MULTIPlicative UPDATE FOR FAST OPTIMIZATION OF INFORMATION RETRIEVAL BASED NEIGHBOR EMBEDDING

Jaakko Peltonen and Ziyuan Lin

Helsinki Institute for Information Technology HIIT, Department of Information and Computer Science, Aalto University, P.O. Box 15400, FI-00076 Aalto, Finland
\{jaakko.peltonen, ziyuan.lin\}@aalto.fi

ABSTRACT

Dimensionality reduction of high-dimensional data for visualization has recently been formalized as an information retrieval task where original neighbors of data points are retrieved from the low-dimensional display, and the visualization is optimized to maximize flexible tradeoffs between precision and recall of the retrieval, avoiding misses and false neighbors. The approach has yielded well-performing visualization methods as well as information retrieval interpretations of earlier neighbor embedding methods. However, most of the methods are based on slow gradient search approaches, whereas fast methods are crucial for example in interactive applications. In this paper we propose a fast multiplicative update rule for visualization optimized for information retrieval, and show in experiments it yields equally good results as the previous state of the art gradient based approach but much faster.

Index Terms—dimensionality reduction, visualization, information retrieval, multiplicative update

1. INTRODUCTION

Visualization of high-dimensional data is a prominent application of nonlinear dimensionality reduction (NLDR) methods. Visualization is crucial especially in the first stages of data analysis when strong hypotheses about the data are not yet available. Numerous NLDR methods have been developed [1, 2, 3, 4, 5, 6, 7, 8, 9]. However, many NLDR methods are based on manifold learning and are designed to unfold an underlying manifold of data, and are not designed to reduce data dimensionality beyond the dimensionality of the manifold; thus they perform poorly in low-dimensional visualization where all original properties cannot be represented perfectly on the low-dimensional display and therefore minimization of the errors that necessarily happen is essential.

A recent well-performing NLDR approach for visualization formalizes visualization by scatter plots as an information retrieval task [10], where original neighbors are retrieved from the low-dimensional display, and the task of the visualization method is to minimize two kinds of errors, misses and false positives, that happen in such retrieval, which is shown to be equivalent to optimizing recall and precision of the retrieval. The resulting formalism yielded rigorous ways to quantitatively measure quality of visualizations by their performance in the information retrieval task, information retrieval interpretations of several recent neighbor embedding methods, as well as a new well-performing visualizer called the Neighbor Retrieval Visualizer (NeRV; [10]) which optimizes flexible tradeoffs between precision and recall. In comparisons NeRV has yielded state of the art performance in visualization, outperforming other recent NLDR methods [10].

NeRV and other neighbor embedding methods having an information retrieval interpretation [1, 2] have been proposed using gradient search based optimization for their respective cost functions, requiring numerous expensive evaluations of the cost functions and gradients to find the local optimum. Fast optimization is essential when working with large data sets, or when working in interactive scenarios [11] where fast response to user input is crucial.

In Section 2 we recap the principle of information retrieval based visualization and the Neighbor Retrieval Visualizer (NeRV). Section 3 introduces our multiplicative fixed-point update rule for NeRV. In Section 4 we show by experimental comparisons on several data sets that the multiplicative updates yield equal quality state of the art performance as the original NeRV, outperforming several other NLDR methods, and we show it yields the results clearly faster than original NeRV. Section 5 concludes the paper.

2. INFORMATION RETRIEVAL PERSPECTIVE TO DIMENSIONALITY REDUCTION FOR VISUALIZATION

We consider visualization by scatter plots. Let \( \{x_i\}_{i=1}^N \) be the set of \( N \) input data samples to be visualized. The task...
of a visualization method is to find low-dimensional output coordinates \( \{y_i\}_{i=1}^N \) for each data point on the display which is typically two- or three-dimensional.

Let each sample \( i \) have an input neighborhood defined as a distribution \( p_i = \{p_{ij}\} \) telling for each neighbor \( j \) the probability \( p_{ij} \) that \( j \) would be chosen as a neighbor for \( i \). A simple definition is

\[
p_{ij} = \frac{\exp \left( -d^2(x_i, x_j)/(2\sigma_i^2) \right)}{\sum_{k \neq i} \exp \left( -d^2(x_i, x_k)/(2\sigma_i^2) \right)} \tag{1}
\]

where \( d(\cdot, \cdot) \) is a distance measure between two input data samples. Here \( \sigma_i^2 \) controls the size of the neighborhood around point \( i \), which can be set to make the entropy of the distribution \( p_i \) equal to \( \log k \) where \( k \) can be seen as an upper limit of the number of relevant neighbors.

On the display an output neighborhood can be defined around each sample, as the distribution \( q_i = \{q_{ij}\} \) where

\[
q_{ij} = \frac{\exp \left( -\|y_i - y_j\|^2/(2\sigma_j^2) \right)}{\sum_{k \neq j} \exp \left( -\|y_i - y_k\|^2/(2\sigma_j^2) \right)} \tag{2}
\]

is the probability that an analyst starting from a central point \( i \) picks the neighbor \( j \) to be inspected.

All properties of a high-dimensional data set cannot be represented on a low-dimensional scatter plot. Two kinds of errors will happen, shown in Fig. 1: misses are true neighbors of a point \( i \) (high \( p_{ij} \)) that are not neighbors on the display (low \( q_{ij} \)), and false neighbors are neighbors on the display (high \( q_{ij} \)) that are not true neighbors (low \( p_{ij} \)). Misses and false neighbors can have a cost to the analyst; good visualizations should be optimized to minimize the total cost of errors.

Minimizing the total cost of misses has been shown to correspond to optimizing the information retrieval measure recall, and minimizing total cost of false neighbors corresponds to optimizing precision [10]. In particular, it can be shown that the Kullback-Leibler divergence \( D_{KL}(p_i, q_i) = \sum_{j \neq i} p_{ij} \log \frac{p_{ij}}{q_{ij}} \) generalizes recall and \( D_{KL}(q_i, p_i) = \sum_{j \neq i} q_{ij} \log \frac{q_{ij}}{p_{ij}} \) generalizes precision. To measure quality of an entire visualization we take the simple average \( \mathbb{E}_t[D_{KL}(p_i, q_i)] \) over query points, denoted mean smoothed recall, and the average \( \mathbb{E}_t[D_{KL}(q_i, p_i)] \), denoted mean smoothed precision. The total cost of errors is proportional to

\[
J = \lambda \mathbb{E}_t[D_{KL}(p_i, q_i)] + (1 - \lambda) \mathbb{E}_t[D_{KL}(q_i, p_i)] \tag{3}
\]

where \( \lambda \) controls the tradeoff between recall and precision (cost of misses vs. false neighbors); \( \lambda \) tells whether the analyst prefers to avoid misses or false neighbors.

Equation (3) is a rigorous measure of the performance of a visualization in the neighbor retrieval task of retrieving neighbors; moreover, as it is a smooth function of the output coordinates, it can be used as a cost function for an NLDR method. The Neighbor Retrieval Visualizer (NeRV; [10]) optimizes visualizations by minimizing the cost (3) with respect to the output coordinates by gradient methods, thus it optimizes visualizations for flexible tradeoffs between precision and recall. It can be shown Stochastic Neighbor Embedding [1] and its t-distributed version [2] are special cases of the information retrieval approach that only optimize recall. Next we give a fast multiplicative update rule for NeRV.

### 3. MULTIPLICATIVE UPDATE RULE FOR NERV

The NeRV method optimizes visualizations by minimizing (3) using conjugate gradient descent, with a golden section search along each conjugate direction. As a detail, the conjugate gradient search is repeated several times shrinking neighborhood sizes \( \sigma_i \) from wide initial values towards the final value, to help reach better local optima. Other neighbor embedding methods involve similar search strategies.

As gradient search is iterative and involves numerous cost function and gradient evaluations, it can take a long time to complete, which can hinder applying the method to large data sets or to interactive applications where fast response to user feedback can be crucial. We next propose a multiplicative fixed-point update rule for NeRV which finds good visualizations faster than the previous conjugate gradient search.

In addition to the approach proposed here, methods have been proposed to compute the cost and gradient approximatively to reducing its quadratic computational complexity \( O(N^2) \) to \( O(N \log N) \) [12, 13, 14]; such methods speed up the cost function and gradient computation but do not essentially change the optimization procedure, and they are compatible with our approach which reduces the number of evaluations needed due to the fast multiplicative optimization.

#### 3.1. The General idea of Multiplicative Update

A typical gradient descent based iterative update rule has the form

\[
z_{id}^{(t+1)} = z_{id}^{(t)} - \alpha \nabla z_{id} J(z^{(t)})
\]

where \( J \) is a cost func-
tion to be minimized with respect to a set of parameters. We
index the parameters with two indices so that \( z_{id}^{(t)} \) is the value
of the \((i, d)\)-th parameter at the \(t\)-th iteration, \( z^{(t)} \) is the set
of all parameter values at the \(t\)-th iteration, \( \nabla_{id}\mathcal{J}^{(t)} \) is the
derivative of \( \mathcal{J} \) with respect to the \((i, d)\)-th parameter evalu-
atied at \( z^{(t)} \), and \( \alpha_{id} \) is the learning rate or step size. This is
sometimes also called an additive update. In the additive rule
the learning rates \( \alpha_{id} \) must be set, typically by choosing them
manually beforehand, or by applying a line search strategy at
each iteration as done in the original NeRV.

Instead of such additive updates, we now create a multipli-
cative update rule which does not require a user-assigned
learning rate parameter or a line search: we first derive it
in general form and then apply it to the NeRV cost func-
tion. We start by decomposing the gradient of the cost func-
tion into a positive and negative part, so that \( \nabla_{id}\mathcal{J}^{(t)} = \nabla_{id}^{+}\mathcal{J}^{(t)} - \nabla_{id}^{-}\mathcal{J}^{(t)} \). Here \( \nabla_{id}^{+}\mathcal{J}^{(t)} \) and \( \nabla_{id}^{-}\mathcal{J}^{(t)} \) are
nonnegative terms that respectively contain the positive par-
t and negative part of \( \nabla_{id}\mathcal{J}^{(t)} \), as detailed in the next
section. Given the decomposition of the gradient, the multipli-
cative update rule is obtained by generalizing the learning rate \( \alpha_{id} \) to be different for each parameter \((i, d)\), and taking
the choice \( \alpha_{id} = \frac{z_{id}^{(t)}}{\nabla_{id}\mathcal{J}^{(t)}} \). This yields the rule
\[ z_{id}^{(t+1)} = z_{id}^{(t)} \frac{\nabla_{id}^{+}\mathcal{J}^{(t)}}{\nabla_{id}^{+}\mathcal{J}^{(t)}} \] (4)

This rule has the same stationary points as the additive rule:
\( \nabla_{id}\mathcal{J}^{(t)} = 0 \) implies \( \nabla_{id}^{+}\mathcal{J}^{(t)} = 0 \) and thus \( \nabla_{id}^{-}\mathcal{J}^{(t)} = 0 \) whenever the denominator is nonzero
which can be ensured by adding a very small regularization constant
to the numerator and denominator.

3.2. Details of Multiplicative Update Rule for NeRV

Equation (4) preserves nonnegativity (or nonpositivity) of pa-
ter values. To make the update rule suitable for optimiz-
ing display coordinates of data points in NeRV, we set
\( z_{id} = \exp(y_{id}) \) in (4), where \( y_{id} \) is the \(d\)-th output coordinate
(dimension) of the \(i\)-th data point. This yields
\[ \nabla_{id}\mathcal{J} = \frac{\partial \mathcal{J}}{\partial y_{id}} = \frac{\partial \mathcal{J}}{\partial y_{id}} \frac{1}{z_{id}} \] (5)

By straightforward arithmetic it can be shown that
\[ \frac{\partial \mathcal{J}}{\partial y_{id}} = \left( \sum_{j \neq i} D_{ij} q_{ij} \right) \left( \sum_{j \neq i} \frac{y_{id} - y_{jd}}{\sigma_{i}^{2}} \right) \]
\[ + \sum_{j \neq i} \left( w_{j} - D_{j,i} \right) q_{ij} \frac{y_{id} - y_{jd}}{\sigma_{i}^{2}} - \sum_{j \neq i} D_{ij} q_{ij} \frac{y_{id} - y_{jd}}{\sigma_{i}^{2}} \]
\[ = \sum_{j \neq i} \left[ (w_{i} - D_{i,j}) q_{ij} \frac{y_{id} - y_{jd}}{\sigma_{i}^{2}} + (w_{j} - D_{j,i}) \frac{q_{ij}}{\sigma_{j}} \right] (y_{id} - y_{jd}) \]
\[ (6) \]

where
\[ D_{i,j} = (1 - \lambda) \left( \log q_{j|i} - \log p_{j|i} + 1 \right) - \lambda \frac{p_{j|i}}{q_{j|i}} \] (7)
\[ w_{i} = \sum_{j \neq i} D_{i,j} q_{ij} \] (8)

We next rewrite the gradient into a form where we can
separate the positive and negative components. For simplicity
we use the following notation:
\[ K_{ij} = \frac{1 - \lambda}{\sigma_{i}^{2}}\lambda \] (6)
\[ k_{ij} = \frac{1 - \lambda}{\sigma_{i}^{2}} q_{ij} \log \frac{q_{ij}}{p_{ij}} \]
\[ p_{ij} = \frac{\lambda}{\sigma_{j}^{2}} \]
\[ Q_{ij} = \frac{\lambda}{\sigma_{j}^{2}} \]

By plugging (7) and (8) into the summands of (6), the gradient
(6) can be rewritten by straightforward arithmetic steps as
\[ \frac{\partial \mathcal{J}}{\partial y_{id}} = \sum_{j \neq i} \left[ \left( (K_{ij} - k_{ij})^{+} + (P_{ij} - Q_{ij})^{+} \right) \left( y_{id}^{+} + y_{jd}^{+} \right) \\
+ \left( (K_{ij} - k_{ij})^{-} + (P_{ij} - Q_{ij})^{-} \right) \left( y_{id}^{-} + y_{jd}^{-} \right) \right] \]
\[ - \sum_{j \neq i} \left[ \left( (K_{ij} - k_{ij})^{+} + (P_{ij} - Q_{ij})^{+} \right) \left( y_{id}^{+} + y_{jd}^{+} \right) \\
+ \left( (K_{ij} - k_{ij})^{-} + (P_{ij} - Q_{ij})^{-} \right) \left( y_{id}^{-} + y_{jd}^{-} \right) \right] \]
\[ = \left( \frac{\partial \mathcal{J}}{\partial y_{id}} \right)^{\text{positive}} - \left( \frac{\partial \mathcal{J}}{\partial y_{id}} \right)^{\text{negative}} \] (9)

To decompose the derivative into two nonnegative parts
\( \left( \frac{\partial \mathcal{J}}{\partial y_{id}} \right)^{\text{positive}} \) and \( \left( \frac{\partial \mathcal{J}}{\partial y_{id}} \right)^{\text{negative}} \) we further rewrite the derivative as
\[ \frac{\partial \mathcal{J}}{\partial y_{id}} = \sum_{j \neq i} \left[ \left( (K_{ij} - k_{ij})^{+} + (P_{ij} - Q_{ij})^{+} \right) \left( y_{id}^{+} + y_{jd}^{+} \right) \\
+ \left( (K_{ij} - k_{ij})^{-} + (P_{ij} - Q_{ij})^{-} \right) \left( y_{id}^{-} + y_{jd}^{-} \right) \right] \]
\[ - \sum_{j \neq i} \left[ \left( (K_{ij} - k_{ij})^{+} + (P_{ij} - Q_{ij})^{+} \right) \left( y_{id}^{+} + y_{jd}^{+} \right) \\
+ \left( (K_{ij} - k_{ij})^{-} + (P_{ij} - Q_{ij})^{-} \right) \left( y_{id}^{-} + y_{jd}^{-} \right) \right] \]
\[ = \left( \frac{\partial \mathcal{J}}{\partial y_{id}} \right)^{\text{positive}} - \left( \frac{\partial \mathcal{J}}{\partial y_{id}} \right)^{\text{negative}} \] (10)

where we denote \( (a)^{+} = (a + a)/2 \) and \( (a)^{-} = (a - a)/2 \) for any variable \( a \). By (5), \( \left( \frac{\partial \mathcal{J}}{\partial y_{id}} \right)^{\text{positive}} / z_{id} \) and
\( \left( \frac{\partial \mathcal{J}}{\partial y_{id}} \right)^{\text{negative}} / z_{id} \) are the positive and negative parts of
\( \nabla_{id}\mathcal{J} \). Inserting the result into (4), we have the final multipli-
cative update rule as
\[ z_{id}^{(t+1)} = z_{id}^{(t)} \frac{\left( \frac{\partial \mathcal{J}}{\partial y_{id}} \right)^{\text{negative}}}{\left( \frac{\partial \mathcal{J}}{\partial y_{id}} \right)^{\text{positive}}} \]
\[ = z_{id}^{(t)} \frac{\sum_{j \neq i} A_{ij} Y_{ij} - A_{ij} Y_{ij}^{d+}}{\sum_{j \neq i} A_{ij} Y_{ij}^{d+} + A_{ij} Y_{ij}^{d-}} \] (11)
where

\[ A_{ij}^+ = (KL_{ij} - kl_{ij})^+ + (P_{ij} - Q_{ij})^+ + (KL_{ji} - kl_{ji})^+ + (P_{ji} - Q_{ji})^+ , \]
\[ A_{ij}^- = (KL_{ij} - kl_{ij})^- + (P_{ij} - Q_{ij})^- + (KL_{ji} - kl_{ji})^- + (P_{ji} - Q_{ji})^- , \]
\[ Y_{ij}^{d+} = (y_{id}^+ + y_{jd}^-), \quad Y_{ij}^{d-} = (y_{id}^- + y_{jd}^+) \]

and the output coordinates can be recovered after the update simply as \( y_{id}^{(t+1)} = \log(z_{id}^{(t+1)}) \). Note that (11) does not need a step size parameter.

A similar approach was taken in [15] to propose a multiplicative update for t-SNE; Stochastic Neighbor Embedding can be seen as a special case of NeRV optimizing recall only (corresponding to \( \lambda = 1 \) in NeRV), and t-SNE is the corresponding special case of t-distributed NeRV [10].

We briefly mention that several terms in the update rule have information retrieval interpretations: it can be shown that \( KL_{ij} = \sum_{km} P(y_i=k) P(y_m=j) \cdot KL(p_{ij}, q_{ij}) \) measures whether neighbor \( j \) contributed a greater proportion of the total false neighbor cost for query point \( i \) than a proportion based only on the output probability \( q_{ij} \), weighted by neighborhood size. Terms \( P_{ij} - Q_{ij} \) and \( KL_{ji} - kl_{ji} \) indicate the worst false neighbors of \( i \). Similarly, \( P_{ij} - Q_{ij} \) is positive if neighbor \( j \) is missed for point \( i \) (higher true neighborhood probability than output probability), weighted by the cost of misses and the neighborhood size. The multiplicative update then pushes \( y_{id} \) away from the worst false neighbors of \( i \) and towards missed neighbors.

A theoretical convergence analysis for the algorithm is a direction of future work. The algorithm does not decrease cost in each step, but over several iterations it empirically minimizes the cost function and yields good visualizations on all our data sets as shown in our experiments.

4. EXPERIMENTS

We compare the new multiplicative update rule (here “multiplicative NeRV”) to the original NeRV implementation (here “additive NeRV”) and to earlier NLDR methods. We first show that the results of multiplicative NeRV are visually equally good as in additive NeRV and are achieved much faster as shown by evolution of the cost function. We then quantitatively compare performance of multiplicative NeRV to additive NeRV and earlier methods on several benchmark data sets and show it yields state of the art performance.

Experiment 1: Comparison with additive NeRV. We first run the proposed method as well as the original NeRV to create 2D visualizations of six data sets, plain s-curve (artificial data sampled from an S-shaped 2D surface embedded in 3D space), sphere (data sampled on the surface of a 3D sphere), phoneme (phoneme samples from 13 classes), land-sat satellite (satellite images from 6 terrain types) a subset of MNIST [16] and Olivetti faces (400 face images of 40 people). See [10] for details of the data. We emphasize precision (\( \lambda = 0.1 \)) to avoid false neighbors. We run additive NeRV for its default 10 iterations, and our method for 300 iterations which is faster than additive NeRV and gives good results. We follow the setup of [10], running each method five times from random initializations and picking the best run by highest F-measure, \( 2(P \cdot R)/(P + R) \) where \( P \) and \( R \) denote rank-based smoothed precision and recall; the rank-based measures are easier-to-interpret alternatives that replace distances in definitions of \( p_{ij} \) and \( q_{ij} \) by ranks of the distances [10].

Fig. 2 shows the resulting visualizations. Both methods yield good quality visualizations with similar cluster shapes and relative positions, despite of some rotations, mirrorings or local minor deformation. We next draw curves of the cost function value versus running time in seconds in Fig. 3; the curves show that for each data set the multiplicative algorithm decreases the cost much faster than the original additive algorithm. The multiplicative algorithm improves the visualization much faster especially at the start of optimization, which is promising for interactive tools where yielding reasonable visualizations quickly in response to user input can be crucial. In conclusion, the faster multiplicative algorithm has achieved speedup while preserving the quality of the results.

Experiment 2: Quantitative comparison with other NLDR methods. We compare the multiplicative NeRV to the original additive NeRV and other NLDR methods by quantitative measures of information retrieval performance: mean smoothed precision and recall achieved in retrieving ground truth neighbors (scale of ground truth neighborhoods was fixed to 20 relevant neighbors). We apply Curvilinear Component Analysis and Curvilinear Distance Analysis (CCA and CDA; see [5]), Laplacian Eigenmap (LE; [3]), ISOMAP [7], Locally Linear Embedding (LLE; see [6]), Hessian LLE (HLLE; [6]), Maximum Variance Unfolding (MVU; see [9]), Landmark MVU (LMVU; [9]), metric multidimensional scaling (MDS; see [4]), Local MDS (LMDS; [8]), Principal Component Analysis (PCA), NeRV [10], and the multiplicative NeRV, to data sets of plain s-curve, noisy s-curve (same as plain s-curve but with significant normally distributed noise added to each data point), Olivetti faces mouse gene expression (gene expression profiles from different mouse tissues) gene expression compendium (human gene expression arrays from http://dags.stanford.edu/cancer) and sea-water temperature time series (time windows of seawater temperature measurements). For details of the data see [10].

We follow the experimental procedure of Venna et al. [10] and choose for each method the best hyperparameters and initialization by highest F-measure as in Experiment 1. For methods LocalMDS, NeRV, and Multiplicative NeRV involving a parameter \( \lambda \) for a user-desired precision-recall tradeoff we plot a curve of results over \( \lambda \). We plot mean smoothed precision and mean smoothed recall; we show each plot with reversed axis directions so that the best results are at top right.
Fig. 2. Visualizations by multiplicative NeRV (“Multiplicative”) and additive NeRV (“Additive”) on six data sets, emphasizing precision ($\lambda = 0.1$). Colors correspond to original 3D coordinates for Plain S-curve and Sphere, and to class labels of data points (not used in optimization) for other data sets. In a good result, similar colors (for S-curve and Sphere) or same-class points (for other data sets) are nearby. Multiplicative NeRV visualizations preserve the good result quality of additive NeRV.

Fig. 4 shows the result; multiplicative NeRV and the original NeRV achieve the best performances on all data sets, thus multiplicative NeRV has achieved speedup while maintaining the state of the art quantitative performance.

5. CONCLUSIONS

We introduced a multiplicative update rule for the information retrieval based visualization method Neighbor Retrieval Visualizer (NeRV). It needs no user-assigned learning rate parameter or line search; in experiments it yields strong speed-up over original NeRV and maintains state of the art performance, in terms of qualitative appearance of visualizations and quantitative information retrieval performance measures.

6. REFERENCES


Fig. 3. Graphs of running time in seconds (horizontal axes) versus information retrieval performance (vertical axes; lower values are better) for the original additive NeRV and for our new multiplicative NeRV. Methods were run for the same input data with the same settings, and were set to optimize a weighted tradeoff of precision and recall with $\lambda = 0.1$. The multiplicative NeRV performs 300 multiplicative update iterations and the original NeRV performs 10 rounds of conjugate gradient descent, which empirically produce similar outputs. We plot the running time and performance as the iterations progress. For both methods we show the best curve out of five random initializations as chosen by best final F-measure.


Fig. 4. Information retrieval performance comparison. In each data set (subfigure), we plot mean smoothed precision on the vertical axis and mean smoothed recall on the horizontal axis for each method; best results are at top right. Multiplicative NeRV (M-NeRV) and original additive NeRV (NeRV) yield best results outperforming other NLDR methods. Results of comparison methods (additive NeRV and previous NLDR methods) are from Venna et al. [10].