Entropic Affinities: Properties and Efficient Numerical Computation

Max Vladymyrov and Miguel Carreira-Perpiñán

Electrical Engineering and Computer Science
University of California, Merced
http://eecs.ucmerced.edu

June 18, 2013
Summary

- The entropic affinities define affinities so that each point has an effective number of neighbors equal to $K$.

- First introduced in:

- Not in a widespread use, even though they work well in a range of problems.

- We study some properties of entropic affinities and give fast algorithms to compute them.
Affinity matrix

Defines a measure of similarity between points in the dataset.

Used in:
- Dimensionality reduction:
  - Stochastic Neighbor Embedding, t-SNE, Elastic Embedding, Laplacian Eigenmaps.
- Clustering:
  - Mean-Shift, Spectral clustering.
- Semi-supervised learning.
- and others

The performance of the algorithms depends crucially of the affinity construction, govern by the bandwidth $\sigma$.

Common practice to set $\sigma$:
- constant,
- rule-of-thumb (e.g. distance to the 7th nearest neighbor, Zelnik & Perona, 05).
Motivation: choice of $\sigma$

COIL-20: Rotations of objects every 5°; input are greyscale images of $128 \times 128$.

Affinity matrices:

- Constant sigma
- Rule-of-thumb: Dist. to the 7th nn (Zelnik & Perona, 05)
- Entropic affinities
Motivation: choice of $\sigma$

COIL-20: Rotations of objects every 5°; input are greyscale images of $128 \times 128$.

Dimensionality Reduction with Elastic Embedding algorithm:

- **Constant sigma**
- **Rule-of-thumb:** Dist. to the 7th nn (Zelnik & Perona, 05)
- **Entropic affinities**
Search for good $\sigma$

Good $\sigma$ should be:

- Set separately for every data point.
- Take into account the whole distribution of distances.
Entropic affinities

In the entropic affinities, the $\sigma$ is set individually for each point such that it has a distribution over neighbors with fixed perplexity $K$ (Hinton & Rowies, 2003).

- Consider a distribution of the neighbors $\mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathbb{R}^D$ for $\mathbf{x} \in \mathbb{R}^D$:
  \[
p_n(\mathbf{x}; \sigma) = \frac{K(\| (\mathbf{x} - \mathbf{x}_n) / \sigma \|^2)}{\sum_{k=1}^N K(\| (\mathbf{x} - \mathbf{x}_k) / \sigma \|^2)}
  \]

- The entropy of the distribution is defined as
  \[
  H(\mathbf{x}, \sigma) = - \sum_{n=1}^N p_n(\mathbf{x}, \sigma) \log(p_n(\mathbf{x}, \sigma))
  \]

- Consider the bandwidth $\sigma$ (or precision $\beta = \frac{1}{2\sigma^2}$) given the perplexity $K$:
  \[
  H(\mathbf{x}, \beta) = \log K
  \]

- Perplexity of $K$ in a distribution $p$ over $N$ neighbors provides the same surprise as if we were to choose among $K$ equiprobable neighbors.

- We define entropic affinities as probabilities $p = (p_1, \ldots, p_N)$ for $\mathbf{x}$ with respect to $\beta$. Those affinities define a random walk matrix.
Entropic affinities: example
Entropic affinities: properties

\[ H(x_n, \beta_n) \equiv - \sum_{n=1}^{N} p_n(x_n, \beta_n) \log(p_n(x_n, \beta_n)) = \log K \]

- This is a 1D root-finding problem or an inversion problem \( \beta_n = H_{x_n}^{-1}(\log K) \).
- Should be solved for \( x_n \in x_1, \ldots, x_N \).
- We can prove that:
  - The root-finding problem is well defined for a Gaussian kernel for any \( \beta_n > 0 \), and has a unique root for any \( K \in (0, N) \).
  - The inverse is a uniquely defined continuously differentiable function for all \( x_n \in \mathbb{R}^N \) and \( K \in (0, N) \).
Entropic affinities: bounds

The bounds $[\beta_L, \beta_U]$ for every $K \in (0, N)$ and $x_n \in \mathbb{R}^N$:

$$\beta_L = \max \left( \frac{N \log \frac{N}{K}}{(N - 1) \Delta_2^2}, \sqrt{\log \frac{N}{K}} \sqrt{d_N^4 - d_1^4} \right),$$

$$\beta_U = \frac{1}{\Delta_2^2} \log \left( \frac{p_1}{1 - p_1} (N - 1) \right),$$

where $\Delta_2^2 = d_2^2 - d_1^2$, $\Delta_N^2 = d_N^2 - d_1^2$, and $p_1$ is a unique solution of the equation

$$2(1 - p_1) \log \frac{N}{2(1-p_1)} = \log \left( \min(\sqrt{2N}, K) \right)$$

The bounds are computed in $\mathcal{O}(1)$ for each point.
Entropic affinities: computation

For every $x_n \in x_1, \ldots, x_N$

$$H(x_n, \beta_n) = \log K$$

1. Initialize $\beta_n$ as close to the root as possible.
2. Compute the root $\beta_n$. 
I. Computation of $\beta_n$; the root-finding

<table>
<thead>
<tr>
<th>Methods</th>
<th>Convergence order</th>
<th>Derivatives order</th>
<th>Number of $\mathcal{O}(N)$ evaluations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Derivative-free</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bisection</td>
<td>linear</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Brent</td>
<td>linear</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Ridder</td>
<td>quadratic</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Derivative-based</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Newton</td>
<td>quadratic</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Halley</td>
<td>cubic</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Euler</td>
<td>cubic</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

- The cost of the objective function evaluation and each of derivative is $\mathcal{O}(N)$.
- Derivative-free methods above generally converge globally. They work by iteratively shrinking an interval bracketing the root.
- Derivative-based methods have higher convergence order, but may diverge.
Robustified root-finding algorithm

• We embed the derivative-based algorithm into bisection loop for global convergence.
• We run the following algorithm for each $x_n \in x_1, \ldots, x_N$

**Input:** initial $\beta$, perplexity $K$, distances $d_1^2, \ldots, d_N^2$, bounds $B$.

```plaintext
while true do
  for $k = 1$ to maxit do
    compute $\beta$ using a derivative-based method
    if tolerance achieved return
    if $\beta \notin B$ exit for loop
    update $B$
  end for
  compute $\beta$ using bisection
  update $B$
end while
```

• We embed the derivative-based algorithm into bisection loop for global convergence.
• We run the following algorithm for each $x_n \in x_1, \ldots, x_N$
Robustified root-finding algorithm

• We embed the derivative-based algorithm into bisection loop for global convergence
• We run the following algorithm for each $x_n \in X_1, \ldots, X_N$

**Input:** initial $\beta$, perplexity $K$, distances $d_{1}^{2}, \ldots, d_{N}^{2}$, bounds $B$.

```plaintext
while true do
    for $k = 1$ to maxit do
        compute $\beta$ using a derivative-based method
        if tolerance achieved return
        if $\beta \notin B$ exit for loop
        update $B$
    end for
    compute $\beta$ using bisection
    update $B$
end while
```

![Diagram showing bisection: step is outside the brackets](image.png)
Robustified root-finding algorithm

• We embed the derivative-based algorithm into bisection loop for global convergence
• We run the following algorithm for each $x_n \in x_1, \ldots, x_N$

**Input:** initial $\beta$, perplexity $K$, distances $d^2_1, \ldots, d^2_N$, bounds $B$.

```plaintext
while true do
    for $k = 1$ to maxit do
        compute $\beta$ using a derivative-based method
        if tolerance achieved return
        if $\beta \notin B$ exit for loop
        update $B$
    end for
    compute $\beta$ using bisection
    update $B$
end while
```

- We embed the derivative-based algorithm into bisection loop for global convergence
- We run the following algorithm for each $x_n \in x_1, \ldots, x_N$
Robustified root-finding algorithm

- We embed the derivative-based algorithm into bisection loop for global convergence.
- We run the following algorithm for each $x_n \in X_1, \ldots, X_N$

**Input:** initial $\beta$, perplexity $K$, distances $d_1^2, \ldots, d_N^2$, bounds $B$.

while true do
  for $k = 1$ to maxit do
    compute $\beta$ using a derivative-based method
    if tolerance achieved return
    if $\beta \notin B$ exit for loop
    update $B$
  end for
  compute $\beta$ using bisection
  update $B$
end while
Robustified root-finding algorithm

- We embed the derivative-based algorithm into bisection loop for global convergence
- We run the following algorithm for each $x_n \in x_1, \ldots, x_N$

**Input:** initial $\beta$, perplexity $K$, distances $d_1^2, \ldots, d_N^2$, bounds $B$.

```python
while true do
    for $k = 1$ to maxit do
        compute $\beta$ using a derivative-based method
        if tolerance achieved return
            if $\beta \notin B$ exit for loop
        update $B$
    end for
    compute $\beta$ using bisection
    update $B$
end while
```

• We embed the derivative-based algorithm into bisection loop for global convergence
• We run the following algorithm for each $x_n \in x_1, \ldots, x_N$
2. Initialization of $\beta_n$

1. Simple initialization:
   - midpoint of the bounds,
   - distance to $k$th nearest neighbor.
   Typically far from root and require more iterations.

2. Each new $\beta_n$ is initialized from the solution to its predecessor:
   - sequential order;
   - tree order.

We need to find orders that are correlated with the behavior of $\beta$. 
2. Initialization of $\beta_n$

1. Simple initialization:
   - middle of the bounds,
   - distance to $k$th nearest neighbor.
   
   Typically far from root and require more iterations.

2. Each new $\beta_n$ is initialized from the solution to its predecessor:
   - sequential order;
   - tree order.

We need to find orders that are correlated with the behavior of $\beta$. 
Sequential or tree order

- $D_k$, density strategy: for the fixed entropy value, $\beta$ is larger in dense regions and smaller in sparser ones.
  - Use nearest neighbor density estimate.
  - $\beta_n$ is proportional to the distance to $k$th nearest neighbor of $x_n$.

- MST, local strategy: nearby points have similar $\beta$ values.
  - Build a MST around the data.
  - Process the points in level-order, so parents are solved for before children.
Experimental evaluation: setup

We set the perplexity to $K = 30$ and the tolerance to $10^{-10}$.

Initializations:

- "oracle": processes the points in the order of their true $\beta$ values,
- MST: local-based order,
- $D_K$: density-based order,
- bounds: initialize from the midpoint of the bounds,
- random: initialize from one of $x_n$ chosen at random.

Root-finding methods:

- Derivative-free: Bisection, Brent, Ridder.
- Derivative-based: Newton, Euler, Halley.
Experimental evaluation: Lena

Bisection: > 10 min.
Our method: 1 min.
Computing just the affinities given $\beta$s: 20 s.
Experimental evaluation: image

512 × 512 Lena image. Each data point is a pixel represented by spatial and range features $(i, j, L, u, v) \in \mathbb{R}^5$:
• $(i, j)$ is the pixel location;
• $(L, u, v)$ the pixel value.

$N = 262,144$ points, $D = 5$ dimensions

---

**Number of iterations**

- Oracle
- MST
- $D_K$
- Raster
- Bounds
- Random

**Runtime**

- Oracle
- MST
- $D_K$
- Raster
- Bounds
- Random

**Number of points converged after $i$ iterations**

- Euler
- Newton
- Halley
- Ridder
- Brent
- Bisection

---

$N = 262,144$ points, $D = 5$ dimensions
Experimental evaluation: digits

60,000 handwritten digits from the MNIST dataset. Each datapoint is a $28 \times 28$ grayscale image.

$N = 60,000$ points, $D = 784$ dimensions
Experimental evaluation: text

Articles from Grolier’s encyclopedia. Each point is a word count of the most popular 15,275 words from 30,991 articles.

\[ N = 30,991 \text{ points}, D = 15,275 \text{ dimensions} \]
Conclusions

• We studied the behavior of entropic affinities and their properties.
• Search for the affinities involves finding the root of non-linear equation.
• We can find the root almost to machine precision in just over one iteration per point on average using:
  ‣ bounds for the root,
  ‣ root-finding methods with high-order convergence,
  ‣ warm-start initialization based on local or density orders.
• In applications such as spectral clustering and embeddings, semi-supervised learning using entropic affinities should give better results than fixing the bandwidth to a single value or using a rule-of-thumb.
• The only user parameter is the global perplexity value $K$.
• MATLAB code online at http://eecs.ucmerced.edu. Run it simply like $[W,s] = ea(X,K)$. 