Apprendre les contraintes topologiques dans les cartes auto-organisatrices

Guénaël Cabanes*, Younès Bennani*

*LIPN-CNRS, UMR 7030
99 Avenue J-B. Clément, 93430 Villetaneuse, France
cabanes@lipn.univ-paris13.fr

Résumé. La Carte Auto-Organisatrice (SOM : Self-Organizing Map) est une méthode populaire pour l'analyse de la structure d'un ensemble de données. Cependant, certaines contraintes topologiques de la SOM sont fixées avant l'apprentissage et peuvent ne pas être pertinentes pour la représentation de la structure des données. Dans cet article nous nous proposons d'améliorer les performances des SOM avec un nouvel algorithme qui apprend les contraintes topologiques de la carte à partir des données. Des expériences sur des bases de données artificielles et réelles montrent que l'algorithme proposé produit de meilleurs résultats que SOM classique. Ce n'est pas le cas avec une relaxation triviale des contraintes topologiques, qui résulte en une forte augmentation de l'erreur topologique de la carte.

1 Introduction

Une Carte Auto-Organisatrice ou Self-Organizing Map (SOM : Kohonen, 2001) se compose d'un ensemble de neurones artificiels, qui représentent la structure des données. Les neurones sont connectés avec des connexions topologiques pour former une grille à deux dimensions. Deux neurones connectés devraient représenter le même type de données, deux neurones distants (sur la carte) doivent représenter des données différentes. Ces propriétés sont assurées pendant le processus d'apprentissage grâce aux informations de voisinage qui imposent des contraintes topologiques.

Toutefois, dans l'algorithme SOM, l'information topologique est fixée avant le processus d'apprentissage et peut ne pas être pertinente par rapport à la structure des données. Pour résoudre ce problème, certains travaux ont été réalisés afin d'adapter le nombre de neurones au cours du processus d'apprentissage en fonction des données à analyser (Fritzke, 1995). Les résultats ont montré que la qualité du modèle est améliorée lorsque le nombre de neurones est appris à partir des données.

En dépit de ces résultats, il y a très peu de travaux qui abordent le problème de l'apprentissage des contraintes topologiques en fonction de la structure des données. Pourtant, à la fin du processus d'apprentissage, des neurones "voisins" peuvent ne pas représenter des données similaires (Cabanes et Bennani, 2007, 2008; Matsushita et Nishio, 2008). Dans l'algorithme False Neighbor-SOM (FN-SOM : Matsushita et Nishio, 2008), les auteurs proposent de conserver la topologie bidimensionnelle de la SOM, mais en associant à chaque "ligne" ou "colonne"

de connexions un indice de voisinage qui est lié à la validité globale de la “ligne” ou de la “colonne” (dans cet algorithme, chaque neurone est limité à seulement quatre voisins ; cf. Fig. 1). Comme la même valeur est utilisée pour toutes les connexions appartenant à la même “ligne” ou “colonne”, la méthode peut conduire à la structure de topologie incorrecte, en particulier pour les bases de données de grande dimension.

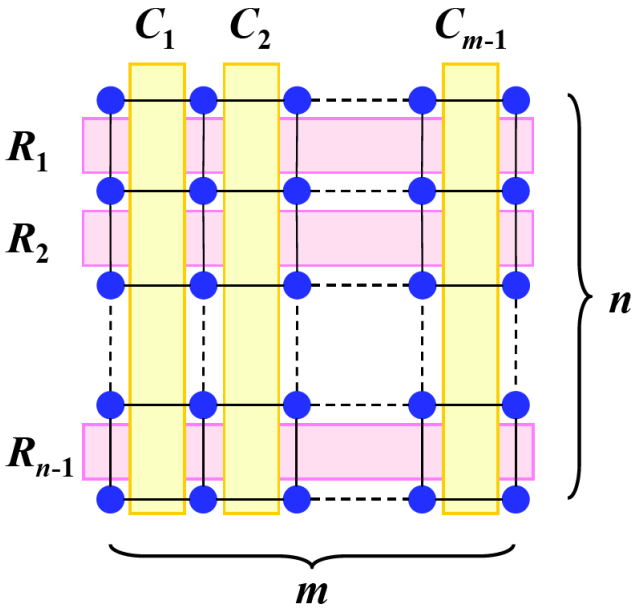


FIG. 1 – *Dans FN-SOM des valeurs sont associées à chaque ligne et colonne d'une SOM de topologie rectangulaire.*

Nous proposons donc dans cette partie d'améliorer les performances SOM avec un nouvel algorithme, Data-Driven Relaxation - SOM (DDR-SOM), capable d'apprendre la topologie de la carte à partir de la structure de données. Cet algorithme permet une visualisation en deux dimensions des résultats et le nombre de voisins n'est pas limité (un voisinage de six est le plus couramment utilisé dans SOM). L'idée principale est d'utiliser des valeurs associées aux connexions, afin de réduire certaines contraintes topologiques entre les neurones qui représentent des données différentes. Ces relâchements de contraintes sont censés améliorer la qualité de la SOM, notamment en réduisant l'erreur de quantification de la carte et l'augmentation du nombre de neurones qui participent réellement à la représentation des données.

Le reste de cet article est organisé comme suit. La Section 2 présente l'algorithme d'auto-organisation de la carte. La section 3 décrit l'algorithme DDR-SOM. La Section 4 montre les validations expérimentales que nous avons effectuées et les résultats obtenus. Enfin, une conclusion est donnée dans la section 5.

2 Algorithme des Cartes Auto-Organisatrices

Une SOM consiste en une carte de neurones en deux dimensions. Ces neurones sont connectés avec leurs voisins selon des connexions topologiques (où connexions de voisinage) (Kohonen, 1984, 2001). L'ensemble de données à analyser est utilisé pour organiser la SOM sous des contraintes topologiques de l'espace d'entrée. Ainsi, une correspondance entre l'espace d'entrée et l'espace de la carte est construit. Deux observations proches dans l'espace d'entrée doivent activer le même neurone ou deux neurones voisins de la SOM. Pour vérifier ces contraintes, les neurones voisins du neurone le plus représentatif d'une donnée mettent à jour leur prototype pour une meilleure représentation de cette donnée. Cette mise à jour est d'autant plus importante que les neurones sont de proches voisins du meilleur neurone.

L'apprentissage de la SOM peut être vue comme la minimisation d'une fonction de coût :

$$\tilde{R}(w) = \frac{1}{N} \sum_{k=1}^{N} \sum_{j=1}^{M} K_{ju^*(x^{(k)})} \parallel x^{(k)} - w_j \parallel^2$$

Avec N le nombre de données, M le nombre de neurones de la carte, $u^*(x^{(k)})$ est le neurone i dont le vecteur prototype w_i est le plus proche de la donnée $x^{(k)}$. Il s'agit donc de minimiser la distance entre les données et les prototypes, selon une pondération donnée par la fonction Noyau K_{ij} qui représente les contraintes topologiques à respecter, par exemple :

$$K_{ij} = \frac{1}{\lambda(t)} \times e^{-\frac{d_M^2(i,j)}{\lambda^2(t)}}$$

Avec $\lambda(t)$ la fonction température qui fait diminuer les contraintes topologiques avec le temps, de façon à assurer la convergence de l'apprentissage (pour un rayon de voisinage fixe, cf. Cheng (1997)). $d_M(i,j)$ est la distance de voisinage entre deux neurones i et j de la SOM, calculée par la distance de Manhattan. Il s'agit du nombre minimal de connexions topologiques entre i et j (voir la Fig. 2 pour un exemple).

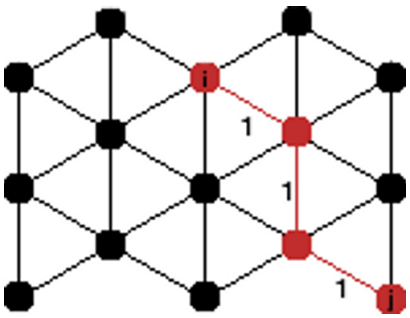


FIG. 2 – *Distance de Manhattan pondérée entre les neurones i et j pour une topologie hexagonale. Ici $d_M(i,j) = 3$, il s'agit du nombre minimal de connexions topologiques entre i et j.*

Notre algorithme est adapté de la version Batch de SOM, qui se présente de la façon suivante :

1. **Phase d'initialisation** :
   – Choisir la topologie de la SOM.
   – Initialiser des prototypes w de la carte.
2. **Phase d'affectation** :
   – Déterminer le prototype le plus représentatif (ou BMU : Best Mach Unit) u^* pour chaque donnée d'entrée $x^{(k)}$:

$$u^*(x^{(k)}) = \underset{1 \leq i \leq M}{Argmin} \parallel x^{(k)} - w_i \parallel^2$$

3. **Phase de représentation** :
   – Mettre à jour les prototypes w_i de la carte pour chaque neurone i de façon à minimiser la fonction de coût :

$$w_i = \frac{\sum_{k=1}^{N} K_{iu^*(x^{(k)})}.x^{(k)}}{\sum_{k=1}^{N} K_{iu^*(x^{(k)})}}$$

4. **Répéter les phases 2 et 3** jusqu'à ce que $t = t_{max}$

3 Relâchement des contraintes topologiques guidé par les données

3.1 Principe

Dans l'algorithme DDR-SOM, nous proposons d'associer à chaque connexion de voisinage une valeur réelle v qui indique la pertinence des neurones connectés. Compte tenu de la contrainte d'organisation de la SOM, les deux neurones les plus représentatifs d'un ensemble de données doivent être reliés par une liaison de voisinage. Ainsi, une paire de neurones voisins, qui sont tout deux de bons représentants d'un même ensemble de données devraient être fortement connectés, tandis qu'une paire de neurones voisins qui ne représentent pas le même type de données doit être faiblement connectés. On associe donc à chaque connexion de voisinage une valeur réelle variant de 1 pour un lien fort à 2 pour un lien faible. On utilise pour cela une fonction logistique qui dépend du nombre de données bien représentées par chacun des deux neurones connectés.

Ces valeurs sont ensuite utilisées pour estimer une distance de Manhattan pondérée entre chaque paire de neurones voisins i et j, notée $d_{WM}(i, j)$. Cette distance est le nombre de connexions entre i et j, pondéré par les valeurs associées à chaque connexion (voir la Fig. 3 pour un exemple).

Nous utilisons l'algorithme de Johnson (Johnson, 1977) pour calculer :
– le chemin le plus court le long des connexions de voisinages entre chaque paire de neurones.
– la longueur de ce chemin en fonction des valeurs v associées à chaque connexion.

De cette manière, la distance entre deux neurones est supposée refléter le voisinage "réel" de ces deux neurones.

Les valeurs de connexion et les distances pondérées entre neurones sont mises à jour au cours de l'apprentissage. De cette façon, la SOM obtenue devrait être un meilleur représentant de la base de données que l'algorithme SOM classique.

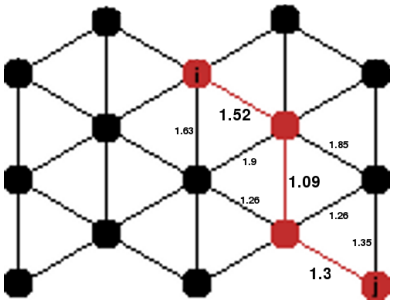


FIG. 3 – *Distance de Manhattan pondérée entre les neurones i et j pour une topologie hexagonale. Ici $d_{WM}(i,j) = 1.52 + 1.09 + 1.3 = 3.91$, il s'agit du plus court chemin entre i et j en fonction des valeurs de connexion v.*

3.2 Nouvel Algorithme

L'algorithme DDR-SOM procède en trois phases :

1. **Phase d'initialisation** :
   - Choisir la topologie de la SOM.
   - Initialiser des prototypes w de la carte.
   - Initialiser à zéro toutes les valeurs des connexions de voisinage v.
2. **Phase de compétition** :
   - Déterminer le premier BMU u^* et le second BMU u^{**} pour chaque donnée d'entrée $x^{(k)}$:

$$u^*(x^{(k)}) = \underset{1 \leq i \leq M}{Argmin} \parallel x^{(k)} - w_i \parallel^2$$

   et

$$u^{**}(x^{(k)}) = \underset{1 \leq i \leq M,\ i \neq u^*}{Argmin} \parallel x^{(k)} - w_i \parallel^2$$

   - Mettre à jour les valeurs v des connexions de voisinage selon la règle suivante :

$$v_{i,j} = \frac{1 + 2e^{\frac{Nth - N(i,j)}{\sigma Nth}}}{1 + e^{\frac{Nth - N(i,j)}{\sigma Nth}}}$$

   Avec $v_{i,j}$ une fonction logistique variant entre 1 (lorsque i et j représentent les mêmes données) et 2 (lorsque i et j représentent des données différentes). $N(i,j)$ est le nombre de données (x) ayant i et j dans $\{u^*(x), u^{**}(x)\}$. Nth est la valeur théorique de $N(i,j)$ sous hypothèse d'homogénéité, autrement dit Nth est la moyenne de $N(i,j)$ sur toutes les connexions de voisinage. Pour finir, σ est un paramètre de la fonction logistique ; il doit être choisi par l'utilisateur.

3. **Phase d'adaptation** :

– Mettre à jour les prototypes w_i de la carte pour chaque neurone i. Notez que les valeurs de K_{ij} dépendent des valeurs $v_{i,j}$:

$$w_i = \frac{\sum_{k=1}^{N} K_{iu^*(x^{(k)})}.x^{(k)}}{\sum_{k=1}^{N} K_{iu^*(x^{(k)})}}$$

4. **Répéter les phases 2 et 3** jusqu'à ce que $t = t_{max}$

4 Résultats expérimentaux

4.1 Description des bases de données utilisées

Pour tester la validité du nouvel algorithme, nous avons utilisé 10 bases de données artificielles et réelles de tailles et de dimensions variables. Les caractéristiques de chaque base sont décrites dans la Table 1.

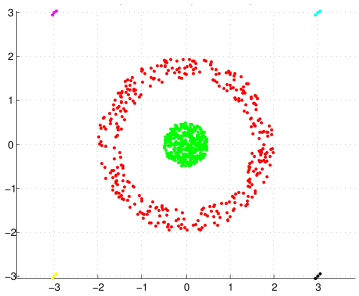
TAB. 1 – *Description des bases de données utilisées.*

Base de données	Type	Taille	Dimension
Target	Artificielle	770	2
TwoDiamonds	Artificielle	800	2
Hepta	Artificielle	212	3
Tetra	Artificielle	400	3
Iris	Réelle	150	4
Harot	Réelle	132	6
Housing	Réelle	506	13
Wine	Réelle	178	13
Cockroach	Réelle	1369	3
Chromato	Réelle	134	60

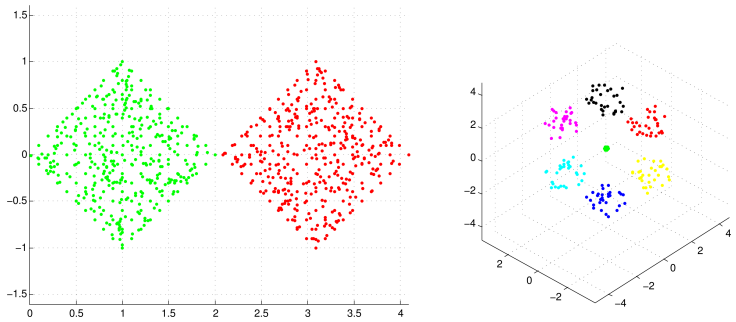
Les bases de données "Target", "TwoDiamonds", "Tetra" et "Hepta" viennent du "Fundamental Clustering Problem Suite" (FCPS : Ultsch, 2005). Il s'agit de données artificielles de faible dimension dont la structure est parfaitement connue. Elles sont souvent utilisées comme bases de test pour les algorithmes de clustering (Cabanes et Bennani, 2008; Matsushita et Nishio, 2008; Ultsch, 2005).

Les bases de données "Housing", "Harot", "Iris" et "Wine" sont des données réelles bien connues de l'UCI repository (Frank et Asuncion, 2010). Elles sont de tailles et de dimensions variables. Pour finir, "Cockroach" et "Chromato" sont des bases de données réelles très bruitées provenant d'expériences dans le domaine de la biologie.

Ces bases de données représentent en partie la diversité des problèmes pouvant être rencontrés par les utilisateurs de SOM lors de l'analyse de bases de données.



(a) “Target” et “Tetra”.



(b) “TwoDiamonds” et “Hepta”

FIG. 4 – *Visualisation des bases de données.*

4.2 Estimation de la qualité d’une SOM

Pour évaluer les performances en apprentissage de notre algorithme, nous utilisons trois indices classiques de qualité pour les algorithmes de type SOM :

Erreur de Quantification Qe :
Cet indice mesure la distance moyenne entre chaque donnée et son BMU (Kohonen, 2001).

$$Qe = \frac{1}{N} \sum_{k=1}^{N} \parallel x^{(k)} - w_{u^*(x^{(k)})} \parallel^2$$

Plus la valeur de Qe est petite, plus la qualité de l’algorithme est grande.

Erreur Topographique Te :
Te décrit la façon dont la SOM préserve la topologie de l’ensemble des données étudiées (Kiviluoto, 1996). Elle mesure la proportion des données ayant les deux premiers BMU non adjacents (non reliés par une connexion de voisinage). Une faible valeur de Te est signe de qualité. Contrairement à l’erreur de Quantification, Te prend en compte la structure de la SOM.

Utilisation des Neurones Ne :
Ne mesure le pourcentage de neurones qui ne sont jamais le BMU d’une donnée de la base (Cheung et Law, 2007). Une bonne SOM devrait avoir une faible Ne, c’est-à-dire que chaque neurone devrait être utilisé dans la représentation des données, afin de ne pas gaspiller

de ressources de calcul (en particulier pour l'analyse de grandes bases de données pour les applications réelles).

Dans toutes les expériences suivantes, les indices sont normalisés afin de pouvoir comparer efficacement les résultats sur les différentes bases de données. Pour représenter un gain ou une perte par rapport à l'algorithme classique de SOM, chaque indice est divisé par la valeur obtenue avec l'algorithme SOM (l'erreur de SOM est donc toujours égale à 1). Pour toutes les expériences de cette section, nous avons utilisé le paquet Matlab SOM-Toolbox (Vesanto et al., 1999), tous les paramètres des SOM ont été fixés aux valeurs par défaut (en particulier, nous utilisons une grille hexagonale comme topologie initiale de la carte).

4.3 Effet d'un relâchement trivial des contraintes topologiques

Le principe essentiel du nouvel algorithme est de réduire la contrainte topologique de la SOM en augmentant la distance entre les neurones. Dans DDR-SOM, ces modifications sont guidées par les données pour optimiser la qualité finale de la SOM.

La première étape de notre expérimentation est d'analyser comment se comporte une SOM si on diminue de façon triviale les contraintes topologiques. Nous prévoyons qu'une plus faible contrainte conduit à un meilleur modèle (c.-à-d. de plus faibles erreurs de Quantification et d'Utilisation des Neurones), mais conduit également à une augmentation de l'erreur Topologique, puisque la fonction des contraintes topologiques de la SOM est de réduire cette erreur.

Pour le vérifier, nous avons calculé les erreurs de Quantification, d'Utilisation des Neurones et Topologique pour chaque base de données à partir des résultats de différents relâchements des contraintes topologiques de l'algorithme SOM, où chaque distance entre deux neurones est multipliée par une valeur constante (voir Fig. 5). Plus cette constante est grande, plus les contraintes topologiques sont faibles. Par exemple, SOM(α) est similaire à l'algorithme SOM, mais $d(i,j) = \alpha \times d_M(i,j)$. Nous avons testé différentes valeurs de cette constante, de SOM(1), similaire à SOM, à SOM(10), où les neurones sont pratiquement indépendants (dans ce cas le comportement de l'algorithme est similaire à celui des K-Moyennes).

Les résultats sont résumés dans les Fig. 5. Cette figure montre la valeur moyenne de Qe, Ne et Te sur toutes les bases de données pour différents relâchements des contraintes topologiques. Comme prévu, Ne et Qe diminuent lorsque les contraintes topologiques s'affaiblissent, tandis que Te augmente fortement.

Puisque le gain en Ne et Qe est associé à une perte en Te, nous proposons de définir une Erreur Générale qui reflète les compromis entre Ne, Qe et Te :

$$Ge = Te^2 \times Ne \times Qe$$

Ge est le produit de deux compromis : Ne vs. Te et Qe vs. Te. La valeur de Ge est plus faible lorsque le gain en Ne et Qe est supérieur à la perte en Te par rapport à l'algorithme SOM. Ge est plus grande dans le cas contraire. Les valeurs moyennes de Ge sur toutes les bases de données pour différentes valeurs de α sont représentées dans la Fig. 5.

Ces résultats montrent que, sous l'effet d'une diminution triviale des contraintes topologiques, le gain en Ne et Qe ne peut pas compenser la perte en Te. Ainsi, le meilleur compromis est d'utiliser l'algorithme SOM classique !

Maintenant la question est : peut-on utiliser les données d'apprentissage pour trouver une relaxation des contraintes topologiques qui soit un meilleur compromis que la SOM ?

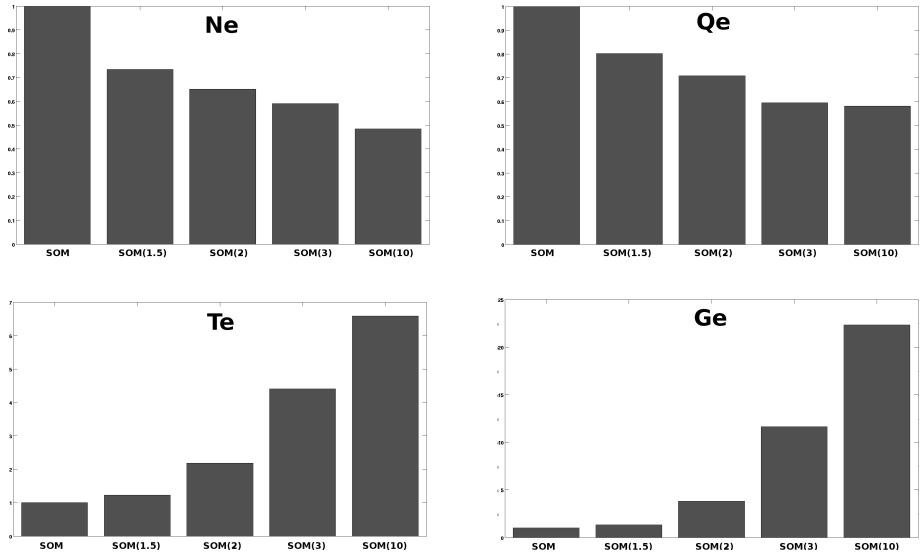


FIG. 5 – *Visualisation de la valeur moyenne de Qe, Ne, Te et Ge sur l'ensemble des bases de données pour différents relâchements des contraintes topologiques dans SOM.*

4.4 Évaluation du nouvel algorithme guidé par les données

Pour évaluer les performances de DDR-SOM, nous avons comparé SOM et différentes versions de DDR-SOM avec différentes valeurs du paramètre σ. La Fig. 6 montre les valeurs moyennes de Qe, Ne, Te et Ge sur toutes les bases de données. Les valeurs de Ge pour chaque base de données et chaque version de DDR-SOM sont indiquées dans la Table 2. Enfin, la Fig. 7 illustre la qualité de DDR-SOM en comparaison de l'algorithme SOM.

TAB. 2 – *Valeurs de Ge obtenues par DDR-SOM pour chaque base de données avec différentes valeurs de σ.*

	DDR(1)	DDR(1/2)	DDR(1/5)	DDR(1/10)
Target	0,57	0,39	0,87	0,89
TwoDiamonds	0,22	0,27	0,22	0,11
Hepta	1,82	0,62	0,97	0,57
Tetra	1,17	0,53	1,61	1,12
Iris	0,09	0,21	0,29	0,28
Harot	0,79	0,17	0,06	0,38
Housing	0,83	0,54	0,35	0,55
Wine	0,51	0,14	0,13	0,33
Cockroach	0,62	0,42	0,52	0,54
Chromato	0,12	0,08	0,09	0,09

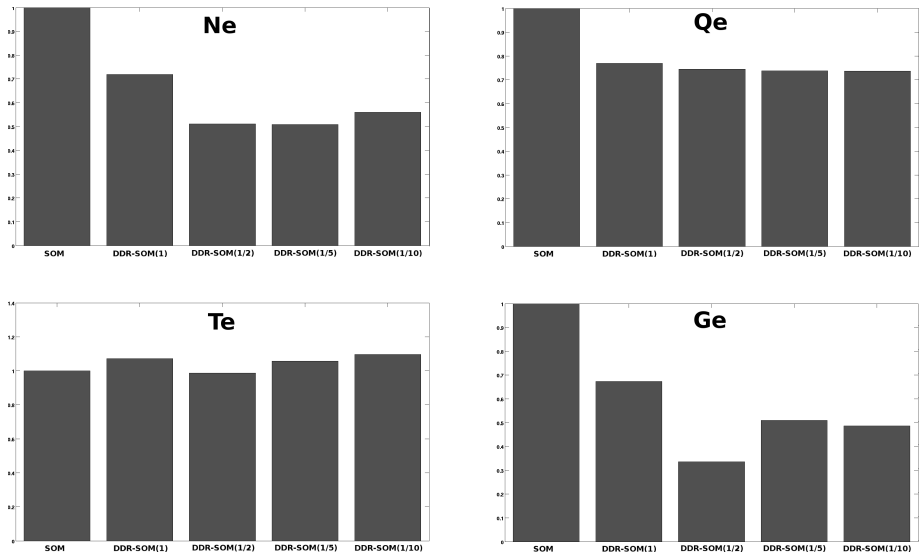


FIG. 6 – *Visualisation de la valeur moyenne de Qe, Ne, Te et Ge sur l'ensemble des bases de données pour différentes valeurs de σ avec DDR-SOM.*

Nous pouvons noter que les contraintes topologiques moyennes de DDR-SOM sont similaires à celles de SOM(1.5). Cependant, comme nous pouvons le constater, les performances de l'algorithme DDR-SOM sont bien meilleures que celles de SOM(1.5) et de SOM, autrement dit le gain en Ne et Qe est supérieur à la perte en Te, par rapport à SOM. En effet, avec DDR-SOM, l'erreur de Quantification est semblable à celle de SOM(2) et Ne est très faible, semblable à SOM(10), tandis que l'erreur Topologique de DDR-SOM est semblable à l'erreur obtenue par l'algorithme SOM classique.

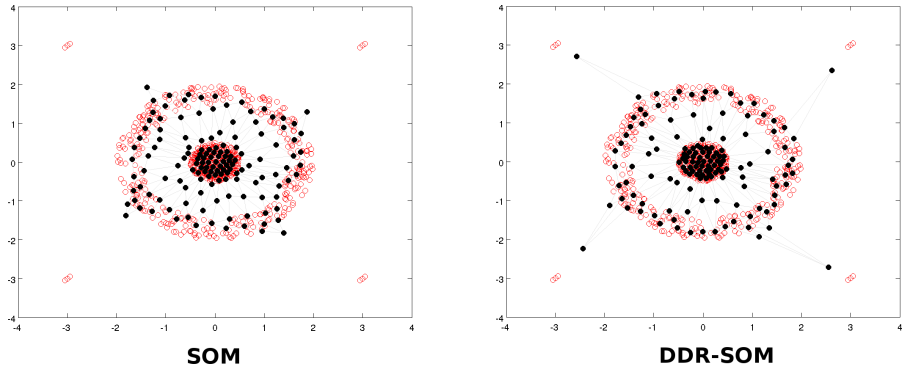
Nous pouvons faire deux commentaires à partir de ces résultats :

1. Les performances de DDR-SOM sont meilleures que celles de SOM pour toutes les valeurs de σ, cependant une valeur de $\sigma = 1/2$ semble donner de meilleurs résultats pour ces bases de données.
2. Le gain en Ge a tendance à être plus élevé pour les bases de données de grandes dimensions (par exemple "Chromato", "Wine", etc ...).

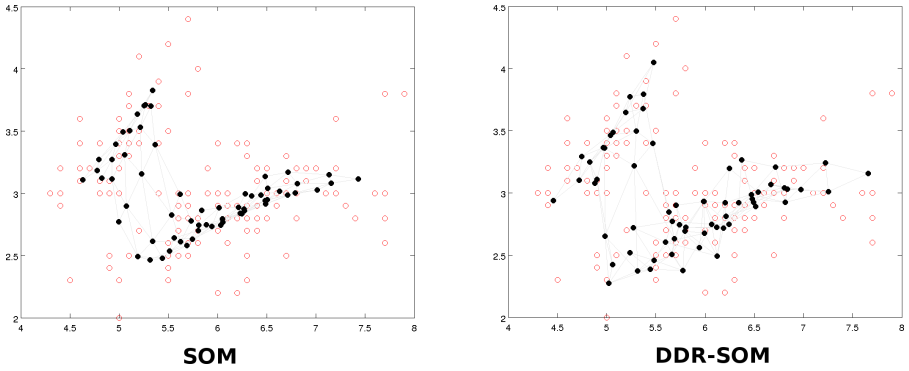
De plus, si l'on se base sur l'article de (Matsushita et Nishio, 2008), on constate que dans le cas des cartes 10x10 à topologie rectangulaire, DDR-SOM est en moyenne 25% plus performant que FN-SOM pour les quelques bases de données utilisées par les auteurs ("Tetra", "Hepta", "Target" et "Iris").

5 Conclusion

Dans cet article, nous proposons un nouvel algorithme adapté de SOM dans le but d'améliorer la qualité du modèle obtenu par apprentissage. Pour cela nous utilisons un relâchement



(a) Données “Target”.



(b) Données “Iris” (2 premières variables).

FIG. 7 – *Visualisation des résultats obtenus avec SOM et DDR-SOM(1/2) pour deux bases de données. Les données sont en rouge et la grille de prototypes à la fin de l'apprentissage est représentée en noir.*

des contraintes topologiques de la SOM guidé par les données. Nous avons défini une erreur globale qui représente le compromis entre les erreurs Topologiques, de Quantification et d'Utilisation des Neurones.

Les expériences sur des bases de données artificielles et réelles montrent que l'algorithme DDR-SOM obtient de meilleurs résultats que l'algorithme SOM. Nous avons également montré que cette amélioration n'est pas obtenue avec une relaxation triviale des contraintes topologiques, en raison d'une forte augmentation de l'erreur Topologique. Une diminution des contraintes guidée par les données semble être une bonne solution pour améliorer le compromis $NeQe/Te$ de la SOM.

6 Remerciements

Ce travail a été soutenu en partie par le projet $CADI$ (N^o ANR-07 TLOG 003), financé par l'ANR (Agence Nationale de la Recherche).

Références

Cabanes, G. et Y. Bennani (2007). A simultaneous two-level clustering algorithm for automatic model selection. In *Proceedings of the International Conference on Machine Learning and Applications (ICMLA'07)*, pp. 316–321.

Cabanes, G. et Y. Bennani (2008). A local density-based simultaneous two-level algorithm for topographic clustering. In *Proceeding of the International Joint Conference on Neural Networks*, pp. 1176–1182.

Cheng, Y. (1997). Convergence and ordering of kohonen's batch map. *Neural Comput. 9*, 1667–1676.

Cheung, Y. et L. Law (2007). Rival-Model Penalized Self-Organizing Map. *IEEE Trans. Neural Networks 18*(1), 289–295.

Frank, A. et A. Asuncion (2010). UCI machine learning repository.

Fritzke, B. (1995). Growing grid - a self-organizing network with constant neighborhood range and adaptation strength. *Neural Processing Letters 2*(5), 9–13.

Johnson, D. (1977). Efficient algorithms for shortest paths in sparse networks. *Journal of the ACM 24*(1), 1–13.

Kiviluoto, K. (1996). Topology Preservation in Self-Organizing Maps. *International Conference on Neural Networks*, 294–299.

Kohonen, T. (1984). *Self-Organization and Associative Memory*. Berlin : Springer-Verlag.

Kohonen, T. (2001). *Self-Organizing Maps*. Berlin : Springer-Verlag.

Matsushita, H. et Y. Nishio (2008). Self-Organizing Map with False-Neighbor Degree between Neurons for Effective Self-Organization. *IEICE Transactions on Fundamentals E91-A*(6), 1463–1469.

Ultsch, A. (2005). Clustering with SOM : U*C. In *Procedings of the Workshop on Self-Organizing Maps*, pp. 75–82.

Vesanto, J., J. Himberg, E. Alhoniemi, et J. Parhankangas (1999). Self-Organizing Map in Matlab : the SOM Toolbox. *Proceedings of the Matlab DSP Conference*, 35–40.

Summary

The Self-Organizing Map (SOM) is a popular algorithm to analyze the structure of a dataset. However, some topological constraints of the SOM are fixed before the learning and may not be relevant regarding to the data structure. In this paper we propose to improve the SOM performance with a new algorithm which learn the topological constraints of the map using data structure information. Experiments on artificial and real databases show that algorithm achieve better results than SOM. This is not the case with trivial topological constraint relaxation because of the high increase of the Topological error.