the function at a point:

$$(\mathbf{L}\boldsymbol{f})(i) = \sum_{j \in \text{nbr}_i} W_{ij}[f(i) - f(j)] \tag{20.127}$$

where nbr_i is the set of neighbors of node i. We can also compute an overall measure of "smoothness" of the function f by computing its **Dirichlet energy** as follows:

$$\boldsymbol{f}^T\mathbf{L}\boldsymbol{f} = \boldsymbol{f}^T\mathbf{D}\boldsymbol{f} - \boldsymbol{f}^T\mathbf{W}\boldsymbol{f} = \sum_i d_i f_i^2 - \sum_{i,j} f_i f_j w_{ij} \tag{20.128}$$

$$= \frac{1}{2}\left(\sum_i d_i f_i^2 - 2\sum_{i,j} f_i f_j w_{ij} + \sum_j d_j f_j^2\right) = \frac{1}{2}\sum_{i,j} w_{ij}(f_i - f_j)^2 \tag{20.129}$$

By studying the eigenvalues and eigenvectors of the Laplacian matrix, we can determine various useful properties of the function. (Applying linear algebra to study the adjacency matrix of a graph, or related matrices, is called **spectral graph theory** [Chu97].) For example, we see that $\mathbf{L}$ is symmetric and positive semi-definite, since we have $\boldsymbol{f}^T\mathbf{L}\boldsymbol{f} \geq 0$ for all $\boldsymbol{f} \in \mathbb{R}^N$, which follows from Equation (20.129) due to the assumption that $w_{ij} \geq 0$. Consequently $\mathbf{L}$ has N non-negative, real-valued eigenvalues, $0 \leq \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_N$. The corresponding eigenvectors form an orthogonal basis for the function f defined on the graph, in order of decreasing smoothness.

In Section 20.4.9.1, we discuss Laplacian eigenmaps, which is a way to learn low dimensional embeddings for high dimensional data vectors. The approach is to let $z_{id} = f_i^d$ be the d'th embedding dimension for input i, and then to find a basis for these functions (i.e., embedding of the points) that varies smoothly over the graph, thus respecting distance of the points in ambient space.

There are many other applications of the graph Laplacian in ML. For example, in Section 21.5.1, we discuss normalized cuts, which is a way to learn a clustering of high dimensional data vectors based on pairwise similarity; and [WTN19] discusses how to use the eigenvectors of the state transition matrix to learn representations for RL.

20.4.10 t-SNE

In this section, we describe a very popular nonconvex technique for learning low dimensional embeddings called **t-SNE** [MH08]. This extends the earlier **stochastic neighbor embedding** method of [HR03], so we first describe SNE, before describing the t-SNE extension.

20.4.10.1 Stochastic neighborhood embedding (SNE)

The basic idea in SNE is to convert high-dimensional Euclidean distances into conditional probabilities that represent similarities. More precisely, we define $p_{j|i}$ to be the probability that point i would pick point j as its neighbor if neighbors were picked in proportion to their probability under a Gaussian centered at $\boldsymbol{x}_i$:

$$p_{j|i} = \frac{\exp(-\frac{1}{2\sigma_i^2}||\boldsymbol{x}_i - \boldsymbol{x}_j||^2)}{\sum_{k\neq i}\exp(-\frac{1}{2\sigma_i^2}||\boldsymbol{x}_i - \boldsymbol{x}_k||^2)} \tag{20.130}$$

Author: Kevin P. Murphy. (C) MIT Press. CC-BY-NC-ND license

Here σ_i^2 is the variance for data point i, which can be used to "magnify" the scale of points in dense regions of input space, and diminish the scale in sparser regions. (We discuss how to estimate the length scales σ_i^2 shortly).

Let $\boldsymbol{z}_i$ be the low dimensional embedding representing $\boldsymbol{x}_i$. We define similarities in the low dimensional space in an analogous way:

$$q_{j|i} = \frac{\exp(-||\boldsymbol{z}_i - \boldsymbol{z}_j||^2)}{\sum_{k \neq i} \exp(-||\boldsymbol{z}_i - \boldsymbol{z}_k||^2)} \tag{20.131}$$

In this case, the variance is fixed to a constant; changing it would just rescale the learned map, and not change its topology.

If the embedding is a good one, then $q_{j|i}$ should match $p_{j|i}$. Therefore, SNE defines the objective to be

$$\mathcal{L} = \sum_i D_{\mathbb{KL}}(P_i \parallel Q_i) = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}} \tag{20.132}$$

where P_i is the conditional distribution over all other data points given $\boldsymbol{x}_i$, Q_i is the conditional distribution over all other latent points given $\boldsymbol{z}_i$, and $D_{\mathbb{KL}}(P_i \parallel Q_i)$ is the KL divergence (Section 6.2) between the distributions.

Note that this is an asymmetric objective. In particular, there is a large cost if a small $q_{j|i}$ is used to model a large $p_{j|i}$. This objective will prefer to pull distant points together rather than push nearby points apart. We can get a better idea of the geometry by looking at the gradient for each embedding vector, which is given by

$$\nabla_{\boldsymbol{z}_i} \mathcal{L}(\mathbf{Z}) = 2 \sum_j (\boldsymbol{z}_i - \boldsymbol{z}_j)(p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j}) \tag{20.133}$$

Thus points are pulled towards each other if the p's are bigger than the q's, and repelled if the q's are bigger than the p's.

Although this is an intuitively sensible objective, it is not convex. Nevertheless it can be minimized using SGD. In practice, it helps to add Gaussian noise to the embedding points, and to gradually anneal the amount of noise. [Hin13] recommends to "spend a long time at the noise level at which the global structure starts to form from the hot plasma of map points" before reducing it.[6]

20.4.10.2 Symmetric SNE

There is a slightly simpler version of SNE that minimizes a single KL between the joint distribution P in high dimensional space and Q in low dimensional space:

$$\mathcal{L} = D_{\mathbb{KL}}(P \parallel Q) = \sum_{i<j} p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{20.134}$$

This is called **symmetric SNE**.

6. See [Ros98; WF20] for a discussion of annealing and phase transitions in unsupervised learning. See also [CP10] for a discussion of the **elastic embedding** algorithm, which uses a homotopy method to more efficiently optimize a model that is related to both SNE and Laplacian eigenmaps.

The obvious way to define p_{ij} is to use

$$p_{ij} = \frac{\exp(-\frac{1}{2\sigma^2}||\boldsymbol{x}_i - \boldsymbol{x}_j||^2)}{\sum_{k \neq l} \exp(-\frac{1}{2\sigma^2}||\boldsymbol{x}_k - \boldsymbol{x}_l||^2)} \tag{20.135}$$

We can define q_{ij} similarly.

The corresponding gradient becomes

$$\nabla_{\boldsymbol{z}_i} \mathcal{L}(\mathbf{Z}) = 4 \sum_j (\boldsymbol{z}_i - \boldsymbol{z}_j)(p_{ij} - q_{ij}) \tag{20.136}$$

As before, points are pulled towards each other if the p's are bigger than the q's, and repelled if the q's are bigger than the p's.

Although symmetric SNE is slightly easier to implement, it loses the nice property of regular SNE that the data is its own optimal embedding if the embedding dimension L is set equal to the ambient dimension D. Nevertheless, the methods seems to give similar results in practice on real datasets where $L \ll D$.

20.4.10.3 t-distributed SNE

A fundamental problem with SNE and many other embedding techniques is that they tend to squeeze points that are relatively far away in the high dimensional space close together in the low dimensional (usually 2d) embedding space; this is called the **crowding problem**, and arises due to the use of squared errors (or Gaussian probabilities).

One solution to this is to use a probability distribution in latent space that has heavier tails, which eliminates the unwanted attractive forces between points that are relatively far in the high dimensional space. An obvious choice is the Student-t distribution (Section 2.7.1). In t-SNE, they set the degree of freedom parameter to $\nu = 1$, so the distribution becomes equivalent to a Cauchy:

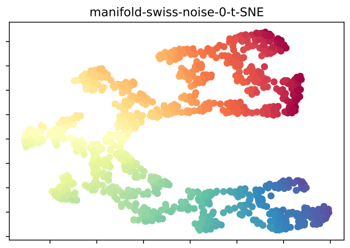
$$q_{ij} = \frac{(1 + ||\boldsymbol{z}_i - \boldsymbol{z}_j||^2)^{-1}}{\sum_{k<l}(1 + ||\boldsymbol{z}_k - \boldsymbol{z}_l||^2)^{-1}} \tag{20.137}$$

We can use the same global KL objective as in Equation (20.134). For t-SNE, the gradient turns out to be

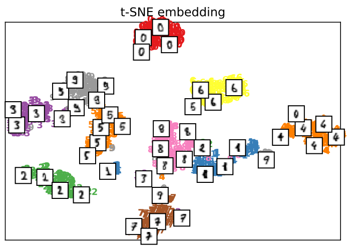
$$\nabla \boldsymbol{z}_i \mathcal{L} = 4 \sum_j (p_{ij} - q_{ij})(\boldsymbol{z}_i - \boldsymbol{z}_j)(1 + ||\boldsymbol{z}_i - \boldsymbol{z}_j||^2)^{-1} \tag{20.138}$$

The gradient for symmetric (Gaussian) SNE is the same, but lacks the $(1 + ||\boldsymbol{z}_i - \boldsymbol{z}_j||^2)^{-1}$ term. This term is useful because $(1 + ||\boldsymbol{z}_i - \boldsymbol{z}_j||^2)^{-1}$ acts like an inverse square law. This means that points in embedding space act like stars and galaxies, forming many well-separated clusters (galaxies) each of which has many stars tightly packed inside. This can be useful for separating different classes of data in an unsupervised way (see Figure 20.41 for an example).

Author: Kevin P. Murphy. (C) MIT Press. CC-BY-NC-ND license



(a)



(b)

Figure 20.41: tSNE applied to (a) Swiss roll. Generated by manifold_swiss_sklearn.ipynb. (b) UCI digits. Generated by manifold_digits_sklearn.ipynb.

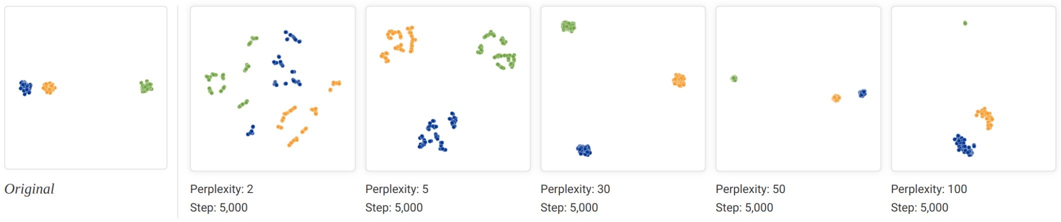


Figure 20.42: Illustration of the effect of changing the perplexity parameter when t-SNE is applied to some 2d data. From [WVJ16]. See `http://distill.pub/2016/misread-tsne` for an animated version of these figures. Used with kind permission of Martin Wattenberg.

20.4.10.4 Choosing the length scale

An important parameter in t-SNE is the local bandwidth σ_i^2. This is usually chosen so that P_i has a perplexity chosen by the user.[7] This can be interpreted as a smooth measure of the effective number of neighbors.

Unfortunately, the results of t-SNE can be quite sensitive to the perplexity parameter, so it is wise to run the algorithm with many different values. This is illustrated in Figure 20.42. The input data is 2d, so there is no distortion generating by mapping to a 2d latent space. If the perplexity is too small, the method tends to find structure within each cluster which is not truly present. At perplexity 30 (the default for scikit-learn), the clusters seem equi-distant in embedding space, even though some are closer than others in the data space. Many other caveats in interpreting t-SNE plots can be found in [WVJ16].

7. The perplexity is defined to be $2^{\mathbb{H}(P_i)}$, where $\mathbb{H}(P_i) = -\sum_j p_{j|i} \log_2 p_{j|i}$ is the entropy; see Section 6.1.5 for details. A big radius around each point (large value of σ_i) will result in a high entropy, and thus high perplexity.

20.4.10.5 Computational issues

The naive implementation of t-SNE takes $O(N^2)$ time, as can be seen from the gradient term in Equation (20.138). A faster version can be created by leveraging an analogy to N-body simulation in physics. In particular, the gradient requires computing the force of N points on each of N points. However, points that are far away can be grouped into clusters (computationally speaking), and their effective force can be approximated by a few representative points per cluster. We can then approximate the forces using the **Barnes-Hut algorithm** [BH86], which takes $O(N \log N)$ time, as proposed in [Maa14]. Unfortunately, this only works well for low dimensional embeddings, such as $L = 2$.

20.4.10.6 UMAP

Various extensions of tSNE have been proposed, that try to improve its speed, the quality of the embedding space, or the ability to embed into more than 2 dimensions.

One popular recent extension is called **UMAP** (which stands for "Uniform Manifold Approximation and Projection"), was proposed in [MHM18]. At a high level, this is similar to tSNE, but it tends to preserve global structure better, and it is much faster. This makes it easier to try multiple values of the hyperparameters. For an interactive tutorial on UMAP, and a comparison to tSNE, see [CP19].

20.5 Word embeddings

Words are categorical random variables, so their corresponding one-hot vector representations are sparse. The problem with this binary representation is that semantically similar words may have very different vector representations. For example, the pair of related words "man" and "woman" will be Hamming distance 1 apart, as will the pair of unrelated words "man" and "banana".

The standard way to solve this problem is to use **word embeddings**, in which we map each sparse one-hot vector, $\boldsymbol{s}_{n,t} \in \{0,1\}^M$, representing the t'th word in document n, to a lower-dimensional dense vector, $\boldsymbol{z}_{n,t} \in \mathbb{R}^D$, such that semantically similar words are placed close by. This can significantly help with data sparsity. There are many ways to learn such embeddings, as we discuss below.

Before discussing methods, we have to define what we mean by "semantically similar" words. We will assume that two words are semantically similar if they occur in similar contexts. This is known as the **distributional hypothesis** [Har54], which is often summarized by the phase (originally from [Fir57]) "a word is characterized by the company it keeps". Thus the methods we discuss will all learn a mapping from a word's context to an embedding vector for that word.

20.5.1 Latent semantic analysis / indexing

In this section, we discuss a simple way to learn word embeddings based on singular value decomposition (Section 7.5) of a term-frequency count matrix.

20.5.1.1 Latent semantic indexing (LSI)

Let C_{ij} be the number of times "term" i occurs in "context" j. The definition of what we mean by "term" is application-specific. In English, we often take it to be the set of unique tokens that are separated by punctuation or whitespace; for simplicity, we will call these "words". However, we may