

# THÈSE

Présentée par

**Pierre DEMARTINES**

Pour obtenir le titre de

**DOCTEUR**

**DE L'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE**

(arrêté ministériel du 30 mars 1992)

Spécialité : **Signal, Image, Parole**

---

## **ANALYSE DE DONNÉES PAR RÉSEAUX DE NEURONES AUTO-ORGANISÉS**

---

Soutenue le 25 Novembre 1994 devant le jury :

Président	Prof. Jean-Marc DOLMAZON
Rapporteurs	Prof. Marie COTTRELL Prof. Christian WELLEKENS
Examineurs	Prof. Alain GERMOND Dr. Vincent LORQUET
Directeur	Prof. Jeanny HÉRAULT

Thèse préparée au sein du laboratoire TIRF  
Laboratoire de Traitement d'Images et de Reconnaissance des Formes

# DATA ANALYSIS THROUGH SELF-ORGANIZED NEURAL NETWORKS

## Keywords

Data structure (submanifold), Self-Organizing Maps (Kohonen), Fractal Dimension, Dimension Reduction, Nonlinear Projection, Unfolding, “VQP” Algorithm, Diffeomorphism, Interpolation, Extrapolation, Multidimensional Scaling, Nonlinear Mapping, Industrial Applications.

---

## Abstract

Data *understanding* is often based on hidden informations retrieval within a big amount of collected variables. It is a search for linear or non linear *dependencies* between these observed variables, and consists in reducing these variables to small number of parameters.

A classical method, widely used for this purpose, is the so-called Principal Component Analysis (PCA). Unfortunately, this method is only linear, and fails to reduce data that are redundant in a non linear way.

The Kohonen’s Self-Organizing Maps are a type of artificial neural networks, the functionality of which can be viewed as a non linear extension of PCA: data samples are mapped onto a grid of neurons. A major drawback of these maps, however, is their *a priori* defined shape (generally a square or a rectangle), which is rarely adapted to the shape of the parametric space to represent.

We relax this constraint with a new algorithm, called “Vector Quantization and Projection” (VQP). It is a kind of self-organizing map, the output space of which is continuous and takes automatically the relevant shape. From a mathematical point of view, VQP is the search for a diffeomorphism between the raw data set and an unknown parametric representation to be found. More intuitively, this is an *unfolding* of data structure towards a low-dimensional space, which dimension is the number of degrees of freedom of the observed phenomenon, and can be determined through fractal analysis of the data set.

In order to illustrate the generality of VQP, we give a wide range of application examples (real or simulated), in several domains such as data fusion, graphes matching, industrial process monitoring or analysis, faults detection in devices and adaptive routing in telecommunications.

# ANALYSE DE DONNÉES PAR RÉSEAUX DE NEURONES AUTO-ORGANISÉS

## Mots-clés

Structure de données (variété), cartes auto-organisantes (Kohonen), dimension fractale, réduction de dimension, projection non-linéaire, dépliage, algorithme “VQP”, difféomorphisme, interpolation, extrapolation, “*Multidimensional Scaling*”, “*Nonlinear Mapping*”, applications industrielles.

---

## Résumé

Chercher à *comprendre* des données, c’est souvent chercher à trouver de l’information cachée dans un gros volume de mesures redondantes. C’est chercher des dépendances, linéaires ou non, entre les variables observées pour pouvoir résumer ces dernières par un petit nombre de paramètres.

Une méthode classique, l’Analyse en Composantes Principales (ACP), est abondamment employée dans ce but. Malheureusement, il s’agit d’une méthode exclusivement linéaire, qui est donc incapable de révéler les dépendances non linéaires entre les variables.

Les cartes auto-organisantes de Kohonen sont des réseaux de neurones artificiels dont la fonction peut être vue comme une extension de l’ACP aux cas non-linéaires. L’espace paramétrique est représenté par une grille de neurones, dont la forme, généralement carrée ou rectangulaire, doit malheureusement être choisie *a priori*. Cette forme est souvent inadaptée à celle de l’espace paramétrique recherché.

Nous libérons cette contrainte avec un nouvel algorithme, nommé “*Vector Quantization and Projection*” (VQP), qui est une sorte de carte auto-organisante dont l’espace de sortie est continu et prend automatiquement la forme adéquate. Sur le plan mathématique, VQP peut être défini comme la recherche d’un difféomorphisme entre l’espace brut des données et un espace paramétrique inconnu à trouver. Plus intuitivement, il s’agit d’un *dépliage* de la structure des données vers un espace de plus petite dimension. Cette dimension, qui correspond au nombre de degrés de liberté du phénomène étudié, peut être déterminée par des méthodes d’analyse fractale du nuage de données.

Afin d’illustrer la généralité de l’approche VQP, nous donnons une série d’exemples d’applications, simulées ou réelles, dans des domaines variés qui vont de la fusion de données à l’appariement de graphes, en passant par l’analyse ou la surveillance de procédés industriels, la détection de défauts dans des machines et le routage adaptatif en télécommunications.

*A mes parents  
et à mon frère*

# Table des matières

Remerciements	7
Préambule	13
<b>1 La recherche de structure</b>	<b>19</b>
1.1 Un exemple . . . . .	20
<b>2 Distributions en grandes dimensions</b>	<b>23</b>
2.1 Norme de vecteurs aléatoires . . . . .	23
2.2 Espace vide . . . . .	27
2.3 Visualisation par rotation dynamique . . . . .	30
2.4 Analyse en composantes principales . . . . .	33
2.4.1 Méthode . . . . .	33
2.4.2 Equivalents neuronaux . . . . .	35
2.4.3 Exemples . . . . .	36
2.4.4 Limitations . . . . .	38
2.5 Dimension intrinsèque . . . . .	40
2.5.1 Notion de variété . . . . .	40
2.5.2 Dimension fractale . . . . .	41
2.5.3 La dimension fractale en pratique . . . . .	44
2.6 Résumé des problèmes . . . . .	48
<b>3 Quantification vectorielle</b>	<b>49</b>
3.1 Généralités . . . . .	49
3.2 Algorithme de Lloyd généralisé (GLA) . . . . .	51
3.3 Apprentissage compétitif (CL) . . . . .	52
3.4 Mécanismes de fatigue ou “conscience” . . . . .	53
3.5 Méthodes “ <i>winner-take-most</i> ” . . . . .	55
3.5.1 Relaxation stochastique (SRS) . . . . .	56
3.5.2 “ <i>Soft Competition Scheme</i> ” (SCS) . . . . .	58
3.5.3 Cartes de Kohonen . . . . .	60

3.5.4	“ <i>Neural Gas</i> ” . . . . .	61
3.6	Quantification par arbre . . . . .	61
3.7	Apprentissage compétitif avec régularisation (CLR) . . . . .	63
<b>4</b>	<b>Cartes auto-organisées</b>	<b>69</b>
4.1	Introduction . . . . .	69
4.2	Algorithme de Kohonen . . . . .	72
4.3	Voisinage gaussien . . . . .	76
4.4	Projection non-linéaire . . . . .	79
4.5	Représentations et mesures de qualité . . . . .	81
4.5.1	Représentation curviligne . . . . .	81
4.5.2	Mesure de désordre $\Theta$ . . . . .	84
4.5.3	Représentation $dy - dx$ . . . . .	85
4.6	Limitations et problèmes non-résolus . . . . .	92
4.6.1	Forme de la carte inadaptée, unités mortes . . . . .	92
4.6.2	Projection discrète . . . . .	94
4.6.3	Occultation de points isolés (événements rares) . . . . .	94
<b>5</b>	<b>“Neural Gas”</b>	<b>97</b>
5.1	Algorithme original . . . . .	97
5.2	Algorithme modifié . . . . .	99
5.2.1	Tri limité . . . . .	100
5.2.2	Suppression des unités mortes . . . . .	101
5.2.3	Apprentissage continu (réglage automatique des paramètres)	102
5.2.4	Résumé des paramètres . . . . .	103
<b>6</b>	<b>Algorithme VQP</b>	<b>105</b>
6.1	Introduction . . . . .	105
6.2	Algorithme . . . . .	107
6.3	Projection sans réduction de dimension . . . . .	111
6.4	Réduction de dimension (“dépliage”) . . . . .	111
6.5	Qualité de la projection . . . . .	113
6.6	Exemples . . . . .	114
<b>7</b>	<b>Propriétés particulières de VQP</b>	<b>121</b>
7.1	Projection continue (...) . . . . .	121
7.2	Précision de la projection continue . . . . .	125
7.3	Projection continue dans le bruit . . . . .	129
7.4	Projection inverse . . . . .	131

<b>8</b>	<b>Représentation non linéaire</b>	<b>133</b>
8.1	“ <i>Multidimensional Scaling</i> ” . . . . .	133
8.1.1	Notion de continuité en une dimension . . . . .	133
8.1.2	Extension au cas multidimensionnel . . . . .	134
8.1.3	Normalisation . . . . .	135
8.1.4	Mise en œuvre . . . . .	136
8.2	Lien avec les cartes de Kohonen . . . . .	137
8.2.1	Premier terme : $E^{vq}$ . . . . .	138
8.2.2	Second terme : $E^{mds}$ . . . . .	139
8.2.3	Analyse . . . . .	140
8.3	“ <i>Nonlinear Mapping</i> ” . . . . .	141
8.4	Comparaison avec VQP . . . . .	142
<b>9</b>	<b>Applications</b>	<b>149</b>
9.1	Vers une procédure d’analyse de données . . . . .	149
9.2	Fusion multi-capteurs . . . . .	152
9.2.1	Localisation . . . . .	152
9.2.2	Fusion auditive et visuelle . . . . .	154
9.2.3	Apprentissage de séquences temporelles . . . . .	157
9.2.4	Algorithme de super-résolution . . . . .	160
9.2.5	Détection de fautes dans des circuits électriques . . . . .	162
9.3	Analyse de procédé . . . . .	165
9.3.1	Extraction de séquence d’automate . . . . .	165
9.3.2	Surveillance de centrale nucléaire . . . . .	170
9.4	Fabrication de métrique . . . . .	173
9.4.1	Routage adaptatif de paquets en télécommunications . . . . .	174
9.4.2	Cartographie de concepts en fonction de distances subjectives . . . . .	177
9.5	Appariement de graphes . . . . .	178
<b>10</b>	<b>Conclusion</b>	<b>181</b>
10.1	Synthèse . . . . .	181
10.2	Perspectives . . . . .	184
	<b>Bibliographie</b>	<b>189</b>





*Il y a au bureau :*

*1 Moi.*

*2 Les autres.*

*De moi, je ne dirai rien. Vous savez la nature d'élite que je suis. Très capable et très comme il faut. De l'intelligence, de la conduite, et 34 000 francs de rente, ce qui ne gêne rien. Les autres, c'est plus mêlé.*

*Alphonse Allais*

## Remerciements

Lorsque je suis arrivé au laboratoire TIRF, j'étais très heureux et fier d'avoir été accepté dans une équipe aussi dynamique, compétente et sympathique. J'avais déjà pu connaître au gré de projets européens les hautes qualités humaines et scientifiques de ses leaders Jeanny Hérault et Christian Jutten. J'étais pourtant loin de mesurer à quel point cette expérience allait être extraordinaire et enrichissante.

La qualité de l'ambiance bon enfant et la bonne humeur qui règnent au TIRF sont en grande partie l'œuvre de son directeur Christian Jutten. Il est très attentif à entretenir l'esprit d'équipe du laboratoire et veille au confort moral de chacun. J'ai souvent pu éprouver sa bienveillance, sa gentillesse et sa courtoisie. Pour moi comme pour beaucoup d'autres étudiants, il est une sorte de grand frère qui arbitre avec un grand souci d'équité les petites vagues de la vie du labo. Toujours prêt à appeler ses étudiants "collègues", il s'abstient de faire peser l'autorité que lui confère son titre, lui préférant celle, naturelle, que lui donne sa rigueur et sa force de chercheur hors pair. Par ailleurs, cet implacable joueur de tennis possède sur son bureau une imposante culture de bouquins, papiers, dessins et autres objets divers qui font la joie de tous ceux qui, vaguement inquiets au sujet de l'état de leur propre table, se disent depuis six mois qu'ils devraient faire un peu de ménage. . .

Je ne crois pas qu'on puisse rêver d'un meilleur directeur de thèse que Jeanny Hérault. Il allie l'humour et l'enthousiasme à la capacité stupéfiante de générer cent idées par jour. En outre, il possède cette rare qualité d'écoute si importante pour un jeune chercheur. Je ne compte plus le nombre de fois où, tandis que je réfléchissais à haute voix dans son bureau, je le voyais, une étincelle dans l'œil, attendre patiemment que j'aie fini de formuler une idée encore confuse, ou bien l'observation de quelque phénomène curieux et (pour moi) incompréhensible. Après une réflexion en profondeur, ponctuée de quelques volutes de sa Gitane, il fournissait alors une explication saisissante de limpidité, aux implications multiples. Jeanny montre une très grande confiance et une tolérance exceptionnelle envers ses étudiants. Cette absence de contrainte galvanise l'initiative qui doit être le moteur principal du "métier de curieux professionnel" (selon un mot de

David Ruelle). Mon grand souhait est de pouvoir encore souvent partager le goût des belles idées, de la belle mécanique, des bons vins et de la bonne chère avec cet ami précieux.

Je voudrais exprimer ma gratitude aux institutions qui m'ont permis d'effectuer ce travail de doctorat, soit :

- le Fonds National de la Recherche Scientifique en Suisse, pour le financement de ma première année de thèse,
- la région Rhône-Alpes, pour le financement des deux suivantes;
- la société Industrie et Techniques de la Machine Intelligente (ITMI), ainsi que
- le groupe PSA (Peugeot & Citroën), tous deux partenaires du projet ANVAR (Agence Nationale pour la Valorisation de la Recherche) sur la surveillance de procédé industriel qui était le cadre du travail.

Je souhaite également remercier les membres du jury, pour la caution qu'ils ont bien voulu apporter à ce travail :

- le Professeur Jean-Marc Dolmazon, Institut de la Communication Parlée de l'Institut National Polytechnique de Grenoble (ICP-INPG).
- le Professeur Marie Cottrell, responsable du groupe SAMOS de l'Université Paris I.
- le Professeur Christian Wellekens, Eurecom, Sophia-Antipolis.
- le Professeur Alain Germond, Directeur du Laboratoire de Réseaux Electriques de l'Ecole Polytechnique Fédérale de Lausanne (LRE-EPFL).
- le Docteur Vincent Lorquet, Chef de projet à la société ITMI, Grenoble.

et en particulier les rapporteurs, Marie Cottrell et Christian Wellekens, pour l'important travail de lecture critique du manuscrit.

Enfin, tous mes chaleureux remerciements vont à mes copains, collègues anciens ou nouvellement arrivés au TIRF et compagnons de tous les jours, soit dans le désordre<sup>1</sup> :

- Ali Chams, ancien "system manager", qui croit en Dieu plus qu'en UNIX,
- Patricia Planet, dite "Patoucnet" en raison de sa satellisation pendant quelques années au CNET,

---

<sup>1</sup>désordre le plus complet, d'ailleurs, puisqu'il résulte d'un tirage aléatoire

- Maciek Orkisz, qui enchaîne les périlleux avant dans la poudreuse comme on enfile des perles,
- Christian Dumontier, standardiste téléphonique de mon bureau,
- Giansalvo Cirrincione, personnage onctueux dont les circonlocutions verbales démontrent la grande maîtrise du sujet, quel qu'il soit, et qui est particulièrement redoutable aux échecs paraît-il,
- Alfred Stork, dont les méditations manifestement torturées devant l'écran faisaient plaisir à voir, et qui a bien compensé ses vacances pour motif de vendanges en rapportant quelques puissantes bouteilles de blanc,
- Lan Nguyen Thi, chez qui ce qui frappe d'emblée, ce n'est pas ce qu'elle essaie d'expliquer mais le fait qu'elle se marre beaucoup en le disant,
- Christian Croll, qui rêve de me battre un jour au billard, et dont la supériorité sur moi au tennis, absolument intolérable, ne saurait durer,
- Ali Mansour, le seul à ce jour capable de m'extirper d'un programme particulièrement ardu, malgré mes écouteurs incontournables marquant bien ma répugnance à répondre à quelque question que ce soit pour l'instant, rien qu'en restant avec obstination et un grand sourire planté à côté de moi,
- Aude Oliva, ma compagne de trois ans, dont les mousses au chocolat ont été appréciées par tout le labo et les idées géniales sur le traitement visuel par toute la communauté scientifique,
- Adrian Spinei, le seul à avoir remarqué (avec un dépit silencieux mais perceptible quand même) l'infâme ruse me permettant d'avoir la priorité en temps de calcul par rapport aux hôtes indésirables de ma station : un petit script nommé "calme\_autres", ce qui veut tout dire, qui donnait 90% de CPU à mon "Mandelbrot" et 10% à son "Markov",
- Alain Chéhikian, ancien directeur du laboratoire depuis sa création, d'une rigueur imparable, aimant les mots d'esprit et grand disciple de Chéops ou alors c'est moi qui n'ai rien compris,
- Pierre-Yves Coulomb, dit "PYC", très au courant des mouvements de personnel à l'INPG et qui nous a toujours bien fait profiter des pommes sinistrées de son jardin (si si, c'était bien des pommes),
- Jean-Christophe Lawson, pour son magnifique et vain combat contre la fumée au laboratoire (et non contre les fumeurs, contrairement à ce qu'ont parfois cru les intéressés),

- Michel Crépin-Jourdan, fumeur, promu administrateur système, dont le bureau est appelé “la cuisine” pour des raisons qu’il déplore à grand bruit,
- Guillaume Vernat, contraint depuis peu à renifler des fromages plus ou moins avancés, d’où son ardeur compréhensible à développer un nez artificiel opérationnel,
- Patricia Palagi, le soleil de Rio au labo,
- Jean-Pierre Charras : ses PC, il les aime, il les chérit, il les démonte, modifie, remonte, il les bidouille, il les rend périodiquement amnésiques, il essaie même de les mettre en réseau, bref, il les fait vivre (ou mourir selon les cas et alors là, il appelle vite Nabil pour que tout soit réparé dans les nuits qui suivent); à part ça, il fut un adversaire honorable dans nos joutes verbales opposant ses odieux PC à nos splendides machines UNIX,
- Marie-Noëlle Matraire, notre secrétaire bien aimée, qui tape plus vite que son ombre quand elle en a envie, et qui est toujours d’accord de dépanner les gens surchargés, enfin, pendant les heures ouvrables,
- Vincent Fristot, qui a si bien su planquer les softs Mac,
- Alice Caplier, terrible prédateur dans la niche informatique,
- Nawar Al Awa, pour son extrême gentillesse et pour ses grands galops pleins de style d’un bout à l’autre du labo, c’est-à-dire entre son PC et son téléphone,
- Denis Pellerin, barbu attachant et amateur de “Fjords” (à moins que ce ne soient les Fjords qui attachent à sa barbe),
- Gérard Bouvier, dit “Gégé”, grand sensible d’abord bougon, fiché à l’INPG pour sa déplorable propension à casser les barrières fermées avec sa motocyclette (comment ça va, la France, Gégé ?),
- François Devillard, surnommé “la taupe” pour ses tendances spéléomanes ainsi que pour sa faible acuité visuelle,
- Olivier Fambon, surfeur en poudreuse comme en programmation (*I mean “free style”*),
- Frank Luthon, dont l’aspiration secrète consiste à l’évidence à parvenir un jour à trouver un éclairage nouveau des théories de Nietzsche par les chaînes de Markov (ou peut-être est-ce l’inverse),
- William Horace-Ambroise Beudot (c’est pas ça, “H.A.” ?), rétinophile émérite et travailleur acharné,

- Nabil Maria, collègue de nuit, compagnon de “system management”, de whisky, de billard, et grand hacker impénitent devant l’Ethernet,
- Anne Guérin, dont la pugnacité au travail a toujours fait l’admiration de tous, et la mienne en particulier lors de la préparation commune du tutorial de Neuronîmes sur les réseaux auto-organisés,
- Carlos Avilès, très discret sauf par son temps de calcul sur les stations des autres,
- Rachida Chentouf, aficionada de Packlib, mais elle le cache bien,
- Nabil Charkani, dont les lunettes cassées à une seule branche trahissent un sérieux scientifique que ses travaux ne contredisent pas d’ailleurs,

et enfin, je remercie de leur existence mes précieux compatriotes d’outre-mail, Laurent Tettoni, Yves Cheneval et Igor Löbl, qui ont eu le bon goût de rire et même de surenchérir aux plaisanteries gaillardes de mes gaulois collègues.



# Préambule

Chercher à *comprendre* des données, c'est souvent chercher à trouver de l'information cachée dans un gros volume de mesures. C'est chercher des corrélations, des dépendances, des liens de cause à effet, c'est tenter de simplifier un grand nombre d'observables en un petit nombre de paramètres.

Une étape fondamentale dans cette démarche de compréhension est de chercher la structure du nuage des données dans l'espace d'observation. Quelle est la dimension intrinsèque de ce nuage, ou, en d'autres termes, de combien de paramètres libres dépend le problème observé ? Quelle est la forme de l'espace paramétrique ? Quelles sont les relations entre les données et ces paramètres cachés ?

Au long de ce mémoire, nous analysons des techniques qui tentent de répondre à ces questions. Tout d'abord, dans le chapitre 1, nous montrons d'une façon intuitive les enjeux et les difficultés d'une telle démarche, que nous illustrons à l'aide d'un exemple simple où l'espace paramétrique est connu. Cet exemple permet de voir que la dimension intrinsèque (le nombre de paramètres) peut être beaucoup plus petite que la dimension de l'espace d'observation, et que les liens entre les paramètres et les données peuvent être fortement non-linéaires.

Au chapitre 2, on s'intéresse à quelques caractéristiques statistiques propres aux espaces de grandes dimensions. On y décrit quelques moyens préliminaires d'analyse, comme la vue directe du nuage de données selon différents angles (projection), selon une rotation continue, ainsi que l'Analyse en Composantes Principales, qui limite la recherche de structure aux formes linéaires. On y précise aussi la notion de dimension intrinsèque d'un nuage, liée à la notion de variété ou de sous-variété (qui est une généralisation de la notion de surface en géométrie différentielle). Dans ce chapitre donc, on se préoccupe essentiellement de dimensions (brute et intrinsèque), de leur définition, de leurs effets, et de la perspective de passer de l'une à l'autre par réduction.

De façon complémentaire, le chapitre 3 traite du nombre d'échantillons disponibles, et de la réduction de ce nombre à un ensemble de représentants pertinents. Cette réduction est l'objet de la *quantification vectorielle*, dont nous rapportons quelques principes et méthodes. En général, ces méthodes souffrent de quelques

problèmes connus, comme celui de la faible efficacité (la lenteur) des algorithmes, leur non-optimalité (les fonctions de distorsion ou de coût à minimiser ne sont pas convexes), et celui des unités mortes (inutiles parce que laissées en dehors de la distribution des données). Si la vitesse des algorithmes peut être améliorée en adaptant tous les prototypes (et non le plus proche seulement) lors de la présentation d'une donnée, il est alors nécessaire de prendre des précautions pour éviter un "collage" (convergence vers un point unique) de tous ces prototypes. En analysant les méthodes qui parviennent à éviter ce phénomène bien qu'adaptant tous les points à chaque itération, nous trouvons qu'il est nécessaire de découpler d'une façon quelconque l'adaptation de la pure distance entre échantillon et prototype. Nous étudions aussi des méthodes permettant d'éviter des minima locaux de la distorsion, et également des principes pour éviter les unités mortes.

Enfin, nous soulevons un problème qui n'est généralement pas abordé dans l'étude de la quantification vectorielle. C'est celui de l'antagonisme qui existe entre la fonction de coût généralement minimisée par les algorithmes, et les buts que l'on cherche à atteindre dans certains problèmes. Généralement, la minimisation de cette fonction de coût force les prototypes à suivre à peu près la même densité de probabilité que les données. Cette tâche est différente de ce que l'on souhaite parfois, non seulement dans notre démarche de recherche de structure (où l'on désire séparer la *forme* du support des données et la *densité* de celles-ci sur ce support), mais aussi dans certains problèmes de représentation, où les événements rares sont précisément les plus importants. La théorie de l'information indique justement que les événements rares sont les plus informatifs (sauf quand c'est du bruit), or la quantification vectorielle tend généralement à les négliger. La difficulté de la discrimination entre bruit et point informatif montre l'importance des connaissances *a priori* du phénomène observé. Nous proposons une méthode originale de régularisation qui permet de choisir de façon continue entre respect de la densité de distribution et régularité de la quantification.

Réduction de dimension et réduction du nombre d'échantillons sont deux tâches effectuées en parallèle par l'algorithme d'auto-organisation de Kohonen. Les *cartes auto-organisantes*, dont l'étude fait l'objet du chapitre 4, peuvent en effet être vues comme une quantification vectorielle, dans l'espace des données, par les poids d'un ensemble de neurones, sous contrainte d'un voisinage préétabli dans l'espace physique des neurones. La mise en relation de l'espace des données et de l'espace physique des neurones (de dimension plus petite, généralement une grille à une ou deux dimensions) est une sorte de projection non-linéaire des données. Malheureusement, on dispose de peu de résultats sur la convergence de cet algorithme quand l'espace de départ est multi-dimensionnel. Ce fait est d'autant plus regrettable que l'algorithme est assez sensible à certains de ses paramètres, et qu'il est difficile de s'apercevoir, en grande dimension, d'un défaut d'organisa-



tion. C'est pour cette raison que nous avons développé des méthodes d'analyse de la qualité du respect de topologie entre les espaces d'entrée et de sortie. Parmi ces méthodes, la plus riche en information est la représentation  $dy - dx$  (§ 4.5.3), qui est un diagramme de dispersion entre les distances dans l'espace de sortie et celles dans l'espace d'entrée.

Le principal problème des cartes de Kohonen est que leur forme et leur dimension sont figées; ces paramètres doivent être fixés *a priori*. Lorsque la dimension et la forme de la grille ne conviennent pas au problème traité, on obtient des défauts de représentation importants sur le plan topologique. Si notre mesure  $dy - dx$  permet de mettre ces défauts en évidence, elle n'indique pas comment choisir forme et dimension de façon plus adéquate. En fait, l'idéal serait d'obtenir la même fonctionnalité que le réseau de Kohonen, mais en se libérant de la contrainte d'une grille prédéfinie en sortie. C'est la raison pour laquelle un certain nombre d'algorithmes d'"auto-organisation" qui abandonnent cette notion de grille ont vu le jour, comme par exemple le "*Neural Gas*" (chapitre 5), qui est en réalité surtout une méthode de quantification vectorielle, l'information topologique étant donnée par un graphe de connectivité locale.

Le problème de ces algorithmes qui abandonnent la *structure* régulière de sortie du réseau de Kohonen est de perdre la possibilité de représentation qu'offre cette structure. En effet, l'aspect synthétique de la projection de données sur une carte de Kohonen, qui permet de voir du premier coup d'œil des ressemblances entre certaines données, des proximités, des classes, n'est absolument pas accessible dans un graphe de proximité, beaucoup plus difficile à analyser.

Dans le chapitre 6, nous présentons un algorithme d'auto-organisation original, que nous appelons "*Vector Quantization and Projection*" (VQP), et qui s'inspire des cartes de Kohonen. Les deux fonctions de ces dernières, quantification vectorielle et projection non-linéaire, sont aussi réalisées ici, mais en étant clairement séparées et traitées par deux couches différentes de vecteurs-poids. Le plus important est qu'il n'y a plus de contrainte de forme (grille) sur la sortie, qui est ici un espace continu où les vecteurs-poids de la couche de projection viennent se positionner pour copier aussi bien que possible la topologie des vecteurs-poids trouvés en entrée par quantification vectorielle. Seule la dimension de sortie est fixée (par exemple sur la base des résultats trouvés par analyse de la dimension fractale des données, voir § 2.5.2). La représentation dans l'espace de sortie se fait par minimisation d'un critère quadratique de distorsion basé sur les distances entre points dans les deux espaces. Grâce à ces propriétés, l'algorithme peut traiter les mêmes problèmes que les cartes de Kohonen, mais d'une façon beaucoup plus précise. Surtout, VQP permet de représenter des structures de données beaucoup plus complexes, là où le réseau de Kohonen ne fonctionne pas correctement : quand le support des données n'a pas la même forme carrée ou rectangulaire que

la grille support des neurones.

VQP met en relation deux espaces (c'est un *morphisme*), celui des données et celui de représentation (ou espace paramétrique), à l'aide de quelques échantillons. Le chapitre 7 montre comment on transforme cette relation discrète en relation continue, c'est-à-dire valable pour tout point de la distribution, et de façon bijective (de l'entrée vers la sortie, mais également en sens inverse, de la sortie vers l'entrée). Contrairement aux méthodes habituelles, généralement capables d'interpolation seulement, notre algorithme fonctionne en extrapolation aussi bien qu'en interpolation. De plus, les précisions obtenues sont en général excellentes.

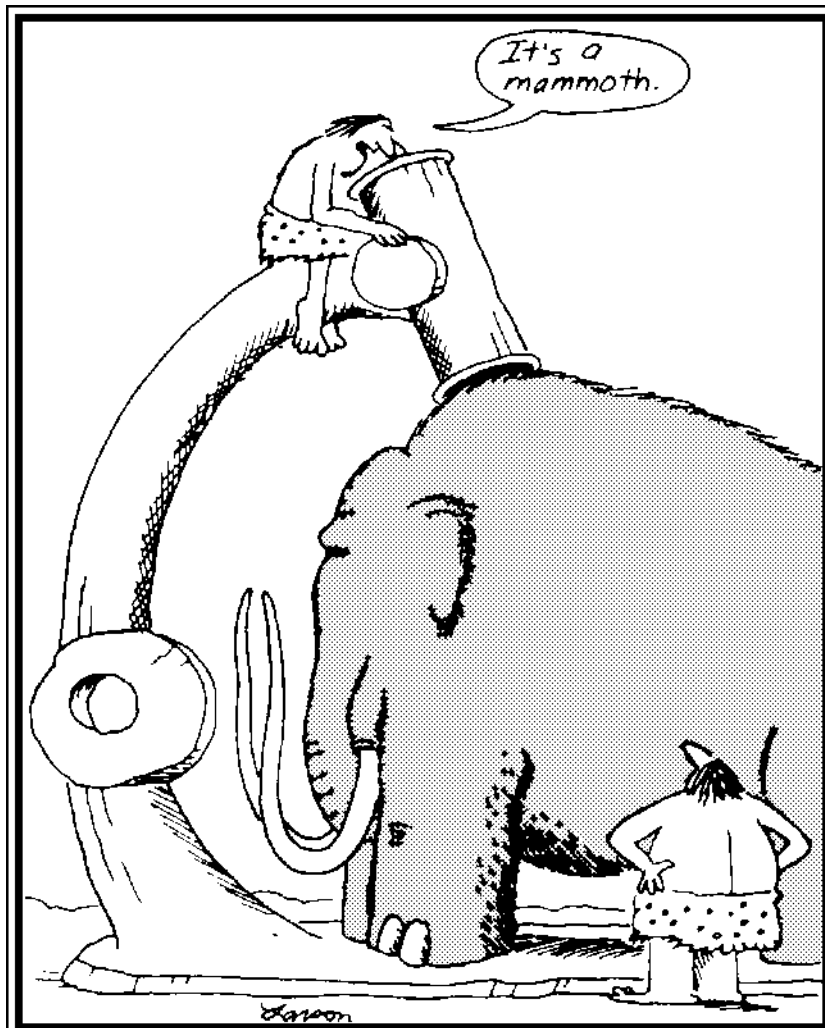
Au chapitre 8, nous présentons d'autres algorithmes statistiques de représentation de données par dépliage ou minimisation d'un critère de distorsion, le "*Multidimensional Scaling*" (MDS) et le "*Nonlinear Mapping*" (NLM), qui ont vu le jour dans les années soixante déjà. Ces algorithmes sont comparés avec VQP, sur le plan de la fonction et de l'efficacité. Fonctionnellement, malgré la grande similarité qui existe dans la forme des critères de distorsion minimisés par le NLM et par VQP, notre méthode permet de représenter des structures beaucoup plus fortement pliées que le NLM. La raison de ce fait provient de la façon dont on stipule que le respect de la topologie peut n'être que local dans le cas de structures pliées. Alors que cette localité s'exprime dans l'espace d'entrée avec le NLM, nous l'exprimons dans l'espace de sortie avec VQP (par analogie avec les cartes de Kohonen). Cette sorte de *récurtivité* dans la solution permet de couper les structures fermées ou simplement fortement pliées. Sur le plan de l'efficacité, notre méthode de minimisation du critère de distorsion se distingue de la méthode du gradient utilisée dans le NLM par le fait qu'elle est beaucoup plus légère (chaque itération est d'ordre  $O(N)$  au lieu de  $O(N^2)$ , aussi bien en occupation mémoire qu'en complexité de calcul). D'autre part, l'évolution du critère de distorsion n'est pas strictement décroissante avec VQP, ce qui permet d'échapper à des minima locaux du critère.

Dans le même chapitre, nous proposons un découpage de la règle d'adaptation de Kohonen qui révèle le lien entre les cartes auto-organisantes et le MDS (projection non linéaire) d'une part, et avec la quantification vectorielle d'autre part.

Finalement, nous proposons une méthodologie d'analyse de données qui s'appuie sur les divers points analysés dans ce mémoire : recherche de dimension intrinsèque, application de VQP, examen de la qualité de représentation, projection et projection inverse de la distribution avec VQP pour déterminer si l'on a trop (ou pas assez) "lissé les virages" de la structure des données. Cette procédure doit être appliquée plusieurs fois pour adapter les différents paramètres (dimension de sortie, nombre de neurones, degré de lissage). Nous formons l'hy-

pothèse que la solution du problème de recherche de structure doit être *récursive* d'une certaine manière, puisque la dimension que l'on trouve par analyse fractale dépend de l'échelle d'observation, donc du nombre de neurones utilisés, et que ce dernier dépend à son tour de la dimension de l'espace de sortie.

Le dernier chapitre est consacré à la présentation d'un petit nombre d'exemples d'applications très diverses, qui illustrent la généralité du propos et l'étendue du champ d'application de l'algorithme VQP.



Early microscope

The Far Side<sup>2</sup>

By Gary Larson

---

<sup>2</sup>The Far Side cartoon by Gary Larson is reprinted by permission of Chronicle Features, San Francisco, CA. All rights reserved.

# Chapitre 1

## La recherche de structure

La recherche de structure dans une distribution multi-variée est la plus répandue des formes d'analyse de données. De la médecine à la sociologie, de la reconnaissance des formes au contrôle de procédé industriel en passant par l'analyse géopolitique, cette opération qui vise à simplifier une masse de données complexes et redondantes, car liées entre elles, est une des démarches fondamentales pour analyser un phénomène.

La Nature, qui a tout inventé des millions d'années avant nous (*nécessité oblige*), n'est pas en reste dans ce domaine. La recherche de structure dans les stimuli que reçoit tout être doté d'un système nerveux me semble la base et même la raison d'être du traitement neuronal : c'est par cette recherche que l'animal perçoit sa propre structure ainsi que celle de son environnement. En cherchant des corrélations, des régularités, des similitudes (pour faire des classes), des co-occurrences, des liens entre les centaines de millions de variables qu'il traite en continu, cet être organise son formidable volume de mesures en des représentations synthétiques et simplifiées, nouveaux espaces dans lesquels les informations sont plus sûres, plus résistantes aux bruits et aux pannes de capteurs (c'est l'une des découvertes de la fameuse *fusion de données*). A partir de ces représentations simplifiées, les décisions sont aussi beaucoup plus faciles à prendre.

Il ne faut donc pas s'étonner de la magnifique organisation des cartes sensorielles (ou "topiques") qu'on rencontre abondamment chez les vertébrés supérieurs par exemple. Ces cartes participent justement à cette simplification de données. Leur omniprésence, tant au niveau des modalités sensorielles et motrices où on les observe qu'à celui de la variété des espèces qui les utilisent dans des contextes fort différents, semble suggérer qu'elles constituent des méthodes très efficaces<sup>1</sup> pour

---

<sup>1</sup>Je n'utiliserai pas ici le terme "*optimal*" si cher à certains ingénieurs mais dont je me méfie. Je me rappelle une personne qui s'étonnait de l'apparente complexité de l'oscillateur neuronal produisant le battement des ailes chez le criquet, alors qu'il suffit de deux transistors

traiter les données, et que les mécanismes qui les produisent sont relativement économiques.

En cherchant à comprendre la formation de ces cartes topiques, Von der Malsburg [118] a proposé son modèle d'auto-organisation, fondé sur l'observation de la rétinitopie. Par la suite, la simplification du modèle par Kohonen [66] pour aboutir aux *cartes auto-organisantes* a fourni un algorithme d'analyse de données très compact, où l'essence même de l'auto-organisation était produite. Cet algorithme, qui fait l'objet du chapitre 4, a depuis lors été utilisé par les ingénieurs dans nombre d'applications techniques et commerciales, comme la reconnaissance statistique des formes et de la parole, le contrôle de bras de robots, le contrôle de procédé industriel, la synthèse automatique de systèmes digitaux, des systèmes adaptatifs en télécommunications, la compression d'image, la classification radar et des problèmes d'optimisation. On a donc eu ici la naissance d'une nouvelle technique d'analyse de données inspirée de la biologie<sup>2</sup>.

Nous avons dit que les méthodes descriptives de l'analyse de données permettent une *réduction du volume des échantillons* sans perdre (trop) d'information. Il existe deux façons d'effectuer cette réduction, correspondant à la réduction des lignes et respectivement des colonnes de la matrice des observations (par exemple, nous choisirons la représentation par vecteurs-lignes : un échantillon par ligne). On peut réduire le nombre d'échantillons, par exemple par quantification vectorielle des lignes de la matrice. On peut aussi réduire le nombre de composantes de chaque échantillon, c'est-à-dire la dimension de l'espace d'observation (les colonnes de la matrice). Les réseaux auto-organisés, d'une façon générale, mettent en œuvre les deux méthodes simultanément !

## 1.1 Un exemple

Pour fixer les idées, imaginons que nous voulons réaliser un système de localisation (par exemple, un “*Global Positioning System*” — GPS, ou encore un digitaliseur). Pour cela, on dispose de  $n$  unités fixes  $\mathbf{r}_i$  qui servent de référentiel. On veut trouver

---

pour faire un oscillateur. Il serait prématurément présomptueux de croire qu'une structure n'est responsable que de la fonction à laquelle nous avons songé. Ensuite, la Sélection Naturelle ne s'occupe que peu de l'individu, son action se portant plutôt sur les espèces. J'entends par là qu'une structure optimale pour l'individu — au sens de l'ingénieur, c'est-à-dire minimale en composants — n'aura pas toute la potentialité d'évolution souhaitable. Le moindre changement dans l'environnement fera alors s'éteindre l'espèce concernée, piégée dans “son minimum local de complexité”.

<sup>2</sup>Cet exemple de démarche connexionniste est particulièrement réussi. On a vu là (et on voit encore) une véritable synergie entre les modélisateurs, animés par un souci académique de compréhension du monde vivant, et les ingénieurs qui veulent s'inspirer des techniques élaborées par l'Évolution pour résoudre des problèmes sur lesquels ils butent eux-mêmes.

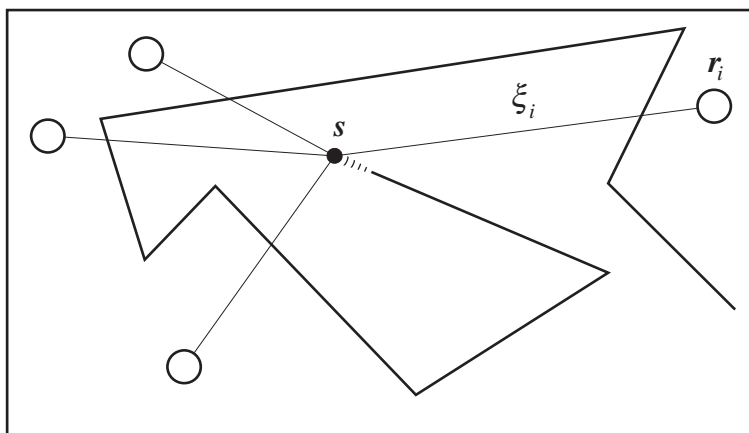


Figure 1.1: Un exemple simple pour générer une distribution redondante d'un nombre arbitraire de dimensions : chaque échantillon  $\xi$  est la collection des distances mesurées entre  $n$  unités fixes  $r_i$  de référence et un point mobile  $s$ .

la position d'un point mobile  $s$ , uniquement à partir des distances entre  $s$  et les  $r_i$ . Techniquement, la mesure de distance peut être réalisée de différentes façons. Par exemple, dans le cas d'un digitaliseur, ceci peut être fait avec un fil tendu entre  $s$  et  $r_i$  qui s'enroule sur un codeur incrémental. On peut aussi mesurer le temps de propagation d'un signal sonore émis au point  $s$  et reçu par les  $r_i$  qui sont alors des microphones. Dans le cas du GPS, c'est l'inverse : ce sont les  $n$  satellites qui envoient des signaux, et c'est le point  $s$  qui les reçoit et en déduit les distances. Quelle que soit la technique utilisée, il s'agit d'un problème de triangulation, et l'ingénieur sait qu'il faut  $n = p + 1$  unités fixes si le point  $s$  peut se mouvoir selon  $p$  directions (dans l'espace  $\mathbb{R}^p$ ). Classiquement, on suppose les  $r_i$  connus et on se lance dans de laborieux calculs pour trouver la position de  $s$  à partir des distances. Nous allons corser la difficulté en supposant *inconnues* les positions des unités fixes qui servent de référentiel. De plus, pour des raisons de robustesse et de tolérance à la faible qualité des capteurs que nous choisirons, on décide de mettre un nombre redondant (arbitrairement grand)  $n > p + 1$  d'unités fixes (figure 1.1).

Avec les  $n$  distances mesurées à un instant donné, on fabrique un vecteur  $\xi$  (à  $n$  composantes). L'espace qui contient les  $\xi$  est donc  $\mathbb{R}^n$ . Cependant, on voit bien dans cet exemple que les  $n$  distances ne sont pas indépendantes entre elles et l'espace qu'occupent effectivement les points  $\xi$  n'est qu'une petite partie de l'espace  $\mathbb{R}^n$ . En réalité, le nombre de degrés de liberté du problème correspond à la dimension  $p$  de l'espace euclidien où se meut le point  $s$  dont on cherche la position. Dans le cas du GPS ou d'un digitaliseur 3D, cette dimension est 3, alors que pour une tablette à digitaliser, c'est seulement 2.

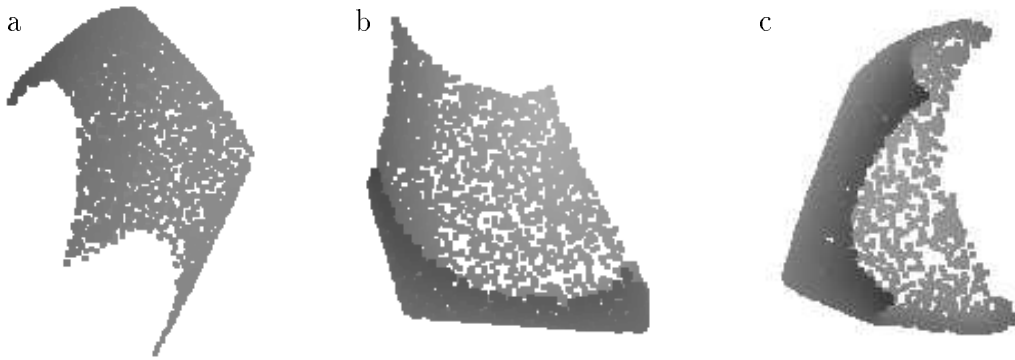


Figure 1.2: Vues selon des angles différents de la distribution “localisation” (20 dimensions, 2 degrés de liberté), à l’aide du “viewer” (voir section 2.3).

D’autre part, on peut facilement vérifier que la nature des dépendances entre les différentes composantes de  $\boldsymbol{\xi}$  est fortement non linéaire. En effet, deux composantes  $\xi_i$  et  $\xi_j$  (correspondant aux distances mesurées entre  $\mathbf{s}$  et  $\mathbf{r}_i$ , respectivement  $\mathbf{r}_j$ ) sont liées par les équations du triangle  $\langle \mathbf{s}, \mathbf{r}_i, \mathbf{r}_j \rangle$  :

$$a^2 = \xi_i^2 + \xi_j^2 - 2\xi_i\xi_j \cos \alpha \quad (1.1)$$

Où  $a$  est la distance qui sépare  $\mathbf{r}_i$  de  $\mathbf{r}_j$ , correspondant à l’angle  $\alpha$  depuis  $\mathbf{s}$ .

On tire de ces différentes observations que la structure des données dans  $\mathbb{R}^n$  est en fait une *variété* à seulement  $p$  dimensions intrinsèques, mais *pliée* dans  $\mathbb{R}^n$ . On remarque donc qu’une recherche d’une projection linéaire est vouée à l’échec, ce qui montre l’insuffisance des méthodes du type ACP (Analyse en Composantes Principales), que nous rappellerons à la section 2.4.

Comment donc retrouver la structure des données ? Le problème est d’autant plus délicat quand de surcroît on ne connaît rien de la variété — ni sa forme, ni sa dimension intrinsèque — ce qui est souvent le cas dans les applications réelles.

Examinons un cas particulier : l’espace libre de  $\mathbf{s}$  est un plan ( $\mathbb{R}^2$ ), et l’on place  $n = 20$  unités fixes de référence à des positions aléatoires dans le plan. Chaque vecteur  $\boldsymbol{\xi}$  est composé de 20 distances. Pourtant, la distribution des  $\boldsymbol{\xi}$  dans  $\mathbb{R}^{20}$  n’a que  $p = 2$  degrés de liberté. Il s’agit donc d’une surface, courbe, dans l’espace à 20 dimensions. La figure 1.2 montre quelques vues sous différents angles de cette distribution à 20 dimensions.

L’objectif est donc de déplier cette surface pour la représenter sur un plan. Nous verrons que les réseaux auto-organisés offrent une solution pour effectuer un tel dépliement, après quantification vectorielle de la distribution des  $\boldsymbol{\xi}$ .



# Chapitre 2

## Distributions en grandes dimensions

### 2.1 Norme de vecteurs aléatoires

Les distributions formées de variables indépendantes ne sont pas très intéressantes (c'est la redondance qui permet de dégager des formes et des propriétés dans les données [57], et à ce titre *la redondance produit de la connaissance* [4]). Cependant, il est instructif d'examiner quelques propriétés particulières des vecteurs dont les composantes sont aléatoires. Notamment, à la suite d'expériences sur le réseau de Kohonen qui donnait des résultats curieux en grandes dimensions (voir chapitre 4.5), j'avais conjecturé (par intuition et surtout grâce à la fonction `Fit[ ]` de Mathematica™...) que la norme de tels vecteurs possédait les propriétés statistiques suivantes, assez surprenantes : si l'espérance mathématique de la norme augmente comme prévu en  $\sqrt{n}$  (avec  $n$  le nombre des composantes indépendantes), la variance reste à peu près constante, en tout cas quand  $n$  devient suffisamment grand pour pouvoir appliquer le théorème central-limite. Plus précisément :

**Théorème 2.1** *Soit  $\mathbf{x}$  un vecteur en  $n$  dimensions  $[x_1, \dots, x_n]^T$  dont les composantes sont aléatoires, indépendantes et de loi identique<sup>1</sup>. On a :*

$$\begin{aligned}\mu_{\|\mathbf{x}\|} &= \text{E}(\|\mathbf{x}\|) &= \sqrt{an - b} + O(1/n) \\ \sigma_{\|\mathbf{x}\|}^2 &= \text{Var}(\|\mathbf{x}\|) &= b + O(1/\sqrt{n}),\end{aligned}\tag{2.1}$$

où  $a$  et  $b$  sont des paramètres dépendant uniquement des moments centrés d'ordre 1, 2, 3 et 4 des  $x_i$  :

$$a = \mu^2 + \sigma^2$$

---

<sup>1</sup>Et possédant un moment d'ordre 8 fini :  $\mu'_8 = \text{E}(x_k^8) \neq \infty$ .

$$b = \frac{4\mu^2\sigma^2 - \sigma^4 + 4\mu\mu_3 + \mu_4}{4(\mu^2 + \sigma^2)} \quad (2.2)$$

où  $\mu_r$  est le moment centré d'ordre  $r$  :  $\mu_r = \mathbb{E}[(x_k - \mu)^r]$ ,  $\mu$  est la moyenne  $\mathbb{E}(x_k)$  et  $\sigma^2$  la variance  $\text{Var}(x_k)$ .

La transformation de la conjecture en théorème est faite par la démonstration suivante, due à Jean-Claude Fort :

### Démonstration

Moyenne des  $x_k$  :  $\mu = \mathbb{E}(x_k)$ . Variance :  $\sigma^2 = \text{Var}(x_k)$ .

Moments d'ordre  $r$  :  $\mu'_r = \mathbb{E}(x_k^r)$ . Moments centrés :  $\mu_r = \mathbb{E}[(x_k - \mu)^r]$ .

Soit  $U_k = \frac{x_k^2}{\mu'_2}$ .  $\mathbb{E}(U_k) = \frac{\mathbb{E}(x_k^2)}{\mu'_2} = 1$  et

$$\mathbb{E}[(U_k - 1)^2] = \text{Var}(U_k) = \mathbb{E}(U_k^2) - \mathbb{E}(U_k)^2 = \frac{\mu'_4 - \mu'^2_2}{\mu'^2_2} \quad (2.3)$$

$$Z = \sqrt{\sum_{k=1}^n x_k^2} = \sqrt{\mu'_2} \sqrt{\sum_{k=1}^n \frac{x_k^2}{\mu'_2}} = \sqrt{n\mu'_2} \sqrt{\sum_{k=1}^n \frac{U_k}{n}} \quad (2.4)$$

$$= \sqrt{n\mu'_2} \sqrt{1 + \sum_{k=1}^n \frac{U_k - 1}{n}} \quad (2.5)$$

Rappel :  $\forall u \in [-1; +\infty[$ ,  $|\sqrt{1+u} - (1 + \frac{u}{2} - \frac{u^2}{8})| \leq C|u|^3$

Donc :

$$\left| Z - \sqrt{n\mu'_2} \left[ 1 + \sum_{k=1}^n \frac{U_k - 1}{2n} - \frac{1}{8} \left( \sum_{k=1}^n \frac{U_k - 1}{n} \right)^2 \right] \right| \leq \sqrt{n\mu'_2} C \left| \sum_{k=1}^n \frac{U_k - 1}{n} \right|^3 \quad (2.6)$$

C'est-à-dire :

$$\text{LT} \leq \text{RT} \quad \left( \begin{array}{l} \text{LT : Terme de gauche} \\ \text{RT : Terme de droite} \end{array} \right)$$

Terme de gauche

Comme  $|\mathbb{E}(A) - \mathbb{E}(B)| \leq \mathbb{E}(|A - B|)$ , on a :

$$\left| \mathbb{E}(Z) - \mathbb{E} \left( \sqrt{n\mu'_2} \left[ 1 + \sum_{k=1}^n \frac{U_k - 1}{2n} - \frac{1}{8} \left( \sum_{k=1}^n \frac{U_k - 1}{n} \right)^2 \right] \right) \right| \leq \mathbb{E}(\text{LT}) \leq \mathbb{E}(\text{RT}) \quad (2.7)$$

et comme, d'autre part (sous l'hypothèse que les  $U_k$  sont indépendants) :

$$\mathbb{E} \left[ \left( \sum_{k=1}^n \frac{U_k - 1}{n} \right)^2 \right] = \frac{1}{n^2} \sum_{k=1}^n \mathbb{E} [(U_k - 1)^2] \quad (2.8)$$

et

$$\mathbb{E} \left( \sum_{k=1}^n \frac{U_k - 1}{n} \right) = 0 \quad \text{car } \mathbb{E}(U_k) = 1, \quad (2.9)$$

on obtient :

$$\begin{aligned} \left| \mathbb{E}(Z) - \sqrt{n\mu'_2} \left[ 1 + 0 - \frac{1}{8n} \text{Var}(U_k) \right] \right| &\leq \mathbb{E}(\text{LT}) \\ \left| \mathbb{E}(Z) - \left( \sqrt{n\mu'_2} - \frac{\sqrt{n\mu'_2} \mu'_4 - \mu'_2{}^2}{2n\mu'_2} \right) \right| &\leq \mathbb{E}(\text{LT}). \end{aligned} \quad (2.10)$$

Rappel :  $\sqrt{u} - \frac{\beta\sqrt{u}}{2u} = \sqrt{u - \beta} + O(1/u)^{3/2}$ .

Donc :

$$\begin{aligned} \left| \mathbb{E}(Z) - \sqrt{n\mu'_2} - \frac{\mu'_4 - \mu'_2{}^2}{4\mu'_2} \right| &\leq \mathbb{E}(\text{LT}) + O(1/n)^{3/2} \\ &\leq \mathbb{E}(\text{RT}) + O(1/n)^{3/2}. \end{aligned} \quad (2.11)$$

*Terme de droite*

On peut montrer que  $R_n = \sum_{k=1}^n \frac{U_k - 1}{\sqrt{n}}$  converge en loi vers une normale et que si  $U_k$  possède un moment d'ordre 4 (donc si les  $x_k$  possèdent un moment d'ordre 8), alors  $\mathbb{E}(|R_n|^3)$  reste bornée.

Donc :

$$\begin{aligned} \mathbb{E}(\text{RT}) &= \mathbb{E} \left( \sqrt{n\mu'_2} C \left| \sum_{k=1}^n \frac{U_k - 1}{n} \right|^3 \right) \\ &= C \sqrt{\mu'_2} \mathbb{E} \left( \frac{\sqrt{n}}{(\sqrt{n})^3} \left| \sum_{k=1}^n \frac{U_k - 1}{\sqrt{n}} \right|^3 \right) \\ &\leq \frac{K}{n}. \end{aligned} \quad (2.12)$$

Ce qui mène finalement à :

$$\left| \mathbb{E}(Z) - \sqrt{n\mu'_2} - \frac{\mu'_4 - \mu'_2{}^2}{4\mu'_2} \right| \leq \frac{K}{n} + O(1/n)^{3/2}. \quad (2.13)$$

Et donc :

$$E(Z) = \sqrt{n\mu'_2 - \frac{\mu'_4 - \mu'_2{}^2}{4\mu'_2}} + O(1/n). \quad (2.14)$$

D'autre part,  $\text{Var}(Z) = E(Z^2) - [E(Z)]^2$  et  $E(Z^2) = n\mu'_2$ . On retrouve donc bien la forme de (2.1) :

$$\begin{aligned} E(\|\mathbf{x}\|) &= \sqrt{an - b} + O(1/n) \\ \text{Var}(\|\mathbf{x}\|) &= b + O(1/\sqrt{n}). \end{aligned} \quad (2.15)$$

Enfin, par application des formules de changement entre moments et moments centrés :

$$\mu'_r = \sum_{j=0}^r \binom{r}{j} \mu^{r-j} \mu_j \quad \text{avec} \quad \begin{aligned} \mu_0 &= 1 \\ \mu_1 &= 0 \\ \mu_2 &= \sigma^2 \end{aligned} \quad (2.16)$$

on retrouve pour les paramètres  $a$  et  $b$  indiqués en (2.2) :

$$\begin{aligned} a &= \mu'_2 = \mu^2 + \sigma^2 \\ b &= \frac{\mu'_4 - \mu'_2{}^2}{4\mu'_2} = \frac{4\mu^2\sigma^2 - \sigma^4 + 4\mu\mu_3 + \mu_4}{4(\mu^2 + \sigma^2)}. \end{aligned} \quad (2.17)$$

### CQFD

La signification de ce résultat est que, à partir d'un certain nombre de composantes indépendantes, les vecteurs  $\mathbf{x}$  semblent normalisés. En effet, quel que soit le type de distribution des composantes  $x_k$ , l'écart-type  $\sigma_{\|\mathbf{x}\|}$  de la norme tend vers une constante lorsqu'on augmente la dimension  $n$ , tandis que la moyenne  $\mu_{\|\mathbf{x}\|}$  croît en  $\sqrt{n}$ . Plus précisément, à cause de l'inégalité de Chebychev :

$$P(|\|\mathbf{x}\| - \mu_{\|\mathbf{x}\|}| \geq \varepsilon) \leq \frac{\sigma_{\|\mathbf{x}\|}^2}{\varepsilon^2} \quad (2.18)$$

la probabilité que la norme  $\|\mathbf{x}\|$  tombe en dehors d'un intervalle de taille fixée autour de  $\mu_{\|\mathbf{x}\|}$  devient approximativement constante quand  $n$  augmente. Comme  $\mu_{\|\mathbf{x}\|}$  lui-même continue à augmenter, l'erreur relative commise en prenant  $\mu_{\|\mathbf{x}\|}$  au lieu de  $\|\mathbf{x}\|$  devient négligeable. Ainsi, en grandes dimensions, des vecteurs aléatoires (dont les composantes suivent une loi donnée) semblent tous répartis à la surface d'une sphère de rayon  $\mu_{\|\mathbf{x}\|}$ .

Les implications de ce phénomène sont multiples. Pour un  $n$  fixé, outre la norme elle-même des vecteurs qui semble invariable d'un tirage à l'autre, la distance euclidienne entre deux vecteurs semble elle aussi invariable (quel que soit le couple de vecteurs choisi). En effet, la distance euclidienne est la norme de la

différence entre les deux vecteurs aléatoires, différence qui est aussi un vecteur aléatoire; donc cette distance suit les règles (2.1) (on peut grossièrement imaginer la chose en disant que l'espace se restreint aux sommets d'un "hyper-tétraèdre", où toutes les distances point-à-point sont identiques). On s'explique ainsi les problèmes rencontrés avec certaines mesures d'organisation du réseau de Kohonen (§ 4.5) en grandes dimensions. Le comportement des algorithmes eux-mêmes (Kohonen, voir chapitre 4, ou "*Multi Dimensional Scaling*", voir § 8.1) travaillant sur des espaces en grandes dimensions en est également éclairé.

Il ne faut toutefois pas perdre de vue que c'est le nombre de degrés de liberté des vecteurs  $\mathbf{x}$  qui est déterminant. En effet, comme on l'a dit plus haut, les distributions "intéressantes" sont généralement structurées. Le nombre de degrés de liberté  $p$  peut alors être beaucoup plus petit que la dimension  $n$ , et l'hypothèse d'indépendance entre les  $x_k$  n'est plus vérifiée. Si le nombre de degrés de liberté est malgré tout suffisamment grand, le résultat reste applicable (en changeant  $n$  par  $p$  dans les équations).

## 2.2 Espace vide

Un autre phénomène, plus connu, est celui dit de l'"*espace vide*" : plus la dimension d'un espace est grande, plus celui-ci semble vide. Considérons par exemple une hyper-sphère en  $n$  dimensions. En fonction du rayon  $r$ , son volume  $V_n(r)$  vaut :

$$V_n(r) = \frac{\pi^{n/2} r^n}{\Gamma(1 + n/2)} \quad (2.19)$$

Le fait que  $n$  soit en exposant fait vite prendre des valeurs élevées à cette fonction. Par exemple, si l'on cherche à quantifier un espace sphérique avec un pas de quantification maximum de  $r/10$ , le nombre  $N$  de points augmente comme suit en fonction de la dimension  $n$  :

$n$	1	2	3	4	5	6
$N$	20	314	4 188	49 348	562 379	$> 5 \cdot 10^6$

Non seulement le nombre de points nécessaires devient exorbitant, mais encore faut-il avoir une base de données suffisamment fournie en nombre d'échantillons ! Comme en général ce nombre est relativement limité, l'espace semble quasiment vide si la dimension intrinsèque de la distribution est grande.

Un problème, lié à ce qui précède, concerne la sélectivité d'un noyau gaussien dans les réseaux de type "*Radial Basis Function*" (RBF), abondamment employés

dans l'approximation de fonction, par exemple [100], ou dans l'estimation de densité de probabilité [7]. Dans un tel réseau, chaque unité  $i$  se spécialise dans une région de l'espace centrée sur un point  $\mathbf{x}_i$  qu'on appelle le *centroïde* de l'unité. La fonction d'activation de l'unité autour de son centroïde est une *fonction noyau*  $K_i(r)$ , ou *fonction radiale de base*, dont l'argument est la distance euclidienne entre le vecteur présenté  $\boldsymbol{\xi}$  et le centroïde :  $r = \|\boldsymbol{\xi} - \mathbf{x}_i\|$ . Les noyaux gaussiens sont un cas particulier fréquemment utilisé :

$$K_i(r) = \exp\left(-\frac{r^2}{2\lambda_i^2}\right) \quad (2.20)$$

L'idée est de paver la distribution avec ces noyaux. Lorsqu'on présente un nouveau vecteur, l'activité de toutes les unités est prise en compte dans un calcul de barycentre pour calculer l'approximation de la fonction par interpolation entre les points appris. Par exemple, la reconstruction  $y$  d'une fonction scalaire apprise à l'aide d'échantillons de couples  $(\boldsymbol{\xi}_i, y_i)$  peut s'écrire :

$$y = f(\boldsymbol{\xi}) = \sum_i a_i K_i(\|\boldsymbol{\xi} - \mathbf{x}_i\|) \quad (2.21)$$

avec les paramètres  $a_i$  et  $\lambda_i$  adaptés pendant l'apprentissage pour obtenir une erreur de reconstruction la plus faible possible.

Cette fois, c'est la surface de notre hyper-sphère qui est en cause :

$$S_n(r) = \frac{2\pi^{n/2}r^{n-1}}{\Gamma(n/2)} \quad (2.22)$$

En effet, observons le comportement de ce que nous appellerons le *rayon d'activité* du noyau, c'est-à-dire le rayon de la sphère qui contient la portion de distribution responsable d'un certain pourcentage (prenons arbitrairement 95%) de l'activité totale. En normalisant l'activité totale du noyau à 1, et dans le cas d'une distribution uniforme autour du centroïde, on obtient :

$$\begin{aligned} r_{0.95} &= r \mid A_r = 0.95 \\ &= r \left| \frac{\int_0^r S_n(u)K(u, \lambda)du}{\int_0^\infty S_n(u)K(u, \lambda)du} = 0.95 \right. \end{aligned} \quad (2.23)$$

Le tableau suivant donne le rayon d'activité  $r_{0.95}$  d'un noyau gaussien dans un espace uniforme à  $n$  dimensions en fonction de  $n$  :

$n$	1	2	3	4	5	6
$r_{0.95}$	$1.96\lambda$	$2.45\lambda$	$2.80\lambda$	$3.08\lambda$	$3.33\lambda$	$3.54\lambda$

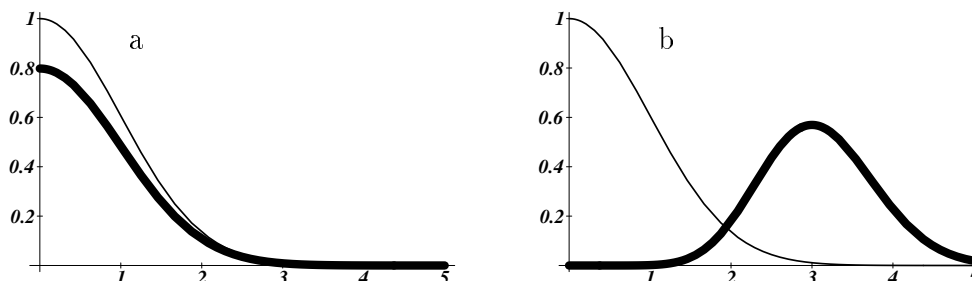


Figure 2.1: Noyau gaussien (ligne fine) et répartition de son activité (en gras) autour d'un point, en fonction de la distance radiale. Selon qu'on est (a) en faible dimension ou (b) en grande dimension (10 ici), la partie du noyau utilisée n'est pas la même. Dans le second cas, l'essentiel de l'activité du noyau est dû à la queue de la gaussienne, ce qui donne une mauvaise représentation de l'espace.

La contribution du noyau dans le calcul de barycentre est donc due à une portion d'espace d'autant plus étalée que la dimension est grande. Examinons maintenant comment cette activité  $A_r$  se répartit dans cette portion, soit la quantité :

$$\frac{dA_r}{dr} = \frac{S_n(r)K(r, \lambda)}{\int_0^\infty S_n(u)K(u, \lambda)du} \quad (2.24)$$

La figure 2.1 montre cette répartition en petite ou grande dimension. En (a), qui correspond au cas monodimensionnel ( $n = 1$ ), on voit que l'essentiel de l'activité est dû, comme prévu, à la portion tombant "dans" le noyau. Par contre, en 10 dimensions (b), l'essentiel de l'activité est dû à des points tombant sur la queue du noyau. Par conséquent, en grande dimension, on voit beaucoup de points qui activent faiblement le noyau, et peu qui l'activent normalement. Le noyau n'est pas très représentatif de la zone qu'il est censé représenter.

En outre dans les grandes dimensions, en raison des relations (2.1), la distance entre un point de la distribution et les centroïdes semble toujours à peu près constante, quel que soit ce point s'il est aléatoire. Tous les noyaux fournissent alors à peu près la même activité. Une faible partie de la fonction noyau est donc utilisée, et c'est la même pour tous les centroïdes (figure 2.1b).

Tout ceci montre à quel point il faut se méfier de la généralisation hâtive à  $n$  dimensions d'un résultat bien connu dans les petites dimensions. Cependant, encore une fois, la dimension à prendre en compte dans un problème n'est pas forcément aussi grande qu'il n'y paraît, et la plupart des applications réelles offrent des distributions fortement structurées, avec des degrés de liberté en bien plus petit nombre que celui des variables observées. Dans tous les cas, c'est ce nombre de degrés de liberté qui doit être considéré. Par exemple, une proposition a été émise afin de contrecarrer l'effet de la dimension  $n$  sur le comportement du noyau : c'est d'adopter des noyaux hyper-gaussiens de la forme  $K(r, \lambda) =$

$\exp(-r^{2n}/2\lambda^{2n})$  au lieu de noyaux simplement gaussiens [7]. Cette idée n'est pas bonne si elle est appliquée machinalement, en affectant simplement à  $n$  le nombre de composantes des vecteurs. Il vaut beaucoup mieux chercher d'abord à connaître la dimension intrinsèque  $p$  de la distribution (on discutera ce sujet à la section 2.5) et utiliser cette valeur  $p$  au lieu de  $n$ . En effet, imaginons une distribution contenue dans un hyperplan dans un espace à 1000 dimensions. Par une simple rotation, on peut remplacer les points de la distribution par des vecteurs dont seules les deux premières composantes ne sont pas nulles. On a beau avoir des vecteurs à 1000 dimensions, toutes les caractéristiques statistiques de l'espace seront celles d'une distribution en deux dimensions, donc où le phénomène d'espace vide n'existe pas.

## 2.3 Visualisation par rotation dynamique

Pour étudier une distribution en grande dimension, nous avons remarqué qu'une simple projection en trois dimensions, mais *dynamique*, c'est-à-dire selon une rotation qui change continûment au cours du temps, donne déjà souvent une bonne idée de la structure des données.

Pour former une telle représentation, il faut tout d'abord définir la rotation dans un espace à  $n$  dimensions. Pour cela, on définit un nouveau repère par une matrice  $Q$  que l'on construit ligne par ligne. On choisira ces lignes (axes) orthonormées et orientées positivement, afin que la prémultiplication par  $Q$  soit une rotation. Pour définir le vecteur-directeur du premier axe (appelons-le  $\mathbf{u}_1$ ), on a besoin de  $n - 1$  angles ( $n$  cosinus-directeurs, l'un d'eux étant calculé par rapport aux autres de façon que la norme soit unitaire). Tous les axes calculés par la suite seront orthogonaux à celui-là, donc on reprend le problème dans le sous-espace à  $n - 1$  dimensions, orthogonal au premier axe. On choisit alors un deuxième axe à l'aide de  $n - 2$  angles. Et ainsi de suite pour définir tous les axes du nouveau repère. On a donc besoin de  $n(n - 1)/2$  angles au total. A chaque étape, il ne suffit cependant pas de choisir la direction de l'axe qu'on veut fixer. Il faut encore trouver une transformation de tout l'espace qui soit compatible avec cette direction. Par exemple pour le premier axe, il faut obtenir, outre la direction  $\mathbf{u}_1$  de ce dernier, un système d'axes orthogonaux à  $\mathbf{u}_1$ , dans lequel on pourra exprimer la direction choisie pour le deuxième axe. De plus, comme on veut faire varier les angles dans le temps pour avoir une rotation dynamique, il faut que les composantes de la matrice soient des fonctions continues de ces angles (il ne doit pas y avoir de sauts).

Toutes ces conditions sont rencontrées si l'on construit  $Q$  à l'aide de *matrices*





bloc la première ligne du reste :

$$A = \begin{bmatrix} \overline{A_1} \\ A_R \end{bmatrix}, \quad (2.27)$$

$$B = \begin{bmatrix} 1 & \overline{B_1} \\ & B_R \end{bmatrix}, \quad (2.28)$$

$$C = \begin{bmatrix} 1 & & \\ & 1 & \overline{C_1} \\ & & C_R \end{bmatrix}, \quad (2.29)$$

on trouve pour les 3 premières lignes de  $Q$  :

$$Q_3 = \begin{bmatrix} A_1 \\ B_1 A_R \\ (C_1 B_R) A_R \end{bmatrix}. \quad (2.30)$$

Faisons le calcul des composantes de chaque bloc dans  $A$ ,  $B$  et  $C$ . Notons ce bloc  $R$ , et sa dimension  $m \times m$  (pour  $A$ ,  $m = n$ , pour  $B$ ,  $m = n - 1$ , et pour  $C$ ,  $m = n - 2$ ).  $R_1$  est donc une ligne de taille  $1 \times m$ , et  $R_R$  une matrice de taille  $m - 1 \times m$ . Pour simplifier, notons les angles  $\alpha_i$  au lieu de  $\alpha_{1,i+1}$  pour  $A$ ,  $\alpha_{2,i+2}$  pour  $B$ , et  $\alpha_{3,i+3}$  pour  $C$ . Enfin, bien que dans chaque cas ni  $\alpha_0$  ni  $\alpha_m$  n'existent, on pose pour la simplicité d'écriture  $\sin(\alpha_m) = 1$  et  $\cos(\alpha_0) = 1$ . On obtient alors :

$$R_{1i} = \sin(\alpha_{m+1-i}) \prod_{k=1}^{m-i} \cos(\alpha_k) \quad (2.31)$$

$$\begin{aligned} R_{Ri,j>i+1} &= 0 \\ R_{Ri,i+1} &= \cos(\alpha_{m-i}) \\ R_{Ri,j \leq i} &= -\sin(\alpha_{m-i}) \sin(\alpha_{m+1-j}) \prod_{k=m+1-i}^{m-j} \cos(\alpha_k) \end{aligned} \quad (2.32)$$

$$(2.33)$$

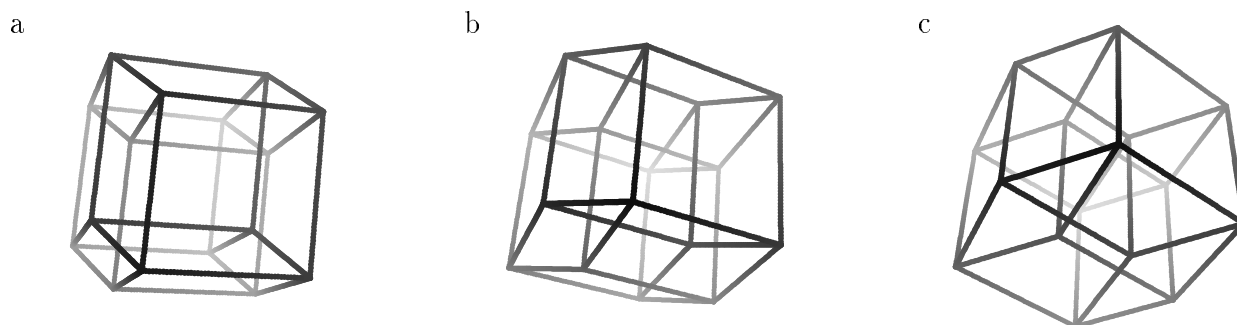


Figure 2.2: Vues selon différents angles des arêtes d'un hyper-cube en 4 dimensions.

Ainsi, le calcul de la projection peut être grandement simplifié et l'on obtient finalement un nombre de multiplications en  $O(6n^2)$  ( $O(2n)$  multiplications pour chaque première ligne de bloc :  $A_1$ ,  $B_1$  et  $C_1$ , et  $O(\frac{3}{2}n^2)$  pour les blocs  $A_R$  et  $B_R$ , plus  $O(n^2)$  pour chaque multiplication ligne-matrice). Au lieu de  $n(n-1)$  angles, on n'en a besoin que de  $(n-1) + (n-2) + (n-3) = 3n-6$ .

Chaque angle  $\alpha_{ij}$  est en réalité une fonction du temps  $\alpha_{ij}(t) = \dot{\alpha}_{ij}t$ . A chaque instant, on calcule une nouvelle matrice  $Q_3$  de projection de  $n$  dimensions vers 3 en fonction des  $3n-6$  angles  $\alpha_{ij}(t)$ . On prémultiplie la distribution  $\Xi$  par  $Q_3$ , ce qui nous donne un nuage en trois dimensions. Il ne reste plus qu'à représenter ce nuage sur l'écran selon les techniques habituelles d'infographie (perspective conique, affichage des points selon la profondeur  $z$  décroissante pour résoudre le problème des faces cachées —on utilise la méthode dite du “ $z$ -buffer”, où l'on quantifie la profondeur par un certain nombre de plans, puis où l'on trace les points d'un plan après l'autre—, taille et luminosité des points dépendant de  $z$ ). L'usage de ces techniques, associé au fait que la rotation est continue sur l'écran, permet une bonne visualisation de la structure des données.

Par exemple, la figure 2.2 montre les arêtes d'un hyper-cube en 4 dimensions.

## 2.4 Analyse en composantes principales

### 2.4.1 Méthode

L'analyse en composantes principales (ACP) est une des méthodes d'analyse de données les plus connues. Elle est parfois référencée sous le nom de “transformation de Karhunen-Loève”. Elle consiste à chercher un sous-espace de dimension plus petite que l'espace de départ et à y projeter la distribution étudiée, en perdant un minimum d'information. Le résultat ainsi obtenu est une représentation

des données avec *réduction de dimension*.

Pour atteindre ce résultat, on cherche les axes autour desquels l'inertie du nuage de points est la plus faible, c'est-à-dire ceux le long desquels la variance est maximale. Au préalable, afin de rendre le résultat indépendant des unités utilisées pour chaque composante, un prétraitement indispensable consiste à centrer et réduire les variables.

On collecte les vecteurs  $\boldsymbol{\xi}_i$  issus du système dans une matrice brute  $\Xi' N \times n$  (les lignes sont les  $N$  vecteurs  $\boldsymbol{\xi}_i$ , les colonnes sont les  $n$  composantes). Puis, on construit une nouvelle matrice  $\Xi$  dont les colonnes ont toutes la même variance (1, par exemple) et une moyenne nulle :

$$\Xi_{ij} = \frac{\Xi'_{ij} - E(\Xi'_{\bullet j})}{\sqrt{\text{Var}(\Xi'_{\bullet j})}}. \quad (2.34)$$

La recherche des axes qui maximisent la variance des points dans l'espace projeté peut se faire de différentes façons. Le long d'un axe représenté par le vecteur unitaire  $\mathbf{u}$ , la variance du nuage  $\Xi$  est :

$$V = \sum_{i=1}^N [\mathbf{u}^T (\boldsymbol{\xi}_i - \boldsymbol{\mu}_\xi)]^2. \quad (2.35)$$

Comme  $\Xi$  est centré, son barycentre  $\boldsymbol{\mu}_\xi$  est nul. On peut donc écrire :

$$V = \sum_{i=1}^N (\mathbf{u}^T \boldsymbol{\xi}_i)^2 = \mathbf{u}^T \Xi^T \Xi \mathbf{u} = \mathbf{u}^T C \mathbf{u}. \quad (2.36)$$

La matrice  $C$  est la matrice d'inertie totale, ou covariance, du nuage  $\Xi$ . Comme  $\Xi$  est centrée-réduite, c'est simplement le calcul de la matrice de corrélation :

$$C = \Xi^T \Xi. \quad (2.37)$$

On peut montrer (c'est un résultat classique) que le vecteur  $\mathbf{u}$  qui maximise cette variance est le vecteur propre de  $C$  associé à la plus grande valeur propre. Notons-les respectivement  $\mathbf{u}_1$  et  $\lambda_1$ . A nouveau, dans l'espace orthogonal à  $\mathbf{u}_1$ , l'axe qui maximise la variance est supporté par le vecteur propre de  $C$  associé à la deuxième valeur propre, et ainsi de suite pour les  $n$  axes principaux. Comme  $C$  est symétrique ( $C^T = C$ ), les valeurs propres sont toutes réelles et les vecteurs propres associés sont orthogonaux. D'autre part,  $C$  est définie semi-positive, donc toutes les valeurs propres sont positives ou nulles. La contribution de chaque axe à la variance est le rapport de la valeur propre qui lui est associée à la somme de toutes les valeurs propres. Soit, pour l'axe  $k$  :

$$V_k = \frac{\lambda_k}{\sum_{j=1}^n \lambda_j}. \quad (2.38)$$

Comme on a rangé les axes principaux en fonction des valeurs propres décroissantes, le premier axe est celui qui supporte le plus de variance, donc selon lequel les individus pourront le mieux être discriminés (voir par exemple [82]). En prenant les  $K$  premiers axes principaux comme base de notre espace de projection, la variance reconstruite est donc :

$$V_K = \sum_{k=1}^K V_k = \frac{\sum_{k=1}^K \lambda_k}{\sum_{k=1}^n \lambda_k} \quad (2.39)$$

En général, on choisit  $K$  de façon à avoir une certaine proportion de la variance conservée (par exemple,  $V_K \geq 90\%$ ).

Cette méthode classique demande le calcul de toutes les valeurs propres, ce qui demande la diagonalisation de la matrice  $C$  ( $n \times n$ ) et peut être lourd quand la dimension  $n$  est grande ( $> 500$ ).

### 2.4.2 Equivalents neuronaux

Il existe d'autres méthodes pour effectuer l'ACP. Par exemple, d'intéressants équivalents neuronaux ont été proposés. Tout d'abord, selon Oja [97], les poids<sup>2</sup>  $\mathbf{x}$  d'un simple neurone convergent vers le premier vecteur propre  $\mathbf{u}_1$ , si ce neurone est muni d'une règle de Hebb modifiée (appelée "règle de Oja") :

$$\Delta \mathbf{x} = \alpha a (\boldsymbol{\xi} - a \mathbf{x}), \quad (2.40)$$

avec  $a = \mathbf{x}^T \boldsymbol{\xi}$ , l'activité du neurone, et  $\alpha$  le facteur d'adaptation.

Sanger [108] et Oja [97] ont alors tous deux proposé des réseaux basés sur ce principe, capables de faire de l'ACP. Pour obtenir une réduction de dimension de  $n$  vers  $p$ , les deux types de réseau n'ont besoin que de  $p$  unités (on ne compte pas les entrées comme des unités). Dans le réseau de Sanger, la règle d'apprentissage est, pour le neurone  $i$  :

$$\Delta \mathbf{x}_i = \alpha a_i \left( \boldsymbol{\xi} - \sum_{k=1}^i a_k \mathbf{x}_k \right), \quad (2.41)$$

tandis que dans celui d'Oja, elle est (pour  $p$  unités) :

$$\Delta \mathbf{x}_i = \alpha a_i \left( \boldsymbol{\xi} - \sum_{k=1}^p a_k \mathbf{x}_k \right). \quad (2.42)$$

Ces deux règles se réduisent à celle de Oja (2.40) quand il n'y a qu'une seule unité dans le réseau. Les deux réseaux fournissent en sortie la projection des  $\boldsymbol{\xi}$

---

<sup>2</sup>On n'utilisera pas la notation  $\mathbf{w}$  pour les poids, afin de rester homogène dans notre notation qui réserve le symbole  $\mathbf{x}$  pour les vecteurs-poids dans l'espace d'entrée.

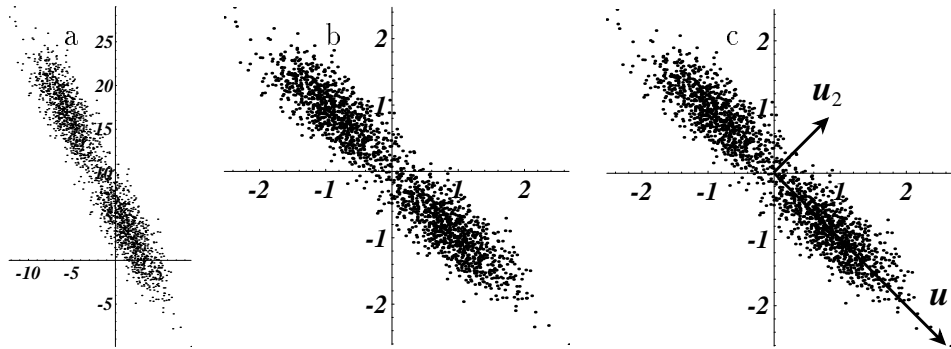


Figure 2.3: Déroulement d'une analyse en composantes principales. (a) Distribution d'entrée. (b) Centrage et réduction de cette distribution, pour obtenir une moyenne nulle, et la même variance selon toutes les coordonnées. (c) Les deux axes principaux, correspondant aux vecteurs propres de la matrice de covariance de la distribution.

dans l'espace des  $p$  premiers axes principaux de la distribution. En fait, la seule différence est qu'avec la méthode proposée par Sanger, les poids  $\mathbf{x}_i$  convergent vers les vecteurs propres triés  $\mathbf{u}_i$  (le premier neurone a pour poids le vecteur propre associé à la plus grande valeur propre), tandis que les poids du réseau de Oja convergent vers n'importe quelle rotation et symétrie des  $\mathbf{u}_i$ . Bien que le résultat soit comparable, il est parfois utile de disposer des vecteurs propres eux-mêmes, pour analyser la nature des dépendances entre les variables et voir celles qui sont prépondérantes, par exemple. En outre, si l'on n'a aucune idée du nombre  $p$  de dimensions nécessaires, le réseau de Sanger tolère un nombre superflu d'unités. *A posteriori*, on peut éliminer les neurones inutiles. Une telle procédure n'est pas possible avec la méthode de Oja, dans laquelle on doit décider *a priori* vers combien de dimensions  $p$  on veut effectuer la projection. En revanche, cette dernière semble plus plausible biologiquement : on ne voit pas bien comment ni pourquoi les axes principaux seraient triés dans le cerveau d'un animal.

### 2.4.3 Exemples

Une illustration des étapes de la méthode est donnée par la figure 2.3 dans un cas très simple où la distribution est en 2 dimensions. La distribution est formée de deux nuages gaussiens. Les deux variables  $x_1$  et  $x_2$  de cette distribution sont manifestement liées. On voit aussi nettement les deux nuages, mais ni selon  $x_1$  ni selon  $x_2$  on ne peut les séparer. Par contre, la projection sur le premier axe principal ( $\mathbf{u}_1$ ) permet aisément de discriminer les deux groupes d'échantillons. Ce premier axe supporte ici environ 90% de la variance.

L'ACP est une technique très largement utilisée. Il existe un très grand nombre d'exemples didactiques pour l'illustrer. Parmi ceux-ci, nous utilisons dans [8] une



Figure 2.4: Projection sur les deux axes principaux de 53 pays en fonction de 6 indices socio-économiques.

analyse géopolitique. L'analyse est faite pour 53 pays du globe d'après les 6 caractéristiques suivantes (la base de données est donc composée de 53 vecteurs à 6 dimensions) :

$x_1$	croissance économique
$x_2$	mortalité infantile
$x_3$	taux d'analphabétisme
$x_4$	taux de scolarisation
$x_5$	produit national brut (PNB)
$x_6$	augmentation du PNB

Former des groupes bien séparés ne paraît pas une tâche simple quand on observe les données brutes. Quand on essaie de représenter les pays sur un plan en fonction de deux variables choisies parmi les  $x_k$ , les résultats ne sont pas probants non plus. Par contre, après traitement par l'ACP, la projection sur les deux premiers axes principaux donne une représentation assez facile à interpréter (figure 2.4). Les 53 pays qui ont été pris en compte<sup>3</sup> forment des groupes assez

clairement marqués. Par exemple, on trouve à gauche les pays du G7 (les 7 nations les plus industrialisées), et, diamétralement opposés, les pays en voie de développement.

Il est intéressant de savoir quelle proportion de la variance totale subsiste après projection (éq. (2.39)). Dans cet exemple, on a 79.5% de la variance dans le plan principal montré ici.

De plus, l'analyse des vecteurs propres trouvés révèle les combinaisons linéaires de facteurs qui sont déterminants.

#### 2.4.4 Limitations

On a vu que l'ACP permettait de faire une réduction de dimension en contrôlant le taux d'information perdue. Malheureusement, comme c'est une opération de *projection linéaire*, seules les dépendances linéaires entre les variables peuvent être révélées.

Dans la figure 2.5(a), on montre une distribution particulière (dite en “fer à cheval”) où l'analyse en composantes principales est incapable de révéler quoi que ce soit. En effet, on s'est arrangé pour obtenir un ellipsoïde d'inertie pour l'ensemble de points qui soit en réalité sphérique. Par conséquent, toutes les valeurs propres de la matrice de covariance sont identiques. Il n'y a pas d'axe principal à proprement parler, chaque axe supporte 1/3 de la variance<sup>4</sup>.

Si l'on s'obstine quand même à tenter une projection sur deux axes, on obtient un plan quelconque (la distribution est estimée par  $N$  individus, et le bruit de la quantification détermine une des solutions parmi l'infinité de systèmes d'axes possibles), par exemple celui de la figure 2.5(a). Le résultat de la projection (figure 2.5(b)) ne révèle pas grand-chose sur la distribution.

Bien que les covariances entre les variables  $x_1$ ,  $x_2$  et  $x_3$  soient nulles (aux erreurs de quantification près, la matrice de covariance est égale à la matrice identité), cela ne veut pas dire que ces variables soient indépendantes ! Simple-ment, cette dépendance n'est pas linéaire et ne peut être révélée par l'ACP.

On pourrait donc souhaiter une sorte de projection non-linéaire, où la distribution serait projetée sur l'abscisse curviligne (la surface gauche sous-tendue par

---

<sup>3</sup>Données datant de 1984.

<sup>4</sup>Comme le même phénomène peut se produire en grande dimension, où alors chaque axe supportera  $1/n$  de la variance, il faut être prudent avec l'interprétation de  $V_K$  dans (2.39). En effet, en 1000 dimensions, on aura l'impression qu'en supprimant quelques axes, on n'aura pas perdu beaucoup d'information. Par exemple, en supprimant 10 axes, on aura encore 99% de la variance, ce qui nous semblera bien. Pourtant chacun de ces 10 axes supportait la même variance que n'importe quel autre. . . C'est pourquoi il est toujours intéressant de faire un petit histogramme des valeurs propres trouvées lors de la procédure de l'ACP, de façon à constater visuellement la prépondérance des axes choisis sur les autres, comme dans la figure 2.6.



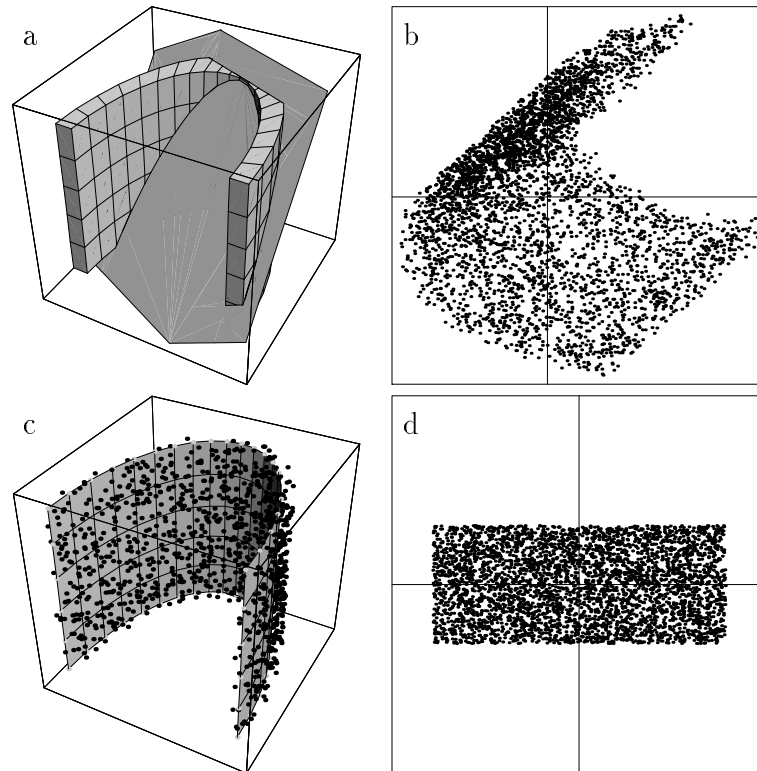


Figure 2.5: Distribution en “fer à cheval”. 1ère ligne : contre-exemple de l’analyse en composantes principales. 2e ligne : Projection non-linéaire “souhaitée”. (a) Distribution et plan principal trouvé par l’ACP (plan quelconque dans ce cas, dépendant uniquement des aléas du tirage des individus de la distribution). (b) Projection de la distribution sur ce plan. (c) Distribution et surface curviligne qui la sous-tend. (d) Projection de la distribution sur cette surface.

les données), comme dans la figure 2.5(c,d). Nous verrons par la suite comment les réseaux auto-organisés peuvent répondre à cette attente.

Reprenons notre exemple de localisation (§ 1.1) dans sa version 2D (la tablette à digitaliser). Comme on collecte en permanence 20 distances, la distribution brute est en 20 dimensions. Cependant, on a vu que comme le problème n’avait que 2 degrés de liberté (le point à localiser se déplace sur un plan) la distribution est en fait une surface pliée dans les 20 dimensions. En effectuant une ACP sur cette distribution, on obtient l’histogramme des valeurs propres montré à la figure 2.6.a.

On déduit des valeurs propres que 97% de la variance peuvent être reconstruits avec les 3 premiers axes principaux. La projection sur ces trois axes est montrée à la figure 2.6.b. En revanche, la projection vers 2 dimensions donnerait une perte de plus de 20% de la variance.

Cette vue en 3 dimensions montre bien qu’il n’y a que deux degrés de liberté



Figure 2.6: ACP de la distribution du problème de localisation (voir texte). (a) Histogramme des valeurs propres. (b) Projection sur les 3 premiers axes principaux.

dans la distribution, mais les dépendances non-linéaires entre les variables ne sont pas réductibles par ACP.

## 2.5 Dimension intrinsèque

### 2.5.1 Notion de variété

On a parlé jusqu'à présent de "dimension brute", par opposition à "dimension intrinsèque" ou encore "nombre de degrés de liberté". Si la notion de dimension brute d'un espace est simple (c'est le nombre  $n$  des composantes d'un vecteur dans cet espace), celle de dimension intrinsèque ou de nombre de degrés de liberté l'est moins. On pourrait prendre comme dimension intrinsèque le rang de la matrice  $X$  des échantillons. Par exemple, en examinant la distribution évoquée à la page 30 (des points sur un hyperplan dans un espace à 1000 dimensions), on trouverait une dimension intrinsèque de 2. Mais cette définition purement linéaire ne nous satisfait pas, car elle ne nous indiquerait pas que la distribution en fer à cheval (figure 2.5) ou celle des capteurs de localisation (§ 1.1) dépendent de deux paramètres. De plus, un seul point bruité (même légèrement) peut changer le rang : il suffit qu'un point s'écarte légèrement de la forme linéaire qui sous-tend tous les autres pour augmenter le rang de 1. Compte tenu des bruits, il y a fort à parier que le rang trouvé sur la distribution en hyperplan dans un espace à 1000 dimensions soit 1000. Dans l'application de localisation, on *sait* que le nombre de degrés de liberté est 2, puisqu'on est parti d'un espace paramétrique à 2 dimensions pour construire nos vecteurs de 20 distances chacun. Si dans ce cas précis on connaît la nature de l'espace paramétrique, il n'en va pas de même pour la plupart des applications, où l'on veut justement retrouver un espace paramétrique.

Comment donc retrouver cette *dimension paramétrique* d'un ensemble de points ?

La notion de dimension paramétrique est étroitement liée à celle de *variété* ou de sous-variété. La notion de sous-variété est une généralisation des surfaces en géométrie différentielle classique. Dans le cas particulier des espaces réels, une définition informelle et intuitive<sup>5</sup> d'une sous-variété  $V$  de dimension  $p$  dans  $\mathbb{R}^n$  est un sous-ensemble de  $\mathbb{R}^n$  obtenu par l'application d'une fonction  $f$  sur un intervalle  $U$  de l'espace  $\mathbb{R}^p$ . En chaque point  $\mathbf{y}$  de  $U$ , la différentielle de  $f$  est de rang  $p$ . Le couple  $(U, f)$  est une *carte différentiable* de la variété, qui est un espace topologique séparé. Deux systèmes de cartes différentiables compatibles qui génèrent le même faisceau de fonctions ne sont que deux façons de définir la variété.

Par exemple, l'enveloppe d'une sphère  $S_n$  centrée de rayon unité dans l'espace  $\mathbb{R}^n$  est l'ensemble des points  $\mathbf{x} = [x_1, \dots, x_n]^T$  de  $\mathbb{R}^n$  liés par la relation :

$$x_1^2 + \dots + x_n^2 = 1 \quad (2.43)$$

Il s'agit d'une sous-variété de dimension  $p = n - 1$ . En effet, soit  $U_p$  la sphère en dimension  $p$  :

$$U_p = \left\{ \mathbf{y} = [y_1, \dots, y_p]^T \mid y_1^2 + \dots + y_p^2 \leq 1 \right\} \quad (2.44)$$

Pour  $p = n - 1$ , on peut définir  $n$  doublets de fonctions  $f_{i+}$  et  $f_{i-}$  :

$$\begin{aligned} f_{i+}(\mathbf{y}) &= \left[ y_1, \dots, y_{i-1}, +\sqrt{1 - \|\mathbf{y}\|^2}, y_{i+1}, \dots, y_p \right]^T \\ f_{i-}(\mathbf{y}) &= \left[ y_1, \dots, y_{i-1}, -\sqrt{1 - \|\mathbf{y}\|^2}, y_{i+1}, \dots, y_p \right]^T. \end{aligned} \quad (2.45)$$

Chaque  $f_{i+}$  est un homéomorphisme de  $U_p$  vers la portion de la sphère  $S_{i+}$  où la  $i$ -ème composante est positive, tandis que  $f_{i-}$  est un homéomorphisme de  $U_p$  vers  $S_{i-}$ . Les différentielles de  $f_{i+}$  et  $f_{i-}$  sont de rang  $p$ . Les  $2n$  cartes différentiables  $(U_p, f_{i+})$  et  $(U_p, f_{i-})$  forment un système de cartes de la sphère  $S_n$ .

En  $n = 3$  dimensions, par exemple, la dimension de cette variété est  $p = 2$ .  $U_2$  est le disque unité dans  $\mathbb{R}^2$ . On obtient 3 couples de cartes  $((U_2, f_{i+}), (U_2, f_{i-}))$  qui définissent la même structure de variété.

## 2.5.2 Dimension fractale

Si la notion de variété permet de fixer les idées sur la nature de la structure que nous voulons retrouver dans une distribution, elle ne donne pas de méthode pour trouver la dimension de cette structure. D'autre part, les variétés sont des objets

<sup>5</sup>Et *imprécise*; mais ça nous permet de transformer un fermé compact de l'espace de compréhension en ouvert de l'espace d'intuition par une application de classe  $\mathcal{C}^\infty \dots$

mathématiques qui ne prennent pas en compte les problèmes de bruits rencontrés dans toute application concrète. Leur formulation semble indiquer cependant que le concept de dimension intrinsèque d'un nuage de points ne peut être qu'une information *locale*. Par exemple, dans  $\mathbb{R}^3$ , une distribution bimodale formée d'une sphère d'un côté et d'un disque de l'autre aura deux dimensions intrinsèques locales différentes (3 pour la sphère, 2 pour le disque). Si les distributions observées avaient la même loi, connue, sur chaque composante, on pourrait se fonder sur notre résultat concernant la norme de vecteurs aléatoires (théorème 2.1) et son extension à la distance entre deux vecteurs aléatoires pour déduire la dimension intrinsèque  $p$  (puisque  $\text{Var}(\|\xi\|) = b + O(1/\sqrt{p})$ ). Malheureusement, cette hypothèse est rarement vérifiée dans la pratique<sup>6</sup>.

On peut néanmoins imaginer un procédé plus simple, qui se base directement sur les propriétés géométriques de la distribution. En reprenant la formule (2.19) du volume d'une sphère, on remarque qu'elle est de la forme :

$$V_n(r) = ar^n \quad (2.46)$$

Ainsi, dans une distribution uniforme (où le nombre de points est proportionnel au volume), on observera un nombre  $N$  de points à l'intérieur d'une sphère de rayon  $r$  qui sera proportionnel à  $r^n$ . Si la distribution n'a localement que  $p$  degrés de liberté, le nombre de points sera en  $r^p$ . En effet, même dans un espace à 1000 dimensions, si la distribution se trouve dans un plan, les points que l'on compte sont situés à l'intersection entre la sphère et le plan, c'est-à-dire un disque. Le nombre de points dénombrés est donc proportionnel à  $r^2$ .

Comme  $N = ar^p$ , on a :

$$\log[N(r)] = \log(a) + p \log(r) \quad (2.47)$$

Le principe de cette méthode simple se résume donc comme suit, pour une distribution finie composée de  $M$  points :

1. Choisir un point  $\xi$  autour duquel on veut mesurer la dimension locale.
2. Pour chaque point de la distribution  $\xi_i \neq \xi$ , calculer la distance  $r_i = \|\xi_i - \xi\|$ .
3. Trier ces distances par ordre croissant, ce qui donne une liste  $\{r_1, r_2, \dots, r_M\}$ , avec  $r_1 \leq r_2 \leq \dots \leq r_M$ . Le nombre  $N$  correspondant à une distance  $r_k$  est égal à l'indice  $k$  de cette distance dans la liste triée.

---

<sup>6</sup>Il n'en demeure pas moins que ça pourrait être un sujet d'investigation; Shepard [111] a déjà remarqué en 1962 que la variance de l'histogramme des distances entre points varie en  $1/p$  et fonde sur cette observation son "Analyse des proximités".

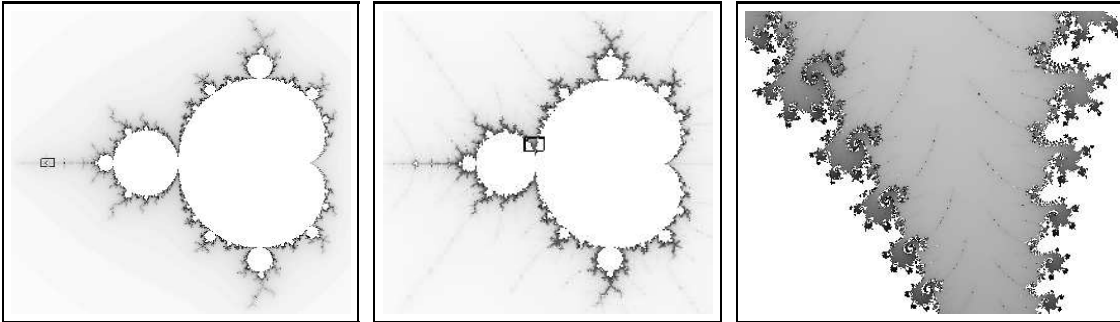


Figure 2.7: Qui pourrait résister, dès lors qu'on parle de fractales, à l'envie de montrer une illustration de l'ensemble de Mandelbrot ? Ici, chaque image est l'agrandissement d'une région de celle qui se trouve à sa gauche.

4. Tracer la caractéristique  $\log(k)$  en fonction de  $\log(r_k)$ .
5. La pente  $p$  trouvée par régression linéaire de cette caractéristique correspond à la dimension locale autour du point  $\xi$  choisi.

Ce type d'approche émane d'un domaine relativement récent : Après s'être demandé quelle était la longueur de la côte de la Bretagne [84], Mandelbrot (voir par exemple [85, 86]) a écrit une nouvelle page de l'histoire des mathématiques, les désormais célèbres *fractales*. Si le monde construit par l'homme est composé d'objets géométriques (parallélépipèdes, cylindres, cônes, ...), la nature fournit des objets si complexes qu'ils ne peuvent pas être décrits par la géométrie classique. Au contraire, leur forme ne trouve ses règles que dans l'irrégularité ou la fragmentation. La côte de la Bretagne justement, la complexité d'une forêt ou d'un arbre, les ramifications d'un éclair, celles des bronches ou la délicate structure d'un flocon de neige sont autant d'exemples d'objets fractals. De tels objets peuvent présenter des propriétés remarquables, comme par exemple l'*auto-similarité* : deux vues à des échelles de grossissement différentes sont similaires (figure 2.7).

Une des grandeurs caractéristiques d'un tel objet est sa dimension fractale, qui n'est généralement pas un nombre entier. Une courbe fractale dans un plan (par exemple la frontière de l'ensemble de Mandelbrot, figure 2.7) a une dimension fractale comprise entre 1 et 2.

Il existe plusieurs définitions de cette dimension fractale, qui se rapprochent plus ou moins du procédé empirique qu'on a décrit ci-dessus, et qui sont liées entre elles [54]. Toutes sont basées sur l'étude de l'évolution, en fonction de l'échelle, d'une propriété mesurable du nuage. Cette propriété est paramétrée par une longueur caractéristique. Prenons par exemple un pavage de l'espace  $\mathbb{R}^n$  en cubes à  $n$  dimensions et de côté  $r$ . Disons que  $N(r)$  est le nombre des cubes qui contiennent

au moins un point. La “*dimension de similarité*” est alors définie par :

$$d_0 = \lim_{r \rightarrow 0} \frac{\log N(r)}{\log(1/r)} \quad (2.48)$$

Une autre définition prend en compte l'évolution de la quantité d'information selon Shannon, lorsqu'on quantifie l'espace par les cubes. C'est la “*dimension d'information*” :

$$d_1 = \lim_{r \rightarrow 0} \frac{I(r)}{\log(1/r)} \quad (2.49)$$

avec  $I(r) = -\sum_{i=1}^{N(r)} p_i(r) \log p_i(r)$  et  $p_i(r)$  la probabilité de la boîte  $i$  (le nombre de points qu'elle contient divisé par  $M$ ).

Une troisième mesure, enfin, est basée sur le nombre  $C(r)$  de couples de points dont l'écart est inférieur à  $r$  ( $C(r) = \text{card} \{ \langle \mathbf{x}_i, \mathbf{x}_j \rangle \mid \|\mathbf{x}_j - \mathbf{x}_i\| < r \}$ ). C'est la “*dimension de corrélation*” :

$$d_2 = \lim_{r \rightarrow 0} \frac{\log C(r)}{\log(r)} \quad (2.50)$$

En fait, toutes ces mesures de dimensions sont des cas particuliers de la “*dimension fractale généralisée*” [54] :

$$d_q = \frac{1}{q-1} \lim_{r \rightarrow 0} \frac{\log \sum_{i=1}^{N(r)} p_i(r)^q}{\log(r)} \quad (2.51)$$

où, comme précédemment,  $N(r)$  est le nombre de boîtes de côté  $r$  nécessaires pour couvrir la distribution, et  $p_i(r)$  la probabilité de chaque boîte.

Dans ces définitions, la notion de “localité” a été transformée en un passage à la limite  $r \rightarrow 0$ . En pratique, comme la distribution (discrète) qu'on étudie possède un nombre fini d'échantillons, cette limite tendrait vers 0 (à partir du moment où l'on n'a plus qu'un point dans chaque boîte, la diminution de  $r$  ne change plus le nombre de boîtes nécessaires). Il faut donc se contenter d'une longueur  $r$  “petite, mais pas trop”, cette notion intuitive étant discutée ci-après.

### 2.5.3 La dimension fractale en pratique

Revenons un instant à la dimension de similarité (dénombrement des boîtes non vides). On voit que cette méthode est plus intéressante que notre procédé initial consistant à compter le nombre de points dans une sphère. En effet, elle dépend surtout de l'*espace couvert* par la distribution (sa variété) et non de la densité de la distribution elle-même, en tout cas tant qu'on a un nombre d'échantillons grand

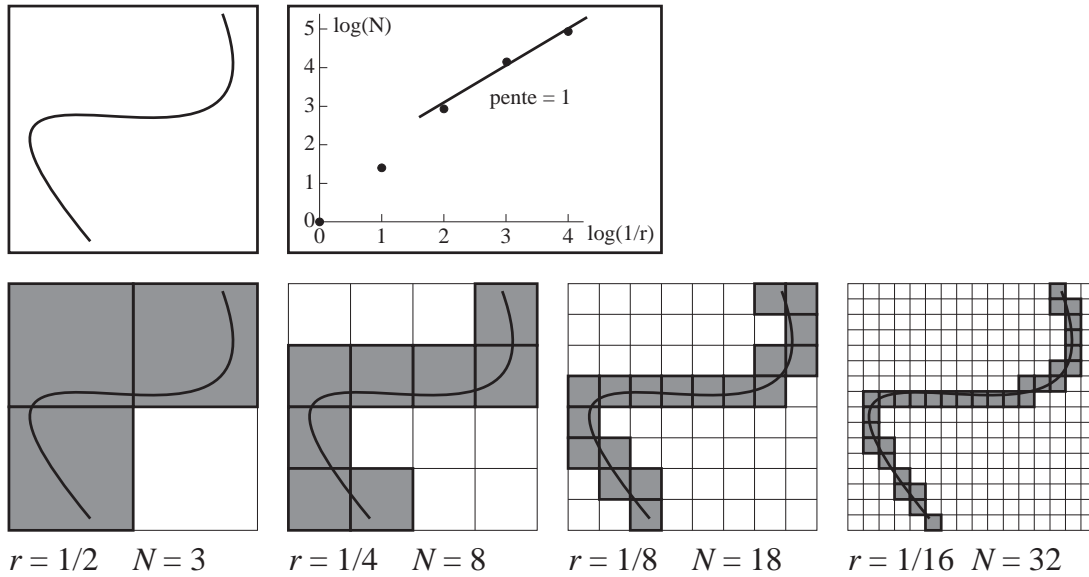


Figure 2.8: Calcul de la dimension fractale d'une courbe par la méthode des boîtes. La représentation en échelle log-log du nombre de boîtes nécessaires pour couvrir la distribution tend vers une pente égale à la dimension intrinsèque de cette distribution.

par rapport au nombre de boîtes. Cette méthode est illustrée dans la figure 2.8 dans le cas d'une distribution sur une ligne courbe.

La mise en œuvre de cette mesure qui consiste à compter des boîtes n'est pas si difficile qu'il y paraît de prime abord. L'algorithme suivant est très satisfaisant du point de vue de la rapidité. L'idée de base est de générer un code unique pour chaque boîte, puis de compter combien de codes différents on a généré avec tous les points de la distribution.

1. Rendre le nuage non négatif par translation, c'est-à-dire soustraire à chaque composante  $k$  des vecteurs  $\xi_i$  la valeur  $\min \xi_k$ . Calculer la longueur  $r_0$  du cube initial (le cube minimal qui englobe complètement la distribution).
2. Calculer une nouvelle longueur  $r$ . Une bonne méthode, qui donnera des échantillons espacés régulièrement sur l'échelle logarithmique, est de prendre une progression géométrique de raison  $q < 1$ .
3. Pour chaque vecteur  $\xi_i$ , assigner un code  $c_i = \{c_{i1}, c_{i2}, \dots, c_{in}\}$  construit de la façon suivante :

$$c_{ik} = \left\lfloor \frac{\xi_{ik}}{r} \right\rfloor \quad (2.52)$$

c'est-à-dire la division entière de la composante  $\xi_{ik}$  par  $r$ . Cette division

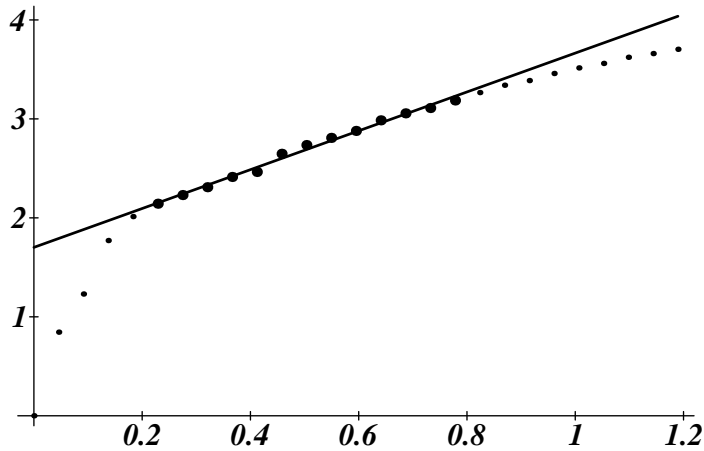


Figure 2.9: Calcul de la dimension fractale de la distribution générée par le problème de localisation avec des capteurs de distance (§ 1.1), dont la dimension brute est 20. La régression effectuée sur un nombre choisi des points de la caractéristique  $\log N(\log r_0/r)$  donne une pente de 2 environ, ce qui est la dimension intrinsèque de la distribution.

entière indique la position, sur l'axe  $k$ , de la boîte à laquelle appartient le point  $\xi_i$ .

4. Compter le nombre  $N(r)$  de codes  $c_i$  différents. Cette opération peut se faire par tri des codes puis dénombrement direct des différences deux à deux, ou mieux, par une méthode de hachage (“*hash-code*”). Imprimer le couple  $(r, N(r))$ .
5. Si  $N(r) < M/2$ , retourner en (2). Expérimentalement,  $N(r) \geq M/2$  semble être une bonne condition d'arrêt de l'algorithme, puisqu'alors il n'y a plus que 2 échantillons en moyenne par boîte; la quantification de la distribution, vue à cette échelle, est trop grossière pour continuer à diminuer  $r$ .
6. Une fois qu'on a obtenu tous les couples  $(r, N(r))$ , on effectue une régression linéaire pour trouver la pente moyenne dans l'échelle log-log, c'est-à-dire la pente moyenne de  $\log N(\log r_0/r)$ .

La charge de calcul, pour cette méthode qui est un prétraitement et ne doit être effectuée qu'une fois comme analyse préalable d'un nuage de données, est relativement modeste : 39.5 secondes sur une Sun Sparc10-41 pour donner 100 couples  $(r, N(r))$ , dans le cas du problème de localisation par 20 capteurs de distance (§ 1.1), qui est une distribution de 10 000 vecteurs à 20 dimensions. La figure 2.9 montre la caractéristique obtenue pour cet exemple. La pente de la droite dessinée vaut 2.05, qui est une bonne approximation du nombre de degrés



de liberté qu'il y a dans ce problème. On obtient un résultat similaire pour la distribution en "fer à cheval".

Il faut faire attention à un point d'importance majeure dans l'utilisation et l'interprétation de ces mesures de dimension fractale : l'échelle d'observation (concrètement, l'intervalle de  $r$  considéré). On voit dans la figure 2.9 que l'on n'a pas considéré tous les points de la caractéristique pour le calcul de la pente. En effet, même si l'on a abandonné la notion de limite et que l'on ne fait pas tendre  $r$  vers 0, mais seulement vers une valeur suffisamment petite, on trouvera la dimension du bruit (celui-ci peut être de plusieurs natures : quantification des données, capteurs défectueux, etc.). Celle-ci correspond, en général, à la dimension brute de l'espace,  $n$ . D'un autre côté, une valeur trop grande de  $r$  risque de "laisser passer" la structure locale de la distribution.

En pratique donc, tout le problème est de déterminer l'échelle à laquelle on veut observer le phénomène, ce qui détermine les valeurs de  $r$  autour desquelles la dimension fractale correspond à la dimension intrinsèque de la structure qu'on veut étudier. Ce problème d'échelle peut être illustré par l'exemple de Mandelbrot de la pelote de ficelle. Vue de très loin, elle ressemble à un point, et sa dimension est 0. Quand on se rapproche, on commence à percevoir sa nature volumique; la dimension passe à 3. En se rapprochant encore, on repère qu'il s'agit d'un fil (une courbe à 1 dimension) replié dans l'espace à 3 dimensions. En s'approchant encore, on voit la nature volumique du fil, qui lui-même est composé de fibres, etc.

En outre, on peut imaginer des processus qui génèrent des distributions dont la dimension locale dépend de l'endroit où l'on se place. C'est le cas de celle de la figure 2.10. L'aspect global de la dimension généralisée ne permet pas de mettre en évidence les propriétés particulières de telles distributions.

Enfin, mentionnons le fait qu'en raison du phénomène d'espace vide (§ 2.2), le nombre de points nécessaires pour estimer avec suffisamment de certitude qu'une structure a une dimension intrinsèque grande peut être exorbitant (de l'ordre du million déjà pour une dimension intrinsèque de 4 seulement; si l'espace de départ est en 100 dimensions — il y a 100 variables mais 4 degrés de liberté —, et avec 32 bits par variable, cela représente 400 MB de données). Il s'agit là d'un problème *irréductible*, au même titre que celui concernant l'estimation *a priori* de l'échelle d'observation précédemment évoqué. Malheureusement, il existe des processus, avec 4 degrés de liberté au moins et un grand nombre de variables, qui ne délivrent qu'un vecteur par heure, par exemple. Dans ce cas, une approche uniquement fondée sur la simple analyse des données n'aboutira pas. Il faut tirer parti au maximum de la connaissance des experts du système.

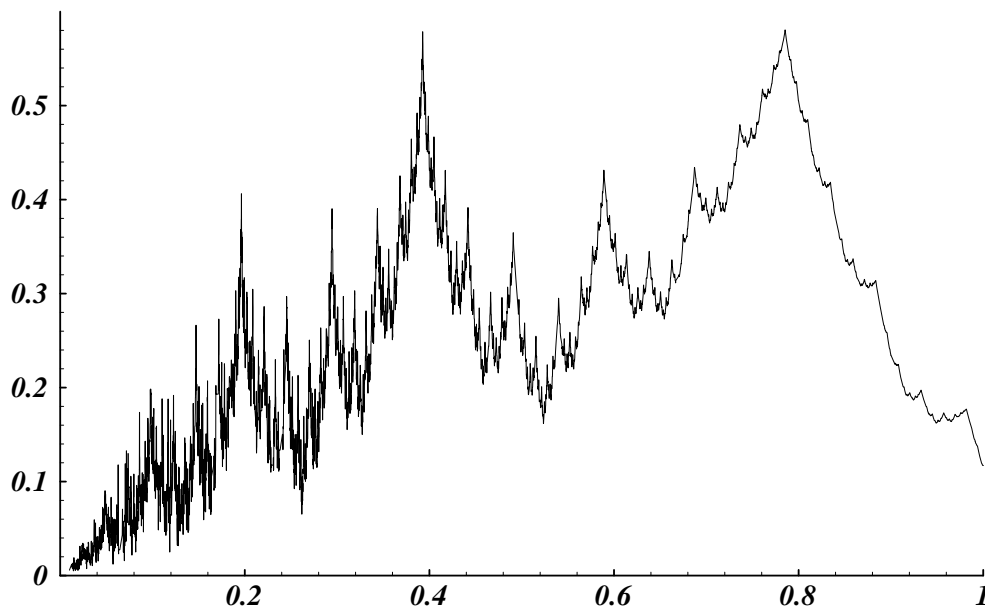


Figure 2.10: Courbe dont la dimension locale varie continûment selon l'abscisse  $t$ . Il s'agit du graphe de la fonction  $t^{3/2} \sum_{k=0}^{\infty} 2^{-kt} \cos(2^k t)$  entre 0 et 1. La dimension locale vaut  $2 - t$  (d'après Tricot [116]). Voir aussi la distribution de la figure 5.1, page 99.

## 2.6 Résumé des problèmes

Nous pouvons résumer les problèmes soulevés jusqu'à présent :

- *Le Bruit.* Quelle est la dispersion des données ?
- *Le Nombre d'échantillons.* Est-il suffisant pour estimer correctement la dimension intrinsèque de la distribution ?
- *L'Échelle.* A quelle échelle faut-il mesurer la dimension intrinsèque ? On a vu qu'une échelle trop grossière "laisait passer" les détails importants, alors qu'une échelle trop fine donnait la dimension du bruit.

Ces problèmes sont liés les uns aux autres, et l'on ne peut pas y apporter de réponse sans une connaissance approfondie du processus qui a généré les données.

# Chapitre 3

## Quantification vectorielle

### 3.1 Généralités

En dehors de la dimension des vecteurs d'une distribution, l'autre axe sur lequel on peut agir pour réduire la quantité de données est le nombre d'échantillons de cette distribution : au lieu de les garder tous, on choisit quelques représentants pertinents (figure 3.1(a)). On parle alors de *quantification*. Au lieu de quantifier chaque axe indépendamment les uns des autres (ce qui est une quantification scalaire, peu intéressante en pratique), on effectue généralement une quantification vectorielle, qui permet de mieux se concentrer sur les zones intéressantes de la distribution, comme le montre la figure 3.1.

Plusieurs domaines utilisent la quantification vectorielle, mais les plus actifs (qui ont le plus contribué aux développements de la méthode) sont sans doute le traitement de signal (compression de parole, d'image pour le stockage et la transmission) et l'analyse de donnée par *regroupement* (“*clustering*”). D'une façon formelle, la quantification vectorielle consiste à représenter une distribution  $\Xi$  (finie ou non) dans un espace  $\mathbb{R}^n$  de dimension  $n$  à l'aide d'un sous-ensemble fini de  $N$  éléments  $\mathbf{x}_i$  (les *vecteurs-codes*, ou *prototypes*) dans cet espace. Ce sous-ensemble, qu'on représente généralement par une matrice  $X$  ( $N \times n$ ), est appelé *dictionnaire* (“*codebook*”). Comme dans les autres chapitres, nous prendrons comme convention que les lignes de la matrice  $X$  sont les vecteurs-codes  $\mathbf{x}_i = [x_{i1}, \dots, x_{in}]$  :

$$X = \begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_N \end{bmatrix} \quad (3.1)$$

Le codage envisagé (par exemple en compression de signal) consiste à remplacer chaque point  $\boldsymbol{\xi}$  de la distribution par le numéro d'un vecteur-code. Ceci permet

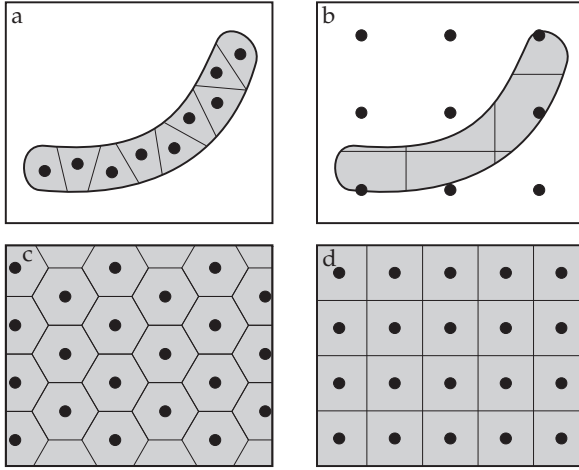


Figure 3.1: La quantification vectorielle (QV) (à gauche : a et c) comparée à la quantification scalaire (à droite : b et d). Du point de vue de la distorsion, dans le cas d'une distribution structurée (en haut) l'avantage de la QV est évident, mais même dans le cas d'une distribution uniforme (en bas), la QV permet une erreur de reconstruction plus faible en moyenne, à cause de l'arrangement hexagonal qu'adoptent les vecteurs-codes.

une réduction de données qui peut être considérable (on transforme un vecteur de  $n$  valeurs réelles en un entier indiquant le numéro du code), au détriment de la qualité de représentation qui subit une distorsion moyenne  $D(X)$  :

$$D(X) = E[d(\boldsymbol{\xi}, \mathbf{x}_{i^*})] \quad (3.2)$$

où  $E(r)$  est l'espérance (la moyenne) de  $r$ ,  $\mathbf{x}_{i^*}$  est le vecteur-code qui représente  $\boldsymbol{\xi}$ , et  $d(\mathbf{a}, \mathbf{b})$  une fonction de distorsion entre les vecteurs  $\mathbf{a}$  et  $\mathbf{b}$ , par exemple :

$$d(\mathbf{a}, \mathbf{b}) = \|\mathbf{a} - \mathbf{b}\|^2 = (\mathbf{a} - \mathbf{b})^T (\mathbf{a} - \mathbf{b}) \quad (3.3)$$

La détermination du “*codebook*” a pour objectif de réduire la distorsion moyenne  $D(X)$ . Un quantifieur vectoriel est dit optimal si aucun autre quantifieur (sous les mêmes conditions de nombre fixé de vecteurs-codes et de distribution) ne donne une distorsion moyenne plus faible. Deux conditions nécessaires pour obtenir un quantifieur optimal sont :

- *Condition du plus proche voisin* : Le vecteur  $\mathbf{x}_{i^*}$  qui représente un  $\boldsymbol{\xi}$  doit être le vecteur-code  $\mathbf{x}_i$  le plus proche, au sens de la mesure  $d(\boldsymbol{\xi}, \mathbf{x}_i)$  ci-dessus.
- *Condition des centroïdes* : Chaque vecteur-code  $\mathbf{x}_i$  doit être le centre de gravité de tous les  $\boldsymbol{\xi}$  qui sont représentés par lui, c'est-à-dire que  $\mathbf{x}_i = E[\{\boldsymbol{\xi} \mid \mathbf{x}_{i^*}(\boldsymbol{\xi}) = \mathbf{x}_i\}]$ .

Dans le cas d'une distorsion définie par la distance euclidienne (éq. (3.3)), la partition de l'espace  $\mathbb{R}^n$  en  $N$  régions  $S_i$ , chacune représentée par un vecteur-code  $\mathbf{x}_i$  :

$$\begin{aligned} S_i &= \{ \boldsymbol{\xi} \mid \mathbf{x}_{i^*}(\boldsymbol{\xi}) = \mathbf{x}_i \} \\ &= \{ \boldsymbol{\xi} \mid \|\boldsymbol{\xi} - \mathbf{x}_i\| \leq \|\boldsymbol{\xi} - \mathbf{x}_j\| \forall j \}, \end{aligned} \quad (3.4)$$

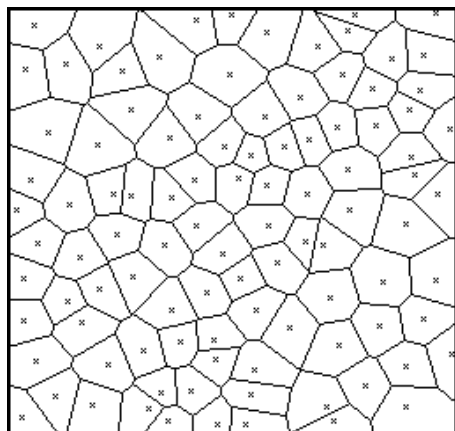


Figure 3.2: Pavage de Voronoï pour 100 points. Chaque segment du pavage est la médiatrice de deux points proches.

est appelée *pavage de Voronoï*. Un exemple de pavage de Voronoï d'un carré en fonction de 100 vecteurs-codes aléatoires est donné dans la figure 3.2.

Grâce à cette définition, on peut réécrire (3.2) plus précisément, ce qui fait apparaître la densité de probabilité  $P(\boldsymbol{\xi})$  :

$$D(X) = \sum_i \int_{\boldsymbol{\xi} \in S_i} \|\boldsymbol{\xi} - \mathbf{x}_i\|^2 P(\boldsymbol{\xi}) d\boldsymbol{\xi} \quad (3.5)$$

## 3.2 Algorithme de Lloyd généralisé (GLA)

L'algorithme de Lloyd généralisé [81] n'est autre qu'une mise en œuvre directe des conditions du plus proche voisin et des centroïdes. En effet, à chaque itération, on calcule la partition de la distribution en *régions de dominance* selon la condition du plus proche voisin (éq. (3.4)), puis on applique la condition des centroïdes pour adapter les vecteurs-codes :

$$\mathbf{x}_i \leftarrow E(S_i) \quad (3.6)$$

avec  $E(S_i)$  la moyenne (le centre de gravité) de  $S_i$ .

Lors de l'application de (3.6), si l'on tombe sur une région  $S_i = \emptyset$  (parce que le vecteur-code  $\mathbf{x}_i$  correspondant n'a représenté aucun  $\boldsymbol{\xi}$  de la distribution), on réinitialise le  $\mathbf{x}_i$  incriminé à n'importe quel vecteur  $\boldsymbol{\xi}$  de la base.

La distorsion euclidienne quadratique  $D(X) = \sum_i \int_{\boldsymbol{\xi} \in S_i} \|\boldsymbol{\xi} - \mathbf{x}_i\|^2 P(\boldsymbol{\xi}) d\boldsymbol{\xi}$  de cet algorithme décroît de façon monotone, et donc tombe souvent dans un minimum local puisque  $D(X)$  n'est généralement pas convexe (voir [122]).

### 3.3 Apprentissage compétitif (CL)

L'équivalent neuronal "en ligne" du GLA est généralement appelé *apprentissage compétitif* ("Competitive Learning" — CL) [1]. Dans cet algorithme, chaque itération consiste à présenter un échantillon de la base à un réseau compétitif à une couche dont le rôle est de trouver un "gagnant" et à rapprocher les poids de ce gagnant en direction du vecteur présenté.

La recherche du gagnant est faite selon la condition du plus proche voisin :

$$\begin{aligned} i^* &= i \mid \|\boldsymbol{\xi} - \mathbf{x}_i\| \leq \|\boldsymbol{\xi} - \mathbf{x}_j\| \forall j \\ i^* &= i \mid \|\boldsymbol{\xi} - \mathbf{x}_i\|^2 \leq \|\boldsymbol{\xi} - \mathbf{x}_j\|^2 \forall j. \end{aligned} \quad (3.7)$$

(la deuxième forme est plus rapide à calculer, puisque  $\|\mathbf{a}\|^2 = \mathbf{a}^T \mathbf{a}$ ).

L'adaptation de son centroïde est un *déplacement* vers l'échantillon :

$$\mathbf{x}_{i^*} \leftarrow \mathbf{x}_{i^*} + \alpha(t)(\boldsymbol{\xi} - \mathbf{x}_{i^*}) \quad (3.8)$$

avec  $\alpha(t)$  une fonction décroissante du temps (du nombre d'itérations). A l'équilibre, la moyenne du vecteur-poids de chaque neurone ne bouge plus s'il est au centre de sa région de dominance, ce qui correspond à la condition des centroïdes.

Dans sa version neuronale "pure", la recherche du gagnant est effectuée par la relaxation d'une couche d'inhibitions/excitations latérale récurrente (ILR) sur les activités des neurones. Plusieurs variantes ont été proposées pour faire émerger le maximum d'activité et désigner ainsi le neurone gagnant dont on va ensuite adapter les poids [120, 118, 66, 79]. Dans tous les cas, les connexions de cette couche entre les neurones sont excitatrices à courte distance et inhibitrices à longue distance. S'il est intéressant d'étudier cette question du point de vue académique (pour trouver des parallèles dans la biologie) et technologique (pour localiser le traitement en vue de son intégration sur circuit), la façon dont on effectue cette recherche n'affecte normalement pas les résultats obtenus. Un cas particulier, toutefois, qui sera détaillé au chapitre 4, survient quand la couche ILR est paramétrée de façon à faire apparaître une *bulle d'activité*, et non le seul maximum. On obtient alors un algorithme d'auto-organisation, plus riche que la simple quantification comme on le verra.

Le CL fournit des résultats comparables à ceux du GLA, à l'exception notable toutefois de produire parfois des *unités mortes* (ce qui n'arrive pas avec le GLA, puisque le cas y est explicitement traité). Celles-ci ont des vecteurs qui restent en dehors de la distribution (elles ne gagnent jamais) et ne participent donc pas à l'apprentissage.

### 3.4 Mécanismes de fatigue ou “conscience”

Pour éviter les unités mortes, il a été proposé à divers endroits [1, 35, 29] de mettre en œuvre un principe permettant aux unités rarement ou jamais gagnantes d’être favorisées dans le mécanisme de sélection.

Dans le “*frequency sensitive competitive learning*” (FSCL) [1], l’équation (3.7) est remplacée par :

$$i^* = i \mid n_i \|\boldsymbol{\xi} - \mathbf{x}_i\|^2 \leq n_j \|\boldsymbol{\xi} - \mathbf{x}_j\|^2 \forall j, \quad (3.9)$$

avec  $n_i$  le nombre de fois où le neurone  $i$  a déjà gagné. Quand  $n_i$  est faible par rapport aux autres, l’élection du neurone  $i$  est facilitée. Au bout d’un certain temps, même les unités mortes finissent par gagner et sont donc rapprochées de la distribution. On peut voir dans la figure 3.3(b) que le pavage par régions de dominance des neurones n’est plus un pavage de Voronoï. Au lieu de séparatrices qui sont des médiatrices (droites), on a des séparatrices en segments de cercles<sup>1</sup>. Les régions ne sont plus des polygones convexes : l’effet de  $n_i$  est de former des excroissances des régions de dominance vides ou presque *vers* la distribution. Si  $\alpha$  diminue suffisamment lentement, l’algorithme tend vers un état où tous les  $N$  neurones sont gagnants  $1/N$ ème du temps en moyenne. Non seulement les unités mortes sont évitées, mais les distributions de densités non uniformes sont quantifiées de telle façon que la répartition des vecteurs-poids suive la même densité, ce qui est parfois considéré comme un avantage (on verra plus loin — § 3.7 — qu’on peut parfois chercher à éviter cela).

Le mécanisme de conscience de De Sieno [35] suit à peu près la même idée, sauf qu’au lieu de pondérer les distances, on leur soustrait une valeur (un biais). L’équation (3.7) devient :

$$i^* = i \mid \|\boldsymbol{\xi} - \mathbf{x}_i\|^2 - C \left( \frac{1}{N} - q_i \right) \leq \|\boldsymbol{\xi} - \mathbf{x}_j\|^2 - C \left( \frac{1}{N} - q_j \right) \forall j, \quad (3.10)$$

avec  $C$  une constante fixée au départ et  $q_i$  la fraction de temps où le neurone  $i$  est celui qui a le vecteur-poids le plus proche de l’entrée (c’est-à-dire où il serait gagnant dans un simple CL). Cette fraction de temps est estimée par un filtre passe-bas sur la variable  $z_i$ , qui vaut 1 pour le neurone le plus proche, et 0 pour les autres :

$$\begin{aligned} q_i &\leftarrow q_i + \frac{1}{\tau}(z_i - q_i) \\ z_i &= \begin{cases} 1 & \text{si } i = i^*_{\text{CL}} \\ 0 & \text{sinon} \end{cases}, \end{aligned} \quad (3.11)$$

<sup>1</sup>Par exemple, pour deux unités dont les poids sont respectivement  $\mathbf{x}_1 = [0, 0]^T$  et  $\mathbf{x}_2 = [1, 0]^T$ , avec  $n_2 = (1 + \gamma)n_1$ , la séparatrice est le cercle de centre  $[\frac{1+\gamma}{\gamma}, 0]^T$  et de rayon  $\frac{\sqrt{1+\gamma}}{\gamma}$ ; quand  $\gamma$  tend vers 0, le cercle dégénère en une droite verticale d’abscisse  $\frac{1}{2}$ .

avec  $i^*_{\text{CL}}$  l'indice du gagnant euclidien (celui dont le vecteur-poids est le plus proche du vecteur présenté, sans tenir compte du principe de conscience). Le pavage par régions de dominance des neurones, bien que n'étant pas non plus un pavage de Voronoï, reste constitué de polygones convexes. Au lieu d'avoir des séparatrices qui sont des médiatrices (qui passent entre deux points), la droite est déportée vers le neurone qui a le plus grand  $q_i$  (d'un facteur  $\frac{1}{2}(q_i - q_j)$  de la distance qui sépare  $\mathbf{x}_i$  et  $\mathbf{x}_j$ , pour être précis) (fig. 3.3(c)).

Remarquons dans (3.10) que la constante  $C$  a la même unité que  $\xi$ ; ce n'est pas une valeur adimensionnelle. Il faut donc soit choisir  $C$  en fonction de la variance de  $\xi$ , soit toujours centrer-réduire les  $\xi$  (comme pour une ACP), ce qui peut être assez gênant.

Dans [29], ce qu'on a appelé "fatigue" est une *condition de validation* du gagnant en fonction de la fréquence à laquelle il a déjà gagné : un neurone qui a beaucoup gagné récemment est "fatigué" et ne peut momentanément plus gagner (pendant une certaine période, dite de récupération ou de repos). L'idée pourrait avoir une certaine plausibilité biologique, si l'on considère des neurones individuellement incapables d'avoir trop souvent un haut degré d'activité. Un tel phénomène peut être expliqué par le mécanisme d'*accommodation* et par la présence de connections inhibitrices re-bouclées [83]. Si l'on fait un peu de "science-fiction" biologique, comme dit Poggio, notons par le *potentiel*  $p_i \in [0, 1]$  la quantité de neurotransmetteur disponible pour chaque neurone  $i$ . Pour être éligible, un neurone doit avoir un potentiel supérieur à un seuil défini à l'avance :  $p_i \geq p_{\min}$ . Quand il gagne, un neurone utilise  $p_{\min}$  de son potentiel, qui en est diminué d'autant. En continu, les neurones sont tous rechargés de  $1/N$  (avec  $N$  le nombre de neurones) à chaque itération. Tous les potentiels sont toujours tronqués à 1. Plus prosaïquement, c'est un peu le modèle d'une chasse d'eau dont, chaque fois qu'on tire la chaîne, une proportion  $p_{\min}$  du contenu est utilisée. On a donc la séquence d'opérations :

$$\begin{aligned} S &= \{i \mid p_i \geq p_{\min}\} \\ i^* &= i \in S \mid \|\xi - \mathbf{x}_i\| \leq \|\xi - \mathbf{x}_j\| \forall j \in S \\ p_{i^*} &\leftarrow p_{i^*} - p_{\min} \\ \forall i \in \{1, \dots, N\}, p_i &\leftarrow \min(1, p_i + 1/N) \end{aligned} \tag{3.12}$$

avec  $S$  l'ensemble des unités éligibles.

Dans cette méthode, le pavage reste un pavage de Voronoï, mais réalisé sur l'ensemble des vecteurs-poids des unités qui ne sont pas trop fatiguées. Ainsi, les neurones qui gagnent trop souvent sont momentanément retirés du processus, pour être remis plus tard quand ils sont "reposés" (fig. 3.3(d)).

Avec  $p_{\min} = 0$ , on obtient un CL classique, tandis qu'avec  $p_{\min} = 1$ , on a un processus dégénéré où tous les neurones sont élus à tour de rôle, sans aucune



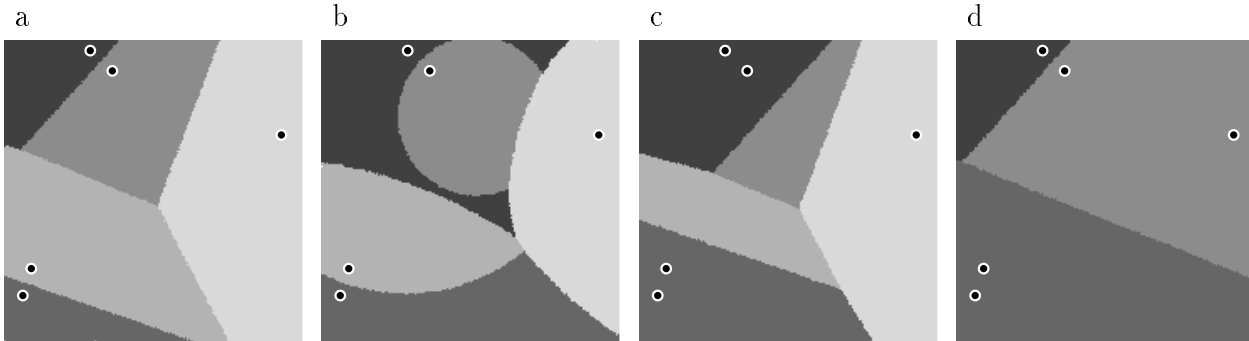


Figure 3.3: Pavages en régions de dominances des neurones en fonction du type d'algorithme. (a) CL. (b) FSCL. (c) CL avec conscience. (d) CL avec fatigue. (Voir texte).

considération pour le vecteur présenté. On peut envisager de faire varier  $p_{\min}$  au cours du temps, en partant de 1 et en diminuant jusqu'à 0, mais les simulations nous ont montré qu'il suffisait de fixer  $p_{\min}$  à une constante, par exemple 0.75, pour obtenir d'excellents résultats (convergence bonne et rapide).

On peut imaginer encore bien d'autres procédés (par exemple des méthodes stochastiques, dans lesquelles la *probabilité* de gagner dépend de la distance et de la fréquence des gains passés) pour éviter les unités mortes en les ramenant dans la distribution. Il existe aussi quelques méthodes un peu “simplistes”, comme initialiser les vecteurs-poids avec des échantillons de la distribution pour être sûr que toutes les unités seront dans cette dernière, mais qui ne donnent pas de très bons résultats de quantification (avec une distribution multi-modale, une fois les vecteurs-poids placés dans un mode par l'initialisation, ils n'en changent plus; la probabilité *a priori* de tous les modes n'aura alors été estimée qu'à l'aide des  $N$  premiers échantillons tirés pour l'initialisation).

### 3.5 Méthodes “winner-take-most”

Jusqu'à présent, on n'a considéré que les méthodes de type “*winner-take-all*”, où seul le gagnant était adapté à chaque itération. Ces algorithmes ont des vitesses de convergence relativement faibles, surtout lorsque le nombre d'unités  $N$  est grand, puisque chaque échantillon présenté ne cause l'adaptation que d'une unité. Il apparaît donc souhaitable de disposer d'algorithmes où plus d'une unité est adaptée à chaque itération. Le problème est de définir avec soin la fonction  $G(i, \xi)$  qui intervient pour chaque unité dans la règle d'adaptation :

$$\mathbf{x}_i \leftarrow \mathbf{x}_i + \alpha(t)G(i, \xi)(\xi - \mathbf{x}_i) \quad (3.13)$$

Le gagnant est déplacé vers le vecteur d'entrée, mais les autres unités aussi, avec toutefois un gain plus faible, ce qui amène au concept d'algorithmes de type

“winner-take-most”.

Hélas ! Comme quelques essais peuvent rapidement nous en convaincre, on ne peut pas simplement prendre  $G(i, \boldsymbol{\xi})$  comme une fonction décroissant avec la distance, sous peine d’assister à une agglomération des unités vers un seul point (la moyenne géométrique de la distribution) à une vitesse surprenante. En effet, il faut conserver à l’algorithme un aspect compétitif pour permettre aux différents neurones de se “spécialiser” dans une zone qui leur est propre. En l’absence d’une telle compétition (ou *inhibition mutuelle*), c’est-à-dire si  $G(i, \boldsymbol{\xi})$  est une fonction continue de la distance entre  $\boldsymbol{x}_i$  et  $\boldsymbol{\xi}$ , deux unités “collées” n’ont aucune chance de se séparer. Il apparaît donc souhaitable de *découpler* d’une quelconque manière le facteur d’adaptation  $G(i, \boldsymbol{\xi})$  de la distance euclidienne.

On va voir comment quelques algorithmes connus mettent en œuvre cette compétition<sup>2</sup> et arrivent à adapter toutes les unités à chaque itération sans que tous les vecteurs-poids ne convergent vers un point.

### 3.5.1 Relaxation stochastique (SRS)

D’une façon générale, la quantification vectorielle est un problème d’optimisation (voir par exemple [99]). Dans le cas d’un ensemble d’apprentissage discret (il y a un nombre fini d’échantillons), Yair, Zeger *et al.* [122, 123] remarquent que les solutions sont aussi un ensemble discret :

Soit  $\Xi = [\boldsymbol{\xi}_1, \dots, \boldsymbol{\xi}_M]^T$ , l’ensemble d’apprentissage composé de  $M$  échantillons (vecteurs  $\boldsymbol{\xi}_m$  à  $n$  dimensions). La quantification de cet ensemble par  $N$  vecteurs-codes, c’est-à-dire sa partition en  $N$  classes distinctes, peut se noter en associant à chaque échantillon  $\boldsymbol{\xi}_m$  le numéro de la classe  $s_m \in \{1, \dots, N\}$ . La partition de tout l’ensemble est complètement décrite par le vecteur d’état à  $M$  composantes entières  $\boldsymbol{s} = [s_1, \dots, s_M]^T$ . Si l’on considère que la condition des centroïdes tient toujours, le “codebook” est complètement défini par  $\boldsymbol{s}$ , puisque chaque vecteur  $\boldsymbol{x}_i$  est la moyenne des  $\boldsymbol{\xi}_m$  dont la classe est  $s_m = i$ . Une distorsion (selon éq. (3.2))  $D(\boldsymbol{s})$  peut être associée à chaque état  $\boldsymbol{s}$ .

Une façon générale de trouver un *minimum global* à une fonction de coût  $D(\boldsymbol{s})$  non-convexe définie sur un ensemble fini d’états est le “recuit simulé”, méthode introduite par Kirkpatrick *et al.* [64] et qui s’inspire de la mécanique statistique (le terme de “recuit” est emprunté à la métallurgie, où il désigne l’opération de réchauffage d’un alliage qui a subi un trempage dur, afin d’améliorer ses qualités mécaniques). Le principe est une procédure stochastique permettant de faire décroître  $D(\boldsymbol{s})$  d’une façon non-monotone dans le but d’éviter les minima locaux.

<sup>2</sup>Le cas du CL est un exemple extrême de compétition :  $G(i, \boldsymbol{\xi})$  vaut 1 si  $i = i^*$  et 0 sinon; mais comme déjà dit, c’est un algorithme “winner-take-all” et non “winner-take-most”.

Généralement, le recuit simulé est mis en œuvre à l’aide de l’algorithme de Metropolis : A chaque itération, un nouvel état  $\mathbf{s}$  est tiré au hasard et va être accepté en fonction de la modification  $\Delta D$  de la fonction de coût qu’il entraîne. L’état est accepté s’il fait diminuer le coût ( $\Delta D < 0$ ). En outre, il peut aussi être accepté, même quand  $\Delta D \geq 0$ , selon une probabilité  $p = \exp(-\Delta D/T)$  qui dépend du paramètre  $T$ , appelé *température* par référence aux systèmes physiques. Pour une température  $T$  constante, ce processus de transition est une chaîne de Markov [23] qui converge en loi vers la loi stationnaire :

$$P(\mathbf{s}) = \frac{1}{Z} \exp(-D(\mathbf{s})/T) \quad (3.14)$$

où  $Z$  est un facteur de normalisation tel que la somme des probabilités de tous les états possibles donne bien 1 (pour voir les relations qui existent entre  $Z$ , l’énergie libre et l’entropie, voir [57]). Cette loi est connue sous le nom de *distribution de Boltzmann-Gibbs* des états  $\mathbf{s}$ .

Lorsque la température n’est pas fixe, mais qu’elle décroît suffisamment lentement, il a été montré qu’une telle procédure évite les minima locaux et converge vers un minimum global de  $D(\mathbf{s})$ . Plus précisément, il existe une constante  $C_0$  telle que, si :

$$\lim_{t \rightarrow \infty} T(t) \ln(t) \leq C_0 \quad (3.15)$$

et que  $T(t)$  est décroissante, alors la répartition des états converge en loi vers la loi uniforme sur tous les minima globaux de  $D(\mathbf{s})$ . En pratique, on n’utilise pas une décroissance logarithmique  $C/\ln(t)$ , mais plutôt une décroissance exponentielle  $T(t) = a^t$  avec  $a$  faiblement inférieur à 1 [23].

Une variante de mise en œuvre, mieux adaptée pour les problèmes comme celui-ci où le vecteur d’état n’est pas binaire mais multivalué, est connue sous le nom d’*échantillonneur de Gibbs*. Au lieu de proposer un nouvel état au hasard et de l’accepter conditionnellement, un nouvel état est tiré selon la distribution de Gibbs et est toujours accepté. Alors que l’algorithme de Metropolis peut proposer beaucoup d’états qui seront rejetés (surtout aux basses températures), l’échantillonneur de Gibbs va toujours prendre un nouvel état. Pour parvenir à faire le tirage selon cette distribution, on ne procède au changement que d’une composante du vecteur d’état, et on calcule la distribution conditionnelle de cette composante *sachant les autres inchangées*. Considérons le changement de la composante  $s_m$  de  $\mathbf{s}$ , qui va passer, disons, de sa valeur actuelle  $j$  à une nouvelle valeur  $i$  tirée au hasard ( $i, j \in \{1, \dots, N\}$ ); on choisit donc désormais de représenter  $\xi_m$  par  $\mathbf{x}_i$  et non plus par  $\mathbf{x}_j$ ). Le nouvel état proposé est ainsi  $\mathbf{s} = [s_1, \dots, s_m = i, \dots, s_M]^T$ , toutes les autres composantes  $s_{k \neq m}$  restant inchangées. Pour rester dans le cadre d’une distribution de Boltzmann-Gibbs des états, Yair *et al.* montrent qu’il suffit

de choisir  $i$  suivant la distribution conditionnelle :

$$P_m(i) = P(s_m = i \mid s_k \forall k \neq m) = \frac{\exp(-\|\boldsymbol{\xi}_m - \mathbf{x}_i\|/T)}{\sum_{j=1}^N \exp(-\|\boldsymbol{\xi}_m - \mathbf{x}_j\|/T)} \quad (3.16)$$

qui s'exprime en fonction du “*codebook*”, et dont on remarque en passant qu'elle est aussi du type Boltzmann-Gibbs.

Lorsque la température est infinie,  $P_m(i)$  vaut  $1/N$  quel que soit  $i$  : tous les neurones ont la même chance de gagner. A l'inverse, quand  $T \rightarrow 0$ , la distribution tend vers une fonction delta autour du gagnant euclidien (1 pour celui-ci et 0 pour les autres), c'est-à-dire que l'algorithme tend vers un simple CL.

Une itération de l'algorithme proposé par Yair *et al.* , appelé “*Stochastic Relaxation Scheme*” (SRS), se résume donc à quatre étapes :

1. Choisir aléatoirement un vecteur  $\boldsymbol{\xi}_m$  de la base  $\Xi$ . Sa partition actuelle est  $s_m = j$ .
2. Calculer les  $N$  valeurs de  $P_m(i)$  selon (3.16), et tirer au hasard un indice  $i = i^*$  selon cette distribution. Le vecteur correspondant  $\mathbf{x}_{i^*}$  est appelé “gagnant stochastique”, tandis que le vecteur  $\mathbf{x}_j$ , qui représentait précédemment  $\boldsymbol{\xi}_m$ , est appelé le “perdant stochastique”.
3. Rapprocher le gagnant de  $\boldsymbol{\xi}_m$ , et éloigner le perdant, selon :

$$\begin{aligned} \mathbf{x}_{i^*} &\leftarrow \mathbf{x}_{i^*} + \frac{1}{N_{i^*} + 1}(\boldsymbol{\xi}_m - \mathbf{x}_{i^*}) \\ \mathbf{x}_j &\leftarrow \mathbf{x}_j - \frac{1}{N_j}(\boldsymbol{\xi}_m - \mathbf{x}_j) \end{aligned} \quad (3.17)$$

avec  $N_i$  le nombre de vecteurs représentés par  $\mathbf{x}_i$  :  $N_i = \text{card} \{s_k \mid s_k = i\}$ .

4. Adapter la température  $T$ .

Si cet algorithme n'est pas encore tout à fait un “*winner-take-most*” (seules deux unités sont adaptées à chaque itération), il permet d'introduire le “*soft competition scheme*” qu'on va voir au paragraphe suivant.

### 3.5.2 “Soft Competition Scheme” (SCS)

Toujours dans [122], les auteurs proposent une version déterministe du SRS, le “*Soft Competition Scheme*” (SCS). Au lieu du principe stochastique où l'on choisit selon la distribution conditionnelle (3.16) l'indice d'un gagnant et d'un perdant,

dans le SCS *tous* les vecteurs-codes sont adaptés à chaque itération selon (3.13), en prenant pour  $G(i, \boldsymbol{\xi})$  la valeur de cette distribution conditionnelle. Ainsi,

$$G(i, \boldsymbol{\xi}) = P_m(i) = \frac{\exp(-\|\boldsymbol{\xi} - \mathbf{x}_i\|/T)}{\sum_{j=1}^N \exp(-\|\boldsymbol{\xi} - \mathbf{x}_j\|/T)}. \quad (3.18)$$

De plus, pour favoriser les unités qui n’ont pas bougé depuis longtemps (pour éviter les unités mortes), chaque unité possède son propre facteur de gain :

$$\alpha_i(t) = \left(1 + \sum_{k=1}^{t-1} G_k(i)\right)^{-1}, \quad (3.19)$$

où  $k$  représente toutes les itérations précédentes ( $k \in \{1, \dots, t-1\}$ ) et  $G_k(i)$  représente le gain de l’unité  $i$  à l’itération  $k$ . Ce mécanisme est une généralisation de la méthode employée par le FSCL (§ 3.4) pour éviter les unités mortes au cas où plusieurs unités sont adaptées à chaque itération.

L’équation (3.18) montre comment une sorte de compétition plus ou moins franche est instaurée entre les unités. Pour une température élevée, la compétition est faible voire inexistante : tous les  $G(i)$  sont à peu près identiques (ils tendent vers  $1/N$  quand  $T \rightarrow \infty$ ). Par contre, quand  $T \rightarrow 0$ ,  $G(i)$  vaut 1 pour le seul gagnant euclidien et 0 pour les autres.

Malheureusement, sauf quand  $T \rightarrow 0$ ,  $G(i)$  est une fonction continue avec la distance, et deux unités qui se trouvent superposées ou presque par les hasards de la simulation (ou de l’initialisation) sont impossibles à séparer. De plus, l’algorithme aux températures élevées tend à agglutiner toutes les unités à la moyenne géométrique de la distribution. Ce phénomène n’arrivait pas avec la version stochastique (SRS), où le facteur d’adaptation était réellement découplé de la distance. C’est pour cette raison que les auteurs ont développé un profil de température assez délicat à contrôler, en exponentielles décroissantes par morceaux avec des réinitialisations périodiques, et mettent également en œuvre une réinitialisation périodique des  $\alpha_i$ . Les simulations que nous avons faites montrent que l’algorithme est difficile à contrôler, et, d’autre part, qu’il converge beaucoup plus lentement que le GLA. Toutefois, quand il fonctionne il permet d’atteindre une meilleure quantification (environ 1.5dB SNR de mieux en moyenne, le signal sur bruit —SNR— étant ici défini comme le rapport entre la distorsion moyenne et la norme de l’écart-type des vecteurs).

Le SCS n’est donc pas le meilleur exemple de “*winner-take-most*” que nous puissions imaginer et illustre la difficulté d’éviter le collage des unités.

### 3.5.3 Cartes de Kohonen

L'algorithme de Kohonen, sous ses diverses formes, sera repris en détail dans le chapitre 4, et nous nous intéresserons ici uniquement à son aspect “*winner-take-most*”.

Comme on le verra, l'adaptation faite par cet algorithme peut s'écrire sous la forme générale (3.13), où  $G(i)$  représente alors une fonction du voisinage physique des unités, qui sont réparties sur une grille. Appelons  $\mathbf{y}_i$  cette position des unités sur la grille, et  $d(\mathbf{y}_i, \mathbf{y}_j)$  la distance de grille entre deux unités. Cette distance peut être n'importe quelle norme  $L^p$  de la différence des coordonnées sur la grille.

En général,  $G(i)$  est choisi selon une des deux variantes suivantes :

1. *Voisinage gaussien* :

$$G(i) = \exp\left(-\frac{d^2(\mathbf{y}_i, \mathbf{y}_{i^*})}{\lambda^2}\right), \quad (3.20)$$

avec  $d(.,.)$  la distance euclidienne (norme  $L^2$ ).

2. *Voisinage rectangulaire* :

$$G(i) = \begin{cases} 1 & \text{si } d(\mathbf{y}_i, \mathbf{y}_{i^*}) \leq \lambda \\ 0 & \text{sinon} \end{cases}, \quad (3.21)$$

avec le plus souvent  $d(.,.)$  la distance “échiquier” (norme  $L^\infty$ ), qui est le maximum des différences de composantes (la portion de grille où  $G(i) = 1$  est un carré centré autour du gagnant).

Dans les deux cas,  $\lambda$  représente le *rayon du voisinage*, et ce voisinage est centré autour du gagnant euclidien,  $i^*$ .

Il faut bien voir que ce voisinage définit une région connexe dans l'espace physique des neurones (la grille), mais pas forcément dans l'espace des poids de ces neurones. En effet, la proximité des neurones dans l'espace des poids peut être complètement différente de celle dans la grille (les deux finissent par correspondre si l'auto-organisation s'est bien déroulée).

Les neurones ont une position de grille fixe; ils ne sont jamais superposés dans cet espace. Par conséquent, deux neurones dont les vecteurs-poids seraient collés à un instant donné peuvent toujours être séparés par la suite, puisque leur facteur d'adaptation selon (3.20) ou (3.21) peut être différent.

Dans cet algorithme, c'est donc la structure de voisinage dans un autre espace que celui des poids qui permet de *découpler* le facteur d'adaptation de la simple distance entre  $\boldsymbol{\xi}$  et les  $\mathbf{x}_i$ .

De fait, on constate dans les simulations que même lorsque les poids ont été collés (à la suite par exemple du maintien prolongé d'un grand rayon  $\lambda$ , ce qui produit un effet de "pincement"), ils finissent rapidement par se séparer dès que les conditions redeviennent "normales".

### 3.5.4 "Neural Gas"

Un autre algorithme, qui fera également l'objet d'un chapitre séparé (chapitre 5), a récemment été proposé par Martinetz *et al.* [89, 119].

Dans cet algorithme, appelé "*Neural Gas*", toutes les unités sont triées à chaque itération en fonction de leur proximité par rapport au vecteur d'entrée  $\xi$ . Un *rang de proximité*  $k$  est donc attribué à chaque unité (notons par commodité  $k(i)$  le rang de l'unité  $i$  et  $i(k)$  le numéro de l'unité de rang  $k$ ) de façon que :

$$\delta_0 < \delta_1 < \dots < \delta_k < \dots < \delta_{N-1} \quad (3.22)$$

avec la distance de rang  $k$

$$\delta_k = \|\xi - \mathbf{x}_{i(k)}\| \quad (3.23)$$

L'unité  $i$  de rang  $k(i) = 0$  est le gagnant euclidien, celle de rang  $k = 1$  vient juste après, etc. L'adaptation suit toujours la forme générale (3.13), avec le facteur d'adaptation

$$G(i) = \exp\left(-\frac{k(i)}{\lambda}\right). \quad (3.24)$$

Toutes les unités sont adaptées à chaque itération. L'algorithme peut être vu comme une sorte de carte de Kohonen avec une grille à une seule dimension, dont la topologie est redéfinie à chaque itération et dont le gagnant se trouve toujours à une extrémité. Le découplage entre distance et facteur d'adaptation provient ici du fait que leur relation n'est pas continue, mais *ordinale*. Quelle que soit la proximité des vecteurs-poids, la différence minimum de rang est 1 (pour deux unités *ex æquo*, l'ordre est choisi au hasard).

## 3.6 Quantification par arbre

Repenchons-nous un instant sur la partie où l'on recherche un gagnant. Sur une machine séquentielle, cette partie qui implique le calcul de  $N$  distances en  $n$  dimensions est très coûteuse quand  $N$  et  $n$  sont grands. Ceci est assez ennuyeux puisque non seulement cette recherche est faite à chaque itération lors de l'apprentissage (elle en prend souvent la part prépondérante), mais aussi à chaque itération de codage, lors de la phase d'exploitation après que le "*codebook*" a été établi.

Plutôt que de rechercher le gagnant d’une façon séquentielle, on pourrait souhaiter une structure organisée qui permette d’accélérer la procédure. C’est le cas des représentations par arbre binaire, dont chaque nœud fait une dichotomie de la distribution et dont les feuilles indiquent les vecteurs-codes.

Dans [47], Gersho et Gray indiquent qu’on peut transformer n’importe quel “codebook” en un tel arbre. Chaque nœud  $k$  représente un sous-ensemble  $X_k$  des vecteurs-codes  $\mathbf{x}_i$ , ainsi qu’un hyperplan séparateur :

$$\pi_k = \left\{ \mathbf{x} \in \mathbb{R}^n \mid \mathbf{s}_k^T \mathbf{x} - c_k = 0 \right\}. \quad (3.25)$$

On assigne au sous-arbre de gauche tous les  $\mathbf{x}_i$  qui représentent des points d’un côté du plan et au sous-arbre de droite ceux qui représentent des points de l’autre côté. Dans chaque sous-arbre, les points ainsi affectés sont à nouveau partitionnés en deux classes par un nouveau plan, et ainsi de suite jusqu’aux nœuds terminaux qui ne représentent qu’un seul vecteur-code. Les nœuds intermédiaires ne stockent que la normale  $\mathbf{s}_k$  et la constante  $c_k$  de leur plan, tandis que les nœuds terminaux stockent le vecteur-code  $\mathbf{x}_i$  qu’ils représentent.

Lorsqu’un vecteur d’entrée  $\boldsymbol{\xi}$  est présenté, on teste de quel côté il se trouve du plan séparateur du nœud racine, et on descend dans le sous-arbre correspondant. On recommence ce test récursivement jusqu’à aboutir au nœud terminal qui représente le gagnant  $\mathbf{x}_{i^*}$ .

L’aspect crucial qui rend possible cette représentation est d’admettre que, à chaque niveau, certains vecteurs-codes peuvent se trouver à la fois représentés par le sous-arbre de gauche et par celui de droite. En effet, au nœud  $k$ , les  $\mathbf{x}_i \in X_k$  dont la zone de dominance (dans le pavage de Voronoï par tous les  $\mathbf{x}_i$ ) est coupée par le plan  $\pi_k$  représentent des points des deux côtés du plan. Pour ces vecteurs-codes particuliers, le plan n’est pas discriminant et, par conséquent, on ne peut les omettre ni d’un côté ni de l’autre. En revanche, les points dont la zone de dominance est entièrement d’un côté du plan ne figurent que dans un des sous-arbres. On obtient ainsi un arbre d’une profondeur un peu plus grande que  $\log_2 N$ , mais dont l’emploi reste néanmoins très efficace (on remplace  $N$  calculs de distances par  $a \log_2 N$  produits scalaires). Les deux problèmes de cette méthode sont :

1. Pour chaque nœud  $k$ , trouver un “bon” plan séparateur, c’est-à-dire qui répartisse de la façon la plus équitable le sous-ensemble  $X_k$ , et en minimisant le nombre des vecteurs-codes qui doivent apparaître dans les deux sous-arbres parce que leur région de dominance est coupée par  $\pi_k$ .
2. Estimer efficacement si la région de dominance d’un  $\mathbf{x}_i$  est coupée par un plan. Ce point est particulièrement délicat en raison de la difficulté de calculer le pavage de Voronoï (surtout en grandes dimensions).



L'arbre est donc assez difficile à construire et, malgré le développement de méthodes heuristiques (voir [47]), on réserve en pratique son usage pour l'exploitation du “*codebook*”, en l'établissant une fois pour toutes après que ce dernier a été déterminé.

Il existe par contre des méthodes de quantification par arbre qui n'ont pas pour objet de construire l'arbre à partir d'un “*codebook*”, mais qui établissent les deux en même temps. Parmi ces méthodes, l'une des plus récentes semble particulièrement efficace, c'est la *quantification vectorielle par composante principale* (“*Principal Component Vector Quantization Algorithm*” — PCVQA), développée par Huang et Huang [58].

PCVQA est un algorithme qui établit un arbre similaire à celui précédemment décrit, de façon évolutive en partant de la racine. Tout d'abord, la moyenne de la distribution est estimée :  $\boldsymbol{\mu}_0 = E[\{\boldsymbol{\xi} \in \Xi\}]$ . Ensuite, un vecteur unitaire  $\mathbf{u}_0$  supportant la plus grande variance de la distribution est calculé, selon une des méthodes étudiées pour l'ACP (§ 2.4). Le premier plan séparateur est alors  $\pi_0 : \{\mathbf{x} \mid \mathbf{u}_0^T(\mathbf{x} - \boldsymbol{\mu}_0) = 0\}$ . Il partitionne la distribution  $\Xi$  en deux parties  $\Xi_0$  et  $\Xi_1$  de part et d'autre du plan. On sélectionne alors la région la plus “allongée” (c'est-à-dire celle dont la matrice de covariance possède la plus grande première valeur propre) et l'on recommence la procédure pour cette région. Cette sélection se fait à chaque étape parmi toutes les régions terminales trouvées jusqu'alors (celles des feuilles de l'arbre). Quand on a suffisamment subdivisé l'espace (c'est-à-dire qu'on a obtenu  $N$  régions terminales), le “*codebook*” est composé des  $N$  centres de gravité  $\mu_k$  des régions terminales, et l'arbre est déjà prêt à l'emploi !

En pratique, Huang et Huang décrivent des méthodes numériques puissantes pour trouver les  $\mathbf{u}_k$  ainsi que la région la plus “allongée”. Ces méthodes, que nous ne décrirons pas ici, sont le quotient de Rayleigh pour estimer la plus grande valeur propre et la méthode des puissances (et son accélération par le  $\delta^2$  d'Aitken) pour chercher une approximation du premier vecteur propre.

Le résultat de ces optimisations est un algorithme nécessitant beaucoup moins de temps calcul que le GLA par exemple, non seulement à l'exploitation mais également lors de la construction du “*codebook*”. De plus, les régions créées sont très régulières du point de vue de l'entropie.

### 3.7 Apprentissage compétitif avec régularisation (CLR)

Jusqu'ici, on a considéré que le but de la quantification vectorielle était de minimiser la distorsion moyenne de reconstruction (3.2). Même sans l'utilisation des techniques pour éviter les unités mortes (§ 3.4), cela conduit à faire respecter

la densité de distribution des données  $\xi$  par celle des vecteurs-codes  $\mathbf{x}_i$ . Cette tendance est explicitement renforcée dans les algorithmes de type FSCL (voir § 3.4).

Si cela est utile en compression de donnée, où l'on souhaite reconstruire les vecteurs d'entrée après quantification avec une erreur moyenne minimale, il existe des cas où ce respect de la densité est superflu, parce que l'on souhaite seulement connaître l'*espace occupé* par les données. Autrement dit, on désire obtenir une quantification régulière préalablement à l'estimation des fonctions de densité de probabilité. Par exemple, en classification (où l'on cherche seulement à discriminer les données en classes et non à les reconstruire après compression), on s'intéresse surtout aux frontières entre classes, et non à la distribution à l'intérieur de ces classes. D'autre part, dans notre contexte général d'analyse de donnée en grandes dimensions, on aimerait séparer le support de la distribution (la variété, voir § 2.5.1) de la densité sur ce support.

Une méthode que nous avons développée pour diminuer l'importance de la densité dans la quantification est de *régulariser* la répartition des vecteurs-codes.

Généralement, on régularise des valeurs qui sont attachées à des positions sur une grille. Dans ce cas, la structure régulière de voisinage entre points permet d'appliquer des filtres passe-bas par la convolution d'un noyau sur les valeurs. Ce genre de procédé est abondamment employé en traitement d'image, par exemple lorsqu'on veut "adoucir" une image. Les valeurs à régulariser sont alors celles des pixels de l'image, dont la position est régulière. Chaque pixel possède 4 voisins (sauf aux bords) et un filtrage par convolution est possible.

Dans notre cas, le problème est plus compliqué puisque les valeurs à régulariser sont les positions elles-mêmes, et aucune structure de voisinage régulière n'est *a priori* disponible. Une exception cependant : les cartes de Kohonen, où les unités ont un arrangement régulier dans un espace annexe de basse dimension (généralement  $\leq 3$ ). La régularisation des vecteurs-codes qui sont attachés aux unités et qui quantifient la distribution provient ici de ce que, lors de l'adaptation, le gagnant entraîne avec lui les vecteurs-codes de ses proches voisins dans la grille. Lorsque la fonction de voisinage est "douce" (par exemple le voisinage gaussien), il y a un effet d'interpolation (voir § 4.3). En dehors de ce cas particulier, on ne sait pas sur quel voisinage s'appuyer, puisque l'échantillonnage est irrégulier. On pourrait bien chercher comme voisins des points proches et "bien répartis" en direction, mais ce critère est difficile à définir, surtout en grandes dimensions.

Notre première étape va donc consister à fabriquer une telle structure de voisinage, à l'aide d'une technique intéressante empruntée à la version originale du "*Neural Gas*" [89]. Cette technique, qu'on pourrait appeler celle des *liens dynamiques*, consiste à créer des liens entre des unités qui ont des activités corrélées. C'est un peu l'idée de la règle de Hebb, sauf qu'ici ces liens ne transmettent au-

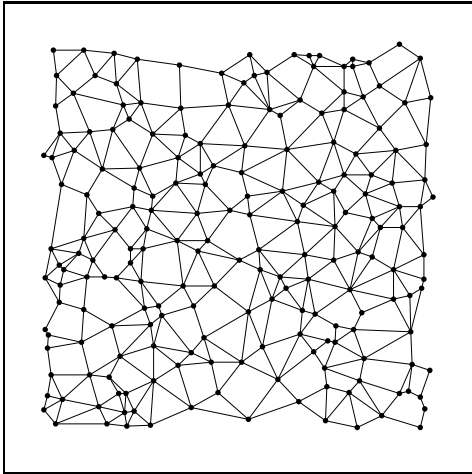


Figure 3.4: Les “liens dynamiques”.  
Ils approximent assez bien la  
triangulation de Delaunay.

cune activité, ce ne sont pas des connexions fonctionnelles; ils ne sont là que pour nous informer de la structure locale de l’espace. La méthode peut s’appliquer à n’importe quel type de quantification par réseau compétitif. L’évolution des liens est régie selon deux règles :

- A chaque itération, un lien est *créé* ou *rafraîchi* entre le gagnant euclidien (l’unité  $i^*$  dont le vecteur  $\mathbf{x}_{i^*}$  est le plus proche de l’entrée  $\boldsymbol{\xi}$ ) et la deuxième plus proche unité (celle qui serait gagnante si  $i^*$  n’existait pas).
- Un lien qui n’a pas été rafraîchi dans les  $T$  dernières itérations disparaît.  $T$  est la période de vie des liens et peut varier au cours du temps (en général, elle est inversement proportionnelle à  $\alpha(t)$ , de façon à avoir une courte durée de vie des liens quand les unités bougent beaucoup).

Une fois que les vecteurs-codes commencent à se stabiliser (lorsque le gain  $\alpha(t)$  est devenu suffisamment petit) et si la période  $T$  est “adéquate”, l’ensemble des liens est une bonne approximation de la triangulation de Delaunay des vecteurs-codes (figure 3.4). En effet, où qu’il tombe, un vecteur  $\boldsymbol{\xi}$  va toujours créer un lien entre deux zones de Voronoï adjacentes, et le lien sera perpendiculaire au plan séparateur de ces deux zones.

Appelons  $c_{ij}$  l’indicateur de la présence d’un lien entre  $\mathbf{x}_i$  et  $\mathbf{x}_j$ , et notons l’ensemble des  $N_i$  voisins de  $\mathbf{x}_i$  par  $V_i$  :

$$c_{ij} = \begin{cases} 1 & \text{si } \exists \text{ lien}(i, j) \\ 0 & \text{sinon} \end{cases}$$

$$V_i = \{j \mid c_{ij} = 1\} \tag{3.26}$$

$$N_i = \text{card}(V_i) \tag{3.27}$$

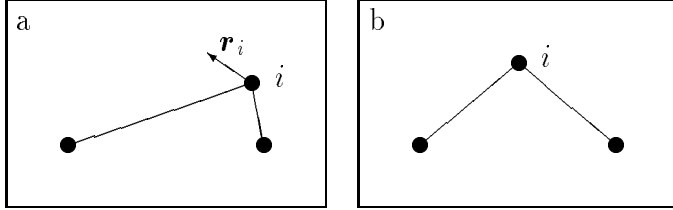


Figure 3.5: La régularisation sans effet de bord. (a) Situation de départ, avec le vecteur de régularisation  $\mathbf{r}_i$  correspondant à cette situation. (b) Résultat de plusieurs itérations de régularisation sur le point  $i$ .

Sur la structure de voisinage ainsi définie, nous pouvons appliquer une régularisation par l'ajout d'un terme à la règle d'adaptation. Ce terme ne peut pas être simplement un rapprochement du point vers le barycentre de ses voisins, comme il est coutume de le faire avec un filtre passe-bas. En effet, pour les points près du bord, dont tous les voisins sont du côté de l'intérieur de la distribution, on assisterait dans ce cas à une attraction vers le centre de la distribution (c'est d'ailleurs ce qui se passe dans les cartes de Kohonen : l'*effet de bord*). Nous allons plutôt chercher à égaliser les distances qui séparent le point de ses voisins, en minimisant la variance des longueurs de liens, *mais sans modifier leur moyenne*. On obtient, en considérant que la moyenne  $\mu_i$  des longueurs des liens varie peu dans l'opération (on ne la dérive donc pas) :

$$\begin{aligned} \text{Var}(\|\mathbf{x}_i - \mathbf{x}_j\|) &= \mathbb{E}_{j \in V_i} [(\|\mathbf{x}_i - \mathbf{x}_j\| - \mu_i)^2] \\ &= \frac{1}{N_i} \sum_{j \in V_i} (\|\mathbf{x}_i - \mathbf{x}_j\| - \mu_i)^2 \end{aligned} \quad (3.28)$$

$$\begin{aligned} \mathbf{r}_i = -\frac{\partial}{\partial \mathbf{x}_i} \text{Var}(\|\mathbf{x}_i - \mathbf{x}_j\|) &= -\frac{\partial}{\partial \mathbf{x}_i} \frac{1}{N_i} \sum_{j \in V_i} (\|\mathbf{x}_i - \mathbf{x}_j\| - \mu_i)^2 \\ &= \frac{2}{N_i} \sum_{j \in V_i} (\|\mathbf{x}_j - \mathbf{x}_i\| - \mu_i) \frac{\mathbf{x}_j - \mathbf{x}_i}{\|\mathbf{x}_j - \mathbf{x}_i\|} \end{aligned} \quad (3.29)$$

Observons la forme du gradient de régularisation  $\mathbf{r}_i$  (3.29). La contribution de chaque lien est son vecteur directeur pondéré par la différence entre sa propre norme et la norme moyenne. Il n'y a pas d'effet de bord : un point qui se trouverait à "un coin" de la structure de voisinage n'est pas attiré par ses voisins, mais ses liens avec eux sont égalisés (figure 3.5).

Un apprentissage compétitif (CL) peut être muni de ce terme de régularisation, ce qui donne la règle d'adaptation de l'algorithme que nous appelons "*Competitive Learning with Regularization*" (CLR) :

$$\begin{aligned} \mathbf{x}_{i^*} &\leftarrow \mathbf{x}_{i^*} + \alpha(t)(\boldsymbol{\xi} - \mathbf{x}_{i^*}) + \gamma(t)\mathbf{r}_i \\ \mathbf{x}_{i^*} &\leftarrow \mathbf{x}_{i^*} + \alpha(t)(\boldsymbol{\xi} - \mathbf{x}_{i^*}) + \end{aligned}$$

### 3.7. APPRENTISSAGE COMPÉTITIF AVEC RÉGULARISATION (CLR) 67

$$\gamma(t) \frac{2}{N_{i^*}} \sum_{j \in V_{i^*}} (\|\mathbf{x}_j - \mathbf{x}_{i^*}\| - \mu_{i^*}) \frac{\mathbf{x}_j - \mathbf{x}_{i^*}}{\|\mathbf{x}_j - \mathbf{x}_{i^*}\|} \quad (3.30)$$

où  $\gamma(t)$  est un paramètre similaire à  $\alpha(t)$ , qui contrôle l'importance de la régularisation. Il est intéressant de voir qu'en faisant fonctionner l'algorithme avec  $\alpha = 0$  et  $\gamma \neq 0$ , on obtient en quelques itérations un arrangement hexagonal centré. On peut contrôler  $\gamma(t)$  pour obtenir une régularisation plus ou moins forte. En réduisant  $\gamma$  à 0, on obtient évidemment le simple CL. En général,  $\gamma(t)$  varie en  $1/t$  entre 0.5 et 0, mais on peut aussi le laisser constant pendant tout l'apprentissage. Dans ce cas, comme  $\alpha(t)$  varie aussi en  $1/t$  (comme pour le CL simple), l'importance relative de la régularisation devient prépondérante.

Dans la figure 3.6, on montre la différence entre le CL et le CLR dans le cas de distributions de densité non-uniforme. On remarque que la répartition des vecteurs-codes est indépendante de la densité dans le cas du CLR.

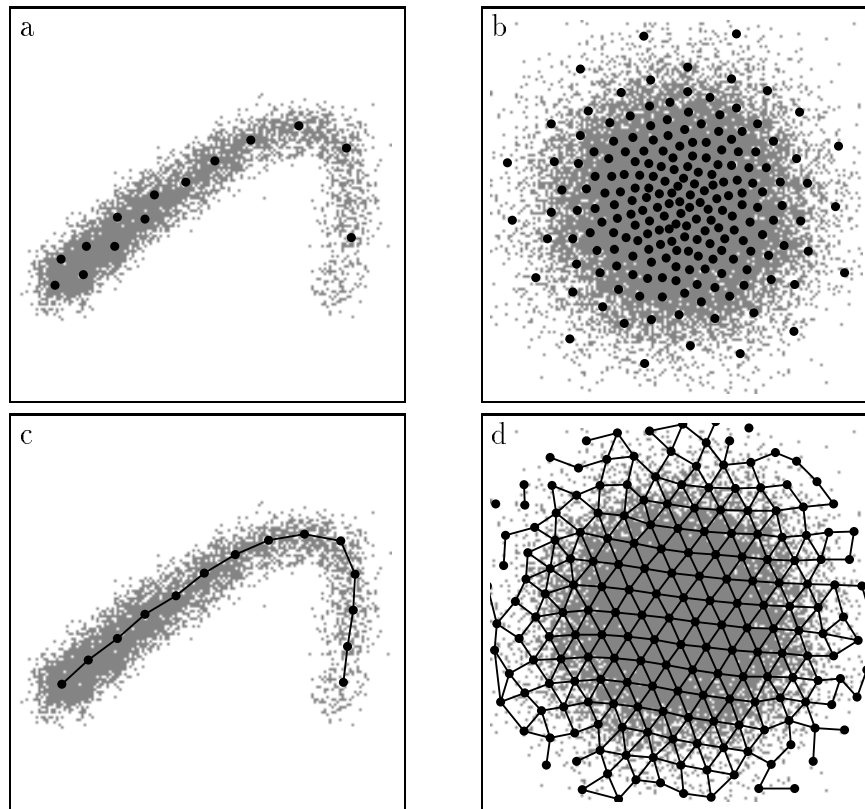


Figure 3.6: Comparaison de quantification vectorielle avec ou sans régularisation. (a et b) CL simple. (c et d) CL avec régularisation sur le support des liens dynamiques (CLR). Le facteur de régularisation a été fixé ici de façon permanente à  $\gamma = 0.5$ . A gauche : la densité est une exponentielle décroissant avec l'abscisse curviligne. A droite : distribution gaussienne.

# Chapitre 4

## Cartes auto-organisées

### 4.1 Introduction

De manière générale, la quantification vectorielle ne s'intéresse pas à la disposition physique des unités. On pourrait toutefois placer ces dernières sur une ligne ou sur un plan et modifier un peu les règles d'adaptation pour que cette position représente une information, par exemple que les unités dont les vecteurs-poids sont proches dans l'espace d'entrée soient aussi proches par leur emplacement physique. C'est précisément ce que nous apportent les *réseaux auto-organisés*, et la fonction ainsi créée s'appelle "*extraction ou mapping de caractéristiques*", ou encore "*mapping préservant la topologie*".

L'un des premiers à aborder cette question est Von der Malsburg qui, en 1973, cherchait à comprendre la formation des colonnes d'orientation dans le cortex visuel [118], puis celle des cartes rétinotopiques en 1976 avec Willshaw [120]. Le modèle qu'il a proposé est un réseau de la forme illustrée à la figure 4.1(a), où la couche inférieure représente des capteurs (visuels en l'occurrence) organisés en grille bidimensionnelle (une image de  $n = w_i \times h_i$  pixels) et la couche supérieure un réseau de neurones également organisés en grille (de  $N = w_o \times h_o$  neurones). Les deux couches sont complètement interconnectées, c'est-à-dire que chaque neurone a un vecteur-poids de  $n$  dimensions (chaque pixel est connecté, par une connexion spécifique, avec chaque neurone).

Par ailleurs, au sein de la grille de neurones, des connexions sont présentes entre les neurones, excitatrices à courte distance et inhibitrices à longue distance, selon un profil en "chapeau mexicain" (figure 4.2). Lorsqu'un motif d'activité  $\xi$  des capteurs est présenté, les neurones répondent par une activité plus ou moins forte selon la proximité de leur vecteur-poids à ce motif  $\xi$ . Une coopération/compétition a lieu entre les neurones qui s'activent ou s'inhibent mutuellement au moyen des connexions latérales. Contrairement au "*Competitive Lear-*

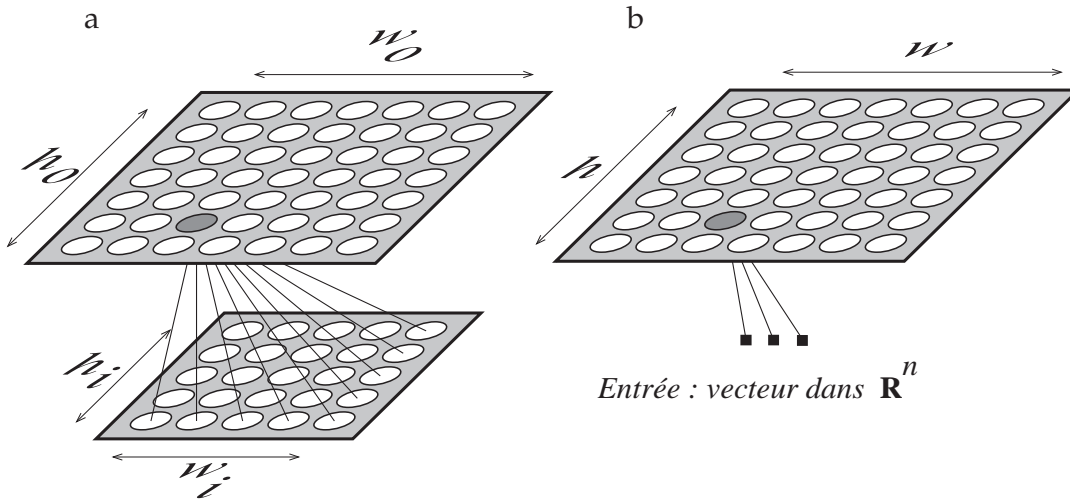


Figure 4.1: Architectures des cartes auto-organisées. (a) Modélisation de la rétinotopie selon Von der Malsburg. Les capteurs sont disposés sur un plan (une “rétine” de  $n = w_i \times h_i$  pixels). Les neurones sont aussi disposés sur un plan (le “cortex”, selon une grille de  $N = w_o \times h_o$  neurones). Les deux couches sont complètement connectées (on ne montre que quelques connexions ici). (b) Simplification : le plan de capteurs est remplacé par des entrées avec des valeurs continues.

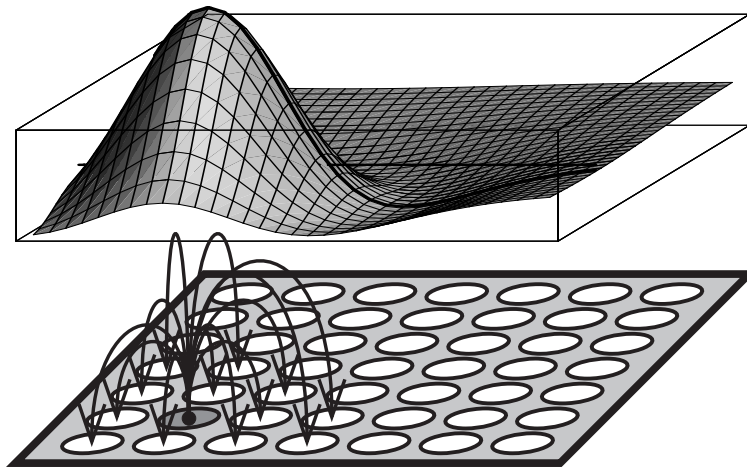


Figure 4.2: Les connexions récursives entre les neurones ont une valeur qui varie en fonction de la distance latérale, selon un profil en “chapeau mexicain”. Les neurones proches s’activent entre eux (coopération), parce que les connexions à courte distance sont excitatrices. À l’inverse, les connexions distantes sont inhibitrices (compétition). En réalité, on peut fabriquer le chapeau mexicain par la différence de deux gaussiennes. Ce profil résulterait ainsi de la superposition de connexions exclusivement inhibitrices selon une gaussienne large, et de connexions exclusivement excitatrices selon une gaussienne plus étroite mais moins aplatie. Ceci permet éventuellement de faire varier le rayon du chapeau mexicain sans que certaines connexions soient obligées de changer de signe (ce qui serait peu plausible du point de vue biologique).



*ning*” (§ 3.3), il n’y a pas qu’un seul gagnant qui émerge de cette compétition, mais une *bulle d’activité* centrée sur des neurones voisins. L’adaptation est faite selon une règle de Hebb suivie d’une re-normalisation des poids; il en résulte un *rapprochement des poids des neurones de la bulle vers  $\xi$* . La différence fondamentale par rapport à la simple quantification vectorielle (par un CL, par exemple) tient dans cette adaptation de plusieurs neurones voisins, et non du seul gagnant.

Dans le cas du modèle de rétinotopie, le but est d’apprendre, d’une façon non supervisée, à projeter un stimulus localisé à un endroit du plan de capteur vers un endroit correspondant dans la carte de neurones. C’est donc une cartographie position-position. En présentant des taches localisées sur plusieurs capteurs voisins, on informe le réseau de l’organisation spatiale de ces capteurs, par une redondance qui est ici la corrélation entre capteurs voisins. Le réseau détecte cette redondance et peut s’organiser en conséquence. Un tel résultat serait impossible en présentant des taches localisées sur un seul capteur à la fois. Willshaw et Von der Malsburg ont utilisé des taches minimales : seuls deux capteurs adjacents sont actifs. Plus tard, Takeuchi et Amari [115] ont montré (dans le cas monodimensionnel de lignes continues de capteurs et de neurones) qu’il suffisait que la largeur du “chapeau mexicain” soit suffisamment grande par rapport à celle des taches pour obtenir une organisation correcte.

Dans le premier modèle, celui des orientations, les motifs appris sont des lignes passant au centre du plan de capteurs et d’orientation aléatoire. Le réseau ne fait pas alors une correspondance position-position, mais apprend à reconnaître les orientations. L’orientation pour laquelle se spécialisent les unités change doucement (avec des discontinuités par endroit) dans le plan des neurones (figure 4.3).

Ce genre de cartes est très utilisé dans le cerveau, et de telles projections se retrouvent aussi bien en provenance d’afférences sensorielles (vue, ouïe, toucher) que vers des zones motrices, ou même entre différentes aires corticales. Il semble hautement improbable que les connexions qui participent à ces projections topiques soient entièrement déterminées génétiquement. Au contraire, on a observé qu’elles dépendaient également de l’expérience dans les premières semaines de la vie (voir les expériences de Hubel et Wiesel sur les chats [59]). Même si l’on pense actuellement que d’autres mécanismes que l’apprentissage non supervisé sont également en jeu dans la formation des cartes, le modèle de Von der Malsburg a le mérite de prouver que les mécanismes qui produisent de l’auto-organisation peuvent être très simples.

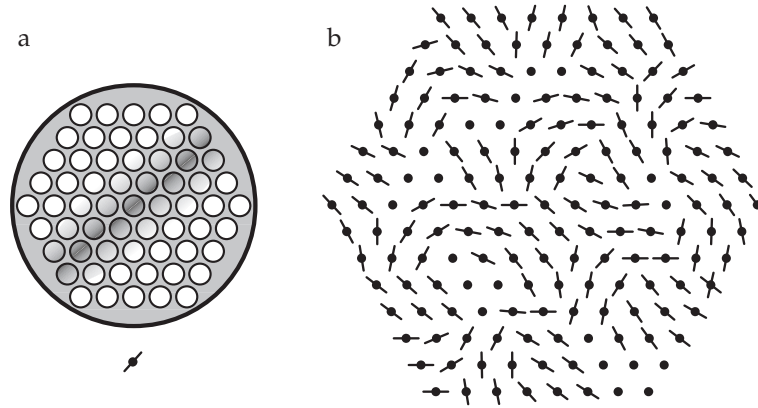


Figure 4.3: Apprentissage de la sélectivité à l'orientation. (a) Un des motifs d'activités des capteurs présentés pendant l'apprentissage. (b) Champs récepteurs des neurones. Chaque vecteur-poids des neurones représente un motif d'une certaine orientation. Il y a une certaine continuité locale dans ces orientations, avec parfois des sauts.

## 4.2 Algorithme de Kohonen

Deux simplifications algorithmiques du modèle de Von der Malsburg permettent d'obtenir les *cartes auto-organisantes de Kohonen* [66, 70]. Premièrement, ce dernier a proposé de remplacer le motif d'activité des capteurs par un vecteur à composantes continues (par exemple deux composantes qui définissent une position dans le plan des capteurs). La structure du réseau prend donc la forme de la figure 4.1(b). D'autre part, le mécanisme de formation de la bulle par le jeu des connexions latérales en profil de chapeau mexicain, qui prend beaucoup de temps en simulation, est remplacé par la prise en compte de l'effet de cette bulle : il n'y a plus de connexion latérale, et on procède à un simple calcul de gagnant, mais l'adaptation se fait pour tous les neurones dans le voisinage du gagnant.

Avec ces deux modifications, les cartes peuvent être vues comme *une quantification vectorielle sous contrainte d'une structure de voisinage entre les unités*. Non seulement elles quantifient la distribution en fonction de sa densité, mais de plus elles réalisent une *projection* de cette distribution vers la grille des neurones en *conservant la notion de proximité* (au moins localement). Dans la figure 4.5, on peut voir le résultat obtenu sur quelques distributions simples en deux dimensions<sup>1</sup>. Dans ce type de représentation, illustrée à la figure 4.4, les petites croix représentent la position des vecteurs-poids des unités. Ces croix sont

<sup>1</sup>Ici, la propriété de projection n'est pas très intéressante, puisqu'on ne change pas de dimension (entrée 2D, grille 2D). Néanmoins, deux vecteurs  $\xi_i$  proches vont activer deux neurones proches dans la carte. On reviendra plus tard sur la propriété de projection avec réduction de dimension.

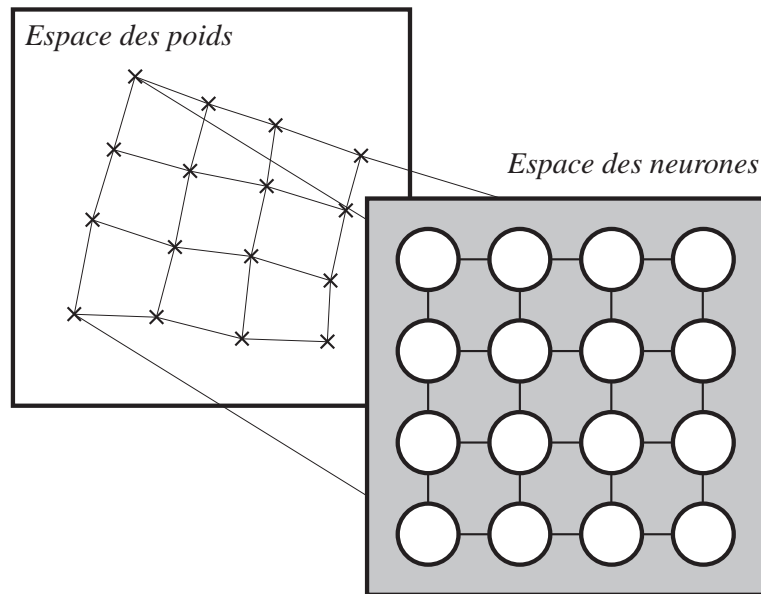


Figure 4.4: Principe de la représentation usuelle du réseau de Kohonen dans l'espace des poids : le vecteur-poids de chaque neurone est un point. Des lignes relient les points correspondant à des neurones adjacents dans la grille, ce qui permet de visualiser l'organisation des poids.

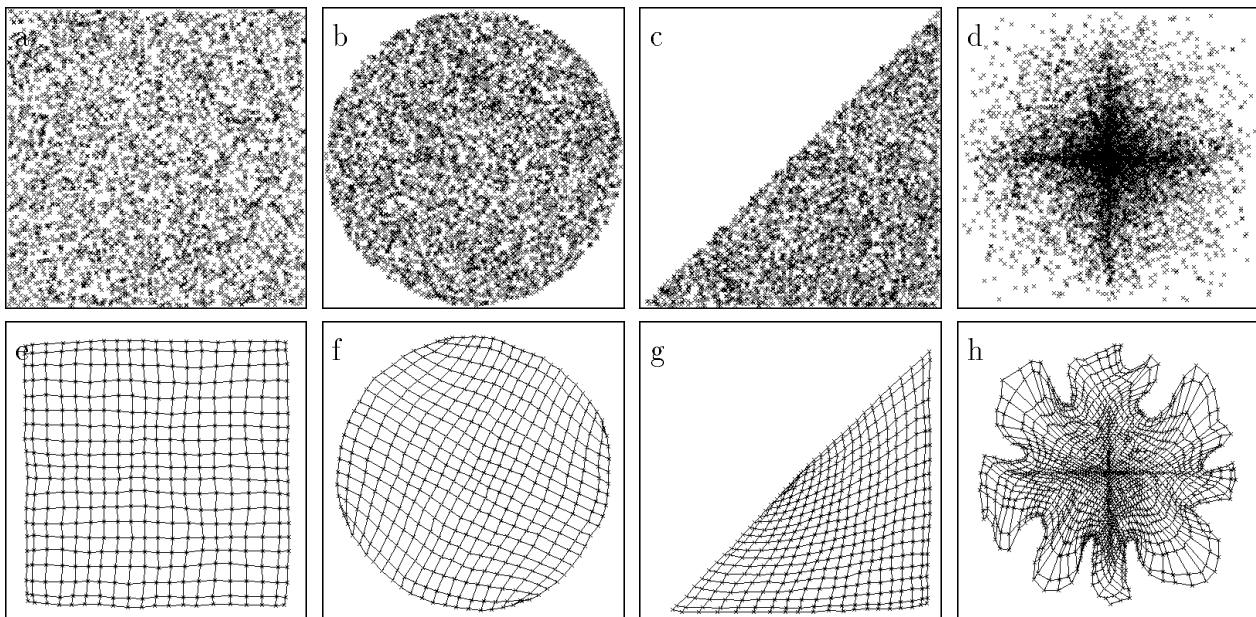





Figure 4.5: Quelques distributions dans le plan et leur quantification par le réseau de Kohonen. La représentation du réseau (e,f,g,h) est faite dans l'espace des poids (voir figure 4.4).

reliées par une ligne quand les poids qui correspondent sont ceux de deux unités adjacentes dans la grille de neurones. On obtient ainsi la vue d'une sorte de filet élastique dont les nœuds correspondent aux vecteurs-poids.

Une autre modification par rapport au modèle de Von der Malsburg et au premier algorithme de Kohonen concerne la détermination du gagnant. Au lieu de considérer le maximum d'activité ( $\max.$  du produit scalaire  $\boldsymbol{\xi}^T \mathbf{x}_i$ ), on recherche le minimum des distances  $d(\boldsymbol{\xi}, \mathbf{x}_i)$ . Dans le cas de la distance euclidienne ( $\|\boldsymbol{\xi} - \mathbf{x}_i\|$ ), et si les vecteurs-poids  $\mathbf{x}_i$  sont normés, cela revient au même. En effet,

$$\|\boldsymbol{\xi} - \mathbf{x}_i\|^2 = \|\boldsymbol{\xi}\|^2 - 2\boldsymbol{\xi}^T \mathbf{x}_i + \|\mathbf{x}_i\|^2. \quad (4.1)$$

Comme  $\|\boldsymbol{\xi}\|^2$  est indépendant de  $i$  et que  $\|\mathbf{x}_i\|^2$  est constant si les poids sont normalisés, l'unité pour laquelle le produit scalaire est maximum est aussi celle qui est la plus proche de  $\boldsymbol{\xi}$  en distance euclidienne. Remarquons d'ailleurs que le carré de la norme convient très bien, puisqu'il s'agit seulement de détecter un minimum; on évite ainsi le calcul de racine carrée dans l'algorithme de recherche du gagnant. L'avantage d'utiliser une distance plutôt que le produit scalaire permet d'éviter la normalisation des poids. On peut ainsi travailler avec des espaces d'entrée pas forcément normés. Les distances le plus couramment utilisées dérivent de la distance généralisée de Minkowski ( $d_p(\mathbf{a}, \mathbf{b}) = (\sum_{k=1}^n |a_k - b_k|^p)^{\frac{1}{p}}$ ) et sont :

Euclidienne ( $p = 2$ )	$d_e(\mathbf{a}, \mathbf{b}) = \sqrt{(\mathbf{a} - \mathbf{b})^T (\mathbf{a} - \mathbf{b})}$	
Manhattan ( $p = 1$ )	$d_m(\mathbf{a}, \mathbf{b}) = \sum_{k=1}^n  a_k - b_k $	
Chessboard ( $p = \infty$ )	$d_c(\mathbf{a}, \mathbf{b}) = \max_k  a_k - b_k $	

La figure 4.6 montre le pavage de l'espace par zones de dominance en fonction de la distance utilisée.

Contrairement à ce qui est parfois dit, distance de Manhattan et distance euclidienne ne donnent pas des résultats similaires, même en considérant globalement le processus d'auto-organisation. La distance de Manhattan peut avoir des

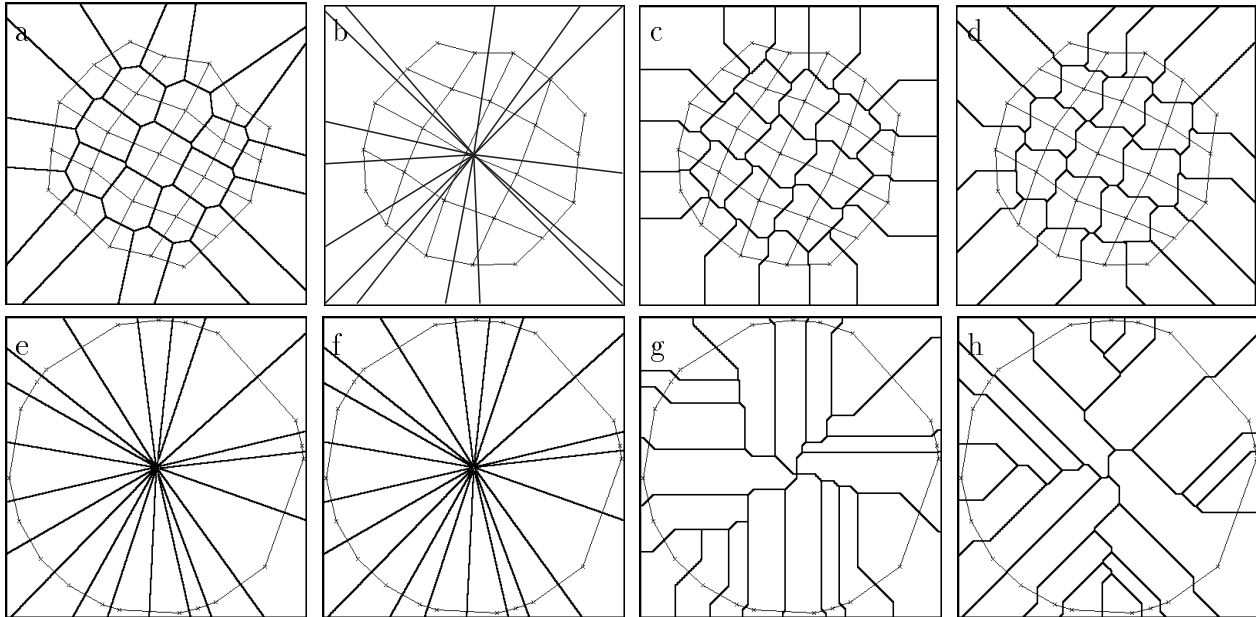


Figure 4.6: Pavage de l'espace par zones de dominance en fonction de la distance utilisée : (a,e) euclidienne, (b,f) produit scalaire, (c,g) de Manhattan, (d,h) Chessboard.

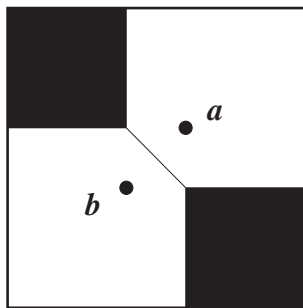


Figure 4.7: La distance de Manhattan peut donner une frontière "surfactive" par endroit. Dans ce cas dégénéré, tous les points de la zone en noir sont à la même distance de  $a$  que de  $b$ .

effets pervers : par exemple, sur un plan, la frontière des zones de dominance de deux points situés sur une diagonale est surfactive par endroits (figure 4.7). Autrement dit, pour tous les points de cette surface-frontière, le gagnant est indéterminable, ce qui peut être assez gênant. Bien sûr, il s'agit là d'un cas particulier dont la probabilité est nulle quand on travaille en valeurs réelles. Mais, quand les poids sont quantifiés (ce qui est justement le cas dans les applications sur circuits numériques), la probabilité d'avoir deux points exactement sur une diagonale n'est plus nulle et, de surcroît, elle augmente avec le nombre de dimensions. Enfin et surtout, la distance euclidienne est une mesure isotrope, ce qui n'est pas le cas de celle de Manhattan, et dans les simulations on constate une sorte de polarisation des poids le long des axes des coordonnées.

L'algorithme simplifié est similaire au “*Competitive Learning*” (§ 3.3), sauf que l'adaptation ne se fait pas que pour un neurone, mais sur tout un voisinage dans la grille. Appelons  $\mathbf{y}_i$  la position des unités sur la grille, et  $d(\mathbf{y}_i, \mathbf{y}_j)$  la distance de grille entre deux unités  $i$  et  $j$ . L'adaptation des vecteurs-poids  $\mathbf{x}_i$  s'écrit alors :

$$\mathbf{x}_i \leftarrow \mathbf{x}_i + \alpha(t) G(\mathbf{y}_i, \mathbf{y}_{i^*}) (\boldsymbol{\xi} - \mathbf{x}_i) \quad (4.2)$$

avec  $G(\mathbf{y}_i, \mathbf{y}_{i^*})$  un facteur de pondération fonction de la distance dans la grille autour du gagnant  $i^*$ . A l'origine, le voisinage proposé était simplement rectangulaire :

$$G(i) = \begin{cases} 1 & \text{si } d(\mathbf{y}_i, \mathbf{y}_{i^*}) \leq \lambda(t) \\ 0 & \text{sinon} \end{cases}, \quad (4.3)$$

avec  $d(.,.)$  la distance “échiquier”. La portion de grille où  $G(i) = 1$  est ainsi un carré de largeur  $2\lambda(t)$  centré sur le gagnant. L'adaptation est faite pour tous les neurones de ce carré, et pour aucun autre.

Une variante souvent mentionnée [71, 104, 103] est d'utiliser un voisinage gaussien, qui donne des résultats plus “doux” (figure 4.8). Cette fonction de voisinage s'écrit :

$$G(i) = \exp\left(-\frac{d^2(\mathbf{y}_i, \mathbf{y}_{i^*})}{\lambda^2(t)}\right), \quad (4.4)$$

avec  $d(.,.)$  la distance euclidienne.

Dans les deux cas,  $\lambda(t)$  est le rayon du voisinage, qui diminue au cours du temps en  $1/t$ , au même titre que le facteur de gain  $\alpha(t)$ .

Le réseau est assez sensible à ces paramètres. Par exemple, si  $\alpha(t)$  diminue trop vite, d'une part la distribution n'est pas bien couverte (comme avec le CL), et d'autre part le respect de topologie n'est pas bon (figure 4.9(b)). Si  $\lambda(t)$  diminue trop rapidement par rapport à  $\alpha(t)$ , on peut assister à des replis, par exemple en deux dimensions à une configuration “papillon” (figure 4.9(c)). Si au contraire il ne diminue pas assez vite, on peut alors avoir un effet de “pincement” (figure 4.9(b)).

### 4.3 Voisinage gaussien

Comme on le voit dans la figure 4.8, l'usage du voisinage gaussien donne des résultats plus doux. Dans les cas où le nombre de points est faible par rapport au nombre de neurones, on remarque en pratique un *effet d'interpolation* : les vecteurs-poids viennent couvrir l'espace entre les points  $\boldsymbol{\xi}$ . Nous allons étudier ce phénomène à l'aide d'un cas simple : le réseau et la distribution sont en une dimension, et cette dernière n'est formée que des deux points 0 et 1 apparaissant

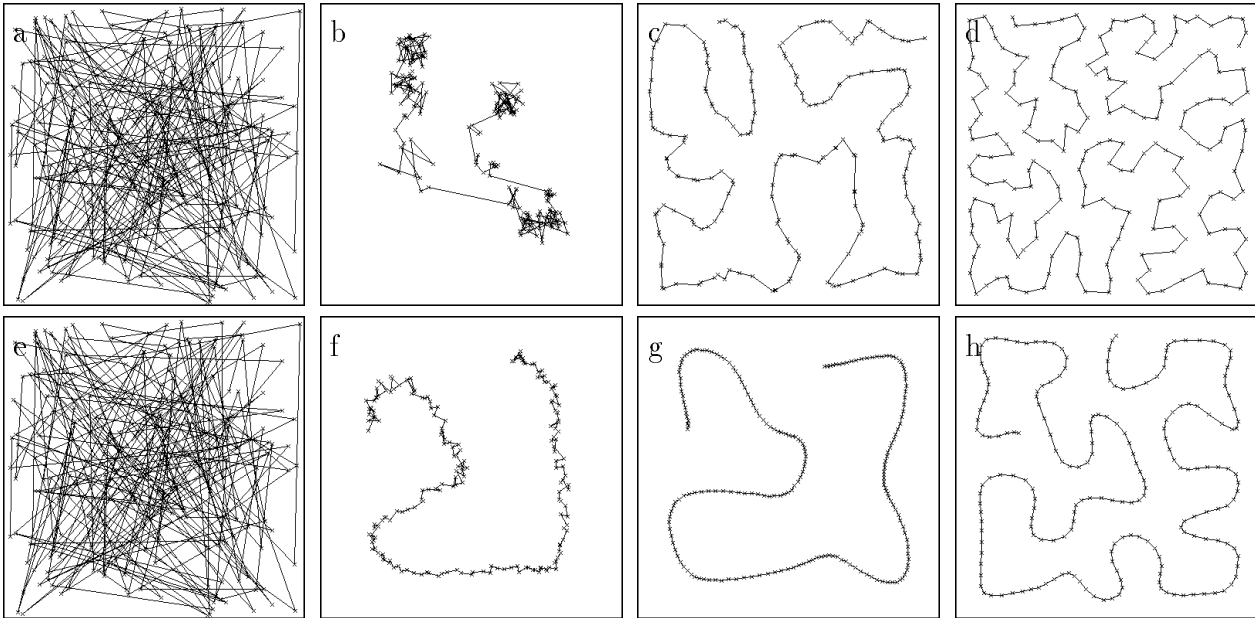


Figure 4.8: Comparaison entre voisinage rectangulaire (en haut) et voisinage gaussien (en bas) sur 10 000 itérations au total. Le voisinage gaussien donne une configuration des poids plus “douce” en cours d’apprentissage.

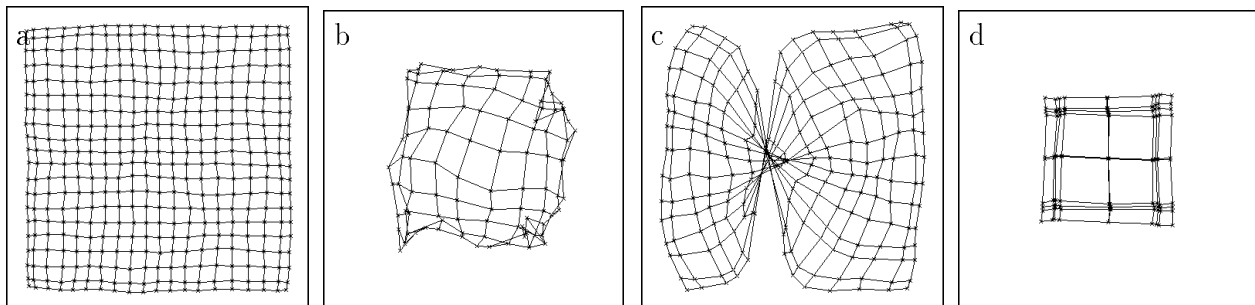


Figure 4.9: Quelques problèmes qui peuvent survenir quand les paramètres  $\alpha(t)$  ou  $\lambda(t)$  ont un profil inapproprié. (a) Cas “normal”, pour comparaison. (b)  $\alpha(t)$  a diminué trop vite. (c) Configuration en “papillon”, qui résulte d’une diminution trop rapide de  $\lambda(t)$  par rapport à  $\alpha(t)$ . (d) Effet de “pincement”, dû à une diminution trop lente de  $\lambda(t)$ .

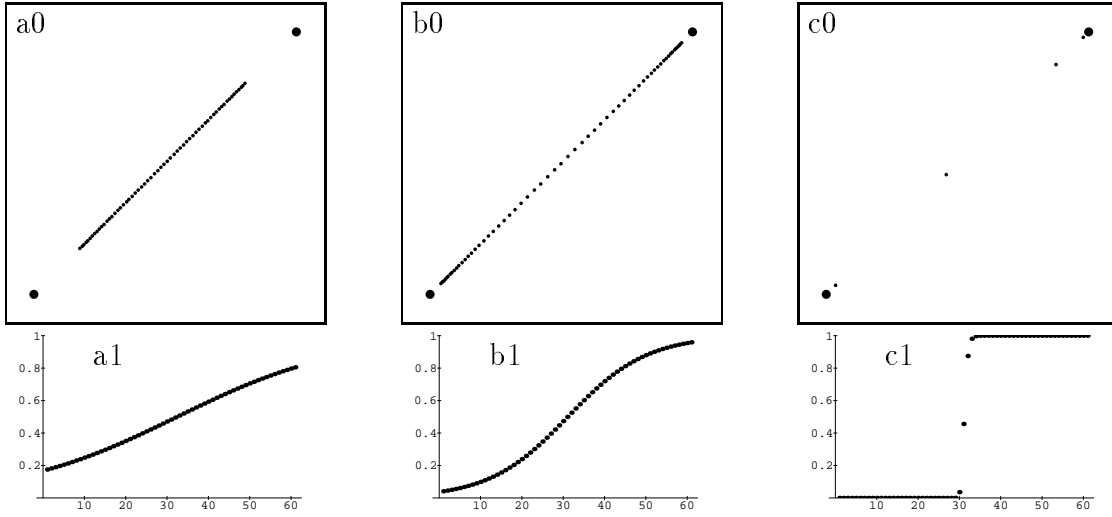


Figure 4.10: Explication de l'effet d'interpolation produit par le voisinage gaussien : les poids ont une position d'équilibre qui est répartie selon une sigmoïde. (a) Réseau partiellement ordonné. (b) Situation intermédiaire. (c) Situation obtenue si l'on poursuit l'apprentissage très longtemps avec un  $\lambda$  faible.

chacun avec une probabilité  $\frac{1}{2}$ . Admettons que le réseau se soit déjà partiellement ordonné, comme dans la figure 4.10, c'est-à-dire que le gagnant se trouve toujours à l'une des deux extrémités. Disons par exemple :

$$i^* = \begin{cases} 1 & \text{si } \xi = 0 \\ N & \text{si } \xi = 1 \end{cases} . \quad (4.5)$$

Comme la position des unités dans la carte peut se noter (par exemple)  $y_i = i$ , on a le déplacement du vecteur-poids de la  $i$ ème unité :

$$\Delta x_i = \alpha(t) \exp\left(-\frac{(i - i^*)^2}{\lambda^2(t)}\right) (\xi - x_i) \quad (4.6)$$

Les deux cas possibles  $\xi = 0$  et  $\xi = 1$  donnent respectivement :

$$\Delta_0 x_i = -\alpha(t) \exp\left(-\frac{(i - 1)^2}{\lambda^2(t)}\right) x_i \quad (4.7)$$

$$\Delta_1 x_i = \alpha(t) \exp\left(-\frac{(i - N)^2}{\lambda^2(t)}\right) (1 - x_i) \quad (4.8)$$

A l'équilibre (quand les  $x_i$  ont atteint une position asymptotiquement stable), la somme pondérée des déplacements doit s'annuler en moyenne :

$$\Delta_0 x_i P(\xi = 0) + \Delta_1 x_i P(\xi = 1) = 0 \quad (4.9)$$



On a posé  $P(\xi = 0) = P(\xi = 1) = \frac{1}{2}$ . Donc, après quelques manipulations algébriques, on trouve comme position d'équilibre moyenne pour les poids :

$$x_i = \frac{1}{1 + \exp\left(-\frac{2(N-1)}{\lambda^2(t)}\left(i - \frac{N+1}{2}\right)\right)} \quad (4.10)$$

L'équation (4.10) est une sigmoïde centrée sur la moitié du réseau ( $\frac{N+1}{2}$ ) et passant en  $\frac{1}{2}$  à ce point. La pente en ce point vaut  $\frac{N-1}{\lambda^2(t)}$ . Quand  $\lambda(t)$  diminue au cours du temps, on obtient ainsi une répartition des poids qui évolue comme dans la figure 4.10.

On constate que quand  $\lambda$  tend vers 0 (parce que  $t \rightarrow \infty$ ), aucun poids ne se trouve entre 0 et 1 (la sigmoïde tend vers une fonction-seuil). Si toutefois on tient compte du fait que le gain  $\alpha(t)$  est devenu petit lui aussi, et qu'on arrête la simulation quand  $\lambda$  n'est pas trop petit, on obtient un *effet d'interpolation* des unités entre 0 et 1.

Cet effet est encore renforcé par l'ajout d'un terme en  $(1 - a_{i*})$  dans la règle d'apprentissage, où  $a_{i*}$  représente l'activité du neurone gagnant entre 0 et 1. En effet, dès que les deux neurones qui gagnent tour à tour sont suffisamment proches des stimuli (0 et 1), ils stabilisent automatiquement les poids de tout le réseau. C'est la *règle de Schyns*, qui, originellement [110], est définie lorsque les poids sont normés et qu'on utilise le produit scalaire comme mesure de proximité (et d'activité), mais qu'on peut étendre au cas général en remplaçant  $a_i$  par une fonction de la distance.

## 4.4 Projection non-linéaire

Dans le cas général, les vecteurs d'entrée  $\xi$  ont plus de deux composantes. Cela ne va en principe pas gêner l'auto-organisation, pour autant que la dimension de la grille soit adaptée au nombre de degrés de liberté de la distribution. Jusqu'à présent, on a vu des grilles en une ou deux dimensions (en une dimension, c'est une ligne), mais rien n'empêche d'utiliser des grilles à trois dimensions (des "cristaux") ou davantage. Cette dimension n'intervient que dans la définition du voisinage<sup>2</sup>.

Reprenons la distribution en "fer à cheval", qui nous avait servi à illustrer les limites des méthodes d'analyse de données linéaires, comme l'ACP (§ 2.4.4). La carte de Kohonen place les poids de ses neurones dans la distribution, comme on le voit à la figure 4.11. En considérant la position du vainqueur dans la grille  $\mathbf{y}_{i*}$

---

<sup>2</sup>Remarquons tout de même qu'avec une grille de dimension  $> 3$ , la fonction de voisinage devient lente à calculer; d'autre part, il faut beaucoup d'unités pour quantifier correctement le problème.

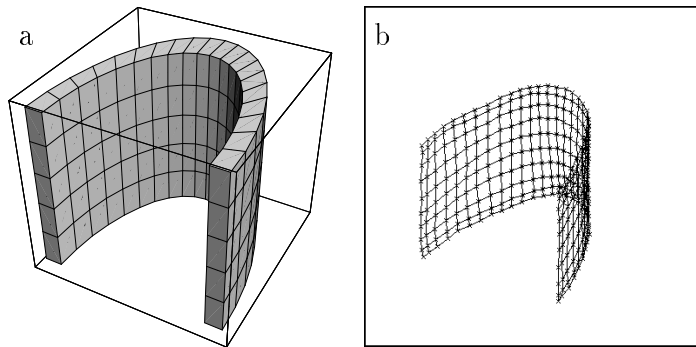


Figure 4.11: Projection non-linéaire par réseau de Kohonen. (a) Distribution en “fer à cheval”. (b) Placement des poids sur la surface gauche qui sous-tend la distribution.

pour chaque vecteur d’entrée, on obtient la projection non-linéaire qu’on avait souhaitée à la section 2.4.4 (figure 2.5, page 39).

De même, en analysant la distribution du problème de localisation (§ 1.1) à l’aide d’une carte de Kohonen, on retrouve les coordonnées du point à mesurer par projection dans la carte (les coordonnées du gagnant,  $\mathbf{y}_{i^*}$ , correspondent à un facteur près à celles du point qu’on voulait retrouver). On a ainsi un système de transformation de coordonnées, non-linéaire, qui permet de retrouver la position à partir d’un vecteur de 20 dimensions.

Les cartes auto-organisantes peuvent donc être vues comme une extension non-linéaire de l’Analyse en Composantes Principales [8]. La carte va aller se placer sur la variété des données et permettre une projection révélatrice de la structure de ces données.

Cette technique a rencontré un grand succès dans le monde des ingénieurs, qui ont trouvé un grand nombre d’applications à cette “cartographie de caractéristiques”, à partir de données complexes vers une grille de neurones. Des applications ont vu le jour dans un grand nombre de domaines, dont la robotique (transformation de coordonnées, apprentissage d’environnement [50, 103]), la reconnaissance des formes (appariement de formes, reconnaissance de parole [69]), le contrôle de procédés (par exemple les réseaux électriques [94], ou la fabrication de circuits VLSI [87]), la compression d’image [93] ou de palette de couleur [63]. La liste n’est pas exhaustive. Une revue d’applications a été donnée en 1990 par Kohonen dans [71], et depuis il en est apparu beaucoup d’autres. Remarquons cependant que parmi ces applications, un certain nombre n’utilisent pas la propriété de projection topologique, mais seulement le fait que l’algorithme de Kohonen fait de la quantification vectorielle. Par exemple, dans les applications mentionnées de compression d’image ou de palette de couleur, on ne s’intéresse absolument pas à la structure de la distribution. Les cartes auto-organisantes peuvent néanmoins être plus intéressantes qu’un autre algorithme de quantification vectorielle, parce qu’elles convergent relativement vite, même en dimensions élevées, du fait qu’il s’agit d’un algorithme “*winner-take-most*” (voir § 3.5). Toutefois, au vu des

performances d'algorithmes de quantification comme le PCVQA (§ 3.6), le SRS (§ 3.5.1) ou le “*Neural Gas*” (chapitre 5), nous aurions tendance à déconseiller les utilisations purement “quantification vectorielle” des cartes de Kohonen. Il vaut mieux réserver l'emploi de ces dernières pour leur propriété particulière de projection topologique avec réduction de dimension.

Ces cartes sont-elles donc l'outil universel d'analyse de données non-linéaire que nous appelions de nos vœux en introduction (§ 1.1) ? La réponse est non. Nous verrons à la section 4.6 que leur usage suppose des *a priori* bien gênants sur la forme et la dimension de la variété des données.

## 4.5 Représentations et mesures de qualité

En 1992, j'avais relevé la relative pauvreté des moyens d'analyse de la qualité d'auto-organisation et des méthodes de représentation des cartes de Kohonen [27, 28]. En effet, malgré les nombreuses applications, l'étude de la configuration topologique des poids a été peu étudiée.

En fait, les seuls résultats dont on disposait étaient ceux de Cottrell et son équipe, qui s'étaient attaqués à la démonstration de convergence vers un état ordonné dans le cas monodimensionnel, avec une densité uniforme [22] ou non-uniforme [12]. Ces démonstrations se basent sur la notion de *classes absorbantes*, c'est-à-dire de classes de configurations dont on ne ressort pas. Celles-ci se définissent par un nombre d'inversions dans l'ordre des poids plus petit ou égal à une certaine valeur. L'essence de ces démonstrations est de prouver que le nombre d'inversions ne peut que diminuer ou rester constant. La difficulté d'un tel schéma de démonstration quand  $n \geq 2$  est de définir une classe absorbante, ce qui semble pour l'instant sans espoir.

Pour le reste, la plupart des articles ne font des considérations que sur la qualité de la quantification vectorielle (“Kohonen à 0 voisins”), tout à fait indépendante de la qualité de l'organisation. Par exemple, dans [76], différentes variantes de l'algorithme sont comparées en fonction de la variance d'excitation des différents neurones, qui n'est rien d'autre qu'une mesure de la qualité de la quantification.

### 4.5.1 Représentation curviligne

Dans [29], on a proposé une représentation “curviligne” du réseau qui rappelle la représentation usuelle dans l'espace des poids, mais qui demeure applicable pour un nombre arbitraire de dimensions. L'idée consiste à considérer que la grille de neurones décrit une surface gauche dans l'espace des poids (comme dans la figure 4.11). On représente les neurones selon les coordonnées curvilignes de cette

surface, c'est-à-dire en fonction de leurs distances respectives dans l'espace des poids. On peut imaginer cette opération comme le "dépliage" de la grille et sa "mise à plat" sur la surface de dessin. En pratique, on commence par positionner les neurones des deux lignes médianes de la grille comme les axes sur lesquels on va s'appuyer, puis les unités de toute la grille, de proche en proche, du centre vers l'extérieur.

Considérons le quadrant supérieur droit (les autres se construisent de la même manière, par symétrie). Faisons référence aux unités par leur position dans la grille  $\mathbf{y}_i = [j, k]^T$ , au lieu de le faire par leur indice  $i$ . Notons le poids correspondant par  $\mathbf{x}_{j,k}$ . La position dans le dessin de cette unité sera  $[px(i, j), py(i, j)]^T$ . Plaçons l'origine des indices au centre de la carte;  $j$  est l'indice horizontal dans la grille ( $j \in \{0, \dots, w/2\}$  pour la moitié droite de la grille), et  $k$  l'indice vertical ( $k \in \{0, \dots, h/2\}$  pour la moitié supérieure), pour une grille de  $w \times h$  unités. On commence par fixer les unités des lignes médianes de la grille sur les axes du dessin à partir du centre :

$$px(0, 0) = py(0, 0) = 0 \quad (4.11)$$

$$px(0, k) = 0 \quad (4.12)$$

$$(a) \quad py(0, k) = py(0, k-1) + \|\mathbf{x}_{0,k} - \mathbf{x}_{0,k-1}\| \quad (4.13)$$

$$(b) \quad px(j, 0) = px(j-1, 0) + \|\mathbf{x}_{j,0} - \mathbf{x}_{j-1,0}\| \quad (4.14)$$

$$py(j, 0) = 0 \quad (4.15)$$

Ensuite, on positionne de proche en proche les unités sur la base de celles qui sont déjà placées :

$$(c) \quad px(j, k) = px(j-1, k) + \|\mathbf{x}_{j,k} - \mathbf{x}_{j-1,k}\| \quad (4.16)$$

$$(d) \quad py(j, k) = py(j, k-1) + \|\mathbf{x}_{j,k} - \mathbf{x}_{j,k-1}\| \quad (4.17)$$

Cette construction est schématisée à la figure 4.12. La figure 4.13 illustre la représentation curviligne avec une séquence d'organisation d'un réseau  $30 \times 10$  avec la distribution en "fer à cheval".

Un contre-exemple intéressant en grande dimension est lorsqu'on vient d'initialiser le réseau. La représentation curviligne donne alors l'impression (à tort bien sûr) que le réseau est déjà parfaitement bien organisé : elle donne l'image d'une grille parfaitement régulière ! Ceci provient du fait qu'on a initialisé les poids avec des valeurs aléatoires. Or on a vu à la section 2.1 qu'en grandes dimensions la norme d'un vecteur aux composantes aléatoires avait un écart-type faible ou même négligeable par rapport à sa moyenne. Ceci est également vrai

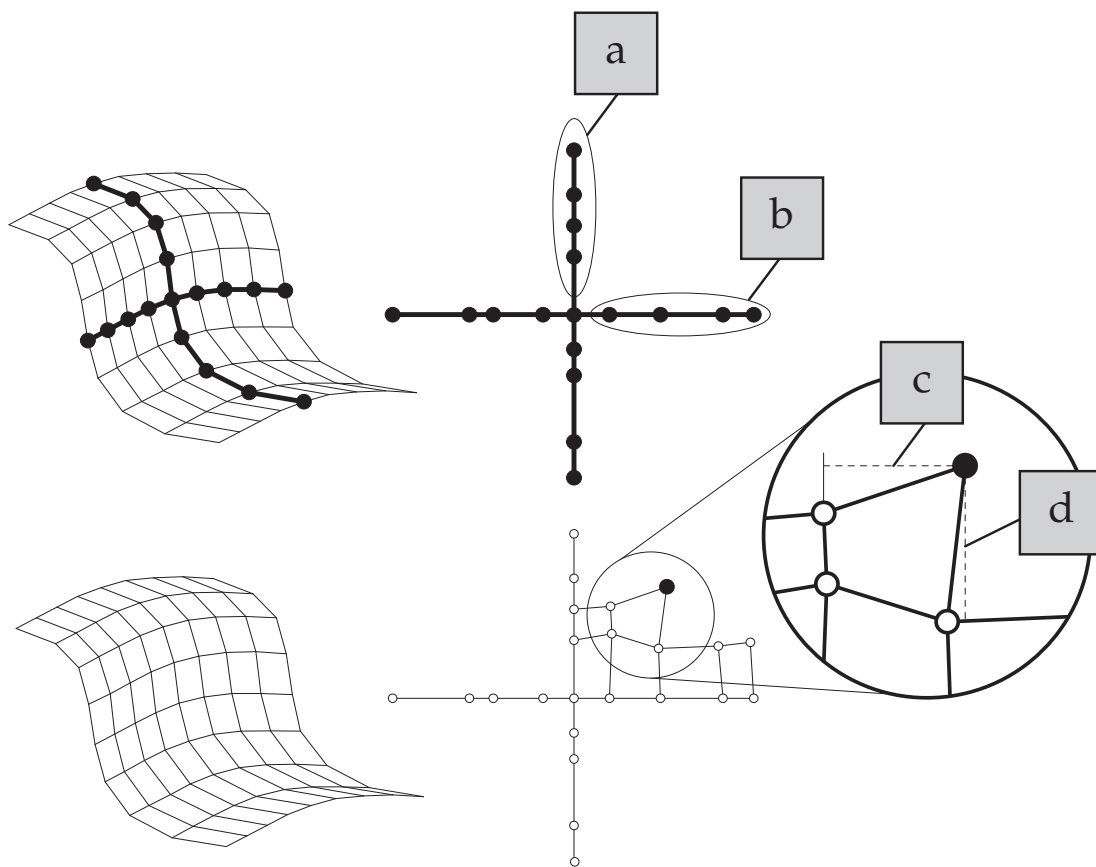


Figure 4.12: Construction de la représentation “curviligne”. Les labels ‘a’, ‘b’, ‘c’ et ‘d’ renvoient aux équations (4.11) à (4.17) (voir texte).

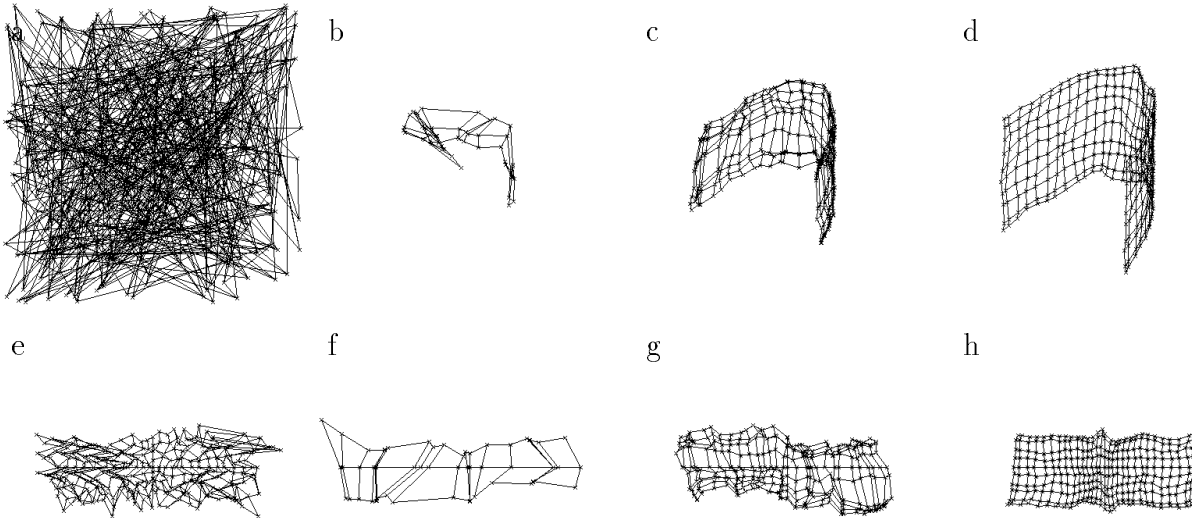


Figure 4.13: Séquence d'organisation d'un réseau  $30 \times 10$  avec la distribution en "fer à cheval". (a,b,c,d) Représentation usuelle dans l'espace 3D des poids. (e,f,g,h) Représentation curviligne.

pour la distance entre deux vecteurs aléatoires, ce qui explique cette anomalie dans la représentation.

### 4.5.2 Mesure de désordre $\Theta$

Toujours dans [29], nous définissons aussi une mesure scalaire de l'organisation qui représente le degré de désordre de la grille. Visuellement (quand on peut représenter le réseau dans l'espace des poids, en 2 ou 3 dimensions), on juge la qualité d'organisation par la régularité de la grille dessinée. Cette régularité peut se définir facilement dans n'importe quelle dimension.

Soit  $A_1$  l'ensemble des couples  $(i, j)$  d'unités adjacentes sur la grille :

$$A_1 = \left\{ (i, j) \mid \|\mathbf{y}_i - \mathbf{y}_j\| = 1 \right\} \quad (4.18)$$

La mesure  $\Theta$  est définie comme le rapport entre l'écart-type  $\sigma_1$  et la moyenne  $\mu_1$  des distances entre les poids de ces unités adjacentes, c'est-à-dire :

$$\Theta = \frac{\sigma_1}{\mu_1} \quad (4.19)$$

avec :

$$\mu_1 = \mathbb{E}_{i,j \in A_1} (\|\mathbf{x}_i - \mathbf{x}_j\|) \quad (4.20)$$

$$\sigma_1 = \sqrt{\text{Var}_{i,j \in A_1} (\|\mathbf{x}_i - \mathbf{x}_j\|)} \quad (4.21)$$

Si la grille est parfaitement régulière,  $\Theta$  tend vers 0. A l'inverse, plus la grille est désordonnée, plus  $\Theta$  est grand.

Cette mesure est scalaire et instantanée. Elle offre donc l'avantage de pouvoir être tracée au cours du temps. On observe alors une première étape croissante correspondant à la phase de mise en ordre du réseau (dépliement), suivie d'une décroissance représentant l'augmentation de régularité au cours du temps.

Pour les mêmes raisons que pour la représentation curviligne du chapitre précédent, la mesure  $\Theta$  donne une valeur très faible juste après l'initialisation aléatoire des poids. Autrement dit, elle nous indique (à tort toujours) que le réseau est parfaitement organisé juste après l'initialisation... Encore une fois, le théorème 2.1 fait sentir ses effets.

### 4.5.3 Représentation $dy - dx$

En proposant cette représentation [27], nous avons généralisé la mesure  $\Theta$  pour ne pas tenir compte uniquement des unités consécutives, mais de toutes les unités du réseau, quelle que soit leur distance de grille. Cette représentation est en effet la *distribution jointe* des distances de grille et de poids, qu'on appelle parfois aussi *diagramme de dispersion* de ces deux types de distances. Pour la tracer, on tire des couples aléatoires  $(i, j)$  de neurones. Chacun de ces couples est distant de  $dx(i, j) = \|\mathbf{x}_i - \mathbf{x}_j\|$  dans l'espace des poids, et de  $dy(i, j) = \|\mathbf{y}_i - \mathbf{y}_j\|$  dans l'espace de la grille. On affiche un point en  $[dy, dx]^T$  pour tous les couples de neurones possibles<sup>3</sup>. L'opération est illustrée à la figure 4.14. Le résultat est un nuage comme ceux des figures 4.15 et suivantes.

Lorsque la topologie n'est pas respectée,  $dy$  et  $dx$  ne sont pas corrélées, et on obtient un nuage diffus, sans polarisation, comme à la figure 4.15(b). En revanche, quand il y a une proportionnalité parfaite entre la distance des poids et la distance de grille, le nuage se ramène à une droite, signe d'une bonne organisation comme à la figure 4.15(d).

L'intérêt de cette représentation est multiple. Elle fournit une information très riche sur l'ordre des unités, quelle que soit la dimension de l'espace d'entrée. Elle ne souffre pas des effets du théorème 2.1, contrairement à la représentation curviligne ou à la mesure  $\Theta$  (en effet, lors de l'initialisation aléatoire de poids en grandes dimensions, le nuage est moins diffus que dans la figure 4.15(b); il peut éventuellement avoir un écart-type très faible par rapport à sa moyenne, mais l'absence d'organisation le laisse tout-à-fait horizontal). Elle renseigne sur

<sup>3</sup>Pas tout à fait... On en trace seulement un certain nombre.

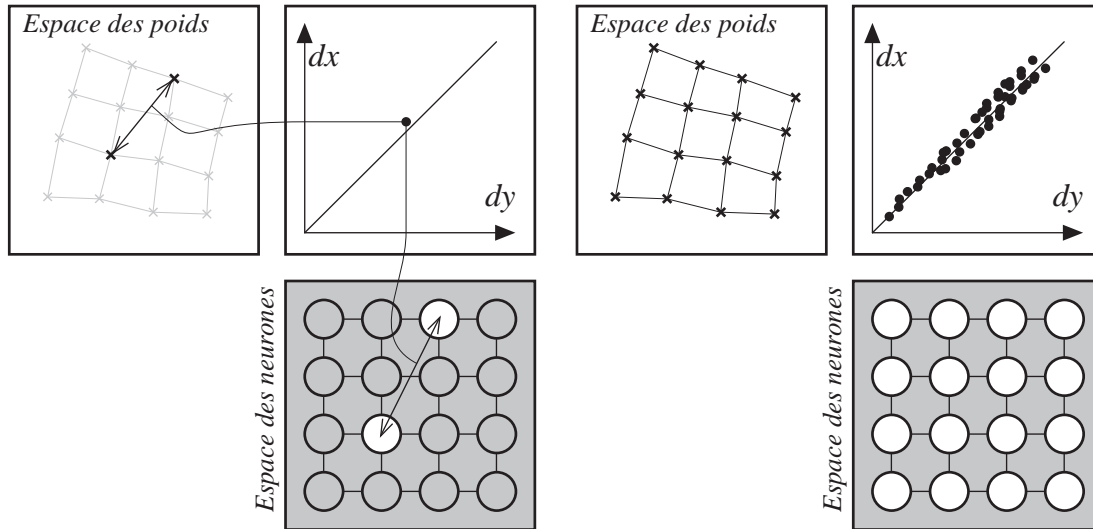


Figure 4.14: Construction de la représentation  $dy - dx$  : on trace la distribution jointe des distances entre les poids (axe vertical) et des distances de grille (axe horizontal). (Voir texte).

l'adéquation entre la forme et la dimension de la grille par rapport à celles de la variété des données. Enfin, elle est extrêmement simple à construire.

Dans la figure 4.16, on montre ce qui se passe quand la dimension de la grille est trop petite par rapport à la dimension intrinsèque des données. Ici, la grille est une ligne (1D), tandis que les données sont réparties sur un plan. Le comportement bien connu du réseau dans cette situation est de faire tendre les poids vers une courbe de Peano, qui couvre au mieux l'espace. Les nombreux replis qui en résultent font que la conservation de topologie est seulement (très) locale : trois ou quatre unités au mieux. La caractéristique  $dy - dx$  met en évidence ce problème par un nuage qui est bien organisé tout près de l'origine, mais qui devient rapidement diffus et globalement horizontal : après quelques unités, distance de grille et distance des poids ne sont plus corrélées. On constate d'ailleurs qu'on peut ajouter autant d'unités qu'on voudra sans changer fondamentalement le problème. Le nuage garde globalement la même forme quel que soit leur nombre (est-il *auto-similaire* ?). Il y a dans ce cas *inadéquation entre la dimension de la variété et celle de la grille*. On peut encore mentionner le concept de "longueur d'organisation" que nous avons introduit de façon intuitive dans [27] : il s'agit de la distance maximum  $dy$  depuis l'origine où les points de la représentation  $dy - dx$  sont "bien alignés en moyenne" (on peut définir formellement cette notion de "bon alignement" à l'aide de l'écart de la moyenne des  $dx(dy)$  par rapport à la droite représentée sur les figures, qui passe par l'origine et la moyenne des  $dx(dy = 1)$ ; cet écart devrait être inférieur à un certain pourcentage). Cette "longueur



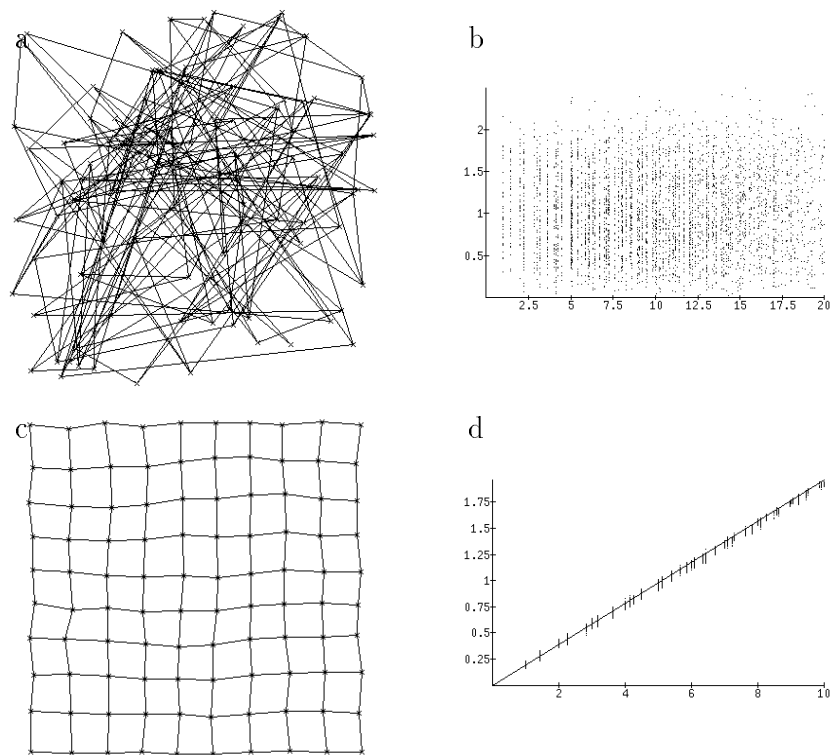


Figure 4.15: Représentation  $dy - dx$  (à droite). (a,b) à l'initialisation aléatoire des poids, il n'y a pas de conservation de topologie. (c,d) après apprentissage, si les conditions sont bonnes (dimension et forme de la carte adéquate pour la distribution, bon profil des paramètres, etc.), on obtient une organisation excellente.

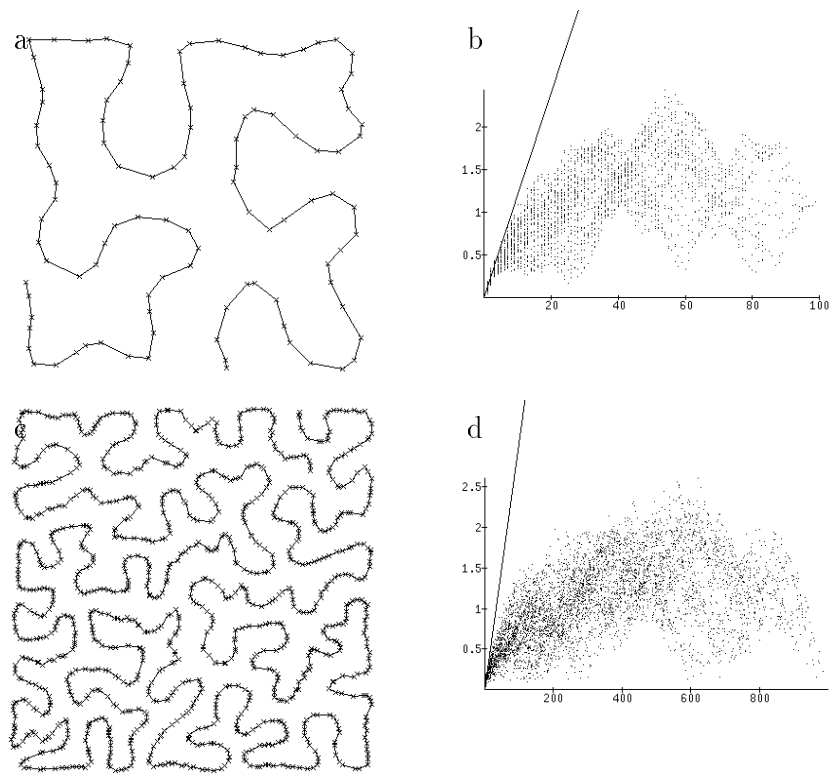


Figure 4.16: La représentation  $dy - dx$  montre la mauvaise organisation qui résulte de conditions initiales inappropriées. Ici, la dimension de la carte (1) est trop petite par rapport à celle de la variété de la distribution (2). L'augmentation du nombre de neurones ne change rien à l'affaire.

d'organisation" indique donc le diamètre moyen (en nombre de neurones) des portions de carte bien organisées. Elle précise la notion de "localité" dans le respect de la topologie obtenu par le réseau de Kohonen.

D'autres cas d'école sont encore donnés à la figure 4.17. La projection d'un cercle sur une ligne donne une organisation assez bonne, mais comportant un important repli. La projection de la distribution bimodale (deux carrés) sur une grille rectangulaire donne une représentation  $dy - dx$  montrant qu'il y a plusieurs modes. La partie inférieure du nuage (assez droite) correspond à l'organisation intra-classe, qui est bonne. Le reste du nuage correspond au respect de topologie inter-classe, qui n'est pas très bon. Par comparaison, lors de la projection de la même distribution sur une ligne, la représentation  $dy - dx$  montre également l'aspect bimodal de la distribution, mais les éléments qui composent le nuage sont différents. Notamment, la partie correspondant aux relations intra-classe est similaire au nuage de la courbe de Peano.

Pour illustrer l'utilité de cette représentation, nous allons analyser l'organisation d'une carte bidimensionnelle dans un espace à 20 dimensions, en comparant les résultats obtenus en utilisant soit une distribution structurée selon deux degrés de liberté, soit une distribution non structurée (à 20 degrés de liberté). La distribution structurée est celle du problème de localisation sur un plan à l'aide de 20 capteurs (§ 1.1), dont on a vu que le nombre de degrés de liberté est 2. La distribution non structurée est simplement uniforme dans  $[0, 1]^{20}$ ; toutes les composantes sont indépendantes. Dans les deux cas, on a utilisé un réseau de  $20 \times 20$  unités. La figure 4.18 montre les résultats obtenus après 10 000 itérations. Dans la partie gauche, on a donné la représentation curviligne des deux réseaux. On constate que celle-ci ne met pas en évidence les replis formés par le réseau alors qu'il tente de représenter l'espace à 20 dimensions avec les deux dimensions de sa grille. Idem pour la mesure  $\Theta$  ou toute autre mesure locale. De même, une mesure de quantification (par exemple la variance d'excitation des neurones) ne donnerait aucune différence significative entre les deux cas. En revanche, la représentation  $dy - dx$  montre clairement le problème de replis, présents avec la distribution non structurée et absents (ou très faibles) dans le cas de la distribution structurée. Dans le cas de la distribution uniforme à 20 degrés de liberté, la "longueur d'organisation" est à peu près de 2 ou 3 unités, ce qui est très local.

Il est bien entendu que sur la base de la caractéristique  $dy - dx$  on peut calculer toutes sortes de valeurs statistiques, comme la variance de  $dx$  en fonction de  $dy$ , ou bien pondérée par une fonction décroissante de  $dy$ , ou encore la distance  $dy$  en dessous de laquelle les  $dx$  correspondants ont une variance inférieure à une certaine valeur (ce qui correspondrait à une autre définition de notre "longueur d'organisation"). L'algorithme "VQP" que nous proposons plus loin (chapitre 6) peut se définir comme une minimisation d'une de ces statistiques sur le nuage

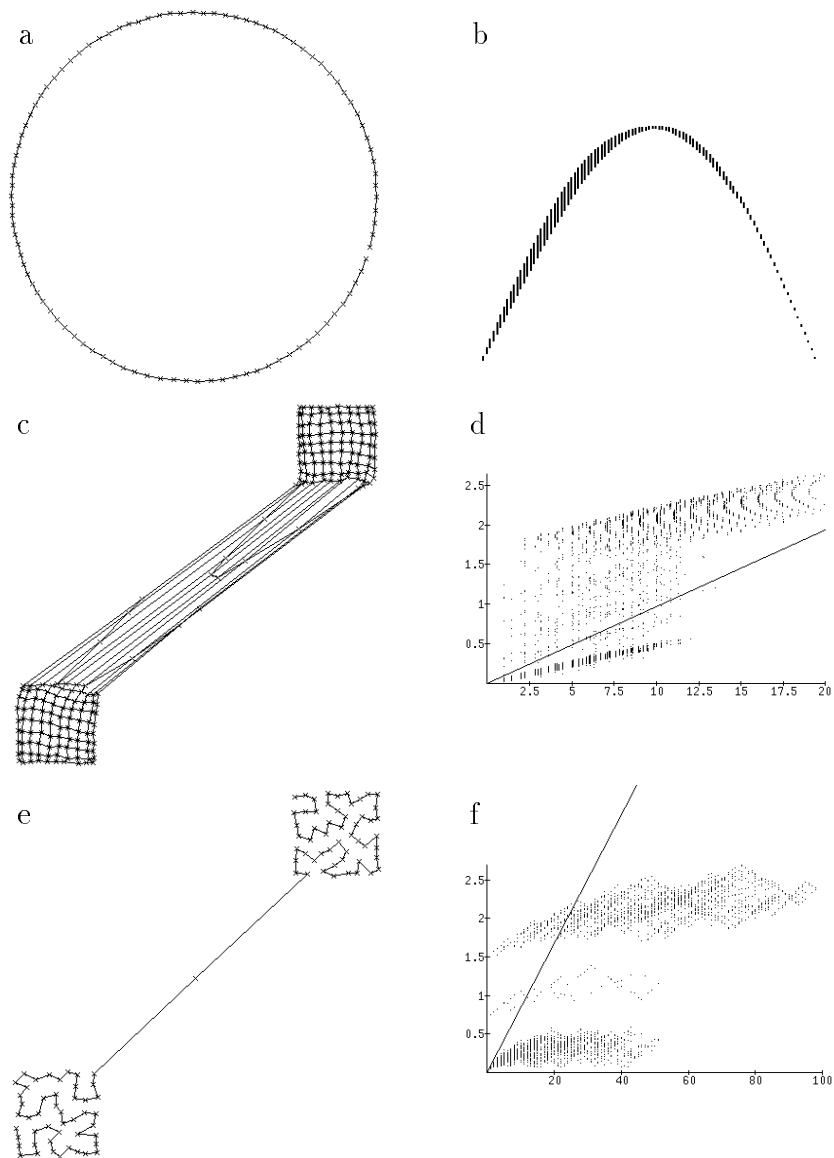


Figure 4.17: Encore des exemples de  $dy - dx$  (voir texte).

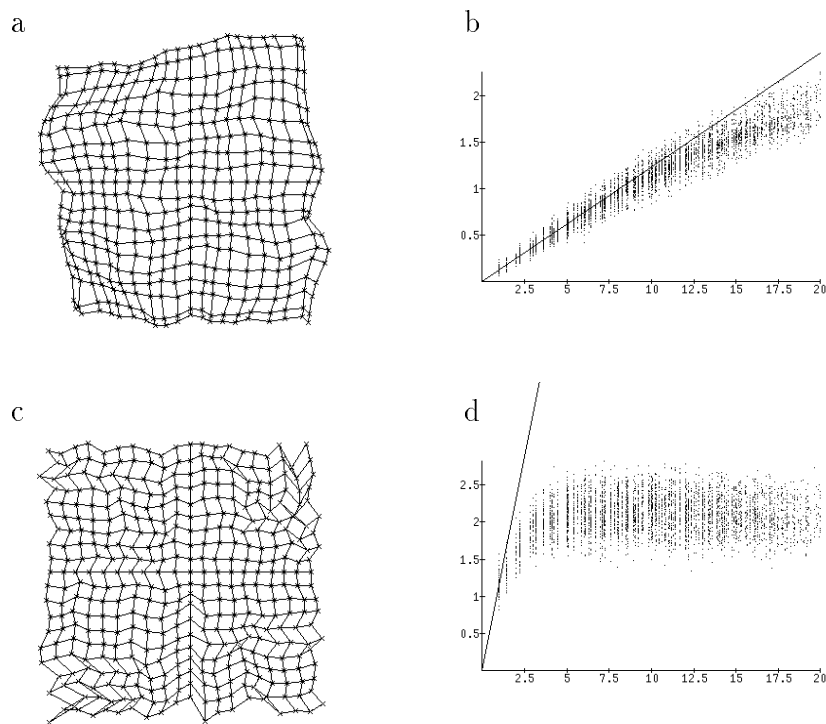


Figure 4.18: Comparaison des résultats obtenus sur deux distributions en 20 dimensions, l'une structurée (a,b), l'autre non structurée (c,d). La première est la distribution de localisation sur un plan à l'aide de 20 capteurs de distance (§ 1.1). La seconde est uniforme dans les 20 dimensions. A gauche, la représentation curviligne n'indique pas clairement de différence. A droite par contre, la représentation  $dy-dx$  montre bien que dans le deuxième cas la dimension de la carte est inappropriée (il n'y a conservation de topologie que le long de 2 ou 3 unités adjacentes, et plus loin il y a des replis).

## 4.6 Limitations et problèmes non-résolus

Les limitations et problèmes des cartes auto-organisantes sont multiples. Ce sont essentiellement :

- La forme et la dimension de la grille définies *a priori*.
- La présence possible d'unités mortes.
- La projection discrète.
- L'occultation des points isolés (événements *rare*s).

On va expliquer ces problèmes dans les sections qui suivent.

### 4.6.1 Forme de la carte inadaptée, unités mortes

Le problème le plus sérieux est que la forme de la carte doit être définie *a priori*. Le plus souvent, il s'agit d'une grille à maille rectangulaire ou hexagonale. Dans la plupart des applications réelles (celles qui ne sont pas des exemples pédagogiques), la forme du sous-espace qui sous-tend la distribution (la forme de la variété) est inconnue et ne peut souvent pas être estimée parce que la dimension de l'espace d'entrée est grande. Faute de mieux, on en est alors réduit à choisir une carte de forme carrée. Ceci est très ennuyeux, parce qu'en pratique il y a très peu de chances que la forme ainsi choisie *a priori* convienne pour couvrir la distribution des données. En considérant l'algorithme de Kohonen comme un outil d'analyse de données, il faudrait déjà connaître une partie du résultat escompté (la forme de la variété) pour fixer judicieusement la forme de la carte, qui est une des conditions initiales !

Dans l'exemple de projection non-linéaire sur la distribution en forme de fer à cheval (§ 4.4, page 80), on a fixé de façon adéquate le rapport largeur/hauteur de la grille à  $1/3$ , ce qui correspond au rapport entre la hauteur du fer à cheval et la longueur de son abscisse curviligne, qu'on connaissait dans cet exemple simple. En abordant l'étude de ce fer à cheval en supposant qu'on ne sait rien de sa forme, et en prenant une grille carrée, on n'obtient pas du tout une projection aussi intéressante. Au contraire, la figure 4.19(a) indique que les coins du rectangle qui sous-tend la distribution sont projetés au milieu des côtés de la carte : le respect de la topologie n'est pas fameux.

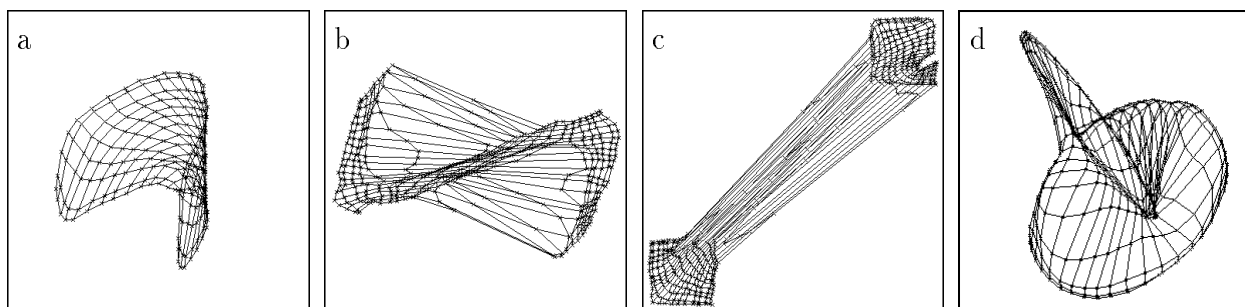


Figure 4.19: Quelques contre-exemples des cartes de Kohonen. La forme carrée de la carte ne convient ni à la distribution en fer à cheval (a), ni à celle en forme d’hélice (b). Pour la distribution dans deux boîtes (c), il faudrait deux cartes carrées. Enfin, dans le cas de deux anneaux imbriqués (d), il faudrait plutôt deux réseaux monodimensionnels bouclés. Malheureusement, quand l’entrée est en grande dimension, on ne voit pas ceci aussi facilement.

D’autres exemples simples montrent à quel point une forme rectangulaire, hexagonale ou carrée peut être inadaptée pour la représentation de certaines distributions (figure 4.19). Dans ces exemples, on voit également le problème des unités mortes (les unités qui pointent en dehors de la distribution), qui provient aussi de l’inadéquation de la forme de la carte à celle des données, ou parfois simplement d’une descente trop rapide des paramètres  $\alpha(t)$  et  $\gamma(t)$  (gain et voisinage), même dans le cas de distributions convexes. Si l’on essaie d’éviter ces unités mortes par une technique de fatigue ou de conscience (§ 3.4), la conservation de topologie sera encore moins bonne.

Ce problème de forme de carte est à l’origine de plusieurs propositions de réseaux “auto-organisés” sans forme prédéfinie, comme les cartes évolutives de Fritzke [44, 46, 45], le “*Neural Gas*” (chapitre 5), les cartes multiples avec test d’appartenance [121], ou encore un réseau où la structure de voisinage est définie par le “*minimal spanning tree*” (MST) des poids et est réactualisée périodiquement [62]. Malheureusement, dans tous ces cas, l’intérêt des cartes de Kohonen comme outil d’analyse de données par projection et réduction de dimension est perdu. En effet, si l’on peut montrer des liens entre des unités dans une représentation des poids et produire ainsi des illustrations du meilleur effet (qui ressemblent à la représentation usuelle du réseau de Kohonen, mais sans unités mortes), on ne dispose par contre plus d’une structure régulière vers laquelle projeter les données de façon révélatrice. On est alors absolument incapable de tirer profit de ces liens. On ne peut pas parler de respect de topologie, puisqu’il n’y a qu’un espace : celui des poids.

### 4.6.2 Projection discrète

Dans l'algorithme de base (et dans la plupart des applications du réseau de Kohonen à l'analyse de données), la projection d'un échantillon  $\xi$  de la base de données se fait de façon discrète dans la carte : cette projection est la position  $\mathbf{y}_{i^*}$  du neurone gagnant lors de la présentation de  $\xi$ . La précision ainsi obtenue n'est pas excellente et incite souvent les utilisateurs à augmenter le nombre d'unités dans la carte pour améliorer les résultats.

Une solution est d'effectuer une interpolation entre les neurones les plus actifs, c'est-à-dire de définir la projection de  $\xi$  par le barycentre des activités des neurones [52, 53, 18, 17]. Cette activité est calculée par une fonction noyau sur la distance entre  $\xi$  et  $\mathbf{x}_i$ , généralement une gaussienne. Comme pour les RBF, la précision de l'interpolation dépend fortement de la façon dont on a adapté le rayon des noyaux : des noyaux trop étroits vont donner une projection où les points sont resserrés par paquets autour des positions  $\mathbf{y}_i$  des neurones, tandis que des noyaux trop larges vont concentrer la projection vers le centre de gravité des vecteurs-poids. En outre, comme la plupart des méthodes de quantification vectorielle, l'algorithme de Kohonen ne place pas de vecteurs tout au bord de la distribution. Il reste donc une frange au bord de la distribution qui est mal représentée.

Une autre possibilité a été présentée par Göppert [49]. Il s'agit d'un procédé d'interpolation où la différence entre  $\xi$  et  $\mathbf{x}_{i^*}$  est projetée sur les axes qui passent par  $\mathbf{x}_{i^*}$  et les  $\mathbf{x}_i$  des neurones voisins du gagnant  $i^*$ . On se base sur ces projections de la différence pour placer le point dans  $\mathbf{y}$  correspondant à la projection de  $\xi$ . Cette méthode permet, dans une certaine mesure, d'améliorer la précision d'interpolation et aussi de faire de l'extrapolation, en tout cas à faible distance. Cependant, dans les cas où la structure des données est assez pliée, il faut limiter le mécanisme à un petit nombre de voisins, sans quoi la projection est mauvaise [49].

### 4.6.3 Occultation de points isolés (événements rares)

Bien que l'intérêt majeur des cartes de Kohonen soit la conservation de topologie qu'elles peuvent offrir, plusieurs applications ne les utilisent que pour leur aspect de quantification vectorielle. La raison de ce fait tient probablement à la vitesse assez grande de convergence de l'algorithme, ainsi qu'à l'impression de régularité dans l'emplacement des vecteurs-poids dans les cas simples, qui donne à penser que la qualité de quantification est bonne (ce qui est parfois vrai, mais pas toujours).

Lorsqu'on ne s'intéresse pas à l'information sur la structure de la distribution (que révèle la projection topologique fournie par l'algorithme), mais uniquement à la quantification vectorielle, il est inutile d'utiliser l'algorithme de Kohonen. En



effet, des algorithmes de QV très efficaces en rapidité et en qualité sont disponibles et donnent de bien meilleurs résultats. Par exemple, ils ne souffrent pas du problème d'unités mortes qu'on a vu précédemment.

Un problème important de la plupart des méthodes de quantification vectorielle, mais qui est encore renforcé dans le cas des cartes de Kohonen en raison des liens entre les unités, est la mauvaise représentation des événements rares. Un exemple concret va illustrer ce problème.

En étudiant une application de compression de palette de couleur par réseau de Kohonen, en vue de représenter les pixels d'une image à l'aide d'un nombre réduit de bits (par exemple en n'utilisant que 8 bits/pixels au lieu de 24), on a constaté que des images pouvaient perdre certains détails importants. Par exemple, sur une photographie de New-York (non illustrée ici, le support étant en noir et blanc), il y a quelques taxis jaunes qui, bien que tout petits en raison de l'éloignement, sont très visibles parce qu'ils se détachent très nettement du fond. Ces taxis sont à peu près les seuls pixels jaunes de l'image (qui est plutôt dans les tons gris, bleus et verts). Ces pixels jaunes sont très peu nombreux par rapport au reste de l'image qui représente des centaines de milliers de pixels. Lors de la quantification de l'espace tridimensionnel rouge-vert-bleu (RGB) de la distribution de pixels par un réseau avec 256 neurones, en vue d'obtenir une palette de 256 couleurs parmi 16 millions (passage de 24 à 8 bits/pixel), la couleur jaune n'est pas représentée, c'est-à-dire que les quelques pixels jaunes de l'image sont en nombre insuffisant pour qu'un neurone les représente. En revanche, les tonalités grises, bleues et vertes sont représentées avec beaucoup de nuances, puisque tous les neurones s'en occupent. Dans l'image en 8 bits, les taxis jaunes ont disparu, parce qu'ils sont devenus gris (la couleur disponible la plus proche) et se confondent avec la route.

Cet exemple montre d'une façon dramatique la contradiction qu'il y a entre le critère de qualité généralement utilisé dans les méthodes de quantification vectorielle et ce qu'on souhaite parfois obtenir. La distorsion quadratique de reconstruction n'est qu'une estimation de l'erreur moyenne. Sa minimisation conduit à produire des "*codebooks*" qui donnent une faible erreur pour le plus grand nombre de points, et une erreur forte mais peu fréquente pour les points isolés. En fait, cela va parfaitement à l'encontre de la théorie de l'information, qui dit que les événements rares sont les plus porteurs d'information<sup>4</sup> !

Cela complète la discussion faite en introduction de la section 3.7 et souligne le besoin d'une méthode telle que le CLR permettant une quantification vectorielle

---

<sup>4</sup>Sauf quand les points isolés sont du bruit, auquel cas il est approprié de les éliminer. Une fois de plus, on voit là l'importance des connaissances du problème, qui dans le cas précis permettent de répondre à la question : "les points rares sont-ils du bruit ou au contraire sont-ils très importants ?".

équitable de l'*espace* et non de la *densité* de distribution sur cet espace. En effet, dans le cas de “l’image aux taxis jaunes”, les milliers de pixels gris-bleu du fond occupent dans l’espace RGB une “boule” du même volume environ (mais plus dense) que les quelques pixels jaunes, et une quantification équitable de l’espace permettrait d’accorder à ceux-ci autant d’importance qu’à ceux-là.

# Chapitre 5

## “Neural Gas”

### 5.1 Algorithme original

Pour éviter les problèmes du réseau de Kohonen qu'on a vus à la section 4.6, Martinetz *et al.* [89, 119] ont proposé un algorithme où aucune structure n'est fixée *a priori*. Plus exactement, la structure de voisinage est redéfinie à chaque itération, en fonction de la proximité entre les vecteurs-poids et le vecteur d'entrée  $\xi$ . Il s'agit d'un voisinage linéaire où le gagnant se trouve à une extrémité et le “perdant” à l'autre. Entre ces deux extrémités, les unités se suivent en fonction de la proximité de leur vecteur-poids et de l'entrée. Bien sûr, les unités elles-mêmes ne sont pas déplacées. On leur attribue simplement un *rang* qui indique combien d'unités sont plus proches de  $\xi$ . Comme on l'a déjà vu à la section 3.5.4, ce rang est attribué par un simple tri des unités en fonction de leur proximité par rapport au vecteur d'entrée  $\xi$ . En reprenant la notation de la section 3.5.4 ( $k(i)$  est le rang de l'unité  $i$  et  $i(k)$  le numéro de l'unité de rang  $k$ ), on a :

$$\delta_0 < \delta_1 < \dots < \delta_k < \delta_{k+1} < \dots < \delta_{N-1} \quad (5.1)$$

avec la distance de rang  $k$

$$\delta_k = \left\| \xi - \mathbf{x}_{i(k)} \right\| \quad (5.2)$$

L'unité  $i$  de rang  $k(i) = 0$  est le gagnant euclidien, celle de rang  $k = 1$  vient juste après, etc.

On a vu qu'il s'agissait d'un algorithme de type “*winner-take-most*”, et non “*winner-take-all*”, c'est-à-dire que tous les neurones (pas seulement le gagnant) sont adaptés à chaque itération. L'adaptation du vecteur-poids de chaque unité  $i$  en fonction de son rang  $k(i)$  s'écrit :

$$\Delta \mathbf{x}_i = \alpha(t) \exp \left( -\frac{k(i)}{\lambda(t)} \right) (\xi - \mathbf{x}_i) \quad (5.3)$$

L'unité dont le facteur d'adaptation est le plus grand est le gagnant euclidien  $i^*$  pour lequel le rang  $k$  est 0.

Les paramètres  $\alpha(t)$  et  $\lambda(t)$  sont les mêmes que dans l'algorithme de Kohonen :  $\alpha(t)$  est comme d'habitude le gain d'adaptation, et  $\lambda(t)$  est le “rayon” de l'adaptation. Les deux paramètres diminuent au cours du temps, selon un profil en

$$r(t) = r_i \left( \frac{r_f}{r_i} \right)^{t/t_{\max}}, \quad (5.4)$$

qui est une exponentielle qui passe par  $r_i$  (paramètre initial) à  $t = 0$  et par  $r_f$  (paramètre final) à  $t = t_{\max}$ .

Comme on l'a expliqué à la section 3.5, le principe-clé qui permet d'éviter un collage des unités dans un schéma d'adaptation de type “*winner-take-most*” (où l'on veut adapter toutes les unités à chaque présentation de  $\xi$ ) est un découplage entre la distance euclidienne  $\|\xi - \mathbf{x}_i\|$  qui sépare le vecteur-poids d'un neurone du vecteur d'entrée, et le gain d'adaptation  $G(i)$ . Ici, ce découplage entre distance et facteur d'adaptation provient de ce que leur relation n'est pas continue, mais *ordinaire*. Quelle que soit la proximité des vecteurs-poids, la différence minimum de rang est 1 (pour deux unités *ex æquo*, l'ordre est choisi au hasard).

En outre, dans leur premier article [89], Martinetz et Schulten introduisent une méthode pour visualiser la proximité des vecteurs-poids  $\mathbf{x}_i$ . Il s'agit de liens placés à chaque itération entre l'unité de rang 0 et celle de rang 1. L'évolution de ces liens est régie selon deux règles :

- A chaque itération, un lien est *créé* ou *rafraîchi* entre  $i(k = 0)$  et  $i(k = 1)$ .
- Un lien qui n'a pas été rafraîchi dans les  $T(t)$  dernières itérations disparaît.  $T(t)$  est la durée de vie des liens et suit la forme générale d'évolution des paramètres (5.4). La valeur initiale de cette durée de vie est faible. Ainsi, quand les poids bougent beaucoup (parce que  $\alpha(t)$  et  $\gamma(t)$  sont encore importants), les liens sont assez volatils. Quand le réseau se stabilise, les liens ont une durée de vie  $T$  plus grande. Si tout se passe bien (bonne stabilité des poids et durée de vie adéquate), les liens forment une bonne approximation de la triangulation de Delaunay des  $\mathbf{x}_i$ .

L'algorithme ainsi défini est une méthode de quantification vectorielle assez rapide, avec en plus une création de liens qui captent les relations de voisinage existant entre les vecteurs-codes. Ces liens peuvent par exemple être comptés pour qu'on ait une information sur la dimension locale autour de chaque neurone. Dans la figure 5.1, par exemple, on voit une distribution dont la dimension locale est différente selon l'endroit où l'on se place. A l'intérieur de la partie de dimension 1, chaque neurone a développé deux connexions, sauf celui qui se trouve sur la

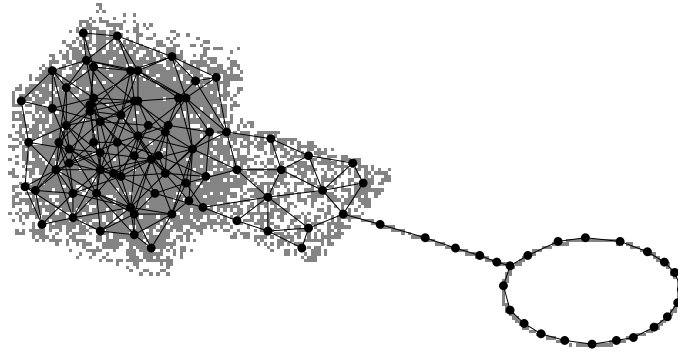


Figure 5.1: Distribution dont la dimension locale est 1, 2 ou 3 suivant l'endroit où l'on se place, proposée par Martinetz et Schulten [89].

bifurcation à l'endroit où la ligne arrive sur le cercle, qui possède 3 connexions. Dans la partie bidimensionnelle de la distribution, il y a en moyenne 6 liens par unité, et 14 dans la partie en 3 dimensions.

Cependant, contrairement aux cartes de Kohonen où les liens reflètent une structure régulière permettant une projection dans le sous-espace correspondant, aucun espace adjoint ne peut être utilisé ici pour faire une telle projection avec réduction de dimension. En dehors de la fonction d'information sur la dimension locale (mais qui peut également être trouvée par les méthodes vues à la section 2.5.2), on n'a pas encore vu d'autre application de ces liens. Toutefois, comme nous l'avons montré à la section 3.7, le principe de création des liens dynamiques n'est pas restreint au cadre du Neural Gas. Il peut au contraire être utilisé dans n'importe quel algorithme compétitif, comme nous l'avons fait dans l'algorithme CLR pour obtenir une structure discrète de voisinage servant de support à une régularisation (§ 3.7).

## 5.2 Algorithme modifié

Dans [31], nous avons proposé quelques modifications à l'algorithme du Neural Gas :

- Dans nos simulations, nous avons remarqué que des unités mortes sont produites par l'algorithme original, sauf si l'on fait diminuer très lentement les paramètres ( $t_{\max} > 50\,000$  itérations). Nous mettons en œuvre un mécanisme de fatigue similaire à ceux de la section 3.4 qui élimine ces unités mortes, même quand on cherche une convergence rapide.

- D’autre part, la phase de l’algorithme la plus coûteuse est la détermination du rang  $k(i)$  de chaque unité à l’aide d’un tri des distances. Comme le font remarquer Martinetz *et al.*, la modification des poids des unités dont le rang  $k(i) \gg \lambda(t)$  est négligeable. Donc, pour les faibles valeurs de  $\lambda(t)$ , on peut chercher à ne trier que les unités dont le gain sera appréciable.
- Enfin, l’apprentissage contraint dans le temps (par le profil prédéfini de descente des paramètres selon (5.4)) est remplacé par un apprentissage perpétuel, qui s’auto-stabilise quand la distribution est stationnaire, mais qui peut redevenir fortement adaptatif en cas de changement de distribution. On a donc un *réglage automatique des paramètres d’apprentissage*.

L’algorithme qui résulte est une méthode de quantification vectorielle rapide, dont la convergence dépend seulement de la stationnarité de la distribution et qui ne laisse pas d’unités mortes.

Dans les sections ci-dessous, on expose cette implantation particulière du Neural Gas sur machine séquentielle, et pour la commodité du développement on reprend la notation introduite au chapitre 3.5 pour les algorithmes de type “*winner-take-most*” :

$$\Delta \mathbf{x}_i = \alpha(t)G(i)(\boldsymbol{\xi} - \mathbf{x}_i) \quad (5.5)$$

$$G(i) = \exp\left(-\frac{k(i)}{\lambda(t)}\right) \quad (5.6)$$

### 5.2.1 Tri limité

En observant (5.6), on note que  $G(i) \cong 0$  pour les unités de rang  $k(i) \gg \lambda(t)$ . Il est donc adéquat de limiter le tri des distances seulement pour les  $K$  premières valeurs. Disons par exemple  $K = 3\lambda + 1$  (“+1” pour avoir toujours au moins le gagnant). Ensuite, on n’adapte que les unités qui ont été triées (les autres ayant un rang inconnu, mais plus grand que  $K$  et donc un facteur d’adaptation négligeable).

Parmi les algorithmes de tri, l’un des plus largement utilisés est sans doute le “*quick-sort*”, qui est une méthode “en place” et a une complexité en moyenne de  $O(N \log N)$ . Il se formule aisément de façon récursive :

- La liste à trier est  $L = \{a_1, \dots, a_N\}$ . Le premier élément  $a_1$  sera appelé *pivot*.
- On partitionne la liste en deux :  $L_1 = \{a_{i \neq 1} \mid a_i < a_1\}$  et  $L_2 = \{a_{i \neq 1} \mid a_i \geq a_1\}$ , ce qui est fait rapidement et en place, par des déplacements au sein de la liste  $L$ . La première partie de  $L$  est  $L_1$ , la fin est  $L_2$ , et entre  $L_1$  et  $L_2$ , on replace  $a_1$ .

- On recommence ce processus séparément pour  $L_1$  et  $L_2$ , jusqu'à ce que les listes ne contiennent plus qu'un élément<sup>1</sup>.

On peut facilement adapter cet algorithme pour qu'il ne fournisse que les  $K$  premières valeurs triées d'une liste. Pour ce faire, on "coupe" la récursion de l'algorithme sur certaines sous-listes. On peut en effet abandonner le tri de  $L_2$  si celle-ci est entièrement située dans les positions  $i > K$  de  $L$ , car on est sûr que les  $K$  premières valeurs se trouvent dans la première partie ( $L_1$  plus le pivot). Le *quick-sort* ainsi modifié a une complexité moyenne de  $K \log N$  au lieu de  $N \log N$ , ce qui est très intéressant pour les petites valeurs de  $K$ . La mise en œuvre de cette méthode de tri augmente très considérablement la vitesse du Neural Gas.

### 5.2.2 Suppression des unités mortes

Suivant ce qui a été présenté à la section 3.4, pour éviter les unités mortes, on va favoriser les neurones qui gagnent peu souvent. Considérons le gain d'adaptation moyen de l'unité  $i$ , estimé à l'aide d'un filtre passe-bas :

$$p_i \leftarrow p_i + \frac{1}{\tau} (G(i) - p_i) \quad (5.7)$$

où  $\tau$  est la constante de temps du filtre (à peu près 1000 dans nos simulations).

Si, pour chaque unité  $i$ , le rang  $k(i)$  est équiprobable dans  $\{0, \dots, N-1\}$ , alors  $p_i$  converge vers l'espérance de  $G(i)$ , c'est-à-dire la valeur  $\frac{1}{N} \sum_{k=0}^{N-1} \exp(-k/\lambda)$ .

Par contre, si le rang n'est pas équiprobable, on voit que les unités qui sont adaptées trop rarement ont un  $p_i$  faible, alors que celles qui gagnent trop souvent ont comparativement une valeur  $p_i$  trop élevée. Pour favoriser les unités peu sollicitées, l'opération de tri doit tenir compte des distances pondérées par les  $p_i$ , et (5.1), respectivement (5.2), sont remplacées par :

$$q_0 < q_1 < \dots < q_k < q_{k+1} < \dots < q_{N-1} \quad (5.8)$$

avec :

$$q_k = p_i \left\| \boldsymbol{\xi} - \mathbf{x}_{i(k)} \right\| \quad (5.9)$$

L'élection des unités avec un gain moyen trop bas est facilitée, et les unités mortes ne sont plus observées. Il s'agit d'une généralisation des mécanismes de fatigue ou de conscience (§ 3.4) au cas du "*winner-take-most*".

---

<sup>1</sup>Ou suffisamment peu d'éléments pour être triées par tri séquentiel, lequel, bien qu'étant de complexité  $O(N^2)$ , peut être plus rapide quand le nombre d'éléments  $N$  est très petit (environ 10).

### 5.2.3 Apprentissage continu (réglage automatique des paramètres)

Les paramètres  $\alpha$  et  $\lambda$  calculés selon (5.4) :

$$r = r_i \left( \frac{r_f}{r_i} \right)^{t/t_{\max}} \quad (5.10)$$

suivent un profil en exponentielle décroissante prédéterminé, avec une valeur initiale  $r_i$  à l’itération  $t = 0$ , et une valeur “finale”  $r_f$  à l’itération  $t = t_{\max}$ . Comme ce profil dépend uniquement du temps, il n’y a pas moyen, après  $t_{\max}$  itérations, de “revenir” à de grandes valeurs des paramètres afin de rendre le réseau à nouveau adaptatif quand la distribution change.

L’idée ici est de supprimer ce profil préétabli et de “détecter” automatiquement quand la quantification est “bonne” pour diminuer les paramètres d’apprentissage. Pour cela, on conserve la forme générale de (5.10), mais on remplace  $t/t_{\max}$  par une mesure convenable de la convergence, définie dans le même intervalle, c’est-à-dire dans  $[0, 1]$ . Une telle mesure peut être obtenue en considérant l’ensemble des valeurs  $p_i$  calculées par (5.7). Comme déjà expliqué, une “bonne” quantification (avec des zones de Voronoï équiprobables) devrait pousser tous les  $p_i$  à prendre à peu près la même valeur. Donc leur écart-type doit être faible. Mieux que l’écart-type, le quotient :

$$P = \frac{\min(p_i)}{\max(p_i)} \quad (5.11)$$

indique une convergence incomplète même si une seule unité reste en dehors de la distribution. Pour des  $p_i$  dispersés (situation typique d’une mauvaise quantification),  $P$  prend une petite valeur. A l’inverse, si tous les  $p_i$  sont égaux, alors  $P = 1$ . On a donc choisi comme forme des paramètres :

$$r = r_i \left( \frac{r_f}{r_i} \right)^{P/P_0} \quad (5.12)$$

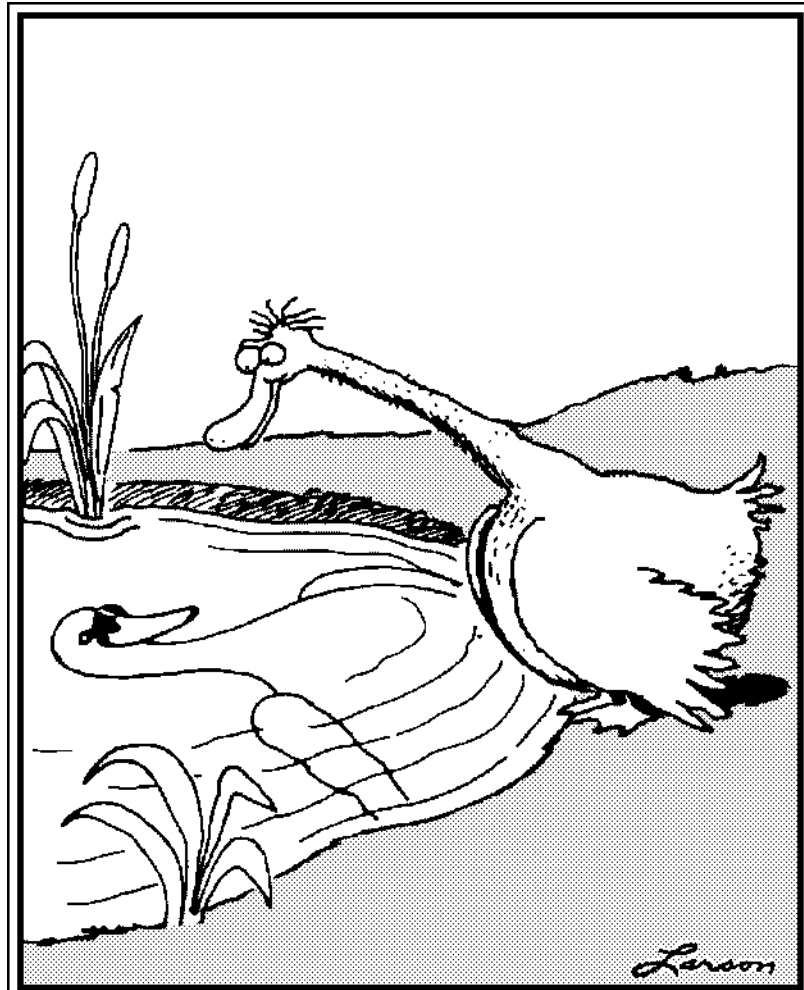
où  $P_0 \in [0, 1]$  est la valeur de  $P$  pour laquelle le paramètre prend sa valeur “finale”  $r_f$ . En pratique, il est quasiment impossible d’obtenir une valeur  $P = 1$  selon (5.11), parce qu’il subsiste toujours quelques fluctuations (bien que de faible valeur) entre les  $p_i$ . Plus l’exigence est élevée ( $P_0$  proche de 1), meilleur sera le résultat quand le réseau est stabilisé, mais plus il faudra de temps à l’algorithme pour converger. Une valeur trop élevée ne permet d’ailleurs pas de converger. D’après les simulations, une valeur qui semble convenable est  $P_0 = 0.7$ .



### 5.2.4 Résumé des paramètres

L'algorithme ainsi modifié est assez robuste face aux valeurs “initiales” et “finales” des paramètres. Nous donnons dans le tableau suivant un ensemble de valeurs numériques qui donnent une convergence rapide dans divers cas de distributions de formes très variées. De plus, la transition vers une nouvelle distribution (après un changement) est étonnamment rapide compte tenu de la bonne stabilité obtenue :

Paramètre	symbole	min	max	typique
Voisinage initial	$\lambda_i$	10	50	30
Voisinage final	$\lambda_f$	0.1	2	1
Gain initial	$\alpha_i$	0.5	1	0.8
Gain final	$\alpha_f$	0.001	0.05	0.01
Constante de temps	$\tau$	500	5000	1000
Limite de tri	$K$	1	$N$	$3\lambda + 1$
Valeur de $P$ attendue	$P_0$	0.4	1	0.7



The Far Side<sup>2</sup>

By Gary Larson

---

<sup>2</sup>The Far Side cartoon by Gary Larson is reprinted by permission of Chronicle Features, San Francisco, CA. All rights reserved.

# Chapitre 6

## Algorithme VQP

### 6.1 Introduction

On a vu que la réduction du nombre de dimensions, permettant une description de notre ensemble de données dans un espace à  $p$  dimensions avec  $p < n$ , était une chose délicate. L'analyse en composantes principales (§ 2.4) ne permet que des transformations linéaires et est donc incapable de “déplier” une variété courbe (§ 2.4.4).

On a aussi vu que les cartes auto-organisantes de Kohonen pouvaient fournir une représentation simplifiée en dimension et en nombre d'éléments d'un ensemble de données en grandes dimensions (chapitre 4). Des parallèles ont par exemple été faits ([22, 71]) avec les cartes sensorielles (rétinotopiques, somatotopiques, tonotopiques, etc.). La fonction réalisée par le modèle correspond donc, *sous certaines conditions*, à ce qu'on s'est fixé comme objectif (chapitre 1) pour l'analyse de données : la recherche de structure d'une distribution pour retrouver sa variété (§ 2.5.1). On a cependant montré (§ 4.6) les limites de ces cartes. Leur défaut majeur est que leur forme est fixée *a priori*, ce qui conduit à un grand nombre d'unités mortes (ne pointant pas dans la distribution d'entrée). Le plus souvent il s'agit d'une grille rectangulaire ou hexagonale, et dans nombre d'applications on choisit même une grille simplement carrée, faute d'informations sur la forme du sous-espace qui sous-tend la distribution. Ces cartes auto-organisantes, utilisées à bon escient, permettent donc une “projection non-linéaire” (§ 4.4), mais malheureusement leur forme doit être adaptée à celle de la variété, laquelle est justement inconnue.

C'est pour cette raison que plusieurs algorithmes “auto-organisés” comme le “*Neural Gas*” (chapitre 5) ou le réseau évolutif de Fritzke [44, 46, 45], ou des variantes sans structure de grille du réseau de Kohonen lui-même [62], ont vu le jour. Malheureusement, en abandonnant une structure régulière dans les

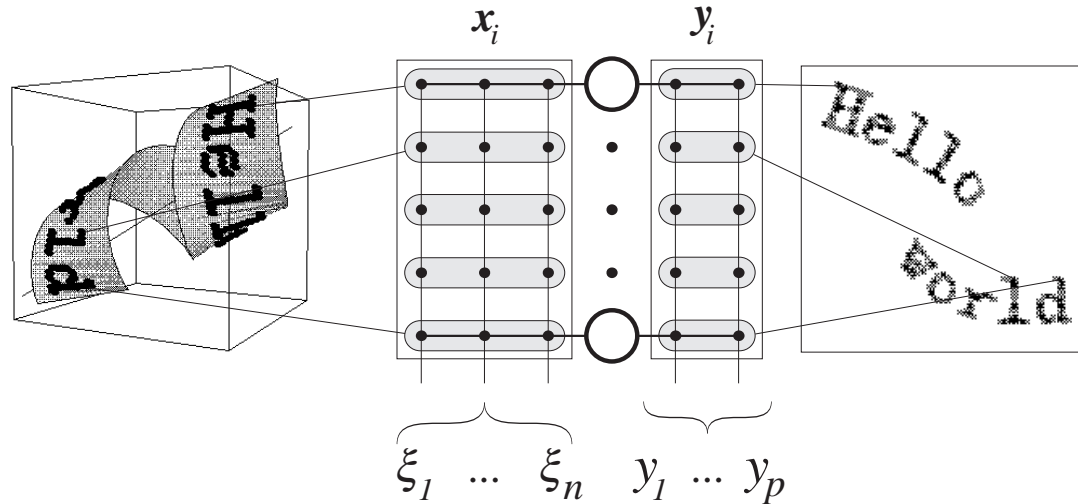


Figure 6.1: La structure du réseau VQP : un ensemble non structuré de neurones dotés de deux vecteurs-poids chacun. Cet ensemble met en relation la distribution d'entrée et sa projection non-linéaire (morphisme). Ici, la distribution (en 3 dimensions) est un texte sur un ruban enroulé. L'espace de sortie, qu'on a contraint d'avoir 2 dimensions seulement, contient la projection trouvée et nous révèle le texte. La fonction du réseau est ainsi de *déplier* la distribution pour rendre sa structure visible en faible dimension.

liens entre neurones, on perd le moyen de représenter simplement la variété des données. On transforme simplement le problème de la recherche de structure dans les données en un problème de recherche de structure dans un graphe, ce qui est tout aussi compliqué !

Nous avons proposé un nouveau modèle, le réseau VQP (*“Vector Quantization and Projection”*, [32, 31, 33]) qui s'affranchit de cette limitation. La fonctionnalité ressemble à celle du réseau de Kohonen, c'est-à-dire une quantification de l'espace d'entrée assortie d'une représentation respectant — au moins localement — la topologie de cet espace. Le principe est toutefois complètement différent :

Au lieu de faire la quantification *sous contrainte d'un voisinage préétabli entre les neurones*, les neurones “cherchent” leur position dans l'espace de sortie pour respecter — au moins localement — la topologie de l'entrée.

Les fonctions de quantification et de représentation sont séparées et déléguées à deux couches de connexions différentes. La structure du réseau comprenant ces deux couches de connexions est illustrée à la figure 6.1 (sur laquelle on a également porté l'espace d'entrée et l'espace de sortie, de dimensions différentes, avec une distribution particulière).

Ainsi les vecteurs d'entrée des neurones cherchent des prototypes de la distribution par quantification vectorielle (avec une des multiples méthodes existantes,

par exemple une de celles vues au chapitre 3). Les mêmes neurones ont chacun un vecteur de sortie dans un espace continu dont seule la dimension a été fixée au départ. Ces vecteurs de sortie, initialement placés aléatoirement, cherchent à se positionner les uns par rapport aux autres de façon à reproduire le mieux possible la topologie des vecteurs d'entrée. Le fait que ces positions en sortie soient libres (et non contraintes sur une grille) permet de représenter, avec une réduction de dimension par projection non-linéaire, une palette de distributions beaucoup plus large qu'avec les cartes de Kohonen, et tout en évitant les "unités mortes".

Cet algorithme fournit une représentation utile de structures de données redondantes et non-linéaires là où l'Analyse en Composante Principale (ACP) ou des techniques similaires sont incapables de trouver un sous-ensemble adéquat de paramètres donnant une représentation révélatrice. Par exemple, la figure 6.1 illustre cette capacité à représenter des structures fortement non-linéaires.

La structure du réseau est identique à celle des réseaux de "Radial Basis Functions" (RBF), mais la fonction est radicalement différente. Au lieu de mettre en relation de façon supervisée deux espaces prédéfinis, le deuxième espace est la projection non-linéaire du premier et est trouvé de façon auto-organisée.

## 6.2 Algorithme

Les  $N$  neurones possèdent chacun deux vecteurs-poids. Les vecteurs d'entrée  $\{\mathbf{x}_i ; i = 1, \dots, N\}$  sont en dimension  $n$ , tandis que les vecteurs de sortie  $\{\mathbf{y}_i\}$  sont en dimension  $p$  ( $p \leq n$ ).

Les vecteurs de la première couche doivent pointer dans la distribution. Cette première couche est adaptée à l'aide d'une méthode de quantification vectorielle (voir chapitre 3). Comme avec VQP on se soucie surtout de retrouver la *variété* des données, et non leur densité sur cette variété, on peut doter cette QV de notre méthode de régularisation (§ 3.7).

La couche de sortie doit faire une projection non-linéaire des vecteurs d'entrée. Pour ce faire, on se base sur les distances entre les  $\mathbf{x}_i$  :  $X_{ij} = \delta(\mathbf{x}_i, \mathbf{x}_j)$ , avec  $\delta$  la distance euclidienne, par exemple. On a les distances correspondantes en sortie :  $Y_{ij} = \delta(\mathbf{y}_i, \mathbf{y}_j)$ . Le but est de rendre équivalentes les  $Y_{ij}$  aux  $X_{ij}$ , pour tous les couples  $(i, j)$ . Comme ceci n'est pas possible si l'on doit "déplier" une structure en faisant la réduction de dimension de  $n$  vers  $p$ , on impose de tenir compte davantage des petites distances que des grandes *dans l'espace de sortie*. On obtient finalement une fonction d'erreur (ou d'énergie) à minimiser :

$$E = \frac{1}{2} \sum_i \sum_{j \neq i} (X_{ij} - Y_{ij})^2 F(Y_{ij}) \quad (6.1)$$

avec  $F(Y_{ij})$  positive, monotone et décroissante, de façon à favoriser le respect *local* de topologie (comme dans les cartes de Kohonen). La minimisation de cette fonction conduit à trouver la représentation révélatrice souhaitée.

La dérivée de  $E$  par rapport à  $y_{ik}$  (composante  $k$  du vecteur  $\mathbf{y}_i$ ) est :

$$\begin{aligned} \frac{\partial E}{\partial y_{ik}} &= \frac{\partial E}{\partial Y_{ij}} \frac{\partial Y_{ij}}{\partial y_{ik}} \\ &= \sum_{j \neq i} \frac{X_{ij} - Y_{ij}}{Y_{ij}} [2F(Y_{ij}) - (X_{ij} - Y_{ij})F'(Y_{ij})] (y_{jk} - y_{ik}) \end{aligned} \quad (6.2)$$

ou encore, de façon vectorielle :

$$\nabla_i E = \sum_{j \neq i} \frac{X_{ij} - Y_{ij}}{Y_{ij}} [2F(Y_{ij}) - (X_{ij} - Y_{ij})F'(Y_{ij})] (\mathbf{y}_j - \mathbf{y}_i) \quad (6.3)$$

où  $\nabla_i E$  dénote le gradient de  $E$  par rapport à  $\mathbf{y}_i$ .

Une minimisation de  $E$  par descente de gradient donne la règle d'adaptation :

$$\Delta \mathbf{y}_i = \alpha \sum_{j \neq i} \frac{X_{ij} - Y_{ij}}{Y_{ij}} [2F(Y_{ij}) - (X_{ij} - Y_{ij})F'(Y_{ij})] (\mathbf{y}_i - \mathbf{y}_j) \quad (6.4)$$

avec  $\alpha(t)$  le facteur d'adaptation. On remarque que cette modification est une somme de contributions de chaque  $j \neq i$ . Chaque contribution est une attraction ou une répulsion de l'unité  $i$  vers l'unité  $j$  qui peut s'exprimer comme un coefficient  $\beta_{ij}$  qui multiplie le vecteur unitaire  $(\mathbf{y}_i - \mathbf{y}_j)/Y_{ij}$ . Ce coefficient est :

$$\beta_{ij} = (X_{ij} - Y_{ij}) [2F(Y_{ij}) - (X_{ij} - Y_{ij})F'(Y_{ij})] \quad (6.5)$$

Sous réserve que  $2F(Y_{ij}) > (X_{ij} - Y_{ij})F'(Y_{ij})$  (on verra plus loin ce qu'implique cette condition), le coefficient  $\beta_{ij}$  est négatif quand  $Y_{ij}$  est trop grand par rapport à  $X_{ij}$ , et positif dans le cas contraire. Ainsi, quand  $i$  est trop loin de  $j$ , il est rapproché de  $j$ , alors qu'il est éloigné dans le cas contraire, ce qui est conforme à ce qu'on attendait intuitivement. Comme tous les termes de  $\beta_{ij}$  sont symétriques,  $\beta_{ij}$  est lui-même symétrique.

La règle (6.4) souffre de plusieurs défauts :

- Elle est relativement lourde à calculer. Pour chaque  $i$ , il faut faire une somme sur tous les  $j$ , d'où une complexité d'une itération en  $O(N^2)$  si l'on adapte toutes les unités à chaque fois.
- La somme de toutes les contributions radiales autour d'un  $i$  particulier produit un effet de "moyenne", qui ralentit l'apprentissage (les unités bougent peu).

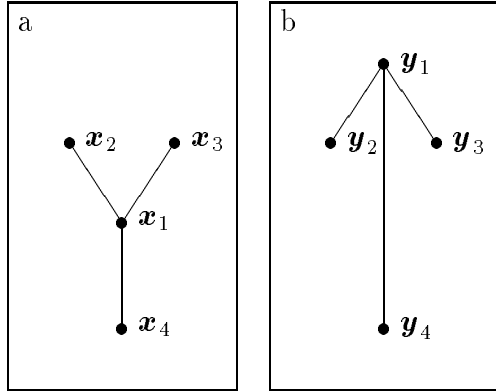


Figure 6.2: Problème du “verrou”. Dans l’espace d’entrée (a), le point  $i = 1$  se trouve au milieu des autres. Dans l’espace de sortie, il est placé malencontreusement à l’extérieur du triangle 2 – 3 – 4. Avec un mécanisme de pure descente de gradient, les points 2 et 3 l’empêchent de rejoindre sa bonne place, parce que les distances  $Y_{12}$  et  $Y_{13}$ , qui sont correctes (elles sont égales à  $X_{12}$  et  $X_{13}$ ), devraient diminuer, ce qui augmenterait momentanément l’énergie globale. La seule distance qui ne correspond pas à son équivalent en entrée est  $Y_{14}$ , qui est trop grande. Un équilibre se forme, et l’on se trouve dans un minimum local de l’énergie. Par contre, la règle d’adaptation que nous proposons, qui n’est égale à l’opposé du gradient qu’*en moyenne*, n’impose pas une descente strictement monotone de l’énergie et permet de résorber ce problème.

- Le processus peut tomber dans un minimum local de  $E$ , ainsi qu’un exemple simple permet de le constater (figure 6.2), où la somme s’annule alors que ses termes isolés permettraient de sortir du minimum local.

Nous avons remarqué qu’une procédure empirique simple, qui donne toutefois un déplacement *moyen* identique à (6.4), permet d’éviter ces défauts. Au lieu de modifier  $\mathbf{y}_i$  en fonction de la somme des contributions de tous les autres  $\mathbf{y}_j$ , on choisit un point fixe  $\mathbf{y}_i$ , et l’on déplace radialement tous les autres points  $\mathbf{y}_{j \neq i}$  par rapport à lui, sans tenir compte des interactions de ces autres points entre eux. Nous définissons donc la règle d’adaptation de la sortie du réseau VQP par :

$$\Delta \mathbf{y}_j = \alpha \frac{X_{ij} - Y_{ij}}{Y_{ij}} [2F(Y_{ij}) - (X_{ij} - Y_{ij})F'(Y_{ij})] (\mathbf{y}_j - \mathbf{y}_i) \quad \forall j \neq i \quad (6.6)$$

Cette règle est plus légère en calculs ; bien que tous les points sauf  $\mathbf{y}_i$  soient modifiés, la complexité n’est que de  $O(N)$ . On n’a pas besoin de calculer toutes les distances  $X_{ij}$  et  $Y_{ij}$ , mais seulement celles qui vont de l’unité  $i$  aux autres.

Il n’y a pas d’effet de moyenne. L’agitation des points est plus forte et permet facilement de sortir du cas du “verrou” illustré à la figure 6.2. En effet, la minimisation de la fonction d’énergie n’est pas monotone, ce qui offre une chance de sortir des minima locaux. Si l’on écrit  $E$  sous la forme :

$$E = \frac{1}{2} \sum_i \sum_{j \neq i} E_{ij} \quad (6.7)$$

$$E_{ij} = (X_{ij} - Y_{ij})^2 F(Y_{ij})$$

la règle d'adaptation que nous utilisons s'écrit alors :

$$\Delta \mathbf{y}_j \mid i = -\alpha \nabla_j E_{ij} \quad (6.8)$$

La variation d'énergie associée à une itération selon cette règle est :

$$\begin{aligned} \Delta E \mid i &= \sum_{j \neq i} (\nabla_j E)^T (\Delta \mathbf{y}_j) \\ &= \sum_{j \neq i} \left[ \left( \sum_{p \neq j} \nabla_j E_{pj} \right)^T (-\alpha \nabla_j E_{ij}) \right] \\ &= -\alpha \sum_{j \neq i} \left[ \|\nabla_j E_{ij}\|^2 + \sum_{p \neq i, j} (\nabla_j E_{pj})^T (\nabla_j E_{ij}) \right] \end{aligned} \quad (6.9)$$

Sous la somme, le premier terme est toujours positif, mais le second peut prendre n'importe quel signe, et même une valeur relativement importante puisqu'il est lui-même une somme. Par exemple, si le déplacement particulier de  $\mathbf{y}_j$ , dû à  $\nabla_j E_{ij}$ , va dans le sens contraire de ce qu'aurait donné tout autre point fixe  $p \neq i$ , donc dans le sens contraire à tous les  $\nabla_j E_{pj}$ , alors il y a de fortes chances pour que globalement  $\sum_{p \neq i, j} (\nabla_j E_{pj})^T (\nabla_j E_{ij})$  soit inférieur à  $-\|\nabla_j E_{ij}\|^2$ , et donc que l'énergie remonte. C'est le cas dans l'exemple du verrou lorsque l'on tire  $i = 4$  et  $j = 1$ . En effet, à ce moment-là, le rapprochement de  $\mathbf{y}_1$  vers  $\mathbf{y}_4$  va dans le sens d'une augmentation des termes de l'énergie  $E_{12}$  et  $E_{13}$ . Globalement, l'énergie remonte temporairement, mais pour atteindre par la suite un niveau plus bas quand le problème du verrou a été résorbé.

Si temporairement l'énergie peut augmenter, son espérance est par contre une fonction décroissante. En effet, l'espérance du déplacement  $\Delta \mathbf{y}_j$  s'exprime :

$$\begin{aligned} E(\Delta \mathbf{y}_j) &= \frac{1}{N} \sum_i (\Delta \mathbf{y}_j \mid i) \\ &= -\frac{\alpha}{N} \sum_i \nabla_j E_{ij} \\ &= -\frac{\alpha}{N} \nabla_j \sum_i E_{ij} \\ &= -\frac{\alpha}{N} \nabla_j E \end{aligned} \quad (6.10)$$

Donc, l'espérance de  $\Delta \mathbf{y}_j$  correspond (au facteur  $\alpha/N$  près) à l'opposé du gradient de l'énergie, ce qui signifie qu'en moyenne l'énergie décroît.

Cette règle empirique simple se révèle très efficace dans la pratique. Par exemple, pour un millier de points, il faut en général une cinquantaine d'itérations pour atteindre un état parfaitement ordonné (on verra plus loin comment



représenter la qualité de cet état). Une comparaison sera faite avec une méthode de gradient au § 8.4. On verra que, si le nombre d'itérations nécessaires est du même ordre, le *temps nécessaire*, lui, est beaucoup plus grand (environ  $N$  fois plus grand pour un réseau de  $N$  neurones).

Le choix de  $i$  à chaque itération peut être fait de différentes manières. Dans nos simulations, nous avons choisi  $i = i^*$ , c'est-à-dire le numéro de l'unité gagnante dans la couche de quantification vectorielle. Enfin, on peut choisir une valeur fixe pour  $\alpha$ , par exemple 0.3 ou 0.4, ou, comme dans nos simulations, faire décroître  $\alpha$  au cours du temps, de la même manière que le gain dans la couche de quantification vectorielle.

### 6.3 Projection sans réduction de dimension

Étudions le cas où  $p = n$ , c'est-à-dire qu'il n'y a pas de réduction de dimension. Dans ce cas, n'importe quelle rotation et translation des  $\mathbf{x}_i$  fournit une solution pour les  $\mathbf{y}_i$  qui minimise l'énergie (6.1) (et la rend même nulle). En effet, la distance entre deux points est invariante pour une translation ou une rotation quelconques. Dans le cas où  $\mathbf{y}_i$  est une telle transformation de  $\mathbf{x}_i$ , les distances en sortie sont égales aux distances correspondantes en entrée ( $Y_{ij} = X_{ij}$ ), et donc  $E = 0$ . Comme  $E$  est une erreur quadratique et ne peut donc être négative, il s'agit d'un minimum.

Il existe donc une infinité de solutions, correspondant à toutes les rotations, translations et opérations miroir possibles dans l'espace. Par ailleurs, comme il n'est pas nécessaire de déplier la distribution, on peut se passer de  $F(Y_{ij})$  dans (6.1).

On sait qu'il existe des minima locaux de l'énergie (par exemple le problème du verrou, figure 6.2). Cependant on a vu que notre algorithme pouvait le cas échéant sortir de ces minima.

### 6.4 Réduction de dimension (“dépliage”)

Lors de la réduction de dimension, si la variété de la distribution n'est pas une forme linéaire (comme dans le cas du fer à cheval, par exemple, voir page 39 ou 80), il est indispensable de *déplier* cette variété. Pour ce faire, nous avons introduit le facteur  $F(Y_{ij})$  dans (6.1). Ce facteur est une fonction de pondération qui décroît avec la distance *dans l'espace de sortie*, comme c'est le cas dans l'algorithme de Kohonen.

Toutefois, dans l'algorithme de Kohonen les points  $\mathbf{y}_i$  sont fixes. Au contraire, dans VQP nous minimisons l'énergie en faisant bouger les  $\mathbf{y}_i$ . D'une certaine

manière, comme le facteur de pondération décroît avec la distance dans  $\mathbf{y}$ , on a une méthode *réursive*.

En pratique, on constate une grande facilité dans le dépliage de structures fortement non-linéaires, voire même bouclées (qu'il faut donc couper quelque part). Cette facilité est similaire à celle des cartes auto-organisantes dans les mêmes conditions. Les essais que nous avons faits en tentant plutôt de mettre une pondération en fonction des distances d'entrée (donc  $F(X_{ij})$ ), comme dans la "*Non Linear Mapping*" que l'on présentera plus loin (§ 8.3), montrent qu'il est alors impossible de couper des variétés bouclées (cercles, sphères, tores, ...).

On peut utiliser plusieurs formes pour  $F(Y_{ij})$ , par exemple :

$$F(Y_{ij}) = \exp\left(-\frac{Y_{ij}}{\lambda_y(t)}\right) \quad (6.11)$$

avec  $\lambda_y(t)$  décroissant au cours du temps, comme  $\alpha(t)$ , en  $1/t$ .

Il faut toutefois faire attention à un point. Pour que la règle soit conforme à ce qu'on attend d'elle (un rapprochement des points trop éloignés, et *vice-versa*), nous avons dit (après l'équation (6.5), § 6.2) qu'il fallait assurer la condition :

$$2F(Y_{ij}) > (X_{ij} - Y_{ij})F'(Y_{ij}) \quad (6.12)$$

soit, comme  $F$  est définie positive, mais décroissante ( $F' < 0$ ) :

$$\left|\frac{F(Y_{ij})}{F'Y_{ij}}\right| > \frac{1}{2}(Y_{ij} - X_{ij}) \quad (6.13)$$

Dans le cas particulier, il vient :

$$\lambda_y(t) > \frac{1}{2}(Y_{ij} - X_{ij}) \quad (6.14)$$

condition qui est difficile à assurer dans la pratique. En effet, lors de la simulation (avec  $\lambda_y(t)$  qui décroît au cours du temps), il finit par arriver que des distances  $Y_{ij}$  soient suffisamment grandes par rapport aux  $X_{ij}$  correspondantes pour que la condition ne soit pas vérifiée. On assiste alors à l'inverse de ce qu'on cherche : les points concernés, "trop éloignés" des autres, s'éloignent encore davantage<sup>1</sup>.

Pour cette raison, il vaut mieux prendre une fonction de pondération comme :

$$F(Y_{ij}) = u[\lambda_y(t) - Y_{ij}] \quad (6.15)$$

où  $u(x)$  est la fonction échelon :  $u(x) = \begin{cases} 1 & \text{pour } x \geq 0 \\ 0 & \text{pour } x < 0 \end{cases}$ .

<sup>1</sup>Bien sûr, quand ils arrivent à l'infini (ou dans ses parages), l'énergie est nulle. Ce n'est toutefois pas exactement ce qu'on recherche.

Cette fonction correspond en quelque sorte au voisinage rectangulaire dans les cartes de Kohonen. Bien que ne donnant du poids dans l'énergie qu'aux distances inférieures à  $\lambda_y(t)$ , sa dérivée est nulle. Ainsi, on peut garantir la validité de la condition (6.12), quelque soient les paramètres.

Une autre façon de procéder peut être d'abandonner purement et simplement la partie  $(X_{ij} - Y_{ij})F'(Y_{ij})$  dans la règle (6.6). Cela revient à considérer une fonction dont la dérivée est nulle partout (du moins, partout où cette dérivée est définie), c'est-à-dire une fonction en marches d'escalier  $F_{\leftarrow}(Y_{ij})$  dont le profil suit globalement la fonction désirée  $F(Y_{ij})$ .

Dans tous les cas, le paramètre  $\lambda_y(t)$  correspond à un voisinage dans l'espace de sortie, à une distance autour de  $\mathbf{y}_{i^*}$  en deçà de laquelle les autres unités sont préférablement adaptées (dans le cas du voisinage rectangulaire fait par la fonction échelon, seules ces unités sont adaptées). Nous avons constaté que, comme dans les cartes auto-organisantes, il était utile (voire nécessaire pour les variétés fortement pliées) de procéder à un *balayage* des distances de l'espace de sortie avec  $\lambda_y(t)$ , en partant d'une valeur puis en la diminuant au cours des itérations. Pratiquement, nous partons d'une valeur correspondant à 4 fois le maximum par composante de l'écart-type des données en entrée  $\lambda_y(0) = 4 \max_k(\sqrt{\text{Var}(\xi_k)})$  et nous descendons jusqu'à  $\lambda_y(t_{\max}) = 1/5 \max_k(\sqrt{\text{Var}(\xi_k)})$  par un profil exponentiel identique à celui des paramètres du "Neural Gas" (équation (5.4)).

## 6.5 Qualité de la projection

Comme pour les cartes de Kohonen, il est important de connaître la fiabilité de la projection qu'on obtient avec l'algorithme VQP. En effet, la qualité de cette projection dépend de l'adéquation de la dimension de sortie  $p$  et de la dimension intrinsèque des données (la dimension de la variété), ainsi que de la configuration de départ, du nombre d'itérations et des paramètres  $\alpha(t)$  et  $\lambda_y(t)$ .

Nous avons proposé (§ 4.5.3) une représentation de la qualité de la projection des cartes de Kohonen basée uniquement sur les distances en entrée et sur la grille. Il s'agit du diagramme de dispersion des valeurs  $Y_{ij}$  et  $X_{ij}$ . Les distances en sortie ( $dy = \{Y_{ij} \mid \forall i, j\}$ ) sont sur l'axe horizontal, et les distances en entrée ( $dx = \{X_{ij} \mid \forall i, j\}$ ) sur l'axe vertical. Pour cette raison, nous avons appelé cette représentation " $dy - dx$ ".

Cette représentation est directement utilisable pour représenter la qualité de la projection réalisée par VQP et doit être interprétée de la même façon que pour les cartes de Kohonen (voir § 4.5.3). La seule différence, peu remarquable à vrai dire, est que dans les cartes les distances  $Y_{ij}$  sont quantifiées puisqu'il s'agit de la distance entre les nœuds de la grille. On a donc pour Kohonen  $Y_{ij} \in$

$\{1, \sqrt{2}, 2, \sqrt{5}, 2\sqrt{2}, 3, \sqrt{10}, \dots\}$ , alors qu'ici toutes les distances  $Y_{ij} \in \mathbb{R}_+$  sont possibles. D'autre part, la pente souhaitée dans la caractéristique pour Kohonen dépend de la variance des  $\xi$  et de la taille de la grille, tandis qu'ici, comme VQP est censé copier en sortie les distances d'entrée, on veut une pente de 1.

L'algorithme de Kohonen peut être vu, d'une certaine manière, comme essayant de faire tendre la caractéristique  $dy - dx$  vers une droite en faisant bouger des points *verticalement* dans le diagramme. A l'inverse, VQP essaie d'atteindre ce même résultat mais en faisant bouger *horizontalement* les points de la caractéristique.

Dans les exemples qui suivent (§ 6.6), on montrera généralement, en plus des poids dans les espaces d'entrée et de sortie, cette caractéristique  $dy - dx$ .

Une autre source d'information sur la qualité de la conservation topologique se trouve dans les “liens dynamiques” que l'on peut toujours créer entre les unités en fonction de leur proximité dans l'espace d'entrée (on obtient ainsi un graphe de proximité, cf. § 3.7). On montre ces liens dans l'espace d'entrée, mais surtout dans l'espace de sortie. Des replis, ou ruptures de projection, sont alors très visibles : il y a des croisements entre des liens, ce qui ne doit pas arriver normalement.

Ces deux moyens de représentation de la qualité de projection peuvent être disponibles tout au long de l'apprentissage. Leur coût en calcul et en mémoire est relativement modeste (de même que pour la représentation dans les espaces des poids, on rafraîchit l'affichage sporadiquement, par exemple une fois par seconde). L'information qu'ils apportent en cours de route donne une idée intuitive sur le choix des différents paramètres. Avec un peu d'expérience, on peut parfaitement identifier et corriger un paramètre qui évoluerait trop ou pas assez rapidement, ou qui aurait une valeur initiale ou finale inadéquate.

## 6.6 Exemples

Tout d'abord, nous allons observer le processus quand il n'y a pas de réduction de dimension, c'est-à-dire quand  $p = n$ . Le dessin des “liens dynamiques” permet de voir la mise en ordre des points en sortie. Dans la figure 6.3, on voit une séquence d'auto-organisation de l'espace de sortie, alors que les points en entrée ( $\mathbf{x}_i$ ) sont maintenus fixes<sup>2</sup> ( $\alpha_{in} = 0$ ).

Le premier exemple avec dépliage est la distribution en “fer à cheval”. On a vu que l'ACP ne donnait strictement aucune information dans ce cas (les 3 valeurs propres sont identiques, et les 3 axes trouvés sont aléatoires). Les cartes de Kohonen ne donnent une représentation correcte que si l'on choisit *a priori* la

---

<sup>2</sup>Dans cet exemple, on a d'abord initialisé les poids  $\mathbf{x}_i$  sur une grille, puis lancé les itérations après avoir initialisé les vecteurs de sortie ( $\mathbf{y}_i$ ) à des valeurs aléatoires.

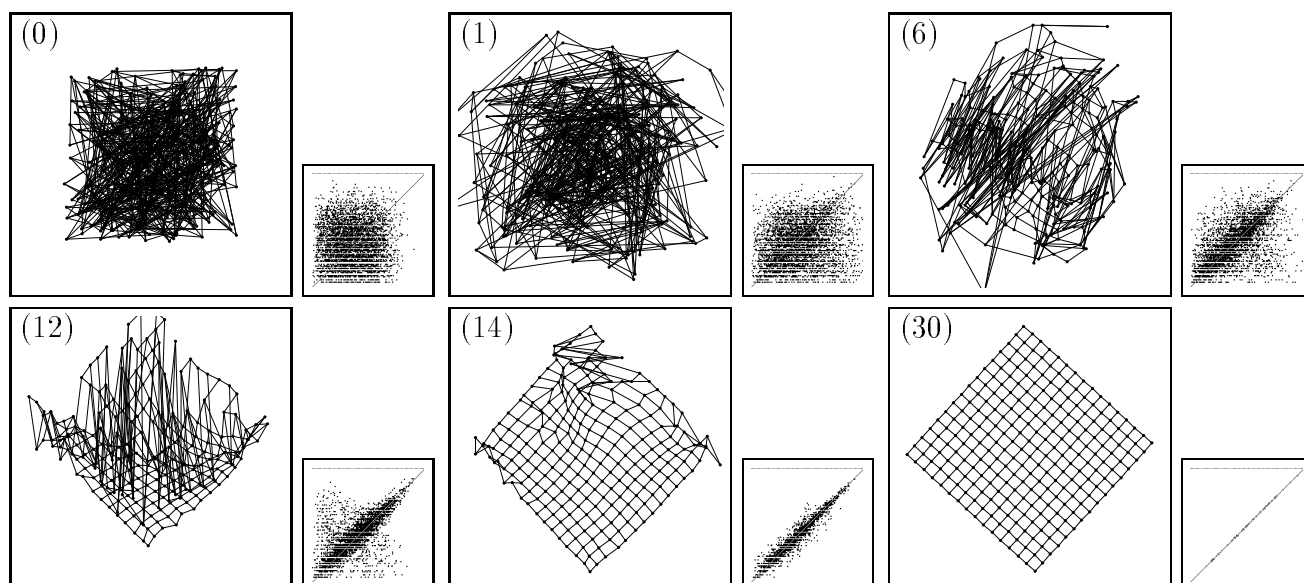


Figure 6.3: Séquence d'organisation de l'espace de sortie, l'espace d'entrée étant fixe (256 points positionnés sur une grille carrée). Pour chaque état, on montre la représentation  $dy - dx$  correspondante. Le nombre d'itérations est indiqué entre parenthèses. Après 30 itérations seulement, le réseau est déjà parfaitement organisé.

bonne forme (hauteur de la grille / largeur  $\cong 1/3$ ).

Avec VQP, par contre, il suffit d'indiquer que l'on désire une projection vers deux dimensions. La forme de la variété est automatiquement extraite et est correctement projetée par le réseau, ce qui révèle sa forme rectangulaire. Le résultat de l'apprentissage est montré à la figure 6.4.

L'exemple suivant montre le dépliage d'un cercle vers un espace en une seule dimension. Pour effectuer ce dépliage, il est nécessaire de couper le cercle quelque part. La sortie  $y$  (scalaire dans ce cas) correspond à l'abscisse curviligne depuis le point de coupure. Cette abscisse est proportionnelle à un angle. A une distance  $Y$  donnée dans  $\mathbf{y}$ , quel que soit l'endroit où on la mesure, correspond un segment du cercle d'ouverture  $\alpha(Y)$  constante. La distance  $X$  correspondante dans l'espace d'entrée est la corde  $X = d \sin(Y/d)$ , où  $d$  est le diamètre du cercle. La caractéristique  $dy - dx$  suit donc cette équation (figure 6.5).

Un exemple similaire est la projection non-linéaire d'une enveloppe de sphère vers une sortie en 2 dimensions. Les points présentés au réseau sont sur des latitudes ou des longitudes. On repère facilement ces latitudes et longitudes dans l'espace de sortie (figure 6.6). Les points sont, comme dans l'exemple du cercle, sur une structure fermée. Il faut donc couper cette structure quelque part pour pouvoir la mettre à plat (c'est le problème de la représentation cartographique

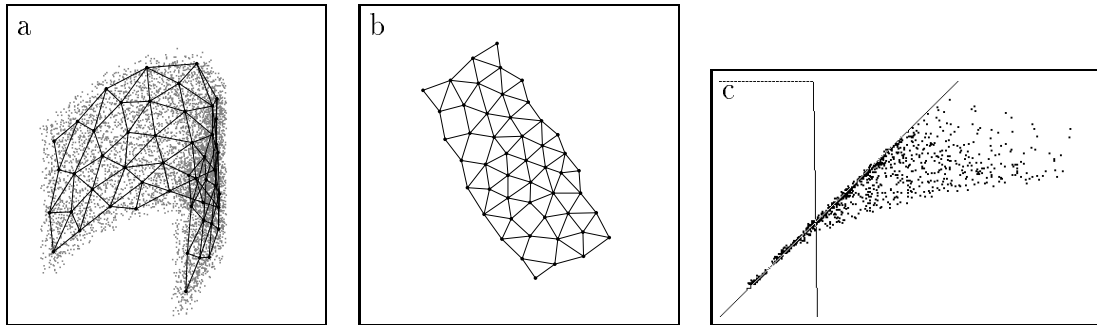


Figure 6.4: Résultat obtenu pour la distribution en “fer à cheval”. La représentation est faite dans l’espace des poids d’entrée (a) et de sortie (b), et l’on montre aussi le diagramme de dispersion  $dy - dx$  (c). Ce diagramme montre que la projection est fortement non linéaire : au-delà d’une certaine distance en sortie,  $Y_{ij}$  et  $X_{ij}$  ne sont plus corrélées, d’où une forte dispersion du nuage. Cependant, localement la topologie est bien conservée.

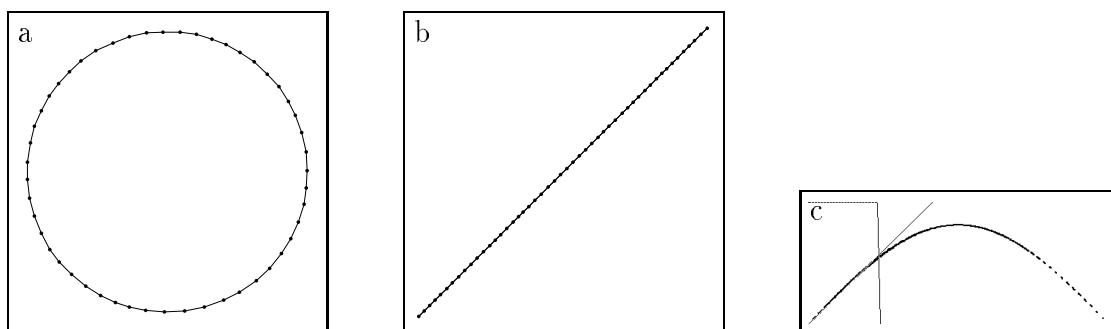


Figure 6.5: Représentation en une dimension d’un cercle. Celui-ci doit être coupé en un point pour pouvoir être déplié. La caractéristique  $dy - dx$  (c) suit l’équation de la corde :  $X = d \sin(Y/d)$  ( $Y$  : distance de sortie, montrée en abscisse;  $X$  : distance d’entrée, montrée en ordonnée).

du planisphère). Là aussi, la caractéristique  $dy - dx$  suit globalement un profil de fonction de corde. Cependant, la distance  $X$  ne dépend pas uniquement de la distance  $Y$ , mais aussi de l'endroit où on l'a mesurée dans l'espace de sortie, d'où une certaine dispersion du nuage pour les distances  $Y$  moyennes et grandes.

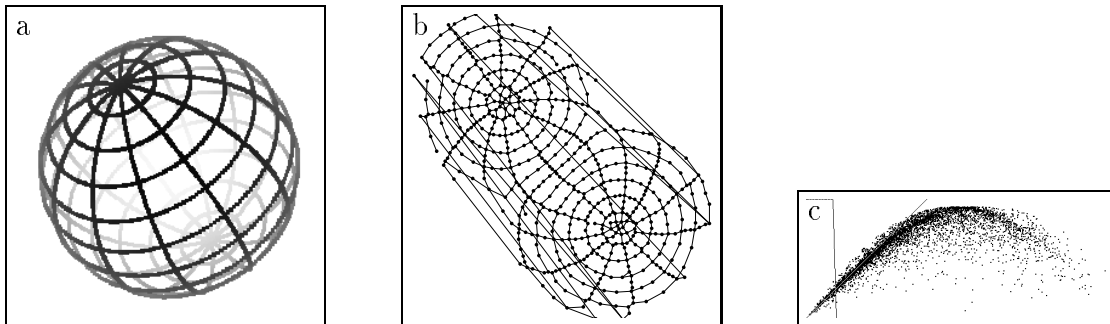


Figure 6.6: Résultat obtenu pour une distribution sur une sphère. Sur la représentation  $dy-dx$ , on voit que, localement, la topologie est respectée. Par contre, pour les distances moyennes la dispersion est assez grande. Enfin, pour les valeurs extrêmes de  $Y_{ij}$ , on trouve de petites valeurs  $X_{ij}$  : cela provient de ce que les plus grandes distances dans  $\mathbf{y}$  se trouvent en faisant le tour de la sphère, c'est-à-dire en revenant presque au point de départ dans l'espace d'entrée.

Un autre exemple intéressant de dépliage est celui des “anneaux imbriqués”<sup>3</sup>. La distribution se trouve à l'intérieur de deux tores enlacés. Lorsqu'on projette la distribution de 3 dimensions vers 2, chaque anneau est cassé à l'endroit où il passe au centre de l'autre, ce qui donne un bon respect de la topologie partout, à l'exception du voisinage de ces points de coupure (figure 6.7). Nous pouvons

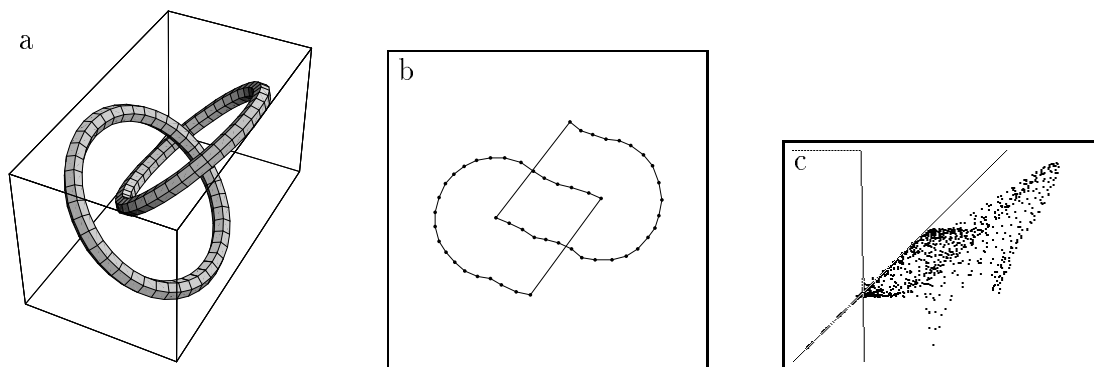


Figure 6.7: Mise à plat des “anneaux imbriqués”. L'espace d'entrée est ici en 3 dimensions.

ajouter une supervision dans l'apprentissage des “anneaux imbriqués”, en ajoutant aux vecteurs une 4<sup>e</sup> composante, qui est une étiquette (valeur) différente

<sup>3</sup>Ou du “vélo qui s'est planté”, comme dit Jeanny...

selon l’anneau (la classe) d’appartenance. Dans l’espace à 4 dimensions qui en résulte, les anneaux ne sont plus imbriqués et il n’y a pas besoin de les casser pour les mettre à plat. On obtient ainsi le résultat illustré à la figure 6.8. Les anneaux sont préservés et sont simplement placés à une certaine distance l’un de l’autre. Cette distance dépend en fait de la différence numérique des étiquettes, puisque cette différence est transformée en distance. Ici, on a une différence de  $2d$ , où  $d$  est le diamètre d’un tore, soit  $x_{i4} = +d$  pour les vecteurs d’un anneau et  $x_{i4} = -d$  pour l’autre. Après apprentissage, la reconnaissance d’un vecteur peut se faire en mettant  $x_{i4} = 0$ , et en regardant dans quel cercle il est projeté.

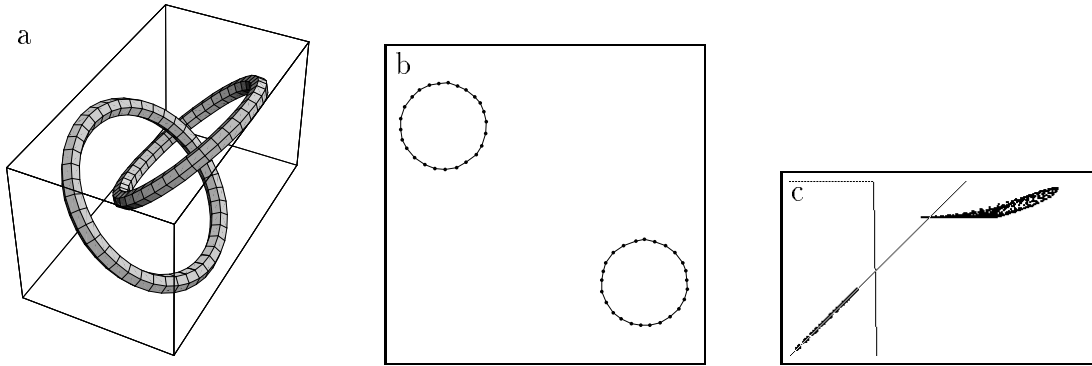


Figure 6.8: Mise à plat des “anneaux imbriqués” avec supervision (indication de classe). L’espace d’entrée est ici en 4 dimensions, la 4<sup>e</sup> étant une étiquette de classe.

Un dernier exemple est repris de [71], où Kohonen montre la possibilité de faire une taxonomie de données abstraites à l’aide d’une carte auto-organisante. Les données sont des vecteurs où 5 attributs hypothétiques sont collectés (donc l’espace d’entrée est à 5 dimensions). Les 32 vecteurs, étiquetés de ‘A’ à ‘Z’ puis de ‘0’ à ‘6’ proposés par Kohonen sont :

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	1	2	3	4	5	6
$x_1$	1	2	3	4	5	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
$x_2$	0	0	0	0	0	1	2	3	4	5	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
$x_3$	0	0	0	0	0	0	0	0	0	1	2	3	4	5	6	7	8	3	3	3	3	6	6	6	6	6	6	6	6	6	6	6
$x_4$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	3	4	1	2	3	4	2	2	2	2	2	2	2
$x_5$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	3	4	5	6

On peut dessiner à la main le “*Minimal Spanning Tree*” (MST) correspondant à ces données. A chaque “point de branchement” (par exemple ‘C’, qui est proche de ‘B’, ‘D’ et ‘F’), on fait partir une nouvelle branche dans la direction orthogonale



à celle de la branche actuelle (figure 6.9).

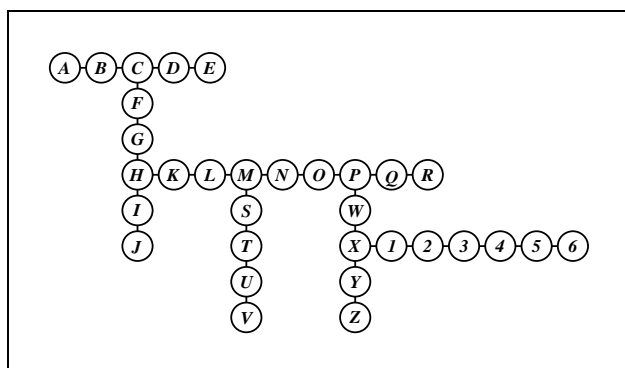


Figure 6.9: (D'après Kohonen [71]). “Minimal Spanning Tree” (tracé à la main) correspondant aux données du problème de “taxonomie”.

La figure 6.10 montre ce que l'on obtient avec une carte auto-organisante à maille hexagonale  $10 \times 7$ . Après apprentissage, on repère chaque neurone sur la grille par le label du vecteur pour lequel il est gagnant. Les neurones qui ne gagnent jamais (unités mortes) sont indiqués par un ‘ $\odot$ ’. On retrouve plus ou moins la structure du MST, mais avec d'importantes distorsions.

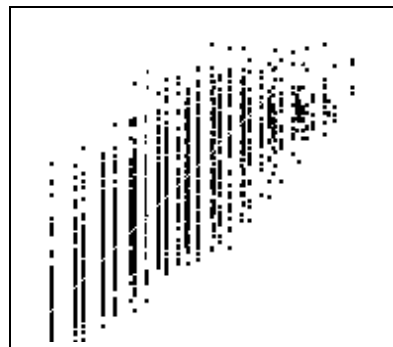
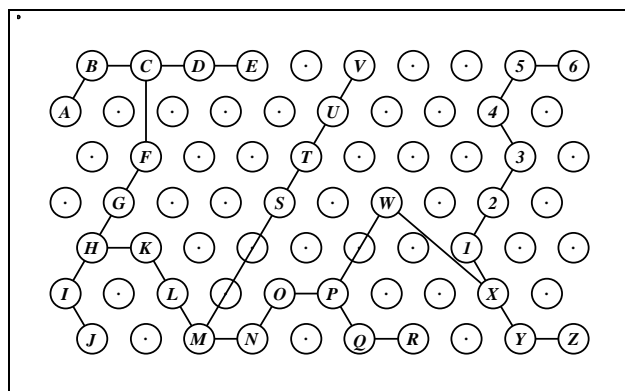


Figure 6.10: Projection des données du problème de taxonomie dans une carte auto-organisante (figure de gauche). Après l'apprentissage, on passe en revue tous les vecteurs, en marquant à chaque fois le neurone gagnant avec l'étiquette du vecteur. La figure de droite montre la représentation  $dy - dx$  correspondant à la carte.

Avec VQP, par contre, il suffit de mettre autant de neurones que de vecteurs et de fixer la sortie à deux dimensions. Après apprentissage, l'espace de sortie révèle directement la structure des données d'une façon très claire (figure 6.11). Ici, comme pour la figure 6.10, on a identifié les neurones par le label du vecteur qui les fait gagner dans la couche d'entrée.

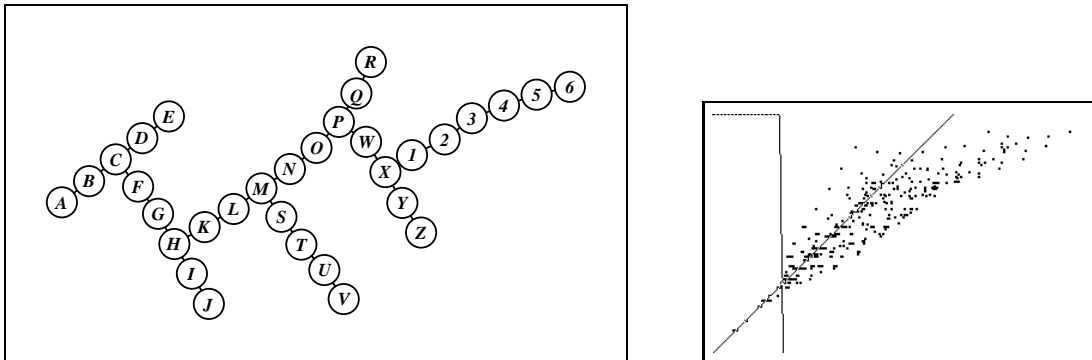


Figure 6.11: Projection des données du problème de taxonomie par VQP. Après l'apprentissage, on passe en revue tous les vecteurs, en marquant, dans l'espace de sortie ( $\mathbf{y}$ ), le neurone gagnant avec l'étiquette du vecteur. La représentation  $dy - dx$  montre que le respect de topologie est bon localement. La corrélation devient mauvaise après une distance en sortie correspondant à 5 ou 6 fois l'écart moyen entre deux vecteurs adjacents.

# Chapitre 7

## Propriétés particulières de VQP

### 7.1 Projection continue : interpolation et extrapolation

La relation  $\mathbf{x} \rightarrow \mathbf{y}$  est quantifiée par les  $N$  prototypes  $(\mathbf{x}_i \rightarrow \mathbf{y}_i)$ . Pour obtenir un continuum dans la *projection* de n'importe quel point  $\mathbf{x}$  de la distribution, on ne minimise la même fonction (6.1) que lors de l'apprentissage, mais pour le seul point  $\mathbf{y}$  qui doit être la projection de  $\mathbf{x}$ , en s'appuyant sur tous les couples  $(\mathbf{x}_i, \mathbf{y}_i)$  existants. Donc, au lieu de positionner tous les points les uns par rapport aux autres, on ne positionne qu'un point en gardant tous les autres fixes. On place donc ce point cherché  $\mathbf{y}$  par rapport aux  $\mathbf{y}_i$  en fonction des distances mesurées entre  $\mathbf{x}$  et les  $\mathbf{x}_i$ . Cela revient à considérer un nouveau neurone fictif avec les poids  $\mathbf{x}_0$  et  $\mathbf{y}_0$  (nous désignons ce neurone "virtuel" par l'indice 0). L'énergie à minimiser est celle du neurone 0, tous les autres points étant considérés comme fixes. Soit :

$$E_0 = \sum_i (X_{i0} - Y_{i0})^2 F(Y_{i0}) \quad (7.1)$$

L'algorithme de projection est un simple algorithme de descente du gradient pour placer  $\mathbf{y}_0$ . Le résultat obtenu est le point  $\mathbf{y}$  image de  $\mathbf{x}$ .

La figure 7.1 illustre la projection de tous les points de la distribution en "fer à cheval", sur la base de 50 neurones. On remarque que non seulement les points situés "entre" les neurones sont bien projetés (c'est une sorte d'interpolation), mais aussi que les points qui se trouvent à l'extérieur de tout polygone de neurones (la frange du bord de la distribution) sont également projetés de façon correcte (c'est une extrapolation).

Cette propriété d'extrapolation est plus particulièrement illustrée dans les figures 7.2 à 7.3. A la figure 7.2, un réseau formé de seulement 4 neurones a appris une distribution en carré. Toute rotation, miroir, ou translation des 4

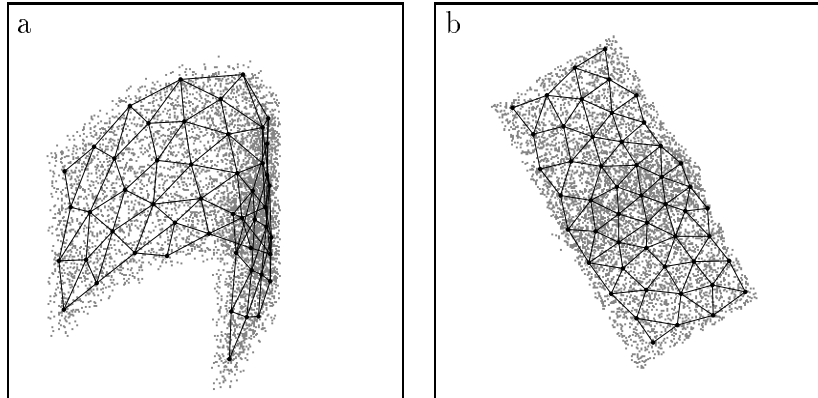


Figure 7.1: Projection de la distribution en “fer à cheval”. Le réseau VQP composé ici de 50 neurones a quantifié et déplié la distribution. Après cette phase d’apprentissage, on utilise le mécanisme de projection pour représenter en sortie l’équivalent de *tous* les points de la distribution, sur la base des prototypes  $\mathbf{x}_i$  et  $\mathbf{y}_i$  du réseau.

vecteurs-poids de l’entrée convient et donne une erreur (6.1) nulle. Bien qu’en ne s’appuyant que sur 4 échantillons ( $\mathbf{x}_i \rightarrow \mathbf{y}_i$ ), la distribution complète est projetée sans distorsion visible à l’œil nu.

Sans changer les poids après apprentissage du carré (figure 7.2), on présente maintenant une distribution en forme de cercle, qui ne coupe la zone apprise (le carré) que sur un quart (figure 7.3). On voit que cette partie est parfaitement projetée, bien que le nombre de neurones soit très faible. Les capacités d’extrapolation sont tout aussi remarquables : la partie du cercle en dehors de la zone apprise est correctement représentée. Sur la figure, on a volontairement exagéré l’erreur commise pour la rendre visible. En réalité, on ne la voit pas : avec 200 itérations de minimisation de (6.1) pour chaque point de la projection, on obtient une erreur moyenne de positionnement de 0.0008 fois le diamètre du cercle (soit une erreur relative de 0.08%).

Un autre exemple (figure 7.4) montre la projection en 2 dimensions d’une distribution particulière sur une sphère, avec les poids qui avaient été appris dans l’exemple de la figure 6.6. La distribution en question correspond aux contours des côtes sur la terre.

La propriété d’extrapolation est particulièrement nouvelle. Elle constitue une difficulté pour la plupart des réseaux dont le but est d’associer continûment une sortie avec une entrée, et qui se contentent généralement de l’interpolation entre les vecteurs-poids. Par exemple, dans les réseaux de type RBF (“*Radial Basis Functions*” [100]), ou dans certaines utilisations du réseau de Kohonen ([18, 17]) ou de la “*CounterPropagation*” [52], cette interpolation est un calcul de barycentre pondéré par des fonctions noyaux. Par le fait que les noyaux sont positifs

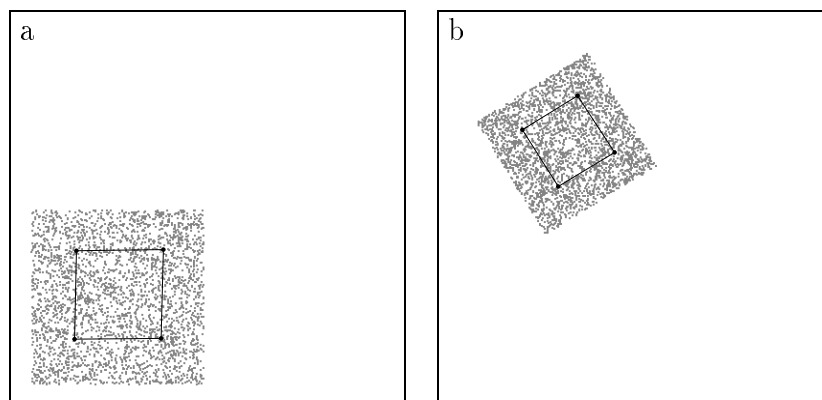


Figure 7.2: Projection d'un carré par un réseau VQP formé de 4 neurones seulement. (a) Espace d'entrée, avec les poids ayant quantifié la distribution, et, en superposition, la distribution. (b) Espace de sortie, avec les poids de sortie, et la projection du carré.

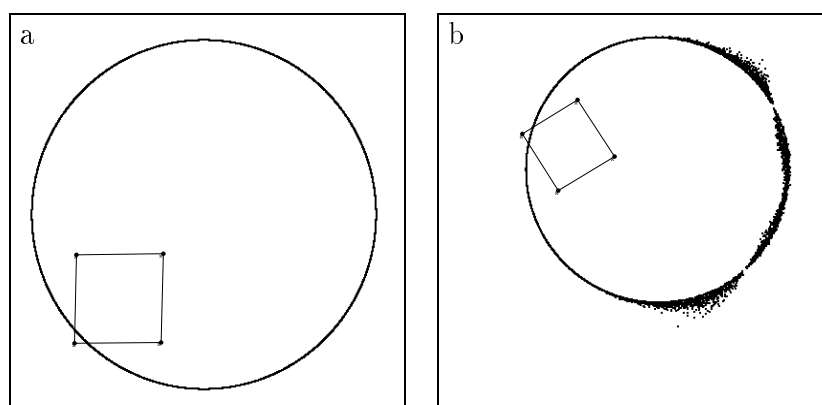


Figure 7.3: Projection d'un cercle avec un réseau VQP ayant appris un petit carré. (a) Espace d'entrée, avec les poids ayant quantifié la distribution apprise (petit carré), et la distribution de test en forme de cercle. (b) Espace de sortie, avec les poids de sortie, et la projection du cercle. Sur cette illustration, on a volontairement rendu visible l'erreur en ne laissant au réseau que 2 itérations pour projeter chaque point.

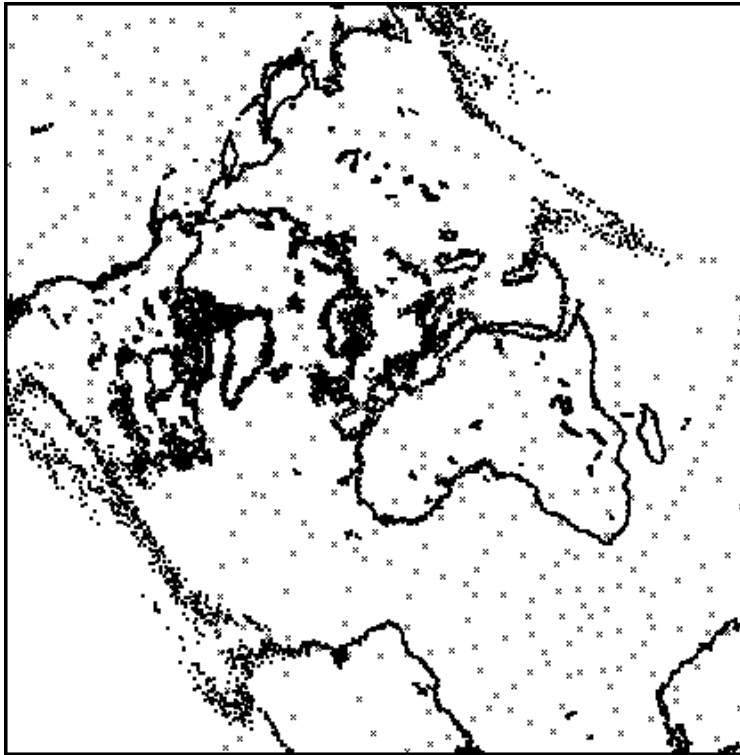


Figure 7.4: Exemple d'interpolation dans un cas où le mapping est fortement non-linéaire. Avec les poids  $x_i$  et  $y_i$  de la figure 6.6, on projette ici le contour des côtes de la terre (les données sont extraites du programme 'xearth' du domaine public, écrit par Kirk Lauritz Johnson, Jim Frost et Jamie Zawinski). En superposition, on voit les poids de l'espace de sortie, relativement espacés par rapport à la finesse des détails.

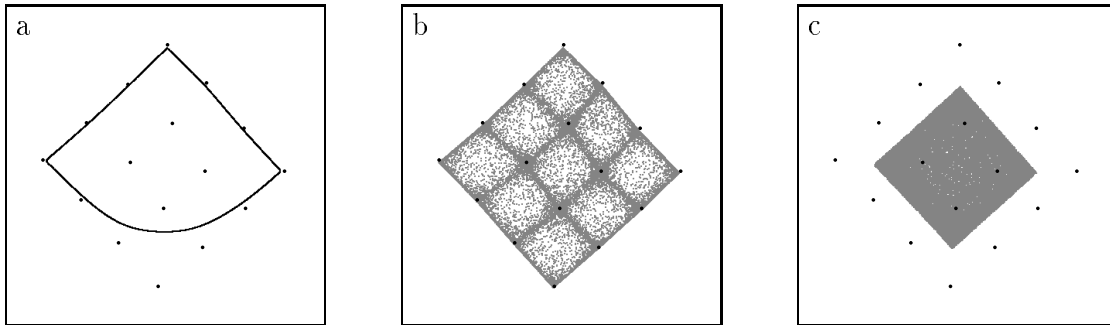


Figure 7.5: Quelques-uns des problèmes rencontrés avec une interpolation par noyaux. En (a), on demande la même tâche qu'à la figure 7.3, sauf qu'on utilise une interpolation par noyaux, et 16 neurones au lieu de 4. L'extrapolation est impossible (la partie du cercle qui sort du carré est projetée sur les bords du carré). Même avec 16 points au lieu de 4, l'interpolation est de moins bonne qualité qu'avec VQP, comme on le voit en (b) et (c), où l'on projette la distribution carrée qui avait servi pour l'apprentissage des poids. La qualité de cette interpolation dépend fortement de la taille des noyaux : en (b), le rayon des noyaux est trop faible (agglomérats), alors qu'en (c), il est trop grand (concentration vers le centre des  $\mathbf{y}_i$ ).

(généralement des gaussiennes), l'extrapolation est impossible. On voit une illustration de ce phénomène dans la figure 7.5a, qui est l'équivalent de la figure 7.3, mais avec une méthode d'interpolation par noyaux et un nombre de neurones plus élevé (16 au lieu de 4).

D'une manière générale, comme la quantification ne place pas de vecteurs tout au bord de la distribution, de telles méthodes d'interpolation par noyaux gaussiens donnent de très mauvais résultats sur les bords de la distribution. Un autre problème inhérent à ce type de méthodes est que la précision de l'interpolation dépend fortement de la façon dont on a adapté le rayon des noyaux : des noyaux trop étroits vont donner une projection où les points sont resserrés par paquets autour des vecteurs  $\mathbf{y}_i$  (figure 7.5b), tandis que des noyaux trop larges vont concentrer la distribution vers le centre de gravité des  $\mathbf{y}_i$  (figure 7.5c). Le réglage adaptatif des rayons est donc une tâche complexe qui de surcroît ralentit l'apprentissage. VQP ne souffre pas de ce problème puisqu'il se base uniquement sur les distances pour définir une projection continue.

## 7.2 Précision de la projection continue

Nous pouvons analyser la précision de cette projection continue dans le cas particulier où l'on n'a pas de réduction de dimension ( $p = n$  : la dimension en sortie est identique à celle d'entrée). En effet, dans ce cas, il n'est pas nécessaire de pondérer les termes de l'énergie par une fonction décroissant avec la distance.

Cette énergie s'écrit alors :

$$E = \frac{1}{2} \sum_i \sum_{j \neq i} (X_{ij} - Y_{ij})^2 \quad (7.2)$$

On a vu que n'importe quelle rotation, translation et miroir des  $\mathbf{x}_i$  convenait comme solution pour  $\mathbf{y}_i$  et donnait même  $E = 0$  (§ 6.3). Pour les mêmes raisons, l'algorithme de projection continue dans le cas  $p = n$  dérive de la fonction sans pondération :

$$E_0 = \sum_i (X_{i0} - Y_{i0})^2 \quad (7.3)$$

(qui est l'équivalent de (7.1) en posant  $F(Y_{i0}) = 1$ ).

Pour aborder ce calcul de précision de la projection continue, nous posons la question : "Comment  $\mathbf{y}_0$  varie-t-il lorsqu'on déplace *un seul point* (désignons-le par  $\mathbf{y}_m$ ) de  $d\mathbf{y}_m$  ?"

Nous notons par  $(.)'$  les valeurs du nouvel équilibre obtenu, c'est-à-dire :

$$\begin{aligned} \mathbf{y}'_m &= \mathbf{y}_m + d\mathbf{y}_m \\ \mathbf{y}'_0 &= \mathbf{y}_0 + d\mathbf{y}_0 \\ E'_0 &= E_0(\mathbf{y}_m + d\mathbf{y}_m, \mathbf{y}_0 + d\mathbf{y}_0) \end{aligned} \quad (7.4)$$

L'algorithme de projection est défini par la recherche du  $\mathbf{y}_0$  qui minimise  $E_0$ . On peut donc écrire, à l'équilibre avant perturbation :  $\nabla_0 E_0 = 0$ . De même, à l'équilibre après perturbation, on a :  $\nabla_0 E'_0 = 0$  (le  $d\mathbf{y}_0$  correspondant est la variation de  $\mathbf{y}_0$  qui compense au mieux la variation  $d\mathbf{y}_m$ ).

On peut développer  $E'_0$  autour de  $(\mathbf{y}_m, \mathbf{y}_0)$ . On a (en se limitant au premier ordre) :

$$E'_0 = E_0 + \begin{bmatrix} \nabla_m E_0 \\ \nabla_0 E_0 \end{bmatrix}^T \begin{bmatrix} d\mathbf{y}_m \\ d\mathbf{y}_0 \end{bmatrix} + \dots \quad (7.5)$$

Le gradient de  $E'_0$  par rapport à  $\mathbf{y}_0$  s'exprime par dérivation de (7.5), et s'écrit :

$$\nabla_0 E'_0 = \nabla_0 E_0 + \nabla_0 \begin{bmatrix} \nabla_m E_0 \\ \nabla_0 E_0 \end{bmatrix}^T \begin{bmatrix} d\mathbf{y}_m \\ d\mathbf{y}_0 \end{bmatrix} + \dots \quad (7.6)$$

où l'on voit apparaître un terme de second ordre, nécessaire pour calculer le  $d\mathbf{y}_0$  qui compense  $d\mathbf{y}_m$ . Comme on a supposé la convergence de l'algorithme de projection continue aussi bien avant qu'après la perturbation,  $\nabla_0 E'_0 = \nabla_0 E_0 = 0$ , et l'on obtient ainsi la condition qui va nous permettre d'exprimer  $d\mathbf{y}_0$  en fonction de  $d\mathbf{y}_m$  :

$$0 = \nabla_0 \begin{bmatrix} \nabla_m E_0 \\ \nabla_0 E_0 \end{bmatrix}^T \begin{bmatrix} d\mathbf{y}_m \\ d\mathbf{y}_0 \end{bmatrix} \quad (7.7)$$



soit :

$$\nabla_0(\nabla_0 E_0)^T d\mathbf{y}_0 = -\nabla_0(\nabla_m E_0)^T d\mathbf{y}_m \quad (7.8)$$

On a pour les dérivées :

$$\begin{aligned} \nabla_0 E_0 &= 2 \sum_i (X_{i0} - Y_{i0}) \mathbf{u}_{0i} \\ \nabla_m E_0 &= -2(X_{m0} - Y_{m0}) \mathbf{u}_{0m} \\ \nabla_0(\nabla_0 E_0)^T &= 2 \sum_i \left( \frac{X_{i0}}{Y_{i0}} \mathbf{u}_{0i} \mathbf{u}_{0i}^T - \frac{X_{i0} - Y_{i0}}{Y_{i0}} I \right) \\ \nabla_0(\nabla_m E_0)^T &= -2 \left( \frac{X_{m0}}{Y_{m0}} \mathbf{u}_{0m} \mathbf{u}_{0m}^T - \frac{X_{m0} - Y_{m0}}{Y_{m0}} I \right) \end{aligned} \quad (7.9)$$

avec  $\mathbf{u}_{ij}$  le vecteur directeur de  $\mathbf{y}_i$  vers  $\mathbf{y}_j$ , soit  $\mathbf{u}_{ij} = (\mathbf{y}_j - \mathbf{y}_i)/Y_{ij}$ .

Si l'on suppose que les points  $\mathbf{y}_i$  eux-mêmes sont bien placés (et donnent une énergie totale  $E = 0$ ), c'est-à-dire si l'on suppose que  $X_{ij} = Y_{ij} \forall i, j$ , alors les termes se simplifient et (7.8) devient :

$$\left( \sum_i \mathbf{u}_{0i} \mathbf{u}_{0i}^T \right) d\mathbf{y}_0 = \mathbf{u}_{0m} \mathbf{u}_{0m}^T d\mathbf{y}_m \quad (7.10)$$

On peut réécrire le terme de gauche en utilisant une matrice  $U_0$  de taille  $N \times p$  dont les colonnes sont les  $\mathbf{u}_{0i}$  :

$$\sum_i \mathbf{u}_{0i} \mathbf{u}_{0i}^T = U_0 U_0^T \quad (7.11)$$

$U_0 U_0^T$  est la matrice de corrélation du nuage des  $\mathbf{u}_{0i}$ . Comme elle est symétrique, elle est orthogonalement diagonalisable, c'est-à-dire qu'on peut l'exprimer sous la forme :

$$U_0 U_0^T = R \Lambda R^T \quad (7.12)$$

avec  $\Lambda$  la matrice diagonale des valeurs propres de  $U_0 U_0^T$  et  $R$  la matrice de rotation (orthonormée) dont les colonnes  $\mathbf{r}_i$  sont les vecteurs propres correspondants aux  $\lambda_i$  (rappel :  $R^{-1} = R^T$ ). La somme des valeurs propres est la trace de  $\Lambda$ . A cause des propriétés de la trace (qui n'est pas changée par rotation), c'est également la trace de  $U_0 U_0^T$ . On vérifie aisément que cette dernière est :

$$\begin{aligned} \text{Trace} \left( U_0 U_0^T \right) &= \sum_{k=1}^p \sum_{i=1}^N u_{ik}^2 = \sum_{i=1}^N \left( \sum_{k=1}^p u_{ik}^2 \right) \\ &= \sum_{i=1}^N (1) = N \end{aligned} \quad (7.13)$$

Donc, la somme des  $p$  valeurs propres vaut  $N$ .

En utilisant ces notations, on peut écrire dans le cas général (pour  $\mathbf{y}_0$  situé n'importe où par rapport aux  $\mathbf{y}_i$ , c'est-à-dire pour n'importe quelle distribution des  $\mathbf{u}_{0i}$ ) :

$$R\Lambda R^T d\mathbf{y}_0 = \mathbf{u}_{0m} \mathbf{u}_{0m}^T d\mathbf{y}_m \quad (7.14)$$

On peut chercher la projection de  $d\mathbf{y}_0$  sur l'axe principal  $\mathbf{r}_i$ , en prémultipliant les deux termes par  $\mathbf{r}_i^T$  :

$$\begin{aligned} \mathbf{r}_i^T R\Lambda R^T d\mathbf{y}_0 &= \mathbf{r}_i^T \mathbf{u}_{0m} \mathbf{u}_{0m}^T d\mathbf{y}_m \\ \lambda_i \mathbf{r}_i^T d\mathbf{y}_0 &= \mathbf{r}_i^T \mathbf{u}_{0m} \mathbf{u}_{0m}^T d\mathbf{y}_m \end{aligned} \quad (7.15)$$

Les matrices  $\Lambda$  et  $U_0 U_0^T$  sont inversibles si et seulement si  $\text{rang}(U_0) = p$ , c'est-à-dire si les  $\mathbf{u}_{0i}$  sont générateurs de l'espace  $\mathbb{R}^p$ . Dans le cas contraire, cela signifie que tous les  $\mathbf{y}_i$  sont contenus dans un sous-espace de plus faible dimension, et il suffit alors de reconsidérer le problème dans ce sous-espace (en se plaçant dans le repère  $R$ , en éliminant les composantes nulles et repartant avec  $p = \text{rang}(U_0)$ ).

Si les matrices sont inversibles, aucune valeur propre n'est nulle, et l'on peut écrire d'après (7.15) :

$$\mathbf{r}_i^T d\mathbf{y}_0 = \frac{1}{\lambda_i} \mathbf{r}_i^T \mathbf{u}_{0m} \mathbf{u}_{0m}^T d\mathbf{y}_m \quad (7.16)$$

La norme de  $d\mathbf{y}_0$  se trouve maintenant facilement :

$$\begin{aligned} \|d\mathbf{y}_0\| &= \sqrt{\sum_i (\mathbf{r}_i^T d\mathbf{y}_0)^2} \\ &= \sqrt{\sum_i \left( \frac{1}{\lambda_i^2} \|\mathbf{r}_i^T \mathbf{u}_{0m} \mathbf{u}_{0m}^T d\mathbf{y}_m\|^2 \right)} \\ &= \sqrt{\sum_i \frac{\cos^2 \theta(\mathbf{r}_i, \mathbf{u}_{0m}) \cos^2 \theta(\mathbf{u}_{0m}, d\mathbf{y}_m)}{\lambda_i^2}} \|d\mathbf{y}_m\| \end{aligned} \quad (7.17)$$

L'ennui, c'est que certaines valeurs propres  $\lambda_i$  peuvent être petites, même si leur somme vaut  $N$ . En fait, le pire des cas (celui qui donne une déviation  $\|d\mathbf{y}_0\|$  maximale pour une modification  $\|d\mathbf{y}_m\|$  donnée) survient quand  $d\mathbf{y}_m$  ainsi que  $\mathbf{u}_{0m}$  sont tous deux alignés avec l'axe  $\mathbf{r}_j$  correspondant à la plus petite valeur propre  $\lambda_j$ . D'autre part, cette valeur propre est minimale dans le cas suivant : tous les  $\mathbf{u}_{0,i \neq m}$  sont orthogonaux à  $\mathbf{r}_j$ , et seul le point  $\mathbf{y}_m$  contribue à la variance sur cet axe. On peut montrer facilement que dans ce cas,  $\lambda_j = \mathbf{r}_j^T \mathbf{u}_{0m}$ . Il vient alors immédiatement de (7.15) que dans ce cas :

$$\mathbf{r}_j^T d\mathbf{y}_0 = \mathbf{u}_{0m}^T d\mathbf{y}_m \quad (7.18)$$

Comme par ailleurs on a posé pour ce cas particulier que  $\mathbf{u}_{0m}$  était colinéaire avec  $\mathbf{r}_j$ , il s'ensuit que  $\mathbf{r}_{i \neq j}^T \mathbf{u}_{0m} = 0$ , puisque les  $\mathbf{r}_i$  sont orthogonaux entre eux. Par conséquent, on peut poser pour ce cas :

$$\|d\mathbf{y}_0\| = \|d\mathbf{y}_m\| \quad (7.19)$$

Il faut bien remarquer qu'il s'agit là d'un cas limite. Dans la pratique, on espère que les  $\mathbf{y}_i$  sont mieux répartis.

Un autre cas, plus fréquent, est intéressant à considérer : le point  $\mathbf{y}_0$  se trouve dans la distribution des  $\mathbf{y}_i$ , et ces derniers sont répartis de façon isotrope dans tout l'espace environnant. Dans ce cas, l'ellipsoïde d'inertie des  $\mathbf{u}_{0i}$  tend vers une sphère. Toutes les valeurs propres de  $U_0 U_0^T$  sont égales. Comme il y a  $p$  valeurs propres et que leur somme vaut  $N$ , elles valent alors  $\lambda = N/p$ , et l'on a  $\Lambda = \lambda I = N/pI$ . Il vient donc (en écrivant les normes au carré de chaque terme de (7.14)) :

$$\begin{aligned} (R\Lambda R^T d\mathbf{y}_0)^T (R\Lambda R^T d\mathbf{y}_0) &= (\mathbf{u}_{0m} \mathbf{u}_{0m}^T d\mathbf{y}_m)^T (\mathbf{u}_{0m} \mathbf{u}_{0m}^T d\mathbf{y}_m) \\ d\mathbf{y}_0^T R\Lambda^T R^T R\Lambda R^T d\mathbf{y}_0 &= d\mathbf{y}_m^T \mathbf{u}_{0m} \mathbf{u}_{0m}^T \mathbf{u}_{0m} \mathbf{u}_{0m}^T d\mathbf{y}_m \\ \lambda^2 \|d\mathbf{y}_0\|^2 &= \|\mathbf{u}_{0m}^T d\mathbf{y}_m\|^2 \\ \text{on encore :} & \\ \|d\mathbf{y}_0\| &= \frac{p}{N} \|\mathbf{u}_{0m}^T d\mathbf{y}_m\| \end{aligned} \quad (7.20)$$

c'est-à-dire que, en normes, la variation du point  $\mathbf{y}_0$  vaut  $p/N$  de la projection de la variation de  $\mathbf{y}_m$  sur l'axe entre  $\mathbf{y}_0$  et  $\mathbf{y}_m$ .

### 7.3 Projection continue dans le bruit

Lorsque la distribution possède un certain bruit autour de sa "variété moyenne" (voir figure 7.6), les points  $\boldsymbol{\xi}$  peuvent se trouver à côté de cette variété, disons à une distance  $\varepsilon$ . Soit  $X_{i0}$  la distance mesurée entre  $\boldsymbol{\xi}$  (le point bruité) et  $\mathbf{x}_i$ , et  $X'_{i0}$  la distance entre  $\boldsymbol{\xi}'$  (la meilleure approximation de  $\boldsymbol{\xi}$  sur la variété) et  $\mathbf{x}_i$ . On peut dire que  $X_{i0}$  est une version bruitée de  $X'_{i0}$ . Malheureusement, l'erreur relative entre  $X_{i0}$  et  $X'_{i0}$  est d'autant plus forte que  $X_{i0}$  est petite. En particulier, pour les points  $\mathbf{x}_i$  les plus proches de  $\boldsymbol{\xi}$  (là où la variété est localement orthogonale à  $\boldsymbol{\xi} - \boldsymbol{\xi}'$ ), on a :

$$X_{i0}^2 + \varepsilon^2 = X'_{i0}^2 \quad (7.21)$$

Comme c'est justement des petites distances que l'on tient le plus compte dans VQP, ainsi qu'on l'a déjà expliqué, cette perturbation  $\varepsilon$  est fort gênante pour le calcul de la projection continue.

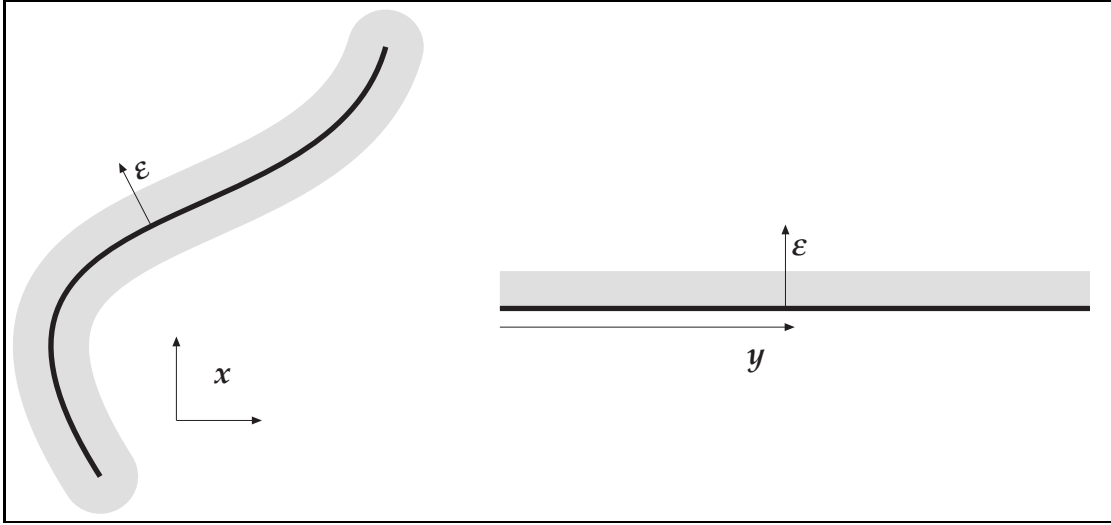


Figure 7.6: Projection dans le cas de données se trouvant à côté de la variété apprise. En donnant un degré de liberté supplémentaire à  $\mathbf{y}_0$ , on améliore considérablement la précision de la projection. De plus, la valeur trouvée pour cette variable supplémentaire correspond, en valeur absolue, à  $\varepsilon$ .

Une astuce permet d'éliminer pratiquement complètement ce problème. Comme l'erreur est orthogonale à la variété au voisinage de  $\boldsymbol{\xi}$ , l'idée consiste à donner un degré de liberté de plus à la projection, sur un axe orthogonal à l'espace de sortie. Tout se passe comme si l'on travaillait avec une dimension de sortie égale à  $p + 1$ , mais avec la  $p + 1$ ème composante nulle pour tous les  $\mathbf{y}_i$ , sauf pour le  $\mathbf{y}_0$  dont on cherche la position par minimisation de (7.1). On peut extraire la composante en question (nommons-la  $\sigma$ ), afin d'éviter de stocker toutes les composantes nulles. La règle utilisée dans l'algorithme de projection devient alors :

$$\Delta \mathbf{y}_0 = \alpha \sum_i \frac{X_{i0} - Y_{i0}}{Y_{i0}} [2F(Y_{i0}) - (X_{i0} - Y_{i0})F(Y_{i0})] (\mathbf{y}_0 - \mathbf{y}_i) \quad (7.22)$$

$$\Delta \sigma = \alpha \sum_i \frac{X_{i0} - Y_{i0}}{Y_{i0}} [2F(Y_{i0}) - (X_{i0} - Y_{i0})F(Y_{i0})] \sigma \quad (7.23)$$

avec la nouvelle définition de  $Y_{i0}$  :

$$Y_{i0} = \sqrt{\sigma^2 + \sum_{k=1}^p (y_{ik} - y_{0k})^2} \quad (7.24)$$

Il faut initialiser  $\sigma$  avec une valeur non-nulle, par exemple la distance  $X_{i^*0}$  trouvée avec le point le plus proche de  $\boldsymbol{\xi}$  : le gagnant euclidien  $i^*$  de la couche de quantification vectorielle.

Les simulations montrent qu'ainsi modifié, l'algorithme de projection continue est nettement plus précis dans le cas de données bruitées. Par ailleurs, il est intéressant de constater que  $\sigma$  donne, en valeur absolue, une estimation de  $\varepsilon$ . Cette information peut être utilisée pour tester si les points présentés après apprentissage sont éloignés de la variété apprise (par exemple, au § 9.3.2).

## 7.4 Projection inverse

Il est intéressant de pouvoir retrouver dans l'espace d'entrée l'équivalent d'un point en sortie. Avec VQP, pour obtenir cette *projection inverse*, on permute simplement les rôles de l'entrée et de la sortie et on applique le même principe que pour la projection continue. Bien qu'ici cette projection se fasse avec augmentation de dimension (de  $p$  vers  $n$ ) la méthode consistant à minimiser l'énergie (7.1) est applicable directement puisque cette énergie n'est définie qu'en fonction des distances dans les espaces d'entrée et de sortie.

Un premier exemple permet d'illustrer la capacité de *généralisation* du réseau VQP muni de ces possibilités de projection et de projection inverse. Il s'agit en l'occurrence de l'apprentissage et de la généralisation d'un arc de cercle à l'aide de 4 neurones seulement. Une distribution en forme de demi-cercle est apprise par un réseau VQP dont la sortie est en une dimension seulement. Nous avons vu (§ 6.6) que dans ce cas la sortie correspondait à l'abscisse curviligne du cercle. Ici, la relation (demi-cercle  $\rightarrow$  droite) est quantifiée par 4 points seulement. La projection de l'ensemble de la distribution donne un segment de droite, grâce aux propriétés d'interpolation et d'extrapolation vues précédemment (§ 7.1).

Plus intéressante est la projection inverse vers l'entrée du segment de droite en sortie. Cette opération, illustrée à la figure 7.7, montre que *le cercle n'est pas approximé par des segments de droites* comme on pourrait s'y attendre, mais que *son arrondi est reconstitué* entre les points  $\mathbf{x}_i$  ! Cette propriété très particulière s'explique par les possibilités d'extrapolation du mécanisme de projection que l'on a illustrées plus haut.

Un autre exemple reprend le problème des anneaux imbriqués (figure 6.7, page 117). Après apprentissage, on projette en arrière une distribution de points (en forme d'ellipse) englobant les  $\mathbf{y}_i$  trouvés par le réseau, et on analyse le résultat dans l'espace d'entrée. Les points trouvés en entrée sont sur une sorte de surface lissée sous-tendue par les deux anneaux (figure 7.8).

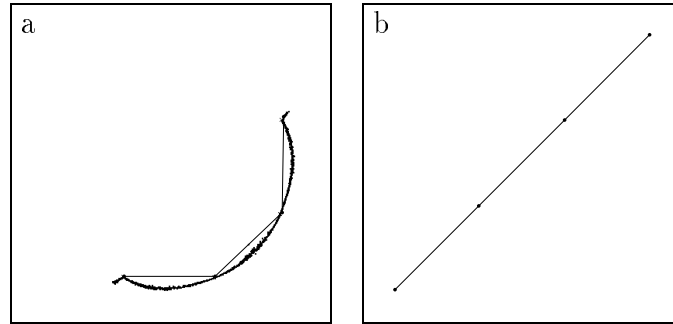


Figure 7.7: Exemple de projection inverse, depuis le segment de droite de l'espace de sortie (b) vers l'espace d'entrée (a). La reconstruction du demi-cercle montre la généralisation faite par le réseau VQP.

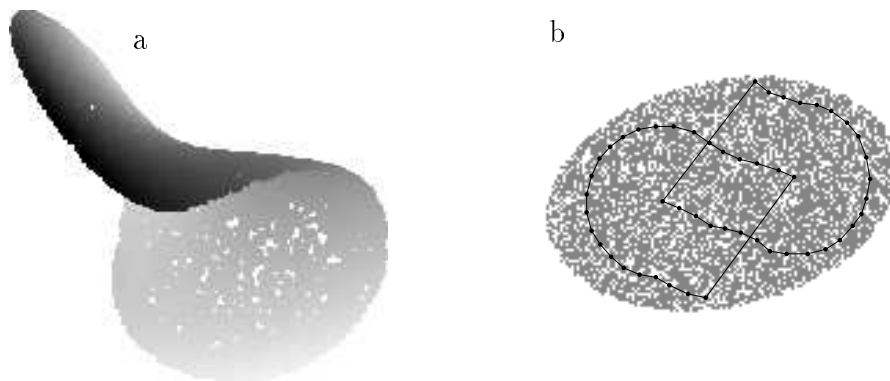


Figure 7.8: Projection inverse et généralisation. La configuration des poids en entrée et en sortie correspond au problème des “anneaux imbriqués” (voir page 117). On a fait ici la projection inverse d'une distribution englobant les poids dans l'espace de sortie  $\mathbf{y}_i$  (b). Le résultat obtenu en entrée est le nuage de points illustré en (a).

# Chapitre 8

## Représentation non linéaire

### 8.1 “Multidimensional Scaling”

Bien avant le développement de VQP ou même des cartes de Kohonen, des psychologues américains se sont intéressés à la recherche de structure de variété par dépliage ou minimisation d’une fonction d’énergie. En 1962, Shepard [111, 114] a ouvert une nouvelle voie, celle du “*Multidimensional Scaling*” (MDS). Nous allons exposer cette méthode en prenant nos notations habituelles.

Pour retrouver la structure d’une distribution par dépliage, la méthode consiste à placer des points  $\mathbf{y}_i$  dans un nouvel espace de dimension réduite  $p < n$ , par minimisation d’un critère de distorsion. Chaque point  $\mathbf{y}_i$  représente un point  $\mathbf{x}_i$  de l’espace d’entrée. L’espace de  $\mathbf{y}$ , de dimension  $p$ , peut être vu comme l’espace paramétrique d’une fonction inconnue dont la valeur est le vecteur à  $n$  dimensions :  $\mathbf{x} = \mathbf{f}(\mathbf{y})$ .

#### 8.1.1 Notion de continuité en une dimension

Pour construire le critère de distorsion à minimiser, l’idée centrale est la notion de *continuité*, ou de *progressivité* dans les données. Une bonne progressivité est obtenue quand, pour une faible variation des paramètres, la fonction varie peu.

Cette notion est plus facile à définir en une dimension que dans le cas général multidimensionnel. Shepard mentionne notamment son utilisation par Von Neumann dans le cadre de l’analyse de séquences temporelles. Les valeurs  $x_i$  sont alors des mesures faites selon un échantillonnage régulier (l’intervalle de temps entre  $i$  et  $i + 1$  est constant), et monodimensionnel. Dans ce contexte, Von Neumann utilise une mesure (inverse) de continuité basée sur le carré moyen de la différence

entre deux échantillons successifs, le tout étant normalisé par la variance :

$$\eta = \frac{\delta^2}{S^2} = \frac{E[(x_{i+1} - x_i)^2]}{\text{Var}(x_i)} \quad (8.1)$$

En laissant pour l'instant de côté le facteur de normalisation  $1/\text{Var}(x_i)$ , on a l'estimateur :

$$\delta^2 = E[(x_{i+1} - x_i)^2] = \frac{1}{N-1} \sum_{i=1}^{N-1} (x_{i+1} - x_i)^2 \quad (8.2)$$

De faibles valeurs de cette mesure  $\delta^2/S^2$  indiquent une bonne progressivité dans la série de  $x_i$ . Il faut remarquer que cela n'implique pas que les  $x_i$  doivent être sur une droite, ou même qu'ils doivent être monotones : il suffit que les changements soient "doux".

Si l'échantillonnage n'est pas fait de façon régulière, comme c'est généralement le cas dans les données (où l'on ne connaît pas *a priori* les intervalles dans l'espace paramétrique), il faut tenir compte de cette irrégularité. Cela peut être réalisé par un changement d'échelle tel que l'écart entre deux  $y_i$  soit ramené à l'unité :

$$\delta^2 = \frac{1}{N-1} \sum_{i=1}^{N-1} \frac{(x_{i+1} - x_i)^2}{(y_{i+1} - y_i)^2} \quad (8.3)$$

(ou, dans un cas continu,  $\delta^2 = \int_a^b f'(y)^2 dy$  si  $[a, b]$  est le domaine de  $y$ ).

### 8.1.2 Extension au cas multidimensionnel

La mesure (8.3) (donc la notion de continuité) souffre encore de deux problèmes :

1. Elle ne peut pas être étendue aisément à  $p > 1$  dimensions, c'est-à-dire à des paramètres  $\mathbf{y}$  vectoriels, car la notion d'adjacence n'est pas définie pour des échantillonnages irréguliers à plus d'une dimension.
2. Même en une dimension, la restriction aux couples d'échantillons adjacents est un peu arbitraire. Il peut arriver que des échantillons non adjacents soient plus proches que d'autres qui sont adjacents.

Pour ces raisons, Shepard et Carroll proposent (toujours dans [114]) de tenir compte de *tous* les couples  $(i, j)$  de points. Toutefois, si la notion d'adjacence est abandonnée, il devient alors nécessaire de pondérer les termes  $(\mathbf{x}_i - \mathbf{x}_j)^2$  par un facteur  $w_{ij}$  décroissant de façon monotone avec la séparation entre les paramètres (c'est l'équivalent fonctionnel de  $F(Y_{ij})$  dans VQP). Pour reprendre nos conventions, nous noterons par  $X_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|$  la distance entre deux points



de l'espace d'entrée, et  $Y_{ij} = \|\mathbf{y}_i - \mathbf{y}_j\|$  la distance entre deux points de l'espace paramétrique. On obtient alors :

$$\delta^2 = \sum_i \sum_{j \neq i} \frac{X_{ij}^2}{Y_{ij}^2} w_{ij} \quad (8.4)$$

Une pondération simple est :

$$w_{ij} = \frac{1}{Y_{ij}^m} \quad (8.5)$$

avec  $m > 0$ . Par exemple, avec  $m = 2$ , on obtient :

$$\delta^2 = \sum_i \sum_{j \neq i} \frac{X_{ij}^2}{Y_{ij}^4} \quad (8.6)$$

### 8.1.3 Normalisation

Dans le cas de la mesure introduite par Von Neumann (8.1), la normalisation (par le facteur  $1/\text{Var}(x_i)$ ) est faite de telle façon qu'un changement d'échelle dans les  $x_i$  soit sans effet sur la mesure. D'autre part, l'écart du paramètre (le temps, en l'occurrence) entre deux échantillons adjacents est supposé constant (échantillonnage régulier). Ici, on est dans le cas contraire : les  $\mathbf{x}_i$  sont les données du problème et ne changent donc pas, alors que les  $\mathbf{y}_i$  forment un échantillonnage irrégulier de l'espace paramétrique à retrouver par optimisation de l'indice de continuité. En l'absence d'une normalisation basée sur les  $\mathbf{y}_i$ , la minimisation de (8.6) conduira à la solution dégénérée où les  $\mathbf{y}_i$  ont des valeurs très grandes (les distances entre elles étant très grandes aussi, on peut rendre par ce moyen le critère arbitrairement petit). Il est donc nécessaire d'introduire une normalisation qui rende le critère indépendant de l'échelle en  $\mathbf{y}$ . Un facteur de normalisation adéquat, qui rend numérateur et dénominateur homogènes par rapport à  $Y_{ij}$ , est  $1/(\sum_i \sum_{j \neq i} \frac{1}{Y_{ij}^2})^2$ . On obtient donc en définitive :

$$\kappa = \frac{\sum_i \sum_{j \neq i} \frac{X_{ij}^2}{Y_{ij}^4}}{\left( \sum_i \sum_{j \neq i} \frac{1}{Y_{ij}^2} \right)^2} \quad (8.7)$$

L'usage de la distance euclidienne est arbitraire, et ce critère  $\kappa$  peut être

généralisé :

$$\kappa = \frac{\sum_i \sum_{j \neq i} \frac{(X_{ij}^2)^a}{(Y_{ij}^2)^b}}{\left( \sum_i \sum_{j \neq i} \frac{1}{(Y_{ij}^2)^c} \right)^{b/c}} \quad (8.8)$$

avec  $a - b + c = 0$ .

### 8.1.4 Mise en œuvre

Une bonne représentation de la variété d'un ensemble de points  $n$ -dimensionnels  $\mathbf{x}_i$  peut être définie comme la recherche d'un espace paramétrique à  $p$  dimensions, supporté par des points  $\mathbf{y}_i$  tels que l'indice de continuité  $\kappa$  (8.7) soit optimisé. Comme  $\kappa$  est en réalité un indice inverse (une faible valeur indique une grande continuité), cette optimisation est une minimisation.

Pratiquement, Shepard et Carroll proposent de fixer *a priori* la dimension intrinsèque estimée  $p$ , puis d'initialiser des points  $\mathbf{y}_i$  à des valeurs aléatoires. A partir de là,  $\kappa$  est minimisé par un processus de descente de gradient, c'est-à-dire qu'à chaque itération, pour la  $k$ -ième composante du point  $i$  :

$$\Delta y_{ik} = -\alpha \frac{\partial \kappa}{\partial y_{ik}} \quad (8.9)$$

Le processus itératif est arrêté lorsque  $\kappa$  ne paraît plus diminuer. Plusieurs essais sont faits avec des valeurs de départ différentes pour les  $\mathbf{y}_i$ , et l'on retient la solution qui donne, après convergence, un  $\kappa$  minimum.

Le principal problème de cette méthode est la complexité du gradient de  $\kappa$  (et, pour une implantation "neuronal", sa non-localité). A chaque étape du calcul, il faut en effet connaître toutes les  $N(N-1)$  distances  $X_{ij}$  et  $Y_{ij}$ . Cela représente une charge de calcul et une occupation mémoire qui rendent l'algorithme inutilisable pour un grand nombre de points (la limite se situant, selon le type de machine, entre  $N = 100$  et  $N = 1000$ ).

Néanmoins, l'approche est très intéressante parce qu'elle est la première (à notre connaissance) qui a permis de déplier une structure de données dans un espace de dimension inférieure, sans *a priori* sur la forme de la variété. Seule la dimension  $p$  doit être estimée. Shepard propose de faire plusieurs essais avec des valeurs de  $p$  de plus en plus petites, en retenant la plus faible ayant donné un critère de distorsion final "acceptable". La solution correspondante est alors une représentation non linéaire, en dimensions réduites, des points  $\mathbf{x}_i$  de l'espace d'entrée.

Développée au fil des ans par Shepard et ses collègues Carroll et Kruskal [75, 114], puis par Koontz et Fukunaga [74], cette voie dans l'analyse de données s'est diversifiée et a été illustrée par un grand nombre d'exemples d'applications dans de nombreux domaines (une revue très complète de Carroll et Arabie datant de 1980 [14] mentionne plus de 300 références). Un des développements les plus utilisés à présent est le “*nonmetric MDS*” [112, 113], qui permet d'analyser des matrices de dissimilitudes pouvant ne pas avoir les caractéristiques des distances. En effet, lorsque les  $X_{ij}$  ne respectent pas l'inégalité du triangle, on ne peut obtenir une représentation où les distances  $Y_{ij}$  ne correspondent aux dissimilitudes  $X_{ij}$  que d'une façon *ordinaire*.

## 8.2 Lien avec les cartes de Kohonen

La recherche de continuité ou de progressivité expliquée plus haut (§ 8.1.1) n'est pas sans évoquer ce que l'on perçoit intuitivement comme le but de l'algorithme d'auto-organisation de Kohonen. Nous avons affirmé (§ 4.2, § 4.4) que la fonction du réseau de Kohonen était double :

- quantification vectorielle d'une part,
- représentation non linéaire dans un espace de dimensions réduites (la carte, généralement à une ou deux dimensions) d'autre part.

Nous allons donner ici des éléments pour justifier cette affirmation. Rappelons que dans notre notation les vecteurs-poids sont appelés  $\mathbf{x}_i$ , la position dans la grille est notée  $\mathbf{y}_i$ , et les distances dans ces deux espaces sont  $X_{ij}$  respectivement  $Y_{ij}$ . Repartons de la règle d'adaptation des vecteurs-poids (4.2), où  $j$  représente le numéro de l'unité gagnante (dont le vecteur-poids est le plus proche de l'entrée  $\xi$ ) :

$$\Delta \mathbf{x}_i = \alpha(t)G(\mathbf{y}_i, \mathbf{y}_j, t)(\xi - \mathbf{x}_i) \quad (8.10)$$

$G(\mathbf{y}_i, \mathbf{y}_j, t)$  est généralement une fonction radiale (qui ne dépend que de la distance  $Y_{ij}$  entre  $\mathbf{y}_i$  et  $\mathbf{y}_j$ ). De plus, on peut séparer  $(\xi - \mathbf{x}_i)$  en deux parties. On obtient alors :

$$\Delta \mathbf{x}_i = \alpha(t)G(Y_{ij}, t)(\xi - \mathbf{x}_j + \mathbf{x}_j - \mathbf{x}_i) \quad (8.11)$$

On peut écrire l'espérance du déplacement des vecteurs-poids comme étant :

$$E(\Delta \mathbf{x}_i) = \alpha(t) \left[ \sum_{j=1}^N G(Y_{ij}) \int_{\xi \in S_j} (\xi - \mathbf{x}_j) P(\xi) d\xi + \sum_{j=1}^N (\mathbf{x}_j - \mathbf{x}_i) G(Y_{ij}) \right] \quad (8.12)$$

où  $S_j$  est la région de dominance de l'unité  $j$  (région de Voronoï dans le cas euclidien; voir § 3.1). Les sommes se font sur tous les  $j$ , parce que l'on considère

que tous ont la même chance de gagner (bien que ce ne soit pas tout à fait vrai, en tout cas au début de l'apprentissage).

On vérifie alors que (8.12) dérive de la fonction :

$$E = E^{vq} + E^{mds} \quad (8.13)$$

$$E^{vq} = \frac{1}{2} \sum_i \sum_j G(Y_{ij}) \int_{\boldsymbol{\xi} \in S_j} (\boldsymbol{\xi} - \mathbf{x}_j)^T (\boldsymbol{\xi} - \mathbf{x}_i) P(\boldsymbol{\xi}) d\boldsymbol{\xi} \quad (8.14)$$

$$E^{mds} = \frac{1}{2} \sum_i \sum_j X_{ij}^2 G(Y_{ij}) \quad (8.15)$$

En réalité, (8.14) n'est pas dérivable partout. En particulier, le gradient n'existe pas sur les hyperplans séparateurs des régions  $S_j$ . Il a d'ailleurs été démontré [40] qu'il était impossible d'associer une fonction de potentiel globalement décroissante à l'algorithme de Kohonen dans le cas d'une distribution continue. Par contre, une telle fonction de potentiel existe dans le cas où les vecteurs d'entrée sont tirés d'un ensemble fini d'échantillons [103]. Dans la pratique, toutefois, cette distinction paraît quelque peu académique puisque l'algorithme effectue toujours un nombre fini d'itérations et ne "voit" par conséquent qu'un échantillonnage fini de la distribution. De plus, cette distribution elle-même est généralement un ensemble fini, disponible sous forme d'un fichier de mesures.

### 8.2.1 Premier terme : $E^{vq}$

Dans la double somme de (8.14), on peut distinguer les couples  $i = j$  des autres. Dans ce cas, en effet, on a  $Y_{ii} = 0$  par définition (distance d'un point à lui-même). Comme on utilise en général une fonction  $G(Y_{ij})$  décroissante et qui donne un maximum de 1 pour  $Y_{ij} = 0$ , on obtient :

$$E_{i=j}^{vq} = \sum_i \int_{\boldsymbol{\xi} \in S_i} \|\boldsymbol{\xi} - \mathbf{x}_i\|^2 P(\boldsymbol{\xi}) d\boldsymbol{\xi} \quad (8.16)$$

ce qui est précisément la fonction (3.5) minimisée par les méthodes de *quantification vectorielle* GLA et CL (§ 3.1).

Pour les termes  $i \neq j$ , l'expression :

$$E_{i \neq j}^{vq} = \frac{1}{2} \sum_i \sum_{j \neq i} G(Y_{ij}) \int_{\boldsymbol{\xi} \in S_j} (\boldsymbol{\xi} - \mathbf{x}_j)^T (\boldsymbol{\xi} - \mathbf{x}_i) P(\boldsymbol{\xi}) d\boldsymbol{\xi} \quad (8.17)$$

est un peu plus difficile à interpréter. En première approximation, on peut dire que si la droite passant par  $\mathbf{x}_j$  et  $\mathbf{x}_i$  est un axe de symétrie de la région  $S_j$ , alors  $E_{i \neq j}^{vq}$  s'annule (il y a autant de produits scalaires positifs que négatifs). Dans le cas

contraire, le vecteur  $\mathbf{x}_i$  est rapproché d'un tel axe. Toutefois, quand on considère le déplacement de  $\mathbf{x}_i$  :

$$\Delta \mathbf{x}_i = \alpha(t)G(Y_{ij})(\boldsymbol{\xi} - \mathbf{x}_j) \quad (8.18)$$

on constate que son espérance est nulle si l'espérance de  $\boldsymbol{\xi} - \mathbf{x}_j$  est nulle. Ainsi, un vecteur  $\mathbf{x}_j$  qui se trouve au barycentre de sa région de dominance  $S_j$ , comme le premier terme  $E_{i=j}^{vq}$  tend à l'y conduire, ne perturbe pas, globalement, les autres vecteurs.

Lorsqu'on effectue la simulation de l'algorithme dont la règle d'adaptation (8.18) dérive du seul terme  $E^{vq}$  (en faisant décroître  $\alpha(t)$  et  $\lambda(t)$  comme dans l'algorithme de Kohonen), on remarque que l'état final obtenu est similaire à celui qu'on obtient avec un simple "*Competitive Learning*" (CL). Cependant, en début de simulation, lorsque  $G(Y_{ij})$  est encore large, on observe un mouvement commun de tous les vecteurs-poids, dont le nuage est déplacé en bloc au gré des entrées  $\boldsymbol{\xi}$  (il y a une polarisation générale selon  $(\boldsymbol{\xi} - \mathbf{x}_j)$ ). Au fur et à mesure que  $G(Y_{ij})$  devient étroite autour du gagnant  $j$ , les neurones prennent leur indépendance et l'on converge vers le CL.

Une différence intéressante apparaît quand tous les vecteurs  $\mathbf{x}_i$  ont été initialisés en dehors de la distribution : tout le nuage des  $\mathbf{x}_i$  est alors déplacé vers la distribution, et non le seul vecteur qui en est le plus proche. Le nombre d'unités mortes est ainsi inférieur (mais non nul) à celui produit dans les mêmes conditions par le CL.

Ainsi donc, le terme  $E^{vq}$  correspond simplement à un type particulier de quantification vectorielle.

### 8.2.2 Second terme : $E^{mds}$

Par définition,  $X_{ii}$  et  $Y_{ii}$  sont nulles (distance d'un point à lui-même). On peut donc enlever les termes ( $i = j$ ) de la double somme, et l'on obtient :

$$E^{mds} = \frac{1}{2} \sum_i \sum_{j \neq i} X_{ij}^2 G(Y_{ij}) \quad (8.19)$$

Pour comprendre la signification de cette expression, il faut refaire une démarche analogue à celle de Shepard lorsqu'il part de l'indice (inverse) de continuité utilisé par Von Neumann pour aboutir au critère de distorsion  $\kappa$  (8.7). Cette démarche a été expliquée plus haut (§ 8.1.1 à 8.1.3).

Tout d'abord, le critère de continuité de Von Neumann est (8.1) :

$$\frac{\delta^2}{S^2} = \frac{E[(x_{i+1} - x_i)^2]}{\text{Var}(x_i)} \quad (8.20)$$

Comme pour le MDS, on laisse pour l'instant de côté le facteur de normalisation  $1/\text{Var}(x_i)$ . Ensuite, on considère aussi tous les couples  $X_{ij}$  (et non simplement ceux correspondant à des unités adjacentes dans la grille). Il faut donc favoriser les courtes distances par un facteur de pondération décroissant avec la distance, par exemple la fonction  $G(Y_{ij})$ . On trouve alors :

$$\delta^2 = \sum_i \sum_{j \neq i} X_{ij}^2 G(Y_{ij}) \quad (8.21)$$

qui, à un facteur  $\frac{1}{2}$  près, est notre fonction  $E^{m ds}$ .

La division de chaque terme par  $(y_{i+1} - y_i)$ , faite dans le développement du MDS pour tenir compte de l'irrégularité de l'échantillonnage en  $y$ , n'est pas nécessaire ici puisque l'échantillonnage est régulier (c'est la grille). Enfin, comme ce sont les  $\mathbf{x}_i$  que l'on modifie pour minimiser l'énergie, on n'a pas besoin de normaliser la fonction par rapport aux  $Y_{ij}$ , comme Shepard a été obligé de le faire dans son algorithme.

En revanche, il aurait été souhaitable ici de normaliser la fonction selon  $\mathbf{x}$ , comme le fait Von Neumann avec  $1/\text{Var}(x_i)$ . L'absence d'une telle normalisation explique l'effet de "pincement" (concentration et même collage des poids entre eux) qui survient lorsque le rayon de voisinage  $\lambda(t)$  reste grand trop longtemps par rapport à  $\alpha(t)$ . Si l'on considère toutefois que le premier terme d'énergie,  $E^{vq}$ , maintient une variance des  $\mathbf{x}_i$  comparable à celle des données, alors le terme  $E^{m ds}$  peut être vu comme une mesure de distorsion dans la projection non linéaire de  $\mathbf{x}$  vers  $\mathbf{y}$ , distorsion qui doit bien sûr être minimisée.

### 8.2.3 Analyse

La fonction  $E$  que nous avons trouvée se partage en deux termes,  $E^{vq}$  qui est un critère de quantification vectorielle ressemblant à celui du GLA et du CL, et  $E^{m ds}$  qui est une contrainte analogue à celle utilisée dans le MDS, c'est-à-dire qui impose un respect de topologie entre les  $\mathbf{x}$  et les  $\mathbf{y}$ . Cependant, la partie  $E^{m ds}$  n'étant pas normalisée par rapport à un changement d'échelle dans  $\mathbf{x}$ , elle n'est pas utilisable seule (elle ferait tendre les poids vers la solution dégénérée  $\mathbf{x}_i = \mathbf{x}_j \forall i, j$ ). L'équilibre entre les deux aspects antagonistes de  $E$  ( $E^{m ds}$  qui tend à contracter l'espace des  $\mathbf{x}$  tout en remplissant son rôle de contrainte topologique, et  $E^{vq}$  qui fait une expansion par l'attraction qu'il engendre vers les données) est nécessaire pour une bonne "convergence" de l'algorithme. Même lorsque les paramètres sont bien réglés, on constate après apprentissage un "effet de bord" déjà mentionné (le bord de la distribution est mal couvert) dû à  $E^{m ds}$ . Dans le cas où l'on annule ce terme en fin d'apprentissage (en forçant le rayon de voisinage à

0), cet effet de bord se résorbe peu à peu, mais parfois au détriment de la qualité du respect topologique<sup>1</sup>.

On peut imaginer un algorithme modifié où la partie  $E^{m ds}$  serait normalisée (par exemple, par analogie avec ce qui est proposé par Shepard pour le MDS, par un facteur en  $1/\sum\sum X_{ij}^2$ ). La règle d’adaptation qui découlerait de cette fonction d’énergie modifiée serait toutefois assez complexe, et l’on perdrait un des aspects séduisants de l’algorithme de Kohonen : sa simplicité.

### 8.3 “Nonlinear Mapping”

Sammon [107] a proposé une variante, le “*Nonlinear Mapping*” (NLM). Cette méthode est aussi basée sur la minimisation d’un critère de distorsion à l’aide d’une descente de gradient. Le critère est ici une erreur quadratique explicite, plus simple à interpréter :

$$E = \frac{\sum_i \sum_{j < i} \frac{(X_{ij} - Y_{ij})^2}{X_{ij}}}{\sum_i \sum_{j < i} X_{ij}} \quad (8.22)$$

qu’on peut écrire :

$$\begin{aligned} E &= \frac{1}{c} \sum_i \sum_{j < i} (X_{ij} - Y_{ij})^2 F(X_{ij}) \\ F(X_{ij}) &= \frac{1}{X_{ij}} \\ c &= \sum_i \sum_{j < i} X_{ij} \end{aligned} \quad (8.23)$$

On remarque une équivalence presque parfaite avec la fonction (6.1) que nous minimisons dans la partie “projection” de VQP. Ici également, le critère est minimisé lorsque les distances  $Y_{ij}$  correspondent aux  $X_{ij}$ . Par contre, la fonction de pondération qui sert à favoriser le respect de proximité ne dépend ici que des distances d’entrée. On verra plus loin (§ 8.4) que cela limite considérablement les possibilités de dépliage des structures non linéaires.

La minimisation se fait par une méthode *pseudo-Newton*<sup>2</sup> :

$$\Delta y_{ik} = -\alpha \frac{\partial E}{\partial y_{ik}} \left/ \left| \frac{\partial^2 E}{\partial y_{ik}^2} \right| \right. \quad (8.24)$$

<sup>1</sup>Le “*mapping est moins smooth*”, comme dirait un certain ministre...

<sup>2</sup>Si l’on regroupe toutes les composantes de tous les points  $\mathbf{y}_i$  dans un seul vecteur d’état  $\boldsymbol{\psi}$  (à  $m = Np$  composantes) représentatif de l’espace de sortie à un instant donné, et que  $E(\boldsymbol{\psi})$  est l’erreur en fonction de cet état, la vraie règle de Newton s’exprime par :  $\Delta \boldsymbol{\psi} = -H^{-1} \nabla E(\boldsymbol{\psi})$ .

Une façon efficace (pour les calculs) d'écrire  $\Delta y_{ik}$  est :

$$\begin{aligned}
 C_{ij} &= \frac{X_{ij} - Y_{ij}}{X_{ij} Y_{ij}} \\
 \mathbf{1} &= \underbrace{[1, \dots, 1]}_p^T \\
 \mathbf{a}_i &= \sum_{j \neq i} C_{ij} (\mathbf{y}_j - \mathbf{y}_i) \\
 \mathbf{b}_i &= \sum_{j \neq i} \left[ C_{ij} \mathbf{1} - \frac{1}{Y_{ij}^3} (\mathbf{y}_j - \mathbf{y}_i) * (\mathbf{y}_j - \mathbf{y}_i) \right] \\
 \Delta y_{ik} &= \alpha a_{ik} / |b_{ik}|
 \end{aligned} \tag{8.25}$$

**Remarques :**

- L'opérateur '\*' dénote le produit terme-à-terme de deux vecteurs, qui donne également un vecteur.
- '1' dénote un vecteur de même dimension que  $\mathbf{y}_i$  ( $p$ ), dont toutes les composantes sont à 1.
- $C_{ij}$  est, comme  $X_{ij}$  et  $Y_{ij}$ , une matrice symétrique.
- La constante  $c = \sum \sum X_{ij}$ , présente dans (8.23), disparaît puisqu'elle se trouve à la fois au numérateur et au dénominateur de (8.24).

## 8.4 Comparaison avec VQP

Sammon note deux limitations au NLM : La première est la faible fiabilité de la représentation obtenue lors de l'utilisation de l'algorithme sur des structures complexes en grandes dimensions. La seconde est que les distances  $X_{ij}$  et  $Y_{ij}$  doivent être stockées en mémoire, ce qui limite le nombre de points pouvant être traités. De nos jours, cette limitation est moins importante, compte tenu des tailles des mémoires et de la puissance des machines actuelles, qui permettent le cas échéant de re-calculer à chaque itération une partie des distances, au prix bien sûr d'une forte augmentation du temps de calcul.

---

L'opérateur  $\nabla$  (*nabla*) est l'opérateur différentiel  $[\frac{\partial}{\partial \psi_1}, \dots, \frac{\partial}{\partial \psi_m}]^T$ , et  $\nabla E$  est le gradient de  $E$ .  $H$  est la *matrice Hessienne* qui regroupe les  $m^2$  dérivées secondes de  $E$  ( $H = \nabla^T \nabla E$ ). Mais, comme dans le cas multidimensionnel  $H^{-1}$  est généralement très lourde à calculer (inversion d'une matrice  $m \times m$ ), on pousse parfois, comme ici, l'audace jusqu'à l'assimiler à une matrice diagonale (c'est-à-dire qu'on néglige tous les termes non-diagonaux). Cela donne la méthode dite "pseudo-Newton".



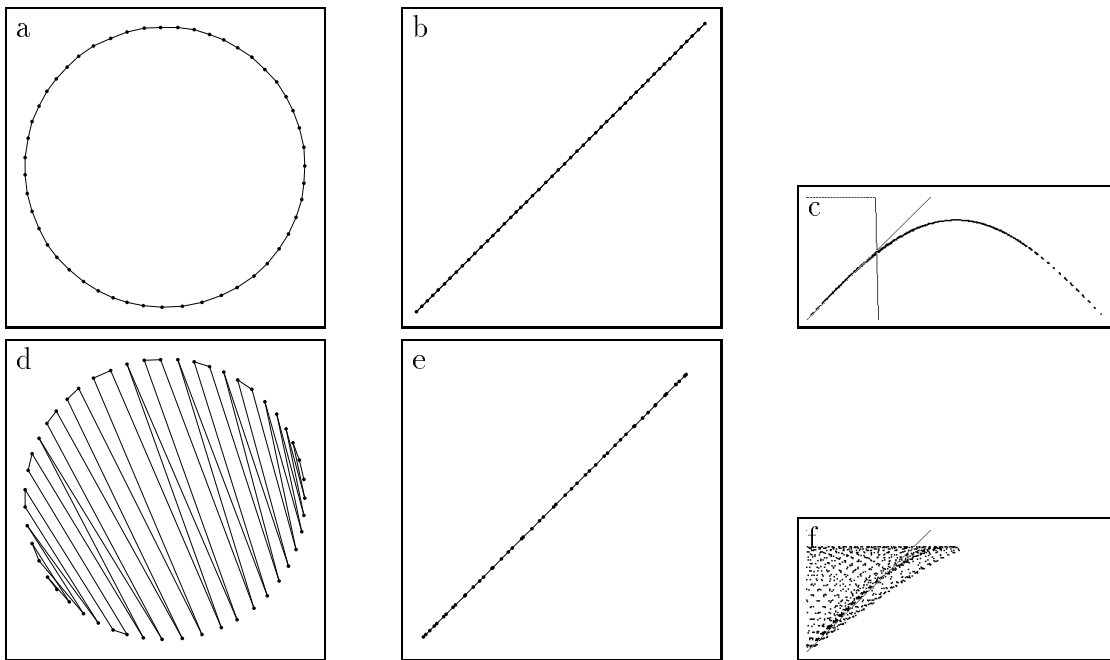


Figure 8.1: Comparaison du VQP (en haut) et du NLM (en bas) sur l'exemple du cercle projeté vers une dimension. Afin de visualiser la correspondance entre les espaces d'entrée et de sortie, on a relié les points à leurs plus proches voisins dans l'espace de sortie. On constate ainsi qu'avec VQP c'est l'abscisse curviligne du cercle qui est projetée sur la sortie, alors qu'avec le NLM la projection n'a pas déplié le cercle. La comparaison des représentations  $dy-dx$  confirme ce fait : en (f), on voit que la topologie du cercle n'est pas conservée par le NLM.

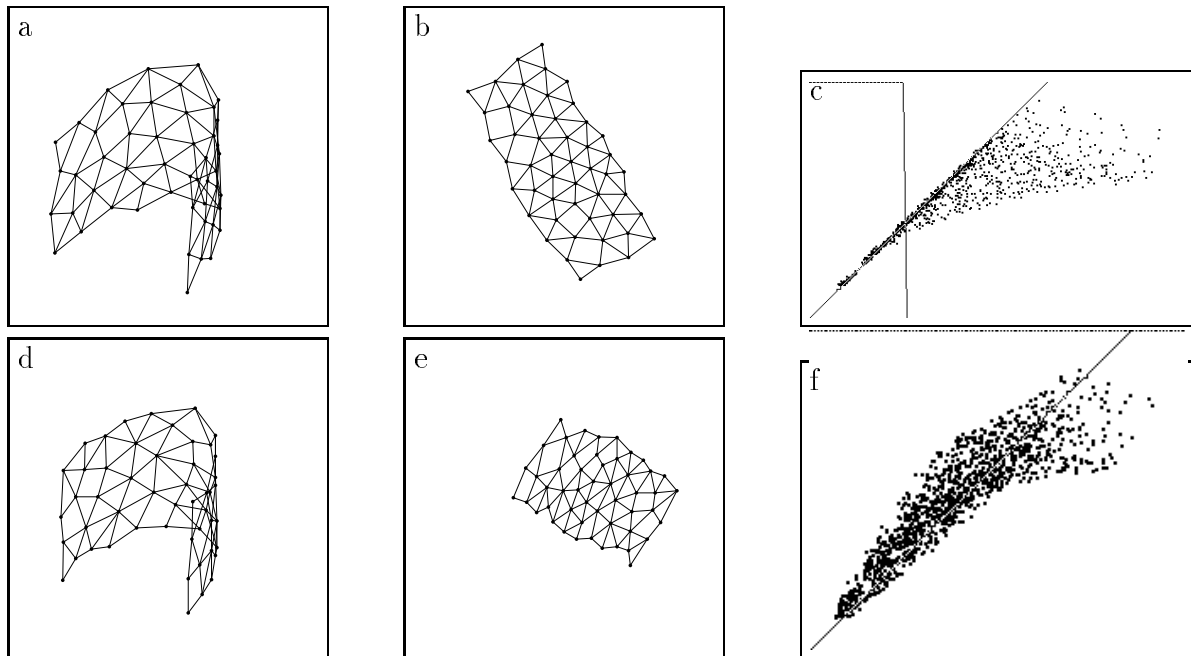


Figure 8.2: Comparaison de VQP (en haut) et du NLM (en bas) sur l'exemple du fer à cheval. On constate que le NLM ne déplie pas aussi bien l'abscisse curviligne du fer à cheval, qui reste un peu "contractée".

Par expérience, nous ajouterons cependant une troisième limitation au NLM : l'algorithme tente une projection directe en fonction des distances mesurées dans l'espace d'entrée. La pondération des termes  $(X_{ij} - Y_{ij})^2$  dans l'énergie est calculée en fonction des distances d'entrée  $X_{ij}$ . On a ainsi remarqué qu'il était très difficile de "couper" des structures refermées, ou même simplement de représenter des structures fortement pliées. Ainsi, les possibilités de dépliage du NLM sont assez réduites. Au contraire, dans les cartes de Kohonen comme dans VQP, le respect de topologie est imposé dans un voisinage dans l'espace de sortie, ce qui permet cette coupure nécessaire dans certaines structures de données.

Les exemples des figures 8.1 à 8.2 illustrent cette différence fonctionnelle importante.

Une autre différence est d'ordre technique, mais elle est aussi importante. Elle concerne les performances en vitesse et en occupation mémoire. Dans le NLM, la fonction d'énergie (8.23) est minimisée par une méthode pseudo-Newton, dont la complexité est équivalente à celle du calcul du gradient de l'énergie. Cette complexité pour  $N$  neurones,  $n$  dimensions en entrée et  $p$  dimensions en sortie, est d'ordre  $O(pN^2)$  pour le calcul de chaque itération, et  $O(N^2)$  pour l'occupation mémoire (par les distances). Si, de plus, on n'a pas la possibilité de stocker les  $N(N-1)/2$  distances  $X_{ij}$ , le calcul de chaque itération devient d'ordre  $O[(p+n)N^2]$ . En outre, pour des raisons déjà discutées (§ 6.2), les méthodes de type "descente du gradient" appliquées dans le cas particulier souffrent d'un "effet de moyenne" qui amortit les modifications faites à chaque itération et favorise le blocage dans des minima locaux de la fonction d'énergie.

La règle d'adaptation que nous avons proposée pour VQP (6.6) ne souffre pas de ces défauts. Elle ne montre pas d'effet de moyenne, et, jusqu'à présent, nous n'avons jamais observé, lors des simulations, de blocage dans un minimum local. Enfin, elle est beaucoup plus légère en temps de calcul (d'ordre  $O[(p+n)N]$  par itération) et aussi en occupation mémoire (les distances n'ont pas besoin d'être stockées; on calcule les  $2(N-1)$  distances nécessaires à chaque itération). La figure 8.3 montre l'évolution temporelle de l'énergie pour VQP et le NLM. La tâche est simplement la projection d'un carré vers un espace en deux dimensions (pas de réduction de dimension, il s'agit seulement de mettre en ordre les unités). Il y a 200 points (pour le NLM, on doit stocker environ 40 000 distances, dont la moitié sont re-calculées à chaque itération). Puisqu'il n'y a pas de réduction de dimension, on peut calculer l'énergie sans pondération par les distances, qui est aussi la partie commune dans l'expression de l'énergie pour les deux algorithmes (et permet ainsi de les comparer) :

$$E = \frac{1}{2} \sum_i \sum_{j \neq i} (X_{ij} - Y_{ij})^2 \quad (8.26)$$

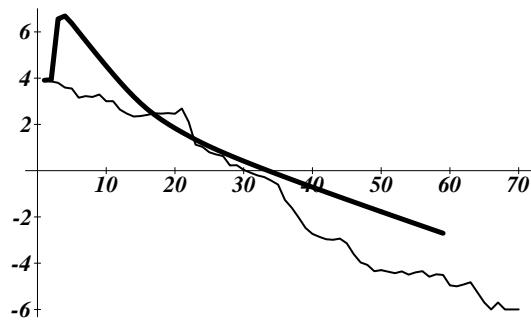


Figure 8.3: Evolutions comparées de l'énergie entre l'algorithme VQP (ligne fine) et le NLM (ligne épaisse), en fonction du nombre d'itérations. Pour les deux algorithmes, on trace  $\log_{10}(E)$  en fonction du nombre d'itérations. Cette graduation en nombre d'itérations ne fait pas apparaître l'importante différence de vitesse qui existe entre les deux algorithmes (voir figure 8.4).

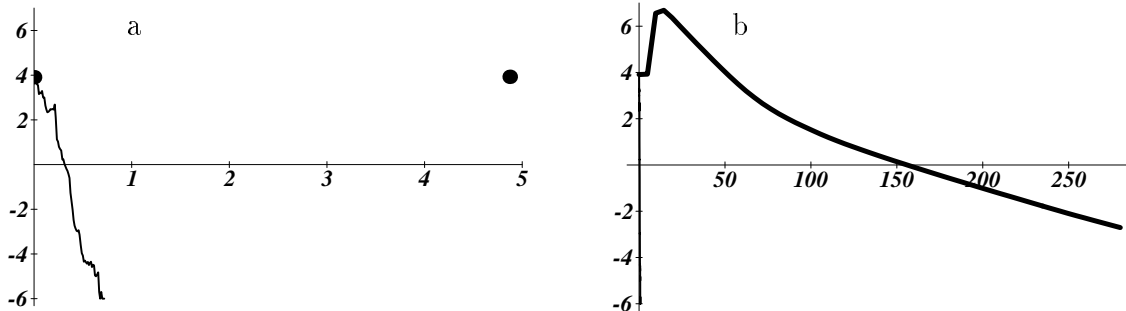


Figure 8.4: On trace à nouveau l'évolution des énergies de VQP et du NLM, mais cette fois-ci en fonction du temps de calcul. Compte tenu de la grande différence d'échelle (0.72 [s] pour VQP contre 280 [s] pour le NLM), il faut faire deux graphiques différents. En (a), on voit l'évolution de VQP (jusqu'à 0.72 [s], instant où son énergie affichée est tombée en dessous de  $10^{-6}$ ); il faut attendre presque 5 [s] pour que le NLM ait fini sa *première* itération ! En (b), les deux algorithmes sont aussi représentés, mais avec l'échelle de temps du NLM : à cette échelle, la courbe pour VQP est confondue avec l'axe vertical.

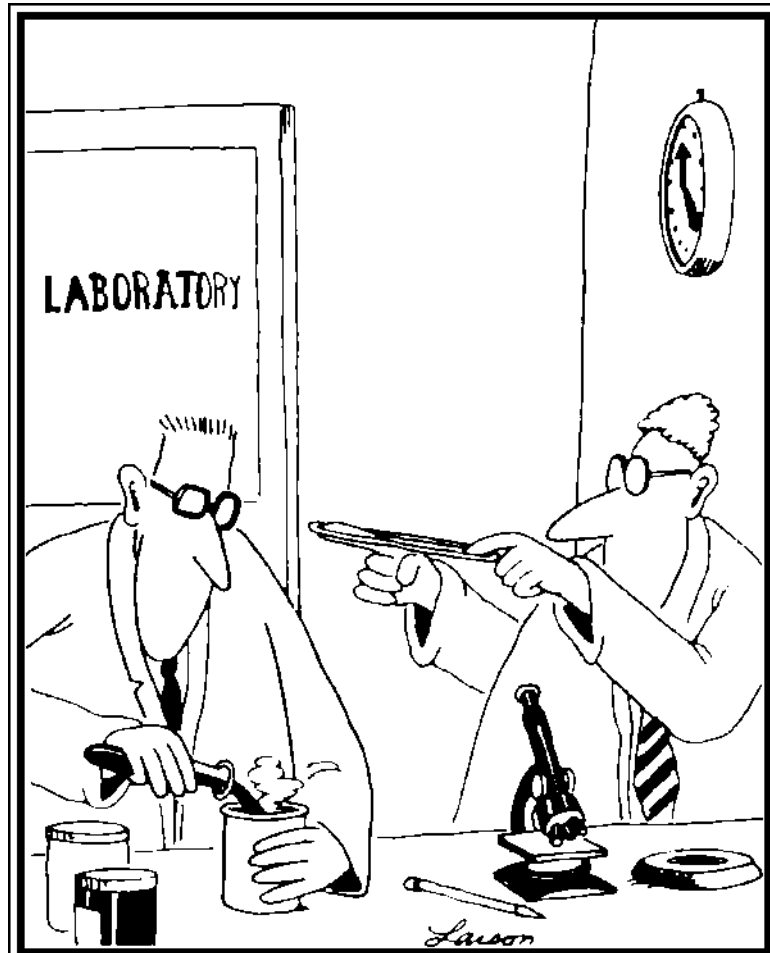
La figure 8.3 montre  $\log_{10}(E)$  en fonction du nombre d'itérations. Compte tenu de la grande différence de complexité des calculs, on donne aussi une illustration de  $\log_{10}(E)$  en fonction du *temps de calcul*<sup>3</sup>, qui est comme prévu nettement à l'avantage de VQP (figure 8.4). Les valeurs finales obtenues sont :

	VQP	NLM
Temps total	0.719 sec	280 sec
Itérations	70	80
Erreur finale	$< 10^{-6}$	0.0025

On observe qu'après une courte phase croissante (due aux trop grands pas faits au début par la méthode pseudo-Newton), l'énergie du NLM décroît de façon monotone et douce. Au contraire, l'énergie de VQP évolue de façon non monotone, conformément à ce qu'on avait prévu (§ 6.2), mais beaucoup plus rapidement. Il est intéressant de constater que, même selon le nombre d'itérations, l'énergie de VQP décroît globalement plus vite que celle du NLM.

---

<sup>3</sup>Sur une station de travail SunSPARC 10 mod. 41.



On Oct. 23, 1927, three days after its invention, the first rubber band is tested.

The Far Side<sup>4</sup>

By Gary Larson

---

<sup>4</sup>The Far Side cartoon by Gary Larson is reprinted by permission of Chronicle Features, San Francisco, CA. All rights reserved.

# Chapitre 9

## Applications

### 9.1 Vers une procédure d'analyse de données

Pour retrouver les informations cachées qui sont contenues dans les données dont on dispose, nous avons analysé ou développé dans les chapitres précédents plusieurs techniques complémentaires, dirigées selon plusieurs axes :

- Recherche de la dimension intrinsèque du nuage.
- Réduction du nombre de points, par quantification vectorielle (réduction à un nombre d'échantillons pertinents).
- Réduction de la dimension, par projection non linéaire des échantillons conservés (dépliage de la variété).

Ces axes et les problèmes qu'ils soulèvent sont liés entre eux. Considérons par exemple le problème de la réduction en nombre de points, c'est-à-dire le problème de la quantification vectorielle. On doit se poser les questions suivantes (voir aussi la figure 9.1) :

- Manque-t-il des points dans nos données ?
- Y a-t-il des points aberrants (des erreurs de mesure) ?
- Quelle est l'importance du bruit autour de la variété des données ?
- Combien faut-il prendre de points représentatifs ? (Ce nombre est lié à la dimension intrinsèque des données, et non à la dimension brute).

Toutes ces questions n'ont *a priori* pas de réponse. On peut tirer parti des connaissances des experts du phénomène dont on analyse les données, lorsqu'elles sont

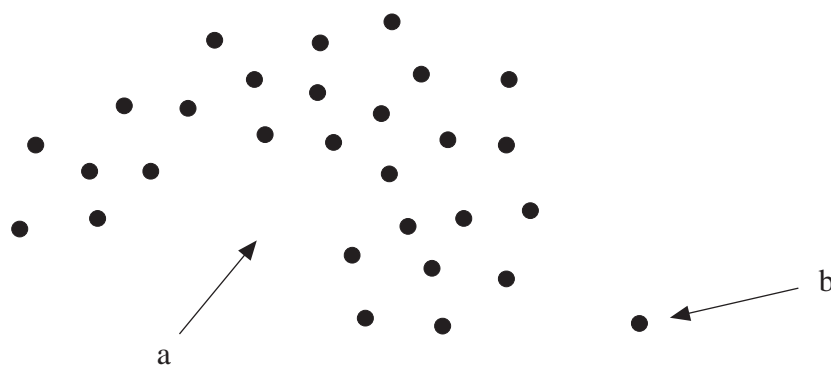


Figure 9.1: Quelques questions difficiles : Y a-t-il du bruit de mesure ? En (a), manque-t-il des points dans les données ? En (b), le point est-il aberrant ? (ou au contraire très porteur d’information parce que rare ?).

disponibles. Dans le cas contraire, il faut faire des hypothèses de *régularité* (selon quel critère ?), par exemple en “lissant” les données, au risque de combler artificiellement des trous. On peut aussi éliminer les points aberrants selon un critère d’homogénéité sur les données (comment le définir ?). En fait, toutes ces questions sont liées à la notion d’*échelle* : échelle de ce que l’on doit détecter, échelle du bruit, échelle dans le lissage, ...

Le choix du nombre d’échantillons est encore un problème d’échelle : avec un nombre d’échantillons trop important, on représente aussi le bruit. A l’inverse, prendre un nombre de points trop faible revient à trop lisser la distribution. Par ailleurs, le nombre de points nécessaires pour obtenir une quantification vectorielle à une certaine échelle (le “pas” de quantification) dépend de la dimension intrinsèque de la distribution et non de la dimension brute.

Nous avons vu que la recherche de dimension intrinsèque (§ 2.5.2) faisait également apparaître le même type de problèmes. En effet, une échelle trop grossière laisse “échapper” des détails importants de la distribution. A l’inverse, une échelle trop fine fait prendre l’inévitable bruit de mesure pour des dimensions structurantes. Enfin, une échelle encore plus fine (à laquelle on voit la distribution comme un ensemble de points très distants les uns des autres) indique la dimension des points isolés, c’est-à-dire 0.

Pour déterminer le nombre d’échantillons “pertinents” à garder, il faut avoir déterminé la dimension intrinsèque du nuage. Mais la détermination de cette dimension dépend de l’échelle à laquelle on observe ce nuage. Cette échelle elle-même dépend du pas de quantification, donc du nombre de points que l’on va garder... A l’inverse, pour déterminer la dimension des données, il faut quantifier l’espace par un nombre de points qui dépend de cette dimension : la *solution est récursive*.



Nous proposons ainsi une procédure d'analyse de données qui permet d'ajuster empiriquement les différents paramètres. Cette procédure peut être appliquée de façon itérative :

- Fixer une échelle d'observation (en fonction du nombre de points disponibles, et, si cette information est disponible, en se basant sur la connaissance *a priori* de l'importance du bruit).
- Rechercher la dimension intrinsèque globale du nuage.
- Décider un nombre de points en fonction de cette dimension.
- Faire la quantification vectorielle du nuage avec ce nombre de points.
- Effectuer la projection non linéaire de ces points, vers un espace dont la dimension est un nombre arrondi autour de la dimension intrinsèque du nuage.
- A l'aide de la représentation  $dy - dx$  (qualité de la projection), on voit si la dimension de sortie est suffisante (si la corrélation entre  $X_{ij}$  et  $Y_{ij}$  n'est pas bonne, même localement, la dimension de sortie est probablement insuffisante) ou non et si la projection est fortement non linéaire ou pas (un diagramme incurvé est signe d'un dépliage important).
- Faire une projection et projection inverse des données. La distance entre  $\xi$  et  $\text{proj}^{-1}(\text{proj}(\xi))$  permet de déterminer si l'on a trop "lissé les virages" ou au contraire si l'on a appris le bruit.
- S'il y a beaucoup de points "aberrants" (trop loin de la variété), cela peut signifier que l'on a trop lissé les données par un nombre d'échantillons insuffisant, ou bien aussi que l'on dispose de données brutes en nombre insuffisant. Il faut donc essayer avec d'autres valeurs pour le nombre de points de quantification vectorielle, ou bien avec une autre dimension de sortie.

Dans les sections suivantes, nous utilisons cette procédure dans des exemples tirés de domaines très différents, afin de montrer l'étendue des applications potentielles d'une telle méthode d'analyse de données. Ces applications vont de la fusion de données à l'appariement de graphes, en passant par l'analyse de procédé et la fabrication de métrique.

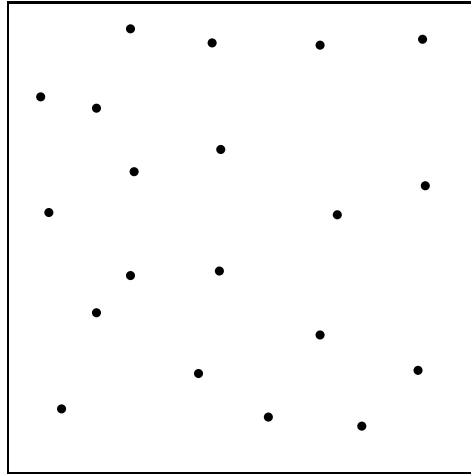


Figure 9.2: Placement sur un plan des 20 capteurs de distances pour le problème de localisation (voir texte).

## 9.2 Fusion multi-capteurs

La biologie fournit une multitude d'exemples où un très grand nombre de variables se trouvent liées par une dimension structurante relativement faible. Par exemple, la perception du mouvement propre fait intervenir un grand nombre de capteurs visuels (flot optique), des données de notre centrale à inertie (l'oreille interne), éventuellement le sifflement de l'air... Pourtant toutes ces valeurs de modalités différentes sont liées par le nombre limité de degrés de liberté du mouvement.

### 9.2.1 Localisation

A titre d'illustration, nous reprenons l'exemple du problème de localisation cité en introduction (§ 1.1).

Nous avons placé 20 capteurs au hasard sur un plan (dans un carré  $[0, 1]^2$ ). Le placement de ces capteurs est donné à la figure 9.2.

Ensuite, nous générons une distribution d'apprentissage constituée de 10 000 vecteurs. Pour chaque vecteur, on a tiré au hasard un point  $s$  dans le plan (toujours dans le carré  $[0, 1]^2$ ), et l'on mesure les distances entre ce point et les 20 capteurs, comme illustré à la figure 1.1 (page 21). La collection de ces distances constitue le vecteur, qui est donc à 20 composantes.

La distribution en 20 dimensions qui résulte est vue sous plusieurs angles dans la figure 1.2 (page 22).

L'analyse de dimension fractale par la méthode des boîtes (§ 2.5.3) indique une dimension intrinsèque de la distribution de 1.96 (idéalement, on devrait trouver 2,

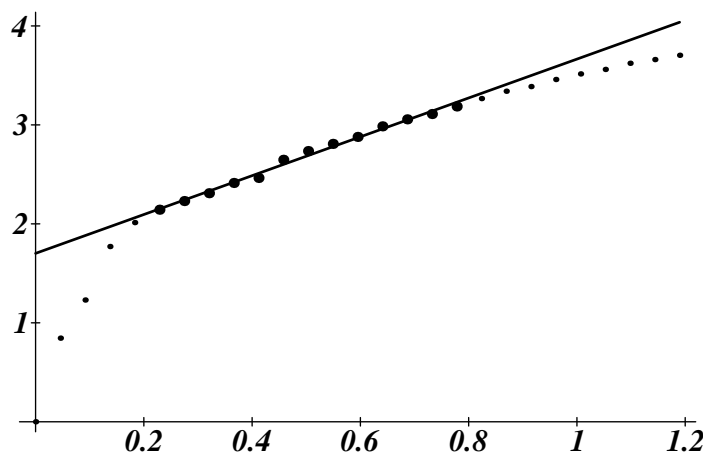


Figure 9.3: Estimation de la dimension fractale du problème de localisation. La dimension brute est 20. La régression effectuée ici sur la relation  $\log N$  (ordonnée) en fonction de  $\log r_0/r$  donne une pente de 2 environ, ce qui est la dimension intrinsèque du problème. La portion de la caractéristique choisie pour faire la régression correspond aux points donnant un nombre de boîtes entre 100 et 1000 (le nombre total de points disponibles pour cette distribution étant 10 000).

puisque le problème possède deux degrés de liberté). La régression linéaire qui a permis de trouver cette valeur (c'est la pente de la caractéristique  $\log N(\log r_0/r)$ ) est illustrée à la figure 9.3 (identique à la figure 2.9 de la page 46). L'intervalle dans lequel on a sélectionné les points pour effectuer la régression a été choisi empiriquement sur la base de l'apparence de la caractéristique complète  $\log N(\log r_0/r)$ . Nous avons sélectionné les points correspondant à un nombre de boîtes compris entre 100 et un peu plus de 1000 (entre 2 et 3 sur l'échelle logarithmique). En dessous de 100 boîtes, l'échelle est trop grossière, et la pente locale est trop élevée (proche de la dimension brute). A l'inverse, pour un nombre de boîtes trop important, il n'y a plus suffisamment de points par boîte pour évaluer correctement la dimension fractale (à l'extrême, quand il n'y a plus qu'un point par boîte, on trouve une pente nulle).

On voit l'aspect empirique de ce choix d'échelle pour l'estimation de la dimension intrinsèque, même dans un cas comme celui-ci où il n'y a pas de bruit de mesure.

Comme il semble toutefois que c'est à partir de 100 boîtes que la caractéristique permettant de trouver la dimension intrinsèque est stable (ce qui signifie que c'est à partir d'une subdivision en 100 boîtes que l'on repère la structure bi-dimensionnelle de la variété), on utilise un réseau VQP composé de 100 neurones, et de 2 dimensions pour l'espace de sortie.

Le résultat obtenu est montré à la figure 9.4.a. On y voit les poids de l'espace

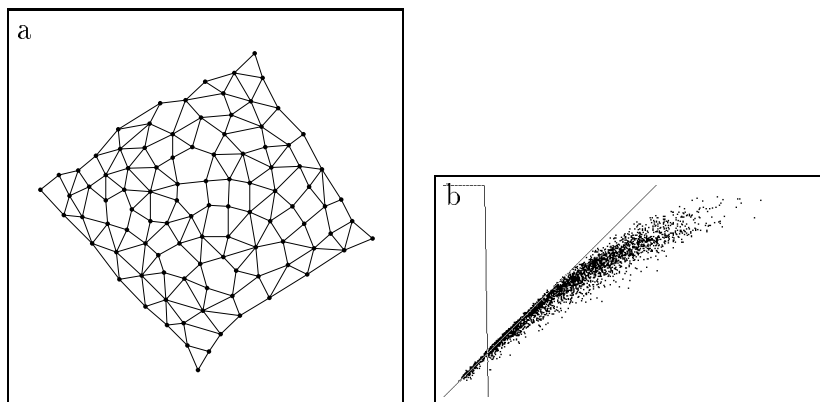


Figure 9.4: Résultat de VQP pour le problème de localisation. (a) Les poids dans l'espace de sortie. (b) Représentation  $dy - dx$ .

de sortie, c'est-à-dire la projection en 2 dimensions des 100 centroïdes trouvés par quantification vectorielle de l'entrée.

La représentation  $dy - dx$  (figure 9.4.b) indique une bonne conservation locale de la topologie. On voit cependant que la distribution était pliée dans l'espace de départ (comme on peut le vérifier dans la réduction à trois dimensions par ACP, figure 2.6, page 40). En effet, la caractéristique  $dy - dx$  s'incurve vers l'horizontale pour les grandes distances  $Y_{ij}$ , ce qui est typique des replis.

Enfin, dans la figure 9.5, on a projeté une distribution de test, formée de la même façon que la distribution d'apprentissage, mais avec des points  $s$  tirés sur une grille, et non dans tout le carré  $[0, 1]^2$ . Cette projection montre que, à quelques défauts près sur les bords, l'espace paramétrique du problème a bien été retrouvé par VQP.

Des essais avec un nombre différent de capteurs (10, 12, 15, ...) donnent les mêmes résultats, tant sur le plan de l'analyse de dimension que sur celui de la projection par VQP. En revanche, l'analyse de dimension fractale semble très sensible au bruit : nous avons ajouté une petite valeur aléatoire à chaque composante; le diagramme  $\log N(\log r_0/r)$  ne présente plus alors de zone où la pente est stable. On a constaté que le fonctionnement du réseau VQP, par contre, est fort peu affecté par ce bruit. Nous n'avons pas d'explication pour ce fait étonnant; il mériterait une étude approfondie.

## 9.2.2 Fusion auditive et visuelle

Dans cet exemple, nous illustrons une fusion de données multi-modales. Des données visuelles et auditives représentatives de phonèmes correspondant aux dix voyelles principales du français sont analysées et représentées.

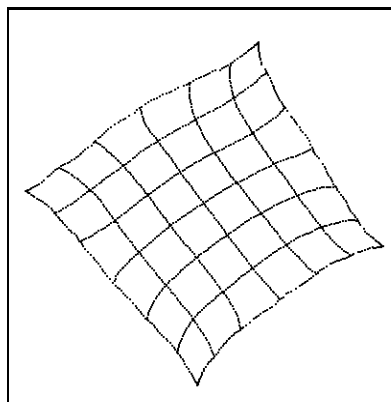


Figure 9.5: Grille de test, fabriquée dans l'espace des distances de la même façon que la distribution complète du problème de localisation, puis projetée dans l'espace de sortie de VQP en s'appuyant sur les poids trouvés à la figure 9.4.

Les résultats que nous présentons ici sont extraits d'une étude préliminaire menée en collaboration avec l'Institut de la Communication Parlée de l'Institut National Polytechnique de Grenoble (ICP-INPG). Le thème de cette collaboration est l'étude de la fusion de données pour améliorer la reconnaissance de parole en milieu bruité. Nous avons utilisé les données récoltées et mises en forme par l'équipe de l'ICP.

Chaque échantillon est un phonème représenté par un vecteur à 23 dimensions. Celui-ci est formé par concaténation de 20 dimensions de nature auditive et de 3 dimensions de nature visuelle. Les 20 composantes auditives sont les sorties d'un banc de filtres. Elles représentent un échantillonnage du spectre d'amplitude entre 0 et 5 KHz selon une échelle non-linéaire (les "barks"<sup>1</sup>). La source visuelle comporte 3 paramètres géométriques de la forme des lèvres pendant la prononciation de la voyelle (largeur, hauteur et surface).

Une normalisation par centrage et réduction à 1 de la variance des composantes a été faite. On sait d'emblée que les données sont des amas de points pour chaque phonème, avec un bruit assez important (la différence d'une façon de prononcer à l'autre). On n'est donc pas dans le cas d'une distribution continue sur une variété, et l'analyse de dimension ne fournit aucune indication. La figure 9.6.a montre la caractéristique  $\log N(\log r_0/r)$  de cette distribution. On vérifie qu'il serait malaisé d'y choisir une région pour trouver une pente stable correspondant à une dimension intrinsèque. Dans le cas particulier, il faut considérer que la distribution ne contient que 10 points (les 10 voyelles), mais avec plusieurs exemplaires bruités de chacun d'eux.

<sup>1</sup>La transformation entre Hz et bark est donnée par :  $\text{bark} = 7 \operatorname{arcsinh}(\text{Hz}/650)$ . Il s'agit d'une échelle fréquemment utilisée en analyse de parole.

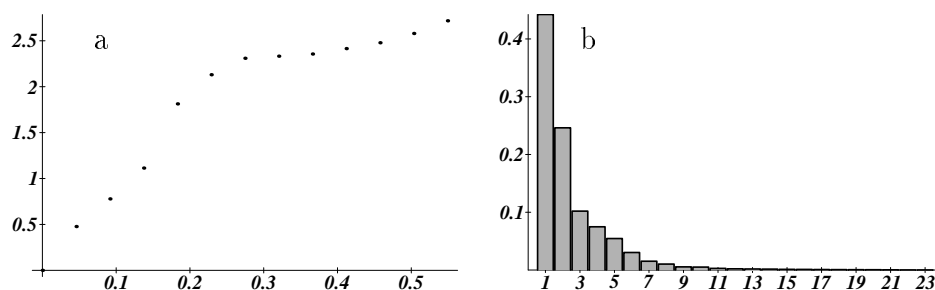


Figure 9.6: (a) Caractéristique  $\log N(\log r_0/r)$  de la distribution des phonèmes. On constate qu’il n’y a pas de zone où la pente est stable. On ne peut pas définir de dimension intrinsèque. (b) Histogramme des valeurs propres trouvées par ACP sur la distribution de phonèmes.

L’analyse en composantes principales permet une réduction jusqu’à 12 ou 13 dimensions. L’histogramme des valeurs propres est donné à la figure 9.6.b.

Avec VQP, nous projetons directement l’ensemble de données vers 2 dimensions, après une quantification vectorielle avec 10 points. Le résultat de cette projection est donné à la figure 9.7, avec la représentation  $dy - dx$  associée. Les 10 vecteurs  $\mathbf{y}_i$  sont représentés par des cercles avec un numéro. Les points de la distribution complète sont projetés en surimpression. La répartition des phonèmes telle qu’elle est reconstruite par VQP rappelle fortement une représentation connue sous le nom de “triangle vocalique”. Cette représentation est une vue des voyelles dans le plan F1-F2, où F1 est le premier formant du spectre (la position du premier maximum) et F2 le second.

La correspondance entre les numéros et les phonèmes est la suivante (selon le même arrangement qu’à la figure 9.7) :

3	/i/ (pile)	6	/y/ (rue)	9	/u/ (poux)
2	/e/ (thé)	5	/ø/ (jeu)	8	/o/ (eau)
1	/ɛ/ (raie)	4	/œ/ (beurre)	7	/ɔ/ (fort)
		0	/a/ (patte)		

En réalité, les modèles actuels de la cavité buccale qui rendent le mieux compte du mécanisme de formation de voyelle font intervenir trois paramètres géométriques fondamentaux. Il s’agit de la position de la langue (avant-arrière, haut-bas) et de l’ouverture des lèvres.

Selon une idée qui rejoint la *théorie motrice* (notre perception passerait par la capacité à se représenter les mouvements permettant de reproduire le percept), la reconnaissance des voyelles se ferait dans cet espace tridimensionnel des paramètres de leur *production*. Ainsi, nous avons proposé de fixer l’espace de sortie

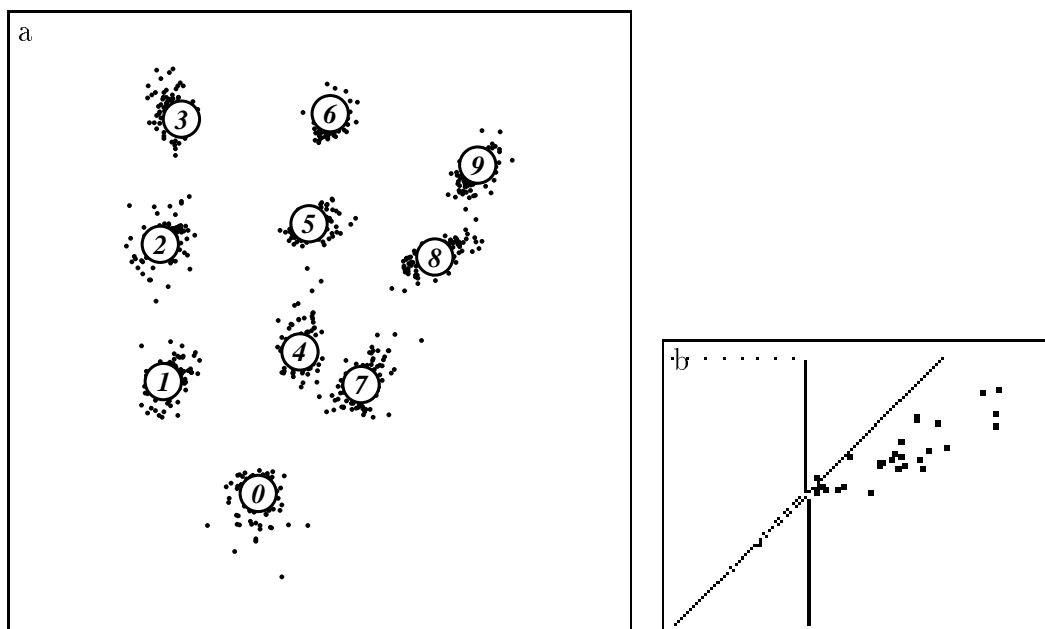


Figure 9.7: Résultat de VQP pour la distribution des phonèmes. (a) les 10 vecteurs-poids dans l'espace de sortie, ainsi que la projection de la distribution complète. (b) représentation  $dy - dx$  des poids.

de VQP à 3 dimensions plutôt qu'à 2, et de contraindre certains points connus dans cet espace, afin de fixer l'orientation de l'espace (normalement, avec VQP, l'espace de sortie est libre en rotation, translation et symétrie). L'idée est de permettre l'association entre l'espace de perception (les 23 dimensions d'entrée) et une représentation motrice. On espère ainsi reconnaître les phonèmes directement dans l'espace moteur, avec une sensibilité au bruit moins grande que dans l'espace d'entrée. Cette étude fera l'objet d'un stage de Diplôme d'Etudes Approfondies (DEA).

### 9.2.3 Apprentissage de séquences temporelles

Ici, nous nous intéressons au problème très général de l'apprentissage d'une séquence de motifs d'activité d'un ensemble de capteurs. Bien que le nombre des capteurs et celui de leurs motifs d'activité puissent être grands, nous allons voir comment VQP peut retrouver le temps comme dimension structurante du problème.

Imaginons une surface sensible (un écran tactile, par exemple) composée de  $11 \times 11$  pixels. Nous traçons un huit sur cette surface, générant ainsi des motifs d'activité des pixels différents au cours du temps. Un élément essentiel du système,

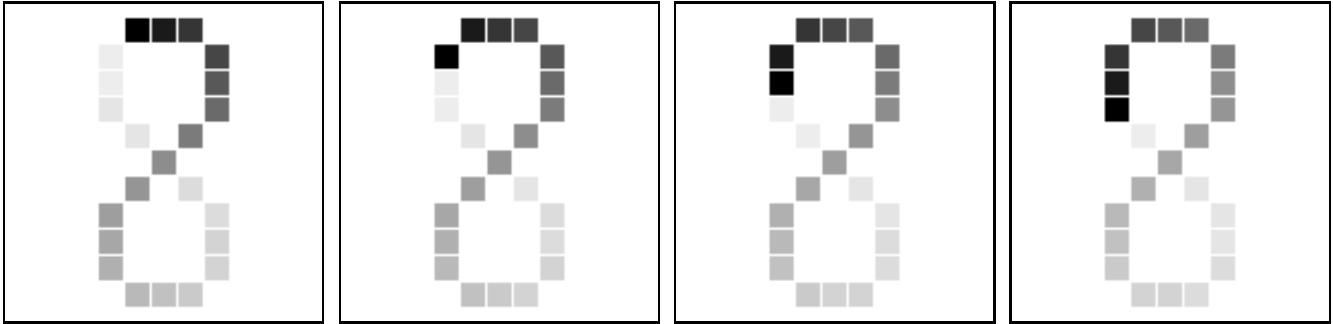


Figure 9.8: Les quatre premiers vecteurs de l'ensemble d'apprentissage d'une séquence temporelle.

qui va permettre une forme d'auto-organisation, est la propriété de *rémanence* que possèdent les capteurs de notre surface sensible. Ainsi, un capteur qui a été activé par le passage du stylo ne retombe pas tout de suite à 0, mais garde encore une petite activité après le passage. Le résultat sur les motifs d'activité va être une traînée derrière le passage du stylo. C'est cette traînée qui va fournir toute l'information nécessaire à l'organisation de VQP. Par exemple, bien que passant deux fois par le pixel central, la trajectoire ne se croise pas dans l'espace d'activité à 121 dimensions, grâce à la rémanence qui permet de désambiguïser le sens de passage.

Les quatre premiers motifs de la séquence apprise sont montrés à la figure 9.8. La façon dont on arrange les  $11 \times 11$  pixels du motif en un vecteur à 121 dimensions est sans importance. On peut par exemple simplement dérouler les pixels ligne par ligne pour former le vecteur.

Le huit complet est formé de 24 pixels. Chaque fois que l'on change de pixel avec le stylo, un nouveau motif est généré. Donc, on ne dispose que de 24 motifs différents. L'analyse de dimension fractale n'est pas applicable sur un nombre si réduit de points. Nous avons donc choisi de passer directement à une représentation par VQP. Compte tenu du faible nombre de points, on peut affecter un neurone à chacun d'entre eux (il n'y a donc pas de quantification vectorielle). Le résultat, avec un espace de sortie fixé à une seule dimension, est illustré à la figure 9.9. On remarque que l'arrangement des points est régulier, que la représentation  $dy - dx$  indique une bonne conservation de topologie, et enfin que la structure des données est re-bouclée (la caractéristique  $dy - dx$  redescend presque à 0 vers la droite).

La question la plus intéressante est de rechercher à quoi correspond le voisinage de l'espace de sortie. Pour trouver cela, on projette en sens inverse des points de l'espace de sortie vers l'entrée, et l'on visualise à quoi correspondent ces vecteurs d'entrée. Lorsqu'on passe ainsi d'un vecteur de sortie à son voisin, puis



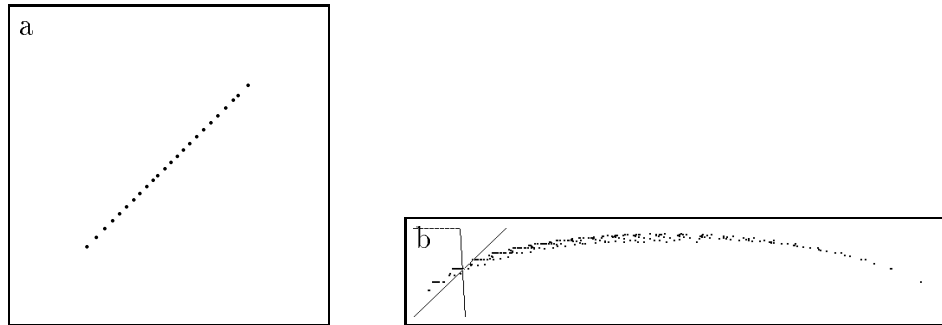


Figure 9.9: Résultat de VQP dans l'exemple de séquence temporelle. (a) Espace de sortie, en 1 dimension (où VQP a appris à retrouver le temps, d'une façon non supervisée). (b) Représentation  $dy - dx$ .

au suivant, et ainsi de suite, le parcours équivalent dans l'espace d'entrée (trouvé par projection inverse) correspond au parcours de la séquence temporelle dans l'espace des capteurs. Ainsi, grâce à la rémanence, VQP a arrangé les vecteurs dans un espace à une seule dimension en fonction de leur proximité temporelle. Peu importe l'ordre dans lequel les différents pixels sont rangés dans les vecteurs d'entrée (pourvu que ce soit toujours le même ordre). C'est la rémanence qui indique, en faisant apparaître une corrélation d'activité entre des pixels voisins dans la séquence, la topologie de l'espace.

De la même façon, nous pouvons reproduire les expériences de Von der Malsburg, en créant cette fois une corrélation spatiale (et non temporelle) entre pixels voisins, en présentant au réseau des taches *connexes* sur l'image (on obtient alors le phénomène de rétinotopie décrit au § 4.1).

Par contre, bien sûr, la présentation de motifs dans lesquels un pixel seul est actif ne fournit aucune information sur l'arrangement (spatial ou temporel), et aucune organisation n'est alors possible (il n'y a pas de redondance, donc pas de connaissance).

On peut aller plus loin dans notre exemple. En supposant des neurones qui peuvent se fatiguer (c'est-à-dire qui ont une période réfractaire après avoir été activés), on peut imaginer un système qui, partant d'une "amorce" (un motif proche d'un de ceux qui ont été appris), active le neurone qui a codé le motif le plus ressemblant. Cette activité se propage aux neurones voisins dans l'espace de sortie. Un de ces voisins, ayant une forte activité, devient gagnant à son tour (à la place du premier neurone gagnant qui est déjà entré dans sa période réfractaire). La répétition de ce processus permet de passer en revue tous les états successifs de la séquence. On peut faire ce qu'on veut de ces états : simplement les *évoquer* (comme dans une activité mentale, pour utiliser une image provocatrice), ou générer un plan d'actions.

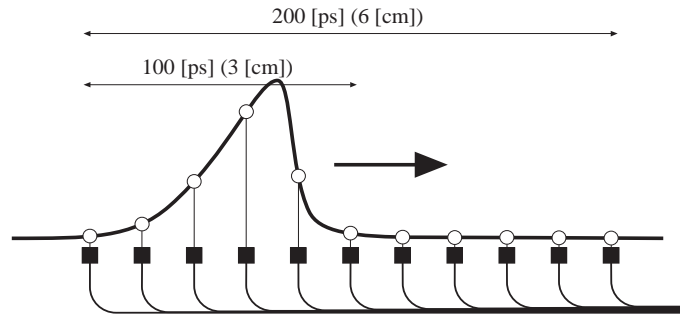


Figure 9.10: Schéma de principe du problème de détection précise d'impulsions (voir texte).

### 9.2.4 Algorithme de super-résolution

Dans le même ordre d'idées, on peut se servir de VQP pour retrouver une caractéristique, comme un pic ou quelque autre point identifiable d'un motif, à partir d'un échantillonnage de ce motif, avec une précision inférieure au pixel.

Par exemple, un problème de grande importance dans le traitement de données de détecteurs en physique des hautes énergies consiste à mesurer avec précision l'instant d'arrivée sur un capteur d'une impulsion, de forme connue, mais échantillonnée assez grossièrement [13].

Nous avons imaginé un système équivalent, mais qui met en jeu plusieurs capteurs, et qui est illustré à la figure 9.10. A intervalles de temps réguliers, la valeur de tous les capteurs est analysée. Lorsqu'une impulsion est présente, on souhaite connaître la position de son maximum avec une précision plus grande que l'écart entre deux capteurs (par exemple, dans la figure 9.10, on aimerait savoir que le maximum de l'impulsion se trouve entre les capteurs 4 et 5).

Supposons par exemple que l'on doive mesurer l'instant d'arrivée d'une impulsion dont la longueur est 100 ps (picosecondes), ce qui correspond à 3 cm pour une onde électromagnétique dans le vide. L'échantillonnage des valeurs fournies par les capteurs est fait à 5 GHz (fréquence qui correspond à une période de 200 ps, soit une longueur d'onde de 6 cm). Avec un seul capteur, on aurait environ une chance sur deux de détecter l'impulsion, et même dans ce cas on ne pourrait donner son instant d'arrivée qu'avec une erreur moyenne de  $\pm 100$  ps. Le système envisagé dans cet exemple est donc de disposer 11 capteurs répartis sur 6 cm. Quel que soit le moment d'arrivée de l'impulsion par rapport à la trame d'échantillonnage, cette impulsion est détectée simultanément par plusieurs capteurs.

Nous avons simulé un tel système en générant à des instants aléatoires des impulsions de la forme de la figure 9.10. Les activités des capteurs sont collectées dans des vecteurs à 11 dimensions. Quatre exemples de ces vecteurs sont donnés

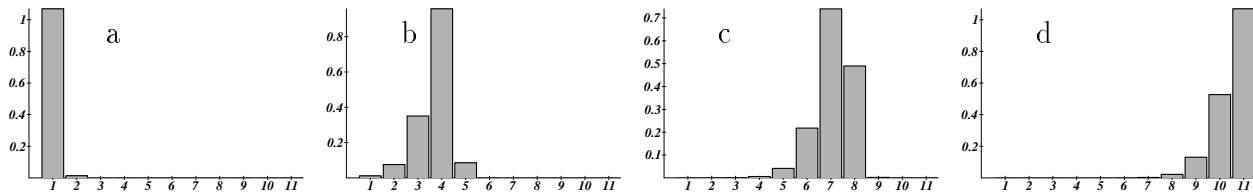


Figure 9.11: Quatre exemples de vecteurs pour le problème de la détection du moment d'arrivée des impulsions, correspondant à différents moments d'arrivée (0 ps, 66 ps, 133 ps et 200 ps) par rapport au coup d'horloge d'échantillonnage. La base d'apprentissage contient un grand nombre de ces vecteurs, dont on ne connaît pas le moment d'arrivée (celui-ci est aléatoire et on ne le conserve pas : l'apprentissage est *non supervisé*).

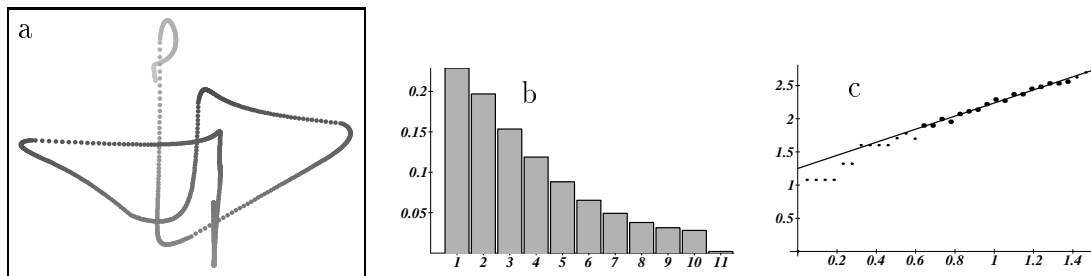


Figure 9.12: La distribution à 11 dimensions du problème de détection des impulsions (voir texte). Vue d'une projection (a). Histogramme des valeurs propres de l'ACP (b). Analyse de dimension fractale (c).

à la figure 9.11.

La distribution résultante est fortement non-linéaire (figure 9.12.a). Une analyse en composantes principales (on voit l'histogramme des valeurs propres à la figure 9.12.b) confirme qu'on ne peut pas réduire le nombre de dimensions par projection linéaire. En revanche, une analyse de dimension fractale (figure 9.12.c) indique que la dimension intrinsèque du problème est 1.

Lorsqu'on fait apprendre cette distribution à un réseau VQP, avec 11 dimensions d'entrée et une dimension de sortie, cette dernière s'auto-organise en une représentation continue de l'instant d'arrivée (à un facteur près) de l'impulsion, c'est-à-dire de l'information recherchée ! La représentation  $dy - dx$ , donnée pour contrôle à la figure 9.13.b, montre que le respect de topologie est très local : la variété de l'espace d'entrée est fortement pliée.

L'instant d'arrivée ainsi reconstruit est tracé en fonction de sa véritable valeur à la figure 9.13.a. On constate que la précision obtenue est bonne; la moyenne de l'erreur est d'environ 1.8 ps, soit 0.9% de la période d'échantillonnage (200 ps), ou encore 9 % de l'écart entre deux capteurs. Il faut bien comprendre que dans cette expérience la tâche a été apprise de façon non-supervisée, c'est-à-dire uniquement

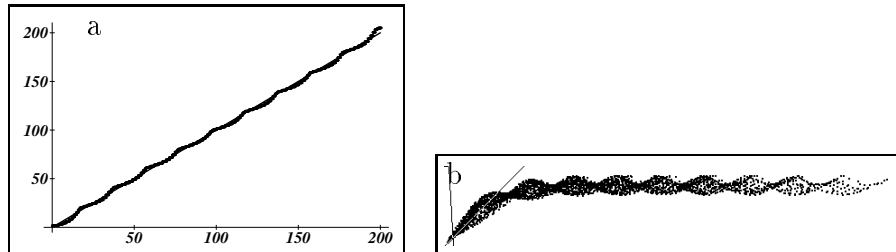


Figure 9.13: Résultat de VQP. En (a), l’instant d’arrivée de l’impulsion reconstruit par VQP en fonction de sa véritable valeur (cette “véritable” valeur étant celle qui a servi à générer le vecteur d’entrée). La représentation  $dy - dx$  (b) montre que la correspondance entre l’espace d’entrée et celui de sortie est fortement non-linéaire.

par auto-organisation sur la base des vecteurs d’activité des capteurs.

D’autres essais faits en mode *supervisé*, c’est-à-dire où l’on apprend le temps associé aux motifs présentés, permettent d’obtenir des précisions nettement supérieures (après apprentissage, on ne présente que les activités des capteurs; par projection depuis l’espace  $\mathbf{x}$  sans le temps vers  $\mathbf{y}$ , puis projection inverse de  $\mathbf{y}$  vers  $\mathbf{x}$ , on retrouve le temps). Nous nous sommes alors placés dans le même cadre expérimental que celui décrit dans [13] : il y a seulement 1 capteur, mais celui-ci est échantillonné suffisamment rapidement pour “voir” toujours trois valeurs à l’intérieur de l’impulsion. On a donc un système à 3 dimensions au lieu de 11. En outre, le système d’encodage assure que le maximum recherché se trouve toujours entre la première et la deuxième composante. Dans ces conditions, on arrive à une erreur maximum sur la position du pic égale à 0.075 % de l’écart entre 2 points d’échantillonnage, soit trois fois mieux que les résultats mentionnés dans [13]. Par contre, notre système ne travaille pas en temps réel, contrairement à celui décrit dans cet article.

### 9.2.5 Détection de fautes dans des circuits électriques

Il existe beaucoup d’applications des réseaux de neurones au contrôle et au diagnostic de *machines électriques* et de *systèmes de puissance* (une revue très complète est faite dans [96]; voir aussi [6, 94, 95]). Dans [19], une utilisation de VQP a été proposée pour le diagnostic de *convertisseurs triphasés* (“*pulse width modulator inverters*”).

Un convertisseur de puissance statique requiert souvent un haut degré de fiabilité de ses éléments pour bien fonctionner. Malheureusement, cette fiabilité n’est pas toujours garantie, et il est alors nécessaire de faire un diagnostic en temps réel pour éviter des dommages majeurs à la machine. Uniquement sur la base de mesures externes, un traitement rapide, bon marché et fiable doit être fait

pour indiquer, si possible d'une façon synthétique, l'état du système. Plusieurs méthodes ont été proposées dans ce but (par exemple des algorithmes heuristiques simples [92] ou des systèmes experts [101]). Elles répondent relativement bien aux exigences posées, mais ne sont pas capables de fournir à l'opérateur la visualisation synthétique et simple des différents états de marche du système.

L'utilisation d'une carte de Kohonen a déjà été proposée [20] pour résoudre ce problème. Une telle carte fournit une représentation simple de l'état du système et permet ainsi un diagnostic rapide et facile. En effet, l'opérateur peut suivre l'évolution du point de fonctionnement dans la carte, où les zones correspondant aux différents problèmes ont été préalablement marquées. Il repère ainsi immédiatement une tendance du système à se mettre en panne et sait aussi tout de suite de quelle panne il s'agit.

Le principal problème de l'utilisation du réseau de Kohonen, toutefois, est que dans le cas particulier la variété de la distribution est fortement pliée et que la région correspondant à une panne donnée peut être projetée dans des zones de la carte qui peuvent être loin les unes des autres (la région est coupée en morceaux).

Les vecteurs d'entrée de cette distribution sont constitués après une FFT (Transformée de Fourier Rapide) des courants des trois phases. Pour chaque phase, les 5 premières harmoniques du courant, normalisées par la valeur continue (DC), sont considérées. Les valeurs ainsi calculées sont collectées dans un vecteur à 15 dimensions, représentatif de l'état du système. Pour neuf états de charge considérés, 12 pannes différentes plus l'état de bon fonctionnement ont été simulés, formant une base d'apprentissage de 117 vecteurs.

L'utilisation d'un réseau VQP avec une dimension de sortie fixée à 2 donne, comme avec la carte de Kohonen, un résultat où certaines régions sont séparées (figure 9.14.a). Par contre, en donnant 3 dimensions à l'espace de sortie, les auteurs ont obtenu des régions simplement connexes (figure 9.14.b), c'est-à-dire non coupées, ce qui signifie que chaque état est représenté par un et un seul amas de points. Leur conclusion est que la variété du problème est 3, et qu'elle ne peut pas, même par dépliage, être ramenée à deux sans qu'il soit nécessaire de couper des régions.

Ils remarquent d'autre part que, s'il est concevable d'utiliser un réseau de Kohonen avec une grille à plus de deux dimensions, le calcul de voisinage devient lourd, et, par-dessus tout, le problème du choix de la forme de la carte (§ 4.6) devient encore plus difficile qu'avec deux dimensions. Avec VQP par contre, ce problème n'existe pas. Les tests de classification réalisés avec une centaine de vecteurs inconnus (c'est-à-dire n'appartenant pas à l'ensemble d'apprentissage) donnent un taux d'erreur de 5% environ [19].

Les mêmes auteurs proposent dans [21] d'utiliser VQP pour un autre problème de diagnostic, connu dans la littérature sous le nom de "diagnostic des

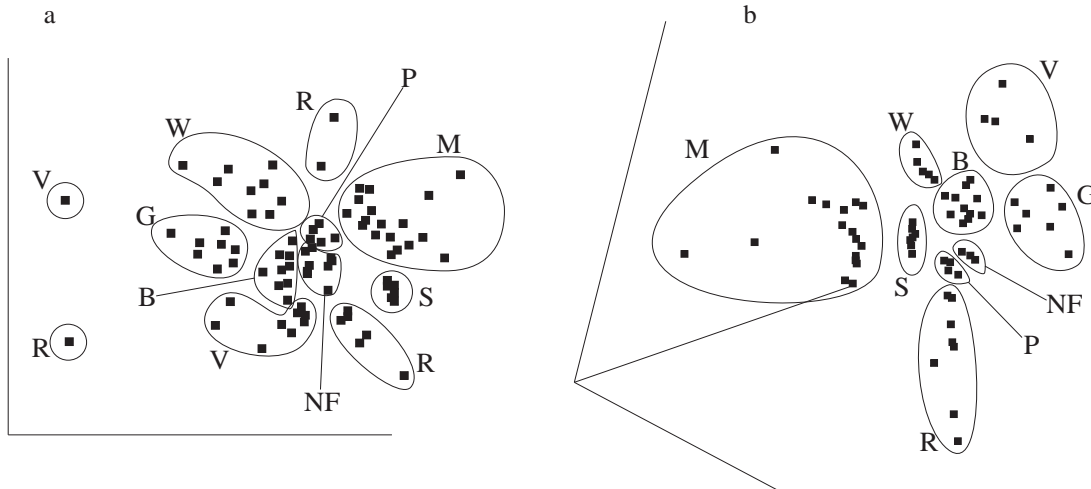


Figure 9.14: Résultats en sortie de VQP pour le problème de détection de fautes dans un convertisseur triphasé. En (a), un réseau avec deux dimensions de sortie a été utilisé : des régions sont coupées en deux, voire en trois. Par contre, lorsqu'on donne trois dimensions de sortie, les groupes ne sont pas coupés. Les différentes lettres correspondent à des types de pannes. Re-dessiné d'après Cirrincione *et al.* [19].

pannes naissantes” (“*incipient faults*”). Le problème standard, étudié dans de nombreux travaux, concerne un moteur à induction. Deux types de problèmes peuvent principalement se produire :

- Court-circuit entre deux spires ou davantage du bobinage du stator, se traduisant par une diminution du nombre de spires équivalentes, donc une baisse de couple, une augmentation du courant et un échauffement du moteur.
- Usure des paliers, provoquant l'augmentation du couple de frottement (et un échauffement des parties concernées).

De même que dans le cas des convertisseurs triphasés, on ne doit pas transformer le système ou attenter à son intégrité pour prendre les mesures. Les seules mesures que l'on peut faire sont des mesures *externes*. En l'occurrence, il s'agit du courant d'une phase du stator (mesuré avec un capteur à effet Hall), et de la vitesse angulaire du rotor (mesurée avec une dynamo tachymétrique).

Un réseau VQP est entraîné pour détecter l'état normal, en pré-panne ou en panne du système. Les considérations qui sont faites et les résultats obtenus par les auteurs sont similaires à ceux du problème du convertisseur triphasé ([21]).

## 9.3 Analyse de procédé

### 9.3.1 Extraction de séquence d'automate

La surveillance de procédé est actuellement un problème clé dans le domaine de la productique. En effet, l'automatisation des procédés industriels a permis une forte augmentation de la productivité, mais cette automatisation s'accompagne de risques de dysfonctionnement des éléments impliqués (actionneurs, capteurs, liens de communication, calculateurs, etc.). Bien que la cadence de production ait augmenté grâce à cette automatisation, le nombre de personnes présentes sur le lieu même de cette production a diminué (jusqu'à être nul dans certains ateliers). Cela signifie que les opérateurs de contrôle qui subsistent sont responsables d'une plus grande partie du procédé et doivent rapidement traiter, en cas de problème, un grand nombre d'informations.

Il se pose immédiatement le problème d'ergonomie et d'efficience dans la présentation des informations aux opérateurs, deux qualités qui sont souvent absentes ou peu présentes dans les systèmes actuels. Le plus souvent, les automatismes qui participent au contrôle d'un procédé ont une vue très locale de leurs actions. Ils sont pilotés et contrôlés par des micro- ou minicalculateurs qui rétro-agissent automatiquement sur le procédé ou confient ce rôle à l'homme. Le modèle utilisé lors de la définition des automatismes et de la programmation des calculateurs est généralement limité (pour permettre de réagir dans un délai très bref), hiérarchique et incomplet.

Ce modèle, par le nombre de variables en jeu (de l'ordre du millier voire de la dizaine de milliers), ne met pas en évidence certaines dépendances entre ses différents paramètres. Il existe donc des situations potentielles face auxquelles le comportement du contrôle de procédé n'est pas connu. En cas d'anomalie, il se produit souvent un phénomène bien connu des techniciens des salles de contrôle, appelé "avalanche des alarmes". En effet, le système de contrôle définit, pour chaque paramètre, des seuils à partir desquels une alarme se déclenche. Lorsqu'un problème grave apparaît, un nombre important — et dans un délai parfois court — de telles alarmes sollicite alors simultanément l'opérateur, qui ne sait plus quelle importance relative leur donner et laquelle traiter prioritairement. Une telle situation se termine généralement par l'arrêt du procédé. Ce type d'arrêt "d'urgence" est particulièrement ennuyeux parce qu'il laisse l'automate n'importe où dans sa séquence. La remise en route est alors souvent longue et difficile.

L'un des premiers pas pour tenter d'apporter une solution à ce problème consiste à tenter de présenter à l'utilisateur un "résumé" de la situation, c'est-à-dire une vue synthétique de l'état du système, de ses tendances, voire de mettre au point un mécanisme d'anticipation des alarmes. Lors de l'annonce d'une alarme

prochaine, l'opérateur pourrait tenter d'en identifier les causes et d'y remédier. Le cas échéant, on pourrait encore procéder à temps à un arrêt du système dans une configuration saine, ce qui réduirait le temps de remise en marche.

Dans le projet "Vigilance Neuromimétique", en collaboration avec le groupe automobile PSA, la société ITMI et la société Christol Consultants, nous avons tenté de mettre au point un tel mécanisme d'analyse et d'anticipation des alarmes.

Le procédé choisi pour ce développement est une station sur un module de l'atelier de ferrage de véhicules Citroën AX et Peugeot 106. Cette station réalise les premiers points de soudure après positionnement et serrage du bloc avant du véhicule avec le plancher ou "soubassement". Elle est composée de deux outils permettant la saisie des blocs à assembler et de deux robots réalisant la soudure. Le but de cette opération, appelée "conformation", est la mise en référence correcte des deux éléments avant qu'ils soient traités par les autres stations de la ligne d'assemblage.

La durée moyenne du cycle est d'environ 40 s, durant lesquelles il y a environ une centaine de changements d'états de variables (parmi les 600 variables environ observées). En combinant les différentes options possibles (AX/106, direction gauche/droite, avec/sans toit ouvrant, 3/5 portes, avec/sans cale hydraulique, essence/diesel, avec/sans cycle de rodage des électrodes de soudure, ...), on obtient environ 80 types de cycles différents. Les données disponibles sont une liste de changements de variables horodatées (on voit un extrait d'une de ces listes à la figure 9.15.a).

Initialement, nous pensions que le problème majeur pour l'analyse d'un tel procédé était le grand volume de données générées à chaque cycle.

En réalité, les problèmes rencontrés ne se réduisent pas "simplement" à cela. Le problème majeur se situe au niveau du codage de l'information à fournir au réseau de neurones. Cet aspect est en effet très délicat en raison de la nature discontinue du procédé. Dans les procédés continus, la tâche à réaliser consiste à surveiller des variables continues en fonction du temps. On peut alors regrouper simplement les différentes variables dans un vecteur et analyser le nuage correspondant, par exemple à l'aide des méthodes décrites jusqu'ici.

Tout d'abord, nous avons pensé fabriquer un vecteur continu, représentatif d'un cycle complet, en assignant à chaque composante l'instant d'arrivée du changement d'état d'une variable donnée (par exemple, on peut ainsi décider que la 1ère composante code l'instant du premier passage à 1 du signal 'E784' depuis le début du cycle, que la 2e code celui du premier passage à 0 de la même variable, etc.). On s'est alors heurté à plusieurs problèmes relatifs à la nature irrégulière et polymorphe des cycles. Par exemple, il s'est avéré impossible de définir un début et une fin de cycle entre lesquels on pouvait voir toujours le même nombre de changements d'états. De plus, il est fréquent de rencontrer certaines variables



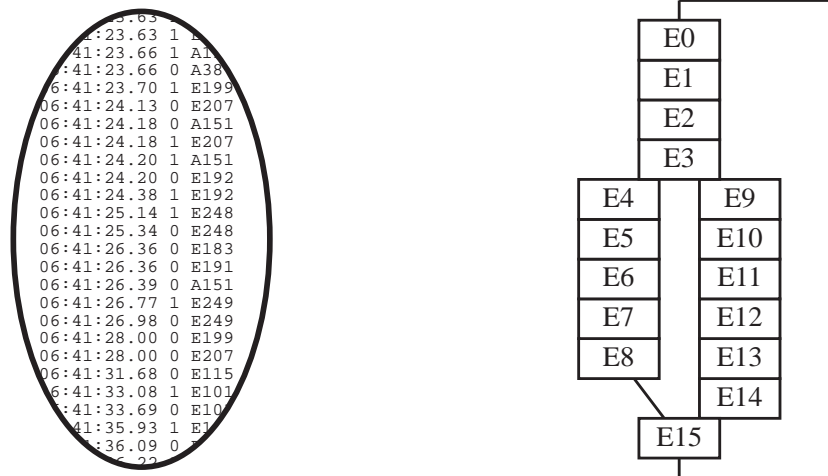


Figure 9.15: Exemples de programmes d’automates. A gauche, les données collectées sur l’automate réel. A droite, un automate très simple, permettant de faire des tests (voir texte).

qui changent  $K$  fois d’état au cours d’un type de cycle donné, et  $K' \neq K$  fois au cours d’un autre type de cycle. En outre, certaines variables plus “statiques” (qui sont des indicateurs de mode, par exemple) peuvent se combiner en un grand nombre d’états possibles différents. Comme elles changent rarement, elles risquent de ne pas être prises en compte dans le codage ci-dessus. Une identification de ces variables s’avère difficile.

Enfin, nous avons pris conscience, au fil de l’analyse, d’un certain nombre d’autres problèmes délicats, par exemple :

- On ne dispose pas de bases de données représentatives de cycles “bons” ou de cycles “mauvais” (ou encore, de cycles “pré-fautifs”), mais d’un long historique de changements d’états de variables isolées, dans lequel tout le problème est d’ancrer une telle symbolique.
- Les conditions de pannes elles-mêmes ne sont pas clairement définies.
- On remarque bien des plages de temps sans aucun changement d’état, ou avec un nombre de changement d’états très faible, ce qui signifie que le procédé a été stoppé. Cela ne signifie pourtant pas nécessairement qu’on soit en situation de panne. En effet, il peut se produire des attentes dues aux autres stations sur la ligne, par exemple, ou à l’ouverture d’un portillon de visite par un opérateur, ou encore à toute autre interruption extérieure et impondérable.

- Certains arrêts proviennent parfois aussi d'un défaut de positionnement de forme du bloc, panne imputable à une station en amont, c'est-à-dire en dehors de notre champ d'observation.
- Un cycle n'est, comme on l'a vu, pas une séquence immuable. Tout au contraire, il faut le considérer comme le parcours dans un graphe d'états. De par la nature du procédé, il arrive que certaines séquences d'opérations soient lancées en parallèle, générant ainsi des "peignes" de changements d'états entremêlés.
- Des trois conditions qui précèdent, il découle que certains écarts entre variables vont être grands, avec une forte variance, bien que sans importance. A l'inverse, des écarts de temps courts entre deux signaux peuvent être très importants. Une analyse de l'espace à base de quantification vectorielle risque donc de "gaspiller" un nombre important de neurones dans des régions inintéressantes, et, réciproquement, de ne pas quantifier certains intervalles vitaux, bien que petits.

Finalement donc, ce problème de codage dans le cas réel semble difficile à résoudre. Le schéma imaginé initialement était l'utilisation d'un réseau de neurones de type VQP ou SOM comme prétraitement pour des méthodes d'intelligence artificielle faisant l'interface avec l'utilisateur. Compte tenu de l'utilisation toujours plus importante de ces dernières méthodes pour préparer le problème en vue de l'application d'un réseau de neurones, ce schéma risque de s'inverser.

Afin de valider les possibilités d'analyse d'un automate par VQP, sans se laisser arrêter par les difficultés rencontrées dans cette application de grande complexité, nous avons testé l'algorithme sur des séquences simples, comprenant toutefois plusieurs des difficultés rencontrées jusqu'à présent dans l'application réelle.

A titre d'illustration, la figure 9.15.b montre la définition d'un programme automate fictif. Les changements d'états se suivent jusqu'à **E3**, après quoi les séquences de gauche ou de droite sont empruntées au hasard, selon une répartition de probabilité fixée *a priori*. Entre chaque succession d'états, il s'écoule un temps tiré aléatoirement dans une fourchette différente pour chaque état, et également fixée *a priori*.

Le codage se fait de la façon suivante :  $k$  composantes binaires servent à définir l'état dans lequel se trouve l'automate. On ajoute encore à ce vecteur d'état 2 composantes supplémentaires qui codent le temps sous forme circulaire :  $a[\cos(\omega t), \sin(\omega t)]^T$ , avec  $\omega = 2\pi/T$  et  $T$  la période moyenne du cycle. Le facteur  $a$  sert à donner plus ou moins d'importance à la dimension temporelle du codage par rapport à l'état binaire des variables. Ce codage sous forme circulaire permet d'indiquer la nature répétitive des cycles.

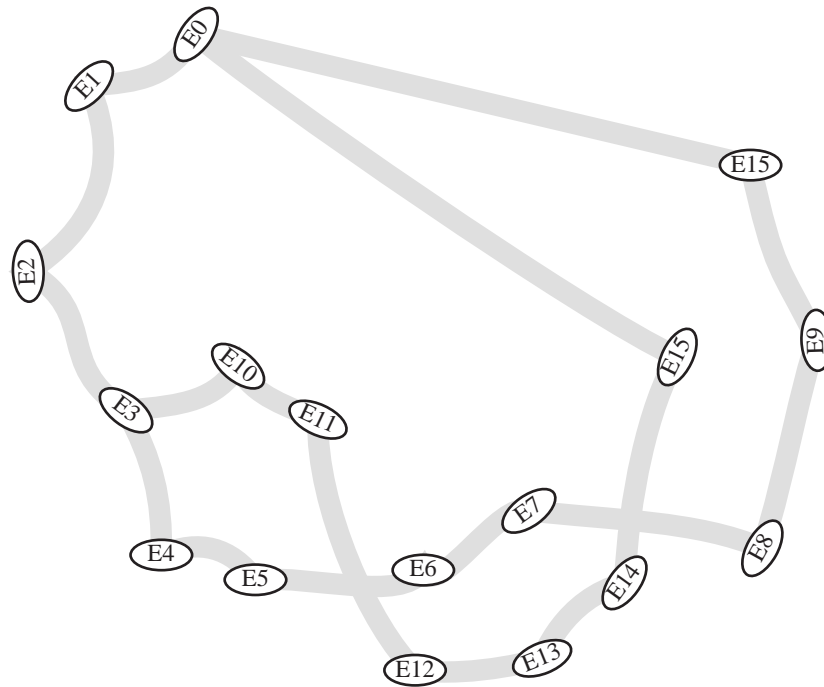


Figure 9.16: Résultat de VQP sur le problème de la figure 9.15.b, avec un espace de sortie fixé à 3 dimensions.

La figure 9.16 donne le résultat d'une représentation de la séquence d'exemple par un réseau VQP dont la sortie est fixée à trois dimensions. L'idée est d'obtenir une représentation sur un cylindre, avec la position angulaire sur le cylindre dépendant du temps, et avec la composante le long de l'axe du cylindre dépendant de la distance de Hamming entre deux états binaires. Dans la projection trouvée par VQP et illustrée à la figure 9.16, on retrouve bien cette structure, avec une bonne séparation des séquences alternatives le long de l'axe du cylindre. On retrouve la variation du délai entre deux événements successifs le long de la circonférence du cylindre. Après apprentissage, lorsqu'on fait défiler des cycles, on observe leur déroulement dans l'espace cylindrique de sortie. Les dérives temporelles sont aisément repérées sur le cylindre : par exemple, un intervalle de temps entre deux événements particuliers qui a tendance à s'allonger va se traduire par le passage du point de sortie à une position toujours plus éloignée de sa position apprise. On arrive ainsi à détecter le défaut, ainsi que l'événement concerné, ce qui peut permettre une localisation de ce défaut dans le procédé.

### 9.3.2 Surveillance de centrale nucléaire

Nous présentons ici des résultats obtenus dans le cadre d'un projet de collaboration avec le Laboratoire de Réseaux Electriques de l'École Polytechnique Fédérale de Lausanne (LRE-EPFL) et une grande société belge spécialisée dans les problèmes de l'énergie, la société Tractebel.

L'étude porte sur le problème de surveillance de centrale nucléaire. Les données sont issues d'un simulateur d'une unité de centrale électrique nucléaire située en Belgique. Cette unité est du type réacteur à eau pressurisée, à 3 boucles primaires. Chaque boucle possède son générateur de vapeur (GV) ainsi que sa pompe primaire. La pompe primaire a pour rôle de faire circuler le fluide au sein de la boucle. Le GV fournit la chaleur produite par le circuit primaire au circuit secondaire.

La section de boucle située entre la sortie du réacteur et l'entrée du GV est appelée "branche chaude". Celle du GV vers le réacteur est appelée "branche froide".

Le circuit primaire possède un pressuriseur, connecté à la branche chaude d'une des boucles primaires. Le pressuriseur est l'organe qui permet, par adjonction d'eau froide ou par réchauffement de l'eau qui s'y trouve, de réguler la pression du circuit primaire.

L'échange de chaleur entre le circuit primaire et secondaire se fait dans le GV. Il produit de la vapeur dans le secondaire. Cette vapeur entraîne une turbine couplée à un alternateur pour produire l'électricité.

Le système est observé à l'aide de 42 variables, qui sont :

- les températures de branches froides et chaudes de chaque boucle, ainsi que celles à l'entrée des GV, côté secondaire,
- les débits d'eau dans chaque boucle, et les débits dans les GV, côté secondaire,
- les niveaux d'eau dans les GV, côté secondaire,
- la pression de vapeur dans chaque GV,
- la pression de vapeur après le 6e étage de la turbine, et celle dans le pressuriseur,
- des variables caractérisant le flux neutronique dans le réacteur,
- des débits de fuites (vapeur, eau, totale),
- la vitesse des 3 pompes,

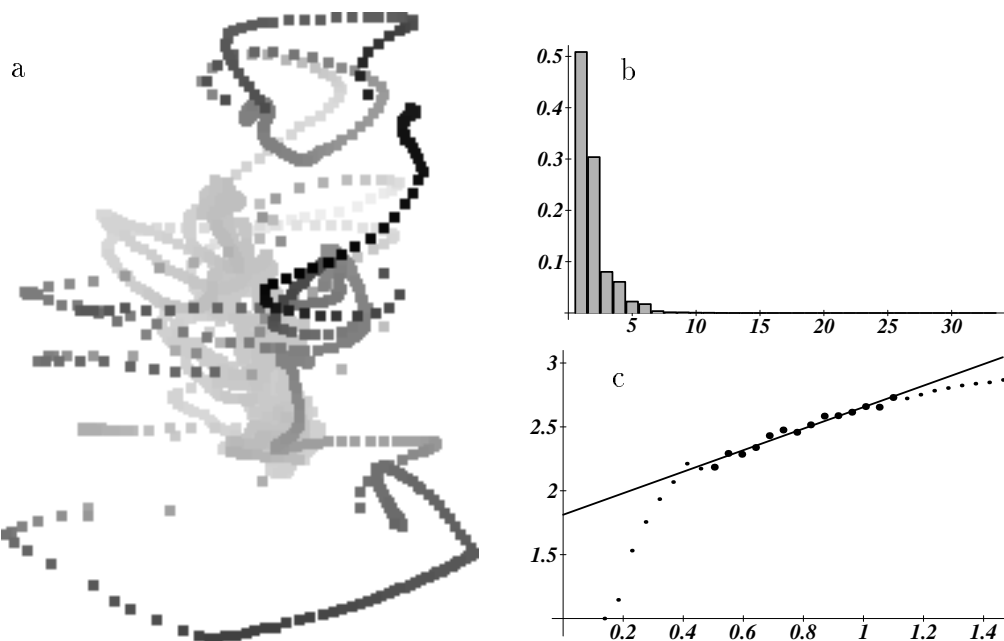


Figure 9.17: Analyse préliminaire des données récoltées sur le réacteur (simulé) en bon fonctionnement : (a) vue directe des 42 dimensions (par le “viewer”), (b) histogramme des valeurs propres, (c) analyse de dimension fractale.

- la position (binaire) de 4 vannes,
- un débit de charge du circuit,
- le niveau du réservoir

Les données correspondant à 5 situations nous ont été mises à disposition par Tractebel. Quatre d’entre elles (que nous nommons E,G,H et I) sont des situations normales, correspondant à l’exploitation sous différentes conditions de la centrale. Dans la 5e par contre (que nous appelons J), il a été simulé un accident consistant en une fuite de liquide réfrigérant (brèche dans la branche chaude d’une boucle primaire). Le simulateur rend compte des actions automatiques qui seraient prises en centrale pour compenser la fuite, notamment l’usage d’une réserve d’eau qui intervient pour compenser toute perte de réfrigérant au niveau primaire. L’incident est initié 12 minutes après le début de la simulation. Il est repéré par le système de supervision de la centrale 3’30” (c’est-à-dire 210 secondes) après son apparition. On va voir comment une utilisation adéquate de VQP va réduire ce temps à 5 secondes seulement.

Une première analyse des données en 42 dimensions (figure 9.17) indique que l’on peut réduire la dimension par projection linéaire (par ACP) jusqu’à 10. Avec ces 10 axes principaux, on rend compte de 99.8% de la variance totale du nuage

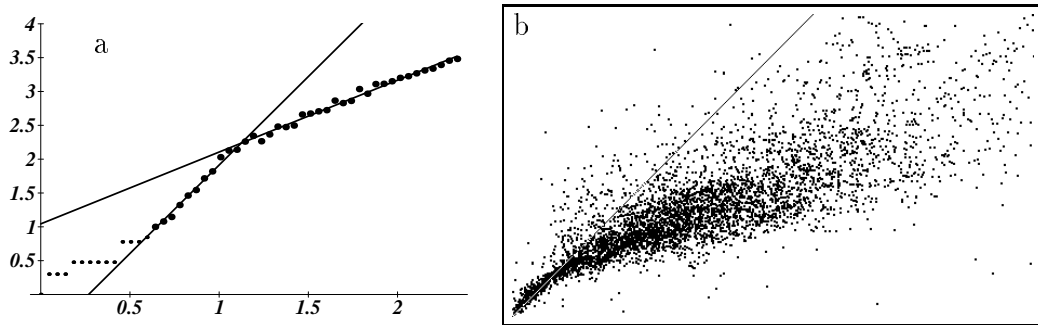


Figure 9.18: (a) Analyse de dimension fractale de l'espace des phases. (b) Représentation  $dy - dx$  d'un VQP à 200 neurones avec l'espace des phases en entrée et un espace de sortie fixé à deux dimensions.

en 42 dimensions. Une analyse de dimension fractale montre que la dimension intrinsèque est proche de 1. Cette analyse est confirmée par la vue directe du nuage en 42 dimensions : la distribution semble avoir une structure “fil de fer”. Cette structure (qui contient des boucles) est compréhensible étant donné qu'elle correspond à l'espace d'état, au cours du temps, d'un processus *régulé*. Ainsi, le système en boucle ouverte aurait tendance à diverger (c'est le départ des boucles), mais le régulateur le ramène vers son point de fonctionnement idéal.

Pour cette raison, il nous a semblé intéressant de considérer l'espace des phases :  $\begin{bmatrix} \xi(t) \\ \dot{\xi}(t) \end{bmatrix} \cong \begin{bmatrix} \xi(t) \\ \frac{1}{\Delta t}(\xi(t + \Delta t) - \xi(t)) \end{bmatrix}$  centré-réduit comme espace d'entrée de VQP, en prenant pour  $\xi$  les vecteurs après ACP. Une analyse fractale de ce nouveau nuage à 20 dimensions (figure 9.18.a) montre *deux zones* où l'estimation de la dimension est stable : à une première échelle, relativement grossière (de  $r_0/3$  à  $r_0/16$  environ, où  $r_0$  est le grand côté de la boîte qui englobe tout le nuage), on trouve une dimension de 2.6, alors que pour une échelle plus fine (de  $r_0/16$  à  $r_0/300$  environ), on retrouve une dimension de 1. Cela peut s'expliquer en voyant que, dans l'espace des phases, la trajectoire du système décrit une courbe dont les segments sont quasiment situés dans des plans.

Pour apprendre cette structure, nous avons donc utilisé un réseau VQP avec une sortie à deux dimensions, en prenant comme ensemble d'apprentissage l'espace des phases de l'ensemble des situations normales : E,G,H et I. On voit la représentation  $dy - dx$  en fin d'apprentissage à la figure 9.18.b. Après apprentissage, nous avons pratiqué la projection continue (selon § 7.3) de la base de données correspondant à la situation accidentelle (J, qui a subi le même prétraitement que les autres<sup>2</sup>). Pour chaque point, nous relevons la valeur  $\varepsilon$  qui est

<sup>2</sup>Il faut bien faire attention de ne pas centrer-réduire chaque nuage indépendamment. Les axes principaux et les variances des différents nuages sont en général différents, et l'on obtien-

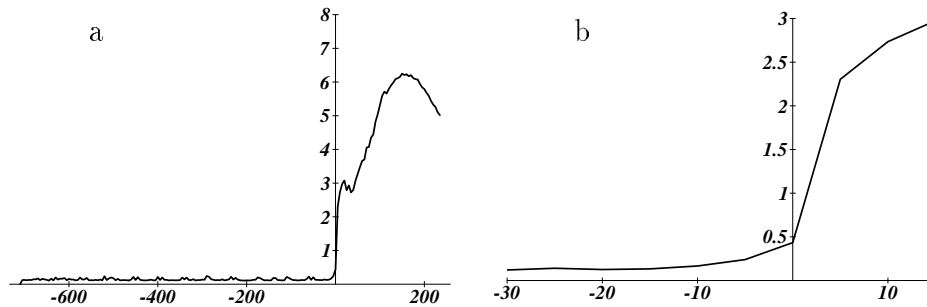


Figure 9.19: (a) Graphique de l'évolution, au cours du temps, de l'erreur de projection  $\varepsilon$  entre l'état du système et la variété apprise par VQP. On repère aisément l'incident (qu'on a placé au temps  $t = 0$  sur cette échelle en secondes). A droite (b), un agrandissement autour de cet instant montre que l'on peut détecter l'incident relativement tôt (les systèmes classiques le détectent après 210 s).

l'erreur entre  $\xi$  et sa projection sur la variété apprise (voir § 7.3). Cette valeur  $\varepsilon$  est tracée en fonction du temps à la figure 9.19, où l'on a placé l'origine du temps au moment où l'incident a été initié (12 minutes après le début de la simulation). La partie où le temps est négatif correspond donc à la présentation d'états qui n'ont jamais été appris, mais où la situation est encore normale. On voit que l'erreur de projection est faible dans cette partie, ce qui signifie que l'état est proche de la surface (pliée) des états appris (mais pas forcément proche des états appris eux-mêmes). La partie droite correspond à ce qui se passe après l'instant où l'incident a commencé. On constate qu'il est facile de discriminer les deux situations, dans un temps relativement court après le début de l'incident (pratiquement au premier échantillon qui suit, soit 5 secondes après l'incident).

Par comparaison, sur cette échelle de temps en secondes, le superviseur de la centrale détecte le défaut après 210 s, soit tout à droite du graphique 9.19.a.

## 9.4 Fabrication de métrique

Dans les applications qui précèdent, on a toujours considéré que l'on disposait de données vectorielles dans un espace euclidien.

Cet espace, de dimension  $n$ , est quantifié par la première couche de VQP, c'est-à-dire que les vecteurs-poids de cette couche sont un échantillonnage  $n$ -dimensionnel non régulier de la distribution. En parallèle, la seconde couche tente

---

drait ainsi des rotations et des facteurs d'échelle différents de cas en cas. La procédure correcte consiste à faire l'ACP d'un nuage, par exemple celui de la situation E, puis de retenir l'application linéaire  $P$  qui permet de passer de E dans son espace principal après centrage-réduction. Ensuite, on applique  $P$  sur tous les autres nuages.

en permanence de copier avec un minimum de distorsion les poids de la première couche dans un espace de sortie de dimension réduite  $p < n$ .

Le critère de distorsion n'est pas fondé sur la variance du nuage, comme c'est le cas pour l'ACP, mais est défini par l'erreur quadratique sur le respect des distances entre les points. En mettant une pondération plus importante pour les petites distances, la minimisation du critère conduit au "dépliage" de la variété moyenne représentée par les poids d'entrée.

Il faut remarquer que la recherche des positions des vecteurs-poids de sortie se base donc uniquement sur une liste des distances mesurées dans l'espace d'entrée.

Cette dernière observation permet de considérer des problèmes pour lesquels *on ne dispose pas* d'une distribution de points dans un espace euclidien, mais uniquement d'une liste de distances entre des individus.

Dans les deux sections qui suivent, nous montrons comment VQP permet, dans ces conditions, de *construire* une représentation euclidienne.

### 9.4.1 Routage adaptatif de paquets en télécommunications

Nous rapportons ici les résultats d'une étude préliminaire pour un projet que nous venons de commencer, en collaboration avec le CNET et France Telecom.

Les réseaux de transmission de données en mode asynchrone (ATM) doivent permettre les plus hauts débits d'information, en s'affranchissant des resynchronisations d'horloge et en utilisant le support optique. Si l'aspect physique de la commutation est relativement bien défini, il n'en va pas de même pour l'aspect "décision de routage" [3, 91]. En effet, dans ce cas, un message (paquet) peut difficilement être stocké, il doit être routé vers (ou dérouté de) sa destination en quelques nanosecondes. Au mieux, dans certains cas, peut-on décider de le retarder (200 m de fibre optique pour 1 ms) mais ceci doit être exceptionnel. A un nœud de réseau, la décision de routage (pour choisir le meilleur chemin) doit être prise en un temps extrêmement réduit en fonction des occupations de lignes et des nœuds à traverser pour atteindre le destinataire.

Sur le plan stratégique, le routage adaptatif permet d'optimiser l'utilisation des ressources afin d'augmenter les débits, de limiter la congestion et de réduire le coût financier des communications. Diverses procédures de routage adaptatif sont exploitées dans des réseaux existants (ETHERNET, ARPANET, TRANSPAC...), mais la complexité croissante de ces réseaux et l'augmentation des débits liés à la technologie impliquent une remise en cause des protocoles de routage conventionnels [37], peu efficaces dans la résolution des problèmes à combinatoire importante. De nouvelles techniques (par exemple celles décrites dans [3, 91]) doivent être définies en fonction de l'aspect temps réel, qui est ici primordial.



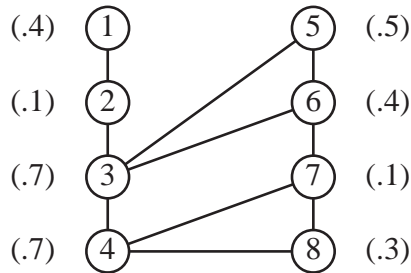


Figure 9.20: Exemple d'un graphe de réseau affecté des coûts (entre parenthèses) d'utilisation des nœuds. La somme des coûts le long d'un trajet définit le coût total de ce dernier, donc la "distance" équivalente du destinataire.

Il est bien évident que le routage adaptatif doit se faire en fonction d'informations sur la structure et l'état du réseau, ce dernier pouvant évoluer au cours du temps. Grandes dimensions des réseaux et hauts débits d'information interdisent toute décision centralisée pour le routage : le processus de décision doit être local, au niveau du commutateur. Cependant, le principe de routage adaptatif optimisé implique une connaissance globale de l'état de l'ensemble du réseau. Ces deux considérations ne peuvent être rendues compatibles qu'en délocalisant les processus d'optimisation au niveau de chaque nœud. La prise de décision doit être rapide, sans calcul, ou selon un calcul strictement parallèle. Elle doit donc utiliser des informations "prédigérées" élaborées par un processeur local. L'élaboration de ces informations doit répondre à un processus d'optimisation : chaque nœud se construisant une "vision locale" de l'ensemble du réseau, qui peut être différente d'un nœud à l'autre. De plus, cette vision locale doit évoluer au cours du temps, en fonction d'informations reçues des nœuds voisins et de l'ensemble du réseau, et remises à jour périodiquement.

L'idée de base est de construire une telle vision locale à l'aide de VQP, en la réactualisant de temps en temps en fonction des informations reçues par les nœuds voisins sur l'état du réseau. Le processus de routage lui-même, à une échelle de temps beaucoup plus fine, utilise cette vision locale pour trouver dans un temps minimal une route optimale par rapport à la vision locale du nœud, c'est-à-dire presque optimale au niveau global si "vision locale" et "état véritable" du réseau ne diffèrent pas trop.

Le fait que les informations prises en compte pour la définition des distances en entrée puissent être de nature différente (distance physique entre deux nœuds, ou logique, attachée au coût d'utilisation d'une liaison ou d'un nœud, ou même floue) autorise le maximum de liberté. Ces distances correspondent aux "coûts" utilisés dans les algorithmes de recherche de parcours dans un graphe.

Le choix d'un espace d'arrivée euclidien (produit scalaire induisant une norme) permet d'utiliser la comparaison de produits scalaires à la place des comparaisons de distances elles-mêmes, ce qui est plus efficace du point de vue de la réalisation matérielle. Afin de fixer les idées, nous donnons, sur un exemple simple que nous avons simulé, la procédure qui permet de trouver la direction dans laquelle envoyer

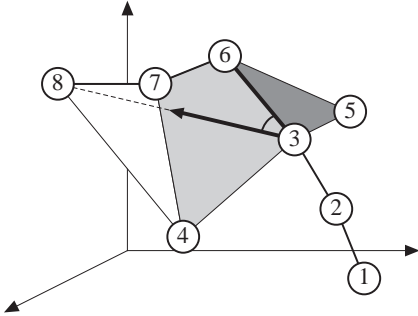


Figure 9.21: Vue en perspective de l'espace 3D de projection construit par le nœud 3. Un message destiné à 8 génère un vecteur qui pointe vers le prototype 8. Son angle minimal avec les directions issues de 3 détermine le chemin 3-6 à emprunter. Le nœud 4, très chargé (0.7), est ainsi évité.

un message à partir d'un nœud, en fonction de l'adresse de destination et des contraintes du trafic dans le réseau. Nous supposons un réseau dont le graphe est celui de la figure 9.20. Des coûts sont affectés à chaque branche et à l'utilisation de chaque nœud en fonction du trafic moyen évalué périodiquement, selon par exemple les recommandations de l'ISO.

Dans un premier temps, puis de temps en temps (par exemple en profitant des périodes d'inactivité d'une branche), tous les nœuds émettent, à destination des autres, des informations sur leur charge moyenne et leur activité. Chaque nœud dispose donc à chaque instant des données nécessaires à l'évaluation des distances entre lui-même et ses voisins ainsi qu'entre les autres nœuds, ceci en fonction du graphe et des coûts liés aux différents chemins possibles. Il établit alors par VQP, dans un espace qui lui est propre, une cartographie de l'ensemble du réseau vu de sa propre position. Cette cartographie est périodiquement remise à jour. Compte tenu de la non-symétrie des chemins et des occupations, il est évident que les nœuds possèdent des représentations du réseau différentes les unes des autres.

Prenons par exemple un message issu du nœud 1 qui possède comme adresse de destination celle du nœud 8. Il doit traverser le réseau selon le chemin de plus faible coût, sachant que la décision de routage appartient à chaque nœud et doit être prise quasi instantanément en fonction du choix de la voie optimale à emprunter et des contraintes locales d'encombrement (voie déjà occupée par exemple). Au début, le routage est unique, et ce n'est qu'à partir du nœud 3 que les problèmes se posent. En raison de ce qui précède, le nœud 3 a établi, à partir des données de la figure 9.20 (structure du graphe et coûts associés), son espace de représentation du réseau. Pour la représentation, nous avons choisi ici un espace en trois dimensions, mais rien n'empêche d'utiliser des dimensions supérieures. Cet espace de représentation est illustré à la figure 9.21. Le message porte l'adresse du destinataire (8). Nous définissons alors le chemin à emprunter comme celui qui est le plus proche de la direction du destinataire. La recherche du voisin à qui il faut envoyer le paquet se fait simplement en prenant le maximum des produits scalaires entre le vecteur directeur  $\mathbf{u}_{38}$  et les  $\mathbf{u}_{3i}$  (pour tous les  $i$  voisins du nœud 3). Cette procédure s'avère plus rapide et plus simple que

l'algorithme classique de Dijkstra [37], qui ne permet que de déterminer l'arbre des chemins les plus courts depuis un nœud, en éliminant les alternatives possibles, et qu'il faut donc re-calculer entièrement chaque fois que les coûts changent. Notre système permet de pondérer ponctuellement les produits scalaires par la disponibilité instantanée des voisins, sans changer toute la représentation locale. De plus, la réactualisation de cette représentation locale n'est en général qu'une petite modification des différents points. Il n'est pas nécessaire de refaire toute une convergence de l'algorithme VQP.

Enfin, au niveau de la couche de routage elle-même, le produit scalaire lui-même peut être effectué de façon optique et parallèle.

### 9.4.2 Cartographie de concepts en fonction de distances subjectives

Cette notion de représentation d'un ensemble de points dans un espace euclidien en fonction d'une liste de distances peut être étendue pour d'autres types d'analyse. Notamment, on peut considérer des relations entre individus qui ne jouissent pas des propriétés d'une distance (c'est-à-dire qui ne respectent ni la réciprocité, ni l'inégalité du triangle) mais qui dénotent tout de même une notion d'éloignement.

Si ces propriétés ne sont pas respectées, on pourra prendre une dimension de sortie aussi grande que l'on voudra, on n'arrivera pas à annuler le critère de distorsion (6.1). Néanmoins, sans l'annuler, il est tout de même possible de le minimiser et de produire par là une représentation euclidienne (munie d'une métrique) approximative des individus.

Par exemple, on peut dresser une liste d'objets plus ou moins hétéroclites (pomme, chat, avion, parapluie, cendrier, cigarette, ...), puis faire une enquête auprès d'un certain nombre de personnes pour connaître la "distance subjective" qu'ils attribuent à quelques couples de ces objets dans un champ sémantique particulier (fonction, apparence, co-occurrence ou encore proximité spatiale moyenne). Une auto-organisation du réseau VQP à partir de cet ensemble de distances fournit une carte des objets dans laquelle la métrique correspond au champ sémantique choisi (par exemple, si celui-ci est "la fonction", il y a fort à parier que la cigarette ne sera pas représentée loin du cendrier). Lorsqu'on présente de nouveaux objets (toujours définis par leurs "distances subjectives" à quelques objets déjà connus) notre algorithme de projection continue permet de les représenter dans l'espace des connaissances et, ainsi, de faire des associations à partir des proximités dans cette projection.

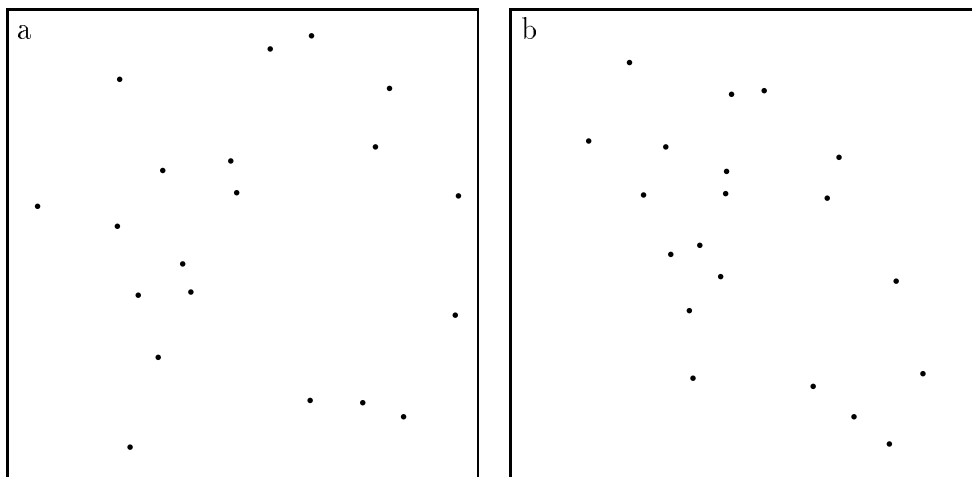


Figure 9.22: Deux ensembles de points à appairer. L'ensemble de points (b) a été généré par rotation, déformation et bruitage de l'autre (a).

## 9.5 Appariement de graphes

Pour terminer cette liste non exhaustive d'exemples d'applications de VQP, nous allons étudier le problème d'appariement de graphes.

Ce problème d'optimisation consiste à chercher la mise en relation des nœuds de deux graphes, en minimisant une fonction objective. C'est un problème qui apparaît notamment en reconnaissance des formes, où les graphes sont alors remplacés par des ensembles de points caractéristiques des objets, et où la tâche consiste à chercher une mise en correspondance de deux de ces ensembles qui minimise une fonction généralement choisie invariante en translation, en rotation, en homothétie et en rotation.

Pour résoudre ce problème, nous proposons l'utilisation d'un algorithme à mi-chemin entre VQP et les cartes de Kohonen. Il s'agit d'une sorte de carte auto-organisante dont les neurones, au lieu de se trouver sur une grille, sont fixés aux positions de l'un des deux ensembles de points. L'algorithme lui-même est identique à celui de Kohonen, mais en utilisant une fonction de voisinage définie sur les positions des neurones ainsi fixées.

Nous pouvons illustrer cet algorithme avec l'exemple de la figure 9.22. L'ensemble de points de droite a été généré par rotation, déformation et bruitage de l'autre.

A la figure 9.23, on montre le résultat après convergence de l'algorithme. Les vecteurs d'entrée  $\mathbf{x}_i$  se sont positionnés sur les points de la distribution, et l'appariement est donné par la relation habituelle  $\mathbf{x}_i \longleftrightarrow \mathbf{y}_i$ . Afin de visualiser cette relation, nous avons utilisé l'algorithme des “liens dynamiques” qui montre

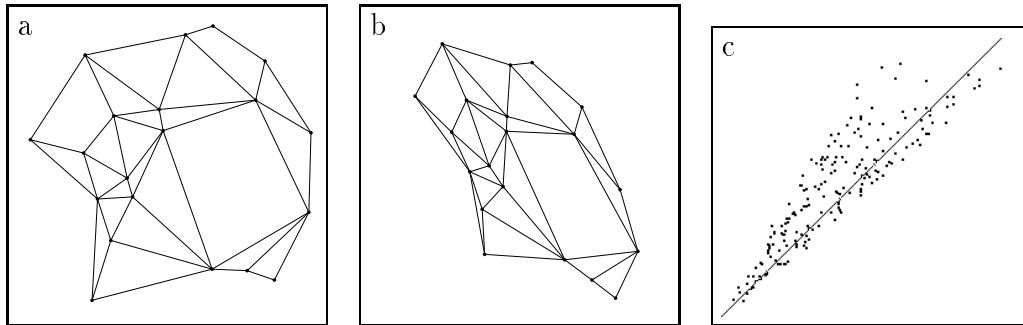


Figure 9.23: Résultat de l'algorithme proposé pour l'appariement de graphes dans le problème de la figure 9.22. (a) Espace d'entrée, avec les vecteurs ayant convergé vers la distribution. (b) Espace de sortie, où les poids ont été fixés sur les points du nuage à apparier.

les plus proches voisins dans l'espace d'entrée. La représentation  $dy - dx$  montre également que l'appariement des distances est bon.

Sur la base de l'appariement de ces points caractéristiques, on peut utiliser notre algorithme de projection continue pour obtenir la représentation de n'importe quel autre point de l'objet, voire de tout l'objet lui-même.



# Chapitre 10

## Conclusion

### 10.1 Synthèse

Le noyau de ce travail sur l'analyse de données par réseaux de neurones auto-organisants est l'algorithme que nous avons appelé "*Vector Quantization and Projection*" (VQP). Cet algorithme permet de trouver le support de la distribution de données et sa représentation par dépliage dans un espace de dimension réduite.

Contrairement à une conventionnelle analyse en composante principale (ACP), il n'est pas limité à la recherche d'une structure linéaire, mais peut révéler une classe beaucoup plus grande d'objets : toute structure "pliée". On pourrait ainsi le renommer "Analyse en Composantes Curvilignes" (ACC).

Il se démarque de l'algorithme des cartes de Kohonen, dont il est pourtant inspiré et qui peut également effectuer un tel dépliage. L'algorithme de Kohonen utilise comme espace de projection une grille que l'on définit au préalable. Celle-ci est de forme généralement rectangulaire ou hexagonale, et même souvent (faute d'information sur la forme qui serait la plus adéquate) carrée. Ainsi, c'est au mécanisme de projection de s'arranger pour représenter les données sur cette grille de forme figée. On conçoit facilement que dans un cas général cette méthode peut conduire à des contraintes contradictoires. Avec VQP au contraire, on ne contraint pas l'espace de représentation à prendre une forme particulière. L'arrangement des neurones est sans importance. C'est leur sortie, vectorielle, qui définit la représentation dans un espace continu. Nous montrons que cette libération de la contrainte d'une forme figée permet d'aborder des structures de données beaucoup plus variées : de forme quelconque, connexe ou non.

Nous devons cependant beaucoup à l'étude du réseau de Kohonen. Par exemple, la mesure de pertinence d'une représentation que nous avons appelée "*dy - dx*" (voir § 4.5.3) et que nous utilisons pour VQP a été développée initialement pour qualifier l'état d'organisation d'une carte de Kohonen. De même, c'est en cher-

chant à comprendre pourquoi la plupart des autres mesures d'organisation donnaient (à tort bien sûr) leur meilleure note à l'état qui résulte immédiatement de l'initialisation que nous avons constaté des propriétés étranges des vecteurs aléatoires de grandes dimensions, en particulier le fait que leur longueur a un écart-type négligeable par rapport à la moyenne quand le nombre de composantes indépendantes est grand.

VQP ne résout pas l'un des problèmes posés lors de l'utilisation de l'algorithme de Kohonen, et qui est d'ailleurs présent dans toutes les formes d'analyse de données : quelle dimension donner à la représentation ? Pour esquisser une réponse, nous avons proposé d'utiliser la dimension fractale du nuage de données. Comme généralement on ne dispose pas d'une distribution continue, mais d'un ensemble fini d'échantillons, on est obligé de s'écarter de la définition mathématique de la dimension fractale qui est une limite. Partant, on soulève l'épineux problème de l'échelle d'observation : si l'on regarde de trop loin, le nuage ressemble à un point, et si l'on regarde de trop près, on voit que le nuage est fait de beaucoup de petits points... Entre ces deux extrêmes se trouve généralement une échelle d'observation appropriée, où la dimension mesurée est stable (la caractéristique suit un segment de droite à cet endroit). Il peut arriver cependant que le nombre de points de la base de données soit insuffisant. Les deux extrêmes sont alors trop proches et l'on n'observe pas de zone où la dimension soit stable. Il peut aussi arriver que la dimension change selon l'endroit de l'espace que l'on considère. La nature globale de l'estimation de la dimension fractale empêche de détecter ce phénomène.

Dans tous les cas, la valeur trouvée par l'analyse de la dimension fractale ne doit être considérée que comme un point de départ pour la dimension à donner à l'espace de sortie de VQP. Au vu de la représentation  $dy - dx$ , on peut décider d'augmenter ou de diminuer cette dimension d'une unité et de recommencer une phase d'apprentissage. Par exemple, si la caractéristique  $dy - dx$  est une droite parfaite, cela signifie qu'il n'y a pas eu besoin de déplier : la dimension de sortie était suffisamment grande pour que la projection soit simplement linéaire. Dans ce cas, on peut essayer de diminuer la dimension de sortie. A l'inverse, une caractéristique  $dy - dx$  dispersée dès l'origine est révélatrice d'une dimension de sortie insuffisante, et l'on aura intérêt à l'augmenter. Il est intéressant de remarquer que la représentation  $dy - dx$  est en dernier ressort plus informative que l'analyse de dimension fractale. Notons d'ailleurs qu'elle peut être vue à tout moment pendant la convergence de VQP, ce qui est extrêmement utile pour le réglage des paramètres.

Un autre aspect de VQP est la quantification de l'espace d'entrée. En étudiant ce vaste chapitre qu'est la quantification vectorielle, nous avons été surpris de constater que la distorsion moyenne est le critère généralement admis sans autre



discussion. Ce critère conduit à une quantification qui respecte la densité des données, ce qui n'est pas toujours souhaitable, même dans la pure optique de la compression de signal. On peut très bien adopter d'autres critères, par exemple la minimisation de l'erreur maximale, comme nous l'avons signalé dans le cas de quantification de palette de couleur. Dans le cadre de VQP, où l'on s'intéresse au *support* des données et non à leur densité sur ce support, c'est d'une quantification de l'espace (et non d'une quantification de la densité) que nous avons besoin. C'est ainsi que nous avons développé l'algorithme "*Competitive Learning with Regularization*" (CLR) qui s'affranchit en partie de la notion de densité. Cette méthode est toutefois encore peu satisfaisante par certains aspects : elle est aussi lente que le simple *Competitive Learning* (c'est un algorithme "*winner-take-all*" et non "*winner-take-most*"). Elle tolère mal qu'on essaie de supprimer les unités mortes qu'elle peut avoir, par un mécanisme de fatigue par exemple, car dans ce cas les deux unités gagnantes ne sont pas forcément proches et cela perturbe l'algorithme des "liens dynamiques" (§ 3.7).

VQP supporte avantageusement la comparaison avec d'autres algorithmes de dépliage, comme le "*Non Linear Mapping*" (NLM) par exemple. Grâce à la forme très particulière de minimisation d'énergie que nous avons développée (qui n'est pas une simple descente de gradient mais plutôt la descente du gradient d'une partie seulement de l'énergie, différente à chaque itération), nous gagnons un ordre de grandeur en temps de calcul en fonction du nombre de neurones. Ainsi, alors que le NLM a une complexité en  $O(N^2)$ , VQP n'a qu'une complexité en  $O(N)$  pour un résultat équivalent. Il est donc beaucoup plus rapide en temps de calcul.

D'autre part, du point de vue purement fonctionnel, une pondération des termes d'énergie en fonction de la distance dans l'espace de sortie (et non de celle dans l'espace d'entrée comme dans le NLM) permet à VQP de représenter correctement des structures beaucoup plus fortement pliées, voire bouclées : pour représenter la surface de la terre sur une carte par exemple, il est nécessaire de "couper" la surface quelque part, à l'endroit qui correspond aux bords de la carte. Avec une pondération en fonction de la distance en sortie, une telle coupure est acceptée (la zone coupée n'introduit qu'une faible énergie puisque les points de part et d'autre sont éloignés dans l'espace de sortie).

Nous montrons également des propriétés particulières de VQP, comme la nature bijective de la relation mise en place entre les données et leur représentation cartographique, ainsi que les possibilités d'interpolation et d'extrapolation dans cette relation. D'un point de vue mathématique, on peut voir VQP comme un difféomorphisme entre le support (la "*variété*") des données et la carte. Ce difféomorphisme est bijectif, c'est-à-dire que l'on peut non seulement trouver l'image d'un point de l'entrée, mais aussi la préimage d'un point de l'espace de sortie.

Bien que, généralement, il y ait réduction de dimension entre l'entrée et la sortie, la préimage d'un point de la sortie est unique : elle se trouve strictement sur la variété des données, de dimension intrinsèque identique à l'espace de sortie. Par exemple, dans le cas de la cartographie du globe terrestre, la projection de la position d'un avion donne une longitude et une latitude sur la carte. A l'inverse, la préimage d'un point donné par sa longitude et sa latitude est unique sur la terre, mais on n'a pas pu reconstruire l'information "altitude". Une telle combinaison de projection et projection inverse donne la possibilité de calculer la distance entre un point inconnu et la variété de la distribution apprise. C'est un moyen de détecter une déviation par rapport à un phénomène appris précédemment (voir l'application de surveillance de centrale nucléaire, § 9.3.2).

Avec VQP, le difféomorphisme est en réalité quantifié par un certain nombre de points (le nombre de neurones dont on dispose). Pour rendre continue la projection, nous avons montré comment les mêmes principes que ceux de l'apprentissage proprement dit peuvent être utilisés pour pratiquer l'interpolation et l'extrapolation autour de ces points.

Finalement, nous mettons à l'épreuve la méthodologie et les algorithmes que nous avons définis avec plusieurs exemples pratiques (réels ou imaginés). La diversité de ces exemples, qui vont de la fusion de données auditives et visuelles au routage intelligent en télécommunications, en passant par la surveillance de centrale nucléaire, montre le potentiel élevé de cette méthode d'analyse de données.

## 10.2 Perspectives

L'algorithme VQP n'est bien sûr pas une conclusion. Bien au contraire, nous le percevons comme un commencement, une voie ouverte dans l'analyse de données. Outre les idées d'applications qu'il suscite et les améliorations dont il pourra faire l'objet (nous y reviendrons), il nécessite un important travail théorique : Peut-on démontrer sa convergence ? Comment régler correctement les différents paramètres ? Quelles sont les limites d'utilisation ?

- Les paramètres de VQP méritent une étude approfondie pour déterminer comment on doit les piloter au cours du temps et dans quelle fourchette de valeurs. Ceci est particulièrement vrai en ce qui concerne l'algorithme de projection continue où le nombre d'itérations doit être très restreint par souci d'efficacité. Peut-on se baser sur la représentation  $dy - dx$  pour commander automatiquement le rayon de voisinage  $\lambda$  et le gain d'adaptation  $\alpha$  ?
- Pour la projection continue (c'est-à-dire le mécanisme qui, après la phase d'apprentissage, permet de représenter n'importe quel point de la distri-

bution), peut-on développer une heuristique qui permette de sélectionner les points  $\mathbf{y}_i$  sur lesquels on va s'appuyer pour placer le point  $\mathbf{y}_0$  (qui est la projection cherchée) le plus efficacement possible ? Pour l'instant, faute de mieux, on procède soit par tirages aléatoires soit en les utilisant tous à chaque itération, mais lorsque le nombre de neurones est grand, on sait qu'un grand nombre d'entre eux aura une contribution minimale.

- Un problème plus ennuyeux est lié à la possibilité d'avoir des zones dans l'ensemble des données avec des dimensions intrinsèques différentes. On a vu que l'analyse de dimension fractale ne révélait pas ce phénomène. Peut-être l'emploi de méthodes locales, comme les liens dynamiques (§ 3.7), permettrait de détecter les dimensions de façon adéquate. Mais alors, comment représenter de telles données ? Quelle dimension donner à l'espace de sortie ? Cependant, dans la grande variété d'applications que nous avons rencontrées jusqu'à présent, ce problème ne s'est pas présenté.
- Un autre problème concerne les limites de VQP en matière de dépliage. Si le mécanisme fonctionne bien lorsque l'espace de départ n'est pas trop "chiffonné", il échoue pour une structure suffisamment complexe, comme une pelote de ficelle par exemple. La raison tient au fait que le nombre d'échantillons n'est peut-être pas suffisant et que la longueur dépliée d'une telle structure est très grande par rapport au diamètre de départ. Ainsi, les points devraient parcourir un chemin très long depuis leur position d'origine résultant de l'initialisation. Encore une fois, où se situe la limite ?
- Le dépliage effectué par VQP est basé sur une conservation de la notion de proximité la meilleure possible, c'est-à-dire que les distances (au moins les courtes) dans l'espace de sortie doivent être égales aux distances d'entrée. Cela introduit une certaine "rigidité" dans le dépliage (un peu comme lorsqu'on déplie une feuille de papier, par contraste avec une feuille de caoutchouc). Dans certaines applications, ou en présence de bruit dans l'espace d'entrée, une certaine souplesse serait nécessaire, y compris pour les courtes distances. Cette souplesse devrait pouvoir être apportée par l'emploi de coefficients de pondération locaux devant les  $Y_{ij}$  de la fonction d'énergie. Une autre approche est celle du "*non metric MDS*", où ce n'est pas une adéquation des *distances* qui est réalisée, mais une adéquation de l'*ordre* de ces distances, ce qui autorise ainsi une certaine déformation.
- Enfin, les cas que nous avons étudiés jusqu'à ce jour possèdent une dimension intrinsèque relativement petite (moins de 5), même si la dimension brute de l'espace d'entrée a pu être relativement grande (jusqu'au millier de dimensions). Comment se comporte VQP face à la tâche de déplier une

structure possédant un grand nombre de degrés de liberté et noyée dans un espace de dimension plus grande encore ?

- Un autre problème concerne le bruit dû aux composantes non significatives (par exemple, un nombre arbitraire de composantes aléatoires indépendantes), comme on en a rencontré dans l'application de surveillance de procédé (§ 9.3.1). Ces composantes sont fort gênantes parce qu'elles dispersent le nuage de données (la variété est étalée dans ces dimensions ajoutées). Les neurones sont gaspillés dans la quantification du nuage ainsi étalé. Surtout, la variété comporte autant de dimensions supplémentaires qui sont prises par VQP comme des degrés de liberté à révéler, ce qui est ennuyeux puisqu'ils ne sont pas informatifs. Comment identifier ces composantes inutiles pour les éliminer ?

Un autre point qui n'a été résolu pour l'instant que de façon empirique concerne la manière de combiner les opérations de la première et de la seconde couche. Faut-il laisser la quantification vectorielle et la projection fonctionner de concert, ou est-il préférable de laisser la première se terminer avant d'entamer la seconde ? Dans nos simulations, on a généralement opté pour le compromis suivant : laisser un temps d'avance à la première couche. Deux solutions me paraissent particulièrement dignes d'intérêt :

- Comme nous l'avons montré pour le “*Neural Gas*” (§ 5.2)), on peut relier les paramètres d'apprentissage à une estimation de la stationnarité de l'espace d'entrée, au lieu de le faire selon une fonction du temps prédéfinie. On obtient ainsi une convergence rapide et stable. Lorsque l'on change la distribution d'entrée, le réseau détecte automatiquement la chose et se remet à apprendre jusqu'à l'obtention d'un nouvel équilibre. Lorsqu'on subordonne les paramètres de la couche de sortie à cette même estimation (toujours mesurée dans l'espace d'entrée), on obtient un comportement similaire en sortie (l'apprentissage de la sortie s'active à nouveau en cas de modification de l'entrée). Ces expériences répondent partiellement au fameux dilemme de Grossberg de “stabilité/plasticité”, mais mériteraient d'être approfondies.
- Une voie encore plus radicale consisterait à considérer la quantification vectorielle et la projection d'une manière globale et couplée. Ce couplage n'est fait que dans un sens actuellement (le résultat de la quantification est recopié par la projection, mais la quantification est autonome). Les tentatives faites jusqu'à présent pour “fermer la boucle” VQ & P se sont soldées par un échec : tous les neurones convergent en un point (le centre de gravité du nuage). Même en ajoutant des contraintes de répulsion entre les vecteurs-poids (en entrée ou en sortie), on obtient des instabilités et une quantifi-

cation peu satisfaisante. Cela tient au fait que ce “rebouclage” à toujours été essayé avec des méthodes similaires à celle de Kohonen (avec l’espace de sortie utilisé à la place de la grille). Or on a vu (§ 8.2.3), lorsqu’on interprète l’algorithme de Kohonen comme une combinaison d’une quantification vectorielle et d’une contrainte de copie topologique de la grille, que ces deux parties sont instables si elles sont prises indépendamment, et que c’est l’aspect figé de la grille qui empêche la solution de dégénérer en un seul point. En revanche, si l’on normalisait la contrainte de copie topologique de l’espace de sortie par rapport à un changement d’échelle, on pourrait espérer obtenir une solution stable. C’est une possibilité attrayante que de partir d’une fonction d’énergie globale, et d’en tirer à la fois l’algorithme d’apprentissage pour l’entrée et la sortie. On obtiendrait ainsi une méthode de quantification vectorielle probablement plus efficace que celles qui existent actuellement, et directement pilotée par la détection de la variété, avec ce que cela apporte comme possibilité de généralisation et d’élimination de bruit.

D’un point de vue technique, il est relativement aisé de mettre en œuvre VQP dans un circuit VLSI. En effet, la quasi totalité de la charge de calcul concerne le calcul des distances  $X_{ij}$  et  $Y_{ij}$ . Avec une fonction de calcul de distance comme bloc de base, et en déléguant le calcul de la fonction non-linéaire  $F(Y_{ij})$  à l’extérieur du circuit, on peut facilement câbler les principales fonctions de VQP : apprentissage de la projection, projection continue et projection continue inverse. On peut même, moyennant l’ajout d’une structure de tri, effectuer la quantification vectorielle à l’aide d’un “*Neural Gas*”, qui lui aussi est essentiellement basé sur des distances. Cette mise sur circuit est envisageable à court ou moyen terme.

Enfin, d’un tout autre point de vue, il est intéressant de se pencher sur la plausibilité biologique du réseau VQP dans le cerveau des vertébrés. La découverte d’une équivalence structurelle me paraît peu probable, du moins si l’on espère une équivalence “neurone-à-transistor”. En revanche, je tiens pour sûr que la *fonction* de VQP est présente dans les processus cognitifs. Toutes les projections non-linéaires qui constituent en partie notre “traitement neuronal” ne s’expliquent pas par de seules cartes dont la forme est peu ou pas modifiable. Il paraît vraisemblable que les représentations de la connaissance et des expériences de la vie se font au moyen de quantifications et de dépliages de nos espaces perceptifs ou proprioceptifs de millions de dimensions. . .



# Bibliographie

- [1] A. Ahalt, A. K. Krishnamurthy, Chen P., and D. E. Melton. Competitive learning algorithms for vector quantization. *Neural Networks*, 3:277–290, 1990.
- [2] E. Alpaydin, P. Buzzi, F. Blayo, and P. Demartines. Libraries. Deliverables DS2-C1, Esprit Project BRA 3049 “NERVES”. Technical report, Swiss Federal Institute of Technology, LAMI-EPFL, INF-Ecublens, CH-1015 Lausanne, Switzerland, July 1991. about 150 pages.
- [3] E. Ayanoglu. Path enumeration and hot-potato routing analysis in multihop networks. *International journal of digital and analog communication systems*, 5:217–223, 1992.
- [4] H.B. Barlow. Unsupervised learning. *Neural Computation*, 1:295–311, 1989.
- [5] H.U. Bauer and K.R. Pawelzik. Quantifying the neighborhood preservation of self-organizing feature maps. *IEEE Transactions on Neural Networks*, 3:570–579, 1992.
- [6] T. Baumann, A. Germond, and D. Tschudi. Impulse test fault diagnosis on power transformations using Kohonen’s self-organizing neural network. In *Third Symposium on Expert Systems Application to Power Systems, Tokyo & Kobe*, April 1991.
- [7] F. Blayo, Y. Cheneval, P. Comon, O. Fambon, C. Jutten, P. Thissen, and M. Verleysen. Enhanced Learning for Evolutive Neural Architecture: Theory. Deliverables R1-A-P, Esprit Project BRA 6891 “ELENA”, June 1993. About 60 pages.
- [8] F. Blayo and P. Demartines. Data analysis: How to compare Kohonen neural networks to other techniques ? In A. Prieto, editor, *International Workshop on Artificial Neural Networks*, volume 540 of *Lecture Notes in Computer Science*, pages 469–476. Springer-Verlag, 1991.
- [9] F. Blayo and P. Demartines. Simulation de réseaux de neurones. In *Journées du Troisième Cycle Romand d’Informatiques*, Charmey, Switzerland, 1991. Invited presentation.
- [10] F. Blayo and P. Demartines. Algorithme de Kohonen: Application à des données économiques. In *Bulletin de l’ASE-SEV*, Zürich, March 1992.
- [11] F. Blayo and P. Demartines. Simulation de réseaux de neurones. In *Bulletin de l’ASE-SEV*, Zürich, March 1992.
- [12] C. Bouton and G. Pagès. Self-organization and convergence of the one-dimensional Kohonen algorithm. *Pre-Print de l’Université Paris I*, 1991.
- [13] V. Buzuloiu and D. Colţuc. Fast nonlinear algorithms for superresolution in geometric feature extraction. *ECCTD93 – Circuit Theory and Design*, pages 1441–1446, 1993.

- [14] J. D. Carroll and P. Arabie. Multidimensional scaling. *Annual Review of Psychology*, 31:607–649, 1980.
- [15] H. Caussinus. Un modèle pour fonder la recherche de projections révélatrices. In *XXIVes Journées de Statistiques*, pages 120–123, 1992.
- [16] D. Chaudhuri, C. A. Murthy, and B. B. Chaudhuri. A modified metric to compute distance. *Pattern Recognition*, 25(7):667–677, 1992.
- [17] Y. Cheneval, P. Demartines, and L. Tettoni. Function approximation using an architecture based on Kohonen self-organizing maps. In *Journées Neurosciences et Sciences de l'Ingénieur*, Oléron, May 1992.
- [18] V. Cherkassky and H. Lari-Najafi. Constrained topological mapping for nonparametric regression analysis. *Neural Networks*, 4:27–40, 1991.
- [19] G. Cirrincione, M. Cirrincione, and G. Vitale. Diagnosis of three-phase converters using the VQP neural network. In *2nd IFAC Workshop on Computer Software Structures integrated AI/KBS Systems in Process Control*, Lund, Sweden, August 1994.
- [20] G. Cirrincione, M. Cirrincione, and G. Vitale. Fault diagnosis in three-phase converters by using the Kohonen neural network classifier. In *SPEEDAM 94*, Taormina, Italy, June 1994.
- [21] G. Cirrincione, M. Cirrincione, and G. Vitale. A Kohonen neural network for the diagnosis of incipient faults for induction machines. In *ICEM*, Paris, September 1994.
- [22] M. Cottrell and J. C. Fort. Etude d'un processus d'auto-organisation. *Annales de l'Institut H. Poincaré*, 23(1):1–20, 1987.
- [23] M. Cottrell and J.-C. Fort. Bases mathématiques pour les réseaux de neurones artificiels. Tutorial COMETT-NEURAL, March 1991. About 100 pages.
- [24] M. Cottrell, J.-C. Fort, and G. Pagès. Two or three things that we know about the Kohonen algorithm. In *European Symposium on Artificial Neural Networks*, pages 235–244, Brussels, April 1994.
- [25] P. Demartines. Peut-on se passer de normalisation dans l'algorithme de Kohonen ? In *Annales du Groupe CARNA C*, volume 4, LAMI-EPFL, INF-Ecublens, CH-1015 Lausanne, Switzerland, 1991. Swiss Federal Institute of Technology. Invited presentation.
- [26] P. Demartines. Mesures d'organisation du réseau de Kohonen. In M. Cottrell, editor, *Congrès Satellite du Congrès Européen de Mathématiques: Aspects Théoriques des Réseaux de Neurones*, 1992.
- [27] P. Demartines. Organization measures and representations of Kohonen maps. In J. Héroult, editor, *First IFIP Working Group 10.6 Workshop*, 1992.
- [28] P. Demartines and F. Blayo. Comparaison de l'algorithme de Kohonen et de l'analyse en composantes principales en analyse de données. In *Journées Neurosciences et Sciences de l'Ingénieur*, Oléron, May 1992.
- [29] P. Demartines and F. Blayo. Kohonen self-organizing maps: Is the normalization necessary ? *Complex Systems*, 6(2):105–123, 1992.
- [30] P. Demartines and A. Guérin. Self-organization: Methods and applications. Tutorial number 5 of *Neuronîmes*, October 1993. About 120 pages.



- [31] P. Demartines and J. Héroult. Representation of nonlinear data structures through fast VQP neural network. In *Neuronimes*, pages 411–424, October 1993.
- [32] P. Demartines and J. Héroult. Vector quantization and projection neural network. In A. Prieto J. Mira, J. Cabestany, editor, *International Workshop on Artificial Neural Networks*, volume 686 of *Lecture Notes in Computer Science*, pages 328–333. Springer-Verlag, 1993.
- [33] P. Demartines and J. Héroult. Apprentissage de structures de données par auto-organisation; interpolation, extrapolation : généralisation. In *Journées Neurosciences et Sciences de l'Ingénieur*, Chamonix, May 1994.
- [34] P. Demartines and L. Tettoni. *SOMA: Simulateurs logiciels de modèles neuronaux adaptatifs. User and Reference Manuals*. Swiss Federal Institute of Technology, LAMI-EPFL, INF-Ecublens, CH-1015 Lausanne, Switzerland, 1991. About 130 pages.
- [35] D. DeSieno. Adding a conscience to competitive learning. *IEEE International Conference on Neural Networks*, 1:117–124, 1988.
- [36] E. Diday, J. Lemaire, P. Pouget, and F. Testu. *Éléments d'analyse de données*. Dunod, Paris, 1983.
- [37] E.W. Dijkstra. A note on two problems in connection with graphs. *Numerical Math.*, 1:69–271, 1959.
- [38] M. Duranton and J. A. Sirat. Learning on VLSI: A general purpose digital neurochip. In *IJCNN conf. on Neural Networks*, Washington DC, June 1989.
- [39] Le Bail E. and A. Mitiche. Quantification vectorielle d'images par réseau neuronal de Kohonen. *Traitement du signal*, 6(6):529–539, 1989.
- [40] E. Erwin, K. Obermayer, and K. Schulten. Self-organizing maps: ordering, convergence properties and energy functions. *Biological Cybernetics*, 67:47–55, 1992.
- [41] E. Filipi and J.C. Lawson. A parallel implementation of Kohonen's self-organizing maps on smart neurocomputer. In *New Trends in Neural Computation: IWANN'93*, volume 686 of *Lecture Notes in Computer Science*, pages 715–720, Sitges, Spain, 1993. Springer-Verlag.
- [42] P. Foldiak. Adaptive networks for optimal linear feature extraction. In *IEEE International Conference on Neural Networks*, volume 1, Washington, 1989.
- [43] J.-C. Fort. Solving a combinatorial problem via self-organizing process: an application of the Kohonen algorithm to the traveling salesman problem. *Biological Cybernetics*, 59:33–40, 1988.
- [44] B. Fritzke. Let it grow — self-organizing feature maps with problem dependent cell structure. In T. Kohonen, K. Mäkisara, O. Simula, and J. Kangas, editors, *Artificial Neural Networks*, volume 1, pages 403–408, North-Holland, 1991. Elsevier Science Publishers.
- [45] B. Fritzke. Growing cell structures — a self-organizing network for unsupervised and supervised learning. Technical report, International Computer Science Institute, Berkeley, May 1993. submitted for publication.
- [46] B. Fritzke. Kohonen feature maps and growing cell structures — a performance comparison. In C.L. Giles, S.J. Hanson, and J.D. Cowan, editors, *Advances in Neural Information Processing Systems*, volume 5, San Mateo, 1993. Morgan Kaufmann.

- [47] A. Gersho and Robert M. Gray. *Vector quantization and signal compression*. Kluwer Academic Publishers, London, 1992.
- [48] G. H. Golub and C. F. Van Loan. *Matrix computation*. John Hopkins series in the mathematical sciences. The John Hopkins University Press, 1990.
- [49] J. Göppert and W. Rosenstiel. Topology preserving interpolation in self-organizing maps. In *Neuronîmes*, pages 425–434, October 1993.
- [50] D. H. Graf and W. R. Lalonde. A neural controller for collision-free movement of general robot manipulators. In *IEEE International Conference on Neural Networks*, volume 1, pages 77–84, San-Diego, 1991.
- [51] D. Hammerstrom and N. Nguyen. An implementation of Kohonen’s self-organizing map on the adaptive solutions neurocomputer. In T. Kohonen, K. Mäkisara, O. Simula, and J. Kangas, editors, *Artificial Neural Networks*, volume 1, pages 715–720, North-Holland, 1991. Elsevier Science Publishers.
- [52] R. Hecht-Nielsen. Counterpropagation networks. *Applied Optics*, 26(23):4979–4984, 1987.
- [53] R. Hecht-Nielsen. Applications of counterpropagation networks. *Neural Networks*, 1:131–139, 1988.
- [54] H. G. E. Hentschel and I. Procaccia. The infinite number of generalized dimensions of fractals and strange attractors. *Physica 8D*, pages 435–444, 1983.
- [55] J. Héroult and C. Jutten. *Réseaux neuronaux et traitement du signal*. Hermès, Paris, 1994.
- [56] R.-D. Hersch, P. Demartines, D.-G. Fridman, P. Pisan, A. Decurnex, and R. Rogner. Multiprocessor raster plotting. *IEEE Computer Graphics*, 12(4):79–87, July 1992.
- [57] J. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the theory of neural computation*, volume 1 of *Santa Fe Institute Lecture Notes*. Addison-Wesley Publishing Company, 1991.
- [58] S. C. Huang and Y. F. Huang. A constrained vector quantization scheme for real-time codebook retransmission. *IEEE Transaction on Circuits and Systems for Video Technology*, 4(1):1–7, February 1994.
- [59] D. H. Hubel and J. M. Wiesel. Receptive fields, binocular interaction, and functional architecture in the cat’s visual cortex. *Journal of Physiology*, 160:106–154, 1962.
- [60] P. Ienne. Architectures for neuro-computers: Review and performance evaluation. Technical Report 93/21, LAMI-EPFL, 1993.
- [61] C. Jutten, A. Guérin, and H. L. Nguyen Thi. Adaptive optimisation of neural algorithms. In A. Prieto, editor, *International Workshop on Artificial Neural Networks*, volume 540 of *Lecture Notes in Computer Science*. Springer-Verlag, 1991.
- [62] J.A. Kangas, T. Kohonen, and J.T. Laaksonen. Variants of self-organizing maps. *IEEE Transaction on Neural Networks*, 1:93–99, 1990.
- [63] J. Kennedy and P. Morasso. Application of self-organizing networks to signal processing. In L. B. Almeida and C. J. Wellekens, editors, *Neural Networks*, volume 412 of *Lecture Notes in Computer Science*, pages 225–232, Sesimbra, Portugal, February 1990. EURASIP Workshop, Springer-Verlag.

- [64] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.
- [65] T. Kohonen. Analysis of a simple self-organizing process. *Biological Cybernetics*, 44:135–140, 1982.
- [66] T. Kohonen. Self-organization of topologically correct feature maps. *Biological Cybernetics*, 43:59–69, 1982.
- [67] T. Kohonen. *Self Organization and Associative Memory*. Springer-Verlag, Washington DC, 2nd edition, 1984.
- [68] T. Kohonen. An introduction to neural computing. *Neural Networks*, 1:3–16, 1988.
- [69] T. Kohonen. The ‘neural’ phonetic typewriter. *Computer*, 21:11–22, March 1988.
- [70] T. Kohonen. *Self-Organization and Associative Memory*. Springer-Verlag, Berlin, 3rd edition, 1989.
- [71] T. Kohonen. The self-organizing maps. *Proc. of the IEEE*, 78(9):1464–1480, 1990.
- [72] T. Kohonen. Self-organizing maps: Optimization approaches. In T. Kohonen *et al.*, editor, *Artificial Neural Networks*, volume 2, pages 981–990, Amsterdam, North Holland, 1991.
- [73] T. Kohonen, G. Barna, and R. Chrisley. Statistical pattern recognition with neural networks : Benchmarking studies. In *IEEE int. conf. on Neural networks*, volume 1, pages 61–68, San Diego, 1988.
- [74] W. L. G. Koontz and K. Fukunaga. A nonlinear feature extraction algorithm using distance transformation. *IEEE Trans. Computers*, C-21(1):56–63, 1972.
- [75] J. B. Kruskal. Nonmetric multidimensional scaling: a numerical method. *Psychometrika*, 29:115–129, June 1964.
- [76] J. Lampinen and E. Oja. Fast self-organization by the probing algorithm. *International Joint Conference on Neural Networks*, 2, 1989.
- [77] J. Lampinen and E. Oja. Clustering properties of hierarchical self-organizing maps. *Journal of Mathematical Imaging and Vision*, 2:261–272, 1992.
- [78] L. Lebart, A. Morineau, and J. P. Fénelon. *Traitement des données statistiques*. Dunod, Paris, 1979.
- [79] C. Lehmann. *Réseaux de neurones compétitifs de grandes dimensions pour l'auto-organisation: analyse, synthèse et implantation sur circuits systoliques*. PhD thesis, Ecole Polytechnique Fédérale de Lausanne, 1993. Thèse No 1129.
- [80] C. Lehmann, M. A. Viredaz, and F. Blayo. A generic systolic array building block for neural networks with on-chip learning. *IEEE Transaction on Neural Networks*, 4(3):400–407, May 1993.
- [81] Y. Linde, A. Buzzo, and R. M. Gray. An algorithm for vector quantizer design. *IEEE Trans. Commun.*, COM-28:84–95, 1980.
- [82] R. Linsker. Self-organization in perceptual network. *Computer*, pages 105–117, March 1988.
- [83] R. J. MacGregor. *Neural and brain modelling*. Academic Press, London, 1987.

- [84] B. B. Mandelbrot. How long is the coast of Britain ? *Science*, 155:636–638, 1967.
- [85] B. B. Mandelbrot. *The fractal geometry of nature*. San Francisco; Freeman, 1982.
- [86] B. B. Mandelbrot. *Les objets fractals : forme, hasard et dimension*. Flammarion, Paris, 1984.
- [87] K. M. Marks and K. F. Goser. Analysis of vlsi process data based on self-organizing feature maps. In *Neuronimes*, pages 337–347, 1993.
- [88] T. Martinetz, S. Berkovich, and K. Schulten. “Neural-Gas” network for vector quantization and its application to time-series prediction. *IEEE Transaction on Neural Networks*, 4(4):558–569, July 1993.
- [89] T. Martinetz and K. Schulten. A neural gas network learns topologies. In T. Kohonen *et al.*, editor, *IEEE International Conference on Artificial Neural Networks, Espoo, Finland*, volume 1, pages 397–407. Elsevier, 1991.
- [90] N. Mauduit, M. Duranton, J. Gobert, and J. A. Sirat. LNeuro: A piece of hardware LEGO for building neural network systems. *IEEE Trans. on Neural Networks*, 3(3):414–422, May 1992.
- [91] J. Minot. Routage adaptatif basé sur un automate cellulaire dans un réseau X25. *Revue annuelle LEP*, pages 62–64, 1992.
- [92] Y. V. V. S. Murty and G. K. Dubey. Fault diagnosis in three-phase thyristor converters using microprocessors. *IEEE Transactions on Industry Applications*, IA-20(6), 1984.
- [93] N. M. Nasrabadi and Y. Feng. Vector quantization of images based on Kohonen self organizing feature maps. In *IEEE International Conference on Neural Networks*, pages 1101–1108, San Diego, 1988.
- [94] D. Niebur and A. Germond. Power system static security assessment using the Kohonen neural network classifier. *IEEE Transactions on Power Systems*, 7(2):865–872, 1992.
- [95] D. Niebur and A. J. Germond. Unsupervised neural net classification of power system static security states. In *Third Symposium on Expert Systems Application to Power Systems, Tokyo & Kobe*, April 1991.
- [96] D. Niebur and members of CIGRE TF 38-06-06 on Artificial Neural Networks Applications for Power Systems. Artificial neural networks for power systems: a literature survey. *Eng Int Syst*, 3:133–158, 1993.
- [97] E. Oja. A simplified neuron model as a principal component analyzer. *Journal of Mathematical Biology*, 15:267–273, 1982.
- [98] E. Oja. Principal components, minor components and linear neural networks. *Neural Networks*, 5:927–935, 1992.
- [99] G. Pagès. Voronoï tessellation, space quantization algorithms and numerical integration. In M. Verleysen, editor, *European Symposium on Artificial Neural Networks*, pages 221–228, Brussels, 1993. D facto.
- [100] T. Poggio and F. Girosi. Networks for approximation and learning. *Proceedings of the IEEE*, 78(9):1481–1497, 1990.
- [101] A. C. Renfrew and J. X. Tian. The use of a knowledge system in power electronics circuit fault diagnosis. In *EPE*, Brighton, 1993.

- [102] H. Ritter. Parametrized self-organizing maps. In *IEEE International Conference on Artificial Neural Networks, Espoo, Finland*, Amsterdam, The Netherlands, September 1993.
- [103] H. Ritter, T. Martinetz, and K. Schulten. *Neural computation and self-organizing maps: an introduction*. Addison-Wesley Publishing Company, 1992.
- [104] H. Ritter and K. Schulten. On the stationary state of Kohonen self-organizing sensory mapping. *Biological Cybernetics*, 54:99–106, 1986.
- [105] J.S. Rodrigues and L.B. Almeida. Improving the learning speed in topological maps of patterns. In *International Neural Networks Conference*, 1990.
- [106] N. Samardzija and R. L. Waterland. A neural network for computing eigenvectors and eigenvalues. *Biological Cybernetics*, 65:211–214, 1991.
- [107] J. W. Sammon. A nonlinear mapping algorithm for data structure analysis. *IEEE Trans. Computers*, C-18(5):401–409, 1969.
- [108] T. D. Sanger. Optimal unsupervised learning in a single-layer linear feedforward neural network. *Neural Networks*, 2:459–473, 1989.
- [109] G. Saporta. *Probabilité Analyse de données et Statistiques*. Technip, Paris, 1990.
- [110] P. G. Schyns. A modular neural network model of concept acquisition. *Cognitive Science*, 15:461–508, 1991.
- [111] R. N. Shepard. The analysis of proximities: Multidimensional scaling with an unknown distance function, parts I and II. *Psychometrika*, 27:125–140 and 219–246, 1962.
- [112] R. N. Shepard. Multidimensional scaling, tree-fitting, and clustering. *Science*, 210:390–398, 1980.
- [113] R. N. Shepard. Toward a universal law of generalization for psychological science. *Science*, 237:1317–1323, 1987.
- [114] R. N. Shepard and J. D. Carroll. Parametric representation of nonlinear data structures. In P. R. Krishnaiah, editor, *International Symposium on Multivariate Analysis*, pages 561–592. Academic Press, 1965.
- [115] A. Takeuchi and S. Amari. Formation of topographic maps and columnar microstructures in nerve fields. *Biological Cybernetics*, 35:63–72, 1979.
- [116] C. Tricot. *Courbes et dimension fractale*. Springer-Verlag, Paris, 1993.
- [117] M. Verleysen, P. Thissen, and J. D. Legat. Linear vector classification : an improvement on LVQ algorithms to create classes of patterns. In *New Trends in Neural Computation : IWANN'93*, volume 686 of *Lecture Notes in Computer Science*, pages 340–345, Sitges, Spain, 1993. Springer-Verlag.
- [118] C. von der Malsburg. Self-organization of orientation sensitive cells in the striate cortex. *Kybernetik*, 14:85–100, 1973.
- [119] J. A. Walter and K. J. Schulten. Implementation of self-organizing neural networks for visuo-motor control of an industrial robot. *IEEE Transaction on Neural Networks*, 4(1):86–95, 1993.

- [120] D.J. Willshaw and C. von der Malsburg. How patterned neural connections can be set up by self-organization. *Proceedings of the Royal Society of London*, B 194:431–445, 1976.
- [121] L. Xu. Adding learned expectation into the learning procedure of self-organizing maps. *International Journal of Neural Systems*, 3:269–283, 1990.
- [122] E. Yair, K. Zeger, and A. Gersho. Competitive learning and soft competition for vector quantizer design. *IEEE Transactions on Signal Processing*, 40(2):294–309, 1992.
- [123] K. Zeger, J. Vaisey, and A. Gersho. Globally optimal vector quantizer design by stochastic relaxation. *IEEE Transactions on Signal Processing*, 40(2):310–322, 1992.

# Index

- abscisse curviligne, 38, 115, 143
- accommodation, 54
- ACP, *voir* analyse en composantes principales
- adaptation
  - facteur d', 52, 55, 76, 98, 108
  - réglage automatique des paramètres, 100, 102
- agglomération des unités, 56
- Aitken, 63
- alarmes
  - anticipation d', 166
  - avalanche d', 165
- Amari, 71
- analyse de données (questions ouvertes), 149
- analyse en composantes curvilignes, 181
- analyse en composantes principales, 33-40
- analyse géopolitique, 37
- anneaux imbriqués (distribution en -), 93, 117, 131
- anticipation d'alarmes, 166
- appariement de formes, 80
- appariement de graphes, 178
- apprentissage d'environnement, 80
- approximation de fonction, 28
- Arabie, 137
- arrangement hexagonal, 67
- ATM, 174
- auto-organisées (cartes -), 14, 20, 60-61, 69-96, 137-141
- auto-similarité, 43, 86
- avalanche d'alarmes, 165
- axes principaux, 34
  
- barycentre des activités, 28, 94, 122
- Boltzmann-Gibbs (distribution de -), 57
- Bretagne (longueur de la côte), 43
- bulle d'activité, 52, 71
  
- Carroll, 134, 136, 137
  
- cartes
  - différentiables, 41
  - évolutives, 93
  - de Kohonen, *voir* auto-organisées (cartes -)
  - sensorielles, topiques, 19, 20, 69, 105
- cartographie
  - de caractéristiques, 80
  - de concepts, 177
- centrage-réduction, 34
- central-limite (théorème -), 23
- centrale nucléaire, 170
- centroïde, 28
- chaîne de Markov, 57
- chapeau mexicain, 70, 72
- Chebychev (inégalité de -), 26
- Cirriuncione, 164
- CL, *voir competitive learning*
- classe absorbante, 81
- CLR, *voir competitive learning with regularization*
- codebook, 49, 61
- collaboration avec
  - Christol Consultants, 166
  - CNET, 174
  - France Telecom, 174
  - ICP-INPG, 155
  - ITMI, 166
  - LRE-EPFL, 170
  - PSA, 166
  - Tractebel, 170
- competitive learning*, 52
- competitive learning with regularization*, 14, 63-67, 99, 107, 183
- compression
  - d'image, 80
  - de palette de couleur, 80, 95
- condition des centroïdes, 50
- condition du plus proche voisin, 50, 52

- conscience, 53  
  généralisation au *winner-take-most*, 101
- continuité  
  en 1D, 133  
  en nD, 134
- contrôle de procédé industriel, 80, 166
- coopération/compétition, 69
- corrélation (matrice de -), 34, 127
- Cottrell, 81
- CounterPropagation, 122
- covariance  
  matrice de -, 34, 63  
  nulle, 38
- curviligne  
  abscisse -, 38, 115, 143  
  représentation -, 81, 91
- De Sieno, 53
- degré de désordre, 84
- degrés de liberté, 21, 40, 42, 152
- Delaunay (triangulation de -), 98
- dénombrement de boîtes, 44
- densité de probabilité (estimation de la -), 28
- dépliage  
  des anneaux imbriqués, 117  
  d'un cercle, 115  
  du fer à cheval, 114  
  de la grille de Kohonen, 82  
  limité dans le NLM, 145  
  par le MDS, 133  
  réduction de dimension, 111-113  
  d'une variété, 111-113  
  vu dans  $dy - dx$ , 151
- diagnostic  
  *incipient faults*, 164  
  de machines électriques, 162  
  d'un procédé industriel, 80, 166  
  en temps réel, 162
- difféomorphisme, 183
- digitaliseur, 20
- Dijkstra, 177
- dimension  
  du bruit, 47, 48, 150  
  brute, 39, 40, 185  
  de corrélation, 44  
  fractale (méthode des boîtes), 45  
  fractale généralisée, 44, 47  
  grande, 23-48  
  de la grille, 79, 86  
  d'information, 44  
  intrinsèque, 22, 40-47, 113, 136, 150  
  locale, 42, 47, 98  
  paramétrique, 40  
  réduction de -, 34, 105, 111-113  
  de similarité, 44  
  structurante, 152, 157  
  variable, 48, 99  
  de la variété, voir dimension intrinsèque
- distance  
  échiquier ( $L^\infty$ ), 60, 76  
  euclidienne ( $L^2$ ), 26, 50, 60, 74  
  de grille, 60, 76, 85  
  de Hamming, 169  
  de Manhattan ( $L^1$ ), 74  
  de Minkowski ( $L^p$ ), 74  
  subjective, 177  
  à une variété, 129, 173, 184  
  entre deux vecteurs aléatoires, 26, 29, 42, 82, 182
- distorsion euclidienne quadratique, 51
- distorsion moyenne, 50, 59, 63
- distribution  
  en anneaux imbriqués, 93, 117, 131  
  de Boltzmann-Gibbs, 57  
  en cercle, 89, 116, 122, 131, 143  
  changeante (non-stationnaire), 100, 186  
  continue, 49, 138, 182  
  discrète, 44  
  en fer à cheval, 38, 39, 40, 47, 79, 82, 92, 114, 144  
  finie (discrète), 42, 49, 56  
  *hello world*, 106  
  hyperplan (dans un -), 30  
  de localisation, 20, 22, 39, 46, 80, 91, 152  
  de Martinetz, 99  
  multi-modale, 55, 89, 93  
  non structurée, 91  
  des phonèmes, 154  
  sur une sphère, 117, 122  
  pour la taxonomie, 118  
  uniforme, 28, 42, 91
- dominance (régions de -), 51, 53, 54, 62, 74, 137
- $dy - dx$  (représentation -), 15, 85-92, 113, 151, 181, 182



- échantillonneur de Gibbs, 57
- échantillonnage irrégulier, 64, 134
- échelle d'observation, 17, 47, 48, 150, 153, 182
- échiquier (distance -), 60, 76
- effet de bord (dans Kohonen), 66, 140
- ellipsoïde d'inertie, 38
- énergie
  - des cartes de Kohonen, 138
  - du MDS, 133
  - du NLM, 141
  - de VQP, 107, 126
- ensemble de Mandelbrot, 43
- ergonomie, 165
- espace vide, 27-30, 47
- euclidienne (distance -), 26, 50, 60, 74
- événements rares (occultation des -), 14, 94
- extrapolation
  - par VQP, 121
- fatigue, 53
  - généralisation au *winner-take-most*, 101
- fer à cheval (distribution en -), 38, 39, 40, 47, 79, 82, 92, 114, 144
- ficelle
  - de Kohonen, 86
  - pelote de -, 47, 185
- fonction noyau (radiale de base), 28
- Fort, 24
- fractales, 43
- frequency sensitive competitive learning*, 53
- Fritzke, 93, 105
- FSCL, *voir frequency sensitive competitive learning*
- Fukunaga, 137
- fusion de données, 19, 152
- généralisation (par VQP), 131
- generalized Lloyd algorithm*, 51
- Gersho, 62
- Givens (matrices de -), 31
- GLA, *voir generalized Lloyd algorithm*
- Global Positioning System, 20
- Göppert, 94
- grande dimension
  - visualisation, 30
- graphe d'état (d'un automate), 168
- Gray, 62
- grille de neurones, 69
- dimension, 79
  - figée *a priori*, 92
- distance dans la -, 60, 76, 85
- forme figée *a priori*, 92
- Grossberg (dilemme de stabilité/plasticité), 186
- Hamming (distance de -), 169
- hash-code*, 46
- Hebb (règle de -), 35, 71
- hello world* (distribution -), 106
- Hessienne (matrice -), 142
- hexagonal (arrangement -), 67
- histogramme des valeurs propres, 39, 156, 161, 171
- homéomorphisme, 41
- Huang, 63
- Hubel, 71
- hyper-cube, 33
- hyper-sphère
  - surface, 28
  - volume, 27
- hyper-tétraèdre, 27
- hyperplan (distribution dans un -), 30
- ILR, *voir* inhibitions/excitations latérales récurrentes
- impulsion (détection d<sup>2</sup>-), 160
- incipient faults*, 164
- indépendance, 38
- inertie (la plus faible), 34
- inertie totale (matrice d<sup>2</sup>-), 34, 63
- infographie, 33
- inhibitions/excitations latérales récurrentes, 52
- interpolation
  - par les cartes de Kohonen, 94
  - par noyaux, 28, 125
  - du voisinage gaussien, 76
  - par VQP, 121
- intrinsèque (dimension -), 22, 40-47, 113, 136, 150
- Karhunen-Loève (transformation de -), *voir* analyse en composantes principales
- Kirkpatrick, 56
- Kohonen, *voir* auto-organisées (cartes -)
- Koontz, 137
- Kruskal, 137

- liens dynamiques, 64, 98, 114, 178
- limitations
  - de l'ACP, 38
  - du CLR, 183
  - de la dimension fractale, 47
  - de Kohonen, 92
  - du NLM, 142
  - de VQP, 184
- Lloyd (algorithme de -), *voir generalized Lloyd algorithm*
- localisation (distribution de -), 20, 22, 39, 46, 80, 91, 152
- localité du respect de topologie, 89
- longueur d'organisation, 86
- machine électrique (diagnostic), 162
- Mandelbrot, 43
  - ensemble, 43
  - exemple de la pelote de ficelle, 47
- Manhattan (distance -), 74
- Markov (chaîne de -), 57
- Martinetz, 61, 97, 99
- matrice
  - centrée-déduite, 34
  - de corrélation, 34, 127
  - de covariance, 34, 63
  - de dissimilitude, 137
  - de Givens, 31
  - Hessienne, 142
  - d'inertie totale, 34, 63
  - inversible, 128
  - de rotation généralisée, 30
  - symétrique, 34, 127, 142
- MDS, *voir multidimensional scaling*
- métrique (fabrication d'une -), 173
- Metropolis (algorithme de -), 57
- minimal spanning tree*, 93, 118
- minimum global, 56
- minimum local, 51
- Minkowski (distance de -), 74
- morphisme, 16, 41, 106, 183
- MST, *voir minimal spanning tree*
- multi-modale (distribution -), 55, 89, 93
- multidimensional scaling*, 16, 133-137, 140
  - non métrique, 137
- neural gas*, 61, 81, 93, 97-103
- Newton (méthode de -), 141
- NLM, *voir non linear mapping*
- non linear mapping*, 16, 112, 141-147, 183
- normalisation des poids, 74
- norme de vecteurs aléatoires, 23
- noyaux
  - gaussiens, 27
  - hyper-gaussiens, 29
- occultation des événements rares, 14, 94
- Oja (règle de -), 35
- organisation
  - mesures de l'-, 81-92
  - mesure de désordre  $\Theta$ , 84, 89
  - représentation  $dy - dx$ , 85
  - représentation curviligne, 81
- organisation (longueur d'-), 86
- pannes (détection), 162
- papillon (configuration -), 77
- pavage
  - en cubes, 43
  - en régions de dominance, 53, 54, 74, 137
  - de Voronoï, 51, 53, 62, 102
- PCVQA, *voir principal component vector quantization algorithm*
- Peano (courbe de -), 86, 89
- pelote (de ficelle), 47, 185
- perspective conique, 33
- phonèmes, 154
- pincement (effet de -), 77
- principal component vector quantization algorithm*, 61-63, 81
- procédé industriel (diagnostic), 80, 166
- projection
  - dans le bruit, 129, 173
  - continue, 121
  - discrète, 92
  - inverse (par VQP), 131
  - linéaire, 22, 38
  - non-linéaire, 38
  - révélatrice, 80
- proximité (conservation de la -), 72
- proximité temporelle, 159
- pseudo-Newton (méthode -), 141, 145
- quantification
  - optimale, 50
  - régulière, 64
  - scalaire, 49

- vectorielle, 49–67
  - par arbre, 61
- quick-sort*, 100
  - modifié, 101
- QV, voir quantification vectorielle
- règle de Schyns, 79
- radial basis function*, 27, 94, 107, 122
- rang de proximité, 61, 97
- Rayleigh (quotient de -), 63
- RBF, voir *radial basis function*
- reconnaissance de parole, 80
- recuit simulé, 56
- redondance, 23
  - comme essence de l'auto-organisation, 71
- réduction de dimension, 34, 38, 105, 111–113
- régions de dominance, voir dominance (régions de -)
- règle de Hebb, 35, 71
- règle de Oja, 35
- règle de Sanger, 35
- régularité de la grille, 84
- rémanence, 158
- représentation
  - curviligne, 81, 91
  - $dy - dx$ , 15, 85–92, 113, 151, 181, 182
  - euclidienne (construction d'une -), 174
  - en grande dimension, 30
  - usuelle du réseau de Kohonen, 73, 93
- rétinopathie, 20
- rétinotopiques (cartes -), 69, 105
- rotation dynamique, 30
- rotation généralisée (matrice de -), 30
- routage adaptatif de paquets en télécommunications, 174
- Sammon, 141
- Sanger (règle de -), 35
- Schulten, 98
- Schyns (règle de -), 79
- SCS, voir *soft competition scheme*
- sélectivité
  - d'un noyau gaussien, 27
  - à l'orientation, 72
- sensorielles (cartes -), 19, 69, 105
- séquence temporelle, 157
- Shannon (quantité d'information), 44
- Shepard, 42, 133, 134, 136
- soft competition scheme*, 58
- SOM, voir cartes de Kohonen
- somatotopiques (cartes -), 105
- sous-variété, 41
- SRS, voir *stochastic relaxation scheme*
- stochastic relaxation scheme*, 56, 58, 81
- structure (recherche de -), 19–22
- subjective (distance -), 177
- super-résolution, 160
- surveillance de procédé, 165
- système de puissance (diagnostic), 162
- Takeuchi, 71
- taxis jaunes (problème des -), 95
- taxonomie, 118
- télécommunications (routage adaptatif de paquets), 174
- théorie motrice, 156
- théorème central-limite, 23
- tonotopiques (cartes -), 105
- topiques (cartes -), 19, 20, 105
- topologie (conservation de la -), 69
- transformation de coordonnées, 80
- triangulation de Delaunay, 98
- Tricot, 48
- uniforme (distribution -), 28, 42, 91
- unités mortes, 52, 59, 92, 101, 183
- valeur propre, 34, 63, 128
- variance (maximale), 34
- variance reconstruite, 35
- variété, 22, 41, 64
  - dépliage d'une -, 111–113
  - distance à une -, 129, 173, 184
- vecteur propre, 34, 63
- vecteurs aléatoires, 26
  - distances entre deux -, 26, 29, 42, 82, 182
  - norme des -, 23
- vector quantization and projection*, 15, 89, 105–131, 145, 181
  - règle d'adaptation, 109
- verrou (problème du -), 109
- viewer*, 22, 30, 171
- Vigilance Neuromimétique, 166
- voisinage
  - adaptatif, 98

- adaptation dans un -, 76
- gaussien, 60, 76
- dans Kohonen, 60, 76
- dans le *neural gas*, 97
- en  $p > 2$  dimensions, 79, 163
- rayon de -, 60, 76, 79, 98, 112
- rectangulaire, 60, 76
- dans VQP, 113
- Von der Malsburg, 20, 69–71
- Von Neumann, 133, 135, 139
- Voronoi (pavage de -), 51, 53, 62, 102
- VQP, voir *vector quantization and projection*
  
- Wiesel, 71
- Willshaw, 69
- winner-take-all*, 51–55
- winner-take-most*, 55
  
- Yair, 56
  
- $z$ -buffer, 33
- Zeger, 56
- zones de dominance, voir dominance (régions de -)