Approche analytique pour l'optimisation de réseaux de neurones artificiels

Yohann Bénédic

Année 2007 Numéro d'ordre : **07 MULH 1234 / 56**

Université de Haute-Alsace
U.F.R. des Sciences et Techniques

APPROCHE ANALYTIQUE POUR L'OPTIMISATION DE RÉSEAUX DE NEURONES ARTIFICIELS

THÈSE

présentée pour l'obtention du
Doctorat de l'Université de Haute-Alsace
(Discipline génie informatique)
par

Yohann BÉNÉDIC

Soutenue publiquement le 11 décembre 2007 devant le jury :

Francis BRAUN	professeur de l'université Louis-Pasteur (Strasbourg)
David MONNIN	chercheur de l'institut franco-allemand de recherches de Saint-Louis
Jean MERCKLÉ	professeur de l'université de Haute-Alsace (Mulhouse)
Michel PAINDAVOINE	professeur de l'université de Bourgogne (Dijon)
Nadine PIAT	professeur de l'université de Franche-Comté (Besançon)
Patrice WIRA	maître de conférences de l'université de Haute-Alsace (Mulhouse)

Thèse préparée au sein du laboratoire MIPS sous la direction de Jean MERCKLÉ.
École doctorale Jean-Henri LAMBERT

Sommaire

1

Introduction générale

« Réunir un ensemble cohérent de connaissances relatives à une certaine catégorie de faits, d'objets ou de phénomènes obéissant à des lois et vérifiées par les méthodes expérimentales » tel est le moteur de la science [1]. Au fil des siècles, fidèle à cet objectif, elle a continué de repousser ses propres limites, chaque réponse étant la source de toujours plus de nouvelles questions. Aujourd'hui, elle s'est divisée en une myriade de disciplines : mathématiques, chimie, biologie, physique, mécanique, optique, astronomie, économie, sociologie, ... si bien qu'il n'y a plus un seul sujet qui ne soit pas l'objet de recherches.

Toutefois, s'il devait y avoir une discipline centrale, elle serait probablement les mathématiques ; non parce que celle-ci est qualifiée de « reine » par beaucoup, mais tout simplement parce qu'elle sert de fondement à toutes les autres. D'ailleurs, nombre de nos connaissances reposent fondamentalement sur les mathématiques, dans le sens où ce sont elles qui dictent la logique d'un raisonnement, la justesse d'une mesure, la cohérence d'une théorie. Sans elles, nous ne saurions pas mettre en équation une idée, un principe, ni même confronter ces derniers de façon rigoureuse à la réalité factuelle.

Aujourd'hui, un nouvel avènement des mathématiques a lieu : les prédictions de la relativité générale et de la mécanique quantique, parmi les premières théories de la physique moderne à avoir été élaborées à partir d'expériences de pensée et qui

[1] Définition du mot « science » tirée du Petit Larousse 2007.

ne trouvent donc de justification que par leur cohérence mathématique, concordent aux observations avec une précision étonnante, conduisant à des percées technologiques majeures [Wigner, 1960]. Malgré tout, si certaines théories se révèlent très fructueuses, d'autres butent encore sur un obstacle qui semble infranchissable : dans leurs travaux sur les équations différentielles, T. Li et J. Yorke l'ont appelé le « chaos » [Li et Yorke, 1975].

Ce terme caractérise les phénomènes déterministes dont la sensibilité aux conditions initiales les rend parfaitement imprévisibles à plus ou moins long terme [Alligood et al., 1997]. Dans un langage un peu plus mathématique, on parle de phénomène chaotique lorsque les équations qui le régissent doivent tenir compte d'un nombre exponentiellement croissant d'informations pour conserver le même niveau de précision lors de prédictions à plus longue échéance. Leur caractère aléatoire, *a priori* contradictoire avec les lois déterministes dont ils sont issus, provient des infimes variations qui ont été négligées — à tort — lors de l'établissement des équations les décrivant.

Dans *A New Kind of Science* [Wolfram, 2002], S. Wolfram a construit, à partir des concepts sous-jacents à celui du chaos, le principe d'équivalence calculatoire [2] abrégé en pce. Il énonce que la complexité inhérente à l'établissement d'un résultat est invariante, quel que soit le mode de calcul employé. En d'autres termes, la prédiction de certains phénomènes physiques nécessite un effort calculatoire minimal et, à moins d'en négliger certains aspects, aucune astuce mathématique ni algorithmique ne permettra d'être plus rapide. S'il est avéré, ce principe impliquerait que les mathématiques modernes laissent de côté tout un pan théorique sans lequel l'émergence du chaos est inévitable [3]. En somme, l'adéquation outil/tâche a toujours été quelque chose d'absolument déterminant pour résoudre un problème et le pce suggère que nous n'employons peut-être pas les bons outils en ce qui concerne toute une gamme de questions ouvertes de la physique et de l'informatique.

L'efficacité surprenante avec laquelle le cerveau humain manie les flux d'informations, par opposition aux performances mitigées des traitements informatiques correspondants, en est une illustration parfaite. En effet, on sait depuis longtemps que le cerveau humain a un fonctionnement fondamentalement différent de celui d'un ordinateur ; l'un réalisant en parallèle un nombre astronomique de petits calculs pendant que l'autre exécute successivement les instructions formant un algorithme. Une différence qui se traduit également au niveau de leur tâche de prédilection :
- le cerveau humain excelle pour les traitements de matériels ayant une densité informationnelle élevée (comme la vision ou le langage) ;
- les tâches plus procédurales (comme la bureautique ou le calcul) conviennent pour leur part bien mieux aux ordinateurs.

[2] Il s'agit d'une traduction libre de l'anglais *principle of computational equivalence*, dans la mesure où aucune traduction officielle n'a, semble-t-il, été formulée.

[3] S. Wolfram explique comment, dans le cadre des mathématiques calculationnistes, chaque phénomène naturel est le résultat de l'itération d'un calcul élémentaire simple, dont il convient d'étudier les propriétés en tant qu'entité. Les approches traditionnelles cherchent quant à elles à réduire ces itérations en une expression formelle unique, faisant ainsi émerger le chaos lorsque cette tentative viole le pce.

J. Weng est parvenu à une observation similaire dans ses travaux sur le développement mental autonome en robotique [Huang et Weng, 2004], dénommant *muddy* ces tâches si facilement effectuées par le cerveau humain et *clean* celles pour lesquelles un ordinateur s'avère plus approprié. Il a remarqué que les solutions les plus efficaces pour traiter les premières avaient en commun de ne pas être bâties autour de modèles mathématiques particuliers des problèmes à résoudre, et que leur mode de fonctionnement mettait en jeu, de façon plus ou moins prononcée, un certain parallèlisme [Adouane et Lefort-Piat, 2005, Prague, République Tchèque]. Parmi ces solutions d'un nouveau genre, on trouve les réseaux de neurones artificiels, inventés il y a plus de cinquante ans à partir des connaissances que l'on avait alors du cerveau humain.

Quand W. Mc Culloch et W. Pitts ont publié, en 1943, le premier papier montrant comment réaliser une architecture de calcul inspirée du cerveau humain, la communauté scientifique, alors très enthousiaste, a promis l'arrivée des premiers ordinateurs intelligents pour les prochaines trente années [Mc Culloch et Pitts, 1943]. Aujourd'hui, nous sommes forcés de constater que cette promesse est très loin d'avoir été tenue et que le fait de disposer d'architectures de calcul fonctionnant « à peu près » comme le cerveau humain n'est pas suffisant pour y parvenir : il reste à savoir comment les programmer correctement. Mais comment faire si les mathématiques modernes n'ont effectivement pas encore étudié l'outil adéquat ?

Il faut avant tout correctement positionner le problème ; en l'occurrence, il n'est nullement question de dénigrer les qualités des nombreux résultats obtenus autour des réseaux de neurones artificiels. Il s'agit plutôt de trouver comment mieux exploiter la débauche d'efforts et de tâtonnements qui a été nécessaire à leur mise au point, ainsi que d'améliorer leur manque manifeste de souplesse vis-à-vis de leur contexte de fonctionnement. Deux aspects qui, à notre sens, reflètent l'inadéquation des outils mathématiques communément utilisés dans ce domaine et qui appellent donc au développement de nouveaux.

Dans le présent mémoire, nous allons développer un outil mathématique original que nous utiliserons par la suite pour re-programmer des réseaux de neurones artificiels. Il s'agit de recherches initiées par l'institut franco-allemand de recherches de Saint-Louis et que nous avons poursuivies au laboratoire modélisation, intelligence, processus et système de l'université de Haute-Alsace de Mulhouse dans le cadre d'un doctorat.

Le point de départ de notre travail a été la définition, par D. Monnin, d'une nouvelle opération mathématique, baptisée n-somme [Monnin et al., 2002]. C'est une porte logique à seuil d'un nouveau genre, qui a la caractéristique de mettre en avant les propriétés de symétrie intrinsèques aux traitements qui sont exprimés par son biais, et qui facilite ainsi la mise en œuvre d'un réseau de neurones artificiels les réalisant. Notre travail a alors consisté à esquisser un cadre théorique autour de la n-somme, puis d'en étudier tous les bénéfices possibles dans le cadre de la programmation de

réseaux de neurones artificiels. Le caractère particulier de cette opération, ainsi que le contexte exposé par cette introduction, nous incite à croire que cette approche apportera de nouvelles réponses au domaine encore très ouvert des réseaux de neurones artificiels.

Dans un souci de clareté, un chapitre complet a été consacré à chacune des notions abordées par ce mémoire. Ces derniers ont été regroupés en trois grandes parties :

I une présentation des réseaux de neurones artificiels ;

II l'analyse du comportement d'un neurone artificiel binaire ;

III la programmation par synthèse d'un neurone artificiel binaire et d'un réseau complet.

Dans la première partie, nous passerons en revue les modèles de réseaux de neurones artificiels les plus populaires de la littérature. Ce sera l'occasion de mettre en avant les nombreux points communs de ces modèles, en particulier dans la façon dont ils définissent les neurones artificiels. Nous nous efforcerons également d'y dégager les deux courants de pensée qui continuent de motiver les principales innovations dans le domaine et dont un des aboutissements est la naissance des réseaux de neurones cellulaires. Il s'agit du modèle qui est à l'origine de notre thématique de recherches en étant le premier, par sa sensibilité au bruit provoqué par son propre fonctionnement, à nécessiter d'optimiser la façon dont un traitement est implémenté. Une question sur laquelle les outils mathématiques traditionnels donnent très peu de réponses.

Dans la deuxième partie, nous commencerons par montrer comment la plupart des modèles des chapitres précédents peuvent être exprimés à partir d'un réseau de neurones binaires, c'est-à-dire de neurones artificiels opérant exclusivement sur des données binaires. À partir d'une analyse approfondie de leurs propriétés, nous tirerons un outil mathématique original, appelé « famille génératrice ». Associé à l'opération n-somme, que nous introduirons également, nous montrerons qu'il a plusieurs des qualités de l'outil mathématique calculationniste qui fait tant défaut aux réseaux de neurones artificiels. Nous terminerons cette partie sur un algorithme d'analyse de neurones binaires dont le résultat exprime directement leur fonction à l'aide de familles génératrices.

Exprimés de cette façon, les neurones binaires peuvent alors être reprogrammés analytiquement par synthèse de leur famille génératrice, une méthode originale qui fera l'objet de la troisième et dernière partie de ce mémoire. Nous y présenterons tout d'abord les méthodes de programmation analytiques concurrentes, afin de mieux mettre en relief leurs différences avec notre approche par famille génératrice. Nous relaterons ensuite les étapes qui ont aboutis à cette méthode de synthèse et sans lesquelles elle n'existerait pas. Nous terminerons enfin avec la méthode de synthèse en elle-même, et montrerons comment l'utiliser pour programmer de façon intuitive, un réseaux de neurones artificiels complet dont l'implémentation répond à certains critères extérieurs comme la dynamique de calcul, le nombre de neurones ou encore la façon de les mettre en réseau.

Première partie

Les réseaux de neurones artificiels

Sommaire de la partie

2

Neurones artificiels et réseaux

L'idée de concevoir des réseaux de neurones artificiels fut très largement inspirée des connaissances neurologiques des années cinquante. L'objectif était alors de reproduire artificiellement les comportements intelligents observés chez n'importe quelle espèce animale. C'est W. Mc Culloch et W. Pitts qui donnèrent naissance au premier modèle de neurone artificiel [Mc Culloch et Pitts, 1943]. Il s'agissait alors d'une petite révolution qui fut à l'origine de nombreux travaux, en particulier ceux de B. Widrow [Widrow et Walach, 1996] et F. Rosenblatt [Rosenblatt, 1962].

À l'époque, il y avait une certaine rivalité entre les deux hommes quant à qui mettrait au point le modèle le plus performant. B. Widrow avait choisi de s'attaquer au problème par la voie mathématique, tandis que F. Rosenblatt cherchait avant tout à reproduire les observations biologiques le mieux possible. C'est de cette façon que le premier aboutit au modèle adaline, qui rencontra un certain succès grâce au cadre rigoureux entourant sa mise au point, et qui gageait son bon fonctionnement. Le second conçut le perceptron, mais son maniement basé sur des postulats biologiques était un peu déconcertant pour l'époque. Adaline et perceptron se configuraient

grâce à un mécanisme d'apprentissage [1], ce qui constituait une innovation notable : une opération n'était plus programmée comme une suite d'instructions élémentaires, mais à l'aide d'exemples d'utilisation la décrivant de manière aussi complète que possible [Widrow et Lehr, 1990]. Cette technique s'appelle un *apprentissage supervisé*.

De façon relativement surprenante, ces neurones artificiels furent étudiés individuellement pendant quelques années et leur utilisation au sein d'un réseau n'a été envisagée que plus tard. C'est alors que le perceptron prit l'avantage sur son concurrent, récompensant ainsi l'audace dont ont dû faire preuve ses partisans pour imposer une architecture de calcul dont on ne savait finalement pas grand chose du fonctionnement.

L'histoire de ces deux modèles illustre parfaitement l'opposition entre le courant applicatif et le courant neuromimétique, telle qu'elle a été esquissée dans l'introduction générale. Le premier a conduit à l'adaline, un modèle dont le bon fonctionnement est rigoureusement garanti, tandis que le deuxième a mené au perceptron, dont la programmation est certes plus aléatoire, mais dont les perspectives d'évolution se sont révélées bien meilleures. L'objet de ce chapitre est de présenter de façon un peu plus précise ces deux modèles qui, aujourd'hui encore, sont à la base de pratiquement toutes les recherches du domaine des réseaux de neurones artificiels.

2.1 Le neurone de Mc Culloch et Pitts

2.1.1 Un peu de biologie

Il a fallu attendre le début du vingtième siècle pour que les fameuses « cellules grises » soient reconnues comme étant à l'origine des processus de traitement, de décision et de contrôle, caractéristiques du règne animal. C'est en effet suite à plusieurs observations que l'anatomiste S. R. y Cajal a pu conclure que ces cellules particulières étaient capables d'échanger des informations par le biais de différentes structures biologiques [López-Muñoz et al., 2006]. Son travail fut d'ailleurs à l'origine de la doctrine neuronale, aujourd'hui à la base des neurosciences, et qui fixe les principes suivants :

- les neurones sont des cellules différenciées distinctes ;
- chaque neurone a un corps cellulaire complet, un axone et éventuellement une ou plusieurs dendrites ;
- l'axone d'un neurone est lié à une dendrite d'un autre neurone par le biais d'une synapse ;
- entre deux neurones, l'information se déplace par inversion de polarisation, de proche en proche, des dendrites vers les axones.

[1] Lors d'une conférence, F. Rosenblatt présenta son perceptron sous la forme d'une grande boite en bois ; les différents composants étaient réglés par des potentiomètres eux-mêmes actionnés par des moteurs commandés par le mécanisme d'apprentissage. On pouvait donc « entendre » le perceptron apprendre une opération !

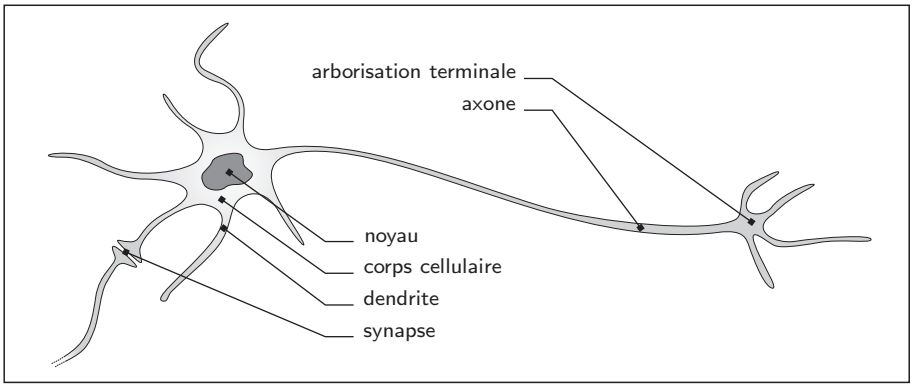


Figure 2.1
Schéma simplifié d'un neurone biologique.

La figure 2.1 schématise grossièrement l'anatomie d'un neurone biologique.

Des études approfondies ont permis d'identifier trois façons avec lesquelles un neurone a peut influencer un neurone b, en fonction du type de synapses qui les relient :

excitatrices l'excitation de a facilite l'excitation de b ;

inhibitrices l'excitation de a rend plus difficile l'excitation de b ;

modulatrices l'excitation de a perturbe de manière globale la capacité d'excitation des neurones avoisinnants.

D'un point de vue fonctionnel, chaque neurone réagit en fonction de l'état d'excitation des neurones qui sont connectés à ses dendrites et des types de synapses impliquées. En fonction de l'intensité de cette activité, le neurone passe, ou non, dans un état d'excitation et transmet à son tour cette information aux neurones suivants par l'intermédiaire de son axone.

Bien que le comportement exact ne soit pas encore complètement compris, les neurosciences s'accordent à dire que l'excitation d'un neurone est grossièrement déclenchée dès que la quantité d'informations apportée par ses dendrites dépasse un seuil critique. C'est de cette assertion qu'est né le concept de neurone formel.

2.1.2 Les neurones formels

Deux principales étapes se dégagent du fonctionnement d'un neurone biologique (voir figure 2.2) :
- une fonction d'agrégation $\mathcal{A}$ caractérise l'environnement amont du neurone (appelé « état »), en utilisant les informations reçues par ses dendrites ;
- un processus de décision Φ, également appelé fonction d'activation, provoque le passage à l'état excité du neurone et donc sa transmission à son environnement aval.

Au travers de ce comportement, chaque neurone biologique peut donc être modélisé mathématiquement comme la composition des fonctions $\mathcal{A}$ et Φ : $\Phi \circ \mathcal{A}$. La première

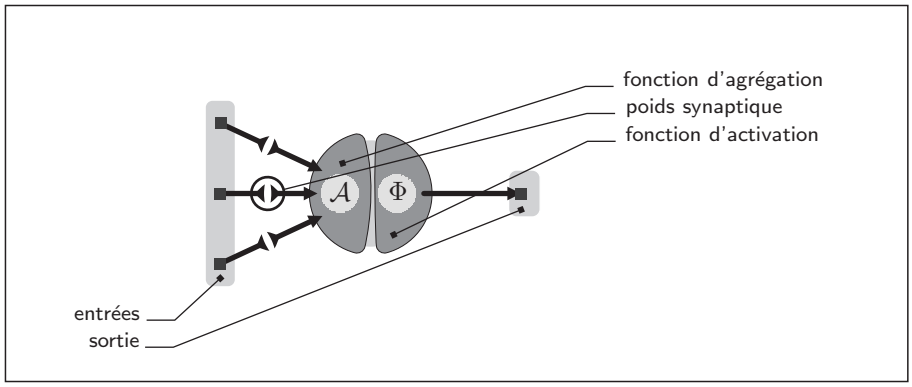


Figure 2.2
Schéma d'un neurone formel.

a une action paramétrique sur n variables d'entrée e_1, e_2, …, e_n, dont les valeurs symbolisent l'activité respective d'autant de dendrites. Le résultat x de cette fonction est passé à la seconde, qui évalue le taux d'activité qui doit être envoyé sur l'axone du neurone.

Dans ce contexte, la fonction composée $\Phi \circ \mathcal{A}$ est appelée *neurone formel*, ou encore neurone artificiel ; terme que nous avons déjà employé.

Définition 2.1 (neurone formel).
Il s'agit de la composition de deux fonctions $\mathcal{A}$ et Φ définies sur deux algèbres quelconques $\mathbb{E}_1$ et $\mathbb{E}_2$:

$$\mathcal{A}\colon \mathbb{E}_1^n \longrightarrow \mathbb{E}_2,$$
$$\Phi\colon \mathbb{E}_2 \longrightarrow \mathbb{E}_1.$$

La fonction $\mathcal{A}$ calcule l'état du neurone formel à partir de ses n entrées dont les influences respectives sont modelées par n paramètres appelés « poids synaptiques » en référence aux synapses dont ils symbolisent le rôle.

Le neurone formel de Mc Culloch et Pitts Le tout premier neurone formel a été imaginé à partir de considérations neurophysiologiques et anatomiques par W. Mc Culloch et W. Pitts en 1943 [Mc Culloch et Pitts, 1943]. Bien que très simpliste, leur neurone formel reste encore aujourd'hui à la base de nombreux travaux, dont ceux exposés dans ce manuscrit. Dans leur modèle (que nous nommerons par la suite neurone « MP »), $\mathcal{A}$ est une somme pondérée prenant chacun de ses arguments dans $\mathbb{E}_1 = \{0\,;1\}$ et a valeur dans $\mathbb{E}_2 = \mathbb{R}$. Les n poids synaptiques du vecteur $\mathbf{w} = \begin{pmatrix} w_1 & w_2 & \dots & w_n \end{pmatrix}^\top$ sont tout simplement les coefficients de pondération appliqués aux entrées ; nous verrons que c'est d'ailleurs le cas dans la majorité des autres modèles. Par définition, Φ est logiquement une fonction de $\mathbb{R}$ dans $\{0\,;1\}$, qui vaut 1 si et seulement si la valeur de la fonction d'agrégation dépasse une valeur seuil fixée par un dernier paramètre w_0.

Ainsi, le signe affecté à un poids synaptique permet de modéliser les synapses excitatrices (cas positif) et les synapses inhibitrices (cas négatif). Par hypothèse[2], les poids synaptiques excitateurs sont tous égaux à w_e, tandis que les inhibiteurs sont égaux à w_i. Les synapses modulatrices ne sont pas modélisées.

La fonction $\Phi \circ \mathcal{A}$ du neurone formel MP est donc :

$$\begin{aligned} \Phi \circ \mathcal{A} \colon \{0\,;1\}^n &\longrightarrow \{0\,;1\} \\ \mathbf{e} = \begin{pmatrix} e_1 & e_2 & \cdots & e_n \end{pmatrix}^\top &\longmapsto \mathrm{H}\left(w_0 + (w_1 e_1 + w_2 e_2 + \cdots + w_n e_n)\right) \\ &= \mathrm{H}\left(w_0 + \mathbf{w}^\top \mathbf{e}\right). \end{aligned} \tag{2.1}$$

La fonction H utilisée dans cette définition est la fonction échelon (voir figure 2.3a).

Les méthodes de programmation associées à ce type de neurones sont empiriques et consistent pour la plupart à choisir les coefficients de pondération suite à une analyse fine du problème à résoudre. C'est exactement sur ce principe qu'est basée l'approche qui sera développée dans les parties II et III.

Les autres neurones formels Depuis, beaucoup d'autres modèles de neurones formels ont fait leur apparition. La motivation derrière leur invention n'était d'ailleurs pas toujours de mieux coller à la réalité biologique, mais plutôt de les rendre moins fastidieux à programmer. Ces autres modèles font naturellement appel à des fonctions d'activation et/ou d'agrégation différentes de celles utilisées pour le neurone MP.

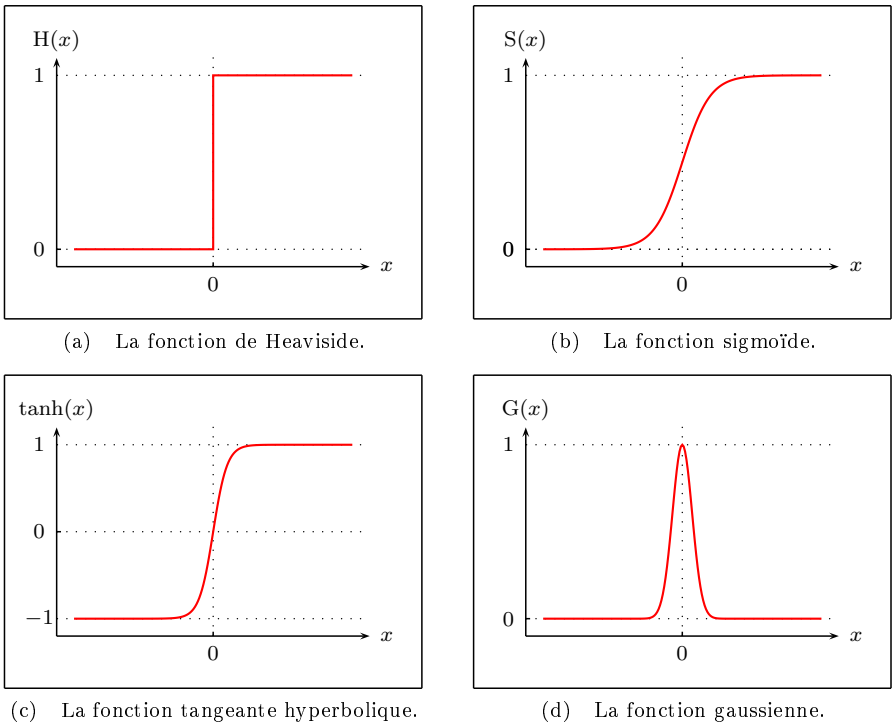
On trouve notamment les fonctions d'activation suivantes, dont les profils sont donnés dans la figure 2.3 :
- la fonction sigmoïde $\mathrm{S}(x) = (1 + \mathrm{e}^{-x})^{-1}$;
- la fonction tangeante hyperbolique $\tanh(x) = (\mathrm{e}^{x} - \mathrm{e}^{-x})/(\mathrm{e}^{x} + \mathrm{e}^{-x})$;
- la fonction gaussienne $\mathrm{G}(x) = \mathrm{e}^{-x^2}$.

Contrairement à la fonction échelon, elles ont l'avantage d'être infiniment dérivables, ce qui est une propriété mathématique non négligeable en analyse, et qui explique donc que ces fonctions soient souvent préférées à la fonction échelon. Du côté des fonctions d'agrégation, les plus connues sont celles du neurone formel à base radiale, dont l'état x est une mesure de la distance entre le vecteur d'entrée $\mathbf{e} = \begin{pmatrix} e_1 & e_2 & \dots & e_n \end{pmatrix}^\top$ et un vecteur de référence défini par les poids synaptiques $\mathbf{w} = \begin{pmatrix} w_1 & w_2 & \dots & w_n \end{pmatrix}^\top$, ainsi que du neurone « sigma-pi » qui, comme son nom l'indique, utilise une somme pondérée (Σ) de produits (Π) de ses entrées.

Formellement, le neurone à base radiale est défini de la façon suivante :

$$\begin{aligned} \Phi \circ \mathcal{A} \colon \mathbb{R}^n &\longrightarrow \mathbb{R} \\ \mathbf{e} &\longmapsto \mathrm{G}\big(\|\mathbf{e} - \mathbf{w}\|\big). \end{aligned} \tag{2.2}$$

[2] Il s'agit d'une hypothèse purement simplificatrice qui permet de ramener le nombre de paramètres à trois : w_0, w_e et w_i.



(a) La fonction de Heaviside.

(b) La fonction sigmoïde.

(c) La fonction tangeante hyperbolique.

(d) La fonction gaussienne.

Figure 2.3
Les principales fonctions d'activation utilisées par les neurones formels.

L'utilisation de la fonction d'activation gaussienne est pertinente en ce qu'elle permet à la sortie d'être d'autant plus proche de l'unité que $\mathbf{e}$ est proche de $\mathbf{w}$.

Le neurone formel sigma-pi n'impose pas de fonction d'activation particulière. Par contre, il nécessite de définir les n produits d'entrées qui sont associés aux n poids synaptiques, ce qui augmente d'autant les degrés de libertés quant à la programmation de ce type de neurones. Sa fonction d'agrégation est donc définie à partir de n ensembles d'indices J_i réunissant les entrées dont le produit est pondéré par w_i :

$$\begin{aligned} \Phi \circ \mathcal{A} : \mathbb{R}^n &\longrightarrow \mathbb{R} \\ e_1, e_2, \ldots, e_n &\longmapsto \Phi\left(\Sigma_{i=1}^{n}(w_i \Pi_{j \in J_i} e_j)\right). \end{aligned} \tag{2.3}$$

2.2 L'ADALINE

L'*adaptive linear neuron* (ADALINE) est né dans un laboratoire de Stanford en 1960 à partir du neurone MP [Widrow et Hoff Jr, 1960]. Il a été conçu par B. Widrow et T. Hoff avec l'objectif d'être simple à programmer et donc à exploiter dans un contexte industriel [3]. À cette fin, ils ont assoupli le modèle du neurone MP, et lui ont surtout adjoint une méthode de programmation aux performances prouvées.

[3] Il en sortira un des exemples d'application les plus impressionnants à ce jour : le pilotage en marche arrière d'un camion-remorque télécommandé [Nguyen et Widrow, 1990] depuis n'importe quelle position jusqu'à un dock de déchargement.

2.2.1 Définition

Un ADALINE se caractérise par une fonction d'agrégation qui, contrairement au neurone MP, est paramétrée par des poids synaptiques indépendants, et qui a recour à une entrée auxiliaire e_0 bloquée à 1 pour inclure le problème du réglage du seuil à celui de la détermination des poids synaptiques. La fonction d'activation est historiquement une fonction échelon. Ce neurone formel s'écrit :

$$\begin{aligned} \Phi \circ \mathcal{A} \colon \{0\,;1\}^{n+1} &\longrightarrow \{0\,;1\} \\ e_0 = 1, \mathbf{e} = \begin{pmatrix} e_1 & e_2 & \dots & e_n \end{pmatrix}^\top &\longmapsto \mathrm{H}\left(w_0 e_0 + w_1 e_1 + \dots + w_n e_n\right) \\ &= \mathrm{H}\left(w_0 + \mathbf{w}^\top \mathbf{e}\right). \end{aligned} \tag{2.4}$$

Outre ces aménagements, ses concepteurs ont aussi mis au point une technique, appelée la règle « delta » qui permet d'obtenir les $n+1$ poids synaptiques à partir d'exemples de la tâche que l'ADALINE doit réaliser. Le principe est d'ajuster petit à petit les poids synaptiques de sorte que l'erreur quadratique observée entre les sorties attendues (celles des exemples) et celles réellement obtenues soit diminuée [WIDROW et HOFF JR, 1960]. Les principales caractéristiques d'un ADALINE sont illustrées par la figure 2.4a.

2.2.2 La règle d'apprentissage delta

Bien que ce ne soit pas l'objet de ce manuscrit, nous allons détailler cette règle afin de faire ressortir la logique analytique qui en est à l'origine. Les exemples de l'opération à programmer sont fournis sous la forme de m couples $\left(\mathbf{e}^{(j)} \quad s^{(j)}\right)|_{j=1}^{m}$, où $s^{(j)}$ correspond à la sortie attendue de l'ADALINE lorsqu'on lui donne les composantes de $\mathbf{e}^{(j)}$ en entrées. Le mécanisme d'apprentissage doit déterminer le poids synaptique auxiliaire w_0 et les poids synaptiques composant $\mathbf{w}$ de sorte que l'erreur E suivante soit aussi petite que possible :

$$E = \sum_{j=1}^{m} E^{(j)} = \sum_{j=1}^{m} \left(s^{(j)} - w_0 - \mathbf{w}^\top \mathbf{e}^{(j)}\right)^2. \tag{2.5}$$

Cela revient à minimiser l'erreur quadratique $E^{(j)}$ de l'ADALINE sur chaque exemple. Cette opération est réalisée en ajustant chaque poids synaptique dans le sens indiqué par le signe de la dérivée partielle de $E^{(j)}$. Cette dérivée s'écrit de façon triviale par rapport au poids synaptique w_i :

$$\frac{\partial E^{(j)}}{\partial w_i} = -2 e_i^{(j)} \left(s^{(j)} - w_0 - \mathbf{w}^\top \mathbf{e}^{(j)}\right). \tag{2.6}$$

La mise à jour du poids w_i, suite à l'utilisation du j^{e} exemple, doit donc être proportionelle à $-\frac{\partial E^{(j)}}{\partial w_i}$ ce qui s'écrit plus classiquement :

$$\Delta w_i = \eta e_i^{(j)} \left(s^{(j)} - w_0 - \mathbf{w}^\top \mathbf{e}^{(j)}\right), \tag{2.7}$$

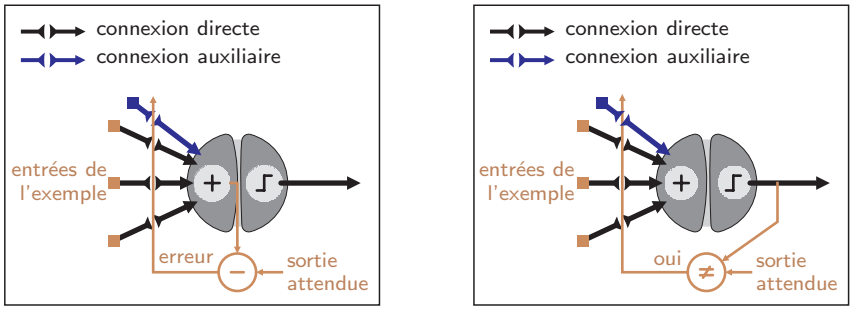


Figure 2.4 Comparaison fonctionnelle entre l'ADALINE et le perceptron.

(a) La règle d'apprentissage de l'ADALINE est basée sur un calcul d'erreur.

(b) Le mécanisme d'apprentissage du perceptron est indépendant de l'erreur.

où η est une constante appelée « taux d'apprentissage ».

Un couple d'exemple après l'autre, tous les poids sont modifiés selon la règle Delta, jusqu'à ce que toutes les énergies $E^{(j)}$ soient négligeables. On dit alors que l'ADALINE a appris à faire cette opération.

2.2.3 Les MADALINEs

Grâce à la règle d'apprentissage delta de l'équation (2.7), un ADALINE semble pouvoir apprendre n'importe quelle opération. C'est malheureusement loin d'être le cas et il faut avoir recours à plusieurs d'entre-eux pour y parvenir. Ils forment alors un réseau de neurones artificiels (voir la figure 2.5), appelé, dans le cas où il est formé d'ADALINEs, MADALINE pour *many* ADALINEs.

Dès lors se pose le problème de l'utilisation de la règle delta pour réaliser l'apprentissage des ADALINEs constituant le réseau. En effet, les exemples de l'opération à apprendre ne sont disponibles qu'à l'échelle du réseau, et non à celle des neurones. Par conséquent, ces derniers calculent ou bien la sortie du réseau à partir de résultats intermédiaires, ou bien un état intermédiaire à partir des entrées du réseau ou, pire encore, un résultat intermédiaire à partir d'autres résultats intermédiaires ; bref, tous ont besoin d'un jeu d'exemples dont on ignore au moins une partie. L'apprentissage d'un tel réseau de neurones artificiels, avec la même rigueur que la règle delta, est ainsi mathématiquement très compliqué.

En pratique, un MADALINE est donc construit sur deux couches, à l'instar du réseau de la figure 2.5, éliminant de cette façon les neurones artificiels dont le jeu d'exemples est totalement inconnu. Deux règles, définies de façon empirique, sont alors principalement utilisées pour l'apprentissage :

la règle MR-I considère que les ADALINEs de sortie n'ont pas besoin d'être modifiés, et utilisent donc l'algorithme d'apprentissage delta pour chaque neurone artificiel de la couche d'entrée [WIDROW et HOFF JR, 1960] ;

la règle MR-II effectue un apprentissage sur tous les ADALINEs du réseau, pourvu qu'ils contribuent à l'erreur observée en sortie, ce qui est déterminé en vérifiant

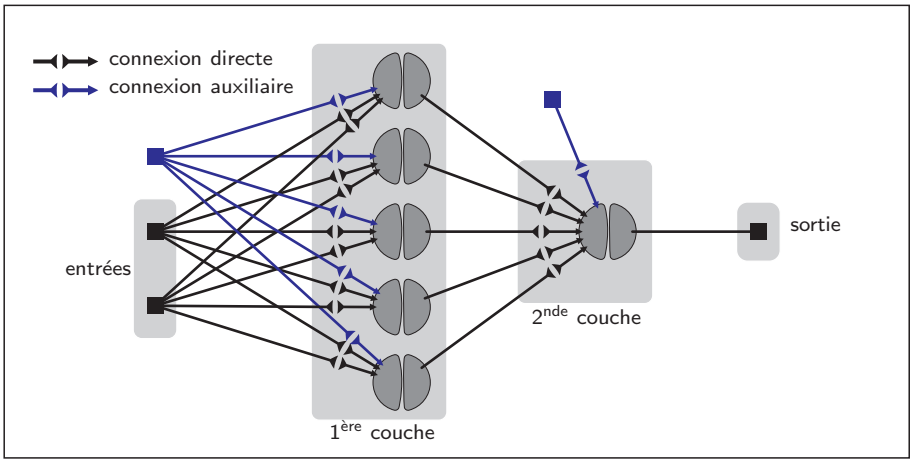


Figure 2.5
Mise en réseau de neurones selon plusieurs couches (deux ici).

si un changement de signe de leur sortie améliore la sortie du réseau [Widrow et al., 1987].

Il est intéressant de constater que leur seul fondement est de sembler mettre en œuvre un processus d'apprentissage « crédible », dont le taux de succès [4] est difficile à quantifier rigoureusement. À la lumière des outils mathématiques actuels, ce type d'architecture est donc une sorte de boîte noire dont on ne sait finalement pas grand chose du fonctionnement. C'est la raison pour laquelle le modèle MADALINE est relativement délaissé au profit du perceptron multicouche ; son équivalent à base de perceptrons.

2.3 Le perceptron

Ce neurone formel a été développé en parallèle de l'ADALINE par F. Rosenblatt qui, en plus d'autoriser plus de liberté dans le choix de ses poids synaptiques, a enrichi le neurone MP de la règle d'apprentissage de Hebb, postulée par D. Hebb à partir d'observations neuropsychologiques [Hebb, 1949]. Cette règle propose l'autorenforcement des connexions les plus utilisées comme mécanisme d'apprentissage et de mémorisation. Cela signifie que si, sur un jeu d'exemples donné, une entrée particulière d'un neurone artificiel est souvent active en même temps que la sortie de ce dernier, alors le poids synaptique correspondant doit avoir une valeur plus élevée que les autres.

[4] Une opération que l'on sait pouvoir être implémentée par un MADALINE pourra-t-elle effectivement l'être en appliquant une de ces deux règles ?

2.3.1 Définition

Le perceptron est un neurone formel parfaitement identique à l'ADALINE, en particulier en ce qui concerne le poids synaptique w_0 qui est également intégré à la fonction d'agrégation [ROSENBLATT, 1958] :

$$\begin{aligned} \Phi \circ \mathcal{A} : \{0\,;1\}^{n+1} &\longrightarrow \{0\,;1\} \\ e_0 = 1, \mathbf{e} = \begin{pmatrix} e_1 & e_2 & \dots & e_n \end{pmatrix}^\top &\longmapsto \mathrm{H}\left(w_0 + \mathbf{w}^\top \mathbf{e}\right). \end{aligned} \tag{2.8}$$

En définitive, la seule différence entre les deux modèles est leur règle d'apprentissage dont les origines sont fondamentalement différentes. Celles du perceptron sont purement empiriques et n'impliquent donc pas de processus de minimisation délibéré. Néanmoins il a été établi, sous certaines hypothèses, que cette règle d'apprentissage conduisait malgré tout à des poids synaptiques réalisant l'opération décrite par le jeu d'exemples utilisé [HERTZ et al., 1991].

La règle de HEBB s'écrit, pour le perceptron et en utilisant le même formalisme que la règle delta de l'ADALINE [ROSENBLATT, 1958] :

$$\Delta w_i = \eta s^{(j)} e_i^{(j)}. \tag{2.9}$$

Un examen superficiel de cette règle montre que la valeur du poids synaptique w_i est effectivement augmentée si et seulement si l'entrée $e_i^{(j)}$ et la sortie $s^{(j)}$ sont activées en même temps dans l'exemple j. La figure 2.4b de la page 16 résume les principales caractéristiques du perceptron.

2.3.2 Le perceptron multicouche

Enthousiasmé par la réussite d'une règle d'apprentissage directement inspirée des mécanismes observés sur le pendant biologique du neurone formel, F. ROSENBLATT proposa qu'un perceptron pouvait résoudre n'importe quel problème de classification en un temps d'apprentissage fini [ROSENBLATT, 1962]. Malheureusement, après quelques recherches, M. MINSKY et S. PAPERT prouvèrent que cette assertion était trivialement fausse, ce qui eu pour effet de jeter un discrédit sur le perceptron qui aura mis presque dix ans à se dissiper [MINSKY et PAPERT, 1969]. En fait, le perceptron se heurtait tout simplement aux mêmes limitations que son concurrent ; une limitation qui n'avait finalement rien à voir avec les performances de leur règle d'apprentissage respective mais qui était d'origine purement architecturale.

Comme pour l'ADALINE, la parade a été de mettre plusieurs perceptrons en réseau, formant ainsi le perceptron multicouche. Toutefois, cette opération fut ici bien plus facile puisque la règle d'apprentissage de HEBB se généralise à un réseau entier sans poser de difficulté particulière : qu'il s'agisse de résultats intermédiaires ou non, la valeur de chaque poids synaptique du réseau est augmentée à chaque fois que l'entrée

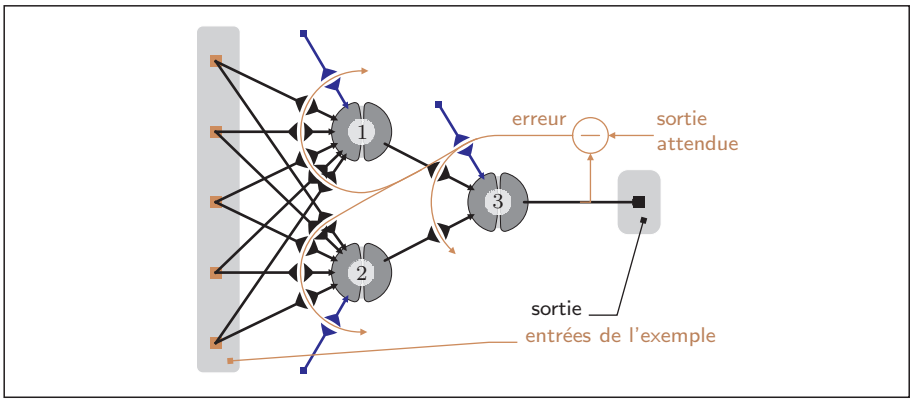


Figure 2.6
Apprentissage d'un perceptron multicouche par rétropropagation du gradient.

et la sortie correspondante sont activées en même temps. Bien entendu, l'efficacité de cette règle, qu'elle soit appliquée à un perceptron isolé ou à un réseau complet, n'en est pas démontrée pour autant.

De façon certes un peu caricaturale, le perceptron et l'ADALINE illustrent ainsi l'incapacité des outils mathématiques actuels à traiter convenablement les systèmes calculationnistes. En effet, les développements de l'ADALINE, bien que basés sur des outils mathématiques, ont très rapidement été limités par la complexité induite par leur mise en réseau. Au contraire, le perceptron a été fondé sur des règles observées sur les neurones biologiques — dont les performances ne sont pas à prouver — et a bien mieux supporté la mise en réseau. Son manque de formalisme mathématique adéquat traduit alors le côté apparemment aléatoire avec lequel un apprentissage sur perceptron multicouche réussit.

L'intérêt des neurones artificiels, sans possibilité d'être mis en réseau, est discutable [MINSKY et PAPERT, 1969]. C'est la raison pour laquelle leur véritable essort ne vint qu'à la fin des années quatre-vingt, lors de la mise au point d'un algorithme d'apprentissage spécialement pensé à l'échelle d'un réseau : la rétropropagation du gradient [WERBOS, 1990, 1994 ; MC CLELLAND et RUMELHART, 1988 ; PARKER, 1985 ; LE CUN, 1987]. Cette technique détermine en particulier la façon dont l'erreur observée en sortie doit être propagée dans les couches internes afin que la règle delta puisse y être appliquée (voir la figure 2.6).

Aujourd'hui, la rétropropagation du gradient est devenue un algorithme d'apprentissage incontournable tant les applications auxquelles elle a donné lieu sont nombreuses [HAYKIN, 2003 ; DREYFUS et al., 2002]. Bien qu'elle ait été améliorée depuis, cette technique reste néanmoins basée sur les outils mathématiques classiques : par conséquent, la convergence vers la solution optimale de chaque neurone est garantie, ce qui n'est en revanche pas le cas du réseau dans son ensemble. Ce dernier peut ne jamais atteindre la configuration qui résolve le problème appris, ou alors en atteindre une mauvaise, d'un point de vue des ressources utilisées par exemple.

L'exemple suivant montre le résultat d'un apprentissage par rétropopagation du gra-

dient. Nous verrons que les poids synaptiques obtenus, s'ils programment la bonne tâche, sont loin d'être les meilleurs.

Exemple 2.1 — Apprentissage d'une fonction logique simple.
Soit une fonction logique $\mathcal{L}$ de cinq variables booléennes, définie par les 2^5 paires entrées/sortie de son tableau de vérité. L'expression algébrique de $\mathcal{L}$ est :

$$\mathcal{L} \equiv e_3(e_5 + \overline{e}_1 + e_4) + e_5(e_2 + \overline{e}_4).$$

Il n'y a naturellement pas besoin de connaître cette expression pour réaliser l'apprentissage.
De façon arbirtraire, nous choisissons d'implémenter ce jeu d'exemples à l'aide d'un perceptron multicouche tel celui de la figure 2.6, c'est-à-dire composé de deux perceptrons en alimentant un troisième. La dynamique de codage des poids synaptiques est fixée, de façon également arbitraire, à 16 bits, et leurs valeur initiale est choisie aléatoirement dans $-5{,}000$ et $5{,}000$ pour les trois neurones artificiels :

neurone 1
$w_{1,0} = 2{,}919$
$w_{1,1} = -0{,}435$
$w_{1,2} = -4{,}815$
$w_{1,3} = 3{,}214$
$w_{1,4} = -0{,}553$
$w_{1,5} = 1{,}154$

neurone 2
$w_{2,0} = 4{,}169$
$w_{2,1} = 4{,}218$
$w_{2,2} = 2{,}382$
$w_{2,3} = -3{,}237$
$w_{2,4} = -0{,}943$
$w_{2,5} = 4{,}355$

neurone 3
$w_{3,0} = 2{,}621$
$w_{3,1} = -0{,}140$
$w_{3,2} = 3{,}913$

Les trois perceptrons sont munis d'une fonction sigmoïde pour fonction d'activation afin de les rendre compatibles avec l'algorithme de rétropropagation du gradient. Ce dernier converge après environ 2 000 présentations du jeu d'exemples et les poids synaptiques obtenus sont :

neurone 1
$w_{1,0} = -10{,}37$
$w_{1,1} = -2{,}661$
$w_{1,2} = -1{,}328$
$w_{1,3} = 7{,}861$
$w_{1,4} = 3{,}393$
$w_{1,5} = -0{,}533$

neurone 2
$w_{2,0} = -7{,}549$
$w_{2,1} = 1{,}812$
$w_{2,2} = 3{,}622$
$w_{2,3} = -8{,}281$
$w_{2,4} = -6{,}294$
$w_{2,5} = 5{,}816$

neurone 3
$w_{3,0} = -3{,}522$
$w_{3,1} = 6{,}734$
$w_{3,2} = 4{,}374$

L'évolution du tableau de vérité réalisé au fil de l'apprentissage du réseau est donné dans la figure 2.7, y représentant respectivement en blanc et en noir les valeurs vrai et faux de $\mathcal{L}$.
Les techniques d'analyse et de programmation présentées dans ce manuscrit permettront d'obtenir les poids synaptiques suivants. Ils sont codés sur seulement 7 bits et dont nous montrerons que les valeurs de seuil sont mieux ajustées :

neurone 1
$w_{1,0} = -3{,}5$
$w_{1,1} = -1{,}0$
$w_{1,2} = 2.0$
$w_{1,3} = 2{,}0$
$w_{1,4} = -2{,}0$
$w_{1,5} = 5{,}0$

neurone 2
$w_{2,0} = -3{,}5$
$w_{2,1} = -2{,}0$
$w_{2,2} = -1{,}0$
$w_{2,3} = 5{,}0$
$w_{2,4} = 2{,}0$
$w_{2,5} = 1{,}0$

neurone 3
$w_{3,0} = -0{,}5$
$w_{3,1} = 1{,}0$
$w_{3,2} = 1{,}0$

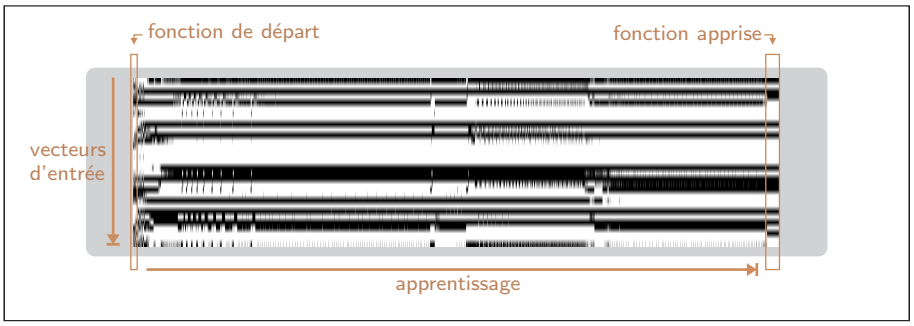


Figure 2.7
Suivi de l'apprentissage par rétropropagation du gradient d'un tableau de vérité (blanc pour la valeur vrai et noir pour la valeur faux).

3

Autres réseaux de neurones artificiels

En s'appuyant sur la complémentarité des deux courants de pensée qui en sont à l'origine, les principes de base des réseaux de neurones artificiels ont brièvement été exposés dans le chapitre précédent. D'un côté, les réseaux neuromimétiques, inspirés de la biologie et dont les développements se révèlent être à l'origine de nombreuses innovations. De l'autre, les réseaux de neurones artificiels à vocation applicative, dont les partisans cherchent à valider les modèles par les mathématiques et qui contribuent à améliorer la maîtrise des modèles imaginés par les premiers.

En suivant un schéma similaire, ce chapitre présente deux outils informatiques dont l'essor est sans aucun doute à mettre au crédit des réseaux de neurones artificiels : les mémoires associatives et les classifieurs. Par ce biais nous soulignerons ainsi la remarquable polyvalence dont font preuve les architectures neuronales. En effet, l'impact des remaniements architecturaux nécessaires à l'implémentation de l'un ou l'autre de ces outils se limite le plus souvent à leur connectivité : une modification fonctionnelle majeure bien qu'elle ne bouleverse pourtant pratiquement pas les modèles. De cette manière, la plupart des développements informatiques étudiés pour un modèle particulier de réseau de neurones artificiels peuvent s'appliquer également aux autres modèles. C'est tout particulièrement le cas de la méthode de programmation analytique proposée dans ce manuscrit.

3.1 Les mémoires associatives

Aristote proposa que le mécanisme de mémorisation humain devait consister en l'établissement de liens entre plusieurs sensations voisines [Fausett, 1994] : l'odeur de l'herbe fraîchement coupée, le bruit de la tondeuse à gazon et l'image d'une pelouse rase par exemple. Cette conclusion se base sur le fait que la perception d'une seule de ces sensations rappelle immédiatement toutes les autres en mémoire ; un processus qui nécessiterait beaucoup de « calculs » si elles étaient stockées indépendamment les unes des autres. L'idée de mémoire associative était née.

Habituellement, le matériel informatique mémorise une information en l'écrivant dans un de ses blocs mémoire, identifiés par des adresses uniques grâce auxquelles l'information pourra être rappelée. Dans le cas d'une mémoire associative, le procédé est totalement différent. En effet, les informations n'y sont pas stockées individuellement mais par paires ; les unes servant d'adresses pour rappeler les autres. En pratique, on peut donc imaginer lire une information complète à partir d'une version partielle de cette dernière — on parle alors de mémoire auto-associative, ou adressable par son contenu — ou à partir d'une information de nature complètement différente — dans ce cas, il s'agit d'une mémoire hétéro-associative. À noter qu'il en existe un troisième type pour laquelle les deux informations jouent un rôle équivalent : les mémoires associatives bidirectionnelles.

En pratique, les mémoires associatives sont utilisées pour la reconnaissance de formes (caractères, visages), le routage des réseaux informatiques, l'exploration de bases de données, la traduction.

3.1.1 Implémentation d'une mémoire associative avec un perceptron monocouche

À l'instar d'une mémoire associative, un perceptron réalise une association entre les entrées et les sorties respectives du jeu d'exemples utilisé pour son apprentissage [Anderson et Rosenfeld, 1988]. Néanmoins, comme il ne dispose que d'une seule sortie binaire, les types d'informations mémorisables par ce biais sont en conséquences très limités. C'est pourquoi on utilise généralement plusieurs perceptrons indépendants dont chacun s'occupe d'un bit de l'information à retrouver : on parle alors de perceptron monocouche. Mémoriser une donnée $\mathbf{s}$ écrite sur n_s bits requiert ainsi n_s perceptrons. Chacun d'eux utilise les n_e entrées au travers desquelles la donnée $\mathbf{e}$ servant d'adresse est codée (voir figure 3.1).

Pour le stockage de la paire $(\mathbf{e} \quad \mathbf{s}) = \left((e_1 \quad \dots \quad e_{n_e})^\top \quad (s_1 \quad \dots \quad s_{n_s})^\top\right)$, la règle de Hebb [1] permet de trouver les poids synaptiques adéquats. Pour n_s perceptrons,

[1] Il s'agit de la règle qui a inspiré la règle d'apprentissage des perceptrons (voir l'équation (2.9), page 18) dont elle est le cas particulier $\eta = 1$.

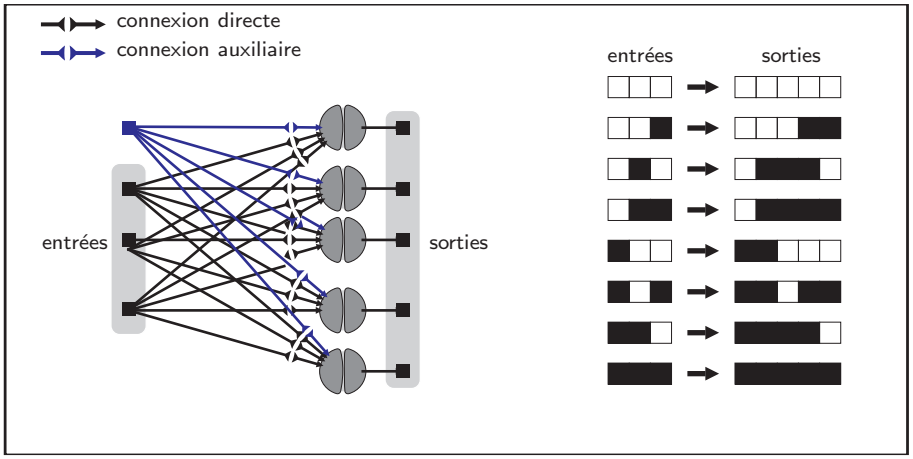


Figure 3.1
Exemple de mémoire associative réalisée à partir d'un perceptron monocouche.

elle s'écrit sous forme matricielle de la façon suivante :

$$\begin{aligned}\mathbf{W} &= \mathbf{s}\mathbf{e}^\top \\ &= \begin{pmatrix} s_1 e_1 & \cdots & s_1 e_{n_e} \\ \vdots & . & \vdots \\ s_{n_s} e_1 & \cdots & s_{n_s} e_{n_e} \end{pmatrix}.\end{aligned} \tag{3.1}$$

L'élément $w_{ij} = s_i e_j$ est le poids synaptique du i^{e} perceptron relatif à sa j^{e} entrée. De cette façon, si l'entrée $\mathbf{e}$ est présentée au perceptron monocouche défini par $\mathbf{W}$ et dont les fonctions d'activation sont des fonctions échelon, alors la sortie est bien $\mathbf{s}$:

$$\Phi\left(\mathbf{W}\mathbf{e}\right) = \Phi\left(\mathbf{s}\left\|\mathbf{e}\right\|^2\right) = \mathbf{s}. \tag{3.2}$$

Naturellement, il est possible de mémoriser plusieurs paires $\left(\mathbf{e}^{(k)} \quad \mathbf{s}^{(k)}\right)|_{k=1}^{m}$ au sein d'un même réseau. Il suffit pour cela de calculer la moyenne des poids synaptiques obtenus individuellement par l'application de la relation (3.1) à chacune des m paires : [Anderson et Rosenfeld, 1988]

$$\mathbf{W} = \frac{1}{m}\sum_{k=1}^{m}\mathbf{W}^{(k)} = \frac{1}{m}\sum_{k=1}^{m}\mathbf{s}^{(k)}\mathbf{e}^{(k)\top}. \tag{3.3}$$

3.1.2 Limitations du perceptron monocouche

La mémoire associative présentée dans la sous-section précédente est extrêmement simpliste et fonctionne relativement rarement en pratique. En effet, si elle est indéniablement efficace pour stocker une association de données, sa généralisation à m paires implique de nombreuses contraintes totalement absentes de la formulation (3.3) :

– des données parasites sont forcément associées aux entrées ne figurant pas dans le jeu d'exemples appris ;

– il est raisonnable de penser que certaines paires puissent être incompatibles entre-elles ; la mémorisation de l'une d'elle provoquant « l'oubli » d'une autre et réciproquement ;
– il y a vraisemblablement une limite au nombre d'associations que le réseau peut mémoriser.

Autant de limitations potentielles évidentes, mais pour lesquelles le modèle perceptron monocouche, bâti sur de simples observations biologiques, n'apporte que très peu de précisions.

Afin d'exprimer ces contraintes le mieux possible, une analyse mathématique du perceptron monocouche est donc nécessaire. Par exemple, la règle d'apprentissage (3.3) suggère que seules les données « adresse » mutuellement orthogonales ont une chance de fonctionner [Anderson et Rosenfeld, 1988]. En effet, si tel est le cas alors la sortie du perceptron monocouche, pour l'entrée $\mathbf{e}^{(k_0)}$ s'exprime :

$$\begin{aligned}\Phi\left(\mathbf{W}\mathbf{e}^{(k_0)}\right) &= \Phi\left(\sum_{k=1}^{m}\mathbf{s}^{(k)}\mathbf{e}^{(k)\top}\mathbf{e}^{(k_0)}\right)\\ &= \Phi\left(\mathbf{s}^{(k_0)}\mathbf{e}^{(k_0)\top}\mathbf{e}^{(k_0)} + \sum_{\substack{k=1\\k\neq k_0}}^{m}\mathbf{s}^{(k)}\underbrace{\mathbf{e}^{(k)\top}\mathbf{e}^{(k_0)}}_{=0}\right)\\ &= \Phi\left(\mathbf{s}^{(k_0)}\left\|\mathbf{e}^{(k_0)}\right\|^2\right),\end{aligned}$$

où il est facile de constater que la sortie associée est alors bien retrouvée.

Au travers de cette brève analyse, une condition sur la nature des associations pouvant être apprises avec succès a pu être établie. Bien qu'elle soit *a priori* très restrictive, elle n'est pourant pas suffisante pour garantir que l'apprentissage fonctionnera. Chaque sortie est en effet obtenue à partir d'un unique perceptron, or le chapitre précédent à indiqué que c'était potentiellement insuffisant pour certains jeux d'exemples.

Une fois encore, le perceptron monocouche, aux règles inspirées de la biologie, souffre de l'incapacité des outils mathématiques actuels à décrire correctement son fonctionnement. Beaucoup de travaux se concentrent donc plutôt sur des architectures alternatives dont les modèles sont plus faciles à cerner. Les paragraphes suivants en présentent sommairement deux d'entre-eux.

Le réseau de neurones artificiels de Hopfield Dans le domaine des mémoires auto-associatives, l'un des modèles les plus étudiés est celui mis au point par J. Hopfield en 1982 [Dreyfus et al., 2002]. Il s'agit d'un réseau de neurones itératif, basé sur le *linear associator* [Mc Clelland et Rumelhart, 1988]. Chacun des neurones artificiels du réseau reçoit ainsi, en plus de sa propre sortie, la sortie de tous les autres (voir la figure 3.2), ce qui rend le modèle dynamique. Les poids synaptiques réglant l'influence de la sortie d'un neurone i sur un neurone j peuvent prendre

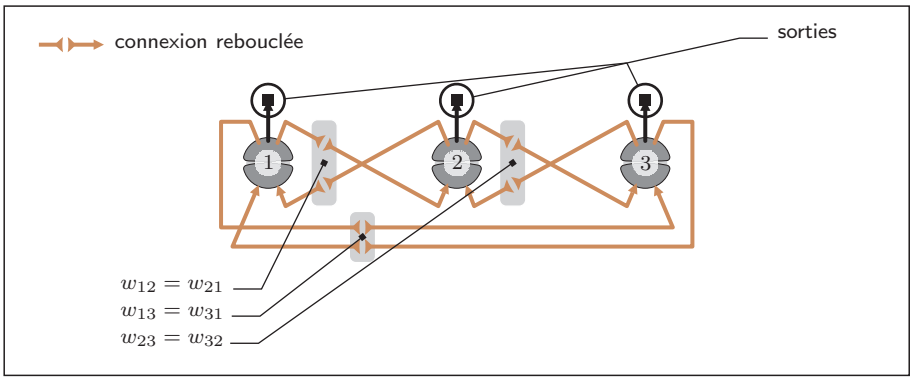


Figure 3.2
Réseau de HOPFIELD composé de trois neurones artificiels.

n'importe quelle valeur pourvu qu'elle satisfasse aux deux conditions suivantes :

$$w_{i,j} = w_{j,i} \quad \text{et} \quad w_{i,i} = 0. \tag{3.4}$$

Les neurones utilisés sont des perceptrons de $\{-1\,;+1\}^n$ dans $\{-1\,;+1\}$, munis d'une fonction d'activation échelon et leur sortie est mise à jour de façon asynchrone[2]. Ces contraintes ont pour principal objectif de garantir la convergence du réseau de neurones en un temps fini [BENNANI, 2006]. Malheureusement, si la sortie se stabilise effectivement quelle que soit la façon dont le réseau ait été initialisé, elle n'est pas forcément celle attendue.

D'un point de vue fonctionnel, l'apprentissage était assuré à l'origine par une règle similaire à celle du perceptron monocouche (voir l'équation (3.3)) à la différence près que $\mathbf{e}^{(k)} = \mathbf{s}^{(k)}|_{k=1}^{m}$ puisqu'il s'agit d'un modèle de mémoire auto-associative :

$$\mathbf{W} = \sum_{k=1}^{m} \left(\mathbf{e}^{(k)} \mathbf{e}^{(k)\top} - \begin{pmatrix} e_1^{2(k)} & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & e_{n_e^2(k)} \end{pmatrix} \right). \tag{3.5}$$

Le réseaux de HOPFIELD étant une architecture dynamique, cet apprentissage a pour conséquence de positionner les données à mémoriser dans des minima locaux. Comme le réseau est stable, il est donc forcément initialisé dans le bassin d'attraction de l'une d'elles. De cette façon, les données mémorisées peuvent être retrouvées à partir de versions partielles, ou dégradées.

Malheureusement, le prix à payer est la mémorisation de données parasites, notamment la version complémentée de chaque information mémorisée. De plus, la condition d'orthogonalité établie plus tôt continue de s'appliquer, ce qui est sans conteste le principal problème de cette règle d'apprentissage.

[2] Aujourd'hui, une version synchrone a été développée mais elle ne montre pas de grande différence avec la version asynchrone originale, sinon que sa stabilité est plus délicate à établir.

Exemple 3.1 — Mémorisation d'imagettes par un réseau de Hopfield.
Les quatre imagettes de la figure 3.3, dont nous avons préalablement vérifié l'orthogonalité, sont mémorisées par un réseau de neurones artificiels de Hopfield de 12 neurones. Chaque neurone calcule la valeur d'un pixel des imagettes : noir pour la valeur -1 et blanc pour $+1$. Par application de l'équation (3.5), les poids synaptiques suivants sont obtenus :

$$\mathbf{W} = \begin{pmatrix} 0 & 0 & 2 & 0 & 0 & 0 & 2 & 4 & -2 & 0 & 0 & -2 \\ 0 & 0 & 2 & -4 & 0 & 0 & -2 & 0 & -2 & 0 & 0 & 2 \\ 2 & 2 & 0 & -2 & -2 & 2 & 0 & 2 & 0 & -2 & -2 & 0 \\ 0 & -4 & -2 & 0 & 0 & 0 & 2 & 0 & 2 & 0 & 0 & -2 \\ 0 & 0 & -2 & 0 & 0 & -4 & -2 & 0 & -2 & 0 & 0 & -2 \\ 0 & 0 & 2 & 0 & -4 & 0 & 2 & 0 & 2 & 0 & 0 & 2 \\ 2 & -2 & 0 & 2 & -2 & 2 & 0 & 2 & 0 & 2 & 2 & 0 \\ 4 & 0 & 2 & 0 & 0 & 0 & 2 & 0 & -2 & 0 & 0 & -2 \\ -2 & -2 & 0 & 2 & -2 & 2 & 0 & -2 & 0 & -2 & -2 & 0 \\ 0 & 0 & -2 & 0 & 0 & 0 & 2 & 0 & -2 & 0 & 4 & 2 \\ 0 & 0 & -2 & 0 & 0 & 0 & 2 & 0 & -2 & 4 & 0 & 2 \\ -2 & 2 & 0 & -2 & -2 & 2 & 0 & -2 & 0 & 2 & 2 & 0 \end{pmatrix}. \quad (3.6)$$

À titre d'exemple d'utilisation, la figure 3.4 montre comment le réseau, initialisé avec l'imagette du chiffre « 1 » (voir la figure 3.3a) dans laquelle 25 % des valeurs ont été inversées, retrouve l'imagette correcte. Il est intéressant de constater que ce niveau de bruit est déjà suffisant pour rendre l'identification visuelle de l'imagette originale difficile.

La méthode de programmation développée dans les parties suivantes de ce manuscrit obtient, pour cet exemple, des poids synaptiques identiques à $\mathbf{W}$ au facteur $\frac{1}{2}$ près. Il arrive cependant que les données à mémoriser s'accomodent moins bien de la règle d'apprentissage utilisée ici et donnent un jeu de poids synaptiques très mauvais, bien que malgré tout fonctionnel.

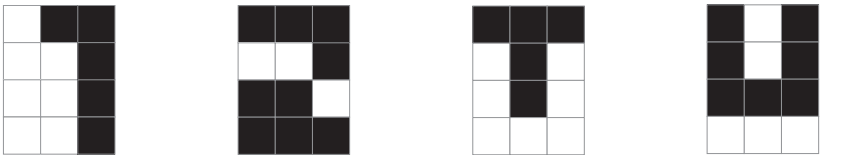
Par exemple, avec les trois imagettes de la figure 3.5, les poids synaptiques obtenus par application de la règle (3.5) sont :

$$\mathbf{W} = \begin{pmatrix} 0 & -1 & -1 & -1 & -1 \\ -1 & 0 & 3 & 3 & -1 \\ -1 & 3 & 0 & 3 & -1 \\ -1 & 3 & 3 & 0 & -1 \\ -1 & -1 & -1 & -1 & 0 \end{pmatrix}. \quad (3.7)$$

Parce que ces valeurs sont mal ajustées, la fonction d'agrégation est amenée à prendre une valeur nulle pour beaucoup de vecteurs d'entrée, c'est-à-dire qu'elle est positionnée exactement sur la valeur de seuil discriminant le cas où la sortie doit valoir $+1$ de celui où elle doit valoir -1.

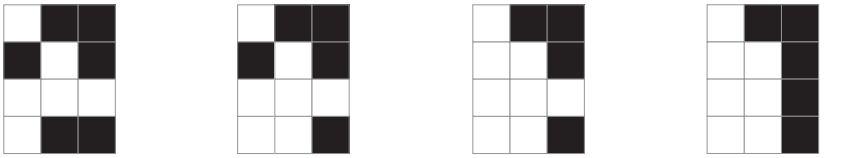
La méthode présentée par ce manuscrit permet quant à elle d'éviter ce problème tout en réduisant la dynamique de codage :

$$\mathbf{W}' = \begin{pmatrix} 0 & -1 & -1 & -1 & -1 \\ -1 & 0 & 2 & 2 & -1 \\ -1 & 2 & 0 & 2 & -1 \\ -1 & 2 & 2 & 0 & -1 \\ -1 & -1 & -1 & -1 & 0 \end{pmatrix}. \quad (3.8)$$



(a) Chiffre « 1 ». (b) Chiffre « 2 ». (c) Lettre « T ». (d) Lettre « U ».

Figure 3.3
Les quatre imagettes, deux à deux orthogonales, utilisées pour l'apprentissage.



(a) Imagette bruitée à $t = 0$. (b) État à $t = 3\tau$. (c) État à $t = 15\tau$. (d) Imagette obtenue à $t = 30\tau$.

Figure 3.4
Reconnaissance d'une imagette mémorisée à partir d'une version bruitée. (τ désigne la durée d'un pas de calcul.)

Les réseaux de neurones artificiels chaotiques Une des contraintes majeures, partagée par toutes les mémoires associatives dont l'implémentation est basée sur un perceptron monocouche (ou multicouche), est leur faible capacité de stockage [3]. Cette limitation est due à l'utilisation de règles d'apprentissage dérivées de la règle de Hebb et qui imposent aux données d'être au moins linéairement indépendantes pour pouvoir être mémorisées [Mc Clelland et Rumelhart, 1988]. Cependant, cela ne veut nullement dire qu'aucun autre type de neurone formel ne peut faire mieux.

En donnant naissance au neurone chaotique, ce sont encore des observations biologiques qui ont permis de multiplier par cinq la capacité de stockage des réseaux de neurones artificiels [Molter et al., 2004 ; Babloyantz et Loureno, 1994 ; Aihara et Matsumoto, 1987]. La principale caractéristique de ce modèle est que chaque neurone artificiel conserve la mémoire d'un certain nombre de ses états passés lors du calcul de ses états futurs [Aihara et al., 1990]. En comparaison, les réseaux de neurones artificiels de Hopfield mémorisent uniquement l'état immédiatement précédent, ce qui n'est pas suffisant pour faire apparaître du chaos. La figure 3.6 présente grossièrement le modèle du neurone chaotique, dont la fonction d'agrégation est :

$$\mathcal{A}(\mathbf{e}, kT) = w_0 + \sum_{i=1}^{n} w_i e_i + \sum_{d=1}^{k} w_d^c \Phi \circ \mathcal{A}(\mathbf{e}, (k-d)T). \tag{3.9}$$

[3] Le nombre d'associations qu'il est possible d'enregistrer dans un réseau de Hopfield constitué de n neurones est environ égal à $0{,}15n$ [Abu-Mostafa et St Jacques, 1985].

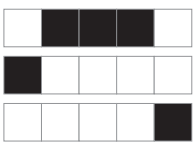


Figure 3.5
Trois imagettes dont la mémorisation par la règle de l'équation (3.5) donne des poids synaptiques très peu robustes.

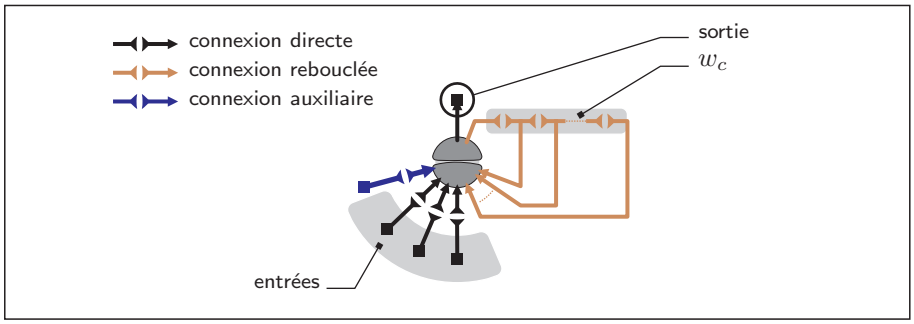


Figure 3.6
Schéma fonctionnel d'un neurone formel chaotique (chaque poids synaptique de bouclage introduit un retard).

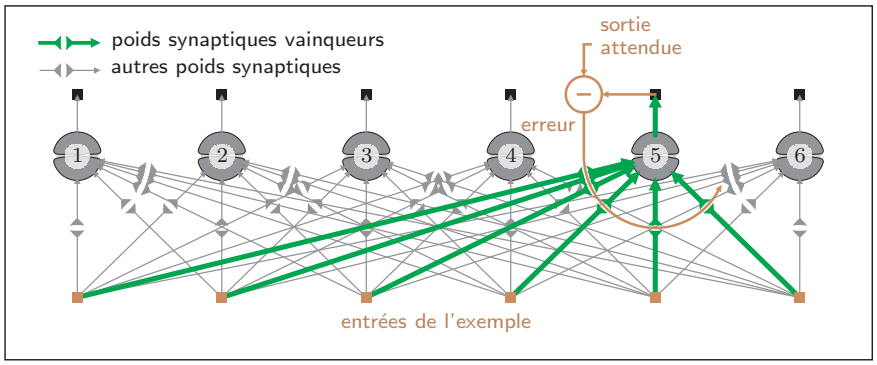
Outre les notations déjà utilisées dans le chapitre précédent, cette fonction d'agrégation introduit les poids synaptiques w_d^c qui pondèrent respectivement la sortie qu'avait le neurone chaotique d unités de temps plus tôt.

Outre les meilleures capacités de stockage affichées, ces réseaux de neurones artificiels permettent également de reconnaître des combinaisons linéaires de données enregistrées. Dans ce cas de figure, la réponse bascule périodiquement d'une donnée retrouvée à l'autre, plutôt que de se stabiliser sur une combinaison linéaire des deux, comme le font les perceptrons monocouches. En se basant sur ce principe, certaines recherches s'intéressent à un nouveau type de mémoire associative, capable d'associer plusieurs données à une seule donnée adresse, ou d'apprendre une nouvelle paire d'exemples si cette dernière est trop éloignée des données mémorisées [Arai et Osana, 2006].

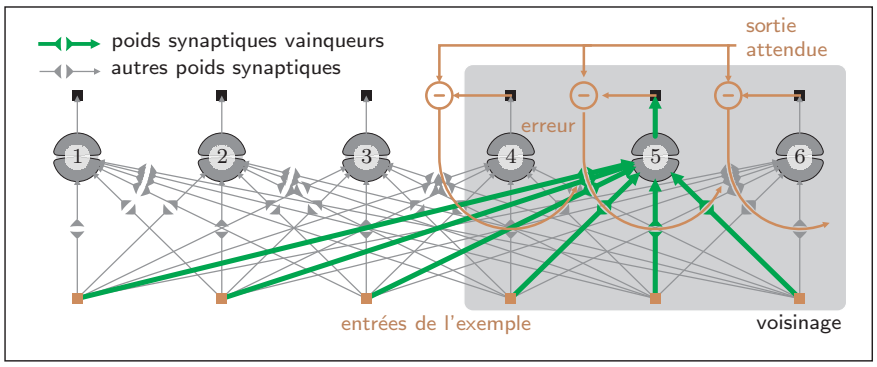
3.2 Les classifieurs

L'évolution progressive des connaissances relatives au cerveau biologique a prouvé que ce dernier était divisé en régions spécialisées et n'était par conséquent pas l'immense réseau de neurones monobloc polyvalent que l'on se représentait volontiers [Anderson et Rosenfeld, 1988]. Or, les mécanismes d'apprentissage utilisés jusqu'alors impliquaient des mises à jour globales des connexions ; un processus totalement incompatible avec le maintien de la localisation des traitements. Pour expliquer ce fait, il a donc fallu imaginer des procédés d'apprentissage d'un nouveau genre, c'est-à-dire permettant à une partie d'un réseau d'apprendre uniquement des exemples qui le concernent.

Cette technique a été nommée « apprentissage compétitif » dans la mesure où seuls les neurones artificiels sortis vainqueurs d'une « compétition » auront leurs poids synaptiques mis à jour par la présentation d'un exemple. Grâce à cette avancée, les réseaux de neurones artificiels peuvent maintenant présenter les mêmes propriétés de localisation que leurs pendants biologiques. Par exemple, s'agissant d'un problème de classification, un ensemble de neurones artificiels peut s'être spécialisé dans la



(a) *Learning Vector Quantization.*



(b) Carte auto-organisatrice.

Figure 3.7
Comparaison du mécanisme d'apprentissage du *Learning Vector Quantization* et des cartes auto-organisatrices.

reconnaissance d'une classe particulière, laissant à d'autres neurones la charge d'en reconnaître une autre, etc.

Learning Vector Quantization T. KOHONEN a proposé la règle d'apprentissage basée sur la compétition suivante : seul le neurone artificiel dont les poids synaptiques ressemblent le plus à l'entrée de l'exemple à apprendre, est mis à jour [KOHONEN, 1989]. Cette ressemblance est mesurée au moyen de la distance euclidienne entre les poids synaptiques de chaque neurone j et le vecteur d'entrée $\mathbf{e}$ de l'exemple présenté :

$$D_j^2 = \sum_i (w_{ij} - e_i)^2. \tag{3.10}$$

Le neurone qui minimise cette distance voit ses poids synaptiques mis à jour selon que sa sortie correspond à celle attendue pour l'exemple ($\Delta w_{ij} > 0$) ou non ($\Delta w_{ij} < 0$) :

$$\Delta w_{ij} = \pm\eta(e_i - w_{ij}). \tag{3.11}$$

La figure 3.7 illustre l'architecture des réseaux de neurones à apprentissage compétitif.

Les cartes auto-organisatrices de KOHONEN La règle du *learning vector quantization* est une règle d'apprentissage supervisé, c'est-à-dire qu'elle nécessite des exemples de la classification à apprendre. Néanmoins, il est parfaitement envisageable de laisser un réseau de neurones artificiel de ce type réaliser sa propre classification. Peuvent alors apparaître des différences pertinentes qui n'auraient pas forcément été incluses à la classification que nous aurions pu imposer.

Pour réussir cette opération, il est indispensable que le résultat ne soit pas trop dépendant de l'initialisation des poids synaptiques et que le réseau soit donc capable de se remodeler en cours d'apprentissage. C'est pourquoi T. KOHONEN a ajouté une notion de topologie aux réseaux de neurones artificiels, les dénommant maintenant « cartes ». Cette notion permet de définir un voisinage à chaque neurone du réseau, c'est-à-dire que la position relative des neurones est devenue une caractéristique importante du réseau (voir la figure 3.7b).

La règle d'apprentissage peut maintenant être modifiée afin de diminuer sa dépendance à l'initialisation : le neurone dont les poids sont les plus proches du vecteur d'entrée est toujours celui qui est mis à jour selon la règle de l'équation (3.11), mais le sont aussi, de façon moindre, les neurones de son proche voisinage.

3.3 Bilan sur les modèles de réseaux de neurones artificiels

Au travers de ce chapitre, il apparaît que les neurones formels introduits dans le chapitre 2 sont très largement à la base des architectures qui se montrent aujourd'hui les plus performantes. Autrement dit, les principales avancées dans le domaine des réseaux de neurones artificiels ont surtout concerné leur connectivité et les techniques de programmation, plus que les neurones eux-mêmes.

Il est intéressant de constater que la plupart des réseaux de neurones abordés dans ces deux derniers chapitres, mis à part le perceptron multicouche, n'utilisent qu'une seule couche de neurones artificiels pour calculer leur sortie. Il faut y voir la conséquence de l'inadéquation des outils mathématiques actuels dont nous avons déjà parlé. Pour voir se profiler un réel besoin de nouveaux outils, il aura fallu attendre l'arrivée d'un nouveau neurone formel, aux propriétés très différentes. Sans lui, la méthode de programmation auquel ont aboutis nos travaux, et que nous avons déjà utilisée par deux fois jusqu'ici, n'aurait pas pu voir le jour. Il s'agit des réseaux de neurones cellulaires [CHUA et YANG, 1988a] qui font l'objet du chapitre suivant.

4

Les réseaux de neurones cellulaires

Dès leur origine, les réseaux de neurones artificiels avaient vocation à être mis en œuvre matériellement, en témoigne la grande « boite en bois » que présenta F. Rosenblatt (voir la page 10). Malheureusement, la complexité des jeux de connexions impliqués par les modèles a rendu cette intégration délicate ; nos capacités de routage sont encore trop limitées pour réaliser les véritables enchevêtrements nécessaires à la réalisation d'un réseau constitué d'un grand nombre de neurones artificiels. Mis à part quelques tentatives d'implémentation couronnées de succès, les réseaux de neurones sont donc avant tout, aujourdhui, une architecture de calcul émulée.

Malgré tout, grâce à l'augmentation constante de la puissance de calcul des ordinateurs, les réseaux de neurones artificiels sont en train de franchir une nouvelle étape de leur développement : le chargement au sein de systèmes embarqués [Malasne et al., 2001, Toulouse, France]. En effet, les technologies de type asic, et plus particulièrement les fpgas, sont devenues tellement performantes qu'elles sont maintenant capables d'émuler un bon millier de neurones formels complètement interconnectés. Il s'agit d'un nombre suffisant pour y embarquer la plupart des réseaux de neurones artificiels développés, mais il faudra cependant passer à l'ordre de grandeur supérieur en ce qui concerne les réalisations les plus gourmandes.

Bien que cette voie soit la plus prometteuse à court terme, il est bien entendu préférable pour le long terme de s'abstraire complètement de l'émulation. Pour cela, il va falloir résoudre, ou du moins contourner, le problème du routage des connexions.

Dès 1988, L. O. Chua et L. Yang, en s'inspirant de travaux sur les automates cellulaires [Packard et Wolfram, 1985], suggérèrent qu'il n'était peut-être pas indispensable d'avoir un réseau de neurones artificiels complètement interconnecté pour réaliser une tâches arbitrairement compliquée. Ils mirent ainsi au point les réseaux de neurones cellulaires (cnns pour l'anglais *cellular neural networks*) à partir d'un nouveau modèle de neurone formel baptisé neurone « cellulaire » et dont une des particularités est d'avoir un nombre limité de connexions [Chua et Yang, 1988a,b]. Grâce à cette propriété, entre autres, les cnns sont à ce jour l'un des seuls modèles qui ait pu effectivement être intégré sans avoir à subir d'optimisation particulière vis-à-vis d'un traitement auquel il serait prédestiné. La puce la plus récente compte d'ailleurs 16 384 neurones, affiche une puissance de calcul équivalente à 330 millions d'opérations analogiques par seconde [Zarándy et Rekeczky, 2004a,b] et supporte l'utilisation combinée de plusieurs jeux de poids synaptiques.

Dans ce chapitre, nous présenterons en détails les réseaux de neurones cellulaires ainsi que leurs différents modes de fonctionnement.

4.1 Architecture d'un réseau de neurones cellulaires

4.1.1 Définition d'un neurone cellulaire

Contrairement aux autres neurones formels, la fonction d'agrégation et la fonction d'activation d'un neurone cellulaire sont directement définies à partir d'un circuit électronique, ce qui garantit ainsi le faible coût matériel de leur intégration. Le schéma de ce circuit est donné dans la figure 4.1 [Chua et Yang, 1988a,b].

Les grandeurs mises en jeu y sont donc analogiques et dépendent du temps, représenté par la variable t. L'équation qui régit le fonctionnement interne de ce neurone formel s'obtient par l'application de la loi de Kirchhoff au nœud A :

$$C\frac{dU_x}{dt}(t) + \frac{1}{R_x}U_x(t) = I + \sum_{i=1}^{n} I_{e_i}(t). \tag{4.1}$$

Dans cette équation, les n entrées du neurone sont les tensions intervennant au travers des sources de courant $I_{e_i}|_{i=1}^{n}$ et des poids synaptiques $w_i|_{i=1}^{n}$ de la façon suivante :

$$\forall i \in [\![1\,;n]\!]\,, \quad I_{e_i}(t) = w_i U_{e_i}(t). \tag{4.2}$$

La dynamique interne d'un neurone cellulaire est donc régie par l'équation différentielle suivante :

$$C\frac{dU_x}{dt}(t) = -\frac{1}{R_x}U_x(t) + I + \sum_{i=1}^{n} w_i U_{e_i}(t), \tag{4.3}$$

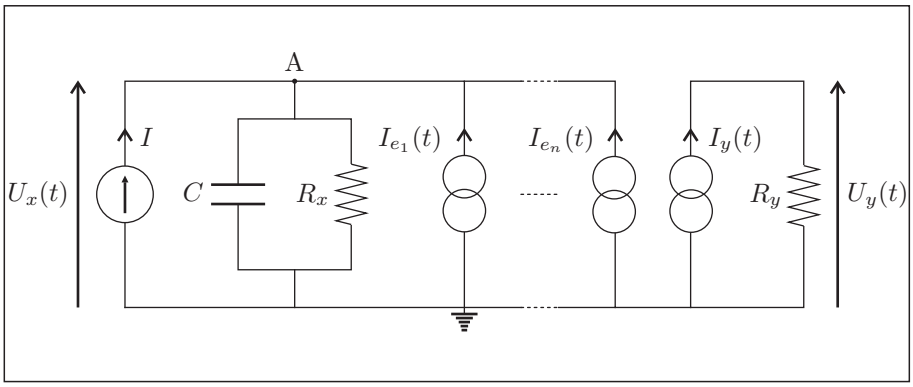


Figure 4.1
Schéma électronique d'un neurone cellulaire.

où la grandeur $U_x(t)$ caractérise l'état interne du neurone.

En admettant que les tensions d'entrées soient constantes tant que la tension interne ne s'est pas stabilisée, il est très facile de trouver le point d'équilibre de ce neurone formel. Il s'agit simplement de la valeur limite $U_x(t \to +\infty)$; ou plus simplement de la valeur de $U_x(t)$ qui annule la dérivée dans (4.3) :

$$\lim_{t \to +\infty} U_x(t) = R_x\big(I + \sum_{i=1}^{n} w_i U_{e_i}\big). \tag{4.4}$$

Cette valeur limite est traditionellement considérée comme le résultat de l'application de la fonction d'agrégation de ce neurone cellulaire aux entrées $U_{e_i}|_{i=1}^{n}$. Formellement, cette fonction s'écrit, avec les notations des chapitres précédents et en considérant que R vaut $1\,\Omega$:

$$\begin{aligned} \mathcal{A}\colon \mathbb{R}^n &\longrightarrow \mathbb{R} \\ \mathbf{e}(t) = \begin{pmatrix} e_1(t) & e_2(t) & \cdots & e_n(t) \end{pmatrix}^\top &\longmapsto w_0 + \mathbf{w}^\top \mathbf{e}(t). \end{aligned} \tag{4.5}$$

En d'autres termes, nous retrouvons la classique somme pondérée du neurone MP, à la différence que l'implémentation proposée est très économe, autant en ressources qu'en surface d'intégration.

Pour sa part, la fonction d'activation est mise en œuvre au moyen de la source de courant liée de l'étage de sortie du montage de la figure 4.1. Cette dernière est commandée en tension par la relation non-linéaire suivante, dont un tracé est donné dans la figure 4.2 :

$$U_y(t) = R_y I_y(t) = \frac{1}{2}\left(|U_x(t) + 1| - |U_x(t) - 1|\right). \tag{4.6}$$

À la lumière de cette expression, il apparaît que la tension de sortie $U_y(t)$ est bornée par les valeurs $-1\,\mathrm{V}$ et $+1\,\mathrm{V}$. La tension interne $U_x(t)$ peut quant à elle prendre

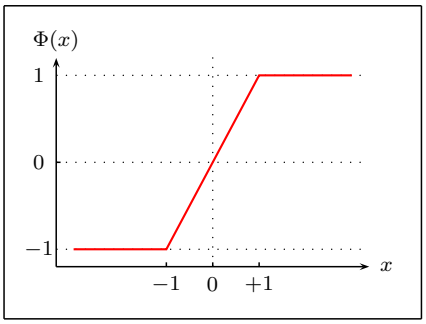


Figure 4.2
Fonction d'activation.

n'importe quelle valeur [1]. D'autre part, signalons que la convention [2] veut que les n tensions d'entrée soient également bornées par $-1\,\mathrm{V}$ et $+1\,\mathrm{V}$.

Afin de rester cohérent avec les notations utilisées jusqu'ici, et de s'abstraire des détails électroniques de ce modèle, les équations (4.3) et (4.6) sont ré-écrites de la façon suivante :

$$\frac{dx}{dt}(t) = -x(t) + w_0 + \mathbf{w}^\top \mathbf{e}(t) \quad \text{et} \quad y(t) = \frac{1}{2}\left(|x(t)+1| - |x(t)-1|\right). \tag{4.7}$$

4.1.2 Mise en réseau de neurones cellulaires

Les réseaux de neurones cellulaires sont traditionnellement monocouches [3], comme le sont les réseaux de HOPFIELD et les cartes auto-organisatrices. D'ailleurs, les astuces permettant de limiter le nombre de connexions d'un CNN sans trop en affecter les fonctionnalités, s'inspirent de ces deux modèles :

- les entrées et sorties d'un CNN sont celles des neurones qui le constituent ;
- les neurones cellulaires d'un CNN sont organisés selon une structure topologique qui permet de leur définir un voisinage dont ils font partie et avec lequel ils interagissent :
  - chaque neurone cellulaire partage son entrée et sa sortie avec ses voisins ;
  - il reçoit l'entrée et la sortie des neurones cellulaires de son voisinage.

La figure 4.3 résume ces caractéristiques à l'aide d'un CNN rudimentaire composé de quatre neurones cellulaires. La structure topologique choisie est monodimensionnelle et le voisinage d'un neurone celullaire regroupe, s'ils existent, les neurones situés immédiatement à sa droite et immédiatement à sa gauche. En règle générale, les CNNs sont utilisés comme architectures de traitement d'images et sont donc plutôt construits selon un maillage rectangulaire bidimensionnel. En fait, plusieurs dispositions sont bien-sûr possibles, parmis lesquelles celles de la figure 4.4 sont les plus classiques.

[1] En réalité, elle est bien évidemment bornée par les limites de fonctionnement des composants du circuit.

[2] Dans l'intégration matérielle proposée, ces tensions sont produites par des cellules CMOS.

[3] Il n'est cependant pas exclu d'imaginer travailler sur des versions multicouches bien qu'à notre connaissance, aucune recherche ne soit menée dans ce sens pour le moment.

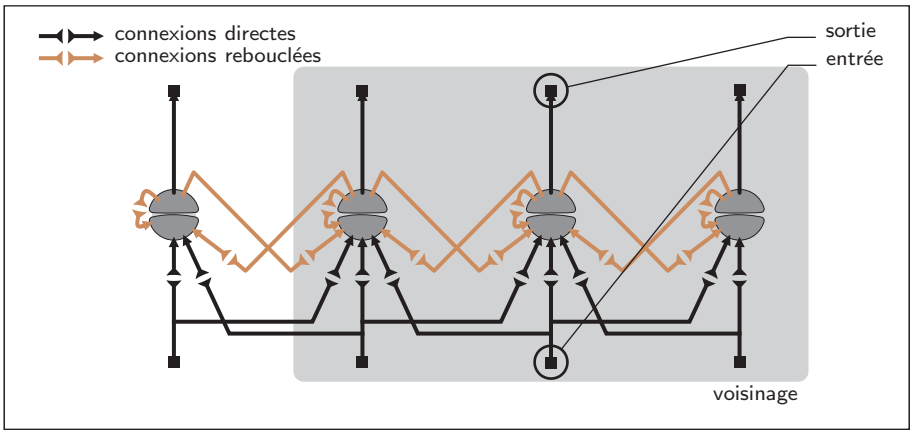
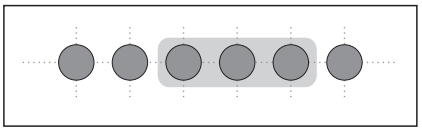
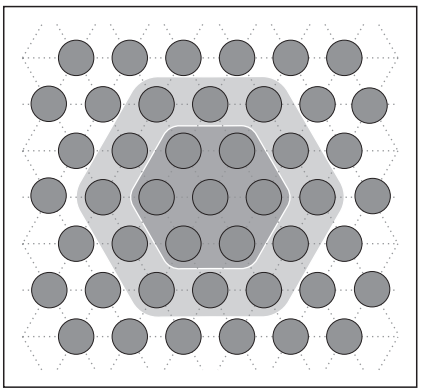


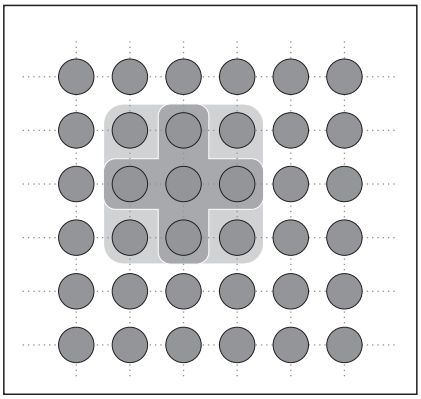
Figure 4.3
Schéma d'un réseau de neurones cellulaires composé de quatre neurones.



(a) En une dimension, avec un voisinage réduit aux deux neurones les plus proches.



(b) En deux dimensions, avec une maille hexagonale et un voisinage 6-connecté (foncé) et 18-connecté (clair).



(c) En deux dimensions, avec une maille carré et un voisinage 4-connecté (foncé) et 8-connecté (clair).

Figure 4.4
Les topologies et voisinages les plus classiques dans le domaine des CNNs.

En tenant compte des règles de mise en réseau qui viennent d'être énoncées, l'équation de fonctionnement interne (4.7) d'un neurone cellulaire devient donc, en appelant $\mathcal{N}$ l'ensemble des indices désignant ses neurones voisins, et en consirédant les entrées comme constantes :

$$\frac{dx}{dt}(t) = -x(t) + w_0 + \sum_{i \in \mathcal{N}} w_{e_i} e_i + \sum_{i \in \mathcal{N}} w_{y_i} y_i(t). \tag{4.8}$$

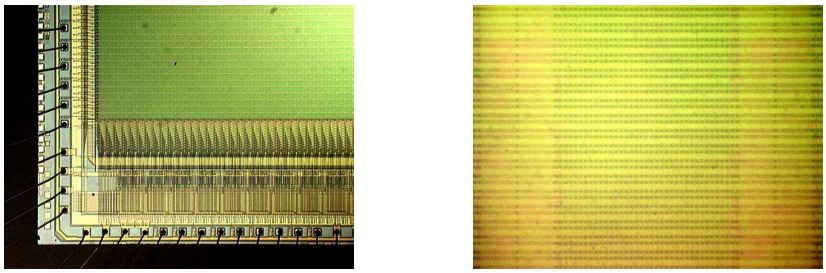
Dans cette équation, e_i et $y_i(t)$ sont respectivement l'entrée et la sortie du i^{e} neurone de $\mathcal{N}$, les coefficients w_{e_i} sont les poids synaptiques des entrées correspondantes et les coefficients w_{y_i} ceux des sorties. Ces derniers s'appellent également poids synaptiques de couplage, dans la mesure où leur présence rend le réseau dynamique et permet à une information de se propager sur tout le réseau malgré la localisation des connexions. Comme un neurone cellulaire est automatiquement contenu dans son propre voisinage, il vient qu'une des sorties rebouclées $y_i(t)$ de $\mathcal{N}$ correspond en fait à la propre sortie $y(t)$ du neurone. Dans la suite, nous serons amenés à traiter de façon différente les sorties provenant des neurones voisins de la sortie y du neurone. Nous utiliserons alors l'ensemble $\mathcal{N}^*$ pour désigner le voisinage strict d'un neurone et nous noterons w_y le poids synaptique de sa propre sortie.

Bien que ces hypothèses aient réduit de façon significative le nombre de connexions — un CNN de 64×64 neurones cellulaires, construit sur le maillage de la figure 4.4c, compte $77\,824$ connexions au lieu de $33\,558\,528$ s'il avait été complètement interconnecté — l'implémentation d'un CNN, en l'état, reste encore problématique. En effet, à chaque connexion correspond un poids synaptique réglable, qui nécessite donc un circuit électronique de contrôle supplémentaire. C'est la place occupée par ces derniers qui rend difficile l'intégration des connexions.

En conséquence, les instigateurs des CNNs ont proposé de n'utiliser que des copies d'un unique neurone cellulaire réglable ; c'est-à-dire que la modification d'un des poids synaptiques de ce dernier modifie également, et de façon identique, le poids synaptique correspondant de tous les autres neurones. De cette façon le nombre de circuits de contrôle, dans l'exemple précédent, est ramené à seulement 19 :
– neuf pour les neuf poids synaptiques des entrées du voisinage ;
– neuf pour les neuf poids synaptiques des sorties du voisinage ;
– un pour le biais des neurones.
La figure 4.5 montre une puce intégrant un réseau de neurones cellulaires de 64×64 neurones, réalisée par l'équipe espagnole de A. Rodríguez-Vázquez [Liñán et al., 2000].

4.2 Stabilité d'un réseau de neurones cellulaires

D'un point de vue pratique, programmer un réseau de neurones cellulaires consiste à déterminer ses 19 poids synaptiques, de telle sorte que tout vecteur d'entrée positionne les états internes du réseau là où ils convergeront sur les valeurs de sorties



(a) Matrice de 64×64 neurones cellulaires. (b) Détail d'un neurone cellulaire (grossissement : $1\,000\times$).

Figure 4.5
La puce ACE-4k.

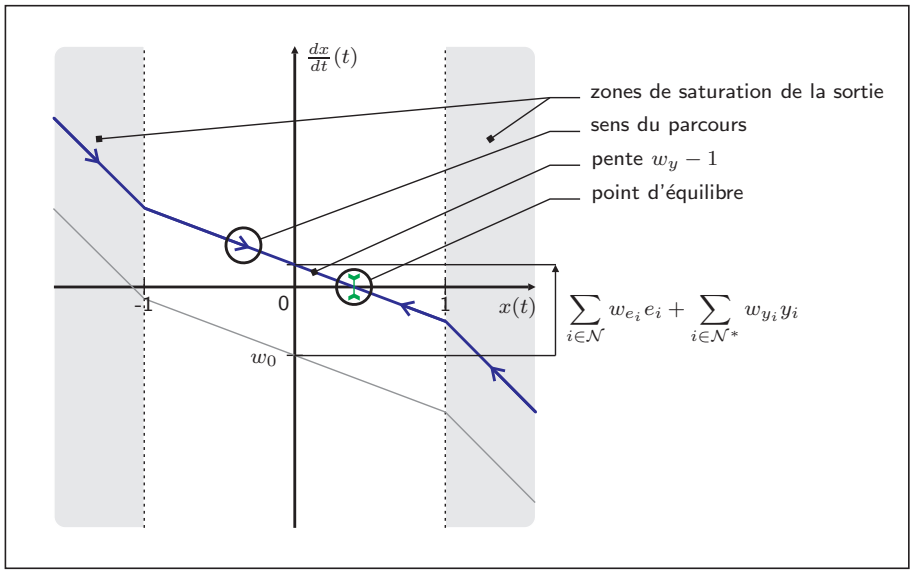


Figure 4.6
Allure générale d'une route dynamique suivie par un neurone cellulaire.

désirées. Naturellement, réaliser un tel positionnement suppose d'avoir identifié au préalable les états internes se stabilisant effectivement sur une valeur de sortie donnée. Dans la sous-section suivante, nous allons montrer qu'à chaque valeur de sortie stable correspond une infinité d'états initiaux s'y stabilisant. Ils sont distribués le long d'une trajectoire appelée « route dynamique » [Chua et Yang, 1988a ; Monnin, 2003 ; Bénédic et al., 2004 ; Bénédic et Mercklé, 2006a].

4.2.1 Routes dynamiques d'un neurone cellulaire

La figure 4.6 montre l'allure typique d'une route dynamique conduisant un neurone cellulaire quelconque à son état stable. Il s'agit du diagramme de phase d'un neurone cellulaire, c'est-à-dire de la représentation, dans le plan $(x(t), \frac{dx}{dt}(t))$, de son équation de fonctionnement interne (4.8), rappelée ci-dessous :

$$\frac{dx}{dt}(t) = -x(t) + w_0 + \sum_{i\in\mathcal{N}} w_{e_i} e_i + \sum_{i\in\mathcal{N}} w_{y_i} y_i(t). \qquad \text{(rappel de (4.8))}$$

En supposant que les poids synaptiques de couplage sont nuls ($w_{y_i} = 0|_{i\in\mathcal{N}}$) alors il s'agit d'une simple équation de droite, de pente -1 et d'ordonnée à l'origine $w_0 + \sum_{i\in\mathcal{N}} w_{e_i} e_i$.

Ce cas très particulier mis à part, les routes dynamiques suivent en général des droites brisées dont l'origine est la linéarité par morceaux de la fonction d'activation (4.7) (voir la figure 4.2 en page 36). En effet, les poids synaptiques de couplage ne sont pas forcément nuls, en particulier w_y qui pondère la propre sortie du neurone. En isolant ce dernier, l'équation (4.8) s'écrit donc également :

$$\frac{dx}{dt}(t) = -x(t) + w_y y(t) + w_0 + \sum_{i\in\mathcal{N}} w_{e_i} e_i + \sum_{i\in\mathcal{N}^*} w_{y_i} y_i(t). \tag{4.9}$$

En décomposant cette équation sur les trois domaines de linéarité de la fonction d'activation, on obtient les trois équations de droite suivantes : [Monnin, 2003]
– pour $x(t) \in]-\infty\,;-1]$:

$$\frac{dx}{dt}(t) = -x(t) - w_y + w_0 + \sum_{i\in\mathcal{N}} w_{e_i} e_i + \sum_{i\in\mathcal{N}^*} w_{y_i} y_i(t); \tag{4.10a}$$

– pour $x(t) \in [-1\,;+1]$:

$$\frac{dx}{dt}(t) = (w_y - 1)x(t) + w_0 + \sum_{i\in\mathcal{N}} w_{e_i} e_i + \sum_{i\in\mathcal{N}^*} w_{y_i} y_i(t); \tag{4.10b}$$

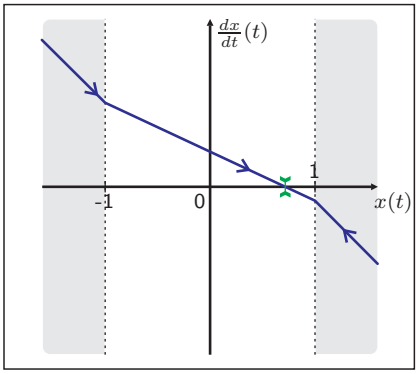
– et pour $x(t) \in [+1\,;+\infty[$:

$$\frac{dx}{dt}(t) = -x(t) + w_y + w_0 + \sum_{i\in\mathcal{N}} w_{e_i} e_i + \sum_{i\in\mathcal{N}^*} w_{y_i} y_i(t). \tag{4.10c}$$

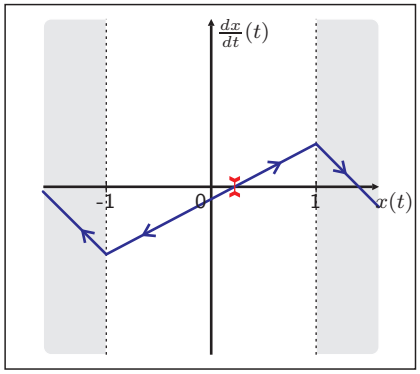
Avec des entrées constantes et des poids synaptiques de couplage nuls, à l'exception de w_y (on parle de neurone cellulaire « non-couplé »), ces trois équations correspondent bien aux trois segments de droites observés dans la figure 4.6.

4.2.2 Points d'équilibre d'un neurone cellulaire et d'un CNN

Les figures 4.8a et 4.8b montrent les deux allures typiques que peuvent prendre les routes dynamiques définies par le jeu d'équations (4.10) qui vient d'être établi. S'agissant de diagrammes de phase, le signe de la valeur représentée en ordonnées indique donc le sens dans lequel ces routes sont parcourues par l'état interne :
– si $\frac{dx}{dt}(t) > 0$ alors la route est parcourue dans le sens des états internes croissants ;
– si $\frac{dx}{dt}(t) < 0$ alors elle l'est dans le sens des états internes décroissants.
Les points d'équilibres sont donc tout naturellement les points d'intersection entre les routes dynamiques et l'axe des abscisses, c'est-à-dire là où la dérivée $\frac{dx}{dt}(t)$ s'annule. Ces derniers sont stables dans la mesure où la pente de la route dynamique y amène

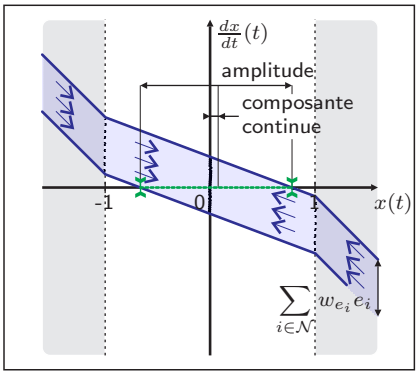


(a) Point d'équilibre stable.

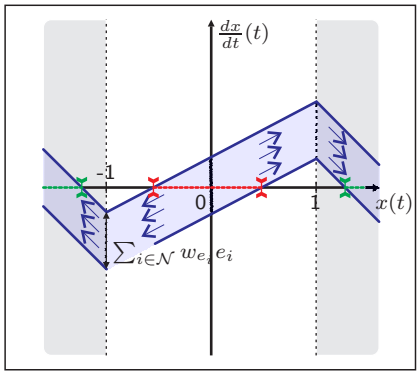


(b) Point d'équilibre instable.

Figure 4.7
Stabilité des points d'équilibre d'un neurone cellulaire.



(a) Routes dynamiques d'un neurone cellulaire fonctionnant en mode analogique.



(b) Routes dynamiques d'un neurone cellulaire fonctionnant en mode bistable.

Figure 4.8
Les deux modes de fonctionnement d'un CNN.

les états internes, c'est-à-dire si la dérivée y est négative. Ainsi, un état d'équilibre tel celui de la figure 4.7a est stable, contrairement à celui de la figure 4.7b qui est instable.

Un examen attentif du jeu d'équations (4.10) montre que les points d'équilibres stables sont dans l'intervalle de sortie $[\![-1\,;+1]\!]$ quand $w_y < 1$, car la pente du segment central est alors négative, et dans l'ensemble $\{-1\,;+1\}$ lorsque $w_y \geq 1$. Un neurone cellulaire non-couplé peut donc fonctionner selon deux modes [MONNIN et al., 1998 ; CHUA et YANG, 1988a] :

le mode analogique lorsque $w_y < 1$ (voir la figure 4.8a) ;

le mode bistable lorsque $w_y \geq 1$ (voir la figure 4.8b).

Ce double mode de fonctionnement ne peut pas s'étendre directement au cas du neurone cellulaire couplé. En effet, les routes dynamiques que nous venons d'établir supposent que les informations reçues par le neurone soient constantes. Or, si cette hypothèse est raisonnable en ce qui concerne les entrées $e_i(t)|_{i\in\mathcal{N}}$, elle est en revanche beaucoup plus discutable pour celles issues des sorties $y_i(t)|_{i\in\mathcal{N}*}$ avoisi-

nantes. Néanmoins, les routes dynamiques continuent malgré tout de rendre compte de façon efficace de l'évolution de l'état interne.

En fait, $x(t)$ suit une des routes dynamiques du cas non-couplé, et « saute » de l'une à l'autre au gré des variations de l'une des sorties du voisinage. Les points d'équilibre de ces réseaux ne sont ainsi pas très difficiles à trouver et beaucoup d'équipes ont apporté leur lot de critères suffisants à garantir leur existence pour un jeu de poids donné [Gilli et al., 2004 ; Takahashi et Chua, 1997, 1998 ; Kaluzny, 1994 ; Chua et Wu, 1992 ; Arik et Tavsanoglu, 1996 ; Bénédic et Mercklé, 2006a]. Néanmoins, ces résultats ne garantissent pas pour autant la convergence d'un cnn, si bien que de façon générale, la stabilité d'un réseau de neurones cellulaires couplés n'a pas encore été démontrée, bien qu'à ce jour aucun cnn non stable n'ait encore été trouvé.

4.3 Programmation des cnns

Les toutes premières applications programmées sur un réseau de neurones cellulaires avaient surtout pour objectif d'illustrer le potentiel de cette architecture, et d'inciter la communauté scientifique à ne pas s'arrêter à la connectivité apparemment réductrice du modèle. Pour cette raison, ces applications étaient principalement le fruit de méthodes empiriques, autant basées sur des intuitions [Matsumoto et al., 1990] que sur des analyses rigoureuses [Chua et Yang, 1988b ; Kozek et al., 1993 ; Hänggi et Moschytz, 1998]. Parmi ces applications pionnières, on trouve des opérateurs de morphologie mathématique [Zarándy et al., 1996], de classification de textures [Kellner et al., 1994], ou encore la réalisation de mémoires associatives [Tan et al., 1990].

À cette époque, chaque nouvelle application nécessitait de nouveaux trésors d'inventivité, ce qui a sans aucun doute été à l'origine de la réputation des cnns : une architecture prometteuse, mais « impossible » à programmer. Malgré tout, grâce à l'accumulation de quinze années d'expérience, ce qui était perçu comme des intuitions a fini par trouver des justifications analytiques et a finalement conduit aux premières méthodes de programmation analytique pour cnns [Merlat, 1998 ; Zarándy, 1999 ; Monnin, 2003 ; Hänggi et Moschytz, 1999]. Les sous-sections suivantes présentent les principes clés qui ont permis à ces méthodes de se développer.

4.3.1 Le mode analogique

Le mode analogique correspond au cas $w_y < 1$. Curieusement, ce mode n'a été traité que très tard dans l'histoire des cnns, à la fin des années quatre-vingt-dix par D. Monnin [Monnin et al., 1998]. Il s'agit d'une étude du cas non-couplé, c'est-à-dire $w_{y_i} = 0|_{i \in \mathcal{N}^*}$, qui montre que les traitements effectués peuvent être globalement assimilés à des filtrages linéaires par convolution de l'information placée en entrée.

Dans ce mode, tous les points d'équilibre sont stables (voir la figure 4.8a). Leur expression en fonction des entrées et des poids synaptiques s'obtient en annulant la dérivée des équations (4.10), ce qui conduit à la fonction d'agrégation suivante :

$$\begin{aligned} &\mathcal{A}\colon [-1\,;+1]^n \longrightarrow \mathbb{R} \\ &e_1, e_2, \ldots, e_n \longmapsto \frac{1}{1-w_y}\Big(w_0 + \sum_{i\in\mathcal{N}} w_{e_i} e_i\Big). \end{aligned} \tag{4.11}$$

En identifiant la somme pondérée à un produit de convolution entre un masque, encodé par les poids synaptiques $w_{e_i}|_{i\in\mathcal{N}}$, et les entrées $e_i|_{i\in\mathcal{N}}$, on retrouve l'analyse proposée dans [Monnin et al., 1998]. Les paramètres w_0 et w_y agissent respectivement sur la composante continue et l'amplitude des valeurs obtenues en sortie [4].

Exemple 4.1 — Détection de contours avec un CNN.

La figure 4.9 montre le résultat d'une détection de contours utilisant le masque de convolution suivant :

$$\mathcal{H} = \frac{1}{8}\begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix}.$$

Les poids synaptiques du CNN s'obtiennent alors à partir de l'équation (4.11). Le choix de w_0 et w_y positionne l'histogramme de l'image obtenue de sorte qu'elle utilise pleinement la plage $[-1\,;+1]$, c'est-à-dire qu'aucun neurone cellulaire ne doit converger dans $]-\infty\,;-1[\,\cup\,]+1\,;+\infty[$ qui correspond à la zone de saturation de la fonction d'activation [Monnin et al., 1998] :

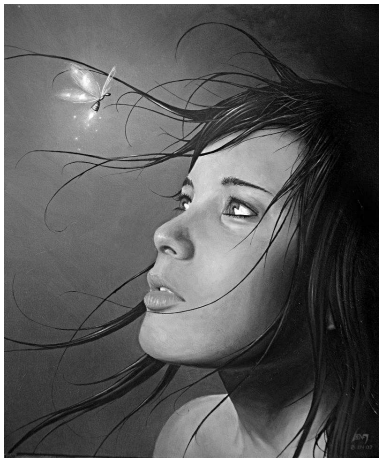
$$\big(w_{y_i}|_{i\in\mathcal{N}}\big) = \begin{pmatrix} 0 & 0 & 0 \\ 0 & -15 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \quad \big(w_{e_i}|_{i\in\mathcal{N}}\big) = \frac{1}{8}\begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix} \quad \text{et} \quad w_0 = 0.$$

Le cas couplé, dont la stabilité est encore un problème ouvert à ce jour, est très mal maitrisé par la communauté scientifique. Certaines recherches, parmi les plus pointues dans le domaine, ont néanmoins réussi à programmer quelques traitements ayant recours à ce mode :
– l'interpolation d'une surface 3D [Tóth, 1995] (voir la figure 4.10) ;
– La détection de textures [Sziranyi et Csapodi, 1998] (voir la figure 4.11),
respectivement implémentés avec les jeux de poids (4.12) et (4.13) :

$$\big(w_{y_i}|_{i\in\mathcal{N}}\big) = \begin{pmatrix} 0 & 0 & -2 & 0 & 0 \\ 0 & -4 & 16 & -4 & 0 \\ -2 & 16 & -39 & 16 & -2 \\ 0 & -4 & 16 & -4 & 0 \\ 0 & 0 & -2 & 0 & 0 \end{pmatrix}, \qquad \big(w_{e_i}|_{i\in\mathcal{N}}\big) = \begin{pmatrix} 0&0&0&0&0 \\ 0&0&0&0&0 \\ 0&0&0&0&0 \\ 0&0&0&0&0 \\ 0&0&0&0&0 \end{pmatrix}, \tag{4.12}$$

$$\text{et} \quad w_0 = 0.$$

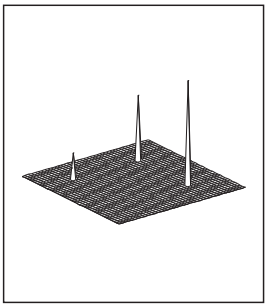
[4] Dans le cas du traitement d'images, w_0 règle la luminosité et w_y le contraste.



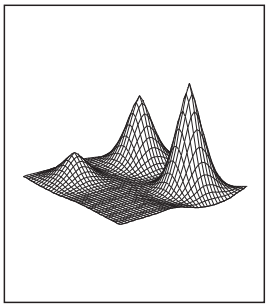
(a) Image originale.

(b) Image traitée.

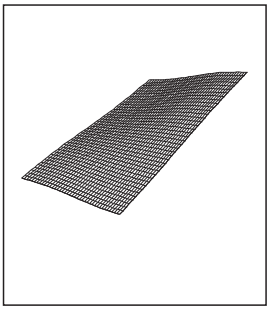
Figure 4.9
Détection de contours par le filtre de Laplace.



(a) Les trois échantillons, avec lesquels la sortie est initialisée, supportent la surface à interpoler.



(b) Valeur de la sortie en cours de calcul.



(c) Résultat de l'interpolation.

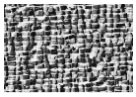
Figure 4.10
Interpolation d'une surface par un CNN 80×80.

$$\left(w_{y_i}|_{i\in\mathcal{N}}\right) = \begin{pmatrix} 2{,}27 & 1{,}80 & 3{,}36 \\ -0{,}70 & -4{,}45 & 1{,}41 \\ 3{,}2 & 0{,}98 & -0{,}31 \end{pmatrix}, \quad \left(w_{e_i}|_{i\in\mathcal{N}}\right) = \begin{pmatrix} -3{,}91 & 1{,}25 & 3{,}05 \\ 0{,}86 & -3{,}05 & 3{,}36 \\ 1{,}72 & -0{,}63 & -4{,}61 \end{pmatrix}, \quad (4{\cdot}13)$$

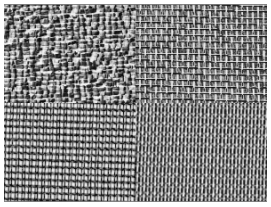
$$\text{et} \quad w_0 = -1{,}64.$$

4.3.2 Le mode bistable

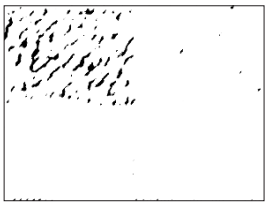
Contrairement au mode analogique, le mode bistable a généré énormément de publications dès les débuts des réseaux de neurones cellulaires [Analogic & Neural Computing Laboratory, 2005 ; Bénédic et Monnin, 2003]. Dans ce mode, les points d'équilibres stables sont situés dans $]-\infty\,;-1[\cup]+1\,;+\infty[$, c'est-à-dire là où la fonction d'activation sature. En conséquence, la sortie $y(t \to +\infty)$ va ou bien se stabiliser en $+1$, ou bien en -1. Il existe donc une ou plusieurs routes dynamiques frontières,



(a) Texture à détecter.



(b) Image de test contenant quatre textures différentes.



(c) La portion d'image de moyenne non nulle correspond bien à la texture recherchée.

Figure 4.11
Détection d'une texture dans une image.

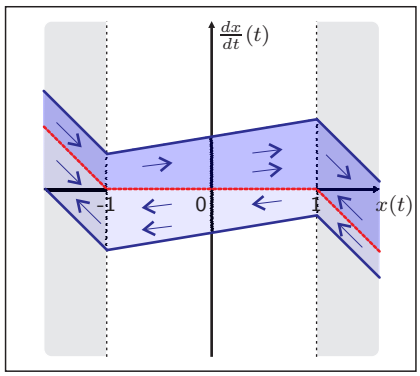


Figure 4.12
Route dynamique frontière d'un neurone cellulaire en mode bistable.

qui séparent les routes conduisant à une stabilisation en -1 de celles conduisant en $+1$. Cette frontière est très simple à déterminer graphiquement (voir la figure 4.12) et son expression est donnée par les trois équations de droite suivantes :

$$\text{– pour } x(t) \in]-\infty\,;-1[: \quad \frac{dx}{dt}(t) = -x(t) - 1; \tag{4.14a}$$

$$\text{– pour } x(t) \in [-1\,;+1]: \quad \frac{dx}{dt}(t) = 0; \tag{4.14b}$$

$$\text{– pour } x(t) \in]+1\,;+\infty[: \quad \frac{dx}{dt}(t) = -x(t) + 1. \tag{4.14c}$$

D'après les équations (4.10), la sortie d'un neurone cellulaire non couplé convergera donc en $+1$ si et seulement si :

$$\text{– pour } x(t) \in]-\infty\,;-1]: \quad -w_y + 1 + w_0 + \sum_{i \in \mathcal{N}} w_{e_i} e_i > 0; \tag{4.15a}$$

$$\text{– pour } x(t) \in [-1\,;+1]: \quad (w_y - 1)x(t) + w_0 + \sum_{i \in \mathcal{N}} w_{e_i} e_i > 0; \tag{4.15b}$$

$$\text{– pour } x(t) \in [+1\,;+\infty[: \quad w_y - 1 + w_0 + \sum_{i \in \mathcal{N}} w_{e_i} e_i > 0. \tag{4.15c}$$

En définitive, un réseau de neurones cellulaires non couplé fonctionnant en mode bistable réalise également un produit de convolution entre un masque codé par les poids synaptiques du réseau et une information placée en entrée. Cependant, à la différence du mode analogique, le résultat de ce produit est maintenant seuillé, comme le montre le jeu d'équations (4.15). Ainsi, ce type de CNNs a un fonctionnement pratiquement identique à celui des modèles présentés dans les chapitres précédents. Comme ces derniers, un CNN non couplé dont les entrées sont binaires évalue donc une fonction logique à seuil (voir le chapitre 7 à la page 67).

Par conséquent, la programmation d'un tel CNN consiste essentiellement à trouver les poids synaptiques qui coderont correctement la fonction logique à seuil désirée [Monnin et al., 2000 ; Hänggi et Moschytz, 1999]. Il s'agit d'une question difficile qui est à l'origine des travaux exposés dans ce manuscrit et sur laquelle nous reviendrons dans la section suivante.

Exemple 4.2 — Opérations de morphologie mathématique avec un CNN.

Les traitements d'images par morphologie mathématique incluent le filtrage, la segmentation, la quantification et la modélisation d'images [Serra, 1982]. Ils sont basés sur deux opérations de base — l'érosion et la dilatation — qui peuvent se définir, dans le cadre d'images binaires, comme l'application des fonctions logiques suivantes à chaque voisinage $p_i|_{i\in\mathcal{N}}$ de pixels de l'image [Zarándy et al., 1996] :

érosion elle est réalisée par le produit logique suivant : $\prod_{i\in\mathcal{N}} p_i$;

dilatation elle est le résulat de la somme logique suivante : $\sum_{i\in\mathcal{N}} p_i$.

Dans ce contexte, le voisinage est nommé « élément structurant ».

Les poids synaptiques permettant de programmer un CNN pour l'érosion s'obtiennent très facilement par l'observation des équations (4.15). En remarquant qu'un produit logique est vrai quand toutes ses variables sont vraies, alors si toutes les entrées du voisinage sont pondérées par la valeur 1, il suffit de positionner le seuil suffisamment haut. En supposant que l'état interne de chaque neurone cellulaire est initialisé à zéro ($x(t=0)=0$), alors ce seuil est simplement réglé par $-w_0$. Pour un voisinage 3×3, la somme pondérée varie entre -9 et $+9$ par incréments de deux unités, ce qui permet de déduire que le seuil doit être compris dans l'intervalle $]7\,;9[$. Les poids synaptiques de l'érosion sont donc :

$$\left(w_{y_i}|_{i\in\mathcal{N}}\right) = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \qquad \left(w_{e_i}|_{i\in\mathcal{N}}\right) = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \qquad \text{et} \qquad w_0 = -8.$$

De la même façon, la dilatation s'écrit à l'aide d'un seuil suffisamment bas — dans l'intervalle $]-9\,;-7[$ — pour que la somme pondérée le dépasse dès qu'une entrée est vraie :

$$\left(w_{y_i}|_{i\in\mathcal{N}}\right) = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \qquad \left(w_{e_i}|_{i\in\mathcal{N}}\right) = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \qquad \text{et} \qquad w_0 = 8.$$

La figure 4.13 présente des exemples de traitements d'images par érosion et par dilatation.

(a) Image originale. (b) Résultat de l'érosion. (c) Résultat de la dilatation.

Figure 4.13
Morphologie mathématique réalisée par un CNN en mode bistable.

Pour sa part, le cas couplé n'est pas très différent du cas non couplé. Bien que cela n'ait encore fait l'objet d'aucune preuve rigoureuse, la plupart des traitements couplés en mode bistable implémentent une fonction logique à seuil qui est itérée jusqu'à stabilisation des sorties [Monnin et al., 2000 ; Bénédic et al., 2004 ; Bénédic et Mercklé, 2006a]. En suivant ce principe, un jeu de poids permettant à un CNN de résoudre un labyrinthe a été obtenu [Nakai et Ushida, 1995]. Un exemple d'utilisation est donné dans la figure 4.14, et les poids synaptiques correspondants sont :

$$\left(w_{y_i}|_{i\in\mathcal{N}}\right) = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 0 \end{pmatrix}, \qquad \left(w_{e_i}|_{i\in\mathcal{N}}\right) = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \qquad \text{et} \qquad w_0 = 2.$$

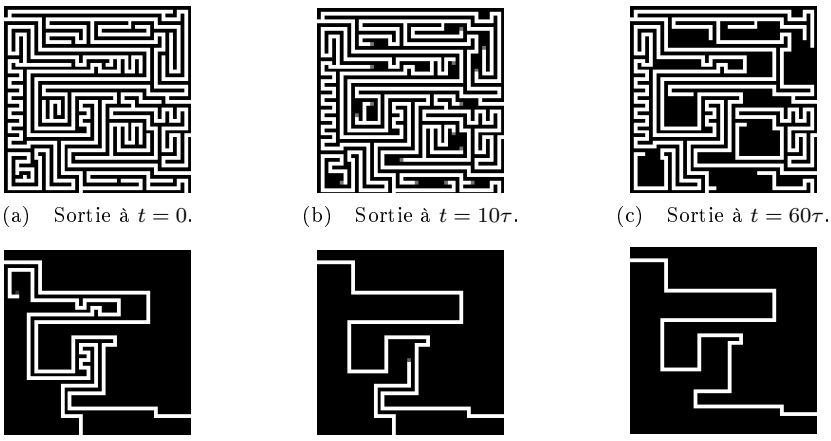
Néanmoins, un CNN couplé en mode bistable peut réaliser des traitements très compliqués, comme par exemple la conversion d'une image en niveaux de gris en une image demi-teintes [Crounse et al., 1993]. Plusieurs implémentations ont été mises au point, dont nous avons retenu la plus concise :

$$\left(w_{y_i}|_{i\in\mathcal{N}}\right) = \begin{pmatrix} -0{,}07 & -0{,}1 & -0{,}07 \\ -0{,}1 & 1{,}05 & -0{,}1 \\ -0{,}07 & -0{,}1 & -0{,}07 \end{pmatrix}, \quad \left(w_{e_i}|_{i\in\mathcal{N}}\right) = \begin{pmatrix} 0{,}07 & 0{,}1 & 0{,}07 \\ 0{,}1 & 0{,}32 & 0{,}1 \\ 0{,}07 & 0{,}1 & 0{,}07 \end{pmatrix} \quad \text{et} \quad w_0 = 0.$$

Un exemple d'application de cette opération est donnée dans la figure 4.15.

4.4 Avantage des méthodes analytiques

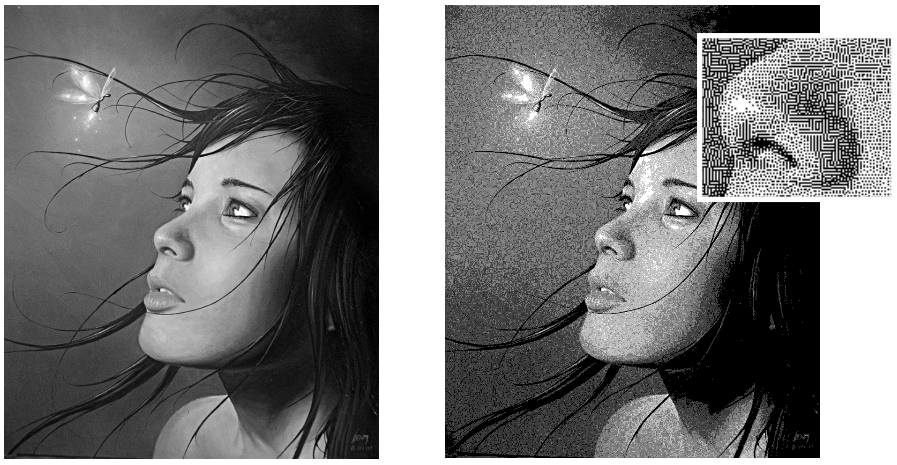
Le modèle des réseaux de neurones cellulaires a été développé avec son intégrabilité pour principal préoccupation. C'est d'ailleurs ce qui a conduit ses instigateurs à le définir à partir d'un circuit électronique. En contrepartie, les signaux internes aux neurones cellulaires sont de nature analogique, c'est-à-dire qu'ils sont sujet au bruit, même lorsque le réseau fonctionne en mode bistable. En pratique, cela signifie qu'un signal théoriquement binaire, représenté par une tension de $\pm 1\,\mathrm{V}$, vaudra en réalité $\pm 1\,\mathrm{V} \pm 10\,\%$ si le niveau de bruit est de dix pourcent, ce qui est environ la valeur maximale que nous avons constatée [Bénédic et Monnin, 2003]. Malheureusement, les applications développées pour le CNN en tiennent très peu compte, si bien qu'un



(a) Sortie à $t = 0$. (b) Sortie à $t = 10\tau$. (c) Sortie à $t = 60\tau$.

(d) Sortie à $t = 500\tau$. (e) Sortie à $t = 900\tau$. (f) Sortie finale.

Figure 4.14
Itération d'une fonction logique à seuil pour résoudre un labyrinthe. (τ désigne la durée d'un pas de calcul.)



(a) Image originale en niveaux de gris. (b) Image obtenue.

Figure 4.15
Conversion en demi-teintes d'une image en niveaux de gris à l'aide d'un CNN couplé en mode bistable.

jeu de poids synaptiques parfaitement fonctionnel dans l'environnement de simulation où il a été développé ne se comporte plus du tout comme prévu une fois embarqué sur une des puces du « marché ».

Il s'agit d'une problématique complètement nouvelle pour le domaine des réseaux de neurones artificiels, qui s'est jusque là toujours appuyé sur des environnements de calcul numériques : émulations informatiques ou réalisations embarquées sur FPGA. Une notion de qualité a alors été introduite pour caractériser l'aptitude d'un jeu de poids synaptiques à réaliser ce pourquoi il a été conçu malgré la présence de bruit déformant les signaux sur lesquels il travaille. Ainsi, deux jeux de poids réalisant la même opération peuvent réagir complètement différemment si leurs entrées deviennent bruités : l'un, par exemple, continuera de fonctionner correctement tandis que l'autre deviendra inopérant (voir l'exemple 4.3 à la fin de ce chapitre).

Ce besoin particulier a fait naître une profusion de méthodes de programmation analytiques propres aux CNNs [Merlat, 1998 ; Hänggi et Moschytz, 1999 ; Zarándy, 1999 ; Monnin, 2003 ; Bénédic, 2002]. C'est en effet le moyen le plus direct pour quantifier cette notion de qualité et surtout pour optimiser un jeu de poids synaptiques selon ce critère. C'est dans ce cadre que les travaux présentés dans ce mémoire ont débuté.

Exemple 4.3 — Comparaison de deux implémentations différentes d'une même opération.

En 2004 a été publiée une banque de 1 882 jeux de poids synaptiques permettant à un CNN non couplé de réaliser toutes les fonctions logiques de quatre variables possible[5] [Chen et Chen, 2005, 2004; Chen et al., 2005]. Le jeu de poids n°972, permettant de réaliser la fonction logique $\overline{e}_7(\overline{e}_9 + e_2\overline{e}_5)$, est donné comme étant :

$$\left(w_{y_i}|_{i\in\mathcal{N}}\right) = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \quad \left(w_{e_i}|_{i\in\mathcal{N}}\right) = \begin{pmatrix} 0 & 1 & 0 \\ 0 & -2 & 0 \\ -4 & 0 & -3 \end{pmatrix} \quad \text{et} \quad w_0 = -3.$$

La méthode de programmation présentée à la fin de ce manuscrit donne quant à elle :

$$\left(w_{y_i}{}'|_{i\in\mathcal{N}}\right) = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \quad \left(w_{e_i}{}'|_{i\in\mathcal{N}}\right) = \begin{pmatrix} 0 & 1 & 0 \\ 0 & -1 & 0 \\ -3 & 0 & -2 \end{pmatrix} \quad \text{et} \quad w_0{}' = -2.$$

Appliquons le vecteur d'entrée suivant à deux neurones cellulaires respectivement programmés avec ces deux jeux de poids synaptiques :

$$\mathbf{e}^{(10)} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & -1 & 0 \\ 1 & 0 & -1 \end{pmatrix}.$$

[5] C'est le nombre de fonctions logiques de quatre variables linéairement séparables.

Naturellement, la sortie des deux neurones se stabilise sur la valeur -1 puisque :

$$w_0 + \sum_{i \in \mathcal{N}} w_{e_i} e_i = -3 + 1 + 2 - 4 + 3 = -1 < 0,$$
$$w_0{}' + \sum_{i \in \mathcal{N}} w_{e_i}{}' e_i = -2 + 1 + 1 - 3 + 2 = -1 < 0.$$

Appliquons leur maintenant le même vecteur d'entrée, auquel du bruit a été ajouté :

$$\mathbf{e}^{(10)\,\prime} = \begin{pmatrix} 0 & 1{,}05 & 0 \\ 0 & -1{,}1 & 0 \\ 0{,}9 & 0 & -1{,}15 \end{pmatrix}.$$

La sortie du neurone cellulaire programmé avec le premier jeu de poids synaptiques est maintenant erronée tandis que la sortie du second reste correcte.

$$w_0 + \sum_{i \in \mathcal{N}} w_{e_i} e_i{}' = -3 + 1{,}05 + 2{,}2 - 3{,}6 + 3{,}45 = 0{,}1 > 0,$$
$$w_0{}' + \sum_{i \in \mathcal{N}} w_{e_i}{}' e_i{}' = -2 + 1{,}05 + 1{,}1 - 2{,}7 + 2{,}30 = -0{,}25 < 0.$$

Bien que ces deux jeux de poids synaptiques réalisent la même fonction logique, le notre a donc une meilleure qualité que le premier, et supportera vraisemblablement bien mieux d'être utilisé sur une implémentation matérielle d'un CNN.

5

Bilan sur les réseaux de neurones artificiels

Dans cette première partie, les principaux modèles de neurones formels ont été présentés : neurone MP, ADALINE, perceptron, neurone chaotique, neurone cellulaire. Il en existe naturellement beaucoup d'autres, et les présenter de façon exhaustive demanderait d'y consacrer plusieurs ouvrages [Dreyfus et al., 2002 ; Fausett, 1994 ; Haykin, 2003]. D'après leurs définitions respectives, il apparaît que la majorité d'entre-eux est défini autour des mêmes fonctions d'agrégation et d'activation : une somme pondérée seuillée. Ainsi, la différence majeure entre ces modèles ne réside pas dans les neurones eux-mêmes, qui sont donc globalement fonctionnellement identiques, mais dans la façon dont ils sont mis en réseau, ainsi que la méthode d'apprentissage qui en découle.

Ce sont d'ailleurs de ces deux caractéristiques que les réseaux de neurones artificiels tirent leur principal atout : celui d'être capable d'apprendre une opération à partir d'une illustration de ses effets sur un jeu d'exemples. Toutefois, aussi souples et efficaces que ces techniques de programmation puissent être, elles ont l'inconvénient de transformer un réseau de neurones artificiels en boite noire, dont on ignore le rôle

exact de ses constituants. Malheureusement, cette méconnaissance freine considérablement leur développement applicatif. En effet, elle impose énormément de choix arbitraires :

- combien de neurones artificiels utiliser ?
- doivent-ils être différents ?
- le réseau doit-il être complètement interconnecté ?
- faut-il des connexions de couplage ?
- combien de couches de neurones sont-elles nécessaires ?
- quelle dynamique de calcul est nécessaire ?

À l'heure actuelle, seules l'expérience et l'intuition apportent un semblant de réponses à ces questions. S'il était possible d'apporter des réponses rigoureuses à certaines d'entre-elles, nul doute que cela améliorerait significativement les temps de convergence des techniques d'apprentissage et la qualité des réseaux obtenus.

Dans ce décor, les réseaux de neurones cellulaires sont arrivés comme une innovation clef. En effet, leur structure rigide a éliminé la plupart de ces questions, mais elle a aussi rendue inefficace les nombreux algorithmes d'apprentissage développés pour les autres modèles, comme en témoigne l'étude de D. Monnin [Monnin, 2003]. En contrepartie, la recherche a donc dû se focaliser sur des techniques de programmation analytiques, développées à partir des nos connaissances du fonctionnement interne des neurones cellulaires.

L'introduction des CNNs a également eu un autre impact important sur la thématique des recherches menées à son sujet, et que nous jugeons toutes autant capitales. Il s'agit de l'introduction d'un critère de robustesse des jeux de poids synaptiques utilisés, vis-à-vis du bruit de fonctionnement inhérent aux réalisations matérielles dont les CNNs font l'objet. Il s'agit d'une contrainte typique de ces réseaux, que les techniques de programmation analytique essaient d'intégrer [Monnin, 2003 ; Hänggi et Moschytz, 1999 ; Bénédic, 2002].

Les CNNs se démarquent donc des autres modèles de réseaux de neurones artificiels par le fait qu'ils se programment de façon analytique, c'est-à-dire à partir d'un algorithme précis de l'opération à implémenter, et non à partir d'un jeu d'exemples de ce dernier. De plus, les jeux de poids obtenus ne sont pas égaux en robustesse, ce qui amène à réfléchir à des méthodes capables de les optimiser en conséquence.

Dans la prochaine partie de ce manuscrit, nous verrons que l'application à d'autres réseaux, de ce type d'approche analytique, apporte un nouvel éclairage sur les questions que nous venons de soulever sur le paramétrage des apprentissages. Ce sera par exemple le cas de la dynamique de calcul minimale d'un réseau pour qu'il puisse réaliser une opération donnée, ou encore son nombre de neurones. Toutefois, l'intérêt des méthodes d'apprentissage étant de ne pas nécessiter la connaissance formelle de l'opération à traiter, ce qui est une condition nécessaire à l'utilisation des méthodes de programmation analytiques.

Aussi, nous proposons une méthode d'analyse dont l'objectif est d'exprimer formellement l'opération apprise par tout réseau de neurones artificiel dont le fonctionnement serait basé sur des sommes pondérées seuillées. Étant donné la complexité de ces opérations, cela nous amènera à développer un formalisme dédié dont nous montrerons qu'il peut servir de base aux méthodes de programmation analytiques existantes. Dans la dernière partie, nous verrons enfin comment de nouvelles méthodes de programmation peuvent être imaginées. Leur principe sera de synthétiser en neurones artificiels, les expressions formelles décrivant les opérations à implémenter. À l'instar des exemples de cette partie, les algorithmes qui en découlent seront capables de « synthétiser » un réseau de neurones aux caractéristiques optimisées, à partir de n'importe quel réseau obtenu par apprentissage.

Deuxième partie

Analyse d'un neurone artificiel

Sommaire de la partie

6

Les neurones artificiels binaires

L'introduction des réseaux de neurones cellulaires a initié une nouvelle campagne de recherches ciblée autour des méthodes de programmation analytiques des neurones cellulaires en mode bistable ; en particulier ceux dont les entrées sont de nature binaire. Bien qu'ils puissent paraître très spécialisés à première vue, les résultats obtenus dans ce domaine peuvent néanmoins être transposés à d'autres modèles de neurones artificiels, pourvu qu'ils soient dotés d'une fonction d'agrégation linéaire, d'une fonction d'activation à sortie binaire [1] et que leurs entrées soient quantifiées. Cette affirmation sera démontrée dans la deuxième section de ce chapitre.

Les perspectives ouvertes par ces méthodes de programmation analytiques, en comparaison avec les techniques basées sur des apprentissages, ont été discutées dans le chapitre précédent. À ce jour, les seules méthodes de ce type existantes sont essentiellement fondées sur l'analyse des mécanismes internes des CNNs, ce qui conduit à une vision du problème peut-être un peu incomplète pour amener des progrès significatifs. C'est la raison pour laquelle nous reposerons notre propre méthode de programmation analytique sur des principes beaucoup plus généraux, ayant trait aux neurones

[1] Il s'agira le plus souvent d'une fonction échelon ou assimilée, mais les neurones cellulaires sont un bon exemple de modèle utilisant une autre fonction d'activation tout en ayant une sortie binaire.

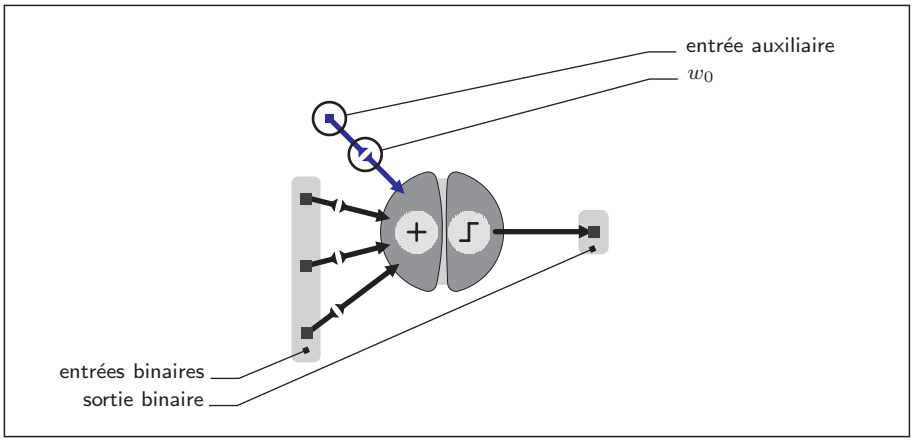


Figure 6.1
Neurone binaire de trois entrées.

binaires de façon générale plutôt qu'à des implémentations particulières. C'est cette analyse que nous allons présenter dans cette deuxième partie, à commencer par le modèle du neurone binaire.

6.1 Définition d'un neurone binaire

Cette section a pour but de décrire précisément le type de neurone concerné par notre analyse et donc par la méthode de programmation qui en découlera. Il s'agit des neurones binaires, dont la définition est la suivante :

Définition 6.1 (Neurone binaire).
Soit $\mathbb{E}$ un ensemble de deux valeurs réelles distinctes. Un neurone binaire sur $\mathbb{E}$ est un modèle de neurone formel ayant les trois propriétés suivantes :
- *ses entrées sont dans $\mathbb{E}$;*
- *sa fonction d'agrégation est une somme pondérée ;*
- *sa sortie est obtenue par un seuillage du résultat de la fonction d'agrégation, dont le niveau y est réglé par un poids synaptique auxiliaire.*

La figure 6.1 représente un neurone binaire de trois entrées en plus de l'entrée auxiliaire.

En règle générale, $\mathbb{E}$ désigne l'ensemble bipolaire $\{-1\,;+1\}$, mais dans la suite de ce manuscrit, nous lui préférerons l'ensemble $\{0\,;1\}$, que nous jugeons beaucoup plus intuitif. Néanmoins, ce choix n'implique aucune perte de généralité puisque ces deux ensembles, munis des lois de composition interne de la logique booléenne, sont isomorphes l'un de l'autre. C'est en fait le cas de n'importe quelle algèbre de Boole construite sur un ensemble binaire $\{a\,;b\}$, où $(a,b) \in \mathbb{R}^2$ et $a < b$. Cela signifie que les résultats obtenus dans ce manuscrit pour des neurones binaires sur $\{0\,;1\}$ s'appliqueront également à des neurones binaires sur $\{a\,;b\}$, moyennant quelques modifications.

En fait, passer d'une représentation basée sur l'ensemble binaire $\{a\,;b\}$ à une autre basée sur $\{0\,;1\}$ demande la définition de deux isomorphismes de corps $\mathcal{M}$ et $\mathcal{M}^{-1}$, réciproques l'un de l'autre :

$$\begin{aligned}\mathcal{M}\colon \{a\,;b\} &\longrightarrow \{0\,;1\}\\ e &\longmapsto \frac{e-a}{b-a},\end{aligned} \tag{6.1}$$

$$\begin{aligned}\mathcal{M}^{-1}\colon \{0\,;1\} &\longrightarrow \{a\,;b\}\\ e &\longmapsto e(b-a)+a.\end{aligned} \tag{6.2}$$

Ces deux applications ont trivialement les propriétés définissant des isomorphismes [2] et la composition de $\mathcal{M}^{-1}$ par $\mathcal{M}$ donne bien l'application identité.

Un neurone binaire sur $\{a\,;b\}$ peut alors être exprimé à partir d'un neurone binaire sur $\{0\,;1\}$ en appliquant $\mathcal{M}^{-1}\circ\mathcal{M}$ — c'est-à-dire la fonction identité — à chacune de ses entrées. Dans ce cas, sa fonction d'agrégation devient, en développant $\mathcal{M}^{-1}$:

$$\mathcal{A}(\mathbf{e}) = w_0 + \sum_{i=1}^{n} w_1\mathcal{M}^{-1}\circ\mathcal{M}(e_i) = w_0 + \sum_{i=1}^{n}\left((b-a)w_i\mathcal{M}(e_i) + aw_i\right). \tag{6.3}$$

En effectuant les changements de variables suivants :

$$w'_0 = w_0 + a\sum_{i=1}^{n} w_i, \tag{6.4a}$$

$$\forall i \in [\![1\,;n]\!]\,, w'_i = (b-a)w_i, \tag{6.4b}$$

$$\text{et}\quad e'_i = \mathcal{M}(e_i), \tag{6.4c}$$

la fonction d'agrégation (6.3) s'écrit maintenant comme la fonction d'agrégation d'un neurone binaire sur $\{0\,;1\}$, puisque chaque entrée $e'_i|_{i=1}^{n}$ vaut 0 ou 1 :

$$\mathcal{A}(\mathbf{e}') = w'_0 + \sum_{i=1}^{n} w'_i e'_i. \tag{6.5}$$

En conclusion, tout neurone binaire défini sur $\{a\,;b\}$ par les poids synaptiques $\begin{pmatrix}w_1 & \cdots & w_n\end{pmatrix}^\top$ et w_0 est équivalent à un neurone binaire défini sur $\{0\,;1\}$ par les poids synaptiques obtenus par les changements de variables du jeu d'équations (6.4) : $\begin{pmatrix}w'_1 & \cdots & w'_n\end{pmatrix}^\top$ et w'_0.

Exemple 6.1 — Étude d'un neurone binaire sur $\{0\,\mathrm{V}\,;5\,\mathrm{V}\}$.

Soit le neurone binaire défini par les poids synaptiques $\begin{pmatrix}2 & 1 & -2{,}4\end{pmatrix}^\top$ et par $w_0 = 0{,}5$. Les circuits électroniques utilisés pour sa mise en œuvre représentent les valeurs logiques vrai et faux par les tensions 0 V et 5 V.

[2] On a en particulier pour la porte logique non-et l'égalité $\mathcal{M}(e_1 \text{ non-et } e_2) = \mathcal{M}(e_1) \text{ non-et } \mathcal{M}(e_2)$.

Son tableau de vérité est par conséquent le suivant :

e_1	e_2	e_3	$\mathcal{A}(e_1, e_2, e_3)$
0 V	0 V	0 V	0,5 V
0 V	0 V	5 V	−11,5 V
0 V	5 V	0 V	5,5 V
0 V	5 V	5 V	−6,5 V
5 V	0 V	0 V	10,5 V
5 V	0 V	5 V	−1,5 V
5 V	5 V	0 V	15,5 V
5 V	5 V	5 V	3,5 V

La transcription de ce neurone dans $\{0\,\text{V}\,;1\,\text{V}\}$, par application des changements de variables proposés plus haut (équations 6.4), donne :

$$w'_0 = w_0 + a\sum_{i=1}^{n} w_i = 0{,}5, \qquad w'_1 = (b-a)w_1 = 10, \qquad w'_2 = (b-a)w_2 = 5,$$

et $\quad w'_3 = (b-a)w_3 = -12.$

Il est immédiat de constater que le tableau de vérité du neurone binaire sur $\{0\,\text{V}\,;1\,\text{V}\}$ défini par ces nouveaux poids synaptiques est le même que celui obtenu précédemment, ce qui illustre l'équivalence des deux neurones binaires :

e'_1	e'_2	e'_3	$\mathcal{A}(e'_1, e'_2, e'_3)$
0 V	0 V	0 V	0,5 V
0 V	0 V	1 V	−11,5 V
0 V	1 V	0 V	5,5 V
0 V	1 V	1 V	−6,5 V
1 V	0 V	0 V	10,5 V
1 V	0 V	1 V	−1,5 V
1 V	1 V	0 V	15,5 V
1 V	1 V	1 V	3,5 V

6.2 Du neurone artificiel au neurone binaire

Dans l'introduction de ce chapitre nous avons fait la proposition suivante :

Proposition 6.1.
Tout neurone artificiel dont les entrées sont le résultat d'une quantification sur un nombre fini de bits, dont la fonction d'agrégation $\mathcal{A}$ est linéaire et dont la sortie est obtenue par seuillage, peut être remplacé, à fonctionnalité équivalente, par un neurone binaire sur $\{0\,;1\}$.

Démonstration. Considérons un neurone formel dont la fonction d'agrégation $\mathcal{A}$ est linéaire et dont la fonction d'activation Φ est la fonction seuil dans $\{0\,;1\}$. Appelons n le nombre d'entrées de ce neurone et d le nombre de bits sur lesquels ces dernières sont quantifiées.

La sortie d'un tel neurone s'écrit donc $\Phi \circ \mathcal{A}(e_1, e_2, \ldots, e_n)$, en vertue de la définition 2.1 de la page 12. Comme la fonction d'agrégation de ce neurone est linéaire, alors il existe nécessairement n poids synaptiques $w_i|_{i=1}^{n}$ tels que :

$$\mathcal{A}(e_1, e_2, \ldots, e_n) = w_1 e_1 + w_2 e_2 + \cdots + w_n e_n.$$

Or les entrées sont quantifiées sur d bits, et se développent donc en des sommes pondérées par les puissances successives de deux. Par conséquent, l'égalité précédente se continue de la façon suivante :

$$\begin{aligned}
\mathcal{A}(e_1, e_2, \ldots, e_n) &= w_1 \sum_{j=0}^{d-1} 2^j e_{1j} + w_2 \sum_{j=0}^{d-1} 2^j e_{2j} + \ldots + w_n \sum_{j=0}^{d-1} 2^j e_{nj} \\
&= \sum_{i=1}^{n} \sum_{j=0}^{d-1} 2^j w_i e_{ij} \\
&\overset{\text{déf}}{=} \mathcal{A}_{n,d}(e_{10}, e_{11}, \ldots, e_{n(d-1)}).
\end{aligned}$$

Il apparaît en définitive que la sortie du neurone formel $\Phi \circ \mathcal{A}$ s'écrit également $\Phi \circ \mathcal{A}_{n,d}$, où $\mathcal{A}_{n,d}$ est une fonction linéaire de $n \times d$ variables binaires. Cette dernière est assimilable à la fonction d'agrégation d'un neurone binaire ayant $n \times d$ entrées et réalisant la même fonction que le neurone formel d'origine. □

Les deux exemples suivants précisent cette démonstration.

Exemple 6.2 — Neurone formel sans biais.

Considérons le neurone formel suivant, qui admet trois entrées échantillonnées sur deux bits :

$$\begin{aligned}
\Phi \circ \mathcal{A} \colon \{0\,;1\,;2\,;3\}^3 &\longrightarrow \{0\,;1\} \\
e_1, e_2, e_3 &\longmapsto \mathrm{H}\left(e_1 + 2e_2 + 3e_3\right).
\end{aligned}$$

Nous vérifions que sa fonction d'agrégation est linéaire, et que sa fonction d'activation correspond à un seuillage. Le résultat de la proposition 6.1 s'applique donc et le neurone binaire équivalent s'obtient en suivant le cheminement de la démonstration correspondante :

$$\begin{aligned}
\mathcal{A}(e_1, e_2, e_3) &= 1 \times e_{10} + 2 \times e_{11} + 1 \times 2e_{20} + 2 \times 2e_{21} + 1 \times 3e_{30} + 2 \times 3e_{31} \\
&= e_{10} + 2e_{11} + 2e_{20} + 4e_{21} + 3e_{30} + 6e_{31} \\
&\overset{\text{déf}}{=} \mathcal{A}_{3,2}(e_{10}, e_{11}, e_{20}, e_{21}, e_{30}, e_{31}).
\end{aligned}$$

En conclusion, le neurone binaire équivalent s'écrit :

$$\begin{aligned}
\Phi \circ \mathcal{A}_{3,2} \colon \{0\,;1\}^6 &\longrightarrow \{0\,;1\} \\
e_{10}, e_{11}, e_{20}, e_{21}, e_{30}, xe_{31} &\longmapsto \mathrm{H}\left(e_{10} + 2e_{11} + 2e_{20} + 4e_{21} + 3e_{30} + 6e_{31}\right).
\end{aligned}$$

Exemple 6.3 — Neurone formel avec biais.

Considérons le neurone formel suivant, qui admet une entrée sur quatre bits :

$$\begin{aligned}\Phi \circ \mathcal{A}\colon \{0\,;1\,;2\,;\ldots\,;15\} &\longrightarrow \{\,0\,;1\}\\ e_1 &\longmapsto \mathrm{H}\,(3+6e_1)\,.\end{aligned}$$

À cause du biais, la fonction d'agrégation n'est pas linéaire[3]. Pour exprimer ce neurone formel à l'aide d'un neurone binaire, il faut donc introduire une fonction d'agrégation différente, utilisant une entrée auxiliaire e_0 : $\mathcal{A}'(e_0, e_1) = 3e_0 + 6e_1$. Notons alors que pour toute valeur de e_1, nous avons $\mathcal{A}'(1, e_1) = \mathcal{A}(e_1)$.
Comme $\mathcal{A}'$ est linéaire, nous pouvons lui appliquer le cheminement de l'exemple précédent :

$$\begin{aligned}\mathcal{A}'(e_0, e_1) &= 1\times 3e_{00} + 2\times 3e_{01} + 4\times 3e_{02} + 8\times 3e_{03} +\\ &\qquad 1\times 6e_{10} + 2\times 6e_{11} + 4\times 6e_{12} + 8\times 6e_{13}\\ &= 3e_{00} + 6e_{01} + 12e_{02} + 24e_{03} + 6e_{10} + 12e_{11} + 24e_{12} + 48e_{13}\\ &\overset{\text{déf}}{=} \mathcal{A}'_{2,4}(e_{00}, e_{01}, e_{02}, e_{03}, e_{10}, e_{11}, e_{12}, e_{13}).\end{aligned}$$

Pour retrouver le neurone original, il faut choisir $e_0 = 1$, c'est-à-dire $e_{00} = 1$, $e_{01} = 0$, $e_{02} = 0$ et $e_{03} = 0$. La fonction d'agrégation $\mathcal{A}'_{2,4}$ devient alors la fonction d'agrégation $\mathcal{A}_{1,4}$ agissant sur les quatre entrées restantes :

$$\begin{aligned}\Phi \circ \mathcal{A}_{1,4}\colon \{\,0\,;1\}^4 &\longrightarrow \{\,0\,;1\}\\ e_{10}, e_{11}, e_{12}, e_{13} &\longmapsto \mathrm{H}\,(3+6e_{10}+12e_{11}+24e_{12}+48e_{13})\,.\end{aligned}$$

Cette démarche a été formalisée dans [Dinu et Cirstea, 2006, 2007]. Elle y avait été introduite afin d'améliorer l'implémentation des réseaux de neurones sur des fpgas.

La généralisation de ce principe à la représentation par des neurones binaires de neurones formels ayant des sorties quantifiées est un problème autrement plus complexe, que nous n'avons pas abordé dans ce manuscrit pour des raisons autant historiques que pratiques. D'une part, si la majorité des modèles sont définis avec des sorties « quasiment » binaires et non réellement binaires[4] c'est principalement parce que les fonctions d'activation correspondantes s'adaptent mieux aux algorithmes d'apprentissage que la traditionelle fonction échelon — elles sont en particulier dérivables — et non pour jouir d'une sortie aux valeurs plus nuancées. D'autre part, lorsqu'une sortie quantifiée est désirée, alors la fonction d'activation est généralement court-circuitée, ce qui aboutit à des modèles de neurones formels d'une nature trop différente pour espérer pouvoir les englober dans l'analyse de neurones binaires que cette partie va développer.

[3] En effet, $\mathcal{A}(2e_1) \neq 2\mathcal{A}(e_1)$, or l'égalité de ces deux expressions est une propriété des applications linéaires.

[4] Voir les profils des fonctions d'activation usuelles présentés dans le chapitre 2 en page 14.

6.3 Objectifs d'une méthode d'analyse de neurones binaires

La méthode d'analyse de neurones binaires, à laquelle cette deuxième partie est consacrée, a pour objectif d'ouvrir une nouvelle voie dans le domaine de la programmation analytique de neurones binaires. Contrairement aux travaux réalisés jusqu'à aujourd'hui, notre étude se veut en effet aussi indépendante que possible des architectures implémentant ces derniers ; en particulier des réseaux de neurones cellulaires, bien qu'ils soient la motivation originelle de nos recherches. Nous montrerons ainsi qu'il est possible d'aboutir à un formalisme original, capable de décrire de façon efficace la famille de fonctions qu'un neurone binaire, tel que nous l'avons défini plus tôt [5], est capable de réaliser.

Afin d'illustrer ce que nous sous-entendons par « efficace » considérons le neurone binaire de la figure 6.2a. Ce dernier réalise en fait la fonction logique et entre ses entrées. Le résultat de l'analyse de la fonction de ce neurone binaire peut donc naïvement être mis sous deux formes :

1. « le neurone binaire de la figure 6.2a compare le résultat de l'opération $1{,}5 + e_1 + e_2$ à la valeur $-1{,}5$ » ;
2. « le neurone binaire de la figure 6.2a réalise la porte logique et ».

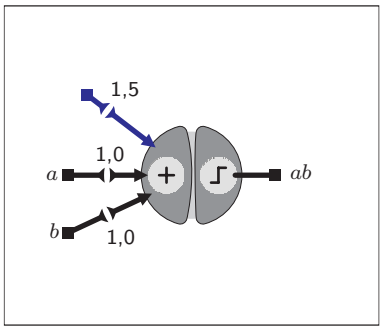
La première proposition est très proche du modèle. En effet, trouver des poids synaptiques réalisant la fonction qu'elle décrit est immédiat : il suffit d'y observer les différents coefficients multiplicateurs. Par contre, comparer de cette façon la fonction formellement réalisée par les deux neurones de la figure 6.2 est beaucoup moins intuitif.

Inversement, la seconde proposition montre tout de suite que les deux neurones binaires de la figure 6.2 sont fonctionnellement identiques. En revanche, c'est maintenant la détermination des poids synaptiques réalisant la porte et identifiée qui pose problème.

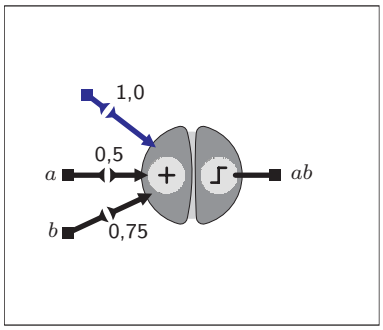
Le formalisme idéal semble donc se situer à mi-chemin entre ces deux propositions, de sorte que :
- la fonction d'un neurone binaire soit identifiable de façon unique ;
- le jeu de poids synaptiques, optimisé selon un critère donné, s'en déduise facilement.

En procédant à une étude approfondie des propriétés des neurones binaires, cette partie a pour objectif d'introduire un nouveau formalisme qui réunit ces deux critères. Nous terminerons par un algorithme permettant d'exprimer automatiquement n'importe quel neurone binaire sous cette forme. L'opération réciproque fera l'objet de la dernière partie de ce manuscrit.

[5] Les réseaux de neurones cellulaires sont par exemple capables de modifier la hauteur du seuillage en fonction de leur état interne initial. Ce n'est cependant pas une caractéristique générale des neurones binaires et nous n'en tiendrons donc pas compte.



(a) Un premier jeu de poids synaptiques donnant un **et** logique.



(b) Un second jeu de poids synaptiques donnant un **et** logique.

Figure 6.2
Réalisation d'un **et** logique avec des neurones binaires.

7

Neurones binaires et fonctions logiques

Une étude sommaire des neurones binaires conduit à la conclusion qu'ils peuvent être modélisés par des assemblages plus ou moins compliqués de portes logiques. En effet, leur fonctionnement se base sur des règles parfaitement déterministes, qui associent une sortie binaire à chaque combinaison de n entrées binaires. Il suffit donc de dresser un tableau de vérité pour déterminer les fonctions logiques ainsi réalisées. Bien-sûr, la nature de ces dernières dépend des valeurs des poids synaptiques.

Il est très facile de montrer qu'un neurone binaire ne peut réaliser qu'un certain type de fonction logique [1]. En conséquence, la première étape dans l'analyse du comportement d'un neurone binaire passe par l'identification de cet ensemble de fonctions. L'étape suivante est une étude de leurs propriétés analytiques dont le but est de trouver une représentation qui soit la plus efficace possible. Nous montrerons en particulier que les opérations logiques classiques, comme la porte et ou la porte ou, ne sont pas les mieux adaptées pour cette tâche. En l'occurrence, il vaut mieux leur préférer une porte logique d'un genre très différent : la n-somme, que nous définirons dans le chapitre suivant.

[1] Nous le ferons dans la suite de ce chapitre.

Avant d'aller plus loin, il est tout d'abord impératif de préciser le cadre mathématique permettant d'identifier un neurone binaire, dont les entrées et sorties sont dans un sous-ensemble binaire de $\mathbb{R}$, à une fonction logique de $\mathbb{B}^n$ dans $\mathbb{B}$. En effet, les calculs opérés au sein d'un neurone binaire impliquent plusieurs sommes et produits travaillant dans $\mathbb{R}$, et qui n'ont plus de sens dès lors qu'un de leurs arguments est booléen. Avec les symboles de la logique des propositions, ces incohérences deviennent évidentes. L'expression suivante n'a par exemple aucun sens :

$$\mathcal{A}(\mathsf{vrai}, \mathsf{vrai}, \mathsf{faux}, \ldots, \mathsf{vrai}) = w_0 + \mathsf{vrai}w_1 + \mathsf{vrai}w_2 + \mathsf{faux}w_3 + \ldots + \mathsf{vrai}w_n.$$

La source du problème est que la fonction d'agrégation attend des entrées binaires, certes, mais à valeurs réelles, et reçoit des entrées booléennes. Il convient donc de « convertir » chaque entrée dans $\mathbb{B}$ avant de les appliquer à la fonction d'agrégation. À cette fin, nous définissons la fonction $\mathcal{T}$ dont le rôle est d'associer une valeur réelle à chacune des deux valeurs booléennes :

$$\begin{aligned} \mathcal{T}\colon \mathbb{B} &\longrightarrow \mathbb{R} \\ \mathsf{faux} &\longmapsto 0 \\ \mathsf{vrai} &\longmapsto 1. \end{aligned} \tag{7.1}$$

Ici, $\mathcal{T}$ identifie vrai et faux à l'ensemble binaire $\{0\,;1\}$ en raison du formalisme retenu pour ce manuscrit, mais elle aurait parfaitement pu diriger $\mathbb{B}$ dans $\{-2\,;5\}$ si cela avait eu un intérêt quelconque.

Pour les mêmes raisons, il est souhaitable de définir une fonction similaire réalisant l'association réciproque : $\mathcal{T}^{-1}$. Dès lors, la fonction logique $\mathcal{L}$ de $\mathbb{B}^n$ dans $\mathbb{B}$ réalisée par un neurone binaire s'écrit :

$$\begin{aligned} \mathcal{L}\colon \mathbb{B}^n &\longrightarrow \mathbb{B} \\ e_1, e_2, \ldots, e_n &\longmapsto \mathcal{T}^{-1} \circ \Phi \circ \mathcal{A}(\mathcal{T}(e_1), \mathcal{T}(e_2), \ldots, \mathcal{T}(e_n)). \end{aligned} \tag{7.2}$$

Afin de ne pas compliquer inutilement les équations, l'utilisation de la fonction $\mathcal{T}$ sera implicite dans toute la suite de ce manuscrit, et nous assimilerons respectivement les valeurs réelles 0 et 1 aux booléens faux et vrai.

7.1 Les fonctions logiques équivalentes aux neurones binaires

Les opérations logiques de base sont très faciles à réaliser à l'aide d'un neurone binaire. Par exemple, une des implémentations de l'opération logique de conjonction (c'est-à-dire le « et » logique) est : $\begin{pmatrix} w_1 & w_2 \end{pmatrix}^\top = \begin{pmatrix} 1 & 1 \end{pmatrix}^\top$ et $w_0 = -1{,}5$ où w_0 pondère une entrée auxiliaire figée sur l'état vrai. En effet, le neurone binaire ainsi

obtenu réagit aux quatre combinaisons d'entrées possibles conformément au tableau de vérité définissant l'opération logique et :

$$\text{faux et faux} \equiv \Phi(-1{,}5+0+0) \equiv \Phi(-1{,}5) \equiv \text{faux}, \tag{7.3a}$$
$$\text{faux et vrai} \equiv \Phi(-1{,}5+0+1) \equiv \Phi(-0{,}5) \equiv \text{faux}, \tag{7.3b}$$
$$\text{vrai et faux} \equiv \Phi(-1{,}5+1+0) \equiv \Phi(-0{,}5) \equiv \text{faux}, \tag{7.3c}$$
$$\text{et} \quad \text{vrai et vrai} \equiv \Phi(-1{,}5+1+1) \equiv \Phi(0{,}5) \equiv \text{vrai}. \tag{7.3d}$$

De la même façon, un neurone binaire peut aussi réaliser l'opération logique non-et avec les poids synaptiques $\begin{pmatrix} w_1 & w_2 \end{pmatrix}^\top = \begin{pmatrix} -1 & -1 \end{pmatrix}^\top$ et $w_0 = 1{,}5$:

$$\text{faux non-et faux} \equiv \Phi(1{,}5+0+0) \equiv \Phi(1{,}5) \equiv \text{vrai}, \tag{7.4a}$$
$$\text{faux non-et vrai} \equiv \Phi(1{,}5+0-1) \equiv \Phi(0{,}5) \equiv \text{vrai}, \tag{7.4b}$$
$$\text{vrai non-et faux} \equiv \Phi(1{,}5-1+0) \equiv \Phi(0{,}5) \equiv \text{vrai}, \tag{7.4c}$$
$$\text{et} \quad \text{vrai non-et vrai} \equiv \Phi(1{,}5-1-1) \equiv \Phi(-0{,}5) \equiv \text{faux}. \tag{7.4d}$$

Comme l'opération non-et est universelle, le fait qu'un neurone binaire puisse la réaliser permet d'affirmer qu'un réseau de neurones binaires peut réaliser n'importe quelle fonction logique.

Toutefois, cette propriété ne se généralise pas à des neurones binaires individuels. Cette affirmation se démontre tout aussi facilement à l'aide d'un troisième exemple : l'opération « ou exclusif ». Bien que le problème de calcul des poids synaptiques ne soit pas l'objet de cette partie, il est très simple de constater qu'il s'agit dans ce cas d'un problème sans solution. En effet, le système d'inégalités suivant impose alors de résoudre les contradictions $w_1 < 0 < w_1$ et $w_2 < 0 < w_2$: [Minsky et Papert, 1969]

$$\text{faux oux faux} \equiv \text{faux} \quad \Rightarrow \quad \mathcal{A}(0,0) = w_0 + 0 + 0 < 0, \tag{7.5a}$$
$$\text{faux oux vrai} \equiv \text{vrai} \quad \Rightarrow \quad \mathcal{A}(0,1) = w_0 + 0 + w_2 > 0, \tag{7.5b}$$
$$\text{vrai oux faux} \equiv \text{vrai} \quad \Rightarrow \quad \mathcal{A}(1,0) = w_0 + w_1 + 0 > 0, \tag{7.5c}$$
$$\text{et} \quad \text{vrai oux vrai} \equiv \text{faux} \quad \Rightarrow \quad \mathcal{A}(1,1) = w_0 + w_1 + w_2 < 0. \tag{7.5d}$$

7.1.1 Définitions préliminaires

Les fonctions logiques réalisables avec un unique neurone binaire sont appelées « fonctions logiques à seuil » en référence au seuil utilisé pour déterminer leur valeur de sortie. Bien que cette dénomination n'en fasse pas mention, le recours à une fonction d'agrégation linéaire n'en est pas moins obligatoire. Formellement cet ensemble de fonctions logiques est défini de la manière suivante :

Définition 7.1 (Fonctions logiques à seuil).
Une fonction logique $\mathcal{L}$ de $\mathbb{B}^n$ dans $\mathbb{B}$ est une fonction logique à seuil si et seulement

s'il existe $n+1$ coefficients réels tels que :

$$\forall \mathbf{e} \in \mathbb{B}^n \begin{cases} w_0 + \sum_{i=1}^{n} w_i e_i > 0 & \Leftrightarrow \mathcal{L}(\mathbf{e}) \equiv \mathsf{vrai}, \\ w_0 + \sum_{i=1}^{n} w_i e_i < 0 & \Leftrightarrow \mathcal{L}(\mathbf{e}) \equiv \mathsf{faux}. \end{cases}$$

On notera $\mathbb{L}_s$ l'ensemble des fonctions logiques à seuil de $\mathbb{B}^n$ dans $\mathbb{B}$.

Dans le contexte de fonctions logiques réalisées par un neurone binaire, les $n+1$ coefficients de cette définition correspondent évidemment aux $n+1$ poids synaptiques, si tant est que le neurone binaire soit bien défini dans $\{0;1\}$. En outre, cette définition nous permet d'établir l'existence d'une infinité de jeux de poids synaptiques réalisant une même fonction logique. Il suffit pour cela de vérifier que si $\mathbf{w}$ et w_0 satisfont la condition nécessaire et suffisante de la définition 7.1 alors c'est nécessairement également le cas de $\mathbf{w}' = k\mathbf{w}$ et $w_0{}' = kw_0$, où $k \in \mathbb{R}^{*+}$.

De plus, les jeux de poids dérivés de cette manière ne sont pas les seuls à implémenter la même fonction logique à seuil que $\mathbf{w}$ et w_0. En effet, pour $n+1$ réels non nuls $\Delta w_i|_{i=0}^{n}$ suffisamment petits en valeur absolue, nous pouvons montrer que si $\mathbf{w}$, w_0 vérifie la condition d'équivalence de la définition 7.1 alors c'est aussi le cas de $\mathbf{w}+\Delta\mathbf{w}$, $w_0+\Delta w_0$. Afin de caractériser la notion de « suffisamment petit », nous introduisons la marge de fonctionnement d'un neurone binaire :

Définition 7.2 (Marge de fonctionnement d'un neurone binaire).
On appelle marge de fonctionnement ϵ d'un neurone binaire la plus petite valeur prise par sa fonction d'agrégation en valeur absolue.

Remarque. Afin de pouvoir comparer entre elles les marges de fonctionnement de plusieurs neurones binaires, il est souvent judicieux de les rendre indépendantes d'éventuels facteurs de proportionalité k : $\bar{\epsilon} = \epsilon / |\mathbf{w}|$. Il s'agit alors de la « marge de fonctionnement normalisée ».

7.1.2 Interprétation de la marge de fonctionnement d'un neurone binaire

Historiquement, la marge de fonctionnement d'un neurone binaire a été introduite comme une mesure de la sensibilité au bruit d'un neurone cellulaire en mode bistable [Dogaru et Chua, 1999]. Pour un CNN, les valeurs binaires 0 et 1, ainsi que les poids synaptiques sont en effet représentées par des grandeurs analogiques sujettes au bruit. Pour un bruit de 10 %, la fonction d'agrégation réellement réalisée est alors :

$$\begin{aligned} \mathcal{A}_{\text{bruit}} : \{0;1\}^n &\longrightarrow \{0;1\} \\ \mathbf{e} &\longmapsto w_0 \pm 10\,\% + \sum_{i=1}^{n} (w_i \pm 10\,\%)(e_i \pm 10\,\%). \end{aligned} \tag{7.6}$$

De toute évidence, cette fonction d'agrégation bruitée peut prendre des valeurs très différentes de celles de la fonction d'agrégation théorique. Cela implique en particulier la possibilité que les deux neurones binaires ainsi définis retournent une réponse différente à partir d'un même vecteur d'entrée. Avoir une idée de la marge disponible pour ces variations peut donc s'avérer cruciale.

Cela étant dit, ce manuscrit étudie les neurones binaires dans un cadre beaucoup plus général et ces remarques ne sont alors pas forcément pertinentes. La tolérance sur les valeurs d'entrée que nous venons d'exprimer peut alors être traduite en tolérance sur les valeurs d'un jeu de poids synaptiques, de sorte que ce dernier puisse être modifié sans que cela ne modifie la fonction logique réalisée *in fine*.

Proposition 7.1.
Soit un neurone binaire défini par le jeu de poids synaptiques w_0, $\mathbf{w}$. *Soit ensuite une perturbation additive* Δw_0, $\Delta\mathbf{w}$. *Si cette perturbation vérifie :*

$$|\Delta w_0| + |\Delta \mathbf{w}| = \sum_{i=0}^{n} |\Delta w_i| < \epsilon,$$

alors le jeu de poids synaptiques $w_0 + \Delta w_0$, $\mathbf{w} + \Delta\mathbf{w}$ *réalise la même fonction logique que* w_0, $\mathbf{w}$.

Démonstration. Considérons une fonction logique $\mathcal{L}$ de $\mathbb{B}^n$ dans $\mathbb{B}$ et w_0, $\mathbf{w}$ un jeu de poids synaptiques réalisant $\mathcal{L}$. Par définition nous avons donc, pour chacun des 2^n vecteurs $\mathbf{e}^{(j)}$ de $\mathbb{B}^n$, l'une des deux relations suivantes :

$$\begin{cases} w_0 + \sum_{i=1}^{n} w_i e_i^{(j)} = \epsilon^{(j)} > 0 & \text{si } \mathcal{L}(\mathbf{e}^{(j)}) \equiv \mathsf{vrai}, \\ w_0 + \sum_{i=1}^{n} w_i e_i^{(j)} = -\epsilon^{(j)} < 0 & \text{sinon.} \end{cases}$$

Choisissons maintenant $n+1$ ajustements synaptiques $\Delta w_i|_{i=0}^{n}$ tels que $\sum_{i=0}^{n} |\Delta w_i| < \epsilon$ et appelons $\mathbf{e}^{(1)} \in \mathbb{B}^n$ un vecteur d'entrée pour lequel $\mathcal{L}(\mathbf{e}^{(1)}) \equiv \mathsf{vrai}$, c'est-à-dire :

$$w_0 + \sum_{i=1}^{n} w_i e_i^{(1)} = \epsilon^{(1)} \geq \epsilon > 0.$$

Après ajustement des poids synaptiques, l'égalité de gauche devient :

$$\Delta w_0 + w_0 + \sum_{i=1}^{n} w_i e_i^{(1)} + \sum_{i=1}^{n} \Delta w_i e_i^{(1)} = \epsilon^{(1)} + \Delta w_0 + \sum_{i=1}^{n} \Delta w_i e_i^{(1)}.$$

Il s'agit maintenant de vérifier que la prise en compte de cette perturbation ne modifie pas le signe de cette expression, autrement dit que le membre de droite de l'égalité précédente est bien positif. Or, nous avons par inégalité triangulaire, et puisque $e_i^{(1)}|_{i=1}^{n} \in \mathbb{B}$:

$$\left| \Delta w_0 + \sum_{i=1}^{n} \Delta w_i e_i^{(1)} \right| \leq |\Delta w_0| + \sum_{i=1}^{n} |\Delta w_i| < \epsilon \leq \epsilon^{(1)}.$$

Cette relation prouve que le terme correspondant à l'altération des poids synaptiques n'est dans le pire des cas pas assez négatif pour provoquer un changement de signe.

En effectuant un raisonnement tout à fait similaire avec un vecteur $\mathbf{e}^{(2)} \in \mathbb{B}^n$ pour lequel $\mathcal{L}$ s'évalue à faux, nous pouvons conclure que le jeu de poids synaptiques $w_0 + \Delta w_0$, $\mathbf{w} + \Delta\mathbf{w}$ réalise également la fonction logique $\mathcal{L}$. □

L'établissement de cette proposition est un résultat capital pour la problématique de nos travaux. En effet, cela démontre qu'il est possible d'agir séparément sur chacun des poids synaptiques d'un neurone binaire sans altérer sa fonction logique. Les degrés de liberté ainsi dégagés laissent entrevoir la possibilité d'orienter le choix des poids synaptiques par des critères supplémentaires à celui de la réalisation de $\mathcal{L}$. Avoir la marge de fonctionnement la plus grande possible est certainement l'exemple de critère supplémentaire le plus évident à la vue de ce qui vient d'être exposé, mais nous en aborderons d'autres dans la suite de ce manuscrit.

L'exemple suivant illustre les implications de cette proposition sur un neurone binaire ayant deux entrées.

Exemple 7.1 — Altération d'un jeu de poids synaptiques.

Soit le jeu de poids synaptiques $w_0 = 2{,}6$ et $\mathbf{w} = \begin{pmatrix} -1{,}4 & -2{,}2 \end{pmatrix}^\top$ qui réalise l'opération logique non-et. Le jeu d'équations suivant donne le résultat de la fonction d'activation pour les quatre vecteurs d'entrée possibles :

$$\begin{aligned} &\text{faux non-et faux} \equiv \Phi(2{,}6) \equiv \text{vrai}, \\ &\text{faux non-et vrai} \equiv \Phi(0{,}4) \equiv \text{vrai}, \\ &\text{vrai non-et faux} \equiv \Phi(1{,}2) \equiv \text{vrai}, \\ \text{et}\quad &\text{vrai non-et vrai} \equiv \Phi(-1) \equiv \text{faux}. \end{aligned}$$

D'après la définition 7.2, la marge de fonctionnement de ce neurone binaire est $\epsilon = 0{,}4$. En altérant les poids synaptiques par $\Delta w_0 = +0{,}1$ et $\Delta\mathbf{w} = \begin{pmatrix} -0{,}2 & -0{,}05 \end{pmatrix}^\top$, dont la somme des valeurs absolues est bien inférieure à ϵ, on vérifie rapidement que la fonction logique réalisée est toujours non-et :

$$\begin{aligned} &\text{faux non-et faux} \equiv \Phi(2{,}7) \equiv \text{vrai}, \\ &\text{faux non-et vrai} \equiv \Phi(0{,}45) \equiv \text{vrai}, \\ &\text{vrai non-et faux} \equiv \Phi(1{,}1) \equiv \text{vrai}, \\ \text{et}\quad &\text{vrai non-et vrai} \equiv \Phi(-1{,}15) \equiv \text{faux}. \end{aligned}$$

En outre, la marge de fonctionnement a été améliorée puisque nous avions $\bar{\epsilon} = 0{,}4/6{,}2 = 0{,}065$ avant altération et $\bar{\epsilon} = 0{,}45/6{,}55 = 0{,}069$ après, mais ce n'est évidemment pas un résultat généralisable à toute altération des poids synaptiques satisfaisant la proposition 7.1.

L'altération suivante ne satisfait pas les conditions énoncées dans la proposition 7.1 : $\Delta w_0 = -0{,}15$ et $\Delta\mathbf{w} = \begin{pmatrix} +0{,}15 & -0{,}3 \end{pmatrix}^\top$. Il est par conséquent possible que celle-ci

modifie la fonction logique réalisée par le neurone binaire original, ce qui est le cas ici :

$$\begin{aligned}
&\text{faux non-et faux} \equiv \Phi(2{,}45) \equiv \text{vrai},\\
&\text{faux non-et vrai} \not\equiv \Phi(-0{,}05) \equiv \text{faux},\\
&\text{vrai non-et faux} \equiv \Phi(1{,}2) \equiv \text{vrai},\\
\text{et}\quad &\text{vrai non-et vrai} \equiv \Phi(-1{,}3) \equiv \text{faux}.
\end{aligned}$$

Il s'agit maintenant de la fonction logique $\overline{e}_2$ et non plus e_1 non-et e_2.
En revanche, la dernière altération proposée par cet exemple, bien qu'elle ne satisfasse pas non plus les hypothèses de la proposition 7.1 conserve malgré tout la fonction logique réalisée par le neurone binaire original : $\Delta w_0 = 0{,}1$ et $\Delta\mathbf{w} = \begin{pmatrix}-0{,}2 & -0{,}15\end{pmatrix}^\top$. En effet :

$$\begin{aligned}
&\text{faux non-et faux} \equiv \Phi(2{,}7) \equiv \text{vrai},\\
&\text{faux non-et vrai} \equiv \Phi(0{,}35) \equiv \text{vrai},\\
&\text{vrai non-et faux} \equiv \Phi(1{,}1) \equiv \text{vrai},\\
\text{et}\quad &\text{vrai non-et vrai} \equiv \Phi(-1{,}25) \equiv \text{faux}.
\end{aligned}$$

Dans cet exemple, nous avons montré qu'il était possible d'agir sur les poids synaptiques d'un neurone binaire afin d'en améliorer la qualité (ici mesurée par sa marge de fonctionnement) tout en conservant la fonction logique que celui-ci réalise. À ce stade, il est néanmoins prématuré de chercher à établir des règles de modification précises.

7.2 Propriétés des fonctions logiques à seuil

Comme l'a montré la section précédente, la modélisation des neurones binaires par des fonctions logiques à seuil permet d'obtenir quelques résultats intéressants quant à leur optimisation. Néanmoins, afin d'aller beaucoup plus loin dans la partie dédiée à la synthèse de réseaux de neurones binaires, il est indispensable d'affiner ce début d'analyse et en particulier notre compréhension des fonctions logiques à seuil. Dans cette optique, nous allons étudier plusieurs de leurs propriétés afin de mettre en évidence les raisons pour lesquelles les portes logiques classiques et, ou, non-et, etc. ne sont pas les mieux adaptées à l'écriture de telles fonctions.

7.2.1 Hypothèse simplificatrice

Afin de simplifier les explications des sous-sections suivantes, nous allons montrer qu'il est possible, sans perte de généralité, de considérer que tous les poids synaptiques d'un neurone binaire, à l'exception de w_0, sont positifs. Nous appliquerons cette hypothèse dans la suite de ce manuscrit.

Proposition 7.2 (Absorption de signe).
Si l'implémentation w_0, $\mathbf{w}$ d'une fonction logique à seuil fait intervenir un poids synaptique négatif w_i, alors il est toujours possible de le rendre positif en complémentant l'entrée correspondante et en modifiant w_0 de la façon suivante :

$$w_0{}' = w_0 + w_i.$$

La démonstration de cette proposition est triviale, nous allons simplement l'illustrer par un exemple :

Exemple 7.2 — Absorption du signe négatif d'un poids synaptique.
Considérons le neurone binaire défini par les trois poids synaptiques $w_0 = 0{,}5$, $w_1 = 1$ et $w_2 = -1$ et qui réalise donc la fonction logique $\overline{e}_1$ non-et e_2. Le poids synaptique w_2 étant négatif, nous déduisons de la proposition précédente que sa fonction logique peut-être analysée en complémentant l'entrée e_2, en changeant le signe de w_2 et en modifiant la valeur de w_0 en conséquence.
Le nouveau neurone binaire s'écrit donc $w'_0 = -0{,}5$, $w_1 = 1$ et $w''_2 = 1$ et réalise la fonction logique e_1 ou e'_2. Or comme la seconde entrée est en fait la négation de l'entrée e_2, on a e_1 ou $e'_2 \equiv e_1$ ou $\overline{e}_2$ qui est bien sémantiquement équivalent à l'écriture de départ.

On désignera l'ensemble des fonctions logiques de $\mathbb{L}_s$ réalisables avec des poids synaptiques positifs, sauf éventuellement w_0, par $\mathbb{L}_s^+$.

7.2.2 Étude des symétries

En logique booléenne, la symétrie d'une fonction est définie de la façon suivante.

Définition 7.3 (Fonction logique symétrique).
Une fonction logique est dîte symétrique (complètement) si et seulement si son évaluation est stable pour toute permutation de ses entrées.

Elle est partiellement symétrique pour un groupe d'entrées si cette propriété est vérifiée sur ce groupe d'entrées uniquement.

D'après cette définition, la valeur prise par une fonction logique symétrique ne dépend pas de l'ordre de ses variables d'entrée. En d'autres termes, le seul paramètre susceptible d'influencer la valeur de la sortie d'une telle fonction est le nombre de variables dans un état donné. Par exemple, la fonction logique e_1 et e_2 est équivalente à la fonction $\mathcal{S}_{\mathsf{et}}(\hat{e})$ définie par :

$$\begin{aligned} \mathcal{S}_{\mathsf{et}}\colon \{0\,;1\,;2\} &\longrightarrow \mathbb{B} \\ \hat{e} = 0 &\longmapsto \mathsf{faux} \\ \hat{e} = 1 &\longmapsto \mathsf{faux} \\ \hat{e} = 2 &\longmapsto \mathsf{vrai}, \end{aligned} \tag{7.7}$$

où la variable $\hat{e}$, appelée « entrée réduite de e_1 et e_2 » et que l'on notera $\hat{e} \to \{e_1\,;e_2\}$, a pour valeur le nombre d'entrées à vrai parmi e_1 et e_2. La fonction $\mathcal{S}_{\text{et}}$ se nomme « spectre » de la fonction logique et.

En fait, il est possible de définir une fonction spectre pour n'importe quelle fonction logique, qu'elle soit complètement symétrique ou non.

Définition 7.4 (Spectre d'une fonction logique).
Soit une fonction logique $\mathcal{L}(e_1, e_2, \ldots, e_n)$. En observant les symétries partielles de $\mathcal{L}$, il est possible de former $d \leq n$ groupes de n_i variables symétriques chacuns. Un vecteur réduit $\hat{\mathbf{e}}$, de dimension d, peut alors être utilisé pour décrire n'importe quel vecteur d'entrée de $\mathcal{L}$. Les entrées réduites $\hat{e}_i|_{i=1}^{d}$ correspondantes sont alors obtenues en répertoriant le nombre de variables dans l'état vrai *dans chacun de ces groupes.*

On appelle spectre de $\mathcal{L}$ la fonction $\mathcal{S}_{\mathcal{L}}(\hat{e}_1, \hat{e}_2, \ldots, \hat{e}_d) \equiv \mathcal{L}(e_1, e_2, \ldots, e_n)$. L'espace des entrées réduite sera appelé espace réduit, et nous désignerons par vecteur réduit tout vecteur de cet espace.

En général, ce spectre est représenté sous la forme d'un « tableau de vérité » (voir les exemples suivants).

Remarque. Par convention, les variables réduites sont ordonnées selon leurs influences décroissantes sur la valeur de la fonction logique à seuil : un faible indice indiquant une forte influence.

Exemple 7.3 — Spectre d'une fonction logique symétrique.

Soit la fonction logique suivante :

$$\mathcal{L}_1(e_1, e_2, e_3) \equiv e_1 e_2 \bar{e}_3 + e_1 \bar{e}_2 e_3 + \bar{e}_1 e_2 e_3.$$

Les trois variables sont ici trivialement symériques et se réduisent donc en une seule variable réduite $\hat{e}$ à valeur dans $[\![0\,;3]\!]$, ce qui signifie : ou bien aucune des variables e_1, e_2 et e_3 n'est à vrai, ou bien une seule, ou bien deux, ou bien les trois. Le spectre de cette fonction se déduit du tableau de vérité en substituant l'entrée réduite $\hat{e}$ aux trois entrées binaires e_1, e_2 et e_3.

$\hat{e}$	$\mathcal{S}_{\mathcal{L}_1}$
0	faux
1	faux
2	vrai
3	faux

Ce dernier traduit le fait que la seule façon qu'a $\mathcal{L}_1$ de valoir vrai est d'avoir exactement deux de ses trois entrées dans l'état vrai.

Exemple 7.4 — Fonction logique partiellement symétrique.

La fonction logique $\mathcal{L}_2(e_1, e_2, e_3) \equiv e_1(e_2 + e_3)$ est partiellement symétrique pour e_2 et e_3. Il est donc possible de représenter ces deux variables par la variable réduite $\hat{e}_2 \in [\![0\,;2]\!]$. Bien que la variable e_1 ne soit symétrique avec aucune autre variable, il

est tout de même possible de la représenter par la variable réduite $\hat{e}_1 \in [\![0\,;1]\!]$ qui vaut simplement 0 quand $e_1 \equiv$ faux et 1 sinon. Le spectre de $\mathcal{L}_2$ est donc :

$\hat{e}_1$	$\hat{e}_2$	$\mathcal{S}_{\mathcal{L}_2}$
0	0	faux
0	1	faux
0	2	faux
1	0	faux
1	1	vrai
1	2	vrai

Le lien avec les fonctions logiques à seuil est visible au niveau de leur implémentation sous forme de neurones binaires. Comme ces derniers utilisent une foncion d'agrégation linéaire, deux entrées pondérées par la même valeur de poids synaptique sont par conséquent symétriques au sens des fonctions booléennes (voir la définition 7.3). Cela signifie que celles-ci sont interchangeables sans incidence sur la sortie du neurone et donc sur la nature de la fonction logique réalisée [2]. Comme le montrent les exemples précédents, décrire les neurones binaires dans l'espace des entrées réduites — par le biais du spectre de la fonction logique réalisée — est certainement un choix plus pertinent en ayant sa resynthèse pour perspective. En effet, cet espace a l'avantage de conserver les informations de symétrie liant les variables d'entrées, c'est-à-dire les relations d'égalité sur les poids synaptiques correspondants.

7.2.3 Étude des variations

L'analyse des variations d'une fonction logique consiste en l'étude de la croissance ou de la décroissance de cette dernière en fonction de ses variables d'entrée. Cela suppose tout d'abord de pouvoir ordonner les fonctions logiques entre elles, ce qui est typiquement fait en posant :

$$\mathcal{L}_1 < \mathcal{L}_2 \text{ si et seulement si } \mathcal{L}_1 \Rightarrow \mathcal{L}_2. \tag{7.8}$$

Cette relation d'ordre implique en particulier faux $<$ vrai.

Une fonction logique peut ainsi avoir les propriétés suivantes :

Définition 7.5 (Constance selon une variable).
La fonction logique $\mathcal{L}(e_1, e_2, \ldots, e_n)$ est constante selon la variable e_i si et seulement si $\mathcal{L}$ ne dépend pas de e_i.

Définition 7.6 (Croissance/décroissance selon une variable).
La fonction logique $\mathcal{L}(e_1, e_2, \ldots, e_n)$ est croissante/décroissante selon la variable e_i si et seulement si la valeur de $\mathcal{L}$ croit/décroit (au sens large) quand e_i croit.

[2] Il est à noter que les deux poids synaptiques n'ont pas besoin d'être strictement identiques pour rendre les deux variables correspondantes symétriques, mais c'est un sujet que nous aborderons dans la partie suivante.

Exemple 7.5 — Fonction logique constante et croissante.
La fonction logique $\mathcal{L}_3(e_1, e_2, e_3) \equiv e_1 e_2 (e_3 + e_1)$ est constante selon sa troisième variable. En effet, le tableau de vérité de $\mathcal{L}_3$ montre que cette dernière ne dépend pas de e_3 :

	e_1	e_2	e_3	$\mathcal{L}_3$
1	faux	faux	faux	faux
2	faux	faux	vrai	faux
3	faux	vrai	faux	faux
4	faux	vrai	vrai	faux
5	vrai	faux	faux	faux
6	vrai	faux	vrai	faux
7	vrai	vrai	faux	vrai
8	vrai	vrai	vrai	vrai

Cette fonction est également croissante selon e_1 et e_2. En effet, le passage de faux à vrai de ces variables ne provoque jamais le passage de vrai à faux de $\mathcal{L}_3$.

Exemple 7.6 — Fonction logique quelconque.
La fonction logique $\mathcal{L}_4$ définie par le tableau de vérité suivant n'est ni constante ni croissante, ni décroissante selon aucune de ses variables.

	e_1	e_2	e_3	$\mathcal{L}_4$
1	faux	faux	faux	vrai
2	faux	faux	vrai	faux
3	faux	vrai	faux	faux
4	faux	vrai	vrai	vrai
5	vrai	faux	faux	faux
6	vrai	faux	vrai	vrai
7	vrai	vrai	faux	faux
8	vrai	vrai	vrai	vrai

En effet, les deux premières lignes suggèrent que $\mathcal{L}_4$ est décroissante selon e_3 ce qui est contredit par les lignes 3 et 4. Ces quatre lignes montrent de la même façon que $\mathcal{L}_4$ n'est ni croissante/décroissante, ni constante selon e_2. Enfin les lignes 1, 2, 5 et 6 conduisent à la même conclusion pour e_1.

Les fonctions logiques à seuil de $\mathbb{L}_s^+$, de par leur définition, sont des fonctions croissantes selon toutes leurs variables. On parle de monotonie complète. En effet, sous l'hypothèse des poids synaptiques positifs, le passage d'une entrée de faux à vrai, c'est-à-dire de 0 à 1, augmente la fonction d'agrégation de la valeur de son poids synaptique. En conséquence, cette dernière ne peut que rester du même côté du seuil défini par la fonction d'activation, ou bien le franchir de 0 vers 1, conformément à la définition 7.6. Cette propriété implique la proposition suivante [Bénédic et Mercklé, 2006b] :

Proposition 7.3 (Génération d'un vecteur).
Soit une fonction logique à seuil $\mathcal{L}$ de $\mathbb{L}_s^+$ définie par les poids synaptiques w_0 et $\begin{pmatrix} w_1 & \dots & w_n \end{pmatrix}^\top$. Si $\mathbf{e}$ est un vecteur d'entrée tel que $\mathcal{L}(\mathbf{e}) \equiv$ vrai alors tout vecteur $\mathbf{e}'$ obtenu à partir de $\mathbf{e}$ en passant à vrai une au moins de ses composantes à faux évalue aussi $\mathcal{L}$ à vrai. On dit que $\mathbf{e}$ « génère » $\mathbf{e}'$, ce qui est noté formellement $\mathbf{e} \rightrightarrows \mathbf{e}'$.

Démonstration. Considérons une fonction logique à seuil $\mathcal{L} \in \mathbb{L}_s^+$ de n variables et soit $\mathbf{e}$ un vecteur d'entrée tel que $\mathcal{L}(\mathbf{e}) \equiv \mathsf{vrai}$ et tel que sa j^{e} composante e_j soit dans l'état faux. Par définition, la relation suivante, écrite en isolant la contribution de e_j dans la fonction d'agrégation du neurone binaire correspondant, est vérifiée :

$$\mathcal{A}(\mathbf{e}) = w_0 + 0 \times w_j + \sum_{\substack{i=1 \\ i \neq j}}^{n} w_i e_i > 0.$$

Soit le vecteur $\mathbf{e}'$ obtenu à partir de $\mathbf{e}$ en passant la valeur de sa j^{e} composante à vrai. On a alors :

$$\mathcal{A}(\mathbf{e}') = w_0 + w_j + \sum_{\substack{i=1 \\ i \neq j}}^{n} w_i e_i > \mathcal{A}(\mathbf{e}) > 0.$$

□

Corollaire. Le processus de génération de vecteurs fonctionne également dans l'espace des entrées réduites. En effet, si $\hat{\mathbf{e}}$ est un vecteur réduit tel que $\mathcal{S}(\hat{\mathbf{e}}) \equiv \mathsf{vrai}$, alors tout vecteur $\hat{\mathbf{e}}'$ obtenu à partir de $\hat{\mathbf{e}}$ en augmentant la valeur d'une ou plusieurs de ses composantes vérifie également $\mathcal{S}(\hat{\mathbf{e}}') \equiv \mathsf{vrai}$.

La proposition 7.3 implique que le tableau de vérité d'une fonction logique à seuil peut-être simplifié en y enlevant toutes les lignes générées par au moins une ligne pour laquelle la fonction logique s'évalue à vrai. Ce principe est illustré par l'exemple suivant :

Exemple 7.7 — Tableau de vérité simplifié d'une porte ou.

Le tableau de vérité de la porte ou, qui est également une fonction logique à seuil, s'écrit :

	e_1	e_2	ou
1	faux	faux	faux
2	faux	vrai	vrai
3	vrai	faux	vrai
4	vrai	vrai	vrai

Il est trivial de constater que la quatrième ligne est générée à la fois par la deuxième et par la troisième ligne. En conséquence, son tableau de vérité peut être simplifié en :

	e_1	e_2	ou
1	faux	faux	faux
2	faux	vrai	vrai
3	vrai	faux	vrai

Par contre, la troisième ligne de ce tableau simplifié n'est pas générée par le seconde et réciproquement. En effet, pour passer de la seconde à la troisième ligne, il faudrait diminuer la valeur de e_2, de vrai à faux, ce qui est contraire au processus de génération de la proposition 7.3.

Les concepts de fonction spectre et de génération de vecteurs d'entrée vont nous permettre d'introduire la notion de « famille génératrice », qui constitue en quelque sorte le tableau de vérité optimisé d'un neurone binaire. C'est cette représentation des fonctions logiques à seuil qui est à la base de la méthode de synthèse que nous présenterons dans la partie III. Le chapitre suivant en développe le cadre mathématique, afin que nous puissions utiliser ce formalisme nouveau pour exprimer le résultat de l'analyse de neurones binaires et surtout comme point de départ de leur resynthèse.

8

Un nouveau formalisme

L'étude analytique menée dans le chapitre précédent, a ouvert la voie à l'ébauche d'une formulation arithmétique des fonctions logiques à seuil alternative au tableau de vérité. En effet, les propriétés de symétrie des fonctions logiques à seuil suggèrent d'utiliser des fonctions spectres pour décrire la fonction logique d'un neurone binaire plutôt que sa fonction logique brute. Par ailleurs, la monotonie implique une certaine redondance entre les différentes lignes d'un tableau de vérité, ce qui a été traduit par le processus de génération de la proposition 7.3.

Par conséquent, la formulation recherchée pour exprimer le résultat de l'analyse d'un neurone binaire est certainement basée sur une association de la fonction spectre et du processus de génération. Dans ce chapitre, nous présentons celle à laquelle nous avons abouti, baptisée « famille génératrice », et qui fera l'objet de la première section de ce chapitre. Dans la deuxième section, nous aborderons une autre proposition basée sur un nouveau genre de porte logique : la n-somme. Il s'agit d'une fonction logique à seuil très simple, proposée par D. Monnin comme opération de base des expressions algébriques de fonctions logiques à seuil. La dernière section établira enfin le lien entre familles génératrices et portes n-somme, et montrera en particulier comment traduire les premières en expressions à base de portes n-somme.

8.1 Les familles génératrices

Comme le montre l'exemple suivant, la famille génératrice d'une fonction logique à seuil est un tableau de vérité amélioré, qui tire partie des propriétés de symétrie et de monotonie des fonctions logiques à seuil. Il s'agit en effet du tableau de vérité de la fonction spectre d'une fonction logique à seuil, dont les lignes identifiées comme redondantes par le processus de génération auront été enlevées.

Exemple 8.1 — Famille génératrice de la porte ou.

L'opération logique ou est commutative, c'est-à-dire que ses deux entrées e_1 et e_2 sont symétriques. D'après la définition 7.4 du chapitre précédent, elles se réduisent donc en une unique variable réduite $\hat{e}$ et le spectre $\mathcal{S}_{\mathsf{ou}}$ est alors la fonction suivante :

$$\begin{aligned}\mathcal{S}_{\mathsf{ou}}\colon [\![0\,;2]\!] &\longrightarrow \mathbb{B}\\ \hat{e}^{(0)} = 0 &\longmapsto \mathsf{faux}\\ \hat{e}^{(1)} = 1 &\longmapsto \mathsf{vrai}\\ \hat{e}^{(2)} = 2 &\longmapsto \mathsf{vrai}.\end{aligned}$$

La porte logique ou est également croissante, ce qui signifie que certains vecteurs d'entrée sont générés par d'autres. En l'occurrence, il s'agit du vecteur d'entrée $\mathbf{e}^{(11)} = \begin{pmatrix}1 & 1\end{pmatrix}^\top$ qui est généré aussi bien par $\mathbf{e}^{(10)} = \begin{pmatrix}0 & 1\end{pmatrix}^\top$ que par $\mathbf{e}^{(01)} = \begin{pmatrix}1 & 0\end{pmatrix}^\top$. Dans l'espace réduit, cette relation se traduit par :

$$\hat{e}^{(1)} \rightrightarrows \hat{e}^{(2)}.$$

Ainsi, la famille génératrice s'obtient en ne conservant que les valeurs d'entrées réduites de $\mathcal{S}_{\mathsf{ou}}$ pour lesquelles ce dernier vaut vrai et qui ne sont générées par aucune autre. Ici, la famille génératrice de la porte ou serait donc simplement $\hat{e}^{(1)} = 1$. Sémantiquement, cette représentation traduit la règle de fonctionnement de l'opération ou : « n'est vrai que si au moins une de ses entrées est vraie ».

De la même façon, la famille génératrice de la porte logique et n'est constituée que de la valeur $\hat{e}^{(2)} = 2$ puisque e_1 et e_2 n'est vraie que si au moins deux des entrées sont vraies, c'est-à-dire exactement deux.

Dans cet exemple, nous avons implicitement supposé que le processus de génération était stable par réduction de symétrie. Cette hypothèse fait partie de la proposition suivante sur l'équivalence du processus de génération dans les deux espaces.

Proposition 8.1.
Si $\mathbf{e}$ *et* $\mathbf{e}'$ *sont deux vecteurs de* $\mathbb{B}^n$ *et si* $\hat{\mathbf{e}}$ *et* $\hat{\mathbf{e}}'$ *sont leurs expressions respectives dans l'espace réduit, alors :*

$$(\mathbf{e} \rightrightarrows \mathbf{e}') \Rightarrow (\hat{\mathbf{e}} \rightrightarrows \hat{\mathbf{e}}').$$

Réciproquement, si $\hat{\mathbf{e}} \rightrightarrows \hat{\mathbf{e}}'$*, alors chaque vecteur* $\mathbf{e}$*, dont la réduction donne* $\hat{\mathbf{e}}$*, génère au moins un des vecteurs* $\mathbf{e}'$ *de* $\mathbb{B}^n$ *donnant* $\hat{\mathbf{e}}'$ *par réduction.*

Démonstration. Dans le sens direct, si $\mathbf{e}'$ est généré par $\mathbf{e}$ alors cela signifie, par définition, qu'une au moins des composantes à faux de ce dernier est à vrai dans $\mathbf{e}'$, les autres restant inchangées. Cette modification a pour conséquence l'augmentation de la valeur des variables réduites correspondantes. Le vecteur réduit $\hat{\mathbf{e}}'$ obtenu est donc composé de variables réduites dont la valeur est bien supérieure ou égale à celles de $\hat{\mathbf{e}}$, ce qui signifie $\hat{\mathbf{e}} \rightrightarrows \hat{\mathbf{e}}'$.

Réciproquement, si $\hat{\mathbf{e}} \rightrightarrows \hat{\mathbf{e}}'$, et si $\mathbf{e}$ est un vecteur de $\mathbb{B}^n$ dont la réduction donne $\hat{\mathbf{e}}$, alors il est possible de construire un vecteur $\mathbf{e}'$ de $\mathbb{B}^n$ à partir de $\mathbf{e}$ de sorte que $\mathbf{e} \rightrightarrows \mathbf{e}'$. Pour cela, il suffit en effet de déterminer les composantes de $\hat{\mathbf{e}}$ dont la valeur a augmenté lors du processus de génération donnant $\hat{\mathbf{e}}'$, et de passer à vrai un nombre similaire de variables dans les groupes de symétrie correspondants de $\mathbf{e}$. □

8.1.1 Définition du nouveau formalisme

L'exemple de la page précédente donne une définition intuitive de la notion de famille génératrice d'une fonction logique à seuil : il s'agit d'un ensemble de vecteurs réduits qui évaluent à vrai la fonction logique représentée, et qui ne peuvent pas être obtenus à partir d'autres vecteurs par génération. Ces vecteurs particuliers sont appelés « vecteurs générateurs ».

Définition 8.1 (Vecteur générateur).
Soit une fonction logique à seuil $\mathcal{L}$ de $\mathbb{L}_s^+$, dont le spectre est $\mathcal{S}_\mathcal{L}$. On appelle vecteur générateur de $\mathcal{L}$, ou de $\mathcal{S}_\mathcal{L}$, tout vecteur réduit $\hat{\mathbf{e}}$ tel que :

- $\mathcal{S}_\mathcal{L}(\hat{\mathbf{e}}) \equiv$ vrai ;
- *pour tout* $\hat{\mathbf{e}}' \neq \hat{\mathbf{e}}$ *tel que* $\mathcal{S}_\mathcal{L}(\hat{\mathbf{e}}') \equiv$ vrai, *on a* $\hat{\mathbf{e}}' \not\rightrightarrows \hat{\mathbf{e}}$.

Une définition équivalente est donnée par la proposition suivante :

Proposition 8.2 (Définition équivalente d'un vecteur générateur).
Si $\mathcal{L}$ est une fonction logique à seuil de $\mathbb{L}_s^+$ et de spectre $\mathcal{S}_\mathcal{L}$, alors $\hat{\mathbf{e}}$ est un vecteur générateur de $\mathcal{L}$ si et seulement si :

- $\mathcal{S}_\mathcal{L}(\hat{\mathbf{e}}) \equiv$ vrai ;
- *pour tout $\hat{\mathbf{e}}'$ obtenu à partir de $\hat{\mathbf{e}}$ en diminuant la valeur d'une au moins de ses variables réduites, on a* $\mathcal{S}_\mathcal{L}(\hat{\mathbf{e}}') \equiv$ faux.

Démonstration. Le premier point de la proposition est présent dans la définition 8.1. Il suffit donc d'établir l'équivalence entre le second point de la proposition8.2 et le second point de cette dernière.

Dans le sens direct, si un des vecteurs $\hat{\mathbf{e}}'$ obtenus dans le second point de la proposition évaluait $\mathcal{S}_\mathcal{L}$ à vrai alors il générerait $\hat{\mathbf{e}}$ par définition, ce qui est impossible par hypothèse. Réciproquement, si $\hat{\mathbf{e}}$ est un vecteur vérifiant le second point de la proposition, alors aucun des vecteurs réduits pouvant le générer n'évalue $\mathcal{S}_\mathcal{L}$ à vrai, ce qui est la définition de vecteur générateur. □

De façon plus formelle, la famille génératrice d'une fonction logique à seuil se défini, à partir des vecteurs générateurs, de la façon suivante :

Définition 8.2 (Famille génératrice).
La famille génératrice d'une fonction logique à seuil $\mathcal{L}$ de $\mathbb{L}_s^+$ et de spectre $\mathcal{S}_\mathcal{L}$, est l'ensemble de tous ses vecteurs générateurs. Les vecteurs y sont ordonnés selon l'ordre lexicographique décroissant.

La famille génératrice de $\mathcal{L}$ est notée $\{\mathcal{S}_\mathcal{L}\}$ ou $\{\mathcal{L}\}$.

Remarque. Par définition de l'espace réduit, les variables réduites composant les vecteurs générateurs d'une famille génératrice sont indicées de la plus influente sur la valeur de la fonction logique décrite à la moins influente.

Remarque. La famille génératrice de la fonction logique vrai est $\{0\}$ tandis que la famille génératrice de la fonction logique faux est $\emptyset$.

Afin de pouvoir utiliser les familles génératrices comme représentations des fonctions logiques dont elles sont issues, il est impératif de s'assurer qu'elles leurs sont bien équivalentes de manière générale. Dans le cas contraire, alors deux familles génératrices différentes pourraient représenter la même fonction logique à seuil, ou réciproquement, deux fonctions logiques à seuil différentes pourraient avoir la même famille génératrice. La proposition suivante établi l'équivalence entre familles génératrices et fonctions spectres. Nous verrons plus loins que l'extension de cette proposition à l'équivalence entre familles génératrices et fonctions logiques à seuil demande une hypothèse supplémentaire quant à la façon dont la réduction de l'espace d'entrée a été menée.

Proposition 8.3 (Équivalence spectre/famille génératrice).
La représentation d'une fonction logique à seuil par sa fonction spectre est équivalente à sa représentation sous la forme d'une famille génératrice.

Démonstration. Le sens direct est en fait la conséquence de l'application du processus de génération aux vecteurs réduits du spectre d'une fonction logique à seuil.

La réciproque tient quant-à-elle au fait qu'un vecteur réduit ne peut pas évaluer le spectre d'une fonction logique à vrai sans être ou bien un vecteur générateur de celui-ci ou bien un vecteur réduit généré par l'un d'eux (voir la définition 8.1). Dès lors, tous les vecteurs réduits du spectre évaluant la fonction logique à seuil à vrai sont contenu directement dans la famille génératrice ou sont le fruit d'une génération. Les vecteurs réduits évaluant la fonction spectre à faux sont tous les vecteurs réduits restants. □

Calcul de quelques familles génératrices Intuitivement, le nombre de vecteurs regroupés au sein d'une famille génératrice peut être considéré comme une mesure de la complexité de la fonction logique correspondante. Il s'agit d'ailleurs là de la principale motivation derrière son utilisation aux dépends des formulations plus classiques que sont les tableaux de vérité ou les expressions algébriques à base des portes et et ou. Les quelques exemples suivants illustrent cette différence à l'avantage des familles génératrices.

Exemple 8.2 — Famille génératrice de quelques fonctions logiques à seuil.

La fonction logique à seuil $\mathcal{L}_1$ de cinq variables est définie par l'expression algébrique suivante :

$$\mathcal{L}_1(e_1, e_2, e_3, e_4, e_5) \equiv e_1(e_2 + e_3 + e_4 + e_5).$$

Son tableau de vérité est donc : (seules les lignes évaluant $\mathcal{L}_1$ à vrai sont données)

e_1	e_2	e_3	e_4	e_5	$\mathcal{L}_1$
vrai	faux	faux	faux	vrai	vrai
vrai	faux	faux	vrai	faux	vrai
vrai	faux	faux	vrai	vrai	vrai
vrai	faux	vrai	faux	faux	vrai
vrai	faux	vrai	faux	vrai	vrai
vrai	faux	vrai	vrai	faux	vrai
vrai	faux	vrai	vrai	vrai	vrai
vrai	vrai	faux	faux	faux	vrai
vrai	vrai	faux	faux	vrai	vrai
vrai	vrai	faux	vrai	faux	vrai
vrai	vrai	faux	vrai	vrai	vrai
vrai	vrai	vrai	faux	faux	vrai
vrai	vrai	vrai	faux	vrai	vrai
vrai	vrai	vrai	vrai	faux	vrai
vrai	vrai	vrai	vrai	vrai	vrai

A priori, ces quinze lignes sont autant d'informations définissant la fonction logique à seuil et dont une méthode de programmation analytique de neurones binaires devrait tenir compte. Pourtant, la famille génératrice qui en est dérivée ne compte qu'un seul vecteur dans l'espace des entrées réduites, signifiant ainsi que la complexité réelle de la fonction logique est bien moindre :

$$\{\mathcal{L}_1\}(\hat{e}_1, \hat{e}_2) = \begin{Bmatrix} 1 & 1 \end{Bmatrix},$$

où $\hat{e}_1 \rightarrow \{e_1\}$ et $\hat{e}_2 \rightarrow \{e_2\,; e_3\,; e_4\,; e_5\}$.

Paradoxalement, le tableau de vérité de la fonction logique à seuil $\mathcal{L}_2$, définie par l'expression suivante :

$$\mathcal{L}_2(e_1, e_2, e_3, e_4, e_5) \equiv e_1(e_2(e_3 + e_4e_5) + e_3e_4e_5),$$

semble à première vue moins contraignant puisqu'il ne contient plus que six lignes évaluant la fonction à vrai :

e_1	e_2	e_3	e_4	e_5	$\mathcal{L}_2$
vrai	faux	vrai	vrai	vrai	vrai
vrai	vrai	faux	vrai	vrai	vrai
vrai	vrai	vrai	faux	faux	vrai
vrai	vrai	vrai	faux	vrai	vrai
vrai	vrai	vrai	vrai	faux	vrai
vrai	vrai	vrai	vrai	vrai	vrai

Pourtant, la famille génératrice correspondante a visiblement gagné en complexité. D'une part, elle nécessite trois variables d'entrée réduites : $\hat{e}_1 \to \{e_1\}$, $\hat{e}_2 \to \{e_2\,;e_3\}$ et $\hat{e}_3 \to \{e_4\,;e_5\}$. D'autre part, elle compte un vecteur générateur supplémentaire :

$$\{\mathcal{L}_2\}(\hat{e}_1,\hat{e}_2,\hat{e}_3) = \begin{Bmatrix} 1 & 2 & 0 \\ 1 & 1 & 2 \end{Bmatrix}.$$

Cette dernière fonction logique démontre un peu plus l'efficacité avec laquelle les familles génératrices décrivent les fonctions logiques à seuil. Toutefois, les deux fonctions $\mathcal{L}_1$ et $\mathcal{L}_2$ ont en commun un nombre important de symétries, ce qui peut conduire à leur attribuer la responsabilité du si petit nombre de vecteurs générateurs obtenu. Dans l'exemple suivant, l'impact de la réduction de variables a été neutralisé par le choix d'une fonction logique à seuil n'ayant aucune symétrie entre ses variables.

Exemple 8.3 — Famille génératrice d'une fonction logique sans symétrie.

La fonction logique à seuil $\mathcal{L}_3$ suivante n'a aucune symétrie :

$$\mathcal{L}_3(e_1,e_2,e_3,e_4,e_5) \equiv e_1(e_2+e_3e_5)+e_3e_4(e_1+e_2).$$

Ainsi, les cinq variables réduites $\hat{e}_i|_{i=1}^5$ de la fonction spectre correspondante sont directement les cinq variables d'entrées e_i. Les treize lignes du tableau de vérité pour lesquelles $\mathcal{L}_3$ vaut vrai sont :

e_1	e_2	e_3	e_4	e_5	$\mathcal{L}_3$
faux	vrai	vrai	vrai	faux	vrai
faux	vrai	vrai	vrai	vrai	vrai
vrai	faux	vrai	faux	vrai	vrai
vrai	faux	vrai	vrai	faux	vrai
vrai	faux	vrai	vrai	vrai	vrai
vrai	vrai	faux	faux	faux	vrai

vrai	vrai	faux	faux	vrai	vrai
vrai	vrai	faux	vrai	faux	vrai
vrai	vrai	faux	vrai	vrai	vrai
vrai	vrai	vrai	faux	faux	vrai
vrai	vrai	vrai	faux	vrai	vrai
vrai	vrai	vrai	vrai	faux	vrai
vrai	vrai	vrai	vrai	vrai	vrai

Malgré cela, la famille génératrice de $\mathcal{L}_3$ continue à afficher un nombre réduit de vecteurs générateurs :

$$\{\mathcal{L}_3\}(\hat{e}_1,\hat{e}_2,\hat{e}_3,\hat{e}_4,\hat{e}_5) = \begin{Bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{Bmatrix}.$$

Au travers de ces exemples, le gain en lisibilité apporté par la famille génératrice est évident : chaque vecteur générateur correspond à une condition minimale d'activation de la fonction logique, si bien que leur nombre évalue effectivement la complexité de cette dernière. Par équivalence, un tableau de vérité contient également ces informations, mais la redondance importante de ce mode de représentation les noie. Les y retrouver demande donc une certaine expertise, c'est-à-dire des traitements supplémentaires (la transcription en famille génératrice par exemple).

Afin d'éviter ces calculs, et donc réellement jouir de la simplicité des familles génératrices, il est nécessaire de développer des outils mathématiques qui leur sont dédiés.

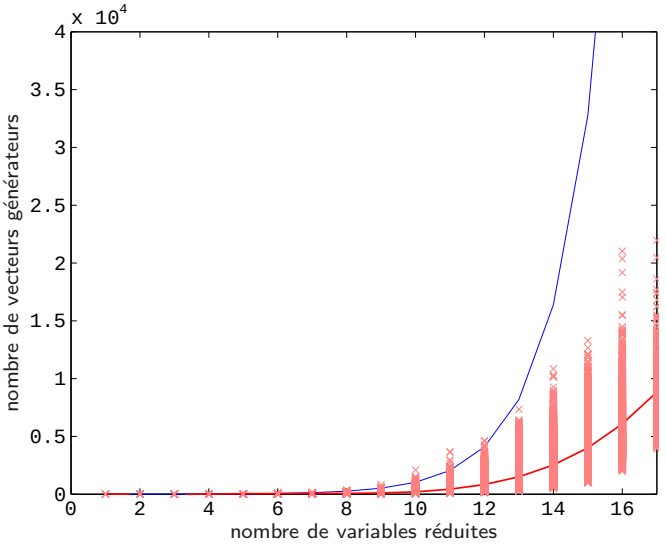


Figure 8.1
Nombre moyen de vecteurs générateurs en fonction du nombre de variables réduites. La courbe moyenne $k = d^4$ y est superposée, ainsi que la courbe $k = 2^d$ (en bleu).

En d'autres termes, l'analyse d'un neurone binaire doit directement fournir une famille génératrice et non un tableau de vérité transcrit par la suite. De la même façon, la programmation analytique d'un neurone binaire devra elle aussi s'abstraire de la représentation sous forme de tableaux de vérité, et ainsi de suite pour tout calcul affectant une famille génératrice.

Dans la figure 8.1, le nombre de vecteurs générateurs moyen pour d variables réduites a été expérimentalement évalué à d^4 au moyen de 170 000 neurones binaires de 1 à 16 entrées réduites. En comparaison avec l'évolution en 2^n du nombre de lignes d'un tableau de vérité ce résultat indique que non seulement une familles génératrice est constituée de moins de « lignes » qu'un tableau de vérité, mais qu'en plus, leur nombre augmente plus lentement avec le nombre de variables que pour ce dernier.

À ce jour, nous ne savons pas expliquer qualitativement ce résultat dont les implications seront capitales pour la complexité des algortihmes de programmation analytique de la partie III. Quantitativement, cette différence se justifie par le fait que le nombre de vecteurs réduits générés augmente plus rapidement avec le nombre de variables réduites que le nombre de vecteurs générateurs.

Anatomie d'une famille génératrice Afin de faciliter la description des procédures qui vont suivre, il est important d'introduire plusieurs définitions relatives à la façon dont une famille génératrice est structurée.

Définition 8.3 (Racine d'un vecteur générateur).
On désigne par « racine d'ordre $i \in [\![i\,;d]\!]$ » d'un vecteur générateur ses i premières composantes.

Définition 8.4 (Famille génératrice extraite).
La famille génératrice extraite de $\{\mathcal{L}\}$ pour une racine particulière est composée des vecteurs générateurs de $\{\mathcal{L}\}$ qui ont cette racine.

Définition 8.5 (Sous-famille génératrice).
La sous-famille génératrice de $\{\mathcal{L}\}$ relative à une racine d'ordre i donnée est composée des vecteurs générateurs de la famille génératrice extraite correspondante, les entrées correspondant aux racines en moins. Afin d'alléger certaine phrases, nous parlerons également de sous-famille génératrice d'ordre i pour préciser qu'elle est issue d'une racine d'ordre i. Par ailleurs, une telle sous-famille génératrice n'a plus que $d - i$ entrées réduites.

Par convention, la sous-famille génératrice d'ordre 0 d'une famille génératrice est la famille génératrice elle-même.

Exemple 8.4 — Racines et familles génératrices extraites.

Soit la fonction logique de $\mathbb{L}_s^+$ suivante :

$$\mathcal{L}_4 \equiv e_1e_2 + (e_1 + e_2)(e_3(e_4 + e_5) + e_4e_5).$$

Dans l'espace réduit constitué de $\hat{e}_1 \to \{e_1\}$, $\hat{e}_2 \to \{e_2\}$, $\hat{e}_3 \to \{e_3\}$, $\hat{e}_4 \to \{e_4\}$ et $\hat{e}_5 \to \{e_5\}$, sa famille génératrice est :

$$\{\mathcal{L}_4\} = \left\{\begin{matrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \end{matrix}\right\}.$$

Conformément à la définition 8.3, ses cinq racines d'ordre 3 sont :

$$\begin{pmatrix} 1 & 1 & 0 \end{pmatrix}, \quad \begin{pmatrix} 1 & 0 & 1 \end{pmatrix}, \quad \begin{pmatrix} 1 & 0 & 0 \end{pmatrix}, \quad \begin{pmatrix} 0 & 1 & 1 \end{pmatrix} \quad \text{et} \quad \begin{pmatrix} 0 & 1 & 0 \end{pmatrix}.$$

D'après les définitions 8.4 et 8.5, les cinq familles génératrices extraites et sous-familles génératrices correspondantes sont :

		familles extraites		sous-familles
	$\{\mathcal{L}_4\}_{(1\,1\,0)}$	$= \{1 \quad 1 \quad 0 \quad 0 \quad 0\}$	$\to$	$\{0 \quad 0\}$,
	$\{\mathcal{L}_4\}_{(1\,0\,1)}$	$= \left\{\begin{matrix} 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{matrix}\right\}$	$\to$	$\left\{\begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix}\right\}$,
	$\{\mathcal{L}_4\}_{(1\,0\,0)}$	$= \{1 \quad 0 \quad 0 \quad 1 \quad 1\}$	$\to$	$\{1 \quad 1\}$,
	$\{\mathcal{L}_4\}_{(0\,1\,1)}$	$= \left\{\begin{matrix} 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{matrix}\right\}$	$\to$	$\left\{\begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix}\right\}$,
et	$\{\mathcal{L}_4\}_{(0\,1\,0)}$	$= \{0 \quad 1 \quad 0 \quad 1 \quad 1\}$	$\to$	$\{1 \quad 1\}$.

8.1.2 Unicité de la famille génératrice d'une fonction logique à seuil

Dans le chapitre 6, nous sommes arrivés à la conclusion qu'une des propriétés que devait avoir un outil analytique adapté à la description d'un neurone binaire était l'unicité. S'agissant de familles génératrices, cela signifie qu'à une fonction logique à seuil ne doit correspondre qu'une seule famille génératrice, quelle que soit la façon dont un neurone binaire — ou un réseau — la réalise.

A priori, la famille génératrice d'une fonction logique à seuil, telle qu'elle a été définie à la page 84, hérite de l'unicité de la représentation en tableau de vérité dont elle est issue. En réalité, le processus de réduction vient ajouter un bémol à cette intuition. En effet, il y a unicité d'une famille génératrice pour une fonction spectre donnée (voir la proposition 8.3 de la page 84), mais l'établissement de l'unicité d'une famille génératrice pour un tableau de vérité suppose que la réduction de l'espace d'entrée soit complète ; c'est-à-dire que tous les groupes de variables symétriques aient été proprement identifiés et réduits.

En pratique, il peut arriver que ce ne soit pas le cas, notamment lors de l'analyse d'un neurone binaire dont les poids synaptiques ne rendraient pas compte de certaines symétries. Nous en arrivons donc à formuler la proposition suivante :

Proposition 8.4 (Unicité d'une famille génératrice).
Soit une fonction logique à seuil $\mathcal{L} \in \mathbb{L}_s^+$ de n variables e_1, e_2, …, e_n. Pour une réduction donnée de l'espace d'entrée, caractérisée par les variables réduites $\hat{e}_1$, $\hat{e}_2$, …, $\hat{e}_d$, la famille génératrice de $\mathcal{L}$ s'écrit de façon unique.

Démonstration. Montrons que deux familles génératrices distinctes $\{\mathcal{L}\}_1$ et $\{\mathcal{L}\}_2$ écrites sur un même espace d'entrée réduit décrivent des fonctions logiques à seuil nécessairement différentes.

Pour cela, appelons $\hat{\mathbf{e}}$ un vecteur générateur de $\{\mathcal{L}\}_1$ ne se trouvant pas dans $\{\mathcal{L}\}_2$ (ou inversement). Si les deux familles génératrices représentent la même fonction logique à seuil, alors $\hat{\mathbf{e}}$ est nécessairement généré par au moins un vecteur générateur de $\{\mathcal{L}\}_2$. Soit $\hat{\mathbf{e}}'$ un tel vecteur.

Pour les même raisons, $\hat{\mathbf{e}}'$ est lui aussi nécessairement généré par un vecteur générateur de $\{\mathcal{L}\}_1$: $\hat{\mathbf{e}}''$. On a alors $\hat{\mathbf{e}}'' \rightrightarrows \hat{\mathbf{e}}' \rightrightarrows \hat{\mathbf{e}}$, ce qui contredit l'hypothèse selon laquelle $\hat{\mathbf{e}}$ est un vecteur générateur de $\{\mathcal{L}_1\}$ et donc l'égalité des fonctions logiques à seuil réprésentées par $\{\mathcal{L}\}_1$ et $\{\mathcal{L}\}_2$. □

L'exemple suivant illustre le fait qu'une fonction logique à seuil trouve une écriture en famille génératrice différente en fonction de la façon dont l'espace d'entrée est réduit.

Exemple 8.5 — Famille génératrice écrite dans différents espaces.
Reprenons la famille génératrice de l'exemple 8.4 de la page 88 :

$$\{\mathcal{L}_4\} = \begin{Bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \end{Bmatrix}.$$

En remarquant que e_1 et e_2 sont symétriques, il est possible de réduire un peu plus l'espace réduit. Celui-ci devient donc $\hat{e}_1 \to \{e_1\,;e_2\}$, $\hat{e}_2 \to \{e_3\}$, $\hat{e}_3 \to \{e_4\}$ et $\hat{e}_4 \to \{e_5\}$, et la famille génératrice s'écrit maintenant :

$$\{\mathcal{L}_4\}' = \begin{Bmatrix} 2 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{Bmatrix}.$$

Enfin, en incorporant la symétrie de e_3, e_4 et e_5 à la constitution de l'espace réduit, il vient $\hat{e}_2{}' \to \{e_3\,;e_4\,;e_5\}$ et :

$$\{\mathcal{L}_4\}'' = \begin{Bmatrix} 2 & 0 \\ 1 & 2 \end{Bmatrix}.$$

Comme le montre cet exemple, plus l'espace dans lequel les familles génératrices sont écrites est réduit et plus ces dernières sont concises. Dans la suite, nous considérerons que les familles génératrices sont toujours écrites dans un espace complètement réduit, c'est-à-dire que toutes les symétries de leur fonction logique à seuil ont été réduites. À cette fin, la suite de cette sous-section présente une procédure algorithmique rapide permettant de déterminer si l'écriture d'une famille génératrice peut être réduite et de la réduire au maximum le cas échéant.

Cette opération consiste à étudier les symétries d'une fonction logique à seuil afin de déterminer l'espace réduit le plus petit possible. À cette fin, beaucoup de méthodes efficaces existent [Kohavi, 1978 ; Chow, 1961], mais elles sont basées sur l'observation d'un tableau de vérité ; une représentation dont nous souhaitons justement nous affranchir. Il est donc nécessaire de développer une nouvelle méthode, propre aux familles génératrices.

D'après la définition 7.3 de la page 74, plusieurs variables d'entrée sont symétriques pour une fonction logique $\mathcal{L}$ si et seulement si la valeur de $\mathcal{L}$ ne dépend que du nombre d'entres-elles dans un état donné. Trivialement, cette définition est également valable dans l'espace des entrées réduites. Il suffit ainsi de vérifier cette propriété sur deux entrées réduites pour en déduire, ou non, leur symétrie pour la fonction logique étudiée.

Proposition 8.5 (Symétrie de deux variables réduites).
Deux variables réduites $\hat{e}_i$ et $\hat{e}_{i+1}$ pour $i \in [\![1\,;d-1]\!]$ sont symétriques pour une

fonction logique $\mathcal{L}$ de $\mathbb{L}_s^+$ si et seulement si chacune des familles génératrices extraites d'ordre $i-1$ vérifie :

1. *les sommes $\hat{e}_{i,i+1} = \hat{e}_i + \hat{e}_{i+1}$ obtenues pour chaque vecteur générateur prennent toutes les valeurs entières possibles entre la somme maximale et minimale ;*
2. *chaque valeur $\hat{e}_{i,i+1}$, est obtenue par toutes les combinaisons possibles de $\hat{e}_i$ et $\hat{e}_{i+1}$;*
3. *les sous-familles génératrices relatives à des racines $\begin{pmatrix}\hat{e}_i & \hat{e}_{i+1}\end{pmatrix}$ dont la somme $\hat{e}_{i,i+1}$ est identique sont identiques.*

Démonstration. Ces trois points sont les implications naturelles de la vérification que la perte d'information inhérente à la réduction de $\hat{e}_{i,i+1} \rightarrow \{\hat{e}_i\,;\hat{e}_{i+1}\}$ ne modifie pas la fonction logique à seuil représentée. □

Les algorithmes 8.1 et 8.2 proposent une implémentation en pseudo-code de ce test de symétrie. Comme le test demande l'observation des k vecteurs générateurs de la famille génératrice et que $k = \mathcal{O}(d^4)$, nous pouvons en conclure que la complexité de cet algorithme est également de l'ordre de $\mathcal{O}(d^4)$.

Pour réduire complètement une famille génératrice, il faut tester la symétrie de toutes ses variables réduites deux à deux. Or, les variables étant ordonnées selon leur influence sur la valeur de la fonction logique qu'elles représentent, il n'y a besoin de tester que les symétries entre deux variables réduites successives, de sorte que seuls $d-1$ tests suffisent : $\hat{e}_1$ avec $\hat{e}_2$, $\hat{e}_2$ avec $\hat{e}_3$, …, $\hat{e}_{d-1}$ avec $\hat{e}_d$. La complexité totale de la réduction d'une famille génératrice est donc en $\mathcal{O}(d^5)$. À titre de comparaison, la méthode la plus rapide pour les fonctions logiques à seuil [Chow, 1961] requiert une unique observation du tableau de vérité et évolue donc en $\mathcal{O}(2^n)$ où $n \geq d$.

La proposition suivante détaille la procédure de réduction une fois que deux variables réduites symétriques ont été identifiées [Bénédic et al., soumis].

Proposition 8.6 (Réduction d'une famille génératrice).
Si les deux entrées réduites $\hat{e}_i$ et $\hat{e}_{i+1}$ d'une famille génératrice sont symétriques, alors cette dernière peut être réduite de la façon suivante :

- *dans chaque vecteur générateur, remplacer les deux entrées réduites symétriques par leur somme ;*
- *enlever les vecteurs générateurs doublons.*

8.1.3 Familles génératrices conjointes et complémentation

Dans la manipulation de fonctions logiques, la complémentation est une opération incontournable qu'il convient de savoir répercuter sur les familles génératrices. En effet, si cette opération a des implications triviales sur un tableau de vérité, c'est un peu moins le cas pour les familles génératrices.

Algorithme 8.1 Réduction d'une famille génératrice.

```
/* Fonction réduire:
Réduit les symétries d'une famille génératrice.
   Paramètres:
/Fg/ - famille génératrice (tableau de k lignes et d colonnes)*/

fonction reduire(Fg)=neant {
        // test de la symétrie des deux premières variables
        // et réduit tant que c'est le cas
        tantque (symetrique(Fg)) {
                Fg.reduire_variables(1,2);
        }

        // /d/ - nombre de variables réduites
        entier d = nbrevariables(Fg);
        // /ordre/ - ordre des sous-familles génératrices
        entier ordre = 1;

        // test de la symétries des variables indicées ordre+1 et ordre+2:
        // toutes les sous-familles d'ordre ordre doivent être symétriques selon leur
        // deux premières variables
        tantque (ordre<=d-1) {
                // /racines/ - tableau contenant toutes les racines de l'ordre désiré
                allouer(racines);
                racines = Fg.extraire_racines(ordre);
                // /nbreracines/ - nbre de lignes du tableau racines
                entier nbreracines = nbre_lignes(racines);

                entier i=2;

                // /sousFg/ - une sous-famille génératrice de Fg
                famillegeneratrice sousFg = Fg.sousfamille(racines[1]);

                tantque ((i <= nbreracines) && symetrique(sousFg)) {
                        sousFg = Fg.sousfamille(racine[i]);
                        i++;
                }

                // si la dernière sous-famille obtenue est symétrique alors elles
                // le sont toutes
                si (symetrique(sousFg)) {
                        Fg.reduire_variables(ordre+1,ordre+2);
                        d--; // diminue la dimension réduite
                }
                sinon { // on teste les deux variables suivantes
                        ordre++;
                }
        }
}
```

Algorithme 8.2
Test de la symétrie des deux premières variables réduites d'une famille génératrice.

```
/* Fonction symetrique:
Indique si deux premières variables réduites sont symétriques.
   Paramètres:
/Fg/ - famille génératrice (tableau de k lignes et d colonnes)*/

fonction symetrique(Fg)=booleen {
        // /k/ - nombre de vecteurs générateurs de Fg
        entier k = Fg.nbrevecteurs();
        // /racines/ - tableau contenant toutes les racines d'odre de 2 de Fg
        allouer(racines);
        racines = Fg.racines(2);
        // /sommes/ - tableau des sommes des deux variables
        allouer(sommes);
        pour (i=1;i<k;i++) {
                sommes[i] = racines[i][1]+racines[i][2];
        }

        // TEST 1
        // vérifier que le tableau somme contient toutes les valeurs entières
        // de l'intervalle [min(sommes);max(sommes)] au moins une fois
        si (!test1()) {
                retourner (faux);
        }

        // TEST 2
        // pour chaque valeur sum de [min(sommes);max(sommes)],
        // toutes les combinaisons variable1,variable2, resp. dans l'intervalle
        // d'entiers [0 Fg.cardinal(1)] et [0;Fg.cardinal(2)],
        // telles que variable1+variable2=sum sont dans le tableau sommes.
        entier sum = max(sommes);
        tantque (sum >= min(sommes)) {
                si (!test2(sum)) {
                        retourner (faux);
                }
                sum--;
        }

        // TEST 3
        // les sous-familles génératrices d'ordre 2 obtenues pour une racine d'ordre \
            2
        // dont la somme des éléments est identique sont identiques
        sum = max(sommes);
        tantque (sum >= min(sommes)) {
                si (!test3(sum)) {
                        retourner (faux);
                }
                sum--;
        }

        // Si l'on arrive jusqu'ici alors on a passé les tests
        retourner (vrai);
}
```

L'exemple suivant donne un premier aperçu des difficultés liées à la négation d'une famille génératrice simple.

Exemple 8.6 — Négation d'une fonction logique à seuil simple.

Soit $\mathcal{L}_5 \in \mathbb{L}_s^+$ une fonction logique à seuil de cinq variables symétriques $e_i|_{i=1}^5$. Le spectre de $\mathcal{L}_5$ est donc la fonction $\mathcal{S}_{\mathcal{L}_5}(\hat{e})$ et l'unique vecteur générateur de $\{\mathcal{L}_5\}$ est par exemple :

$$\{\mathcal{L}_5\} = \{2\}.$$

Verbalement, cela signifie que $\mathcal{L}_5(\mathbf{e}) \equiv \mathsf{vrai}$ dès qu'au moins deux des cinq entrées de $\mathbf{e}$ sont vraies. Autrement dit, $\mathcal{L}_5(\mathbf{e}) \equiv \mathsf{vrai}$ si au plus $5 - 2 = 3$ entrées sont fausses, ou encore $\mathcal{L}_5(\mathbf{e}) \equiv \mathsf{faux}$ dès qu'au moins quatre entrées sont fausses.

Comme $\mathcal{L}_5(\mathbf{e}) \equiv \mathsf{faux}$ est sémantiquement équivalent $\neg\mathcal{L}_5(\mathbf{e}) \equiv \mathsf{vrai}$, cela signifie que le vecteur $\big(4\big)$ peut être interprété comme l'unique vecteur générateur de la fonction logique $\neg\mathcal{L}_5$. Il est alors énoncé dans l'espace réduit formé par la variable réduite $\overline{\hat{e}} \rightarrow \{\overline{e}_i|_{i=1}^5\}$.

La suite de cette sous-section a pour objectif de généraliser ce raisonnement à la complémentation d'une famille génératrice quelconque, avant d'introduire le concept de famille génératrice conjointe.

Complément d'une famille génératrice Bien que l'exemple 8.6 puisse paraître simplet, l'algorithme général permettant de complémenter une famille génératrice donnée s'en déduit relativement aisément. Il s'agit d'une procédure récursive sur les sous-familles génératrices d'ordre de plus en plus élevé.

Proposition 8.7 (Complémentation d'une famille génératrice).
Soit $\mathcal{L}$ une fonction logique à seuil de $\mathbb{L}_s^+$, dont la famille génératrice $\{\mathcal{L}\}$ est composées de vecteurs générateurs écrits sur d variables réduites. Dans la suite, nous supposerons que la première variable réduite $\hat{e}_1$ de l'espace réduit prend ses valeurs dans l'intervalle d'entiers $[\![0\,;n_1]\!]$.

On distingue alors deux cas de figure :

- *si $d = 1$, alors $\{\mathcal{L}\}$ ne compte qu'un vecteur générateur, $\big(\eta_1\big)$, et la famille génératrice complémentée est composée de l'unique vecteur $\big(n_1 - \eta_1 + 1\big)$, s'il existe ;*
- *si $d > 1$, alors la famille génératrice complémentée est obtenue à partir des racines d'ordre 1 $\eta_{1,1} > \eta_{1,2} > \cdots > \eta_{1,k}$ en formant, s'il existe, le vecteur générateur $\big(n_1 - \eta_{1,k} + 1 \quad 0 \quad \ldots \quad 0\big)$, puis en inversant toutes les sous-familles génératrices d'ordre 1, et en leur affectant respectivement les nouvelles racines d'ordre 1 suivantes : $n_1 - \eta_{1,i}|_{i=2}^k$.*

La figure 8.2 précise cette procédure de complémentation.

Démonstration. Au travers de l'exemple 8.6, le cas où $d = 1$ est évident.

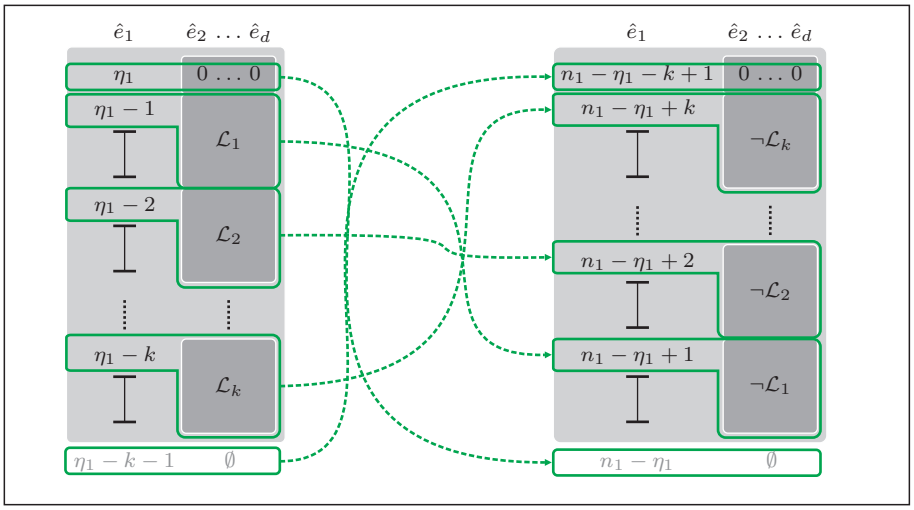


Figure 8.2
Complémentation d'une famille génératrice.

Pour le cas $d > 1$, supposons tout d'abord savoir complémenter une famille génératrice de $d - 1$ variables réduites. Les k racines d'ordre 1 représentent une plage de valeurs pour l'entrée réduite $\hat{e}_1$ telles que la fonction logique représentée peut être vraie. En d'autres termes, si $\eta_{1,k}$ est la plus petite des racines d'ordre 1, alors la fonction logique ne peut plus être vraie si $\hat{e}_1 \leq \eta_{1,k} - 1$. Par conséquent, le vecteur $\begin{pmatrix} n_1 - \eta_{1,k} + 1 & 0 & \ldots & 0 \end{pmatrix}$ est bien un vecteur générateur de la famille génératrice complémentée.

Pour chaque racine d'ordre 1, la sous-famille correspondante énonce les conditions permettant à la fonction logique d'être vraie. Par conséquent, en complémentant chacune de ces sous-familles, nous obtenons les conditions permettant à la fonction logique d'être fausse. En exprimant les racines correspondantes dans l'espace réduit complémenté, on obtient la famille génératrice complémentée. □

L'algorithme 8.3 est une implémentation en pseudo-code de la procédure de complémentation suggérée par cette proposition. Étant donné que chaque vecteur générateur n'est traité qu'une seule fois lors de la complémentation d'une famille génératrice, la complexité de cet algorithme est en $\mathcal{O}(k) = \mathcal{O}(d^4)$.

Famille génératrice conjointe Dans le processus de complémentation, la famille génératrice obtenue est exprimée dans un espace réduit où toutes les variables d'entrée ont été niées. Bien que surprenant, il faut voir dans ce résultat l'expression d'un processus de complémentation similaire à la loi de De Morgan, qui complémente toutes les variables et transforme les sommes logiques en produits et réciproquement. Ce comportement trouvera sa justification dans la section suivante.

Nous verrons dans la partie III de ce manuscrit que le complément d'une famille génératrice joue un rôle déterminant dans la programmation analytique de neurones binaires. Il y sera utilisé sous une forme alternative, dans laquelle ses variables ré-

Algorithme 8.3
Complémentation d'une famille génératrice.

```
/* Fonction complementer:
Retourne le complément d'une famille génératrice.
   Paramètres:
/Fg/ - famille génératrice (tableau de k lignes et d colonnes)*/

fonction complementer(Fg)=famillegeneratrice {
        // /complement_Fg/ - tableau stockant la famille complémentée de Fg
        famillegeneratrice complement_Fg;
        // /k/ - nombre de vecteurs de Fg
        entier k = Fg.nbrevecteurs();
        // /d/ - nombre de variables réduites de Fg
        entier d = Fg.nbrevariables();
        // /n/ - valeur maximale de la première variable réduite
        entier n = Fg.max(1);
        // /racine/ - la racinde d'ordre 1 en cours de traitement
        entier racine = Fg[k][1];

        // calcul du premier vecteur s'il existe
        si (racine != 0) {
                complement_Fg[1][1] = n - Fg[k][1] + 1;
                pour (i=2;i<=d;i++) {
                        famille[1][d] =0;
                }
        }

        // racine suivante:
        racine++;

        // calcul des vecteurs suivants, pour chaque valeur de racine
        pour (i=racine;i<Fg[1][1];i++) {
                // /sousFg/ - sous-famille génératrice extraite pour racine
                famillegeneratrice sousFg = Fg.sousfamille(racine);

                // /complement_sousFg/ - sousFg complémentée
                famillegeneratrice complement_sousFg = complementer(sousFg);

                // ajoute une colonne à gauche à complement_sousFg
                //remplie par la valeur racine en cours
                complement_sousFg.ajout_racine(i);

                // concatene complement_sousFg avec complement_Fg
                complement_Fg.concatene(complement_sousFg);
        }
        retourner(complement_Fg);
}
```

duites sont exprimées dans le même espace réduit que la famille génératrice de départ. On l'appelle alors famille génératrice conjointe.

Définition 8.6 (Famille génératrice conjointe).
La famille génératrice conjointe d'une famille génératrice $\mathcal{L}$ est son complément, exprimé dans l'espace réduit de $\mathcal{L}$. On l'obtient en effectuant la transformation suivante à chaque composante de toutes les vecteurs générateurs de $\neg\{\mathcal{L}\}$: $\eta_{i,j}$ devient $n_i - \eta_{i,j}|_{i=1,j=1}^{d,k}$, où $n_i|_{i=1}^{d}$ est la valeur maximale admissible de la variable réduite $\hat{e}_i|_{i=1}^{d}$.

8.2 La porte logique n-somme

La porte logique n-somme a été définie à l'origine dans le but « d'unifier » la description logique des neurones binaires symétriques [Monnin et al., 2000]. À l'instar de ces derniers, elle a donc un nombre d'entrées arbitrairement grand et dispose d'un paramètre qui régle l'équivalent du seuil d'activation. Dans cette section, nous allons montrer que des expressions algébriques à base de portes n-somme sont également un formalisme possible pour décrire les fonctions logiques à seuil réalisées par des neurones binaires.

Définition 8.7 (La n-somme).
Une n-somme de paramètre $\eta \in \mathbb{N}$ est une opération logique entre un certain nombre de variables d'entrée. Elle est évaluée à vrai *dès lors qu'au moins η de ces variables sont dans l'état* vrai*, elle vaut* faux *sinon.*

Cette opération est représentée par le symbole $\overset{\eta}{\top}$, où la lettre η est son paramètre. Avec n variables d'entrées, les deux écritures suivantes sont équivalentes :

$$\overset{\eta}{\top}(e_1, e_2, \ldots, e_n) \equiv (e_1 \overset{\eta}{\top} e_2 \overset{\eta}{\top} \ldots \overset{\eta}{\top} e_n).$$

Elles sont toutes les deux lues : « η-somme de e_1, e_2, jusqu'à e_n. »

Au travers de cette définition, la similitude entre le comportement d'un neurone binaire et celui d'une n-somme est évidente. En particulier, le fait que les deux aient une sortie qui bascule dès que l'« activité » perçue au travers des entrées dépasse un certain seuil, résume à lui seul le caractère monotone et symétrique de ces deux entités.

La suite présente quelques exemples de fonctions logiques à seuil écrites avec des portes n-somme.

Exemple 8.7 — Quelques écritures utilisant la porte n-somme.
La fonction logique $\mathcal{L}_1$ définie par :

$$\mathcal{L}_1(e_1, e_2, e_3) \equiv e_1e_2 + e_1e_3 + e_2e_3,$$

s'évalue à vrai dès qu'au moins deux variables parmi les trois sont dans l'état vrai. D'après la définition de la n-somme, elle s'écrit donc également :

$$\mathcal{L}_1(e_1, e_2, e_3) \equiv {}^{2}\top(e_1, e_2, e_3) \equiv (e_1 \overset{2}{\top} e_2 \overset{2}{\top} e_3).$$

La fonction logique $\mathcal{L}_2$ définie par :

$$\mathcal{L}_2(e_1, e_2, e_3, e_4) \equiv e_1 + e_2 e_3 + e_4(e_2 + e_3),$$

est vraie dès lors que $e_1 = \mathsf{vrai}$ ou qu'au moins deux des entrées e_2, e_3, et e_4 sont à vrai. Elle s'écrit donc aussi :

$$\mathcal{L}_2(e_1, e_2, e_3, e_4) \equiv {}^{1}\top(e_1, {}^{2}\top(e_2, e_3, e_4)) \equiv (e_1 \overset{1}{\top} (e_2 \overset{2}{\top} e_3 \overset{2}{\top} e_4)).$$

8.2.1 Propriétés

Au premier abord, les caractères « n-aire » et paramétrable de la n-somme en font une opération déroutante dans sa manipulation. Elle a néanmoins quelques propriétés remarquables qu'il convient d'étudier afin de mieux se familiariser avec elle :

Commutativité Une n-somme est commutative par définition, c'est-à-dire que l'ordre de ses entrées n'influence pas la valeur de sa sortie. On a par exemple :

$$(e_1 \overset{2}{\top} e_2 \overset{2}{\top} e_3) \equiv (e_2 \overset{2}{\top} e_3 \overset{2}{\top} e_1).$$

Associativité Une n-somme n'est généralement pas associative, même lorsque le même paramètre est utilisé :

$$((e_1 \overset{2}{\top} e_2 \overset{2}{\top} e_3) \overset{2}{\top} e_4 \overset{2}{\top} e_5) \not\equiv (e_1 \overset{2}{\top} e_2 \overset{2}{\top} (e_3 \overset{2}{\top} e_4 \overset{2}{\top} e_5)).$$

En effet, dans le membre de gauche de cet exemple, avoir $e_4 \equiv \mathsf{vrai}$ et $e_5 \equiv \mathsf{vrai}$ suffit à rendre la fonction logique vraie or ce n'est pas le cas avec l'expression du membre de droite qui nécessite au moins une variable vraie supplémentaire parmi e_1 et e_2.

Élément neutre L'élément neutre de la n-somme est par définition la valeur booléenne faux :

$$e_1 \overset{\eta}{\top} \ldots \overset{\eta}{\top} e_n \overset{\eta}{\top} \mathsf{faux} \equiv e_1 \overset{\eta}{\top} \ldots \overset{\eta}{\top} e_n.$$

Élément absorbant La n-somme n'a pas d'élément absorbant au sens strict du terme. Néanmoins l'action d'une variable à vrai dans une n-somme se résume de la façon suivante :

$$e_1 \overset{\eta}{\top} \ldots \overset{\eta}{\top} e_n \overset{\eta}{\top} \mathsf{vrai} \equiv e_1 \overset{\eta-1}{\top} \ldots \overset{\eta-1}{\top} e_n.$$

Au travers de ces deux dernières propriétés, nous sommes donc en mesure d'établir une relation de décomposition d'une n-somme selon une de ses variables.

Proposition 8.8 (Décomposition selon une variable).
Soit une n-somme de paramètre η appliquées à n variables $e_i|_{i=1}^{n}$. On peut la décomposer selon sa première variable de la façon suivante :

$$(e_1 \overset{\eta}{\top} \dots \overset{\eta}{\top} e_n) \equiv ((e_2 \overset{\eta}{\top} \dots \overset{\eta}{\top} e_n) + (e_1(e_2 \overset{\eta-1}{\top} \dots \overset{\eta-1}{\top} e_n)))$$

Par commutativité, on peut utiliser cette relation pour effectuer une décomposition selon la i^e variable.

Cette formule de décomposition, dérivée du théorème de Shannon, fait apparaître les deux opérations logiques standards et et ou. La proposition suivante a pour but de montrer que l'on peut s'en passer et n'utiliser que des négations et des opérations n-somme pour exprimer n'importe quelle formule logique.

Proposition 8.9 (Universalité de la n-somme).
La porte n-somme et l'opération de négation forment un système d'opérations universel, c'est-à-dire que n'importe quelle formule logique peut s'exprimer en utilisant exclusivement des opérations de négation et des portes n-somme.

Démonstration. Le triplet d'opérations {non ; et ; ou} est connu pour être universel. La négation faisant partie du système d'opérateur dont l'universalité doit être démontrée, il suffit donc de montrer comment fabriquer un et et un ou à partir d'une n-somme pour prouver la proposition.

Par définition, l'opération et est vraie lorsque ses deux entrées sont vraies, et l'opération ou lorsqu'au moins une est vraie. Nous en déduisons donc les deux équivalences sémantiques suivantes :

$$e_1 \,\mathsf{et}\, e_2 \equiv (e_1 \overset{2}{\top} e_2) \qquad\qquad e_1 \,\mathsf{ou}\, e_2 \equiv (e_1 \overset{1}{\top} e_2),$$

ce qui clôt la démonstration. □

La relation de décomposition selon une variable de la proposition 8.8 peut donc s'écrire également :

$$(e_1 \overset{\eta}{\top} \dots \overset{\eta}{\top} e_n) \equiv ((e_2 \overset{\eta}{\top} \dots \overset{\eta}{\top} e_n) \overset{1}{\top} (e_1 \overset{2}{\top} (e_2 \overset{\eta-1}{\top} \dots \overset{\eta-1}{\top} e_n))). \tag{8.1}$$

Proposition 8.10 (Factorisation d'une variable).
Soit une fonction logique $\mathcal{L}$ dont l'expression algébrique est sous la forme :

$$\mathcal{L} \equiv (\overset{\eta}{\top}(\hat{e})) \overset{1}{\top} (\mathcal{L}_1 \overset{2}{\top} (\overset{\eta-1}{\top}(\hat{e}))) \overset{1}{\top} (\mathcal{L}_2 \overset{2}{\top} (\overset{\eta-2}{\top}(\hat{e}))) \overset{1}{\top} \dots \overset{1}{\top} (\mathcal{L}_k \overset{2}{\top} (\overset{\eta-k}{\top}(\hat{e}))),$$

où $k \leq \eta$.

Si les fonctions logiques $\mathcal{L}_1$, $\mathcal{L}_2$, ..., $\mathcal{L}_k$ vérifient $\mathcal{L}_1 > \mathcal{L}_2 > \cdots > \mathcal{L}_k$, alors $\mathcal{L}$ peut se factoriser en :

$$\mathcal{L} \equiv (\hat{e} \overset{\eta}{\top} \mathcal{L}_1 \overset{\eta}{\top} \mathcal{L}_2 \overset{\eta}{\top} \ldots \overset{\eta}{\top} \mathcal{L}_k).$$

Démonstration. Ce théorème se démontre en décomposant successivement l'expression factorisée selon les fonctions logiques $\mathcal{L}_k$, $\mathcal{L}_{k-1}$, etc.

La décomposition selon $\mathcal{L}_k$ donne :

$$\mathcal{L} \equiv (\hat{e} \overset{\eta}{\top} \mathcal{L}_1 \overset{\eta}{\top} \mathcal{L}_2 \overset{\eta}{\top} \ldots \overset{\eta}{\top} \mathcal{L}_{k-1}) \overset{1}{\top} (\mathcal{L}_k \overset{2}{\top} (\hat{e} \overset{\eta-1}{\top} \mathcal{L}_1 \overset{\eta-1}{\top} \mathcal{L}_2 \overset{\eta-1}{\top} \ldots \overset{\eta-1}{\top} \mathcal{L}_{k-1})).$$

La 2-somme ne peut pas être vraie sans que $\mathcal{L}_k$ soit vraie. Or, $\mathcal{L}_k \equiv$ vrai implique par hypothèse que les autres fonctions logiques $\mathcal{L}_i|_{i=1}^{k-1}$ sont vraies également. La $(\eta-1)$-somme contient donc $k-1$ termes vrais et se simplifie en vertue de la propriété d'absorption. En définitive, on a donc :

$$\mathcal{L} \equiv (\hat{e} \overset{\eta}{\top} \mathcal{L}_1 \overset{\eta}{\top} \mathcal{L}_2 \overset{\eta}{\top} \ldots \overset{\eta}{\top} \mathcal{L}_{k-1}) \overset{1}{\top} (\mathcal{L}_k \overset{2}{\top} (\overset{\eta-k}{\top}(\hat{e}))).$$

Par récurrence, la décomposition de la η-somme selon les fonctions $\mathcal{L}_i|i = 1^{k-1}$ restantes donne l'expression recherchée :

$$\mathcal{L} \equiv (\overset{\eta}{\top}(\hat{e})) \overset{1}{\top} ((\overset{\eta-1}{\top}(\hat{e})) \overset{2}{\top} \mathcal{L}_1) \overset{1}{\top} ((\overset{\eta-2}{\top}(\hat{e})) \overset{2}{\top} \mathcal{L}_2) \overset{1}{\top} \ldots \overset{1}{\top} ((\overset{\eta-k}{\top}(\hat{e})) \overset{2}{\top} \mathcal{L}_k).$$

□

D'autres propriétés pourront être trouvées en annexe du manuscrit de thèse de D. Monnin [Monnin, 2003].

8.2.2 Écriture en n-somme de fonctions logiques à seuil

En première approximation [1], un neurone binaire réalise une fonction logique symétrique et monotone : on parle de neurone binaire symétrique. Il peut alors être modélisé par une simple n-somme dont le paramètre η s'obtient à partir du poids synaptique w_0 de l'entrée auxiliaire du neurone de la façon suivante [Bénédic et Mercklé, 2006b] :

$$\eta = 1 + \left\lfloor \frac{n}{2} \left(1 - \frac{w_0}{\sum_{i=1}^{n} w_i} \right) \right\rfloor . \tag{8.2}$$

Il s'agit simplement de compter le nombre minimal d'entrées dans l'état vrai permettant à la sortie d'entrer dans l'état vrai. En utilisant le théorème de décomposition établi dans la sous-section précédente, il est possible de décomposer cette n-somme

[1] Lorsque tous les poids synaptiques ont des valeurs semblables et que w_0 n'est pas « mal » positionné.

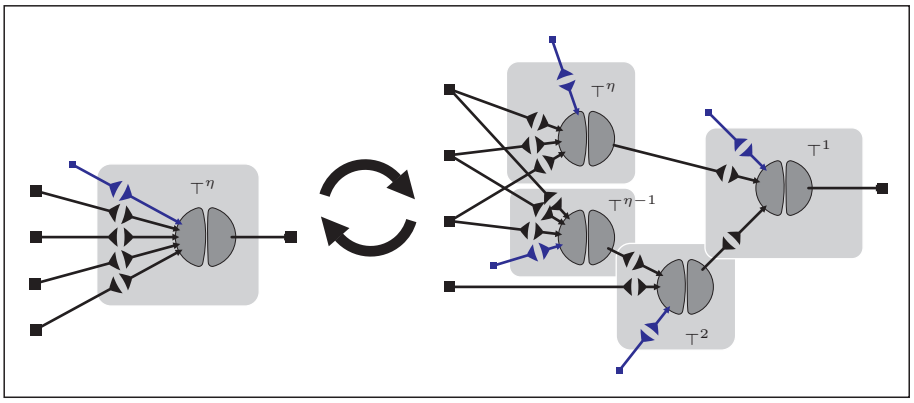


Figure 8.3
Décomposition d'un neurone binaire en un réseau de neurones binaires.

sous la forme présentée dans l'équation (8.1), c'est-à-dire en utilisant quatre opérations n-somme. Or si chaque neurone binaire symétrique correspond à une porte n-somme, alors il doit être possible d'effectuer l'opération inverse, c'est-à-dire de traduire chaque n-somme en un neurone binaire symétrique. Autrement dit, cette nouvelle écriture suggère une façon de mettre en réseau quatre neurones binaires symétriques de sorte que la fonction logique réalisée par ce dernier soit la même que celle réalisée par l'unique neurone binaire de départ et réciproquement (voir figure 8.3).

Pour cela le poids synaptique auxiliaire de chaque neurone binaire du réseau est calculé en fonction du paramètre η de la n-somme correspondante au travers de la relation suivante :

$$w_0 = \frac{(n - 2\eta + 1)}{2} \sum_{i=1}^{n} w_i, \tag{8.3}$$

qui trouvera sa justification dans la partie III.

Au travers de cet exemple naïf, nous avons montré en quoi l'opération logique n-somme est proche du modèle de neurone binaire. Néanmoins, ces derniers ne réalisent une opération logique aussi simple que relativement rarement et s'expriment donc en général sous la forme d'une imbrication de plusieurs portes n-somme. Ces expressions peuvent tout de même être programmés au sein d'un réseau de neurones binaires qui aura la propriété singulière d'attribuer la même valeur à tous ses poids synaptiques (sauf peut-être les poids synaptiques des entrées auxiliaires). Nous aborderons cette question dans la partie III.

Dans la section précédente, nous avons introduit la famille génératrice comme moyen de représenter efficacement une fonction logique à seuil. Cette dernière peut être interprétée comme un ensemble de conditions d'activation décrites par des vecteurs générateurs. De façon similaire, une n-somme permet d'implémenter rapidement une condition d'activation simple : avoir, dans un groupe de symétrie, un nombre minimal

de variables à vrai pour faire passer la porte dans l'état vrai. Nous allons voir dans la section suivante comment ce lien peut être mis à profit pour trouver très rapidement l'écriture en n-somme — et donc un réseau de neurones binaires symétriques — correspondant à une famille génératrice.

8.3 Portes n-somme et familles génératrices

Un tableau de vérité peut-être synthétisé de façon unique en une expression algébrique appelée forme normale disjonctive. De façon similaire, une famille génératrice peut-être synthétisée sous la forme d'une expression algébrique basées sur des portes n-somme. Dans un tableau de vérité, chaque ligne se traduit par un minterme qui est assemblé aux mintermes des autres lignes par une porte ou. Dans une famille génératrice, la synthèse est sensiblement différente puisque fondée sur un processus récursif.

Un vecteur générateur formalise un jeu de conditions sur les entrées permettant l'activation d'une fonction logique à seuil. Ces conditions sont exprimées par les composantes des vecteurs générateurs, qui indiquent la valeur minimale que les variables d'entrée réduites respectives doivent atteindre. En appelant $e_{i,j}$ les entrées représentées par la variable réduite $\hat{e}_i|_{i=1}^{d}$, le vecteur générateur $\hat{\mathbf{e}} = \begin{pmatrix}\eta_1 & \eta_2 & \dots & \eta_d\end{pmatrix}^\top$ se traduit donc en le produit de portes n-somme suivant :

$$\begin{pmatrix}\eta_1 & \eta_2 & \dots & \eta_d\end{pmatrix} \equiv (e_{1,1} \overset{\eta_1}{\top} e_{1,2} \overset{\eta_1}{\top} \dots)(e_{21} \overset{\eta_2}{\top} e_{2,2} \overset{\eta_2}{\top} \dots) \cdots (e_{d1} \overset{\eta_d}{\top} e_{d,2} \overset{\eta_d}{\top} \dots)). \quad (8.4)$$

Exemple 8.8 — Écriture en n-somme d'une famille génératrice constituée d'un seul vecteur générateur.

La famille génératrice de la fonction logique à seuil $\mathcal{L}_1$ définie dans l'exemple 8.2 de la page 85, contenait pour seul vecteur générateur $\hat{\mathbf{e}} = \begin{pmatrix}1 & 1\end{pmatrix}$. D'après l'équation (8.4), cette famille génératrice se traduit donc en :

$$\{\mathcal{L}_1\} \equiv (\overset{1}{\top}\hat{e}_1)(\overset{1}{\top}\hat{e}_2),$$

soit, avec les entrées binaires d'origine :

$$\begin{aligned}\{\mathcal{L}_1\} &\equiv e_1(e_2 \overset{1}{\top} e_3 \overset{1}{\top} e_4 \overset{1}{\top} e_5) \\ &\equiv e_1 \overset{2}{\top} (e_2 \overset{1}{\top} e_3 \overset{1}{\top} e_4 \overset{1}{\top} e_5).\end{aligned}$$

Dans cet exemple, l'expression en n-somme a été obtenue en appliquant des règles similaires à celles régissant la synthèse d'un tableau de vérité. L'exemple suivant montre qu'il est possible de faire mieux.

Exemple 8.9 — Écriture en n-somme d'une famille génératrice plus compliquée.

La seconde fonction logique à seuil de l'exemple 8.2 a une famille génératrice constituée des deux vecteurs suivants :

$$\hat{\mathbf{e}}^{(1)} = \begin{pmatrix}1 & 2 & 0\end{pmatrix} \quad \text{et} \quad \hat{\mathbf{e}}^{(2)} = \begin{pmatrix}1 & 1 & 2\end{pmatrix}.$$

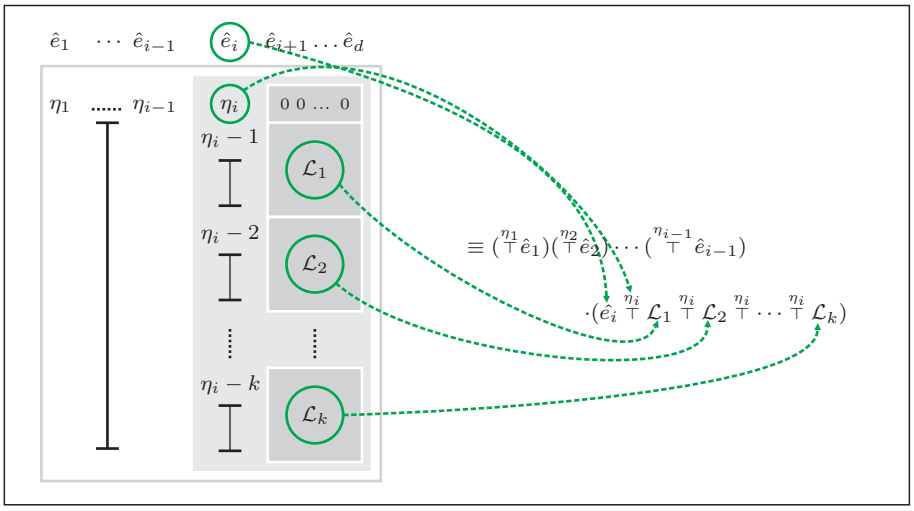


Figure 8.4
Synthèse factorisée selon une variable.

En synthétisant séparément ces deux vecteurs grâce à l'équation (8.4) puis en les réunissant par une porte ou, on trouve l'expression algébrique suivante :

$$\begin{aligned}\{\mathcal{L}_2\} &\equiv (e_1(e_2 \, {}^{2}_{\top} \, e_3)) + (e_1(e_2 \, {}^{1}_{\top} \, e_3)(e_4 \, {}^{2}_{\top} \, e_5)) \\ &\equiv e_1 \, {}^{2}_{\top} \, ((e_2 \, {}^{2}_{\top} \, e_3) \, {}^{1}_{\top} \, ((e_2 \, {}^{1}_{\top} \, e_3) \, {}^{2}_{\top} \, (e_4 \, {}^{2}_{\top} \, e_5))).\end{aligned}$$

En factorisant les variables e_2 et e_3 (voir la proposition 8.10 de la page 99), cette expression devient :

$$\{\mathcal{L}_2\} \equiv (e_1 \, {}^{2}_{\top} \, (e_2 \, {}^{2}_{\top} \, e_3 \, {}^{2}_{\top} \, (e_4 \, {}^{2}_{\top} \, e_5))).$$

Contrairement aux apparences, ce résultat ne peut pas être simplifié puisque l'opération n-somme n'est pas associative. La forme en n-somme de la famille génératrice de $\mathcal{L}_2$ est donc constituée de trois portes 2-somme.

Ces deux exemples tendent à montrer que d'une manière générale, c'est un groupe de vecteurs générateurs qu'il faut chercher à traduire en expression algébrique, et non chacun des vecteurs séparément. En opérant ainsi, le résultat obtenu est alors déjà factorisé, de sorte qu'il est considéré comme unique pour la fonction logique décrite.

La figure 8.4 montre quel groupe de vecteurs générateurs choisir pour factoriser selon la variable réduite $\hat{e}_i|_{i=1}^{d}$. Il s'agit de chaque sous-famille génératrice d'ordre $i-1$. Cette factorisation fait intervenir les sous-familles génératrices d'ordre i, nommées $\mathcal{L}_1$, $\mathcal{L}_2$, …, $\mathcal{L}_k$, qui peuvent donc être préalablement synthétisées de la même manière, et ainsi de suite.

Ce procédé de synthèse récursif s'arrête lorsque les sous-familles génératrices intervenant lors de la factorisation peuvent être traitées directement, c'est-à-dire lorsqu'elles ne dépendent plus que d'une seule variable réduite. L'expression algébrique est alors simplement une opération n-somme sur cette variable.

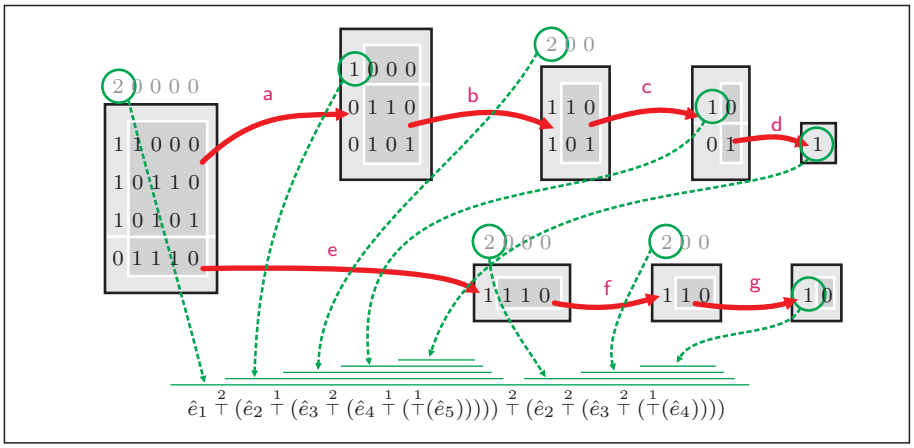


Figure 8.5
Étapes de la mise sous forme en n-somme de la famille génératrice de l'exemple 8.3.

La forme en n-somme est donc le résultat d'un algorithme récursif, dont le principe est de réaliser la synthèse factorisée de l'intégralité des vecteurs générateurs. Avant de détailler la procédure algorithmique ainsi obtenue, cette dernière est illustrée par un exemple, qui reprend la famille génératrice de l'exemple 8.3 de la page 86.

Exemple 8.10 — Forme en n-somme d'une famille génératrice.
La famille génératrice à synthétiser est la suivante :

$$\{\mathcal{L}_3(\hat{e}_1, \hat{e}_2, \hat{e}_3, \hat{e}_4, \hat{e}_5)\} \equiv \begin{Bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{Bmatrix}.$$

La figure 8.5 montre comment se distribuent les lancements récursifs de la procédure de synthèse. Le résultat obtenu est :

$$\{\mathcal{L}_3\} \equiv (\hat{e}_1 \overset{2}{\top} (\hat{e}_2 \overset{1}{\top} (\hat{e}_3 \overset{2}{\top} (\hat{e}_4 \overset{1}{\top} (\overset{1}{\top}(\hat{e}_5))))) \overset{2}{\top} (\hat{e}_2 \overset{2}{\top} (\hat{e}_3 \overset{2}{\top} (\overset{1}{\top}(\hat{e}_4))))).$$

Pour y parvenir, il faut veiller à attribuer le bon paramètre aux différentes opérations n-somme. Dans la figure 8.5, ces derniers sont cerclés de vert, et correspondent aux valeurs des racines d'ordre 1 dont les sous-familles génératrices sont la fonction logique vrai ; que ces derniers fassent effectivement partie de la famille génératrice ou non. C'est par exemple le cas de la factorisation de $\hat{e}_1$, qui demande la synthèse préalable de la fonction logique obtenue pour $\hat{e}_1 = 1$ et de celle obtenue pour $\hat{e}_1 = 0$ et qui réuni le tout avec une porte 2-somme, bien que le vecteur $\begin{pmatrix} 2 & 0 & 0 & 0 & 0 \end{pmatrix}^\top$ ne fasse pas partie de la famille génératrice.

La forme en n-somme d'une famille génératrice complète est donc le résultat d'une synthèse factorisant récursivement toutes les variables réduites de l'espace d'entrée. L'algorithme 8.4 est une implémentation en pseudo-code de cette synthèse. Sa complexité est évaluée dans la figure 8.6 par le nombre de lancements récursifs en fonction du nombre de vecteurs générateurs k de la famille génératrice. Ce test, réalisé sur 170 000 neurones binaires ayant de 1 à 16 entrées réduites et obtenus aléatoirement, montre que la complexité est en $\mathcal{O}(k)$, c'est-à-dire en $\mathcal{O}(d^4)$.

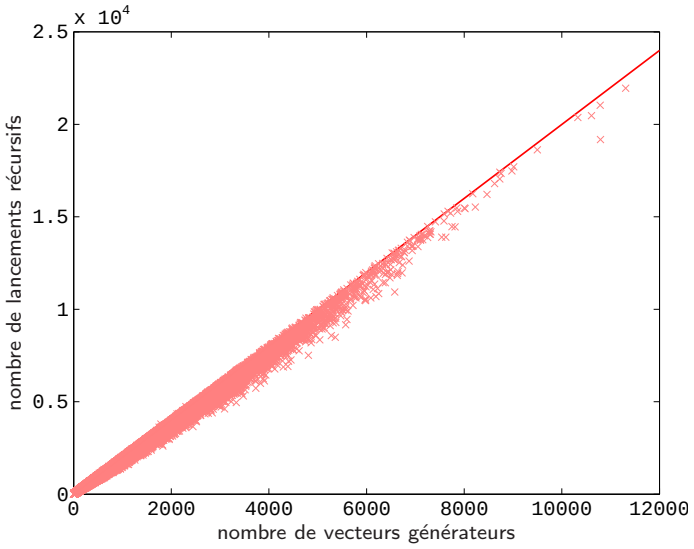


Figure 8.6
Nombre de lancements récursifs en fonction du nombre de vecteurs générateurs. À titre de comparaison, la droite $y = 2k$ est également tracée.

L'algorithme présenté à la fin de cette section permet de synthétiser, par factorisations récursives, une famille génératrice sous la forme d'une expression algébrique composée de portes `n-somme`. En supposant disposer de la famille génératrice d'un neurone binaire, ce premier résultat permet donc de l'exprimer sous la forme de plusieurs portes `n-somme` — c'est-à-dire de plusieurs neurones binaires symétriques — connectées les unes aux autres (voir la sous-section 8.2.2 de la page 100). Nous montrerons dans la partie suivante que le réseau de neurones binaires qui en découle est celui qui réalise la famille génératrice avec la dynamique de calcul la plus faible.

Algorithme 8.4
Mise sous forme en n-somme d'une famille génératrice.

```
/* Fonction factoriser:
Affiche l'expression algébrique factorisée d'une famille génératrice.
   Paramètres:
/Fg/ - tableau dont chaque ligne est un vecteur générateur
/d/ - nombre de variables réduites
/k/ - nombre de vecteurs générateurs
/variable/ - indice de la variable réduite factorisée*/

fonction factoriser(Fg,d,k,variable)=neant {
        // /depart/ - 1er vecteur générateur du groupe traitant L1
        entier depart;
        // /parametre/ - valeur du paramètre de la n-somme calculée
        entier parametre;
        // /vecteurtraite/ - indice parcourant les vecteurs générateurs
        entier vecteurtraite;
        // /nbrevecteur/ - nombre de vecteurs générateurs lus
        entier nbrevecteur;

        // cas de base du processus récursif: une seule variable réduite
        si (d == 1) {
                parametre = Fg[1][1];
                afficher<<"T"<<parametre<<"(x"<<variable<<")";
        }
        sinon {
                // cas normal du processus récursif
                si (somme(Fg[1][j],j=2:d) == 0) {
                        // vecteur du type (n 0 0 0 ...)
                        parametre = Fg[1][1];
                        depart = 2;
                }
                sinon { // autres cas
                        parametre = 1 + Fg[1][1];
                        depart = 1;
                }

                afficher<<"(x"<<variable<<"T"<<parametre;

                vecteurtraite = depart;

                // lancement récursif sur les fonctions logiques Li
                tantque (vecteurtraite <= k) {
                        // /sousFg/ - stocke la famille génératrice de Li
                        allouer(sousFg);
                        nbrevecteur = 0;

                        tantque (Fg[vecteurtraite][1] == Fg[depart][1]) {
                                // copie du vecteur générateur
                                pour (j=2;j<=d;j++) {
                                    sousFg[vecteurtraite][j] = Fg[vecteurtraite][j];
                                }
                                vecteurtraite++;
                                nbrevecteur++;
                        }

                        factoriser(variable+1,d-1,nbrevecteur,sousFg);
                        detruire(sousFg);
                        depart=vecteurtraite;
                }
                afficher<<")";
        }
}
```

9

Analyse d'un neurone binaire

L'analyse d'un neurone binaire consiste en la détermination de la fonction logique qu'il réalise au travers de ses n poids synaptiques et de son poids synaptique auxiliaire. La méthode la plus immédiate est sans nul doute le remplissage, ligne après ligne, d'un tableau de vérité. Malheureusement, dans l'optique d'une reprogrammation du neurone binaire ainsi analysé, les chapitres précédents ont montré que cette représentation souffrait de plusieurs inconvénients. En effet, si le tableau de vérité décrit effectivement de façon convenable la fonction logique réalisée par un neurone binaire, sa structure en est néanmoins beaucoup trop éloignée pour permettre un aller-retour efficace entre une expression purement logique et des poids synaptiques.

Au contraire, le chapitre précédent a montré que l'utilisation des familles génératrices permettait de rester très proche d'une forme implémentée en neurone binaire. Par le biais de la factorisation, il est par exemple très rapide de programmer un réseau de neurones binaires réalisant une fonction logique à seuil décrite sous cette forme. Par ailleurs, le faible nombre de vecteurs générateurs, comparé au nombre de lignes d'un tableau de vérité, laisse présager un temps de calcul moindre pour ce genre d'opérations.

Si la seule manière d'obtenir la famille génératrice de la fonction logique d'un neurone binaire était de passer par son tableau de vérité — comme nous l'avons fait dans le chapitre précédent — alors ce bénéfice serait complètement noyé, voire anihilé, dans la masse d'opérations que cette transcription engendredait. Il est par conséquent nécessaire de réfléchir à un algorithme d'analyse de neurones binaires dont le résultat serait directement une famille génératrice.

Dans ce chapitre, les principes de fonctionnement d'un tel algorithme d'analyse sont présentés comme conclusion de cette partie. Grâce à cet algorithme, n'importe quel neurone binaire peut être analysé en une famille génératrice dont nous montrerons dans la partie III que la manipulation permettra de choisir les propriétés du neurone ou du réseau de neurones qui en sera dérivé.

9.1 Principes de l'algorithme

Quitte à nier les entrées adéquates, conformément au principe d'absorption de signe du chapitre 7 à la page 73, nous supposons disposer d'un neurone binaire dont tous les poids synaptiques, sauf peut-être w_0, sont positifs. Par ailleurs, nous rappelons que nous utilisons l'ensemble binaire $\{0\,;1\}$ comme représentation des valeurs booléennes faux et vrai. Sous ces hypothèses, la fonction logique réalisée par un neurone binaire se définit par :

$$\begin{aligned} \mathcal{L}\colon \mathbb{B}^n &\longrightarrow \mathbb{B} \\ e_1, e_2, \ldots, e_n &\longmapsto \mathrm{H}\Big(w_0 + \sum_{i=1}^{n} w_i e_i\Big). \end{aligned} \tag{9.1}$$

9.1.1 Mise en forme du neurone binaire

La représentation en familles génératrices est construite à partir de la fonction spectre d'une fonction logique. Il convient donc dans un premier temps de transposer les entrées du neurone binaire à analyser en entrées réduites. Cette opération sous-entend deux étapes :
– réduire chaque groupe d'entrées symétriques en une variable réduite ;
– ordonner les entrées réduites en fonction de leur influence sur la sortie du neurone.
De ces deux étapes, la seconde est indéniablement la plus simple. Il suffit en effet de comparer la valeur des poids synaptiques associés à chacune des entrées pour identifier lesquelles sont les plus influentes : plus le poids synaptique a une valeur élevée et plus le basculement de l'entrée correspondante a de chances de modifier la sortie.

En revanche, la première étape nécessite quelques éclaircissements. Il est évident que deux entrées seront symétriques si leurs poids synaptiques sont égaux. Cette condition n'est pourtant pas nécessaire et deux entrées dont les poids synaptiques ne

sont pas suffisamment différents peuvent parfaitement être symétriques. La difficulté se situe ici dans le jugement de cette différence qui dépend énormément de la valeur de w_0. L'exemple suivant montre qu'aussi petite qu'elle puisse être, elle est toujours potentiellement suffisante pour briser la symétrie entre les entrées correspondantes.

Exemple 9.1 — Symétrie de deux entrées.

Soit un neurone binaire dont la sortie est calculée à partir des trois entrées e_1, e_2 et e_3 et des coefficients de pondération positifs w_1, $w_2 \ll w_1$ et $w_3 = w_2 + \epsilon$, où ϵ est un réel positif arbitrairement petit. Le poids synaptique servant de seuil a par exemple la valeur suivante : $w_0 = -w_1 - w_2 - \frac{\epsilon}{2}$.
Le tableau de vérité correspondant à ce neurone binaire est :

	e_1	e_2	e_3	$\mathcal{A}(e_1, e_2, e_3)$	sortie
1	0	0	0	$-w_1 - w_2 - \frac{\epsilon}{2}$	faux
2	0	0	1	$-w_1 + \frac{\epsilon}{2}$	faux
3	0	1	0	$-w_1 - \frac{\epsilon}{2}$	faux
4	0	1	1	$-w_1 + w_2 + \frac{\epsilon}{2}$	faux
5	1	0	0	$-w_2 - \frac{\epsilon}{2}$	faux
6	1	0	1	$\frac{\epsilon}{2}$	vrai
7	1	1	0	$-\frac{\epsilon}{2}$	faux
8	1	1	1	$w_2 + \frac{\epsilon}{2}$	vrai

Dans ce tableau, il est facile de constater que les entrées e_2 et e_3 ne sont pas symétriques puisque les lignes 6 et 7 ne donnent pas le même résultat. En revanche, si le poids synaptique w_0 avait eu la valeur $-w_1 - 2w_2$, alors le tableau de vérité se serait écrit :

	e_1	e_2	e_3	$\mathcal{A}(e_1, e_2, e_3)$	sortie
1	-1	-1	-1	$-w_1 - 2w_2$	faux
2	-1	-1	$+1$	$-w_1 - w_2 + \epsilon$	faux
3	-1	$+1$	-1	$-w_1 - w_2$	faux
4	-1	$+1$	$+1$	$-w_1 + \epsilon$	faux
5	$+1$	-1	-1	$-2w_2$	faux
6	$+1$	-1	$+1$	$-w_2 + \epsilon$	faux
7	$+1$	$+1$	-1	$-w_2$	faux
8	$+1$	$+1$	$+1$	ϵ	vrai

L'écart entre les poids synaptiques w_2 et w_3, bien qu'identique au cas précédent, n'est plus suffisant pour briser la symétrie entre e_2 et e_3.

L'enseignement à tirer de cet exemple est l'impossibilité de juger de la symétrie de deux entrées par la seule valeur de leur poids synaptique. En d'autres termes, la réduction des entrées demande *a priori* de tester la symétrie de toutes les entrées deux à deux ; une opération qui demande le calcul préalable d'un tableau de vérité, ce qui est justement ce que nous voulons éviter.

Déterminer les symétries, autrement que par la simple égalité des poids synaptiques, est donc une opération qui devra être effectuée postérieurement au calcul de la famille génératrice à l'aide de l'algorithme de réduction du chapitre 8.

En définitive, la mise en forme des informations se limite donc à réduire les entrées $e_i|_{i=1}^n$ ayant la même valeur de poids synaptique, puis à indicer les d entrées réduites obtenues de telle sorte que celles dont l'indice est le plus élevé soient pondérées par le poids synaptique le plus faible. Dans la section suivante, nous appellerons donc $\hat{w}_i|_{i=1}^d$ le poids synaptique de l'entrée réduite $\hat{e}_i|_{i=1}^d$. De plus, si $i < j$ alors $\hat{w}_i > \hat{w}_j$.

9.1.2 Calcul des vecteurs générateurs

Dans le chapitre 8 une définition équivalente d'un vecteur générateur a été formulée : un vecteur générateur de la fonction logique $\mathcal{L}$ est un vecteur réduit $\hat{\mathbf{e}}$ de l'espace réduit tel que $\mathcal{L}(\hat{\mathbf{e}}) \equiv \mathsf{vrai}$ et tel que tout vecteur $\hat{\mathbf{e}}'$ obtenu en diminuant au moins une des composantes de $\hat{\mathbf{e}}$ évalue $\mathcal{L}$ à faux.

Premier vecteur générateur Dès lors, un premier vecteur générateur du neurone binaire défini par w_0 et les $\hat{w}_i|_{i=1}^d$ s'obtient très facilement en cherchant la valeur $\hat{e}_1^{(+)}$ à partir de laquelle le vecteur $\hat{\mathbf{e}}^{(1)} = \begin{pmatrix}\hat{e}_1^{(+)} & 0 & \dots & 0\end{pmatrix}^\top$ évalue systématiquement le neurone binaire à vrai. L'équation de fonctionnement du neurone binaire implique que cette valeur vérifie à la fois les deux équations suivantes :

$$\hat{w}_1 \hat{e}_1^{(+)} > -w_0, \qquad \hat{w}_1(\hat{e}_1^{(+)} - 1) \leq -w_0. \tag{9.2}$$

Le poids synaptique $\hat{w}_1$ étant strictement positif, cela revient à positionner $\hat{e}_1^{(+)}$ comme le seul entier de l'intervalle suivant :

$$-\frac{w_0}{\hat{w}_1} < \hat{e}_1^{(+)} \leq 1 - \frac{w_0}{\hat{w}_1}. \tag{9.3}$$

Cet encadrement permet d'exprimer la valeur de $\hat{e}_1{}^{(+)}$ à l'aide de la fonction « arrondi inférieur » :

$$\hat{e}_1^+ = 1 - \left\lfloor \frac{w_0}{\hat{w}_1} \right\rfloor. \tag{9.4}$$

En supposant que cette valeur minimale puisse être effectivement atteinte par la première entrée réduite, alors le vecteur $\hat{\mathbf{e}}^{(1)}$ est par construction un vecteur générateur du neurone binaire. De plus, étant donné que les vecteurs générateurs d'une famille génératrice sont ordonnés dans l'ordre lexicographique décroissant, nous pouvons conclure que $\hat{\mathbf{e}}^{(1)}$ en est le premier vecteur générateur. En effet, il génère tous les vecteurs qui pourraient être classés devant lui.

Deuxième vecteur générateur Le vecteur générateur suivant s'obtient de façon très similaire. Le processus de génération implique que la composante $\hat{e}_1$ de tous les autres vecteurs générateurs aura une valeur strictement plus petite que $\hat{e}_1^{(+)}$. En fait la première composante du deuxième vecteur générateur de la famille génératrice

doit valoir exactement $\hat{e}_1^{(+)} - 1$. Dans le cas contraire, cela signifierait en effet que la contribution maximale combinée de toutes les autres entrées réduites n'est pas suffisante pour compenser le passage de $\hat{e}_1 = \hat{e}_1^{(+)}$ à $\hat{e}_1 = \hat{e}_1^{(+)} - 1$. Si tel était le cas, la famille génératrice ne compterait pas d'autres vecteurs générateurs.

Le vecteur générateur $\hat{\mathbf{e}}^{(2)}$ s'écrit donc $\begin{pmatrix}(\hat{e}_1^{(+)} - 1) & \hat{e}_2^{(+)} & 0 & \dots & 0\end{pmatrix}^\top$ et satisfait les deux équations suivantes, où $w_0^{(1)} = w_0 + \hat{w}_1(\hat{e}_1^{(+)} - 1)$:

$$\hat{w}_2\hat{e}_2^{(+)} > -w_0^{(1)}, \qquad \hat{w}_2(\hat{e}_2^{(+)} - 1) \leq -w_0^{(1)}. \tag{9.5}$$

En nous inspirant de la résolution qui a conduit à la valeur de $\hat{e}_1^{(+)}$, nous pouvons en déduire la valeur de $\hat{e}_2^{(+)}$:

$$\hat{e}_2^{(+)} = 1 - \left\lfloor \frac{w_0^{(1)}}{\hat{w}_2} \right\rfloor = 1 - \left\lfloor \frac{w_0}{\hat{w}_2} + \frac{\hat{w}_1}{\hat{w}_2}\left(\hat{e}_1^{(+)} - 1\right) \right\rfloor. \tag{9.6}$$

Dans la mesure où la deuxième entrée réduite peut atteindre cette valeur minimale, $\hat{\mathbf{e}}^{(2)}$ est par construction le deuxième vecteur de la famille génératrice.

Généralisation La similitude entre ces deux constructions est le signe d'un processus récursif visant à retrouver la famille génératrice d'un neurone binaire de d entrées réduites à partir de celles de plusieurs neurones binaires « extraits » ayant chacun $d-1$ entrées réduites. Le deuxième vecteur générateur calculé s'avère ainsi être le premier vecteur de la famille génératrice obtenue récursivement pour le neurone binaire extrait ayant pour entrées réduites $\hat{e}_i|_{i=2}^{d}$ et dont le poids synaptique auxiliaire est $w_0^{(1)}$.

Le traitement de la variable réduite $\hat{e}_1$ provoque ainsi plusieurs lancements récursifs, affectant aux neurones binaires extraits les poids synaptiques auxiliaires suivants :
- $w_0{}^{(1)} = w_0 + \hat{w}_1(\hat{e}_1^{(+)} - 1)$ pour $\hat{e}_1 = \hat{e}_1^{(+)} - 1$;
- $w_0{}^{(2)} = w_0 + \hat{w}_1(\hat{e}_1^{(+)} - 2)$ pour $\hat{e}_1 = \hat{e}_1^{(+)} - 2$;
- ...
- $w_0{}^{(\hat{e}_1^{(+)})} = w_0$ pour $\hat{e}_1 = 0$.

Les familles génératrices obtenues pour les neurones binaires extraits sont alors constituées de vecteurs générateurs de $d-1$ composantes. La composante manquante est simplement la valeur de l'entrée $\hat{e}_1$ pour laquelle le poids synaptique auxiliaire correspondant a été calculé.

Ce processus récursif prend naturellement fin lorsqu'il s'agit de retrouver la famille génératrice d'un neurone binaire extrait constitué d'une seule entrée réduite. L'unique vecteur générateur qu'elle contient est alors l'équivalent du premier vecteur générateur obtenu pour le cas général.

Exemple 9.2 — Calcul d'une famille génératrice simple.
Soit le neurone binaire constitué des quatre entrées e_1, e_2, e_3 et e_4, respectivement pondérées par $w_1 = 2$, $w_2 = 3$, $w_3 = 2$ et $w_4 = 3$, et dont le poids synaptique auxiliaire est $w_0 = -3{,}75$.

La mise en forme de ce neurone conduit à utiliser les deux variables réduites suivantes, dont l'ordre des indices est fixé par les poids synaptiques correspondants :
– $(\hat{e}_1, \hat{w}_1 = 3)$ pour e_2 et e_4 ;
– $(\hat{e}_2, \hat{w}_2 = 2)$ pour e_1 et e_3.
Le premier vecteur générateur est calculé à l'aide de l'expression (9.4) :

$$\hat{\mathbf{e}}^{(1)} = \begin{pmatrix} 2 & 0 \end{pmatrix}^\top.$$

Il va donc falloir lancer l'analyse récursivement sur les deux neurones binaires extraits constitués de l'entrée réduite restante $\hat{e}_2$, respectivement pour $w_0^{(1)} = -0{,}75$ (cas $\hat{e}_1 = 1$) et $w_0^{(2)} = -3{,}75$ (cas $\hat{e}_1 = 0$).

Le premier vecteur générateur du premier neurone binaire extrait s'obtient également avec l'expression (9.4). Néanmoins, ce n'est plus sur l'entrée réduite $\hat{e}_1$ que le calcul s'effectue, puisque ce neurone binaire extrait ne dispose pas de cette entrée, mais plutôt sur $\hat{e}_2$, ce qui donne le vecteur générateur :

$$\hat{\mathbf{e}}^{(2)\,\prime} = \begin{pmatrix} 1 \end{pmatrix}.$$

Comme il n'y a qu'une seule entrée réduite, il n'y a pas d'analyse récursive supplémentaire et la famille génératrice de ce neurone binaire extrait est simplement constituée du vecteur $\hat{\mathbf{e}}^{(2)\,\prime}$. Il est transformé en vecteur générateur du neurone binaire original en lui ajoutant une première composante dont la valeur est celle qui avait permis d'obtenir $w_0^{(1)}$. L'analyse récursive de ce premier neurone binaire extrait enrichi donc la famille génératrice du vecteur générateur suivant :

$$\hat{\mathbf{e}}^{(2)} = \begin{pmatrix} 1 & 1 \end{pmatrix}^\top.$$

Le second neurone binaire extrait se traite de façon similaire. Son unique vecteur générateur est :

$$\hat{\mathbf{e}}^{(3)\,\prime} = \begin{pmatrix} 2 \end{pmatrix}.$$

Comme ce neurone binaire extrait est obtenu pour $\hat{e}_1 = 0$, le troisième et dernier vecteur générateur du neurone binaire original est donc :

$$\hat{\mathbf{e}}^{(3)} = \begin{pmatrix} 0 & 2 \end{pmatrix}^\top.$$

Ce neurone binaire extrait n'a qu'une seule entrée et ne provoque pas non plus d'analyse récursive.

La famille génératrice de la fonction logique $\mathcal{L}_1$ réalisée par le neurone binaire, défini par $\mathbf{w}$ et w_0, est donc constituée des trois vecteurs générateurs suivants :

$$\{\mathcal{L}\} \equiv \left\{ \begin{matrix} 2 & 0 \\ 1 & 1 \\ 0 & 2 \end{matrix} \right\}.$$

9.2 Optimisation de l'algorithme

Le principe algorithmique décrit dans la section précédente présente l'avantage d'être complètement déterministe dans le sens où il ne requiert aucune opération de comparaison. Par contre, il a l'inconvénient de calculer beaucoup de vecteurs générateurs

superflus, c'est-à-dire faisant intervenir des valeurs trop élevées pour une ou plusieurs de leurs composantes. En effet, rien ne garanti que le résultat de l'expression (9.4), utilisée pour construire le premier vecteur générateur de chaque analyse, soit dans l'intervalle des valeurs admissibles pour l'entrée réduite concernée. Si tel n'est pas le cas, alors le vecteur générateur obtenu n'existe pas en réalité ; il est superflu.

Par exemple, si le poids synaptique auxiliaire de l'exemple 9.2 avait été fixé à -14, alors le premier vecteur générateur aurait été :

$$\hat{\mathbf{e}}^{(1)} = \begin{pmatrix} 5 & 0 \end{pmatrix}^\top, \tag{9.7}$$

bien que l'entrée réduite correspondante $\hat{e}_1$ ne puisse valoir que 2 au maximum. Ce vecteur générateur, ainsi que tous ceux ayant une valeur strictement supérieure à 2 comme première composante, doivent donc être retirés de la famille génératrice d'un tel neurone binaire.

Heureusement, ces vecteurs générateurs superflus ne remettent pas pour autant en cause la validité des vecteurs générateurs suivants. Éviter le calcul inutile de ces nombreux vecteurs est donc purement une question d'optimisation algorithmique.

En désignant par n_1 la valeur maximale que peut prendre l'entrée réduite $\hat{e}_1$, il suffit alors de comparer le résultat de l'expression (9.4) avec celui-ci. Si $\hat{e}_1^{(+)} > n_1$ alors le vecteur $\hat{\mathbf{e}}^{(1)} = \begin{pmatrix} \hat{e}_1^{(+)} & 0 & \dots & 0 \end{pmatrix}^\top$ ne fait pas partie de la famille génératrice et les lancements récursifs doivent commencer sur le neurone binaire extrait pour $\hat{e}_1 = n_1$. Dans le cas contraire, le vecteur $\hat{\mathbf{e}}^{(1)}$ fait partie de la famille génératrice et la première analyse récursive a lieu sur le neurone binaire extrait pour $\hat{e}_1 = \hat{e}_1^{(+)} - 1$, conformément à la section précédente.

De façon similaire, il est fort probable que certains neurones binaires extraits pour des valeurs trop faibles de $\hat{e}_1$ ne puissent plus être vrais et aient une famille génératrice vide. C'est par exemple le cas du neurone binaire de l'exemple 9.2 avec $w_0 = -4{,}75$. Le premier vecteur générateur est toujours $\hat{\mathbf{e}}^{(1)} = \begin{pmatrix} 2 & 0 \end{pmatrix}^\top$, tandis que le premier neurone binaire extrait ajoute $\hat{\mathbf{e}}^{(2)} = \begin{pmatrix} 1 & 2 \end{pmatrix}^\top$. D'après la section précédente, une dernière analyse récursive doit être lancée sur le neurone binaire extrait pour $\hat{e}_1 = 0$. Il est pourtant évident que ce dernier ne pourra jamais être vrai. Pour qu'il le soit il faudrait en effet pouvoir augmenter encore plus la valeur de $\hat{e}_2$ afin de compenser le passage de $\hat{e}_1 = 1$ à $\hat{e}_1 = 0$, ce qui est impossible puisqu'elle est déjà à son maximum dans $\hat{\mathbf{e}}_2$.

Il existe donc une valeur limite $\hat{e}_1^{(-)}$ en-deça de laquelle il n'est plus nécessaire d'analyser les neurones binaires extraits. Cette dernière est donnée par la mise en équation du fait que le neurone binaire extrait pour cette valeur peut encore être vrai, tandis qu'il ne peut plus l'être pour $\hat{e}_1^{(-)} - 1$:

$$\hat{w}_1 \hat{e}_1^{(-)} + \sum_{i=2}^{d} \hat{w}_i n_i > -w_0 \qquad \text{et} \qquad \hat{w}_1 (\hat{e}_1^{(-)} - 1) + \sum_{i=2}^{d} \hat{w}_i n_i \leq -w_0, \tag{9.8}$$

où les valeurs $n_i|_{i=2}^d$ sont les valeurs maximales admissibles pour les entrées réduites $\hat{e}_i|_{i=2}^d$.

La valeur de $\hat{e}_1^{(-)}$ correspond donc au seul entier compris dans l'intervalle :

$$-\frac{1}{\hat{w}_1}\left(w_0+\sum_{i=2}^{d}\hat{w}_i n_i\right) < \hat{e}_1^{(-)} \leq 1-\frac{1}{\hat{w}_1}\left(w_0+\sum_{i=2}^{d}\hat{w}_i n_i\right). \tag{9.9}$$

Elle s'exprime donc, comme $\hat{e}_1^{(+)}$, au moyen de la fonction « arrondi inférieur » :

$$\hat{e}_1^{(-)} = 1-\left\lfloor\frac{1}{\hat{w}_1}\left(w_0+\sum_{i=2}^{d}\hat{w}_i n_i\right)\right\rfloor. \tag{9.10}$$

En conclusion, la méthode d'analyse, décrite en pseudo-code par l'algorithme 9.1, doit tout d'abord calculer les deux valeurs $\hat{e}_1^{(+)}$ et $\hat{e}_1^{(-)}$ en utilisant respectivement les expressions (9.4) et (9.10). Elle devra alors lancer récursivement l'analyse des neurones binaires extraits pour $\hat{e}_1 \in [\![\min(\hat{e}_1^{(+)}-1, n_1)\,;\max(0,\hat{e}_1^{(-)})]\!]$. Pour cette variable réduite, il y a donc au pire n_1+1 lancements récursifs, ce qui implique une complexité globale en $\mathcal{O}(\prod_{i=1}^{d}(n_i+1)) \leq \mathcal{O}(2^n)$.

Grâce à cet algorithme, tout neurone binaire peut être traduit en famille génératrice et faire l'objet d'une optimisation par l'une des méthodes exposées dans la partie suivante.

Algorithme 9.1
Analyse d'un neurone binaire en famille génératrice.

```
/* Fonction analyser:
Retourne la famille génératrice d'un neurone binaire.
   Paramètres:
/W/ - tableau trié des poids synaptiques des entrées réduites
/x/ - tableau des valeurs maximales des entrées réduites
/wo/ - poids synaptique auxiliaire
/d/ - nombre d'entrées réduites */

fonction analyser(W,x,wo,d)=famillegeneratrice {
        // /x1min/ - valeur minimale de x1
        entier x1min = 1 - floor(wo/W[1]);
        // /x1max/ - valeur maximale de x1
        entier x1max = 1 - floor(wo/W[1] + somme(W[i],i=2:d)/W[1]);
        // /extraction/ - valeur de x1 pour laquelle un neurone binaire est extrait
        entier extraction = 0;
        // /famille/ - famille génératrice construite
        // /sous-famille/ - famille génératrice d'un neurone binaire extrait
        famillegeneratrice famille = vide;
        famillegeneratrice sous-famille vide;

        // test d'existence du premier vecteur générateur
        si (x1min > x[1]) {
                extraction = x[1];
        }
        sinon {
                extraction = x1min - 1;
                famille += {x1min,0,0,...,0};
        }

        // /nouveau_wo/ - poids synaptique auxiliaire du neurone binaire extrait
        flottant nouveau_wo = wo - extraction * W[1];

        pour (i=x1min:x1max) {
                // lancement récursif
                sous-famille = analyser(W+1,x+1,nouveau_wo,d-1);

                // ajout de la première composante à chaque vecteurs
                sous-famille.ajouter_tete(extraction);

                // concaténation de la sous-famille avec la famille génératrice
                famille += sous-famille;

                // calcul du nouveau poids synaptique auxiliaire
                nouveau_wo = nouveau_wo + W[1];
        }

        retourner(famille);
}
```

Troisième partie

Synthèse de réseaux de neurones artificiels

Sommaire de la partie

10

Un regard sur la programmation de neurones binaires

L'arrivée du premier neurone artificiel conduisit de nombreuses recherches à s'intéresser à la logique à seuil (logique TL pour *threshold logic*), principalement comme alternative à la logique TTL [1] alors naissante [Dertouzos, 1964 ; Muroga, 1971 ; Kohavi, 1978]. Contrairement aux portes logiques standards, telle la porte non-et, les portes logiques à seuil réalisent en effet des fonctions logiques bien plus complexes, promettant ainsi des circuits logiques globalement beaucoup plus concis. Néanmoins, les première portes logiques à seuil, basées sur des circuits résistifs, étaient handicapées par leur coup de fabrication élevé et leur temps de commutation supérieur à celui du transistor.

L'invention du *neuristor* en 1962 par H. Crane [Crane, 1962], inspirée des fibres nerveuses biologiques, finit d'assoire la supériorité de la logique TL en termes de densité d'intégration puisqu'elle permettait également à deux lignes d'un circuit intégré de se croiser sans interférer. Néanmoins, la très rapide miniaturisation des transistors a rendu cette innovatiom caduc pratiquement dès sa sortie. Malgré tout, plusieurs équipes travaillent toujours dans ce domaine et espèrent bien pouvoir proposer une

[1] Technologie basée sur le transistor bipolaire.

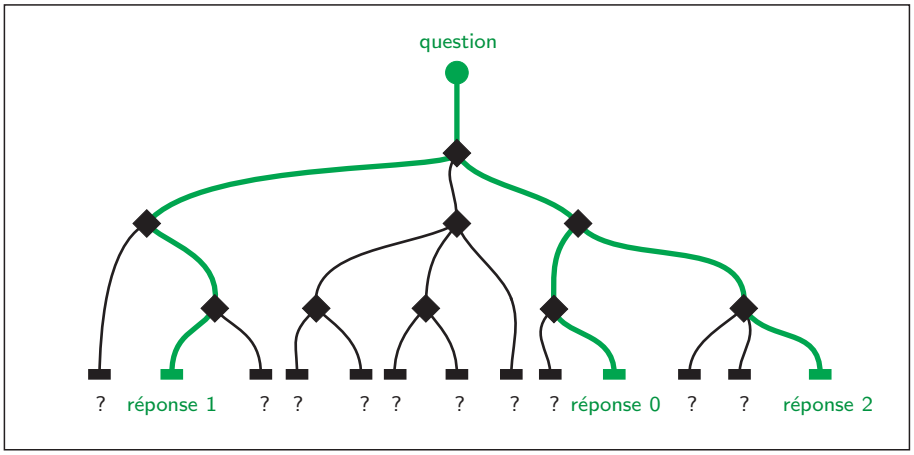


Figure 10.1
Résolution d'un problème de complexité NP par un algorithme polynomial non déterministe.

porte logique à seuil compétitive dès que la course à la miniaturisation dont fait l'objet le transistor aura atteint ses limites [Ozdemir et al., 1996 ; Beiu et al., 2003 ; Neville, 2006].

La manque d'intérêt pour la logique TL a également pour origine l'absence d'outil algébrique fiable permettant d'automatiser la conception d'un circuit logique. Comme nous l'avons déjà expliqué dans ce manuscrit, ce manque est dû à la complexité inhérente aux fonctions logiques à seuil. Par exemple, la simple question de l'appartenance d'une fonction logique quelconque à cette famille est connue pour être l'un des nombreux problèmes de classe NP-complet [Hegedüs et Megiddo, 1996], c'est-à-dire soluble par un algorithme non déterministe de complexité polynomiale [2]. Par voie de conséquence, toutes les autres questions relatives à la mise au point d'outils algébriques dédiés à la logique à seuil font alors également partie de la classe NP :

– minimiser le nombre de portes logique à seuil requis ;
– homogénéiser la distribution des portes sur le circuit.

La figure 10.1 illustre la nature de la complexité NP par un parcours d'arbre dont chaque feuille est atteinte depuis la racine par un algorithme polynomial. À la différence de la classe P, seules certaines feuilles correspondent à une réponse au problème et il n'y a *a priori* pas de moyen de savoir à l'avance quel chemin suivre sinon que d'en choisir un arbitrairement en espérant qu'il conduise à l'une d'elles.

Étant donné la très forte similitude existant entre la thématique des recherches exposées par ce manuscrit et le domaine de la synthèse de circuits TL, il est naturel d'y rechercher des solutions à la programmation de neurones binaires. Néanmoins,

[2] En théorie de la complexité, un algorithme déterministe de complexité polynomiale — classe P — est jugé acceptable lorsqu'il s'agit de résoudre une tâche compliquée. Il a déjà été établi l'inclusion $P \subseteq NP$, et la question réciproque $P = NP$ fait l'objet de recherches poussées, dont l'aboutissement sera récompensé par la prix Clay ($ 1 000 000). Un problème de la classe NP-complet est un problème qui, s'il peut être résolu par un algorithme de la classe P, suffit à prouver l'égalité de classes $P = NP$.

aucunes d'entre-elles ne résout de façon satisfaisante la problématique générale de la synthèse d'une fonction logique en logique TL, ou en un réseau de neurones binaires, pour l'une ou l'autre de ces raisons :
– systématisation difficilement envisageable ;
– utilisation déraisonnable de ressources informatiques ;
– domaine d'application trop restrictif ;
– validité du résultat non garantie.
Ce chapitre en présente quelques-unes parmis les plus incontournables et les plus récentes.

10.1 La synthèse par inégalités

Cette section regroupe les méthodes dont le but est d'établir, de façon plus ou moins rapide, un système d'inégalités ordonnant les poids synaptiques entre eux. De cette manière le neurone binaire est obtenu en fixant arbitrairement la valeurs des poids synaptiques de telle sorte à ce qu'ils satisfassent le système. L'inconvénient de ces méthodes est qu'elles ne permettent pas de contrôler la marge de fonctionnement des neurones binaires obtenus.

10.1.1 Synthèse naïve

Afin de présenter concrètement le problème et les difficultés qui l'accompagnent, cette sous-section détaille l'approche dite « naïve » de la synthèse de neurones binaires au travers d'un exemple [Kohavi, 1978]. Nous verrons ainsi que par nature, une telle façon de procéder est très difficile à systématiser et qu'elle ne donne aucun moyen de prévoir la valeur ϵ de la marge de fonctionnement du neurone obtenu (voir le chapitre 7 en page 70) sinon par « essai-erreur ».

La fonction logique que nous utilisons pour la démonstration est la fonction $\mathcal{L}_1$ de $\mathbb{L}_s^+$ suivante :

$$\mathcal{L}_1(e_1, e_2, e_3, e_4) \equiv e_1 e_2 e_3 \overline{e}_4 + e_2 e_3 e_4.$$

Comme $\mathcal{L}_1$ dépend de quatre variables, le neurone binaire qui la réalise aura vraisemblablement quatre entrées lui-aussi. Elles sont respectivement pondérées par $\mathbf{w} = \begin{pmatrix} w_1 & w_2 & w_3 & w_4 \end{pmatrix}$. Conformément aux notations des parties précédentes, le poids synaptique auxiliaire est noté w_0. La synthèse de w_0 et $\mathbf{w}$ suit les trois étapes suivantes :

1. établir le système d'inégalités sur $\mathbf{w}$ définissant $\mathcal{L}_1$;
2. trouver un vecteur $\mathbf{w}$ vérifiant le système ;
3. calculer w_0.

Établissement des inégalités Si $\mathbf{e}^{(+)}$ désigne un vecteur d'entrée pour lequel $\mathcal{L}_1(\mathbf{e}^{(+)}) \equiv \mathsf{vrai}$ et $\mathbf{e}^{(-)}$ un vecteur d'entrée tel que $\mathcal{L}_1(\mathbf{e}^{(-)}) \equiv \mathsf{faux}$, alors la composition de la fonction d'agrégation et de la fonction d'activation du neurone binaire donne les deux inégalités suivantes :

$$\mathbf{w}(\mathbf{e}^{(+)})^\top > -w_0, \tag{10.1a}$$
$$\mathbf{w}(\mathbf{e}^{(-)})^\top < -w_0. \tag{10.1b}$$

De ces deux inégalité peut se déduire l'inégalité (10.2), indépendante de w_0 :

$$\mathbf{w}(\mathbf{e}^{(+)})^\top > \mathbf{w}(\mathbf{e}^{(-)})^\top. \tag{10.2}$$

En regroupant de cette façon tous les vecteurs pour lesquels $\mathcal{L}_1$ est vrai avec tous ceux pour lesquelles $\mathcal{L}_1$ vaut faux, on obtient le système d'inéquations suivant [3] :

$$\begin{array}{lll} w_1+w_2+w_3 > w_1+w_2+w_4, & & w_2+w_3+w_4 > w_1+w_2+w_4, \\ w_1+w_2+w_3 > w_1+w_3+w_4, & & w_2+w_3+w_4 > w_1+w_3+w_4, \\ w_1+w_2+w_3 > w_2+w_3 & \text{et} & w_2+w_3+w_4 > w_2+w_3. \end{array} \tag{10.3}$$

Résolution du système En observant le système (10.3), il apparaît que l'ensemble des inégalités se simplifie en :

$$\begin{array}{lll} w_3 > w_4, & & w_3 > w_1, \\ w_2 > w_4, & & w_2 > w_1, \\ w_1 > 0, & \text{et} & w_4 > 0. \end{array} \tag{10.4}$$

Parmi les nombreuses affectations permises par ces inégalités, nous avons arbitrairement retenu $w_1 = w_4 = 1$ et $w_2 = w_3 = 2$.

Calcul du poids synaptique auxiliaire Pour les poids synaptiques retenus dans l'étape précédente, la valeur du poids synaptique auxiliaire doit être celle donnant la meilleure marge de fonctionnement au neurone binaire à synthétiser. L'intervalle des valeurs admissibles pour w_0 se déduit des inégalités suivantes, qui sont obtenues à partir des inégalités (10.1a) où $\mathbf{e}^{(+)}$ et $\mathbf{e}^{(-)}$ balayent les 2^n vecteurs d'entrée possibles [4] :

$$\begin{array}{ll} w_1+w_2+w_4+w_0 < 0, & w_1+w_2+w_3+w_0 > 0, \\ w_1+w_3+w_4+w_0 < 0, & w_2+w_3+w_4+w_0 > 0, \\ \text{et} \quad w_2+w_3+w_0 < 0. & \end{array} \tag{10.5}$$

Il vient donc $w_0 \in]-5\,;-4[$, ce qui suggère de choisir $w_0 = -4{,}5$ afin de maximiser ϵ.

[3] Pour plus de clareté, nous avons omis les inégalités qui peuvent se déduire des autres.

[4] Nous avons une fois encore omis les inégalités qui peuvent se déduire des autres.

Bilan Bien que très simple en apparence, cette procédure manuelle est très difficile à systématiser de façon efficace en une procédure algorithmique. En effet, le caractère NP complexe du problème, évoqué dans les pages précédentes, se manifeste plusieurs fois sous les formes suivantes :

- lors de l'établissement du système d'inéquations, savoir si une inégalité particulière se déduit des autres ne peut se faire qu'une fois tout le système obtenu ;
- la détermination d'un poids synaptique vérifiant une inégalité particulière peut être remise en cause par une autre inégalité ;
- si la marge de fonctionnement obtenue ne convient pas, il faut recommencer l'affectation.

Apport des familles génératrices En remarquant que $\mathcal{L}$ se factorise en $e_2e_3(e_1+e_4)$, on déduit facilement la famille génératrice de $\mathcal{L}$ ainsi que sa famille génératrice conjointe :

$$\{\mathcal{L}\} = \begin{Bmatrix} 2 & 1 \end{Bmatrix}, \qquad {}^{c}\{\mathcal{L}\} = \begin{Bmatrix} 2 & 0 \\ 1 & 2 \end{Bmatrix}.$$

où les deux variables réduites utilisées sont respectivement $\hat{e}_1 \to \{e_2\,;e_3\}$ et $\hat{e}_2 \to \{e_1\,;e_4\}$. En reprenant la méthode en trois étapes décrite par cette sous-section, et en appellant $\hat{\mathbf{w}} = \begin{pmatrix} \hat{w}_1 & \hat{w}_2 \end{pmatrix}^\top$ les poids synaptiques des entrées réduites correspondantes, il vient trivialement :

$$2\hat{w}_1 + \hat{w}_2 > -w_0 \tag{10.6a}$$

$$2\hat{w}_1 < -w_0 \tag{10.6b}$$

$$\text{et} \quad \hat{w}_1 + 2\hat{w}_2 < -w_0. \tag{10.6c}$$

L'observation de ce système conduit alors très simplement à l'inégalité suivante :

$$\hat{w}_2 > \hat{w}_1 > 0, \tag{10.7}$$

à partir de laquelle des valeurs arbitraires de $\hat{w}_1$ et $\hat{w}_2$ peuvent être obtenues. Bien que l'utilisation des familles génératrices ne permette toujours pas à cette méthode de déterminer à l'avance la marge de fonctionnement du neurone binaire synthétisé, son gain en complexité est particulièrement évident.

10.1.2 Synthèse par descripteurs de Chow

Dans son article de 1961 [Chow, 1961 ; Picton, 2001], C. K. Chow montra qu'un ensemble de descripteurs permettait d'identifier de façon unique chaque fonction logique à seuil — chaque neurone binaire. Il proposa ainsi d'utiliser ces derniers comme clefs d'une table de correspondance qui stockerait les valeurs des poids synaptiques de l'intégralité des fonctions logiques à seuil d'un certain nombre de variables.

Si cette méthode s'avère naturellement très efficace en temps de calcul pour des neurones binaires ayant un très petit nombre d'entrées, elle devient vite problématique dès que ce dernier augmente. D'une part, les descripteurs dont il est question n'identifient plus de façon unique une fonction logique à seuil dès lors qu'elle a plus de huit variables [Picton, 2001]. D'autre part, la taille des tableaux de correspondance devient vite rédhibitoire ; l'ensemble des fonctions logiques à seuil de huit variables représentant déjà quelques 17 561 539 552 946 entrées [Hassoun, 1995], ce qui nécessite un adressage sur un minimum de 44 bits et une mémoire de 2 500 000 Go.

Descripteurs Soit une fonction logique à seuil $\mathcal{L}$ de $\mathbb{L}_s^+$ et soit $\mathbb{B}_+^n(\mathcal{L})$ l'ensemble des vecteurs d'entrée de $\mathbb{B}^n$ évaluant $\mathcal{L}$ à vrai et $\mathbb{B}_-^n(\mathcal{L})$ celui des vecteurs l'évaluant à faux. Si $n \leq 8$, alors les descripteurs suivant décrivent $\mathcal{L}$ de façon unique :

$$\mathbf{a}(\mathcal{L}) = \begin{pmatrix} a_1(\mathcal{L}) & a_2(\mathcal{L}) & \dots & a_n(\mathcal{L}) \end{pmatrix} = \sum_{\mathbf{e} \in \mathbb{B}_+^n(\mathcal{L})} \mathbf{e} \tag{10.8a}$$

$$\text{et} \quad m(\mathcal{L}) = \operatorname{card}\left(\mathbb{B}_+^n(\mathcal{L})\right), \tag{10.8b}$$

où la fonction card retourne le nombre d'éléments de l'ensemble qui lui est donné en argument.

Ces descripteurs, au travers du vecteur $\mathbf{a}(\mathcal{L})/m(\mathcal{L})$, peuvent s'interpréter comme le centre de gravité de l'ensemble $\mathbb{B}_+^n(\mathcal{L})$. La limite de validité à huit variables s'explique alors par un gain de complexité [5] permettant à deux fonctions logiques à seuil distinctes de partager le même centre de gravité.

Utilisation des descripteurs pour la synthèse Dans la mesure où $\mathcal{L}$ est une fonction logique à seuil, C. K. Chow a démontré les relations suivantes, entre ses descripteurs et les poids synaptiques $\mathbf{w}$ des neurones binaires réalisant $\mathcal{L}$:

$$\text{si} \quad a_i(\mathcal{L}) < \frac{1}{2} m(\mathcal{L}) \quad \text{alors} \quad w_i < 0, \tag{10.9a}$$

$$\text{si} \quad a_i(\mathcal{L}) > \frac{1}{2} m(\mathcal{L}) \quad \text{alors} \quad w_i > 0, \tag{10.9b}$$

$$\text{si} \quad a_i(\mathcal{L}) = \frac{1}{2} m(\mathcal{L}) \quad \text{alors} \quad w_i = 0, \tag{10.9c}$$

$$\text{si} \quad a_i(\mathcal{L}) < a_j(\mathcal{L}) \quad \text{alors} \quad w_i < w_j, \tag{10.9d}$$

$$\text{si} \quad a_i(\mathcal{L}) = a_j(\mathcal{L}) \quad \text{alors} \quad w_i = w_j. \tag{10.9e}$$

À la manière de la méthode naïve de la sous-section précédente, l'utilisation des descripteurs permet elle aussi de créer — bien plus simplement — un système d'inégalités dont la résolution permet de trouver $\mathbf{w}$. Le poids synaptique auxiliaire se

[5] Au-delà de huit variables, l'équivalence entre fonction logique à seuil et fonction logique totalement monotone n'est plus vraie, ce qui invalide toutes les méthodes qui s'appuient dessus [Muroga, 1971].

déduit alors de façon similaire, en déterminant son intervalle de valeurs admissibles et en se positionnant au milieu.

Toutefois, le système obtenu ici ne contient pas forcément toutes les contraintes relationnelles nécessaire à la synthèse correcte d'un neurone binaire. Cela implique une assez grande probabilité que l'intervalle des valeurs possibles de w_0 soit l'ensemble vide, signifiant alors qu'il faut recommencer l'affectation des poids synaptiques en espérant avoir plus de chance.

Exemple 10.1 — Conception d'un neurone binaire par les descripteurs de Chow.

Soit la fonction logique de $\mathbb{L}_s^+$ suivante :

$$\begin{aligned}\mathcal{L}_2(e_1,e_2,e_3,e_4,e_5) &\equiv e_1e_2 + (e_1+e_2)(e_3e_4+e_3e_5+e_4e_5) + e_3e_4e_5\\ &\equiv (e_1 \overset{2}{\top} e_2 \overset{2}{\top} (e_3 \overset{2}{\top} e_4 \overset{2}{\top} e_5) \overset{2}{\top} (e_3 \overset{3}{\top} e_4 \overset{3}{\top} e_5)).\end{aligned}$$

Les vecteurs d'entrée de $\mathbb{B}_+^n(\mathcal{L})$ sont :

$$\begin{array}{lll}
\mathbf{e}^{(7)} = \begin{pmatrix}0&0&1&1&1\end{pmatrix}^\top, & \mathbf{e}^{(11)} = \begin{pmatrix}0&1&0&1&1\end{pmatrix}^\top, & \mathbf{e}^{(13)} = \begin{pmatrix}0&1&1&0&1\end{pmatrix}^\top,\\
\mathbf{e}^{(14)} = \begin{pmatrix}0&1&1&1&0\end{pmatrix}^\top, & \mathbf{e}^{(15)} = \begin{pmatrix}0&1&1&1&1\end{pmatrix}^\top, & \mathbf{e}^{(19)} = \begin{pmatrix}1&0&0&1&1\end{pmatrix}^\top,\\
\mathbf{e}^{(21)} = \begin{pmatrix}1&0&1&0&1\end{pmatrix}^\top, & \mathbf{e}^{(22)} = \begin{pmatrix}1&0&1&1&0\end{pmatrix}^\top, & \mathbf{e}^{(23)} = \begin{pmatrix}1&0&1&1&1\end{pmatrix}^\top,\\
\mathbf{e}^{(24)} = \begin{pmatrix}1&1&0&0&0\end{pmatrix}^\top, & \mathbf{e}^{(25)} = \begin{pmatrix}1&1&0&0&1\end{pmatrix}^\top, & \mathbf{e}^{(26)} = \begin{pmatrix}1&1&0&1&0\end{pmatrix}^\top,\\
\mathbf{e}^{(27)} = \begin{pmatrix}1&1&0&1&1\end{pmatrix}^\top, & \mathbf{e}^{(28)} = \begin{pmatrix}1&1&1&0&0\end{pmatrix}^\top, & \mathbf{e}^{(29)} = \begin{pmatrix}1&1&1&0&1\end{pmatrix}^\top,\\
\mathbf{e}^{(30)} = \begin{pmatrix}1&1&1&1&0\end{pmatrix}^\top, & \mathbf{e}^{(31)} = \begin{pmatrix}1&1&1&1&1\end{pmatrix}^\top. &
\end{array}$$

Il est ainsi très simple d'en déduire la valeur des quatre descripteurs de Chow en appliquant directement le jeu d'équations (10.8) les définissant :

$$m(\mathcal{L}) = 17 \qquad \text{et} \qquad \mathbf{a}(\mathcal{L}) = \begin{pmatrix}12&12&11&11&11\end{pmatrix}^\top.$$

Grâce aux relations entre descripteurs et poids synaptiques, on en déduit les inégalités suivantes :

$$w_1 > 0, \quad w_2 > 0, \quad w_3 > 0, \quad w_4 > 0, \quad w_5 > 0 \quad \text{et} \quad w_1 = w_2 > w_3 = w_4 = w_5.$$

Les poids synaptiques sont alors fixés arbitrairement de la façon suivante : $w_1 = 1{,}5$, $w_2 = 1{,}5$, $w_3 = 1$, $w_4 = 1$ et $w_5 = 1$, en accord avec ce système. De la même façon que dans la sous-section précédente, w_0 est choisi au centre de l'intervalle de ses valeurs admissibles $]-2{,}5\,;-3[$.
Si, de façon toute autant légitime, nous avions choisi les poids synaptiques $w_1 = 2$, $w_2 = 2$, $w_3 = 1$, $w_4 = 1$ et $w_5 = 1$, alors cette fois-ci l'intervalle admissible pour w_0 aurait été vide et $\mathcal{L}_1$ aurait été jugée non réalisable avec ces valeurs. Les descripteurs de Chow ne conduisent donc pas à un système d'inégalités complet.

Bilan En comparaison avec la méthode naïve, les descripteurs de Chow ont l'avantage de conduire directement à un système d'inéquations très simple. Malheureusement, l'utilisation de ce raccourci se paye par l'absence potentielle de certaines

contraintes qui peuvent conduire à la fausse conclusion que la fonction logique à synthétiser n'est pas une fonction logique à seuil. Par ailleurs, le choix des poids synaptiques étant là encore arbitraire, il ne permet pas lui non plus de contrôler la valeur de la marge de fonctionnement du neurone binaire obtenu.

En définitive, l'intérêt majeur de cette méthode réside dans l'unicité des descripteurs, qui leur permet d'associer très simplement une fonction logique à seuil d'au plus huit variables aux poids synaptiques qui la définissent. Ce recours aux tables de correspondances en fait d'ailleurs la méthode de synthèse de neurones binaires la plus rapide de toutes celles que nous présenterons dans cette partie, la nôtre comprise.

Apport des familles génératrices Dans le cas des descripteurs de Chow, l'utilisation des familles génératrices n'apporte pas grand chose. Pire encore, la restriction d'une fonction logique à seuil aux seules données de ses vecteurs générateurs ne permet pas de calculer le centre de gravité de son ensemble $\mathbb{B}^n_+$ et donc ses descripteurs. Pour cela, il faudrait en effet régénérer tous les vecteurs de cet ensemble à partir de la famille génératrice. En plus du temps de calcul supplémentaire, il faudrait également éliminer les vecteurs réduits générés plusieurs fois.

10.1.3 Synthèse par bifurcation de seuils

À la fin de l'année 2006, F. Chen et al. publièrent un nouveau moyen d'obtenir le système d'inégalités définissant les poids synaptiques d'un neurone binaire [Chen et al., 2006]. Leur méthode utilise la base naturelle $\{\mathbf{b}^{(1)} \quad \mathbf{b}^{(2)} \quad \cdots \quad \mathbf{b}^{(2^{n-1})}\}$, de l'espace d'entrée $\mathbb{B}^n$, pour exprimer la valeur de la fonction d'agrégation pour un vecteur d'entrée donné :

$$\mathbf{b}^{(1)} = \begin{pmatrix} 0 & 0 & \cdots & 0 & 1 \end{pmatrix}^\top, \tag{10.10a}$$

$$\mathbf{b}^{(2)} = \begin{pmatrix} 0 & 0 & \cdots & 1 & 0 \end{pmatrix}^\top, \tag{10.10b}$$

$$\vdots$$

$$\text{et} \quad \mathbf{b}^{(2^{n-1})} = \begin{pmatrix} 1 & 0 & \cdots & 0 & 0 \end{pmatrix}^\top. \tag{10.10c}$$

La valeur de la fonction d'agrégation d'un neurone binaire défini par $\mathbf{w}$ et w_0 se décompose ainsi de la façon suivante :

$$\begin{aligned}
&\mathcal{A}(\mathbf{e}^{(0)}) = w_0, && \mathcal{A}(\mathbf{e}^{(6)}) = \mathcal{A}(\mathbf{e}^{(2)}) + \mathcal{A}(\mathbf{b}^{(4)}) - w_0,\\
&\mathcal{A}(\mathbf{e}^{(1)}) = \mathcal{A}(\mathbf{b}^{(1)}), && \mathcal{A}(\mathbf{e}^{(7)}) = \mathcal{A}(\mathbf{e}^{(3)}) + \mathcal{A}(\mathbf{b}^{(4)}) - w_0,\\
&\mathcal{A}(\mathbf{e}^{(2)}) = \mathcal{A}(\mathbf{b}^{(2)}), && \mathcal{A}(\mathbf{e}^{(8)}) = \mathcal{A}(\mathbf{b}^{(8)}),\\
&\mathcal{A}(\mathbf{e}^{(3)}) = \mathcal{A}(\mathbf{e}^{(1)}) + \mathcal{A}(\mathbf{b}^{(2)}) - w_0, && \mathcal{A}(\mathbf{e}^{(9)}) = \mathcal{A}(\mathbf{e}^{(1)}) + \mathcal{A}(\mathbf{b}^{(8)}) - w_0,\\
&\mathcal{A}(\mathbf{e}^{(4)}) = \mathcal{A}(\mathbf{b}^{(4)}), && \vdots\\
&\mathcal{A}(\mathbf{e}^{(5)}) = \mathcal{A}(\mathbf{e}^{(1)}) + \mathcal{A}(\mathbf{b}^{(4)}) - w_0, && \mathcal{A}(\mathbf{e}^{(2^{n+1}-1)}) = \mathcal{A}(\mathbf{e}^{(2^n-1)}) + \mathcal{A}(\mathbf{b}^{(2^n)}) - w_0.
\end{aligned} \tag{10.11}$$

Par exemple, si le neurone binaire a quatre entrées, alors sa fonction d'agrégation s'écrit, pour le vecteur d'entrée $\mathbf{e}^{(5)} = \begin{pmatrix} 0 & 1 & 0 & 1 \end{pmatrix}^\top$:

$$\begin{aligned} \mathcal{A}(\mathbf{e}^{(5)}) &= \mathbf{w} \left(\mathbf{b}^{(4)} + \mathbf{b}^{(1)} \right)^\top + w_0, \\ &= \mathcal{A}(\mathbf{b}^{(4)}) + \mathcal{A}(\mathbf{b}^{(1)}) - w_0. \end{aligned} \tag{10.12}$$

Synthèse des poids synaptiques En utilisant deux propriétés particulières de ces décompositions, appelées « symétrie » et « auto-reproduction [6] », les auteurs montrent qu'il est possible d'établir les inégalités de base que les différentes fonctions d'agrégation doivent vérifier ; en particulier les valeurs $\mathcal{A}(\mathbf{b}^{(2^i)})|_{i=0}^{n}$, puisqu'elles dépendent directement des poids synaptiques à déterminer.

Ces deux propriétés découlent directement de l'arithmétique de l'écriture en base 2 des entiers successifs. En donner une définition formelle sur n variables étant inutilement compliquée, nous les exprimerons pour deux et trois variables et laisserons le lecteur généraliser.

Pour $n = 2$, la symétrie implique les égalités suivantes :

$$\forall i \in [\![0\,;1]\!]\,, \quad \mathcal{A}(\mathbf{e}^{(i)}) + \mathcal{A}(\mathbf{e}^{(3-i)}) = \text{cste}. \tag{10.13}$$

Pour $n = 3$, les égalités deviennent :

$$\forall i \in [\![0\,;3]\!]\,, \quad \mathcal{A}(\mathbf{e}^{(i)}) + \mathcal{A}(\mathbf{e}^{(7-i)}) = \text{cste}, \tag{10.14a}$$
$$\forall i \in [\![0\,;1]\!]\,, \quad \mathcal{A}(\mathbf{e}^{(i)}) + \mathcal{A}(\mathbf{e}^{(3-i)}) = \text{cste}, \tag{10.14b}$$
$$\forall i \in [\![0\,;1]\!]\,, \quad \mathcal{A}(\mathbf{e}^{(4+i)}) + \mathcal{A}(\mathbf{e}^{(7-i)}) = \text{cste}. \tag{10.14c}$$

Pour sa part, l'auto-reproduction implique les égalités suivantes si $n = 2$:

$$\forall i \in [\![0\,;1]\!]\,, \quad \mathcal{A}(\mathbf{e}^{(2+i)}) - \mathcal{A}(\mathbf{e}^{(i)}) = \text{cste}, \tag{10.15}$$

Elles deviennent, pour $n = 3$:

$$\forall i \in [\![0\,;3]\!]\,, \quad \mathcal{A}(\mathbf{e}^{(4+i)}) - \mathcal{A}(\mathbf{e}^{(i)}) = \text{cste}, \tag{10.16a}$$
$$\forall i \in [\![0\,;1]\!]\,, \quad \mathcal{A}(\mathbf{e}^{(2+i)}) + \mathcal{A}(\mathbf{e}^{(i)}) = \text{cste}, \tag{10.16b}$$
$$\forall i \in [\![0\,;1]\!]\,, \quad \mathcal{A}(\mathbf{e}^{(6+i)}) + \mathcal{A}(\mathbf{e}^{(4+i)}) = \text{cste}. \tag{10.16c}$$

La procédure de synthèse est dictée par les remarques suivante, pour une fonction logique à seuil $\mathcal{L}$ connue :
- $\mathcal{L}$ permet de positionner chaque valeur formelle de l'équation (10.11) par rapport à zéro ;
- $\mathcal{L}$ permet également d'établir plusieurs inégalités entre les valeurs formelles de l'équation (10.11) en exprimant que si $\mathcal{L}(\mathbf{e}^{(i)}) \equiv \mathsf{faux}$ et $\mathcal{L}(\mathbf{e}^{(j)}) \equiv \mathsf{vrai}$, alors $\mathcal{A}(\mathbf{e}^{(i)}) < \mathcal{A}(\mathbf{e}^{(j)})$;
- la symétrie et l'auto-reproduction permettent d'éliminer automatiquement les redondances du système d'inégalités obtenu après les deux points précédents.

[6] Traduction libre de *self-reproduction*.

Le système d'inégalités ainsi obtenu est ensuite exprimé en fonction des n fonctions d'agrégation de base $\mathcal{A}(\mathbf{b}^{(2^i)})|_{i=0}^{n-1}$, dont des valeurs peuvent maintenant être fixées.

Enfin ,les n poids synaptiques s'en déduisent de la façon suivante :

$$w_0 = \mathcal{A}(\mathbf{b}^{(0)}), \tag{10.17a}$$
$$w_1 = \mathcal{A}(\mathbf{b}^{(1)}) - \mathcal{A}(\mathbf{b}^{(0)}), \tag{10.17b}$$
$$w_2 = \mathcal{A}(\mathbf{b}^{(2)}) - \mathcal{A}(\mathbf{b}^{(0)}), \tag{10.17c}$$
$$\vdots \tag{10.17d}$$
$$w_n = \mathcal{A}(\mathbf{b}^{(2^{n-1})}) - \mathcal{A}(\mathbf{b}^{(0)}). \tag{10.17e}$$

Exemple 10.2 — Synthèse par bifurcation de seuils.
Synthétisons la fonction logique à seuil $\mathcal{L}_3$ de cinq variables suivante :

$$\mathcal{L}_3(\mathbf{e}) \equiv (e_1 \overset{3}{\top} e_2 \overset{3}{\top} e_3 \overset{3}{\top} (e_4 \overset{2}{\top} e_5)),$$

dont les sept vecteurs d'entrée qu'elle évalue à vrai sont :

$$\mathbf{e}^{(15)} = \begin{pmatrix}0 & 1 & 1 & 1 & 1\end{pmatrix}^\top, \qquad \mathbf{e}^{(23)} = \begin{pmatrix}1 & 0 & 1 & 1 & 1\end{pmatrix}^\top,$$
$$\mathbf{e}^{(27)} = \begin{pmatrix}1 & 1 & 0 & 1 & 1\end{pmatrix}^\top, \qquad \mathbf{e}^{(28)} = \begin{pmatrix}1 & 1 & 1 & 0 & 0\end{pmatrix}^\top,$$
$$\mathbf{e}^{(29)} = \begin{pmatrix}1 & 1 & 1 & 0 & 1\end{pmatrix}^\top, \qquad \mathbf{e}^{(30)} = \begin{pmatrix}1 & 1 & 1 & 1 & 0\end{pmatrix}^\top,$$
$$\text{et} \quad \mathbf{e}^{(31)} = \begin{pmatrix}1 & 1 & 1 & 1 & 1\end{pmatrix}^\top.$$

Dès lors, aucun des vecteurs d'entrée de la base naturelle de $\mathbb{B}^n$ n'étant dans cette liste, il vient les cinq inégalités suivantes :

$$\mathcal{A}(\mathbf{b}^{(0)}) < 0, \qquad \mathcal{A}(\mathbf{b}^{(1)}) < 0, \qquad \mathcal{A}(\mathbf{b}^{(2)}) < 0,$$
$$\mathcal{A}(\mathbf{b}^{(4)}) < 0, \qquad \mathcal{A}(\mathbf{b}^{(8)}) < 0, \qquad \text{et} \quad \mathcal{A}(\mathbf{b}^{(16)}) < 0.$$

En écrivant, par exemple, que $\mathcal{A}(\mathbf{e}^{(28)}) > \mathcal{A}(\mathbf{e}^{(25)})$ alors il vient $\mathcal{A}(\mathbf{b}^{(4)}) > \mathcal{A}(\mathbf{b}^{(1)})$. De façon similaire, d'autres inégalités permettent d'établir :

$$\mathcal{A}(\mathbf{b}^{(4)}) > \mathcal{A}(\mathbf{b}^{(2)}), \qquad \mathcal{A}(\mathbf{b}^{(8)}) > \mathcal{A}(\mathbf{b}^{(2)}), \qquad \mathcal{A}(\mathbf{b}^{(8)}) > \mathcal{A}(\mathbf{b}^{(1)}),$$
$$\mathcal{A}(\mathbf{b}^{(16)}) > \mathcal{A}(\mathbf{b}^{(2)}) \quad \text{et} \quad \mathcal{A}(\mathbf{b}^{(16)}) > \mathcal{A}(\mathbf{b}^{(1)}).$$

Par ailleurs, puisque $\mathcal{A}(\mathbf{e}^{(15)}) > 0 > \mathcal{A}(\mathbf{e}^{(14)})$, alors :

$$\mathcal{A}(\mathbf{e}^{(8)}) + \mathcal{A}(\mathbf{e}^{(4)}) + \mathcal{A}(\mathbf{e}^{(2)}) + \mathcal{A}(\mathbf{e}^{(1)}) - 3\mathcal{A}(\mathbf{e}^{(0)}) > 0$$
$$> \mathcal{A}(\mathbf{e}^{(8)}) + \mathcal{A}(\mathbf{e}^{(4)}) + \mathcal{A}(\mathbf{e}^{(2)}) - 2\mathcal{A}(\mathbf{e}^{(0)}).$$

Une affectation admissible est donc :

$$\mathcal{A}(\mathbf{b}^{(0)}) = -3{,}25, \qquad \mathcal{A}(\mathbf{b}^{(1)}) = -3, \qquad \mathcal{A}(\mathbf{b}^{(2)}) = -3,$$
$$\mathcal{A}(\mathbf{b}^{(4)}) = -2, \qquad \mathcal{A}(\mathbf{b}^{(8)}) = -2, \qquad \text{et} \quad \mathcal{A}(\mathbf{b}^{(16)}) = -2.$$

Les poids synaptiques vallent quant à eux :

$$w_0 = \mathcal{A}(\mathbf{b}^{(0)}) = -3{,}25, \qquad w_1 = \mathcal{A}(\mathbf{b}^{(1)}) - \mathcal{A}(\mathbf{b}^{(0)}) = 0{,}25,$$
$$w_2 = \mathcal{A}(\mathbf{b}^{(2)}) - \mathcal{A}(\mathbf{b}^{(0)}) = 0{,}25, \qquad w_3 = \mathcal{A}(\mathbf{b}^{(4)}) - \mathcal{A}(\mathbf{b}^{(0)}) = 1{,}25,$$
$$w_4 = \mathcal{A}(\mathbf{b}^{(8)}) - \mathcal{A}(\mathbf{b}^{(0)}) = 1{,}25, \quad \text{et} \quad w_5 = \mathcal{A}(\mathbf{b}^{(16)}) - \mathcal{A}(\mathbf{b}^{(0)}) = 1{,}25.$$

Bilan Au travers de cet exemple, il apparaît de façon très claire que la synthèse par bifurcation de seuil est une méthode très difficile à systématiser tant l'intuition qu'elle requiert pour le choix des inégalités à établir est grande. Malgré tout, le fait de travailler avec des valeurs de fonction d'agrégation plutôt que des poids synaptiques permet d'entrevoir la possibilité d'orienter la synthèse pour qu'elle donne au neurone binaire une marge de fonctionnement prédéterminée. D'ailleurs, les auteurs vantent par anticipation [7] les qualités des neurones binaires qu'ils obtiennent. Pourtant, nos tests prouvent qu'ils sont globalement loin d'avoir une marge de fonctionnement optimale, et la méthode de synthèse que nous développerons dans les chapitres suivants en trouve de bien meilleurs.

En réalité, cette méthode avait été conçue à l'origine pour la réalisation d'une banque de neurones binaires réalisant toutes les fonctions logiques à seuil d'un nombre de variables donné [Chen et Chen, 2004, 2005 ; Chen et al., 2005]. C'est-à-dire qu'en étudiant les répercussions, sur la fonction logique à seuil, de la modification de la valeur d'une des fonctions d'agrégation de base, il est possible de parcourir itérativement l'ensemble des profils $\{\mathcal{A}(\mathbf{e}^{(i)})|_{i=0}^{2^n-1}\}$ réalisant des fonctions logiques à seuil différentes. Ainsi, et bien que les auteurs n'en fassent pas mention, la perspective de pouvoir générer ces banques automatiquement est relativement intéressante dans la mesure où cela permettrait d'utiliser pleinement les descripteurs de Chow ; les tables de correspondances n'étant pas encore disponibles jusqu'à huit variables. Il reste néanmoins à résoudre les contraintes de stockage matériel que cela soulève.

Apport des familles génératrices Comme pour la procédure naïve, dont cette méthode n'est finalement pas si éloignée, le principal apport des familles génératrices est qu'elles listent directement les vecteurs d'entrée significatifs pour l'établissement du système d'inégalités. Cependant, les propriétés de symétrie et d'autoreproduction, qui permettent d'éviter le calcul d'inégalités redondantes, n'existent plus avec les familles génératrices, mais peuvent parfaitement s'étendre aux fonctions spectre. Cela étant, nous pensons que le gain engendré par l'utilisation de la représentation en famille génératrice est plus important que celui apporté par l'existence de ces propriétés.

10.2 La synthèse directe

Les méthodes de synthèse par inégalités ont en commun le handicap d'être très difficiles à systématiser par une procédure algorithmique. Malgré toutes les astuces imaginées par la communauté, elles restent en effet bien trop proches des mécanismes qui donnent à la synthèse de neurones binaires sa complexité de classe NP. En revanche, les méthodes directes sont en général très procédurales et se prêtent bien mieux à une implémentation algorithmique. Elles parviennent à contourner les

[7] L'article [Chen et al., 2006] nous invite à surveiller la parution d'une discussion sur le sujet.

étapes de tâtonnements, caractéristiques des problèmes de classe NP, en acceptant une certaine approximation sur le résultat obtenu. Dans cette section, il s'agit en l'occurrence des deux aléas suivants :
- le neurone binaire synthétisé ne réalise peut être pas exactement la fonction logique à seuil désirée ;
- un réseau de neurones binaires sera peut-être obtenu au lieu d'un neurone binaire unique ;

10.2.1 Synthèse par pseudo-inversion

En conclusion de ses travaux sur les réseaux de neurones cellulaires, M. HÄNGGI publia en 1999 la première méthode analytique de synthèse de neurones binaires capable de prendre en charge une contrainte de marge de fonctionnement minimum [HÄNGGI et MOSCHYTZ, 1999]. De la même façon que les méthodes de la section précédente, la fonction logique à seuil $\mathcal{L}$ doit tout d'abord être traduite en inégalités, en y incluant cette fois-ci la marge de fonctionnement ϵ désirée. Pour $j \in [\![0\,;2^n-1]\!]$ celles-ci sont :

$$\mathbf{w}^\top \mathbf{e}^{(j)} + w_0 \geq \epsilon > 0 \quad \text{si} \quad \mathcal{L}(\mathbf{e}^{(j)}) \equiv \mathsf{vrai}, \tag{10.18a}$$

$$-\mathbf{w}^\top \mathbf{e}^{(j)} - w_0 \geq \epsilon > 0 \quad \text{si} \quad \mathcal{L}(\mathbf{e}^{(j)}) \equiv \mathsf{faux}. \tag{10.18b}$$

Là encore, le système d'inégalités obtenu est vraisemblablement très redondant et il convient donc de le simplifier en conséquence. Sous forme matricielle, le système s'exprime alors de la façon suivante, où les $\mathbf{e}^{(j_i)}|_{i=1}^{k}$ sont les vecteurs d'entrée retenus pour former les inégalités non redondantes et où le poids synaptique auxiliaire et les poids synaptiques sont regroupés dans une même matrice $\mathbf{w}_0$:

$$\begin{pmatrix} \pm 1 & \pm e_1^{(j_1)} & \pm e_2^{(j_1)} & \cdots & \pm e_n^{(j_1)} \\ \pm 1 & \pm e_1^{(j_2)} & \pm e_2^{(j_2)} & \cdots & \pm e_n^{(j_2)} \\ \vdots & \vdots & \cdot & \vdots & \\ \pm 1 & \pm e_1^{(j_k)} & \pm e_2^{(j_k)} & \cdots & \pm e_n^{(j_k)} \end{pmatrix} \begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix} \geq \begin{pmatrix} \epsilon \\ \epsilon \\ \vdots \\ \epsilon \end{pmatrix}, \tag{10.19}$$

$$\Leftrightarrow \mathbf{E}_{\mathcal{L}}{}^\top \mathbf{w_0} \geq \boldsymbol{\epsilon}.$$

Les signes + et − y sont affectés selon que $\mathcal{L}(\mathbf{e}^{(j_i)})$ est vraie ou fausse, conformément aux équations (10.18).

L'intuition géniale de M. HÄNGGI est de transformer l'inégalité matricielle (10.19) en égalité et de la résoudre avec les outils classiques de l'algèbre linéaire en inversant la matrice $\mathbf{E}_{\mathcal{L}}$. Or comme cette dernière n'est en général pas carrée — elle a plus de lignes que de colonnes — il faut souvent avoir recours à sa matrice pseudo-inverse $\mathbf{E}_{\mathcal{L}}^{+}$. Les poids synaptiques réalisant $\mathcal{L}$ sont alors donnés par la relation :

$$\mathbf{w}_0 = \mathbf{E}_{\mathcal{L}}^{+\top} \boldsymbol{\epsilon}. \tag{10.20}$$

En réalité, les poids synaptiques obtenus de la sorte ne constituent pas une solution exacte du système (10.19), mais une solution minimale au sens des moindres carrés. Par conséquent, il se peut que la marge de fonctionnement réelle du neurone binaire synthétisé soit inférieure [8] à ϵ, voire négative, ce qui signifierait que le neurone binaire ne réalise pas la fonction logique pour laquelle il a été conçu.

Par ailleurs, l'ajustement de la marge de fonctionnement se fait de manière grossière ; en jouant sur la dynamique des poids synaptique. En effet, les poids synaptiques $\mathbf{w}_0{}'$ réalisant $\mathcal{L}$ avec une marge de fonctionnement double de celle utilisée dans l'équations (10.20) sont donnés par :

$$\mathbf{w}_0{}' = 2\mathbf{E}_{\mathcal{L}}^{+\top}\boldsymbol{\epsilon} = 2\mathbf{w}_0, \tag{10.21}$$

ce qui correspond bien à une dynamique double de celle de $\mathbf{w}_0$.

Exemple 10.3 — Synthèse d'un neurone binaire par pseudo-inversion.
La fonction logique $\mathcal{L}_4$ de $\mathbb{L}_s^+$ $e_1(e_2+e_3+e_4)+e_2e_3e_4$ donne le système d'inégalités suivant :

$$\begin{array}{lll} w_1+w_4+w_0>0, & w_1+w_3+w_0>0, & w_1+w_2+w_0>0, \\ w_2+w_3+w_4+w_0>0, & w_2+w_3+w_0<0, & w_2+w_4+w_0<0, \\ w_3+w_4+w_0<0, & \text{et} \quad w_1+w_0<0. & \end{array}$$

L'expression matricielle de ce système, telle qu'elle est définie dans l'équation (10.19), s'écrit alors :

$$\mathbf{E}_{\mathcal{L}_4} = \begin{pmatrix} 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ -1 & 0 & -1 & -1 & 0 \\ -1 & 0 & -1 & 0 & -1 \\ -1 & 0 & 0 & -1 & -1 \\ -1 & -1 & 0 & 0 & 0 \end{pmatrix}.$$

Le calcul de la pseudo-inverse de $\mathbf{E}_{\mathcal{L}}$ donne la matrice :

$$\mathbf{E}_{\mathcal{L}_4}^{+} = \frac{1}{8}\begin{pmatrix} -3 & -3 & -3 & -7 & -5 & -5 & -5 & -9 \\ 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \\ 0 & 0 & 4 & 4 & 0 & 0 & 4 & 4 \\ 0 & 4 & 0 & 4 & 0 & 4 & 0 & 4 \\ 4 & 0 & 0 & 4 & 4 & 0 & 0 & 4 \end{pmatrix}.$$

Pour une valeur de marge quelconque, l'équation (10.20) donne les poids synaptiques suivants :

$$w_0 = -5\epsilon, \qquad w_1 = 4\epsilon, \qquad w_2 = 2\epsilon, \qquad w_3 = 2\epsilon \quad \text{et} \quad w_4 = 2\epsilon.$$

On peut vérifier que la marge de fonctionnement réellement obtenue est bien ϵ.

[8] Ou supérieure, mais cela n'est généralement pas un problème.

Bilan Plutôt que de fixer arbitrairement les poids synaptiques en fonction des contraintes exprimées par le système d'inégalités, cette méthode de synthèse par pseudo-inversion effectue un calcul de minimisation globale par le critère des moindres carrés. Pourtant, il n'est absolument pas évident que cela conduise systématiquement à une implémentation possible d'une fonction logique à seuil. En pratique, c'est néanmoins souvent le cas lorsque le nombre de variables est faible, ce qui explique sans doute le succès rencontré par cette méthode dans la communauté des réseaux de neurones cellulaires.

La possibilité affichée de régler la marge de fonctionnement des implémentations obtenues est à notre avis beaucoup plus discutable. Si cette méthode a effectivement le mérite de positionner de façon optimale le poids synaptique auxiliaire, elle se révèle incapable d'optimiser les valeurs des poids synaptiques de $\mathbf{w}$. Or nous avons montré dans la partie précédente que cela pouvait apporter un gain significatif sur la marge de fonctionnement. À dynamique de codage fixe pour les poids synaptiques, cette méthode ne permet donc pas d'ajuster la marge de fonctionnement des neurones binaires qu'elle génère.

Apport des familles génératrices La principale difficulté de la synthèse par pseudo-inversion réside une fois de plus dans l'établissement du système d'inégalités de départ. Comme nous l'avons vu dans la section précédente, l'utilisation des familles génératrices facilite grandement cette opération. C'est ici d'autant plus crucial que la qualité du système d'inégalités utilisé pour la pseudo-inversion influence beaucoup la marge de fonctionnement effective des neurones binaires obtenus.

Par exemple si les seize inégalités, tirées des seize vecteurs d'entrée de la fonction logique $\mathcal{L}_4$ de l'exemple 10.3, avaient été utilisées telles quelles pour la composition de $\mathbf{E}_{\mathcal{L}_4}$, alors les poids synaptiques obtenus auraient été :

$$w_0 = -1{,}5\epsilon, \qquad w_1 = 1{,}5\epsilon, \qquad w_2 = 0{,}5\epsilon, \qquad w_3 = 0{,}5\epsilon \text{ et } \quad w_4 = 0{,}5\epsilon,$$

et la marge de fonctionnement réellement obtenue aurait valu 0 quelle que soit la valeur fixée pour ϵ !

Formellement, l'utilisation des familles génératrices se traduit sur la relation (10.20) de la façon suivante :

$$\hat{\mathbf{w}}_0 = \hat{\mathbf{E}}_{\mathcal{L}}^{+\top} \boldsymbol{\epsilon}, \tag{10.22}$$

où $\hat{\mathbf{E}}_{\mathcal{L}}^{+}$ représente le système d'inégalités obtenu à partir des vecteurs générateurs de $\mathcal{L}$ et de ceux de sa famille conjointe.

Exemple 10.4 — Utilisation de la famille génératrice pour la synthèse d'un neurone binaire par pseudo-inversion.

La famille génératrice de la fonction logique utilisée dans l'exemple 10.3, ainsi que sa conjointe, s'écrivent, en fonction des deux variables réduites $\hat{e}_a \rightarrow \{e_1\}$ et $\hat{e}_b \rightarrow$

$\{e_2\,;e_3\,;e_4\}$:

$$\{\mathcal{L}_4\} = \begin{Bmatrix} 1 & 1 \\ 0 & 3 \end{Bmatrix} \qquad {}^{c}\{\mathcal{L}_4\} = \begin{Bmatrix} 1 & 0 \\ 0 & 2 \end{Bmatrix}.$$

Ces quatre vecteurs donnent le système d'inégalités suivant :

$$\begin{aligned} \hat{w}_a + \hat{w}_b + w_0 > 0, \\ -\hat{w}_a - w_0 < 0, \end{aligned} \qquad \text{et} \qquad \begin{aligned} 3\hat{w}_b + w_0 > 0, \\ -\hat{w}_b - w_0 < 0. \end{aligned}$$

Sa forme matricielle ainsi que sa pseudo-inverse sont donc :

$$\hat{\mathbf{E}}_{\mathcal{L}_4} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 3 \\ -1 & -1 & 0 \\ -1 & 0 & -2 \end{pmatrix} \qquad \text{et} \qquad \hat{\mathbf{E}}^{+}_{\mathcal{L}_4} = \frac{1}{4} \begin{pmatrix} -5 & -3 & -5 & -7 \\ 6 & 2 & 2 & 6 \\ 2 & 2 & 2 & 2 \end{pmatrix}.$$

L'application de l'expression (10.22) donne les poids synaptiques suivants, qui sont bien les mêmes que ceux obtenus dans l'exemple 10.3 :

$$w_0 = -5\epsilon, \qquad \hat{w}_a = 4\epsilon = w_1, \qquad \text{et} \qquad \hat{w}_b = 2\epsilon = w_2 = w_3 = w_4.$$

10.2.2 Synthèse par tableaux de Celinski

La méthode de synthèse proposée par P. Celinsky s'inspire des techniques de minimisation utilisées en synthèse de circuits logiques [Celinski et al., 2001]. À l'instar de la minimisation par tables de Karnaugh, cette méthode consiste à regrouper les vecteurs d'entrée formant une fonction logique à seuil identifiée, puis de combiner les groupes obtenus pour réaliser la fonction logique à synthétiser. C'est une idée très intéressante dans la mesure où elle autorise la synthèse de fonctions logiques n'appartenant pas à $\mathbb{L}_s$ sous la forme de réseaux de neurones binaires. De plus, en décomposant volontairement une fonction logique à seuil — normalement réalisable avec un seul neurone binaire — en plusieurs sous-fonctions logiques à seuil, elle permet d'augmenter la marge de fonctionnement des neurones obtenus.

La synthèse d'une fonction logique de n variables demande ainsi la constitution d'une table sur le modèle des tables de Karnaugh : c'est-à-dire en utilisant le codage Gray pour rassembler au maximum les vecteurs d'entrée adjacents. Les vecteurs d'entrée évaluant la fonction logique à vrai doivent ensuite être regroupés en suivant des schémas de regroupements pré-établis et qui correspondent à des fonctions logiques dont une implémentation robuste est déjà connue. Ainsi, si tous ces vecteurs peuvent être couverts par un seul groupement, alors un seul neurone binaire est synthétisé. Dans le cas contraire, chaque groupement est synthétisé séparément en autant de neurones binaires qui seront par la suite combinés au travers d'une porte ou.

Les schémas de groupement pour $n = 2$ à 4 sont respectivement donnés dans les figures 10.3, 10.4 et 10.5, à des substitutions ou inversions de variables près. D'une manière générale, il est assez difficile de prévoir l'allure de ces schémas pour un

nombre arbitraire de variables [EMAMY-K, 1999], ce qui handicape un peu cette technique. Les segments visibles sur certains des schémas représentés indiquent les vecteurs d'entrée dont la méthode de synthèse devra tenir compte pour établir le système d'inégalités à résoudre. Dans ses travaux, l'auteur a recours à une méthode de synthèse dérivée de la pseudo-inversion présentée dans la sous-section précédente. Cela étant, il est en pratique très difficile d'imaginer un schéma de groupement valide sans devoir calculer les poids synaptiques qui le réalisent. Il est donc à notre avis préférable de mémoriser ces derniers conjointement aux schémas et d'ainsi éviter de devoir les recalculer à chaque fois qu'un schéma sera utilisé.

Exemple 10.5 — Synthèse par tables de CELINSKI.

Soit la fonction logique $\mathcal{L}_5 \notin \mathbb{L}_s$ de quatre variables suivante :

$$\mathcal{L}_5(e_1, e_2, e_3, e_4) \equiv \overline{e}_1 e_3 + e_1 \overline{e}_3) + \overline{e}_2 e_4 + e_2 \overline{e}_4.$$

La couverture de la table de CELINSKI de $\mathcal{L}_5$ demande deux groupements (voir la figure 10.2) ce qui signifie qu'elle sera synthétisée sous la forme $\mathcal{L}_5 \equiv \mathcal{L}_5^{(1)} + \mathcal{L}_5^{(2)}$. Comme ces deux groupements ont été réalisés à l'aide du même schéma (voir la figure 10.5h), dont les poids synaptiques sont les suivants :

$$w_0 = -1{,}5, \qquad w_1 = -1, \qquad w_2 = 2, \qquad w_3 = -1, \qquad \text{et} \qquad w_4 = 2,$$

il suffit d'identifier les inversions et substitutions permettant de positionner correctement le groupe de la figure 10.5h et de les répercuter sur les poids synaptiques pour synthétiser $\mathcal{L}_5^{(1)}$ et $\mathcal{L}_5^{(2)}$ (voir la proposition 7.2 de la page 73). Cela donne en définitive pour $\mathcal{L}_5^{(1)}$:

$$w_0 = -0{,}5 \qquad w_1 = -2 \qquad w_2 = -1 \qquad w_3 = 2 \qquad \text{et} \qquad w_4 = 1,$$

et pour $\mathcal{L}_5^{(2)}$:

$$w_0 = -0{,}5 \qquad w_1 = 2 \qquad w_2 = 1 \qquad w_3 = -2 \qquad \text{et} \qquad w_4 = -1.$$

L'analyse de ces deux fonctions logique donne :

$$\mathcal{L}_5^{(1)} = \overline{e}_1 e_3 + \overline{e}_1 \overline{e}_2 e_4 + e_3 \overline{e}_2 e_4$$
$$\text{et} \quad \mathcal{L}_5^{(2)} = e_1 \overline{e}_3 + e_1 e_2 \overline{e}_4 + \overline{e}_3 e_2 \overline{e}_4,$$

dont la disjonction redonne bien $\mathcal{L}_5$.

Bilan et apport des familles génératrices Cette méthode de synthèse s'avère surtout intéressante par sa capacité à générer des réseaux de neurones binaires, plutôt que des neurones individuels. Dans ce dernier cas, il faut d'ailleurs reconnaitre qu'elle n'apporte aucune innovation. Néanmoins, sa gestion de la marge de fonctionnement est aussi simple qu'efficace : en limitant les schémas de regroupements à ceux dont une implémentation optimale est connue, il n'y a en effet aucun risque d'obtenir un réseau dont la marge de fonctionnement ne conviendrait pas.

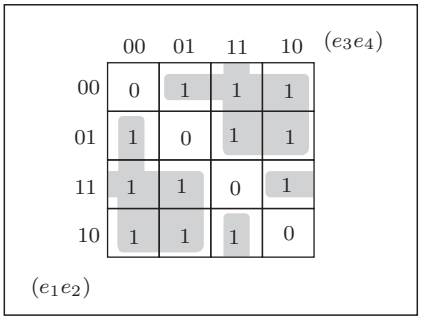
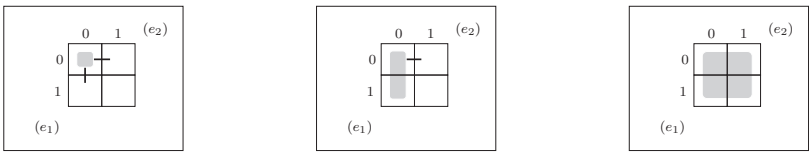


Figure 10.2
Synthèse de la fonction logique de l'exemple 10.5 à l'aide du schéma de regroupement de la figure 10.5h.



(a) (b) (c)

Figure 10.3
Groupements de Celinski pour la synthèse de neurones binaires de deux entrées.

Néanmoins, cette méthode n'est pas satisfaisante dans la mesure où elle ne se suffit pas à elle même et suppose la disponibilité de schémas de regroupements dont l'obtention n'est pas évidente. Nous montrerons dans le chapitre 13 que la représentation sous forme de famille génératrice permet, dans une certaine mesure, de solutionner ce problème. Nous y reprendrons alors cette démarche pour synthétiser des réseaux de neurones binaires lorsque cela sera nécessaire.

Ces cinq méthodes donnent un bref apperçu des difficultés rencontrées lors de la synthèse de jeux de poids synaptiques réalisant une fonction logique à seuil donnée. Chaque fois que cela était possible, nous avons montré comment l'utilisation des familles génératrices permettait de les améliorer de façon significative. Néanmoins, ces méthodes ont l'avantage d'introduire plusieurs points clefs que nous utiliserons dans les chapitres suivants pour l'élaboration de nos propres méthodes de synthèse :
- les descripteurs de Chow s'avéreront très performants pour transformer un tableau de vérité en famille génératrice ;
- l'interprétation géométrique de la synthèse par pseudo-inverse conduira à notre méthode de synthèse ;
- la synthèse par tableaux de Celinksi inspirera notre propre méthode de décomposition de fonction logique à seuil.

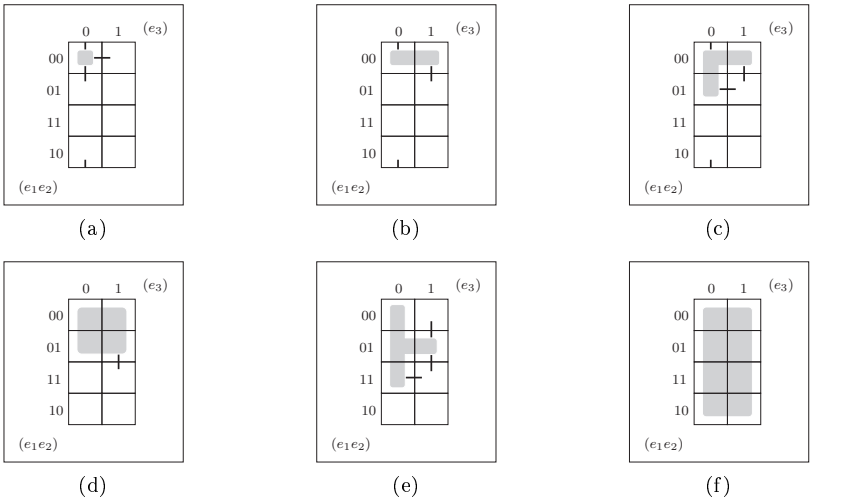


Figure 10.4
Groupements de CELINSKI pour la synthèse de neurones binaires de trois entrées.

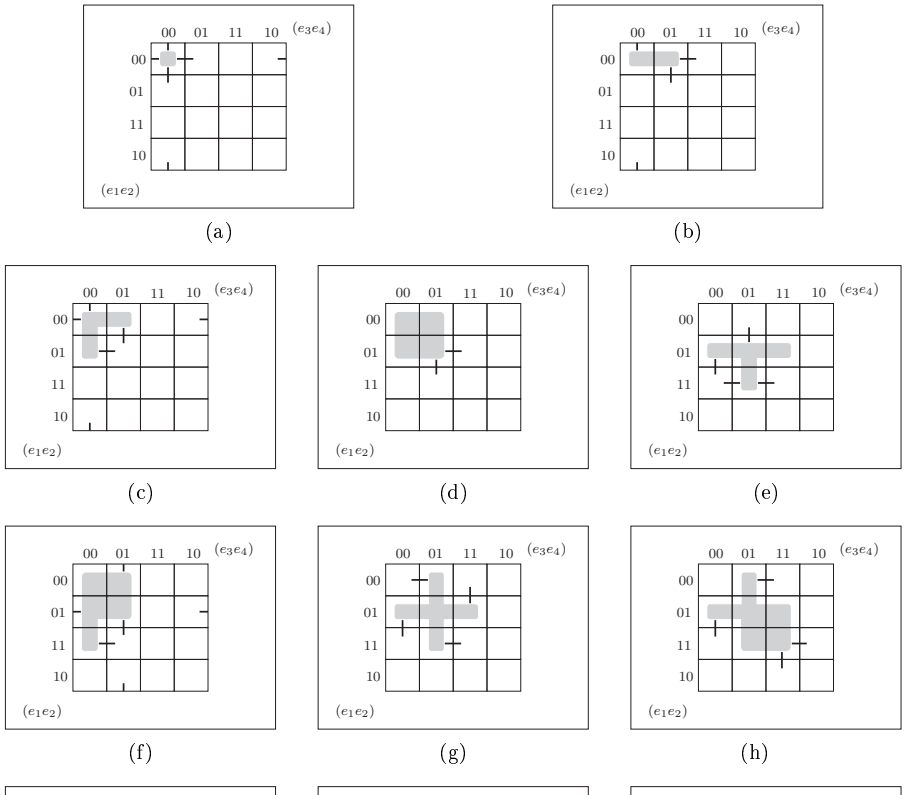


Figure 10.5
Groupements de CELINSKI pour la synthèse des neurones binaires de quatre entrées.

11

Synthèse spectrale d'une fonction logique à seuil

Le chapitre précédent a présenté quelques-unes des méthodes de synthèse de fonctions logiques à seuil prises dans la littérature, et qui constituent de réelles alternatives aux méthodes par apprentissage. Néanmoins, le gain en qualité annoncé pour les neurones binaires obtenus par leur biais n'est pas nécessairement au rendez-vous. Bien que ces méthodes n'aient pas été spécifiquement mises au point pour utiliser des familles génératrices, les améliorations apportées par ces dernières sont indéniables ; en particulier pour la formation des systèmes d'inégalités caractérisant les fonctions logiques à seuil synthétisées. Cela confirme l'intérêt d'utiliser un formalisme proche des propriétés des neurones binaires et, avec eux, des fonctions logiques à seuil.

Ce chapitre aborde la synthèse analytique de neurones binaires à partir de représentations en familles génératrices de la fonction logique qu'ils doivent implémenter. Le fait de spécialiser ainsi une méthode de synthèse permet de tirer pleinement partie des caractéristiques des familles génératrices : puisqu'elles sont très peu redondantes, l'effort calculatoire habituellement consenti pour éliminer les inégalités superflues peut maintenant être recentré sur l'optimisation des marges de fonctionnement. Ainsi, en exploitant la récursivité inhérente aux familles génératrices, deux méthodes de synthèse peuvent être élaborées en fonction de la morphologie des familles génératrices.

11.1 Morphologie des familles génératrices

Par morphologie, nous désignons la structure selon laquelle une famille génératrice se décompose en sous-familles génératrices. Or la signification logique de cette structure hiérarchique est le résultat de deux ordonnancements fondamentaux sur lesquels repose la définition même de famille génératrice : les variables réduites sont indicées par ordre d'influence décroissante et les vecteurs générateurs dans l'ordre lexicographique décroissant. Cela implique donc qu'une famille génératrice dont les composantes ont été choisies de façon arbitraire ne correspond pas nécessairement à une famille génératrice valide. Dans une optique de synthèse, il est donc impératif de veiller au respect de ces deux relations d'ordre, sous peine d'obtenir des neurones binaires au comportement totalement aléatoire.

11.1.1 Détection de la mauvaise mise en forme d'une famille génératrice

Une famille génératrice dont les vecteurs générateurs sont dans le désordre est relativement facile à identifier et à corriger, comme l'illustre l'exemple suivant :

$$\{\mathcal{L}_1\} = \begin{Bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \end{Bmatrix}, \tag{11.1}$$

et aurait en réalité dû s'écrire :

$$\{\mathcal{L}_1\}' = \begin{Bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{Bmatrix}. \tag{11.2}$$

Par contre, détecter qu'une famille génératrice a été écrite avec les entrées réduites dans le désordre est bien plus difficile et demande une certaine expertise pour déceler les incohérences que cela introduit. Par exemple, les deux derniers vecteurs générateurs de la famille génératrice de $\mathcal{L}_2$, donnée ci-après dans l'espace réduit formé des variables $\hat{e}_1$, $\hat{e}_2$ et $\hat{e}_3$, sont incohérents :

$$\{\mathcal{L}_2\} = \begin{Bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 0 & 3 & 0 \end{Bmatrix}. \tag{11.3}$$

En effet, le dernier vecteur générateur suggère que la variable réduite $\hat{e}_2$ suffit à rendre $\mathcal{L}_2$ vraie si elle est égale à trois. Or l'indice de cette variable étant 2, elle est normalement moins influente que $\hat{e}_1$, ce qui est contredit par l'avant dernier vecteur générateur. Ce dernier montre en effet que la perte d'une unité au niveau de $\hat{e}_2$ n'est

pas complètement compensée par le gain d'une unité au niveau de $\hat{e}_1$; dans le cas contraire, il n'aurait pas été nécessaire d'augmenter également la valeur de $\hat{e}_3$ pour finir de compenser la perte sur $\hat{e}_2$.

Cela indique donc que $\hat{e}_2$ est plus influente que $\hat{e}_1$ ce qui n'est pas normal étant donnés leurs indices respectifs. Cette famille génératrice doit donc s'écrire avec l'ordre $\hat{e}_2$, $\hat{e}_3$, $\hat{e}_1$ et en ré-ordonnant les vecteurs générateurs en conséquence :

$$\{\mathcal{L}_2\}' = \begin{Bmatrix} 3 & 0 & 0 \\ 2 & 1 & 1 \\ 1 & 1 & 2 \end{Bmatrix}. \tag{11.4}$$

Si les familles génératrices sont obtenues à l'aide des techniques décrites dans ce manuscrit (voir les chapitres 9 et 13), alors elles seront nécessairement correctement mises en forme, de sorte que la question de savoir comment corriger l'écriture d'une famille génératrice mal formée n'a pas d'intérêt immédiat. Toutefois, la réalisation d'une double complémentation d'une famille génératrice est un moyen très rapide [1] de déterminer si une famille génératrice est correctement mise en forme.

Par exemple, la réalisation de ce test sur la famille génératrice précédente $\{\mathcal{L}_2\}$ donne :

$$\overline{\{\mathcal{L}_2\}} = \begin{Bmatrix} 2 & 1 & 0 \\ 1 & 2 & 0 \\ 1 & 1 & 1 \\ 0 & 3 & 0 \\ 0 & 2 & 1 \end{Bmatrix} \quad \rightarrow \quad \overline{\overline{\{\mathcal{L}_2\}}} = \begin{Bmatrix} 2 & 2 & 0 \\ 2 & 1 & 1 \\ 1 & 2 & 0 \\ 1 & 1 & 1 \\ 0 & 3 & 0 \end{Bmatrix} \neq \{\mathcal{L}_2\}, \tag{11.5}$$

tandis que la même opération sur $\{\mathcal{L}_2\}'$ donne :

$$\overline{\{\mathcal{L}_2\}'} = \begin{Bmatrix} 3 & 0 & 0 \\ 2 & 1 & 0 \\ 2 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 2 \end{Bmatrix} \quad \rightarrow \quad \overline{\overline{\{\mathcal{L}_2\}'}} = \begin{Bmatrix} 3 & 0 & 0 \\ 2 & 1 & 1 \\ 1 & 1 & 2 \end{Bmatrix} = \{\mathcal{L}_2\}', \tag{11.6}$$

confirmant ainsi que la mise en forme de $\{\mathcal{L}_2\}$ n'est pas bonne, mais que celle de $\{\mathcal{L}_2\}'$ l'est.

Remarque. La famille génératrice obtenue après double complémentation est toujours une famille génératrice correctement mise en forme. Il s'agit d'un résultat général qui ne sera pas démontré dans ce manuscrit. Néanmoins, comme le montre le test sur $\{\mathcal{L}_2\}$, la famille génératrice obtenue ne constitue pas une version corrigée de la famille génératrice originale.

[1] La complexité de cette méthode hérite de la linéarité de l'algorithme de complémentation.

En revanche, cette opération de double complémentation peut être utilisée pour « réparer » une famille génératrice bien formée, mais dont la définition serait, pour une raison ou pour une autre, incomplète. Le processus de complétion réalisé par ce biais est à rapprocher de la propriété de généralisation observée dans les méthodes de programmation par apprentissage. Toutefois l'étude de la pertinence logique d'une telle généralisation est un sujet d'études que nous n'aborderons pas non plus dans ce manuscrit.

11.1.2 Repliement spectral d'une famille génératrice

Le repliement spectral est une caractéristique morphologique des familles génératrices décrivant la façon dont les sous-familles qui la constituent s'imbriquent les unes dans les autres. Il s'agit d'une propriété importante dans la mesure où nous montrerons qu'elle caractérise le degré d'inter-dépendance des poids synaptiques du vecteur $\hat{\mathbf{w}}$ vis-à-vis de la détermination de la marge de fonctionnement ϵ obtenue. Elle permet ainsi d'aiguiller automatiquement vers la méthode de synthèse la mieux appropriée à une famille génératrice donnée.

Par ailleurs, les familles génératrices de spectre replié se synthétisent en neurones binaires dont la marge de fonctionnement est en moyenne plus faible que celle les familles génératrices de spectre non replié. Cela justifie d'autant plus l'étude de cette caractéristique morphologique, dans la perspective de pouvoir décomposer des familles génératrices en plusieurs familles de spectre non replié, lorsqu'une marge de fonctionnement trop élevée sera requise (voir le chapite 13).

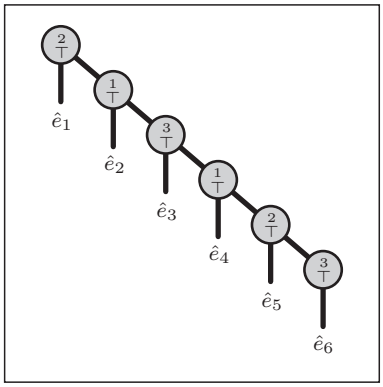
Définition intuitive du repliement spectral Avant d'aborder une définition formelle de cette propriété morphologique, considérons les deux familles génératrices suivantes :

$$\{\mathcal{L}_3\} = \begin{Bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 2 & 1 & 0 & 0 \\ 1 & 0 & 2 & 0 & 2 & 0 \\ 1 & 0 & 2 & 0 & 1 & 3 \end{Bmatrix} \quad \text{et} \quad \{\mathcal{L}_4\} = \begin{Bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 2 & 1 & 0 & 0 \\ 1 & 0 & 2 & 0 & 2 & 0 \\ 1 & 0 & 1 & 1 & 2 & 3 \end{Bmatrix}. \tag{11.7}$$

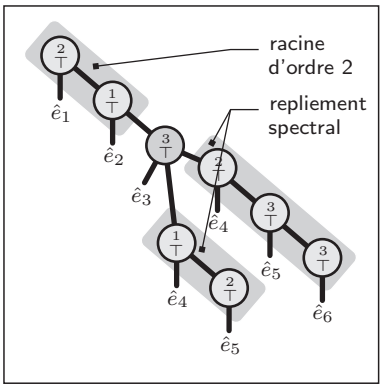
Ces deux familles comptent les six mêmes variables réduites et ont le même nombre de vecteurs générateurs. Pourtant, nous allons voir que le spectre de $\{\mathcal{L}_3\}$ n'est pas replié, alors que celui de $\{\mathcal{L}_4\}$ l'est. En effet, mises sous leur forme en n-somme (équations (11.8)) ces deux familles s'avèrent avoir une structure logique différente : $\mathcal{L}_4$ a en particulier une structure d'arbre bien plus riche en ramifications que $\mathcal{L}_3$, comme le confirme la figure 11.1.

$$\mathcal{L}_3(\hat{e}_i|_{i=1}^6) \equiv (\hat{e}_1 \overset{2}{\top} (\hat{e}_2 \overset{1}{\top} (\hat{e}_3 \overset{3}{\top} (\hat{e}_4 \overset{1}{\top} (\hat{e}_5 \overset{2}{\top} (\overset{3}{\top}\hat{e}_6)))))), \tag{11.8a}$$

$$\text{et} \quad \mathcal{L}_4(\hat{e}_i|_{i=1}^6) \equiv (\hat{e}_1 \overset{2}{\top} (\hat{e}_2 \overset{1}{\top} (\hat{e}_3 \overset{3}{\top} (\hat{e}_4 \overset{1}{\top} (\overset{2}{\top}\hat{e}_5)) \overset{3}{\top} (\hat{e}_4 \overset{2}{\top} (\hat{e}_5 \overset{3}{\top} (\overset{3}{\top}\hat{e}_6)))))). \tag{11.8b}$$



(a) Forme en n-somme de la fonction logique $\mathcal{L}_3$ définie dans l'équation (11.8a).



(b) Forme en n-somme de la fonction logique $\mathcal{L}_4$ définie dans l'équation (11.8b).

Figure 11.1
Visualisation du repliement spectral au travers des structures d'abres des fonctions logiques à seuil.

Chaque ramification fait intervenir un poids synaptique supplémentaire dans le calcul de la marge de fonctionnement, compliquant d'autant son expression analytique [Bénédic, 2002].

Une définition intuitive du repliement spectral des familles génératrices pourrait donc être formulée comme le nombre de ramifications que son expression en n-somme fait intervenir. Toutefois, nous préfèrerons à cette définition un critère directement exprimé sur les familles génératrices. En effet, ces dernières étant uniques, il sera bien plus facile de vérifier si elles satisfont à ce critère plutôt que de chercher à savoir si leur expression en n-somme — qui n'est pas unique — peut se mettre sous une forme qui ne fait pas intervenir de ramifications.

Définition formelle du repliement spectral La définition suivante, équivalente à la définition intuitive du paragraphe précédent est maintenant proposée.

Définition 11.1 (Repliement spectral d'une famille génératrice).
La famille génératrice d'une fonction logique à seuil de $\mathbb{L}_s^+$ n'ayant qu'une seule variable réduite ne présente pas de repliement spectral.

Par récurrence, une famille génératrice de d variables réduites ne présente pas de repliement spectral si et seulement si il n'y a au plus que deux racines d'ordre 1 de valeur différente, et les sous-familles génératrices correspondantes vérifient :

- *s'il n'y en a qu'une, alors elle ne présente pas de repliement spectral ;*
- *s'il y en a deux, alors la sous-famille génératrice d'ordre 1 attachée à la plus grande racine est la fonction logique* vrai*, tandis que la sous-famille génératrice attachée à l'autre racine ne présente pas de repliement spectral.*

Dans le cas contraire, la famille génératrice a un spectre replié.

Afin d'illustrer les critères introduits par cette définition, et explicités dans l'algorithme 11.1, ceux-ci sont appliqués aux familles génératrices de $\mathcal{L}_3$ et $\mathcal{L}_4$ de la page 142 dans l'exemple suivant.

Exemple 11.1 — Test du repliement spectral de deux familles génératrices.

La décomposition de $\mathcal{L}_3$ en sous-familles génératrices donne :

$$\{\mathcal{L}_3\} \to \begin{Bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 2 & 1 & 0 & 0 \\ 0 & 2 & 0 & 2 & 0 \\ 0 & 2 & 0 & 1 & 3 \end{Bmatrix} \to \begin{matrix} 1 \\ 0 \\ 0 \\ 0 \end{matrix}\!\!\begin{Bmatrix} 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 2 & 0 & 2 & 0 \\ 2 & 0 & 1 & 3 \end{Bmatrix}$$
$$\to \begin{matrix} 2 \\ 2 \\ 2 \end{matrix}\!\!\begin{Bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 1 & 3 \end{Bmatrix} \to \begin{matrix} 1 \\ 0 \\ 0 \end{matrix}\!\!\begin{Bmatrix} 0 & 0 \\ 2 & 0 \\ 1 & 3 \end{Bmatrix} \to \begin{matrix} 2 \\ 1 \end{matrix}\!\!\begin{Bmatrix} 0 \\ 3 \end{Bmatrix}. \quad (11.9)$$

Chaque étape correspond bien à la définition 11.1, ce qui conduit à la conclusion que le spectre de $\{\mathcal{L}_3\}$ n'est pas replié.

Par contre, la famille génératrice de $\mathcal{L}_4$ présente un repliement spectral :

$$\{\mathcal{L}_4\} \to \begin{Bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 2 & 1 & 0 & 0 \\ 0 & 2 & 0 & 2 & 0 \\ 0 & 1 & 1 & 2 & 3 \end{Bmatrix} \to \begin{matrix} 1 \\ 0 \\ 0 \\ 0 \end{matrix}\!\!\begin{Bmatrix} 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 2 & 0 & 2 & 0 \\ 1 & 1 & 2 & 3 \end{Bmatrix} \to \begin{matrix} 2 \\ 2 \\ 1 \end{matrix}\!\!\begin{Bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 1 & 2 & 3 \end{Bmatrix} \Leftarrow, \quad (11.10)$$

puisque la famille génératrice repérée par une double-flèche aurait dû être la famille génératrice de la fonction logique vrai et viole par conséquent la définition 11.1.

Algorithme 11.1
Test du repliement d'une famille génératrice.

```
/* Fonction test_repliement:
Retourne les poids synaptiques approchant une famille génératrice donnée.
   Paramètres:
/Fg/ - la famille génératrice à tester*/

fonction test_repliement(Fg) = booleen {
        // /eta1/ - première racine d'ordre 1
        entier eta1 = Fg[1][1];

        // /sousFg/ - sous-famille d'ordre 1 relative à eta1
        famillegeneratrice sousFg = Fg.sousfamille(eta1);

        // test si sousFg est la fonction vrai, on sélectionne la racine suivante
        si (sousFg.fonction == vrai) {
                eta1 = Fg[2][1];
                sousFg = Fg.sousfamille(eta1);
        }

        si (!test_repliement(sousFg)) {
                retourner(faux);
        }
        sinon { // test s'il y a des racines d'ordre 1 supplémentaires
                si (Fg.estracine(eta1-1)) {
                        retourner(faux);
                }
        }

        retourner(vrai);
}
```

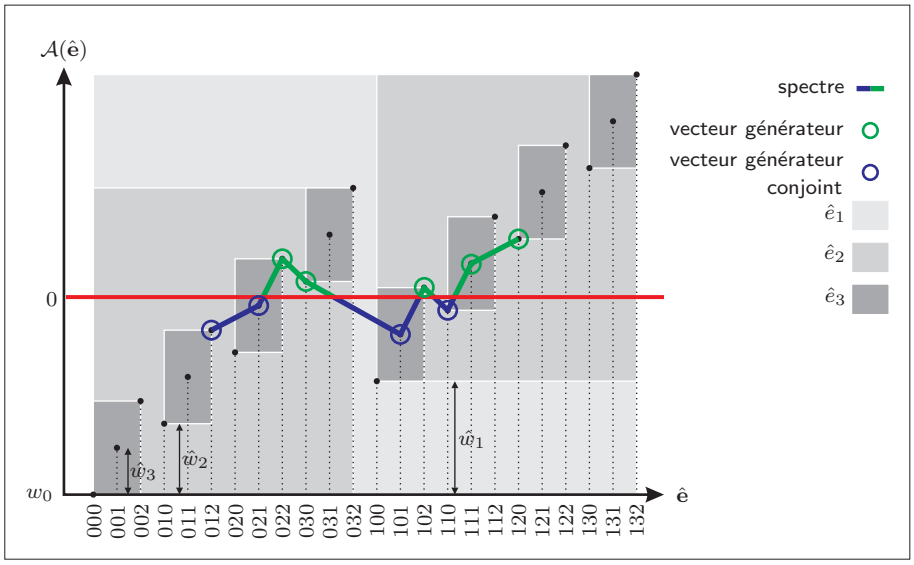


Figure 11.2
Représentation graphique de la fonction spectre d'une fonction logique à seuil ainsi que du spectre de la famille génératrice correspondante.

Afin d'apporter une aide visuelle aux sections consacrées à la synthèse spectrale d'une famille génératrice, une représentation graphique du spectre de ces dernières a été imaginée. Elle est présentée dans la sous-section suivante.

11.1.3 Visualisation du spectre d'une famille génératrice

Lorsque la valeur des poids synaptiques implémentant une fonction logique à seuil est connue, la fonction spectre de cette dernière, telle que nous l'avons définie à la page 75 peut être visualisée au travers du tracé de sa fonction d'agrégation en fonction des vecteurs réduits donnés en entrée (voir la figure 11.2). Ces différents vecteurs sont disposés sur l'axe des abscisses selon l'ordre lexicographique croissant afin de faire ressortir le mieux possible les propriétés de monotonie des fonctions logiques à seuil. Sur cette figure sont représentés les vecteurs générateurs et les vecteurs générateurs conjoints. La courbe supportée par les valeurs prises par la fonction d'agrégation pour ces vecteurs particuliers correspond à ce que nous appelons « spectre de la famille génératrice ».

Malheureusement, une telle représentation graphique n'est bien entendue possible que si les poids synaptiques ont des valeurs fixées, ce qui n'est pas le cas avec les familles génératrices puisque ces dernières ont été construites pour en être indépendantes. Toutefois, l'étude des descripteurs de Chow réalisée dans le chapitre précédent permet de tirer parti de l'ordonnancement des entrées réduites dans les familles génératrices. En effet, si une entrée e_i a plus d'influence sur la valeur de la sortie d'une fonction logique à seuil que e_j alors nécessairement $w_i > w_j$ [Chow, 1961]. Une relation d'ordre sur les poids synaptiques des d entrées réduites s'en déduit alors

immédiatement :

$$\hat{w}_1 > \hat{w}_2 > \cdots > \hat{w}_d. \tag{11.11}$$

Ajoutée à la connaissance de l'identité des vecteurs générateurs d'une famille génératrice, ainsi que de ses vecteurs générateurs conjoints, cette relation d'ordre permet, malgré l'indétermination des poids synaptiques, de positionner suffisamment de valeurs de la fonction d'agrégation pour avoir une idée précise de l'allure du spectre de sa famille génératrice. Ce résultat n'a d'ailleurs rien de surprenant puisqu'il traduit le fait qu'une famille génératrice contient suffisamment d'informations pour décrire complètement la fonction logique qu'elle représente.

Observation du repliement spectral Grâce à ce moyen d'investigation, la réalité du repliement spectral dont font l'objet, ou non, les familles génératrices se manifeste de façon très explicite. Dans la figure 11.3, la corrélation entre repliement et nombre de franchissements de la valeur de seuil par le spectre de la famille génératrice, est évidente :
– si elle ne le traverse qu'une fois, alors la famille génératrice a un spectre non replié ;
– sinon elle a un spectre replié.
De cette manière, si nous arrivons, d'une façon ou d'une autre, à trouver des poids synaptiques permettant de positionner correctement le spectre d'une famille génératrice autour de la valeur de seuil, alors nous aurons synthétisé un neurone binaire réalisant cette opération. Il s'agit précisément du principe des méthodes de synthèse spectrale dont ce chapitre fait l'objet.

11.2 Synthèse de familles génératrices de spectre non replié

Cette méthode a pour origine un travail de recherches effectué au *non-linear electronics laboratory* de l'université de Berkeley, en partenariat avec le groupe *Optronik und Signalverarbeitung* de l'institut franco-allemand de recherches de Saint-Louis [Bénédic, 2002 ; Monnin, 2003]. Bien qu'elle ne soit capable de synthétiser que les familles génératrices sans repliement spectral, nous verrons au début de la section suivante que les neurones binaires obtenus pour des familles génératrices quelconques ne sont pas totalement fantaisistes.

La raison pour laquelle les familles génératrices à spectre non replié peuvent faire l'objet d'une méthode de synthèse séparée est l'indépendance de leurs poids synaptiques dans le réglage de la marge de fonctionnement. En effet, comme le montre la figure 11.4, celle-ci est nécessairement fixée par les deux vecteurs encadrant le point de franchissement du spectre. Or cet écart vaut exactement $\hat{w}_d$, ce qui implique que les autres poids synaptiques n'ont aucun effet sur la marge de fonctionnement. En réalité, si les autres poids synaptiques ont des valeurs trop faibles, ils peuvent

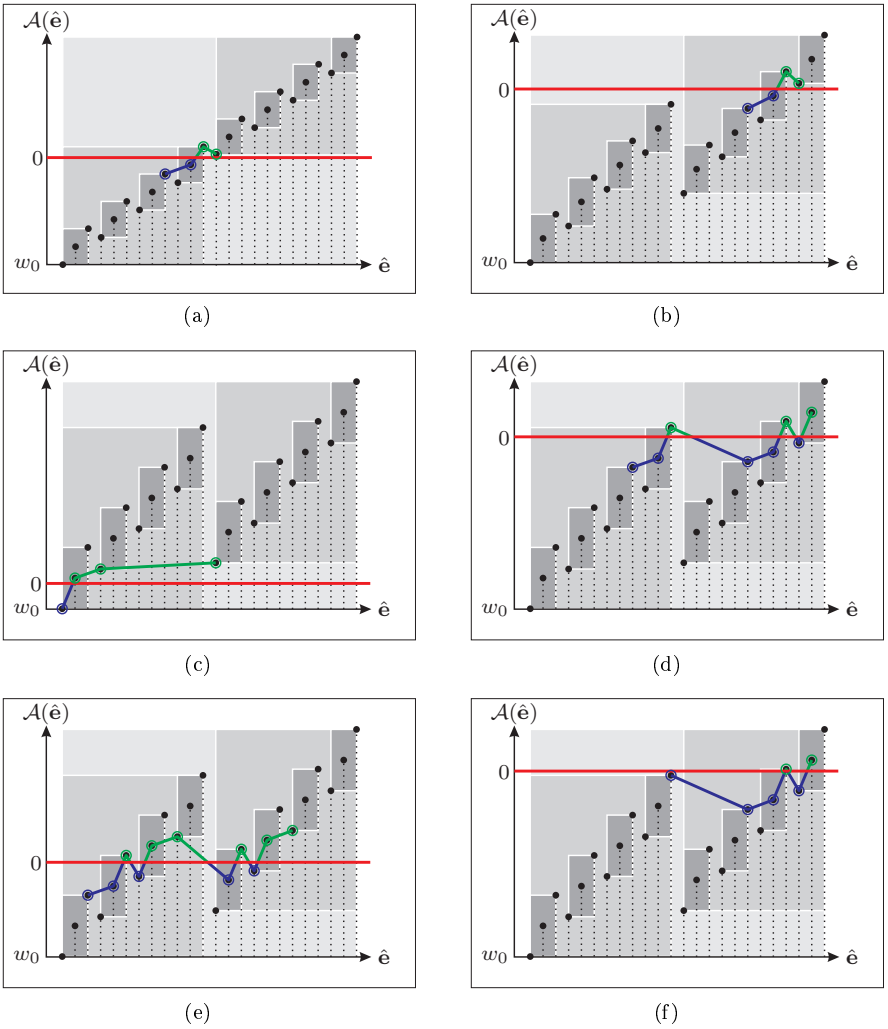


Figure 11.3
Quelques allures de fonctions spectres montrant du repliement (11.3d, 11.3e, 11.3f) ou non (11.3a, 11.3b, 11.3c).

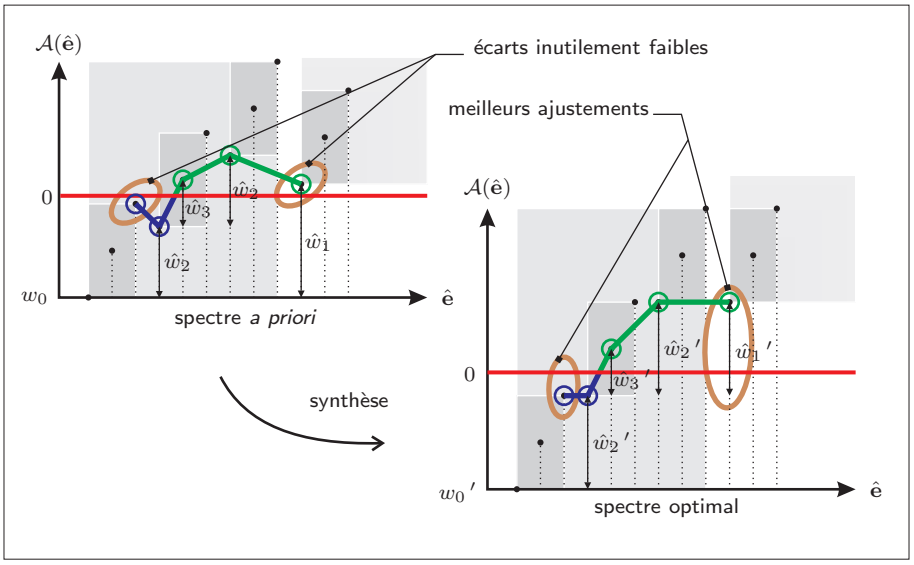


Figure 11.4
Influence des poids synaptiques sur la marge de fonctionnement lors de la synthèse d'une famille génératrice de spectre non replié.

dégrader la marge de fonctionnement imposée par $\hat{w}_d$, mais ils ne pourront jamais l'augmenter.

Proposition 11.1 (Indépendance des poids synaptiques pour la synthèse d'une famille génératrice de spectre non replié).
Lorsque le spectre d'une famille génératrice de d variables réduites n'est pas replié, alors la marge de fonctionnement maximale qu'il est possible de donner à tout neurone binaire réalisant cette famille est réglée par $\hat{w}_d$ et lui seul.

Corollaire. Cette proposition implique que les $d-1$ autres poids synaptiques réduits, ainsi que le poids synaptique auxiliaire, ne peuvent que dégrader cette marge s'ils sont mal réglés.

Démonstration. Dans un premier temps, étudions le cas $d = 1$, c'est-à-dire le cas des familles génératrices de la forme $\{\eta_d\}$, ce qui correspond à une famille conjointe $\{\eta_d - 1\}$. Pour ces deux vecteurs réduits, la fonction d'agrégation vaut respectivement :

$$\eta_d \hat{w}_d + w_0 = \epsilon \qquad \text{et} \qquad (\eta_d - 1)\hat{w}_d + w_0 = -\epsilon', \tag{11.12}$$

où ϵ et ϵ' sont des réels positifs non nuls. Par définition, la marge de fonctionnement est la plus petite de ces deux valeurs, et son optimisation consiste donc, en jouant sur les valeurs de $\hat{w}_d$ et w_0, à maximiser $\min(\epsilon, \epsilon')$. Or, ces deux grandeurs varient de manière opposée ce qui implique que la marge de fonctionnement optimale est atteinte pour $\epsilon = \epsilon'$. En reportant cette égalité dans le système (11.12), il vient alors :

$$w_0 = \epsilon(1 - 2\eta_d) \qquad \text{et} \qquad \hat{w}_d = 2\epsilon, \tag{11.13}$$

ce qui clos la démonstration du premier cas de figure.

Dans le cas plus général où $d > 1$, nous supposerons que la proposition est vraie pour $d-1$ variables réduites, de sorte que la marge de fonctionnement soit fixée par $2\epsilon = \hat{w}_d$ pour n'importe quelles valeurs des poids synaptiques $\hat{w}_i|_{i=2}^{d}$ et w_0. Nous noterons ce dernier $w_{0,d-1}$ pour le différencier du poids synaptique auxiliaire du cas général.

Pour $d > 1$ variables réduites, une famille génératrice sans repliement spectral se décompose, par définition, de l'une des deux façons suivantes :

$$\{\mathcal{L}_d\}_1 = \begin{Bmatrix} \eta_1 & \{0 & \cdots & 0\} \\ \eta_1 - 1 & & \{\mathcal{L}_{d-1}\} & \end{Bmatrix} \qquad \text{ou} \qquad \{\mathcal{L}_d\}_2 = \begin{Bmatrix} \eta_1 - 1 & \{\mathcal{L}_{d-1}\} \end{Bmatrix},$$

où $\{\mathcal{L}_{d-1}\}$ est une sous-famille génératrice de $d-1$ variables réduites pour laquelle la marge de fonctionnement vaut donc ϵ et n'est réglé que par $\hat{w}_d$. Montrons que la valeur de $\hat{w}_1$ ne peut en aucun cas améliorer la marge de fonctionnement imposée par $\{\mathcal{L}_{d-1}\}$.

Pour les vecteurs générateurs de $\{\mathcal{L}_d\}_1$, la fonction d'agrégation vaut respectivement :

$$\eta_1 \hat{w}_1 + w_0 = \epsilon^{(0)} \tag{11.14a}$$

$$\text{et} \quad (\eta_1 - 1)\hat{w}_1 + \epsilon - w_{0,d-1} + w_0 = \epsilon^{(1)}, \tag{11.14b}$$

où $\epsilon - w_{0,d-1}$ est, par définition, la valeur minimale de la fonction d'agrégation pour la sous-famille génératrice $\{\mathcal{L}_{d-1}\}$. Dans le cas où $\{\mathcal{L}_d\}_2$ est la famille génératrice de $\mathcal{L}_d$, alors seule la seconde équation est obtenue.

De la même manière, la famille génératrice conjointe de $\{\mathcal{L}_d\}$ se décompose de l'une des deux façons suivantes :

$${}^c\{\mathcal{L}_d\}_1 = \begin{Bmatrix} \eta_1 - 1 & {}^c\{\mathcal{L}_{d-1}\} \\ \eta_1 - 2 & \{n_2 \quad \dots \quad n_d\} \end{Bmatrix} \qquad \text{ou} \qquad {}^c\{\mathcal{L}_d\}_2 = \begin{Bmatrix} \eta_1 - 1 & {}^c\{\mathcal{L}_{d-1}\} \end{Bmatrix}.$$

Le cas où elle s'écrit ${}^c\{\mathcal{L}_d\}_1$, conduit aux deux équations suivantes :

$$(\eta_1 - 1)\hat{w}_1 - \epsilon - w_{0,d-1} + w_0 = -\epsilon^{(1)\prime} \tag{11.15a}$$

$$\text{et} \quad (\eta_1 - 2)\hat{w}_1 + \sum_{i=2}^{d} w_i n_i + w_0 = -\epsilon^{(2)\prime}, \tag{11.15b}$$

où $-\epsilon - w_{0,d-1}$ est, par définition, la valeur maximale de la fonction d'agrégation pour la sous-famille génératrice conjointe de $\mathcal{L}_{d-1}$. Là encore, si la famille génératrice conjointe avait été $\{\mathcal{L}_d\}_2$, alors seule la première équation aurait été obtenue.

Formellement, la marge de fonctionnement optimale est donnée par les valeurs de $\hat{w}_1$ et w_0 qui maximisent $\min(\epsilon^{(0)}, \epsilon^{(1)}, \epsilon^{(1)\prime}, \epsilon^{(2)\prime})$. Or, la figure 11.5 montre que pour $\hat{w}_1$ suffisamment grand, on a $\min(\epsilon^{(0)}, \epsilon^{(1)}) = \epsilon^{(1)}$ et $\min(\epsilon^{(1)\prime}, \epsilon^{(2)\prime}) = \epsilon^{(1)\prime}$. Une fois de

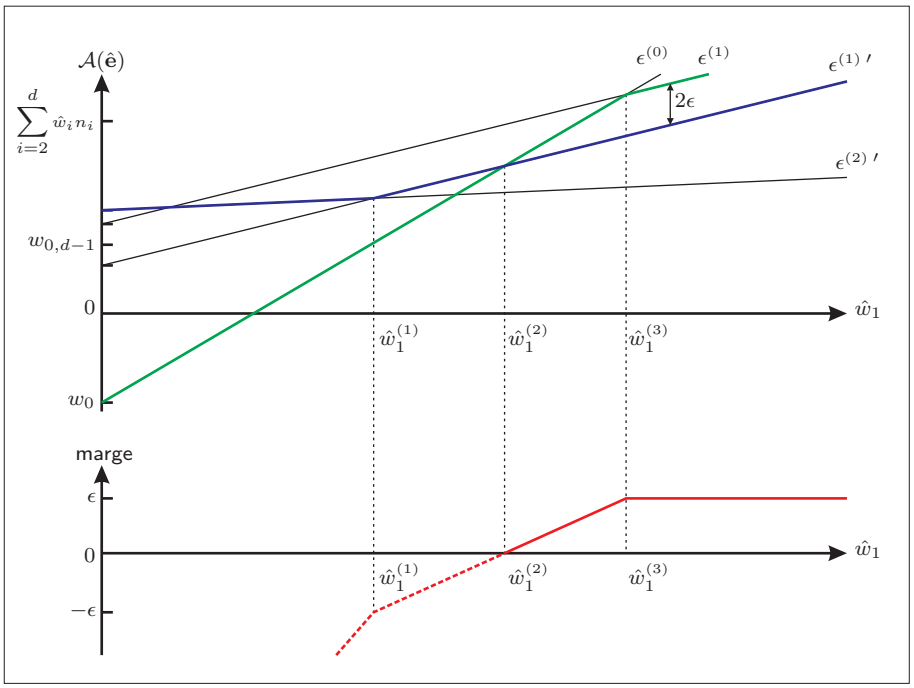


Figure 11.5
Influence de $\hat{w}_1$ sur la marge de fonctionnement d'un neurone binaire réalisant une famille génératrice à spectre non replié.

plus, les variations de ces deux grandeurs sont opposées, ce qui implique que la valeur optimale de la marge de fonctionnement est obtenue lorsque $\epsilon^{(1)} = \epsilon^{(1)\prime}$, c'est-à-dire pour :

$$w_0 = (1 - \eta_1)\hat{w}_1 + w_{0,d-1} \tag{11.16a}$$

$$\text{et} \quad \hat{w}_1 \geq \hat{w}_1^{(3)} = \max\left(\epsilon + \sum_{i=2}^{d} w_i n_i + w_{0,d-1}, \epsilon - w_{0,d-1}\right). \tag{11.16b}$$

En substituant par exemple la valeur de w_0 dans l'équation (11.15a), il vient l'égalité $\epsilon = \epsilon^{(1)} = \epsilon^{(1)\prime}$, qui démontre que ni $\hat{w}_1$ ni w_0, pourtant choisis de façon optimale, ne peuvent améliorer la marge de fonctionnement fixée par $\hat{w}_d$.

Par récurrence sur le nombre de variables réduites d'une famille génératrice, la proposition est donc vérifiée. □

En plus de prouver que les poids synaptiques peuvent être choisis librement sans influencer la marge de fonctionnement optimale du neurone binaire obtenu, cette démonstration fournit un moyen très simple de synthétiser une famille génératrice de spectre non replié. À l'image de celle-ci, l'algorithme qui en découle est naturellement récursif, fixant tout d'abord le poids synaptique $\hat{w}_d$ conformément à la marge de fonctionnement désirée, avant de calculer les poids synaptiques suivants les uns après les autres. Puisqu'il n'y a qu'un lancement récursif par poids synaptique, cette méthode de synthèse affiche par conséquent une complexité linéaire en $\mathcal{O}(d)$, ce qui est de loin la meilleure à l'heure actuelle.

Toutefois, il reste une question dont la résolution n'était pas nécessaire pour démontrer la proposition 11.1 : le choix exact de la valeur de $\hat{w}_1$. S'il est vrai que la valeur de la marge de fonctionnement n'est pas affectée par ce choix tant qu'il vérifie la condition exprimée par l'inégalité (11.16b), il en est tout autrement de la valeur de la marge de fonctionnement normalisée introduite à la page 70 du chapitre 7. Par définition, cette dernière augmente proportionnellement à la marge de fonctionnement et à l'inverse de la norme du vecteur $\hat{\mathbf{w}}$. En conséquence, le choix de $\hat{w}_1$ doit se porter sur la valeur la plus petite permise par l'inégalité (11.16b) ; à savoir :

$$\begin{aligned} 2\hat{w}_1 &= 2\hat{w}_1^{(2)} + \left|\hat{w}_1^{(3)} - \hat{w}_1^{(1)}\right| \\ &= 2\epsilon + \sum_{i=2}^{d} n_i \hat{w}_i + \left| 2w_{0,d-1} + \sum_{i=2}^{d} n_i \hat{w}_i \right|. \end{aligned} \tag{11.17}$$

Dès lors, l'algorithme de synthèse optimale de neurones binaires à partir de familles génératrices de spectre non replié est complètement décrit. L'algorithme 11.2 en propose une implémentation en pseudo-code. Un exemple d'application est proposé ci-après.

Exemple 11.2 — Synthèse d'une famille génératrice à spectre non replié.
Soit la fonction logique $\mathcal{L}_5$ de quatre variables réduites dont la famille génératrice est la suivante :

$$\{\mathcal{L}_5\} = \begin{Bmatrix} 2 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 2 \end{Bmatrix},$$

où les entrées réduites sont définie telles que $\hat{e}_4 \in [\![0\,;2]\!]$, $\hat{e}_3 \in [\![0\,;1]\!]$, $\hat{e}_2 \in [\![0\,;1]\!]$, $\hat{e}_1 \in [\![0\,;2]\!]$. Nous supposons avoir besoin d'une marge de fonctionnement de deux unités : $\epsilon = 2$. Comme cette famille génératrice a un spectre non replié, l'algorithme 11.2 peut être utilisé pour la synthétiser en un neurone binaire optimal.
La décomposition de $\{\mathcal{L}_5\}$ en sous-familles donne :

$$\{\mathcal{L}_5\} \to \begin{matrix} 2 \\ 1 \\ 1 \end{matrix}\!\!\begin{Bmatrix} 0 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 2 \end{Bmatrix} \to \begin{matrix} 1 \\ 1 \end{matrix}\!\!\begin{Bmatrix} 1 & 0 \\ 0 & 2 \end{Bmatrix} \to \begin{matrix} 1 \\ 0 \end{matrix}\!\!\begin{Bmatrix} 0 \\ 2 \end{Bmatrix} \to 0\{2\}.$$

Par l'application du jeu d'équations (11.13), il vient immédiatement $\hat{w}_4 = 2\epsilon = 4$ et $w_{0,4} = \epsilon(1 - 2\eta_4) = -6$ puisque $\eta_4 = 2$.
Pour la détermination de $\hat{w}_3$, les équations (11.16a) et (11.17) doivent être utilisées, avec $\eta_3 = 1$:

$$\hat{w}_3 = \epsilon + \frac{1}{2}(n_4\hat{w}_4) + |w_{0,4} + \frac{1}{2}(n_4\hat{w}_4)| = 8$$
$$\text{et} \quad w_{0,3} = (1 - \eta_3)\hat{w}_3 + w_{0,4} = -6.$$

De la même façon, la valeur de $\hat{w}_2$ s'obtient avec $\eta_2 = 2$:

$$\hat{w}_2 = \epsilon + \frac{1}{2}(n_4\hat{w}_4 + n_3\hat{w}_3) + |w_{0,3} + \frac{1}{2}(n_4\hat{w}_4 + n_3\hat{w}_3)| = 12$$
$$\text{et} \quad w_{0,2} = (1 - \eta_2)\hat{w}_2 + w_{0,3} = -18.$$

Algorithme 11.2
Synthèse spectrale d'une famille génératrice de spectre non replié.

```
/* Fonction synthese_spectrale_nonreplie:
Retourne les poids synaptiques réalisant une famille génératrice donnée.
   Paramètres:
/Fg/ - la famille génératrice de spectre non replié à synthétiser
/marge/ - marge de fonctionnement désirée*/

fonction synthese_spectrale_nonreplie(Fg,marge) = flottant[] {
        // /d/ - nombre de variables réduites de Fg
        entier d = Fg.nbrevariables();
        // /eta1/ - première racine d'ordre 1
        entier eta1 = Fg[1][1];

        // cas de base de la procédure récursive
        si (d == 1) {
                // on retourne le tableau [w0 w1]
                retourner ([marge*(1-2*etat1) 2*marge]);
        }

        // sinon il faut calculer w1 à partir des autres obtenus par récursivité
        // /sousFg/ - la seule sous-famille génératrice d'ordre 1 de Fg
        famillegeneratrice sousFg=Fg.sousfamille(1);

        // /W/ - tableau de poids [w0 w2 w3 ... wd] obtenu récursivement
        W = synthese_spectrale_nonreplie(sousFg,marge);

        // /eta1p/ - valeur de la deuxième racine d'ordre 1
        entier eta1p = Fg[2][1];

        // si c'est la même, alors eta1 vaut en fait eta1+1
        si (eta1p == eta1) {
                eta1++;
        }

        // /somme/ - calcul de a somme des wi*ni pour i=2 à d
        flottant somme = 0;
        pour (i=2;i<=d;i++) {
                somme += W[i] * Fg.cardinal(i);
        }

        // /w1/ - poids synaptique que nous devons calculer
        flottant w1 = marge + (somme/2) + abs(W[1]+(somme/2));

        // /w0/ - poids synaptique auxiliaire
        flottant w0 = (1 - eta1)*w1 + W[1];

        retourner([w0 w1 W[i=2:d]]);
}
```

Enfin, avec $\eta_1 = 2$, on a :

$$\hat{w}_1 = \epsilon + \frac{1}{2}(n_4\hat{w}_4 + n_3\hat{w}_3 + n_2\hat{w}_2) + |w_{0,2} + \frac{1}{2}(n_4\hat{w}_4 + n_3\hat{w}_3 + n_2\hat{w}_2)| = 20$$

et $\quad w_0 = w_{0,1} = (1 - \eta_1)\hat{w}_1 + w_{0,2} = -38.$

En définitive, le neurone binaire obtenu est caractérisé par :

$$\hat{\mathbf{w}} = \begin{pmatrix} 20 & 12 & 8 & 4 \end{pmatrix}^\top \text{ et } w_0 = -38.$$

Le lecteur pourra vérifier que la marge de fonctionnement de ce neurone binaire est bien 2, conformément à la contrainte de départ. Par ailleurs, la marge de fonctionnement normalisée vaut $\bar{\epsilon} = 0{,}029$, ce qui peut paraître faible, mais qui est pourtant, par construction, la meilleure valeur possible pour cette famille génératrice.

Grâce à la méthode de synthèse spectrale proposée dans cette section, toute famille génératrice peut être synthétisée de façon optimale en neurone binaire, pourvu que le spectre de cette dernière ne soit pas replié. Cette contrainte peut être vérifiée à l'avance en utilisant la définition 11.1, de sorte qu'une famille génératrice de spectre replié ne sera jamais synthétisée par ce biais. Dans le cas contraire, le neurone binaire obtenu ne réaliserait pas exactement la fonction logique à seuil décrite par la famille génératrice synthétisée.

11.3 Synthèse de familles génératrices de spectre quelconque

Le repliement spectral des familles génératrices traduit la contrainte selon laquelle la marge de fonctionnement des neurones binaires correspondants est fixée par au moins deux poids synaptiques et non plus un seul. En conséquence, une méthode de synthèse n'est plus libre de choisir leur valeur de façon indépendante : ils doivent maintenant être calculés simultanément. En effet, les spectres repliés présentent plusieurs points de franchissement du seuil, réglés chacun par un poids synaptique spécifique. À cela s'ajoutent des contraintes supplémentaires, sous la forme de combinaisons linéaires de poids synaptiques, dont le rôle est de garantir l'existence simultanée des différents franchissements. (Voir la figure 11.6.)

Malheureusement, cela pose une difficulté majeure dans le contexte de la synthèse spectrale. En effet, l'expression formelle de ces combinaisons linéaires varie énormément d'une famille génératrice à l'autre. Elles ne sont par conséquent pas connues, à moins de consentir à un lourd effort calculatoire dont la complexité demanderait rapidement un temps de calcul rédhibitoire. Sans cette information cruciale, la synthèse optimale d'une famille génératrice par une méthode spectrale n'est ainsi pas possible.

Les deux approches présentées par cette section ne sont ainsi pas totalement satisfaisantes du point de vue des objectifs fixés dans ce manuscrit, mais leurs performances méritent néanmoins d'être soulignées.

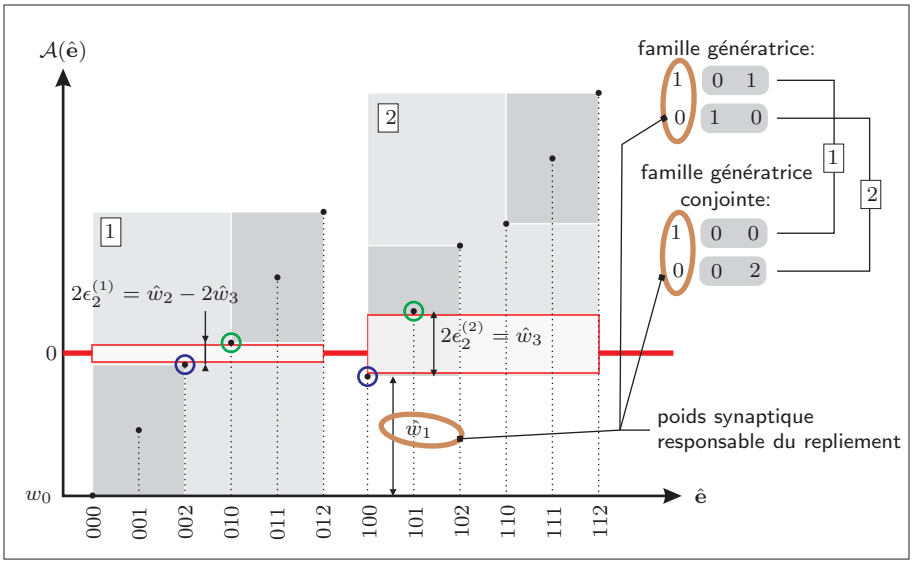


Figure 11.6
Contraintes fixant les points de franchissement d'un spectre replié.

11.3.1 Synthèse spectrale approchée

Cette sous-section n'aborde pas vraiment la question de la synthèse de fonctions logiques à seuil étant donné que le neurone calculé ne réalisera pas exactement la fonction logique recherchée. Nous avons néanmoins choisi de présenter cette option de synthèse parce qu'elle illustre la souplesse avec laquelle les familles génératrices permettent de manipuler les fonctions logiques à seuil.

Le principe de cette méthode de synthèse par approximation consiste à modifier une famille génératrice de spectre replié de façon à obtenir une famille de spectre non replié. Il y a vraisemblablement de nombreuses façons de faire, en fonction de ce qui nuira le moins au contexte d'utilisation du neurone binaire obtenu. Les paragraphes suivants présentent les trois que nous jugeons les plus pertinentes.

Approximation par troncature Dans cette approche, l'idée est d'éliminer les variables réduites les moins influentes d'une famille génératrice, jusqu'à ce que les repliements spectraux de cette dernière aient disparu. En pratique, cela se fait très rapidement au travers de la définition 11.1 : la première sous-famille génératrice ne vérifiant pas un de ces critères est éliminée de la famille génératrice originale.

Exemple 11.3 — Synthèse spectrale approchée par troncature.

En reprenant la fonction logique $\mathcal{L}_4$ de la page 142, et en lui appliquant l'algorithme 11.3, la famille génératrice suivante est obtenue :

$$\{\mathcal{L}_4\}^{(\text{tronquée})} = \begin{Bmatrix} 1 & 0 \\ 0 & 2 \end{Bmatrix}.$$

Algorithme 11.3
Approximation par troncature d'une famille génératrice.

```
/* Fonction troncature:
Retourne la famille génératrice approchée par troncature
  d'une famille génératrice donnée.
   Paramètres:
/Fg/ - la famille génératrice de spectre replié à approcher*/

fonction troncature(Fg) = famillegeneratrice {
        // /FgTronquee/ - famille génératrice approchée de Fg
        allouer(FgTronquee);
        // /racines/ - nombre de racines d'ordre 1 de Fg
        entier racines = Fg.nbreRacines(1);

        // cas où la variable réduite la plus influente est
        // trivialement responsable d'un repliement
        si (racines >= 3) {
                retourner(FgTronquee);
        }
        sinon {
                // /eta1/ - première racine d'ordre 1
                entier eta1 = Fg[1][1];
                // /sousFg/ - sous-famille d'ordre 1 relative à eta1
                famillegeneratrice sousFg = Fg.sousfamille(eta1);

                // si la sous-famille de cette racine est la fonction vrai
                // alors il faut l'ajouter à la famille tronquée
                si (sousFg.fonction == vrai) {
                        Fgtronquee.ajouter(eta1,vrai);
                }
                // On affecte à sousFg la sous-famille de racine eta1-1
                // Si elle n'existe pas, sousFg est vide
                sousFg = Fg.sousfamille(eta1-1);
                // ensuite, ajouter de la famille tronquée de sousFg
                FgTronquee.ajouter(eta1-1,troncature(sousFg));
        }
        retourner(FgTronquee);
}
```

La synthèse de cette famille génératrice par la méthode spectrale de la section précédente donne les poids synaptiques suivants, pour $\epsilon = 1$:

$$\hat{\mathbf{w}} = \begin{pmatrix} 4 & 2 & 0 & 0 & 0 \end{pmatrix}^\top \text{ et } w_0 = -3.$$

Approximation par extraction Ici, nous proposons d'extraire d'une famille génératrice de spectre replié, le plus grand nombre de vecteurs générateurs possibles formant une famille génératrice de spectre non replié. Cette opération, bien que plus complexe que la précédente, peut également être réalisée en parallèle de la vérification du repliement spectral de la famille génératrice à synthétiser. À chaque fois qu'un critère de la définition 11.1 n'est pas vérifié, une seule sous-famille génératrice est conservée ; celle qui compte le plus de vecteurs générateurs.

Exemple 11.4 — Synthèse spectrale approchée par extraction.
En reprenant la fonction logique $\mathcal{L}_4$ de la page 142, et en lui appliquant l'algorithme 11.4, la famille génératrice suivante est obtenue :

$$\{\mathcal{L}_4\}^{(\text{extraite})} = \left\{ \begin{matrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 2 & 3 \end{matrix} \right\}.$$

La synthèse de cette famille génératrice par la méthode spectrale de la section précédente donne les poids synaptiques suivants, pour $\epsilon = 1$:

$$\hat{\mathbf{w}} = \begin{pmatrix} 72 & 36 & 18 & 6 & 2 \end{pmatrix}^\top \text{ et } w_0 = -71.$$

À première vue, ce résultat est radicalement différent de celui obtenu par la synthèse de $\{\mathcal{L}_4\}^{(\text{tronquée})}$. Néanmoins, en divisant les poids ci-dessus par dix-huit, on obtient des ordres de grandeurs similaires et il devient évident que l'approche par troncature a « simplement » négligé la valeur des poids synaptiques $\hat{w}_3$, $\hat{w}_4$ et $\hat{w}_5$ devant celle de $\hat{w}_1$ et $\hat{w}_2$.

Approximation par moyennage Cette dernière approche est très certainement celle qui illustre le mieux le degré de maitrise qu'offrent les familles génératrices en matière de synthèse de fonctions logiques à seuil. Dans les deux méthodes d'approximation précédentes, l'objectif était de ne conserver qu'un seul franchissement de seuil ; le plus représentatif selon des critères subjectifs. Ici, nous proposons de fabriquer une famille génératrice de spectre non replié dont le point de franchissement de seuil est positionné le plus près possible du point de franchissement moyen, observé dans le spectre de la famille génératrice originale.

Il s'agit d'une opération qui se décompose en quatre étapes :

1. calculer la famille génératrice conjointe ;
2. réunir les deux familles et ordonner les vecteurs selon l'ordre lexicographique croissant ;
3. localiser les franchissements de seuil et calculer le point de franchissement moyen ;

Algorithme 11.4
Approximation par extraction d'une famille génératrice.

```
/* Fonction extraction:
Retourne la famille génératrice approchée par extraction
  d'une famille génératrice donnée.
   Paramètres:
/Fg/ - la famille génératrice de spectre replié à approcher*/

fonction extraction(Fg) = famillegeneratrice {
        // /FgExtraite/ - famille génératrice approchée de Fg
        allouer(FgExtraite);
        // /nbracines/ - nombre de racines d'ordre 1 de Fg
        entier nbracines = Fg.nbreracines(1);

        // cas où la variable réduite la plus influente
        // provoque trivialement du repliement:
        // on ajoute la sous-famille qui a le plus de vecteurs
        si (nbracines >= 3) {
                // /sousFg/ - sous-famille qui a le plus de vecteurs générateurs
                famillegeneratrice sousFg = Fg.sousfamillemax(1);
                // /eta1/ - racine correspondant à sousFg
                entier eta1 = Fg.racine(sousFg);
                // fabrication de FgExtraite
                FgExtraite.ajouter(eta1,extraction(sousFg));
        }
        sinon {
                // /eta1/ - première racine d'ordre 1 de Fg
                eta1 = Fg[1][1];

                // si la sous-famille de cette racine n'est pas la fonction vrai
                // il faut ajouter la plus grande des familles restantes
                si (sousFg.fonction != vrai) {
                        // /sousFg/ - sous-famille qui a le plus de vecteurs |
                             générateurs
                        famillegeneratrice sousFg = Fg.sousfamillemax(1);
                        // /eta1/ - racine correspondant à sousFg
                        entier eta1 = Fg.racine(sousFg);
                        // fabrication de FgExtraite
                        FgExtraite.ajouter(eta1,extraction(sousFg));
                }
                sinon {
                        sousFg = Fg.sousfamille(eta1-1);
                        FgExtraite.ajouter(eta1,vrai);
                        FgExtraite.ajouter(eta1-1,extraction(sousFg));
                }
        }
        retourner(FgExtraite);
}
```

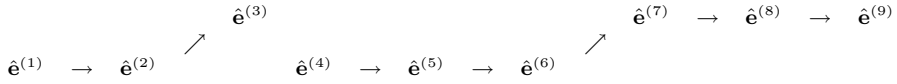
4. transférer tous les vecteurs générateurs conjoints supérieurs à ce point dans la famille génératrice et tous les vecteurs générateurs inférieurs dans la famille génératrice conjointe.

L'exemple suivant montre comment appliquer cette méthode, dont une implémentatiopn est proposée par l'algorithme 11.5, sur $\mathcal{L}_4$.

Exemple 11.5 — Synthèse spectrale approchée par moyennage.
La famille génératrice et la famille génératrice conjointe de $\mathcal{L}_4$ sont :

$$\{\mathcal{L}_4\} = \left\{\begin{array}{l} \hat{\mathbf{e}}^{(9)\top} = \begin{pmatrix}1 & 0 & 0 & 0 & 0\end{pmatrix} \\ \hat{\mathbf{e}}^{(8)\top} = \begin{pmatrix}0 & 2 & 1 & 0 & 0\end{pmatrix} \\ \hat{\mathbf{e}}^{(7)\top} = \begin{pmatrix}0 & 2 & 0 & 2 & 0\end{pmatrix} \\ \hat{\mathbf{e}}^{(3)\top} = \begin{pmatrix}0 & 1 & 1 & 2 & 3\end{pmatrix} \end{array}\right\}. \quad \text{et} \quad {}^{c}\{\mathcal{L}_4\} = \left\{\begin{array}{l} \hat{\mathbf{e}}^{(6)\top} = \begin{pmatrix}0 & 2 & 0 & 1 & 3\end{pmatrix} \\ \hat{\mathbf{e}}^{(5)\top} = \begin{pmatrix}0 & 1 & 1 & 2 & 2\end{pmatrix} \\ \hat{\mathbf{e}}^{(4)\top} = \begin{pmatrix}0 & 1 & 1 & 1 & 3\end{pmatrix} \\ \hat{\mathbf{e}}^{(2)\top} = \begin{pmatrix}0 & 1 & 0 & 2 & 3\end{pmatrix} \\ \hat{\mathbf{e}}^{(1)\top} = \begin{pmatrix}0 & 0 & 1 & 2 & 3\end{pmatrix} \end{array}\right\}.$$

Schématiquement, le spectre de $\mathcal{L}_4$ se présente de la façon suivante :



Algorithme 11.5 Approximation par moyennage d'une famille génératrice.

```
/* Fonction moyennage:
Retourne la famille génératrice approchée par moyennage
  d'une famille génératrice donnée.
   Paramètres:
/Fg/ − la famille génératrice de spectre replié à approcher*/

fonction moyennage(Fg) = famillegeneratrice {
        // /FgMoyenne/ − famille génératrice approchée de Fg
        allouer(FgMoyenne);
        // /FgConjointe/ − famille génératrice conjointe de Fg
        famillegeneratrice FgConjointe = Fg.conjointe();
        // /vecteurG/ − vecteur générateur après un franchissement
        vecteurreduit vecteurG = nul;
        // /vecteurC/ − vecteur générateur conjoint avant un franchissement
        vecteurreduit vecteurC = nul;
        // /nbrevecteur/ − nombre de vecteurs retenus pour la moyenne
        entier nbrevecteur = 0;

        // sélection des vecteurs générateurs juste après un franchissement
        pour (vecteur dans Fg) {
                si (vecteur.apres_franchissement()) {
                        vecteurG += vecteur;
                        nbrevecteur++;
        }

        // calcul du point de franchissement moyen
        vecteur /= nbrevecteur;

        // calcul de la nouvelle famille génératrice avec
        // les vecteurs de Fg et FgConjointe supérieurs à vecteur
        FgMoyenne = superieurs(vecteur,Fg) + superieurs(vecteur,FgConjointe);
        }
        retourner(FgConjointe);
}
```

Les deux points de franchissement de seuil sont donc respectivement délimités par $\hat{\mathbf{e}}^{(2)}$ et $\hat{\mathbf{e}}^{(3)}$ et par $\hat{\mathbf{e}}^{(6)}$ et $\hat{\mathbf{e}}^{(7)}$. Le point de franchissement moyen correspond donc au vecteur réduit suivant :

$$\frac{1}{4}\left(\hat{\mathbf{e}}^{(2)}+\hat{\mathbf{e}}^{(3)}+\hat{\mathbf{e}}^{(6)}+\hat{\mathbf{e}}^{(7)}\right)=\begin{pmatrix}0 & 1{,}5 & 0{,}25 & 1{,}75 & 2{,}25\end{pmatrix}.$$

Comme l'ordre lexicographique situe ce point entre $\hat{\mathbf{e}}^{(5)}$ et $\hat{\mathbf{e}}^{(6)}$, la famille génératrice approchée a par conséquent un spectre de la forme :

$$\cdots \rightarrow \hat{\mathbf{e}}^{(3)} \rightarrow \hat{\mathbf{e}}^{(4)} \rightarrow \hat{\mathbf{e}}^{(5)} \nearrow \hat{\mathbf{e}}^{(6)} \rightarrow \hat{\mathbf{e}}^{(7)} \rightarrow \cdots$$

ce qui correspond à la famille génératrice suivante :

$$\{\mathcal{L}_4\}^{(\text{moyennée})}=\left\{\begin{array}{l}\hat{\mathbf{e}}^{(9)\top}=\begin{pmatrix}1 & 0 & 0 & 0 & 0\end{pmatrix}\\ \hat{\mathbf{e}}^{(8)\top}=\begin{pmatrix}0 & 2 & 1 & 0 & 0\end{pmatrix}\\ \hat{\mathbf{e}}^{(7)\top}=\begin{pmatrix}0 & 2 & 0 & 2 & 0\end{pmatrix}\\ \hat{\mathbf{e}}^{(6)\top}=\begin{pmatrix}0 & 2 & 0 & 1 & 3\end{pmatrix}\end{array}\right\}.$$

La synthèse de cette famille génératrice par la méthode spectrale de la section précédente donne, pour $\epsilon=1$:

$$\hat{\mathbf{w}}=\begin{pmatrix}52 & 20 & 12 & 6 & 2\end{pmatrix}^{\top} \text{ et } w_0=-51.$$

Remarque. Dans la sous-section suivante, nous montrerons que les poids synaptiques réalisant exactement et optimalement $\mathcal{L}_4$ sont en fait :

$$\hat{\mathbf{w}}=\begin{pmatrix}60 & 24 & 18 & 6 & 2\end{pmatrix}^{\top} \text{ et } w_0=-59.$$

En pratique, l'intérêt de savoir synthétiser des neurones binaires réalisant une approximation de la fonction logique à seuil désirée peut sembler discutable. En effet, dans le contexte de la conception de circuits logiques, les fonctions logiques à seuil sont toujours parfaitement déterminées et synthétiser un circuit qui n'implémenterait pas ces dernières dans les moindres détails dénaturerait complètement la fonctionalité désirée. Pourtant, il est relativement facile d'imaginer des cas de figures pour lesquels disposer de ce type de solutions pourrait s'avérer profitable.

Au début du chapitre 6, nous avons en effet montré comment traduire un neurone artificiels dont les entrées sont quantifiées en un neurone binaire. Une telle transformation augmente évidemment le nombre d'entrées de façon exponentielle, ce qui n'est pas sans conséquences sur les temps de calcul de la plupart des méthodes de synthèse [2]. Dans ce contexte il est ainsi évident que certaines de ces entrées, celles affectées de poids synaptiques faibles en l'occurrence, puissent avoir une influence négligeable sur la fonctionalité globale du neurone binaire. Une synthèse spectrale approchée par troncature peut alors convenir.

[2] La synthèse spectrale de familles génératrices de spectre non replié a une complexité linéaire, et peut donc être éxécutée sur un nombre d'entrées bien plus grand que les méthodes concurrentes dont la complexité est au moins polynomiale.

Par ailleurs, dans le cas de neurones artificiels obtenus par apprentissage, on peut raisonnablement penser que les entrées de poids synaptiques faibles expriment la généralisation inhérente à l'apprentissage. La synthèse approchée par extraction et par moyennage peuvent alors être interprétées comme la suggestion d'une généralisation différente, dont l'intérêt serait de permettre au neurone binaire de pouvoir être resynthétisé de manière robuste. On préfèrera alors la méthode par moyennage dans le cas où aucune raison subjective ne conduit à préférer la synthèse d'un franchissment particulier.

11.3.2 Synthèse spectrale exacte

D'après les considérations précédentes, il apparaît qu'une fonction logique à seuil quelconque peut toujours être ramenée à la combinaison de plusieurs sous-familles génératrices dont le spectre n'est pas replié et qui peuvent donc individuellement faire l'objet de la synthèse spectrale de la section 11.2. La synthèse spectrale exacte proposée dans cette section ne consisterait alors qu'en la réunion des multiples jeux de poids synaptiques obtenus. Pour y parvenir, il faut toutefois apporter une réponse aux questions suivantes :

- que faire si les synthèses conduisent à des valeurs différentes pour un poids synaptique donné ?
- comment calculer un poids synaptique responsable d'un repliement, c'est-à-dire chargé de recomposer plusieurs sous-familles génératrices ?

Ces deux questions sont illustrées par la figure 11.7, qui montre les différentes phases de synthèse d'une famille génératrice de spectre replié. En plus de la méthode de synthèse de poids synaptiques pour les spectres non repliés (flèches vertes), elle fait apparaître deux autres techniques : la synthèse d'un poids synaptique responsable d'un repliement (flèche rouge) et la synthèse parallèle (pointillé vert). Ces deux apports sont décrits dans les paragraphes suivants.

Synthèse d'un poids synaptique responsable d'un repliement Comme le suggère la figure 11.7, il s'agit des cas où le poids calculé rassemble au moins deux sous-familles génératrices, de sorte que la famille génératrice obtenue s'écrive de l'une des deux façons suivantes :

$$\{\mathcal{L}_d\}_1 = \begin{Bmatrix} \eta_1 & \{0 \quad \cdots \quad 0\} \\ \eta_1 - 1 & \{\mathcal{L}_{d-1}^{(1)}\} \\ \eta_1 - 2 & \{\mathcal{L}_{d-1}^{(2)}\} \\ \vdots & \\ \eta_1 - k & \{\mathcal{L}_{d-1}^{(k)}\} \end{Bmatrix} \quad \text{ou} \quad \{\mathcal{L}_d\}_2 = \begin{Bmatrix} \eta_1 - 1 & \{\mathcal{L}_{d-1}^{(1)}\} \\ \eta_1 - 2 & \{\mathcal{L}_{d-1}^{(2)}\} \\ \vdots & \\ \eta_1 - k & \{\mathcal{L}_{d-1}^{(k)}\} \end{Bmatrix},$$

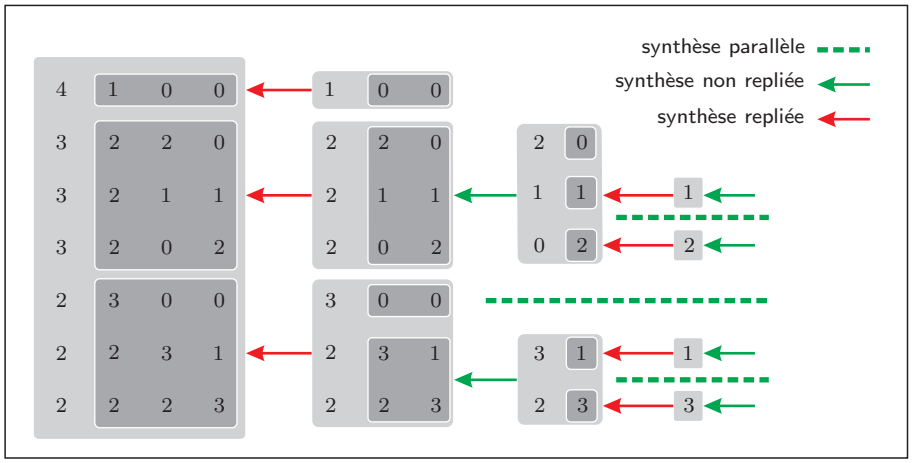


Figure 11.7
Principe de la synthèse spectrale exacte.

ce qui se traduit par les valeurs formelles suivantes de la fonction d'agrégation du neurone binaire synthétisé :

$$\eta_1 \hat{w}_1 + w_0 = \epsilon^{(0)}, \tag{11.18a}$$
$$(\eta_1 - 1)\hat{w}_1 + \epsilon_{d-1}^{(1)} - w_{0,d-1}^{(1)} + w_0 = \epsilon^{(1)}, \tag{11.18b}$$
$$\vdots$$
$$\text{et} \quad (\eta_1 - k)\hat{w}_1 + \epsilon_{d-1}^{(k)} - w_{0,d-1}^{(k)} + w_0 = \epsilon^{(k)}, \tag{11.18c}$$

où, comme pour le cas de la synthèse de spectres non replié, les marges de fonctionnement des sous-familles ainsi que leurs poids synaptiques intermédiaires respectifs interviennent au travers des variables $\epsilon_{d-1}^{(i)}|_{i=1}^{k}$ et $w_{0,d-1}^{(i)}|_{i=1}^{k}$. La présence de l'équation (11.18a) est conditionnée par la forme réelle de la famille génératrice : $\{\mathcal{L}_d\}_2$ ou $\{\mathcal{L}_d\}_2$.

De même, la famille génératrice conjointe est alors une des deux familles suivantes :

$${}^{c}\{\mathcal{L}_d\}_1 = \begin{Bmatrix} \eta_1 - 1 & {}^{c}\{\mathcal{L}_{d-1}^{(1)}\} \\ \eta_1 - 2 & {}^{c}\{\mathcal{L}_{d-1}^{(2)}\} \\ \vdots & \\ \eta_1 - k & {}^{c}\{\mathcal{L}_{d-1}^{(k)}\} \\ \eta_1 - k - 1 & \{n_2 \quad \cdots \quad n_d\} \end{Bmatrix} \quad \text{ou} \quad {}^{c}\{\mathcal{L}_d\}_2 = \begin{Bmatrix} \eta_1 - 1 & {}^{c}\{\mathcal{L}_{d-1}^{(1)}\} \\ \eta_1 - 2 & {}^{c}\{\mathcal{L}_{d-1}^{(2)}\} \\ \vdots & \\ \eta_1 - k & {}^{c}\{\mathcal{L}_{d-1}^{(k)}\} \end{Bmatrix}.$$

La fonction d'agrégation prend les valeurs formelles suivantes ; la présence de la dernière dépendant là aussi de l'écriture générale de la famille génératrice conjointe

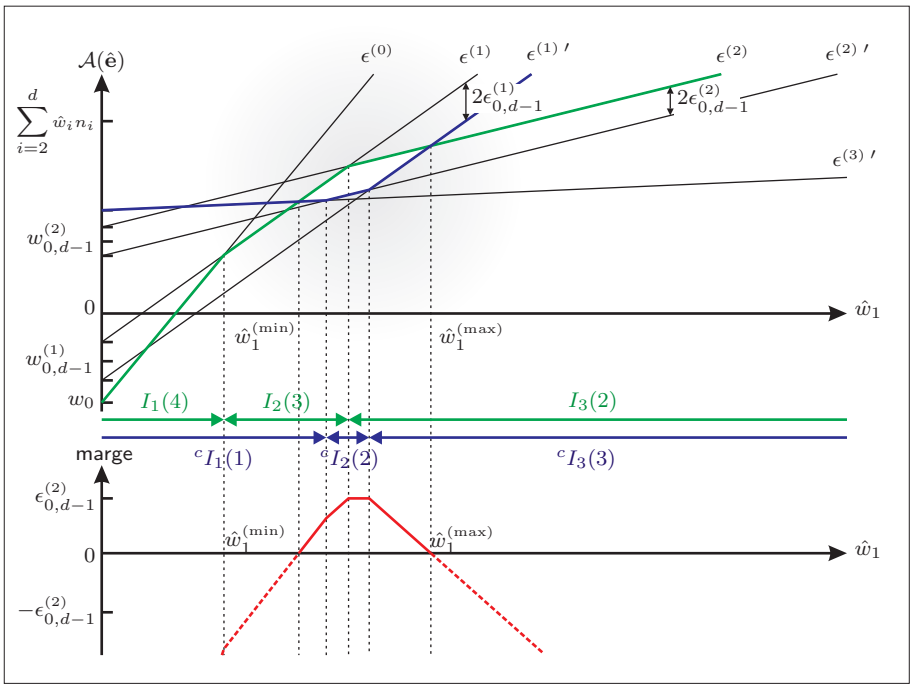


Figure 11.8
Influence de $\hat{w}_1$ sur la marge de fonctionnement d'un neurone binaire réalisant une famille génératrice de spectre replié.

réelle :

$$(\eta_1 - 1)\hat{w}_1 - \epsilon^{(1)}_{d-1} - w^{(1)}_{0,d-1} + w_0 = -\epsilon^{(1)\prime}, \tag{11.19a}$$

$$\vdots$$

$$(\eta_1 - k)\hat{w}_1 - \epsilon^{(k)}_{d-1} - w^{(k)}_{0,d-1} + w_0 = -\epsilon^{(k)\prime}, \tag{11.19b}$$

$$\text{et} \quad (\eta_1 - k - 1)\hat{w}_1 + \sum_{i=2}^{d} n_i \hat{w}_i + w_0 = -\epsilon^{(k+1)\prime}. \tag{11.19c}$$

En réalisant un tracé similaire à celui de la figure 11.5 de la page 150 il est possible de déterminer la meilleure valeur de $\hat{w}_1$. Ici, la figure 11.8 montre que l'intervalle de valeurs admissibles est cette fois-ci borné. Cela traduit l'essence du problème de la dépendance des poids synaptiques dont nous avons parlé au début de cette section. En effet, les valeurs de ces bornes sont fixées par les poids synaptiques $\hat{w}_i|_{i=2}^{d}$. Or leur calcul n'en a absolument pas tenu compte. Ainsi, une valeur de $\hat{w}_i|_{i=2}^{d}$ qui pouvait sembler optimale au moment de son calcul peut entrainer un intervalle de valeurs admissibles pour $\hat{w}_{j<i}$ trop étroit, voire même vide. Si ce cas de figure devait être rencontré au cours de la synthèse spectrale exacte, alors la méthode conclurait qu'il n'est pas possible de réaliser la famille génératrice avec un seul neurone binaire et qu'il faut soit l'approcher, soit synthétiser un réseau de neurones binaires. En réalité, cette conclusion ne serait donc pas nécessairement exacte.

La valeur de ces bornes sont données par les deux points d'intersection des droites

brisées définies par $\min(\epsilon^{(0)}, \epsilon^{(1)}, \ldots, \epsilon^{(k)})$ et $\max(\epsilon^{(1)\prime}, \ldots, \epsilon^{(k)}, \epsilon^{(k+1)})$. La valeur optimale de $\hat{w}_1$ est alors la plus petite valeur de cette intervalle pour laquelle la marge est la plus grande.

En pratique, elle est obtenue en construisant deux séries d'intervalles de valeurs pour $\hat{w}_1$: une première pour traduire les contraintes induites par la famille génératrice et une seconde pour celles induites par la famille génératrice conjointe :

$$I = \left[\hat{w}_1^{(\min)}; \hat{w}_1^{(\max)}\right] \cap I_1 \cup I_2 \cup \cdots \cup I_{k+1} \tag{11.20}$$

$$\text{et} \quad {}^cI = \left[\hat{w}_1^{(\min)}; \hat{w}_1^{(\max)}\right] \cap {}^cI_1 \cup {}^cI_2 \cup \cdots \cup {}^cI_{k+1}. \tag{11.21}$$

À chacun des intervalles $I_i|_{i=1}^{k+1}$ et ${}^cI_i|_{i=1}^{k+1}$ de ces séries est associée la valeur $\rho_i|_{i=1}^{k+1}$ et ${}^c\rho_i|_{i=1}^{k+1}$ du coefficient multiplicateur de $\hat{w}_1$ leur correspondant dans les jeux d'équations (11.18) et (11.19). Par conséquent, pour une valeur de $\hat{w}_1 \in I_i \cap {}^cI_j$ donnée, la marge de fonctionnement augmente avec $\hat{w}_1$ dans cette intersection d'intervalles si et seulement si $\rho_i > {}^c\rho_j$. Comme $\rho_i|_{i=1}^{k+1}$ décroit avec i et ${}^c\rho_i|_{i=1}^{k+1}$ croit avec i, la valeur optimale de $\hat{w}_1$ est, en parcourant I et ${}^cI$ des plus petites valeurs vers les plus grandes, la première valeur de $\hat{w}_1$ telle que $\hat{w}_1 \in I_i \cap {}^cI_j$ et $\rho_i \leq {}^c\rho_j$.

Le poids synaptique auxiliaire est alors très simplement obtenu par la relation suivante :

$$\hat{w}_0 = w_{0,d-1}^{(\eta_1-\rho_i)} - \rho_i \hat{w}_1. \tag{11.22}$$

La marge de fonctionnement est quant à elle calculée à partir de celle de la sous-famille génératrice $\{\mathcal{L}_{d-1}^{(\eta_{\rho_i}-1)}\}$ au travers de la relation suivante :

$$\epsilon = w_0 + \rho_i \hat{w}_1 + \epsilon_{d-1}^{(\eta_1-\rho_i)} - w_{0,d-1}^{(\eta_1-\rho_i)}. \tag{11.23}$$

Synthèse parallèle d'un poids synaptique Il s'agit d'un principe destiné à assurer que la synthèse séparée de plusieurs sous-familles génératrices conduise au même jeu de poids synaptiques afin de faciliter leur réunion en une seule famille. Cela est réalisé très simplement en calculant les deux séries d'intervalles des paragraphes précédents à partir de la réunion de toutes les sous-familles génératrices et génératrices conjointes concernées. Il n'y a alors plus qu'à calculer la valeur optimale du poids synaptique recherché.

La seule différence avec le cas de la synthèse d'un poids synaptique responsable d'un repliement est le fait que les poids synaptiques auxiliaires, ainsi que les marges de fonctionnement, sont calculés séparément sur chaque sous-familles génératrices. Cette synthèse a donc pour résultat une seule valeur de poids synaptique, et autant de poids synaptiques auxiliaires et de marges de fonctionnement que de sous-familles synthétisées en parallèle.

À titre d'explications complémentaires, nous allons utiliser la synthèse spectrale exacte décrite par l'algorithme 11.6, pour calculer le neurone binaire réalisant exactement $\mathcal{L}_4$ avec pour marge de fonctionnement la valeur $\epsilon = 1$.

Algorithme 11.6
Synthèse exacte d'une famille génératrice.

```
/* Fonction synthese_exacte:
Retourne les poids synaptiques réalisant exactement une famille génératrice.
   Paramètres:
/tableau_Fg/ - tableau de familles génératrices à synthétiser en parallèle*/

fonction synthese_exacte(tableau_Fg) = flottant[] {
        // /nbrefamilles/ - nombre de familles dans tableau_Fg
        entier nbrefamilles = tableau_Fg.nbrefamilles();
        // /poids1/ - valeur du poids synaptique à synthétiser
        flottant poids1 = 1;

        // si les familles dans tableau_Fg n'ont qu'une dimension réduite
        // synthétiser par la méthode spectrale non repliée
        si (tableau_Fg.nbrevariables() == 1) {
                pour (i=1;i<=nbrefamilles;i++) {
                        poids1 = max(poids1,synthese_spectrale_nonreplie(tableau[i\
                            ]);
                }
                retourner (poids1);
        }

        // /sous_tableau_Fg/ - tableau des sous-familles d'ordre 1
        sous_tableau_Fg = tableau_Fg.sousfamilles(1);
        // /poids/ poids synaptiques obtenus par récurrence
        flottant[] poids = synthese_exacte(sous_tableau_Fg);

        // construction des de I et de cI
        intervalles I = tableau_Fg.intersections(poids);
        intervalles cI = tableau_Fg.intersections_conjoints(poids);

        // recherche du meilleurs w1:
        flottant w1 = meilleurs(I,cI);

        retourner([w1 poids]);
}
```

Exemple 11.6 — Synthèse spectrale parallèle.
La famille génératrice de $\mathcal{L}_4$ et sa famille conjointe s'écrivent :

$$\{\mathcal{L}_4\} = \begin{Bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 2 & 1 & 0 & 0 \\ 0 & 2 & 0 & 2 & 0 \\ 0 & 1 & 1 & 2 & 3 \end{Bmatrix} \qquad \text{et} \qquad {}^c\{\mathcal{L}_4\} = \begin{Bmatrix} 0 & 2 & 0 & 1 & 3 \\ 0 & 1 & 1 & 2 & 2 \\ 0 & 1 & 1 & 1 & 3 \\ 0 & 1 & 0 & 2 & 3 \\ 0 & 0 & 1 & 2 & 3 \end{Bmatrix}.$$

La décomposition de $\{\mathcal{L}_4\}$ en sous-familles génératrices donne :

$$\{\mathcal{L}_4\} \rightarrow \begin{matrix} 1 \\ 0 \\ 0 \\ 0 \end{matrix} \begin{Bmatrix} 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 2 & 0 & 2 & 0 \\ 1 & 1 & 2 & 3 \end{Bmatrix} \rightarrow \begin{matrix} 2 \\ 2 \\ \rightarrow 1 \end{matrix} \begin{Bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 1 & 2 & 3 \end{Bmatrix} \rightarrow \begin{matrix} 1 \\ 0 \\ 1 \end{matrix} \begin{Bmatrix} 0 & 0 \\ 2 & 0 \\ 2 & 3 \end{Bmatrix} \rightarrow 2\{0\} \rightarrow 2\{3\}.$$

La synthèse du poids synaptique $\hat{w}_5$, est du ressort de la synthèse spectrale non repliée, avec pour marge de fonctionnement $\epsilon = 1$. Elle doit être effectuée en parallèle pour les deux sous-familles d'ordre 4, mais comme seule celle dont la racine d'ordre 4 vaut $\begin{pmatrix} 0 & 1 & 1 & 2 \end{pmatrix}$ dépend de cette variable réduite, la synthèse de $\hat{w}_5$ se fait en suivant l'équation (11.13) :

$$\hat{w}_5 = 2\epsilon = 2 \quad \text{et} \quad w_{0,5}^{(2)} = (1 - 2.3) = -5.$$

Le calcul du poids synaptique $\hat{w}_4$ fait intervenir deux sous-familles d'ordre 3. Néanmoins, ce poids synaptique n'est responsable d'aucun repliement et il faut donc mener deux synthèses spectrales non repliées en parallèle :

$$\hat{w}_4 = \max(\hat{w}_4^{(1)}, \hat{w}_4^{(2)}) = \max(2, 6) = 6,$$

où $\hat{w}_4^{(1)}$ et $\hat{w}_4^{(2)}$ ont été obtenus par l'application de l'équation (11.13). Comme il s'agit d'une synthèse parallèle, les poids synaptiques auxiliaires intermédiaires doivent en revanche être calculés séparément grâce aux équations (11.13) et (11.22) :

$$w_{0,4}^{(1)} = -11 \quad \text{et} \quad w_{0,4}^{(2)} = -17.$$

Les deux marges de fonctionnement intermédiaires sont ici identiques et valent 1.
Le calcul du poids synaptique $\hat{w}_3$ correspond à un cas de figure exactement similaire, qui donne donc :

$$\hat{w}_3 = 18, \quad w_{0,3}^{(1)} = -11 \quad \text{et} \quad w_{0,3}^{(2)} = -35.$$

Les marges de fonctionnement sont toujours identiques à 1.
Le poids synaptique $\hat{w}_2$ est en revanche responsable d'un repliement. Il doit donc faire l'objet de la méthode de synthèse présentée dans cette section. Afin de visualiser la signification pratique des intervalles qui doivent être calculés, la figure 11.9 montre les différentes droites dont la synthèse doit tenir compte. Dans cet exemple, les deux séries d'intervalles sont :

$$I = [22\,;24]\,(2) \cup [24\,;26]\,(1) \quad \text{et} \quad {}^cI = [22\,;24]\,(1) \cup [24\,;26]\,(2).$$

Le coefficient directeur des segments de droite sont inscrits entre parenthèses pour chaque intervalle. La meilleure valeur de $\hat{w}_2$ est donc 24 puisque c'est la valeur à

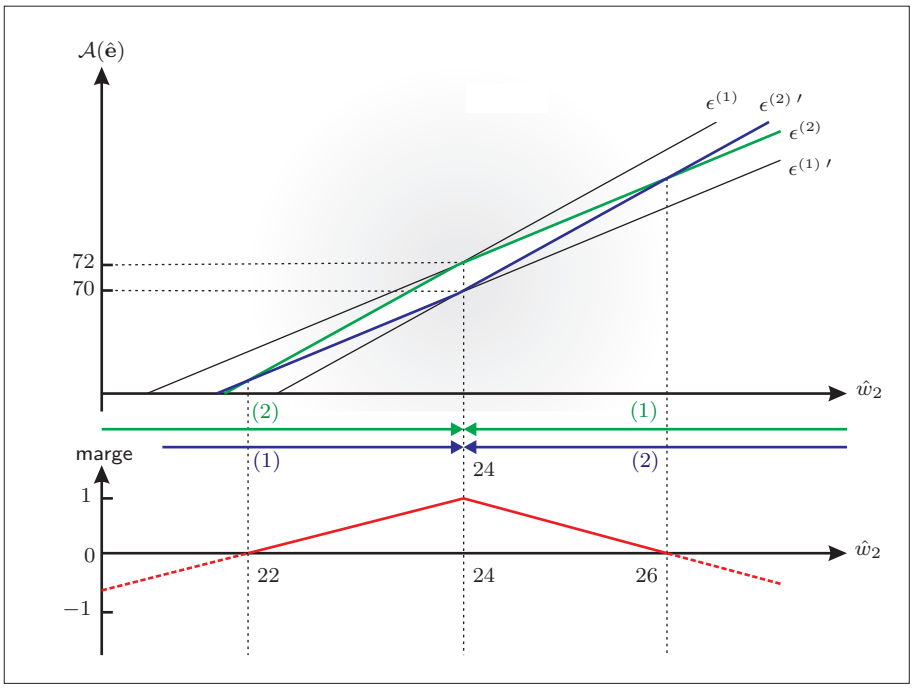


Figure 11.9
Calcul du poids $\hat{w}_2$ de l'exemple 11.6.

partir de laquelle le coefficient directeur de I vaut 1 et celui de ${}^{c}I$ vaut 2. Le poids synaptique auxiliaire intermédiaire de la sous-famille génératrice d'ordre 1 est donnée par l'équation (11.22) :

$$w_{0,2} = -w_{0,3}^{(1)} - 2\hat{w}_2 = -59,$$

et la marge de fonctionnement intermédiaire vaut toujours 1.
Le dernier poids synaptique n'étant pas responsable d'un repliement, la règle de synthèse spectrale non repliée peut lui être appliquée :

$$\hat{w}_1 = 60 \text{ et } w_0 = -59.$$

Les poids obtenus sont finalement :

$$\hat{\mathbf{w}} = \begin{pmatrix} 60 & 24 & 18 & 6 & 2 \end{pmatrix}^{\top} \text{ et } w_0 = -59.$$

La marge de fonctionnement correspond bien à la consigne, ce qui permet en outre d'affirmer qu'il s'agit de la solution la plus robuste possible. Si cela n'avait pas été le cas, alors il aurait été possible qu'une solution permettant d'avoir la marge de fonctionnement désirée existe. Le neurone binaire obtenu n'en réaliserait pas moins $\mathcal{L}_4$.

Bilan de la synthèse spectrale exacte Étant donné que, d'après les considérations du début de cette section, cet algorithme de synthèse n'est théoriquement pas capable de synthétiser une famille génératrice quelconque de façon optimale, cette méthode a été expérimentée sur plusieurs neurones binaires obtenus aléatoirement.

Malheureusement, si de nombreux cas de neurones binaires sont effectivement venus confirmer cette limitation, nous n'avons pas pu en dégager de caractéristique morphologique qui leur soit commune et qui permette, comme dans le cas de la synthèse spectrale non repliée, de caractériser les familles génératrices pour lesquelles la synthèse spectrale exacte fonctionne.

Cette lacune est malgré tout relativisée par le fait que la procédure suivie par cette méthode de synthèse n'aboutira pas si jamais la famille génératrice ne convenait pas. Cela donne ainsi l'assurance que les neurones binaires obtenus, à défaut d'être nécesairement optimaux, réalisent tout de même la fonction logique à seuil attendue. C'est la raison pour laquelle nous jugeons que la synthèse spectrale exacte, pour imparfaite qu'elle puisse être, a sa place parmi les méthodes de synthèse analytiques de fonctions logiques à seuil.

D'une manière générale, l'approche spectrale de la synthèse de neurones binaires souffre, commes les méthodes de synthèse par inégalités, de l'absence de condition nécessaire et suffisante sur laquelle s'appuyer. En particulier, la synthèse spectrale s'appuie sur la structure récursive des familles génératrice, qui est un critère nécessaire mais pas suffisant. Dans ces conditions, la capacité des familles génératrices à représenter de façon efficace une fonction logique à seuil est donc elles aussi limitées, à l'instar des tableaux de vérité, par le nombre de variables de celles-ci. Plus ce nombre augmente, plus la proportion de familles génératrices correctement mises en forme et correspondant réellement à des fonctions logiques à seuil diminue. Pour les tableaux de vérité, cette proportion décroit rapidement de 100 % pour une entrée à 3 % pour quatre entrées, puis à 0,22 ‰ pour cinq entrées, etc. [Chen et al., 2006]. Une étude similaire pour les familles génératrices montrerait vraisemblablement que cette proportion diminue beaucoup moins vite avec le nombre de variables.

L'enseignement majeur apporté par ce chapitre est donc qu'une méthode de synthèse analytique se voulant générale doit nécessairement reposer sur un critère nécessaire et suffisant décrivant les fonctions logiques à seuil. À ce jour, le seul critère de ce type en est la définition originale (voir le chapitre 7) : celle-ci stipule qu'une fonction logique est une fonction logique à seuil si et seulement s'il existe des poids synaptiques permettant à un neurone binaire de la réaliser. Nous allons montrer dans le chapitre suivant qu'une interprétation algébrique de cette définition donne une vision neuve du problème de synthèse et révèle une nouvelle nature à la famille génératrice ; y dévoilant notament des redondances supplémentaires que l'approche que nous avons suivie jusque là a été incapable de déceler.

12

Synthèse géométrique d'une fonction logique à seuil

Dans les chapitres précédents, nous sommes arrivés à la conclusion que les méthodes de synthèse spectrales et, de façon plus générale, les méthodes de synthèse par inégalités, ne pouvaient pas constituer de solution satisfaisante au problème de synthèse analytique de neurones binaires. Optimiser la marge de fonctionnement au travers des méthodes par inégalités est en effet utopique sur de grands nombres de variables. Les méthodes de synthèse spectrale sont quant à elles beaucoup moins affectées par le nombre de variables, mais elles sont tout de même limitées par l'augmentation polynomiale du nombre de vecteurs générateurs dont elles doivent tenir compte.

Cet « échec [1] » a été attribué à l'absence de critère nécessaire et suffisant sur la morphologie d'une famille génératrice pour en faire une fonction logique à seuil. Pour espérer aller plus loin, il faut donc soit énoncer une telle condition, soit déplacer le problème dans un contexte où une telle condition existe déjà. C'est cette deuxième option qui est présentée dans ce chapitre.

[1] La synthèse spectrale n'a pas conduit à une solution globale du problème de la synthèse exacte et optimale de fonctions logiques à seuil.

12.1 Interprétation géométrique des fonctions logiques à seuil

Comme nous l'avons vu dans le chapitre 7, une fonction logique de $\mathbb{B}^n$ dans $\mathbb{B}$ est une fonction logique à seuil si et seulement s'il existe $n+1$ coefficients $w_i|_{i=0}^n$ tels que :

$$\forall \mathbf{e} \in \mathbb{B}^n, \quad (\mathcal{L}(\mathbf{e}) \equiv \mathsf{vrai}) \Leftrightarrow (w_0 + \mathbf{w}^\top \mathbf{e} > 0)\,. \tag{12.1}$$

Dans la partie II, ces coefficients ont été interprétés comme les poids synaptiques d'un neurone binaire qui réalise $\mathcal{L}$. D'après cette définition, le basculement de la sortie de $\mathcal{L}$ se produit lorsque la fonction d'agrégation de ce neurone s'annule :

$$w_0 + \mathbf{w}^\top \mathbf{e} = 0, \tag{12.2}$$

La sortie du neurone, et de la fonction logique à seuil, est alors indéterminée. Toutefois, cela ne justifie pas d'abandonner l'étude de ce cas de figure, dont l'existence est en fait au cœur de la définition des fonctions logiques à seuil.

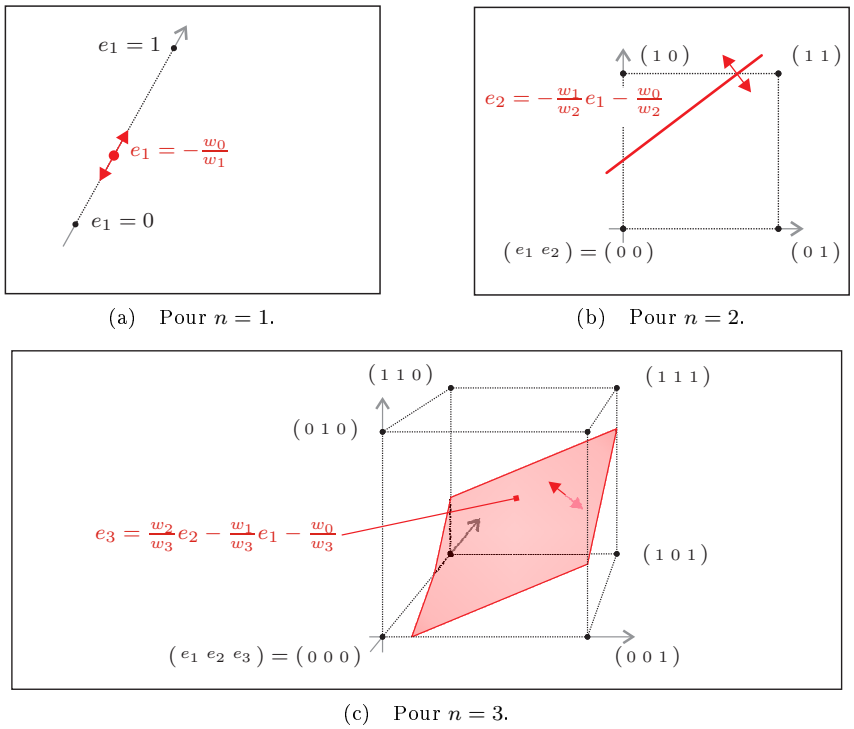
Géométrie pour de faibles dimensions Pour $d = 1$, l'équation (12.2) s'écrit $w_0 + w_1 e_1 = 0$. La seule valeur d'entrée vérifiant celle-ci s'en déduit immédiatement : $e_1 = -\frac{w_0}{w_1}$. Géométriquement, il s'agit d'un point de l'espace d'entrée de dimension 1, dont la position par rapport à $e_1 = 0$ et $e_1 = 1$ détermine la nature de la fonction logique réalisée ; comme le montre la figure 12.1a.

Dans le cas $d = 2$, l'espace d'entrée a deux dimensions et l'équation (12.2) s'écrit maintenant $w_0 + w_1 e_1 + w_2 e_2 = 0$. Pour une entrée e_1 fixée, il y a maintenant une infinité de valeurs de e_2 vérifiant cette équation : $e_2 = -\frac{w_0}{w_2} - \frac{w_1}{w_2} e_1$. Géométriquement, il s'agit d'une équation de droite, comme le confirme la figure 12.1b. En fonction de sa position relativement aux vecteurs d'entrée $\begin{pmatrix}0 & 0\end{pmatrix}^\top$, $\begin{pmatrix}0 & 1\end{pmatrix}^\top$, $\begin{pmatrix}1 & 0\end{pmatrix}^\top$ et $\begin{pmatrix}1 & 1\end{pmatrix}^\top$, elle fixe là encore la fonction logique réalisée.

Enfin, pour $d = 3$, l'équation (12.2) devient $w_0 + w_1 e_1 + w_2 e_2 + w_3 e_3 = 0$, ce qui correspond à l'équation d'un plan en dimension 3 dont la séparation des huit vecteurs d'entrée détermine la fonction logique réalisée (voir la figure 12.1c).

Dans cette section, ce raisonnement, ébauché sur des espaces dont la dimension est suffisament faible pour être intuitivement explicite, va être généralisé à des espaces de dimension quelconque. Il sera ensuite appliqué à l'espace réduit dans lequel les méthodes de synthèse spectrale du chapitre précédent ont été élaborées.

Géométrie en dimension quelconque Pour des valeurs de n supérieures, l'interprétation géométrique précédente devient difficile à représenter graphiquement. Algébriquement, ce raisonnement se généralise toutefois sans aucune difficulté. L'équation (12.2) représente alors un « hyperplan » de dimension $n-1$ séparant les 2^n



(a) Pour $n = 1$.

(b) Pour $n = 2$.

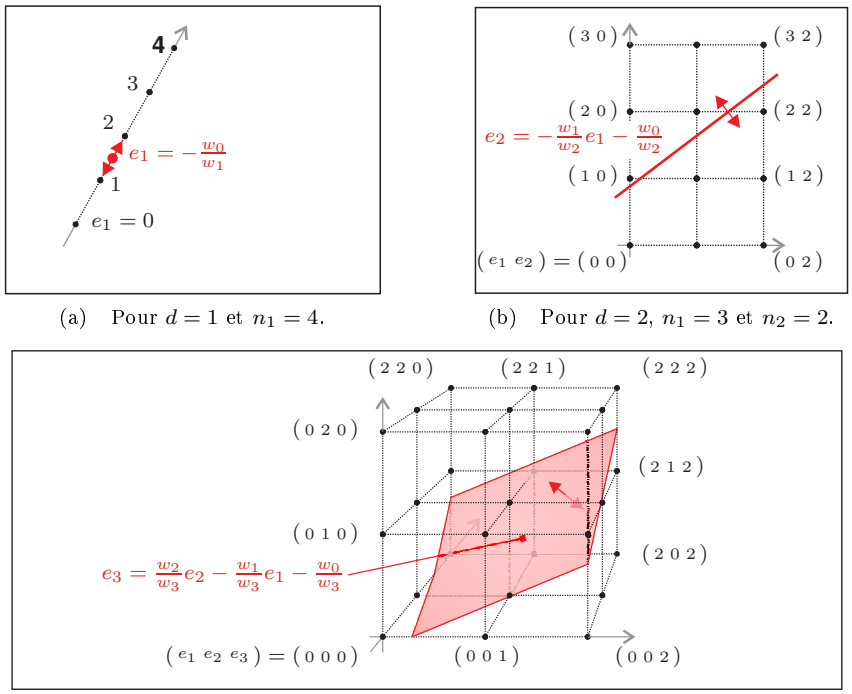
(c) Pour $n = 3$.

Figure 12.1
Interprétation géométrique du fonctionnement d'un neurone binaire.

vecteurs d'entrée d'un espace de dimension n. Ces derniers sont disposés aux sommets d'un « hypercube » qui est défini comme la généralisation à n dimensions d'un segment de droite en dimension 1, d'un carré en dimension 2 et d'un cube en dimension 3 [Kohavi, 1978].

Dans ce contexte, une fonction logique à seuil est également appelée « fonction logique linéairement séparable » afin de traduire l'existence d'un hyperplan séparant les sommets de l'hypercube pour lesquels la fonction logique est vraie de ceux pour lesquels elle est fausse. La synthèse d'un neurone binaire peut par conséquent s'interpréter comme le calcul des coordonnées d'un hyperplan séparant exactement ces deux types de sommets. Le réglage de la marge de fonctionnement correspond alors au positionnement de celui-ci de sorte qu'il passe le plus loin possible du sommet de l'hypercube dont il est le plus proche.

Géométrie dans un espace réduit L'analyse des fonctions logiques réalisées par les neurones binaires, menée dans la partie II, a conduit à la définition d'un espace réduit au sein duquel une fonction logique à seuil $\mathcal{L}$ s'exprime par le biais de sa fonction spectre $\mathcal{S}_\mathcal{L}$. La construction de cet espace est régie par le processus de réduction, qui représente par une entrée réduite les entrées symétriques de la fonction logique à seuil. En conséquence, les entrées réduites ne sont pas des variables binaires, mais n-aires, c'est-à-dire qu'elles varient dans $[\![0\,;n_i]\!]$, où $n_i|_{i=1}^{d}$ est le nombre de variables d'entrées symétriques réduites dans la i^{e} variable réduite $\hat{e}_i$.



(a) Pour $d = 1$ et $n_1 = 4$.

(b) Pour $d = 2$, $n_1 = 3$ et $n_2 = 2$.

(c) Pour $d = 3$, $n_1 = 2$, $n_2 = 2$ et $n_3 = 2$.

Figure 12.2
Interprétation géométrique de l'espace réduit.

Par ailleurs, nous avons vu dans le chapitre 10 que les descripteurs de CHOW impliquaient que ces variables symétriques devaient nécessairement être affectées de poids synaptiques identiques. L'équation (12.2) peut donc être transposée dans l'espace réduit, en appelant $\hat{\mathbf{w}}$ les poids synaptiques correspondant aux variables réduites $\hat{\mathbf{e}}$:

$$w_0 + \hat{\mathbf{w}}^\top \hat{\mathbf{e}} = 0, \tag{12.3}$$

Il s'agit alors de l'équation d'un hyperplan de dimension $d-1$ dans l'espace réduit de dimension d. Dans cet espace, les vecteurs réduits ne sont plus nécessairement disposés sur les sommets d'un hypercube, mais sur les intersections d'un « pavage » d'hypercubes dont l'étendue est bornée par les valeurs $n_i|_{i=1}^{d}$. La figure 12.2 illustre ce pavage pour les dimensions 1, 2 et 3. La généralisation à une dimension quelconque est triviale, bien qu'il soit impossible de la représenter. Par abus de langage, nous continuerons à dénommer cette structure hypercube.

12.2 Interprétation géométrique des familles génératrices

Grâce à la monotonie des fonctions logiques à seuil, le chapitre 7 a pu formuler un processus de génération par lequel chaque vecteur réduit évaluant une fonction logique à vrai implique l'évaluation à vrai d'autres vecteurs réduits. L'ensemble formé des vecteurs réduits évaluant une fonction logique à seuil à vrai, sans être eux-mêmes

générés, constitue la famille génératrice de cette fonction logique. Les vecteurs réduits d'une telle famille sont appelés vecteurs générateurs, et cette section traite de leur interprétation géométrique, dans l'espace réduit formé de l'hypercube défini dans la section précédente.

Proposition 12.1 (Échantillonnage d'un hyperplan).
Soit une fonction logique à seuil $\mathcal{L}$ de $\mathbb{L}_s^+$ définie par les poids synaptiques $\hat{\mathbf{w}}$ et w_0, et dont l'analyse donne la famille génératrice $\{\mathcal{L}\}$ et la famille génératrice conjointe ${}^c\{\mathcal{L}\}$.

Les vecteurs générateurs de $\{\mathcal{L}\}$ sont, parmi les vecteurs évaluant $\mathcal{L}$ à vrai, *les plus proches de l'hyperplan défini par $\hat{\mathbf{w}}$ et w_0. De même, les vecteurs générateurs conjoints de ${}^c\{\mathcal{L}\}$ sont, parmi les vecteurs évaluant $\mathcal{L}$ à* faux, *les plus proches de l'hyperplan.*

La réunion de $\{\mathcal{L}\}$ et ${}^c\{\mathcal{L}\}$ réalise en quelque sorte un échantillonnage de l'hyperplan.

Démonstration. Dans l'espace réduit borné par la structure d'hypercube, montrons que les vecteurs générateurs ainsi que les vecteurs générateurs conjoints sont exactement les points les plus proches de l'hyperplan défini par $\hat{\mathbf{w}}$ et w_0.

Soit $\hat{\mathbf{e}}$ un vecteur générateur de $\mathcal{L}$. Sa distance à l'hyperplan est évaluée en effectuant la projection orthogonale de $\hat{\mathbf{e}}$ sur la normale de l'hyperplan dirigée par $\hat{\mathbf{w}}$ et en comparant la coordonnée obtenue avec celle de n'importe quel vecteur de l'hyperplan, $-w_0$:

$$\mathcal{D}(\hat{\mathbf{e}}) = |\hat{\mathbf{w}}^\top \hat{\mathbf{e}} - (-w_0)|.$$

Tous les termes de cette somme étant positifs, sauf peut-être w_0, la seule façon pour une entrée $\hat{\mathbf{e}}'$ d'être plus proche de l'hyperplan que $\hat{\mathbf{e}}$ est qu'elle ait ses composantes inférieures ou égales à celles de $\hat{\mathbf{e}}$. Or si un tel vecteur évaluait $\mathcal{L}$ à vrai, il générerait $\hat{\mathbf{e}}$ ce qui est impossible par hypothèse.

Réciproquement si $\hat{\mathbf{e}}$ est un vecteur réduit tel que $\mathcal{L}(\hat{\mathbf{e}}) \equiv$ vrai et tel que pour tout $\hat{\mathbf{e}}'$ vérifiant $\mathcal{L}(\hat{\mathbf{e}}') \equiv$ vrai on ait :

$$\mathcal{D}(\hat{\mathbf{e}}') > \mathcal{D}(\hat{\mathbf{e}}),$$

alors $\hat{\mathbf{e}}$ est nécessairement un vecteur générateur de $\mathcal{L}$ puisqu'aucun des vecteurs $\hat{\mathbf{e}}'$ ne peut le générer.

De façon similaire, les vecteurs générateurs conjoints de ${}^c\{\mathcal{L}\}$ vérifient la proposition 12.1. □

Remarque. Par unicité de la famille génératrice, il apparaît que cette dernière est un échantillonnage de n'importe quel hyperplan séparant l'hypercube conformément à une même fonction logique à seuil.

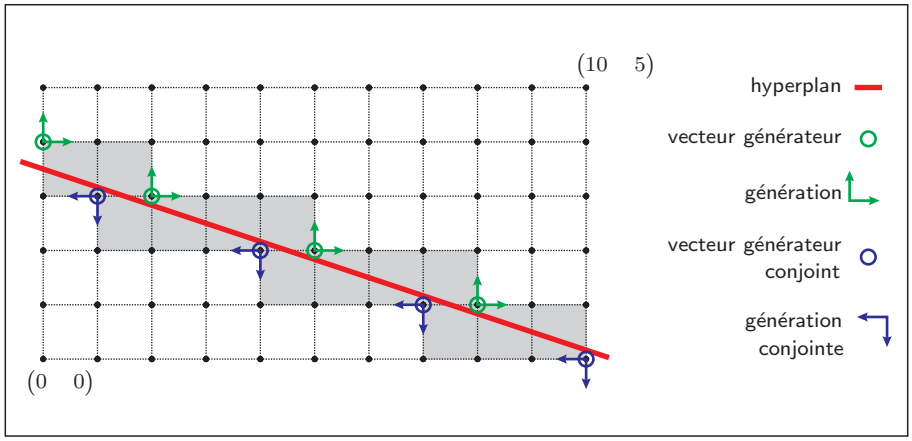


Figure 12.3
Illustration de l'échantillonnage réalisé par une famille génératrice et sa famille génératrice conjointe.

La figure 12.3 précise le rôle d'échantillons que nous venons d'attribuer aux vecteurs générateurs des deux familles caractérisant une fonction logique à seuil. Elle représente un espace réduit de deux dimensions et montre que ces vecteurs délimitent une bande qui encadre exactement, et au plus près, l'hyperplan.

En utilisant un vocabulaire emprunté au domaine de l'imagerie numérique, les familles génératrices et génératrices conjointent définissent les pixels qu'il faudrait noircir pour tracer l'hyperplan ; faisant ainsi apparaître l'*aliasing* caractéristique d'une résolution trop faible. En ces termes, synthétiser un neurone binaire optimal à partir d'une famille génératrice revient à retrouver l'hyperplan le mieux décrit par cet échantillonnage.

12.2.1 Les vecteurs générateurs supports

Algébriquement, pour un espace vectoriel de dimension d, un hyperplan est un sous-espace vectoriel de dimension $d-1$. Il est donc complètement engendré par $d-1$ vecteurs libres $\hat{\mathbf{v}}_i|_{i=1}^{d-1}$, c'est-à-dire d points : un point origine, et $d-1$ points dirigeant les $d-1$ vecteurs.

Remarque. Nous appelons « point » un vecteur dont l'origine est intrinsèquement fixée en $\begin{pmatrix}0 & 0 & \cdots & 0\end{pmatrix}^\top$. C'est entre autres le cas des vecteurs générateurs qui sont donc, dans le contexte de ce chapitre, considérés comme des points. En revanche, ce n'est pas le cas des vecteurs libres $\hat{\mathbf{v}}_i|_{i=1}^{d-1}$ formant la base de l'hyperplan recherché, dans le sens où l'information qu'ils véhiculent n'est pas la position du point dont ils portent les coordonnées, mais la direction qu'ils indiquent.

La proposition suivante établi qu'une base de vecteurs libres de l'hyperplan réalisant de façon optimale une fonction logique à seuil, est nécessairement définie par des points appartenant aux familles génératrice et génératrice conjointe de celle-ci. Ce résultat permettra de ramener le problème de synthèse à celui de l'identification de

ces points parmi ces deux familles.

Proposition 12.2 (Orientation optimale d'un hyperplan).
Un hyperplan, séparant de façon optimale les sommets de l'hypercube défini par une fonction logique à seuil $\mathcal{L} \in \mathbb{L}_s^+$, est dirigé par $d-1$ vecteurs $\hat{\mathbf{v}}_i|_{i=1}^{d-1}$ définis par :

1. *ou bien exactement d vecteurs générateurs (générateurs conjoints) ;*
2. *ou bien $d-p$ vecteurs générateurs et $p+1$ vecteurs générateurs conjoints ; p étant un entier positif caractéristique de la fonction logique à seuil.*

Démonstration. La démonstration de cette proposition se base sur le fait qu'un hyperplan optimal est défini comme celui donnant la plus grande marge de fonctionnement au neurone binaire correspondant. En d'autres termes, sa distance avec le vecteur générateur ou générateur conjoint de $\mathcal{L}$ le plus proche est maximale. Or nous avons vu qu'il fallait définir $d-1$ vecteurs libres pour engendrer un hyperplan, c'est-à-dire d points bien choisis[2], ce qui revient à fixer ses d degrés de liberté.

En choisissant les d points les plus proches comme contraintes, il est ainsi possible d'orienter l'hyperplan de sorte qu'il leur soit distant de façon égale. Le résultat obtenu est un hyperplan parallèle[3] à celui supporté par ces points. Les $d-1$ vecteurs, obtenus en les soustrayant deux à deux, forment donc une base des deux hyperplans parallèles, donc de l'hyperplan optimal recherché[4].

Les deux possibilités de la proposition se déduisent alors de la localisation de ces d points : s'ils sont tous dans la même famille — génératrice ou génératrice conjointe — alors les $d-1$ vecteurs libres s'en déduisent immédiatement. S'ils sont répartis dans les deux familles, alors les vecteurs obtenus en soustrayant deux points pris dans des familles différentes représentent une direction traversale à l'hyperplan recherché et ne font donc pas partie de son espace vectoriel. Par conséquent, seuls les vecteurs $\hat{\mathbf{v}}_i$ obtenus par la soustraction deux à deux de points d'une même famille doivent être considérés, ce qui implique la présence d'un point supplémentaire dans l'une ou l'autre famille. Ainsi, si $d-p$ points sont dans la famille génératrice, ils supportent $d-p-1$ vecteurs et il faut alors $p+1$ points dans la famille génératrice conjointe pour supporter p vecteurs, et ainsi atteindre le total de $d-1$. □

Les vecteurs générateurs et générateurs conjoints supportant les vecteurs $\hat{\mathbf{v}}_i|_{i=1}^{d-1}$ sont appelés vecteurs supports et leur identification fera l'objet d'un algorithme décrit dans la dernière section de ce chapitre.

Définition 12.1 (Vecteurs supports).
Les vecteurs supports d'une fonction logique à seuil $\mathcal{L}$ de $\mathbb{L}_s^+$ sont les $k \in [\![d\,;d+1]\!]$ vecteurs générateurs ou générateurs conjoints $\tilde{\mathbf{e}}$ de $\{\mathcal{L}\} \cup {}^c\{\mathcal{L}\}$ définissant une base de $d-1$ vecteurs libres $\hat{\mathbf{v}}_i|_{i=1}^{d-1}$ de l'hyperplan séparant $\mathcal{L}$ de façon optimale.

[2] Ils doivent supporter des vecteurs libres.

[3] Ou encore admettant le même sous-espace vectoriel supplémentaire orthogonal.

[4] En réalité, ce qui discrimine ces deux hyperplans est leur valeur w_0 et non celle de $\hat{\mathbf{w}}$

12.2.2 Synthèse de vecteurs supports

Soit une fonction logique à seuil de d variables réduites dont on a identifié les vecteurs supports. Dès lors, leur soustraction deux à deux donne une base de $d-1$ vecteurs engendrant l'hyperplan optimal correspondant à cette fonction. Soient $\hat{\mathbf{v}}^{(i)}|_{i=1}^{d-1}$ ces vecteurs. Dans l'espace vectoriel de dimension d, le sous-espace vectoriel représenté par cet hyperplan admet un espace vectoriel supplémentaire de dimension 1, autrement dit une droite vectorielle.

Or, le vecteur $\hat{\mathbf{w}}$ qui définit l'orientation de l'hyperplan de séparation est justement un vecteur engendrant un des espaces supplémentaires de ce dernier. En l'occurrence, il s'agit de l'unique espace supplémentaire qui lui soit orthogonal. Retrouver $\hat{\mathbf{w}}$ étant donnés les vecteurs $\hat{\mathbf{v}}^{(i)}|_{i=1}^{d-1}$ devient donc un problème d'algèbre linéaire consistant à trouver un vecteur engendrant l'espace supplémentaire orthogonal à l'hyperplan de séparation [5].

Calcul des poids synaptiques La solution la plus immédiate pour obtenir $\hat{\mathbf{w}}$ est de l'exprimer comme le résultat du produit vectoriel des vecteurs $\hat{\mathbf{v}}^{(i)}|_{i=1}^{d-1}$:

$$\hat{\mathbf{w}} = \hat{\mathbf{v}}^{(1)} \wedge \hat{\mathbf{v}}^{(2)} \wedge \cdots \wedge \hat{\mathbf{v}}^{(d-1)}. \tag{12.4}$$

Le vecteur résultat de ce produit est en effet défini de sorte que pour tout vecteur réduit $\hat{\mathbf{e}}$, on ait :

$$\hat{\mathbf{w}}^\top \hat{\mathbf{e}} = \det(\hat{\mathbf{v}}^{(1)}, \ldots, \hat{\mathbf{v}}^{(d-1)}, \hat{\mathbf{e}}), \tag{12.5}$$

$$\sum_{i=1}^{d} \hat{w}_i \hat{e}_i = \begin{vmatrix} v_1^{(1)} & \cdots & v_1^{(d-1)} & e_1 \\ \vdots & . & \vdots & \vdots \\ v_d^{(1)} & \cdots & v_d^{(d-1)} & e_d \end{vmatrix}. \tag{12.6}$$

Autrement dit, les produits $\hat{\mathbf{w}}^\top \hat{\mathbf{v}}^{(i)}|_{i=1}^{d-1}$ sont nuls puisque les déterminants calculés comptent alors systématiquement deux vecteurs identiques. Par conséquent, $\hat{\mathbf{w}}$ est bien orthogonal à chacun de ces vecteurs et se compose donc bien des poids synaptiques définissant l'hyperplan recherché. Ceux-ci s'expriment respectivement par le calcul du déterminant suivant :

$$\hat{w}_i = (-1)^{d+i} \begin{vmatrix} v_{1,1} & \cdots & v_{d-1,1} \\ \vdots & . & \vdots \\ v_{1,i-1} & \cdots & v_{d-1,i-1} \\ v_{1,i+1} & \cdots & v_{d-1,i+1} \\ \vdots & . & \vdots \\ v_{1,d} & \cdots & v_{d-1,d} \end{vmatrix}. \tag{12.7}$$

[5] En algèbre linéaire, l'existence de ce dernier est garantie par le fait que l'espace réduit a une dimension finie.

Cette expression est obtenue en décomposant le déterminant de l'équation (12.5) selon sa dernière colonne. On peut alors identifier chaque valeur w_i au sous-déterminant obtenu puisque l'égalité doit être valable pour tout $\hat{\mathbf{e}}$.

Calcul du poids synaptique auxiliaire L'hyperplan calculé dans le paragraphe précédent a une orientation qui le fait reposer sur les vecteurs supports de la famille génératrice et génératrice conjointe. Par conséquent, la position optimale de cet hyperplan, au sens de la valeur de w_0 et non plus de celle de $\hat{\mathbf{w}}$, est celle qui place les vecteurs supports des deux familles à la même distance. Ainsi, pour deux vecteurs supports $\tilde{\mathbf{e}}$ et $\tilde{\mathbf{e}}'$, respectivement dans la famille génératrice et génératrice conjointe de la fonction logique, on a :

$$w_0 + \hat{\mathbf{w}}^\top \tilde{\mathbf{e}} = \epsilon \quad \text{et} \quad w_0 + \hat{\mathbf{w}}^\top \tilde{\mathbf{e}}' = -\epsilon, \tag{12.8}$$

qui se traduit immédiatement par :

$$w_0 = -\hat{\mathbf{w}}^\top \left(\frac{\tilde{\mathbf{e}} + \tilde{\mathbf{e}}'}{2} \right). \tag{12.9}$$

D'un point de vue complexité, le calcul d'un déterminant par des méthodes inspirées du pivot de Gauss varie typiquement en $\mathcal{O}(d^3)$. La complexité du calcul complet des d poids synaptiques varie donc en $\mathcal{O}(d^4)$.

12.3 Détection des vecteurs supports

Les sections précédentes ont établi l'existence de vecteurs générateurs particuliers parmi les vecteurs des familles génératrice et génératrice conjointe d'une fonction logique à seuil : les vecteurs supports. Ceux-ci sont remarquables par le fait qu'ils supportent l'hyperplan réalisant cette fonction de façon optimale. Géométriquement, leur existence révèle donc une redondance dans les vecteurs générateurs puisque, finalement, les vecteurs supports suffisent à décrire complètement une fonction logique à seuil. Il s'agit donc maintenant d'identifier un mécanisme capable de trahir cette redondance et d'éliminer les vecteurs générateurs et générateurs conjoints qui ne sont pas des vecteurs supports.

Supposons savoir que $\hat{\mathbf{e}} \in \{\mathcal{L}\}$ n'est pas un vecteur support de $\mathcal{L}$. Dans ce cas, il est possible d'affirmer que tout vecteur $\hat{\mathbf{v}} = \tilde{\mathbf{e}} - \hat{\mathbf{e}}$, où $\tilde{\mathbf{e}}$ est un vecteur support de la famille génératrice, engendre un espace vectoriel supplémentaire de l'hyperplan optimal. Dans le cas contraire, $\hat{\mathbf{e}}$ serait par définition un vecteur support, puisque $\hat{\mathbf{v}}$ serait alors un des vecteurs engendrant l'hyperplan.

Dès lors, la direction donnée par $\hat{\mathbf{v}}$ est transversale à l'hyperplan, de sorte qu'il existe un vecteur réduit $\hat{\mathbf{e}}'$ tel que $\hat{\mathbf{e}}' \pm \hat{\mathbf{v}}$ ne soit pas un vecteur générateur de $\mathcal{L}$.

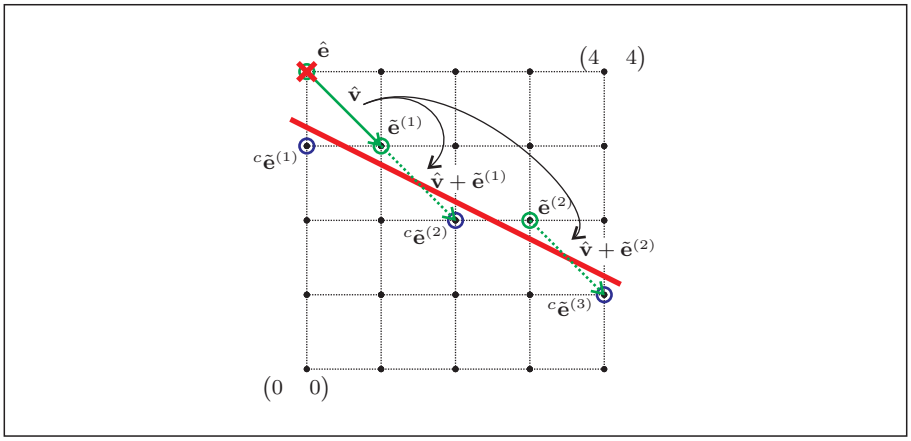


Figure 12.4
Élimination d'un vecteur générateur à l'origine d'un vecteur engendrant un espace vectoriel supplémentaire de l'hyperplan.

Naturellement, cette remarque vaut pour un vecteur $\hat{\mathbf{v}}$ construit à partir d'un vecteur générateur conjoint et d'un vecteur support de ${}^{c}\{\mathcal{L}\}$. La figure 12.4 illustre ce phénomène en dimension 2.

Nous proposons d'utiliser cette propriété pour éliminer les vecteurs générateurs qui ne seraient pas des vecteurs supports. L'algorithme obtenu doit être exécuté séparément dans la famille génératrice de la fonction logique et dans sa famille génératrice conjointe, afin d'identifier la totalité des vecteurs supports.

Proposition 12.3 (Détection des vecteurs supports d'une famille génératrice).
Soient $\hat{\mathbf{e}}$ *et* $\hat{\mathbf{e}}'$ *deux vecteurs générateurs d'une fonction logique* $\mathcal{L}$ *et soit* $\hat{\mathbf{v}} = \hat{\mathbf{e}} - \hat{\mathbf{e}}'$. *Dès lors pour tout vecteur générateur* $\hat{\mathbf{e}}'' \in \{\mathcal{L}\}$ *:*

- *si* $\mathcal{L}(\hat{\mathbf{e}}'' + \hat{\mathbf{v}}) \equiv$ faux *alors* $\hat{\mathbf{e}}'$ *n'est pas un vecteur support ;*
- *si* $\mathcal{L}(\hat{\mathbf{e}}'' + \hat{\mathbf{v}}) \equiv$ vrai *alors* $\hat{\mathbf{e}}$ *n'est pas un vecteur support ;*
- *si* $\mathcal{L}(\hat{\mathbf{e}}'' \pm \hat{\mathbf{v}}) \in \{\mathcal{L}\}$ *alors* $\hat{\mathbf{e}}''$ *n'est pas un vecteur support ;*

Démonstration. La preuve de cette proposition est au cœur des recherches postérieures aux travaux présentés dans ce manuscrit. À l'heure actuelle, ces résultats nous semblent trouver une justification — à défaut d'une démonstration — dans le fait que tout vecteur, obtenu par la soustraction d'un vecteur générateur par un vecteur support, est transversal à l'hyperplan. Il conduit donc nécessairement à un vecteur hors de la famille génératrice. Pour chaque cas, ces trois tests identifient alors le vecteur responsable de la transversalité de $\hat{\mathbf{v}}$.

Le dernier test est un peu particulier puisqu'il conduit à éliminer des vecteurs générateurs qui pourraient être des vecteurs supports mais qui se trouvent être redondants, comme le montre la figure 12.5. □

Remarque. Il arrivera souvent que les vecteurs réduits obtenus à partir de $\hat{\mathbf{v}}$ soient positionnés en dehors de l'hypercube, c'est-à-dire qu'une au moins de leurs coordon-

Algorithme 12.1
Synthèse géométrique d'une famille génératrice.

```
/* Fonction synthese_ geometrique:
Retourne le complément d'une famille génératrice.
   Paramètres:
/Fg/ - famille génératrice (tableau de k lignes et d colonnes)*/

fonction synthese_geometrique(Fg) = flottant[] {
        // /conjointe_Fg/ - tableau stockant la famille conjointe de Fg
        famillegeneratrice conjointe_Fg;
        // /k/ - nombre de vecteurs générateurs
        entier k = Fg.nbrevecteurs();
        // /ck/ - nombre de vecteurs générateurs conjoints
        entier ck = conjointe_Fg.nbrevecteurs();

        // /support/ - tableau de vecteurs supports de la famille génératrice
        vecteurreduit[] support = Fg;

        // calcul des vecteurs supports de la famille génératrice
        pour (i=1;i<=k;i++) {
                pour (j=i;j!=i;j<=k;j++) {
                        // /v/ - direction potentiellement transversale
                        vecteurreduit v =Fg[i]-Fg[j];
                        // les trois tests de la transversalité éliminant
                        // un vecteur support potentiel
                        v.test(support);
                }
        }

        // /csupport/ - tableau de vecteurs supports de la famille génératrice |
             conjointe
        vecteurreduit[] csupport = conjointe_Fg;
        // calcul des vecteurs supports de la famille génératrice
        pour (i=1;i<=k;i++) {
                pour (j=i;j!=i;j<=k;j++) {
                        // /v/ - direction potentiellement transversale
                        vecteurreduit v =conjointe_Fg[i]-conjointe_Fg[j];
                        // les trois tests de la transversalité éliminant
                        // un vecteur support potentiel
                        v.test(csupport);
                }
        }

        // /vecteurs/ - tableau de vecteurs engendrant l'hyperplan
        // obtenus en faisant la différence deux à deux des vecteurs supports
        vecteurgenerateur[] vecteurs = [support.difference() + csupport.difference()];

        // /poids/ - tableau de poids synaptiques
        flottant[] poids = produit_vectoriel(vecteurs);

        retourner(poids);
}
```

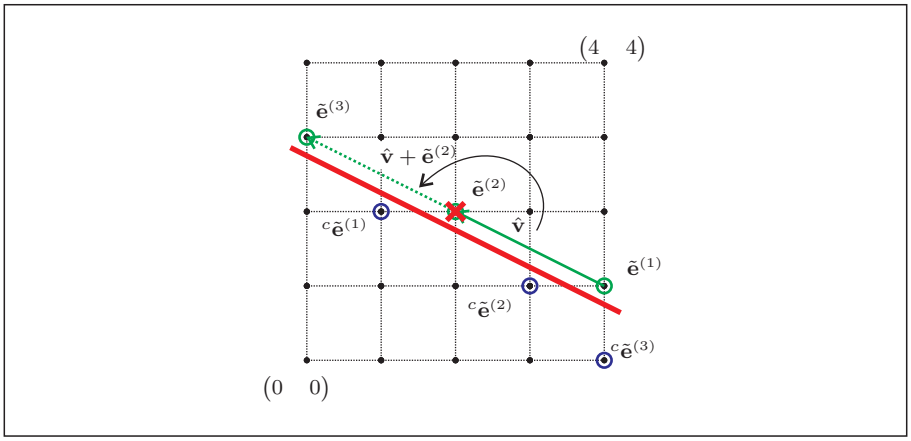


Figure 12.5
Élimination d'un vecteur générateur redondant avec les vecteurs supports.

nées ne soit pas une valeur admissible de la variable réduite correspondante. Il s'agit d'un cas neutre qui ne conduit à aucune élimination.

D'un point de vue complexité, cette procédure, décrite en pseudo-code dans l'algorithme 12.1, demande un nombre d'opérations conséquent, même si nous allons montrer qu'il s'agit d'une évolution polynomiale et non exponentielle. Former tous les vecteurs $\hat{\mathbf{v}}$, à partir des vecteurs générateurs deux à deux, représente, s'il y a k vecteurs générateurs, $\frac{(k)(k-1)}{2}$ possibilités. Chacune de ces possibilités doit ensuite être traitée par la proposition 12.3, ce qui engendre $3k$ tests. Cela conduit à une complexité de l'ordre de $\mathcal{O}(k^3)$, soit $\mathcal{O}(d^7)$ pour d variables réduites.

Autrement dit, pour seize variables réduites, il faut réaliser environ 400 million de tests sur sa famille génératrice[6] et vraisemblablement autant sur sa famille génératrice conjointe. Le degré de cette complexité polynomiale est donc tel qu'un algorithme dont le nombre d'opérations évoluerait exponentiellement ne demanderait que 65 536 opérations. Néanmoins, un tel algorithme n'existe pas encore à l'heure actuelle.

Exemple 12.1 — Synthèse géométrique d'une famille génératrice.
Soit la fonction logique à seuil $\mathcal{L}_1 \in \mathbb{L}_s^+$ définie par la famille génératrice suivante :

$$\{\mathcal{L}_1\} = \begin{Bmatrix} \hat{\mathbf{e}}^{(1)} = \begin{pmatrix} 2 & 0 & 1 & 0 \end{pmatrix}^\top \\ \hat{\mathbf{e}}^{(2)} = \begin{pmatrix} 2 & 0 & 0 & 1 \end{pmatrix}^\top \\ \hat{\mathbf{e}}^{(3)} = \begin{pmatrix} 1 & 2 & 0 & 0 \end{pmatrix}^\top \\ \hat{\mathbf{e}}^{(4)} = \begin{pmatrix} 1 & 1 & 1 & 1 \end{pmatrix}^\top \\ \hat{\mathbf{e}}^{(5)} = \begin{pmatrix} 0 & 2 & 1 & 1 \end{pmatrix}^\top \end{Bmatrix}.$$

La valeur maximale des quatre entrées réduites sont respectivement : $n_1 = 2$, $n_2 = 2$, $n_3 = 1$ et $n_4 = 1$. Comme $\mathcal{L}_1$ a quatre entrées réduites, il faut trouver quatre vecteurs générateurs supports dans sa famille génératrice. En réalité, il peut y en avoir moins si le complément se trouve dans la famille génératrice conjointe, mais nous avons

[6] En supposant qu'un test dure une milliseconde, cela conduit à un temps de calcul d'environ tcinq jours.

volontairement choisi une fonction logique à seuil pour laquelle ce n'est pas le cas. La famille génératrice conjointe est néanmoins calculée comme étant :

$$ {}^{c}\{\mathcal{L}_1\} = \begin{Bmatrix} {}^{c}\hat{\mathbf{e}}^{(1)} = \begin{pmatrix} 2 & 1 & 0 & 0 \end{pmatrix}^\top \\ {}^{c}\hat{\mathbf{e}}^{(2)} = \begin{pmatrix} 2 & 0 & 1 & 1 \end{pmatrix}^\top \\ {}^{c}\hat{\mathbf{e}}^{(3)} = \begin{pmatrix} 1 & 2 & 0 & 1 \end{pmatrix}^\top \end{Bmatrix}. $$

L'objet est donc de détecter les vecteurs générateurs de $\{\mathcal{L}_1\}$ qui ne sont pas des vecteurs générateurs supports. En appliquant la proposition 12.3 à tous les vecteurs $\hat{\mathbf{v}}^{(i,j)} = \hat{\mathbf{e}}^{(j)} - \hat{\mathbf{e}}^{(i)}$, il apparait que seul le vecteur générateur $\hat{\mathbf{e}}^{(5)}$ est éliminé.
Comme il ne reste que quatre vecteurs générateurs, ceux-ci sont nécessairement les quatre vecteurs générateurs supports recherchés. On peut alors en dériver trois vecteurs libres, par exemple :

$$ \begin{aligned} & \hat{v}^{(1)} = \begin{pmatrix} 0 & 0 & 1 & -1 \end{pmatrix}^\top, \\ & \hat{v}^{(2)} = \begin{pmatrix} 1 & -2 & 1 & 0 \end{pmatrix}^\top \\ \text{et} \quad & \hat{v}^{(3)} = \begin{pmatrix} 1 & -1 & 0 & -1 \end{pmatrix}^\top. \end{aligned} $$

L'utilisation de l'équation (12.7) permet de calculer les quatre poids synaptiques de $\hat{\mathbf{w}}$:

$$ \begin{aligned} \hat{w}_1 &= -\begin{vmatrix} 0 & -2 & -1 \\ 1 & 1 & 0 \\ -1 & 0 & -1 \end{vmatrix} = 3, \\ \hat{w}_2 &= \begin{vmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \\ -1 & 0 & -1 \end{vmatrix} = 2, \\ \hat{w}_3 &= -\begin{vmatrix} 0 & 1 & 1 \\ 0 & -2 & -1 \\ -1 & 0 & -1 \end{vmatrix} = 1, \\ \text{et} \quad \hat{w}_4 &= \begin{vmatrix} 0 & 1 & 1 \\ 0 & -2 & -1 \\ 1 & 1 & 0 \end{vmatrix} = 1. \end{aligned} $$

Le poids synaptique auxiliaire est ensuite calculé grâce à l'équation (12.9) avec un des quatre vecteurs générateurs supports détectés dans $\{\mathcal{L}_1\}$ et un vecteur générateur support de ${}^{c}\{\mathcal{L}_1\}$, par exemple ${}^{c}\hat{\mathbf{e}}^{(3)}$:

$$ w_0 = -\hat{\mathbf{w}}^\top \frac{\hat{\mathbf{e}}^{(1)} + {}^{c}\hat{\mathbf{e}}^{(3)}}{2} = -7{,}5. $$

Le neurone binaire obtenu est donc caractérisé par les poids synaptiques suivants :

$$ \hat{\mathbf{w}} = \begin{pmatrix} 3 & 2 & 1 & 1 \end{pmatrix}^\top \text{ et } w_0 = -7{,}5. $$

Dans l'exemple précédent, la famille génératrice n'était pas exprimée dans un espace complètement réduit. L'algorithme de réduction, à la page 91 du chapitre 8, montre en effet que les deux variables les moins influentes sont symétriques l'une de l'autre, ce qui est confirmé par la synthèse géométrique qui leur a affecté le même poids synaptique. Cela nous permet de souligner que travailler dans un espace complètement réduit n'est pas une condition nécessaire au bon fonctionnement de la synthèse

géométrique, mais qu'il est toutefois préférable de réduire au maximum les familles génératrices afin de diminuer leur nombre de vecteurs générateurs, et donc les temps de calculs.

12.4 Bilan et optimisation de la synthèse géométrique

Étant donné la complexité algorithmique élevée de la méthode de synthèse géométrique présentée dans ce chapitre, ainsi que sa formulation définitive, peu d'expérimentations ont pu être menées pour des nombres de variables élevés. Nous avons néanmoins pu constater que cette méthode parvenait à synthétiser toutes les familles génératrices pour lesquelles la synthèse spectrale exacte ne donnait rien.

Afin de pouvoir mener des tests un peu plus poussés et donc appuyer la véracité de la proposition 12.3, plusieurs pistes conduisant à diminuer la complexité de cette méthode peuvent être explorées. Tout d'abord, l'optimisation la plus évidente consiste à éliminer les vecteurs générateurs au fur et à mesure que les tests les disqualifient, soulageant ainsi la suite de l'algorithme. Il est de plus possible d'anticiper le résultat de certains tests, pour lesquels les vecteurs obtenus sortiraient de façon triviale de l'hypercube.

Enfin, une idée bien plus prometteuse est que les vecteurs non supports puissent n'occuper que des positions particulières dans les familles génératrices. Si cela s'avérait exact, comme nos quelques essais semblent le suggérer, de nombreux vecteurs générateurs et générateurs conjoints pourraient être identifiés d'office comme vecteurs supports.

En tout état de cause, puisque le nombre de vecteurs générateurs augmente comme d^4, alors que le nombre de vecteurs supports augmente seulement comme d, les préceptes éliminant les vecteurs non supports doivent éliminer un nombre de vecteurs croissant comme d^3. Dans le cas contraire, trop de vecteurs supports seraient détectés, faisant échouer la synthèse géométrique. Il s'agit là de la principale piste pour cerner les éventuelles limitations de la synthèse géométrique.

13

Vers un environnement de synthèse assistée par ordinateur

Au fil des chapitres de ce manuscrit, un formalisme complet a été développé autour des fonctions logiques à seuil et des neurones binaires. Il s'agit d'une représentation abstraite, appelée famille génératrice, au travers de laquelle la fonction d'un neurone binaire peut être exprimée, manipulée, puis resynthétisée avec une marge de fonctionnement plus robuste. Ainsi éparpillés dans ce documents, l'unité formée par ces procédures ne transparaît pas de façon évidente. Ils forment pourtant une première base d'outils algorithmiques, que nous avons pu réunir au sein d'une petite application nommée CNNstudio, développée en C++ sous licence GPL pour l'environnement GNU/Linux. La présence de l'acronyme des réseaux de neurones cellulaires dans le nom de ce logiciel est l'héritage des motivations originelles des travaux de recherches exposés dans ce manuscrit.

En réalité, nos résultats s'étendent sur de nombreux modèles de réseaux de neurones artificiels, pourvu qu'ils puissent se décomposer en réseaux de neurones binaires. C'est ainsi que nous avons pu utiliser CNNstudio pour optimiser les exemples de réseaux de

neurones de la partie I. Nous avonc pour cela eu recours aux différentes techniques décrites dans ce manuscrit, et que nous avons regroupé sous trois catégories :

analyse l'algorithme du chapitre 9 permet d'exprimer n'importe quel neurone binaire sous la forme d'une famille génératrice ;

conditionnement les chapitres 8 et 11 apportent des outils permettant de mettre en forme des familles génératrices et d'en extraire des informations d'intérêt :
- la réduction,
- la complémentation,
- la détection de repliement spectral,
- l'approximation ;

synthèse la troisième partie présente plusieurs méthodes originales permettant de synthétiser une famille génératrice :
- les méthodes de synthèse de la littérature adaptées aux familles génératrices,
- la synthèse en un réseau de neurones binaires symétriques,
- la synthèse spectrale exacte et, avec elle, la synthèse de spectres non repliés,
- la synthèse géométrique.

La partie analyse de cette liste se revèle étonnamment pauvre. La seule façon d'obtenir une famille génératrice à laquelle appliquer ces techniques est en effet d'analyser la fonction d'un neurone binaire déjà programmé ; que ce soit par apprentissage ou par une technique de synthèse non robuste. Nous justifions cette lacune par le caractère prioritaire de l'étude des propriétés des familles génératrices et surtout de leur aptitude à être synthétisées en neurones binaires.

Pouvoir fabriquer des familles génératrices indépendament des neurones binaires, c'est-à-dire à partir d'une description algébrique ou arithmétique de la fonction logique à réaliser, est néanmoins une question centrale et incontournable. Elle pose plusieurs problèmes que ce chapitre tente de résoudre : comment traduire un tableau de vérité en une famille génératrice, comment traiter une fonction logique qui ne relèverait pas de la logique à seuil. Cette dernière question nous conduira à proposer une méthode de synthèse d'un réseau de neurones binaires, et non plus de neurones binaires isolés.

13.1 Du tableau de vérité aux familles génératrices

Rendre l'existence des familles génératrices indépendante des autres méthodes de programmation de neurones binaires peut se révéler hautement stratégique. En effet, les méthodes de synthèse que nous avons développées prendraient alors le statut de méthodes de programmation de neurones binaires à part entière et ne seraient ainsi plus tributaires des techniques concurrentes. Cette évolution se découpe en deux étapes : la traduction du tableau de vérité d'une fonction logique à seuil en famille génératrice, puis celle d'une fonction logique quelconque.

13.1.1 Cas des fonctions logiques à seuil

Savoir qu'un tableau de vérité représente une fonction logique à seuil est une hypothèse très confortable dans la mesure où elle allège la procédure de transcription en familles génératrices. Celle-ci n'a en particulier pas à vérifier que la fonction logique ainsi décrite peut effectivement être représentée par une famille génératrice ; avec toutes les contraintes fonctionnelles que cela impose, comme la validité du principe de génération.

En outre, les descripteurs de Chow, définis dans le chapitre 10 à la page 125, sont alors des outils formidables pour réaliser cette opération [Chow, 1961]. Obtenus par la simple observation des mintermes d'une fonction logique à seuil, ces derniers permettent d'identifier les entrées dont le poids synaptique est négatif, les entrées symétriques, ainsi que les influences relatives de ces dernières. L'espace complètement réduit d'une fonction logique à seuil peut ainsi être déterminé dès le départ et les variables devant être complémentées pour avoir un poids synaptique positif le sont également.

Une fois ce travail effectué, la suite de la procédure se développe de façon naturelle, à partir des ordonnancements sur lesquels reposent les familles génératrices. En l'occurence, il faut tout d'abord ré-arranger l'ordre des variables réduites obtenues selon leurs influences croissantes sur la valeur de la fonction logique. Il faut ensuite éliminer les combinaisons d'entrées ne correspondant pas à des mintermes. Enfin, les lignes restantes doivent être ordonnées dans l'ordre lexicographique décroissant.

La représentation obtenue est alors très proche de celle d'une famille génératrice. Il ne reste qu'à éliminer les lignes, autrement dit les vecteurs réduits, qui s'avèrent redondantes vis-à-vis du processus de génération ; une opération facilitée par l'ordre lexicographique, comme le montre l'exemple suivant.

Exemple 13.1 — Famille génératrice d'une fonction logique à seuil.
Soit la fonction logique à seuil $\mathcal{L}_1 \in \mathbb{L}_s$ définie par :

$$\mathcal{L}_1(e_1, e_2, e_3, e_4) \equiv e_1 + (e_2 \overset{2}{\top} e_3 \overset{2}{\top} \overline{e}_4).$$

Le tableau de vérité de $\mathcal{L}_1$ est le suivant :

e_1	e_2	e_3	e_4	$\mathcal{L}_1$
faux	faux	faux	faux	faux
faux	faux	faux	vrai	faux
faux	faux	vrai	faux	vrai
faux	faux	vrai	vrai	faux
faux	vrai	faux	faux	vrai
faux	vrai	faux	vrai	faux
faux	vrai	vrai	faux	vrai
faux	vrai	vrai	vrai	vrai

vrai	faux	faux	faux	vrai
vrai	faux	faux	vrai	vrai
vrai	faux	vrai	faux	vrai
vrai	faux	vrai	vrai	vrai
vrai	vrai	faux	faux	vrai
vrai	vrai	faux	vrai	vrai
vrai	vrai	vrai	faux	vrai
vrai	vrai	vrai	vrai	vrai

En ne conservant que les lignes pour lesquelles $\mathcal{L}_1$ vaut vrai, les cinq descripteurs de

CHOW valent respectivement, d'après leur définition à la page 126 :

$$\mathbf{a}(\mathcal{L}_1) = \begin{pmatrix} 8 & 7 & 7 & 5 \end{pmatrix}^\top$$
$$\text{et} \quad m(\mathcal{L}_1) = 12.$$

Comme $a_4 < \frac{m}{2}$, son poids synaptique devra être négatif. Cette variable est donc complémentée dans l'expression de $\mathcal{L}_1$ et son nouveau descripteur de CHOW vaut $m - a_4 = 7$. Les variables e_2, e_3 et $\overline{e}_4$ sont donc symétriques, conduisant à la définition d'un espace réduit de deux dimensions indicées de la façon suivante : $\hat{e}_1 \rightarrow \{e_1\}$ et $\hat{e}_2 \rightarrow \{e_2\,;e_3\,;\overline{e}_4\}$.
La fonction spectre peut ainsi être dérivée de ce tableau de vérité. En y supprimant les vecteurs réduits pour lesquelles $\mathcal{L}_1$ vaut faux, celui-ci s'écrit :

$\hat{e}_1$	$\hat{e}_2$	$\mathcal{S}_{\mathcal{L}_1}$
1	3	vrai
1	2	vrai
1	1	vrai
1	0	vrai
0	3	vrai
0	2	vrai

Les vecteurs générateurs s'en déduisent par simple observation :

$$\{\mathcal{L}_1\} = \left\{ \begin{matrix} 1 & 0 \\ 0 & 2 \end{matrix} \right\}.$$

13.1.2 Cas des fonctions logiques quelconques

L'obtention de la famille génératrice relative à une fonction logique quelconque pose des problèmes beaucoup plus fondamentaux. En effet, le formalisme dont les familles génératrices sont le résultat, a été mis au point spécifiquement pour les fonctions logiques à seuil, en tenant compte de leurs propriétés analytiques. Espérer pouvoir exprimer une fonction logique qui n'ait pas ces propriétés est donc un non sens théorique.

Toutefois, pouvoir être exprimée sous la forme d'une famille génératrice n'est pas une condition suffisante pour être une fonction logique à seuil. Cela signifie qu'une famille génératrice représente en réalité une classe de fonctions logiques bien plus large ; celle des fonctions logiques complètement monotones qui est connue pour être équivalente aux fonctions logiques à seuil lorsque ces dernières dépendent de moins de huit variables [PAULL et MC CLUSKEY, 1960]. Pour toutes ces fonctions logiques, la représentation en familles génératrices est donc parfaitement adaptée, et s'obtient de la même façon que pour les fonctions logiques à seuil, au travers des descripteurs de CHOW.

En ce qui concerne les autres fonctions logiques, une formulation précise, compatible avec nos techniques de synthèse, reste encore à déterminer. Elle devra vraisemblablement être basée sur l'étude de la décomposition d'une fonction logique en

fonctions logiques complètement monotones, c'est-à-dire en plusieurs familles génératrices. Formellement, il s'agira donc d'une description des différentes façons de séparer des groupes de mintermes de ces fonctions logiques. Cette décomposition ne doit cependant pas revêtir de caractère définitif. Elle doit pour cela rester purement fonctionelle, à l'instar de ce que nous avons pu faire en introduisant les familles génératrices, qui se contentent de lister les contraintes à implémenter sans pour autant restreindre l'espace des poids synaptiques solutions.

La section suivante illustre les avantages qu'auraient une telle formulation, par le biais d'une méthode de synthèse de réseaux de neurones binaires réalisant des fonctions logiques complètement monotones.

13.2 Synthèse de familles génératrices par couvertures

La capacité d'englober la synthèse de fonctions logiques quelconques représente un enjeu important pour l'essor des familles génératrices. Dans ce contexte, savoir synthétiser une fonction logique à seuil en un neurone binaire, ou en une porte logique TL, est une étape clef. Il manque néanmoins une procédure de synthèse plus globale, capable de « décider » quels neurones binaires doivent être synthétisés vis-à-vis d'une fonction logique donnée et de contraintes diverses sur le résultat. L'implémentation d'un tel procédé, autour des méthodes développées dans ce manuscrit, serait le fondement solide d'environnement de conception de réseaux de neurones binaires assistée par ordinateur.

Du point de vue des contraintes extérieures dont un tel environnement doit pouvoir tenir compte, l'ensemble des techniques auxquelles ont abouties nos recherches apportent déjà de nombreuses réponses. En effet, le chapitre 8 a par exemple montré comment la structure récursive des familles génératrices pouvait donner lieu à un réseau de neurones binaires, par le biais de son expression algébrique en portes n-somme. Formellement, cette méthode revient à implémenter séparément chacune des sous-familles d'une famille génératrice, de sorte que les neurones binaires obtenus ne réalisent que des fonctions n-somme, dont les poids synaptiques sont calculés par la synthèse spectrale pour spectres non repliés.

Cette réflexion a conduit à la méthode de synthèse par couvertures, présentée dans cette section. Elle a pour objectif de s'intéresser aux différents choix de synthèse qu'il est possible de faire pour une famille génératrice : dans quelle mesure est-il intéressant de synthétiser séparémment toutes ses sous-familles génératrices, faut-il plutôt en synthétiser certaines spécifiquement, etc.

13.2.1 Principes de la synthèse par couvertures

La synthèse par couverture est inspirée de la synthèse par tableaux de Celinski présentée à la page 135 du chapitre 10, elle-même dérivée de la synthèse de fonctions

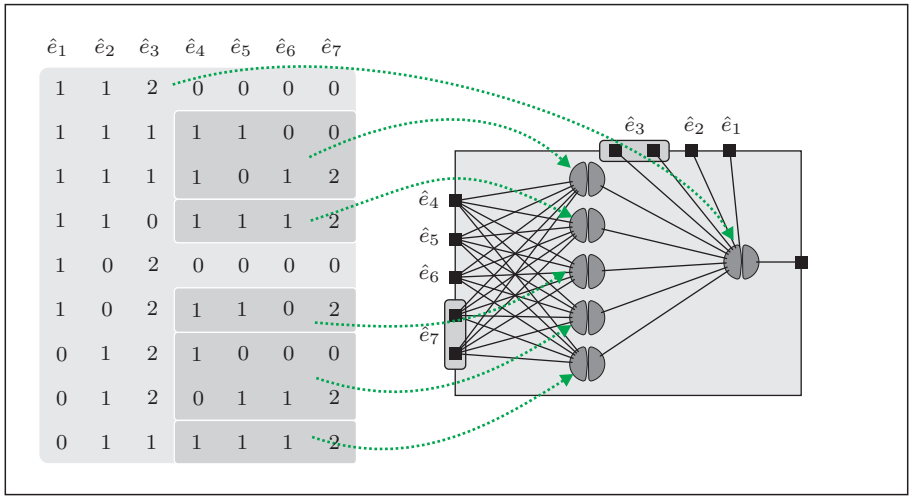


Figure 13.1
Principe de la synthèse par couvertures.

logiques par tables de KARNAUGH. En synthétisant toutes les sous-familles d'une famille génératrice, ensemble ou séparément, autant de neurones binaires sont obtenus et mis en réseau selon la façon dont les sous-familles correspondantes s'imbriquent les unes dans les autres. La figure 13.1 illustre ce principe en synthétisant à part toutes les sous-familles d'odre 3, de sorte que les neurones binaires obtenus servent d'entrées au neurone binaire obtenu par synthèse de la famille génératrice entière.

Dans l'exemple traité par la figure 13.1 et en appelant $\mathcal{L}_1$ la fonction logique à seuil correspondant à la première sous-famille génératrice d'ordre 3, $\mathcal{L}_2$ la suivante et ainsi de suite, la famille génératrice globale s'écrit :

$$\left\{\begin{array}{cccc} 1 & 1 & 2 & 0 \\ 1 & 1 & 1 & \mathcal{L}_1 \\ 1 & 1 & 0 & \mathcal{L}_2 \\ 1 & 0 & 2 & 0 \\ 1 & 0 & 2 & \mathcal{L}_3 \\ 0 & 1 & 2 & \mathcal{L}_4 \\ 0 & 1 & 1 & \mathcal{L}_5 \end{array}\right\}. \tag{13.1}$$

En remarquant que les propriétés des familles génératrices impliquent $\mathcal{L}_5 < \mathcal{L}_4 < \cdots < \mathcal{L}_1$ (voir le chapitre 8), alors ces fonctions peuvent-être remplacées par une variable réduite équivalente : $\hat{e} \in [\![0\,;5]\!]$. Sa valeur vaut :

- $\hat{e} = 0$ lorsqu'aucune des fonctions $\mathcal{L}_1, \ldots, \mathcal{L}_5$ n'est vraie ;
- $\hat{e} = 1$ lorsque $\mathcal{L}_1$ est vraie ;
- $\hat{e} = 2$ lorsque $\mathcal{L}_2$ est vraie, ce qui implique que $\mathcal{L}_1$ est vrai puisque $\mathcal{L}2 < \mathcal{L}_1$;
- ...
- $\hat{e} = 5$ lorsque $\mathcal{L}_5$ est vraie, impliquant que les quatre autres fonctions logiques sont vraies.

La famille génératrice se met alors sous la forme suivante :

$$\left\{\begin{matrix} 1 & 1 & 2 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 2 \\ 1 & 0 & 2 & 0 \\ 1 & 0 & 2 & 3 \\ 0 & 1 & 2 & 4 \\ 0 & 1 & 1 & 5 \end{matrix}\right\}, \tag{13.2}$$

et peut ainsi faire l'objet d'une synthèse par une des méthodes de ce manuscrit.

D'une manière générale, la méthode de synthèse préconisée pour une sous-famille génératrice dépend de sa morphologie. Si son spectre n'est pas replié, alors le calcul du neurone binaire correspondant est du ressort de la synthèse spectrale pour spectres non repliés, sinon, il faudra recourir à la synthèse spectrale exacte, voire à la synthèse géométrique pour les morphologies les plus compliquées. Après synthèse, la façon dont les neurones sont connectés les uns aux autres dépends de la façon dont s'imbriquent les familles génératrices correspondantes. Comme pour la synthèse par tableaux de Celinksi, la synthèse par couvertures s'arrête dès que toutes les composantes de la famille génératrice sont couvertes par la synthèse d'au moins une sous-famille génératrice à laquelle elle appartient.

Les différentes manières de choisir quelles sous-familles génératrices synthétiser dépend des propriétés désirées pour le réseau de neurones binaires obtenu. Elles sont résumées par plusieurs principes, décrits ci-après.

Dimension maximale d'un regroupement Afin de pouvoir utiliser les techniques de synthèse développées dans ce manuscrit, il est impératif de s'assurer que les sous-familles génératrices choisies correspondent bien à des fonctions logiques à seuil. C'est la raison pour laquelle nous exploiterons l'équivalence, pour moins de huit variables, entre fonctions logiques à seuil et familles génératrices en limitant à huit variables la dimension des sous-familles choisies.

Couverture maximale L'objectif est ici de couvrir l'ensemble d'une famille génératrice par la synthèse de toutes ses sous-familles génératrices. Le réseau obtenu a alors de nombreux neurones binaires reliés les uns aux autres par un jeu de connexions souvent compliqué. En revanche, la dynamique de calcul du réseau obtenu est réduite au maximum. En effet, chaque sous-famille génératrice a alors une morphologie très simple, de spectre souvent non replié, et pour laquelle la synthèse donnera une marge de fonctionnement élevée. Dès lors, la tolérance au bruit des poids synaptiques sera également élevée, et leur valeur pourra ainsi supporter une plus grande imprécision et, par conséquent, une dynamique de calcul plus faible.

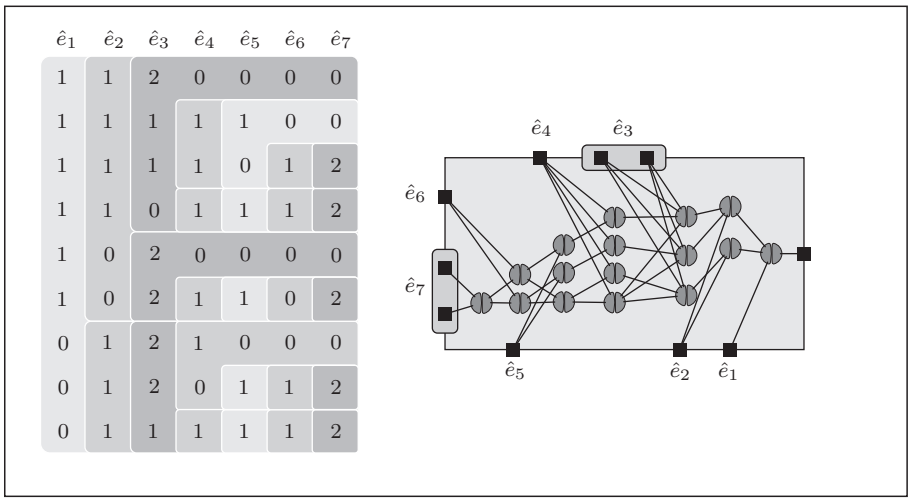


Figure 13.2
Synthèse d'un réseau de neurones binaires par couverture maximale.

Ce type de synthèse peut par exemple être souhaitable pour des réseaux de neurones binaires embarqués sur des architectures de type FPGA, faisant ainsi l'économie de nombreuses structures de calcul au profit d'unités logiques très simples.

La figure 13.2 effectue la synthèse par couverture maximale d'une famille génératrice. Le découpage qui y est réalisé est en fait exactement celui obtenu en exprimant la famille génératrice avec des portes n-somme.

Couverture minimale Au contraire de la couverture maximale, la stratégie ici proposée consiste à synthétiser le moins de sous-familles possibles pour couvrir entièrement une famille génératrice. D'une manière générale, cela conduit naturellement à des réseaux très légers en nombre de neurones binaires, et donc en nombre de connexions. Toutefois, les neurones binaires vont globalement réaliser des fonctions logiques dont la morphologie est plus compliquée, nécessitant l'utilisation de la synthèse spectrale exacte voire de la synthèse géométrique.

Les marges de fonctionnement ainsi obtenues seront globalement plus faibles, impliquant du même coût une dynamique de calcul élevée. C'est pourquoi ce type de réalisation est très certainement préférable pour les réseaux de neurones binaires destinés à être exécutés sur des plateformes informatiques dont les ressources peuvent être considérées comme illimitées. Par ailleurs, ce type de plateforme sera également plus rapide à exécuter les réseaux obtenus de cette façon puisque la sortie de chaque neurone est calculée sériellement et non parallèlement.

La figure 13.3 montre un réseau de neurones binaires obtenu par cette méthode.

Couvertures redondantes Cette dernière possibilité permet d'introduire de la redondance dans un réseau de neurones binaires en synthétisant plusieurs fois les

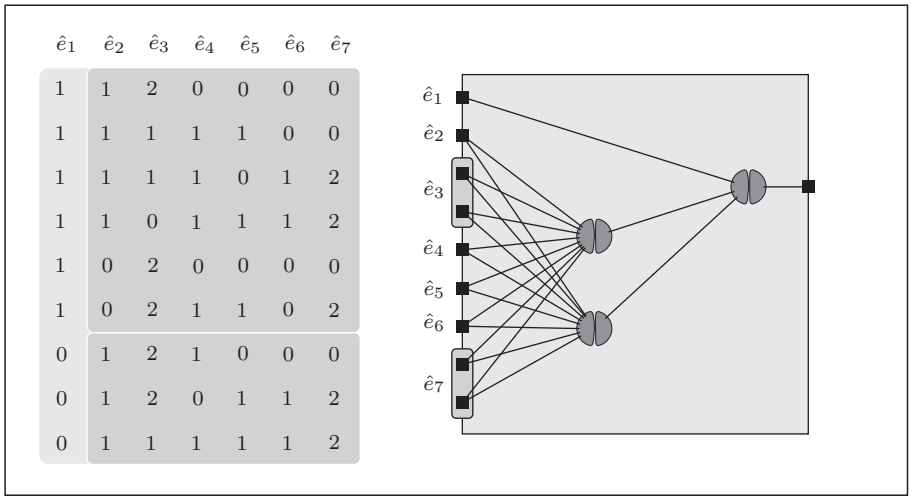


Figure 13.3
Synthèse d'un réseau de neurones binaires par couverture minimale.

mêmes composantes d'une famille génératrice, au travers de plusieurs de ses sous-familles génératrices. Cela peut avoir l'avantage de rendre le réseau robuste à la défaillance d'un de ses constituants.

13.2.2 La synthèse assistée par ordinateur basée sur la synthèse par couvertures

D'une manière générale, la méthode de synthèse par couvertures conduit à des réseaux de neurones binaires dont la structure est bien mieux équilibrée que ce que les solutions concurrentes ont à offrir en matière de synthèse de réseaux de neurones [Chen et al., 2006 ; Celinski et al., 2001 ; Nemes et al., 1998 ; Crounse et al., 1997 ; Roska et Chua, 1993]. En effet, notre méthode est basée sur des découpages cohérents avec la logique à seuil, symbolisés par la décomposition en sous-familles génératrices, tandis que les alternatives de la littérature ne réalisent que les décompositions classiques du théorème de Shannon. En conséquence, leurs décomposition conduit systématiquement à deux couches de neurones binaires : une première couche réalisant les mintermes de la fonction logique à synthétiser, et un neurone binaire de sortie réunissant les sorties de la couche inférieure par une porte ou. En comparaison, notre méthode offre la possibilité séduisante de pouvoir sortir de la structure multicouche classique des réseaux de neurones artificiels, en autorisant certaines entrées à attaquer directement les couches supérieures.

Bien qu'une formulation générale des fonctions logiques sous la forme de familles génératrices soit encore à établir, ce chapitre a montré comment une telle représentation pourraient être utilisée pour synthétiser des fonctions logiques quelconques en un réseau de neurones binaires ou en un circuit en logique tl. En fonction des propriétés recherchées, plusieurs façons de choisir les sous-familles génératrices à synthétiser ont

été proposées, de sorte que l'ensemble peut former l'ossature d'un véritable environnement de conception de circuits logiques assistée par ordinateur.

Il ne s'agit toutefois que d'une étude préliminaire. Il est ainsi probable que la politique la plus pertinente pour choisir les sous-familles à synthétiser soit située à mi-chemin entre la couverture maximale et minimale. La synthèse proposée dans la figure 13.1 en est un exemple.

14

Conclusion générale

En 1965, G. Moore, le co-fondateur aujourd'hui retraité d'Intel®, énonça la fameuse loi empirique selon laquelle le nombre de transistors d'un microprocesseur devait doubler tous les deux ans. Quarante ans plus tard il prédit que d'ici 2020, les technologies à base de transistors buteront sur des limitations physiques incontournables, mettant un terme à près de soixante ans de progrès [Richards, 2007]. Et si le temps était venu pour les systèmes neuromimétiques, au travers de la logique à seuil, de prendre le relai ?

Pour qu'ils puissent constituer une alternative crédible, de puissantes techniques d'optimisation manquent encore aux réseaux de neurones artificiels, qui ne bénéficient ainsi que de très peu d'environnements de conception assistée par ordinateur. Dans ces conditions, leur intégration matérielle à grande échelle est aujourd'hui très irréaliste et seule une approche analytique approfondie du problème pourra y remédier. Le succès d'une telle étude, étant donnée la complexité considérable des problèmes mis en jeu, dépendra de la capacité de la communauté scientifique à imaginer de nouveaux outils mathématiques.

Ce manuscrit propose un formalisme analytique original dédié aux neurones binaires : les familles génératrices. Nées de la recherche de la meilleure adéquation entre outil mathématique et problématique, nous montrons que leur synthèse en neurones

binaires s'effectue de manière naturelle, en contournant beaucoup des difficultés habituellement rencontrées. Ces travaux ont ainsi mené à plusieurs algorithmes de synthèse automatique de neurones binaires dont les performances surpassent les solutions concurrentes. Ces résultats sont le fruit d'une réflexion qui s'est échelonnée sur trois phases.

Comme point de départ de notre étude, un bilan pragmatique des principaux modèles de réseaux de neurones artificiels a permis d'identifier le neurone binaire comme leur élément central. Son implémentation matérielle sous la forme d'un neurone cellulaire a alors révélé la relative faiblesse, en terme de fiabilité, des méthodes de conception de neurones binaires par apprentissage. Ce constat a motivé l'exploration de nouvelles pistes, en particulier la voie analytique.

Les techniques auxquelles nous avons abouties sont ainsi le fruit d'une analyse fonctionnelle approfondie des neurones binaires. Par l'étude des propriétés analytiques des fonctions logiques qu'ils mettent en œuvre, nous sommes arrivés à la conclusion que la traditionnelle représentation par tableau de vérité leur était fondamentalement inadaptée et excessivement redondante. Ces considérations ont conduit à la formulation d'un espace réduit, dans lequel seuls quelques vecteurs sont retenus pour former la famille génératrice équivalente d'un neurone binaire. La forme inhabituelle de cet outil rendant sa manipulation déroutante, l'établissement de plusieurs règles d'utilisation s'est révélé nécessaire. Cela a permis de montrer que, contrairement aux tableaux de vérité, le formalisme des familles génératrices conservait plusieurs informations que leur synthèse en neurones binaires aurait autrement dû recalculer. Un algorithme d'analyse a enfin été proposé pour retrouver automatiquement la famille génératrice d'un neurone binaire quelconque.

Grâce au développement de ce formalisme, de nombreuses techniques existantes pour la conception de neurones binaires ont pu être accélérées. L'étude des progrès ainsi apportés a par la suite permis d'énoncer un premier mécanisme par lequel la synthèse de certaines classes morphologiques de familles génératrices en neurones binaires est possible : la synthèse spectrale. Basée sur une procédure récursive, visant à positionner chacun des vecteurs d'une famille génératrice par rapport à un seuil, cet ensemble de méthodes se démarque des autres par des performances nettement supérieures. Une identification plus précises de ces classes permettra de spécifier les conditions d'application de ces méthodes. Une méthode de synthèse plus générale, basée sur l'identification de vecteurs supports, a ensuite été proposée : la synthèse géométrique. Algorithmiquement plus compliquée que la synthèse spectrale, elle tire parti des propriétés géométriques des familles génératrices pour étendre son domaine de validité à virtuellement toutes les fonctions logiques à seuil. Si son bon fonctionnement, tel que nous avons pu l'observer sur de nombreux exemples, peut être démontré, il s'agira de la première méthode capable de synthétiser analytiquement l'intégralité de la classe des fonctions logiques à seuil.

Toutes ces techniques d'analyse, de manipulation et de synthèse ont été implémentées au sein d'une application dont l'ambition est de devenir le premier outil informatique

de synthèse analytique de neurones binaires. À cette fin, les propriétés d'une formulation générale des fonctions logiques, autour de laquelle nos techniques de synthèse pourraient s'articuler, ont été établies. À titre d'exemple, la synthèse par couvertures, destinée aux fonctions logiques complètement monotones, a été présentée. En fonction de la façon dont sont synthétisés les nombreux éléments d'information rassemblées par les familles génératrices correspondantes, les réseaux de neurones calculés peuvent être optimisés selon l'un ou l'autre de deux critères antagonistes : le nombre de neurones et la dynamique de calcul. En outre, les circuits neuronaux conçus de cette manière présentent une charge de calcul répartie de façon bien plus équilibrée que ceux résultant des techniques de synthèse concurrentes.

L'apport formel des familles génératrices a été un élément décisif dans l'élaboration des méthodes de synthèse de ce manuscrit. Grâce à lui, les principales difficultés rencontrées par les approches traditionnelles ont en effet pu être contournées. De cette façon, les algorithmes auxquels nous avons abouti sont capables, pour des complexités comparables voire moindre, de synthétiser des neurones binaires optimisés et dont le nombre d'entrées est théoriquement illimité. Ces algorithmes n'ayant pas fait l'objet d'implémentations poussées, on peut facilement imaginer que leur optimisation parviendra à des solutions encore plus rapides qui tireront alors pleinement parti de ce nouveau formalisme.

D'un point de vue théorique, l'étude des mécanismes de fonctionnement des neurones binaires par le biais de leurs familles génératrices n'en est qu'à ses débuts. Maintenant que leur intérêt pour la synthèse de neurones binaire a été établie, il est en effet temps d'initier des recherches encore plus pointues afin d'en identifier les limites. Par exemple, étudier comment les familles génératrices peuvent rendre compte des mécanismes d'apprentissage propres aux réseaux de neurones artificiels pourrait ouvrir la voie à des circuits neuronaux capables de s'adapter à leur environnement de fonctionnement.

Avant cela, il nous semble stratégique de nous intéresser à la définition d'un formalisme, basé sur les familles génératrices, capable de décrire efficacement une fonction logique quelconque. En lui appliquant la synthèse par couvertures, nous disposerons ainsi de l'un des tous premiers environnements de conception assistée par ordinateur ; un outil indispensable pour tester les résultats de nos travaux à l'échelle de circuits complets. À densité d'intégration égale avec les microprocesseurs actuels, de tels circuits afficheront, normalement, des performances très supérieures, héritées de la richesse fonctionnelle des neurones binaires.

Table des matières

Table des figures

Liste des algorithmes

Bibliographie

Analogic & Neural Computing Laboratory,
Cnn Software Library,
Mta SzTaki, 2005.

Y. S. Abu-Mostafa et J.-M. St Jacques,
« Information Capacity of the Hopfield Model »,
Ieee *Transactions on Information Theory*, vol. 31, p. 461–464, 1985.

L. Adouane et N. Lefort-Piat,
« Methodology for parameters optimization of an hybrid architecture of control. »,

Dans *Proceedings of the World Congress of the International Federation of Automatic Control*, 2005, Prague, République Tchèque.

K. Aihara et G. Matsumoto,
Chaos in Biological Systems,
New York : Plenum, 1987.

K. Aihara, T. Takabe et M. Toyoda,
« Chaotic Neural Networks »,
Physics Letters A, vol. 144, p. 333–340, 1990.

K. Alligood, T. Sauer et J. A. Yorke,
Chaos: An Introduction to Dynamical Systems,
Springer-Verlag, 1997.

J. A. Anderson et E. Rosenfeld (coordinateurs),
Neurocomputing: Foundations of Research,
Cambridge, ma : mit Press, 1988.

T. Arai et Y. Osana,
« Hetero Chaotic Associative Memory for Successive Learning with Give Up Function: One-to-many Associations »,
Dans *Proceedings of* iasted *Artificial Intelligence and Applications, Innsbrück, Autriche*, 2006.

S. Arik et V. Tavsanoglu,
« Equilibrium Analysis of Nonsymmetric cnns »,
International Journal of Circuit Theory and Applications, vol. 24, p. 269–274, 1996.

A. Babloyantz et Loureno,
« Computation with chaos: A paradigm for cortical activity »,
Proceedings of National Academy of Sciences, vol. 91, p. 9027–9031, 1994.

V. Beiu, J. M. Quintana et M. J. Avedillo,
« Vlsi Implementations of Threshold Logic — a Comprehensive Survey »,
Ieee *Transactions on Neural Networks*, vol. 14, p. 1217–1243, 2003.

Y. Bénédic,
Realization of the First cnn *Image Processing Template Design and Analysis Software*,
Thèse de maîtrise, École Nationale Supérieure de Physique de Strabourg, 2002.

Y. Bénédic et J. Mercklé,
« Cellular Neural Networks: a Unified Analysis of the Stability Issue »,
Dans *Proceedings of* iasted *Artificial Intelligence and Applications, Innsbrück, Autriche*, 2006a.

Y. Bénédic et J. Mercklé,
« Optimization of Binary-Output Cnns: First Step of an Analytical Design Process »,
Dans *Proceedings of* ieee *World Congress on Computational Intelligence, Vancouver, Canada*, 2006b.

Y. Bénédic et D. Monnin,
« Acquisition et traitement d'images au moyen d'une rétine artificielle fondée sur les réseaux de neurones cellulaires »,
Rapport technique N 802/2003, Institut franco-allemand de recherches de Saint-Louis, 2003.

Y. Bénédic, D. Monnin et J. Mercklé,
« Fast Analysis Method for Both Coupled and Uncoupled Binary-Output Cloning Templates »,
Dans *Proceedings of the* ieee *International Workshop on Cellular Neural Networks and their Applications, Budapest, Hongrie*, 2004.

Y. Bénédic, P. Wira et J. Mercklé,
« A New Method for re-Implementation of Threshold Logic Functions with Cellular Neural Networks »,
International Journal of Neural Systems, soumis.

Y. Bennani,
Traité ic2 *: Information - Commande - Communication*, chap. Apprentissage connexionniste, p. p. 360,
B. Dubuisson, 2006.

P. Celinski, G. D. Sherman et D. Abbott,
« A Technique for Implementing Arbitrary Boolean Functionsin Threshold Logic »,

Dans *Proceedings of* spie*, San Diego*, vol. 4236, 2001.

F. Chen, G. He et G. Chen,
A Complete List of Genes and Decimal Codes of 94 572 lsbf that can be Realized via cnn of Five Input Variables,
`http://www.ee.cityu.edu.hk/~gchen/pdf/list2.pdf`, 2005.

F. Chen, G. He et G. Chen,
« Realization of Boolean Functions via cnn: Mathematical Theory, lsbf and Template Design »,
Ieee *Transactions on Circuit and Systems* i, vol. 53, n° 10, p. 2203–2213, 2006.

F. Y. Chen et G. Chen,
« Realization and Bifurcation of Boolean Functions via Cellular Neural Networks »,

International Journal of Bifurcation and Chaos, vol. 15, n° 7, p. 2109–2129, Juillet 2005.

F. Y. Chen et G. Chen,
A Complete List of Genes, Binary Decoding Tapes, and Decimal Codes of the 1 882 lsbf that Can be Realized via a cnn of Four Input Variables,
`http://www.ee.cityu.edu.hk/~gchen/pdf/list.pdf`, Mai 2004.

C. Chow,
« On the Characterization of Threshold Functions »,
Dans *Proceedings of the Symposium on Switching Circuit Theory and Logical Design, New York,* usa, p. 34–38, 1961.

L. O. Chua et C. W. Wu,
« On the Universe of Stable Cellular Neural Networks »,
International Journal of Circuit Theory and Applications, vol. 20, p. 497–517, 1992.

L. O. Chua et L. Yang,
« Cellular Neural Networks: Theory »,
Ieee *Transactions on Circuits and Systems*, vol. 35, p. 1257–1272, 1988a.

L. O. Chua et L. Yang,
« Cellular Neural Networks: Applications »,
Ieee *Transactions on Circuits and Systems*, vol. 35, p. 1273–1290, 1988b.

H. D. Crane,
« Neuristor — A Novel Device and System Concept »,
Proceedings of the Institute of Radio Engineers, vol. 50, p. 2048–2060, 1962.

K. R. Crounse, T. Roska et L. O. Chua,
« Image halftoning with Cellular Neural Networks »,
Ieee *Transactions on Circuits and Systems – ii : Analog and Digital Signal Processing*, vol. 40, n° 4, p. 267–283, 1993.

K. R. Crounse, E. L. Fung et L. O. Chua,
« Efficient Implementation of Neighborhood Logic for Cellular Automata via the Cellular Neural Network Universal Machine »,
Ieee *Transactions on Circuits and Systems I*, vol. 44, p. 355–361, 1997.

M. Dertouzos,
« An Approach to Single-Threshold-Element Synthesis »,
Ire *Transactions on Electronic Computers*, vol. 13, p. 519–528, 1964.

A. Dinu et M. Cirstea,
« A Novel Digital Neural Network Hardware Implementation Technique Targeting Fpgas »,
Journal of Electrical Engineering, vol. 6, n° 4, 2006.

A. Dinu et M. Cirstea,
« A Digital Neural Network fpga Direct Hardware Implementation Algorithm »,
Dans *International Symposium on Industrial Electronics, Vigo, Spain*, 2007.

R. Dogaru et L. O. Chua,
« Universal cnn cells »,
International Journal of Bifurcation and Chaos, vol. 9, p. 1–48, 1999.

G. Dreyfus, J.-M. Martinez, M. Samuelides, M. B. Gordon, F. Badran, S. Thiria et L. Hérault,
Réseaux de neurones, méthodologie et applications,
Eyrolles, 2002.

W. R. Emamy-K,
« Geometry of Cut-Complexes and Threshold Logic »,
Journal of Geometry, vol. 65, p. 91–100, 1999.

L. Fausett,
Fundamentals of Neural Networks,
Prentice-Hall, 1994.

M. Gilli, M. Biey et P. Checco,
« Equilibrium Analysis of Cellular Neural Networks »,
Ieee *Transactions on Circuits and Systems*, vol. 51, p. 903–912, 2004.

M. Hänggi et G. S. Moschytz,
« Genetic Optimization of Cellular Neural Networks »,
Dans *Proceedings of the* ieee *International Conference on Evolutionnary Computation*, p. 381–386, 1998.

M. Hänggi et G. S. Moschytz,
« An exact and Direct Analytical Method for the Design of Optimally Robust CNN Templates »,
Ieee *Transactions on Circuits and Systems*, vol. 46, p. 304–311, 1999.

M. H. Hassoun,
Fundamentals of Artificial Networks,
Cambridge ms: mit Press, 1995.

S. Haykin,
Neural Networks, A Comprehensive Foundation,
Mac Millan, 2003.

D. O. Hebb,
The Organization of Behavior,
New York : Wiley, 1949.

T. Hegedüs et N. Megiddo,
« On the Geometric Separability of Boolean Functions »,
Discrete Applied Mathematics, vol. 66, n° 3, p. 205–218, 1996.

J. Hertz, A. Krogh et R. G. Palmer,
Introduction to the Theory of Neural Computation,
Addison-Wesley, 1991.

X. Huang et J. Weng,
« Value System Development for a Robot »,
Dans *Proceedings of the International Joint Conference on Neural Networks, Budapest, Hongrie*, 2004.

P. Kaluzny,
« Number of Stable Equilibrium States of Cellular Neural Networks »,
Ieee *Transactions on Circuits and Systems*, vol. 41, p. 608–610, 1994.

A. Kellner, H. Magnussen et J. A. Nossek,
« Texture Segmentation and Text Segmentation with Discrete-Time Cellular Neural Networks »,
Dans *Proceedings of the* ieee *Workshop on Cellular Neural Networks and their Applications*, p. 243–248, 1994.

S. Kohavi,
Switching and Finite Automata Theory,
McGraw Hill, 1978.

T. Kohonen,
Self-organization and Associative Memory,
Berlin : Springer-Verlag, 1989.

T. Kozek, T. Roska et L. O. Chua,
« Genetic Algorithm for cnn Template Learning »,
Ieee *Transactions on Circuits and Systems* i, vol. 40, n° 6, p. 302–402, 1993.

Y. Le Cun,
Modèles connexionnistes de l'apprentissage,
Thèse de doctorat, Université P. et M. Curie (Paris 6), June 1987.

T.-Y. Li et J. A. Yorke,
« Period three implies chaos »,
American Mathematical Monthly, vol. 82, p. 985–992, 1975.

G. Liñán, S. Espejo, R. Domínguez-Castro et A. Rodríguez-Vázquez,
« The cnnuc3: An Analogical i/o 64×64 cnn Universal Machine Chip Prototype with 7-Bit Analog Accuracy »,
Dans *Proceedings of Cellular Neural Networks and their Applications*, p. 201–206, 2000.

F. López-Muñoz, J. Boya et C. Alamo,
« Neuron theory, the cornerstone of neuroscience, on the centenary of the Nobel Prize award to Santiago Ramón y Cajal »,
Brain Research Bulletin, vol. 70, p. 391–405, 2006.

N. Malasne, F. Yang, M. Paindavoine et D. Ginhac,
« Implantation temps réel d'un algorithme de localisation et de reconnaissance de visage sur un fpga »,
Dans *Colloque* gretsi, 2001, Toulouse, France.

T. Matsumoto, T. Yokohama, H. Suzuki, R. Furukawa, A. Oshimoto, T. Shimmi, Y. Matsushita, T. Seo et L. O. Chua,
« Several Image Processing Examples by cnn »,
Dans *Proceedings of the International Workshop on Cellular Neural Networks and their Applications*, 1990.

J. L. Mc Clelland et D. E. Rumelhart,
Explorations in Parallel Distributed Processing,
Cambridge, ma : mit Press, 1988.

W. S. Mc Culloch et W. Pitts,
« A Logical Calculus of the Ideas Immanent in Nervous Activity »,
Bulletin of Mathematical Biophysics, vol. 5, p. 115–133, 1943.

L. Merlat,
Systèmes dynamiques et connexionnisme : une approche différente du calcul neuronal,
Thèse de doctorat, Université de Haute-Alsace, 1998.

M. L. Minsky et S. A. Papert,
Perceptrons: An Introduction to Computational Geometry,
Cambridge, ma : mit Press, 1969.

C. Molter, U. Salihoglu et H. Bersini,
« How chaos boosts the encoding capacity of small Recurrent Neural Networks: learning consideration »,
Dans *Proceedings of* ieee *International Joint Conference on Neural Networks, Budapest, Hongrie*, 2004.

D. Monnin,
Réseaux de neurones cellulaires et traitement d'images : une approche analytique de la conception d'opérateurs,
Thèse de doctorat, Université Joseph Fourier – Grenoble i, 2003.

D. Monnin, L. Merlat, A. Köneke et J. Hérault,
« Design of Cellular Neural Networks for Binary and Gray Level Image Processing »,
Dans *Proceedings of* icann, 1998.

D. Monnin, L. Merlat, A. Köneke et J. Hérault,
« Straightforward Design of Robust Cellular Neural Networks for Image Processing »,
Dans *Proceedings of the International Workshop on Cellular Neural Networks and their Applications*, 2000.

D. Monnin, A. Köneke et J. Hérault,
« Boolean Design of Binary Initialized and Coupled cnn Image Processing Operators »,
Dans *Proceedings of the International Workshop on Cellular Neural Networks and their Applications, Frankfort, Allemagne*, 2002.

S. Muroga,
Threshold Logic and Its Applications,
Wiley, Ne York, 1971.

K. Nakai et A. Ushida,
« Design Technique of Cellular Neural Network »,
Electronics and Communications in Japan, part iii, vol. 78, n° 3, p. 97–107, 1995.

L. Nemes, L. O. Chua et T. Roska,
« Implementation of Arbitrary Boolean Functions on a cnn Universal Machine »,
International Journal of Circuit Theory and Applications, vol. 26, p. 593–610, 1998.

R. Neville,
« A Hardware Implementation of Multi-level Threshold Logic for Artificial Neural Net »,
Dans *Proceedings of the* ieee *World Conference on Computational Intelligence*, p. 5152–5158, 2006.

D. H. Nguyen et B. Widrow,
« Neural Networks for Self-Learning Control Systems »,
Ieee *Control Systems Magazine*, vol. 10, p. 18–23, 1990.

H. Ozdemir, A. Kepkep, B. Pamir, Y. Leblebici et U. Cilingiroglu,
« A Capacitive Threshold-Logic Gate »,
Ieee *Journal of Solid State Circuits*, vol. 31, p. 1141–1150, 1996.

N. H. Packard et S. Wolfram,
« Two-Dimensional Cellular Automata »,
Journal of Statistical Physics, vol. 38, n° 5/6, p. 901–946, 1985.

D. Parker,
« Learning-logic: Casting the cortex of the human brain in silicon »,
Rapport technique tr-47, Center for Computational Research in Economics and Management Science, mit, Cambridge, ma, 1985.

M. C. Paull et E. J. Mc Cluskey,
« Boolean Functions Realizable with Single Threshold Devices »,
Proceedings of the Institute of Radio Engineers, vol. 48, p. 1335–1337, 1960.

P. D. Picton,
« Deriving Weights for Single Threshold Logic Gates Using Decomposition »,
Dans *Proceedings of the International Conference on Neural Information Processing, Stockholm*, 2001.

J. Richards,
« Will computers reach top speed by 2020? »,
Times Online, 19 septembre 2007, adresse : `http://www.timesonline.co.uk/tol/news/`.

F. Rosenblatt,
Principles of neurodynamics,
New York : Spartan, 1962.

F. Rosenblatt,
« The Perceptron: A probabilistic model for information storage and organization in the brain »,
Physical Review, vol. 65, p. 386, 1958.

T. Roska et L. O. Chua,
« The cnn Universal Machine: an Analogic Array Computer »,
Ieee *Transactions on Circuit and Systems* ii, vol. 40, p. 163–173, 1993.

J. Serra,
Image Analysis and Mathematical Morphology, vol. 1,
Academic Press, 1982.

T. Sziranyi et M. Csapodi,
« Texture classification and Segmentation by Cellular Neural Network using Genetic Learning »,
Computer Vision and Image Understanding, vol. 71, p. 255–270, 1998.

N. Takahashi et L. O. Chua,
« A New Sufficient Condition for Nonsymmetric cnn's to Have a Stable Equilibrium Point »,
Ieee *Transactions on Circuits and Systems*, vol. 44, p. 1092–1095, 1997.

N. Takahashi et L. O. Chua,
« On the Complete Stability of Nonsymmetric Cellular Neural Networks »,
Ieee *Transactions on Circuits and Systems*, vol. 45, p. 754–758, 1998.

S. Tan, J. Hao et J. Vandewalle,
« Cellular Neural Networks as a Model of Associative Memory »,
Dans *Proceedings of the* ieee *Workshop on Cellular Neural Networks and their Applications*, p. 26–35, 1990.

G. Tóth,
« Analogic cnn Algorithm for 3d Interpolation-Approximation »,
Rapport technique DNS-2-1995, Analogical and Neural Computing Laboratory, Computer and Automation Research Institute, Académie des Sciences Hongroise (Mta SzTaki), Budapest, 1995.

P. Werbos,
« Backpropagation through time: what it does and how to do it »,
Proceedings of the ieee *78*, vol. 10, p. 1550–1560, 1990.

P. Werbos,
The Roots of Backpropagation,
New York : Wiley, 1994.

B. Widrow et M. E. Hoff Jr,
« Adaptive Switching Circuits »,
Dans Ire *Wescon Convention Record*, p. 96–104, 1960.

B. Widrow et M. A. Lehr,
« Thirty Years of Adaptive Neural Networks: Perceptron, Madaline, and Backpropagation »,
Proceedings of the ieee, vol. 78, n° 9, p. 1415–1442, septembre 1990.

B. Widrow et E. Walach,
Adaptive Inverse Control,
Prentice Hall, 1996.

B. Widrow, R. G. Winter et R. A. Baxter,
« Learning Phenomena in Layered Neural Networks »,
Dans *Proceedings of* ieee *First International Conference on Neural Networks*, vol. 2, p. 411–429, 1987.

E. Wigner,
« The Unreasonnable Effectiveness of Mathematics in the Natural Sciences »,
Communications on Pure and Applied Mathematics, vol. 13, n° 1 :Wiley, Février 1960.

S. Wolfram,
A New Kind of Science,
Wolfram research, 2002.

A. Zarándy,
« The Art of cnn Template Design »,
International Journal of Circuits Theory and Applications, vol. 27, p. 5–23, 1999.

Á. Zarándy et C. Rekeczky,
« Bi-i: a Standalone Cellular Vision System, Part i. Architecture and Ultra High Frame Rate Processing Examples »,
Dans *In proceedings of* ieee *Cellular Neural Networks and their Applications, Budapest, Hongrie*, p. 4–9, 2004a.

Á. Zarándy et C. Rekeczky,
« Bi-i: a Standalone Cellular Vision System, Part ii. Topographic and Non-topographic Algorithm and Related Applications »,
Dans *In proceedings of* ieee *Cellular Neural Networks and their Applications, Budapest, Hongrie*, p. 10–15, 2004b.

Á. Zarándy, T. Roska et L. O. Chua,
« Morphological Operators on the cnnUniversal Machine »,
Dans *Proceedings of the* ieee *Workshop on Cellular Neural Networks and their Applications*, p. 151–156, 1996.