ÉCOLE
SUPÉRIEURE
D'ÉLECTRICITÉ

Support de cours

Quantification Vectorielle

Didacticiel sommaire

Hervé Frezza-Buet, Supélec

23 juin 2014

Herve.Frezza-Buet@supelec.fr

Table des matières

1 Préambule

Le lecteur est invité à consulter [Fritzke, 1997], qui est un document synthétique et très bien fait concernant la quantification vectorielle.

2 Généralités

2.1 Principe de la quantification vectorielle

La quantification vectorielle permet d'*analyser* une distribution, de la décrire, par une représentation simplifiée. On parle dans ce cas d'*apprentissage non supervisé*, puisque l'on ne dit pas au système ce qu'il doit apprendre, on lui demande simplement de décrire ce qu'il reçoit. L'analyse en composantes principales (ACP) est également une technique d'apprentissage non supervisé, puisqu'elle permet de représenter une distribution par des axes principaux.

Ce qui est propre à la quantification vectorielle (VQ), c'est que la représentation simplifiée de la distribution soumise à l'algorithme est un ensemble *fini* de vecteurs, d'où le nom *quantification*.

> *Représenter par une structure discrète une distribution continue peut se concevoir comme le fait d'ancrer une structure discrète dans une réalité continue. On peut y voir un mécanisme à la base de l'ancrage de représentations symboliques dans un monde analogique. Une prise de décision est quelque chose de discret (faire ou ne pas faire), et aborder cette question par quantification vectorielle est bien plus puissant que d'utiliser des seuils.*

2.2 Formalisation et notations

On se donne un espace X dit espace d'exemples, auquel appartiennent les échantillons que l'on reçoit d'un phénomène donné. Par exemple, si l'on s'intéresse à la distribution du poids et de la taille des gens, il y a une distribution (la nature et le mode d'alimentation) qui fait que, quand on choisit quelqu'un dans une population, on n'a pas n'importe quoi comme taille et comme poids. L'espace X, pour cet exemple, c'est $\mathbb{R}^2$ (taille et poids). Choisir une personne revient à choisir un échantillon $\xi \in X$.

On notera $p(\xi) \in \mathbb{R}^+$ la densité au point ξ de la distribution qui préside au choix des échantillons. Si on la normalise, ce qu'on ne fera pas, on obtient une densité de probabilité du tirage des échantillons. On se donne k vecteurs de X, que l'on notera $\omega_1, \cdots, \omega_k$, et que l'on appellera des *prototypes*. L'ensemble des prototypes $\omega_i \in X, 1 \leq i \leq k$ a vocation à représenter p.

Donnons nous également une mesure de similarité d de $X^2 \to \mathbb{R}^+$. Cette fonction est telle que $d(x, y)$ est faible si x « ressemble » à y, et forte si x « ne ressemble pas » à y. Usuellement, d est choisit comme le carré de la distance euclidienne sur X.

> *En pratique, d n'a pas besoin d'être une distance, on n'impose pas l'inégalité triangulaire par exemple. Le choix d'une fonction d adéquate est crucial, car c'est là que l'on injecte une connaissance du problème à résoudre. Voir figure 1.*

Notons $\omega(\xi)$ le prototype le plus proche de ξ, défini par l'équation 1.

$$\omega(\xi) = \underset{\omega \in \{\omega_1 \cdots \omega_k\}}{\operatorname{argmin}} \; d(\xi, \omega) \tag{1}$$

On définit [1] alors la distorsion $E \in \mathbb{R}^+$ qui est l'erreur que l'on commet à considérer que les prototypes représentent bien la distribution. Si la distribution p est normalisée, il s'agit de l'espérance de la distance entre un échantillon ξ et son prototype le plus proche, ce qu'exprime l'équation 2.

$$E = \int_X d(\xi, \omega(\xi)) p(\xi) \mathrm{d}\xi \tag{2}$$

> *Le but de la quantification vectorielle, c'est de placer les prototypes dans l'espace X de sorte à minimiser la distorsion E. Voir figure 4.*

2.3 Propriétés à l'optimum

Le propriétés de la répartition des prototypes au minimum de distorsion permettent de comprendre ce que calcule la quantification vectorielle. Pour exprimer ces propriétés, nous introduisons la notion de pavage de Voronoï.

On appellera cellule de Voronoï du prototype ω_i la région V_i de l'espace X qui contient les points qui sont plus proches de ω_i que des autres $\{\omega_j\}_{1 \leq j \neq i \leq k}$. Formellement, une cellule de Voronoï est définie par l'équation 3.

1. En toute rigueur, $\omega(\xi)$ n'est pas correctement défini si ξ est équidistant à plusieurs ω_i, ce qui n'arrive jamais en pratique. On peut dans ce cas compléter la définition en choisissant en cas d'ex-æquos d'attribuer à l'exemple ξ le prototype d'indice minimal parmis ces ex-æquos.

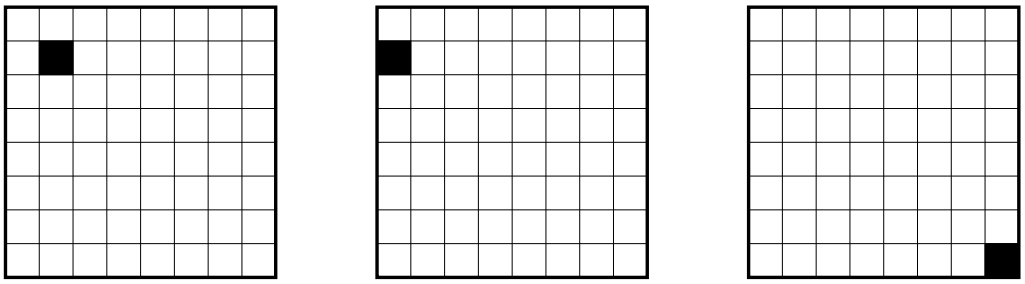

FIGURE 1 – Influence d'un choix correct de d. Si l'on considère ces images comme des vecteurs de 64 dimensions, et qu'un pixel noir (resp.blanc) vaut 1 (resp. 0), et si l'on utilise la distance euclidienne sur $\mathbb{R}^{64}$, alors ces images forment un triangle équilatéral, dont les côtés ont une longueur $\sqrt{2}$. Or, à l'oeil, on trouve l'image du milieu bien plus proche de l'image de gauche que de l'image de droite. C'est cette ressemblance que doit refléter un choix correct de d, qui incite ici à ne pas utiliser la distance euclidienne.

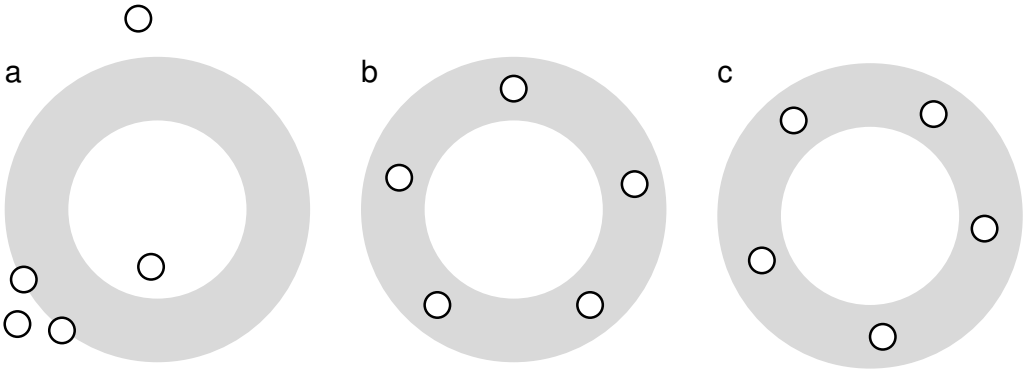

FIGURE 2 – Sur cet exemple, X est le plan de la feuille, et $p(\xi)$ a une valeur constante dans la zone grisée, et nulle en dehors. Les prototypes ω_i sont représentés par des cercles blancs. Sur la figure a, la distorsion occasionnée par la disposition des prototypes est forte, alors qu'elle est proche du minimum sur les figures b et c. Les figures b et c illustrent que la valeur minimal de distorsion peut être atteinte pour plusieurs configurations de prototypes.

$$V_i = \{\xi \in X \ : \ \omega(\xi) = \omega_i\} \tag{3}$$

Les cellules de Voronoï forment un pavage de X (voir figure 3(a)). On peut alors décomposer l'intégrale de l'équation 2 sur chacune des cellules de Voronoï, ce qui donne l'équation 4.

$$E = \int_X d(\xi, \omega(\xi))p(\xi)\mathrm{d}\xi = \sum_{i=1}^{k} \int_{V_i} d(\xi, \omega(\xi))p(\xi)\mathrm{d}\xi \tag{4}$$

En remarquant que, par définition des cellules de Voronoï, $\xi \in Vi \Leftrightarrow \omega(\xi) = \omega_i$, on peut réécrire l'équation 4 en 5.

$$E = \sum_{i=1}^{k} E_i \text{ en posant } E_i = \int_{V_i} d(\xi, \omega_i)p(\xi)\mathrm{d}\xi \tag{5}$$

> *Lorsque la configuration des prototypes conduit à une distorsion minimale, on a* $\forall 1 \leq i \leq k,\ E_i \approx T$. *La valeur* T *dépend de la distribution* p *et du nombre de prototypes* k. *Voir figure 3*

La conséquence de cette remarque est que la taille de la cellule de Voronoï est inversement proportionnelle à p. Intuitivement, on le comprend si l'on fait l'approximation que $p(\xi)$ est constant, valant p_i, sur la cellule V_i. On a alors $E_i = p_i \int_{V_i} d(\xi, \omega_i)\mathrm{d}\xi$. Comme les E_i sont tous égaux au minimum de distorsion, une cellules V_i où p_i est faible a une valeur $\int_{V_i} d(\xi, \omega_i)\mathrm{d}\xi$ plus forte qu'une cellule V_j où p_i est fort. La cellule V_i est donc plus étendue que V_j. En fait, l'étendue dont ont parle n'est pas celle de la cellule de Voronoï, mais la région de cette cellule où p_i est significativement non nul (cf. figure 3(c)). La conséquence est qu'à l'équilibre, les prototypes sont « plus serrés » là ou p est plus dense (cf. figure 3(b)).

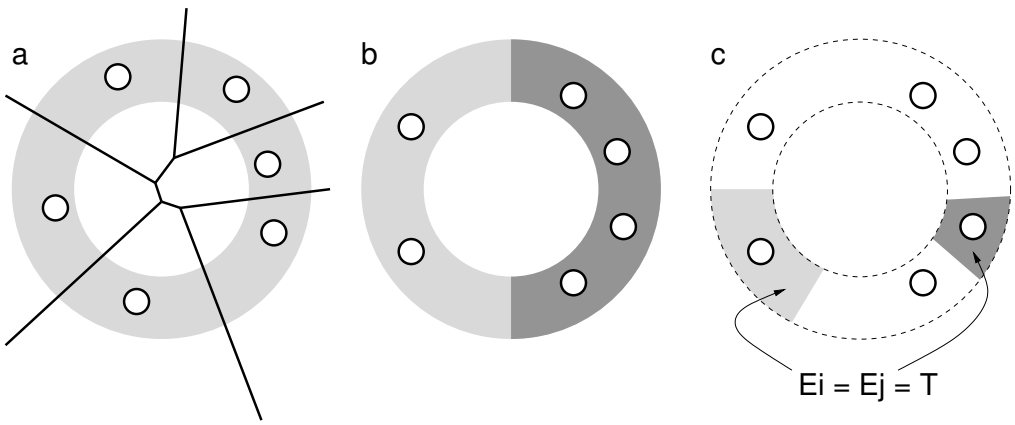

FIGURE 3 – En a, pavage de Voronoï induit par la répartition des prototypes. En b, la zone sombre est plus dense ($p(\xi)$ supérieur dans cette région) et les prototypes sont placés de sorte à minimiser la distorsion. En c, les régions de densité non nulle d'un V_i et d'un V_j sont isolées. Ce sont elles qui contribuent respectivement aux intégrales E_i et E_j de l'équation 5, on a $E_i = E_j = T$.

Attention, la répartition des prototypes est sensible à la valeur de p : les prototypes s'agglutinent autour des régions les plus denses, au détriment des autres. Voir figure 4.

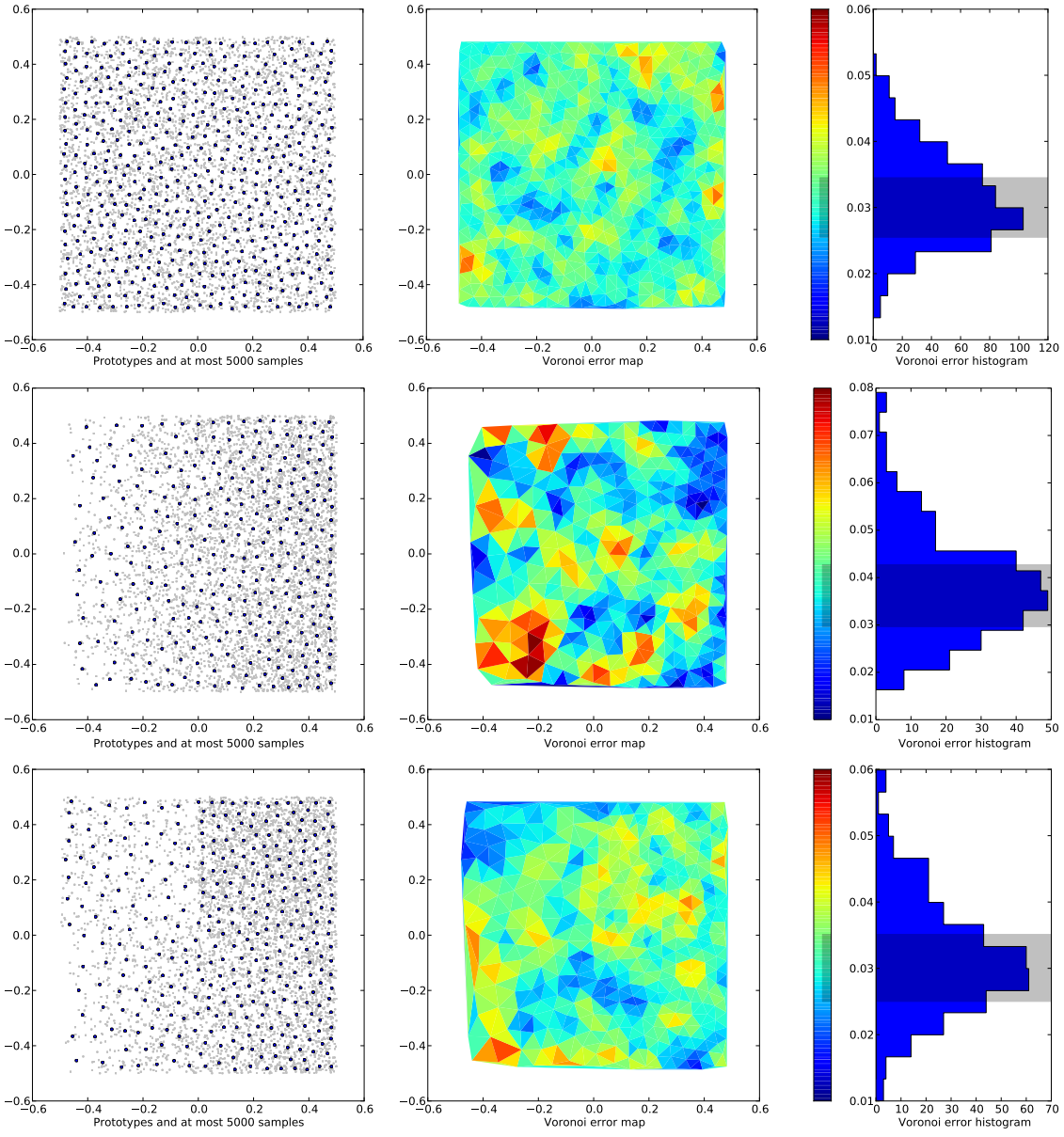

FIGURE 4 – La répartition des prototypes est sensible aux différences de concentration des exemples.

2.4 Extraction de structure

On s'intéresse ici à la structure de la distribution p : Est-elle filiforme, surfacique, volumique, contient-elle plusieurs composantes connexes, des trous ? Il est possible assez facilement d'extraire une bonne approximation de cette structure. Pour ce faire, définissons au préalable la *triangulation de Delaunay* de nos prototypes.

La triangulation de Delaunay est un graphe où les sommets sont les prototypes. On crée un arête entre ω_i et ω_j si et seulement si V_i et V_j ont un côté commun. Le graphe obtenu a des propriétés géométriques non décrites ici. Ce qui nous intéresse, c'est d'élaguer ce graphe en lui supprimant les arêtes qui ne recouvrent pas suffisamment des régions où p est fort. Le graphe élagué donne alors une bonne mesure de la topologie de la variété p, comme l'illustre la figure 5

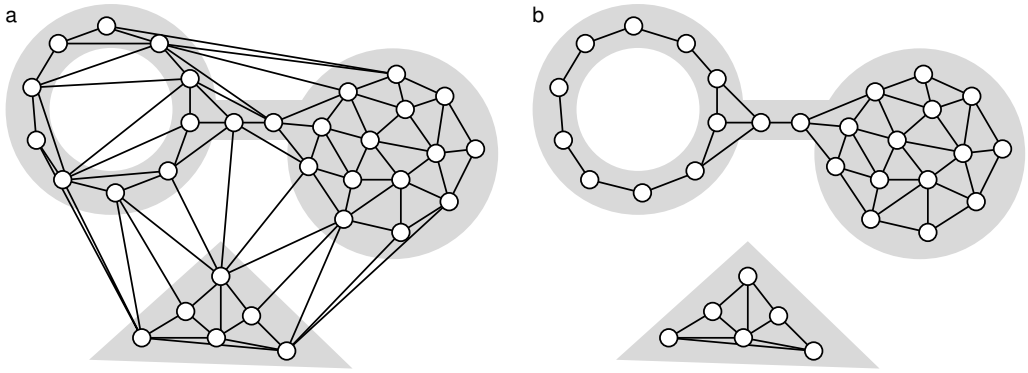

FIGURE 5 – Si l'on minimise la distorsion avec 32 prototypes, on obtient la répartition illustrée sur les deux figures, pour une distribution p représentée en gris. À gauche, on représente la triangulation de Delaunay de ces prototypes. À droite, on enlève de cette triangulation les arête qui ne recouvrent pas assez les zones grisées. Le résultat est un maillage 1D des prototypes là où p est filiforme, et une triangulation 2D des régions où p recouvre des surfaces. Le graphe obtenu a bien deux composantes connexes.

Une bonne approximation du graphe de la figure 5(b) peut s'obtenir par l'algorithme 1 tout simple [Martinez and Schulten, 1994].

Algorithm 1 Competitive Hebbian Learning

1: **while true do**
2: Tirer ξ suivant p.
3: Déterminer ω^1 et ω^2 les deux prototypes les plus proches de ξ.
4: Créer l'arête $\omega^1 \leftrightarrow \omega^2$ si elle n'existe pas.
5: **end while**

2.5 Bilan

Les méthodes de quantifications vectorielles sont des techniques de réduction de la distorsion. Elles permettent de plaquer une structure discrète, qui est un ensemble de points voire un graphe, sur un phénomène continu p qui fournit des échantillons.

Pour les algorithmes que nous abordons dans la suite, nous chercherons à voir si :

- ils sont *on-line* : On les nourrit successivement d'échantillons ξ, et ils travaillent à partir de ce flux d'échantillons sans avoir besoin d'en stocker une quantité finie dans une base.
- ils impliquent un graphe : bien qu'on puisse toujours construire un graphe a posteriori sur un ensemble de prototypes, certains algorithmes utilisent explicitement ce graphe.
- ils sont stationnaires : il n'est pas nécessaire de faire varier les paramètres de l'algorithme en cours d'utilisation.
- ils font du *tracking* de distribution : ils sont robustes aux changements de p au cours du temps.

– si le nombre de prototype s'ajuste : pour certains algorithmes, il faut définir le nombre de prototypes à l'avance.

On répondra à ces questions pour chacun des algorithmes abordés dans ce qui suit.

3 Algorithmes

À l'exception de la section 3.1, on ne présente ici que des algorithmes utilisables en ligne. Pour ceux-ci, on présente successivement des exemples tirés de la distribution que l'on souhaite quantifier. Il peut alors se présenter un piège :

> *Lorsque l'on tire des exemples d'une distribution en ligne, il faut veiller à ce que ces exemples ne soient pas corrélés spatialement.*

Autrement dit, on ne présente pas les exemples suivant leur composante x, puis y, etc... mais bien dans un ordre qui ne dépend que de la probabilité définie par la distribution. Si on ne prend pas garde à ce point, les algorithmes en ligne interprètent le problème comme une distribution non-stationnaire, qui dérive dans l'espace.

3.1 k-means

Cet algorithme [Lloyd, 1982; Linde *et al.*, 1980; Kanungo *et al.*, 2002] est le plus connu des algorithmes de quantification vectorielle.

Algorithm 2 k-means

1: Constituer un ensemble $S = \{\xi_i\}_{1 \leq i \leq N}$ de N échantillons tirés suivant p.
2: Calculer ω_1, barycentre de S.
3: $\Omega \leftarrow \{\omega_1\}$ // On commence avec un seul prototype
4: **while** $|\Omega| <$ k **do**
5: Choisir aléatoirement $n = \min(k - |\Omega|, |\Omega|)$ prototypes dans Ω.
6: Cloner ces n prototypes, et ajouter les n clones dans Ω.
7: **repeat**
8: $\forall\, 1 \leq i \leq N,\ \text{label}(i) \leftarrow k\ :\ \omega(\xi_i) = \omega_k$
9: $\forall\, 1 \leq k \leq |\Omega|,\ \omega_k \leftarrow \text{barycentre}_{i\,:\,\text{label}(i)=k}\ \xi_i$
10: **until** La procédure n'a pas conduit à un changement de label.
11: **end while**

À la ligne 6 de l'algorithme 2, cloner un prototype, c'est créer un nouveau prototype dont la valeur est tirée aléatoirement dans une région très proche du prototype initial. La terminaison de l'algorithme 2 est montrée, mais il peut rester capturé dans un minimum local de distorsion. L'initialisation requiert que les prototypes soient tirés suivant p.

k-means	
on-line	non
graphe	non
algorithme stationnaire	oui car off-line
distribution non stationnaire	non car off-line
ajustage du nombre de prototypes	non

3.2 k-means on-line

Cet algorithme [MacQueen, 1967] est une version on-line du précédent, encore plus facile à programmer. Sa structure se retrouvera dans les autres algorithmes que nous étudierons.

Algorithm 3 k-means online

1: Tirer $\omega_1 \cdots \omega_K$ suivant p.
2: **while true do**
3: Tirer ξ suivant p.
4: Déterminer $k^\star$: $\omega_{k^\star} = \omega(\xi)$. // Compétition.
5: $\omega_{k^\star} \leftarrow \omega_{k^\star} + \alpha(\xi - \omega_{k^\star})$ // Apprentissage, $\alpha \in]0, 1]$.
6: **end while**

La valeur α est une constante positive. La ligne 5 réalise ce qu'on appelle une *hard competition*, ou *winner-take-all*. En effet, une fois que $k^\star$ est déterminé, seul le prototype $\omega_{k^\star}$ apprend.

k-means online	
on-line	oui
graphe	non
algorithme stationnaire	si α = cste
distribution non stationnaire	non
ajustage du nombre de prototypes	non

3.3 Growing Neural Gas (GNG)

3.3.1 L'algorithme initial de Fritzke

Cet algorithme [Fritzke, 1995] consiste à gérer des prototypes qui sont d'emblée structurés comme un graphe. On en trouvera une description complète dans [Fritzke, 1997]. L'algorithme commence avec deux prototypes, et fait croître régulièrement le nombre de prototypes.

Les arêtes du graphe sont mises à jour à chaque tirage d'un exemple, selon le principe de l'algorithme de restitution de structure page **??**. Lorsqu'un prototype est le plus proche de ξ, ses arêtes vieillissent, sauf celle qui le connecte avec le deuxième plus proche, qui est soit créée, soit rajeunie à 0 si elle existait déjà. Les arêtes trop vieilles sont supprimées.

Le prototype $\omega_{k^\star} = \omega(\xi)$, à chaque tirage de ξ, met à jour un accumulateur $e_{k^\star} \leftarrow e_{k^\star} + d(\xi, \omega_{k^\star})$. Régulièrement, on décide d'ajouter un prototype près des noeuds du graphe qui ont l'accumulateur le plus fort. C'est ainsi que le graphe grandit et se déploie sur la distribution p.

GNG	
on-line	oui
graphe	oui
algorithme stationnaire	oui
distribution non stationnaire	non
ajustage du nombre de prototypes	non

3.3.2 GNG-T

Il s'agit d'une extension que nous avons proposée de l'algorithme précédent [Frezza-Buet, 2008]. L'idée est de contrôler le nombre de neurones par une estimation de la valeur T définie dans l'encart page 5. Une implémentation `C++` de cet algorithme est disponible sur le site de l'équipe IMS de Supélec [Frezza-Buet, 2010].

Dans cet algorithme, il s'agit de présenter les échantillons sous forme d'*epoch*, c'est-à-dire par paquets. Un paquet sert à estimer p pour un instant t donné.

Algorithm 4 `gngt_epoch`(G, S, α)

1: // $\alpha \in [0, 1]$ is the learning rate.
2: $S' \leftarrow S$
3: `gngt_open_epoch`(G)
4: **repeat**
5: $\quad \xi \sim \mathcal{U}_{S'},\ S' \leftarrow S' \setminus \{\xi\}$
6: $\quad$ `gngt_submit`(G, ξ, α)
7: **until** $S' = \emptyset$
8: `gngt_close_epoch`(G)

Algorithm 5 `gngt_open_epoch`(G)

1: **for all** $v \in \mathrm{V}(G)$ **do**
2: $\quad v_\mathrm{e} \leftarrow 0$ // v_e is the Voronoï error accumulated by v.
3: $\quad v_\mathrm{n} \leftarrow 0$ // v_n is the number of times v_ω has been the closest to an input sample.
4: **end for**

L'algorithme permet, selon la valeur de T, d'ajuster le nombre de prototypes pour avoir la précision souhaitée dans l'approximation. Il remet à jour automatiquement le nombre de prototypes si p change, en essayant au maximum d'adapter les prototypes existants (effet *tracking*). La figure 6 montre le comportement de cet algorithme suivant les variations de p et la figure 7 illustre son application à l'analyse de scènes vidéo.

GNG-T	
on-line	oui
graphe	oui
algorithme stationnaire	oui
distribution non stationnaire	oui
ajustage du nombre de prototypes	oui

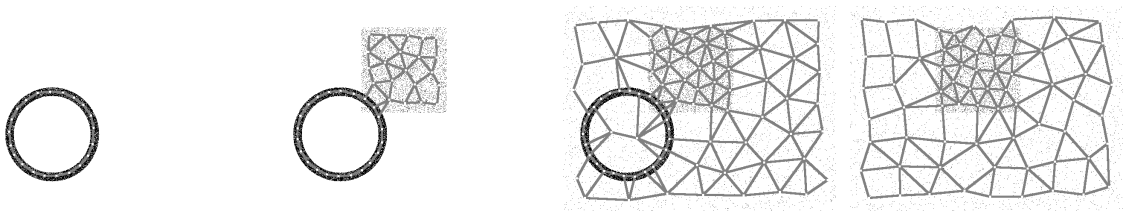

FIGURE 6 – Comportement de GNG-T sur une distribution artificielle, d'après [Frezza-Buet, 2008].

Algorithm 6 `gngt_submit`(G, ξ, α)

if $|\mathrm{V}(G)| < 2$ **then**
  Add a new vertex v to G.
  $v_\omega \leftarrow \xi$ // The prototype takes the value of the sample.
  $v_\mathrm{e} \leftarrow 0$
  $v_\mathrm{n} \leftarrow 1$
  return // Exit the algorithm here.
end if
$v^\star \leftarrow \mathrm{argmin}_{v \in \mathrm{V}(G)} d(v_\omega, \xi)$ // $v^\star$ is the closest vertex.
$v^{\star\star} \leftarrow \mathrm{argmin}_{v \in \mathrm{V}(G) \setminus \{v^\star\}} d(V_\omega, \xi)$ // $v^{\star\star}$ is the second closest vertex.
create the edge $\mathrm{e}^{v^\star \leftrightarrow v^{\star\star}}$ if it does not exist yet.
$\mathrm{e}_\mathrm{a}^{v^\star \leftrightarrow v^{\star\star}} \leftarrow 0$ // Set/reset the age of the edge at a 'newborn' value.
for all $\mathrm{e} \in \mathrm{E}(v^\star) \setminus \left\{\mathrm{e}^{v^\star \leftrightarrow v^{\star\star}}\right\}$ **do**
  $e_\mathrm{a} \leftarrow e_\mathrm{a} + 1$
  if $e_\mathrm{a} > \varphi$ **then**
    remove e // Too old edges are removed.
  end if
end for
$v_\mathrm{e}^\star \leftarrow v_\mathrm{e}^\star + d(v_\omega^\star, \xi)$ // Update winner statistics.
$v_\mathrm{n}^\star \leftarrow v_\mathrm{n}^\star + 1$
`vq_learn`$(\alpha, v_\omega^\star, \xi)$
for all $v \in \mathrm{V}(v^\star)$ **do**
  `vq_learn`$(\zeta\alpha, v_\omega, \xi)$ // $\zeta \in]0, 1]$ so that the learning of neighbors is weaker.
end for

Algorithm 7 `vq_learn`(α, ω, ξ)

Require: $\alpha \in [0, 1]$
$\omega \leftarrow \omega + \alpha.(\xi - \omega)$

FIGURE 7 – Application à l'analyse de scenes vidéo. L'algorithme est alors nourri par les coordonnées des pixels du contour des zones qui ne sont pas du fond, d'après [Frezza-Buet, 2008].

Algorithm 8 `gngt_close_epoch`(G)

for all $v \in \mathrm{V}(G)$ **do**
  if $v_\mathrm{n} = 0$ **then**
    remove v from G. // v has never won any competition.
  end if
end for
if $|\mathrm{V}(G)| = 0$ **then**
  return // Exit the algorithm here.
end if
evol $\leftarrow$ `compute_evolution`(G)
if evol $=$ `None` **then**
  return // Exit the algorithm here.
else if evol $=$ `Remove` **then**
  remove $\mathrm{argmin}_{v \in \mathrm{V}(G)} v_\mathrm{e}$
else if evol $=$ `Add` **then**
  find $v^\star = \mathrm{argmax}_{v \in \mathrm{V}(G)} v_\mathrm{e}$
  create v and add it in the graph.
  $v_\omega \leftarrow v^\star_\omega$ // The winning node is cloned
  if $\mathrm{E}(v^\star) = \emptyset$ **then**
    add $\mathrm{e}^{v \leftrightarrow v^\star}$ // Connect the two clones...
    $\mathrm{e}_\mathrm{a}^{v \leftrightarrow v^\star} \leftarrow 0$ // ... with a 'newborn' edge.
  else
    find $v^{\star\star} = \mathrm{argmax}_{v \in \mathrm{V}(v^\star)} v_\mathrm{e}$ // $v^{\star\star}$ is the neighbor of $v^\star$ with the highest accumulated error.
    `vq_learn`$(\lambda, v_\omega, v^{\star\star}_\omega)$ // The clone is moved slightly toward $v^{\star\star}$.
    add $\mathrm{e}^{v \leftrightarrow v^\star}$ and $\mathrm{e}^{v \leftrightarrow v^{\star\star}}$ to the graph.
    $\mathrm{e}_\mathrm{a}^{v \leftrightarrow v^\star} \leftarrow 0$, $\mathrm{e}_\mathrm{a}^{v \leftrightarrow v^{\star\star}} \leftarrow 0$ // Make the new edges 'newborn'.
  end if
end if

Algorithm 9 `compute_evolution`(G)

$\mu = \frac{1}{|\mathrm{V}(G)|} \sum_{v \in G} v_\mathrm{e}$ // μ is the average of accumulated errors.
if $T < \mu$ **then**
  return `Add`
else
  return `Remove`
end if

3.4 Cartes auto-organisatrices

Les cartes auto-organisatrices, *Self-Organizing Maps (SOM)* [Kohonen, 1989], sont des méthodes pour lesquelles le graphe qui relie les prototypes est fixé au départ, a priori, et reste constant tout au cours de la quantification vectorielle. Cela permet de mettre en correspondance le graphe avec la topologie de la distribution p.

Soit G un graphe dont les sommets sont des prototypes, et dont les arêtes définissent une topologie. On utilise classiquement une grille comme topologie, mais n'importe quel graphe peut convenir. Soit $\nu(\omega_i, \omega_j)$ la distance entre deux sommets ω_i et ω_j du graphe. Ne confondez pas $\nu(\omega_i, \omega_j)$ avec $d(\omega_i, \omega_j)$. En effet, ν ne tient compte que de la position des prototypes *dans le graphe*, c'est par exemple lié au nombre d'arêtes du chemin le plus court qui relie les deux sommets. La fonction d en revanche réside, elle, dans l'espace X d'où sont tirés les exemples, et les prototypes.

Les SOM font en sorte que, à l'équilibre, les prototypes minimisent la distorsion, mais en assurant également que deux prototypes proches suivant ν le soient aussi suivant d. La réciproque, elle, n'est pas assurée. Soit h une fonction décroissante de $\mathbb{R}^+$ dans $\mathbb{R}^+$, telle que $h(0) = 1$ et $h(\infty) = 0$. On choisit en général une gaussienne.

Algorithm 10 Cartes auto-organisatrices

1: On se donne a priori un graphe G de prototypes et une distance ν sur ce graphe.
2: On initialise les ω_i, sommets du graphe, avec des valeurs aléatoires.
3: **while true do**
4: Tirer ξ suivant p.
5: Déterminer $k^\star$: $\omega_{k^\star} = \omega(\xi)$. // Compétition.
6: $\forall \omega \in G,\ \omega \leftarrow \omega + \alpha \times h(\nu(\omega_{k^\star}, \omega))(\xi - \omega)$ // Apprentissage, $\alpha \in]0, 1]$.
7: **end while**

L'algo est identique à celui des k-means on-line page 9, mais tous les prototypes sont mis à jour, selon leur proximité dans le graphe avec $\omega(\xi)$ (voir ligne 6). On parle alors de *soft competition* ou *winner-take-most*. Les figures 8 et 9 montrent le comportement de l'algorithme. En général, on fait varier h au cours du temps, ce qui aide la convergence (mais n'est pas nécessaire). Cette variation consiste à accélérer la décroissance de h. Par exemple, si h est une gaussienne, on commencera par une variance forte, que l'on réduira au cours du temps.

Cartes auto-organisatrices	
on-line	oui
graphe	oui
algorithme stationnaire	oui si h fixe
distribution non stationnaire	oui
ajustage du nombre de prototypes	non

Références

[Frezza-Buet, 2008] Hervé Frezza-Buet. Following non-stationary distributions by controlling the vector quantization accuracy of a growing neural gas network. *Neurocomputing*, 71(7-9) :1191–1202, 2008.

[Frezza-Buet, 2010] Hervé Frezza-Buet, 2010. http://ims.metz.supelec.fr/spip.php?rubrique22.

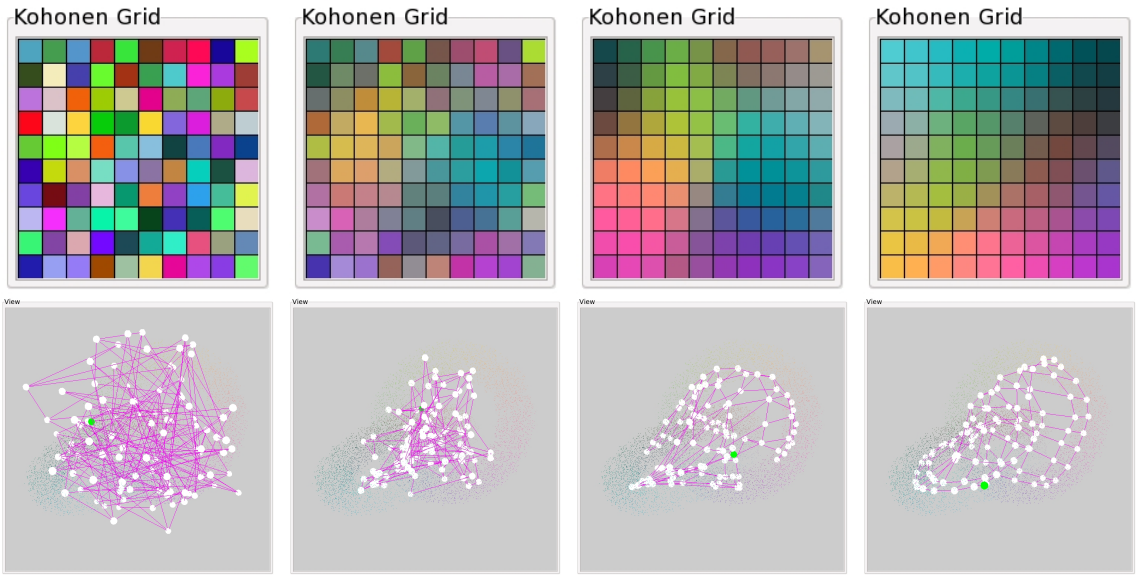

FIGURE 8 – Cartes auto-organisatrices de Kohonen. X est un espace 3D de couleurs (r,g,b). On peut représenter les prototypes au sein de la grille (en haut) en affichant leur couleur, ou dans l'espace X selon leur valeur (en bas). De gauche à droite, dépliement de la carte.

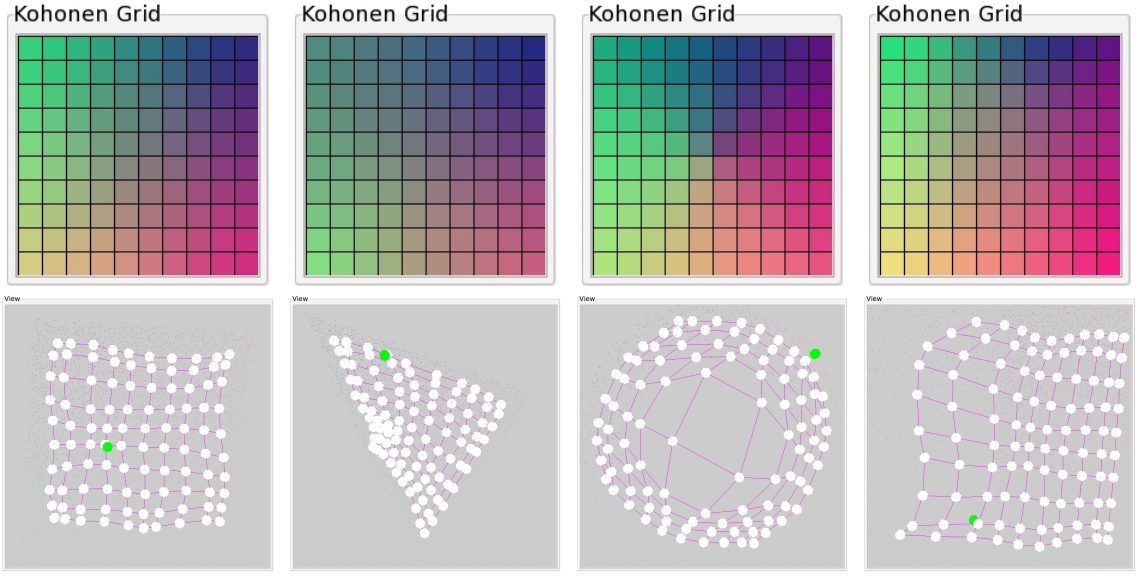

FIGURE 9 – Même algorithme que sur la figure 8, pour différentes distributions.

[Fritzke, 1995] Bernd Fritzke. A growing neural gas network learns topologies. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, *Advances in Neural Information Processing Systems 7*, pages 625–632. MIT Press, Cambridge MA, 1995.

[Fritzke, 1997] Bernd Fritzke. *Some Competitive Learning Methods*, 1997. http://www.ki.inf.tu-dresden.de/~fritzke/JavaPaper/.

[Kanungo *et al.*, 2002] T. Kanungo, D. M. Mount, N. Netanyahu, C. Piatko, R. Silverman, and A. Y. Wu. An efficient k-means clustering algorithm : Analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24 :881–892, 2002.

[Kohonen, 1989] Tuevo Kohonen. *Self-Organization and Associative Memory*, volume 8 of *Springer Series in Information Sciences*. Springer-Verlag, 1989.

[Linde *et al.*, 1980] Yoseph Linde, Andres Buzo, and Robert M. Gray. Algorithm for vector quantization design. *IEEE transactions on communications systems*, 28(1) :84–95, 1980.

[Lloyd, 1982] S. P. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2) :129–137, 1982.

[MacQueen, 1967] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In L. M. Le Cam and J. Neyman, editors, *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967.

[Martinez and Schulten, 1994] T. M. Martinez and K. J. Schulten. Topology representing networks. *Neural Networks*, 7(3) :507–522, 1994.