

## Chapitre 2

# Modèle d'architecture CxSOM

### Sommaire

---

<b>2.1</b>	<b>Carte de Kohonen classique</b>	<b>12</b>
2.1.1	Algorithme et notations	12
2.1.2	Paramétrage d'une carte de Kohonen	13
<b>2.2</b>	<b>Motivations du modèle CxSOM</b>	<b>16</b>
2.2.1	Champ d'application : mémoire associative	16
2.2.2	Description de l'architecture	16
<b>2.3</b>	<b>Présentation de CxSOM : exemple d'une architecture de deux cartes</b>	<b>18</b>
2.3.1	Détail des étapes	18
2.3.2	Résumé	23
<b>2.4</b>	<b>Formalisation : cas pour <math>n</math> cartes</b>	<b>23</b>
2.4.1	Entrées et calcul d'activité	23
2.4.2	Calcul du BMU par relaxation	24
2.4.3	Mise à jour des poids	25
<b>2.5</b>	<b>Choix des paramètres</b>	<b>26</b>
2.5.1	Paramétrage d'une carte	26
2.5.2	Paramètres de l'architecture	27
<b>2.6</b>	<b>Conclusion</b>	<b>27</b>

---

Nous proposons dans cette thèse un modèle d'architecture de cartes auto-organisatrices, CxSOM. Par ce modèle, on associe des cartes auto-organisatrices en architecture afin de réaliser des tâches de mémoire associative : apprendre un modèle de relation entre des ensembles de données issues de plusieurs modalités. On souhaite que ce modèle soit générique, permette de construire n'importe quel forme et taille d'architecture, et aie la possibilité d'intégrer des connexions récurrentes. Notre démarche est d'abord de proposer un nouveau modèle de calcul général à base de cartes auto-organisatrices ; des applications plus spécifiques pourront être développées à partir de cette méthode. Nous avons publié le modèle CxSOM en [11].

On définit une *architecture* de carte par un réseau composé de plusieurs modules qui sont chacun des cartes de Kohonen et dans lequel des connexions sont définies entre ces modules. Ces connexions sont orientées : on parle d'une connexion d'une carte A vers une carte B. Le but de ces connexions est de coupler l'apprentissage de plusieurs cartes. Sur une telle architecture, on peut construire un graphe  $G$  orienté, dont les nœuds sont des cartes. La connexion d'une carte A vers une carte B est indiquée par la présence d'une arête de A vers B. On appelle architecture

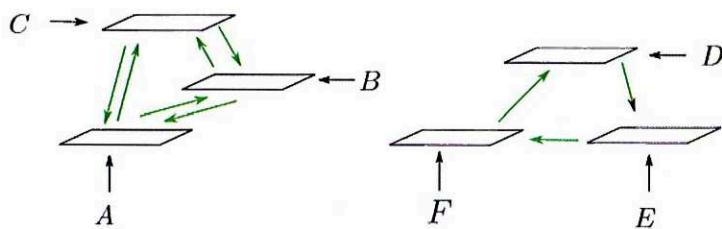


FIGURE 2.1 – Deux exemples d'architectures *non-hiéronymiques* à 3 cartes de Kohonen : des connexions sont réciproques, ou des boucles sont présentes au sein de l'architecture. Les entrées sont  $A, B, C, D, E, F$  quelconques.

*non-hiéronymique* une architecture pour laquelle le graphe  $G$  n'est pas un arbre : il présente des boucles. Un exemple d'architecture non-hiéronymique est par exemple représenté en figure 2.1. Certaines cartes sont connectées dans les deux directions, d'autres en boucle. Dans cette thèse, nous chercherons à utiliser de telles architectures non-hiéronymiques pour des tâches de mémoire associative. Ici, chaque carte de l'architecture cherche à apprendre une représentation ~~d'une des~~ des entrées  $A, B, C, D, E$  qui lui est fournie. Lorsque ces entrées sont dépendantes les unes des autres, l'architecture dans sa globalité doit également, d'une façon ou d'une autre extraire les relations, ~~les dépendances, existant entre les distributions de données~~. Cet apprentissage de relations est au cœur de cette thèse et sera détaillé dans le chapitre d'analyse de l'architecture.

Dans ce chapitre, nous détaillons d'abord le modèle CxSOM développé et étudié durant cette thèse, permettant de construire des architectures non-hiéronymiques de cartes auto-organisatrices. Nous présentons en premier lieu le formalisme d'une carte de Kohonen classique, dont sont dérivées les cartes auto-organisatrices utilisées dans les architectures CxSOM. Nous expliquerons ensuite les choix de développement sur lesquels nous nous sommes appuyés pour développer le modèle. Nous présenterons le modèle sur un exemple d'architecture à deux cartes, puis nous le formaliserons pour le cas général de  $n$  cartes connectées au sein d'une architecture. Le glossaire des notations est fourni en annexe.

## 2.1 Carte de Kohonen classique

*Vraiment préciser dans cette section  
activité voisiniage) ?  
tx d'app*

Chaque carte de Kohonen d'une architecture CxSOM est directement dérivée de l'algorithme d'une carte de Kohonen classique introduite en [16], et décrite extensivement par Kohonen en [17]. Le principe général d'une carte de Kohonen a été décrit dans le chapitre précédent ; nous définissons ici plus précisément le modèle et les équations qui serviront de base pour la définition de l'algorithme CxSOM.

### 2.1.1 Algorithme et notations

Une carte de Kohonen est un graphe, généralement une ligne 1D ou une grille 2D de  $N$  noeuds. Nous utiliserons dans cette thèse des cartes en une et deux dimensions, c'est à dire des lignes et des grilles. Les notations et le modèle présentés ici sont toutefois applicables à des cartes de dimensions et topologie quelconques.

L'algorithme et les notations sont résumés en figure 2.2. Une entrée fournie à une carte de Kohonen est notée  $X_t$ , tirée dans un espace d'entrée  $\mathcal{D}$ . À chaque noeud de la carte est associé

un poids, appelé aussi prototype, noté  $\omega_e \in D$ . Sa *position* dans la carte est indexée par  $p$ . Nous choisissons d'indexer les positions entre 0 et 1 : l'ensemble des positions  $p$  est donc un ensemble de points discrets entre 0 et 1. L'ensemble des poids est noté  $\{\omega_e(p), p \in \{0, \dots, \frac{i}{N}, \dots, 1\}\}$ , avec  $i$  l'indice entier d'un noeud de la carte. On peut faire la même discrétisation de l'espace pour une carte en 2D.

Une étape  $t$  de l'algorithme de mise à jour d'une carte de Kohonen contient les étapes suivantes :

1. Une entrée  $X_t$  est tirée et présentée à la carte.
2. Une *activité*  $a_e(X_t, p)$  est calculée dans la carte pour chaque noeud de position  $p$ . La fonction d'activité choisie est gaussienne :

$$a_e(X_t, p) = \exp \frac{\|X_t - \omega_e(p)\|^2}{2\sigma^2} \quad (2.1)$$

3. L'unité ayant l'activité maximale est la *Best Matching Unit* de la carte. Sa position est notée  $\Pi$ . Cette étape est déjà une modification de l'algorithme original de Kohonen. Dans la version classique, on calcule plutôt les distances entre l'entrée et les poids  $\|X_t - \omega_e(p)\|$ , et le BMU est choisi comme l'unité dont le poids présente la plus petite distance à l'entrée. Ici, on prendra comme BMU l'unité ayant l'activité la plus élevée.
4. Chaque poids  $\omega_e$  est déplacé vers l'entrée  $X$ . Le déplacement est pondéré par une *fonction de voisinage*  $H(\Pi, p)$ . Cette fonction est maximale en  $p = \Pi$  et décroissante autour de cette position. Dans notre étude, la fonction de voisinage est triangulaire, donc maximale en  $\Pi$ , décroissante sur un *rayon de voisinage* noté  $h_e$  et nulle sinon.

$$\omega_e(p, t+1) = \omega_e(p, t) + \alpha H(\Pi, p)(X_t - \omega_e(p, t)) \quad (2.2)$$

### 2.1.2 Paramétrage d'une carte de Kohonen

L'organisation d'une carte de Kohonen est gérée par plusieurs paramètres. Nous détaillons ici les choix de paramètres effectués. Les paramètres supplémentaires introduits par la version CxSOM seront présentés [plus tard] en partie 2.5.

#### Taux d'apprentissage $\alpha$

Le taux d'apprentissage  $\alpha$  détermine la proportion dans laquelle chaque poids est déplacé vers l'entrée lors de sa mise à jour, selon l'équation 2.2. Dans l'algorithme classique, le taux d'apprentissage décroît au cours de l'apprentissage. Au début de l'apprentissage,  $\alpha$  est élevé, ce qui assure un déploiement rapide de la carte.  $\alpha$  est ensuite diminué manuellement tout au long de l'apprentissage. Cette décroissance assure la convergence des poids de la carte au cours de l'apprentissage.

Un objectif à long terme de développement de l'architecture CxSOM est de développer des systèmes de cartes autonomes, intégrant le traitement de données séquentielles. Dans ce cas d'utilisation, il n'est pas souhaitable de faire décroître le taux d'apprentissage qui introduit un début et une fin d'apprentissage fixés par avance. Le calcul d'une itération dépend alors non seulement de l'état précédent de la carte, mais aussi de l'itération  $t$  courante. Dans une logique de développement futur, nous utiliserons un taux d'apprentissage constant au cours des itérations dans CxSOM. Les calculs réalisés lors d'une itération  $t$  dépendent alors uniquement de l'état précédent.

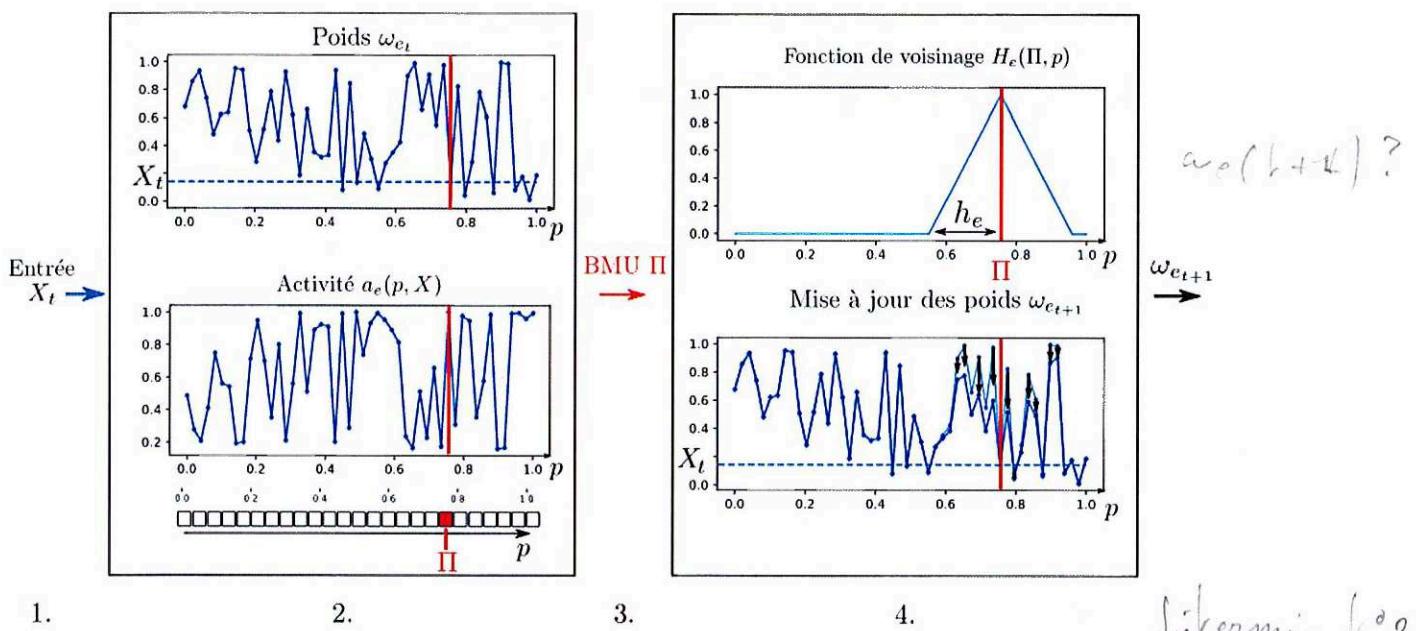


FIGURE 2.2 – Notations utilisées dans une carte de Kohonen simple. Les 4 étapes d'une itération d'apprentissage sont présentées : 1. Présentation de l'entrée, 2. Calcul de l'activité, 3. Choix du BMU, 4. Mise à jour des poids.

### Topologie de la carte

Le graphe supportant la carte de Kohonen peut présenter diverses formes, comme détaillé en section 1.3 : grilles, lignes, arbres, graphes ... Les notations et l'algorithme CxSOM que nous présentons dans ce chapitre sont applicables à toutes les formes de cartes. Les expériences et l'évaluation du modèle se concentrent quant à elles sur des lignes 1D et des grilles 2D, et omettent les formes de graphes quelconques. Ce choix est d'abord motivé par le fait que les lignes et les grilles sont les formats de cartes les plus courants rencontrés dans la littérature. On parle souvent de cartes de Kohonen 1D et cartes 2D, en sous-entendant le format de ligne ou de grille du graphe support. + visualisatio<sup>e</sup> données ?

Ensuite, la spécificité des cartes de Kohonen tient à l'organisation des prototypes de façon continue. Lorsqu'on parle de continuité des prototypes dans une carte de Kohonen, il s'agit d'abord d'une relation de proximité et d'ordre entre des prototypes discrets : *si deux unités sont proches dans la carte, alors leurs prototypes sont proches dans l'espace d'entrée*. Un exemple d'organisation des poids d'une SOM en ligne 1D sur des données dans  $[0, 1]$  est tracé en figure 2.3. Les prototypes sont répartis aléatoirement dans l'espace d'entrée  $[0, 1]$  à l'itération 0 ; au cours de l'apprentissage, ils s'organisent de façon ordonnée. A partir de l'itération 500, on observe cette continuité des prototypes.

Le format particulier de ligne et de grille d'une carte de Kohonen permet d'étendre cette notion de proximité entre prototypes à une continuité des poids au sens mathématique, par interpolation. Dans ces formats 1D et 2D, l'ensemble des noeuds et leurs arêtes est inclus dans une ligne ou un plan : chaque arête peut être vue comme un ensemble de positions. Les poids de la carte sont dans ce cas une approximation discrète d'une fonction continue  $\widetilde{\omega}_e$ , à valeurs

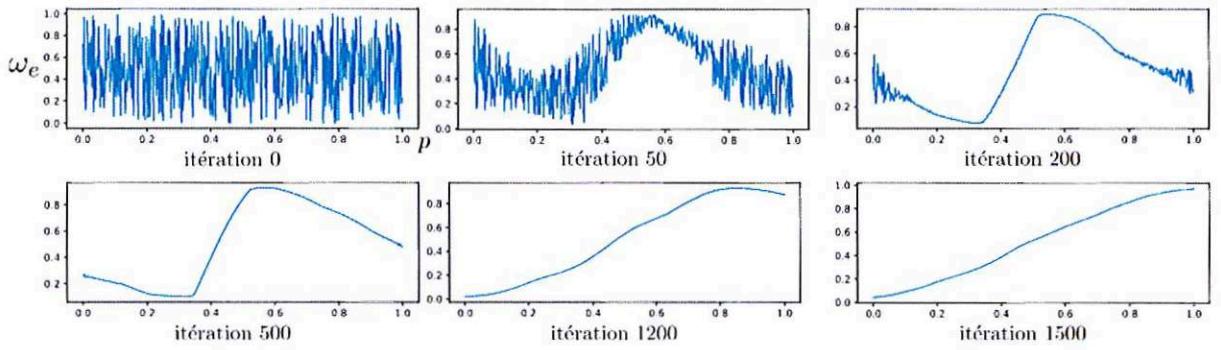


FIGURE 2.3 – Exemple de déploiement d’une carte 1D de taille 500, sur des données 1D  $X \in [0, 1]$ . Les paramètres  $h_e = 0.2$ ,  $\alpha = 0.2$  ont été gardés constants dans cet exemple. On s’attend à ce que les poids de la carte soient organisés selon un ordre strictement croissant ou décroissant à la fin de l’apprentissage.

dans  $\mathcal{D}$ .

$$\begin{array}{ccc} M & : & [0, 1]^2 \text{ ou } [0, 1] \rightarrow \mathcal{D} \\ & & p \mapsto \widetilde{\omega}_e(p) \end{array}$$

Cette continuité est une des puissances d’une carte de Kohonen en tant qu’algorithme de quantification vectorielle. Des opérations réalisées dans l’espace des positions  $[0, 1]$  correspondent directement à des opérations dans l’espace d’entrée  $\mathcal{D}$ , par la fonction  $\widetilde{\omega}_e$ . *Apprendre ?*

Au cours de l’apprentissage, les poids d’une carte se rapprochent de la distribution des données. On parlera de *dépliement* d’une carte pour parler de son apprentissage. Pour une carte 1D sur des données 1D, il est démontré en [17] que les poids évolueront au cours de l’apprentissage vers un ordre strictement croissant ou strictement décroissant ; ordre qui ne sera plus modifié une fois atteint. Lorsque la dimension des données est plus grande que celle de la carte, par exemple des points 2D ou des images (256 dimensions), la carte formera des plis de manière à remplir l’espace  $\mathcal{D}$  (voir figure 1.4, section 1.3).

### Rayon de voisinage

Le choix de la fonction de voisinage est déterminant dans la topologie de la carte. Elle dépend en particulier du rayon de voisinage  $h_e$ . Cette valeur détermine quelles unités voisines du BMU auront leurs poids mis à jour. Plus le rayon  $h_e$  est grand, plus la partie de la carte déplacée vers l’entrée lors de la mise à jour est étendue. Le rayon de voisinage détermine l'*élasticité* d’une carte. Une carte ayant un grand rayon de voisinage est moins sensible aux variations locales des données et parvient à se déplier selon les variations à grande échelle de la distribution des entrées. Un petit rayon d’apprentissage permet au contraire de déplacer les poids concentrés dans une petite région sans affecter toute la carte. Les poids s’ajustent ainsi aux variations locales des entrées. Par contre, choisir un rayon de voisinage petit au début de l’apprentissage empêche la carte de se déplier globalement de façon ordonnée ; au contraire, on verra apparaître des portions distinctes de cartes s’organisant de façon discontinue. Le choix de l’élasticité est donc un compromis entre apprentissage d’une structure globale des entrées et ajustement aux variations locales. Dans l’algorithme classique, ce compromis est trouvé en faisant décroître le rayon de voisinage au cours de l’apprentissage. Un grand rayon de voisinage permet à la carte de se déplier rapidement en apprenant une structure globale des données. Sa décroissance au cours des itérations permet

d'affiner l'apprentissage des données à un niveau plus fin. Contrairement à la plupart des SOM classiques, nous garderons des rayons de voisinage constants dans CxSOM. Tout comme le fait de garder le taux d'apprentissage constant, garder le rayon de voisinage constant est motivé par les objectifs de traitement de données séquentielles.

*Réduire le temps avec l'immersion.*

## 2.2 Motivations du modèle CxSOM

*de telles architectures ?*

A partir du modèle de carte de Kohonen détaillé en section 2.1, nous proposons une version de carte auto-organisatrice servant de bloc de base pour construire des architectures non-hierarchiques de cartes. L'idée de construire une telle architecture est de traiter plusieurs ensembles de données de façon jointe, pour réaliser de la mémoire associative. Nous présentons tout d'abord les choix de développement effectués pour créer le modèle d'architecture.

*hétérogénies ?*

### 2.2.1 Champ d'application : mémoire associative

L'architecture CxSOM a pour but de développer une mémoire *associative* au sein d'une architecture de cartes. On considère donc un ensemble d'espaces  $\mathcal{D}^{(1)}, \dots, \mathcal{D}^{(n)}$ , différentes modalités sur lesquelles on effectue de la quantification vectorielle. Les entrées présentées à une architecture de cartes sont  $(X^{(1)}, \dots, X^{(n)}) \in \mathcal{D}^{(1)} \times \dots \times \mathcal{D}^{(n)}$ . Pour pouvoir développer une mémoire associative, on se place dans des cas où les modalités considérées ne sont pas indépendantes les unes des autres : les distributions marginales  $X^{(i)}$  ne sont pas indépendantes. La tâche de mémoire associative cherche alors à extraire des schémas de dépendance entre modalités. Lorsqu'on tire une entrée pour la présenter à une carte, on tire une entrée jointe  $\mathbf{X} = (X^{(1)}, \dots, X^{(n)})$ , puis chaque composante est présentée à la carte qui lui correspond. Pour respecter l'homogénéité des entrées nécessaires à l'apprentissage d'une carte auto-organisatrice, on normalise les espaces  $\mathcal{D}^{(i)}$  pour que toutes les entrées soient à valeur dans  $[0, 1]^k$ ,  $k$  la dimension de  $\mathcal{D}^{(i)}$ .

Dans les exemples de cette thèse, on tire des entrées jointes en 3 dimensions, donc chaque composante 1D est présentée à une carte. Chaque modalité est la coordonnée sur un des axes du point 3D tiré. Nous utiliserons par exemple, en tant qu'espace dont les modalités sont dépendantes, un ensemble de points sur un cercle en une dimension dans un espace en trois dimensions. Chaque coordonnée  $X, Y, Z$  dépend alors des deux autres coordonnées. On évaluera comment l'architecture que nous présentons dans cette partie apprend les données mais surtout leurs relations.

*figure 2.4 ?*

Les exemples porteront sur des modalités en une dimension, mais les dimensions de chaque modalité peuvent être quelconque.

### 2.2.2 Description de l'architecture

Nous avons vu au chapitre précédent la notion de contexte transmis entre cartes. Dans CxSOM, on choisit de se placer dans le paradigme de transmission de la position du BMU entre cartes : on connecte une carte B à une carte A en donnant la position du BMU de B en entrée à la carte A. Ce paradigme de partage de positions rappelle à la fois le modèle hiérarchique HSOM [21], et les modèles de cartes récurrentes s'appuyant sur SOMSD [13, 12, 9].

Contrairement aux cartes hiérarchiques HSOM dans lesquelles la position du BMU est la seule entrée d'une carte de plus haut niveau, chaque carte de l'architecture peut posséder une entrée

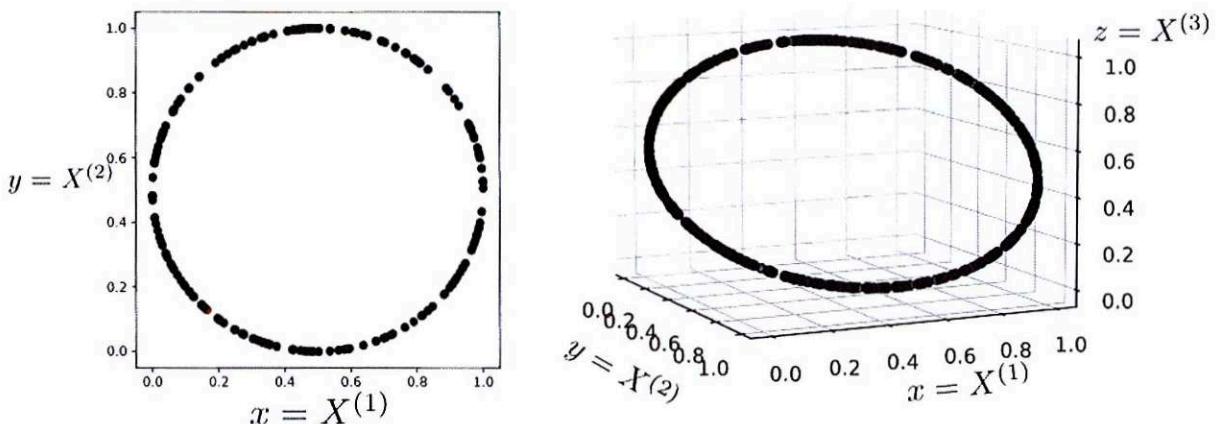


FIGURE 2.4 – Exemple de disposition d’entrées. Les modalités associées à différentes cartes sont les coordonnées  $x, y, z$  de chaque point. Dans une telle disposition, les modalités dépendent les unes des autres : développer une mémoire associative signifie apprendre le modèle de relation existant entre  $x, y$  et  $z$ , c’est à dire le cercle.

principale propre issue d’une modalité  $X^{(i)}$ , l’entrée *externe*. L’entrée ou les entrées correspondant aux positions des BMUs d’autres cartes sont considérées comme une entrée supplémentaire d’une carte. Les cartes auto-organisatrices dans le modèle CxSOM prennent donc un nombre arbitraire d’entrées, dont certaines sont les BMUs d’autres cartes. On appelle ces entrées internes à l’architecture les entrées *contextuelles* d’une carte. L’algorithme d’apprentissage d’une carte auto-organisatrice prenant une position de BMU en tant que contexte est similaire à celui d’une carte classique, comprenant :

1. Présentation de son entrée à la carte
2. Recherche du BMU par calcul d’activité
3. Mise à jour des poids selon une fonction de voisinage

Chaque carte aura maintenant plusieurs entrées : une entrée *externe* dans un espace d’entrée, et  $k$  entrées *contextuelles* qui sont les positions des BMUs des cartes qui lui sont connectées. La carte peut aussi ne pas prendre d’entrée externe, seulement des entrées contextuelles. La recherche du BMU doit être modifiée par rapport à la méthode originale : les rétroactions entre les cartes sont autorisées, la position du BMU de la carte A va donc influencer la position du BMU de la carte B, lequel modifie à nouveau le BMU de la carte A, etc.

Notre algorithme implémentera deux modifications principales par rapport à l’algorithme d’apprentissage d’une carte de Kohonen classique :

- Les cartes possèdent plusieurs entrées, externes et contextuelles ; les entrées contextuelles sont les positions des BMUs d’autres cartes. Le calcul de l’activité est modifié afin de prendre en compte ces différentes couches d’entrées.
- La recherche du BMU est modifiée afin de gérer les rétroactions entre cartes.

L’architecture CxSOM couple ainsi l’apprentissage de plusieurs cartes. Elles apprennent à la fois sur leurs données  $X^{(i)}$ , mais contextualisées selon les informations issues des autres cartes.

- ☒ Notons que les cartes apprennent de façon jointe dès le début de leur apprentissage. Seule la position du BMU est utilisée comme information transmise entre carte. Cette valeur a l’avantage d’apporter une homogénéité dans l’architecture de cartes : quelles que soient les entrées d’une carte et leurs dimensions, le BMU sera une position en 1 ou 2 dimensions. Si on prenait le poids du BMU comme valeur transmise, comme peut le faire la carte récurrente

New fine ?

MSOM [25], l'information circulant entre les cartes dépendrait des dimensions des entrées. De plus, transmettre seulement le BMU est une avantage en terme de quantité d'information à transmettre. Certains modèles, tel que le modèle de carte récurrente [27] transmettent l'activité complète d'une carte en contexte. Certes, cette information reste indépendante du type d'entrée, est complète, mais lourde. La transmission du seul BMU, on le verra dans le chapitre d'analyse, est suffisante pour permettre l'apprentissage du modèle de relations entre données, qui est ce qu'on recherche avec les entrées multimodales.

On laisse aussi la possibilité d'utiliser des cartes ne prenant que des entrées contextuelles. Ces cartes agissent alors comme des cartes intermédiaires, connectant des cartes prenant des entrées externes.

L'algorithme CxSOM est détaillé en algorithme 1 ; les parties suivantes expliquent et illustrent le modèle. Nous présentons d'abord le modèle sur un exemple de deux cartes, puis le formalisme étendu dans un cadre général d'architecture.

### 2.3 Présentation de CxSOM : exemple d'une architecture de deux cartes

Avant de présenter le modèle général de CxSOM sur une architecture quelconque, présentons le fonctionnement de l'architecture la plus simple qui soit : deux cartes  $M^{(1)}$  et  $M^{(2)}$ , connectées réciproquement, présentée en figure 2.5. Toutes les équations seront formalisées dans le cas général en section 2.4. On prend dans cet exemple des cartes en une dimension, indexées par  $p \in [0, 1]$ . Les étapes sont d'abord détaillées, puis résumées en deuxième sous-partie.

#### 2.3.1 Détail des étapes

Structure des cartes La carte  $M^{(1)}$  prend une entrée externe notée  $X^{(1)}$  et une entrée contextuelle qui est le BMU de la carte  $M^{(2)}$ ,  $\Pi^{(2)}$ . Les entrées externes  $X^{(1)}$  et  $X^{(2)}$  sont deux modalités interdépendantes ; dans cet exemple, il s'agit des coordonnées  $x$  et  $y$  d'un point sur un cercle

tel que présenté en figure 3.4.1.0. Les entrées externes sont donc également en une dimension.

La structure de chaque carte de l'architecture est adaptée à partir du modèle classique de SOM pour prendre deux entrées au lieu d'une. On indicera les éléments des cartes par (1) et (2) pour désigner les éléments appartenant à la carte  $M^{(1)}$  et  $M^{(2)}$ . Une carte possède ainsi deux couches

de poids : les poids *externes*  $\omega_e^{(i)}$ ,  $i = 1, 2$  qui se déplient sur les entrées  $X^{((i))}$  et les poids contextuels  $w_c^{(i)}$ , qui se déplient sur l'espace des positions en une dimension de l'autre carte. Ces deux couches de poids sont représentées en figure 2.6. La position du BMU de  $M^{(2)}$ ,  $\Pi^{(2)}$  est utilisée comme entrée contextuelle de  $M^{(1)}$ , et  $\Pi^{(1)}$  comme entrée contextuelle de  $M^{(2)}$ . Les deux cartes apprennent donc de façon couplée.

Calcul d'activité L'activité de la carte  $a_g$  résulte de la combinaison entre l'activité *externe* et l'activité *contextuelle*. Ces deux activités sont calculées comme dans le modèle classique, équation 2.1 et tracées en figure 2.8; l'activité externe compare les poids externes à l'entrée externe  $X_t$ , et l'activité contextuelle les poids contextuels à l'entrée contextuelle.

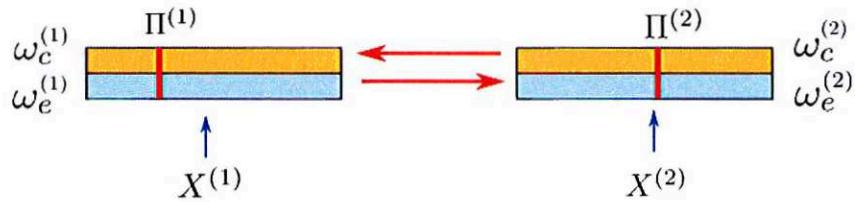


FIGURE 2.5 – Architecture la plus simple possible de deux cartes. Le BMU  $\Pi^{(1)}$  de la carte  $M^{(1)}$  est utilisé en entrée de  $M^{(2)}$ , et le BMU  $\Pi^{(2)}$  de  $M^{(2)}$  en entrée de  $M^{(1)}$ . Chaque carte possède donc deux couches de poids.

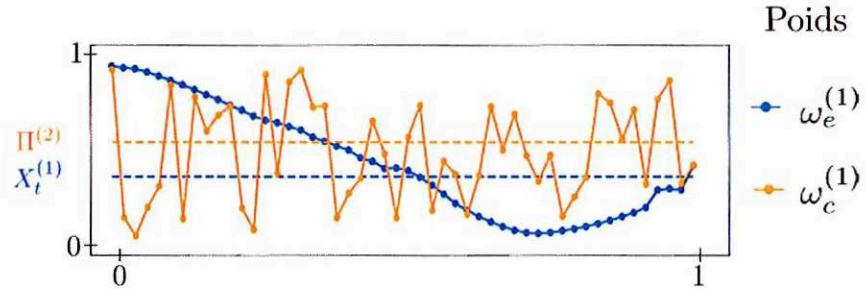


FIGURE 2.6 – Représentation des poids de  $M^{(1)}$ . L'entrée externe  $X_t$  présentée à l'itération  $t$ , tirée d'un espace d'entrée 1D  $[0, 1]$ , est indiquée en bleu sur le graphique. L'entrée contextuelle est le BMU de la carte  $M^{(2)}$ . Sa valeur est indiquée en jaune ; il s'agit d'une position 1D dans la carte  $M^{(2)}$ , à valeur entre 0 et 1. La configuration des poids présentée dans cet exemple est atteinte durant le processus d'apprentissage de deux cartes  $M^{(1)}$  et  $M^{(2)}$ , dont les entrées  $X^{(1)}$  et  $X^{(2)}$  sont les coordonnées de points tirés sur un cercle.

Pour la carte  $M^{(1)}$ , au pas de temps  $t$ , on a :

$$\begin{cases} a_e^{(1)}(X_t^{(1)}, p) = \exp \frac{-\|\omega_e^{(1)}(p) - X_t^{(1)}\|^2}{2\sigma^2} \\ a_c^{(1)}(\Pi^{(2)}, p) = \exp \frac{-\|\omega_c^{(1)}(p) - \Pi^{(2)}\|^2}{2\sigma^2} \end{cases} \quad (2.3)$$

$a_c$  et  $a_e$  sont combinées en une activité globale :

$$a_g^{(1)}(X_t^{(1)}, \Pi^{(1)}, \Pi^{(2)}, p) = \sqrt{a_e(X_t, p) \times \frac{a_e(X_t, p) + a_c(\Pi^{(2)}, p)}{2}} \quad (2.4)$$

Par la différence de contribution de  $a_c$  et  $a_e$  au sein de l'activité globale –  $a_c$  ne contribue qu'à la puissance  $\frac{1}{2}$  – on assure que l'activité contextuelle vient seulement moduler l'activité externe. On peut observer cette modulation sur la courbe noire de la figure 2.8 : l'activité globale suit la même progression que l'activité externe, mais est localement modifiée par les variations de l'activité contextuelle. De cette façon, les entrées contextuelles ne viennent pas donner d'hallucinations à la carte : elle apprend en priorité ses entrées externes, conditionnées aux entrées contextuelles. Cette façon d'assembler les entrées est inspirée des travaux conduits en [23]. Dans ces travaux, les auteurs combinent des activités par leur moyenne géométrique. Le sens de la moyenne géométrique est une sorte de "et" entre les activités, ce qu'on recherche ici aussi. temp

**Relaxation** Le calcul de  $\Pi^{(1)}$  dépend donc de  $\Pi^{(2)}$  et inversement. Contrairement à une carte simple, on ne peut pas calculer tous les BMUs de l'architecture un par un en prenant  $\hat{p}$ , l'argmax

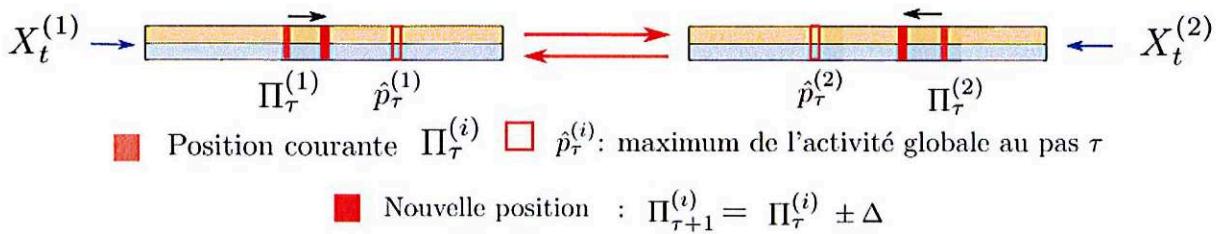


FIGURE 2.7 – description d'une étape de la relaxation dans l'architecture, aboutissant à un consensus entre cartes. Au sein d'une même itération  $t$ , les position des BMU  $\Pi$  sont légèrement déplacées jusqu'à ce que toutes les positions  $\Pi$  des cartes de l'architecture soient stables. Ces positions maximisent collectivement les activités globales de chaque carte.

de  $a_g$ , comme BMU dans chaque carte. On remplace l'étape de simple calcul d'argmax par un processus global à l'architecture de recherche de BMU. Cette recherche est réalisée par un processus dynamique que l'on appellera *relaxation*, menant à un *consensus* entre cartes : on cherche un point, s'il en existe, où le BMU de chaque carte est au plus proche du maximum de son activité globale  $\hat{p}$ .

Cette recherche est une boucle imbriquée dans un pas d'apprentissage  $t$ , indexée par  $\tau$ , et appelée *relaxation*. On définit une suite de BMUs intermédiaires,  $(\Pi_\tau^{(1)}, \Pi_\tau^{(2)})$ ,  $\tau$  variant de 0 à un nombre d'itérations maximum  $\tau_{max}$  fixé pour assurer une fin de la boucle. Le processus de relaxation est le suivant :

1. Les entrées externes sont présentées au début de la boucle, donc  $a_e$  peut être calculée ;  $\Pi_0^{(1)}$  et  $\Pi_0^{(2)}$  sont initialisées à la position où les activités externes sont maximales dans chaque carte.
2. Tant que la suite de positions  $(\Pi_\tau^{(1)}, \Pi_\tau^{(2)})$  n'a pas convergé :
  - (a) Dans chaque carte, calculer les activités contextuelles et globales, définissant ainsi  $\hat{p}_\tau^{(1)} = \arg \max_{p^{(1)}} (a_g^{(1)}(p^{(1)}, X^{(1)}, \Pi^{(2)}))$ , de même pour  $\hat{p}^{(2)}$ .
  - (b) Déplacer  $\Pi^{(1)}$  vers  $\hat{p}^{(1)}$  et  $\Pi^{(2)}$  vers  $\hat{p}^{(2)}$  d'un pas  $\Delta$  :  $\Pi_{\tau+1}^{(1)} = \Pi_\tau^{(1)} \pm \Delta$ . Si une des valeurs est plus proche de  $\hat{p}$  que  $\Delta$ , on déplacera  $\Pi$  directement sur  $\hat{p}$  pour éviter les oscillations autour du point. Cette étape est illustrée en figure 2.7.  
→ c'est bizarre % ω(b) est X<sup>2</sup>, non ?
3. Le BMU de chaque carte est pris comme la valeur finale stable de ce processus dynamique. On note cette position finale  $\Pi_t^{(i)}$ , désignant que cette valeur correspond à l'itération d'apprentissage  $t$ . Cette valeur est utilisée pour les mise à jour des poids. Si la relaxation n'atteint pas de point stable, on fixe tout de même un nombre d'itérations maximum après lequel on arrête la relaxation.

**Mise à jour** Enfin, chaque couche de poids  $\omega_e^{(i)}$ ,  $\omega_c^{(i)}$  est mise à jour indépendamment dans chaque carte relativement au BMU  $\Pi_t^{(i)}$  et les entrées externes  $X_t^{(i)}$  et contextuelles  $\Pi_t^{(j)}$ . Cette mise à jour correspond à la figure 2.9. Les rayons d'apprentissage diffèrent entre couche externe et couche contextuelle.

↑  
 A mordre que ?

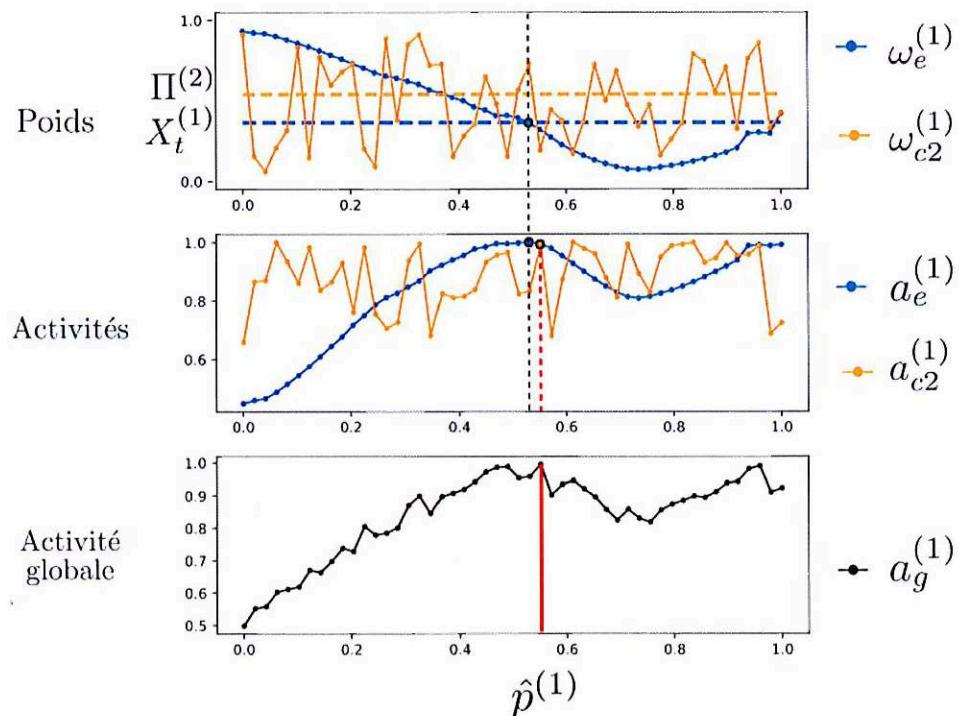


FIGURE 2.8 – Calcul d'activité dans une SOM au sein d'une architecture de deux cartes. La carte prend une entrée externe et une entrées contextuelle. L'indice (1) permet de distinguer les objets relatifs à cette carte. L'entrée externe est  $X_t^{(1)}$ . La carte possède deux couches de poids, permettant de calculer deux activités. L'activité globale prend en compte tout les couches d'activités afin de trouver un BMU commun pour toutes les couches de poids. Ce calcul favorise l'activité externe et est modulé par l'activité contextuelle, ce qu'on observe sur la courbe du bas. Le maximum de l'activité globale est noté  $\hat{p}$ . A partir de l'activité globale, le BMU  $\Pi_t^{(1)}$  sera trouvé par le processus de relaxation décrit en partie 2.4.2

*+ ouvrir que le module ne que module ?*

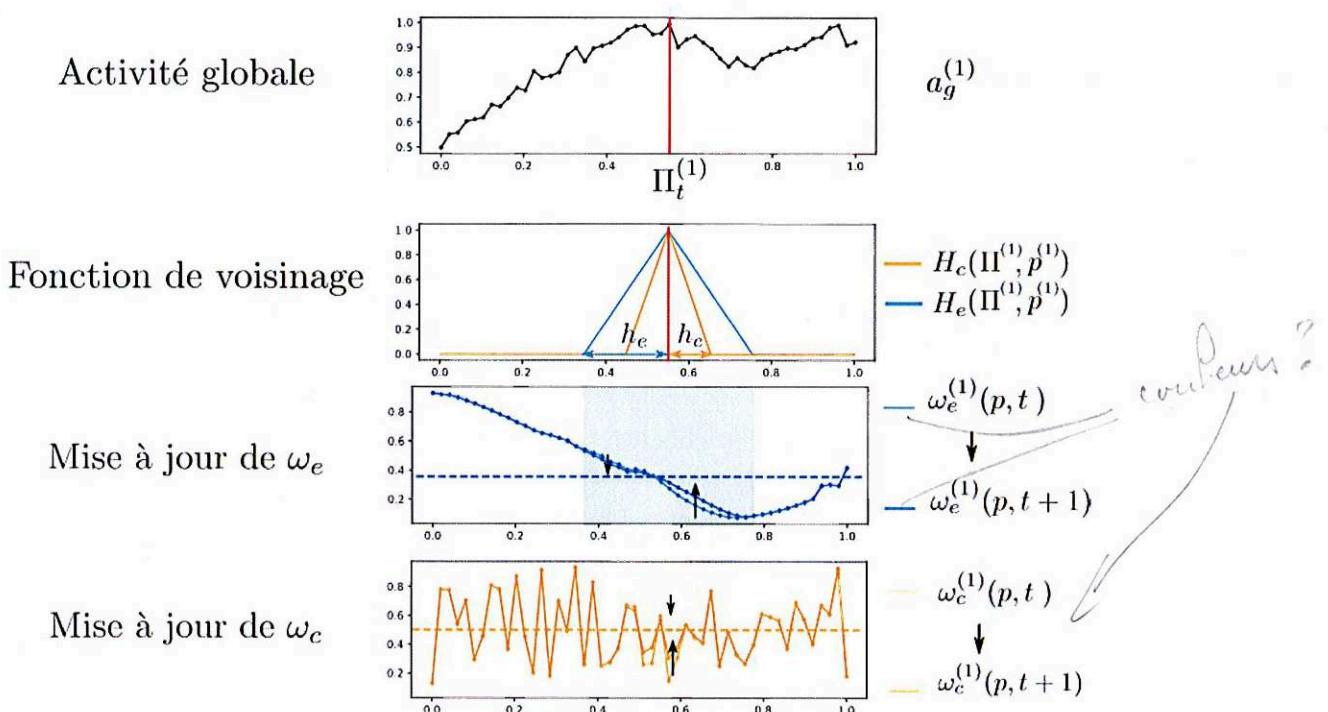


FIGURE 2.9 – Mise à jour de chaque couche de poids indépendamment, relativement au BMU commun  $\Pi_t^{(1)}$ , calculé par relaxation. Par définition de la relaxation, la position  $\Pi_t^{(1)}$  est proche de la position  $\hat{p}^{(1)}$  à la fin de la relaxation. Le rayon de voisinage  $h_e$  est utilisé pour mettre à jour les poids externes, le rayon  $h_c$  pour mettre à jour les poids contextuels. On choisit  $h_e > h_c$ . Cette différence permet une différence d'échelle d'apprentissage entre couches de poids. Elle est détaillée en section suivante.

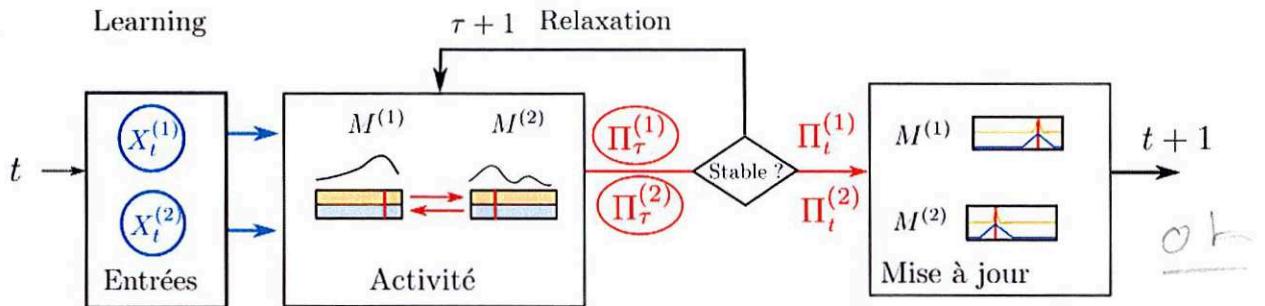


FIGURE 2.10 – Résumé des étapes de l'algorithme d'apprentissage d'une architecture.

### 2.3.2 Résumé

Les étapes d'un pas d'apprentissage  $t$  d'une architecture de deux cartes sont les suivantes ; elles sont schématisées en figure 2.10.

1. Présentation des entrées  $X_t^{(1)}$  et  $X_t^{(2)}$  à chaque carte
2. Relaxation :
  - (a) Calcul de l'activité externe  $a_e(X^{(i)}, p)$  dans chaque carte et initialisation des BMUs ( $\Pi_0^{(1)}, \Pi_0^{(2)}$ ) pour la relaxation.
  - (b) Relaxation par petits déplacements de  $\Pi_\tau^{(1)}, \Pi_\tau^{(2)}$  dans chaque carte, avec calcul de l'activité contextuelle et globale à chaque pas  $\tau$ , jusqu'à stabilisation selon  $\tau$  du couple de valeurs  $(\Pi_\tau^{(1)}, \Pi_\tau^{(2)})$
  - (c) Choix des positions de BMU  $\Pi_t^{(1)}, \Pi_t^{(2)}$  comme la valeur de  $(\Pi_\tau^{(1)}, \Pi_\tau^{(2)})$  à l'issue de la relaxation
3. Mise à jour des poids  $\omega_e^{(i)}$  et  $\omega_c^{(i)}$  dans chaque carte, selon la position du BMU  $\Pi_t^{(i)}$ , les entrées externes  $X_t^{(i)}$  et contextuelles  $\Pi_t^{(j)}$ , la position du BMU calculée par relaxation dans l'autre carte.

## 2.4 Formalisation : cas pour $n$ cartes

Nous présentons dans cette partie un formalisme pour l'algorithme général pour une architecture quelconque de  $n$  cartes. Les notations sont valables pour des cartes de dimension quelconque ; les entrées peuvent être de dimension quelconque. La différence principale avec l'exemple à deux cartes est qu'une carte peut prendre plusieurs entrées contextuelles, qui sont les BMUs de toutes les cartes qui lui sont connectées dans l'architecture, au lieu d'une dans le cas de deux cartes. On retrouvera donc les notations de la partie précédente. Cette partie concentre toutes les notations et l'algorithme utilisé dans cette thèse. L'algorithme est résumé en algorithme 1.

### 2.4.1 Entrées et calcul d'activité

Nous présentons d'abord la structure et les notations utilisées dans une carte ; toutes ces notations sont retrouvables en figure 2.8, pour l'exemple d'architecture de 2 cartes. Dans une architecture composée de  $n$  cartes, les cartes sont indexées par  $i \in \{1, \dots, n\}$ . On indicera chaque élément d'une carte  $M^{(i)}$  par l'exposant  $(i)$ . Pour faciliter la lecture, nous omettrons par abus de

langage l'exposant ( $i$ ) dans les équations, lorsqu'on se concentre sur une seule carte.  $X_t$  désigne donc  $X_t^{(i)}$ ,  $\omega_e$  désigne  $\omega_e^{(i)}$ , etc.

A un pas d'apprentissage  $t$ , une carte  $M^{(i)}$  reçoit en entrée une entrée *externe* notée  $X_t$  et  $K$  entrées *contextuelles*. Notons-les pour le moment  $\Gamma = (\gamma_{i_1}, \dots, \gamma_{i_K})$ ; elles seront les positions de BMU  $\Pi^{(i_k)}$  des cartes d'indice  $i_k$  qui lui sont connectées. La gestion des entrées contextuelles sera décrite avec le processus de relaxation en section suivante; notons pour le moment que les entrées contextuelles sont des positions 1D ou 2D dans des cartes. *cf. 5?*

La carte possède donc  $K+1$  couches de poids. On note  $\omega_e(p)$  les poids externes et  $\omega_{ci_1}(p), \dots, \omega_{ci_K}(p)$  les poids correspondant aux entrées contextuelles, *les poids contextuels*.  $\omega_{ci_k}$  correspond à la couche de poids relative à l'entrée contextuelle  $\gamma_{i_k} = \Pi^{(i_k)}$ . Les poids externes sont à valeur dans  $\mathcal{D}^{(i)}$ , la modalité associée à la carte  $i$ . Les poids contextuels sont à valeur dans l'espace des positions d'une carte, soit  $[0, 1]$  en 1D ou  $[0, 1]^2$  en 2D.

Les activités externes et contextuelles s'expriment de la façon suivante :

$$\begin{cases} a_e(X_t, p) = \exp \frac{-\|\omega_e(p) - X_t\|^2}{2\sigma^2} \\ a_{ci_k}(\gamma_{i_k}, p) = \exp \frac{-\|\omega_{ci_k}(p) - \gamma_{i_k}\|^2}{2\sigma^2}, \end{cases} \quad (2.5)$$

*$i_1, \dots, i_K$  indices des cartes connectées à  $i$  dans l'architecture*

Notons  $a_c(\Gamma, p)$  la moyenne des activités contextuelles, avec  $\Gamma = (\gamma_{i_1}, \dots, \gamma_{i_K})$ .

$$a_c(\Gamma, p) = \frac{1}{K} \sum_{k=1}^K a_{ci_k}(\gamma_{i_k}, p) \quad (2.6)$$

L'activité globale  $a_g$  est calculée en combinant l'activité externe et la moyenne des activités contextuelles :

$$a_g(X_t, \Gamma, p) = \sqrt{a_e(X_t, p) + a_c(\Gamma, p)} \quad (2.7)$$

On note  $\hat{p}$  la position du maximum de l'activité globale :

$$\hat{p} = \arg \max_p a_g(X_t, \Gamma, p) \quad (2.8)$$

Notons qu'une carte peut ne pas avoir d'entrée externe : toutes ses entrées sont des positions des BMUs d'autre cartes. Dans ce cas, on prendra comme activité globale  $a_c$ , la moyenne des activités contextuelles (équation 2.6). *pourquoi pas  $\sqrt{a_c}$  ?*

## 2.4.2 Calcul du BMU par relaxation

Dans chaque carte  $i$ , l'entrée contextuelle  $\gamma_{i_k}^{(i)}$  est le BMU  $\Pi^{(i_k)}$  de la carte  $i_k$ . Le BMU  $\Pi^{(i)}$  dépend donc de ceux des autres cartes, qui dépendent eux-mêmes de  $\Pi^{(i)}$ . Au lieu de chercher un maximum d'activité indépendamment dans chaque carte, on doit donc résoudre un problème d'optimisation global à l'architecture. On cherche les positions  $\Pi = (\Pi^{(1)}, \dots, \Pi^{(n)})$ , si elles existent, telles que dans chaque carte,  $\Pi^{(i)}$  soit la position du maximum de l'activité globale, c'est-à-dire  $\hat{p}^{(i)}$ .

$$\forall i, \Pi^{(i)} = \arg \max_{p^{(i)}} a_g^{(i)}(X_t^{(i)}, \Pi^{(i_1)}, \dots, \Pi^{(i_K)}, p^{(i)}) \quad (2.9)$$

*place binnaire de position ?*

$\Pi^{(i_1)}, \dots, \Pi^{(i_K)}$  est un sous ensemble de  $\Pi$ . On peut donc écrire

$$\Pi^{(i)} = \arg \max_{p^{(i)}} a_g^{(i)}(X_t^{(i)}, \Pi, p^{(i)})$$

Si cette position n'existe pas, on veut chercher le meilleur compromis, c'est à dire les positions  $\Pi^{(i)}$  telles que l'activité globale de chaque carte soit la plus élevée possible.

Pour formuler le problème en terme d'optimisation, on cherche le vecteur  $\Pi = (\Pi^{(i)})_{i=1 \dots n}$  minimisant, pour tout  $i$ ,  $\|\Pi^{(i)} - \arg \max_{p^{(i)}} a_g^{(i)}(X_t^{(i)}, \Pi, p^{(i)})\|$

Le processus de relaxation est une boucle imbriquée dans un pas d'apprentissage de l'architecture, indexée par  $\tau$ . Il s'agit d'une heuristique cherchant à aller vers cette position minimum. Notons  $\Pi^{(i)}$  la position du BMU de la carte  $i$ , et  $\Pi = (\Pi^{(0)}, \dots, \Pi^{(n)})$ , avec  $n$  le nombre de cartes de l'architecture.

On cherche, dans chaque carte  $i$ , la position  $\Pi^{(i)}$  telle que  $\forall i, a_g^{(i)}(\Pi^{(i)}, X^{(i)}, \Pi^{(i_0)}, \dots, \Pi^{(i_k)})$  soit maximale. Au début d'un pas d'apprentissage, chaque carte est nourrie avec une entrée externe  $X_t^{(i)}$  qui restera constante au cours de la relaxation. Les activités externes  $a_e^{(i)}(X_t^{(i)}, p)$  de chaque carte peuvent être calculées. La recherche du BMU suit le processus de relaxation suivant :

1. Dans chaque carte  $i$ , la position  $\Pi^{(i)}$  est initialisée à  $\hat{p}_0^{(i)} = \arg \max_{p^{(i)}} a_e^{(i)}(X_t^{(i)}, p)$ . Dans chaque carte  $i$ , on assigne  $\gamma_{i_k}^{(i)} = \Pi_\tau^{(i)}$
2. Tant que toutes les positions  $\Pi^{(i)}$  ne sont pas stables,
  - (a) Dans chaque carte  $i$ , calculer les activités contextuelles et globales, définissant ainsi  $\hat{p}_\tau^{(i)} = \arg \max_{p^{(i)}} a_g^{(i)}(p^{(i)}, X^{(i)}, \Pi_\tau^{(i_0)}, \dots, \Pi_\tau^{(i_k)})$ , avec  $i_0, \dots, i_k$  les indices des cartes connectées à  $i$  dans l'architecture.
  - (b) Déplacer  $\Pi^{(i)}$  vers  $\hat{p}^{(i)}$  :  $\Pi_{\tau+1}^{(i)} \leftarrow \Pi_\tau^{(i)} + \Delta \times \text{sgn}(\hat{p}^{(i)} - \Pi^{(i)})$  si  $|\Pi^{(i)} - \hat{p}^{(i)}| \geq \Delta$ ,  $\Pi^{(i)} \leftarrow \hat{p}^{(i)}$  sinon
3. Le BMU de chaque carte est pris comme la valeur finale stable de ce processus dynamique. Cette valeur est utilisée pour les mises à jour des poids.

Il peut arriver que les positions ne trouvent pas de point de stabilité. Dans ce cas, on arrêtera la relaxation arbitrairement ; ce phénomène étant ponctuel, il n'influence pas l'apprentissage. Les paramètres des cartes de l'architecture sont choisis pour éviter de telles situations.

Expérimentalement, nous observons que ce processus permet de trouver des positions pour lesquelles l'activité globale de chaque carte est proche de son maximum. Nous observons également que ces positions sont un point de convergence du processus de relaxation. Nous détaillerons ces expériences dans le chapitre 4.

### 2.4.3 Mise à jour des poids

Les poids sont mis à jour par rapport à leurs entrées respectives suivant l'équation 2.2. Le BMU d'une carte est ainsi commun à toutes les couches. Les rayons de voisinage  $h_e$  et  $h_c$  ont des valeurs différentes. Ainsi, la mise à jour des poids d'une carte est indépendante à chaque couche, avec des paramètres propres, ayant simplement le BMU en commun.

$$\omega_e^{(i)}(p, t+1) = \omega_e^{(i)}(p, t) + H_e(\Pi^{(i)}, p)(\omega_e^{(i)}(p) - X_t^{(i)}) \quad (2.10)$$

$$\forall k, \omega_{ci_k}^{(i)}(p, t+1) = \omega_{ci_k}^{(i)}(p) + H_c(\Pi^{(i)}, p)(\omega_{ci_k}^{(i)}(p) - \Pi_t^{(i)}) \quad (2.11)$$

**Algorithme 1 :** Déroulement d'une itération d'apprentissage  $t$ 


---

**Données :**  $X_t^{(1)}, \dots, X_t^{(K)}$  tirés dans  $\mathcal{D}^{(1)} \times \dots \times \mathcal{D}^{(n)}$

$$\tau \leftarrow 0$$

**pour chaque carte  $i$  faire**  $\Pi_0^{(i)} \leftarrow \arg \max_{p^{(i)}} a_e(X_t^{(i)}, p^{(i)})$ ;

**tant que**  $\Pi_\tau \neq \Pi_{\tau-1}$  et  $\tau < \tau_{max}$  **faire**

**pour chaque carte  $i$  faire**

Avec  $i_1, \dots, i_K$  indices des cartes connectées à  $i$  dans l'architecture : Calcul de  $a_{ci_1}^{(i)}(\Pi^{(i_1)}, p^{(i)}), \dots, a_{ci_K}^{(i)}(\Pi^{(i_K)}, p^{(i)})$

Calcul de  $a_g^{(i)}(X^{(i)}, \Pi_\tau^{(i_0)}, \dots, \Pi_\tau^{(i_K)})$  (équation 2.7)

$\hat{p}_\tau^{(i)} = \arg \max_{p^{(i)}} a_g^{(i)}(X^{(i)}, \Pi_\tau^{(i_1)}, \dots, \Pi_\tau^{(i_K)})$

Déplacement de  $\Pi_\tau^{(i)}$  vers  $\hat{p}_\tau^{(i)}$  d'un pas  $\Delta$  :

$\Pi_{\tau+1}^{(i)} \leftarrow \Pi_\tau^{(i)} + \min(\Delta, |\hat{p}_\tau^{(i)} - \Pi_\tau^{(i)}|) \times \text{sgn}(\hat{p}_\tau^{(i)} - \Pi_\tau^{(i)})$

**fin**

$\tau \leftarrow \tau + 1$

**fin**

$\Pi_t^{(1)}, \dots, \Pi_t^{(n)} \leftarrow \hat{p}_\tau^{(1)}, \dots, \hat{p}_\tau^{(n)}$

**pour chaque Carte  $i$  faire**

$\omega_e^{(i)}(p) \leftarrow \omega_e^{(i)}(p) + H_e(\Pi_t^{(i)}, p)(\omega_e^{(i)}(p) - X_t^{(i)})$

**pour chaque  $k$  faire**  $\omega_{ci_k}^{(i)}(p) \leftarrow \omega_{ci_k}^{(i)}(p) + H_c(\Pi_t^{(i)}, p)(\omega_{ci_k}^{(i)}(p) - \Pi_t^{(i_k)})$ ;

**fin**

---

## 2.5 Choix des paramètres

Le modèle CxSOM introduit des paramètres supplémentaires par rapport à une carte classique. Les plages de valeurs utilisées pour tous les paramètres d'une architecture sont résumées en tableau 2.1

### 2.5.1 Paramétrage d'une carte

On retrouve les mêmes paramètres dans CxSOM que sur une carte classique : taille de la carte, topologie et dimensions. Contrairement à une carte simple, on a maintenant un jeu de paramètre d'apprentissage par couche de poids d'une carte : pour chaque couche de poids  $\omega_e$  et  $\omega_{ci_k}$ , on peut faire varier le taux d'apprentissage  $\alpha$  et le rayon de voisinage  $h_e$  ou  $h_{ci_k}$ . Le taux d'apprentissage  $\alpha$  est commun à toutes les couches dans un souci de simplicité.

On choisit de prendre une valeur  $h_{ci_k} = h_c$  commune à toutes les couches de poids contextuels d'une carte. Ainsi, on apporte une symétrie dans les connexions : les cartes réagissent de la même façon aux autres cartes. Le rayon externe, par contre, est choisi  $h_e$  très supérieur au rayon contextuel. Nous prendrons au moins  $h_e = 10h_c$ . Cette différentiation de paramètres apporte deux elasticités dans l'apprentissage, et induit deux vitesses de dépliement dans la carte sans avoir à modifier les paramètres au cours de l'apprentissage. Les poids externes se déplient alors très rapidement sur les données, quand les poids contextuels se déplacent très peu au début. Lorsque les poids externes sont organisés, l'apprentissage n'influence plus que les poids contextuels et ces derniers se déplient. Nous analyserons plus en détail l'organisation des cartes dans le chapitre suivant.  $\alpha$  et  $h_e, h_c$  resteront constants au cours de l'apprentissage. Ce jeu de paramètres est

TABLE 2.1 – Tableau récapitulatif des paramètres ayant une influence sur le comportement de l'algorithme CxSOM. Tous les paramètres relativ à une carte sont les mêmes pour chacune des cartes de l'architecture, mais il est possible de les différencier. L'analyse de l'influence des paramètres sera détaillée au chapitre 4.

Paramètres	Description	Plage de valeur
$\alpha$	Taux d'apprentissage	0.1
$N$	Taille de la carte	de 500 à 1000 en 1D, $50 \times 50$ en 2D
$h_e$	Rayon de voisinage externe	autour de 0.2
$h_c$	Rayon de voisinage contextuel	D'ordre $\frac{h_e}{10}$ ou inférieur
$\Delta$	Pas de relaxation	0.1
$\tau_{max}$	Nombre de pas de relaxation maximum	200

ajustable indépendamment dans chaque carte de l'architecture ; dans nos travaux, on prendra en général les mêmes valeurs pour toutes les cartes d'une architecture.

### 2.5.2 Paramètres de l'architecture

Certains paramètres sont relatifs à l'architecture. Il s'agit d'abord de  $\Delta$ , le pas d'apprentissage. Nous avons pris la même valeur de pas pour toutes les cartes. Cette valeur sera en général d'ordre 0.1, c'est à dire un déplacement de 10% de la taille de la carte, dans les expériences présentées dans les chapitres suivants. Nous verrons que la valeur de ce paramètre a finalement peu d'influence sur la relaxation ; il faut juste veiller à ne pas le prendre trop petit, pour ne pas augmenter les temps de relaxation. Le deuxième paramètre relatif à la relaxation est  $\tau_{max}$ , nombre maximum de pas de relaxation. Il sera fixé à 200 dans la plupart de nos expériences ; nous verrons que la relaxation, si elle converge, se réalise en une dizaine de pas. Les connexions entre cartes sont prédéfinies et fixes. On choisit au préalable la taille de l'architecture et les connexions entre cartes.

## 2.6 Conclusion

le modèle CxSOM permet de construire ~~des~~ architectures de carte auto-organisatrices apprenant chacune sur des entrées de différents types, mais liées par un modèle : des entrées multimodales. L'apprentissage est couplé entre cartes par l'utilisation d'entrées contextuelles dans chaque carte, qui sont les positions des BMU qui lui sont connectées. Nous avons décrit dans ce chapitre l'algorithme permettant d'effectuer l'apprentissage de données dans une telle architecture ainsi que les notations associées.

L'algorithme d'apprentissage a été conçu de façon large : nous avons développé un modèle général permettant d'associer des cartes. Les objectifs des chapitres suivants sont d'abord de montrer le comportement de l'algorithme et comment la relaxation permet de déterminer un BMU dans chaque carte. Nous étudierons ensuite en détail comment les données fournies en entrées à une architecture sont représentées dans les couches de poids. Nous verrons enfin une application d'une architecture de cartes dans un contexte de prédiction d'entrée.