Following non-stationary distributions by controlling the vector quantization accuracy of a growing neural gas network

Hervé Frezza-Buet

Supélec, 2 rue Édouard Belin, F-57070 Metz, France

Abstract

In this paper, an original method (GNG-T) extended from Growing Neural Gas [6] is presented. The method performs continuously vector quantization over a distribution that changes over time. It deals with both sudden changes and continuous ones, and is thus suited for the video tracking framework, where continuous tracking is required as well as fast adaptation to incoming and outgoing people. The central mechanism relies on the management of the quantization resolution, that copes with stopping condition problems of usual Growing Neural Gas inspired methods. Application to video tracking is presented.

Key words: vector quantization, non stationary distribution, video tracking, growing neural gas

1 Introduction

Unsupervised learning consists in mapping the distribution of some phenomenon with some reduced information, like principal components in PCA (Principal Components Analysis) for example. In this paper, analysis by vector quantization is rather addressed. This consists in finding a finite set of vectors that are located in order to cover places where the phenomenon happens. Representing an unknown continuous density probability function by few vectors reduces the information and allows us to analyse, compress or represent the complexity of the problem.

* Many thanks to Georges Adrian Drumea for having started the design of GNG-T, and to the Region Lorraine (France) for substantial support to this work. Email address: Herve.Frezza-Buet@supelec.fr (Hervé Frezza-Buet).
In this paper, that is an extended version of [4], unsupervised learning by vector quantization is applied to non stationary distributions, which requires to adapt the usual techniques to this issue. The problem with non stationary distributions is that the process should keep a high level of plasticity, in order to adapt to sudden changes, as well as being able to stabilize on constant or smoothly changing parts, which rather requires stability. The algorithm proposed here deals with both these features, and can thus be applied to video scene analysis, since objects in a scene can appear, disappear, move continuously or just be motionless.

2 Vector quantization and non-stationary distributions

Since a decade, many methods have been proposed for vector quantization. Some of them are related to information theory and signal processing, as reviewed in [27], but more recent ones deal with the mapping of a distribution by a finite set of vectors that try to cover at best a continuous probability density function. These latter are reported in [3,16]. Let us first sketch the basic elements of any vector quantization procedure, and then show how it has been extended to deal with non-stationary distributions.

2.1 Vector quantization basics and notation

In order to introduce vector quantization (VQ), let us recall some of its features and define some notations as in [8]. Let \( \xi \in X \) be a sample of an unknown distribution \( P_X \) over space \( X \), according to a stationary density of probability \( p(\xi) \). Vector quantization consists classically in finding a discrete set \( \{w_i\}_{1 \leq i \leq n} \subset X \) of prototypes such that this set “matches” \( P_X \). The so called neuron \( i \) is the computational element that gathers information related to the \( w_i \) prototype. Let \( w(\xi) \) be \( \text{argmin}_{w_i} \{d(\xi, w_i)\} \), where \( d \) is some proximity function (usually \( d(\xi, w_i) = \|\xi - w_i\|^2 \)), less restrictive than pure distances, returning low values for similar arguments and higher values for less similar arguments. The quality of the fitting depends on how well the prototypes \( \{w_i\} \) are scattered over the distribution. More formally, this scattering has to minimize distortion defined in equation 1, where \( V_i \) is the Voronoï cell around \( w_i \) (i.e. the region in space where points are closer to \( w_i \) than to any other \( w_j \)). The \( V_i \)s form a partition of \( X \).

\[
E = \int_X d(w(\xi), \xi)p(\xi)d\xi = \sum_{i=1}^{n} E_i
\]

where \( E_i = \int_{V_i} d(w_i, \xi)p(\xi)d\xi \) and \( V_i = \{\xi \in X : w(\xi) = w_i\} \)
The minimization of $E$ is performed by successive stages, until some stopping condition is met. Some methods proceed minimization from a given finite set of examples, chosen from $P_X$, and minimizes the distortion measured on this set [15,11]. Some other methods works on-line, since examples are provided continuously. In this latter case, at each stage, an input $\xi$ is first chosen, according to $P_X$. Second, a so called winner-take-all procedure (WTA) allows to determine the winning prototype $w_i = w(\xi)$. Third, $w_i$ is modified so as to be closer to $\xi$.

The previous procedure describes the reduction of distortion by placing vectors in the input space. For many algorithms in the literature, the feature of topology preservation is also addressed. This consists in endowing the prototypes with a graph structure, that defines a neighbourhood relation between them. Preserving topology, roughly speaking, means ensuring that two prototypes that are close in the graph are also close according to the metrics $d$ in the input space. Early self-organizing maps by Kohonen [14] have this feature since they try to map an a priori graph (usually a grid) to the distribution in the input space. This distribution can have a dimension higher than two. In this case, the resulting grid of prototypes is distorted to fit the distribution. The work presented here is rather inspired by the Growing Neural Gas (GNG) algorithm by Fritzke [6] since it adapts to the actual topology of the distribution by building a suited graph with the prototypes. This prevents from distorting an arbitrary low dimension graph (as a grid) if some higher dimension connectivity is needed. The idea is, each time an example is given, to link the two best matching prototypes. This method, called Competitive Hebbian Learning (CHL), has been introduced by Martinez and Schulten. Authors have demonstrated that it builds a graph that approximates the Delaunay triangulation of the prototypes [17]. One other interesting feature of GNG is its incremental nature. Each prototype $w_i$ has an accumulation variable with it that measures the sum of the errors $w_i$ makes when it actually wins. This accumulated value is used to add neurons where quantization is not accurate enough (i.e. local distortion is too high).

Accumulation of local error, an incremental algorithm, and CHL are of primary relevance in our approach, that directly inherits these features from the GNG algorithm. Some refinements have been proposed in the literature, in order to accelerate the growing of the network [24] or manage the unstability induced by outliers in the distribution [23]. Some other approaches modulate the spatial sensitivity of a prototype, using some variable radius receptive field, to allow different accuracies in the network [1]. In this latter case, adaptation and growth is driven by a value that is a goal to be reached by local errors. Such a criterion is very close to the target that is used to drive the growth of our algorithm.

Last, in [24], the preservation of topology is analyzed numerically. The authors
stress that, for a given distribution, the quality of the topology preservation increases with the number of prototypes until some bound is reached. More prototypes, then, will not improve the preservation. This bound depends on the probability density function $p$ itself, and it may differ drastically from one distribution to another. This is why setting in advance the number of prototypes is difficult, and difficulty increases if the distribution is subject to change with time. This is why, as explain in next section, controlling continuously the number of prototypes is the main difficulty when using vector quantization for non stationary distributions.

2.2 Endowing VQ with the ability to track non-stationary distributions

As explained in [8], algorithms whose plasticity decreases with time, as for example the self-organizing maps, are not suitable for non stationary distributions. What is less intuitive is that even incremental neural networks, where all parameters are constant, encounter difficulties when fed with changing distributions. They are actually able to create new prototypes if needed, but the problem is that some neurons may become useless (dead units) in case of a sudden change. So the challenge is to exploit the properties of algorithms like GNG, while controlling the additions and removals of neurons, in order to stick to the changes in the distribution. This requires computing some kind of statistics for each prototype, in order to detect when it needs to have one more neighbour or, rather, that it needs to be removed.

Let us first consider the problem of removing useless prototypes. Some utility measure as been proposed by Fritzke [7], and adapted to GNG [8]. The resulting GNG-U algorithm uses two accumulators for each prototype. One accumulator stores the sum of errors made when the prototype actually wins, as in GNG. The second stores the 'utilities' of a prototype when it wins as well. This utility is the difference between the error made by the second closest prototype and the error made by the winning prototype itself. If this is low, it means that if the winning prototype were missing, distortion would have been approximately the same. Both accumulators for all prototypes are subject to a continuous exponential decrease, in order to enhance the influence of recent events. Such exponential decays are often added for the management of non stationary processes, acting as some decaying temporal traces of past events. With GNG-U, removal occurs if some heuristic criterion, based on the values of all accumulators, is fulfilled.

The second problem is the growth of the network, that occurs every $\lambda$ steps in GNG. With GNG-U, the growth is stopped when some arbitrary upper bound is found. This is why removal is crucial since when this bound is reached, new neurons can be created only if some other are removed. This feature of GNG-
U is also common to many other algorithms. For example, let us mention the BNN (Bi-Counter Neural Network) that deals with a fixed number of units [28], but modulates the learning rates for each prototype. If a prototype wins frequently, its learning rate will decrease, as in the self-organizing maps. Nevertheless, if some change in the distribution occurs, and if this prototype does not win, some decaying traces drive the learning rates and the removals [16], keeping the number of neuron limited to an arbitrary value as well. Some other algorithms like RCN (Representation-burden Conservation Network) consider that representation (i.e low distortion) is a constant value that has to be shared by the prototypes [29]. Each prototype has a “representation burden” that is updated when it wins or looses, so that the total amount of burden is kept constant. Maintaining a value to a constant level while dealing with a non stationary process is also what is used in the GNG-T algorithm presented in this paper.

To sum up, a vector quantization procedure like GNG has to be extended for dealing with non stationary distribution with some algorithmic tools for probabilistic evaluation, and some decision about growing and removal has to be taken, often from an heuristic criterion. Some of these extensions are complex or memory consuming. For example, the LOJ (Law of the jungle) criterion [19,20] requires storing for each prototype a table of examples. Our goal with the GNG-T procedure is to avoid heuristics and keep the algorithm as simple as possible, since it is only slightly different from the original GNG. This is detailed in next section.

3 The GNG-T algorithm

3.1 Main ideas

The idea underlying the design of GNG-T (GNG with targeting) comes from considering equation 1. When $E = \sum_{i=1}^{n} E_i$ is minimal, all the $E_i$ reach the same value, denoted $T$. GNG-T drives the quantization in order to control this value, that is thus viewed as a target to be reached. Let us consider an epoch of $N$ examples chosen from the distribution, in order to estimate $T$. Let us note $\{\xi_j\}_{1 \leq j \leq n}$ the $n_i$ examples for which $w_i$ has won. First, when a prototype $w_i$ wins for an example $\xi$ (i.e $\xi \in V_i$, the Voronoi cell around $w_i$) the quantization error $d(w_i, \xi)$ can be easily added by the neuron $i$ in an accumulator $e_i = \sum_{1 \leq j \leq n} d(\xi_j, w_i)$, as in GNG. The quantity $e_i/n_i$ estimates $E_i$, given by equation 2.

$$E_i = \int_{V_i} d(\xi, w_i)p(\xi/\xi \in V_i)d\xi = \left(\int_{V_i} d(\xi, w_i)p(\xi)d\xi\right) / \left(\int_{V_i} p(\xi)d\xi\right)$$  (2)
This leads immediately to equation 3, where expression on the right is the actual contribution of the Voronoï cell $V_i$ to the overall distortion.

$$\bar{E}_i \int_{V_i} p(\xi) d\xi = \int_{V_i} d(\xi, w_i) p(\xi) d\xi = E_i$$  \hspace{1cm} (3)$$

So, during an epoch of $N$ successive inputs to the algorithm, we can estimate $E_i$ from equation 3, as in equation 4.

$$\bar{E}_i \approx \frac{1}{n_i} \sum_{j=1}^{n_i} d(\xi^i_j, w_i) = \frac{e_i}{n_i} \text{ and } \int_{V_i} p(\xi) d\xi \approx n_i/N$$

$$\Rightarrow E_i \approx \frac{1}{N} \sum_{j=1}^{n_i} d(\xi^i_j, w_i) = \frac{e_i}{N}$$  \hspace{1cm} (4)$$

Let us stress here two points. First, equation 4 shows that a neuron $i$ does not need $n_i$ to estimate $E_i$, but it has to know the epoch size $N$. Second, the core of GNG-T is to use this estimation of $E_i$ to drive the growth of the network. As all the $E_i$s reach the same value at equilibrium, this common value $T'$ can be compared to some desired target $T$. If $T$ is lower than $T'$, quantization is not accurate enough, and more neurons have to be added. On the contrary, if $T$ is greater than $T'$, the current quantization is too much accurate, and some neurons have to be remove so that Voronoï cells of remaining ones become wider. This is what is presented in [4], but some refinements are added here.

Let us note that factor $1/N$ in equation 4 acts as a normalization, making computation independent from the actual epoch size $N$ that is used to sample the distribution. Therefore, a given value $T$ leads to the same accuracy in the quantization of a density $p$, whatever the epoch size used for sampling. This is suitable for a stationary distribution, since changing epoch size $N$ only changes the number of samples to estimate $p$. Nevertheless, when $p$ actually changes over time, the relation between the value $T$ and quantization accuracy may change. For example, let us consider a distribution made of a single cluster, and only one prototype $w_1$ at its center. Let suppose that we use $N = 10$, and we find $T' = 100$ (i.e $e_1 = 1000$). If we use $N = 20$, we may find $e_i = 2000$, since twice as many examples are provided in the cluster, and $T' = 100$ is kept. Let us now add a new cluster in the distribution, far from the previous one, but with the same shape. Let us also put a prototype $w_2$ for it, at its center. The accuracy is the same as previously: one prototype per cluster. Nevertheless, for $N = 10$, we will have only 5 examples in first cluster, and 5 in the second one. So $e_1 = e_2 = 500$, and $T' = 50$. It means that the target $T$ has to be divided by two to keep the accuracy constant. In more general cases, having the accuracy obtained for a value $T$ dependent on the shape of the distribution is not convenient.

Such consideration has led us to remove the $1/N$ factor. It means that we
control the network size so that each $e_i$ reaches the $T$ parameter. For this to lead to a reliable non stationary quantization, the epoch has to be chosen carefully. It should actually correspond to a time interval that is meaningfully in the application context. For example in a video analysis framework, an epoch could be a fixed number of successive frames. In some other process, an epoch could be a constant duration of some example collection. During the epoch, if many examples $\xi$ are given by the process whose distribution is analyzed, the $e_i$s will have high values, and the system will tend to reduce them to $T$ by adding more neurons. If during some other epoch, the distribution has changed so that it produces fewer examples, the $e_i$s will decrease and some neurons will have to be removed to make them grow and reach $T$. So if all epochs are comparable, in the sense of the semantics of the problem, having more or less examples correspond to an actual change in the distribution, and normalizing by epoch size $N$ should be avoided. Let us mention that the use of epochs to deal with statistics has also been proposed in the BNN algorithm [28].

The choice of the actual value for $T$ is crucial when GNG-T is applied, since $T$ drives the accuracy of the quantization. The numerical value required to obtain a desired accuracy depends on what is considered as being an epoch. As each $e_i$ theoretically reaches $T$ during an epoch, $T$ can be set from considering what a desired quantization should be, and for one prototype of it, what value is reached by the sum of the error it makes during an epoch. This sum is the actual $T$ value. As any value of $T$ leads to a stable quantization, one can also tune $T$ empirically, from successive tries, starting from high values i.e. sparse quantization.

To sum up, we have shown that, if an epoch consists of the interval between meaningful events, accordingly to the problem, and if these epochs are comparable, the number of examples given during an epoch has to be considered, since it may indicate that the distribution probability changes. This is easily done by only using the accumulated errors $e_i$s at each prototype, during the epoch, to control the total number of prototypes.

3.2 Implementation

The GNG-T algorithm consists in processing successive epochs. When an epoch starts, it is given a graph of $\nu$ nodes, resulting from previous computation. At beginning, an empty graph ($\nu = 0$) has to be provided. The computation of an epoch of size $N$ is then the following, where $\star$ denotes steps that are different from original GNG. The parameter $T$ corresponds to the actual target, $\alpha_1 > \alpha_2$ are two positive learning rates, and $A$ is the age limit of the edges.
Initialization A new epoch starts. Make sure that the graph has two nodes.
If it is not the case, and if some examples are actually available, choose 2
prototypes \( w_1 \) and \( w_2 \) initialized according to \( p \). \( n \leftarrow 0 \).

Step* 1 Reset \( e_i \) to 0 for all neurons \( i \).
Step* 2 if \( n = N \), then the epoch is completed, go to step 8. Else, go to next
step.
Step 3 \( n \leftarrow n + 1 \), Get input \( \xi \) according to \( p \), and determine the winning
prototype \( w_{i_1} \) and the second closest prototype \( w_{i_2} \), among the \( w_i \)s.
Step 4 Create (or refresh) an edge between \( w_{i_1} \) and \( w_{i_2} \) with 0 age. Increment
the age of all the edges from \( w_{i_1} \), and remove the ones older than \( \lambda \).
Step 5 Update accumulator: \( e_{i_1} \leftarrow e_{i_1} + d(\xi, w_{i_1}) \).
Step 6 Update weights as follows, index \( j \) denoting the neighbours of \( w_{i_1} \).
\[
w_{i_1} \leftarrow w_{i_1} + \alpha_1(\xi - w_{i_1}), \quad \forall j \quad w_j \leftarrow w_j + \alpha_2(\xi - w_j)
\]
Step 7 Go to step 2.
Step* 8 Remove nodes that have not won.
Step* 9 Compute \( \bar{e} = \frac{1}{\nu} \sum_{i=1}^{\nu} e_i \), the average node distortion.
Step* 10 If \( T < \bar{e} \), go to step 11, else go to step 12.
Step 11 Quantization needs to be more accurate. Determine \( w_a \) the proto-
type \( w_i \) with strongest \( e_i \), and find among its \( w_j \) neighbours the prototype
with strongest \( e_j \). Let us call \( w_b \) this prototype. Remove edge between neu-
ron \( a \) and \( b \), add neuron \( c \) with prototype \( w_c = 0.5 \times (w_a + w_b) \), add a new
0 aged edge between \( a \) and \( c \), and between \( b \) and \( c \). Go to step 13.
Step* 12 Too much accuracy in quantization. Remove the neuron \( i \) for which
\( e_i \) is the smallest.
Step 13 End of epoch.

It can be noticed that this algorithm is straightforward, and it does not involve
any decaying activity. Moreover, the \( T \) value is easy to interpret since it is
directly the accumulated error expected from each neuron during an epoch.

4 Experiments

4.1 Description

The way GNG-T deals with topology conservation will not be discussed here,
since this algorithm has exactly the same properties as the original GNG
by Fritzke whose topological features are the ones analyzed by Martinez and
Schulten [17]. This topological feature is of primary interest in many applica-
tions, and GNG-T can be used in any applicative framework where building
a graph with prototypes is relevant. In this section, we rather analyze empiri-
cally the quantization itself in order to show how this quantization is controlled
during the evolution of a non-stationary process.
As mentioned previously, one has to define epochs to apply GNG-T, according to the nature of the problem. Let us here consider an artificial bi-dimensional distribution over a 640x480 image, driven by a probability $p(x,t)$ of the pixel $x$ to be coloured black at time $t$. At each time $t$, we build an image by applying the following procedure: each pixel $x$ is coloured black or white, according to the value of $p(x,t)$ at this pixel. Once the image is built, we compute a collection of bi-dimensional vectors, that are the actual coordinates of black pixels. This collection is shuffled, and all its elements are given successively to the GNG-T procedure. The latter defines an epoch, and so the number of examples given during an epoch depends on the probability values. Let us define a chunk as ten successive epochs, fed with the same collection of examples. Chunks will be used as units in further plotting, they play no role in the algorithm.

The experiments are the following. The numerical parameters are set to $T = 50000$, $\alpha_1 = 0.05$, $\alpha_2 = 0.005$ and $A = 50$. $T$ has been empirically tuned so that density of prototypes on figure 1 leads to readable graphics. First, 100 empty epochs are presented (i.e. 10 chunk), for representation convenience in the following graphs. Then during the 100 next chunks (chunk 110 to 210), a probability of 0.8 is set at some thick ring in the picture. During the next 100 chunks, another region is superposed to the existing one. This is a square with 0.1 probability for each pixel to be black. During the 100 next chunk, an overall probability of 0.01 is added everywhere, in order to simulate some global noise. Last, during the next 100 chunks, the ring-shaped region is removed. The distribution of examples, as well as the GNG-T graph of prototypes, are shown on figure 1.

In order to describe the whole quantization performed during an epoch, one can measure, for each neuron, the variance of the error made when it wins. This variance is $e_i/n_i$, following the notations of equation 4. From these measurements, for all neurones, let us build an histogram of last epoch of each chunk. These are shown for four chunks in figure 2.

Histograms of node variance allows us to analyse the quantization according to the sizes of the Voronoï regions. A neuron whose prototype is in a region where examples are sparse will have a high variance, as opposed to neurons quantifying a region where examples have a high density. On figure 2, it can be seen that each shape is associated to one hill in the histogram. The ring with 0.8 probability is covered with neurons whose variance is around 100 (left hill). The 0.1 probability square is given neurons with variance around 300 (hill in the middle), and the noise is sampled by neurons with high variance (more that 700). Such association of the local probability density with some region in the histogram is kept constant during the whole experiment, which is one interesting feature of GNG-T.
Fig. 1. Result of the GNG-T procedure at different chunks. The four images correspond to the status of the algorithm at four different chunk boundaries. The following graphical convention is used: small dots are the examples chosen from the distribution, and larger white dots are the nodes of the graph (i.e. the \( w_i \)). Edges of the graph are overprinted to show the CHL triangulation. Upper left is chunk 95, upper right is 195, lower left 295 and lower right 400. These represent the status of the network before changing the probability (see text).

4.2 Analysis

The experiments presented in the previous paragraph allow us to highlight the dynamical behavior of the network. To do so, let us trace during the whole experiment the mean of the \( e_i \) values in the population of neurons (see figure 3). This average value is maintained by the algorithm as close as possible to \( T \) (see steps 10, 11 and 12 in paragraph 3.2). On this figure, chunks from 0 to about 20 correspond to the creation of neurons in order to cover the ring shape with enough accuracy. Then, the network is like upper left picture in figure 1, and this state is stable until next change. When the square is added (around chunk 100), some few neurons of the graph get a high value, since they win for the whole wide square shape. The network then create new neurons to cover the square, and the mean returns to the stable \( T \) value (around chunk 120), corresponding to upper right picture of figure 1. The same behaviour can be observed when noise is added (from chunks 200 to 250), and stable state is the
Fig. 2. Histograms of the variance $e_i/n_i$ of the neurons measured at four different chunks boundaries: chunks 95 (upper left), 195 (upper right), 295 (lower left) and 400 (lower right). Variance is computed at last epoch in the chunk from distances (expressed in the pixel unit), and histogram is built by counting the number of neurons (ordinate) that have a variance in a given interval (abscissa). These correspond to respective pictures in figure 1, the layout is the same.

lower left picture of figure 1. What happens when the ring is removed is slightly different (see chunks 300 to 320). When the ring disappears, the neurons that were used for it are still there, quantifying the noise. Their $e_i$ values get low, since they have to share the sparse examples in this region of space and thus each neuron wins for very few of them. The region is over represented and the mean value $\bar{e}$ decreases. In this case, the algorithm removes the neurons having the smallest $e_i$, to retrieve the desired $T$ value (see step 12 in paragraph 3.2).

The behavior of the network when this scenario is played can also be understood by considering the evolution of variance histograms with time. Let us consider histograms on figure 2. They are instantaneous snapshots of the evolution of the graph through successive chunks. One could have built a movie from such histograms, made from snapshots taken at each chunk. Figure 4 stands for such a movie, since it illustrates the dynamics of the algorithm. It can be seen that the adding of a shape in the distribution make the variance grow (see around chunks 10, 100, 200) for a short transient duration, until stabilization is reached. The removing of the ring has the opposite effect. The neurons that were mapping the ring are too numerous for the sparse noise distribution, and their variance get transiently low, until neurons are removed and stabilization is reach. The fact that, from chunks 300 to 400, the middle hill, corresponding to the square, is conserved, suggest that the neurons that where covering this shape are preserved: they are always affected to the square,
Fig. 3. At each chunk (abscissa), $e_i$ is measured for all neurons. The maximal, minimal and mean values are plotted. When the probability changes, the mean differs suddenly from the target $T$, but the network modifies the number of neurons to bring back the mean to $T$.

Fig. 4. Evolution through time of the variance histogram. In the graph, each column corresponds to one histogram. The chunk number is reported horizontally. Vertically, values from 0 to 20 correspond to the twenty one intervals in histograms. Line 0 is the “0 to 50” interval, and line 20 is the “more than 1000” interval. Each vertical slice of the graph is the gray-scaled representation of the variance histogram in the corresponding chunk. Slices 95, 195, 295 and 400 are the actual ones that are plotted in figure 2. The whole picture shows the variance histogram evolution through time.
even if the rest of the distribution dramatically changes. This can be confirmed by seeing videos of the network, whose figure 1 is just some snapshots.

4.3 Stability versus plasticity

More generally, our experiments have shown that the neurons that are dedicated to the representation of one shape are associated to this shape until it disappears. Moreover, if the shape is sliding smoothly, the sub-graph of neurons associated to it tracks it, instead of creating new neurons on the new place and removing the previously associated ones at the previous place. This is visible on videos, and difficult to show in this paper.

From these remarks and from the observation of the behaviour of GNG-T, it can be said that this algorithm is able to adapt quickly to abrupt changes in the distribution, when shapes are added or removed suddenly. This plasticity is mandatory for dealing with non-stationary distribution. Nevertheless, it is also able to track some smooth changes while keeping the same computational resource (the neurons) associated to it. This ensures some stability of the algorithm. This is of primary importance if one needs to label the neurons, for example (objects have to be clearly separated to do so and must not cross). This tracking feature, inherited from GNG, has also been used for gesture recognition in videos [5], since when mapping the hand in one frame, the network obtained from the previous frame can be used as an initial state, in order to accelerate reaching an equilibrium.

The stability-plasticity dilemma is central in the machine learning literature, and GNG-T addresses it by exploiting the adaptivity of the original GNG for smooth changes in the one hand, while using the target $T$ to drive dramatic growths or reductions when high plasticity is needed. It can be noticed that the latter plasticity is the actual effect of some other kind of stability that keeps the mean of the $e_i$'s at a value $T$ (see figure 3).

So with GNG-T, both stability and plasticity are provided by some stabilization process. The first comes from GNG and the second is the originality of this algorithm. Both these features ensure the robustness and the reliability of GNG-T for tracking non-stationary distributions.
5 Application to video tracking

5.1 Context

In the context of video processing, and more precisely concerning the detection of elements of interest, designers often built integrated algorithms by combining a filtering process with a clustering procedure. The tracking itself consists in locating a physical object in the image, and being able to continuously update the location of this object as long as it is viewed by the video device. We refer to [13] for an overview of video tracking techniques, as well as [26] that addresses subsea video tracking, but provide a wide and recent overview on general video tracking techniques. Briefly, the idea is to separate objects of interest from the background, which is in itself a problem addressed by many studies (see [12] for a survey) to deal with luminosity variation or shadows of objects [2], and then manage the points in the image that have been associated with actual non background objects. This latter management involves for example clustering or the fitting with pre-defined models [22], and then tracking of clusters by predictive methods, like Kalman filters for example [21].

Detecting moving people can be done by using motion detection filters on the images, labeling some pixels as moving parts of the image. This is used for traffic analysis for example [10, 9], where patches of moving salient points are built from the image flow. Such preliminary stages are also performed by using content based features of ISO/MPEG-4 [18]. The difficulty is then to relate the detected pixels to the different objects in the scene. This latter stage requires clustering the pixels so that each cluster contains pixels related to each respective object. To sum up, three stages are commonly involved in video tracking system: Detection of the pixels belonging to objects of interest, then gathering of these pixels in clusters, corresponding to actual objects, and last labelling the clusters in a reliable way. GNG-T, in the applicative example presented here, is used in second stage.

Let us consider one image in the video stream, where object related pixels have been identified. They form several compact groups, one for each object. Such a pixel distribution can be mapped by a graph through a GNG-like algorithms in [24]. The contour of the groups can be extracted, and it can be reduced to some polygons, as shown on figure 5. Vertexes and edges of the polygons form a graph that can be interpreted for the analysis of the scene (number of connected components could be related to the number of objects, the curvatures of the polygons could help to make a difference between a walking human and a walking dog, etc.). The object detection process that feeds the system is detailed further, but the use of the graph is out of the scope of this paper. Concerning both of these points, let us stress here the hypothesis that obtaining such graphs from a reliable procedure could be
useful to bridge the gap between numerical analysis in the one hand, based
on filtering results over the image, and symbolic scene interpretation in the
other hand, based on the analysis of the graph to extract semantics. Such an
attempt has been made for hand gesture recognition by using directly GNG [5]
or the SGONG algorithm adapted from GNG [25], where the graph is used
to find the orientation of the hand on the picture, as well as the number of
raised fingers.

Fig. 5. Schematic description of the extraction of a graph to describe moving objects
in a video scene. From left to right: input, object detection result, expected kind of
graph built from moving pixel clustering (real results are available on next figures).

Last, let us mention that our experiments are performed in a real video surveil-
lance platform in our lab, installed for research and pedagogical purpose. One
corridor of the Supélec building in Metz (France) is equipped with three pan
tilt zoom video devices, and a screen displays in real time the processing
applied to the video streams. Visitors and students are informed about the
presence of the devices and the display, so that they can see and test the effect
of the algorithms.

5.2 Implementation

The video tracking procedure proposed here consists of two stages. The first
stage is the extraction of object related pixels from a static video device. What
is considered here as objects are the elements in the scene that do not belong
to the background. The extraction procedure labels each pixel in a 300 × 260
image with a Boolean tag, indicating whether the pixel belongs to an object or
not. This defines a Boolean image, with true patches where the image differs from the background. From this Boolean image, a set of examples is built by applying supplementary filtering. The coordinates of the pixels in the resulting image are, for each frame, the examples that are used during an epoch to feed a GNG-T procedure. For each frame, the set of data is used to execute 20 epochs. Different filtering procedures are presented next in this paragraph. Let us first detail in the following how the objects are detected and how the graph is used.

First object detection is made by comparing two RGB color images. The first one is image \( B \) coded with floating point RGB components, and being the result of a first order recursive filtering, as in [9]. This filtering is fed by a 24 bit RGB image \( F \), that is the current frame. The update of the background is given by equation 5, using \( \alpha = 0.0005 \), applied to each component of each pixel.

\[
B \leftarrow (1 - \alpha)B + \alpha F
\]  

Then, the pixel per pixel comparison between the frame \( F \) and background \( B \) is performed to build the object detection Boolean image. This comparison is based on the hue of pixels, in order not to be sensitive to luminosity variations induced by changes in overall lighting or shadows of the objects. Let \( p_B \) and \( p_F \) be two corresponding pixels in background \( B \) and current frame \( F \). \( p_B \) and \( p_F \) are three dimensional vectors (red, green and blue components that are in \([0, 255]\)). Let us note \( N^2 = 3 \times 255^2 \) the maximal possible value of the squared norm of a pixel. If only one of the \( \|p_B\|^2 \) and \( \|p_F\|^2 \) is lesser that \( \beta N^2 \) (i.e. is too dark), the frame and the background are considered different, since color cannot be obtained reliably from the dark pixel. If both are lesser that \( \beta N^2 \), background and foreground are considered to be the same. In other cases, we consider that \( p_F \) belongs to an object if \( \cos(p_F, p_B) < \gamma \), since cosine provides an illumination-proof color detection. The values \( \gamma = 0.997 \) and \( \beta = .3 \) are used in our experiments.

The GNG-T procedure provides continuously a graph of neurons as an output. This graph is expected to map objects, as illustrated on figure 6, 7 and 8. In real videos, when performing a pixel per pixel comparison to detect objects, one has to cope with noise, due to the sensor itself, but also to some small changes in the scene and JPEG quantization. Such noise is also mapped by the GNG-T procedure, as illustrated in lower left and right pictures in figure 1. As only a mapping of objects is desired in this context, nodes and edges mapping the noise should be removed. This can be done by removing edges above a certain length (a threshold of 20 pixels length is used here), or by removing neurons with wide Voronoï cell, i.e. the neurons for which \( e_i/n_i \) is above a threshold. This latter solution is not used in our experiments since results were identical to the former solution.

In our experiment in a real corridor, tracking has to cope with many problems,
also reported in [12]. First, the floor has a ‘skin’ color, which makes faces and hands be considered as background with our color comparison mechanism. Second, glasses on both sides, as well as metal frames produce reflections, that are detected as objects in the foreground. Third, the dark horizontal bars in the background prevent from detecting the head of dark haired people when heads and bars are superposed in the image. Last, as mentioned previously, our video devices sends JPEG images with interlacing, which produces noise in the colors, at the edge of objects mainly. All these problems explain why GNG-T is provided with sometimes altered shapes of bodies in figures 6, 7 and 8, but solving these problems requires improving background detection, which is a major problem in tracking (see discussion in [13]). This latter point is out of the scope of this paper.

Fig. 6. Application of GNG-T to video tracking. Parameters are $A = 20$, $\alpha_1 = 0.05$, $\alpha_2 = 0.05$ and $T = 300$. The result of the object detection is provided directly to GNG-T. First line contain video frames (time goes from left to right), the second line contain respective results of background suppression, and third line is the result of GNG-T clustering.

On figure 6, the reported experiment consists of providing directly the result of the background suppression to GNG-T. As noise is much sparser than object blobs, prototypes are dense on objects only. This allows to build a
graph without any filtering. The drawback is that the growth of the graph until equilibrium is reached is slower, since prototypes have to be numerous enough to cover the entire blob surface. On figure 7, we aim at clustering the contours of objects. As both noise and contour are lines, the sparseness argument used in previous experiment do not stand anymore: both noise and object contours will be mapped with the same density of prototypes. The solution used here is to clean the background suppression result (line 3 on figure 7) by some fast morphological filtering, and then use the morphological contour of the remaining points (line 4 on figure 7) to feed GNG-T. Since both these morphological operators are fast, this can be done without any expensive overhead. In usual cases and in an ordinary PC, this allows more than 5 frames per second. On figure 8, a supplementary smoothing stage has been added once the background extraction has been cleared. This stage uses a morphological closing, that adds a significant computational cost. The conclusion of these experiments are the following. The quality of the clustering obtained by the method is directly dependent on the quality of background suppression. Once this is done, GNG-T allows to track blobs, using its features for non-stationary distribution tracking. The conclusion of these experiments is that GNG-T is ready for being used in a plug and play manner, and any improvement of the object detection process will be rendered by GNG-T without any algorithmic modifications, since only the $T$ parameter may have to be changed.

6 Conclusion and perspectives

We have proposed in this paper an original extension of the GNG algorithm for addressing non-stationary problems. The algorithm deals with sudden changes, while being able to track smoothly changing elements with dedicated computational structures: the same neurons are associated with a smoothly moving object until they leave the image.

One advantage of GNG-T is that it is controlled by few parameters, that do not correspond to any heuristic. The semantics of the parameters is clear. This makes parameter tuning easy for a specific application. Moreover, the semantics of $T$ explains why sparse distributions of examples are represented by a few neurons, each having a high error variance. Such consideration explains that noise will be actually represented by some neurons in the network, but these can be removed according to variance (see figure 4 where noise is above line 15) or according to the length of edges. As far as we know, studies found in the literature do not stress the effect of noise in the probability density on the quantization.

Concerning the application to video surveillance, more efforts have to be made at the level of object detection, as well as further investigation in the way one
Fig. 7. This figure shows an experiment similar to the one in figure 6, except that $T = 100$ and that a cleaning process (line 3) and a contour extraction process (line 4) have been added.
Fig. 8. This figure shows an experiment similar to the one in figure 7, except that a morphological closure is applied just before computing the contour. The resulting contour is shown (line 2).

can use the resulting graph for understanding the semantics of the scene. Both these points are currently being worked on by our team.

References


