t-SNE: Stochastic Neighbour Embedding

Visualisation of whole process:
https://askadvaith-t-sne-visualisation.vercel.app/

- Dim reduction that focuses on capturing local clusters

① Calculate pair-wise distances for every point

$$d_{ij} = \|x_i - x_j\| = \sqrt{\sum_k (x_{ik} - x_{jk})^2}$$

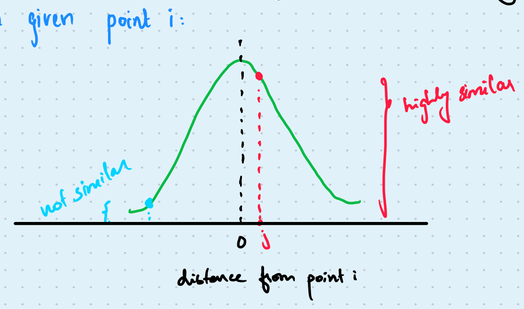
Hyperparameter: Perplexity
Defines local neighbourhood size, i.e., how many neighbours are expected for a given point.

② Create normal distributions for every point.

- Determine σ for every point
- ↑ Density = σ↓ (lower spread to weight similarity correctly)
- Perplexity plays a role → defines expected no. of neighbours → σ defines concept of neighbourhood

③ Calculate similarity to every other point using distance

For given point i:



$$p(j|i) = \frac{\exp\left(\frac{-d_{ij}^2}{2\sigma_i^2}\right)}{\sum_{k \neq i} \exp\left(\frac{-d_{ik}^2}{2\sigma_i^2}\right)}$$

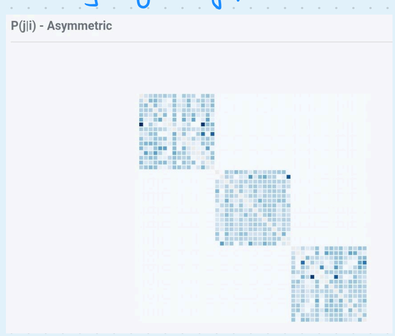
We thus construct a matrix P after doing this for every point. This matrix contains the similarity of every point to each other

RBF for Gaussian:

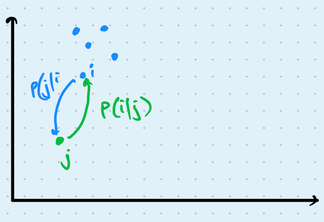
$$\varphi(x) = \exp\left(\frac{-x^2}{2\sigma^2}\right)$$

value determined by distance of x from origin

P(j|i) - Asymmetric



④ Symmetrize P



From earlier, not necessary that

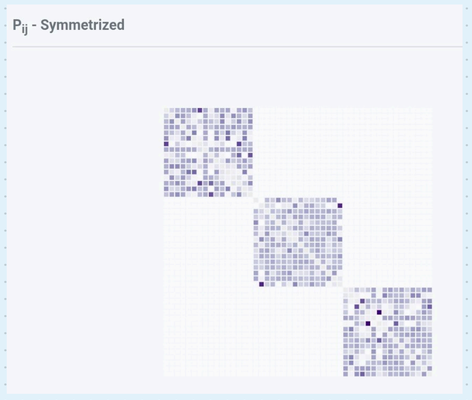
$$P(j|i) = P(i|j)$$

Depends on density etc.

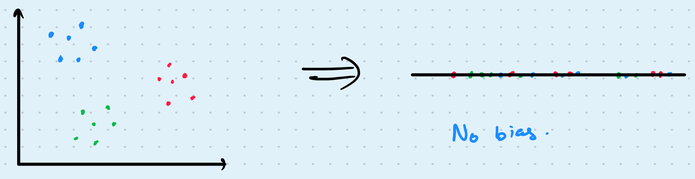
Symmetrize:

$$P(ij) = \frac{P(i|j) + P(j|i)}{2N}$$

P_{ij} - Symmetrized



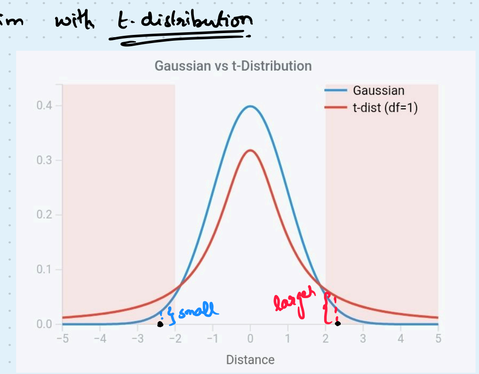
⑤ Randomly init lower dimension representation



No bias.

⑤ Do the same similarity calculation in the lower dim with t-distribution

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



Why t-distribution?

Heavier tails → less similar points given higher probability

↓

prevents distant points from all bunching together

"Crowding problem" } not a problem in high D because more "space"/"volume" for points

⑥ Take matrix Q and make it match P

KL divergence between the two → gradient gives direction to minimise difference between P and Q.

Attractive force: $P_{ij} > Q_{ij}$ → points should be closer

Repulsive force: $P_{ij} < Q_{ij}$ → points should be farther

$$\text{Loss} = KL(P\|Q) = \sum_{i \neq j} P_{ij} \log\left[\frac{P_{ij}}{q_{ij}}\right]$$

} Minimise by gradient descent

When to use	When to stay away
• Exploratory data analysis and visualization	• Need to preserve global structure
• Identifying clusters and local structure	• Want deterministic, reproducible results
• Understanding data distribution	• Working with very large datasets (>10k points)
• Preliminary analysis before other methods	• Need to interpret distances quantitatively

Self Organising Maps (SOM)

- low dim (2D) map that preserves topological relationships
- Map high-D data onto a grid of neurons.
- Similar inputs → activate nearby neurons

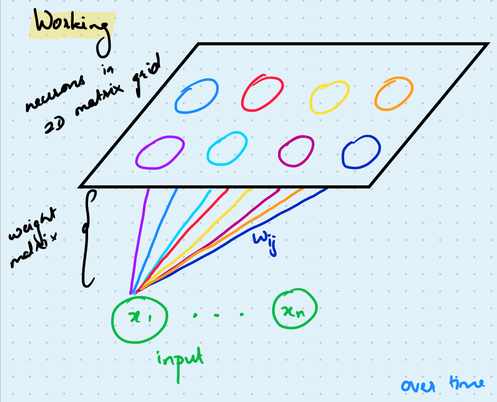
Differences

unlike PCA: nonlinear

unlike t-SNE: creates regular grid structure

unlike clustering: preserves neighbourhood relationships

Working



w_{ij} = weight/code/prototype vector associated with neuron ij

↓ initialised at random

↓ influenced by training inputs/attributes, pulled in diff directions ← over time finds the location that compromises between attribute vectors

Stochastic learning algorithm

```
for each w_i ∈ W
    init w_i
while !(stop condition):
    select random i/p vector from X
    find best neuron for x_i → find closest w_i } competition
    for each w_i ∈ W:          } cooperation + adaptation
        update w_i
    end for
end while
```

Batch learning algorithm

```
for each w_i ∈ W:
    init w_i
while !(stop condition):
    for each w_i ∈ W:
        X_i ← φ   → set of vectors associated w/ w_i → associated with i^th neuron
        if x_i ∈ (neighbourhood of w_i):
            add x_i to X_i
        end if
    end for
    for w_i ∈ W:
        w_i ← update w/ mean over X_i
    end for
end while
```

Outcomes

Topological map

- Winning neuron = $\text{argmin}_i |x - w_i|$
- Adapt weights:

$$w_n' = w_n + \eta(t)\, h_{in}((x - w_n))$$

→ "pull" towards neuron n

η: learning rate
t: time step
neighbourhood function

$t \uparrow \Rightarrow \eta \downarrow$, $h(i) \downarrow$

- Closer neuron = stronger response to pattern

⇒ topological map.

Dimensionality reduction

Everything goes to 2D / low-dim grid

Phases

① Competition

$\text{argmin}_i |x - w_i|$

neuron closest → Best Matching Unit (BMU)

② Cooperation

Find neighbours using h_{ij}

value ↓ as distance ↑ ; symmetrical about winning neuron

③ Adaptation

$$w_{j(n+1)} = w_{j(n)} + \alpha\, h_{ij}(d(x, w_{j(n)}))$$

fn. of distance b/w i, j neurons

Additional notes

- Best lattice dim = $5\sqrt{N}$ → total datapoints
- Very slow, unsupervised
- Not generative

When to use	When to stay away
• Want interpretable, visual representation	• Need exact distance preservation
• Need discrete clustering with spatial relationships	• Want continuous embedding space
• Data has natural neighborhood structure	• Working with very high-dimensional data
• Want to understand data distribution	• Need fast processing for large datasets
• Need robust method for noisy data	• Require deterministic results