國立臺灣大學理學院應用數學科學研究所

碩士論文

Institute of Applied Mathematical Sciences

College of Science

National Taiwan University

Master Thesis

使用伽馬散度之推廣 t-分布隨機鄰域嵌入法

Generalized degrees of freedom t-SNE with γ-divergence

邱郁軒

Yu-Xuan Chiou

指導教授：陳素雲博士

Advisor: Su-Yun Huang, Ph.D.

中華民國 109 年 6 月

June, 2020

doi:10.6342/NTU202001485

doi:10.6342/NTU202001485

誌謝

非常感謝我的指導教授 陳素雲博士，在我碩一初期時就收我當指導學生。一開始讓我參與到好幾位老師的博班學生分享機器學習、類神經網路、醫學影像分析等相關研究介紹或討論實作，使我了解一些新穎的技術與認識厲害的學長姐們。後來修習老師的資料科學之統計基礎，認識到許多機器學習方法、神經網絡的理論及其推廣，也在這堂課準備期末報告的過程中也找到自己的論文題目的主題。之後在老師一年多的指導下，終於把這篇碩士論文的核心內容都完成了，在這個過程中經歷了不少挫折卻也學到很多不只是理論的東西，真的很感謝老師長期耐心指導和不吝於當頭棒喝我這個學生的不足之處!

再來感謝兩位學長姐，第一位是森元俊成學長，學長的程式能力好到沒話說，清楚地指教我解決實作方法在套用資料前程式最難的部分，也跟我一起觀察分析後的結果變化；第二位是李易儒學姐，當初在介紹腦部醫學影像資料時就有協助我和學長及同學一起討論實作的地方，感謝後來學姐和學長都願意幫我在口試前做預口試。最後也謝謝口試委員 洪弘老師與 黃子銘老師，特別感謝黃老師還幫我檢查到文稿裡一些小錯誤。在此誠心謝謝以上各位老師、前輩們慷慨地花時間協助我完成此篇論文。

doi:10.6342/NTU202001485

doi:10.6342/NTU202001485

摘要

本論文主要在推廣 t-分布隨機鄰域嵌入法。在此方法的原始論文中 [9]，是使用 t_1 分布來嵌入低維度的空間來當作模型。t_1 分布有相當厚尾的性質，主要是用來減少擁擠問題的效應。然而對於不同的資料，其大小、維度、性質都不盡相同，不見得都需要用到 t_1 分配來建模。因此本論文就把原本 t-SNE 使用的 t_1 分布推廣到 t_ν 分布。為了衡量資料分布與模型分布間的差異，在 t-SNE 的原始論文中 [9] 使用的是 KL-散度。在這個部分中，我們亦作了換使用伽馬散度的推廣，其中 KL-散度是伽馬散度的一個特例。再來推導出使用伽馬散度的 t_ν-SNE 的梯度，用來做可以執行的程式，並應用在兩個實際的例子上。

關鍵字： 伽馬散度，t-SNE，隨機鄰域嵌入法，視覺化方法，非監督式降維

doi:10.6342/NTU202001485

doi:10.6342/NTU202001485

Abstract

This thesis presents an extended version of the t-SNE visualization method. In the original paper [9] of t-SNE, the t_1-distribution was used to embed the data into a low dimensional space. The distribution t_1 has very fat tails that can reduce the effect of the crowding problem. However, data sets may vary in different aspects, such as the data set size, dimensionality, or feature properties, etc. It seems not adequate to use only the t_1 distribution for modeling the low dimensional similarity. Hence, it is natural to extend the degrees of freedom in t-SNE to general t_ν. To measure the discrepancy between data distribution and model distribution, the original paper [9] used KL-divergence. In this work, we also make an extension by using gamma-divergence, which includes KL-divergence as a special case. The gradient of minimum gamma-divergence t_ν-SNE is derived and used in the implementation algorithm. Two numerical examples are presented.

Keywords: gamma-divergence, t-SNE (stochastic neighborhood embedding), unsupervised dimension reduction, visualization.

doi:10.6342/NTU202001485

doi:10.6342/NTU202001485

Contents

doi:10.6342/NTU202001485

doi:10.6342/NTU202001485

List of Figures

doi:10.6342/NTU202001485

doi:10.6342/NTU202001485

Chapter 1

Introduction

t-SNE (t-Distributed Stochastic Neighbor Embedding) is a nonlinear unsupervised dimension reduction method, also an improvement of an earlier SNE algorithm of Hinton and Roweis (2002) [6]. It was proposed by van der Maaten and Hinton (2008) [9]. They used t-distribution to embed the data to a low-dimensional space and to compute the similarity between two points. The KL-divergence was used to measure the discrepancy between the data distribution and the model distribution. The low-dimensional embedding was solved by minimum KL-divergence estimation.

In the conclusion of van der Maaten and Hinton (2008), they gave a discussion on the degrees of freedom of the t-distribution. It can be helpful for dimension reduction, if the low-dimensional space has many dimensions. As the degrees of freedom increase, t-distribution will get closer to the Gaussian. A more Gaussian-like distribution can be useful for big datasets.

Furthermore, for the purpose of reducing the influence of outliers, we can use more robust divergence measure than the widely used KL-divergence [1, 7]. The main reason is that KL-divergence is more sensitive to outliers than gamma-divergence. The latter includes the KL-divergence as a special case.

doi:10.6342/NTU202001485

doi:10.6342/NTU202001485

Chapter 2

Literature Review

In this chapter, we give a brief review for t-SNE and gamma-divergence.

2.1 SNE and t-SNE

In SNE, the similarity of high-dimensional data points x_j to x_i is a conditional probability, $p_{j|i}$, which is measured by a Gaussian probability density centered at x_i with Euclidean distance between data points. Precisely, the conditional probability $p_{j|i}$ is given by

$$p_{j|i} = \frac{\exp(- \parallel x_i - x_j \parallel^2 / 2\sigma_i^2)}{\sum_{k \neq i} \exp(- \parallel x_i - x_k \parallel^2 / 2\sigma_i^2)}, \tag{2.1}$$

where σ_i is the Gaussian standard deviation and its value is determined by *Perp*, the *perplexity*. Set $p_{i|i} = 0$. The perplexity of the distribution $P_i = \{p_{j|i}\}_{j=1}^n$ is defined as

$$Perp(P_i) = 2^{H(P_i)}, \text{ where } H(P_i) = -\sum_j p_{j|i} \log_2 p_{j|i}.$$

The perplexity can be interpreted as the effective number of local neighbors, and is usually chosen by hand between 5 and 50 [9]. Let $y_i \in \mathbb{R}^d$ denote embedded points in the low-dimensional space. Here the dimensionality d is often set to $d = 2$, or 3. The similarity between points y_i and y_j, denoted by $q_{j|i}$, is based on a similar conditional probability

doi:10.6342/NTU202001485

model and is given by

$$q_{j|i} = \frac{\exp(- \parallel y_i - y_j \parallel^2)}{\sum_{k \neq i} \exp(- \parallel y_i - y_k \parallel^2)}. \tag{2.2}$$

Similarly, set $q_{i|i} = 0$. Consider the cost function

$$C = \sum_i KL(P_i \| Q_i) = \sum_{i,j} p_{j|i} \log \frac{p_{j|i}}{q_{j|i}}, \tag{2.3}$$

where $Q_i = \{q_{j|i}\}_{j=1}^n$. To estimate the low-dimensional embedding $\{y_i \in \mathbb{R}^d\}_{i=1}^n$, we seek the minimum KL-divergence estimation via gradient descent. The gradient is given by

$$\frac{\partial C}{\partial y_i} = 2 \sum_j (p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j})(y_i - y_j). \tag{2.4}$$

The optimization problem of the original SNE is difficult to solve due to the *curse of intrinsic dimensionality* problem. Later, t-SNE was developed as an alternative to SNE. There are two major differences from SNE. The first one is that t-SNE has used symmetrized conditional probabilities and with some adjustment about the weight to every data point x_i that makes them all have enough contribution to the cost function. The probabilities are set as follows.

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}. \tag{2.5}$$

The second difference is that t-SNE has used the Student-t distribution with 1-degree of freedom to replace the Gaussian distribution for low-dimensional embedding. The similarities, or say the joint probabilities q_{ij}, in the model space are defined to be

$$q_{ij} = \frac{(1+ \parallel y_i - y_j \parallel^2)^{-1}}{\sum\limits_{k \neq l} (1+ \parallel y_k - y_l \parallel^2)^{-1}}. \tag{2.6}$$

Use the same KL-divergence-based cost function. The gradient of t-SNE is given by

$$\frac{\partial C}{\partial y_i} = 4 \sum_j \left(1+ \parallel y_i - y_j \parallel^2\right)^{-1} (p_{ij} - q_{ij})(y_i - y_j). \tag{2.7}$$

doi:10.6342/NTU202001485

From now on we will use t_1-SNE to reflect its degree of freedom is one.

2.2 The gamma-divergence

Fujisawa and Eguchi [5] proposed a robust estimation method by gamma-divergence. This divergence also measures the discrepancy between data distribution and the corresponding model distribution, but it can reduce the influence of outliers better than KL-divergence, and is a generalization of KL-divergence.

Definition 2.2.1 (Gamma norm)**.** *For $\gamma > 0$, p is a probability distribution function, then the gamma norm is defined as*

$$\Phi_\gamma(p) = \| p \|_{\gamma+1} = \left(\int p(x)^{\gamma+1} dx \right)^{\frac{1}{\gamma+1}} := \| p \| . \tag{2.8}$$

Definition 2.2.2 (Gamma divergence [2])**.** *Let p, q be both probability distribution functions, then the gamma-divergence is*

$$D_\gamma(p,q) = \frac{1}{\gamma(\gamma+1)} \left\{ \Phi_\gamma(p) - \Phi_\gamma(q) - \langle \nabla\Phi_\gamma(q), p - q \rangle \right\} . \tag{2.9}$$

We expressing the inner product in gamma-divergence and simplify it:

$$\begin{aligned} D_\gamma(p,q) &= \frac{1}{\gamma(\gamma+1)} \left(\| p \| - \| q \| - \int \frac{1}{\gamma+1} \| q \|^{-\gamma} (\gamma+1) q^\gamma (p - q) \right) \\ &= \frac{1}{\gamma(\gamma+1)} \left(\| p \| - \| q \| - \int \left(\frac{q}{\| q \|} \right)^\gamma p + \int \frac{q^{\gamma+1}}{\| q \|^\gamma} \right) \\ &= \frac{1}{\gamma(\gamma+1)} \left(\| p \| - \int \left(\frac{q}{\| q \|} \right)^\gamma p \right) . \end{aligned}$$

From the definition, we can see that when $\gamma \to 0^+$, the gamma-divergence will converge to the KL-divergence.

For minimizing the divergence $D_\gamma(p,q)$, or say minimizing the cost function, since p is fixed, we can turn this minimization problem into a maximization of the following

doi:10.6342/NTU202001485

cross-entropy:

$$C_\gamma = \frac{1}{\gamma(\gamma+1)} \int \left(\frac{q}{\parallel q \parallel} \right)^\gamma p.$$

For the case of t_1-SNE, C_γ takes the following form:

$$C_\gamma = \frac{1}{\gamma(\gamma+1)} \sum_{l \neq k} \left(\frac{q_{lk}}{\parallel q \parallel} \right)^\gamma p_{lk}, \tag{2.10}$$

where $\|q\|$ is given by

$$\|q\| = \|q\|_{\gamma+1} = \left\{ \sum_{l=1, l\neq k}^{n} \sum_{k=1}^{n} q_{lk}^{\gamma+1} \right\}^{1/(\gamma+1)}.$$

doi:10.6342/NTU202001485

Chapter 3

t_ν-SNE with KL-divergence

The main focus of this chapter is to formulate the t_ν-SNE with minimum KL-divergence estimation and to compute the gradient of its corresponding cost function.

3.1 The formulation

Recall that Student's t-distribution has the probability density function given by

$$f(t) = \frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\sqrt{\nu\pi}\,\Gamma\left(\frac{\nu}{2}\right)}\left(1+\frac{t^2}{\nu}\right)^{-\frac{\nu+1}{2}}. \tag{3.1}$$

By applying the t_ν-distribution to the model of the data points, the new affinity function of data i to data j (or data j to data i, or between data i and data j) is

$$q_{ij} = \frac{\left(1+\frac{\|y_i-y_j\|^2}{\nu}\right)^{-\frac{\nu+1}{2}}}{\sum\limits_{k\neq l}\left(1+\frac{\|y_k-y_l\|^2}{\nu}\right)^{-\frac{\nu+1}{2}}}.$$

We can compare to (2.6) and see the similarities and differences.

doi:10.6342/NTU202001485

The cost function computed by using KL-divergence to the data P and the model Q is

$$C = KL(P||Q) = \sum_{i,j} p_{ij} \log \frac{p_{ij}}{q_{ij}} = \sum_{i,j} p_{ij}(\log p_{ij} - \log q_{ij}).$$

Since p_{ij}'s are fixed, so if we want to minimize the cost function, we can just deal with the second term of the above equation, i.e. $\sum\limits_{i,j} -p_{ij} \log q_{ij}$.

3.2 The gradient

Proposition 3.2.1. *The gradient of the cost function of t_ν-SNE with KL-divergence is*

$$\frac{\partial C}{\partial y_i} = \frac{2(\nu+1)}{\nu} \sum_j \left(1 + \frac{\| y_i - y_j \|^2}{\nu}\right)^{-1} (p_{ij} - q_{ij})(y_i - y_j). \tag{3.2}$$

Proof. Define two auxiliary variables:

$$\begin{cases} d_{ij} = \| y_i - y_j \| \\ Z = \sum\limits_{k \neq l} \left(1 + \frac{d_{kl}^2}{\nu}\right)^{-\frac{\nu+1}{2}} \end{cases} \Rightarrow \quad q_{ij} = \frac{\left(1 + \frac{d_{ij}^2}{\nu}\right)^{-\frac{\nu+1}{2}}}{Z}. \tag{3.3}$$

Note if y_i changes, only d_{ij} and d_{ji} changes for all j, and by using norm derivative, we can get $\dfrac{\partial d_{ij}}{\partial y_i} = (y_i - y_j)/d_{ij}$, then with chain rule we have:

$$\frac{\partial C}{\partial y_i} = \sum_j \left(\frac{\partial C}{\partial d_{ij}} + \frac{\partial C}{\partial d_{ji}}\right)(y_i - y_j)/d_{ij} = 2\sum_j \frac{\partial C}{\partial d_{ij}}(y_i - y_j)/d_{ij} \tag{3.4}$$

a rough form of gradient. Then with appropriate assumption $p_{ii} = q_{ii} = 0$, now we

doi:10.6342/NTU202001485

compute $\dfrac{\partial C}{\partial d_{ij}}$:

$$\begin{aligned}
\frac{\partial C}{\partial d_{ij}} &= \frac{\partial \sum\limits_{k\neq l} -p_{kl}\log q_{kl}}{\partial d_{ij}} \\
&= -\sum_{k\neq l} p_{kl}\frac{\partial \log q_{kl}}{\partial d_{ij}} \\
&= -\sum_{k\neq l} p_{kl}\frac{\partial(\log q_{kl}Z - \log Z)}{\partial d_{ij}} \\
&= -\sum_{k\neq l} p_{kl}\left(\frac{1}{q_{kl}Z}\frac{\partial\left(1+\frac{d_{kl}^2}{\nu}\right)^{-\frac{\nu+1}{2}}}{\partial d_{ij}} - \frac{1}{Z}\frac{\partial Z}{\partial d_{ij}}\right).
\end{aligned}$$

Because the term

$$\frac{\partial\left(1+\frac{d_{kl}^2}{\nu}\right)^{-\frac{\nu+1}{2}}}{\partial d_{ij}}$$

is nonzero when $k = i$ and $l = j$, and $\sum\limits_{k\neq l} p_{kl} = 1$, also use representations in (3.3), the above result becomes to

$$\begin{aligned}
\frac{\partial C}{\partial d_{ij}} &= \frac{p_{ij}}{q_{ij}Z}\left[\frac{\nu+1}{2}\left(1+\frac{d_{ij}^2}{\nu}\right)^{-\frac{\nu+3}{2}}\frac{2d_{ij}}{\nu}\right] - \sum_{k\neq l} p_{kl}\left(\frac{1}{Z}\right)\left[\frac{\nu+1}{2}\left(1+\frac{d_{ij}^2}{\nu}\right)^{-\frac{\nu+3}{2}}\frac{2d_{ij}}{\nu}\right] \\
&= \left[\frac{\nu+1}{\nu}p_{ij}\left(1+\frac{d_{ij}^2}{\nu}\right)^{-1} - \frac{\nu+1}{\nu}q_{ij}\left(1+\frac{d_{ij}^2}{\nu}\right)^{-1}\right]d_{ij} \\
&= \frac{\nu+1}{\nu}(p_{ij}-q_{ij})\left(1+\frac{d_{ij}^2}{\nu}\right)^{-1}d_{ij}.
\end{aligned}$$

Therefore, the gradient is

$$\frac{\partial C}{\partial y_i} = (3.4) = \frac{2(\nu+1)}{\nu}\sum_j (p_{ij}-q_{ij})\left(1+\frac{\| y_i - y_j \|^2}{\nu}\right)^{-1}(y_i - y_j).$$

□

doi:10.6342/NTU202001485

doi:10.6342/NTU202001485

Chapter 4

t_1-SNE with gamma-divergence

The main focus of this chapter is to formulate the t_1-SNE with minimum gamma-divergence estimation and to compute the gradient of its corresponding cost function.

4.1 The cost function with gamma-divergence

By replacing the KL-divergence with the gamma-divergence in t-SNE, we have the following cost function (which is the negative cross-entropy in (2.10)):

$$-\frac{1}{\gamma(\gamma+1)}\sum_{k\neq l}\left(\frac{q_{kl}}{\parallel q \parallel}\right)^{\gamma} p_{kl} = -\frac{1}{\gamma(\gamma+1)}\sum_{k\neq l}\frac{q_{kl}^{\gamma}p_{kl}}{\left(\sum\limits_{m\neq n} q_{mn}^{\gamma+1}\right)^{\frac{\gamma}{\gamma+1}}}, \tag{4.1}$$

where $q_{ij} = \dfrac{(1+\parallel y_i - y_j \parallel^2)^{-1}}{\sum\limits_{k\neq l}(1+\parallel y_k - y_l \parallel^2)^{-1}}$.

4.2 The gradient

Proposition 4.2.1. *The cost function of t_1-SNE with gamma-divergence has gradient:*

$$\frac{\partial D_\gamma}{\partial y_i} = \frac{4}{\gamma+1}\sum_j (1+\parallel y_i - y_j \parallel^2)^{-1}\left(\frac{q_{ij}^{\gamma}p_{ij}}{\parallel q \parallel^{\gamma}} - \frac{\sum\limits_{k\neq l} q_{kl}^{\gamma}p_{kl}}{\parallel q \parallel^{2\gamma+1}}q_{ij}^{\gamma+1}\right)(y_i - y_j). \tag{4.2}$$

doi:10.6342/NTU202001485

Proof. Define $d_{ij} = \| y_i - y_j \|$, and known that

$$\frac{\partial d_{ij}}{\partial y_i} = (y_i - y_j)/d_{ij}$$

and define $Z = \sum\limits_{k \neq l} (1 + d_{kl})^{-1}$, then we have:

$$q_{ij} = \frac{(1 + d_{ij}^2)^{-1}}{Z}. \tag{4.3}$$

By using same method in (3.4), the gradient from of C_γ is:

$$\frac{\partial C_\gamma}{\partial y_i} = 2 \sum_j \frac{\partial C_\gamma}{\partial d_{ij}} (y_i - y_j)/d_{ij}. \tag{4.4}$$

Then we compute the partial term $\frac{\partial C_\gamma}{\partial d_{ij}}$ by changing the form of C_γ in d_{ij} first:

$$\begin{aligned}
C_\gamma &= \frac{1}{\gamma(\gamma+1)} \sum_{k \neq l} \frac{q_{kl}^{\gamma} p_{kl}}{\left(\sum\limits_{m \neq n} q_{mn}^{\gamma+1} \right)^{\frac{\gamma}{\gamma+1}}} = \frac{1}{\gamma(\gamma+1)} \left(\sum_{m \neq n} q_{mn}^{\gamma+1} \right)^{-\frac{\gamma}{\gamma+1}} \sum_{k \neq l} q_{kl}^{\gamma} p_{kl} \\
&= \frac{1}{\gamma(\gamma+1)} \left[\sum_{m \neq n} \left(\frac{(1 + d_{mn}^2)^{-1}}{Z} \right)^{\gamma+1} \right]^{-\frac{\gamma}{\gamma+1}} \left[\sum_{k \neq l} \left(\frac{(1 + d_{kl}^2)^{-1}}{Z} \right)^{\gamma} p_{kl} \right] \\
&= \frac{1}{\gamma(\gamma+1)} \left(\sum_{m \neq n} (1 + d_{mn}^2)^{-(\gamma+1)} \right)^{-\frac{\gamma}{\gamma+1}} \left(\sum_{k \neq l} (1 + d_{kl}^2)^{-\gamma} p_{kl} \right).
\end{aligned}$$

doi:10.6342/NTU202001485

By using the similar differential techniques in t_1-SNE with gamma-divergence,

$$\frac{\partial C_\gamma}{\partial d_{ij}} = \frac{1}{\gamma+1}\left(\sum_{m\neq n}(1+d_{mn}^2)^{-(\gamma+1)}\right)^{-\frac{2\gamma+1}{\gamma+1}}(1+d_{ij}^2)^{-\gamma-2}2d_{ij}\left(\sum_{k\neq l}(1+d_{kl}^2)^{-\gamma}p_{kl}\right)$$

$$-\frac{1}{\gamma+1}\left(\sum_{m\neq n}(1+d_{mn}^2)^{-(\gamma+1)}\right)^{-\frac{\gamma}{\gamma+1}}(1+d_{ij}^2)^{-\gamma-1}2d_{ij}p_{ij}$$

$$= \frac{2}{\gamma+1}d_{ij}(1+d_{ij}^2)^{-\gamma-1}\left(\sum_{m\neq n}(1+d_{mn}^2)^{-(\gamma+1)}\right)^{-\frac{\gamma}{\gamma+1}}$$

$$\left[(1+d_{ij}^2)^{-1}\left(\sum_{m\neq n}(1+d_{mn}^2)^{-(\gamma+1)}\right)^{-1}\left(\sum_{k\neq l}(1+d_{kl}^2)^{-\gamma}p_{kl}\right)-p_{ij}\right]$$

$$= \frac{2}{\gamma+1}d_{ij}(Zq_{ij})^{\gamma+1}\parallel Zq\parallel^{-\gamma}\left((Zq_{ij})\parallel Zq\parallel^{-(\gamma+1)}\sum_{k\neq l}(Zq_{kl})^{\gamma}p_{kl}-p_{ij}\right)$$

$$= \frac{2}{\gamma+1}d_{ij}Zq_{ij}^{\gamma+1}\parallel q\parallel^{-\gamma}\left(q_{ij}\parallel q\parallel^{-(\gamma+1)}\sum_{k\neq l}q_{kl}^{\gamma}p_{kl}-p_{ij}\right)$$

$$= \frac{2}{\gamma+1}d_{ij}(1+d_{ij}^2)^{-1}\left(\frac{q_{ij}}{\parallel q\parallel}\right)^{\gamma}\left(\frac{\sum\limits_{k\neq l}q_{kl}^{\gamma}p_{kl}}{\parallel q\parallel^{\gamma+1}}q_{ij}-p_{ij}\right).$$

So the gradient computed in (4.4) is:

$$\frac{4}{\gamma+1}\sum_j(1+d_{ij}^2)^{-1}\left(\frac{q_{ij}}{\parallel q\parallel}\right)^{\gamma}\left(\frac{\sum\limits_{k\neq l}q_{kl}^{\gamma}p_{kl}}{\parallel q\parallel^{\gamma+1}}q_{ij}-p_{ij}\right)(y_i-y_j).$$

Hence, the gradient of the cost function in (4.1) is:

$$\begin{aligned}\frac{\partial D_\gamma}{\partial y_i} &= -\frac{\partial C_\gamma}{\partial y_i}\\ &= \frac{4}{\gamma+1}\sum_j(1+d_{ij}^2)^{-1}\left(\frac{q_{ij}^{\gamma}p_{ij}}{\parallel q\parallel^{\gamma}}-\frac{\sum\limits_{k\neq l}q_{kl}^{\gamma}p_{kl}}{\parallel q\parallel^{2\gamma+1}}q_{ij}^{\gamma+1}\right)(y_i-y_j).\end{aligned} \quad (4.5)$$

□

Then we can observe that as $\gamma\to 0$, we have $\parallel q\parallel\to 1$ by the definition of gamma norm.

doi:10.6342/NTU202001485

The result in (4.5) becomes

$$4\sum_j (p_{ij} - q_{ij})\,(1 + d_{ij}^2)^{-1}(y_i - y_j).$$

This is the gradient of the cost function of t_1-SNE with KL-divergence. It follows that when $\gamma \to 0$, γ-divergence will go to KL-divergence.

4.3 A comparison of gradient with others

In this section, we compare this case to one of the others who also done t_1-SNE with gamma divergence, but with a little different form.

In *Stochastic neighbor embedding (SNE) for dimension reduction and visualization using arbitrary divergences* [1], their gamma-divergence used has the form:

$$D_\gamma(p \parallel q) = \log \frac{[\int p^{\gamma+1}dr]^{1/(\gamma^2+\gamma)} \cdot [\int q^{\gamma+1}dr]^{1/(\gamma+1)}}{(\int p \cdot q^\gamma dr)^{1/\gamma}}$$

It's some different from ours in 2.2.2, especially for taking a logarithm. And they had also computed the gradient of t_1-SNE with gamma-divergence:

$$4\sum_j (1+ \parallel y_i - y_j \parallel^2)^{-1} \left(\frac{p_{ij}q_{ij}^\gamma}{\sum_{kl} p_{kl}q_{kl}^\gamma} - \frac{q_{ij}^{\gamma+1}}{\sum_{kl} q_{kl}^{\gamma+1}} \right)(y_i - y_j) \tag{4.6}$$

We mark the part of our gradient(4.2) in red where is different from above:

$$\frac{4}{\gamma+1}\,\frac{\sum_{kl} p_{kl}q_{kl}^\gamma}{\parallel q \parallel^\gamma} \sum_j (1+ \parallel y_i - y_j \parallel^2)^{-1} \left(\frac{p_{ij}q_{ij}^\gamma}{\sum_{kl} p_{kl}q_{kl}^\gamma} - \frac{q_{ij}^{\gamma+1}}{\parallel q \parallel^{\gamma+1}} \right)(y_i - y_j)$$

Although the gradient is not the same, but the numerical example practice in the last chapter seems almost the same. Probably that it is no different in the direction of the gradient, only the scalar part.

doi:10.6342/NTU202001485

Chapter 5

t_ν-SNE with gamma-divergence

In this chapter we discuss the general case, i.e., t_ν-SNE with minimum gamma-divergence estimation.

5.1 The formulation and the cost function

The cost function with gamma-divergence here is also D_γ in 2.2.2. Here want to compute the gradient of it, then the question is to maximize C_γ in (2.10) when the distribution is t_ν, where

$$C_\gamma = \frac{1}{\gamma(\gamma+1)} \sum_{l \neq k} \left(\frac{q_{lk}}{\parallel q \parallel} \right)^\gamma p_{lk},$$

where $\|q\|$ is given by

$$\|q\| = \|q\|_{\gamma+1} = \left\{ \sum_{l=1, l\neq k}^{n} \sum_{k=1}^{n} q_{lk}^{\gamma+1} \right\}^{1/(\gamma+1)}$$

with

$$q_{ij} = \frac{\left(1 + \frac{\|y_i - y_j\|^2}{\nu}\right)^{-\frac{\nu+1}{2}}}{\sum\limits_{k \neq l} \left(1 + \frac{\|y_k - y_l\|^2}{\nu}\right)^{-\frac{\nu+1}{2}}},$$

and where

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

doi:10.6342/NTU202001485

with

$$p_{j|i} = \frac{\exp(- \parallel x_i - x_j \parallel^2 / 2\sigma_i^2)}{\sum_{k \neq i} \exp(- \parallel x_i - x_k \parallel^2 / 2\sigma_i^2)}.$$

By plugging the new q_{ij} above into C_γ, we can get the general cost function for t_ν-SNE with gamma divergence.

5.2 The gradient

The gradient of the t_ν-SNE with minimum gamma-divergence estimation is derived below.

Proposition 5.2.1. *The cost function of t_ν-SNE with γ-divergence has gradient:*

$$\frac{\partial D_\gamma}{\partial y_i} = \frac{2(\nu+1)}{(\gamma+1)\nu} \sum_j (1 + \frac{\parallel y_i - y_j \parallel^2}{\nu})^{-1} \left(\frac{q_{ij}^\gamma p_{ij}}{\parallel q \parallel^\gamma} - \frac{\sum\limits_{k \neq l} q_{kl}^\gamma p_{kl}}{\parallel q \parallel^{2\gamma+1}} q_{ij}^{\gamma+1} \right) (y_i - y_j). \quad (5.1)$$

We can observe that the above result is quite similar to the gradient form in t_1-SNE with gamma divergence in Chapter 4. When $\gamma \to 0^+$, it is clear that $\frac{\partial D_\gamma}{\partial y_i} \to \frac{\partial C}{\partial y_i}$, the gradient of t_ν-SNE with KL-divergence in (3.2). Moreover, also let ν to be 1, the gradient here will be the original gradient of t-SNE in (2.7).

Proof. By the gradient form in (4.4), we have to compute the partial term $\frac{\partial C_\gamma}{\partial d_{ij}}$ first, where the C_γ is already given in above section, then use the same process in the proof of 4.2.1.

Define two auxiliary variables same as (3.3):

$$\begin{cases} d_{ij} = \parallel y_i - y_j \parallel \\ \\ Z = \sum\limits_{k \neq l} \left(1 + \frac{d_{kl}^2}{\nu}\right)^{-\frac{\nu+1}{2}} \end{cases} \Rightarrow \quad q_{ij} = \frac{\left(1 + \frac{d_{ij}^2}{\nu}\right)^{-\frac{\nu+1}{2}}}{Z}$$

doi:10.6342/NTU202001485

By the form-changing method of C_γ in the proof of 4.2.1, C_γ turns into

$$C_\gamma = \frac{1}{\gamma(\gamma+1)}\left(\sum_{m\neq n}(1+\frac{d_{mn}^2}{\nu})^{-\frac{(\nu+1)(\gamma+1)}{2}}\right)^{-\frac{\gamma}{\gamma+1}}\left(\sum_{k\neq l}(1+\frac{d_{kl}^2}{\nu})^{-\frac{(\nu+1)\gamma}{2}}p_{kl}\right). \tag{5.2}$$

Then the partial term is

$$\begin{aligned}
\frac{\partial C_\gamma}{\partial d_{ij}} &= \frac{\nu+1}{\nu(\gamma+1)}\left(\sum_{m\neq n}(1+\frac{d_{mn}^2}{\nu})^{-\frac{(\nu+1)(\gamma+1)}{2}}\right)^{-\frac{2\gamma+1}{\gamma+1}}(1+\frac{d_{ij}^2}{\nu})^{-\frac{(\nu+1)(\gamma+1)}{2}-1}d_{ij}\left(\sum_{k\neq l}(1+\frac{d_{kl}^2}{\nu})^{-\frac{(\nu+1)\gamma}{2}}p_{kl}\right)\\
&\qquad -\frac{\nu+1}{\nu(\gamma+1)}\left(\sum_{m\neq n}(1+\frac{d_{mn}^2}{\nu})^{-\frac{(\nu+1)(\gamma+1)}{2}}\right)^{-\frac{\gamma}{\gamma+1}}(1+\frac{d_{ij}^2}{\nu})^{-\frac{(\nu+1)\gamma}{2}-1}d_{ij}p_{ij}\\
&= \frac{\nu+1}{\nu(\gamma+1)}\left(\sum_{m\neq n}(1+\frac{d_{mn}^2}{\nu})^{-\frac{(\nu+1)(\gamma+1)}{2}}\right)^{-\frac{\gamma}{\gamma+1}}(1+\frac{d_{ij}^2}{\nu})^{-\frac{(\nu+1)\gamma}{2}-1}d_{ij}\\
&\qquad \left[(1+\frac{d_{ij}^2}{\nu})^{-\frac{\nu+1}{2}}\left(\sum_{m\neq n}(1+\frac{d_{mn}^2}{\nu})^{-\frac{(\nu+1)(\gamma+1)}{2}}\right)^{-1}\left(\sum_{k\neq l}(1+\frac{d_{kl}^2}{\nu})^{-\frac{(\nu+1)\gamma}{2}}p_{kl}\right)-p_{ij}\right]\\
&= \frac{\nu+1}{\nu(\gamma+1)}d_{ij}(Zq_{ij})^{\gamma+\frac{2}{\nu+1}}\parallel Zq\parallel^{-\gamma}\left((Zq_{ij})\parallel Zq\parallel^{-(\gamma+1)}\sum_{k\neq l}(Zq_{kl})^\gamma p_{kl}-p_{ij}\right)\\
&= \frac{\nu+1}{\nu(\gamma+1)}d_{ij}(1+\frac{d_{ij}^2}{\nu})^{-1}\left(\frac{q_{ij}}{\parallel q\parallel}\right)^\gamma\left(\frac{\sum_{k\neq l}q_{kl}^\gamma p_{kl}}{\parallel q\parallel^{\gamma+1}}q_{ij}-p_{ij}\right).
\end{aligned}$$

Hence, the gradient of the cost function about t_ν-SNE with γ-divergence is

$$-\frac{\partial C_\gamma}{\partial y_{ij}} = 2\sum_j\frac{\nu+1}{\nu(\gamma+1)}(1+\frac{d_{ij}^2}{\nu})^{-1}\left(\frac{q_{ij}}{\parallel q\parallel}\right)^\gamma\left(p_{ij}-\frac{\sum_{k\neq l}q_{kl}^\gamma p_{kl}}{\parallel q\parallel^{\gamma+1}}q_{ij}\right)(y_i-y_j) \tag{5.3}$$

□

doi:10.6342/NTU202001485

At last, we show all the gradients computed above together to see their similarities and differences easily:

t_1 -SNE with KL-divergence: $\frac{\partial C}{\partial y_i} = 4\sum_j \left(1+\| y_i - y_j \|^2\right)^{-1}(p_{ij}-q_{ij})(y_i-y_j)$

t_ν -SNE with KL-divergence: $\frac{\partial C}{\partial y_i} = \frac{2(\nu+1)}{\nu}\sum_j \left(1+\frac{\| y_i - y_j \|^2}{\nu}\right)^{-1}(p_{ij}-q_{ij})(y_i-y_j)$

t_1-SNE with γ –divergence: $\frac{\partial D_\gamma}{\partial y_i} = \frac{4}{\gamma+1}\sum_j (1+\| y_i - y_j \|^2)^{-1}\left(\frac{q_{ij}^\gamma p_{ij}}{\| q \|^\gamma} - \frac{\sum_{k\neq l} q_{kl}^\gamma p_{kl}}{\| q \|^{2\gamma+1}} q_{ij}^{\gamma+1}\right)(y_i-y_j)$

t_ν-SNE with γ –divergence: $\frac{\partial D_\gamma}{\partial y_i} = \frac{2(\nu+1)}{(\gamma+1)\nu}\sum_j (1+\frac{\| y_i - y_j \|^2}{\nu})^{-1}\left(\frac{q_{ij}^\gamma p_{ij}}{\| q \|^\gamma} - \frac{\sum_{k\neq l} q_{kl}^\gamma p_{kl}}{\| q \|^{2\gamma+1}} q_{ij}^{\gamma+1}\right)(y_i-y_j)$

Figure 5.1: Comparison of gradients with different case

doi:10.6342/NTU202001485

Chapter 6

Numerical Examples

In this chapter, we present two numerical examples to visualize the low-dimensional embedding by t_ν-SNE with various degrees of freedom ($\nu = 1, 2, 3$), various gamma values ($\gamma = 0.1, 10^{-3}, 10^{-5}$), and various perplelxity levels ($Perp = 50, 100, 150, 200$).

The first data example is the MNIST dataset, a set of well-known hand-written digits data, which can be downloaded from *Kaggle*. Please refer to *Visualizing data using t-SNE* (2008) [9] and van der Maaten's website [8]. The authors have provided Python code for t_1-SNE with minimum KL-divergence estimation. We modified and extended their code to t_ν-SNE with minimum gamma-divergence estimation.

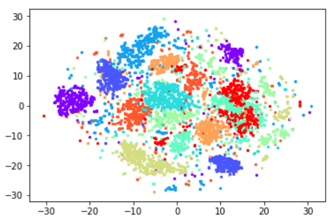
The second data is also from Kaggle and is about the 'Dogs vs. Cats', which was originally used for distinguishing between dog and cat images. This is a very easy task for human eyes, but it might be quite difficult for computer to do it. There is a transfer-learning method to extract dog-cat data features using a pre-trained model VGG16.

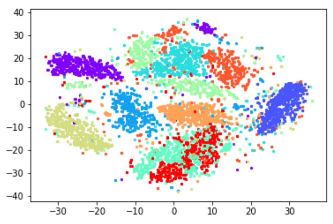
6.1 MNIST data

The original MNIST data set has 60000 digits as training set. Each digit is a 28x28 array (or a 784-vector), with elements taking values in $\{0,1,...,255\}$ by representing the grayscale of an image pixel. Since the size is quite large for a typical personal computer, we ran a random stratified sampling to take 5000 or 10000 digits from the original data and also used fewer iterations than the original code. Here we first demonstrate the part of 5000

doi:10.6342/NTU202001485

data size with $Perp = 100$. From Fig.6.1∼6.3, we can see different effects of γ values.


(a) $\gamma = 10^{-1}$


(b) $\gamma = 10^{-3}$

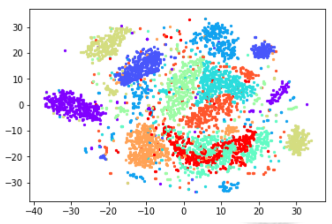
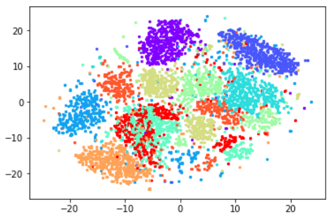
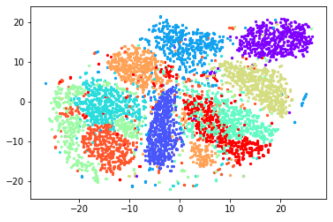

(c) $\gamma = 10^{-5}$

Figure 6.1: t with degree of freedom ν=1 (t_1-MNIST5000)


(a) $\gamma = 10^{-1}$


(b) $\gamma = 10^{-3}$

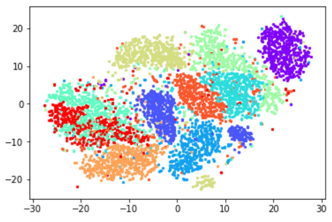
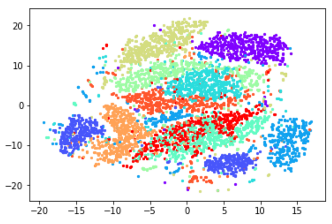
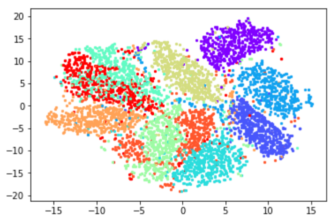

(c) $\gamma = 10^{-5}$

Figure 6.2: t with degrees of freedom ν=2 (t_2-MNIST5000)


(a) $\gamma = 10^{-1}$


(b) $\gamma = 10^{-3}$

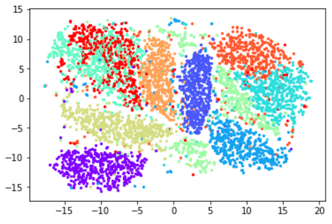
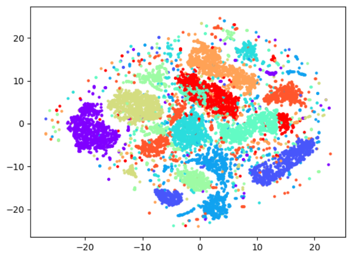

(c) $\gamma = 10^{-5}$

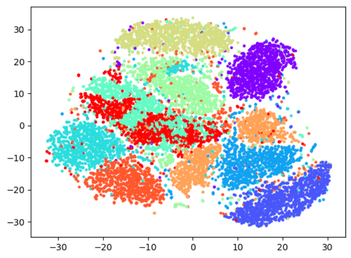
Figure 6.3: t with degrees of freedom ν=3 (t_3-MNIST5000)

For larger γ ($\gamma = 0.1$), the performance is obviously worse than others. It seems similar when $\gamma = 10^{-3}$ or $\gamma = 10^{-5}$. For the effect of degrees of freedom of t, we can see that for larger ν, the group of same digits become looser, and the boundaries between different digit groups are likely more clear but near. Moreover, the influence of the size of γ's is decreased.

doi:10.6342/NTU202001485

For larger data size, it seems that only the density of points in the graphs is changed. Fig.6.4 is the 10000 size MNIST, with $\nu = 2$, and also with $Perp = 100$.


(a) $\gamma = 10^{-1}$


(b) $\gamma = 10^{-3}$

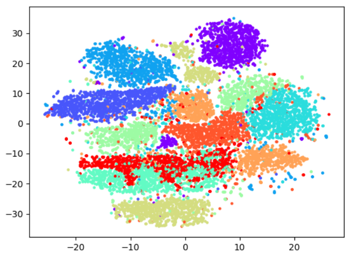

(c) $\gamma = 10^{-5}$

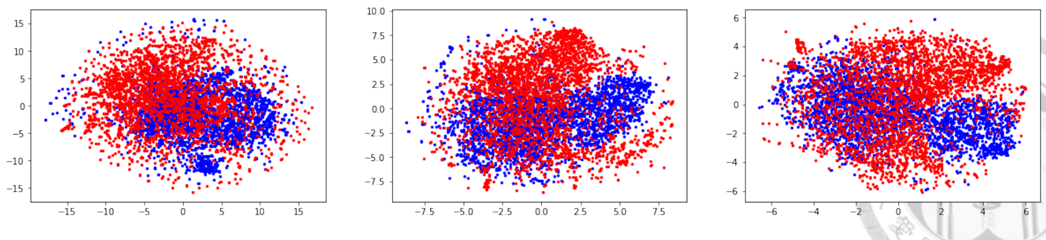
Figure 6.4: $\nu = 2$ (t_2-MNIST10000)

6.2 Dogs v.s. Cats dataset

The original Dogs vs. Cats dataset has size 25,000, with half of the dogs and the other half the cats. These images have different shapes and pixels. We first do some preprocessing for preparation to feed into the pre-trained VGG16 deep neural network. This VGG16 was developed by Karen Simonyan and Andrew Zisserman in 2014, which is a convolution network that has been trained on the ImageNet dataset. The ImageNet dataset has huge labeled images and thousands of different classes, and the images contain various animals include our characters: dogs and cats. Hence the pre-trained VGG16 might be a nice feature extractor for the Dogs and Cats dataset. For the procedures of pre-trained VGG16, we refer the reader to the book *"Deep Learning with Python"* [4] in chapter 5, and the implementation code is available on github [3].

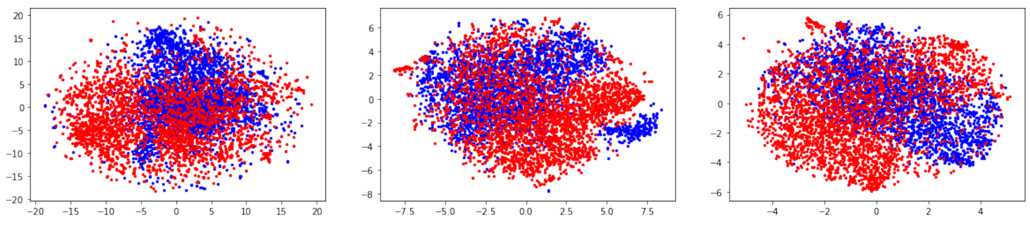
Here we use only partial data ($n = 6000$, half dogs and half cats) for demonstration. Given blue points are the dogs, and the red ones are the cats. We set $Perp = 100$, and vary γ and ν. The graphs are on the next page.

doi:10.6342/NTU202001485



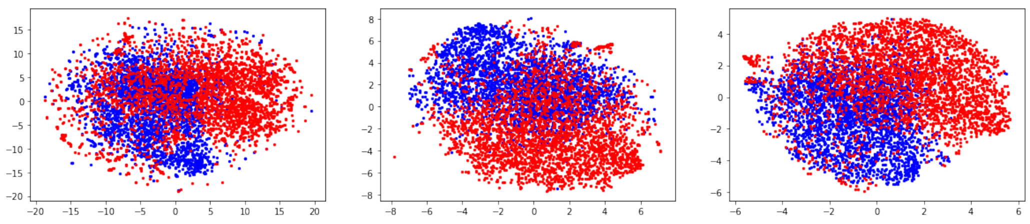
(a) $\nu = 1$ (b) $\nu = 2$ (c) $\nu = 3$

Figure 6.5: $\gamma = 10^{-1}$ (DC6000-g1)



(a) $\nu = 1$ (b) $\nu = 2$ (c) $\nu = 3$

Figure 6.6: $\gamma = 10^{-3}$ (DC6000-g3)



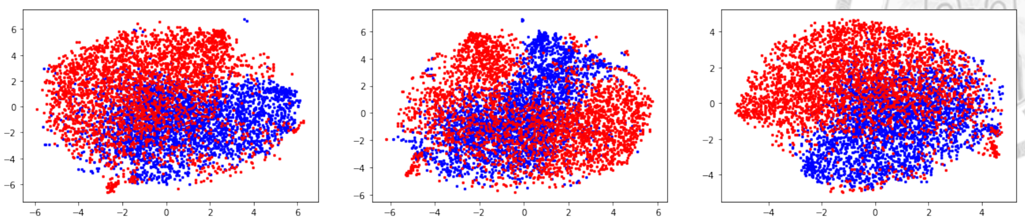
(a) $\nu = 1$ (b) $\nu = 2$ (c) $\nu = 3$

Figure 6.7: $\gamma = 10^{-5}$ (DC6000-g5)

From Fig.6.5∼6.7 we see that with smaller γ or larger ν under the same perplexity 100, the cats and dogs seem to be more away from each other. When the ν becomes larger, the group of points of either the cats or dogs becomes more concentrated.

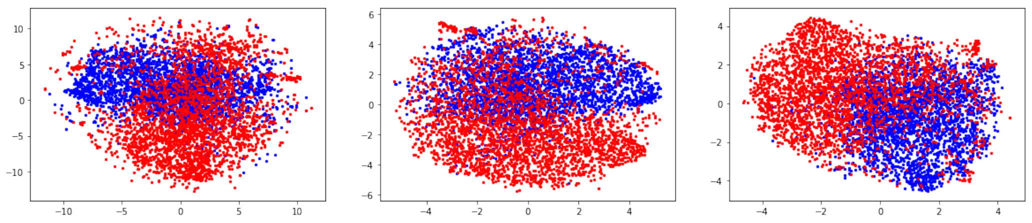
doi:10.6342/NTU202001485

With different perplexity ($Perp$ = 150, 200) and with $\gamma = 10^{-5}$, we have the following results (Fig.6.8, 6.9).



(a) $\nu = 1$ (b) $\nu = 2$ (c) $\nu = 3$

Figure 6.8: *Perp* =150 (DC6000-p150)



(a) $\nu = 1$ (b) $\nu = 2$ (c) $\nu = 3$

Figure 6.9: *Perp* =200 (DC6000-p200)

From Fig.6.8, 6.9, we see that with larger perplexity it will be more suitable to visualize this data since we can see that for $Perp$ = 200, it becomes more clear between cats and dogs.

doi:10.6342/NTU202001485

doi:10.6342/NTU202001485

Bibliography

[1] K. Bunte, S. Haase, M. Biehl, and T. Villmann. Stochastic neighbor embedding (sne) for dimension reduction and visualization using arbitrary divergences. *Neurocomputing*, 08 2012.

[2] T.-L. Chen, D.-N. Hsieh, H. Hung, I.-P. Tu, P.-S. Wu, Y.-M. Wu, W.-H. Chang, and S.-Y. Huang. γ-sup: A clustering algorithm for cryo-electron microscopy images of asymmetric particles. *The Annals of Applied Statistics*, 8, 05 2012.

[3] F. Chollet. Deep Learning with Python 5.3-using-a-pretrained-convnet.ipynb. https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/5.3-using-a-pretrained-convnet.ipynb.

[4] F. Chollet. *Deep Learning with Python*. MANNING, 2017.

[5] H. Fujisawa and S. Eguchi. Robust parameter estimation with a small bias against heavy contamination. *Journal of Multivariate Analysis*, 99:2053–2081, 10 2008.

[6] G. Hinton and S. Roweis. Stochastic neighbor embedding. *In Advances in Neural Information Processing Systems*, 15(1):833–840, 2002.

[7] H. Hung, Z.-Y. Jou, and S.-Y. Huang. Robust mislabel logistic regression without modeling mislabel probabilities. *Biometrics*, 74, 08 2016.

[8] L. van der Maaten. Python implementation standard implementations of t-sne in python. https://lvdmaaten.github.io/tsne.

[9] L. van der Maaten and G. Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605, 11 2008.

doi:10.6342/NTU202001485

doi:10.6342/NTU202001485

Appendix

Python code for t_ν-SNE with gamma-divergence

Please refer to *Visualizing data using t-SNE* (2008) [9] and van der Maaten's website [8] for the original work and code for t_1-SNE with minimum KL-divergence. We modified and extended their code to t_ν-SNE with minimum gamma-divergence estimation.

This is an implementation of MNIST with 5000 data points, t-distribution's degrees of freedom = 3, γ value = 10^{-3}, and $Perp$ = 100.

```
\#  t_nu_gamma−SNE.py
\#
\#  Original version was created by Laurens van der Maaten on 20−12−08.
\#  Source from : https://lvdmaaten.github.io/tsne/
\#  This is an adaptation by Chiou Yu Hsuan in 2019.
import numpy as np
import pylab
import matplotlib.cm as cm

nu_of_t = 3          ## t−distribution 's freedom
Gamma = 0.001        ## gamma value of the used Gamma−divergence
perplexity = 100     ## perplexity variable
data_nums = 5000     ## data form

def Hbeta(D=np.array([]), beta=1.0):
"""
```

doi:10.6342/NTU202001485

```
Compute the perplexity and the P–row for a specific value of the
precision of a Gaussian distribution.
"""

# Compute P–row and corresponding perplexity
P = np.exp(–D.copy() * beta)
sumP = sum(P)
H = np.log(sumP) + beta * np.sum(D * P) / sumP
P = P / sumP
return H, P


def x2p(X=np.array([]), tol=1e–5, perplexity=perplexity):

print("Computing pairwise distances...")
(n, d) = X.shape
sum_X = np.sum(np.square(X), 1)
D = np.add(np.add(–2 * np.dot(X, X.T), sum_X).T, sum_X)
P = np.zeros((n, n))
beta = np.ones((n, 1))
logU = np.log(perplexity)

# Loop over all datapoints
for i in range(n):

# Print progress
if i % 500 == 0:
print("Computing P–values for point %d of %d..." % (i, n))

# Compute the Gaussian kernel and entropy for the current precision
betamin = –np.inf
```

doi:10.6342/NTU202001485

```
betamax = np.inf
Di = D[i, np.concatenate((np.r_[0:i], np.r_[i+1:n]))]
(H, thisP) = Hbeta(Di, beta[i])

# Evaluate whether the perplexity is within tolerance
Hdiff = H - logU
tries = 0
while np.abs(Hdiff) > tol and tries < 50:

# If not, increase or decrease precision
if Hdiff > 0:
betamin = beta[i].copy()
if betamax == np.inf or betamax == -np.inf:
beta[i] = beta[i] * 2.
else:
beta[i] = (beta[i] + betamax) / 2.
else:
betamax = beta[i].copy()
if betamin == np.inf or betamin == -np.inf:
beta[i] = beta[i] / 2.
else:
beta[i] = (beta[i] + betamin) / 2.

# Recompute the values
(H, thisP) = Hbeta(Di, beta[i])
Hdiff = H - logU
tries += 1

# Set the final row of P
P[i, np.concatenate((np.r_[0:i], np.r_[i+1:n]))] = thisP
```

doi:10.6342/NTU202001485

```
# Return final P—matrix
print("Mean value of sigma: %f" % np.mean(np.sqrt(1 / beta)))
return P



def pca(X=np.array([]), no_dims=50):
"""
Runs PCA on the NxD array X in order to reduce its dimensionality to
no_dims dimensions.
"""


print("Preprocessing the data using PCA...")
(n, d) = X.shape
X = X — np.tile(np.mean(X, 0), (n, 1))
(l, M) = np.linalg.eig(np.dot(X.T, X))
Y = np.dot(X, M[:, 0:no_dims])
return Y

def gamma_norm(Array, gamma = Gamma):
"""To compute the gamma norm of Q """
return np.sum(np.power(Array, gamma+1))**(1/(gamma+1))

def tsne(X=np.array([]), no_dims=2, initial_dims=50, perplexity, gamma, nu):

# Check inputs
if isinstance(no_dims, float):
print("Error: array X should have type float.")
return —1
if round(no_dims) != no_dims:
print("Error: number of dimensions should be an integer.")
return —1
```

doi:10.6342/NTU202001485

```
# Initialize variables
X = pca(X, initial_dims).real
(n, d) = X.shape
max_iter = 130  #1000
initial_momentum = 0.5
final_momentum = 0.8
eta = 500
min_gain = 0.01
Y = np.random.randn(n, no_dims)
dY = np.zeros((n, no_dims))
iY = np.zeros((n, no_dims))
gains = np.ones((n, no_dims))

# Compute P−values
P = x2p(X, 1e−5, perplexity)
P = P + np.transpose(P)
P = P / np.sum(P)
P = P * 4                          # early exaggeration
P = np.maximum(P, 1e−12)

# Run iterations
for iter in range(max_iter):

# Compute pairwise affinities
sum_Y = np.sum(np.square(Y), 1)
num = −2. * np.dot(Y, Y.T)
## modified by t_nu distributed :
num = (1. + np.add(np.add(num, sum_Y).T, sum_Y)/nu)**(−(nu + 1)/2)
num_recipr = num**(2/(nu+1))
num[range(n), range(n)] = 0.
```

doi:10.6342/NTU202001485

```
Q = num / np.sum(num)
Q = np.maximum(Q, 1e-12)
Q_n = gamma_norm(Q)

# Compute gradient ## my gradient part
PQ = Q**gamma*P/Q_n**gamma/(gamma+1)
        - Q**(gamma+1)*np.sum(Q**gamma*P)/Q_n**(2*gamma+1)/(gamma+1)

for i in range(n):
dY[i, :] = 2*(nu+1)/nu*np.sum(np.tile(PQ[:, i]
        * num_recipr[:, i], (no_dims, 1)).T * (Y[i, :] - Y), 0)

# Perform the update
if iter < 20:
momentum = initial_momentum
else:
momentum = final_momentum
gains = (gains + 0.2) * ((dY > 0.) != (iY > 0.)) + \
(gains * 0.8) * ((dY > 0.) == (iY > 0.))
gains[gains < min_gain] = min_gain
iY = momentum * iY - eta * (gains * dY)
Y = Y + iY
Y = Y - np.tile(np.mean(Y, 0), (n, 1))

# Compute current value of cost function
if (iter + 1) % 10 == 0:
C = (gamma_norm(P)-np.sum(Q**gamma * P)/gamma_norm(Q))/(gamma*(gamma+1))
print(”Iteration %d: error is %f” % (iter + 1, C))

# Stop lying about P-values
if iter == 100:
```

doi:10.6342/NTU202001485

```
P = P / 4.

# Return solution
return Y


if __name__ == "__main__":
print("Run Y = tsne(X, no_dims, perplexity) to perform modified t-SNE.")
print("Running example on 5,000 MNIST digits...")

with open("mnist_Train.txt", "r") as f:
data = f.read()

labels = []
label_lines = data.split('labels ')

for i in range(data_nums):
for j in range(10):
if label_lines[4*i+1][2*j] == '1':
labels.append(j)

X = []
lines = data.split('features ')
for i in range(data_nums):
raw = lines[4*i+1].split('\n')[0].split(' ')
for j in range(784):
raw[j] = int(raw[j])/255

X.append(raw)
raw = []
X=np.asarray(X)
```

doi:10.6342/NTU202001485

```
# Visualized
Y = tsne(X, 2, 50, perplexity)

Color=cm.rainbow(np.linspace(0, 1, 10))
for i in range(data_nums):
pylab.scatter(Y[i, 0], Y[i, 1], 5, c=Color[int(labels[i])])
pylab.show()
```

Python code of pre-trained Dogs and Cats dataset with VGG16

Here, we list the python code of pre-training the ‘Dogs and Cats’ image data with VGG16 network by using some method to convert the images to the size we want to use for running the t_ν-SNE code above.

```
import tensorflow as tf;
import tensorflow.keras;
from PIL import Image;
from tensorflow.keras.preprocessing import image;
import os;
os.environ['KMP\_DUPLICATE\_LIB\_OK']='True';
import numpy as np;

# load data
test_datagen = image.ImageDataGenerator(rescale=1./255)

data_generator = test_datagen.flow_from_directory(
'train_data',
target_size=(150, 150),
color_mode="rgb",
```

doi:10.6342/NTU202001485

```
shuffle = False ,
class_mode='binary ',
batch_size=1)
filenames = data_generator . filenames ;
n = len ( filenames ) ;

vgg16_model = tf . keras . applications . vgg16 . VGG16( include_top=False ,
weights='imagenet ',
input_tensor=tf . keras . layers . Input ( shape =(150 ,150 ,3))) ;

output = vgg16_model . predict_generator ( data_generator , steps = n) ;
output_vec = np . reshape ( output ,(6000 , −1))
np . save ( 'vggout . npy ', output_vec ) ;
```

doi:10.6342/NTU202001485