Truly Incremental Locally Linear Embedding

Sebastian Schuon, Marko Đurković, Klaus Diepold
Lehrstuhl für Datenverarbeitung
Technische Universität München
Email: schuon@mytum.de, durkovic, kldi@tum.de

Jürgen Scheuerle, Stefan Markward
Department of Mathematics
Technische Universität München
Email: scheurle, markward@ma.tum.de

***Abstract*—We describe an incremental version of the Locally Linear Embedding (LLE) method for manifold learning. Such an incremental LLE algorithm is suitable for real time application, where measurement data is coming in continuously. We are able to incrementally calculate embeddings which provide the same level of accuracy as the original LLE algorithm. For situations where data is appended constantly, the proposed algorithm achieves a significant speed up.**

I. Introduction

Locally Linear Embedding (LLE) has been proposed by Saul and Roweis [1], and is an algorithm for nonlinear dimensionality reduction, belonging to the family of (unsupervised) manifold learning techniques. Manifold learning is particularly interesting, because of recent findings in neuroscience, where it is believed to be similar to the human brain's learning process. In *The Manifold Ways of Perception* Sueng et al. [2] describe this new hypothesis and LLE is suggested as one possible algorithm.

Manifold learning algorithms are very demanding in terms of computing power. One major drawback of these techniques is, that they operate in batch mode, i.e. they process large blocks of data points at once. The algorithm is unable to reuse any information from a previous run and has to restart the whole computation from scratch even for the slightest change in a data set. While this is appropriate for the offline analysis of recorded data sets, it is a huge problem for time critical analysis when operating online mode. Then significant computing resources in terms of CPU power and memory are required when points are being continuously added to a given data set.

Moreover, batch mode processing introduces a significant delay, which may be detrimental for certain applications where the system has to react on an input. In order to make manifold learning suitable for real-time applications, where sensor data is arriving at a high rate, an incremental formulation of the algorithms is required. One possible application for an incremental version of LLE could be the feature extraction from audio data [3] in real-time for tasks like speech recognition or feature extraction from raw sound signals.

There exist a few earlier attempts to formulate an incremental implementation for LLE [1], [4], which all suffer from a loss of accuracy. We present a solution which is based on a reformulation of the originally proposed LLE algorithm, and which is able to handle continuous data flows.

In the course of this paper we will give a short overview over the state of the art in chapter 2, briefly introduce the principles of the LLE algorithm in chapter 3 and show our extension to LLE with some experimental results in chapters 4 and 5. We will conclude with proposals how to further optimize the operation in chapter 6.

II. Related work

Several authors have addressed the problem of handling continuously incoming data efficiently: Saul and Roweis proposed a simple approach [1] to compute an embedding for an initial set of data points. Once new data points arrive, they would linearize the previously computed manifold to determine the embedding of the new data points. This algorithm's remarkable advantages is, that it has very low computational requirements and that its complexity is bounded by $\mathcal{O}(N)$, where N denotes the number of data points to be embedded. However, this algorithm returns satisfactory results only in those cases, where new data points are quite similar to the data points which have been processed previously. If the new data points carry significant new information relevant for the manifold structure, this information will not be incorporated due to the linearization of the manifold. Such new data points are of high interest for most applications at hand and thus this algorithm is generally not to be used.

Bengio et al. [5] try to create a general extension for several manifold learning techniques to allow one to apply a trained model to out-of-sample points without having to recompute eigenvectors.

Another approximate algorithm has been proposed by Kouropteva et al. [4], which delays the approximation to a later stage. Their algorithm approximates the new embedding (by assuming Eigenvalues to be constant, for details see Section III), but over time their estimate drifts away from the true embedding. To compensate for that, regular recomputation of the true embedding and reinitialization of their algorithm is required.

III. The LLE algorithm

LLE is an algorithm for dimensionality reduction of a given data set. LLE takes N data samples (observations) of dimensionality D and calculates a new data set consisting of N points with dimensionality d ($D >> d$), while preserving the structure of the data. This is done, by assuming that all data points are lying on a low dimensional manifold $A \in M^d$,

that is embedded in the high dimensional space D. The basic idea of the algorithm is to compute the embedding, or map of manifold A into the lower d dimensional space.

Accordingly the algorithm takes N data samples which are all points on the surface of A. The input data is stored in a matrix $X \in R^{D\times N}$ where each column represents one sample. The algorithm will find an embedding $Y \in R^{d\times N}$ which preserves neighborhood structure most accurately.

The algorithm itself can be divided into three different steps which will be described separately here. The first step selects the K nearest neighbors of any point, the second step computes the optimal reconstruction weights for each point by its nearest neighbors. Finally the third step performs the embedding by preserving the reconstruction weights of any point in the low dimensional space.

A. Step I: Nearest Neighbors

In the first place, the K nearest neighbors for each data sample (i.e. for each column of X) have to be determined. In terms of differential geometry the geodesic distance would be the most appropriate metric to identify nearest neighbors. Since we are dealing with a sampled manifold, this metric can not be computed. Several other metrics could be imagined, such as infinity norm, but Euclidean distances are the most common ones used.

Hence we construct a matrix $L \in R^{K\times N}$ where each column j includes the data point indices (the i-th column of X refers to the index i) that are the nearest neighbors to the j-th data sample.

B. Step II: Reconstruction Weights

Now one needs to find the optimal reconstruction for each point by its neighbors. Speaking mathematically, this can be expressed in minimizing the reconstruction error, or cost function

$$\epsilon(w_j) = \left\| \mathbf{x} - \sum_{j=1}^{K} w_j \mathbf{n_j} \right\|_2^2 \tag{1}$$

where $\mathbf{x}$ denotes the current data point, $\mathbf{n_j}$ is a nearest neighbor point and w_j the corresponding reconstruction weight. We introduce a normalization constraint $\sum_j w_j = 1$ to fix scale and rewrite Eq. 1:

$$\epsilon = \sum_j \sum_k w_j w_k (\mathbf{x} - \mathbf{n_j}) \cdot (\mathbf{x} - \mathbf{n_k}) \tag{2}$$

introducing a Gram matrix G with the matrix' entries to be $G_{j,k} = (\mathbf{x} - \mathbf{n_j}) \cdot (\mathbf{x} - \mathbf{n_k})$, arriving at

$$\epsilon = \sum_{j=1}^{K} \sum_{k=1}^{K} w_j w_k G_{j,k}\,.$$

Minimizing ϵ is generally a convex problem. In the case of a l_2 norm, it reduces to a least squares problem. It is numerically save to solve first $\sum_{k=1}^{K} G_{j,k} w_k = 1$ and then to normalize the weights afterwards.

If $K > D$, the matrix G becomes singular (i.e. matrix G has only rank D); i.e. infinite linear combinations exist to represent a data point optimally. A particular solution can be enforced by applying any regularizer. Having computed the reconstruction weights for each point, we arrive at a weight matrix $\tilde{W}$, which requires the index information from matrix L. To this end, we store the weights in a matrix $W \in R^{N\times N}$. Here the i,j-th element w_{ij} represents the reconstruction weight of sample x_i by the data point x_j. As normally $N \gg K$, this matrix will be heavily sparse.

C. Step III: Embedding

Using the reconstruction weights for each point, the embedding into d dimensional space is performed. This yields the matrix $Y \in R^{d\times N}$. We formulate this task as a minimization problem using a cost function

$$\sigma(Y) = \sum_{i=1}^{N} \left\| \mathbf{y_i} - \sum_{j=1}^{N} W_{i,j} \cdot \mathbf{y_j} \right\|_2^2 \tag{3}$$

This reflects the idea of minimizing the reconstruction error of low dimensional points. Now we introduce a matrix $M \in R^{N\times N}$ with elements

$$M_{i,j} = \delta_{i,j} - W_{i,j} - W_{j,i} + \sum_{k=1}^{N} W_{k,i} W_{k,j}$$

where $\delta_{i,j} = 1$ for $j = i$, else it equals 0. Given such a matrix M, Eq. 3 can be expressed in quadratic form

$$\sigma(Y) = \sum_{i=1}^{N} \sum_{j=1}^{N} M_{i,j} (\mathbf{y_i} \cdot \mathbf{y_j}) = y^T M y.$$

We solve this problem by computing the singular value decomposition (SVD) of $M = U\Sigma V^T$. Since M is symmetric by construction, eigenvector decomposition (EVD) and SVD coincide.

The weight matrix W can be interpreted as an adjacency matrix and is closely related to the Laplacian in spectral graph theory. If the graph represented by the matrix is connected, the smallest eigenvalue is zero and all components of the corresponding eigenvector have the same value [6]. We leave out the eigenvector for the eigenvalue zero and use the eigenvectors corresponding to the remaining d smallest eigenvalues as the orthogonal basis for our low dimensional map.

IV. Incremental LLE

An incremental version of LLE will speed up the calculation of a new embedding when an existing data set is being modified slightly. When one data sample is added to the set and the existing embedding has to be extended, incremental LLE will try to reuse information and intermediate results gathered by generating the previous embedding. Table I shows roughly what runtime is taken by the different parts of the original algorithm. An incremental formulation needs to update all three steps of LLE, but they themselves can be examined individually. For a hardware implementation this yields the

Step	Action performed	% of runtime
1	Compute distances between samples	9.9
1	Find nearest neighbors	28.4
2	Compute restoration weights	4.6
2	Save restoration weights	20.9
3	Perform embedding	25.2
	Other tasks	11.0

TABLE I
MOST TIME-CONSUMING PARTS OF LLE

advantage, that the algorithm can be implemented in a three step pipeline, thus speeding up overall processing. In the following we will address all three steps involved separately. Furthermore we assume, that only one data point is to be added at a time. More than one point can be handled by repeatedly executing the proposed algorithm.

One of our primary goals for our incremental version was to keep the level of accuracy of the original algorithm. All changes we introduce are constructed in a way to exactly reproduce the same results of the classic LLE.

A. Step I: Nearest Neighbors

By keeping the previous L matrix we are able to create the new L matrix by a simple extension and a few updates. For the new point, we need to determine its nearest neighbors, accordingly the distance between the new point and all existing points has to be computed. The nearest neighbors identified are appended in form of a row to the matrix L. Now we still have the problem, that the new point could be nearest neighbour to one of the other data samples. Therefore we are also caching the distances for each point to its nearest neigbors in a $K \times N$-matrix H, where each entry H_{ij} corresponds to the distance between the i-th sample and its j-th neighbour as denoted by the index in L. If the distance from the new point to an existing data sample is lower than any entry in the corresponding column in H, then L and H have to be updated accordingly.

B. Step II: Reconstruction Weights

Since most distances are not changing between incremental runs of LLE, the W matrix can be cached and kept up to date by a few simple operations. Since the nearest neighbor structure has not only changed for the new point, but also for its neighbors, we have to recompute the weights for these points. It is helpful to keep track which entries of L have changed, in order to determine which rows of W need to be recalculated. This is done in the very same fashion as in the non-incremental case. W is extended by one row and one column to reflect the addition of the new data point.

C. Step III: Embedding

Within this step, the system matrix M is computed as before, since the computational effort is low. To obtain the new embedding, we require the eigenvalue decomposition of M. This is the most time intensive part of the overall algorithm that not lends itsself towards a simple incremental formulation.

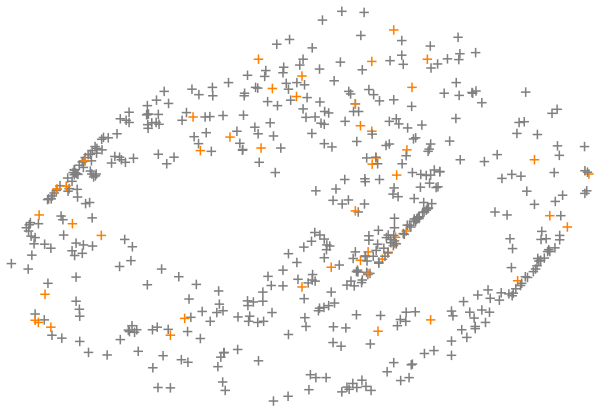

Fig. 1. Swissroll Manifold

In literature, update algorithms for EVD/SVDs are known for such rank one updates [7]. Sadly all this algorithms rely on M being spawned by a wide matrix F as $M = F^T F$. Under these circumstances, the computation of the SVD of M can be reduced to computing the EVD for F. Since this property does not apply, EVD update algorithms are not feasible for LLE.

Nevertheless we can find a incremental formulation by noting that the lower Eigenvalues change only little by appending one data point (see Fig. 2). This is a safe assumption since for a large matrix the changes introduced by one update are rather little. Observing that all modern algorithms for computing an EVD are of iterative nature, we use the old EVD as initialization to these algorithms. This will allow these algorithms to converge with very little iterations.

Although the proposed iterative method represents local optimization, it has to converge to the same solution as with the original global approach, since the embedding's cost function in Eq. 3 is convex.

V. EXPERIMENTAL RESULTS

The algorithm proposed in the previous section is demonstrated on a swissroll manifold, similar to the one found in the first paper on LLE [8]. In Fig. 1 we have depicted this manifold, with data points that are to be added later on being colored. This manifold has been generated by sampling the analytic manifold with $N = 500$ points and randomly sampling another $N_{new} = 50$ points to be added later to the embedding.

Before performing the embedding, it seems reasonable to verify the assumption that only little changes occur in the small Eigenvalues. Hence we added, starting from the initial sampled manifold, points one after another and computed the smallest Eigenvalues. The magnitude of the Eigenvalues of interest has been plotted in Fig. 2. Here we note that magnitude is nearly constant, but for some sudden changes. These changes represent data points which contribute significant new information on the structure of the manifold. Such cases are also well handled by the algorithm, requiring only some more iteration

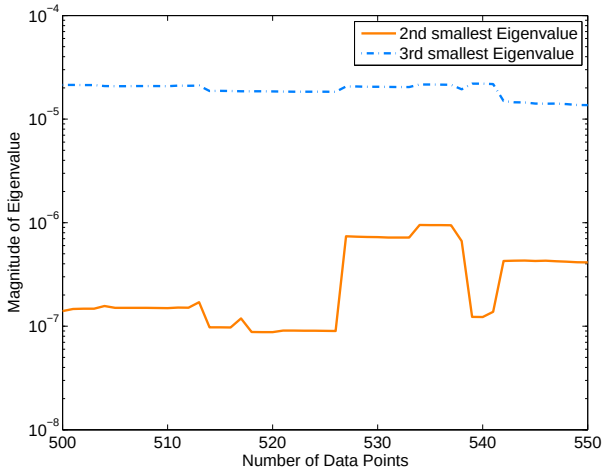

Fig. 2. Plot of Eigenvalues

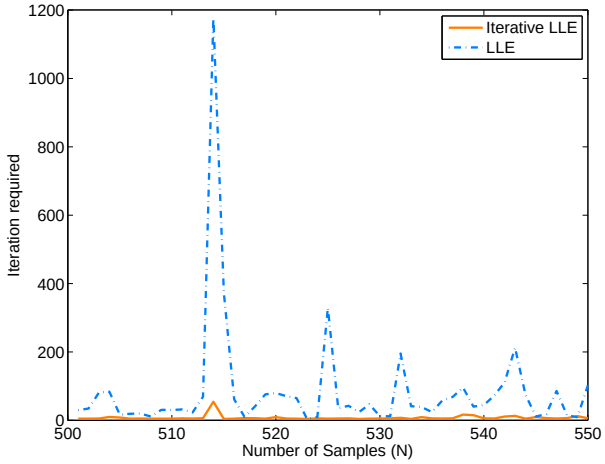

Fig. 3. Comparison of iterations needed to find EVD

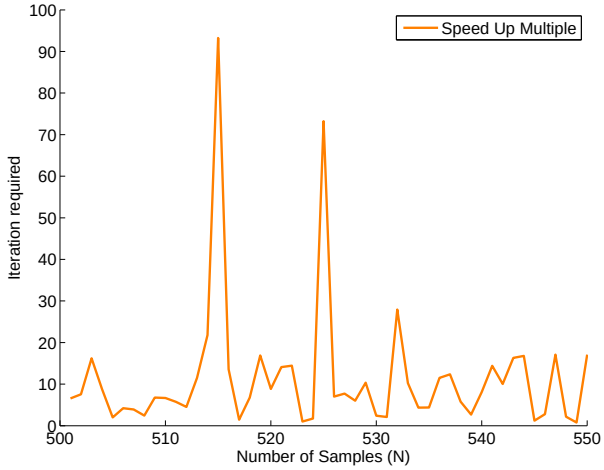

Fig. 4. Speedup by Iterative LLE compared to the original algorithm

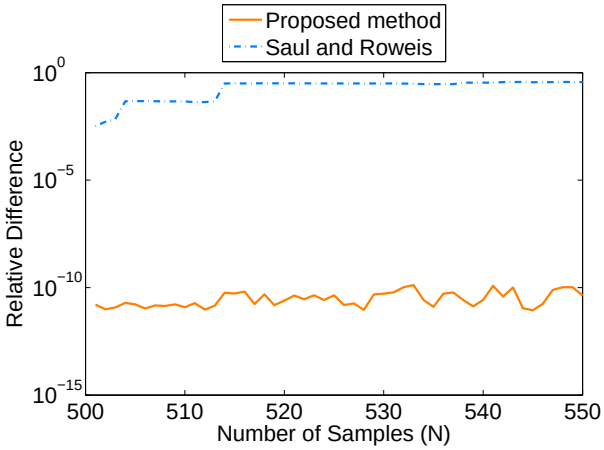

Fig. 5. Relative error for the computed eigenvalues

in the Eigensolver. Overall we can find the assumption of only little changes in Eigenvalue to hold true.

We started with a data set consiting of 500 samples from the swissroll and computed the embedding. Next we compared standard LLE to our incremental LLE by repeatedly adding new samples to the set and recomputing the corresponding embedding. Using Power Iterations to find selective Eigenvalues of a sparse matrix, we recorded the number of iterations required by the proposed algorithm to update the embedding for the same setup described above. In Fig. 3 the iteration count has been plotted, both for computing the EVD with standard LLE and the proposed incremental LLE.

For a better comparison, Fig. 4 shows the speed-up multiple gained by using the incremental algorithm proposed. On average for the time critical step III our new algorithm performs 11 times faster than classic LLE.

For each data point added, we also took measurements of the error introduced by the incremental computation. Fig. 5 shows the cumulative relative error measured using sum of absolute differences between the ground truth manifold and the online updated one. It can be seen that our proposed algorithm is vastly superior to the simple method Roweis et al. suggested. Looking at the relative errors in Fig. 5 it can be seen that our error is around 10^{-10} at all times, which is near the precision of the optimized eigensolvers used (MATLAB) and is therefore negligible. This result in terms of accuracy was expected, since the new formulation of the algorithm has by construction the same precision as the original one.

VI. Conclusion

In this paper we have described an incremental version of the LLE algorithm, that is able to handle constantly incoming data in a much more efficient manner. In terms of conputing power our version is considerably faster and it produces exactly the same results as the original algorithm. Unlike previous work our solution does not suffer from the degradation of accuracy and does not rely on frequent time consuming batch recalculations.

Incremental LLE is especially interesting for applications where data is gathered constantly and information has to be

processed as it arrives. One possible problem with this version of LLE is, that the processing time is not contant and grows as the data set becomes larger. Step I has a computational complexity of $O(n)$ since it basically consists of $2 \times n$ comparisons. Step II is somewhat constant in time, because the weight problem has to be solved for $\approx K$ samples in every iteration. The most significant problem is the spectral decomposition in step III, that has a complexity $> O(n)$.

One way to address this problem is to keep the amount of data low. In future we will need techniques to identify and expunge data samples, that do not contribute significantly or any information at all about the structure of the problem. By aging out those samples we would be able to keep the data set at a manageable size and the computational costs at a reasonable low.

REFERENCES

[1] L. Saul and S. Roweis, "Think Globally, Fit Locally: Unsupervised Learning of Low Dimensional Manifolds," *Journal of Machine Learning Research*, vol. 4, no. 2, pp. 119–155, 2004.

[2] H. Seung and D. Lee, "The manifold ways of perception," *Science*, vol. 290, no. 5500, pp. 2268–2269, 2000.

[3] V. Jain and L. K. Saul, "Exploratory analysis and visualization of speech and music by locally linear embedding." 2004.

[4] O. Kouropteva, O. Okun, and M. Pietikäinen, "Incremental locally linear embedding algorithm," *Image Analysis*, pp. 521–530, 2005.

[5] Y. Bengio, J.-F. Paiement, and P. Vincent, "Out-of-Sample Extensions for LLE, Isomap, MDS, Eigenmaps, and Spectral Clustering," 2003.

[6] F. R. K. Chung, *Spectral Graph Theory*, 1997.

[7] J. R. Bunch and C. P. Nielsen, "Updating the singular value decomposition," *Numerische Mathematik*, vol. 31, no. 2, pp. 111–129, 1978.

[8] S. Roweis and L. Saul, "Nonlinear Dimensionality Reduction by Locally Linear Embedding," 2000.