# t-SNE Notes

Pavlin Poličar

## 1 t-SNE

t-SNE was presented in [Maaten and Hinton, 2008] and aims to preserve local structure of high dimensional spaces $X$ with some low dimensional embedding $Y$. First for each point $i$, we find its nearest nearest neighbours and compute the probability of this point $p_j$ based on the PDF of a Gaussian centred on the point $i$:

$$p_{j|i} = \frac{\exp\left(-||\mathbf{x}_i - \mathbf{x}_j||^2 / 2\sigma_i^2\right)}{\sum_{k \neq i} \exp\left(-||\mathbf{x}_i - \mathbf{x}_k||^2 / 2\sigma_i^2\right)} \tag{1}$$

where $\sigma_i$ is the bandwidth of the Gaussian density. These bandwidths are controlled by the "perplexity" parameter. Perplexity can be thought of as a continuous analogue to the number $k$-nearest neighbours:

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

where $H(P_i)$ is the Shannon entropy of the distribution $P_i$.

t-SNE actually doesn't use Equation 1 directly, but symmetrizes this conditional probability, so the actual $p_{ij}$s used by t-SNE are

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2} \tag{3}$$

In their experiments, van der Maaten et al. found that this doesn't affect embedding quality and simplifies the gradient expression.

Similarly, we represent the embedding $Y$ as a probability distribution. In the original SNE paper [Hinton and Roweis, 2003], a Gaussian was used, however this often led to the crowding problem, where all the points were clumped into a single ball in a single point in space. t-SNE, as the name would suggest, uses a Student-t distribution, therefore the probability density of $Y$ is

$$q_{ij} = \frac{\left(1 + ||\mathbf{y}_i - \mathbf{y}_j||^2\right)^{-1}}{\sum_{k \neq l} \left(1 + ||\mathbf{y}_k - \mathbf{y}_l||^2\right)^{-1}} \tag{4}$$

We now have two probability distributions over the local point affinities. Now we'd like some way to match these two distributions, so the local structure of $X$ is reflected in $Y$. A natural way of doing this is to use Kullback-Leibler divergence (from here on referred to as the KL divergence), which is defined as

$$KL(P \mid\mid Q) = \sum_{ij} p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{5}$$

Our goal is to minimize this error $C$, so we can take the derivative and obtain

$$\frac{\partial C}{\partial \mathbf{y}_i} = 4 \sum_{j \neq i} (p_{ij} - q_{ij}) (\mathbf{y}_i - \mathbf{y}_j) \left(1 + ||\mathbf{y}_i - \mathbf{y}_j||^2\right)^{-1} \tag{6}$$

This is t-SNE in essence. In practice various tricks are used to speed up convergence e.g. using a momentum term helps a lot. The embedding $Y$ is typically initialized using an isotropic Gaussian with small variance (e.g. 0.01). Often times, PCA is used for initialization. This can sometimes be problematic if the PCA embedding provides very scattered embeddings (sometimes most points are clumped to one side with very long stretched out tails). In these cases, using a random initialization produces better embeddings.

## 2 Performance improvements

It quickly became apparent that t-SNE, while nice, was infeasible to run for larger data sets, because of its quadratic time complexity $\mathcal{O}(n^2)$ (due to the normalization term in $q_{ij}$).

For convenience, we will write the gradient in a different form seen in many papers, that makes the attractive and repulsive forces clearer.

$$\frac{\partial C}{\partial \mathbf{y}_i} = 4 \sum_{j \neq i} (p_{ij} - q_{ij}) (\mathbf{y}_i - \mathbf{y}_j) \left(1 + ||\mathbf{y}_i - \mathbf{y}_j||^2\right)^{-1} \tag{7}$$

Notice that the right most term is just the unnormalized $q_{ij}$

$$= 4 \sum_{j \neq i} (p_{ij} - q_{ij}) (\mathbf{y}_i - \mathbf{y}_j) \left(1 + ||\mathbf{y}_i - \mathbf{y}_j||^2\right)^{-1} \frac{Z}{Z} \tag{8}$$

Where $Z$ is the normalization term of $Q$: $Z = \sum_{k \neq l} \left(1 + ||\mathbf{y}_k - \mathbf{y}_l||^2\right)^{-1}$

$$= 4 \sum_{j \neq i} (p_{ij} - q_{ij}) \, q_{ij} Z \, (\mathbf{y}_i - \mathbf{y}_j) \tag{9}$$

$$= 4 \left( \sum_{j \neq i} p_{ij} q_{ij} Z \, (\mathbf{y}_i - \mathbf{y}_j) - \sum_{j \neq i} q_{ij}^2 Z \, (\mathbf{y}_i - \mathbf{y}_j) \right) \tag{10}$$

which can in turn be throught of as attractive and repulsive forces

$$= 4 \left(F_{\text{attr}} + F_{\text{rep}}\right) \tag{11}$$

## 2.1 Landmark points

In fact, van der Maaten and Hinton provide a solution to this in their original paper: instead of visualizing all the points, embed only a sample of carefully chosen landmark points. The points must be carefully chosen because a random subset may not properly describe the manifold. First, we construct the k-neighbourhood graph on all the points. Next, they approximate the $P$ of the landmark points using random walks across the neighbourhood graph. Then, we proceed with t-SNE on the landmark points.

## 2.2 Approximating P

An observation made in [Van Der Maaten, 2014] was that since we use a Gaussian kernel for $P$, points further than 3 standard deviations from the mean have almost zero probabilities, and as such, do not affect the KL divergence term. Therefore, no harm would come if we simply ignored these terms. In practice, this means that we only compute the $p_{ij}$ terms for $\lfloor 3u \rfloor$ neighbours, where $u$ is the perplexity.

In [Van Der Maaten, 2014], exact nearest neighbours are used. These can be efficiently computed in $\mathcal{O}(n \log n)$ using tree structures, thus reducing the complexity from $\mathcal{O}(n^2)$ needed for pairwise distances.

The preferred exact nearest neighbour method are vantage point trees (also referred to as VP trees). [Yianilos, 1993] presented VP trees and compared their performance to another popular tree based nearest neighbour search method – KD trees. VP trees were shown to require far fewer queries when dealing with high dimensions, as t-SNE often does.[Van Der Maaten, 2014] also provide a comparison with dual-trees, where VP trees, again, perform favourably.

More recently, it was shown in [Linderman et al., 2017] that approximate nearest neighbours perform just as well. Approximate nearest neighbour algorithms are often orders of magnitude faster than exact nearest neighbour search, allowing us to scale this step to much larger data than before.

## 2.3 Barnes-Hut

Having drastically improved the complexity of $F_{\text{attr}}$, we are still left with quadratic $\mathcal{O}(n^2)$ complexity for $F_{\text{rep}}$, required by the normalization term $Z$.

[Van Der Maaten, 2014] notice that computing $F_{\text{rep}}$ can be posed as an N-body simulation problem. This problem has been addressed physics simulation community and can be efficiently solved in $\mathcal{O}(n \log n)$ time using Barnes-Hut trees. The main idea behind this approximation is that clusters of points far away for the current point $i$ will have similar contribution, therefore we can summarize entire regions of space (denoted cells in the following) by computing the center of mass of the region $\mathbf{y}_{\text{cell}}$, computing the interaction between $i$ and $\mathbf{y}_{\text{cell}}$ and adding this interaction up $N_{\text{cell}}$ times, where $N_{\text{cell}}$ is the number of points in the given region, given they are far enough from our query point $i$. The space is split into square regions and represented by a space splitting tree (a quad-tree in 2D and an oct-tree in 3D) which can be built in linear time.

The "far enough" is determined by a parameter $\theta$, which controls how accurate our estimations are. If the following relation holds, then the cell is summarized

$$\frac{r_{\text{cell}}}{||\mathbf{y}_i - \mathbf{y}_{\text{cell}}||^2} < \theta \tag{12}$$

where $r_{\text{cell}}$ represents the length of the diagonal of the cell. Larger values of $\theta$ produce more accurate estimates. Setting $\theta$ to 0 computes all the pairwise interactions as the condition can never be met. Scikit-learn recommends values between 0.2 and 0.8, as anything above and below that quickly result in long computation time and large error, respectively.

It is worth noting that this approach scales fairly well for 1, 2 and 3 dimensions, but further than that, the complexity becomes prohibitively expensive. This is not really an issue, since we humans can only perceive 3 dimensions, and most visualizations are 2D.

## 2.4 FFT Accelerated Interpolation

We can write an equivalent expression for the repulsive forces

$$F_{\text{rep}} = \sum_{j \neq i} q_{ij}^2 Z \left(\mathbf{y}_i - \mathbf{y}_j\right) \tag{13}$$

Plugging in the expressions for $q_{ij}$ and $Z$

$$= \sum_{j \neq i} \frac{\left(1 + ||\mathbf{y}_i - \mathbf{y}_j||^2\right)^{-2}}{\sum_{k \neq l} \left(1 + ||\mathbf{y}_k - \mathbf{y}_l||^2\right)^{-2}} \frac{\left(\mathbf{y}_i - \mathbf{y}_j\right)}{\sum_{k \neq l} \left(1 + ||\mathbf{y}_k - \mathbf{y}_l||^2\right)} \tag{14}$$

Putting the top and bottom terms together

$$= \left( \sum_{j \neq i} \frac{\mathbf{y}_i - \mathbf{y}_j}{\left(1 + ||\mathbf{y}_i - \mathbf{y}_j||^2\right)^2} \right) \Bigg/ \left( \sum_{k \neq l} \frac{1 + ||\mathbf{y}_k - \mathbf{y}_l||^2}{\left(1 + ||\mathbf{y}_k - \mathbf{y}_l||^2\right)^2} \right) \tag{15}$$

$$= \left( \sum_{j \neq i} \frac{\mathbf{y}_i - \mathbf{y}_j}{\left(1 + ||\mathbf{y}_i - \mathbf{y}_j||^2\right)^2} \right) \Bigg/ \left( \sum_{k \neq l} \frac{1}{\left(1 + ||\mathbf{y}_k - \mathbf{y}_l||^2\right)} \right) \tag{16}$$

We can also write an expression for each term of $\mathbf{y}_i$ individually:

$$F_{\text{rep},i}(m) = \left( \sum_{j \neq i} \frac{\mathbf{y}_i(m) - \mathbf{y}_j(m)}{\left(1 + ||\mathbf{y}_i - \mathbf{y}_j||^2\right)^2} \right) \Bigg/ \left( \sum_{k \neq l} \frac{1}{\left(1 + ||\mathbf{y}_k - \mathbf{y}_l||^2\right)} \right) \tag{17}$$

where $\mathbf{y}_i(m)$ denotes the $m^{\text{th}}$ component of $\mathbf{y}$ i.e. $m \in \{1, 2\}$ in the 2D case.

[Linderman et al., 2017] make the acute observation that the repulsive forces $F_{\text{rep}}$ can be written as $s + 2$ sums of the form

$$\phi(\mathbf{y}_i) = \sum_j K(\mathbf{y}_i, \mathbf{y}_j) q_{ij} \tag{18}$$

where $K(y, z)$ is either the Cauchy kernel or the squared Cauchy kernel and $s$ is the dimensionality of $Y$

$$K_1(y, z) = \frac{1}{\left(1 + ||\mathbf{y} - \mathbf{z}||^2\right)}, \quad \text{or} \quad K_2(y, z) = \frac{1}{\left(1 + ||\mathbf{y} - \mathbf{z}||^2\right)^2} \tag{19}$$

To make the sums concrete, consider the 2D case:

$$\begin{aligned}
\phi_{1,i} &= \sum_{j\neq i} \frac{1}{(1+||\mathbf{y}_j-\mathbf{y}_i||^2)} \\
\phi_{2,i} &= \sum_{j\neq i} \frac{\mathbf{y}_j(1)}{(1+||\mathbf{y}_j-\mathbf{y}_i||^2)^2} \\
\phi_{3,i} &= \sum_{j\neq i} \frac{\mathbf{y}_j(2)}{(1+||\mathbf{y}_j-\mathbf{y}_i||^2)^2} \\
\phi_{4,i} &= \sum_{j\neq i} \frac{1}{(1+||\mathbf{y}_j-\mathbf{y}_i||^2)^2}
\end{aligned}$$

the the repulsive forces can be expressed in terms of these 4 sums as follows:

$$\begin{aligned}
F_{\mathrm{rep},i}(1) &= \left(\sum_{j\neq i} \frac{\mathbf{y}_i(1)-\mathbf{y}_j(1)}{(1+||\mathbf{y}_i-\mathbf{y}_j||^2)^2}\right) \Big/ \left(\sum_{k\neq l} \frac{1}{(1+||\mathbf{y}_k-\mathbf{y}_l||^2)}\right) \\
&= (\phi_{2,i} - \mathbf{y}_i(1)\phi_{4,i})/Z, \qquad (20) \\
F_{\mathrm{rep},i}(2) &= \left(\sum_{j\neq i} \frac{\mathbf{y}_i(2)-\mathbf{y}_j(2)}{(1+||\mathbf{y}_i-\mathbf{y}_j||^2)^2}\right) \Big/ \left(\sum_{k\neq l} \frac{1}{(1+||\mathbf{y}_k-\mathbf{y}_l||^2)}\right) \\
&= (\phi_{3,i} - \mathbf{y}_i(2)\phi_{4,i})/Z, \qquad (21)
\end{aligned}$$

where

$$Z = \sum_j \phi_{1,j} \qquad (22)$$

The key idea in this approach is that since we have smooth kernels $K_1$ and $K_2$, we can approximate them using polynomial interpolation. Of course, the choice of interpolants is entirely up to us, but we we evaluate our kernel functions at these points and interpolate our true data using these. To make things computationally efficient, we can set the interpolants to be equispaced points on the space spanned by the data.

This is very convenient, because the kernels in question are all translation invariant and when we evaluate them at the interpolants, then the kernel matrix $K$ will be Toeplitz. This means that it is enough to evaluate the Kernel for the left-most point in space in 1D. In 2d, our $K$ is actually a 3D tensor, but is again, Toeplitz.

Linear algebra tells us we can embed any Toeplitz matrix into a circulant matrix. This is desirable, because now we can perform matrix-vector multiplication in the frequency domain in linear time, with the slowest part being the FFT and IFFT transforms in $\mathcal{O}(n \log n)$ time.

Finally, having evaluated the repulsive forces at the interpolants, we just need to interpolate the forces on our true data. This can be done in linear time $\mathcal{O}(n)$.

Doing this, we have successfully made the overall complexity independent of $N$, and have shifted the brunt of the work onto the number of chosen interpolation points, so the time complexity will rely heavily on that. In practice, we split the input space into equally sized intervals, and then have 3 interpolation points in each interval. While we could increase the number of interpolation points, it is preferable to increase the number of intervals (due to the Runge phenomenon in interpolation). Increasing the number of interpolation points also increases the accuracy of the approximation, but comes at a computation cost.

Like the Barnes-Hut variant, this method becomes very inefficient for higher dimensions, as the number of interpolation points needed scales exponentially with $d$. In practice, this isn't an issue because most often, we want to inspect 2D embeddings.

## 3 Implementation details

### 3.1 Perplexity

The following section explains how perplexity is formulated so the code can run efficiently. Perplexity is defined as

$$\text{Perplexity}(P_i) = 2^{H(P_i)} \tag{23}$$

where $H$ is the Shannon entropy of a discrete distribution

$$H(P_i) = -\sum_i p_{j|i} \log_2(p_{j|i}) \tag{24}$$

In code, the following is more practical to avoid computing $2^x$ whereas perplexity stays fixed:

$$\log(\text{Perplexity}(P_i)) = -\sum_i p_{j|i} \log(p_{j|i}) \tag{25}$$

Remember that $P_i$ is just a Gaussian distribution centered on point $i$, given by

$$p_i(d_i) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{d_{ij}^2}{2\sigma^2}\right) \tag{26}$$

however, since we'll be performing row-normalization by hand, something proportional is sufficient

$$\sim \exp\left(-\frac{d_{ij}^2}{2\sigma^2}\right) \tag{27}$$

In most implementations this Gaussian is parameterized with $\beta = 1/2\sigma^2$ and therefore we compute $\exp\left(-d_{ij}^2\beta\right)$ in practice. In our case, we actually compute $\frac{1}{\sigma}\exp\left(-d_{ij}^2\beta\right)$ because we allow a multiscale approach, which mixes several Gaussians together. We also reparameterize our distribution to use the more interpretable precision $\tau = 1/\sigma^2$ instead of $\beta$. Therefore our probability density is given by

$$p_i(d_i) \sim \sqrt{\tau} \exp\left(-\frac{d_{ij}^2\tau}{2}\right) \tag{28}$$

We now plug in our parametrization into the entropy and arrive at a convenient form which can be coded efficiently.

$$H_i = -\sum_j \frac{\sqrt{\tau}\exp\left(-d_{ij}^2\tau/2\right)}{\sum_k \sqrt{\tau}\exp\left(-d_{ik}^2\tau/2\right)} \log\left(\frac{\sqrt{\tau}\exp\left(-d_{ij}^2\tau/2\right)}{\sum_k \sqrt{\tau}\exp\left(-d_{ik}^2\tau/2\right)}\right) \tag{29}$$

The first term is just $p_{j|i}$ and we can split up the log into two parts

$$= -\sum_j p_{j|i}\left[\log\left(\sqrt{\tau}\exp\left(-d_{ij}^2\tau/2\right)\right) - \log\left(\sum_k \sqrt{\tau}\exp\left(-d_{ik}^2\tau/2\right)\right)\right] \tag{30}$$

Notice now that the first term in the square brackets almost has the form $\log(\exp(x))$. For clarity, we will also denote the normalization sum as $Z$.

$$= -\sum_j \left[p_{j|i}\left(\frac{1}{2}\log\tau - d_{ij}^2\tau/2\right)\right] + \sum_j p_{j|i}\log Z \tag{31}$$

$$= -\frac{1}{2}\log\tau\sum_j p_{j|i} + \frac{\tau}{2}\sum_j p_{j|i}d_{ij}^2 + \sum_j p_{j|i}\log Z \tag{32}$$

We move the first term to the end to make the sign unmissable. Since $p_i$ is a proper probability distribution, its elements sum up to 1, leaving us with

$$= \frac{\tau}{2}\sum_j p_{j|i}d_{ij}^2 + \log Z - \frac{1}{2}\log\tau \tag{33}$$

This can be computed in two passes over the data. The first pass computes the unnormalized probabilities $\tilde{p}_{j|i}$ and accumulate the normalization constant $Z$. In the second pass, the first term can be computed.

In other implementation e.g. scikit-learn, the expression is computed without $\sqrt{\tau}$. It's easy to see that the result will be the same (and indeed, this is used in their code), but without the $-1/2\log\tau$ term and parameterized with $\beta = \tau/2$.

## 3.2 Fast KL Divergence

During computation of negative gradients, we do not know the value of the normalization term $Z$ during intermediate steps. Therefore, in order to compute the KL divergence of the embedding, we would need at least two passes over the data points, first to compute the unnormalized $q_{ij}$s, and secondly to normalize them and compute the KL divergence. By rewriting the KL divergence in terms of unnormalized $q_{ij}$s, we can compute the entire error with a single pass over the data points by accumulating the $\sum_{ij} p_{ij}$ and $\sum_{ij} q_{ij}$ in the first pass.

$$KL(P\ ||\ Q) = \sum_{ij} p_{ij}\log\frac{p_{ij}}{q_{ij}} \tag{34}$$

$$= \sum_{ij} p_{ij}\log\left(p_{ij}\frac{Z}{\hat{q}_{ij}}\right) \tag{35}$$

where $\hat{q}_{ij}$ denotes the unnormalized values $q_{ij}$

$$= \sum_{ij} p_{ij} \log \frac{p_{ij}}{\hat{q}_{ij}} + \sum_{ij} p_{ij} \log Z \tag{36}$$

Therefore the first term requires a single pass over all $i, j$s and the second term can be computed in constant time if we accumulate the sums of $P$ and $Q$.

This is already included in most software packages e.g. scikit-learn.

### 3.3 KL Divergence with exaggeration

The implemented optimization methods don't have a notion of exaggeration, they simply take an affinity matrix $P$ containing the probabilities of points $j$ appearing close to $i$. Exaggeration is used to scale $P$ by some constant factor $\alpha$ (this means that entries in the affinity matrix $P$ are not proper probabilities) to help separate clusters in the beginning of the optimization. These methods also compute the KL divergence during optimization (for efficiency), and, as such, the error is incorrect because we don't account for the scaling $\alpha$.

This section derives a simple correction for the KL divergence error term so we can get the true error of the embedding even when $P$ is exaggerated.

$$KL(P \mid\mid Q) = \sum_{ij} p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{37}$$

We need to introduce the scaling i.e. exaggeration factor $\alpha$ to every $p_{ij}$ term, so we multiply some terms by $1 = \alpha / \alpha$.

$$= \sum_{ij} \frac{\alpha}{\alpha} p_{ij} \log \frac{\alpha p_{ij}}{\alpha q_{ij}} \tag{38}$$

Exaggeration means that the $p_{ij}$ terms get multiplied by $\alpha$, so we need to find an expression for the KL divergence that includes only $\alpha p_{ij}$ and $q_{ij}$ and some other factor that will correct for $\alpha$.

$$= \frac{1}{\alpha} \sum_{ij} \alpha p_{ij} \left( \log \frac{\alpha p_{ij}}{q_{ij}} - \log \alpha \right) \tag{39}$$

$$= \frac{1}{\alpha} \left( \sum_{ij} \alpha p_{ij} \log \frac{\alpha p_{ij}}{q_{ij}} \right) - \frac{1}{\alpha} \left( \sum_{ij} \alpha p_{ij} \log \alpha \right) \tag{40}$$

We notice in the first term is exactly the KL divergence where $p_{ij}$s are scaled by $\alpha$. We also notice in the second term that $\sum_{ij} P_{ij} = 1$ and that $\alpha$ cancels out, leaving us with

$$= \frac{1}{\alpha} \left( \sum_{ij} \alpha p_{ij} \log \frac{\alpha p_{ij}}{q_{ij}} \right) - \log \alpha \tag{41}$$

The first term is computed by the gradient method (since it only knows about the scaled $P$), the second term can easily be computed post-optimization, allowing us to get the correct KL divergence.

### 3.4 Single Kernel in interpolation

## 4 Transform

### 4.1 Direct optimization

### 4.2 General framework of cost functions

[Bunte et al., 2012]

### 4.3 MDS interpolation

MDS Interpolation [Bae et al., 2010]. A similar approach might be able to be applied to t-SNE. In essence, they run MDS on a sample of points. Then for each new point, we compute the k-nearest neighbours and optimize the stress function w.r.t. only those points. In their paper, they derive equations that can be used for efficient optimization via majorization.

### 4.4 Kernel t-SNE

[Gisbrecht et al., 2012] claim to outperform direct mapping t-SNE using a direct kernel mapping. This paper is not very useful. The graph is misleading and the table at the end is informative, but run only on small datasets. Their subsequent paper is much better and throughout.

In [Gisbrecht et al., 2015], kernel t-SNE is described in more detail and parameters are chosen in a more principled manner.

Describes how to integrate class labels into embedding using Fischer information.

The issue of kernel t-SNE is that we have to compute the inverse of the interaction matrix K. We can use P as the interaction matrix, and P is sparse, but the inverse of that is very dense, and for any reasonably sized data set, this is unfeasable.

## References

[Bae et al., 2010] Bae, S.-H., Choi, J. Y., Qiu, J., and Fox, G. C. (2010). Dimension reduction and visualization of large high-dimensional data via interpolation. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 203–214. ACM.

[Bunte et al., 2012] Bunte, K., Biehl, M., and Hammer, B. (2012). A general framework for dimensionality-reducing data visualization mapping. *Neural Computation*, 24(3):771–804.

[Gisbrecht et al., 2012] Gisbrecht, A., Lueks, W., Mokbel, B., and Hammer, B. (2012). Out-of-sample kernel extensions for nonparametric dimensionality reduction. In *ESANN*, volume 2012, pages 531–536.

[Gisbrecht et al., 2015] Gisbrecht, A., Schulz, A., and Hammer, B. (2015). Parametric nonlinear dimensionality reduction using kernel t-sne. *Neurocomputing*, 147:71–82.

[Hinton and Roweis, 2003] Hinton, G. E. and Roweis, S. T. (2003). Stochastic neighbor embedding. In *Advances in neural information processing systems*, pages 857–864.

[Linderman et al., 2017] Linderman, G. C., Rachh, M., Hoskins, J. G., Steinerberger, S., and Kluger, Y. (2017). Efficient algorithms for t-distributed stochastic neighborhood embedding. *arXiv preprint arXiv:1712.09005*.

[Maaten and Hinton, 2008] Maaten, L. v. d. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605.

[Van Der Maaten, 2014] Van Der Maaten, L. (2014). Accelerating t-sne using tree-based algorithms. *Journal of machine learning research*, 15(1):3221–3245.

[Yianilos, 1993] Yianilos, P. N. (1993). Data structures and algorithms for nearest neighbor search in general metric spaces. In *SODA*, volume 93, pages 311–321.