Implementing Neural Gas Networks on Distributed Architectures

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 3532268
e-mail(s): {ancona, rovetta, zunino}@dibe.unige.it

Abstract

The paper describes a methodology for the implementation of a neural algorithm for vector quantization on parallel hardware. The final application goal is lossy compression of high-dimensional data for low bit-rate image 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. First, the paper shows a neural algorithm that implement the vector quantization (Neural Gas). Then, the paper presents the strategy to distribute the algorithm over an architecture based on a toroidal mesh topology. Experimental results on an application testbed consisted in an image-compression task, in which low bit-rate coding is achieved by Vector-Quantization encoding, confirm the validity of the parallel approach.

1. Introduction

In the past few years there has been significant progress in video coding. Very low bit rate video coding is still an open reseach area where useful contribution are obtained by Vector-Quantization encoding. Neural training process for VQ implementations has a drawback in the high computational load for compression of high-dimensional data. We have incentived the search to overcome this drawback by a parallel approach implementing the quantization method. To this end, we chose a neural model (Neural Gas [1]) that exhibits remarkable properties in terms of both consistency (quality of the quantization process) and easy implementation, and we describe how a toroidal mesh topology fits the proposed model for a distributed approach. The paper first outlines the basic neural algorithm, and then points out those features making this technique very suitable for efficient HW support. The analysis of the actual implementation methodology shows its notable scaling properties, that is, the satisfactory efficiency of the technique is not affected significantly by the number of processors involved. Preliminary experimental results confirm the validity of the overall approach.

2. The Neural Gas algorithm for vector quantization

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

The Neural Gas (NGAS) algorithm, developed by Martinetz et al. [1], is an iterative algorithm to train a set of prototypes. At each iteration, a sample datum is received, and prototype vectors are ordered according to their distances from the input sample. Prototypes are then adjusted according to their positions on the ordered list: closer vectors undergo larger modifications. The intensity of the adaptation steps and the width of each vector's neighborhood decrease during training, thus providing a stabilization mechanism, also present in similar algorithms (including Kohonen's SOMs [2]). The NGAS 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. The simple rule provides a uniform coverage of input samples with the available number of prototypes, thus maximizing the representation consistency. Although there is no theoretical proof of convergence, the algorithm has often been shown to have notable advantages over similar models.

Another property [3] guarantees the existence of an initialization of prototypes such that they 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. It is worth noting that weights in back-propagation networks do not feature this property.

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 achieve a high degree of parallelism; moreover, the alternation of the computation and communication phases makes synchronization easier.

3. Parallel implementation

The hardware architecture is subject to different constraints, mainly related to its performance, cost, and flexibility. In this research, transputers constitute the basic processing units of the final HW realization. They are less expensive than commercial workstations applied for image processing; besides, thanks to their high structural flexibility, one can configure a system in compliance with target applications.

In the implementation of neural systems, transputers can represent a suitable compromise between opposite constraints (e.g. image size involving computational overhead, timings imposed by the application domain, easy configuration, and availability of resources). On the other hand, higher development costs demand an accurate architectural design. In particular, the data-allocation strategy and the organization of processors play crucial roles in the 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 approch 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 contribution 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.

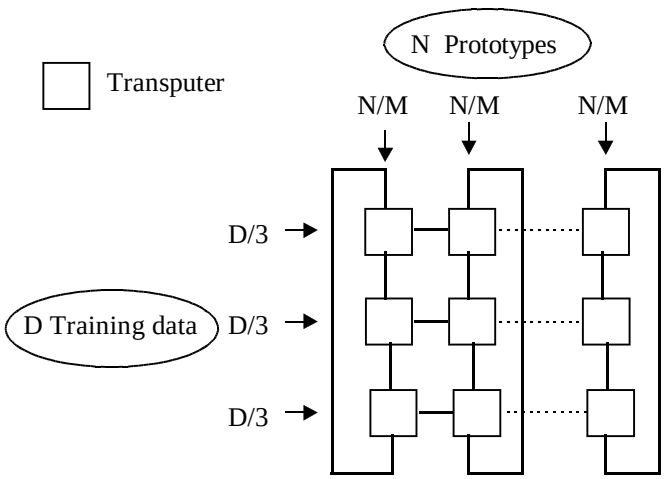

Fig. 1 - The mesh architecture and the related data-allocation strategy:
rows = 3 (always); M = #columns; D= number of training samples; N=number of prototypes.

3.2 - Algorithm implementation

The above allocation approach has important consequences on the actual algorithm implementation and its resulting 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.

As to the communication overhead, 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 vector. Conversely, the communication of adjustment displacements at 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. Actually, the efficiency behaviour of the system when the number of processors increases can be described by observing:

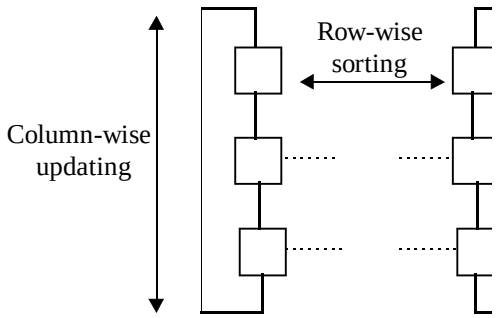

Fig.2 - Communication structure.

$$\lim_{n\to\infty} \eta = 1 \qquad \text{where } \eta \text{ is the system's efficiency}.$$

3.3 - Experimental results

As result of a previous research [5], we have developed a method to estimate the number of prototypes as a good compromise between compression and generalization power of the model. In this research the neural method has been applied to image compression.

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 ($\eta = 1/P \cdot T_{seq}/T_{par} = 0.58$; 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 adopted for the network training, and a different image set for the generalization-based algorithm control. A network with 236 prototypes was used.

Figure 3 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 PSNR of 28.26 (SNR=22.71, MSE=97.90), indicating the method's notable 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 [6]. 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.

5. References

[1] Martinez T. M., Berrkovich S. G., Schulten K. J., "'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] Kohonen T., "Self-organization and associative memories," Heidelberger:Springer, 1982.

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

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

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

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

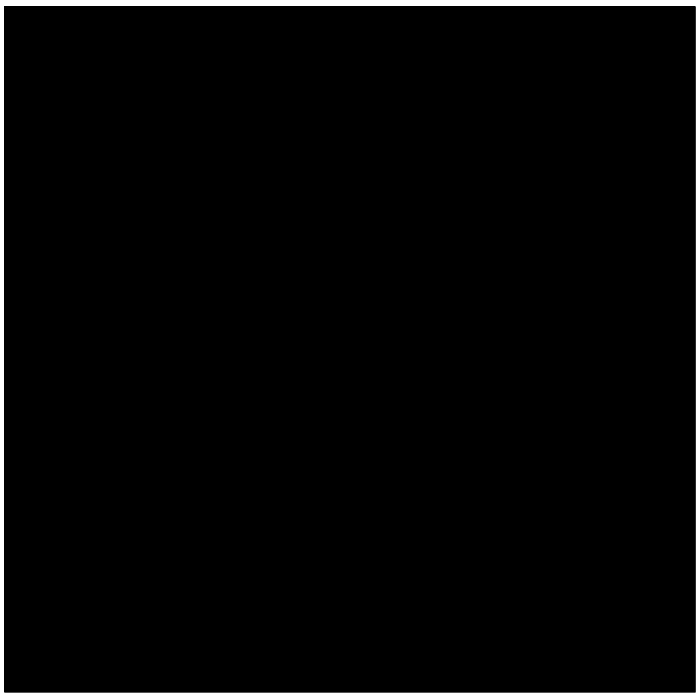

Fig.3 - Validation performance of Neural Gas for image compression