Un algorithme efficace pour les cartes auto-organisatrices de Kohonen appliquées aux tableaux de dissimilarités

Brieuc Conan-Guez†, Fabrice Rossi‡, Aïcha El Golli‡

†*LITA EA3097, Université Paul Verlaine -Metz, Île du Saulcy, 57045 METZ CEDEX 1*
brieuc.conan-guez@univ-metz.fr
‡*Projet AxIS, INRIA, Domaine de Voluceau, Rocquencourt, B.P. 105, 78153, Le Chesnay Cedex, France*
{fabrice.rossi,aicha.elgolli}@inria.fr

RÉSUMÉ. *Dans cet article, nous proposons une nouvelle implémentation d'une adaptation des cartes auto-organisatrices de Kohonen (SOM) aux tableaux de dissimilarités : nous proposons tout d'abord une modification de l'algorithme afin d'obtenir une réduction importante de son coût théorique, puis nous introduisons certaines techniques d'implémentation qui permettent de réduire de manière importante son temps de calcul effectif. Une propriété importante de ces diverses modifications est que les résultats obtenus sont strictement identiques à ceux de l'algorithme initial.*

MOTS-CLÉS : *Tableau de dissimilarités, Carte auto-organisatrice de Kohonen, Implémentation efficace.*

1. Introduction

Dans beaucoup d'applications réelles, les individus étudiés ne peuvent pas être décrits efficacement par des vecteurs numériques : on pense par exemple à des données de tailles variables (séquences de protéines), ou bien à des données (semi-)structurées (graphes, arbres, documents XML). Une solution possible pour traiter de telles données est de s'appuyer sur une mesure de dissimilarité permettant de comparer les individus deux à deux.

Une variante des cartes auto-organisatrices de Kohonen (SOM, [1]) adaptée aux tableaux de dissimilarités a été proposée dans [2, 3] (voir section 2). Cet algorithme souffre malheureusement d'un temps de calcul beaucoup plus élevé que celui du SOM classique. Dans ce travail, nous proposons tout d'abord une modification de cet algorithme afin d'obtenir une réduction importante de son coût théorique (section 3), puis nous introduisons certaines techniques d'implémentation qui permettent de réduire de manière importante son temps de calcul. Les résultats produits par ce nouvel algorithme sont strictement identiques à ceux de l'algorithme initial. Enfin, afin de quantifier l'amélioration des temps de calcul, la section 4 est consacrée à des expériences menées sur des données simulées.

2. Cartes auto-organisatrices de Kohonen adaptées aux tableaux de dissimilarités

Nous rappelons dans cette section l'adaptation du SOM aux tableaux de dissimilarités [2, 3, 4]. On considère N individus $\mathcal{D} = (\mathbf{x}_i)_{1 \leq i \leq N}$ appartenant à un espace arbitraire $\mathcal{X}$, muni d'une mesure de dissimilarité d. On considère un SOM à M modèles (ou neurones), qui sont numérotés de 1 à M. Le modèle j est associé à un élément de $\mathcal{D}$, noté $\mathbf{m}_j$ (pour chaque modèle j, il existe i tel que $\mathbf{m}_j = \mathbf{x}_i$). $\mathbf{m}_j$ est appelé le prototype du modèle j, et on note $\mathcal{M} = (\mathbf{m}_1, \ldots, \mathbf{m}_M)$. On munit l'ensemble des modèles d'une métrique δ qui correspond à la structure *a priori* imposée par le SOM. Le but du SOM est alors double : quantifier correctement l'ensemble $\mathcal{D}$ en trouvant de bons prototypes, organiser ces prototypes dans $\mathcal{D}$ afin qu'ils respectent la topologie induite par δ sur les modèles. On note K une fonction noyau décroissante (par exemple $K(x) = \exp(-x^2)$), et $h(j,k) = K(\delta(j,k))$ la fonction de voisinage.

Le SOM adapté aux tableaux de dissimilarités (DSOM) est un algorithme itératif où les prototypes et la fonction de voisinage évoluent à chaque itération l : on note donc $\mathbf{m}_j^l$ et $h^l(.,.)$ ces quantités. La décroissance de h^l à chaque itération assure la convergence de l'algorithme. L'algorithme du DSOM, qui est basée sur la version non stochastique du SOM (version *batch*) commence par une *phase d'initialisation* : on choisit par exemple $\mathcal{M}^0 = (\mathbf{m}_1^0, \ldots, \mathbf{m}_M^0)$ de manière aléatoire. Puis l'algorithme alterne les *phases d'affectation* et les *phases de représentation* jusqu'à la convergence. Pour l'itération l, on a :

1. *phase d'affectation* : chaque individu $\mathbf{x}_i$ est affecté à son modèle gagnant $c^l(i)$ en appliquant la règle d'affectation usuelle $c^l(i) = \arg\min_{j\in\{1,\ldots,M\}} d(\mathbf{x}_i, \mathbf{m}_j^{l-1})$. L'inconvénient de ce type d'affectation est qu'il ne permet pas de lever les ambiguités relatives aux prototypes (cas relativement fréquent où deux modèles distincts sont associés au même prototype). Dans notre implémentation, nous utilisons donc la modification proposée dans [2]. Finalement, on note $\mathcal{C}_j^l$ la classe associée au modèle j à l'itération l. Les $\mathcal{C}_j^l$ forment une partition de $\mathcal{D}$.

2. *phase de repésentation* : l'algorithme calcule de nouvelles valeurs pour les prototypes (i.e. $\mathcal{M}^l$). Chaque prototype $\mathbf{m}_j^l$ est solution du problème :

$$\mathbf{m}_j^l = \arg\min_{\mathbf{m}\in\mathcal{D}} \sum_{i=1}^{N} h^l(c^l(i), j) d(\mathbf{x}_i, \mathbf{m}). \quad [1]$$

3. Une implémentation efficace du DSOM

Si l'on examine le coût algorithmique du DSOM, on constate que pour une itération de l'algorithme, le coût de la phase d'affectation est en $O(NM^2)$ (voir la règle d'affectation modifiée dans [2]) et que la phase de représentation (voir équation 1) est en $O(N^2M)$[1]. Comme $N > M$ dans tous les cas, la phase de représentation domine nettement le calcul. Les modifications apportées au DSOM dans cette section vont permettre dans un premier temps de réduire la complexité théorique de la phase de représentation, puis, grâce à une implémentation efficace de l'algorithme, de réduire son temps d'exécution.

3.1. *Les sommes partielles*

En analysant le problème de minimisation proposé dans l'équation 1, on constate que sa structure peut être grandement simplifiée. A l'itération l et pour chaque modèle j, on cherche à trouver pour quel k, $S^l(j,k)$ est minimum, où $S^l(j,k) = \sum_{i=1}^{N} h^l(c^l(i), j) d(\mathbf{x}_i, \mathbf{x}_k)$. Si l'on note $D^l(u,k) = \sum_{i\in\mathcal{C}_u^l} d(\mathbf{x}_i, \mathbf{x}_k)$, on peut exprimer $S^l(j,k)$ d'une manière plus simple :

$$S^l(j,k) = \sum_{u=1}^{M} h^l(u,j) \sum_{i\in\mathcal{C}_u^l} d(\mathbf{x}_i, \mathbf{x}_k) = \sum_{u=1}^{M} h^l(u,j) D^l(u,k). \quad [2]$$

Il y a MN différentes valeurs $D^l(u,k)$, qui peuvent être pré-calculées une fois pour toute avant la phase de représentation. Le coût de cette phase de pré-calcul est en $O(N^2)$. En effet, calculer la somme partielle $D^l(u,k)$ coûte $O(|\mathcal{C}_u^l|)$ (où $|\mathcal{C}_u^l|$ est l'effectif de $\mathcal{C}_u^l$). Puis calculer tous les $D^l(u,k)$ pour un u fixé coûte $O(N|\mathcal{C}_u^l|)$. Comme $\sum_{u=1}^{M} |\mathcal{C}_u^l| = N$, le coût total est en $O(N^2)$. Selon l'équation 2, le calcul de $S^l(j,k)$ peut être effectué en $O(M)$ opérations. Comme cela doit être fait pour tous les k et pour tous les j, le coût est en $O(NM^2)$. On a donc un coût total pour la phase de représentation de $O(N^2 + NM^2)$, à comparer avec $O(N^2M)$ pour l'algorithme initial. Comme on a $N > M$ dans toutes les situations, cette approche réduit le coût du DSOM. De plus, les résultats obtenus sont strictement identiques à ceux de l'algorithme initial.

---

1. à comparer avec $O(nNM)$ dans le cas du SOM *batch* classique sur des vecteurs de dimension n

3.2. *L'arrêt prématuré*

Dans la section précédente, les modifications apportées à l'algorithme du DSOM ont permis de réduire sa complexité algorithmique, dans les sections suivantes, nous allons montrer comment grâce à quelques techniques d'implémentation, il est possible d'accélérer son temps d'exécution de manière importante.

Lors de la phase de représentation, à l'itération l et pour chaque modèle j, on cherche à calculer le minimum des $S^l(j,k)$. Ce calcul est réalisé informatiquement au sein d'une boucle en k, que nous appelerons la boucle externe. Cette boucle externe effectue deux opérations à chacune de ses itérations : premièrement elle calcule grâce à une seconde boucle (boucle interne) la valeur de $S^l(j,k)$ pour un k donné (voir la somme en u apparaissant à la droite de l'équation 2), puis elle compare le résultat trouvé avec le meilleur résultat calculé lors des itérations prédédentes. Afin d'améliorer l'efficacité de ce calcul, une idée simple consiste à déplacer cette étape de comparaison dans la boucle interne : le calcul de la boucle interne est arrêté prématurément dès lors que la sous-somme $\Sigma_k = \sum_{u=1}^{m'} h^l(u,j)D^l(u,k)$ calculée à l'itération m' (avec $m' < M$) est supérieure au minimum.

Afin de favoriser la stratégie d'arrêt prématuré, la boucle interne ainsi que la boucle externe doivent être ordonnées. Pour la boucle interne, l'ordre optimal serait d'accroître Σ_k le plus rapidement possible en sommant d'abord les grandes valeurs de $h^l(u,j)D^l(u,k)$. Pour la boucle externe, l'ordre optimal serait de commencer avec les valeurs faibles de $S^l(j,k)$ (i.e. avec un bon candidat pour le prototype du modèle j) : une faible valeur de $S^l(j,k)$ arrêtera la boucle interne plus tôt qu'une grande valeur. En pratique cependant, calculer ces ordres optimaux est extrêmement coûteux. Nous utiliserons donc des ordres efficaces mais non optimaux qui sont induits par la topologie du DSOM. La définition de h^l implique que $h^l(u,j)$ est petit quand $\delta(u,j)$ est grand. Il est donc raisonnable d'ordonner la boucle interne en u dans l'ordre décroissant des $h^l(u,j)$, i.e. dans l'ordre croissant des $\delta(u,j)$. Pour la boucle externe, nous utilisons les propriétés d'organisation du DSOM : les individus sont affectés à la classe du prototype le plus proche. Donc, la qualité *a priori* d'un individu $\mathbf{x}_k$ comme prototype pour le modèle j est plus ou moins l'inverse de la distance δ entre le modèle j et le modèle représentant la classe de $\mathbf{x}_k$. On ordonne donc la boucle externe dans l'ordre croissant de δ. Ces divers optimisations ne modifie pas les résultats produits.

3.3. *La mémorisation*

Une autre source d'optimisation vient du fait que les classes $\mathcal{C}_u^l$ produites par le DSOM ont tendance à se stabiliser lors des dernières itérations de l'algorithme. Il n'est pas rare que d'une itération à l'autre, le contenu d'une (ou plusieurs) classe reste strictement identique. Dans de tels cas, les N valeurs $D^l(u,k)$ correspondantes restent inchangées, et il est donc inutile de les recalculer. La mise en œuvre informatique de cette optimisation peut facilement être réalisée en associant à chaque classe $\mathcal{C}_u$ une variable booléenne. Lors de la phase d'affectation, cette variable est positionnée si on constate un changement de classe. Le $D^l(u,k)$ correspondant sera alors recalculé lors de la phase de représentation.

Bien que cette stratégie de mémorisation se révèle être très efficace, son mode d'action est un peu grossier. En effet, le calcul complet des $D^l(u,k)$ pour deux valeurs de u (i.e., pour deux classes) peut être déclenché par le changement de classes d'un unique individu. Il semble donc intéressant de chercher une solution plus fine. Considérons en effet le cas où la classe ne subit qu'une seule modification : l'individu $\mathbf{x}_i$. On a alors :

$$D^l(c^{l-1}(i),k) = D^{l-1}(c^{l-1}(i),k) - d(\mathbf{x}_i,\mathbf{x}_k) \quad [3]$$

$$D^l(c^l(i),k) = D^{l-1}(c^l(i),k) + d(\mathbf{x}_i,\mathbf{x}_k) \quad [4]$$

Appliquer ces formules de mise-à-jour induit $2N$ additions et N affectations. Si plusieurs individus se déplacent de leur ancienne classe à leur nouvelle classe, les opérations de mise-à-jour doivent être effectuées pour chaque déplacement. Dans le cas extrême où tous les individus changent de classes, le nombre total d'additions serait de $2N^2$ (et N^2 affectations). Ce coût de remise-à-jour est donc supérieur à un recalcul total (N^2 additions et N^2 affectations). Ceci implique que pour un nombre de modifications de classes inférieur à approximativement $\frac{N}{2}$, l'approche par mise-à-jour est plus efficace que le recalcul total. Il est impotant de noter que dans le cas où le recalcul total est nécessaire, la stratégie de mémorisation présentée au paragraphe ci-dessus peut être utilisée.

4. Expériences

Les différentes optimisations ont été évaluées sur un jeu de données simulées. Il consiste en N points de $\mathbb{R}^2$ choisis aléatoirement et uniformément dans le carré unité. $\mathbb{R}^2$ est muni de la distance euclidienne. La topologie du DSOM est une grille hexagonale munie de sa distance de graphe δ. On choisit $L = 100$ itérations et la fonction de voisinage est gaussienne.

N (nb d'individus) M (nb de modèles)	500	1 000	1 500	2 000	3 000
$49 = 7 \times 7$	11.4 / 0.8	53.5 / 2.3	135.4 / 4.8	261.6 / 8.5	865.3 / 22.5
$100 = 10 \times 10$	24.7 / 2.4	115.0 / 5.6	283.4 / 9.8	557.0 / 15.3	1757.0 / 32.8
$225 = 15 \times 15$		313.7 / 30.4	806.6 / 46.4	1594.8 / 63.3	4455.5 / 105.3
$400 = 20 \times 20$			1336.9 / 136.1	2525.2 / 179.1	7151.8 / 264.6

TAB. 1. *Temps d'exécution en secondes pour l'algorithme standard / pour l'algorithme avec sommes partielles*

Le tableau 1 donne les performances[2] de l'algorithme standard et de l'algorithme basé sur les sommes partielles. Pour l'algorithme standard, le coût est quadratique en N et relativement linéaire en M. L'amélioration apportée par les sommes partielles est impressionante. Le rapport entre les deux algorithmes est approximativement proportionnel à $\frac{NM}{N+M^2}$, qui est le rapport théorique (la phase d'affectation n'est pas prise en compte).

N (nb d'individus) M (nb de modèles)	500	1 000	1 500	2 000	3 000
$49 = 7 \times 7$	1.14 / 1.6	1.05 / 1.92	1.00 / 2	0.97 / 2.02	0.98 / 2.39
$100 = 10 \times 10$	1.41 / 1.85	1.33 / 1.81	1.23 / 1.66	1.15 / 1.65	1.08 / 1.83
$225 = 15 \times 15$		2.27 / 2.87	2.13 / 2.67	2.00 / 2.57	1.78 / 2.43
$400 = 20 \times 20$			2.74 / 3.04	2.75 / 3.2	2.48 / 2.89

TAB. 2. *Taux d'accélération pour l'algorithme avec arrêt prématuré et ordre sans / avec mémorisation (référence : algorithme avec sommes partielles)*

Le tableau 2 donne les performances des algorithmes avec arrêt prématuré et ordre avec ou sans mémorisation. Ces expériences montrent clairement que chacune des optimisations proposées apporte un gain important en terme de temps d'exécution.

5. Conclusions

Nous avons proposé dans ce travail un nouvel algorithme pour le SOM appliqué aux tableaux de dissimilarités. Cet algorithme permet une réduction importante du coût théorique. De plus, certaines techniques d'implémentation ont permis d'accélérer d'un facteur 3 le temps d'exécution sous des conditions favorables.

6. Bibliographie

T. Kohonen (1995,1997,2001), *Self-Organizing Maps*, Vol. 30 of Springer Series in Information Sciences, Springer.

T. Kohonen, P. J. Somervuo (1998), Self-Organizing Maps of symbol strings *Neurocomputing* **vol. 21** p. 19-30

T. Kohonen, P. J. Somervuo (2002), How to make large Self-Organizing Maps for nonvectorial data *Neural Networks* **vol. 15 (8)** p. 945-952

A. El Golli, B. Conan-Guez, F. Rossi Self-Organizing Map and symbolic data *Journal of Symbolic Data Analysis* **vol. 2(1)**

2. L'algorithme a été implémenté en Java (JRE 5.0 de Sun) sur un PC équipé d'un processeur Pentium IV (3 GHz) sous le système d'exploitation Linux