A Parallel Approach to Plastic Neural Gas

Fabio Ancona, Stefano Rovetta and Rodolfo Zunino
Dept. of Biophysical and Electronic Engineering (DIBE), University of Genoa
Via all'Opera Pia 11a,16145 Genova, ITALY
phone: +39-10-3532269; fax: +39-10-3532175
e-mail: {ancona, rovetta, zunino}@dibe.unige.it

ABSTRACT

A parallel implementation of unsupervised Vector-Quantization networks can reduce the high computational load of the training process. First, a plastic version of the Neural Gas algorithm is presented. Then, the paper describes how a toroidal mesh topology fits the neural model for a distributed implementation. The architecture adopted and the data-allocation strategy enhance the method's scaling properties and remarkable efficiency. Experimental results on a significant testbed (low bit-rate image compression) confirm the validity of the parallel approach.

1. Introduction

The paper describes a methodology to implement a neural algorithm for vector quantization on parallel HW. The final application goal is lossy a compression of high-dimensional data for low bit-rate communication. The high computational load of the neural training process and the technical importance of the specific application motivate the search for a highly efficient parallel implementation of the quantization method.

To this end, we chose a neural model (Plastic Vector Quantization) that exhibits remarkable properties in terms of both consistency (quality of the quantization process) and easy implementation. This model can be considered as a modified version of the Neural Gas (NG) algorithm [1], whose original formulation exhibits the crucial drawback of an advance setting of the number of prototypes. This algorithm makes it possible to add and prune neurons dynamically, and guarantees a finite-time convergence. Akaike's theoretical framework controls the generalization behaviour of the plastic model, while ensuring a satisfactory placement of all neurons [6]. As the plastic model involves the interaction of several NG networks using different vocabularies, the parallel implementation is most effective in reducing the process computational cost.

The paper first outlines the basic neural algorithm, and then points out the features that make this technique very suitable for an efficient HW support. An analysis of the actual implementation methodology shows its notable scaling properties; the satisfactory efficiency is not affected significantly by the number of processors involved. Preliminary experimental results on an image-compression testbed confirm the validity of the overall approach.

2. The Neural Model for Vector Quantization

2.1. The Neural Gas Algorithm

Vector quantization is the process of approximating a large data set of multidimensional data (e.g., image blocks for image compression) by a limited number of "prototype" vectors, obtained by clustering several similar data. This approximation resembles that used in scalar quantization, and proceeds by minimizing some error function (usually, the mean square error).

The NG algorithm, developed by Martinetz et al. [1], is an iterative algorithm to train a set of prototypes. At each iteration, a training pattern is presented and prototype vectors are ordered according to their Euclidean distances from the input sample. Prototypes are then adjusted according to their positions on the ordered list:

0-7803-3210-5/96 $4.00©1996 IEEE

closer vectors undergo larger modifications. The intensities of the adaptation steps and the width of each vector's neighbourhood decrease during training, thus providing a stabilization mechanism, also present in similar algorithms (including Kohonen's SOMs [2]). The NG training algorithm can be outlined as follows:

1. Set W = a set of randomly initialized prototypes; set I = a fixed number of iterations.
2. Repeat for $i = 1$ to I :
   - 2.1. Input a sample vector **x.**
   - 2.2. Compute the distance $d_k = \| \mathbf{x} - \mathbf{w}_k \|$ from each prototype $\mathbf{w}_k$.
   - 2.3. Sort the list of prototypes according to d_k.
   - 2.4. Compute the adaptation step $\Delta \mathbf{w}_k$ for each prototype $\mathbf{w}_k$.
   - 2.5. Apply adaptations to each prototype.
3. Output the set of prototypes W.

This procedure exhibits interesting properties that can be exploited in an HW realization. A specific feature of this algorithm [6] guarantees the existence of such an initialization such that prototypes always lie in a bounded region, provided that input values are themselves bounded (which is always the case in practice). This is very important when one needs to assess a priori the dynamic range of a stored quantity.

The training algorithm involves a number of independent operations, and the absence of a fixed inter-neuron connectivity simplifies a parallel implementation. The relatively large amount of computations at the local level allows one to to achieve a high degree of parallelism; moreover, the alternation of the computation and communication phases makes synchronization easier.

2.2. The Plastic Neural Gas model

In comparison with the basic NG algorithm, the basic feature of the plastic model is the ability to add and prune neurons dynamically. The Plastic Neural Gas algorithm was first proposed in [3]. Each neuron is provided with a local "analog cost" (typically, the mean square error) that measures the quality of the neuron placement. This quantity can eventually control the algorithm's computational overhead: prototypes showing satisfactory placements are deactivated and do not take part in the training process any more.

Training proceeds by iteratively adding neurons to those regions of the data space which appear to be insufficiently covered with available prototypes ("network growing"); an opposite, "network pruning" mechanism removes insignificant units (dead vectors); finally, cost-checking inhibits from the next training iterations those neurons whose analog costs are smaller than a fixed threshold. All these phases are controlled *locally* by monitoring each neuron's analog cost. The plastic model can be outlined as follows:

1. Input: a training data set, a test data set, a cost threshold.
2. Initialize the set of (at least one) prototypes.
3. Repeat until stop:
   - 3.1. Train the *active* nodes in the current vocabulary by the standard NG algorithm.
   - 3.2. Remove insignificant neurons that do not cover any training sample.
   - 3.3. Deactivate nodes showing satisfactory local costs.
   - 3.4. Compute the overall analog cost on the test data set.
   - 3.5. If the test cost has not improved significantly, as compared with the previous iteration,
     Stop the algorithm
     Else
     Add one neuron in proximity to the prototype with the highest cost.
4. Output the set of prototypes W.

The plastic method can be shown to have a finite-time convergence; more importantly, a network's generalization ability can be easily assessed, as well. In particular, one can control the growing process by a sort of cross-validation procedure: available data are split into a training set and a test set, and the cost of test data operates as a stopping criterion for the overall plastic process. This empirical mechanism aims to estimate the smallest number of prototypes required to achieve a given accuracy of the overall data distribution.

In summary, plasticity increases the performance of a neural structure from both a computational and a generalization perspective. From a computational point of view, through neuron deactivation one can remove entire subregions of the data space from the training process, and limit the training overhead accordingly. At

the same time, generalization is enhanced by avoiding the introduction of insignificant vectors, which might ultimately give rise to overfitting phenomena.

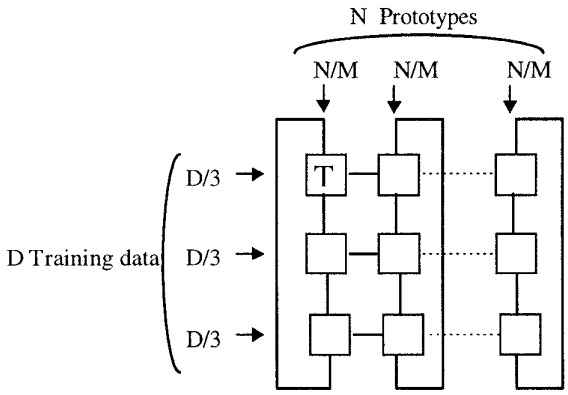
3. Parallel Implementation

The hardware architecture is subject to different constraints, mainly related to its performance, cost, and flexibility. Transputers constitute the basic processing units of the final HW realization, as they are less expensive than commercial workstations. Thanks to their high structural flexibility, one can design systems in compliance with target applications. In neuroimplementations, transputers can represent a suitable compromise between opposite constraints (e.g., image size involving a high computational overhead, timings imposed by the application domain, easy configuration, and availability of resources). On the other hand, higher development costs demand an accurate architecture design, and the data-allocation strategy and the organization of processors play crucial roles for a system's effectiveness [4].

3.1. Architecture and Data Allocation

The realized algorithm is suited to being distributed over an architecture characterized by a toroidal mesh topology, as shown in Fig.1. The basic approach lies in splitting neurons along the mesh columns, whereas the training data set is partitioned along the mesh rows.

The specific nature of the neural algorithm makes it possible to identify three distinct computational phases, namely, steps 2.2, 2.3, and 2.4+2.5. Thus a straightforward and effective data-allocation method is to split the data set into three subsets and to map them into the mesh rows. As a result, each row is entrusted with the training of one third of the entire training data set. Conversely, the mutual topological independence of neurons makes it possible to partition the prototype set into as many subsets as the mesh columns, whose number is not fixed and can be changed according to the number of available processors.



rows = 3 (always); M = #columns; D= number of training samples; N=number of prototypes.
Fig. 1: The mesh architecture and the related data-allocation strategy

3.2. Algorithm Implementation

The above allocation approach has important consequences on the actual algorithm implementation and its efficiency. The system's run-time kernel is arranged in a state machine as follows:

1) Compute *locally* the distances between available samples and local prototypes.
2) Sort prototypes.
3) Update prototypes *locally*.

4) *Send* adjustment steps for each vector to the next row in the mesh.

This approach has several specific features enhancing a parallel performance: the computation-intensive phase, namely, the working out of distances, is performed entirely at the local level, thus yielding the maximum efficiency. Likewise, the vector-adjustment step does not involve any inter-processor communication.

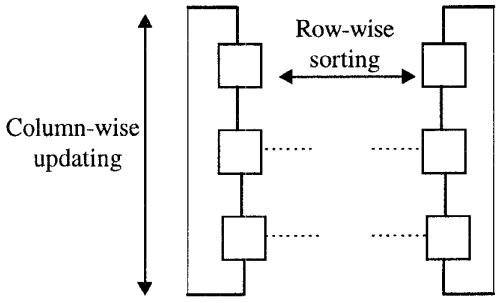


Fig. 2: Communication structure.

As to the communication overhead (Fig. 2), the sorting phase involves row-wise communications; as a result, the sorting process proceeds independently along each row for one third of the allocated data. In addition, the amount of transmitted information (vector index + scalar distance values) is small, as compared with the huge amount of data stored for each datum and each vector. Conversely, the communication of adjustment displacements (step 4) involves a larger amount of information, but its parallelism spreads over *columns*, whose number is unbounded. This property allows the critical part of communication costs to be reduced by increasing the number of processors, hence efficiency is made virtually independent of the problem scale.

3.3. Experimental Results

In our experiments, we used a simple 6-processor network (2 columns and 3 rows) for evaluation purposes. Accuracy results include the *training timing* for the sequential algorithm, the *training timing* for the parallel algorithm, and the system's efficiency ($\sigma = 1/P \cdot T_{seq}/T_{par} = 0.58$; P = # processors).

The application testbed consisted in an image-compression task, in which low bit-rate coding is achieved by VQ encoding. The compression system processed standard (grey-level) images (8bpp) with 512x512 pixels. All pictures were split into 4096 blocks including 8x8 pixels each. In the experiments, a set of classical pictures were used for the network training, and a different image set for the generalization-based algorithm control.

The graph in Fig.3a plots the training and test (the "Lena" picture) costs versus the number of prototypes used. The curves show that the relative improvement on test data decreases progressively; the fact that the test-cost curve becomes "flat", while the training one keeps decreasing, marks an incipient overfitting and triggers the generalization-based stopping condition. This situation indicates the estimated "best" number of neurons which balances the representation accuracy with the size of the vocabulary. In the case considered, the

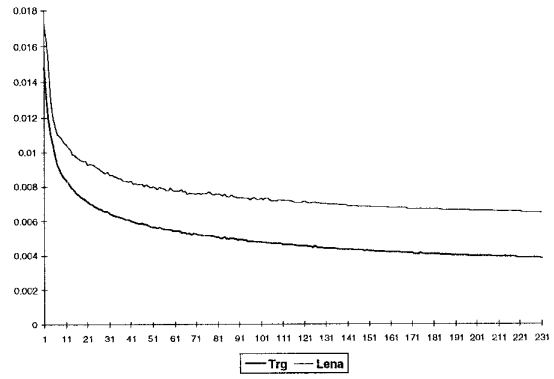


Fig. 3: Plastic Neural Gas for image compression
a) Analog-cost curves (x axis = # neurons) b) Validation performance

estimated optimal cardinality of the prototype set lies in the range [190, 230].

Figure 3b presents the network's performance on a "validation" picture not used for training nor for cross-validation. Results attained a compression ratio of 42.7, with a $\mathrm{PSNR} = 10 \cdot \log(255^2 / \sum_{\#\text{ of pixels}} e^2) = 28.26$, where e is the error between a pixel of the original image and its pixel corresponding of the rebuilding image, (SNR=22.71, MSE=97.90), indicating the method's remarkable performance as compared with classical compression techniques (e.g., JPEG).

4. Concluding Remarks

Vector Quantization can provide an image-coding schema with a remarkable compression ability, thanks to the codebook-indexing mechanism intrinsic to the quantization process. This advantage is often obtained at the cost of some coarseness and blockness affecting the reconstruction quality. In this sense, an adaptive technique to improve the overall generalization ability is described in [5]. A crucial issue inherent in all these methodologies is the computational cost of training, especially in high-dimensional domains with many training samples.

Therefore, a method for a parallel implementation with high efficiency appears very interesting and useful from a practical perspective, too. In this regard, the paper has presented a general methodology that combines a low-cost machinery with a scalable and effective implementation of the neural model. This represents the basic advantage and the main novel point of the described method.

The current lines of research in this area concern the development of more complex architectures integrating several processors for a real-domain utilization.

References

[1] T. M. Martinez, S. G. Berkovich and K. J. Schulten, "'Neural-Gas' network for vector quantization and its application to time-series prediction," *IEEE Transaction Neural Networks*, vol. 4, No. 4, pp. 558–569, 1993.

[2] T. Kohonen, "Self-organization and associative memories," Heidelberger:Springer, 1982.

[3] S. Ridella, S. Rovetta and R. Zunino, "Generalization-based Approach to Plastic Vector Quantization", *World Congr. Neur.Netw. WCNN'95*, Washington, vol.I, 505-508, 1995.

[4] F. Pagano, GC. Parodi and R. Zunino, "Parallel implementations of associative memories for image classification," *Parallel Computing*, vol. 19, No. 6, pp. 667–684, 1993.

[5] D. Anguita, F. Passaggio and R. Zunino, "SOM-based interpolation for image compression", *World Congr. Neur.Netw. WCNN'95*, Washington, vol.I, 739-742, 1995.

[6] S. Ridella, S. Rovetta and R. Zunino, "On the role of generalization in adaptive dynamic vector quantization," submitted to *IEEE Trans. Neural Networks*, 1995.