main Goal

[cite$_s tart]Wewanttoarrangethey_i$ points so that the probabilities Q_i become as similar to the probabilities P_i as possible for all data points[cite: 82].

Step 3: How Do We Measure "Similarity"?

How do we measure the difference between two probability distributions?

The Kullback-Leibler (KL) Divergence

[cite$_s$*tart*]*TheKLDivergenceisastandardwaytomeasurehowoneprobabilitydistribu*
83].

[cite$_s$*tart*]*Ourtotal* $**$*costfunction* $*$
$*$(C)*isthesumofalltheKLdivergencesforeverypoint*[*cite* : 83].$C = \sum_{i=1}^{n} KL(P_i||Q_i)$ *Whichexpandsto* : $C = \sum_{i=1}^{n} \sum_{j \neq i} p_{j|i} \log \frac{p_{j|i}}{q_{j|i}}$
Our task: Find the arrangement of $\{y_i\}$ that **minimizes this cost**!

Intuition on the Cost Function

$$C = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}}$$

Let's break this down.

ite$_s tartThecostfunctionheavilypenalizesputtingpointsfarapartinthemap(sm$
when they were close in the original data (large $p_{j|i}$)[cite:
613].ite$_s tartAlargemismatchwhere$p$_{j|i}$ is large creates a large
cost.ite$_s tartItcareslessaboutputtingpointsclosetogetherinthemap(large$q$_{j|i}$
when they were far apart in the data (small
$p_{j|i}$).ite$_s tartThismeansSNEfocusesmoreonpreservingthe *$
$* localstructure * * ofthedata.$

Step 4: Finding the Best Map

How do we find the positions of y_i that minimize our cost function C?

Gradient Descent!

This is a classic optimization problem.
[cite$_s tart]We'llusethegradientdescentalgorithm[cite : 68$].

The process:

1. Start with a random arrangement of the y_i points on the map.
2. Calculate the "gradient" of the cost function. The gradient tells us, for each point y_i, which direction to move it to decrease the cost the most.
3. Take a small step in that direction.
4. Repeat steps 2 and 3 many times until the map settles down.

The SNE Gradient

[cite$_{s}tart]Thederivative(orgradient)ofthecostfunctionCwithrespecttothepositio$
is surprisingly elegant[cite: 89].

The Gradient Formula

$$\frac{\partial C}{\partial y_i} = 2\sum_j ((p_{j|i} - q_{j|i}) + (p_{i|j} - q_{i|j}))(y_i - y_j)$$

This equation looks complex, but the intuition is simple. It's like a system of springs connecting all the points!

Intuition: A System of Springs

$$\frac{\partial C}{\partial y_i} = 2 \sum_j (\dots)(y_i - y_j)$$

Think of the term $(\dots)$ as the "stiffness" of a spring between points y_i and y_j.

- If points i and j should be closer ($p_{j|i}$ is large but $q_{j|i}$ is small), the "stiffness" is positive. This creates an **attractive force**, pulling y_i and y_j together.
- If points i and j are too close ($p_{j|i}$ is small but $q_{j|i}$ is large), the "stiffness" is negative. This creates a **repulsive force**, pushing them apart.

The algorithm finds the equilibrium state of this complex spring system!

T

- $y_i^{(t-1)}$ is the old position.
- $\frac{\partial C}{\partial y_i}$ is the gradient (the direction of "steepest ascent"). [cite$_s tart$]

- η is the **learning rate**: a small number that controls how big of a step we take[cite: 149].

b make the optimization faster and avoid getting stuck in bad local minima, we can add a **momentum** term[cite: 143, 148].

The Idea

[cite$_s$ $tart]Theupdateforthecurrentstepshouldalsodependalittleontheupdatefrom$ $previous * step[cite:$ $140].It'slikegivingtheballrollingdownthehillsomemomentum, soitdoesn'tgetstuc$

The update rule with momentum is:

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

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

[cite$_s$ $tart]Here, \alpha(t)$ is the momentum term, which can change over time[cite: 140, 145].

Another Trick: A Bit of Randomness

Avoiding Local Optima

[cite$_s$*tart*]*Intheearlystagesofoptimization*, *it′shelpfultoaddasmallamountofrand*
after each iteration[cite: 150].

ite$_s$*tart***Why?** *Thishelps*" *shake*" *thepointsoutofpoorlocalarrangementsand*
151].

SNE: The Story So Far

- For every pair of points in the high-D space, calculate the conditional neighborhood probability $p_{j|i}$ using Gaussians with adaptive variances[cite: 61, 73]. [cite$_s tart$]
- Start with a random scatter plot of points $\{y_i\}$ in the low-D space[cite: 139]. [cite$_s tart$]
- Calculate the conditional neighborhood probability $q_{j|i}$ for these map points using Gaussians with a fixed variance[cite: 77, 81]. [cite$_s tart$]
- Calculate the gradient of the KL divergence cost function[cite: 90]. [cite$_s tart$]
- Update the positions of all y_i points by taking a small step in the direction of the negative gradient, using momentum[cite: 140].
- Repeat for many iterations until the map is stable.

Next Steps

SNE is a fantastic idea, but it has a few wrinkles.

- The probabilities $p_{j|i}$ are not symmetric. This makes the gradient a bit messy.
- It suffers from something called the "Crowding Problem."

C

oming up next: We'll see how **Symmetric SNE** and the revolutionary **t-SNE** solve these problems!