ESANN'2008 proceedings, European Symposium on
Artificial Neural Networks - Advances in Computational Intelligence and Learning
Bruges (Belgium), 23-25 April 2008, d-side publi., ISBN 2-930307-08-0.

Parallelizing single patch pass clustering

Nikolai Alex[1], Barbara Hammer[2]

1- Univ.of Applied Science Braunschweig/Wolfenbüttel - Dept.Computer Science
Salzdahlumer Str. 46/48, 38302 Wolfenbüttel - Germany

2- Clausthal University of Technology - Dept.Informatics
Julius-Albert-Str. 4, 38678 Clausthal-Zellerfeld - Germany

Abstract. Clustering algorithms such as k-means, the self-organizing map (SOM), or Neural Gas (NG) constitute popular tools for automated information analysis. Since data sets are becoming larger and larger, it is vital that the algorithms perform efficient for huge data sets. Here we propose a parallelization of patch neural gas which requires only a single run over the data set and which can work with limited memory, thus it is very efficient for streaming or massive data sets. The realization is very general such that it can easily be transferred to alternative prototype-based methods and distributed settings. Approximately linear relative speed-up can be observed depending on the number of processors.

1 Introduction

Clustering algorithms like SOM, k-means, fuzzy-k-means, NG, etc. are well studied and numerous extensions exist to improve their efficiency or memory usage, including batch optimization, growing strategies, etc. [8]. A vast literature on parallel implementations of k-means, NG, or SOM can be found [2, 4, 9, 11, 12, 13, 14] such as realizations on specific hardware [9, 12, 13], implementations for distributed settings [4], or implementations with particularly efficient cache usage [14]. Most of these implementations, however, rely on online clustering variants which require multiple runs over the data set, or they implement batch optimization which accesses all data at once.

Massive data sets pose particular challenges towards clustering and visualization and standard algorithms are often not applicable due to memory and time constraints: data sets do not fit into main memory (i.e. direct batch optimization is not possible) and at most one pass over the data set is affordable (i.e. standard online methods cannot be applied either). A number of algorithms has been proposed for streaming data clustering, including various efficient heuristics [5] as well as efficient, but in the worst case prohibitive exact approximations [7]. Recently, patch optimization has been proposed for k-means and NG, respectively, which relies on the original cost functions. It dramatically reduces effort and memory requirements, making it appropriate for huge streaming data [1, 6]. Based on this proposal, a parallel implementation of k-means has been investigated in [11] which realizes a simple and efficient implementation of patch clustering for massive data sets. However, like standard k-means, the algorithm is very sensitive to initialization and requires multiple runs for multimodal data.

In this contribution, we propose a parallel implementation of patch NG which processes data in a single run using only limited memory. The algorithm displays robust results also for multimodal data sets. The realization is based on local sufficient statistics of parallel runs derived from the standard NG cost function, such that a canonic, robust, and flexible method results. Communication is very low and almost linear relative speed-up can be observed. We demonstrate

ESANN'2008 proceedings, European Symposium on
Artificial Neural Networks - Advances in Computational Intelligence and Learning
Bruges (Belgium), 23-25 April 2008, d-side publi., ISBN 2-930307-08-0.

the behavior for an artificial multimodal data set as well as a data set from KDD'98. To our knowledge, this is the first proposal of a parallel single pass learning scheme with limited memory for neural clustering such as NG or SOM.

2 Patch clustering

Assume data $\vec{x}_1, \ldots, \vec{x}_N \in \mathbb{R}^n$ are given, the goal of prototype-based clustering is to find cluster centers $\vec{w}_1, \ldots, \vec{w}_k \in \mathbb{R}^n$ such that the average squared distance of data points to their respective closest prototype is as small as possible. Since this function is very sensitive to local optima, NG [10] includes neighborhood cooperation, optimizing the costs (in the discrete case, neglecting constant factors)

$$\frac{1}{2} \cdot \sum_{ij} h_\lambda(r_i(\vec{x}_j))(\vec{w}_i - \vec{x}_j)^2$$

where $r_i(\vec{x}_j) = |\{\vec{w}_l \,|\, (\vec{w}_l - \vec{x}_j)^2 \leq (\vec{w}_i - \vec{x}_j)^2\}|$ denotes the rank of prototype $\vec{w}_i$ measured according to the distance from $\vec{x}_j$ and $h_\lambda(t) = \exp(-t/\lambda^2)$ constitutes an exponential weighting function of the ranks. For vanishing neighborhood $\sigma \to 0$ original k-means is recovered. This cost function is usually optimized online by means of a stochastic gradient descent, however, this method requires multiple runs over the data. Alternatively, batch optimization in turn determines optimum prototype locations

$$\vec{w}_i = \sum_j h_\lambda(r_i(\vec{x}_j))\vec{x}_j / \sum_j h_\lambda(r_i(\vec{x}_j))$$

for fixed ranks, and optimum data assignments $r_i(\vec{x}_j)$ for fixed prototypes until convergence [3]. This displays quadratic convergence, but it needs to store all data in the main memory at once.

Patch optimization as proposed in [1] is based on batch optimization, but it reduces the memory requirement to a fixed size while maintaining the simplicity of original batch NG: Data are processed in patches of fixed size, say P, such that the data fits into main memory. For every patch of data, standard batch NG is applied, and the result is summarized by means of a sufficient statistics, i.e. for every cluster A computed in NG, its sufficient statistics is formed by $(\mathrm{Sum}^{(A)}, n^{(A)})$ where $\mathrm{Sum}^{(A)}$ is the sum of points assigned to cluster A, and $n^{(A)}$ its number. For the next patch of data, the result of the previous run is integrated in form of k additional training points given by the cluster centers $\mathrm{Sum}^{(A)}/n^{(A)}$ of all clusters A found in the previous run, weighted by their multiplicity $n^{(A)}$. This way, a single run over the hole data set is sufficient, whereby every patch can be processed using only limited memory.

Since NG converges fast for (small) patches, an empirical value for the number of necessary epochs being $\sqrt{P}$ for a data set of size P, the overall time complexity is reduced from $\mathcal{O}(kN\sqrt{N})$ for batch clustering to $\mathcal{O}(kN\sqrt{P})$, N being the number of data points, k the number of clusters, and P the patch size, thereby neglecting sorting (which is almost linear in later runs).

Since data points of early patches contribute only by means of the sufficient statistics of clustering, the order of processing influences the final result. However, it has been demonstrated in [3] that patch NG displays virtually no difference in accuracy compared to batch NG, while drastically reducing time and memory requirements.

ESANN'2008 proceedings, European Symposium on
Artificial Neural Networks - Advances in Computational Intelligence and Learning
Bruges (Belgium), 23-25 April 2008, d-side publi., ISBN 2-930307-08-0.

3 Parallel patch clustering

Standard patch clustering processes every patch sequentially. This can easily be parallelized by assigning different patches to different processors. Afterwards, the sufficient statistics of several parallel patches is merged for the next round. This way, no communication between the processes is necessary during the patch runs and the processes share their information by means of merged statistics after every patch run. The principled algorithm is as follows, C denoting the number of available parallel processing units:

```
init
repeat
        distribute the next C patches P_i on the parallel processors
        cluster the data sets P_i with batch NG independent and in parallel
        calculate the sufficient statistics S_i independent and in parallel
        merge the statistics of all runs                                    (*)
        add the merged statistics as additional points to all
                parallel runs of the next patch iteration
```

Note that, by passing the sufficient statistics of the runs to all subsequent parallel runs, full information of the already processed elements is available after every merge step. In the merge step $(*)$, the result of C parallel runs has to be summarized such that it can serve as additional input patterns for all parallel runs in the next patch iteration. Several different strategies are possible:

(merge1) All cluster statistics $(\mathrm{Sum}^{(A)}, n^{(A)})$ are concatenated in a vector of length kC. Since the points are passed to all parallel patches (i.e. they are implicitly multiplied C times without subsequent averaging), their multiplicities are scaled by $1/C$ to prevent an exponential influence of points in early patches. This procedure has complexity $\mathcal{O}(Ck)$.

(merge2) The cluster statistics are merged to a length k vector using a greedy strategy, sequentially combining the two closest centers of two statistics by means of the weighted mean. This procedure has complexity $\mathcal{O}(C \cdot k^2)$. Note that a length k vector could alternatively be obtained applying batch NG to the sufficient statistics. Due to the comparably high complexity $\mathcal{O}(C^2 k\sqrt{Ck})$, this possibility was not further investigated.

We tested both merge strategies. The resulting clustering results are almost identical, while their complexity is different, such that we choose (merge1) for all intermediate runs and (merge2) to obtain the final k cluster centers (since merge1 obviously results in Ck centers). Merging itself is implemented sequentially, obviously, parallelization would be possible. The resulting parallel patch clustering algorithm constitutes an approximation of single patch clustering, since data patches are processed independently and in parallel instead of a sequential line. We will see in experiments that the resulting clustering results remain almost the same for this very intuitive and flexible parallelization.

The implementation of parallel patch NG has been realized on a multiprocessor architecture (8 dual core machine) using Java 5.0 and its powerful API for multi-threaded application. Each patch clustering is realized as a different thread. A scheduler object retrieves, merges, and distributes the sufficient statistics of the patches. For the realization, different design patterns for functionalities such as data access and the clustering strategy are used.

ESANN'2008 proceedings, European Symposium on
Artificial Neural Networks - Advances in Computational Intelligence and Learning
Bruges (Belgium), 23-25 April 2008, d-side publi., ISBN 2-930307-08-0.

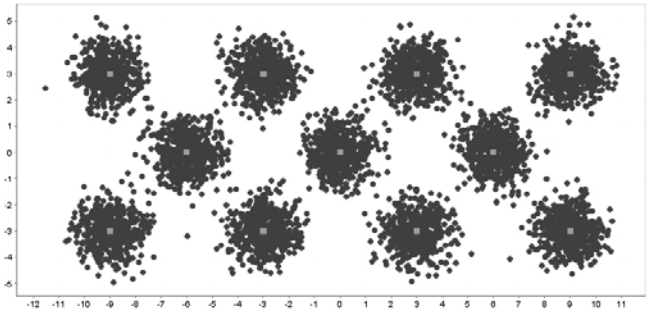

Fig. 1: 11 clouds data set given by a mixture of Gaussians, different sizes of this multimodal two-dimensional data set are generated for testing.

4 Experiments

We tested parallel patch NG for an artificial multimodal data set (11 clouds, see Fig. 1) and a large real life data set from KDD'98. In all cases, the neighborhood range is multiplicatively annealed from 10 to 0. The number of epochs per patch is chosen as 10. The number of clusters is 11 for the 11 clouds and 10 for the KDD data set. The reported results are obtained by a ten-fold cross-validation.

4.1 11 Clouds

The clouds data set is a two dimensional data set as depicted in Fig. 1, given by a mixture of isotropic Gaussians. We create three different sizes in order to test the scalability of the algorithm. The smallest data set contains 110.000 data point (45MB), the medium data set contains 1.1 million records (450MB), the large data set contains 11.1 million data points (4.5GB). For evaluation purposes, we label the data points with the number of the generating Gaussian and evaluate the classification error of the resulting classifier. We test the influence of the patch size P, which equals 1%, 5%, and 10%, respectively, of the data set in three different settings, and the scaling with respect to the number of threads, which is chosen between 1 and 6 (realized on different processors).

We observed the same behaviour of the parallel patch algorithm for settings with different patch sizes due to a sufficient and fixed number of epochs, see

11 Clouds	t avg (ms)	var $\cdot 10^{-3}$	accuracy avg (%)	var	speed-up
batch	201565.48	82.7	99.80	0.18	
1 thread	135308.96	82.7	99.80	0.18	1
2 threads	67703.35	116.1	99.79	0.18	1.9986
3 threads	45161.47	119.3	99.78	0.24	2.9961
4 threads	33907.87	122.1	99.76	0.26	3.9905
5 threads	27366.69	122.5	99.65	0.35	4.9442
6 threads	22631.55	123.6	99.54	0.41	5.9787

Table 1: Experimental results for the 11 clouds data sets, the average execution time (in ms) and the classification accuracy (in %) and the variances are displayed for batch and patch NG. The average speed-up shows linear behavior.

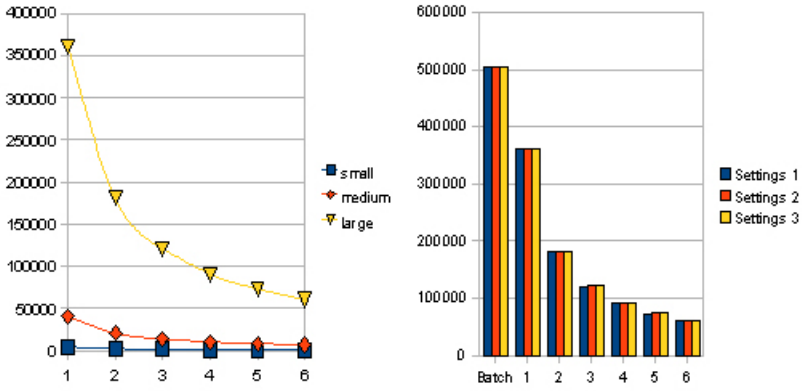

Fig. 2: Experimental results for the 11 clouds data sets: the average execution time in milliseconds is displayed depending on the number of parallel threads. The left graphics shows the scaling for different size data sets, the right graph displays the differences of different patch sizes for the large data set.

Fig. 2. The averaged results are displayed in Tab. 1. For comparison, we also report the result of standard batch clustering (using 15 epochs per run). The table contains the average execution time in milliseconds and the classification error for posterior labeling according to the generating Gaussian centers. Obviously, a classification accuracy close to 100% is reached in all cases, i.e. parallel patch clustering reliably detects all 11 cluster centers, showing only a very small decrease in the optimum positioning of the centers. For the parallelization of patch clustering, close to linear relative speed-up can be observed as documented in Tab. 1. A small overhead is caused by the merge strategy, which is of complexity $\mathcal{O}(Ck)$. This is only a fraction of the overall execution time as can be observed by the results. The classification accuracy is decreased by less than 0.3% in the parallel implementation compared to full batch clustering.

4.2 KDD'98

The second data set stems from the 1998 KDD cup data mining contest. Data contains 95412 records with 481 statistical fields giving information about people who made charitable donations in response to direct mailing requests. We selected 56 features from these fields including numerical features (e.g. donation amount, income, age), date values (e.g. donation date, date of birth), and binary values (e.g. income category). All data are preprocessed to numerical values with zero mean and unit variance.

The performance and accuracy was tested in the same manner as beforehand, whereby the mean quantization error is chosen as evaluation measure. As before, virtually no difference could be observed for different patch sizes (ranging from 1% to 10% of the data). The average execution time and quantization error are displayed in Tab. 2. A linear relative speedup can be observed for the parallel implementation of patch NG, whereby the accuracy measured in terms of the mean quantization error decreases by less than 0.3%.

KDD'98	t avg (ms)	var $\cdot 10^{-3}$	mean error	var $\cdot 10^{5}$	speed-up
batch	92486.48	16.7	0.44654820	0.118	
1 thread	79848.54	744.9	0.44793333	0.118	1
2 threads	39097.12	28.9	0.44803743	0.119	2.0423
3 threads	25383.83	93.2	0.44795150	0.119	3.1456
4 threads	19048.41	19.0	0.44811113	0.119	4.1920
5 threads	15347.82	78.2	0.44800127	0.121	5.2026
6 threads	13548.37	46.9	0.44805333	0.121	5.8938

Table 2: Experimental results of the KDD data set, the average execution time (in ms) and the mean quantization error are displayed together with the variance and the average relative speed-up for parallel threads.

5 Discussion

We proposed a simple parallelization scheme for patch NG which displays almost linear relative speedup while maintaining the classification accuracy of standard batch NG. The algorithm is generically derived from the NG cost function, such that it carries a wide potential for extensions towards further important settings, including extensions of the cost function such as supervised NG or relational NG, general implementation platforms and distributed settings, the incorporation of dynamic growing schemes, extensions to non-stationary data distributions, and the incorporation of additional functionality such as visualization.

References

[1] N. Alex, B. Hammer, F. Klawonn, *Single pass clustering for large data sets*, WSOM'2007

[2] F. Ancona, S. Rovetta, R. Zunino, *A parallel approach to plastic neural gas*, ICNN'1996.

[3] M. Cottrell, B. Hammer, A. Hasenfuss, T. Villmann, *Batch and median neural gas*, Neural Networks 19(6-7): 762-771, 2006.

[4] I. S. Dhillon, D. S. Modha, *A Data-Clustering Algorithm On Distributed Memory Multiprocessors*, Large-Scale Parallel Data Mining, LNAI 1759, pp. 245-260, 2000

[5] F. Farnstrom, J. Lewis, C. Elkan, *Scalability for clustering algorithms revisited, SIGKDD Explorations*, 2(1):51-57., 2000

[6] S. Guha, N. Mishra, R. Motwani, L. O'Callaghan (2000), *Clustering Data Streams*, IEEE Symposium on Foundations of Computer Science, 359-366.

[7] R. Jin, A. Goswami, G. Agrawal (to appear). *Fast and Exact Out-of-Core and Distributed K-Means Clustering*, Knowledge and Information Systems.

[8] T. Kohonen, *Self-Organized formation of topologically correct feature maps, Biological Cybernetics*, 43:59-69., 1982

[9] P. Kolinummi, P. Pulkkinen, T. Hämäläinen, J. Saarinen, *Parallel Implementation of Self-Organizing Map on the Partial Tree Shape Neurocomputer*, NPL 12(2):171-182, 2000

[10] T. Martinetz, S.G. Berkovich, and K.J. Schulten, *'Neural-gas' network for vector quantization and its application to time-series prediction*, IEEE TNN 4:558-569, 1993

[11] S. Nittel, K. T. Leung, *Parallelizing Clustering of Geoscientific Data Sets using Streams*, SSDBM'2004

[12] M. Porrmann, M. Franzmeier, H. Kalte, U. Witkowski, U. Rückert, *A reconfigurable SOM hardware accelerator*, ESANN'2000

[13] U. Seiffert, *Artificial Neural Networks on Massively Parallel Computer Hardware*, ESANN'2000

[14] P. Tomsich, A. Rauber, D. Merkl, *parSOM: Using Parallelism to Overcome Memory Latency in Self-Organizing Neural Networks*, HPCN Europe, pp.136-145, 2000