t-SNE Notes

Iakovidis Ioannis
email iakoviid@auth.gr

January 4, 2020

1 Introduction

t-Distributed Stochastic Neighbor Embedding (t-SNE) is a technique for dimensionality reduction that is particularly well suited for the visualization of high-dimensional datasets.

Let $x_i \in \mathbf{R^d}, i = 1, \dots, n$ be the data samples in high dimensional space and $y_i \in \mathbf{R^s}, i = 1, \dots, n$ be the data samples in low dimensional space, s is usually 2 or 3. The goal of t-SNE is to preserve local structure, points that are similar (close) in the high dimensional space are mapped to similar (close) points in the low dimensional space. This is achieved by minimizing the divergence between similarity weights that are defined in the high and low dimensional space.

More specifically from the distances d_{ij} of $(x_i)_{i=1}^n$ we define

$$p_{i|j} = \frac{e^{-d_{ij}^2/2\sigma_i^2}}{\sum_{l \neq i} e^{-d_{il}^2/2\sigma_i^2}} \quad \text{and} \quad p_{ij} = \frac{p_{i|j} + p_{j|i}}{2N}$$

as the similarity weights in the high dimensional space. The value $p_{j|i}$ is interpreted as the distribution of the other points given x_i or the probability that point j is a neighbor of point i. The parameters N and σ_i are chosen. For the low dimensional space we define

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

as similarity weights but this time we use the t-distribution. For notation purposes let $P = \{p_{ij}\}$, $Q = \{q_{ij}\}$ $n \times n$ matrices and $X = [x_1, \dots, x_n]$, $Y = [y_1, \dots, y_n]$ $d \times n$ and $s \times n$ matrices respectively.

And then we take KL divergence

$$C(Y) = KL(P \mid Q) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}}.$$

Minimization can be done with gradient descent. Note that this function is non convex and hence we will halt at a local minimum.

2 Deriving the Gradient

Here with straight calculations we compute the gradient. First of all:

$$\begin{aligned}
C(Y) &= \sum_{i\neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}} \\
&= \sum_{i\neq j} p_{ij} \log p_{ij} - \sum_{i\neq j} p_{ij} \log q_{ij} \\
&= \sum_{i\neq j} p_{ij} \log p_{ij} + \sum_{i\neq j} p_{ij} \log f_{ij} + \sum_{i\neq j} p_{ij} \log Z \\
&= \sum_{i\neq j} p_{ij} \log p_{ij} + \sum_{i\neq j} p_{ij} \log f_{ij} + \log Z.
\end{aligned}$$

Where $f_{ij} = 1 + \|y_i - y_j\|^2$ and $Z = \sum_{k\neq l}(1 + \|y_k - y_l\|^2)^{-1}$.
With some computation:

$$\begin{aligned}
\frac{\partial \sum_{i\neq j} p_{ij} \log f_{ij}}{\partial y_m} &= \sum_{i\neq m} p_{im} \frac{\partial \log f_{im}}{\partial y_m} + \sum_{i\neq m} p_{mi} \frac{\partial \log f_{mi}}{\partial y_m} \\
&= 2\sum_{i\neq m} p_{mi} \frac{\partial \log f_{mi}}{\partial y_m} \\
&= 2\sum_{i\neq m} p_{mi} \frac{1}{f_{mi}} \frac{\partial f_{mi}}{\partial y_m} \\
&= 2\sum_{i\neq m} p_{mi} \frac{2(y_m - y_i)}{1 + \|y_m - y_i\|^2} \\
&= 4\sum_{i\neq m} p_{mi} \frac{Z(y_m - y_i)}{Z(1 + \|y_m - y_i\|^2)} \\
&= 4\sum_{i\neq m} p_{mi} q_{mi} Z(y_m - y_i).
\end{aligned}$$

$$\begin{aligned}
\frac{\partial \log Z}{\partial y_m} &= \frac{\partial Z}{\partial y_m} \frac{1}{Z} \\
&= \frac{2}{Z} \sum_{i\neq m} \frac{\partial (1 + \|y_m - y_i\|^2)^{-1}}{\partial y_m} \\
&= -\frac{2}{Z} \sum_{k\neq m} \frac{(y_m - y_i)}{1 + \|y_m - y_i\|^2)^2} \\
&= -\frac{2}{Z} \sum_{i\neq m} \frac{2(y_m - y_i)}{1 + \|y_m - y_i\|^2} \\
&= -4\sum_{i\neq m} q_{mi}^2 Z(y_m - y_i).
\end{aligned}$$

So

$$\frac{\partial C(Y)}{\partial y_i} = 4\sum_{j\neq i} p_{ij} q_{ij} Z(y_i - y_j) - 4\sum_{j\neq i} q_{ij}^2 Z(y_i - y_j)$$

Also we can make same the calculation so that it generalizes more easily to changes in the objective function or the similarity weights. Observe how the objective function C depends on the embedded points. Let more general $q_{ij} = \frac{w_{ij}}{\sum_{kl} w_{kl}}$ where w_{ij} is the similarity weight of y_i and y_j depended on their distance d_{ij}.

So when computing the gradient we:

1. The distances d_ij are generated from the coordinates,y_i, y_j.
2. The similarity weights w_{ij} are generated as a function of the distances, d_ij.
3. The output probabilities, q_{ij}, are normalized versions of the similarity weights, w_{ij}.
4. The cost function, C is normally a divergence of some kind, and hence expressed in terms of the output probabilities, q_{ij}.

We're going to chain those individual bits together via the chain rule for partial derivatives. The chain of variable dependencies is $C \to q \to w \to d \to y$. So

$$\frac{\partial C}{\partial \mathbf{y_m}} = \sum_{ij} \frac{\partial C}{\partial q_{ij}} \sum_{kl} \frac{\partial q_{ij}}{\partial w_{kl}} \sum_{pq} \frac{\partial w_{kl}}{\partial d_{pq}} \frac{\partial d_{pq}}{\partial \mathbf{y_m}}$$

In the relationship between w, and d is such that any cross terms are 0, i.e. unless $k = p$ and $l = q$ those derivatives evaluate to 0. Also, either $k = m$ or $l = m$, otherwise $\partial d_{kl}/\partial \mathbf{y_m} = 0$. So

$$\begin{aligned}
\frac{\partial C}{\partial \mathbf{y_m}} &= \sum_{ij} \frac{\partial C}{\partial q_{ij}} \sum_{kl} \frac{\partial q_{ij}}{\partial w_{kl}} \sum_{pq} \frac{\partial w_{kl}}{\partial d_{pq}} \frac{\partial d_{pq}}{\partial \mathbf{y_m}} \\
&= \sum_{ij} \frac{\partial C}{\partial q_{ij}} \sum_{kl} \frac{\partial q_{ij}}{\partial w_{kl}} \frac{\partial w_{kl}}{\partial d_{kl}} \frac{\partial d_{kl}}{\partial \mathbf{y_m}} \\
&= \sum_{ij} \frac{\partial C}{\partial q_{ij}} \sum_{l} \frac{\partial q_{ij}}{\partial w_{ml}} \frac{\partial w_{ml}}{\partial d_{ml}} \frac{\partial d_{ml}}{\partial \mathbf{y_m}} + \sum_{ij} \frac{\partial C}{\partial q_{ij}} \sum_{k} \frac{\partial q_{ij}}{\partial w_{km}} \frac{\partial w_{km}}{\partial d_{km}} \frac{\partial d_{km}}{\partial \mathbf{y_m}}
\end{aligned}$$

Exchanging the summations:

$$\begin{aligned}
\frac{\partial C}{\partial \mathbf{y_m}} &= \sum_{l} \Big(\sum_{ij} \frac{\partial C}{\partial q_{ij}} \frac{\partial q_{ij}}{\partial w_{ml}} \Big) \frac{\partial w_{ml}}{\partial d_{ml}} \frac{\partial d_{ml}}{\partial \mathbf{y_m}} + \sum_{k} \Big(\sum_{ij} \frac{\partial C}{\partial q_{ij}} \frac{\partial q_{ij}}{\partial w_{km}} \Big) \frac{\partial w_{km}}{\partial d_{km}} \frac{\partial d_{km}}{\partial \mathbf{y_m}} \\
&= \sum_{l} f_{ml} \frac{\partial w_{ml}}{\partial d_{ml}} \frac{\partial d_{ml}}{\partial \mathbf{y_m}} + \sum_{k} f_{km} \frac{\partial w_{km}}{\partial d_{km}} \frac{\partial d_{km}}{\partial \mathbf{y_m}} \\
&= \sum_{k} f_{mk} \frac{\partial w_{km}}{\partial d_{km}} \frac{\partial d_{km}}{\partial \mathbf{y_m}} + \sum_{k} f_{km} \frac{\partial w_{km}}{\partial d_{km}} \frac{\partial d_{km}}{\partial \mathbf{y_m}}, \text{Assume w and d are symmetric} \\
&= \sum_{k} \Big(f_{mk} + f_{km} \Big) \frac{\partial w_{km}}{\partial d_{km}} \frac{\partial d_{km}}{\partial \mathbf{y_m}}
\end{aligned}$$

Now

$$\frac{\partial d_{ij}}{\partial \mathbf{y_i}} = \frac{1}{d_{ij}} (\mathbf{y_i} - \mathbf{y_j}) \;\text{ for } d_{ij} = \left[\sum_{l}^{K} (y_{il} - y_{jl})^2 \right]^{1/2}$$

And

$$\frac{\partial w_{ij}}{\partial d_{ij}} = -\frac{2d_{ij}}{(1+d_{ij}^2)} \text{ for } w_{ij} = \frac{1}{(1+d_{ij}^2)^2}$$

So

$$\frac{\partial C}{\partial \mathbf{y_m}} = -2\sum_k \Big(f_{mk} + f_{km}\Big)\frac{(y_m - y_k)}{(1+{d_{km}}^2)^2}$$

And for specifically the KL divergence:

$$\frac{\partial C}{\partial q_{ij}} = -\frac{p_{ij}}{q_{ij}}$$

$$\frac{\partial q_{ij}}{\partial w_{km}} = -\frac{w_{ij}}{(\sum_{lt} w_{lt})^2} = -\frac{q_{ij}}{Z}$$

While

$$\frac{\partial q_{ij}}{\partial w_{ij}} = -\frac{w_{ij}}{(\sum_{lt} w_{lt})^2} + \frac{1}{\sum_{lt} w_{lt}} = -\frac{q_{ij}}{Z} + \frac{1}{Z}$$

Ending

$$f_{km} = -\frac{p_{km}}{Zq_{km}} + \sum_{ij}\frac{p_{ij}}{Z}$$

And as a result we get the same gradient expression.

$$\frac{\partial C(Y)}{\partial y_i} = 4\sum_{j\neq i} p_{ij}q_{ij}Z(y_i - y_j) - 4\sum_{j\neq i} q_{ij}^2 Z(y_i - y_j)$$

3 Gradient Interpretation

The t-SNE gradient computation can be reformulated as an N-body simulation problem:

$$F_{attr,i} = \sum_{j\neq i} p_{ij}q_{ij}Z(y_i - y_j)$$

$$F_{rep,i} = \sum_{j\neq i} q_{ij}^2 Z(y_i - y_j)$$

$$\frac{1}{4}\frac{\partial C(Y)}{\partial y_i} = F_{attr,i} + F_{rep,i}$$

The forces can be also viewed with matrix-vector computations in the following ways:

- $$F_{attr} = (P \odot Q)O \odot Y - (P \odot Q)Y$$
  $$F_{rep} = (Q \odot Q)O \odot Y - (Q \odot Q)Y$$

  Where Y is the $N \times 2$ matrix of embedded points, O is an $N \times 2$ matrix of ones.

-

4 First Complexity Discussion

5 Attractive Term approximation

6 Repulsive Term approximation

6.1 Fast Multipole Methods

Essentially a subproblem of computing the gradient is to compute sums of the form:

$$u_i = \sum_{j=1}^{N} G(y_i, y_j) v_j$$

in our case for computing the repulsive forces $G(y_i, y_j) = \frac{1}{1+\|y_i - y_j\|^2}$. More generally the fast multipole methods are constructed to compute sums of the form:

$$u_i = \sum_{j=1}^{N} G(t_i, y_j) v_j$$

the points $\{t_i\}_1^M$ are called targets and $\{y_i\}_1^N$ are called sources, both of dimension s. While the kernel $G(t_i, y_j)$ depends on the distance of t_i and y_j. Notice that computing these sums naively takes $O(NM)$ (quadratic) time.

Putting all evaluations of the kernel in a matrix $A : A_{ij} = G(t_i, y_j)$ we can see that $u = Ax$. It is known that if the domain of T is different than the domain of Y then we can separate the variables and have $A \approx B(X)C(Y)$, let B be $M \times P$ and C be $P \times N$ matrices. This is true because if the kernel have separate domains (does not blow up) then A can be efficiently approximated by a low rank matrix.

$$\begin{aligned} u_i &= \sum_{j=1}^{N} G(t_i, y_j) v_j \\ &= \sum_{j=1}^{N} \sum_{k=1}^{P} B_{ik} C_{kj} v_j \\ &= \sum_{k=1}^{P} B_{ik} \Big(\sum_{j=1}^{N} C_{kj} v_j \Big) \\ &= \sum_{k=1}^{P} B_{ik} m_k \end{aligned}$$

Note that m_k is only computed once for all i, meaning that the sums can be computed in $P \cdot (N + M)$ computations–a dramatic improvement over $M \cdot N$.

6.2 Separation of Variables

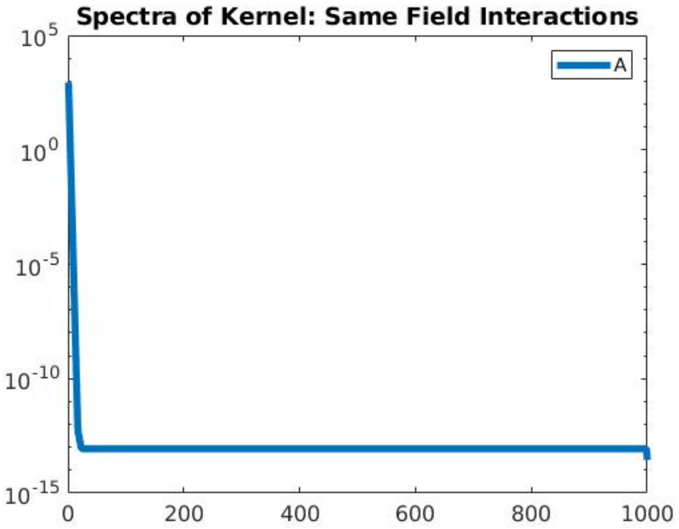
But how can we find such a separation the variables. First we will discuss the optimal separation (optimal in terms of accuracy). Assume the Singular Value Decomposition (SVD) of A, $A \approx U\Sigma V^T$ taking the p highest singular values. Let u_i, v_i be the columns of U and V while σ_i be the singular values. Notice that:

$$A \approx U\Sigma V^T = \sum_{k=1}^{p} \sigma_k u_k {v_k}^T$$

$$\implies A_{ij} = \sum_{k=1}^{p} \sigma_k u_k(i) {v_k}^T(j)$$

This is a separation of variables like what described above. But how is it's properties in terms of accuracy? The SVD for the p highest singular values is the optimal approximation of the matrix in terms of the L_2 norm (Eckart-Young-Minsky theorem). The error is given in terms of the singular that we didn't consider. In our case because of our kernel such low rank approximation is very accurate.

Figure 1: Singular values of the Kernel for 1000 random points.



Also notice that because in our case the set of targets and sources it the same we have $U = V$ and so:

$$A = V\Sigma V^T$$

6.3 Interpolative Separation of Variables

But finding the SVD of a matrix even approximately is slow (for p largest singular values: $O(N \cdot M \cdot p)$). So we need an other way to separate variables. We will use an interpolation based technique. Starting let $s = 1$ (1-dimensional embedding space) and then we will generalize. Instead of the Kernel we use the Lagrange interpolation polynomial for the Kernel defined by equidistant points in the target and source fields.

Let R_y and R_z the intervals denoting the ranges of the source and target points, $y_i \in R_y, \forall i = 1, \ldots, N$ and $z_i \in R_z, \forall i = 1, \ldots, M$. Now suppose that $\tilde{z_1}, \ldots, \tilde{z_p}$ are equidistant points in R_z and $\tilde{y_1}, \dots, \tilde{y_p}$ are equidistant points in R_y, the number of points p will control the error of our approximation method. The importance of the points being equidistant will be explored in the next paragraph. Let $L_{l,\tilde{y}}$ and $L_{l,\tilde{z}}$ be the Lagrange polynomials :

$$L_{l,\tilde{y}}(y) = \frac{\prod_{j\neq l}^{p}(y - \tilde{y_j})}{\prod_{j\neq l}^{p}(\tilde{y_l} - \tilde{y_j})} \text{ and } L_{l,\tilde{z}}(z) = \frac{\prod_{j\neq l}^{p}(z - \tilde{z_j})}{\prod_{j\neq l}^{p}(\tilde{z_l} - \tilde{z_j})}$$

With these polynomials we can make an interpolation polynomial G_p for the Kernel.

$$G_p(z,y) = \sum_{l=1}^{p}\sum_{j=1}^{p} G(\tilde{z_l}, \tilde{y_j}) L_{l,\tilde{z}}(z) L_{j,\tilde{y}}(y)$$

Now with the use of the approximation of G with G_p we can approximate the

$$u_i = \sum_{j=1}^{N} G(t_i, y_j) v_j$$

by

$$\tilde{u_i} = \sum_{j=1}^{N} G_p(t_i, y_j) v_j.$$

So

$$\begin{aligned}
\tilde{u_i} &= \sum_{j=1}^{N} G_p(t_i, y_j) v_j \\
&= \sum_{j=1}^{N}\sum_{l=1}^{p}\sum_{m=1}^{p} G(\tilde{z_l}, \tilde{y_m}) L_{l,\tilde{z}}(z_i) L_{m,\tilde{y}}(y_j) v_j \\
&= \sum_{l=1}^{p} L_{l,\tilde{z}}(z_i) \Big(\sum_{m=1}^{p} G(\tilde{z_l}, \tilde{y_m}) \Big(\sum_{j=1}^{N} L_{m,\tilde{y}}(y_j) v_j \Big)\Big).
\end{aligned}$$

By this separation we can compute the values using three convolutions (by the nested parentheses) of $\{\tilde{u_i}\}_1^M$ in $O((M+N)p + p^2)$ time.

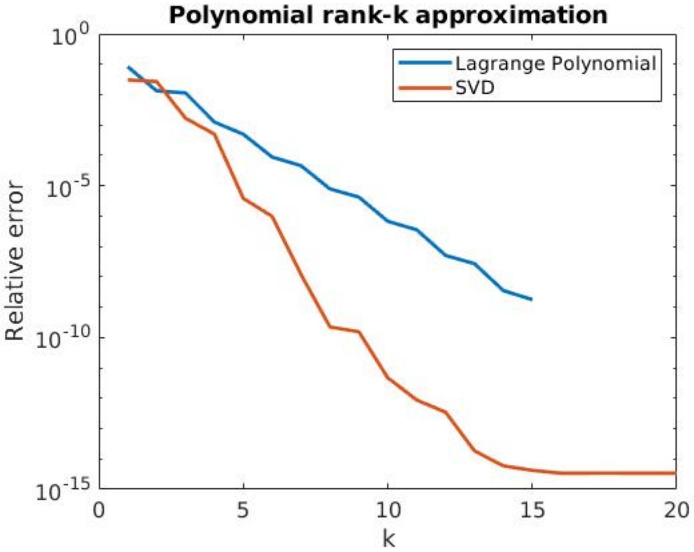
But what are the accuracy properties of this method. The error

$$\begin{aligned}
|u_i - \tilde{u_i}| &= |\sum_{j=1}^{N} (G(z_i, y_j) - G_p(z_i, y_j)) \cdot v_j| \\
&\leq \sum_{j=1}^{N} |(G(z_i, y_j) - G_p(z_i, y_j))| \cdot |v_j| \\
&\leq \epsilon \sum_{1}^{N} |v_j|.
\end{aligned}$$

Provided that the polynomial uniformly approximates the kernel in the ranges. That is:

$$\exists \epsilon : \sup |G(z,y) - G_p(z,y)| \leq \epsilon$$

Figure 2: Relative Error of SVD and Lagrange approximations.



6.4 Use of the FFT

For the following text we will consider the case where the targets and the sources are the same set. In computing the convolution $\sum_{m=1}^{p} G(\tilde{y_l}, \tilde{y_m}) w_m$ notice that can be expressed as a matrix vector product $K \cdot w$ where K the matrix of evaluations of the kernel and w the vector of the values $\{w_m\}_1^p$. But because the points $\tilde{y_l}$ are equidistributed the matrix K is Toeplitz and hence the product can be computed efficiently with the FFT in time $O(p \log p)$. Note that a matrix vector product takes $O(p^2)$ but here we can use the FFT to exploiting the structure of a Toeplitz matrix.

The FFT be used in the following manner.First of all let

$$K = \begin{bmatrix} t_0 & t_1 & \dots & t_{p-1} \\ t_1 & t_0 & \dots & \dots \\ \dots & \dots & \dots & t_1 \\ t_{p-1} & \dots & t_1 & t_0 \end{bmatrix}$$

we construct the matrix

$$C = \begin{bmatrix} K & B \\ B & K \end{bmatrix}$$

where

$$B = \begin{bmatrix} 0 & t_{p-1} & \dots & t_1 \\ t_{p-1} & 0 & \dots & \dots \\ \dots & \dots & \dots & t_{p-1} \\ t_1 & \dots & t_{p-1} & 0 \end{bmatrix}.$$

The matrix C is circulant and we can compute it's matrix vector products with the FFT. Now zero padding the vector w and performing the product should give us the answer.

$$C \cdot \begin{bmatrix} w \\ 0 \end{bmatrix} = \begin{bmatrix} Kw \\ Bw \end{bmatrix}$$

```
%FFT matrix-vector product of a Toeplitz matrix
p=1024;
y = randn(p,1) ;
w = randn(p,1) ;
K=toeplitz(y);

z=K*w;

a=[0;y(p:-1:2)];
B=toeplitz(a);
C2=[K B;B K];

w2=[w;zeros(p,1)];

b=C2*w2;
b_fft=ifft(fft(w2).*fft(C2(1,:))');
disp(norm(b_fft(1:p)-z));
```

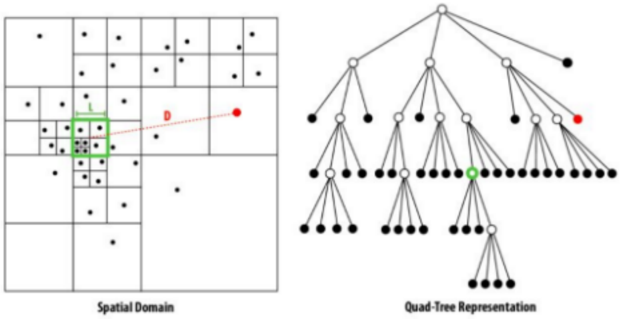
7 Improving Accuracy

7.1 Generalizing to Higher Dimensions

7.2 Barnes–Hut

The Barnes–Hut simulation is an approximation algorithm for performing an n-body simulation. It is notable for having order $O(n \log n)$ compared to a direct-sum algorithm which would be $O(n^2)$. The space is hierarchically divided into cells (rectangular or cubic), so that only points from nearby cells need to be treated individually, and points in distant cells can be treated as a single large point centered at the cell's center of mass. This can dramatically reduce the number of particle pair interactions that must be computed.

More specifically the Barnes–Hut algorithm constructs a octtree or quadtree and for each point it does a depth first search on that tree at each node testing a condition that says if the approximation for this cell is valid.



7.3 Barnes–Hut an algebraic interpretation

It is clear that given a constructed space tree we can reorder the matrix $A = \{q_{ij}^2\}$ to a matrix A' that has the recursive partition:

$$A' = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

for a binary tree $y \in \mathbf{R}$ or

$$A' = \begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{bmatrix}$$

for a quadtree $y \in \mathbf{R}^2$ similarly for an octtree.

Then for the computation for

$$u = A'v = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} v_{near} \\ v_{far} \end{bmatrix} = \begin{bmatrix} A_{11}v_{near} + A_{12}v_{far} \\ A_{21}v_{near} + A_{22}v_{far} \end{bmatrix} \approx \begin{bmatrix} A_{11}v_{near} + a_1 \\ a_2 + A_{22}v_{far} \end{bmatrix}$$

a_1 and a_2 represent the approximation of the the terms of the cells with the combined force from the center of mass. This approximation can be done recursively at any level.

8 FIt-SNE

9 SG-SNE

10 Method Comparisons

11 Theoretical Analysis