Avoiding Prototype Proliferation in Incremental Vector Quantization of Large Heterogeneous Datasets

Héctor F. Satizábal, Andres Pérez-Uribe, and Marco Tomassini

Abstract. Vector quantization of large datasets can be carried out by means of an incremental modelling approach where the modelling task is transformed into an incremental task by partitioning or sampling the data, and the resulting datasets are processed by means of an incremental learner. Growing Neural Gas is an incremental vector quantization algorithm with the capabilities of topology-preserving and distribution-matching. Distribution matching can produce overpopulation of prototypes in zones with high density of data. In order to tackle this drawback, we introduce some modifications to the original Growing Neural Gas algorithm by adding three new parameters, one of them controlling the distribution of the codebook and the other two controlling the quantization error and the amount of units in the network. The resulting learning algorithm is capable of efficiently quantizing large datasets presenting high and low density regions while solving the prototype proliferation problem.

Keywords: Large Datasets, Vector Quantization, Topology-Preserving Networks, Distribution-matching, Prototype Proliferation, Growing Neural Gas Algorithm.

Héctor F. Satizábal
Institut des Systèmes d'Information (ISI), Hautes Etudes Commerciales (HEC), Université de Lausanne, Switzerland
e-mail: Hector.SatizabalMejia@unil.ch

Andres Pérez-Uribe
Reconfigurable and Embeded Digital Systems (REDS), University of Applied Sciences of Western Switzerland (HES-SO) (HEIG-VD)
e-mail: andres.perez-uribe@heig-vd.ch

Marco Tomassini
Institut des Systèmes d'Information (ISI), Hautes Etudes Commerciales (HEC), Université de Lausanne, Switzerland
e-mail: Marco.Tomassini@unil.ch

L. Franco et al. (Eds.): Constructive Neural Networks, SCI 258, pp. 243–260.
springerlink.com

1 Introduction

Processing information from large databases has become an important issue since the emergence of the new large scale and complex information systems (e.g., satellite images, bank transaction databases, marketing databases, internet). Extracting knowledge from such databases is not an easy task due to the execution time and memory constraints of actual systems. Nonetheless, the need for using this information to guide decision-making processes is imperative.

Classical *data mining* algorithms exploit several approaches in order to deal with this kind of dataset [8, 3]. Sampling, partitioning or hashing the dataset drives the process to a *split and merge*, *hierarchical* or *constructive* framework, giving the possibility of building large models by assembling (or adding) smaller individual parts. Another possibility to deal with large datasets is incremental learning [9]. In this case, the main idea is to transform the modelling task into an *incremental task*[1] by means of a sampling or partitioning procedure, and the use of an incremental learner that builds a model from the single samples of data (one at a time).

Moreover, large databases contain a lot of redundant information. Thus, having the complete set of observations is not mandatory. Instead, selecting a small set of prototypes containing as much information as possible would give a more feasible approach to tackle the knowledge extraction problem. One well known approach to do so is *Vector Quantization* (VQ). VQ is a classical quantization technique that allows the modelling of a distribution of points by the distribution of prototypes or reference vectors. Using this approach, data points are represented by the index of their closest prototype. The codebook, i.e. the collection of prototypes, typically has many entries in high density regions, and discards regions where there is no data [1].

A widely used algorithm implementing VQ in an incremental manner is Growing Neural Gas (GNG) [7]. This neural network is part of the group of *topology-representing networks* which are unsupervised neural network models intended to reflect the topology (i.e. dimensionality, distribution) of an input dataset [12]. GNG generates a graph structure that reflects the topology of the input data manifold (topology learning). This data structure has a dimensionality that varies with the dimensionality of the input data. The generated graph can be used to identify clusters in the input data, and the nodes by themselves could serve as a codebook for vector quantization [5].

In summary, building a model from a large dataset could be done by splitting the dataset in order to make the problem an *incremental task*, then applying an *incremental learning* algorithm performing *vector quantization* in order to obtain a reduced set of *prototypes* representing the whole set of data, and then using the resulting codebook to build the desired model.

Growing Neural Gas suffers from prototype proliferation in regions with high density due to the absence of a parameter stopping the insertion of units in sufficiently-represented[2] areas. This stopping criterion could be based on a local

[1] A learning task is incremental if the training examples used to solve it become available over time, usually one at a time [9].

[2] Areas with low quantization error.

measure of performance. One alternative that exploits this approach to overcome the aforementioned drawback was proposed by Cselenyi [4]. In this case, the proposed method introduces eight new parameters to the GNG algorithm proposed by Fritzke [7]. In our case, we propose a modification that adds three new parameters to the original GNG algorithm in order to restrict the insertion of new units due to points belonging to already covered areas. This approach promotes the insertion of new units in areas with higher quantization error in order to produce network structures covering a higher volume of data using the same number of units.

The rest of the article is structured as follows. In section 2 we make a brief description of the original GNG algorithm. Section 3 describes the proposed modifications made to the algorithm. Section 4 describes some of the capabilities of the resulting method using some "toy" datasets. Section 5 shows two tests of the modified algorithm with a real large size dataset and finally, in section 6 we give some conclusions and insights about prototype proliferation and the exploitation of large datasets.

2 Growing Neural Gas

Growing Neural Gas (GNG) [7] is an incremental *point-based network* [2] which performs vector quantization and topology learning. The algorithm builds a neural network by incrementally adding units using a competitive hebbian learning strategy. The resulting structure is a graph of neurons that reproduces the topology of the dataset by keeping the distribution and the dimensionality of the training data [5].

The classification performance of GNG is comparable to conventional approaches [10] but has the advantage of being incremental. This gives the possibility of training the network even if the dataset is not completely available all the time while avoiding the risk of catastrophic interference. Moreover, this feature makes GNG also suitable for incremental modelling taks where processing a large dataset is not possible due to memory constraints. In such cases, one should proceed in two steps. First, the dataset of the process to be modelled should be split in smaller parts having a size that the system can manage. Second, the resulting parts should be used to train a model like GNG by feeding incrementally each one of the individual datasets resulting from the partitioning procedure. In summary, the methodology consists of transforming the modelling task into an incremental-modelling task, and training an incremental learner in order to build a model of the complete dataset [9].

The algorithm proposed by Fritzke is shown in table 1. In such an approach, every λ iterations (step 8), one unit is inserted halfway between the unit q having the highest error and its neighbour f having also the highest error. Carrying out this insertion without any other consideration makes the network converge to a structure where each cell is the prototype for approximately the same number of data points and hence, keeping the original data distribution.

As an example, a GNG network was trained using the dataset shown in figure 1, and the training parameters shown in table 2. These values were selected after several runs of the algorithm.

Table 1 Original growing neural gas algorithm proposed by Fritzke.

Step 0:	Start with two units a and b at random positions w_a and w_b in $\Re^n$
Step 1:	Generate an input signal ξ according to a (unknown) probability density function $P(\xi)$
Step 2:	Find the nearest unit s_1 and the second-nearest unit s_2
Step 3:	Increment the age of all edges emanating from s_1
Step 4:	Add the squared distance between the input signal and the nearest unit in input space to a local counter variable: $$\Delta error(s_1) = \|w_{s_1} - \xi\|^2 \quad (1)$$
Step 5:	Move s_1 and its direct topological neighbours towards ξ by fractions ε_b and ε_n, respectively, of the total distance: $$\Delta w_{s_1} = \varepsilon_b (\xi - w_{s_1}) \quad (2)$$ $$\Delta w_n = \varepsilon_n (\xi - w_n) \quad \text{for all direct neighbours } n \text{ of } s_1 \quad (3)$$
Step 6:	If s_1 and s_2 are connected by an edge, set the age of this edge to zero. If such an edge does not exist, create it
Step 7:	Remove edges with an age larger than a_{max}. If the remaining units have no emanating edges, remove them as well
Step 8:	If the number of input signals generated so far is an integer multiple of a parameter λ, insert a new unit as follows: • Determine the unit q with the maximum accumulated error. • Insert a new unit r halfway between q and its neighbour f with the largest error variable: $$w_r = 0.5 (w_q + w_f) \quad (4)$$ • Insert edges connecting the new unit r with units q and f, and remove the original edge between q and f. • Decrease the error variables of q and f by multiplying them with a constant α. Initialize the error variable of r with the new value of the error variable of q.
Step 9:	Decrease all error variables by multiplying them with a constant d
Step 10:	If a stopping criterion (e.g., net size or some performance measure) is not yet fulfilled go to step 1

Figure 2 shows the position and distribution of the 200 cells of the resulting structure. As we can see, the distribution of each one of the variables is reproduced by the group of prototypes required.

The GNG algorithm is a vector quantizer which places prototypes by performing entropy maximization [6]. This approach, while allowing the vector prototypes

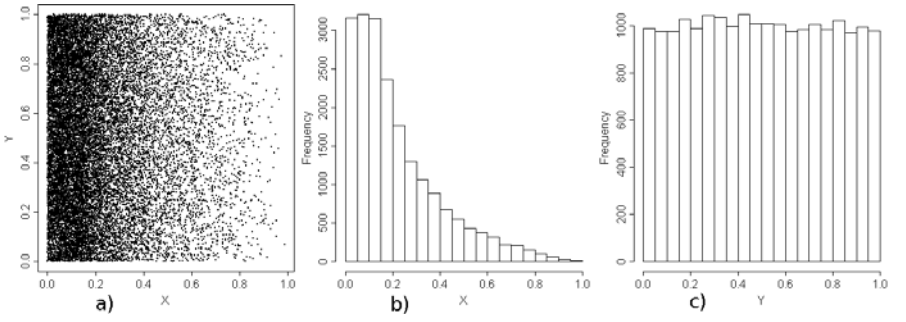

Fig. 1 a) Two dimensional non-uniform data distribution. b) Histogram of variable X. c) Histogram of variable Y.

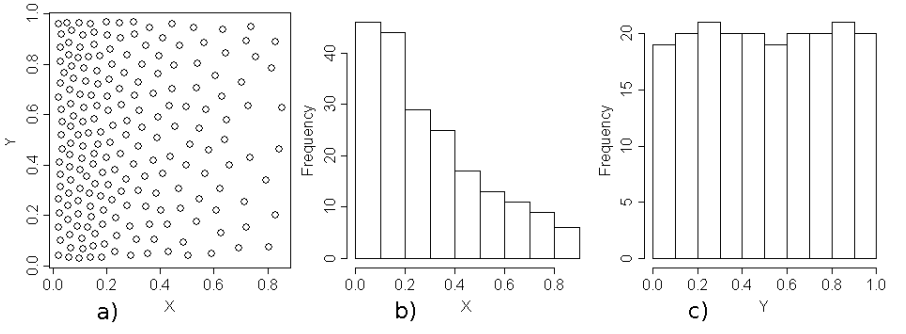

Fig. 2 Positions of neurons of the GNG model. a) Position of the neuron units. b) Distribution of X. c) Distribution of Y.

Table 2 Parameters for the Growing Neural Gas algorithm.

Parameter	ε_b	ε_n	λ	a_{max}	α	d
value	0.05	0.005	100	100	0.5	0.9

to preserve the distribution of the original data, promotes the overpopulation of vector prototypes in high density regions, and therefore, this situation is desirable if the goal is a distribution-matching codebook. Conversely, if we have a huge dataset where there is a lot of redundant information, and we want to keep only a relatively small number of prototypes describing the data with some distortion, then we are not interested in reproducing the data distribution. Instead, we would want to distribute the prototypes over the whole volume of data without exceeding a maximum quantization error or distortion. In such a case, some modifications to the original algorithm are needed in order to modulate the distribution matching property of the algorithm, and therefore, replacing the entropy maximization behaviour in high densitiy regions by an error minimization policy being capable of stopping the insertion of prototypes.

3 Proposed Modifications to the Algorithm

This section describes the three main modifications we propose. As already mentioned, the main goal of modifying the original algorithm is avoiding prototype proliferation in high density regions of data, or in other words, to modulate the distribution matching property of the algorithm. The modified version adds three new parameters to the original algorithm, whose operation is explained in the following subsections.

3.1 Modulating the Local Measure of Error

The criterion driving the insertion of units in the GNG algorithm is the accumulated error of each unit (equation 1). This local error measure grows each time a cell becomes the winner unit (i.e. the closest unit to the current data point), producing the insertion of more cells in zones with higher densities of data.

In order to attenuate that effect, we propose the error signal $\Delta error$ when it is produced by a data point having a quantization error smaller than a threshold qE by multiplying it by a factor h, as show in equation 5.

$$accumulatedError = \begin{cases} accumulatedError + \Delta error & \text{if } \Delta error \geq qE \\ accumulatedError + (h * \Delta error) & \text{if } \Delta error < qE \end{cases} \quad (5)$$
$$\text{where } 0 \leq h \leq 1$$

3.2 Modulating the "Speed" of Winner Units

The proliferation of prototypes could also be due to neuron movement. Neuron units located in zones with higher densities are chosen as winners with higher probability, attracting their neighbours belonging to less populated zones. As in the previous case, parameter h can be used to modulate the change of position, or speed, of the units in each iteration, replacing equations 2 and 3, by equations 6 and 7.

$$\Delta w_{s_1} = \begin{cases} \varepsilon_b (\xi - w_{s_1}) & \text{if } \Delta error \geq qE \\ h * \varepsilon_b (\xi - w_{s_1}) & \text{if } \Delta error < qE \end{cases} \quad (6)$$

$$\Delta w_n = \begin{cases} \varepsilon_n (\xi - w_n) & \text{if } \Delta error \geq qE \\ h * \varepsilon_n (\xi - w_n) & \text{if } \Delta error < qE \end{cases} \quad \text{for all direct neighbours } n \text{ of } s_1 \quad (7)$$

3.3 Modulating the Overlapping of Units

Parameter qE is the radius of a hypersphere determining the region of influence of each prototype. These regions of influence could overlap to a certain extent. In our approach this amount of superposition can be controlled by a parameter sp. Every λ iterations (step 8), one unit is inserted halfway between the unit q having the highest

Table 3 Proposed modification to the original algorithm.

Step 8: If the number of input signals generated so far is an integer multiple of a parameter λ, insert a new unit as follows:

- Determine the unit q with the maximum accumulated error.
- Determine the unit f in the neighbourhood of q with the maximum accumulated error.
- Calculate the distance $dist$ between units q and f.

$$dist = \|q - f\| \tag{8}$$

- Calculate the available space between the two units as follows:

$$available = dist - qE \tag{9}$$

If equation 10 yields true, goto step 9.

$$(available > (sp \times qE)) \quad \text{where } 0 \leq sp \leq 1 \tag{10}$$

Else,

- Insert a new unit r halfway between q and its neighbour f with the largest error variable:

$$w_r = 0.5\left(w_q + w_f\right) \tag{11}$$

- Insert edges connecting the new unit r with units q and f, and remove the original edge between q and f.
- Decrease the error variables of q and f by multiplying them with a constant α. Initialize the error variable of r with the new value of the error variable of q.

error and its neighbour f having also the highest error (see equation 4). Knowing the distance between unit q and unit f (equation 8), and taking the quantization error qE as the radius of each unit, one could change the step 8 of the original algorithm proposed by Fritzke as shown in table 3. Hence, we propose to insert a new unit as in the original version (equation 11), but only if there is enough place between unit q and unit f (equation 10).

3.4 Insights about the Proposed Modification

In summary, the proposed modifications add three parameters, qE, sp and h, to the original GNG algorithm. Parameters *quantization error* qE and *superposition percent* sp depend on the application and are strongly related. Both control the amount of units in the resulting neural network structure, the former by

controlling the region of influence of each unit, and the latter by controlling the superposition of units. These natural meanings allow them to be tuned according to the requirements of each specific application.

In a less obvious sense, parameter h controls the distribution of units between high and low density areas, modulating the distribution-matching property of the algorithm. In order to do so, parameter h modulates the signal that drives the insertion of new units in the network only if the best matching neuron fulfills a given quantization error condition. In this way, the algorithm does not insert unnecessary units in well represented zones, even if the local error measure increases due to high data density. Some examples exploring this parameters are given in section 4.

4 Toyset Experiments

In section 2, a non-uniform distribution of points in two dimensions was used to train a GNG network. Figure 2 shows a high concentration of prototypes in the zone with higher density due to the property of density matching of the model. This is an excellent result if we do not have any constraint on the amount of prototypes. In fact, having more prototypes increases the execution time of the algorithm since

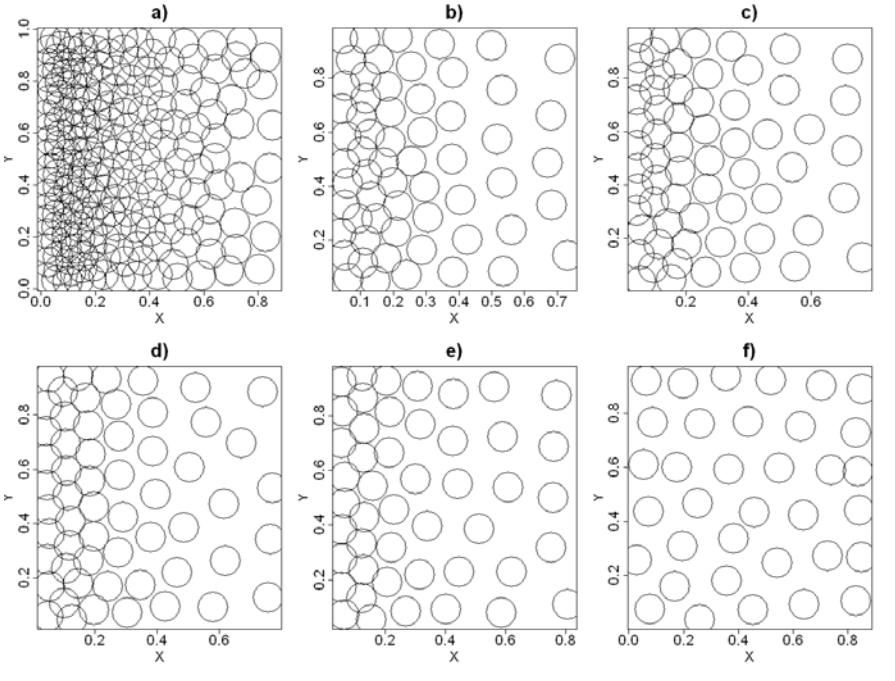

Fig. 3 Results of the modified algorithm varying parameter h ($qE = 0.1$ and $sp = 0.75$). a) Using the original GNG algorithm b) Using $h = 1.00$ c) Using $h = 0.75$ d) Using $h = 0.50$ e) Using $h = 0.25$ f) Using $h = 0.00$.

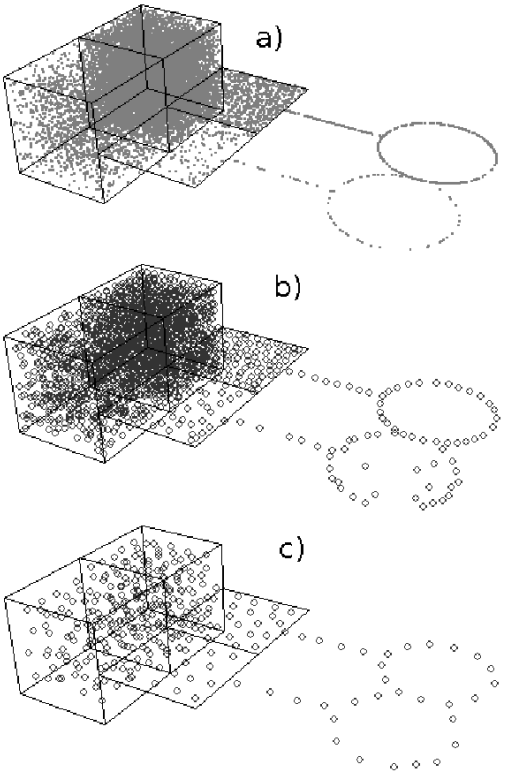

Fig. 4 The original and the modified version of GNG trained with a dataset like the one used by Cselenyi [4]. a) Training data. b) Algorithm of GNG by Fritzke. c) Modified GNG algorithm, $qE = 0.1$, $h = 0.1$, $sp = 0.5$.

there are more units to evaluate each time a new point is evaluated, and this is not desirable if we have a very large dataset. Moreover, we apply vector quantization in order to reduce the number of points to process by choosing a suitable codebok, and therefore, redundant prototypes are not desirable. This section shows how the proposed modification to controlling prototype proliferation allows us to overcome this situation. Two experiments with controlled toysets should help in the testing and understanding of the modified algorithm.

Figure 3 shows the results of the modified algorithm when trained with the data distribution showed in figure 1. Figure 3 shows how parameter h effectively controls the proportion of units assigned to regions with high and low densities. In this case the parameters qE and sp were kept constant ($qE = 0.1$ and $sp = 0.75$) since their effects are more global and depend less on the data distribution. The rest of the parameters were set as shown in table 2.

Another interesting test consists of using a dataset similar to the one proposed by Martinetz [13] in the early implementations of this kind of network (neural gas,

growing cell structures, growing neural gas) [5]. This distribution of data has been used by several researchers [4, 5, 7, 12, 13] in order to show the ability of the *topology-preserving* networks in modelling the distribution and dimensionality of data. The generated dataset shown in figure 4 a) presents two different levels of densities for points situated in three, two and one dimension, and has points describing a circle.

When this dataset is used, the model has to deal with data having different dimensionalities, different densities and different topologies. Figures 4 b) and 4 c) show the position of the units of two GNG networks, one of them using the original algorithm and the other one using the modified version. Both structures preserve the topology of the data in terms of dimensionality by placing and connecting units depending on local conditions. Conversely, the two models behave differently in terms of the distribution of the data. The codebook of the original GNG algorithm reproduces the distribution of the training data by assigning almost the same quantity of data points to each vector prototype. In the case of the modified version, the parameter h set to 0.1 makes the distribution of prototypes more uniform due to the fact that the insertion of new units is conditioned with the quantization error. Other parameters were set as shown in table 2.

5 Obtaining a Codebook from a Large Dataset

This section summarizes a series of experiments using a large database of climate. The database contains information of the temperature (minimum, average, and maximum) and the precipitation over approximately the last fifty years in Colombia, with a spatial resolution of 30 seconds (≈900m) (WORLDCLIM) [11]. This database is part of a cooperative project between Colombia and Switzerland named "Precision Agriculture and the Construction of Field-Crop Models for Tropical Fruits", where one of the objectives is finding geographical zones with similar environmental conditions, in order to facilitate the implementation or migration of some crops. There are 1,336.025 data points corresponding to the amount of pixels covering the region, and each one has twelve dimensions corresponding to the months of the year (i.e. one vector of twelve dimensions per pixel), and each month has four dimensions corresponding to the aforementioned variables.

Processing the whole dataset[3] implies the use of a lot of memory resources and takes hours of calculation. Moreover, the situation could get even worse if we consider the use of the whole set of variables at the same time. Therefore, instead of processing every pixel in the dataset, we could use vector quantization to extract a codebook representing the data, and then to process this set of prototypes finding the zones that have similar properties.

[3] Finding zones with similar environmental conditions (i.e, temperature) by means of some measure of distance.

5.1 Incremental Learning of a Non-incremental Task

The first test was done by taking only the data corresponding to the average temperature from the dataset described in section 5. The resulting dataset has 1,336.025 observations corresponding to the amount of pixels on the map, and each one has twelve dimensions corresponding to the months of the year. Figure 5 shows the resulting quantization errors using both algorithms.

Both neural networks have only 89 neuron units, which means having a codebook with only 0.007% of the original size of the dataset. Nonetheless, the quantization error is astonishingly low. This reduction is possible due to the low local dimensionality of the data, and the low range of the variables. Figure 5 shows that the modified algorithm presents quantization error values that are comparable to those from the original version, but with a slightly different distribution.

Having a dataset which allows a representation over two dimensions has some advantages. In this case, we can draw some information from the geographic distribution of the prototypes. Figure 6 shows the geographic representation of the boundaries (white lines) of the Voronoi region of each prototype. The region delimited with a circle is a wide plain at low altitudes which presents homogeneous conditions in terms of temperature. Therefore, this large amount of pixels belongs to a high density zone in the space of twelve dimensions of our data. In this case, this high density zone does not mean more information to quantize. However, the original GNG algorithm is "forced" to proliferate prototypes due to its property of distribution matching. Thus incurring in a high computational cost when one uses

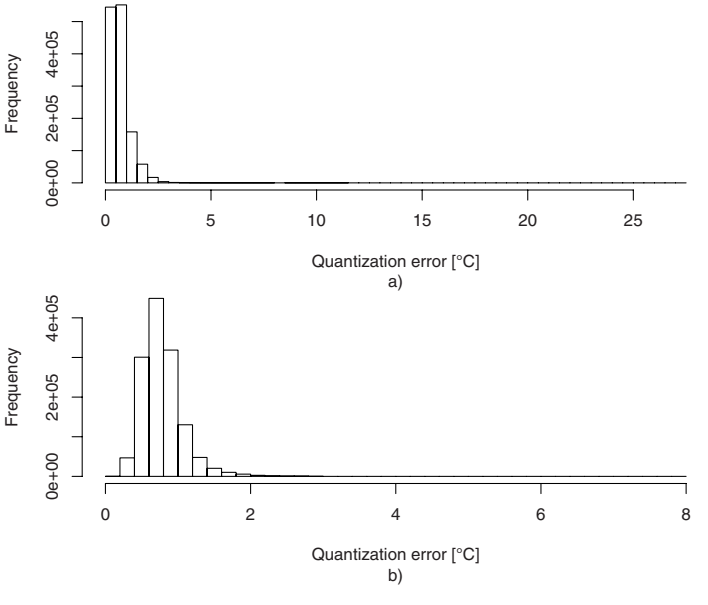

Fig. 5 Histogram of the quantization error for a large dataset. a) Fritzke's original GNG algorithm. b) Modified GNG algorithm, $qE = 1\,^{\circ}\mathrm{C}$, $h = 0.1$, $sp = 0.5$.

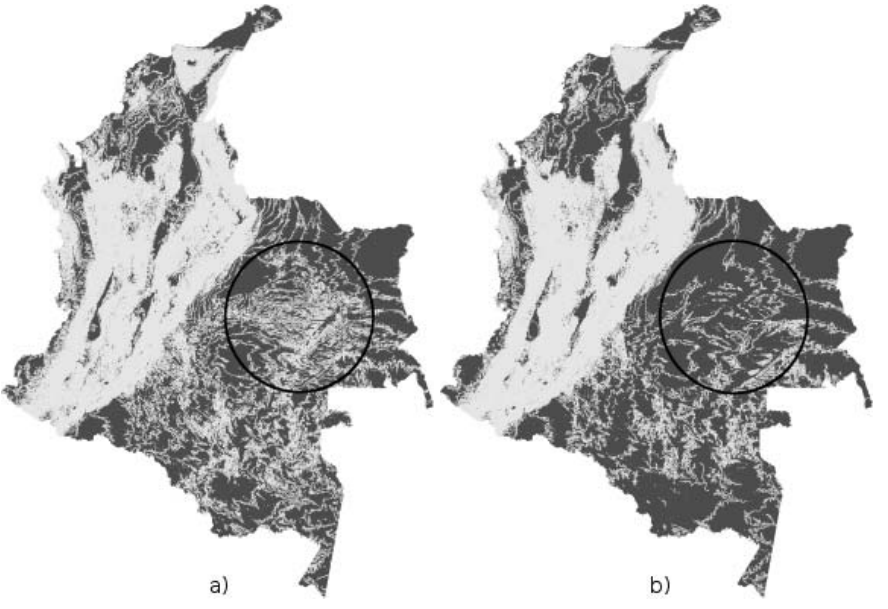

Fig. 6 Prototype boundaries. a) Original algorithm. b) Modified algorithm.

the resulting model. Instead of representing better these areas, our approach is to avoid prototype proliferation in regions with regular conditions in order to better represent heterogeneous zones (e.g., mountains). Figure 6.b) shows that the modified version of GNG places less prototypes in flat areas (i.e., high density regions) than the original version (Figure 6.a), and assigns more prototypes (i.e., cluster centres) to the lower density points belonging to mountain areas (i.e., low density regions).

5.2 Incremental Learning of an Incremental Task

Even if the dataset used in section 5.1 was a large one, and its codebook was extracted by using an incremental algorithm, it was not processed by using the incremental approach. In this case, the whole dataset was used for training, and therefore, the algorithm had access to every point in the dataset at every iteration. Incremental modelling, instead, proposes dividing the dataset, and training on an incremental

Table 4 Parameters for the modified Growing Neural Gas algorithm.

Parameter	ε_b	ε_n	λ	a_{max}	α	d	$qError$	h	sp
value	0.05	0.005	250	1000	0.5	0.9	0.1	0.1	0.5

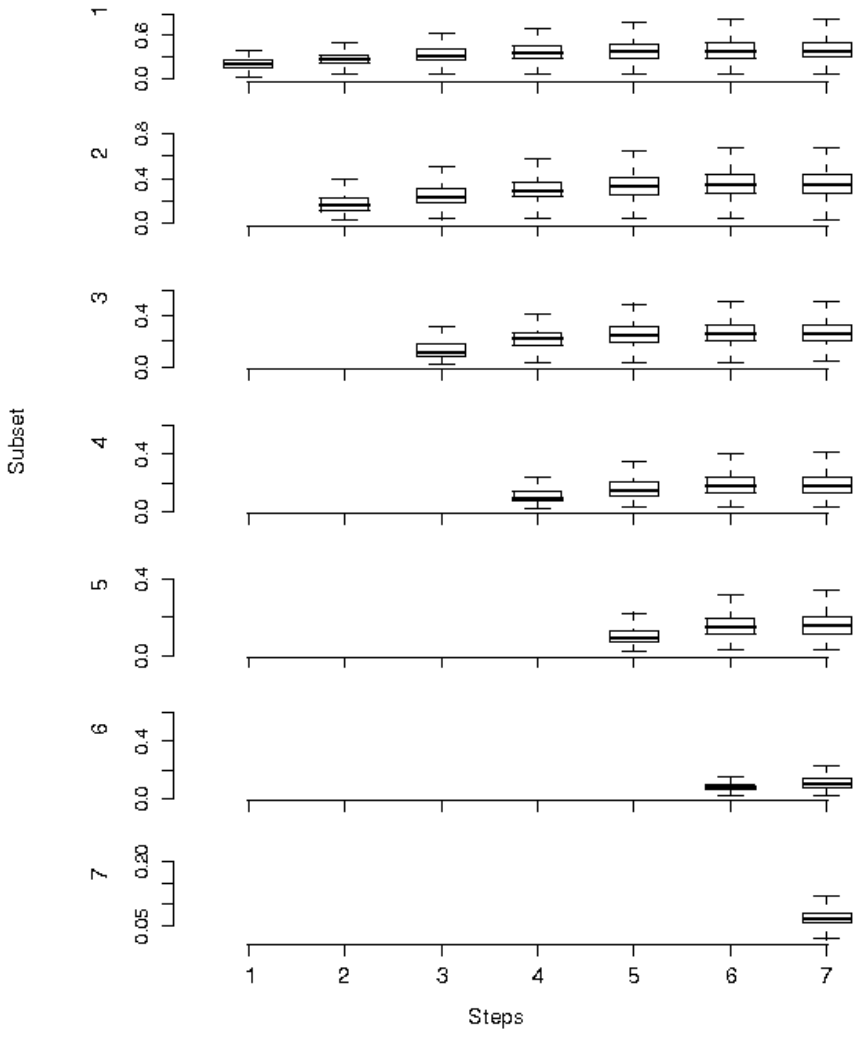

Fig. 7 Error distributions after each incremental step. Each line shows the evolution of the quantization error for a given subset when adding new knowledge to the model.

learner by using the individual parts. This scenario was tested by using the complete dataset mentioned in section 5.

The complete dataset of climate with 1,336.025 data points and forty-eight dimensions was divided in an arbitrary way (i.e. from north to south) into 7 parts; six parts of 200.000 observations, and one final part having 136.025 data points. These individual subsets were used in order to incrementally train a model using our modified version of the GNG algorithm. The parameters used are shown in table 4.

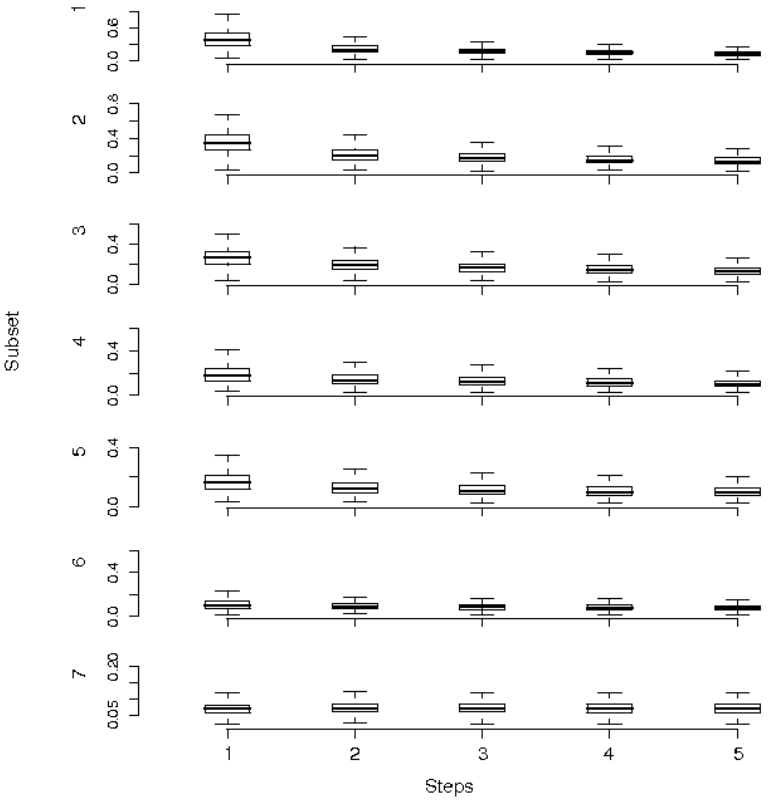

Fig. 8 Error distributions after each training-repetition step. Each line shows the evolution of the quantization error for a given subset.

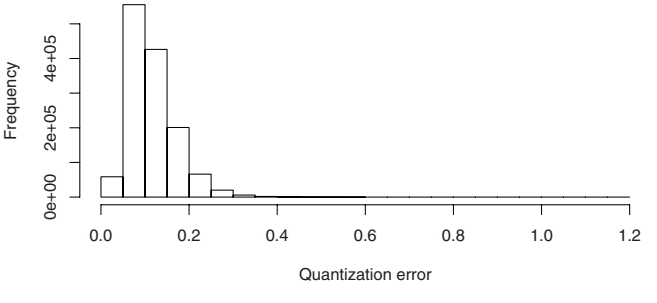

Fig. 9 Histogram of the quantization error for codebook obtained with the modified version of GNG, after feeding in the whole climate dataset

Figure 7 shows the evolution of the quantization error after adding each one of the seven subsets. Each horizontal line of boxplots represents the error for a given subset, and each step means the fact of training the network with a new subset. As

Fig. 10 Prototype boundaries of the final GNG network projected on the geographic map.

can be seen in figure 7, the quantization error for a subset presented in previous steps increases in further steps with the addition of new knowledge. Such behaviour suggests that, even if the GNG algorithm performs incrementally, the addition of new data can produce interference with the knowledge already stored within the network. This undesirable situation happens if the new data are close enough to the existant prototypes to make them move, forgetting the previous information.

In order to overcome this weakness of the algorithm, a sort of repetition policy was added to the training procedure, as follows. The GNG network was trained several times with the same sequence of subsets, from the first to the seventh subset. At each iteration[4] the algorithm had two possibilities: choosing a new observation from the input dataset, or taking one of the prototype vectors from the current codebook as input. The former option, which we named *exploration*, could be taken with a

[4] Each time a new point is going to be presented to the network.

pre-defined probability. Figure 8 shows the evolution of the quantization error for five training-repetition steps of the algorithm, using an exploration level of 50%.

As can be seen in figure 8, the quantization error for each one of the seven subsets decreases with each iteration of the algorithm. Such behaviour means that the GNG network is capable of avoiding catastrophic forgetting of knowledge when using repetition. Moreover, after five iterations the network reaches a low quantization error over the whole dataset. This result is shown in figure 9.

The resulting network has 14.756 prototype vectors in its codebook, which represents 1.1% of the total amount of pixels in the database. The number of prototypes is larger than in the case of the temperature because of the larger dimensionality of the observations (i.e. forty-eight dimensions instead of twelve). Moreover, precipitation data have a wider range than temparature, increasing the area where prototype vectors should be placed.

Figure 10 shows the Voronoi region of each prototype projected over the two-dimensional geographic map. As in the previous case, one can see that prototypes are distributed over the whole map, and they are more concentrated in mountain zones, as desired.

Finally, after quantizing the dataset of 1,336.025 observations, the sets of similar climate zones could be found by analyzing the 14.756 prototypes obtained from the GNG network. This compression in the amount of data to be analyzed is possible due to the existance of redundancy in the original dataset. In other words, pixels with similar characteristics are represented by a reduced number of vector prototypes, even if they are located in regions which are not geographically adjacent.

6 Conclusions

Nowadays, there is an increasing need for dealing with large datasets. A large dataset can be split or sampled in order to divide the modelling task into smaller subtasks that can be merged in a single model by means of an incremental learning technique performing vector quantization. In our case, we chose the Growing Neural Gas (GNG) algorithm as the vector quantization technique. GNG allows us to get a reduced codebook to analyse, instead of analysing the whole dataset. Growing Neural Gas is an excellent incremental vector quantization technique, allowing us to preserve the topology and the distribution of a set of data.

However, in our specific application, we found it necessary to modulate the topology matching property of the GNG algorithm in order to control the distribution of units between zones with high and low density. To achieve this, we modified the original algorithm proposed by Fritzke by adding three new parameters, two controlling the quantization error and the amount of neuron units in the network, and one controlling the distribution of these units. The modified version still has the property of topology-preservation, but contrary to the original version it permits the modulation of the distribution matching capabilities of the original algorithm. These changes allow the quantization of datasets having high contrasts in density

while keeping the information of low density areas, and using a limited number of prototypes.

Moreover, we tested the modified algorithm on the task of quantizing a heterogeneous set of real data. First, the difference in the distribution of prototypes between the original and the modified version was tested by using the classical modelling approach where the whole set of data is available during the training process. By doing so, we verified that the modified version modulates the insertion of prototypes in high density regions of data. Finally, we used the modified version of the algorithm to perform an incremental modelling task over a larger version of the former dataset. A repetition policy had to be added to the incremental training procedure in order to carry out this test. This repetition strategy allowed the GNG network to remember previous information, preventing catastrophic forgetting caused by new data interfering with the already stored knowledge.

Acknowledgements. This work is part of a joint project between BIOTEC, CIAT, CENICAÑA (Colombia) and HEIG-VD (Switzerland) named *"Precision agriculture and the construction of field-crop models for tropical fruits"*. The financial support is given by several institutions in Colombia (MADR, COLCIENCIAS, ACCI) and the State Secretariat for Education and Research (SER) in Switzerland.

References

1. Alpaydin, E.: Introduction to Machine Learning. MIT Press, Cambridge (2004)
2. Bouchachia, A., Gabrys, B., Sahel, Z.: Overview of Some Incremental Learning Algorithms. In: Fuzzy Systems Conference. FUZZ-IEEE 2007, vol. 23-26, pp. 1–6 (2007)
3. Bradley, P., Gehrke, J., Ramakrishnan, R., Srikant, R.: Scaling mining algorithms to large databases. Commun. ACM 45, 38–43 (2002)
4. Cselenyi, Z.: Mapping the dimensionality, density and topology of data: The growing adaptive neural gas. Computer Methods and Programs in Biomedicine 78, 141–156 (2005)
5. Fritzke, B.: Unsupervised ontogenic networks. In: Handbook of Neural Computation, ch. C 2.4, Institute of Physics, Oxford University Press (1997)
6. Fritzke, B.: Goals of Competitive Learning. In: Some Competitive Learning Methods (1997), `http://www.neuroinformatik.rub.de/VDM/research/gsn/JavaPaper/` (Cited October 26, 2008)
7. Fritzke, B.: A Growing Neural Gas Learns Topologies. In: Advances in Neural Information Processing Systems, vol. 7. MIT Press, Cambridge (1995)
8. Ganti, V., Gehrke, J., Ramakrishnan, R.: Mining very large databases. Computer 32, 38–45 (1999)
9. Giraud-Carrier, C.: A note on the utility of incremental learning. AI Commun. 13, 215–223 (2000)
10. Heinke, D., Hamker, F.H.: Comparing neural networks: a benchmark on growing neural gas, growing cell structures, and fuzzy ARTMAP. IEEE Transactions on Neural Networks 9, 1279–1291 (1998)

11. Hijmans, R., Cameron, S., Parra, J., Jones, P., Jarvis, A.: Very High Resolution Interpolated Climate Surfaces for Global Land Areas. Int. J. Climatol 25, 1965–1978 (2005)
12. Martinetz, T., Schulten, K.: Topology representing networks. Neural Networks 7, 507–522 (1994)
13. Martinetz, T., Schulten, K.: A neural gas network learns topologies. Artificial Neural Networks, 397–402 (1991)