Entropic Affinities: Properties and Efficient Numerical Computation

Max Vladymyrov MVLADYMYROV@UCMERCED.EDU
Miguel Á. Carreira-Perpiñán MCARREIRA-PERPINAN@UCMERCED.EDU
Electrical Engineering and Computer Science, School of Engineering, University of California, Merced

Abstract

Gaussian affinities are commonly used in graph-based methods such as spectral clustering or nonlinear embedding. Hinton and Roweis (2003) introduced a way to set the scale individually for each point so that it has a distribution over neighbors with a desired perplexity, or effective number of neighbors. This gives very good affinities that adapt locally to the data but are harder to compute. We study the mathematical properties of these "entropic affinities" and show that they implicitly define a continuously differentiable function in the input space and give bounds for it. We then devise a fast algorithm to compute the widths and affinities, based on robustified, quickly convergent root-finding methods combined with a tree- or density-based initialization scheme that exploits the slowly-varying behavior of this function. This algorithm is nearly optimal and much more accurate and fast than the existing bisection-based approach, particularly with large datasets, as we show with image and text data.

Many machine learning algorithms rely on the choice of meta-parameters that govern their performance. These parameters depend on the data and good values are often hard to find. One such meta-parameter is the bandwidth σ that is used in the construction of affinities in many machine learning problems. These include dimensionality reduction methods such as LLE (Roweis & Saul, 2000), Laplacian eigenmaps (Belkin & Niyogi, 2003), ISOMAP (de Silva & Tenenbaum, 2003), SNE (Hinton & Roweis, 2003), and the elastic embedding (Carreira-Perpiñán, 2010); clustering methods such as spectral clustering (Ng et al., 2002) and mean-shift algorithms (Carreira-Perpiñán, 2006); semi-supervised learning (Zhou et al., 2004; Belkin et al., 2006); and many others. In some of those algorithms σ is the only parameter to tune and a user has to try several values until the desired quality of the algorithm is achieved. When the dataset is large, such a process is not interactive and can lead to frustration and ultimately to the user refusing to use a potentially good algorithm. On top of that, the best results of the algorithm may not be achieved for a single value of σ for all the points, but rather for a separate bandwidth for every datapoint, in which case the existence of automatic procedure is vital.

In their spectral clustering algorithm, Ng et al. (2002) suggest to set σ to the value giving least distorted clusters, but this requires running the algorithm, which is expensive. In a supervised setting, Er et al. (2002) select the bandwidth per cluster of data as the one that captures the variation between points in each cluster, but minimizes the overlapping of nearest neighbors in different classes. The method requires tuning some parameters that depend on the mean and variance of the clusters. Bchir & Frigui (2010) estimate one σ per cluster in an unsupervised manner using a fuzzy logic framework. The objective function of this method maximizes the scaling parameter per cluster up until the clusters start to overlap. There also exist classic rules of thumb, such as setting σ separately for each point to the distance d_k to the kth nearest neighbor of that point, where k is a user parameter (set to 7 in Zelnik-Manor & Perona, 2005). This has the odd behavior that σ would change proportionally to changes in d_k, but would ignore any changes to the rest of the distances, no matter how large, as long as d_k remained the kth distance; or else it would change discontinuously.

In this paper, we study a previously proposed way to set per-point bandwidths that takes into account the whole distribution of distances and is a continuous, differentiable function of them. For a given point $\mathbf{x} \in$

Proceedings of the 30th International Conference on Machine Learning, Atlanta, Georgia, USA, 2013. JMLR: W&CP volume 28.

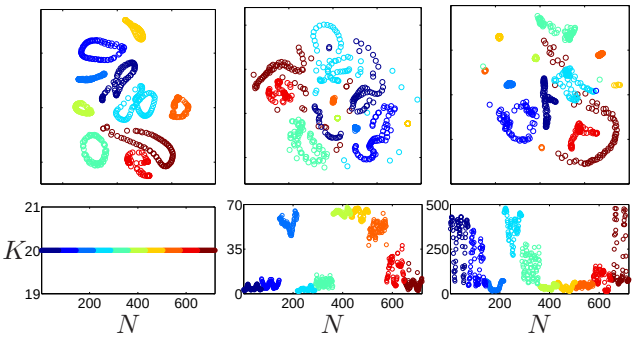

Figure 1. Top plots: embeddings of COIL dataset with the elastic embedding algorithm, using (from left to right): entropic affinities with perplexity $K = 20$; unique $\sigma = 9$ obtained by averaging σs with perplexity $K = 20$; $\sigma_n =$ distance to 7th nearest neighbor. *Bottom plots*: value of the perplexity K for each point $\mathbf{x}_n$ of the dataset. The color corresponds to different COIL manifolds.

$\mathbb{R}^D$, consider the posterior distribution of an isotropic kernel density estimator of width σ defined on a finite set of points $\mathbf{x}_1, \dots, \mathbf{x}_N \in \mathbb{R}^D$. Thus we have a discrete distribution $p(\mathbf{x}; \sigma)$ with probabilities for $n = 1, \dots, N$

$$p_n(\mathbf{x}; \sigma) = \frac{K\left(\left\|\frac{\mathbf{x}-\mathbf{x}_n}{\sigma}\right\|^2\right)}{\sum_{k=1}^N K\left(\left\|\frac{\mathbf{x}-\mathbf{x}_k}{\sigma}\right\|^2\right)} = \frac{K\left((\frac{d_n}{\sigma})^2\right)}{\sum_{k=1}^N K\left((\frac{d_k}{\sigma})^2\right)} \quad (1)$$

where $d_n = \|\mathbf{x} - \mathbf{x}_n\|$. We focus on the case where $K(\|(\mathbf{x} - \mathbf{x}_n)/\sigma\|^2)$ is the Gaussian kernel. We set σ individually for point $\mathbf{x}$ to a value such that the entropy of the distribution $p(\mathbf{x}; \sigma)$, considered as a function of σ for fixed $d_1, \dots, d_N$, equals $\log K$, where K is a user-set perplexity parameter. The perplexity, widely used in natural language processing (Manning & Schütze, 1999), has an intuitive interpretation. A perplexity of K in a distribution p over N neighbors means p provides the same surprise as if we were to choose among K equiprobable neighbors. Having set σ in this way, the resulting value $p_n(\mathbf{x}; \sigma)$ can be used as an affinity between $\mathbf{x}$ and $\mathbf{x}_n$. We call them *entropic affinities*.

These affinities were introduced by Hinton & Roweis (2003) as a better way to define the local scaling of the Gaussian distribution in their stochastic neighbor embedding (SNE) method. Their definition of $p(\mathbf{x}; \sigma)$ was particularized to $\mathbf{x}$ being one of the data points, but our generalization simplifies things later. The affinity p_{nm} between points $\mathbf{x}_n$ and $\mathbf{x}_m$ is then $p_n(\mathbf{x}_m; \sigma)$. If we consider an affinity matrix $\mathbf{W}$ with entries $K(\mathbf{x}_n, \mathbf{x}_m)$ and degree matrix $\mathbf{D} = \operatorname{diag}(\sum_{n=1}^N w_{nm})$, then p defines the random-walk matrix $\mathbf{P} = \mathbf{D}^{-1}\mathbf{W}$, where each row is our distribution $p(\mathbf{x}_n; \sigma)$. Thus, the entropic affinities seek a matrix $\mathbf{P}(\sigma_1, \dots, \sigma_N)$ as a function of the kernel widths for each data point so that each row of $\mathbf{P}$ has perplexity K. To compute each σ_n, Hinton & Roweis (2003) performed a search to find a bracket for the solution, initialized at $[0, 1]$, and then used bisections. This becomes noticeably slow with large datasets.

Fig. 1 illustrates how the entropic affinities indeed improve over using a single σ or simple rule-of-thumb adaptive σ_n (the distance to the 7th nearest neighbor; Zelnik-Manor & Perona, 2005). We applied a nonlinear dimensionality reduction algorithm, the elastic embedding (Carreira-Perpiñán, 2010), to the COIL dataset (rotation sequences of 10 physical objects every 5 degrees, each a grayscale image of 128×128 pixels, total $N = 720$ points in 16 384 dimensions; Nene et al., 1996). The left plot clearly shows the separation between the manifolds as well as the sequential structure of each manifold. The embedding resulting from a single σ or σ_n from the 7th neighbor does not show such a structure. The bottom plots show the σ values in the latter two cases result in a wide range of perplexity values.

This paper improves our understanding of entropic affinities and their numerical computation. Section 1 proves useful properties, in particular that the function $\sigma(\mathbf{x})$ is well defined and continuously differentiable, and give simple bounds for it. Based on this, section 2 describes fast, scalable algorithms that compute σ and the entropic affinities themselves in very few iterations to almost machine precision, by processing points in a certain order. Section 3 shows experimental results with image and text datasets.

1. Some properties of entropic affinities

The entropy of the distribution (1) is defined as

$$\begin{aligned} H(\mathbf{x}, \sigma) &= -\textstyle\sum_{n=1}^N p_n(\mathbf{x}, \sigma) \log(p_n(\mathbf{x}, \sigma)) \\ &= -\textstyle\sum_{n=1}^N p_n(\mathbf{x}, \sigma) \log K_n + \log \sum_n K_n. \end{aligned} \quad (2)$$

In particular, for the Gaussian kernel it becomes

$$H(\mathbf{x}, \beta) = \beta \sum_{n=1}^N p_n(\mathbf{x}, \beta) d_n^2 + \log \sum_{n=1}^N \exp(-d_n^2 \beta) \quad (3)$$

where we will work with the precision parameter $\beta = 1/2\sigma^2$. We can express (3) and its derivatives wrt β using the partition function $Z(\beta) = \sum_{n=1}^N \exp(-d_n^2\beta)$ and moments $m_k(\beta) = \sum_{n=1}^N p_n d_n^{2k}$ as follows:

$$\begin{aligned} &H(\mathbf{x}, \beta) = \beta m_1 + \log Z \quad H'_\beta(\mathbf{x}, \beta) = -\beta(m_2 - m_1^2) \\ &H''_\beta(\mathbf{x}, \beta) = \beta(m_3 - 3m_2 m_1 + 2m_1^3) + m_1^2 - m_2. \end{aligned} \quad (4)$$

m_k can be expressed as a function of Z and its derivatives using the following recursive definition:

$$m_1 = -\tfrac{1}{Z}\tfrac{\partial Z}{\partial \beta}; \; m_{k+1} = m_1 m_k - \tfrac{\partial m_k}{\partial \beta}, \text{ for } k > 1. \quad (5)$$

We now consider the problem of searching for σ (or β), which is implicit given the perplexity K:

$$F(\mathbf{x}, \beta, K) := H(\mathbf{x}, \beta) - \log K = 0. \tag{6}$$

This is a 1D root-finding problem or an inversion problem if $H(\mathbf{x}, \beta)$ is invertible over β.

The first derivative of $F(\mathbf{x}, \beta, K)$ is equal to the one from $H(\mathbf{x}, \beta)$ and is always negative for $\beta > 0$ given that the neighboring points are not equidistant from $\mathbf{x}$. This means that (3) is a monotonically decreasing function of β that decreases from $\log N$ for $\beta = 0$ to 0 for $\beta \to \infty$. Thus, the problem (6) is well defined for any value of $\beta > 0$ and has a unique root $\beta(\mathbf{x})$ for any $K \in (0, N)$ in the same interval.

Next, notice that $H(\mathbf{x}, \beta)$ is continuously differentiable in an open neighborhood of $(\mathbf{x}_0, \beta_0)$ for some fixed $\beta_0 > 0$ and $\mathbf{x}_0 \in \mathbb{R}^D$ and that $H'_\beta(\mathbf{x}, \beta_0) \neq 0$. Thus, we can apply the implicit function theorem to show the existence of a uniquely defined local continuously differentiable function $\beta(\mathbf{x})$ that satisfies $\beta(\mathbf{x}_0) = \beta_0$. Moreover, since $H(\mathbf{x}, \beta)$ is invertible, the function $\beta(\mathbf{x})$ is also a global function defined for all $\mathbf{x}$. The same argument can be applied to $F(\mathbf{x}, \beta, K)$, leading to the existence of a continuously differentiable global function $\beta(\mathbf{x}, K)$ defined for all $K \in (0, \log N)$ and $\mathbf{x} \in \mathbb{R}^D$.

Finally, we give bounds $[\beta_L, \beta_U]$ for $\beta(\mathbf{x}, K)$, i.e., satisfying $H(\mathbf{x}, \beta_L) > \log K > H(\mathbf{x}, \beta_U)$ for every $\mathbf{x}$ and K, that are easy to compute and reasonably tight. Assume w.l.o.g. the squared distances are sorted increasingly: $d_1^2 < d_2^2 < \cdots < d_N^2$. Define $\Delta_N^2 = d_N^2 - d_1^2$ and $\Delta_2^2 = d_n^2 - d_1^2$. (If some of the distances are equal, the results hold taking Δ_2^2 as the first nonzero $d_n^2 - d_1^2$.)

Theorem 1.1. *The following give lower and upper bounds for the root of* (6)*:*

$$\beta_L = \max\left(\frac{N \log \frac{N}{K}}{(N-1)\Delta_N^2}, \sqrt{\frac{\log \frac{N}{K}}{d_N^4 - d_1^4}}\right) \tag{7}$$

$$\beta_U = \frac{1}{\Delta_2^2} \log\Big(\frac{p_1}{1-p_1}(N-1)\Big) \tag{8}$$

where p_1 is the unique solution in the interval $[3/4, 1]$ of the equation:

$$2(1-p_1)\log\frac{N}{2(1-p_1)} = \log\big(\min(\sqrt{2N}, K)\big). \tag{9}$$

The proof is given in the supplementary material. Solving (9) can be done with Newton's method and needs be done only once for all the points in the dataset, since p_1 depends only on K and N. The computation of the bounds is thus $\mathcal{O}(1)$ for each data point, since they only need d_1, d_2 and d_N. Tighter bounds can be obtained by using all distances $d_1, \ldots, d_N$, but at a cost $\mathcal{O}(N)$, which defeats the purpose.

Rescaling the data (or the distances) rescales σ as well, i.e., $H^{-1}(K; \alpha d) = \alpha H(K; d)$ for any $\alpha > 0$. This suggests rescaling the data should rescale the bounds correspondingly, which indeed happens for our bounds.

Our results carry over, suitably modified, to some variations of our problem. The formulation (1) implies $p_{nn} \neq 0$. It is also possible to set self-affinities p_{nn} to 0 (as is sometimes done) by defining p over the distances d_2 to d_N instead. One can also use sparse affinities if defining $p(\mathbf{x}; \sigma)$ on the k nearest neighbors of $\mathbf{x}$ rather than all N points. This means setting $N = k$ with points sorted in increasing distance to $\mathbf{x}$.

2. Computation of entropic affinities

To compute the entropic affinities we need to solve the root-finding problem (6) as efficiently as possible for every point in the dataset. There exist many one-dimensional root-finding algorithms with different convergence orders. Some of the most popular derivative-free methods are the bisection method, Brent's method (1973) and Ridders' method (1979). These methods have universal convergence guarantees and take as an input an interval bracketing the root, which they iteratively shrink. Derivative-based methods such as Newton's, Halley's and Euler's methods (Traub, 1982) construct a sequence of iterates instead. The next iterate is found based on the value of the function and its derivatives at the current iterate. These methods usually do not have global convergence guarantees unless the function has some very specific form (Melman, 1997). However, their convergence order is usually higher than that from derivative-free methods using the same amount of information. Notice that although the cost of (6) as well as its derivatives scales as $\mathcal{O}(N)$, the computation of the function takes about three times as long. For F we need to compute Z (exponentiation and summation over N terms) and m_1 (summation over N), but the derivatives can be computed sequentially and require calculation of just one of the m_k per derivative. Thus, it is beneficial to have as little function evaluations as possible.

We will use derivative-based root-finding methods with a simple modification so they achieve global convergence (from any starting point). We initialize the algorithm with an interval bracketing the root (obtained from the bounds (1.1)) and an initial point. The algorithm consists of two nested loops: an outer loop with the bisection method, which is slow but guarantees global convergence, and an inner loop with a derivative-based method, which is fast but only locally convergent. For each iteration of the inner loop, the algorithm evaluates the function, updates the bounds based on a new function value, computes the necessary

Algorithm 1 Root-finding framework

Input: initial β, perplexity K, distances $d_1^2, \dots, d_N^2$
compute bounds $\mathcal{B}$ using Theorem 1.1.
while *true* **do**
  for $k = 1$ **to** `maxit` **do**
    compute β using any derivative-based method
    if tolerance achieved **return** β
    if $\beta \notin \mathcal{B}$ **exit for loop**
    update $\mathcal{B}$
  end for
  compute β using bisection
  update $\mathcal{B}$
end while

derivatives and applies a derivative-based method. If the output of the method falls outside of the current brackets or the number of iterations exceeds a certain constant `maxit`, the inner loop terminates and the next point is computed using the outer bisection loop. Thus the sequence of iterates contains a subsequence of bisection steps (every `maxit` steps at most), which necessarily converges. Practically, we use a rather big value of `maxit` $= 50$, since our good initialization (see below) makes it very infrequent for the derivative-based method step to fail. Algorithm 1 shows the framework.

Finally, to avoid dealing with negative values of β and to make the function more well-behaved, we find roots over $\log \beta = -2 \log \sigma$ rather than β or σ. This modifies the expressions for the derivatives of the function (6) slightly. The final formulae are available in the supplementary material.

2.1. Choice of the root-finding algorithm

It appears that locally, close to the root, it is not essential which exact derivative-based method is used, but how many derivatives are used. Gander (1985) shows that many of the third-order methods can be described jointly using simple framework. He proves that the iterative procedure $\beta_{k+1} = \beta_k - \frac{f(\beta_k)}{f'(\beta_k)} H(t(\beta_k))$ where $t(\beta_k) = \frac{f(\beta_k) f''(\beta_k)}{f'(\beta_k)^2}$ and H is some function, is of third-order convergence if $H(0) = 1$, $H'(0) = 1/2$. Halley's method is recovered with $H(t) = (1 - \frac{1}{2}t)^{-1}$, Euler's method with $H(t) = 2(1 + \sqrt{1 - 2t})^{-1}$ and $H(t) = 1$ gives Newton's method. Traub (1982) showed that, for any $p > 1$, given $p - 1$ derivatives of the function there exists no method with convergence order higher than p. Thus, if we use only one derivative, we cannot do better than second order convergence and locally, close to the root, the behavior of Newton's method is optimal. Similarly, for two derivatives, both Euler's and Halley's methods are optimal for third-order convergence methods. The differences between methods arise mostly when the iterations are far from the root.

Among the many root-finding methods that we tried, we focus here mostly on the following three: Newton's, Euler's and Halley's method. Newton's method is a second-order method that approximates the function with a line (i.e., up to a first derivative) and the next iteration is found by the intersection of the tangent of the current point with the x-axis. Euler's and Halley's methods are third-order methods that approximate the function with a parabola and a hyperbola respectively (Scavo & Thoo, 1995). Those curves agree with the current iterate up to the first two derivatives. Fig. 2 shows a typical case of search for $\log \beta$ for a given $\log K$. We initialized the three algorithms in different places inside the interval $(-4, 3)$ and computed how many and what kind of iterations they need to find the root to an accuracy `tol` $= 10^{-10}$. Notice that close to the root, Halley's and Euler' method behave almost identically to each other, while for Newton's method the region where the number of iterations equal to 1 is a lot smaller. This is caused by a higher convergence order of the former methods compared to the latter. However, in the region far from the root, the methods behave quite differently. The initial steps of Newton's and Euler's methods are too big and send the next iterate out of the bounds, causing our algorithm to use a bisection. The region where bisection iterations occur is smaller for Euler's method compared to that of Newton's method because the parabolic approximation leads to smaller steps than the linear one. Halley's method never has to use bisection steps because the hyperbolic approximation is too conservative and uses steps that are too small to get out of the flat region.

2.2. Bounds and initialization

For the bounds, our goal is to find a region around the root that is as tight as possible and is efficient to compute from the distances. Theorem 1.1 can be applied for this. It guarantees to contain the root and it takes a constant amount of time to compute. Fig. 3 shows an example of the bounds and the entropy for a typical point in the Lena image dataset. We computed both bounds for different values of K. Notice that the bounds are quite tight and, except for the small region near the upper bound, do not include the flat, numerically challenging region of the function.

As for the initialization, we need to find a good initial iterate for the root-finding algorithm, i.e., as close as possible to the root. One way to do it is to provide precomputed initialization values directly to the algorithm. For example, we can initialize the algorithm from the middle of the bounds from Theorem 1.1, or initialize σ from the distance to the kth neighbor. But

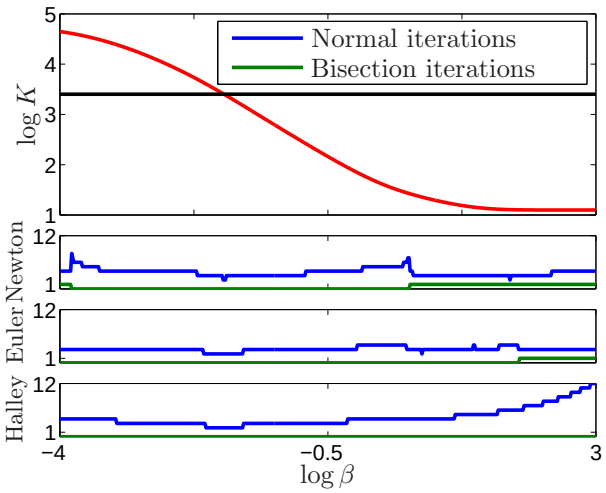

Figure 2. Top: a typical case of the entropy function $H(\mathbf{x}, \beta)$ (shown in red) for one of the points of the Lena image dataset. The desired perplexity $K = 30$ is shown as a black line. Note the function is plotted in logarithmic scale for both β and K. We bracket the root in the interval $[-4, 3]$. Below we show the number of normal and bisection iterations for different initializations needed for three different algorithms: Halley, Euler and Newton.

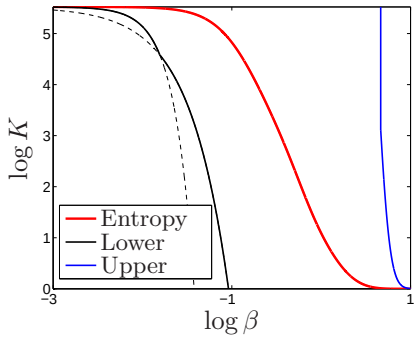

Figure 3. The entropy function (in red) and its bounds. The solid black and blue lines indicate the lower and the upper bound obtained using the formulae (7) and (8), respectively, for different values of K. The dashed black line indicates the minimum of (7).

these initializations ignore the distances to most of the points, which do affect the entropy and so the root. We also do not want to include more information in the initialization if its computational cost becomes commensurate to evaluating the entropy function itself. Instead, we propose to capitalize on the correlation that exists between β and the structure of the dataset. We can then link the points to each other based on some criterion and initialize the algorithm from the solution of the points for which β was already found. This order can be sequential or, more generally, based on a tree. In the sequential order each new point is initialized from the solution to the previous one. In the tree order, the order is not linear, but forms a directed tree (or forest in general) with each point being a node. The points are then processed in an order (such as breadth-first search) which ensures that the root of a parent node is visited before the root of its children (which are initialized from the parent). For both sequential and tree orders, each root point can be initialized, for example, from the middle of the bounds. We now describe two different strategies for choosing the order and show how they are correlated with β.

The first, *local* strategy is based on the existence and continuity of the function $\beta(\mathbf{x})$ defined in section 1. Continuous changes in $\mathbf{x}$ lead to continuous changes in β, so expect nearby points to have similar β values (except where $\beta(\mathbf{x})$ changes quickly). Therefore, we can use a local ordering of the points in the dataset. Among various orders we have explored, the one derived from a minimum spanning tree (or forest) of the dataset works well and is efficient to compute. We used Kruskal's algorithm, which takes $\mathcal{O}(N\kappa \log N)$ time to construct an MST given a κ-nearest-neighbor graph. The MST is faster to compute if using a small κ, but it should not be too small that it loses too much connectivity information. Empirically we found out that $\kappa = 10$ gives a good tradeoff and we use it for all our experiments. We observed that the choice of the root point(s) does not critically affect the results.

Our second strategy takes into account the *density* around the points. The closer the points are to $\mathbf{x}$, the smaller the distances d_k are and the bigger the entropy $H(\mathbf{x}, \beta)$ is. Therefore, for the entropy to remain constant, the resulting β must be larger in dense regions and smaller in sparser ones. Indeed, β (or σ) is related to a nonparametric density estimate of the dataset. Estimating the density in the first place is difficult, but we can use a simple estimate given by the distance from the query point $\mathbf{x}$ to its kth neighbor. Then, we can sort the points $\mathbf{x}_1, \dots, \mathbf{x}_N$ in increasing distance of its kth nearest neighbor, which gives a sequential order. As for the choice of k, we find a correlation with the desired perplexity value K: we observe empirically that the best k (which gives the best initializations) is usually approximately equal to K. We call this the $\mathcal{D}_K$ order. Note this is different from the old rule-of-thumb of setting σ directly to the distance to kth neighbor ($k = 7$ in Zelnik-Manor & Perona, 2005). We use this only to initialize the root finding.

Fig. 4 shows how β changes from point to point and the resulting $\mathcal{D}_K$ and MST orders for a dataset of 1 000 randomly generated uniformly distributed points. β changes according to our predictions above: the changes are gradual in any local neighborhood and

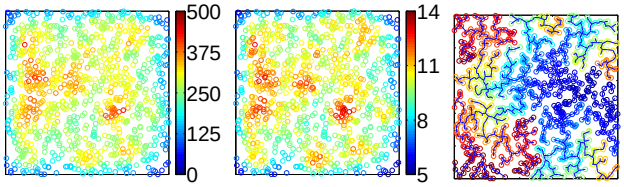

Figure 4. β computed for 1 000 uniformly distributed points with perplexity $K = 30$. The points are colored according to their corresponding β values (*left*) and their distance to the 30th nearest neighbor (*center*). *Right: MST* order. The color corresponds to the order of the points.

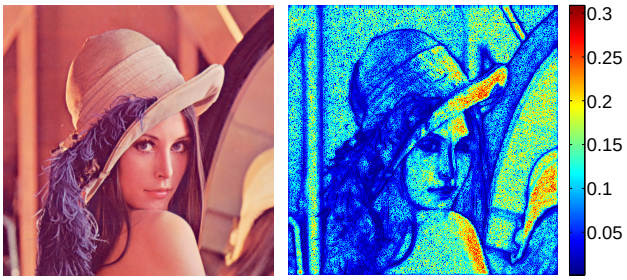

Figure 5. Lena test image and the learned β values for every pixel for a perplexity $K = 30$.

regions of similar density have similar β values.

3. Experimental evaluation

We compare several different root-finding algorithms: Ridder's, Brent's and bisection for derivative-free methods, and Euler's, Halley's and Newton's for derivative-based methods. The former group of methods were run as is without any modification. The latter group were incorporated into our globally convergent framework (algorithm 1). For those methods, we also tried several initialization choices. The *bounds* order simply initializes each point from the middle of the bracket given by theorem 1.1. This is an example of non-sequential initialization, where the order of points does not matter and points can be processed in any order. The *MST* order was used as a local-based order and the $\mathcal{D}_K$ order was used as a density-based order. In addition, we compare with two best- and worst-case orders, that give us best and worst possible performance. The *random* order of the points gives us an idea of the worst-case order. The *oracle* order processes the points in the order of their true β values, which is optimal in terms of initializing the points closest to the solution (although not practical, obviously).

All the user needs to do to obtain entropic affinities for a given dataset is to set the perplexity K and possibly their sparsity level. For all the experiments, we used $K = 30$ and a sparse distance matrix with nonzero elements corresponding to the 250 nearest neighbors. This almost did not alter the final result, because the terms in (2) responsible for farther distances are negligible due to the fast decay of the Gaussian tails. For all the algorithms we set the convergence tolerance to 10^{-10}. We chose such an accurate value because, with quadratic or cubic convergence rate, one needs very few additional iterations to achieve such accuracy, and the user need not worry about setting the tolerance. However, the tolerance should not be too close to the machine precision that oscillations occur because of precision loss. We used three datasets from very different domains: pixels of a single image, an image collection, and word-count vectors from documents.

The first dataset is the 512×512 Lena image (fig. 5 left). Each data point in this dataset is a pixel represented by spatial and range features $(i, j, L, u, v) \in \mathbb{R}^5$ where (i, j) is the pixel location in the image and (L, u, v) the pixel value in a color image (overall $N = 262\,144$ points in $D = 5$ dimensions). Note that this dataset has a very peculiar structure: the (i, j) values are independent of the image itself (as long as the size is the same). This suggests a special, *raster* order, where pixels are processed by zigzagging edge-to-edge left to right and then right to left from the top to the bottom of the image. In this order, the spatial features (i, j) vary slowly and the range features (L, u, v) vary slowly except at edges. It is an example of local order with zero computation cost.

Fig. 5(right) shows the resulting β. It preserves much information about the image, in particular edges. β tends to change gradually from pixel to pixel; small values correspond to dark regions of space and big ones to bright regions. The final β and the corresponding affinity matrix can be used for segmenting the image using mean-shift or spectral methods, for example.

For the second dataset we used 60 000 handwritten digits from the MNIST dataset. Each datapoint is a 28×28 grayscale image of a digit represented by a 784-dimensional vector. Finally, the third dataset is a subset from Grolier's encyclopedia dataset. Each datapoint represents the word count of the most popular 15 275 words from one of the 30 991 articles of the encyclopedia. Compared with the first two datasets (especially the first one), where there are reasonably densely populated areas of input space, the third dataset mostly consists of empty space. The points are located far away from each other and β values of neighboring points are not similar anymore. This will be clearly visible in the results. The affinities resulting from the MNIST and Grolier datasets can be used for visualization using dimensionality reduction, spectral clustering or semi-supervised learning, for example.

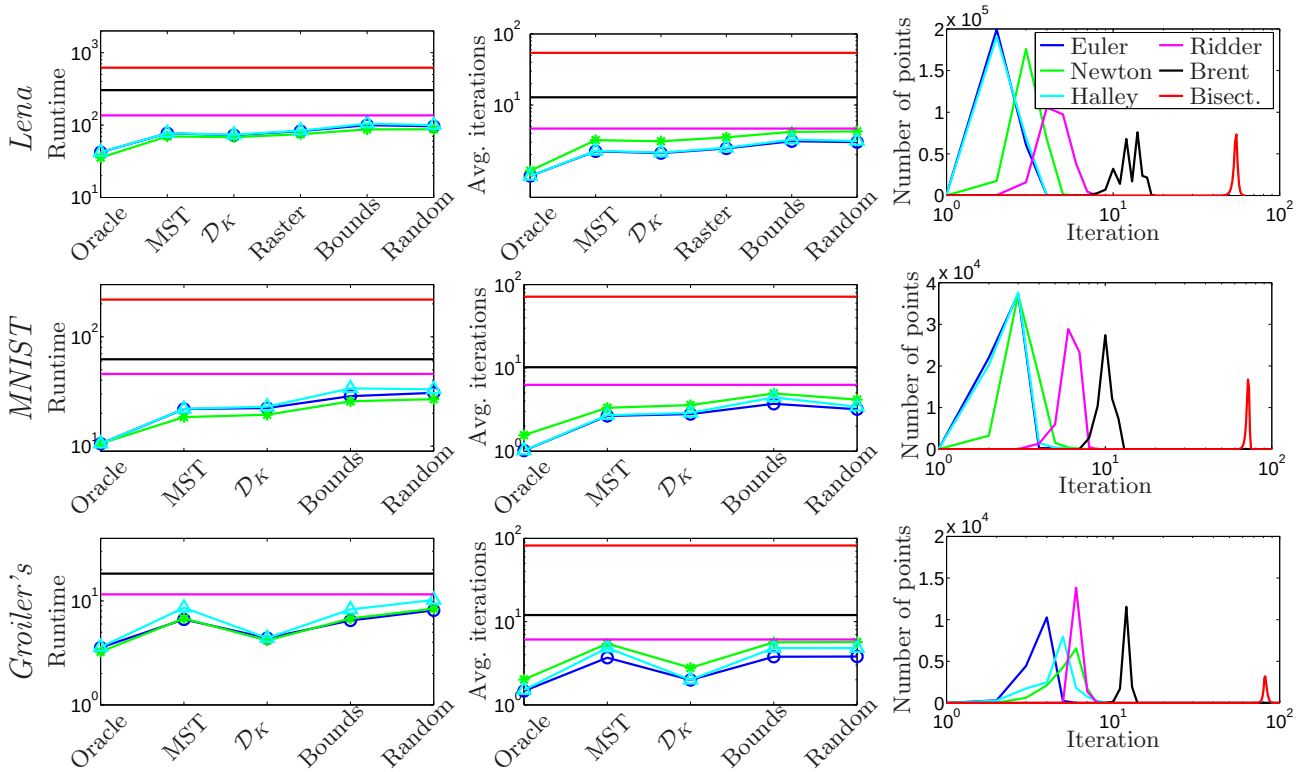

Figure 6. Runtime statistics for 3 different datasets, with 6 root-finding methods (color-coded as in the legend). *From top to bottom*: 512×512 Lena image, 60 000 MNIST digits, 30 991 articles from Grolier's encyclopedia. *First two columns*: runtime and average number of iterations for different methods with different initializations. *Right column*: number of points that converged for a given number of iterations for the *MST* order. Note the log scale in many of the plots.

For each of the datasets, we present three statistics in fig. 6: the total runtime for the different root-finding algorithms with the different initializations; the average number of iterations per point required to achieve the tolerance; and the number of the points that converged after a certain iteration count.

First of all, notice that Halley's and Euler's methods have very similar performance for all the datasets. Both methods require only two iterations for most of the points. However, there is a small difference, in particular for the *bounds* order of the MNIST dataset and for the *MST*, *bounds* and *random* orders in the Grolier dataset. The reason is the initialization in the flat region for many of the points (note that the result of those initializations is not good compared to e.g. the $\mathcal{D}_K$ order). Similar to what we see in the flat region of fig. 2, Halley's method is more conservative and moves slowly towards the solution, whereas Euler's method uses steps that are too big and retreats to bisection, which moves away from the flat region in one iteration.

Compared to the other derivative-based root-finding methods, Newton's method is a second-order method and requires slightly more iterations than Halley's or Euler's methods. However, its runtime is lower because each iteration does not need to compute the second derivative of the entropy, which costs $\mathcal{O}(N)$. For the derivative-free methods, Ridder's method is fastest, but is still approximately twice as slow as Newton's method. Brent's method and the bisection are approximately $5\times$ and $10 - 20\times$ as slow as Newton's method, respectively.

For different initializations, *MST* and $\mathcal{D}_K$ have very similar results for the MNIST and Lena datasets, with $\mathcal{D}_K$ being only slightly better (e.g. for Euler's method in the Lena dataset it takes 2.09 iterations on average for $\mathcal{D}_K$ compared to 2.22 for *MST*). However, for the Grolier dataset the *MST* order does almost as badly as the *random* order. This is due to the spatial emptiness that we described above. This does not seem to affect the $\mathcal{D}_K$ order, which is only 22% slower than the *oracle* order. The *bounds* and *random* orders perform almost identical to each other and not terribly bad, only about 1–2 iterations more than the other initializations. However, for 50% of the points, the extra iterations are the bisection iterations during the first steps of the algorithm when the initial region is flat and the root-finding methods send us away from the

bounds. This also indicates that the bounds are quite tight and one or two extra iterations are able to move very close to the root no matter where the initialization is. The *raster* order for Lena dataset does almost as good as the *MST* order, suggesting it as an fast alternative local order for image pixels. Finally, the *oracle* order achieves nearly 1 iteration per point for Euler's and Halley's method on Lena and MNIST. For example, for Euler's method only 0.1% of all the points needed 2 iterations to converge. However, in terms of the runtime the *MST* and $\mathcal{D}_K$ orders achieve a speed that is only twice as slow as the optimal one. For the Grolier dataset the average number of iterations per point is more than one even for the *oracle* order. This is because, even in the best case, the σs are not as close to each other as in the Lena and MNIST datasets.

4. Discussion

The entropic affinities can give high-quality results with many different machine learning algorithms that are based on graphs (as illustrated in fig. 1), and only require the user to set the perplexity K and possibly the sparsity level of the affinity matrix. However, up to now they have not been in widespread use outside nonlinear embedding methods such as SNE (Hinton & Roweis, 2003) or EE (Carreira-Perpiñán, 2010). Reasons for this could be the lack of a closed-form expression for the bandwidth of each point given the perplexity, and the (up to now) computational cost involved in solving for it. With our numerical algorithms, there is now very little difference between applying a closed-form formula and solving for the implicit bandwidths almost exactly. This is for two reasons. First, even if a user is able to compute bandwidths very efficiently (e.g. with a rule-of-thumb formula), computing the elements of the affinity matrix themselves is still $\mathcal{O}(N^2)$ or $\mathcal{O}(N\kappa)$ in the full and sparse case, respectively. Each iteration of our root-finding method has this same cost, but (1) we require just a few such iterations, and (2) the affinities are produced for free in our last iteration. Thus, the cost of applying the rule-of-thumb formula to compute the affinity values given the bandwidths is comparable to that of computing entropic affinities and their bandwidths. Second, we can achieve near-machine-precision at nearly no extra cost because of the high order of convergence of the root-finding methods.

The bisection algorithm used by Hinton & Roweis (2003), although slow, was not much of a problem up to now because the optimization in methods such as SNE was so costly that the number of points N was limited to a few thousands, for which the bisection time was acceptable. However, recent improvements in embedding optimization (Vladymyrov & Carreira-Perpiñán, 2012) have significantly increased the values of N that are practical: the embedding optimization takes 15 min for 20 000 MNIST images in a workstation, while the bisection-based computation of the entropic affinities takes over 20 min and becomes a bottleneck. With our algorithm, this time is reduced to 55 seconds.

Some work has used the fast Gauss transform to compute a single bandwidth parameter for kernel density estimation (KDE) (Raykar et al., 2010; Raykar & Duraiswami, 2007). These algorithms use an approximate decomposition of the computations to achieve linear runtime with N-body problems, such as KDE. As mentioned before, one needs to add the cost of computing the affinities given the bandwidth, so by Amdahl's law this reduces the speedup. The parameters of the fast Gauss transform are also hard to tune (Raykar & Duraiswami, 2007) and a bad choice can make the runtime even slower that the exact computation. More importantly, as mentioned before, in many cases a single bandwidth parameter is just not good enough. Based on our knowledge there is no work that estimates per-point σ fast, except for rules-of-thumb and cross-validation methods that are slow (Sheather, 2004; Duong & Hazelton, 2005).

5. Conclusion

By extending the entropic affinity function to the entire Euclidean space, we have been able to characterize its behavior, show that it is a well-defined function and give explicit bounds for its implicitly defined value. Based on these properties, we have analyzed different algorithms for the computational problems involved: root-finding and ordering points for best initialization. One of the best and simplest choices is a Newton-based iteration, robustified with bisection steps, using a tree- or density-based order. This achieves just above one iteration per data point on average, which is the optimally achievable performance.

Entropic affinities work better than using a single bandwidth or multiple bandwidths set with a rule of thumb, provide a random-walk matrix for a dataset, and only require a user to set the global number of neighbors. The fact that they define the scale implicitly and require an iterative computation may have prevented their widespread application, but our algorithm makes the computation scale up almost as if they were given in explicit form. Matlab code is available from the authors' web page.

Acknowledgments

Partly funded by NSF CAREER award IIS–0754089.

References

Bchir, Ouiem and Frigui, Hichem. Fuzzy relational kernel clustering with local scaling parameter learning. In *Proc. of the 2010 IEEE Int. Workshop on Machine Learning for Signal Processing (MLSP10)*, pp. 289–294, Kittilä, Finland, Aug. 29 –Sep. 1 2010.

Belkin, Mikhail and Niyogi, Partha. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15(6):1373–1396, Jun. 2003.

Belkin, Mikhail, Niyogi, Partha, and Sindhwani, Vikas. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of Machine Learning Research*, 7: 2399–2434, Nov. 2006.

Brent, Richard P. *Algorithms for Minimization without Derivatives*. Prentice-Hall, Englewood Cliffs, N.J., 1973.

Carreira-Perpiñán, Miguel Á. Fast nonparametric clustering with Gaussian blurring mean-shift. In *Proc. of the 23rd Int. Conf. Machine Learning (ICML 2006)*, pp. 153–160, Pittsburgh, PA, Jun. 25–29 2006.

Carreira-Perpiñán, Miguel Á. The elastic embedding algorithm for dimensionality reduction. In *Proc. of the 27th Int. Conf. Machine Learning (ICML 2010)*, pp. 167–174, Haifa, Israel, Jun. 21–25 2010.

de Silva, V. and Tenenbaum, Joshua B. Global versus local approaches to nonlinear dimensionality reduction. In *NIPS*, volume 15, pp. 721–728, MIT Press, Cambridge, MA, 2003.

Duong, Tarn and Hazelton, Martin L. Convergence rates for unconstrained bandwidth matrix selectors in multivariate kernel density estimation. *J. Multivariate Analysis*, 93(2):417–433, Apr. 2005.

Er, Meng Joo, Wu, Shiqian, Lu, Juwei, and Toh, Hock Lye. Face recognition with radial basis function (RBF) neural networks. *IEEE Trans. Neural Networks*, 13(3):697–710, May 2002.

Gander, Walter. On Halley's iteration method. *Amer. Math. Monthly*, 92(2):131–134, Feb. 1985.

Hinton, Geoffrey and Roweis, Sam T. Stochastic neighbor embedding. In *NIPS*, volume 15, pp. 857–864. MIT Press, Cambridge, MA, 2003.

Manning, Christopher D. and Schütze, Hinrich. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, 1999.

Melman, A. Geometry and convergence of Euler's and Halley's methods. *SIAM Review*, 39(4):728–735, Dec. 1997.

Nene, S. A., Nayar, S. K., and Murase H. Columbia object image library (COIL-20). Technical Report CUCS–005–96, Dept. of Computer Science, Columbia University, Feb. 1996.

Ng, A. Y., Jordan, M. I., and Weiss, Y. On spectral clustering: Analysis and an algorithm. In *NIPS*, volume 14, pp. 849–856. MIT Press, Cambridge, MA, 2002.

Raykar, Vikas C. and Duraiswami, Ramani. The improved fast Gauss transform with applications to machine learning. In *Large Scale Kernel Machines*, Neural Information Processing Series, pp. 175–202. MIT Press, 2007.

Raykar, Vikas C., Duraiswami, Ramani, and Zhao, Linda H. Fast computation of kernel estimators. *Journal of Computational and Graphical Statistics*, 19(1):205–220, 2010.

Ridders, C. J. F. A new algorithm for computing a single root of a real continuous function. *IEEE Trans. Circuits and Systems*, 26(11):979–980, Nov. 1979.

Roweis, Sam T. and Saul, Lawrence K. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, Dec. 22 2000.

Scavo, T. R. and Thoo, J. B. On the geometry of Halley's method. *Amer. Math. Monthly*, 102(5):417–426, May 1995.

Sheather, Simon J. Density estimation. *Statistical Science*, 19(4):588–597, Nov. 2004.

Traub, J. F. *Iterative Methods for the Solution of Equations*. Prentice-Hall, second edition, 1982.

Vladymyrov, Max and Carreira-Perpiñán, Miguel Á. Partial-Hessian strategies for fast learning of nonlinear embeddings. In *Proc. of the 29th Int. Conf. Machine Learning (ICML 2012)*, pp. 345–352, Edinburgh, Scotland, Jun. 26 – July 1 2012.

Zelnik-Manor, Lihi and Perona, Pietro. Self-tuning spectral clustering. In *NIPS*, volume 17, pp. 1601–1608. MIT Press, Cambridge, MA, 2005.

Zhou, Dengyong, Bousquet, Olivier, Lal, Thomas N., Weston, Jason, and Schölkopf, Bernhard. Learning with local and global consistency. In *NIPS*, volume 16, pp. 321–328. MIT Press, Cambridge, MA, 2004.