

École polytechnique de Louvain

Data visualization using nonlinear dimensionality reduction accelerated with fast multipole methods

Author: Cédric ANTOINE

Supervisors: John LEE, Michel VERLEYSEN

Readers: Cyril DE BODT, Pierre LAMBERT

Academic year 2022–2023

Master [120] in Data Sciences Engineering

Acknowledgements

I would like to thank my family and friends for their support and advice. A special thanks goes to my sister, who helped me proofread this thesis.

Moreover, I would like to thank Professors John Lee and Michel Verleysen for trusting me with this thesis topic and for their role as supervisors. Next to that I also thank the assistant supervisor Cyril De Bodt for his availability, advice and feedback. He was a crucial support in the progress of this master thesis.

Finally, I would like to thank Valentin Delchevalerie and his peers for providing me with an implementation of their work as well as their offer of help in understanding it.

Abstract

The following thesis investigates the use of Fast Multipole Methods as part of the acceleration of multi-scale Neighbor Embeddings methods. Given the exponential increase in the amount of available data in recent years, it is essential that the tools and algorithms for processing it become faster. As a matter of fact, this does not exclude dimension reduction techniques such as the family of Neighbour embedding methods. Indeed, given their remarkable results, particularly in the context of visualisation, these methods have enjoyed growing popularity since their development.

As for the current literature, the two most popular approaches for accelerating Neighbor Embedding methods are the Barnes-Hut algorithm as well as Fast Multipole Methods. Accelerated single-scale Neighbor Embeddings algorithms that use these two methods are already widespread. Nevertheless, regarding the multi-scale scheme of Neighbor Embedding methods, only an acceleration using Barnes-Hut is currently proposed. Given the improved results of multi-scale Neighbor Embeddings methods compared to those of the single-scale approaches, as well as the lower time complexity of Fast Multipole Methods when compared to the Barnes-Hut algorithm, this work therefore seeks to combine both.

Hence, the subsequent work is divided into three parts. The first one covers the single-scale and multi-scale versions of several Neighbor Embedding methods in detail. The second then analyses two distinct Fast Multipole Methods and the methodology by which they have been used to accelerate the single-scale methods previously covered. Finally, the third part analyses how these same Fast Multipole Methods can be adapted to accelerate the multi-scale scheme of previously accelerated single-scale methods. Both, an implementation and experimental results of these adaptations are then given and compared with some existing methods.

Ultimately, the paper concludes by highlighting certain limitations as well as several promising results of the carried out implementations. In addition, a number of suggestions and ideas for improvements are put forward in order to not only avoid the limitations mentioned above, but also to improve the obtained results. In addition, the complexity and variability of Fast Multipole Methods are highlighted, although motivating further study for their use. Done implementations are publicly available at <https://github.com/cedantoine/Multi-scale-FGT-and-PM>.

List of Contributions

1. This work gathers summaries of several NE methods (SNE, s-SNE, t -SNE), covering in detail their single- and multi-scale schemes. Furthermore, it introduces two Fast Multipole Methods and how these can be applied to accelerate the single-scale versions of the previous introduced NE methods.
2. An adaptation to the multi-scale schema of the Particle-Mesh single-scale t -SNE algorithm (PM t -SNE), developed in [1], is proposed and implemented in the subsequent work.
3. An attempt is made to adjust the Fast Gaus Transform-accelerated single-scale SNE algorithm of [2] to the multi-scale scheme. The encountered obstacles are then explained and analysed in detail in the work.
4. Finally, the results obtained in the two previous points are analysed and various directions for further developments are proposed.

Contents

1	Introduction	4
1.1	Context and Introductory Notions	5
1.2	Problem Statement	7
1.3	Related Work	8
1.4	Contributions to the Domain	9
2	Neighbor Embedding (NE)	11
2.1	Single-scale Methods	11
2.1.1	Stochastic Neighbor Embedding (SNE)	12
2.1.2	Symmetric Stochastic Neighbor Embedding (s-SNE)	13
2.1.3	t-Distributed Stochastic Neighbor Embedding (<i>t</i> -SNE)	14
2.2	Multi-scale Methods	16
2.2.1	Stochastic Neighbor Embedding (Ms SNE)	16
2.2.2	Symmetric Stochastic Neighbor Embedding (Ms s-SNE)	18
2.2.3	t-Distributed Stochastic Neighbor Embedding (Ms <i>t</i> -SNE)	18
3	Fast Single-scale NE Methods	21
3.1	Fast Multipole Methods versus Barnes-Hut Algorithm	21
3.2	Fast Multipole Methods (FMM)	22
3.2.1	Computation of Single-scale HD similarities	22
3.2.2	Fast Gauss Transform (FGT) for symmetric SNE	24
3.2.3	Particle-Mesh (PM) for t-SNE	30
4	Developing Fast Multipole Methods for Multi-scale NE	37
4.1	Computation of Multi-scale HD similarities	38
4.2	Fast Gauss Transform (FGT) for Multi-scale s-SNE (Ms s-SNE)	41
4.2.1	Comparative Analysis of s-SNE and Ms s-SNE for FGT	42
4.2.2	Implementation Choices - Hybrid s-SNE	44
4.2.3	Experimental Observations	44
4.3	Particle-Mesh (PM) for Multi-scale t-SNE (Ms <i>t</i> -SNE)	46

4.3.1	Comparative Analysis of t -SNE and Ms t -SNE for PM . . .	46
4.3.2	Experimental Results	47
5	Discussions and Further Works	53
5.1	Fast Gauss Transform for Multi-scale s-SNE	54
5.2	Particle-Mesh for Multi-scale t -SNE	54
6	Conclusion	56
7	Appendix	62
7.1	Appendix A	62
7.1.1	Shifting from Gaussians to Taylor series	62
7.1.2	Fast Gauss transform (FGT) time complexity development .	63
7.2	Appendix B	64
7.2.1	Computation of the normalisation factor Z in PM t -SNE . .	64

Chapter 1

Introduction

In our modern world, data are everywhere. The computer revolution of recent decades, which popularised the use of computers among other things, has been followed in recent years by a digital revolution. This digital revolution has completely changed the way we use computers, leading to the emergence of numerous innovations such as Virtual Reality, Augmented Reality, Blockchain, Big Data, Artificial Intelligence and the Internet of Things. These changes have been accompanied by a boom in data volumes. Indeed, a research suggests that more or less $79 \cdot 10^3$ million terabytes of data were generated during 2021 [3].

However, such an increase in data also comes with a major challenge: the analyzability and usability of that same data. A first obstacle is the human brain, indeed, it can easily grasp objects in two or three dimensions whilst having a great difficulty understanding higher-dimensional data. Another problem is the curse of dimensionality. As the number of dimensions increases, the volume in space increases rapidly, making the available data sparse. As a result, the amount of data required to obtain reliable results increases exponentially.

However, as is often the case, humans have found ways to remedy this challenge. A solution that has gained in popularity in recent years and that uses the power of computers in combination with statistics, is **dimensionality reduction (DR)**.

Dimensionality reduction consists in finding a coherent **low-dimensional (LD)** representation of **high-dimensional (HD)** data [4]. As mentioned earlier, humans are predisposed to interact with low dimensions. Consequently, in the context of dimension reduction for visualisation, two or three dimensions are often targeted.

The first section of this chapter lays out a more formal introduction to some of the main aspects of dimensionality reduction. The second section formulates the motivations that led to this work, by building on the previous part. As for the third section, it introduces the work previously done on the subject. Finally, the fourth section presents the additional contributions that this paper brings.

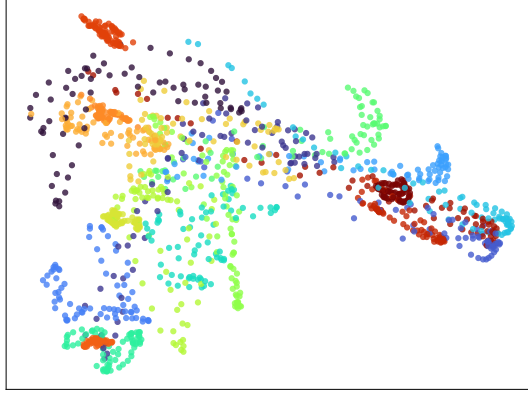
1.1 Context and Introductory Notions

In this section, formal definitions and mathematical formulas are consciously avoided wherever possible. The aim is to provide an overview of the field without going into too much detail.

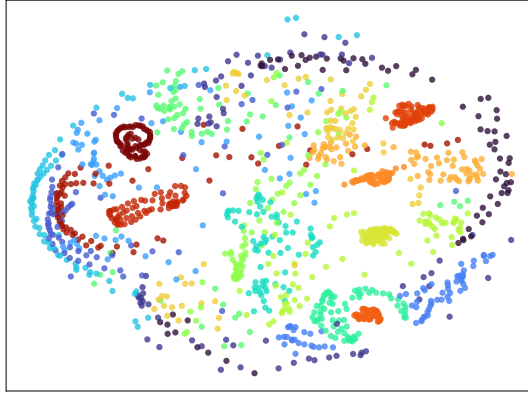
As mentioned earlier, dimensionality reduction is the process of faithfully representing HD data in LD spaces. However, there exist several ways to achieve this. One of the simplest forms is simply to remove features that seem less interesting for the job, thus keeping only those we consider relevant. There are numerous supervised and unsupervised methods available to help select the features to be considered relevant [5], [6]. This form of DR is known as **feature selection**. Next to that, another DR method is called **feature extraction**. In this method, new features are created from original ones. The aim of these new features is to summarise all the information present in the original features as best as possible. In this paper, we solely focus on feature extraction as DR, i.e. the ability to combine multiple HD features into a few LD features [4]. However, it is interesting to note that feature selection is often used as a preprocessing tool for feature extraction.

When it comes to feature extraction, there's one very important notion to consider: the extraction fidelity. Indeed, the aim is to summarise HD information as accurately as possible in LD, and thus have the most faithful representation of the HD embedding. Hence, when developing a DR algorithm, it is essential to first define **which properties must be maintained** between HD and LD in order for the representation to be considered faithful.

In the context of DR for visualization, the relevance of an LD representation of data is often quite intuitively associated with the preservation of distances [7]. The aim of this paradigm is to maintain, as far as possible, the pair-wise distances of HD in LD. Here, when it is spoken of distances, those are not necessarily Euclidean, although they are often related to the latter. Algorithms that use this paradigm include, among others, early principal component analysis (PCA) [8] and classical multidimensional scaling (MDS) [7]. These perform linear projections of the HD vectors, while trying to respectively maintain the variance or the dot product of the HD representations. Embeddings of the same initial dataset can be observed using both algorithms in Figure 1.1 above.



(a) PCA: trying to maintain variance



(b) MDS: trying to maintain dot product

Figure 1.1: 2D Embedding of the initial 16384D COIL-20 dataset

Other more recent algorithms, such as nonlinear metric MDS extensions [9], try to preserve (weighted) distance based on approximations of geodesic or Euclidean measures [10]. However, these algorithms are hardly better in visualisation tasks than their linear predecessors mentioned before [4], [11], [12]. This is probably due to the fact that they can be expressed as a classical MDS but applied in an unknown feature space [13].

Although the distance preservation paradigm seems quite intuitive and very promising, it does have one major drawback. Indeed, when dealing with high dimensional data, a phenomenon known as **norm concentration** [14] appears. In fact one can observe that, the more dimensions there are, the more the pair-wise Euclidean distances between points tend to become similar [15]. However, considering the DR, low dimensions don't have enough degrees of freedom to place

a large number of points so that they are all equidistant.

Another paradigm, meanwhile, effectively counters this phenomenon by trying to equalise the probabilities of neighborhood distributions between HD and LD [15]. This enables the preservation of the point’s neighborhoods [16], [17] rather than the distance between them. Methods using this paradigm are known as **Neighbor embedding techniques (NE)**.

The first algorithm that ever used this paradigm is called stochastic neighbor embedding (SNE) [17]. It aims to match Gaussian similarities between HD and LD. Here, similarities can be considered to represent the previously mentioned conditional probabilities. SNE’s remarkable performances have motivated the development of numerous alternatives based on its principle [12], [18]. These include, among others, *t*-distributed stochastic neighbor embedding (*t*-SNE) [12], [19], [20] using a heavily tailed distribution in LD instead of a Gaussian one as in SNE.

Initially, NE methods required the user to specify a parameter called the perplexity. Data typically present large structures within which are located smaller-scale structures. The perplexity can thus be seen as the parameter setting the neighborhood scale to be maintained during DR. Such methods, that focus on one perplexity, can be seen as single-scale methods. Capturing the organisation of data on multiple scales is a major difficulty in DR. Indeed, most algorithms are obliged to discard information from one scale in order to capture information from another.

Nevertheless, fairly recently, multi-scale NE methods have been introduced. These combine probabilities of neighborhoods of different scales [21], [22] for the data, resulting in meta-parameter-free techniques. Compared to their single-scale counterparts, these new techniques perform much better at maintaining both global and local structures [21]. Hence, employing them enables the preservation both near and far neighborhoods much better than their single-scale equivalents.

Unfortunately, despite their impressive results, the NE, single- and multi-scale techniques respectively have a time complexity of $\mathcal{O}(N^2)$ and $\mathcal{O}(N^2 \log N)$ for N data samples. The presence of the quadratic term in this complexity makes the use of NE methods incompatible with large datasets, inevitably limiting their use to datasets of a few thousand samples.

1.2 Problem Statement

The previous section highlighted that NE techniques, despite looking more promising than distance-based techniques, remain very limited in the scope of their use. As explained above, the root of the problem lies in their quadratic complexity, which

prevents their use for large datasets.

One might think that this complexity problem could easily be avoided by employing an NE based method solely on a random subset of the data. This would make it possible to visualize this subset for exploratory analysis. However, analyzing a subset of the data often only reveals a very limited fraction of the relationships present in HD [23].

Another workaround could be to use the HD and LD positions obtained by applying NE to the random subset, in order to project the rest of the data. Next, these positions could be used to train a parametric function between the HD and LD spaces [24], [25]. Although it's probably faster, learning such mappings turns out to be very complicated because the Neighborhood Embedding (NE) method processes only a tiny fraction of the data [26]. This leads to a situation where most of the information about the high-dimensional (HD) structure is ignored because it is carried by the rest of the points [12]. Furthermore, experimental studies have shown that analyzing only a subset of the data poorly generalises the complete data structure [27], thus greatly impacting the representation of the rest of the data.

Finally, it has been shown that, whether you want to analyze the whole dataset or just a fraction of it, considering the whole data set is always advantageous [27]. For this reason, a number of studies have attempted to integrate information from the entire data set into the visualization of a fraction of it. An example of it is landmark t -SNE [12]. However, it turned out that these methods, in addition to limiting visualization, require considerable running time and memory [27].

All the above reasons clearly motivate the process of finding new techniques to speed up NE methods, permitting the visualization of the entire data while still taking into account the information present in all samples.

1.3 Related Work

To remedy the problem formulated above, lower-complexity approximations were introduced, in a first instance, for single-scale methods. These approximations, either relying on tree-based computations like the Barnes-Hut algorithm (BH) [26], [27], **fast multipole methods (FMM)** [2], Negative sampling techniques (NST) [28] or Gaussian mixture approximation methods [23] helped achieving the above-mentioned goals by different means.

Gaussian mixtures, while effectively reducing time complexity to near-linear, are very difficult to tune for (very) large HD spaces [29]. Moreover, they require prior determination of their number of components. Incorrect tuning or a wrong

choice of number of components naturally degrades the quality of the HD space modeling and therefore of the resulting LD embedding.

On the other hand, tree-based methods, FMMs and NST achieve very impressive results, both in terms of quality of results and reduction of time complexity. These methods have made it possible to accelerate NE single-scale algorithms to a complexity of $\mathcal{O}(N \log N)$ and $\mathcal{O}(N)$.

It's important to note that algorithms based on tree-computations or FMM don't scale well proportionally to the number of LD dimensions, i.e. the greater the number of dimensions required in LD, the faster the time complexity will increase. However, as seen earlier, given that in the context of visualization we're mainly aiming for 2 to 3 dimensions in LD space, this constraint is considered negligible.

Finally, one should also consider that the $\mathcal{O}(N \log N)$ and $\mathcal{O}(N)$ complexities can only be reached using these approximations, in the case where the HD similarities can be encoded sparsely. These are therefore particularly well-suited to single-scale NE, in which neighborhood probability distributions remain local. In contrary, they are as it stands, not suitable for multi-scale approaches which attempt to capture both global and local structures. This results in dense neighborhood probability distributions and thus prevents multi-scale methods from being accelerated likewise single-scale methods.

Nevertheless, workarounds have already been found to efficiently accelerate NE multi-scale algorithms. Indeed, a method using the Barnes-Hut (BH) tree algorithm [30] has succeeded in obtaining multi-scale NE algorithms with a complexity $\mathcal{O}(N \log^2 N)$ instead of $\mathcal{O}(N^2 \log N)$ [31]. This method will partly be presented in the third section of this work as we will use some of its mechanisms in our algorithms.

1.4 Contributions to the Domain

As mentioned in the initial contributions list, the major inputs of this work will be to investigate several methods from the FMM family that have already been shown to accelerate single-scale NE algorithms. One will then build on these methods to try to accelerate even more multi-scale NE algorithms.

Fast multipole methods are a family of numerical techniques originally developed to speed up the evaluation of long-range forces in so-called N-body problems. N-body problems define the set of instances in which, in a group of objects, it is necessary to calculate the influence of every single one, on the rest of the group individually. An example of N-body problem is given in section 3.2.3. Such concerns were originally only envisaged in astronomy in order to predict the motion

of celestial objects that were gravitationally interacting. In recent years however, it has been observed that they are applicable to many other situations. A good example are NE methods, where each datapoint has an influence on the similarities of every other datapoint. Fast multipole methods are given this specific name through their evaluations that are approximated based on multipole expansions.

Unlike BH, which can accelerate N-body problems up to $\mathcal{O}(N \log N)$ complexity, FMM methods can reach down to a $\mathcal{O}(N)$ time complexities. Theoretically, it would therefore be possible to speed up multi-scale methods using FMM even more than is already realisable with BH. The aim is to find out whether this is really possible in practice, and if so, to make it happen.

The structure of this work is laid out as follows:

In chapter two, SNE, symmetric SNE and t-SNE, are introduced, with their respective single-scale and multi-scale versions being covered in detail.

Chapter three presents and motivates the usage of two accelerations of respectively single-scale SNE and t-SNE, using two different fast multipole methods. In chapter four these analyses are used to investigate whether it is possible to accelerate their multi-scale versions using the same methods. Chapter five discusses the obtained results and suggests further paths of work. Finally, chapter six concludes the work as a whole.

Chapter 2

Neighbor Embedding (NE)

The objective of DR is to map HD data to a LD space, while maintaining as much of its structure as possible. In other words, to manage to have the neighborhood of a given point in LD space to reflect that of the original HD space as closely as possible. From this point of view, the paradigm of rigid preservation of pair-wise distances is largely surpassed by that of neighbors embedding [4].

The first section of this chapter presents the SNE algorithm, which was the first NE algorithm to be developed. It will then introduce its symmetric version as well as its t-SNE adaptation. The second section reviews their respective multi-scales variants.

The following chapters make use of the following notation conventions:

Let's define $\Xi = \{\xi_i\}_{i=1}^N$ as the set of N data points lying in a HD space of M dimensions. $\mathbf{X} = \{x_i\}_{i=1}^N$ embodies its representation in a LD space of P dimensions. We naturally have $M \geq P$. Finally let δ_{ij} and d_{ij} respectively indicate the HD and LD distances between the i^{th} and the j^{th} datapoints with $i \in I = \{1, \dots, N\}$ and $j \in I \setminus \{i\}$.

2.1 Single-scale Methods

As mentioned above, this section first presents the single-scale version of the initial NE method called Stochastic Neighbor Embedding (SNE). Its symmetric variant as well as its t -SNE variant are then introduced in subsection two and three.

2.1.1 Stochastic Neighbor Embedding (SNE)

As mentioned above, NE methods use conditional neighborhood probabilities. SNE’s first step therefore consists in transforming the set of pair-wise HD distances into these conditional probabilities, which represent similarities. In the case of SNE, these are Gaussian similarities. The similarity of datapoint x_i to datapoint x_j , denoted $\sigma_{ij\star}$, can thus be considered as the conditional probability that point x_i would choose point x_j as its neighbor if neighbors were chosen according to their probability density under a Gaussian centred around x_i . Nearby datapoints therefore have a relatively high $\sigma_{ij\star}$ while well-separated points have nearly infinitesimal $\sigma_{ij\star}$ values. For SNE, the HD similarities are given by

$$\sigma_{ij\star} = \frac{\exp(-\pi_{i\star}\delta_{ij}^2/2)}{\sum_{k \in I \setminus \{i\}} \exp(-\pi_{i\star}\delta_{ik}^2/2)}, \sigma_{ii\star} = 0, \quad (2.1)$$

where $\pi_{i\star}$ represents the precision to obtain the wanted scale. The targeted scale, in the shape of perplexity $K_\star$, is a user-provided parameter. In general single-scale perplexity values range from 5 to 50. It is nevertheless observed that for reasonable values, SNE’s performances are fairly robust to its variation [12]. In order to find the precision $\pi_{i\star}$, related to the distribution $\sigma_{i\star} = \{\sigma_{ij\star}, j \in I \setminus \{i\}\}$ and that corresponds to the provided $K_\star$, a binary search is performed. The precision is then defined s.t.

$$\log K_\star = - \sum_{j \in I \setminus \{i\}} \sigma_{ij\star} \log \sigma_{ij\star}. \quad (2.2)$$

It is important to note that in most cases it is highly unlikely that there is a single optimal value for $\pi_{i\star}$ within the overall dataset [12]. Indeed, the density of data in space is rarely constant. Another specification that has to be made is that in practice the number of iterations of the binary search is most of the time limited to avoid infinite loops. This would notably happen when all HD distances, δ_{ij} , are equal [31]. Finally, it is also interesting to point out, especially for SNE, that the similarities don’t necessarily have to be symmetrical, i.e. $\sigma_{ij\star}$ and $\sigma_{ji\star}$ don’t need to be equal. This matter will be addressed in more detail further down this section.

The LD space similarities can be computed in a similar way as the HD ones. These are denoted $s_{ij\star}$ and are given by

$$s_{ij\star} = \frac{\exp(-p_{i\star}d_{ij}^2/2)}{\sum_{k \in I \setminus \{i\}} \exp(-p_{i\star}d_{ik}^2/2)}, s_{ii\star} = 0, \quad (2.3)$$

where, the LD precisions, $p_{i\star}$, are fixed to 1. If points x_i and x_j are perfectly mapped in the LD space, the conditional probabilities $\sigma_{ij\star}$ and $s_{ij\star}$ must then be equal.

In order to achieve this goal, SNE maps datapoints into LD, trying to minimize the Kullback-Leiber (KL) divergences between the $\sigma_{i\star}$ and $s_{i\star} = \{s_{ij\star}, j \in I \setminus \{i\}\}$ distributions over all points [17]. The KL divergence is a statistical measure of how a probability distribution differs from another one. The cost function to minimize in SNE is then given by

$$C_\star = \sum_{i \in I, j \in I \setminus \{i\}} \sigma_{ij\star} \log(\sigma_{ij\star}/s_{ij\star}). \quad (2.4)$$

The gradient-descent method is used to minimize it [17]. Its gradient is given by

$$\frac{\delta C_\star}{\delta x_i} = 2 \sum_j (\sigma_{ij\star} - s_{ij\star} + \sigma_{ji\star} - s_{ji\star})(x_i - x_j). \quad (2.5)$$

While being of relatively simple form, it can be interpreted as the resultant force created by a set of springs mapping x_i to all other datapoints x_j . These springs exert an attracting or repelling force on each datapoint, depending on whether it is too close or too far away of x_i in its LD space representation, specifically comparing it to its HD position.

2.1.2 Symmetric Stochastic Neighbor Embedding (s-SNE)

An alternative to the SNE we've just presented is its symmetrical version. As seen in the previous section, SNE's similarities don't have to be symmetrical. In the subsequent version they will be made symmetrical. In other words, we will make sure that $\sigma_{ij, sym\star} = \sigma_{ji, sym\star}$ and $s_{ij, sym\star} = s_{ji, sym\star}$. Here, the LD and HD similarities are thus given by

$$\sigma_{ij, sym\star} = (\sigma_{ij\star} + \sigma_{ji\star})/(2N), \quad (2.6)$$

$$s_{ij, sym\star} = \frac{\exp(-p_{i\star} d_{ij}^2/2)}{\sum_{k \in I, l \in I \setminus \{k\}} \exp(-p_{k\star} d_{kl}^2/2)}, s_{ii, sym\star} = 0. \quad (2.7)$$

By symmetrizing the HD similarities, $\sigma_{ij, sym\star}$, in this way, compared to the symmetry is achieved in the LD ones, it is ensured that $\sum_j \sigma_{ij, sym\star} > \frac{1}{2N}$ for each

datapoint ξ_i . Thus, that each HD space datapoint contributes significantly to the cost function [12].

The main advantage of this version over conventional SNE is its simplified gradient form, which can be computed more quickly. It is given by

$$\frac{\delta C_{sym\star}}{\delta x_i} = 4 \sum_j (\sigma_{ij,sym\star} - s_{ij,sym\star})(x_i - x_j). \quad (2.8)$$

Preliminary experiments have shown that symmetric SNE produces embeddings of the same quality as SNE, and sometimes even better [12].

One of both, SNE's and symmetric SNE's major shortcomings is that they are subject to a phenomenon known as the crowding problem [12]. This phenomenon is somewhat related to the norm concentration problem mentioned earlier. For a better understanding, let's illustrate with an example. In a three-dimensional space one can represent four equidistant points, whereas in two-dimensional space only three can be represented at the same distance. The extra point then has to be placed too far away from the others in its 2D representation. When optimizing SNE, the gradients resulting from such a representation induce a slightly attracting force between the point that is too far away and the others. The combination of these attracting forces leads to a collective crush of all points in the centre of the map, hence the name 'crowding problem'. Finally, the given example concerns a reduction from 2- to 3D, but it describes a phenomenon that is naturally exacerbated when dimensions have to be reduced from higher-dimensional spaces.

2.1.3 t-Distributed Stochastic Neighbor Embedding (*t*-SNE)

SNE's extension, *t*-SNE, has been developed to counter the effects of crowding. Besides using symmetric similarities like symmetric SNE, *t*-SNE has one major difference compared to classical SNE. It uses a Student distribution with one degree of freedom and a much heavier tail, instead of a Gaussian distribution that was previously used, in LD space. The HD and LD similarities, denoted $\tau_{ij\star}$ and t_{ij} respectively, are then given by

$$\tau_{ij\star} = (\sigma_{ij\star} + \sigma_{ji\star})/(2N), t_{ij} = \frac{(1 + d_{ij}^2)^{-1}}{\sum_{k \in I, l \in I \setminus \{k\}} (1 + d_{kl}^2)^{-1}}, t_{ii} = 0. \quad (2.9)$$

The use of a Student distribution in LD allows moderate distances in the HD space to be represented by larger distances in the LD space. Naturally, the

undesired attraction forces between moderately dissimilar data are eliminated, thus eliminating crowding [12].

The cost function of t-SNE remains of the same shape as for SNE and is given by

$$C_{t\star} = \sum_{i \in I, j \in I \setminus \{i\}} \tau_{ij\star} \log(\tau_{ij\star}/t_{ij}). \quad (2.10)$$

The gradient resulting from its minimisation, using gradient-descent, is then given by

$$\frac{\delta C_{t\star}}{\delta x_i} = 4 \sum_j (\tau_{ij\star} - t_{ij})(x_i - x_j)(1 + \|x_i - x_j\|^2)^{-1}. \quad (2.11)$$

It is worth bearing in mind that symmetrization and the use of a Student distribution make t-SNE significantly easier to optimize than SNE.

SNE, symmetric SNE as well as t-SNE, all three have a time complexity in their default version of $\mathcal{O}(N^2)$. This originates as, for each datapoint, similarities to all other points of the dataset have to be computed.



Figure 2.1: 2D t-SNE Embedding of the initial 16384D COIL-20 dataset using a 30.0 perplexity

Since its development, t-SNE has met a great success in the field of DR for visualization. Indeed, it tends to produce embeddings with well-defined clusters, making them much easier to read, statement that can easily be confirmed by Figure 2.1 above. Nevertheless, t-SNE doesn't only have advantages. In fact, it tends

to distort large-scale structures and thus be more efficient in representing small neighborhoods. Indeed, given that the similarities of distant points tend to shrink towards zero, t-SNE has a predisposition to consider sufficiently distant points as equally distant [12].

2.2 Multi-scale Methods

As already mentioned, small and medium perplexities are typically used for SNE and t-SNE. As a result there is a focus on small neighborhoods, leading to neglect long- and medium-distance neighborhoods. Indeed, the majority of HD similarities are then infinitesimal, hence hiding LD similarities from the respective cost functions. The use of large perplexities tends to make all HD similarities uniform, which greatly affects small neighborhoods.

NE multi-scale methods however, combine several similarities of different perplexities to best model both global and local trends.

2.2.1 Stochastic Neighbor Embedding (Ms SNE)

As mentioned above, multi-scale SNE (Ms SNE) combines similarities of different perplexities to model both global and local structures [21]. In this version of SNE, single-scale similarities will therefore be indexed, using a scale counter h . For $i \in I$ and $j \in I \setminus \{i\}$, the single-scale similarities are then given by

$$\sigma_{ijh} = \frac{\exp(-\pi_{ih}\delta_{ij}^2/2)}{\sum_{k \in I \setminus \{i\}} \exp(-\pi_{ih}\delta_{ik}^2/2)}, \sigma_{iih} = 0, \quad (2.12)$$

$$s_{ijh} = \frac{\exp(-p_{ih}d_{ij}^2/2)}{\sum_{k \in I \setminus \{i\}} \exp(-p_{ih}d_{ik}^2/2)}, s_{iih} = 0. \quad (2.13)$$

The multi-scale similarities are calculated by averaging the single-scale ones, taking the following form

$$\sigma_{ij} = H^{-1} \sum_{h=1}^H \sigma_{ijh}, s_{ij} = H^{-1} \sum_{h=1}^H s_{ijh}, \quad (2.14)$$

where H is denoting the number of scales.

To evaluate the multi-scale precisions π_{ih} and p_{ih} , the corresponding K_h perplexities are naturally needed. These multi-scale perplexities, that summarize



Figure 2.2: 2D multi-scale SNE Embedding of the initial 16384D COIL-20 dataset

the entire data range, can be defined, unlike single-scale ones, in a totally data-driven way that requires no user intervention. They are introduced as $K_h = 2^{2h-1}$ [21], with a small base perplexity $K_1 = 2$ and $1 \leq h \leq H = \lfloor \log_2(N/K_1) \rfloor$, where $\lfloor \cdot \rfloor$ denotes the rounding.

Using these perplexities, it is then possible to calculate the HD precisions, π_{ih} , in a way equivalent to the calculation of the HD precision of single-scale SNE. Doing this, the entropy of the resulting HD precisions increases linearly across scales. Since the LD coordinates, $\mathbf{X}$, have not yet been determined, multi-scale LD precisions, p_{ih} , cannot be evaluated in the same way. The simplest alternative then consists in fixing these precisions. Therefore these are defined as $p_{ih} = p_h = K_h^{-2U/P}$ with U fixed using the HD precisions as in [21]. In this manner, the LD precisions follow the exponential growth of the previously defined perplexities.

Having all the necessary tools, Ms SNE’s cost function can be calculated with the same procedure as for SNE. Since the entropy of HD similarities doesn’t depend on $\mathbf{X}$, Equation 2.4 is developed as

$$C = - \sum_{i \in I, j \in I \setminus \{i\}} \sigma_{ij} \log s_{ij}. \quad (2.15)$$

This cost function is then also minimised using gradient descent and results to a gradient of the same form as for SNE (Equation 2.5). An embedding that uses Ms SNE can be observed in Figure 2.2.

2.2.2 Symmetric Stochastic Neighbor Embedding (Ms s-SNE)

The symmetrical version of Ms SNE takes a very similar form to its asymmetrical version. As with the single-scale versions of these methods, in Ms s-SNE the similarities are symmetrized compared to Ms SNE. These are then given by

$$\sigma_{ij,sym} = (\sigma_{ij} + \sigma_{ji})/(2N), \quad (2.16)$$

$$s_{ijh,sym} = \frac{\exp(-p_{ih}d_{ij}^2/2)}{\sum_{k \in I, l \in I \setminus \{k\}} \exp(-p_{kh}d_{kl}^2/2)}, s_{iih,sym} = 0. \quad (2.17)$$

The multi-scale LD similarities are then calculated in the exact same manner as done for classical multi-scale SNE, i.e. by taking the mean of each scale. Consequently the resulting cost function and its following gradient are thus identical as in single-scale symmetric SNE except that multi-scale similarities are used instead of single-scales ones.

2.2.3 t-Distributed Stochastic Neighbor Embedding (Ms t-SNE)

Comparable to Ms SNE, multi-scale t-SNE (Ms t-SNE) [22] combines several similarities of different perplexities to best summarize both local and global structures. The distinction between Ms SNE and Ms t-SNE is quite equivalent to the difference between SNE and t-SNE single-scale. Indeed, Ms t-SNE uses symmetrical similarities and a t-Student rather than Gaussian distributions in LD space. Since the t-Student distribution does not depend on perplexities, it is not necessary to index the LD similarity of Ms t-SNE as it was done for Ms SNE. The LD similarity of Ms t-SNE remains effectively the same as for its single-scale counterpart. Ms t-SNE's similarities are therefore given by

$$\tau_{ij} = \frac{\sigma_{ij} + \sigma_{ji}}{2N}, \text{ with } \sigma_{ij} = H^{-1} \sum_{h=1}^H \sigma_{ijh} \text{ and } \sigma_{ijh} \text{ given in Equation 2.12,} \quad (2.18)$$

$$t_{ij} = \frac{(1 + d_{ij}^2)^{-1}}{\sum_{k \in I, l \in I \setminus \{k\}} (1 + d_{kl}^2)^{-1}}, t_{ii} = 0 \text{ as in Equation 2.9.} \quad (2.19)$$



Figure 2.3: 2D multi-scale t-SNE Embedding of the initial 16384D COIL-20 using a perplexity of 30

Here, the evaluation of the HD precisions, π_{ih} , can be done following the same procedure as for Ms SNE. Concerning the LD precisions, as mentioned, there is not needed to evaluate them, as they don't appear in t-Student similarities.

As with Ms SNE, the cost function retains the same form as its single-scale version. Hence, for the same reason as in Ms SNE, the equation 2.10 can be developed as

$$C_t = \sum_{i \in I, j \in I \setminus \{i\}} \tau_{ij} \log(1/t_{ij}). \quad (2.20)$$

Additionally, the gradient, that is used for its minimisation, remains as in Equation 2.11, only replacing $\tau_{ij\star}$ by τ_{ij} . An embedding that uses MS t-SNE can be observed at Figure 2.3.

NE multi-scale methods can maintain large ranges of neighborhoods, compared to single-scale methods, while relieving the user of having to define a $K_\star$ parameter. However, the $\mathcal{O}(\log N)$ scales bring with them a time complexity of $\mathcal{O}(N^2 \log N)$ for all these methods.

Finally, it may be worth stressing out that SNE-like method optimization is indeed very complex [17], [18], [32], [33]. This is particularly true for small perplexities that induce oscillations in the cost functions [21], which explains why heuristics are often used to avoid local minimums are therefore often useful. These include among other early exaggeration and multi-scale optimization. This involves

to start optimizing the different methods with the large perplexities only, and then progressively include the smaller ones.

Chapter 3

Fast Single-scale NE Methods

The quadratic time complexity of NE single-scale methods, with respect to N , limits their potential of use to medium-sized datasets. This motivates to find a way of speeding them up in order for them to be usable for larger datasets.

In order to speed up single-scale methods, it is possible to use the typical smallness of perplexities, K_* , that are supplied by the user. Indeed, this permits the definition of sparse HD similarities with a time complexity of $\mathcal{O}(N \log N)$. It is then possible to accelerate the evaluation of the cost function of NE methods and their gradient by using tree-based methods such as the Barnes-Hut (BH) algorithm [26], [27] or fast multipole methods (FMM) [2].

NE multi-scale methods, however, are based on perplexities that cover all data scales, which results in dense HD similarities. A consequence is thus their prevention from being accelerated in the same way as single-scale methods.

The first section of the upcoming chapter briefly introduces and motivates the impending choice between the Barnes Hut algorithm and Fast multipole methods. The second section then presents two distinct FMM methods, each using a different multipole expansions to accelerate respectively single-scale symmetric SNE's and single-scale t-SNE's cost function and gradient evaluation.

3.1 Fast Multipole Methods versus Barnes-Hut Algorithm

In the current literature, the Barnes-Hut algorithm and Fast multipole methods are the two most widespread approaches to accelerating NE methods. Both are, at the origin, methods used to accelerate so-called N-body problems. As previously

mentioned, N-body problems define the set of instances in which, in a group of objects, it is necessary to calculate the influence of every single one, on the rest of the group individually. An example of an N-body problem is given in section 3.2.3.

Barns-Hut is a tree-based calculation algorithm. It accelerates N-body problems from a time complexity of $\mathcal{O}(N^2)$ to a complexity of $\mathcal{O}(N \log N)$. On the other hand, Fast multipole methods are based on multipole expansions, which under certain conditions, allow N-body problems to be accelerated down to a time complexity of $\mathcal{O}(N)$.

Motivated by the linear time complexity that Fast multipole methods can achieve, the focus to build on these is then logically derived.

3.2 Fast Multipole Methods (FMM)

As touched upon earlier, Fast multipole methods are a family of numerical techniques originally developed to speed up the evaluation of long-range forces in so-called N-body problems. These methods are given this specific name through their evaluations whose approximations are based on multipole expansions.

It is chosen to base this work on two specific variants of fast multipole methods that have already been used in the literature on accelerating NE methods. The first one, called Fast Gauss Transform (FGT), was chosen because it proved accelerating symmetric SNE up to a linear complexity. The second one, called Particle-Mesh (PM), was selected as it has shown to accelerate t-SNE to a time complexity of $\mathcal{O}(M \log M)$, where M is a grid parameter s.t. $M \ll N$ with a dataset of N items, and thus remains faster than Barns-Hut quoted earlier.

The first part of this section shows that the acceleration of NE methods can be divided into two parts for which the first one can effectively be accelerated by relying on the nature of HD similarities. The second and third subsection introduce and provide a detailed development of the two FMM mentioned above.

3.2.1 Computation of Single-scale HD similarities

As indicated previously, the acceleration of single-scale NE methods can be divided into two parts. Firstly, the acceleration of the evaluation of HD similarities, and secondly, the acceleration of the evaluation of their cost functions and gradients. Since without acceleration these two steps have a time complexity of $\mathcal{O}(N^2)$, there is no point in accelerating only one of the two processes if the aim is to effectively reduce the time complexity of the algorithm as a whole.

Since the accelerated computation of HD similarities is common to both of the investigated acceleration methods, it is presented in the beginning of this section. Throughout the next two parts, the two FMMs and their usage to speed up the evaluation of the cost functions and gradients are then introduced.

In order to efficiently evaluate HD similarities, $\sigma_{ij\star}$, advantage can be taken from their Gaussian normalized nature [27]. Indeed, $\sigma_{ij\star}$ disappears as δ_{ij} grows, thus enabling to make a sparse approximation of $\sigma_{ij\star}$ as

$$\tilde{\sigma}_{ij\star} = \begin{cases} \frac{\exp(-\tilde{\pi}_{i\star}\delta_{ij}^2/2)}{\sum_{k \in I_{i\star}} \exp(-\tilde{\pi}_{i\star}\delta_{ik}^2/2)} & \text{if } j \in I_{i\star} \\ 0 & \text{otherwise} \end{cases}, \quad (3.1)$$

with $\pi_{i\star}$ s.t. $\log K_\star = -\sum_{j \in I_{i\star}} \tilde{\sigma}_{ij\star} \log \tilde{\sigma}_{ij\star}$, and where $I_{i\star}$ is a set regrouping ξ_i 's $\lfloor 3K_\star \rfloor$ nearest neighbors in $\Xi \setminus \{\xi_i\}$ [26].

As one can see, the number of neighbors to consider for $\tilde{\pi}_{i\star}$ increases with $K_\star$. Indeed, the entropy of $\sigma_{i\star}$, in Equation 2.2, decreases monotonically with $\pi_{i\star}$ when the distances $\{\delta_{ij}, j \in I \setminus i\}$ are not all equal [34]. Thus, the larger $K_\star$ is, the smaller the resulting $\pi_{i\star}$ will be, which will have flat HD similarities in $\sigma_{i\star}$ as consequence. Therefore, it is necessary to consider more neighbors in Equation 3.1 in order to get a reliable approximation.

The sets of $\lfloor 3K_\star \rfloor$ nearest neighbors can be exactly evaluated by creating a vantage-point tree [35] on Ξ and by performing one depth-first search for every single point's neighborhood on it. The tree can be created in $\mathcal{O}(N \log N)$ time and each search in it costs $\mathcal{O}(K_\star \log N)$ time. Therefore all the neighborhoods sets, $I_{i\star}$ with $i \in I$, can be derived in $\mathcal{O}(K_\star N \log N)$ time.

It's interesting to note that the $\lfloor 3K_\star \rfloor$ exact neighborhoods of each point are used here. However, authors such as Linderman et al. [36], Pezzotti et al. [37] and Tang et al. [28] have shown that using approximate neighborhood search algorithms does not significantly impact the quality of the final visualizations. It would therefore be possible to speed up the calculation of these neighborhoods even more by using such search algorithms in this particular case.

A sparse approximation of the HD similarities can be obtained for s-SNE and t-SNE with $\tilde{\sigma}_{ij, \text{sym}\star} = \tilde{\tau}_{ij\star} = (\tilde{\sigma}_{ij\star} + \tilde{\sigma}_{ji\star})/(2N)$. The matrices $[\tilde{\sigma}_{ij\star}]_{i,j \in I}$, $[\tilde{\sigma}_{ij, \text{sym}\star}]_{i,j \in I}$ and $[\tilde{\tau}_{ij\star}]_{i,j \in I}$ then have $\mathcal{O}(K_\star N)$ non-zero entries instead of $\mathcal{O}(N^2)$ as in their dense versions.

3.2.2 Fast Gauss Transform (FGT) for symmetric SNE

This section first elaborates on the used fast multipole method based on Fast Gauss Transform and secondly shows how it can be applied to accelerate single-scale SNE's cost function and gradient evaluation.

Fast Gauss Transform (FGT)

As previously explained, FMMs were initially used in astrophysics to evaluate gravitational interactions between many particles [38]. The idea behind FMM is to transform interactions between points into local series expansions around these same points. Truncating these series then reduces the complexity of computing these interactions. The FMM presented in this section is called **Fast Gauss Transform (FGT)** [39] and is used to calculate the sum of Gaussian interactions such that

$$Q(x_n) = \sum_{m=1}^N q_m \exp(-\|(x_n - x_m)/\sigma\|^2), \quad (3.2)$$

for a set of points $\mathbf{X} = \{x_i\}_{i=1}^N$ and a standard deviation σ . As SNE and symmetric SNE's HD and LD similarities are derived from Gaussian interactions, FGT is well suited to their acceleration.

For the sake of clarity, in this explanation, a distinction is made in $\mathbf{X}$, between source points s_j , as points influencing other points, and target points t_i , influenced by sources. In our case, however, since every point influences all the other points, each one of them can be both a source and a target, depending on the situation.

In FGT the dataset is normalized so that all its points lie in a box of dimension $[0, 1]^d$. This box is then subdivided into smaller boxes of size $\sqrt{2}\sigma r$ with $r < 1/2$. Each source point s_j , is then assigned to the box B in which it lies, and each target point t_i , to the box C in which it lies. As mentioned above, we distinguish between B and C boxes, but in this situation each box could be one or the other.

In order to compute the sums of Gaussians, one then needs to calculate the influence, i.e. the Gaussian field, of all the points in all the boxes B on each target in C , for each target box C . In practice, however, given that the Gaussian $e^{-\|(t-s)/\sigma\|^2}$ have a range of $\mathcal{O}(\sqrt{\sigma})$ and that the boxes, the space is divided into, have a side length of $\sqrt{2}r\sigma$. Only a fixed number of "nearest" boxes can make a non-negligible contribution to the fields of a given target box C . In the presented algorithm, this

number of boxes can be defined as a variable k by the user.

Next, it is important to define how FGT calculates the influence of N_B points that are in a box B on M_C points in a box C . FGT allows B to transmit its influence to C in two separate ways, besides allowing C to process the information received by B in two different manners.

On one hand, B can individually send the strength and centre of the Gaussians of its N_B points to C , as in classical SNE. On the other hand it can accumulate all the Gaussians of its points in a truncated Hermite expansion about its centre s_B and only send this expansion to C . Indeed, the Gaussian fields due to all sources in B can be expressed as ¹

$$\sum_{j=1}^{N_B} q_j \exp(-\|(\mathbf{t} - s_j)/\sigma\|^2) = \sum_{\alpha \geq 0} A_\alpha h_\alpha\left(\frac{\mathbf{t} - s_B}{\sqrt{\sigma}}\right), \quad (3.3)$$

where $\mathbf{t}$ is the target point, $h_n(t) = e^{-t^2} H_n(t)$ are Hermite functions with Hermite polynomials $H_n(t)$ and where the coefficients A_α are given by

$$A_\alpha = \frac{1}{\alpha!} \sum_{j=1}^{N_B} q_j \left(\frac{s_j - s_B}{\sqrt{\sigma}}\right)^\alpha. \quad (3.4)$$

Similarly, the first option for the C box is to directly evaluate all fields (Gaussians or Hermite expansions) sent to it, at the M_C targets it contains. As an alternative it can convert all these fields into a single truncated Taylor expansion about its centre t_C . Indeed, a Hermite expansion truncated after p terms can be expressed as ¹

$$\sum_{\alpha \leq p} A_\alpha h_\alpha\left(\frac{\mathbf{t} - s_B}{\sqrt{\sigma}}\right) = \sum_{\beta \geq 0} C_\beta \left(\frac{\mathbf{t} - t_C}{\sqrt{\sigma}}\right)^\beta, \quad (3.5)$$

where the coefficients C_β are given by

$$C_\beta = \frac{(-1)^{|\beta|}}{\beta!} \sum_{\alpha \leq p} A_\alpha h_{\alpha+\beta}\left(\frac{s_B - t_C}{\sqrt{\sigma}}\right). \quad (3.6)$$

¹ We use multi-index notation: $\alpha \geq 0 \Rightarrow \alpha_1, \dots, \alpha_d \geq 0$; $\alpha! = \alpha_1! \cdots \alpha_d!$; $\beta \geq 0 \Rightarrow \beta_1, \dots, \beta_d \geq 0$; $\beta! = \beta_1! \cdots \beta_d!$; $\mathbf{t}^\alpha = t_1^{\alpha_1} \cdots t_d^{\alpha_d}$ for $\alpha, \beta \in \mathbb{N}^d, \mathbf{t} \in \mathbb{R}^d$

Details on how to transform the individual Gaussians into Taylor series are given in Appendix A.

The FGT approximation comes from truncating the series expansions presented above up to terms of order p . Since it then contains p terms per dimension, this reduces the complexity of computing the Gaussian sums from $\mathcal{O}(N^2)$ to $\mathcal{O}(Np^d)$. A detailed analysis of complexity is provided in Appendix A. It is also essential to point out that the FGT approximation has well-known approximation error bounds, which can be expressed as functions of p . It can therefore be shown that the Taylor series developed in Equation 3.5 converge rapidly in a box of size $\sqrt{2}r\sigma$.

Coming back to the previously studied case, there are thus four ways a box B can influence a box C :

1. N_B Gaussians $\rightarrow$ directly sent to C
2. N_B Gaussians $\rightarrow$ accumulated in Hermite series (3.3)
3. Hermite series and Gaussians $\rightarrow$ directly evaluated on M_C points
4. Hermite series and Gaussians $\rightarrow$ accumulated in Taylor series (3.5)

The choice of either making an exact evaluation or approximating using series, whether for the source box B or the target box C , then depends on the number of points the boxes contain. Greengard and Strain (1991) [39] therefore propose to introduce two cutoff parameters N_F and M_L . If the number of points in a box B , $N_B < N_F$, then B sends its N_B Gaussians individually and thus its exact field to C . Otherwise, B sends its approximate field in the form of a Hermite expansion. Similarly, if in a target box C , $M_C < M_L$, C evaluates all fields it has received immediately. If $M_C \geq M_L$, C then transforms all fields it has received into Taylor series, accumulates all the coefficients and finally evaluates the Taylor series. As for the number of boxes to consider for a given target box, these cutoff coefficients can be set in form of a variable by the user. An example of the four cases explained here can be seen in Figure 3.1 below.

After having applied the above strategy to all the target boxes and their respective source boxes, all targets should have accumulated fields that are already evaluated on them (case 1. and 3.) as well as Taylor series coefficients (case 2. and 4.). In order to obtain the total field acting on each of these targets, there is thus one final step to compute. Indeed, for each target, a Taylor series using the target's gotten coefficients needs to be evaluated and added to the already calculated fields, which will then finally result in an approximation of (3.2).

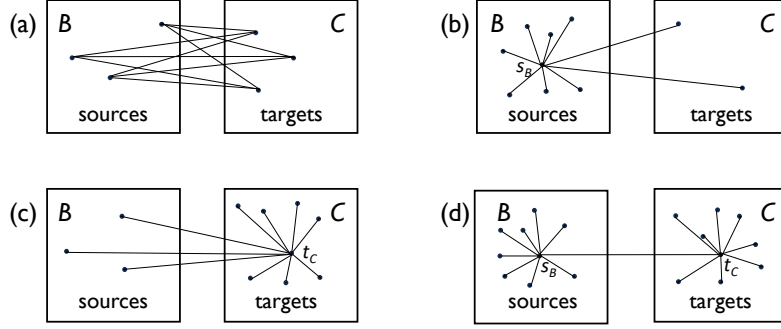


Figure 3.1: Different FGT field approximations. (a) exact interaction (3.2) (few points in both boxes); (b) Hermite expansion around s_B directly calculated on C 's points (many source points and few target points); (c) B 's individual Gaussians sent to C and approximated by a Taylor expansion around t_C (few source points and many target points); (d) Hermite expansion around s_B , then Taylor expansion to the Hermite functions around t_C (many points in both boxes).

In machine learning, FGT has already been used to speed up problems such as kernel density estimation [40] or matrix inversion and eigendecomposition [41]. One of its major shortcomings, however, is its limitation to small d dimensions. Its time complexity $\mathcal{O}(p^d N)$, that depends exponentially on d , scales very poorly for large dimensions. An improved FGT version has been proposed by Yang et al. (2003) [42] that remedies this issue. Improved FGT uses clustering and other techniques to divide the grid into data-dependent regions. Next to that IFGT also uses a modified version of Taylor expansions allowing the complexity to be reduced from $\mathcal{O}(Np^d)$ to $\mathcal{O}(Nd^p)$. This allows for the use of higher dimensions, but does not completely solve the dimension scalability problem. What's more, IFGT requires the careful setting of numerous parameters. If not done correctly, the overhead of IFGT can be so large that an exact field calculation would actually be shorter.

Applying FGT to symmetric Single-scale SNE

In single-scale NE methods, the N-body problem appears in the evaluation of both the cost function and the gradient. In fact, it is necessary to calculate the interactions between all datapoints in the form of similarities, for both. As we have mentioned earlier, FGT uses Gaussian interactions and could therefore be perfectly suitable for SNE and symmetric SNE. Nevertheless, a take-away from this section enlightens that for the appliance of FGT to NE methods, symmetric similarities are needed. As classical SNE doesn't respect this criterion the next

elaborates on how FGT can be used to accelerate single-scale symmetric SNE [2]. In a first step to its appliance, it is essential to start by rewriting symmetric SNE's cost function. Indeed, since the entropy of HD similarities doesn't depend on $\mathbf{X}$, it can be developed as

$$\begin{aligned}\tilde{C}_{sym\star} &= \sum_{i \in I, j \in I \setminus \{i\}} \tilde{\sigma}_{ij, sym\star} \frac{d_{ij}^2 p_{i\star}}{2} + \log\left(\sum_{k \in I, l \in I \setminus \{k\}} \exp\left(\frac{-p_{k\star} d_{kl}^2}{2}\right)\right) \\ &= E^+(\mathbf{X}) + E^-(\mathbf{X})\end{aligned}\quad (3.7)$$

with $\sigma_{ij, sym\star}$ and $s_{ij, sym\star}$ respectively given in Equation 2.6 and Equation 2.7. Here E^+ is an "attractive" term that pulls points, that were close in the HD space, together in the LD space. And E^- is a "repulsive" term that pulls all points apart from each other. Furthermore the gradient of Equation 3.7 can be written as

$$\frac{\delta \tilde{C}_{sym\star}}{\delta \mathbf{X}} = 4\mathbf{X}(\mathbf{L} - \tilde{\mathbf{L}}), \quad (3.8)$$

with the graph Laplacians defined as

$$\begin{aligned}\mathbf{L} &= \text{diag}\left(\sum_{j=1}^N \tilde{\sigma}_{ij, sym\star}\right) - \boldsymbol{\sigma}_{sym\star}, \\ \tilde{\mathbf{L}} &= \text{diag}\left(\sum_{j=1}^N s_{ij, sym\star}\right) - \mathbf{s}_{sym\star},\end{aligned}\quad (3.9)$$

where $\tilde{\boldsymbol{\sigma}}_{sym\star} = \{\tilde{\sigma}_{i, sym\star}, i \in I\}$ with $\tilde{\sigma}_{i, sym\star} = \{\tilde{\sigma}_{ij, sym\star}, j \in I \setminus \{i\}\}$ and $\mathbf{s}_{sym\star} = \{s_{i, sym\star}, i \in I\}$ with $\mathbf{s}_{i, sym\star} = \{s_{ij, sym\star}, j \in I \setminus \{i\}\}$. Specifically the use of Laplacians in the gradient formulation above requires similarities to be symmetrical, limiting to the acceleration of symmetric SNE.

In the new cost function formulation (3.7) it is clear that two N-body problems appear, namely one in each of the attractive and repulsive terms. The evaluation of the attractive term can be mitigated using the normalized Gaussian nature of $\sigma_{ij, sym\star}$. As noted in section 3.2.1, this then results in a sparse matrix of HD similarities, $\boldsymbol{\sigma}_{sym\star}$. The repulsive term, however, is not sparse and implies an N-body problem in the shape of a sum of Gaussian similarities between all LD datapoints.

As for the new gradient formulation, it implies in its first term the graph Laplacian

$\mathbf{L}$ which depends on $\sigma_{sym\star}$, that is, as previously mentioned, sparse and can therefore be calculated quickly. Its second term, however, it involves the graph Laplacian $\tilde{\mathbf{L}}$ which depends on $\mathbf{X}$ through $\mathbf{s}_{sym\star}$ and is as explained not sparse.

Knowing this, one can define two interactions as follows

$$S(x_i) = \sum_{j=1}^N \exp\left(-\frac{d_{ij}^2 p_{i\star}}{2}\right), \quad (3.10)$$

$$S^x(x_i) = \sum_{j=1}^N x_j \exp\left(-\frac{d_{ij}^2 p_{i\star}}{2}\right). \quad (3.11)$$

Using these interactions one can finally rewrite single-scale symmetric SNE's the cost function (3.7) and gradient (3.8) as

$$\tilde{C}_{sym\star} = \sum_{i \in I, j \in I \setminus \{i\}} \tilde{\sigma}_{ij, sym\star} \frac{d_{ij}^2 p_{i\star}}{2} + \log\left(\sum_{i \in I} S(x_i)\right), \quad (3.12)$$

$$\frac{\delta \tilde{C}_{sym\star}}{\delta \mathbf{X}} = 4\mathbf{X}\mathbf{L} - 4Z(\mathbf{X})\mathbf{X} \text{diag}(S(\mathbf{X})) + 4Z(\mathbf{X})S^x(x_i). \quad (3.13)$$

Knowing that for single-scale symmetric SNE the precisions are set as $p_{i\star} = 1$ for $i \in I$. Using FGT, $S(x_i)$ can thus be evaluated by taking, in (3.2), $\sigma = 1$ as well as $q_n = 1$ for $n \in I$. Similarly, $S^x(x_i)$ can be evaluated by taking $\sigma = 1$ as well as $q_n = x_{kn}$ and evaluating the formula d times for $k = 1, \dots, d$ with d the number of dimensions in LD space.

Using the method presented in section 3.2.1 and FGT as described above, the time complexity of single-scale symmetric SNE can be reduced down at two distinct places. Firstly, it allows the evaluation of HD similarities, at initialization of the algorithm, with a time complexity of $\mathcal{O}(N \log N)$. Secondly, it allows the reduction of the time complexity of gradient-based optimization iterations to $\mathcal{O}(N)$. Although the total complexity of the algorithm remains $\mathcal{O}(N \log N)$, a computation of this complexity only needs to be performed once at initialization. The rest of the iterations can then be performed with linear complexity, which is far from being negligible. An embedding using this method can be observed in Figure 3.2 bellow.

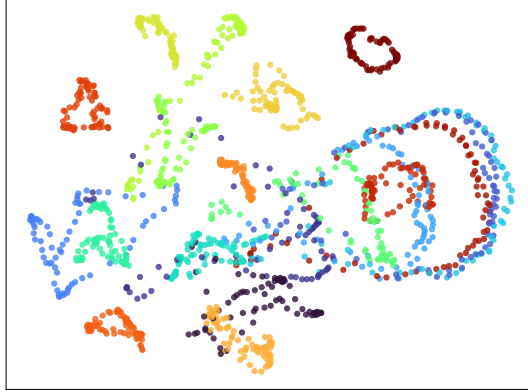


Figure 3.2: 2D FGT SNE Embedding of the initial 16384D COIL-20 dataset

3.2.3 Particle-Mesh (PM) for t-SNE

The organization of this subsection follows that of the paper from which the method is taken [1]. The first part initiates by presenting an example of the N-body problem in Electrostatics. The second part then builds on this problem to present an algorithm, called Particle-Mesh (PM), that uses fast Fourier transforms to solve it. Finally, the third part of the subsection spells out how this algorithm can be adapted to accelerate single-scale t-SNE.

The N-Body Problem in Electrostatics

As explained above, the N-body problem appears in physics whenever multiple bodies interact with each other, and has no analytical solution as soon as $N > 2$. Particle mesh (PM) is an algorithm initially developed to solve the N-body problem in electrostatics. In this section, the very specific case of this N-body problem is thus furnished. It should be noted, however, that developments of N-body problems in other fields are very similar to this one.

Let's consider a set of punctual charges with charge $\{q_i\}_{i=1}^N$ and given positions in 3D space $\{r_i\}_{i=1}^N$. These charges exert forces on each other according to Coulomb's law, given by

$$F(r_i) = \frac{q_i}{4\pi\epsilon_0} \sum_{j=1}^N \frac{r_i - r_j}{\|r_i - r_j\|^3} q_j, \quad (3.14)$$

where ϵ_0 is the electric constant. Note that this force can also be expressed for charges in a space with a different number of dimensions.

Nevertheless, in this case, if the point charges were to be continuously distributed, Equation 3.14 could be expressed as

$$F(r_i) = \frac{q_i}{4\pi\epsilon_0} \iiint \frac{r_i - r'}{\|r_i - r'\|^3} \rho(r') d^3r', \quad (3.15)$$

where $\rho(r)$ is the charge distribution in space, i.e. representing the amount of charge at position r .

As for the N-body problem, here, it then consists in evaluating Equation 3.14 for all $\{r_i\}_{i=1}^N$.

Particle-Mesh algorithm

Now that the N-body problem has been introduced, the next logical step is to describe how the Particle-Mesh (PM) algorithm uses fast Fourier transforms (FFT) to solve it.

To do so, the PM operations are based on the use of the electrostatic potential. The latter is given by

$$\phi(r) = \frac{1}{4\pi\epsilon_0} \iiint \frac{\rho(r')}{\|r - r'\|} d^3r'. \quad (3.16)$$

It can in fact be used to compute the previously seen exerted forces on each particle (3.15). Indeed, one has $F(r) = -q_i \nabla \phi(r)$. The computation of $\phi(r)$ would therefore allow to evaluate the set $\{F(r_i)_{i=1}^N\}$. The main advantage of using $\phi(r)$, instead of directly evaluating $\{F(r_i)_{i=1}^N\}$, is that $\phi(r)$ can be expressed as a convolutional product and that these can be evaluated efficiently. Indeed, Equation 3.16 can be expressed as

$$\phi(r) = \frac{1}{4\pi\epsilon_0} (\rho(r) \star h(r)), \quad (3.17)$$

where $h(r) = 1/\|r\|$ and $\star$ denotes a convolutional product. Using the Fourier Transforms (FTs), this convolutional product, which is difficult to evaluate, can then be transformed into a much easier to compute dot product. This is given by

$$\phi(r) \propto (\rho(r) \star h(r)) \iff \hat{\phi}(r) \propto (\hat{\rho}(r) \hat{h}(r)), \quad (3.18)$$

with $\hat{\cdot}$ denoting the FT of the given function. After evaluating this dot product, $\phi(r)$ can then be recovered by taking the inverse Fourier Transform of $\hat{\phi}(r)$.

To do all this, however, the load distribution $\rho(r)$ is needed. To obtain it from the q_i charges and the r_i positions of the punctual charges, the space first has to be discretized, which can be done in several ways.

In classic PM, the space is first divided into a grid of M equivalent cells and each cell is assigned a point called grid point. The M parameter is here a variable that has to be set by the user.

The simplest way to proceed is then to assign the q_i charge of each punctual charge to the nearest grid point. This method corresponds to a first-order interpolation scheme and is also known as Nearest Grid Point (NGP). Despite its simplicity, however, it can quickly give rise to significant errors for sparsely populated datasets. For very large datasets, nevertheless, these errors tend to be compensated.

Another way of proceeding is to use a 2nd-order interpolation scheme. In 3D space, a part of the q_i charge of each punctual charge is then assigned to the 8 nearest grid points. The portion of charge assigned is inversely proportional to the distance separating the grid point and the punctual charge in question. This solution is also known as Cloud in Cell (CIC). An example of the two methods of charge allocation can be seen in the 2D space example in Figure 3.3 below.

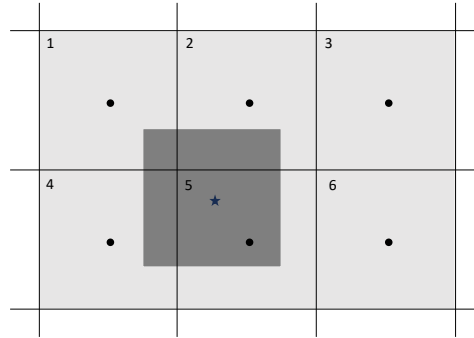


Figure 3.3: Grid points are represented by the centred black points; the punctual charge is represented by the star and the cloud surrounding it is the region of space that indicates its uniformly distributed charge. When using NGP, the charge would be allocated to cell 5; whereas when using CIC the charge would be allocated partly to cell 1, 2, 4 and 5, the bigger fraction added to 5 and the smaller fraction to 1.

It's important to note that the method that is chosen to evaluate the charge distribution influences how the gradient should be calculated later on. Indeed, if a first-order method is used to evaluate $\rho(r)$, it makes no sense to use a second-order

method to evaluate $\nabla\phi(r)$. The same reasoning applies to calculating the force exerted on each particle. The order of the method chosen to approximate $\rho(r)$ will then define the order of the approximation methods to be chosen for the rest of the calculations.

Evaluating $\phi(r)$ without using FTs, i.e. by evaluating the convolutional product directly, would have a complexity of $\mathcal{O}(M^2)$. Given that for very large datasets we can have $M \ll N$, this would already be an improvement on the brute force algorithm. The use of FTs, in our case Fast Fourier Transforms (FFT), however, allows this complexity to be reduced to $\mathcal{O}(M \log M)$.

Applying PM and FFT to Single-scale t-SNE

Just as done earlier when presenting the acceleration of single-scale SNE with FGT, it is easily proven that N-body problems can be found in single-scale t-SNE's cost function and gradient. In order to do this, t-SNE's cost function (2.10) first has to be reformulated as follows

$$\begin{aligned}\tilde{C}_{t\star} &= - \sum_{i \in I, j \in I \setminus \{i\}} \tilde{\tau}_{ij\star} \log(1 + \|x_i - x_j\|^2)^{-1} + \log Z \\ &= E_t^+(\mathbf{X}) + E_t^-(\mathbf{X})\end{aligned}\tag{3.19}$$

where $Z = \sum_{k \in I, l \in I \setminus \{k\}} (1 + \|x_k - x_l\|^2)^{-1}$ and $\tau_{ij\star}$ is given in Equation 2.9. As explained for SNE in the previous section, E_t^+ is here an "attractive" term that pulls points, that were close in the HD space, together in the LD space. Additionally, E_t^- is a "repulsive" term that pulls all points apart from each other.

Similarly, one can rewrite t-SNE's gradient (2.11) as follows

$$\begin{aligned}\frac{-1}{4} \frac{\delta \tilde{C}_{t\star}}{\delta x_i} &= \sum_j \tilde{\tau}_{ij\star} \frac{x_j - x_i}{1 + \|x_i - x_j\|^2} + \sum_j \frac{1}{Z} \frac{x_i - x_j}{(1 + \|x_i - x_j\|^2)^2} \\ &= \sum_j F_{i,j}^{attr} + \sum_j F_{i,j}^{rep}\end{aligned}\tag{3.20}$$

where $F_{i,j}^{attr}$ can be seen as an attractive force and $F_{i,j}^{rep}$ as a repulsive force, both exerted on x_i respectively towards and opposite to x_j . Moreover one can interpret $1/Z$ as a balancing factor between the attractive and repulsive forces.

In the new formulation of the cost function (3.19), two N-body problems come to evidence. The first in E_t^+ , whose evaluation can be mitigated by the

sparsity of $\tau_{ij\star}$ as pointed out in section 3.2.1. The second in E_t^- , however, is not sparse and is expressed in the form of a sum of t -Student distributions.

Besides, the new gradient formulation (3.20) also contains two N-body problems. The first in F_{ij}^{attr} can also be mitigated thanks to the sparsity of $\tau_{ij\star}$ and therefore evaluated using the method presented in section 3.2.1. As for the second in F_{ij}^{rep} , it is again not sparse and is accordingly expressed in the form of a sum of t -Student similarities.

As it has already been described how to efficiently evaluate E_t^+ and F_{ij}^{attr} , the focus now shifts unto calculating E_t^- and F_{ij}^{rep} . To initiate lets lay out F_{ij}^{rep} in Equation 3.20.

When assuming $q_i = 1, \forall i$ in Equation 3.14 a significant similarity with the term F_{ij}^{rep} becomes visible. Posing this, would intuitively mean that no instance in the dataset is favoured. Using PM, the studied term could then be calculated by discretizing the space using the methods seen previously, as follows

$$F_i^{rep} = F^{rep}(x_i) = \frac{1}{Z} \sum_{g=1}^M \rho_g \frac{x_i - x_g}{(1 + \|x_i - x_g\|^2)^2}, \quad (3.21)$$

where ρ_g is the number of instances assigned to the g^{th} grid point and g is an index iterating through M , i.e. the number of cells and thus grid points. If one considers that $M \rightarrow \infty$, the finite summation of Equation 3.21 becomes an integral given by

$$F^{rep}(x_i) = \frac{1}{Z} \iint \rho(r') \frac{x_i - r'}{(1 + \|x_i - r'\|^2)^2} d^2 r'. \quad (3.22)$$

Now, in order to use PM for t -SNE as presented earlier, a potential such as $-\nabla\phi(r) \propto F^{rep}(r)$ is required. This potential can be expressed as

$$\phi(r) = \rho(r) \star w(r), \text{ where } w(r) = \frac{1}{1 + \|r\|^2}. \quad (3.23)$$

Indeed, it can then be shown that

$$\begin{aligned}
\frac{-1}{2Z} \nabla \phi(r) &= \frac{-1}{2Z} \nabla (\rho(r) \star \frac{1}{1 + \|r\|^2}) \\
&= \frac{-1}{2Z} \iint \rho(r') \nabla_r \frac{1}{(1 + \|r - r'\|^2)^2} d^2 r' \\
&\stackrel{2}{=} \frac{1}{Z} \iint \rho(r') \frac{r - r'}{(1 + \|r - r'\|^2)^2} d^2 r' \\
&= F^{rep}(r).
\end{aligned} \tag{3.24}$$

As in the original PM, Equation 3.23 contains a convolutional product that can be efficiently evaluated using FTs. Having been able to express the gradient as it is done above, the difference between PM and t -SNE is almost wiped out. Nevertheless, a successful evaluation of Z , which will allow to finish evaluating not only the gradient but also $E_t^-(X)$ of the cost function (3.19), is still needed. Fortunately the potential expressed above can be relied on. Indeed, now having

$$\phi(r) = \iint \rho(r') \frac{1}{1 + \|r - r'\|^2} d^2 r' \tag{3.25}$$

one can compute Z as follows

$$Z = \sum_k \phi(x_k). \tag{3.26}$$

This can be shown by starting from Z 's original formula and developing it considering a grid of infinite resolution. The succinct development can be found in Appendix B.

The calculation of the potential therefore allows to fully evaluate the terms F_{ij}^{rep} and E_t^- . What's more, since F_{ij}^{rep} can be expressed as a convoluted product, its evaluation can be carried out efficiently using FTs and inverse FTs. A schematic diagram of all the steps involved in calculating Z and F_{ij}^{rep} can be seen in Figure 3.4 below.

It's worth pointing out that this method, here used for accelerating t -SNE, uses data interpolation on a grid and is therefore less effective for small datasets. In fact, using this grid can generate approximation errors in areas of the space that are less data dense, which intuitively are less present in large datasets.

²considering that $\nabla \frac{1}{1 + \|r\|^2} = \frac{\partial}{\partial \|r\|} \frac{1}{1 + \|r\|^2} \frac{r}{\|r\|} = \frac{-2r}{(1 + \|r\|^2)^2}$

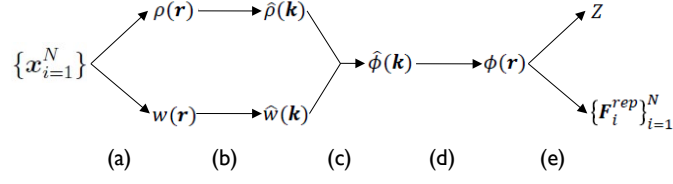


Figure 3.4: Schematic diagram of the computation of F_{ij}^{rep} and Z . In step (a) one computes w and ρ by discretizing the LD space using M grid points. Step (b) consists in applying FT on ρ and w in order to be able to evaluate ϕ efficiently. Step (c) resides in computing the dot product between the two previous obtained FTs. Step (d) consist in applying the inverse FT on $\hat{\phi}$ to get ϕ . Finally, step (e) subsists in computing both F_{ij}^{rep} and Z using Equations 3.24 and 3.26

An embedding resulting from the use of PM t -SNE can be seen in Figure 3.5 below. In these circumstances, for the reason mentioned above, the MINST dataset was used, which is larger than the Coil-20 dataset used previously.

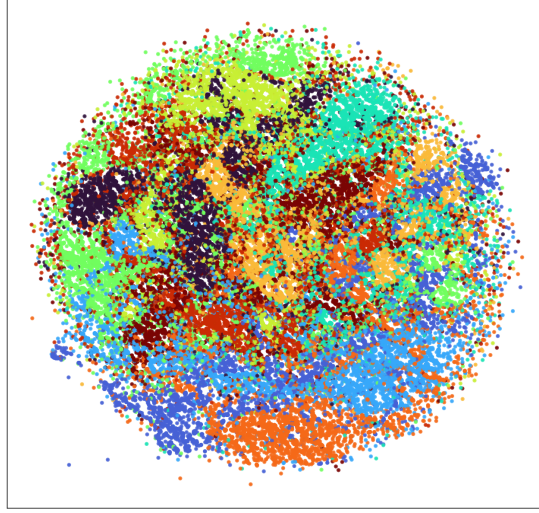


Figure 3.5: 2D single-scale PM t -SNE Embedding of the initial 50D MINST dataset using a perplexity of 30

Chapter 4

Developing Fast Multipole Methods for Multi-scale NE

Using the methods presented in the third chapter, the time complexity of two NE single-scale methods were successfully reduced. Indeed, in the first instance, a fast multipole method using Fast Gauss Transform (FGT) enabled the reduction of the time complexity of symmetric SNE's cost function and gradient computation from $\mathcal{O}(N^2)$ to a linear complexity of $\mathcal{O}(N)$. Then, in a second phase, using a fast multipole method based on Fast Fourier Transform (FFT) and an adapted version of the the Particle Mesh (PM) algorithm, one were able to reduce the time complexity of single-scale t -SNE from $\mathcal{O}(N^2)$ to $\mathcal{O}(M \log M)$, where M is a grid parameter s.t. $M \ll N$ with a dataset of N samples. However, as explained, these accelerations partly rely on the ability to encode HD similarities sparsely.

On the other hand, multi-scale methods with a time complexity of $\mathcal{O}(N^2 \log N)$, rely on perplexities covering data at different scales. These result in dense HD similarities. For this reason, coupled with other factors, it is not possible to accelerate NE multi-scale methods in the same way as it has been done up until here for single-scale methods.

Nevertheless, recalling from the second chapter, multi-scale methods are a better option than single-scale ones at maintaining both the small- and long-range neighborhoods. Consequently, this engenders LD embeddings that retain both the global and local structure of the initial HD embedding in an improved way. Knowing all this, it highly motivates to also find a way of speeding up these methods so that they can be used on large datasets.

Fortunately, as mentioned in the introduction, a method using the Barnes-Hut tree algorithm has already been found to speed up multi-scale NE methods [31]. It has succeeded in developing a way of making multi-scale HD similarities sparse, and has thus been able to reduce the time complexity of multi-scale NE methods to $\mathcal{O}(N \log^2 N)$.

Similar to the acceleration of single-scale methods, the acceleration of multi-scale methods can be divided into two distinct stages. Firstly, the evaluation of HD similarities, and secondly, the evaluation of the cost function and gradients. This section builds on the method developed by de Bodt et al. [31] to make multi-scale HD similarities sparse, as well as on the FMM methods presented in the previous chapter, in order to develop a new method for accelerating multi-scale schemes.

The first section of this chapter thus shows how it is possible to compute the multi-scale HD similarities in a sparse way using the method presented in [31]. The second section then describes how one tried to adapt the first analysed Fast multipole method, which uses the Fast Gauss Transform and is presented in section 3.2.2, to the multi-scale scheme and why it failed. Finally the third section will discuss how the second analysed Fast multipole method, introduced in 3.2.3 and which uses the Particle-Mesh algorithm, can be adapted to multi-scale t -SNE.

4.1 Computation of Multi-scale HD similarities

A crucial aspect of single-scale methods is their sparse HD similarities evaluated with a time complexity of $\mathcal{O}(N \log N)$. As presented in section 3.2.1, these are due to the set of nearest neighbors $I_{i\star}$ such that $|I_{i\star}| = \lfloor 3K_{i\star} \rfloor$.

If the same procedure was to be used for multi-scale methods, there would be dense HD similarities. Indeed, given that the perplexity K_H generally lies in an $\mathcal{O}(N)$ interval and that $|I_{i\star}| = \min(\lfloor 3K_H \rfloor, N - 1)$, the set of nearest neighbors I_{iH} would have a size of $\mathcal{O}(N)$ for $i \in I$. Having that the complete nearest neighbor sets of multi-scale are obtained such that $I_i = \cup_{h=1}^H I_{ih}$, all sets would be such that $I_i = I \setminus \{i\}$. The resulting HD similarities would then have $\mathcal{O}(N^2)$ entries.

To solve this problem, de Bodt et al. [31] started with an intuition. Indeed, single-scale HD similarities with small perplexities excel at modeling local interactions around each point but reduce the influence of other, more distant HD objects. In fact, they tend to concentrate their mass probability on the nearest neighbors of each data point. It is therefore crucial to exactly take these neighbors into account. On the other hand, HD similarities with larger perplexities capture more global interactions. Given that these more global structures emerge from average patterns over a much larger number of data items than just the nearest

neighbors, it's unlikely that this estimation will require exhaustive consideration of all these data. Indeed, while local short-range interactions are due to small, very precise neighborhoods, more global interactions result from a consensus on a much larger number of data items and could be fairly well approximated by a subset of these data items. Evaluating, in the cost function of NE methods, the divergence between the probability distributions on such subsets could then may be sufficient to shape LD embedding in accordance with HD global structures.

HD similarities, however, combine both small and large perplexities. In order to accelerate multi-scale methods effectively, small and relevant subsets would then have to be defined, denoted $\tilde{I}_i$ for $i \in I$, which would gather data modelling the entire data cloud, while at the same time targeting each ξ_i nearest neighbors. Finally, the evaluation of the cost function could be limited to these subsets. Given the hierarchical structure of multi-scale perplexities, from small to large, this intuition suggests that the subset $\tilde{I}_i$ should gather data lying at different ranges from ξ_i . Indeed, for small scales, the exact nearest neighbors are necessary, but as the scale increases, a handful of more distant data, characterising the HD space more roughly, would suffice.

Following this principle, a hierarchy $\{\Xi_h\}_{h=1}^H$ of subsampled datasets for each scale h can be defined. These would be formed in such a way that Ξ_h contains $\lfloor 2^{1-h}N \rfloor$ elements drawn without replacements and at random from Ξ . One can note that doing this one has $\Xi_1 = \Xi$. Hence, this set of HD subsampled datasets is illustrated in Figure 4.1 below.

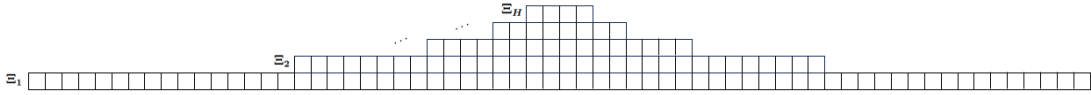


Figure 4.1: Hierarchy of subsampled HD datasets where one square stands for one data sample

Having done this, a vantage-point tree can be created, similar to the put up in section 3.1.1 for the whole dataset, on each subsampled dataset. Each tree can be created with a time complexity of $\mathcal{O}(2^{1-h}N \log(2^{1-h}N))$, which results in all trees being created with a time complexity of $\mathcal{O}(N \log N)$.

The set $\tilde{I}_{ih}$ of the $\lfloor 3K_1 \rfloor$ nearest neighbors of ξ_i in $\Xi_h \setminus \{xi_i\}$ can then be created using the h^{th} tree. Subsequently, the time complexity of evaluating each of these sets then takes $\mathcal{O}(K_1 \log(2^{h-1}N))$ time, allowing to obtain the sets $\tilde{I}_{ih}$ for all $i \in I$ and $h \in \{1, \dots, H\}$ in $\mathcal{O}(N \log^2 N)$ time.

It may be interesting to note that since $\tilde{I}_{ih}$ is based on Ξ_h which contains a fraction $2^{h-1} = K_1/K_h$ of Ξ , it subsamples on average the set $\tilde{I}_{ih}$ of $\lfloor 3K_h \rfloor$ nearest neighbors

of ξ_i in $\Xi \setminus \{\xi_i\}$.

Finally, if $\tilde{I}_i = \cup_{h=1}^H \tilde{I}_{ih}$ is defined, it then clearly follows the intuition expressed earlier. The smallest scale will be represented by the exact $\lfloor 3K_1 \rfloor$ nearest neighbors of ξ_i in $\Xi \setminus \{\xi_i\}$. The larger the scale, the more Ξ_h summarises Ξ approximately. In this way, the neighbors considered in I_{ih} become more and more dispersed, therefore capturing increasingly more of the global properties of HD space.

The $\tilde{I}_i$ sets can then be used to define sparse HD multi-scale similarities that capture both the local and global structure of the HD dataset's structures. For $h \in \{1, \dots, H\}$ and $i, j \in I$, these similarities are given such that

$$\tilde{\sigma}_{ijh} = \begin{cases} \frac{\exp(-\tilde{\pi}_{ih}\delta_{ij}^2/2)}{\sum_{k \in \tilde{I}_i} \exp(-\tilde{\pi}_{ih}\delta_{ik}^2/2)} & \text{if } j \in \tilde{I}_i \\ 0 & \text{otherwise} \end{cases}, \quad (4.1)$$

with

$$\tilde{\pi}_{ih} \text{ s.t. } \log K_1 = - \sum_{j \in \tilde{I}_{ih}} \tilde{\sigma}_{ijh}^\pi \log \tilde{\sigma}_{ijh}^\pi, \quad (4.2)$$

where

$$\tilde{\sigma}_{ijh}^\pi = \begin{cases} \frac{\exp(-\tilde{\pi}_{ih}\delta_{ij}^2/2)}{\sum_{k \in \tilde{I}_{ih}} \exp(-\tilde{\pi}_{ih}\delta_{ik}^2/2)} & \text{if } j \in \tilde{I}_{ih} \\ 0 & \text{otherwise} \end{cases}, \quad (4.3)$$

It here becomes apparent that the precisions $\tilde{\pi}_{ih}$ are fixed in Equation 4.2 using the sets $\tilde{I}_{ih}$. From here sparse multi-scale HD similarities can be defined with $\mathcal{O}(N \log N)$ inputs in $\mathcal{O}(N \log^2 N)$ time. For $i, j \in I$ it becomes

$$\tilde{\sigma}_{ij} = H^{-1} \sum_{h=1}^H \tilde{\sigma}_{ijh}, \tilde{\sigma}_{ij, \text{sym}} = \tilde{\tau}_{ij} = (\tilde{\sigma}_{ij} + \tilde{\sigma}_{ji})/(2N). \quad (4.4)$$

One might think it is wise to define LD similarities in the same way. That is, to match LD similarities to HDs by evaluating them on the same sets $\tilde{I}_i$ for $i \in I$.

Each of these sets could then potentially be extended with $\mathcal{O}(\log N)$ elements to capture local LD embedding properties. However, experiments have shown that minimizing the divergence of the cost function using HD similarities as defined in 4.4 as well as LD similarities using sets as described here, leads to a coordinate crush phenomenon in LD. Indeed, the size of each set I_i for $i \in I$ remains limited to $\mathcal{O}(\log N)$ elements, whether the $\mathcal{O}(\log N)$ elements mentioned earlier are added or not. For this reason it is very likely that $I_k \cap I_l = \emptyset$ for $k, l \in I$. This would then make the points whose coordinates are x_k and x_l and whose similarities are based on I_k and I_l , independent of each other. It would then lead to a lack of repulsive forces between these data, which would therefore tend to overlap.

LD multi-scale similarities can, however, be effectively evaluated by building on the methods used for single-scale formats. This motivates the definition of hybrid cost functions for fast versions of NE multi-scale methods. For multi-scale symmetric SNE and t -SNE, these are given by

$$\tilde{C}_{sym} = - \sum_{i \in I, j \in I \setminus \{i\}} \tilde{\sigma}_{ij, sym} \log s_{ij, sym}, \quad (4.5)$$

and

$$\tilde{C}_t = - \sum_{i \in I, j \in I \setminus \{i\}} \tilde{\tau}_{ij} \log t_{ij}. \quad (4.6)$$

In the equations 4.5 and 4.6, the HD similarities are normalized by the points in I_i , whereas the normalization of the LD similarities involves all the points in I . The correspondence between these two types of similarities reduces the amplitude of the additional terms in the denominators of LD similarities and favours the tendency of LD points to expand.

Now that after having succeeded in evaluating efficiently sparse HD multi-scale similarities, an investment on how to speed up the evaluation of the respective cost functions and gradients of the multi-scale methods can be started.

4.2 Fast Gauss Transform (FGT) for Multi-scale s-SNE (Ms s-SNE)

In this section of the work, one will analyse and try to adapt the Fast Gauss Transform, presented and used in section 3.2.2 for the acceleration of single-scale

symmetric SNE (s-SNE), to the acceleration of its multi-scale version (Ms s-SNE). In order to do this, this part is divided into three pieces. The first sub-section aims not only to identify precisely the differences between the cost function and gradients of the two versions of symmetric SNE; But it also tries to analyze how these differences impact the adaptation of the implementation from the single-scale acceleration to the multi-scale paradigm. The second subsection then describes a dedicated approach to overcome the differences encountered in the first subsection. Finally, the third subsection explains the observations that were made following the implementation.

During this phase of the work, the opportunity was given to use the existing implementation of the accelerated version of multi-scale NE methods with Barnes-Hut [31] as well as the implementation of the FGT adaptation for symmetric SNE [2]. As a first step, one immersed in these implementations and adjusted them to fit this specific work. This not only enabled to get to grips with the subject from a different point of view than that of theory, but also allowed to avoid starting from scratch in order to achieve the desired implementation of multi-scale NE methods using FMMs.

4.2.1 Comparative Analysis of s-SNE and Ms s-SNE for FGT

One of the major differences between multi-scale and single-scale NE variants, as mentioned earlier, is that multi-scale ones take several perplexities into account when calculating HD similarities. However, as these HD similarities are evaluated beforehand, using the method described in the previous section, they have no direct impact on the manner with which the cost function and gradient are evaluated during the gradient-based optimization iterations.

For symmetric SNE nevertheless, another big difference, between its multi-scale and single-scale method, is the form of its LD similarities. In multi-scale these indeed also take into account several perplexities. Here it is important to recall that the cost function of multi-scale symmetric SNE is given by

$$C = - \sum_{i \in I, j \in I \setminus \{i\}} \sigma_{ij, sym} \log s_{ij, sym}, \quad (4.7)$$

where

$$s_{ij, sym} = H^{-1} \sum_{h=1}^H s_{ijh, sym}, \quad (4.8)$$

and

$$s_{ijh, sym} = \frac{\exp(-p_{ih}d_{ij}^2/2)}{\sum_{k \in I, l \in I \setminus \{k\}} \exp(-p_{kh}d_{kl}^2/2)}, s_{iih, sym} = 0. \quad (4.9)$$

The LD precisions are found s.t. $p_{ih} = p_h = K_h^{-2U/P}$ and $K_h = 2^{h-1}K_1$ with $K_1 = 2$ [21].

The introduction of the sum, present in Equation 4.8, into multi-scale s-SNE's cost function (4.7), compared to single-scale s-SNE's one, induces a big change for the rest of the computations.

Indeed in FGT for single-scale SNE, the cost function was transformed such that

$$\begin{aligned} \tilde{C}_{sym\star} &= - \sum_{i \in I, j \in I \setminus \{i\}} \tilde{\sigma}_{ij, sym\star} \log s_{ij, sym\star} \\ &= \sum_{i \in I, j \in I \setminus \{i\}} \tilde{\sigma}_{ij, sym\star} \frac{d_{ij}^2 p_{i\star}}{2} + \log \left(\sum_{k \in I, l \in I \setminus \{k\}} \exp\left(\frac{-p_{k\star} d_{kl}^2}{2}\right) \right) \end{aligned} \quad (4.10)$$

However, the introduction of the new sum of $s_{ij, sym}$ into the cost function no longer allows the use of the quotient property of logarithms to transform it as is done in 4.10.

While the cost function can still be effectively evaluated without this development, it is more tricky for the gradient evaluation. Indeed, one can write the cost function as follows

$$\tilde{C}_{sym} = - \sum_{i \in I, j \in I \setminus \{i\}} \tilde{\sigma}_{ij, sym} \log \left(H^{-1} \sum_{h=1}^H \frac{\exp(-p_{ih}d_{ij}^2/2)}{\sum_{k \in I} S(x_k, p_h)} \right), \quad (4.11)$$

where

$$S(x_i, p_h) = \sum_{j=1}^N \exp\left(-\frac{d_{ij}^2 p_h}{2}\right). \quad (4.12)$$

Equation 4.11 can then effectively be calculated in $\mathcal{O}(HNp^d)$ time, using the sparsity of $\tilde{\sigma}_{ij}$ and FGT to compute $S(X_i, p_h)$ by fixing $q_n = 1$ and $\sigma = 1/\sqrt{p_h}$. The gradient that results from the non-transformed cost function (4.7), however, cannot efficiently be evaluated in its basic form and requires a more thorough mathematical transformation to be adapted to FGT.

4.2.2 Implementation Choices - Hybrid s-SNE

In order to avoid the obstacle of having to find the appropriate multi-scale s-SNE gradient transformation described earlier, it was decided to first implement a hybrid version between single- and multi-scale s-SNE.

As mentioned in chapter two, SNE like methods are difficult to optimize. Therefore, multi-scale optimisations such as starting the gradient descent with the largest perplexity only and gradually including the smaller ones, are used. In the hybrid single/multi-scale version that has been implemented, it has also been decided to use this strategy. However, it was decided to apply it to minimize a cost function using HD similarities, which are multi-scale, while using LD similarities, which are single-scale. That is, during the optimization, the cost function begins to be minimised using the similarities of the highest perplexity on both sides. Nevertheless, when a new perplexity is introduced into the HD similarities, the LD similarities remain single-scale but move on to the next smaller perplexity.

This method avoided the introduction of a sum in the logarithm of the cost function, as the LD similarities remained single-scale. It therefore made possible speeding up a method based on a compromise between single and multi-scale, while retaining the accelerated evaluation methods developed with FGT for single-scale s-SNE.

4.2.3 Experimental Observations

Following the implementation of the method as described above, the algorithm was executed on several datasets. However, several irregularities were quickly observed. The more new scales were introduced into the multi-scale HD similarities, thus progressively reducing the scale of the single-scale LD similarity, the slower the algorithm turned out to be. This outcome contradicts the intended acceleration.

After delving into the code, the reason of this phenomenon became clearer. Indeed, as explained in chapter 3.2.2, at the start of the FGT algorithm, the LD space is discretized so that it lies in a box of dimension $[0, 1]^d$. This box is then subdivided into smaller cells of side-length $\sqrt{2}\sigma r$. Nevertheless, the standard deviation, σ , is directly related to the accuracy p_h . Indeed, $\sigma = 1/\sqrt{p_h}$.

Since the hierarchically growing perplexities are fixed such that $K_h = 2^{h-1}K_1$, the subsequent precisions which are set such that $p_{ih} = p_h = K_h^{-2U/P}$, are exponentially increasing. As a result of this exponentially increasing precision, the standard deviation becomes exponentially smaller. This means that the cell side-length, with which the LD space is subdivided, becomes exponentially smaller and leads to an exponentially greater number of cells in the space. Easily put, the smaller the introduced scales get, the greater the precision of the LD similarities become, and the more the algorithm has to iterate over a large number of cells. To give an idea of magnitude, for the *sklearn* Digits dataset containing 1797 records, 50 000 cells have already to be iterated over at the fifth introduced scale out of ten. Logically seen, this seems to be absurd when taking into account the amount of data.

Certainly, this problem is thus directly related to the way the space is subdivided into cells in FGT. In the single-scale scheme, nevertheless, it does not cause complications. Indeed, as in the single-scale LD similarities of s-SNE the precisions are fixed such that $p_{i*} = 1$, thus permitting to set $\sigma = 1$ in FGT, the number of cells in space always remains constant. For the multi-scale scheme, however, this subdivision method is not suitable.

One might think that only iterating on cells containing data items could get around this problem. However, another issue arose when this solution was considered. Indeed, as explained in section 3.2.2, FGT approximates the interactions between points as soon as a cell contains more data than a fixed threshold. If this is not the case, FGT calculates the exact interactions of the points of that cell, thus with a time complexity of $\mathcal{O}(N_{cell}^2)$ with N_{cell} being the number of points in the concerned cell. However, when dividing up the space with such a large number of cells, these inevitably tend to contain fewer points. This means that FGT has to calculate the exact interactions for almost all points when space is divided into too many cells, which then results in a time complexity of $\mathcal{O}(N^2)$. There is thus no extra acceleration compared to the un-accelerated version. For the Digits dataset mentioned above, this phenomenon has been observed starting with the introduction of the third perplexity. From that point onward, the majority of points found themselves alone in their cell.

One might also believe that it could be possible to change the way in which space is subdivided into cells by possibly trying to implement a data-driven method. However, doing so would no longer allow to assert that the Taylor series used in FGT converge rapidly. A change of subdivision should then automatically lead to a more profound modification of the way of approximating interactions in FGT. However, such changes have been made in the improved version of FGT, a subject

that will be mentioned once again in the next chapter.

4.3 Particle-Mesh (PM) for Multi-scale t -SNE (Ms t -SNE)

The aim of this section is to analyse the Particle-Mesh method, which uses Fast Fourier Transforms to accelerate single-scale t -SNE and that is presented in section 3.1.3. Additionally it is then tried to adapt it to the multi-scale version of the NE algorithm (Ms t -SNE). Similar to the previous section, this part is divided into two parts. The first sub-section namely analyses the differences that may arise when applying this acceleration method to the multi-scale scheme, compared to its application to the single-scale method. Next, the second sub-section aims to explain and present the obtained results.

Similar to the previous section, it was made possible to use an existing implementation of the acceleration of NE multi-scale methods using Barnes-Hut [31]. In addition, an implementation of the PM t -SNE method presented in section 3.2.3 was also used [1]. As before, these have enabled a different perspective to be taken on the subject than that of theory, and have made it possible to avoid having to start the desired implementation from scratch.

4.3.1 Comparative Analysis of t -SNE and Ms t -SNE for PM

As already mentioned, one of the major differences between the single-scale and multi-scale variants of the NE methods are the HD similarities. In multi-scale, these take several different perplexities into account. However, given that the evaluation of these perplexities (section 3.1.3) is done prior to the evaluations of the cost function and the gradient, which take place during the gradient descent iterations, they do not have a direct impact on the way in which these are evaluated.

Opposite to previous findings with s-SNE, there are no other major differences between the single- and multi-scale versions of t -SNE. Indeed, multi-scale t -SNE's cost function is given by

$$\tilde{C}_t = \sum_{i \in I, j \in I \setminus \{i\}} \tilde{\tau}_{ij} \log(1/t_{ij}), \quad (4.13)$$

where

$$\tilde{\tau}_{ij} = \frac{\tilde{\sigma}_{ij} + \tilde{\sigma}_{ji}}{2N}, \tilde{\sigma}_{ij} = H^{-1} \sum_{h=1}^H \tilde{\sigma}_{ijh}. \quad (4.14)$$

and

$$t_{ij} = \frac{(1 + d_{ij}^2)^{-1}}{\sum_{k \in I, l \in I \setminus \{k\}} (1 + d_{kl}^2)^{-1}}, t_{ii} = 0 \quad (4.15)$$

Evidently, the LD similarities of multi-scale t -SNE are exactly the same as those of its single-scale version. In fact, unlike the Gaussian distribution used for SNE and s-SNE, the t-Student distribution is not influenced by a parameter that depends on perplexity. Since the HD similarities are evaluated a priori and the LD similarities remain unchanged, the way of evaluating the cost function and the gradient in the optimisation iterations can remain the same. This observation also explains why t -SNE is often easier to optimise than variants based on SNE.

The desired implementation of multi-scale PM t -SNE, using FFTs, can therefore be obtained in exactly the same way as its single-scale version (section 3.2.3) with the exception of its HD similarities, which must be calculated as shown in section 4.1.

4.3.2 Experimental Results

In this section, the implementation of Ms t -SNE based on FFTs (PM Ms t -SNE) is tested on three different datasets. Firstly it is assessed on the small sklearn **Digits** database containing 1797 tiny 8 x 8 images of ten handwritten digits, thus 64 dimensions. Secondly it is evaluated on the medium-sized **MINST** database containing 70,000 32 x 32 greyscale images of also 10 handwritten digits. Finally, it is tested on the large **SVHN** database composed of 630,420 real-world 32 x 32 color images obtained from google-street. Each are labeled as single-digits. For the **MINST** and **SVHN** database, CNN is used to extract respectively a 50 and 64D embeddings of them.

The performance of the implemented algorithm is tested on one side considering the DR quality of its LD embeddings and on the other side considering the speed of the algorithm. The recorded speed nevertheless has to be taken cautiously as the realised implementation has yet not been fully optimised in terms of coding methods.

Concerning the quality of an embedding, it is generally linked to the preservation

of its HD neighbourhoods in LD space. The method used to quantify this quality is described in the next paragraph.

Let's define v_i^K and n_i^K as the K nearest neighbours of ξ_i and x_i in the HD and LD space respectively. The normalised mean agreement between these two K -length arrays can be derived such that

$$Q_{NX}(K) = \frac{1}{N} \sum_{i \in I} \frac{\|v_i^K \cap n_i^K\|}{K} \in [0, 1]. \quad (4.16)$$

Knowing that a random LD embedding would yield $\mathbb{E}[Q_{NX}(K)] = k/(N - 1)$, one can then define

$$R_{NX} = \frac{(N - 1)Q_{NX}(K) - K}{N - 1 - K}. \quad (4.17)$$

This measure allows to compare $Q_{NX}(K)$ for neighbourhoods of different sizes. Given that the predominant neighbourhoods are mainly close ones, we can represent R_{NX} with a log-scale according to K . What will be of particular interest is then its resulting area under the curve (AUC), which can be obtained as follows

$$AUC = \left(\sum_{K=1}^{N-2} K^{-1} \right)^{-1} \cdot \sum_{K=1}^{N-2} \frac{R_{NX}(K)}{K} \in [-1, 1], \quad (4.18)$$

and which will grow related to DR quality. This method makes it possible to calculate the DR quality with a time complexity of $\mathcal{O}(N^2 \log N)$ while taking into account all the scales, focusing on the smallest ones.

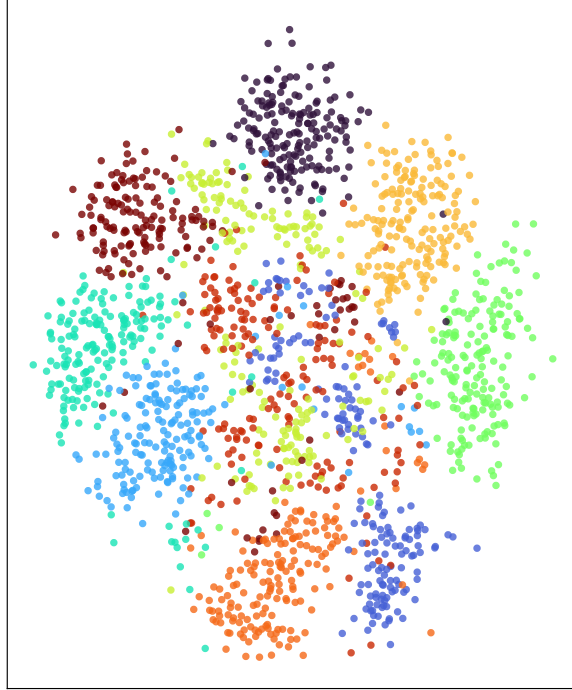
In order to compare the multi-scale method using Fast Fourier Transforms (PM Ms t -SNE), it was tested in parallel with its single-scale variant as well as the Barns-Hut accelerated multi-scale version of t -SNE. This enabled to compare the DR quality as-well as the speed of PM Ms t -SNE to its similarly accelerated but single-scale version and to another multi-scale approach but accelerated differently. All three methods are optimised using the limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) algorithm and are given an initial LD embedding corresponding to the 2D PCA of the used database at the beginning of the optimisations.

For computational reasons, the algorithms is tested only with standard hyper-parameters. It is therefore decided to apply the algorithms using Fast Fourier Transforms, PM t -SNE and PM Ms t -SNE, with a grid of 256^2 cells on the datasets

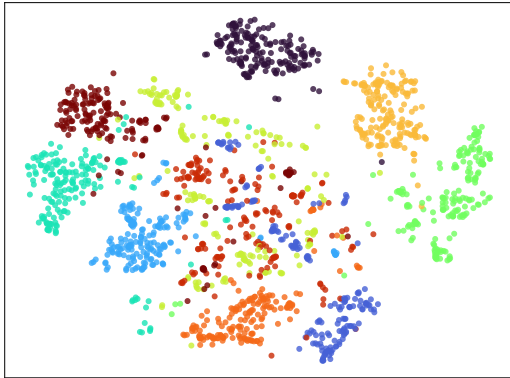
Digits and **MINST**. This exaggerated gridding is chosen in an attempt to limit the impact on the quality of the results of these two datasets which, given their size, are less suited to the two methods. For the **SVHN** dataset, a grid of 512^2 cells is used. The number of optimization iterations for the single-scale PM t -SNE algorithm is limited to 700 iterations. This number corresponds to the number of iterations used to test this algorithm in the work in which it was developed. Finally, the number of optimisation iterations for the two multi-scale algorithms, BH Ms t -SNE and PM Ms t -SNE, is limited to 50 per new scale introduced. Also for computational reasons, a reduced quality criterion is evaluated for the datasets **MINST** and **SVHN**, taking into account neighbourhoods of up to 10,000 and 5,000 points respectively. This is denoted by $\overline{AUC}_E$, where E represents the size of neighborhoods taken up to into account.

For all the reasons mentioned above, i.e. the use of standard hyper-parameters and the evaluation of reduced quality criteria, it is important to note that the obtained results are not representative. Indeed, in order to obtain the real performances of the algorithm, many more tests with different parameters and more datasets would have to be carried out. However, the results obtained may indicate a trend that needs to be confirmed.

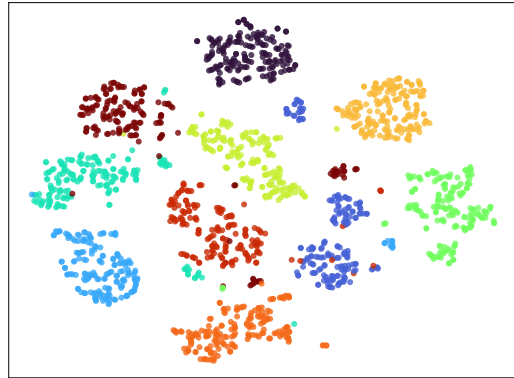
The obtained results for each dataset can respectively be observed in Figures 4.2 , 4.3 and 4.4 below. These results are then discussed in the next chapter.



(a) PM t -SNE; $AUC = 31,39\%$;
Time = 3,24 sec

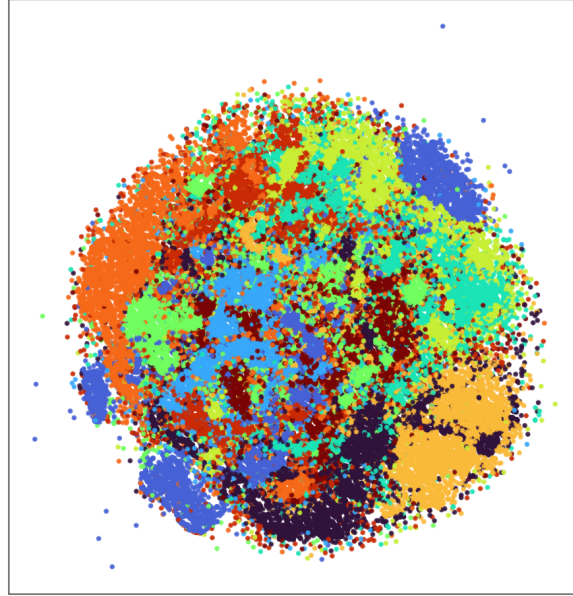


(b) PM Ms t -SNE; $AUC = 41,11\%$;
Time = 12,18 sec

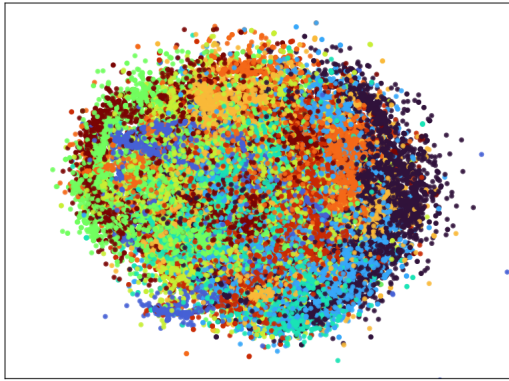


(c) BH Ms t -SNE; $AUC = 55,45\%$;
Time = 5,46 sec

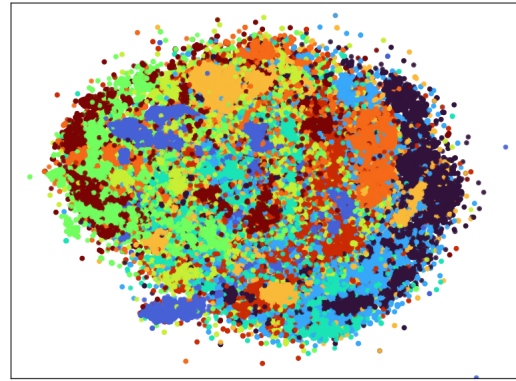
Figure 4.2: 2D Embedding of the Digits dataset



(a) PM t -SNE; $\overline{AUC}_{10000} = 13,03\%$;
Time = 60,68 sec

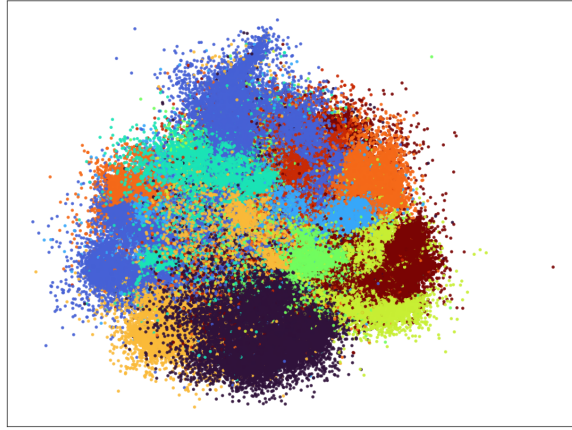


(b) PM Ms t -SNE; $\overline{AUC}_{10000} = 16,22\%$;
Time = 1670,07 sec

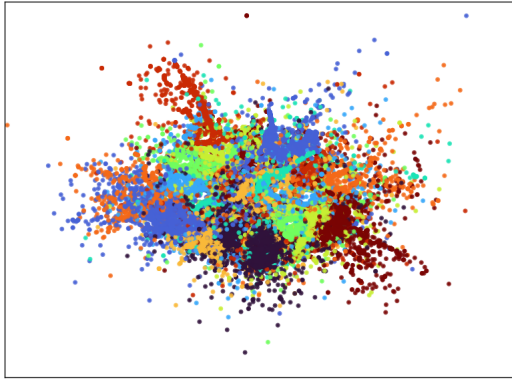


(c) BH Ms t -SNE; $\overline{AUC}_{10000} = 26,59\%$;
Time = 3626,4 sec

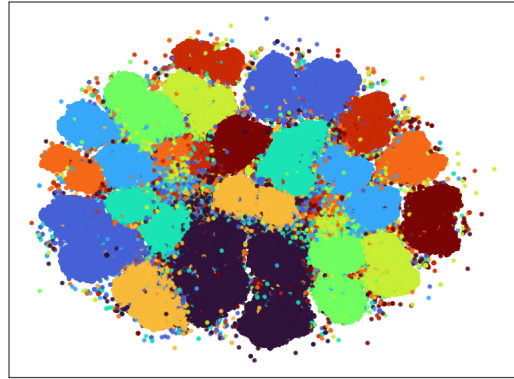
Figure 4.3: 2D Embedding of the MINST dataset



(a) PM t -SNE ; $AUC = 10,74\%$
Time = 922,18 sec



(b) PM Ms t -SNE ; $AUC = 12,37\%$
Time = 9986,37 sec



(c) BH Ms t -SNE ; $AUC = 29.08\%$
Time = 35406,22 sec

Figure 4.4: 2D Embedding of the SVHN dataset

Chapter 5

Discussions and Further Works

This chapter delves into the findings from the preceding chapter and proposes subsequent areas of exploration. In order to be able to discuss the obtained results, Table 5.1, gathering all the investigated methods as well as their complexities, is given as a reminder.

DR method	Time complexity	
	HD similarities computation at initialization	Iterations in gradient-based optimization
SNE, t -SNE	$\mathcal{O}(N^2)$	$\mathcal{O}(N^2)$
FGT SNE	$\mathcal{O}(N \log N)$	$\mathcal{O}(N)$
PM t -SNE	$\mathcal{O}(N \log N)$	$\mathcal{O}(M \log M)$
Multi-scale SNE	$\mathcal{O}(N^2 \log N)$	$\mathcal{O}(N^2 \log N)$
Multi-scale t -SNE	$\mathcal{O}(N^2 \log N)$	$\mathcal{O}(N^2)$
PM Multi-scale t -SNE	$\mathcal{O}(N \log^2 N)$	$\mathcal{O}(M \log M)$

Table 5.1: Time complexities of the NE schemes presented in Chapter 2-4

The first section of this chapter investigates the results obtained in section 4.2 concerning the use of FGT for the multi-scale scheme. Next, the second section then discusses the results obtained by the adaptation of the PM algorithm for multi-scale t -SNE (section 4.3).

5.1 Fast Gauss Transform for Multi-scale s-SNE

In section 4.2 it was shown that classical FGT, as it stands, is not suitable for the multi-scale scheme. Indeed, the division method used to separate the space into cells is not adequate for large scales. Using this method consequently slows down the evaluation of interactions rather than speeding them up. However, despite this obstacle, the method should not be totally rejected. In fact, the limitation encountered so far is restricted solely to this separation method and the impact it has on interaction approximations. Nevertheless, the linear time complexity, $\mathcal{O}(N)$ that this method proposes, is a strong incentive to try and get around this limitation.

One way of avoiding the problem would perhaps be to try to adapt the improved version of FGT instead of the classic one, as attempted in this work. As mentioned earlier, this method uses a data-driven space separation method based on clustering and could therefore avoid the aforementioned limitation. Although IFGT requires the fine-tuning of many parameters, the linear complexity it could offer represents a significant acceleration compared to current methods, which is too important to be ignored. It therefore seems to be a good idea to further develop the subject in this direction.

It is also important to note that this work is limited to the use of the series expansion proposed by Greendard et al. [38], which is only suitable for adaptation to SNE as it is related to Gaussian distributions. In their adaptation of FGT to SNE, Vladymyrov et al. [2] mention that a reasoning equivalent to that done for SNE with FGT could be performed for t -SNE using an adapted series expansion or even an independent kernel Fast Multipole method. Given the significant superiority of t -SNE over SNE, this approach would also seem to be worth pursuing.

5.2 Particle-Mesh for Multi-scale t -SNE

In truth, the results obtained by the implementation of Particle-Mesh multi-scale t -SNE, displayed and compared with Particle-Mesh single-scale t -SNE and Barnes-Hut multi-scale t -SNE in section 4.3, are very informative. Indeed they clearly indicate that PM Ms t -SNE seems more efficient in terms of DR quality, than its single-scale version, but less than its alternative that uses Barnes-Hut.

However, as mentioned above, these results are to be considered with a certain amount of hindsight and are indicative rather than definitive. Indeed, they solely apply for the used standard hyper-parameters and datasets. In order to validate these results, further tests should be carried out, exploring a wider range of

parameters and datasets.

One can clearly observe that for the two larger datasets **MINST** and **SVHN**, PM Ms t -SNE is much faster than BH Ms t -SNE. This is due to the grid-mesh M , which in this case, contrary to the configuration of the small datasets, induces $M \ll N$. Nevertheless it is also important to note that at present, the algorithm has been tested on an implementation version that does not use Cython’s total coding optimisation capacities. The obtained results hence remain solely indicative. The temporal complexities are thus mostly theoretically proven. It therefore seems important to optimize the method so that it can also be correctly tested in terms of practical speed.

Nevertheless, given the promising results and the time complexity $\mathcal{O}(M \log M)$ obtained with $M \ll N$ for very large datasets, it seems very interesting to further investigate this method. Indeed, if the current results are confirmed, given that for large datasets the complexity obtained is lower than that of the Barns-Hut-based alternative, $\mathcal{O}(N \log N)$, the method in its current state could already represent a plausible alternative to it. It would be a lower-quality alternative, but faster and therefore potentially more affordable.

Additionally, it is also very interesting to note that the PM Ms t -SNE method that has been developed uses the first version of Particle-Mesh in its current state. More sophisticated versions of Particle-Mesh have already been developed and could be adapted in the same way for t -SNE. We’re thinking notably of versions of particle mesh using nested grids, where the mesh size depends on the density of the data. This could potentially improve results and make the algorithm as efficient in terms of DR quality as its Barns-Hut version.

Chapter 6

Conclusion

All throughout this work, the use of different Fast Multipole Methods in relation to NE methods is studied. This is followed by an attempt to incorporate these into NE multi-scale method schemes in order to speed them up.

Contrary to the already existing accelerations of multi-scale NE methods that use Barnes-Hut algorithm, the aim of the hereby project is thus to use and focus on Fast Multipole methods. Indeed, these are able to exert a lower time complexity than Barnes-Hut and could therefore further improve the potential acceleration.

It was first shown that adapting Fast Multipole Methods, as done by Vladymyrov et al. for single-scale schemes in [2] using FGT, is unfortunately not possible for multi-scale versions. However, a number of workarounds have been identified and proposed to bypass the current limitations. Additionally although this method has not yet worked, it is still the one that potentially offers the most significant acceleration. In spite of the current results, further research is therefore strongly encouraged on it.

Further, it was then proven that the adaptation for single-scale t -SNE of Particle-Mesh using Fast Fourier Transforms in [1], could also work in the multi-scale framework. Although this method already appears to be faster than the accelerated version using Barnes-Hut, it nonetheless seems to score significantly lower than its counterpart in terms of DR quality. However, several approaches, like the use of more advanced versions of Particle-Mesh, have been identified to remedy this issue.

It can therefore be concluded that Fast Multipole Methods do indeed appear to be a viable and faster alternative to the current acceleration of NE multi-scale methods with Barnes-Hut. However, there is still a great deal of work to be done,

firstly to confirm the results obtained in this work, and secondly to try and improve them. Nevertheless, a variety of very promising developments that could help make this a reality have been suggested.

Finally it also seems that, given the number of Fast Multipole Method alternatives and the varying difficulty of parameterizing and adjusting them for the desired application, the use of FMMs is more complex than that of Barnes-Hut. However, bearing in mind that datasets are getting increasingly larger, it doesn't seem unreasonable to put further work into untangling all the mentioned FMM aspects in order to obtain the accelerations they could perform.

Bibliography

- [1] Valentin Delchevalerie, Alexandre Mayer, Adrien Bibal, and Benoît Frénay. Accelerating t -sne using fast fourier transforms and the particle-mesh algorithm from physics. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2021.
- [2] Max Vladymyrov and Miguel Carreira-Perpinan. Linear-time training of nonlinear low-dimensional embeddings. In *Artificial Intelligence and Statistics*, pages 968–977. PMLR, 2014.
- [3] Arne Holst. Amount of data created, consumed, and stored 2010-2025. *Technology & Telecommunications Retrieved*, pages 06–29, 2021.
- [4] John A Lee, Michel Verleysen, et al. *Nonlinear dimensionality reduction*. Springer Science & Business Media, 2007.
- [5] Xuelong Li, Han Zhang, Rui Zhang, Yun Liu, and Feiping Nie. Generalized uncorrelated regression with adaptive graph for unsupervised feature selection. *IEEE Transactions on Neural Networks and Learning systems*, 30(5):1587–1595, 2018.
- [6] Rui Zhang, Feiping Nie, Yunhai Wang, and Xuelong Li. Unsupervised feature selection via adaptive multimeasure fusion. *IEEE Transactions on Neural Networks and Learning systems*, 30(9):2886–2892, 2019.
- [7] Ingwer Borg and Patrick JF Groenen. *Modern multidimensional scaling: Theory and applications*. Springer Science & Business Media, 2005.
- [8] IT Jolliffe. *Principal Component Analysis*, chapter Principal component analysis and factor analysis, pages 115–128. Springer, 1986.
- [9] John W Sammon. A nonlinear mapping for data structure analysis. *IEEE Transactions on Computers*, 100(5):401–409, 1969.

- [10] Joshua B Tenenbaum, Vin de Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.
- [11] John A Lee and Michel Verleysen. Quality assessment of dimensionality reduction: Rank-based criteria. *Neurocomputing*, 72(7):1431–1443, 2009.
- [12] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008.
- [13] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural computation*, 10(5):1299–1319, 1998.
- [14] Damien François, Vincent Wertz, and Michel Verleysen. The concentration of fractional distances. *IEEE Transactions on Knowledge and Data Engineering*, 19(7):873–886, 2007.
- [15] John A Lee and Michel Verleysen. Shift-invariant similarities circumvent distance concentration in stochastic neighbor embedding and variants. *Procedia Computer Science*, 4:538–547, 2011.
- [16] Teuvo Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480, 1990.
- [17] Geoffrey E Hinton and Sam Roweis. Stochastic neighbor embedding. *NIPS*, 15:833–840, 2002.
- [18] Jarkko Venna, Jaakko Peltonen, Kristian Nybo, Helena Aidos, and Samuel Kaski. Information retrieval perspective to nonlinear dimensionality reduction for data visualization. *Journal of Machine Learning Research*, 11(13):451–490, 2010.
- [19] Zhirong Yang, Irwin King, Zenglin Xu, and Erkki Oja. Heavy-tailed symmetric stochastic neighbor embedding. *NIPS*, pages 2169–2177, 2009.
- [20] Cyril de Bodt, Dounia Mulders, Daniel López Sánchez, Michel Verleysen, and John A Lee. Class-aware t-sne: cat-sne. In *ESANN*, pages 409–414, 2019.
- [21] John A Lee, Diego H Peluffo-Ordóñez, and Michel Verleysen. Multi-scale similarities in stochastic neighbour embedding: Reducing dimensionality while preserving both local and global structure. *Neurocomputing*, 169:246–261, 2015.

- [22] Cyril De Bodt, Dounia Mulders, Michel Verleysen, and John A Lee. Perplexity-free t-sne and twice student tt-sne. In *ESANN*, pages 123–128, 2018.
- [23] Jaakko Peltonen and Konstantinos Georgatzis. Efficient optimization for data visualization as an information retrieval task. In *MSLP*, pages 1–6. IEEE, 2012.
- [24] Laurens Van Der Maaten. Learning a parametric embedding by preserving local structure. In *Artificial intelligence and statistics*, pages 384–391. PMLR, 2009.
- [25] F Crecchi, C De Bodt, M Verleysen, JA Lee, D Bacciu, et al. Perplexity-free parametric t-sne. In *ESANN*, pages 387–392, 2020.
- [26] Laurens Van Der Maaten. Accelerating t-sne using tree-based algorithms. *Machine learning research*, 15(1):3221–3245, 2014.
- [27] Zhirong Yang, Jaakko Peltonen, and Samuel Kaski. Scalable optimization of neighbor embedding for visualization. In *IMCL*, pages 127–135. PMLR, 2013.
- [28] Jian Tang, Jingzhou Liu, Ming Zhang, and Qiaozhu Mei. Visualizing large-scale and high-dimensional data. In *Proceedings of the 25th international conference on world wide web*, pages 287–297, 2016.
- [29] Charles Bouveyron, Stéphane Girard, and Cordelia Schmid. High-dimensional data clustering. *Computational Statistics & Data analysis*, 52(1):502–519, 2007.
- [30] Josh Barnes and Piet Hut. A hierarchical o ($n \log n$) force-calculation algorithm. *nature*, 324(6096):446–449, 1986.
- [31] Cyril de Bodt, Dounia Mulders, Michel Verleysen, and John Aldo Lee. Fast multiscale neighbor embedding. *IEEE Transactions on Neural Networks and Learning Systems*, 33(4):1546–1560, 2020.
- [32] Miguel A Carreira-Perpinán. The elastic embedding algorithm for dimensionality reduction. In *ICML*, volume 10, pages 167–174, 2010.
- [33] Max Vladymyrov and Miguel Á Carreira-Perpiñán. Partial-hessian strategies for fast learning of nonlinear embeddings. In *Proceedings of the 29th International Conference on Machine Learning*, pages 987–994, 2012.

- [34] Max Vladymyrov and Miguel Carreira-Perpinan. Entropic affinities: Properties and efficient numerical computation. In *IMCL*, pages 477–485. PMLR, 2013.
- [35] Peter N Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *SODA*, volume 93, pages 311–21, 1993.
- [36] George C Linderman, Manas Rachh, Jeremy G Hoskins, Stefan Steinerberger, and Yuval Kluger. Fast interpolation-based t-sne for improved visualization of single-cell rna-seq data. *Nature methods*, 16(3):243–245, 2019.
- [37] Nicola Pezzotti, Boudewijn PF Lelieveldt, Laurens Van Der Maaten, Thomas Höllt, Elmar Eisemann, and Anna Vilanova. Approximated and user steerable tsne for progressive visual analytics. *IEEE transactions on visualization and computer graphics*, 23(7):1739–1752, 2016.
- [38] Leslie Greengard and Vladimir Rokhlin. A fast algorithm for particle simulations. *Journal of computational physics*, 73(2):325–348, 1987.
- [39] Leslie Greengard and John Strain. The fast gauss transform. *SIAM Journal on Scientific and Statistical Computing*, 12(1):79–94, 1991.
- [40] Vikas Chandrakant Raykar and Ramani Duraiswami. Fast optimal bandwidth selection for kernel density estimation. In *Proceedings of the 2006 SIAM International Conference on Data Mining*, pages 524–528. SIAM, 2006.
- [41] ND FREITAS. Fast krylov methods for n-body learning. *Advances in Neural Information Processing Systems*, 18:251–258, 2006.
- [42] Yang, Duraiswami, and Gumerov. Improved fast gauss transform and efficient kernel density estimation. In *Proceedings ninth IEEE international conference on computer vision*, pages 664–671. IEEE, 2003.

Chapter 7

Appendix

7.1 Appendix A

7.1.1 Shifting from Gaussians to Taylor series

Lets consider a Gaussian s.t.

$$G(t) = qe^{-|t-s_j|^2}/\sigma. \quad (7.1)$$

It has the following the Taylor expansion [38] about t_c :

$$G(t) = \sum_{\beta \geq 0} B_{\beta} \left(\frac{t - t_c}{\sqrt{\sigma}} \right)^{\beta}. \quad (7.2)$$

Furthermore, it's error in truncation after p^d terms is

$$|E_T(p)| = \left| \sum_{\beta \geq p} B_{\beta} \left(\frac{t - t_c}{\sqrt{\sigma}} \right)^{\beta} \right| \leq Kq \left(\frac{1}{p!} \right)^{d/2} \left(\frac{r^{p+1}}{1-r} \right)^d \quad (7.3)$$

7.1.2 Fast Gauss transform (FGT) time complexity development

Lets break down how, after having chosen the cutoff parameters N_F and M_L , FGT evaluates the wanted interactions:

1. If $N_B < N_F$ then B sends out N_B Gaussians to C
2. If $N_B \geq N_F$ then B sends out a Hermite expansions with its N_B Gaussians accumulated in it to C
3. If $M_C < M_L$ then C evaluates directly all fields sent to it on its M_C points
4. If $M_C \geq M_L$ then C converts all fields sent to it in Taylor series, accumulates the coefficients and only then evaluates the Taylor series.

The work involved in this algorithm can therefore be broken [38] down in the following way

1.

$$\sum_{N_B \geq N_F} \mathcal{O}(p^d N_B)$$

to evaluate the N_B Hermite expansions coefficients;

2.

$$+ \sum_{M_C < M_L} \sum_{N_B < N_F} \mathcal{O}(N_B M_C)$$

to evaluate the N_B Gaussians on the M_C points;

3.

$$+ \sum_{M_C < M_L} \sum_{N_B \geq N_F} \mathcal{O}(p^d M_C)$$

to evaluate Hermite expansions on the M_C points;

4.

$$+ \sum_{M_C \geq M_L} \sum_{N_B < N_F} \mathcal{O}(p^d N_B)$$

to transform the N_B Gaussians into Taylor series;

5.

$$+ \sum_{M_C \geq M_L} \sum_{N_B \geq N_F} \mathcal{O}(dp^{d+1})$$

to transform the Hermite series into a Taylor series;

6.

$$+ \sum_{M_C \geq NM_L} \mathcal{O}(p^d M_C)$$

to evaluate the final Taylor series.

The total work then has a time complexity of the form

$$\mathcal{O}(p^d N) + \mathcal{O}(dp^{d+1}(2n+1)^d \min((r\sqrt{2\sigma})^{-d/2}, M)) + \mathcal{O}(p^d M)$$

which is linear in N and M with another term depending on the precision.

7.2 Appendix B

7.2.1 Computation of the normalisation factor Z in PM t -SNE

As mentioned in section 3.2.3, the normalization term $Z = \sum_{k \in I, l \in I \setminus \{k\}} (1 + \|x_k - x_l\|^2)^{-1}$ can be computed using the expression of the previously computed potential $\phi(r)$. One can indeed show that

$$\begin{aligned} Z &= \sum_{k \in I, l \in I \setminus \{k\}} \frac{1}{1 + \|x_k - x_l\|^2} \\ &= \sum_k \sum_{g=1}^M \rho_g \frac{1}{1 + \|x_k - x_g\|^2} \\ &= \sum_k \iint \rho(r') \frac{1}{1 + \|x_k - r'\|^2} d^2 r' \\ &= \sum_k \phi(x_k) \end{aligned}$$

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | www.uclouvain.be/epl