THÈSE

présentée par

Pascal Koiran

devant

l'ÉCOLE NORMALE SUPÉRIEURE DE LYON

l'UNIVERSITÉ CLAUDE BERNARD-LYON I

pour l'obtention

du DIPLÔME DE DOCTORAT

Spécialité: Informatique

Puissance de calcul des réseaux de neurones artificiels

Date de soutenance:	3 juin 1993	
Composition du Jury:	Président:	Maurice Nivat
	Rapporteurs:	Felipe Cucker Jacques Demongeot
	Examinateurs:	Michel Cosnard (directeur de thèse) Max Dauchet Jacques Mazoyer

Thèse préparée à l'Ecole Normale Supérieure de Lyon,
au sein du Laboratoire de l'Informatique du Parallélisme (LIP),
Unité de Recherche Associée au CNRS n° 1398.

Résumé

Depuis quelques années on s'est beaucoup intéressé à la résolution de problèmes d'«ingénierie» avec des réseaux de neurones artificiels (par exemple en reconnaissance de formes, robotique, prédiction de séries temporelles, optimisation...). La plupart de ces travaux sont de nature empirique, et ne comportent que peu ou pas du tout d'analyse mathématique rigoureuse. Cette thèse se situe dans une perspective tout-à-fait différente : il s'agit d'étudier les relations entre les réseaux de neurones et les modèles de calculs classiques ou moins classiques de l'informatique théorique (automates finis, machines de Turing, circuits booléens, machines de Turing réelles de Blum, Shub et Smale).

Les principaux résultats sont les suivants :

- Simulation d'une machine de Turing universelle par des réseaux récurrents.
- Etude générale de la puissance de calcul des systèmes dynamiques définis par des itérations de fonctions, notamment en petites dimensions.
- Etude de modèles de calculs sur les nombres réels, avec application aux réseaux récurrents et acycliques. On montre que la classe des fonctions (discrètes) calculables en temps polynômial est P/poly.

Remerciements

Il est d'usage d'adresser des remerciements dithyrambiques à son directeur de thèse, alors comment faire pour dire que j'ai *vraiment beaucoup* apprécié de travailler sous la direction de Michel ? Essayons quand même. A mon avis, il s'approche de très près de l'idéal du directeur de thèse, malgré les tâches administratives qui l'accablent sans cesse. Il sait à merveille se tenir à l'écart de ces deux écueils : abandonner l'étudiant à son sort, ou au contraire l'amener à produire un travail qui reflète moins ses idées propres que celles de son directeur de thèse [1]. Il m'a non seulement encouragé et guidé tout au long de ma thèse, tantôt de près, tantôt de loin, mais a également eu une influence décisive sur le choix du sujet. Cela va de soi, me direz vous, mais Michel a tout-de-même été l'un des premiers chercheurs français (voire le premier, et peut-être même le seul) à reconnaître l'importance de ce type d'études très fondamentales sur les réseaux de neurones.

Je remercie Maurice Nivat de l'honneur qu'il m'a fait en acceptant de présider mon jury de thèse.

Je remercie Felipe Cucker pour des discussions très utiles sur les modèles de calcul réels, et pour m'avoir fait bénéficier de ses lumières sur la géométrie semi-algébrique. Il a également accepté d'être rapporteur de ma thèse.

Je remercie Max Dauchet de l'intérêt qu'il a manifesté pour mon travail.

Jacques Demongeot a apporté l'indispensable point de vue de l'expert connexioniste, et a accepté d'être rapporteur de ma thèse.

Jacques Mazoyer a manifesté dès le départ de l'intérêt pour mon travail, et il m'a prodigué de nombreux conseils bien utiles. Qu'il en soit ici remercié.

Je remercie les participants réguliers aux groupes de travail de réseaux de neurones et de modèles réels, qui ont eu la patience de m'écouter et la générosité de partager leurs idées : Pascal Bigot, Michel Cosnard, Vladimir Demian, Frédéric Gruau, Martin Matamala, Jean-Christophe Mignot, Hélène Paugam-Moisy et Bruno Poizat.

Merci à mes collègues du LIP pour l'atmosphère à la fois conviviale et stimulante qui règne dans ce laboratoire. Je pense en particulier à mes camarades normaliens (Alain Darte, Bruno Durand, Christophe Mazenc... que les autres me pardonnent).

Pour finir (mais c'est par là qu'il aurait fallu commencer) je remercie mes parents qui m'ont encouragé tout au long de mes études.

1. On a bien dit à propos d'un mathématicien célèbre dont le nom m'échappe : «plus la thèse est médiocre, plus l'étudiant est brillant».

Table des matières

Chapitre 1

Introduction

Les réseaux de neurones formels ont été introduits dans un but de modélisation biologique. Ce domaine a ensuite évolué dans plusieurs directions. Depuis quelques années on s'est beaucoup intéressé à la résolution de problèmes d'«ingénierie» avec ces systèmes (par exemple en reconnaissance de formes, robotique, prédiction de séries temporelles, optimisation...). La plupart de ces travaux sont de nature empirique, et ne comportent que peu ou pas du tout d'analyse mathématique rigoureuse. Cette thèse se situe dans une perspective tout-à-fait différente : il s'agit d'étudier les relations entre les réseaux de neurones et les modèles de calculs classiques ou moins classiques de l'informatique théorique (automates finis, machines de Turing, circuits booléens, machines de Turing réelles de Blum, Shub et Smale).

Le chapitre 2 présente les modèles de calculs qui interviennent souvent dans ce travail : réseaux de neurones, modèles discrets non uniformes et machines de Turing réelles.

Le chapitre 3 est consacré aux simulations d'automates finis par des réseaux de neurones récurrents. On développe à cette occasion des techniques utiles pour le chapitre suivant.

Hava Siegelmann et Eduardo Sontag ont montré récemment que toute machine de Turing peut être simulée par un réseau récurrent à espace d'états continu utilisant une fonction de sortie linéaire seuillée [43, 44, 46]. Le chapitre 4 généralise ce résultat à une classe assez large de fonctions de sorties (qui n'inclut malheureusement pas la tangente hyperbolique, souvent utilisée en pratique, ni d'ailleurs aucune fonction analytique).

Le chapitre 5 est le fruit d'une collaboration avec Michel Cosnard et Max Garzon (voir référence [h] dans la liste de publications personnelles à la fin de cette thèse). Il généralise les résultats de Siegelmann et Sontag dans une direction différente, en posant le problème général de la puissance de calcul des systèmes dynamiques définis par des itérations de fonctions sur $[0,1]^n$. On montre notamment que les fonctions affines par morceaux sont universelles en dimension deux (le résultat de Siegelmann & Sontag correspond à la dimension 1058) et qu'en dimension un il est nécessaire de recourir à des fonctions plus compliquées (par exemple monotones par morceaux) pour conserver cette propriété d'universalité.

Le chapitre 6 étudie le même problème pour des fonctions analytiques réelles (les travaux de Siegelmann & Sontag ou le chapitre 4 n'apportent pas de réponse dans ce cas). On montre que les fonctions analytiques sont universelles en dimension un, et que la simulation d'une machine de Turing se fait en temps linéaire en dimension deux.

Dans les chapitres précédents on ne construit que des systèmes (réseaux, fonctions analytiques ou affines par morceaux...) paramétrés par des nombres rationnels. Les chapitres 7 et 8 étudient les gains de puissance de calcul qu'on peut obtenir en utilisant des paramètres réels.

On montre au chapitre 7 que tout langage (même non récursif) sur un alphabet fini peut être reconnu en temps exponentiel par itération d'une fonction affine par morceaux, et qu'en dimension supérieure l'ensemble des langages reconnaissables en temps polynômial par itération de fonction affine par morceaux est exactement la classe de complexité P/poly. Ces résultats généralisent les travaux antérieurs sur les réseaux de neurones à poids réels ([45], résumé dans [47]).

Le chapitre 8 poursuit l'étude de la reconnaissance de langages dans un cadre plus général : le modèle de Blum, Shub & Smale. On montre que dans une version affaiblie de ce modèle la classe des langages reconnaissable en temps polynômial reste P/poly, ce qui implique que P≠NP dans ce modèle, sauf si la hiérarchie polynômiale de Stockmeyer s'effondre au deuxième niveau. Ce résultat s'applique en particulier à une classe de réseaux récurrents plus large que dans [45]. Il est basé sur un résultat purement «mathématique» apparemment nouveau : un ensemble non vide défini par des «petites» inégalités polynômiales strictes contient un «petit» point rationnel. L'outil des machines de Turing réelles permet également de définir des familles uniformes de réseaux acycliques réels, ce qui mène à des simulations efficaces de familles uniformes de circuits réels par des familles non uniformes de circuits booléens. Ce résultat constitue une généralisation de travaux très récents de Wolfgang Maass sur les familles non uniformes de circuits réels [30].

On peut naturellement se poser la question de la signification et de la pertinence des modèles de calcul sur les réels. Ce problème a déjà été abordé ailleurs (au moins par les partisans de ces modèles [45, 48]); je préfère ne pas entrer dans ce débat et me contenterai de la remarque suivante. L'intérêt pratique de l'étude des calculs numériques en précision limitée est incontestable. On peut naturellement s'attendre à ce que la puissance de calcul du modèle correspondant augmente avec la précision. Certains de mes résultats montrent que cette puissance reste en fait limitée (à P/poly) même si la précision devient illimitée. Il serait certainement intéressant d'étudier cette question plus en détail; imposer une borne sur la précision des calculs revient probablement à imposer une borne correspondante sur l'espace de travail d'une machine de Turing.

On peut également se demander si l'étude de la puissance de calcul des réseaux neuronaux peut apporter quelque chose aux aspects plus pragmatiques de cette discipline. Il est vrai que l'intérêt pratique des réseaux de neurones ne vient pas de leur puissance de calcul (qui est tout à fait comparable à celle de modèles classiques, ainsi que ce travail contribue à le démontrer) mais de leurs capacités d'auto-organisation et d'apprentissage par l'exemple. A cela on peut répondre que des techniques «analytiques» parfois très proches de celles utilisées ici permettent d'évaluer une quantité fondamentale en apprentissage, la dimension de Vapnik-Chervonenkis de classes de réseaux [4, 49, 30, 20] (pour une présentation des liens entre dimension de Vapnik-Chervonenkis et apprentissage, on pourra consulter [7] ou [34]). Plus généralement, il est probable que s'il existe un jour une véritable algorithmique neuronale, elle devra tirer parti des propriétés de base des modèles sous-jacents.

Chapitre 2

Modèles de calcul

2.1 Réseaux de neurones

Un réseau connexioniste est composé de plusieurs unités de calcul simples (appelées *neurones*) interconnectées selon un graphe orienté. Considérons un réseau de n neurones numérotés de 1 à n. A l'instant t, le neurone i se trouve dans un état $x_i(t)$ appartenant à un ensemble d'états $Q_i \subset \mathbb{R}$. La mise à jour des états se fait de manière synchrone par la règle suivante :

$$x_i(t+1) = \phi_i\left(\sum_{j=1}^{n} w_{ij}x_j(t) - \theta_i + I_i(t)\right). \tag{2.1}$$

La fonction $\phi_i : Q_i \to Q_i$ est appelée *fonction de transfert* du neurone i. Le réel θ_i est le seuil du neurone, w_{ij} est le poids de la connexion du neurone j vers le neurone i, et $I_i(t)$ est une entrée externe. La somme qui est passée en argument à la fonction de transfert est l'*activation* du neurone i. L'interprétation «biologique» des poids est que $w_{ij} > 0$ correspond à une synapse excitatrice entre les neurones j et i, $w_{ij} < 0$ à une synapse inhibitrice. Cette interprétation n'a de sens que si les fonctions de sorties sont croissantes, ce qui est bien vrai dans les trois cas usuels présentés ci-dessous.

L'équation (2.1) semble correspondre au cas d'un graphe complet, mais il suffit bien entendu de prendre certains w_{ij} égaux à 0 pour supprimer des connexions. Si le graphe de connexion contient des circuits on parle de réseau récurrent, dans le contraire de réseau acyclique.

Il arrive souvent que les ensembles d'états et fonctions de transfert de tous les neurones d'un réseau soient identiques. Dans ce cas, on notera Q au lieu de Q_i et ϕ au lieu de ϕ_i (et on parle alors de «ϕ-réseau»). Dans la littérature on rencontre souvent les cas suivants :

- $Q = \{0,1\}$ et $\phi(x) = H(x) = \begin{cases} 1 \text{ pour } x \geq 0 \\ 0 \text{ pour } x < 0 \end{cases}$ (fonction de Heaviside ou fonction à seuil).
- $Q = [0,1]$ et $\phi(x) = \sigma(x) = \begin{cases} 1 \text{ pour } x \geq 1 \\ x \text{ pour } 0 \leq x \leq 1 \\ 0 \text{ pour } x \leq 0 \end{cases}$ (fonction linéaire seuillée).

– $Q = [-1, 1]$ et $\phi(x) = \tanh(x)$ (sigmoïde standard); on peut se ramener à $\phi(x) = \frac{1}{1+e^{-x}}$ par un changement de variable.

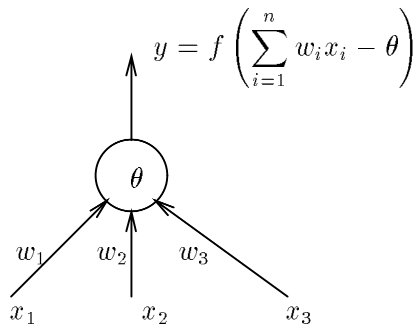

FIG. 2.1 - *Un neurone.*

En général, on appelle sigmoïdes les fonctions dont le graphe ressemble à celui de la tangente hyperbolique (fonctions croissantes, non constantes, bornées). Historiquement on a d'abord considéré des réseaux à seuils, c'est à dire qui utilisent la fonction à seuil (Mc Cullochs & Pitts, 1943), dans un but de modélisation biologique. Si l'activation d'un neurone dépasse un certain seuil il devient actif en émettant des potentiels d'action (état 1), sinon il reste inactif (état 0). En fait un neurone réel peut émettre des potentiels d'actions à des fréquences qui varient entre 0 et une valeur maximum. Il est donc plus réaliste de les modéliser par des systèmes continus comme ceux qu'on obtient avec la sigmoïde standard. Cette fonction est souvent utilisée dans les applications à cause de certaines propriétés sympathiques. Par exemple sa dérivée s'exprime en fonction d'elle-même ($\tanh' = 1 - \tanh^2$), ce qui est utile dans des algorithmes du type «rétropropagation du gradient» (voir par exemple [42]). Malheureusement les réseaux obtenus se livrent très difficilement à une analyse mathématique rigoureuse. C'est pourquoi on utilise souvent dans les travaux théoriques la fonction linéaire seuillée (voir par exemple le chapitre 5).

On peut calculer avec un réseau de neurones en considérant que l'état initial du réseau est la donnée du calcul, ou en fournissant les données grâce aux entrées externes. Le résultat du calcul est lu dans l'état du réseau une fois qu'il s'est stabilisé. Ce point est un peu délicat dans le cas des réseaux récurrents, car même sans entrées externes, il n'y a aucune raison pour qu'on atteigne un point fixe si on ne fait pas d'hypothèse supplémentaire sur la structure du réseau, par exemple une hypothèse de symétrie de la matrice des poids (voir notamment [21] et [24]). Dans ce travail nous ne considérons pas ce type de réseaux. Au chapitre 4 on utilise deux cellules particulières, une «cellule de sortie» et une «cellule de validation». Le résultat du calcul peut être lu sur la cellule de sortie quand la cellule de validation est active. Avec un réseau acyclique on procède en général de la façon suivante : les données du calcul sont contenues dans les neurones de degré entrant nul (appelés *cellules d'entrée*). L'état de ces cellules est maintenu constant pendant tout le calcul en imposant une entrée externe constante. Il n'y a pas d'entrées externes pour les autres neurones du réseau. Celui-ci se stabilise donc en un temps égal (au plus) à la longueur du plus long chemin dans le graphe d'interconnexion,

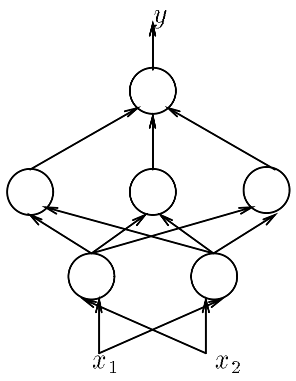

FIG. 2.2 - *Un réseau à deux couches cachées.*

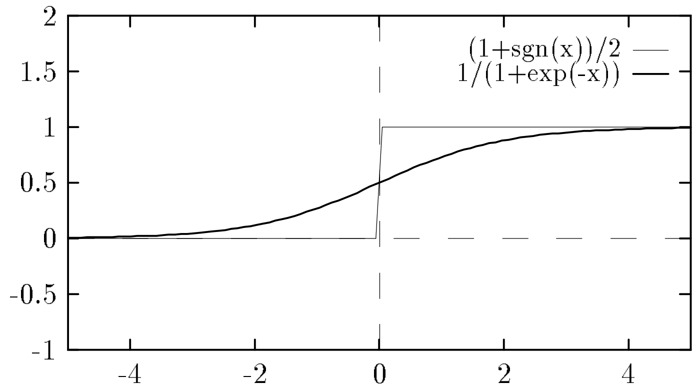

FIG. 2.3 - *La fonction à seuil et la sigmoïde standard*

et le point fixe atteint dépend uniquement de l'état des cellules d'entrée. En général on ne s'intéresse qu'à l'état final des neurones de degré sortant nul (appelés *cellules de sortie*). Les réseaux acycliques sont donc une généralisation de la notion classique de circuit booléen : un réseau acyclique à n cellules d'entrée et p cellules de sortie calcule une fonction de $\mathbb{R}^n$ dans $\mathbb{R}^p$.

Il est parfois possible de partitionner les neurones d'un réseau acyclique en couches, de telle manière que les liens entrants dans un neurone de la couche numéro h ne provienne que de la couche $h-1$, et que les liens sortants ne parviennent qu'à la couche $h+1$. On parle alors de réseau multicouche. Les couches cachées sont celles qui sont situées entre la couche d'entrée (composée des cellules d'entrée) et la couche de sortie (composée des cellules de sortie).

2.2 Modèles discrets non uniformes

En informatique on considère généralement des algorithmes capables de traiter des instances arbitrairement grandes d'un problème. La machine de Turing fournit un modèle abstrait de cette approche. Les circuits booléens permettent au contraire d'étudier le calcul de fonctions définies sur un domaine fini. Un circuit booléen est un graphe acyclique de portes qui sont soit des portes d'entrée booléennes, soit des portes de calcul choisies dans l'ensemble $\{\vee, \wedge, \neg\}$. On suppose souvent qu'il y a une unique porte de sortie (de degré sortant nul), de sorte qu'un circuit C_n à n entrées calcule une fonction de $\{0,1\}^n$ dans $\{0,1\}$. Si on veut se rapprocher de la machine de Turing et reconnaître des langages (c'est à dire des sous-ensembles de $\{0,1\}^*$), il faut considérer des familles de circuits $\mathcal{C} = (C_n)_{n\in\mathbb{N}}$ en prenant un circuit différent pour chaque taille des données. On impose souvent une condition d'uniformité sur $\mathcal{C}$, c'est-à-dire qu'on précise comment cette famille peut être engendrée effectivement (par exemple en temps polynômial ou en espace logarithmique sur une machine de Turing). En effet, on se doute bien par exemple que des circuits qui réalisent des multiplications de nombres de 50 et 100 chiffres respectivement ont un petit air de famille... Ce n'est pas le point de vue que nous adoptons ici : il n'y a pas de condition d'uniformité, d'où le terme de *complexité non uniforme*.

Il est aussi possible de faire de la complexité non uniforme avec des machines de Turing. L'idée est de fournir à la machine de Turing, en plus de la donnée x du problème, une information supplémentaire fournie par une *fonction de conseil* («advice function») dépendant uniquement de la taille $|x|$ de la donnée. On a précisément la définition suivante [2] :

Définition 2.1 *Soit $\mathcal{C}$ une classe de langages et $\mathcal{F}$ une classe de fonctions de conseils de $\mathbb{N}$ dans $\{0,1\}^*$. La classe $\mathcal{C}/\mathcal{F}$ est la classe des langages B tels qu'il existe un langage $A \in \mathcal{C}$ et une fonction $f \in \mathcal{F}$ tels que $B = \{x;\ (x, f(|x|)) \in A\}$.*

Par exemple si poly désigne la classe des fonctions de $\mathbb{N}$ dans $\{0,1\}^*$ telles que la longueur de $f(n)$ est polynômiale en n, on obtient la classe P/poly des langages qui peuvent être reconnus en temps polynômial par une machine de Turing utilisant des conseils de tailles polynômiales. Voici une caractérisation intéressante de P/poly [2].

Théorème 2.1 *Une fonction $F : \{0,1\}^* \to \{0,1\}$ appartient à* P/poly *si et seulement si il existe une famille $(C_n)_{n\in\mathbb{N}}$ de circuits booléens telle que C_n calcule $f_{|\{0,1\}^n}$.*

Il est clair que P/poly contient des fonctions non récursives, puisque la fonction de conseil (où la famille $(C_n)_{n\in\mathbb{N}}$) n'est pas supposée récursive. Réciproquement, certaines fonctions calculables ne sont pas dans P/poly, puisque EXPSPACE$\not\subset$P/poly [2]. On peut dire beaucoup mieux en admettant une hypothèse classique de la théorie de la complexité : si NP $\subset$ P/poly alors la hiérarchie polynômiale s'effondre au deuxième niveau. On pourra trouver des démonstrations de ce résultat dans [3] et dans l'article original [25]. Sur la hiérarchie polynômiale, consulter [2] et [3].

2.3 Le modèle de Blum, Shub & Smale

La complexité algébrique est le domaine de l'informatique qui étudie le nombre d'opérations arithmétiques nécessaire pour résoudre un problème, sans descendre jusqu'au niveau du bit comme on le fait habituellement. Les fondements de cette discipline sont beaucoup moins approfondis que ceux de la complexité discrète usuelle. Cette lacune a été en partie comblée par l'introduction de la machine de Turing réelle de Blum, Shub et Smale [6]. Les motivations de ces auteurs étaient les suivantes : fournir un modèle de calcul proche de la pratique du calcul scientifique, et «to bring the theory of computation into the domain of analysis, geometry and topology». Les modèles de calculs utilisés auparavant en complexité algébrique (les arbres de décision, par exemple) avaient en général une puissance de calcul très limitée. Par contre, la machine de Turing réelle (appelée MTR par la suite) a permis de jeter les bases d'une théorie générale de la calculabilité et de la complexité sur les nombres réels. Blum, Shub et Smale ont notamment montré l'existence de problèmes NP-complets dans ce modèle (existence d'une racine réelle pour un polynôme de degré 4 en n variables).

La machine de Turing réelle est probablement plus proche d'une RAM que d'une machine de Turing. C'est donc de cette manière que nous allons la présenter. On pourra trouver une présentation un peu différente et beaucoup plus formalisée dans [6]. De toute manière les détails précis du modèle n'ont guère d'importance car nous ne nous intéressons qu'au temps polynômial. A ce niveau il en va de même pour les calculs réels que pour les calculs discrets : tous les modèles séquentiels «raisonnables» sont équivalents.

La machine dispose d'une infinité de registres $x_1, \ldots, x_n, \ldots$ pouvant chacun contenir un nombre réel en précision infinie. Un programme contient :

- des instructions du type $x := y$ op z, où x est un registre, y et z des registres ou des constantes, et op l'une des quatre opérations $+, -, *, /$.
- des instructions de branchement du type «if x $\mathcal{R}$ 0 then goto L» où L est un numéro de ligne et $\mathcal{R}$ l'une des relations $<, \leq, =, \geq, >$.
- une instruction de fin de programme.

Chacune de ces instructions est exécutée en temps constant et en précision infinie. Un registre peut également être désigné par un adressage indirect de la forme $x[y]$, où y est un registre (qui doit bien sûr contenir un entier positif).

Un programme peut reconnaître des sous-ensembles de $\mathbb{R}^\infty = \bigcup_{n=1}^{\infty} \mathbb{R}^n$ en convenant que la taille de l'entrée est contenue dans le premier registre et l'entrée elle-même dans les n registres suivants (les autres registres sont initialisés à zéro). Il faut également fixer une convention pour la sortie du programme; on peut par exemple convenir que l'entrée est acceptée si $x_1 \geq 0$ à la fin du calcul. Ceci se généralise de manière évidente au calcul de fonctions de $\mathbb{R}^\infty$ dans $\mathbb{R}^\infty$. Pour parler de complexité il faut définir ce qu'est la taille d'une entrée. Ici c'est simplement le nombre n de registres contenant les données : tous les réels ont une taille unité, quelle que soit leur valeur absolue ou la «quantité d'information» qu'ils contiennent.

Chapitre 3

Simulation d'automates finis

Dans ce court chapitre nous montrons que les automates finis peuvent être simulés par des réseaux récurrents «analogiques» utilisant une sigmoïde quelconque. La situation est donc plus claire que pour les machines de Turing, qu'on ne sait simuler qu'avec des sigmoïdes d'un type particulier (voir le chapitre 4). La simulation est robuste en ce sens qu'elle se déroule correctement même si les neurones sont perturbés par l'injection d'un bruit d'intensité bornée. Techniquement ce résultat est très simple, mais il semble qu'il n'ait jamais été publié auparavant. Il est basé sur une simulation des réseaux à seuils par des réseaux analogiques. Ceci est suffisant car il est bien connu que les réseaux à seuils peuvent simuler les automates finis. Comme notre simulation se fait sans augmentation du nombre de neurones, on en déduit qu'un automate à m états peut être simulé par un réseau analogique de $2m$ neurones en utilisant la construction de Minsky [31], ou de $O(m^{3/4})$ neurones en utilisant une construction beaucoup plus élaborée dûe à A. Dewdney ([17], voir aussi [1]).

Avant de présenter le résultat nous faisons un rappel sur les calculs de fonctions booléennes et les simulations d'automates finis avec des réseaux à seuils. Ces notions et le principe de la simulation seront utiles pour le chapitre 4.

3.1 Calcul de fonctions booléennes

Il est très facile de voir que toutes les fonctions booléennes peuvent être calculées avec des réseaux acycliques à seuils. En effet, les portes de base $\vee, \wedge, \neg$ peuvent être réalisées avec des portes à seuil de la façon suivante : $x \vee y = H(x+y-0.5)$, $x \wedge y = H(x+y-1.5)$ et $\neg x = H(1-x)$. En fait on n'a pas besoin d'une porte supplémentaire pour calculer une négation qui ne se trouve pas en sortie de réseau. En effet, il suffit d'effectuer un changement de poids et de seuils en utilisant directement le fait que $\neg x = 1 - x$. Par exemple, $x \vee (\neg y)$ se calcule avec la porte $H(x-y+0.5)$. Avec une seule porte on peut calculer des disjonctions ou des conjonctions de plusieurs variables. Par exemple, $x_1 \vee x_2 \vee \ldots \vee x_n = H(x_1 + x_2 + \ldots + x_n - 0.5)$. En utilisant la forme normale disjonctive ou conjonctive, on en déduit que toute fonction booléenne peut être calculée par un réseau multicouche à une couche cachée (cette méthode donne un nombre exponentiel de portes dans le pire des cas, ce qui est d'ailleurs inévitable).

On constate que dans les exemples ci-dessus l'activation de la porte à seuil ne s'annule pour aucune valeur (booléenne) des entrées. Ceci est en fait une propriété

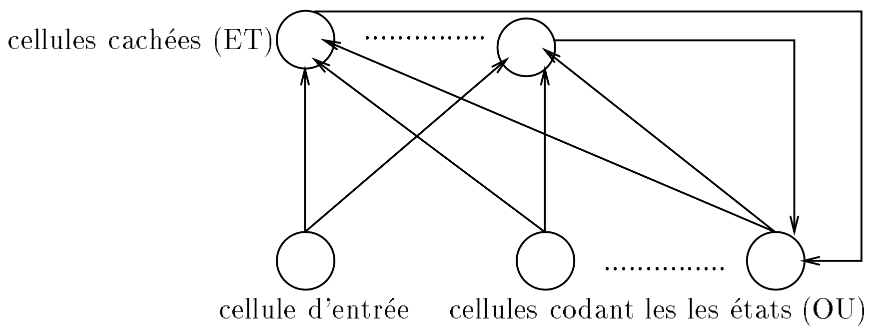

FIG. 3.1 - *Simulation d'un automate avec la forme normale disjonctive (vue partielle).*

générale : toute fonction à seuil $f : \{0,1\}^n \rightarrow \{0,1\}$ peut être calculée par une porte à seuil dont l'activation ne s'annule jamais. Supposons en effet que cette fonction est de la forme

$$f(x) = H\left(\sum_{i=1}^{n} w_i x_i - \theta\right)$$

et soit $\alpha = -\max\{\sum_{i=1}^{n} w_i x_i - \theta;\ x \in \{0,1\}^n \text{ et } f(x) = 0\}$. Alors on a également

$$f(x) = H\left(\sum_{i=1}^{n} w_i x_i - \theta + \frac{\alpha}{2}\right)$$

et sous cette forme l'activation ne s'annule jamais. Cette propriété sera utile dans la section suivante.

Maintenant qu'on sait calculer des fonctions booléennes avec des réseaux acycliques, il est facile de voir qu'on peut simuler des automates finis avec des réseaux récurrents. Pour simuler un automate $\mathcal{M}$ de m états avec entrée sur un alphabet binaire, on construit une première couche composée de $p = \lceil \log m \rceil$ cellules pour encoder les états de $\mathcal{M}$, et d'une cellule d'entrée binaire x_0 qui fournit le mot en entrée de $\mathcal{M}$. Calculer la fonction de transition de l'automate revient à calculer une certaine fonction booléenne de $\{0,1\}^{p+1}$ dans $\{0,1\}^p$, ce qui peut être fait par un réseau à deux couches. Pour obtenir le réseau récurrent cherché, il suffit de «reboucler» les sorties de ce réseau multicouche sur ses p entrées. Une transition de l'automate se simule donc en deux unités de temps. Pour cette raison la lettre lue par $\mathcal{M}$ au temps t ne peut être placée en x_0 qu'au temps $2t$ (ce qui se passe sur cette cellule aux instants impairs n'a pas d'influence sur le calcul). La complexité de cette construction assez naïve est de $O(m \log m)$ neurones : les solutions de Minsky et de Dewdney sont donc nettement meilleures.

3.2 Simulation de réseaux à seuils

On considère un ϕ-réseau à entrées externes dont la dynamique est définie par l'équation suivante :

$$x_i(t+1) = \phi\left(\sum_{j=0}^{n} w_{ij}x_j(t) - \theta_i + b_i(t)\right)$$

pour $1 \leq i \leq n$. La cellule x_0 fournit le mot (infini) lu par le réseau à seuils : ce mot étant $u \in \{0,1\}^\omega$, on a $x_0(t) = u_t$. La seule hypothèse sur le terme de bruit $b_i(t)$ est qu'il est d'amplitude bornée : $\exists B \geq 0, \forall t \geq 0,\ |b_i(t)| \leq B$. Les réseaux à seuils qu'on simule sont du même type avec $\phi = H$, mais sans bruit.

Théorème 3.1 *Soit $\mathcal{N}$ un réseau à seuils, $B \geq 0$ une intensité de bruit, et $\phi : \mathbb{R} \to \mathbb{R}$ une fonction telle que $\lim_{x\to+\infty} \phi(x) = 1$ et $\lim_{x\to-\infty} \phi(x) = 0$. Il existe un ϕ-réseau $\mathcal{N}'$ qui simule $\mathcal{N}$ sous un bruit d'intensité B. Ce réseau se déduit de $\mathcal{N}$ par un changement des poids et des seuils. Si on dénote les états des neurones dans $\mathcal{N}$ et $\mathcal{N}'$ par $x_i(t)$ et $x'_i(t)$ on a la propriété suivante : $\exists x_b, x_s \in]0,1[$ tels que $x_b < x_s$ et*

$$\forall t \geq 0, \begin{cases} x_i(t) = 0 \Longrightarrow x'_i(t) \in [-x_b, x_b]; \\ x_i(t) = 1 \Longrightarrow x'_i(t) \in [x_s, 2 - x_s]. \end{cases} \tag{3.1}$$

Preuve. D'après la section précédente, on peut supposer sans perte de généralité que

$$\exists \alpha > 0, \forall x \in \{0,1\}^{n+1}, \forall i, \left|\sum_{j=0}^{n} w_{ij}x_j - \theta_i\right| \geq \alpha.$$

$\mathcal{N}'$ est obtenu à partir de $\mathcal{N}$ par les transformations suivantes : $w'_{ij} = Aw_{ij}$ et $\theta'_i = A\theta_i$, où A est une constante positive suffisamment grande.

Raisonnons par récurrence en supposant que l'équation (3.1) est satisfaite à l'instant t, pour des valeurs de x_b et x_s qui restent à déterminer. Alors

$$\left|\sum_{j=1}^{n} w_{ij}x_j(t) - \sum_{j=1}^{n} w_{ij}x'_j(t)\right| \leq mne,$$

avec $m = \max_{i,j}|w_{ij}|$ et $e = \max(x_b, 1 - x_s)$. Si on choisit x_b et x_s de manière que $e \leq \alpha/(2mn)$, on a donc

$$\sum_{i=0}^{n} w_{ij}x'_j(t) - \theta_i \geq \alpha/2$$

si $x_i(t+1) = 1$ et

$$\sum_{i=0}^{n} w_{ij}x'_j(t) - \theta_i \leq \alpha/2$$

si $x_i(t+1) = 0$. Soient y_b et y_s deux réels tels que $y \leq y_b \Longrightarrow \phi(y) \in [-x_b, x_b]$ et $y \geq y_s \Longrightarrow \phi(y) \in [x_s, 2 - x_s]$. Prenons alors A de sorte que $A\dfrac{\alpha}{2} - B \geq y_s$ et $-A\dfrac{\alpha}{2} + B \leq y_b$. L'équation (3.1) est clairement satisfaite à l'instant $t+1$. □

Remarquons enfin que l'hypothèse que les limites de ϕ à l'infini soient 0 et 1 n'est pas réellement restrictive. Supposons en effet que ces limites soient L_1 et L_2 ($L_1 \neq L_2$) au lieu de 0 et 1. La fonction $\psi(x) = (\phi(x) - L_1)/(L_2 - L_1)$ satisfait les hypothèses de la définition 4.1, donc pour tout automate il existe un ψ-réseau qui le simule. Ce réseau peut être simulé par un ϕ-réseau grâce aux changements de variables suivants :

- $x'_i = (L_2 - L_1)x_i + L_1$ pour les états des neurones;
- $w'_{ij} = w_{ij}/(l_2 - L_1)$ pour les poids;
- $\theta'_i = \theta_i + L_1 \sum_j w'_{ij}$ pour les seuils.

Chapitre 4

Réseaux récurrents universels

Hava Siegelmann et Eduardo Sontag ont récemment montré que les réseaux récurrents sont universels, c'est à dire qu'ils peuvent simuler une machine de Turing quelconque [46]. Leur preuve est valable uniquement pour des neurones utilisant la sigmoïde linéaire seuillée σ. Dans ce chapitre nous étendons ce résultat à une classe de fonctions de sortie assez large (mais qui n'inclut malheureusement pas la tangente hyperbolique). Cette classe est constituée des fonctions égales à σ au voisinage de $\pm\infty$ et qui sont non constantes (et vérifient une certaine condition de régularité) sur un intervalle I. L'idée de la preuve est d'approximer la partie linéaire de σ par $\phi_{|I}$, ce qui permet de simuler des automates à deux piles unaires. Il est bien connu que ces machines sont universelles [23]. Le principal désavantage de ce codage unaire (utilisé par Hava Siegelmann et Eduardo Sontag dans une version préliminaire de leur article [43]) est qu'une simulation de machine de Turing se fait alors en temps exponentiel. Dans la version définitive [46] ils utilisent des piles binaires, ce qui permet une simulation en temps linéaire. Ce codage plus efficace semble malheureusement mal adapté à notre technique de simulation (il est par contre utilisé au chapitre 5).

La fonction linéaire seuillée σ est particulièrement bien adaptée aux simulations de machines de Turing (ou de manière équivalente d'automates à deux piles) car les opérations sur les piles (empiler – dépiler – ne rien faire) peuvent se réaliser exactement, sans «erreur d'arrondi». Avec ϕ il est possible de réaliser ces opérations avec une précision arbitraire, mais fixée pour un réseau donné. La principale difficulté vient de ce qu'au cours du temps les erreurs s'accumulent, ce qui risque de compromettre la justesse de la simulation. Pour résoudre ce problème on duplique chaque pile, et la mémoire est constamment transvasée d'une pile à l'autre. Cette solution fonctionne car vider une pile permet d'effacer les erreurs qui s'y sont accumulées. Il est amusant de constater qu'une idée similaire est utilisée dans les ordinateurs de tous les jours : certains types de mémoires RAM doivent être lues et écrites en permanence, à cause de «fuites» qui risquent de corrompre leur contenu. Mentionnons enfin que la partie contrôle de l'automate ne pose pas de problème : les transitions d'état peuvent être réalisées exactement car $\phi = \sigma$ à l'infini.

Le résultat de simulation est énoncé précisément dans la section 4.1 et démontré dans les sections suivantes.

4.1 Présentation du résultat

On considère des réseaux récurrents avec entrées externes, dont tous les neurones utilisent même fonction de transfert ϕ. L'évolution du réseau est décrite par l'équation

$$x_i(t+1) = \phi\left(\sum_{j=1}^{n} w_{ij}x_j(t) + a_iE(t) - \theta_i\right).$$

La ligne d'entrée E est binaire : $\forall t, E(t) \in \{0,1\}$. Dans ce chapitre uniquement, on adopte la définition suivante d'une fonction sigmoïdale.

Définition 4.1 *ϕ est une sigmoïde si $\phi(x) = 0$ pour $x \leq 0$, $\phi(x) = 1$ pour $x \geq 1$, et ϕ satisfait la condition de régularité suivante : il existe un intervalle ouvert I tel que ϕ est C^2 sur I, et $\forall x \in I, f'(x) \neq 0$.*

Avec ces réseaux on calcule des fonctions de $\mathbb{N}$ dans $\{0,1\}$ de la manière suivante. Au départ tous les neurones sont dans l'état 0, sauf un neurone désigné à l'avance qui est dans l'état 1. La donnée $n \in \mathbb{N}$ du calcul est lue sur la ligne E de la manière suivante : $E(t) = 1$ pour $0 \leq t \leq 4n-1$, $E(t) = 0$ pour $t \geq 4n$. Le rśultat du calcul est lu sur deux unités spécifiés à l'avance, mettons x_0 (l'unité de validation) et x_1 (l'unité de sortie). Par construction on verra qu'il existe au plus un t tel que $x_0(4t) = 1$. Le résultat du calcul est alors $x_1(4t) \in \{0,1\}$. Si un tel t n'existe pas, le résultat du calcul est non défini. Pour la même raison, seules les valeurs présentes sur la ligne d'entrée à un instant de la forme $4t$ ont une importance.

La présence du terme $4t$ est dû au fait que les réseaux qu'on va construire ont quatre couches, et donc il faut quatre unités de temps pour simuler une transition d'un automate à piles.

Théorème 4.1 *Soit ϕ une sigmoïde. Pour toute fonction récursive partielle $f : \mathbb{N} \to \{0,1\}$ il existe un ϕ-réseau qui calcule f avec les conventions d'entrées-sorties définies ci-dessus.*

On peut voir de la même manière qu'au chapitre 3 que l'hypothèse que les valeurs prises par ϕ à l'infini soient 0 et 1 n'est pas réellement restrictive. On peut également montrer par une manipulation similaire qu'il est possible de supposer uniquement qu'il existe $A > 0$ tel que $\phi(x) = 0$ ou 1 suivant que $x \leq -A$ ou $x \geq A$.

4.2 Les piles

Dans cette section nous montrons comment réaliser les opérations de piles. Des conditions suffisantes pour réaliser ces opérations par itération de fonctions sont établies dans la section 4.2.1, et on montre comment satisfaire ces conditions avec des sigmoïdes dans la section 4.2.2. En fait on ne simule pas des suites quelconques d'opérations, mais uniquement des suites de n empilages suivis de n dépilages. On montre dans la section 4.3 que cela est bien suffisant pour simuler des machines de Turing.

4.2.1 Empiler et dépiler

Dans le résultat suivant, f doit être vue comme une fonction d'empilage ou de dépilage approchée, et l comme l'opération exacte.

Lemme 4.1 *Soient l et f deux fonctions telles que la condition suivante soit satisfaite sur un intervalle $I = [x_\infty, x_m]$ avec $x_m > x_\infty$:*

1. *$l(x) = a(x - x_\infty) + x_\infty$ avec $a > 0$, $a \neq 1$.*
2. *$l(x) \geq f(x) \geq l(x) - M(x - x_\infty)^2$, avec $M \geq 0$.*

Soit $s_n = f^n(s_0)$, $y_n = l^n(y_0)$, avec $s_0 \leq y_0$. Supposons également que $s_p, y_p \in I$ pour $0 \leq p \leq n-1$. On a alors la relation suivante :

$$|s_n - y_n| \leq (|s_0 - y_0| - E)\, a^n + Ea^{2n} \tag{4.1}$$

avec $E = M\dfrac{(y_0 - x_\infty)^2}{a^2 - a}$.

Preuve. Par récurrence. Le résultat est clairement vrai pour $n = 0$. Par l'inégalité triangulaire on a :

$$|s_{n+1} - y_{n+1}| = |f(s_n) - l(y_n)| \leq |f(s_n) - l(s_n)| + |l(s_n) - l(y_n)|,$$

donc d'après l'hypothèse 2 du lemme,

$$|s_{n+1} - y_{n+1}| \leq M(s_n - x_\infty)^2 + a|s_n - y_n| \leq M(y_n - x_\infty)^2 + a|s_n - y_n|.$$

La dernière inégalité vient de ce que $x_\infty \leq s_n \leq y_n$ d'après l'hypothèse 2. Supposons le résultat vrai au rang n. Comme $|y_n - x_\infty| = |y_0 - x_\infty|a^n$, on a :

$$|s_{n+1} - y_{n+1}| \leq a\left[(|s_0 - y_0| - E)\, a^n + Ea^{2n}\right] + M(y_0 - x_\infty)^2 a^{2n}.$$

Le résultat au rang $n+1$ découle donc de la relation $Ea + M(y_0 - x_\infty)^2 = Ea^2$. $\square$

On va maintenant appliquer ce résultat deux fois de suite, d'abord pour empiler n symboles avec $(f, l) = (f_1, l_1)$, puis les dépiler avec $(f, l) = (f_2, l_2)$. Soit $l_1(y) = (y + x_\infty)/2$, $l_2(y) = l_1^{-1}(y) = 2y - x_\infty$, et $x_0 \in]x_\infty, x_m]$. On empile en définissant $s_{1,n}$ et $y_{1,n}$ par : $s_{1,n} = f_1^n(x_0)$, $y_{1,n} = l_1^n(x_0)$. Pour un n fixé, on dépile en définissant $s_{2,p}$ et $y_{2,p}$ par $s_{2,p} = f_2^p(s_{1,n})$, $y_{2,p} = l_2^p(y_{1,n})$, pour $0 \leq p \leq n$.

D'après (4.1) on a :

$$\begin{cases} |s_{2,p} - y_{2,p}| \leq |s_{1,n} - y_{1,n}| 2^p + \dfrac{M}{2}(y_{1,n} - x_\infty)^2 2^{2p} \\ |s_{1,n} - y_{1,n}| \leq \dfrac{4M}{2^n}(x_0 - x_\infty)^2 \end{cases}$$

Comme $y_{1,n} - x_\infty = (x_0 - x_\infty)/2^n$,

$$|s_{2,p} - y_{2,p}| \leq 4M(x_0 - x_\infty)^2 2^{p-n} + \frac{M}{2}(x_0 - x_\infty)^2 2^{2(p-n)}$$

d'où puisque $p \leq n$:
$$|s_{2,p} - y_{2,p}| \leq \frac{9M}{2}(x_0 - x_\infty)^2.$$
Remarquons que $y_{2,n-1} = y_{1,1} = (x_0 + x_\infty)/2$ et $y_{2,n} = y_{1,0} = x_0$. Par conséquent si M est maintenu constant et x_0 pris suffisament proche de x_∞, on peut garantir la propriété de séparation suivante :

$$\begin{cases} s_{2,p} \leq b_i = (2x_0 + x_\infty)/3 \text{ pour } p < n; \\ s_{2,n} \geq b_s = (3x_0 + x_\infty)/4. \end{cases} \tag{4.2}$$

Il est donc facile de détecter si tous les éléments empilés par application de f_1 ont été dépilés par application de f_2 (cas $p = n$), ou seulement une partie d'entre eux (cas $p < n$).

4.2.2 Implémentation avec des fonctions sigmoïdales

Nous faisons les choix suivants :

- Les fonctions d'empilage et de dépilage sont de la forme
$$f_i(x) = \phi(\alpha_i \phi(\beta_i \psi(x) + \gamma_i) + \delta_i) \quad (i = 1, 2)$$
avec
$$\psi(x) = \phi(\alpha\phi(\beta x + \gamma) + \delta);$$
- $x_\infty = \phi(b)$, avec $b \in I$;
- $x_0 = \phi(c)$, avec $c \in I$.

L'utilité du terme $\psi(x)$ n'apparaîtra que dans la section 4.3 : nous construisons un circuit de profondeur 4 pour calculer les f_i. Sans cette contrainte on pourrait très bien prendre $\psi(x) = x$.

Une fois que b et les f_i ont été choisis, on peut prendre c tel que $x_0 = \phi(c)$ soit strictement supérieur à x_∞ et suffisament proche de x_∞ pour que (4.2) soit satisfaite (puisque $\phi'(b) \neq 0$).

Montrons maintenant comment choisir $x_\infty = \phi(b)$ et les f_i. Si $\phi'' = 0$ sur I, ϕ est linéaire et non constante sur cet intervalle. Il suffit donc de prendre $b \in I$ arbitraire, et les conditions du lemme 4.1 sont satisfaites avec $M = 0$. Supposons maintenant que ϕ'' n'est pas identiquement nulle sur I, et soit $b \in I$ tel que $\phi''(b) \neq 0$. Pour que les conditions du lemme 4.1 soient satisfaites avec les fonctions d'empilage et de dépilage exacts l_1 et l_2 définies dans la section (4.2.1), il suffit que :

- $f_i(x_\infty) = x_\infty$.
- $f_i'(x_\infty) = d_i$, avec $d_1 = 1/2$ et $d_2 = 2$.
- $f_i''(x_\infty) < 0$.

Montrons d'abord comment obtenir que $\psi(x_\infty) = x_\infty$, $\psi'(x_\infty) = 1$ et $\psi''(x_\infty) = 0$. Pour que la première condition soit satisfaite, il suffit de choisir α, β, γ, et δ de manière que $\beta x_\infty + \gamma = b$ et $\alpha x_\infty + \delta = b$. On a alors

$$\psi'(x_\infty) = \alpha\beta\phi'(b)^2$$

et

$$\psi''(x_\infty) = \alpha\beta^2\phi'(b)\phi''(b)[\alpha\phi'(b) + 1].$$

On prend donc $\alpha = -1/\phi'(b)$ pour que $\psi''(x_\infty) = 0$, puis $\beta = 1/[\alpha\phi'(b)^2]$ pour que $\psi'(x_\infty) = 1$ et enfin $\gamma = b - \beta x_\infty$, $\delta = b - \alpha x_\infty$.

Choisissons également les α_i, β_i, γ_i, et δ_i de manière que $\beta_i x_\infty + \gamma_i = b$ et $\alpha_i x_\infty + \delta_i = b$. On a alors $f_i(x_\infty) = x_\infty$, et en utilisant les propriétés de ψ obtenues ci-dessus il vient :

$$f_i'(x_\infty) = \alpha_i\beta_i\phi'(b)^2$$

et

$$f_i''(x_\infty) = \alpha_i\beta_i\phi'(b)\phi''(b)[\alpha_i\beta_i\phi'(b) + \beta_i].$$

Il faut que $\alpha_i\beta_i = d_i/\phi'(b)^2$ afin d'avoir $f_i'(x_\infty) = d_i$. On peut obtenir une valeur arbitraire pour $f_i''(x_\infty)$, par exemple -1, en faisant varier β_i et en maintenant la relation $\alpha_i = d_i/[\beta_i\phi'(b)^2]$. On prend enfin $\gamma_i = b - \beta_i x_\infty$ et $\delta_i = b - \alpha_i x_\infty$.

4.3 Contrôle et structure du résau

Il est bien connu que les automates à deux piles unaires sont universels [23] (des simulations plus simples et plus rapides sont possibles en utilisant plus de deux piles). Nous allons montrer qu'on peut simuler tout automate à deux piles (unaires) M par un automate à quatre piles M' d'un type spécial, et que M' peut être simulé par un réseau de neurones. Le réseau commence par empiler la donnée lue sur la ligne E. La simulation de M' proprement dite commence une fois que l'entrée a été entièrement lue. Le passage de la phase d'empilement à la phase de simulation puis à la production du résultat suivant les règles énoncées dans la section 4.1 ne pose pas de problème, puisqu'on sait simuler des automates d'états finis. Dans cette section on ne décrit donc que la phase de simulation.

4.3.1 Contrôle

Les fonctions de transition de M' sont notées θ_i, $i = 0...4$ où

- $\theta_0 : S \times \{0,1\}^4 \to S$ est la fonction de transition d'état. L'ensemble d'états de M' est S et dans les quatre dernières composantes 0 représente une pile vide, 1 une pile non vide. Par la suite ces quatre composantes sont notées $e = (e_1, \ldots, e_4)$.

- Les opérations de piles sont spécifiées par $\theta_i : S \times \{0,1\}^p \to \{u, d, r\}$, $i = 1...4$. Plus précisément, u indique un empilage, d un dépilage, et l'instruction de remise à zéro r est un «no-op» d'un genre spécial : elle ne peut être appliquée qu'à une pile vide.

Chaque pile P de M est représentée par une paire (P_1, P_2) de piles de M'. Cet automate transfère constamment le contenu de P d'une pile sur l'autre par un mouvement de va et vient : P est transféré de P_1 vers P_2, puis quand P_1 est vide de P_2 vers P_1, puis à nouveau de P_1 vers P_2, etc. L'intêret de cette manipulation vient de ce que M' doit être simulé par un réseau de neurones, et d'après la propriété de séparation (4.2) une suite d'empilages suivies d'autant de dépilages peut être réalisée sans que les erreurs s'accumulent trop. Il faut cependant prendre garde à effacer les erreurs qui se sont accumulées sur une pile une fois qu'elle a été vidée. Ceci se fait par l'opération de remise à zéro r. Au niveau de M' cette opération est bien entendu inutile, mais elle est essentielle pour que la simulation par réseau de neurones soit correcte.

Rentrons un peu plus dans le détail du fonctionnement de M', sans aller toutefois jusqu'à une description complètement formalisée. Une fois qu' une pile (par exemple P_1) a été vidée il faut empiler un élément supplémentaire sur P_2 afin d'avoir le temps d'appliquer l'opération r à P_1. On se débarasse ensuite de cet élément en le dépilant à l'instant suivant. Pour simuler une opération d'empilement de M il suffit d'empiler cet élément sur P_1 au lieu de l'effacer. Pour une opération de dépilement, on efface les deux premiers éléments de P_2 (pendant ce temps P_1 est soumise à des opérations de remise à zéro). Enfin pendant toute la durée de simulation d'une transition de M, M' doit conserver et mettre à jour la liste des opérations de piles de M à effectuer. La transition ne peut bien sûr être terminée avant que cette liste soit vide.

4.3.2 Structure du réseau

On simule M' par un réseau à quatre couches, la sortie de la quatrième couche étant rebouclée sur l'entrée de la première. Cette première couche est la partie visible du réseau, celle qui encode l'état et les piles de M'. Soulignons qu'il est essentiel que la structure du réseau soit bien strictement multicouche, c'est à dire qu'il n'y ait pas de «connexion pontée» entre deux couches non consécutives. Dans le cas contraire on aurait un problème de synchronisation dû au rebouclage de la quatrième couche sur la première. Les unités de cette couche sont réparties en deux groupes :

- un ensemble U_P de quatre unités dont chacune code une pile de M';
- un ensemble U_S indexé par S d'«unités de piles» dont chacune correspond à un état de M' : si M' est dans l'état i à l'instant t, $x_i(4t) = 1$ et $x_j(4t) = 0$ pour $j \in S, j{\neq}i$.

La fonction de transition d'une unité de pile $i \in U_P$ est définie par :

$$x_i(t+4) = \phi(\alpha_1\phi(\beta_1\psi(x_i(t)) + B_{1,i}) + \alpha_2\phi(\beta_2\psi(x_i(t)) + B_{2,i}) + B_i).$$

Les valeurs des biais $B_{1,i}$, $B_{2,i}$ et B_i determinent l'opération appliquée à la pile. Par construction $x_i(t)$ appartient à un sous-intervalle borné de I, donc il existe une constante $C > 0$ telle que $\forall i, t, |\psi(x_i(t))| \leq C$. Rappelons également que $\phi(x) = 0$ pour $x \leq 0$. Pour que l'opération sélectionnée soit correcte, il suffit donc que les valeurs des entrées soient les suivantes quand M' est dans l'état j (et donc $x_j = 1$ dans le réseau) :

- si $\theta_i(j, e) = u$ alors $B_{1,i} = \gamma_1$, $B_{2,i} = -C|\beta_2|$, $B_i = \delta_1$;
- si $\theta_i(j, e) = d$ alors $B_{1,i} = -C|\beta_1|$, $B_{2,i} = \gamma_2$, $B_i = \delta_2$;

– si $\theta_i(j,e) = r$ alors $B_{1,i} = -C|\beta_1|$, $B_{2,i} = -C|\beta_2|$, $B_i = c$.

L'entrée $B_{1,i}$ peut être réalisée de la manière suivante :

$$B_{1,i} = \sum_{j \in S, e \in \{0,1\}^4} c_{i,j,e} \mathrm{ET}(x_j(t), \delta(x_{j_1}(t), e_1), \ldots, \delta(x_{j_p}(t), e_p))$$

avec $c_{i,j,e} = \gamma_1$ si $\theta_i(j,e) = u$, $c_{i,j,e} = -C|\beta_1|$ sinon. Le test de pile vide δ est défini par :

$$\begin{cases} \delta(x,0) = 1 \text{ si } x \geq b_s, 0 \text{ si } x \leq b_i; \\ \delta(x,1) = 1 \text{ si } x \leq b_i, 0 \text{ si } x \geq b_s. \end{cases}$$

Ces tests peuvent être implémentés de la manière suivante :

$$\delta(x,0) = \phi\left(\frac{x - b_i}{b_s - b_i}\right)$$

et

$$\delta(x,1) = \phi\left(\frac{b_s - x}{b_s - b_i}\right).$$

Un terme $\mathrm{ET}(x, \delta_1, \ldots, \delta_p)$ peut être implémenté par :

$$\phi(\phi(x) + \delta_1 + \ldots + \delta_p - p - 1).$$

On utilise la même construction pour $B_{2,i}$; pour B_i, il faut ajouter des fonctions ϕ redondantes afin d'obtenir un réseau de profondeur 3.

La fonction de transition d'une unité d'état $i \in U_S$ est définie par :

$$x_i(t+4) = \phi\left(\sum_{(j,e) \in \theta_0^{-1}(i)} \mathrm{ET}(x_j(t), \delta(x_{j_1}(t), e_1), \ldots, \delta(x_{j_p}(t), e_p))\right).$$

Ce terme peut être transformé en un réseau de profondeur 4 en utilisant les mêmes techniques que pour les unités de piles.

Chapitre 5

Calculabilité avec des systèmes dynamiques en basses dimensions

Au chapitre 4 nous avons généralisé le résultat de Siegelmann & Sontag sur l'existence de réseaux universels. Ici nous le généralisons dans une direction tout-à-fait différente. Comme leur réseau utilise la sigmoïde linéaire seuillée σ, sa fonction de transition globale est affine par morceaux (de $[0,1]^d$ dans $[0,1]^d$ s'il est constitué de d unités). Par conséquent leur résultat peut être interprété de la manière suivante : «on peut simuler une machine de Turing universelle par itération d'une fonction affine par morceaux» (ils montrent que $d = 1058$ est suffisant). La question se pose donc de déterminer la dimension minimale pour laquelle on peut simuler les machines de Turing en itérant des fonctions affines par morceaux. Dans ce chapitre nous montrons que $d = 2$ est suffisant. De plus, notre construction peut être utilisée pour donner une preuve différente de l'existence de réseaux universels.

Nous étudions également les simulations d'automates cellulaires et la puissance de calcul des fonctions affines par morceaux et d'autres classes naturelles de fonctions en dimension 1. Nos principaux résultats sont qu'une machine de Turing quelconque ne peut pas être simulée par itération d'une fonction affine par morceaux en dimension 1, et qu'un automate cellulaire ne peut pas être simulé par itération d'une fonction affine par morceaux en dimension 2.

Tous les résultats positifs sont constructifs. Les résultats négatifs sont démontrés pour une classe de simulations naturelles, mais quelque peu restrictive. Nous proposons enfin un cadre général pour étudier les machines universelles. Un tel cadre pourrait être utile pour obtenir des résultats négatifs plus généraux.

Les itérations de fonctions affines par morceaux et monotones par morceaux ont été très étudiées dans la littérature sur les systèmes dynamiques, et on sait que des comportements très riches existent (voir par exemple [11] et [35]). Le comportement de ces itérations en terme de puissance de calcul est-il aussi riche ? Ceci nous semble être un important problème ouvert. Dans [32] et [33], C. Moore a étudié les systèmes dynamiques universels (au sens du calcul). Ses principaux résultats sont similaires à notre théorème 5.1 : une machine de Turing universelle peut être simulée par un décalage généralisé («generalized shift») sur un ensemble de Cantor en dimension 2, ce décalage peut être plongé dans une fonction «régulière» de $\mathbb{R}^2$, et dans un flot «régulier» de $\mathbb{R}^3$.

5.1 Préliminaires

Les définitions et notations fréquemment utilisées dans ce chapitre sont énumérées ci-dessous. L'intervalle unité est $I = [0, 1]$; PL_d est l'ensemble des fonctions continues et affines par morceaux sur I^d. Plus précisément, $f : I^d \to I^d$ appartient à PL_d si

- f est continue;
- il existe une famille $\mathcal{P} = (P_i)_{1 \leq i \leq p}$ de polyèdres convexes fermés (et d'intérieur non vide) telle que $f_i = f_{|P_i}$ est affine, $I^d = \bigcup_{i=1}^{p} P_i$ et $\overset{\circ}{P_i} \cap \overset{\circ}{P_j} = \emptyset$ pour $i \neq j$.

Dans le cas $d = 1$, nous utilisons également la notation $I_i = [c_i, c_{i+1}] = P_i$. Soient $(a_i, b_i)_{1 \leq i \leq p}$ les paramètres tels que $f_i(x) = a_i x + b_i$ pour $x \in [c_i, c_{i+1}]$. Le sous-ensemble de PL_1 composé des fonctions telles que les a_i, b_i et c_i soient tous rationnels est noté RPL_1.Les fonctions analytiques par morceaux et monotones par morceaux sont définies de la même manière évidente. Si l'ensemble des intervalles I_i n'est pas fini mais dénombrable, nous parlerons de fonctions dénombrablement affines (ou analytiques, monotones, etc) par morceaux. Dans ce cas, certains des I_i peuvent être réduits à un point.

Etant donné une fonction $f : I^d \to I^d$, une suite $(x_i)_{0 \leq i \leq k-1}$ de points distincts est un k-cycle si $f(x_i) = x_{i+1 \pmod k}$. Un point est dit de période k s'il appartient à un k-cycle.

Les machines de Turing que nous considérons ont un ruban et une tête de lecture-écriture, et travaillent sur un alphabet fini A contenant un «symbole blanc» B. Leur ensemble d'état (fini) est noté Q. L'espace des configurations de ruban valides est le sous-ensemble $R \subset A^{\mathbb{Z}}$ des configurations pour lesquelles les cellules non blanches sont en nombre fini. Une configuration de la machine est un élément de $C = R \times Q \times \mathbb{Z}$ (la troisième composante indique la position de la tête de lecture sur le ruban). On définit de la manière habituelle une fonction de transition $T : C \to C$.

Un automate cellulaire unidimensionnel est défini par un espace d'états fini Q et une fonction de transition locale $\delta : Q^3 \to Q$. Son espace de configurations est $C = Q^{\mathbb{Z}}$, et sa fonction de transition globale $T : C \to C$ est définie par :

$$(T(x))_i = \delta(x_{i-1}, x_i, x_{i+1}).$$

L'ensemble C est muni de la topologie de Cantor standard [38], c'est à dire de la topologie produit de la topologie discrète sur Q. Intuitivement, ceci signifie que deux configurations c et c' sont proches l'une de l'autre si elle coïncident sur un large segment autour de l'origine (c'est-à-dire qu'il existe un «grand» entier r tel que $c'(i) = c(i)$ pour $i \in [-r, r]$). Il est bien connu que T est continue pour cette topologie, et que C est homéomorphe à l'ensemble de Cantor habituel de l'intervalle unité.

Dans ce chapitre nous ne considérons que des simulations en temps réel, qui sont définies précisément de la façon suivante.

Définition 5.1 *Soit T la fonction de transition d'une machine $\mathcal{M}$ (qui peut être une machine de Turing, un automate cellulaire, un automate à pile, etc). Une fonction $f : I^d \to I^d$ simule $\mathcal{M}$ s'il existe un sous-ensemble D de I^d stable par f et une fonction bijective $\phi : C \to D$ telle que*

$$T = \phi^{-1} \circ f \circ \phi.$$

Intuitivement, ceci signifie que pour appliquer T sur une configuration, on peut encoder cette configuration avec ϕ, appliquer f, puis décoder le résultat avec ϕ^{-1}. Dans le cas des automates cellulaires, nous ferons de plus l'hypothèse que ϕ est un homéomorphisme (comme C est compact, ceci a lieu si et seulement ϕ est continue). Autrement dit, f simule un automate cellulaire s'il existe une conjugaison entre $f_{|D}$ and T. La conjugaison est un outil naturel pour étudier les propriétés dynamiques des automates cellulaires et des réseaux neuronaux (voir par exemple [9]).

On dit habituellement qu'une machine $\mathcal{M}_1$ simule une machine $\mathcal{M}_2$ en temps réel (respectivement, temps linéaire, temps quadratique, ...) si un calcul de $\mathcal{M}_2$ pendant t unités de temps peut être réalisé par $\mathcal{M}_1$ en temps t (respectivement, en temps $O(t)$, $O(t^2)$,...). Puisque la classe des fonctions affines par morceaux (ou analytiques par morceaux, ou monotones par morceaux) est fermée par composition, l'existence de simulations en temps linéaire est équivalente à l'existence de simulations en temps réel.

5.2 Universalité en dimension 2

Dans cette section, nous montrons que les fonctions affines par morceaux en dimension 2 sont universelles.

Théorème 5.1 *Toute machine de Turing peut être simulée en temps linéaire par une fonction de PL_2.*

Preuve. Nous allons montrer comment simuler des automates a deux piles binaires sur l'alphabet $\{1, 3\}$. Il est bien connu que ces automates peuvent simuler les machines de Turing en temps linéaire [23]. L'état d'un automate $\mathcal{M}$ et le contenu de ses piles sont codés par la représentation en base 4 d'un point $(x_1, x_2) \in I^2$. Les p_1 premiers chiffres de x_1 et les p_2 premiers chiffres de x_2 peuvent coder l'état de $\mathcal{M}$ si cet automate a moins de $2^{p_1+p_2}$ états (on pourrait prendre $p_2 = 0$ mais cette forme est plus symétrique). Dans la suite, nous supposons que l'ensemble des états de $\mathcal{M}$ est $Q = \{1, 3\}^{p_1} \times \{1, 3\}^{p_2}$. Un état $(q_{i,j})_{1 \leq j \leq p_i}$ et une pile $(s_{i,j})_{1 \leq j \leq n_i}$ ($i = 1, 2$ et $s_{i,j} \in \{1, 3\}$) sont donc codés par le nombre réel

$$x_i = \sum_{j=1}^{p_i} \frac{q_{i,j}}{4^j} + \sum_{j=1}^{n_i} \frac{s_{i,j}}{4^{p+j}}.$$

Remarquons que x_i appartient à une sorte d'ensemble de Cantor (utilisé auparavant dans [44]), puisque l'utilisation des chiffres 0 et 2 est interdite. Dans la suite, $\overline{0.a_1 a_2 \cdots a_n}$ représente un développement en base 4. Sauf précision contraire, $a_i \in \{1, 3\}$.

La fonction f qui simule $\mathcal{M}$ est affine sur chacun des $3|Q|$ produits $I_{1,l_1} \times I_{2,l_2}$ avec

$$I_{i,l_i} = [l_i, l_i + 1/4^{p_i+1}[\text{ et } l_i = \overline{0.q_{i,1} \cdots q_{i,p_i} s_{i,1}}$$

ou

$$I_{i,l_i} = \{l_i\} \text{ et } l_i = \overline{0.q_{i,1} \cdots q_{i,p_i}}.$$

La pile est non vide dans le premier cas, vide dans le second. Pour $x_i \in I_{i,l_i}$, on note $\Delta x_i = x_i - l_i$. Supposons que (x_1, x_2) encode l'état et les piles de $\mathcal{M}$ à l'instant t. Soit (q'_1, q'_2) l'état suivant de $\mathcal{M}$ (déterminé uniquement par l'état actuel (q_1, q_2) et les lettres de sommet de pile $s_{i,1}$ et $s_{i,2}$). Sur $I_{1,l_1} \times I_{2,l_2}$, f est telle que $f(x_1, x_2) = (x'_1, x'_2)$ avec

$$x'_i = \overline{0.q'_{i,1} \cdots q'_{i,p_i}} + \Delta x'_i$$

et $\Delta x'_i$ défini de la manière suivante :

- $\Delta x'_i = 4\Delta x_i$ si la pile numéro i est dépilée;
- $\Delta x'_i = \dfrac{s_{i,1}}{4^{p_i+1}} + \Delta x_i$ si la pile numéro i est inchangée;
- $\Delta x'_i = \dfrac{a_i}{4^{p_i+1}} + \dfrac{s_{i,1}}{4^{p_i+2}} + \dfrac{\Delta x_i}{4}$ si a_i est empilé sur la pile numéro i.

Ces deux dernières opérations peuvent être appliquées à des piles vides, avec la convention $s_{i,1} = 0$. Il est clair que (x'_1, x'_2) encode l'état et les piles de $\mathcal{M}$ au temps $t + 1$. f est affine par morceaux puisque les opérations appliquées aux piles sont les mêmes pour tous les points d'un produit $I_{1,l_1} \times I_{2,l_2}$.

Pour achever la preuve, il suffit d'étendre f à I^2 tout entier, c'est à dire de l'étendre en dehors de

$$\mathcal{C} = \bigcup_{l_1,l_2} (I_{1,l_1} \times I_{2,l_2}) .$$

Cette extension ne peut pas interférer avec la simulation de $\mathcal{M}$, puisque seuls les points de $\mathcal{C}$ sont utilisés pour cela. Il existe des extensions de f continues et affines par morceaux, puisque la distance entre deux produits distincts est strictement positive. En effet, cette distance (en norme L^∞) est bornée inférieurement par $\max(1/4^{p_1+1}, 1/4^{p_2+1})$. $\square$

La simulation que nous venons de décrire est naïve en ce sens que nous n'avons pas essayé de suivre précisément la définition 5.1. Ceci peut être fait avec quelques modifications que nous présentons brièvement ci-dessous. On simule habituellement une machine de Turing $\mathcal{T}$ par un automate à piles en encodant la partie du ruban située à gauche de la tête de lecture dans la première pile, et la partie située à droite dans la deuxième pile. Le problème de ce codage est que $\phi(c)$ reste inchangé si la tête de lecture et le contenu du ruban de la configuration c sont soumis à une translation. Ceci n'est pas acceptable puisque ϕ est sensée être injective. Prenons une machine de Turing sur l'alphabet $\{1, 3\}$ avec le symbole blanc $B \equiv 1$. L'idée consiste à marquer la case 0 pour la différencier des autres cases du ruban. On pourrait par exemple travailler avec une machine à deux rubans, et laisser la tête de lecture du deuxième ruban sur la case 0. Nous avons en fait choisi de coder une configuration τ_1 du ruban par une nouvelle configuration τ_2 telle que $\tau_1(i)$ est codé par les 2 symboles $\tau_2(2i)\tau_2(2i+1)$. On peut par exemple coder 1 par 11, 3 by 33 si $i \neq 0$, et 1 par 13, 3 par 31 si $i = 0$. Une configuration $c \in C$ de la machine pour laquelle la tête de lecture se trouve sur la case h est donc codée par le point $\phi(c) = (x_1, x_2)$ avec :

$$x_i = q_i + y_i \tag{5.1}$$

$$q_i = \sum_{j=1}^{p_i} \frac{q_{i,j}}{4^j} \tag{5.2}$$

$$y_1 = \sum_{j=1}^{+\infty} \frac{\tau_2(2h - j)}{4^{j+p_1}} \tag{5.3}$$

$$y_2 = \sum_{j=1}^{+\infty} \frac{\tau_2(2h+j-1)}{4^{j+p_2}} \tag{5.4}$$

($q_{i,j}$ et p_i sont définis dans la preuve du théorème 5.1). Ce codage est bien injectif, et conduit à une simulation très proche de celle qui est décrite dans la preuve du théorème.

Dans cette preuve on aurait pu utilisé en codage tout simple en base deux, mais f n'aurait pas été continue. Elle aurait pu être continue avec un codage unaire, mais dans ce cas la simulation n'aurait pas pu se faire en temps linéaire (voir le réseau de neurones universel de [43]).

On peut estimer simplement la complexité de la construction (c'est-à-dire le nombre S de morceaux polygonaux sur lesquels f est affine) de la manière suivante. Avec q états et l lettres ($l = 2$ jusqu'à présent), le nombre de produits de la forme $I_{1,l_1} \times I_{2,l_2}$ est $q(l+1)$. Encodons l'état uniquement dans la première variable, en prenant $p_2 = 0$. L'espace vide rectangulaire entre deux produits consécutifs peut être divisé en deux triangles, et chaque triangle peut être «rempli» par une fonction affine. On se retrouve donc avec $S = 3q(l+1)$. Dans la preuve du théorème 5.1, on peut en fait utiliser un alphabet à plus de deux lettres (pour $l = 3$ par exemple, on construira un codage en base 6 utilisant uniquement les chiffres 1, 3 et 5). Afin de supprimer le test de pile vide, on peu travailler avec des piles de profondeur infinie (telles que celles qui sont définies par les équations (5.3) et (5.4)). La complexité est donc ramenée à $S = 3ql$, et de plus on peut simuler une machine de Turing sans augmenter le nombre d'états ou de symboles. Minsky a construit une machine de Turing universelle ayant 7 états et 4 lettres [31], ce qui donne $S = 84$. En 1982, Rogozhin [39] a construit une machine universelle encore plus petite : elle n'a que 4 états et 6 lettres. Ceci donne $S = 72$.

A partir de notre construction on peut donner une nouvelle preuve du résultat de Siegelmann & Sontag sur l'existence de réseaux universels. L'idée est simplement de construire un réseau qui calcule notre fonction universelle $f : I^2 \to I^2$. Ceci peut être fait facilement car on a besoin de calculer f sur $\mathcal{C}$ uniquement. Soit $\mathcal{R}$ l'ensemble des produits de la forme $I_{1,l_1} \times I_{2,l_2}$. La fonction de transition est donnée par la formule

$$x'_i = \sigma \left[\sum_{\mathcal{P} \in \mathcal{R}} \sigma(a_{i,\mathcal{P}} x_i + b_{i,\mathcal{P}} + \theta_{\mathcal{P}}(x_1, x_2)) \right],$$

où σ est la fonction linéaire seuillée. Sur chaque produit $\mathcal{P}$, $x_i \mapsto a_{i,\mathcal{P}} x_i + b_{i,\mathcal{P}}$ est la fonction affine définie dans la preuve du théorème. La fonction $\theta_{\mathcal{P}}$ est définie de la façon suivante :

- si $(x_1, x_2) \in \mathcal{P}$, $\theta_{\mathcal{P}}(x_1, x_2) = 0$;
- si (x_1, x_2) appartient à un autre produit $\mathcal{P}'$, $\theta_P(x_1, x_2)$ est une valeur négative suffisamment grande pour garantir que $\sigma(a_{i,\mathcal{P}} x + b_{i,\mathcal{P}} + \theta_{\mathcal{P}}(x_1, x_2)) = 0$.

Ceci peut être fait en utilisant σ comme une fonction à seuil (c'est à dire que les entrées de σ doivent toujours être plus petite que 0 ou plus grande que 1), puisque la distance entre deux produits de $\mathcal{R}$ est strictement positive.

Comme on l'a indiquer au début de ce chapitre, la construction de Siegelmann & Sontag peut être vue comme un résultat d'universalité pour les fonctions affines par morceaux en dimension $d = 1058$. Nous avons montré dans cette section que d peut être réduit de 1058 à 2. Il est peut-être intéressant de remarquer que d peut-être réduit très simplement de 1058 à 92 de la manière suivante. Le réseau universel de [44] est composé de 4 couches; les sorties de chaque couche intermédiaire sont reliées aux entrées de la couche suivante, et les sorties de la dernière couche sont reliées aux entrées de la première. Puisque la configuration de la machine simulée par le réseau est encodée dans la première couche, composée de 92 neurones, une fonction universelle en dimension 92 peut être obtenue par composition des fonctions calculées par chacune des couches.

5.3 Simulation d'automates cellulaires

Dans cette section nous montrons que contrairement aux machines de Turing, les automates cellulaires ne peuvent pas être simulés par des fonctions affines par morceaux en dimension deux (nous conjecturons qu'il en va de même en dimension supérieure). Ce résultat est basé sur une hypothèse supplémentaire : La fonction d'encodage est supposée continue. Cette différence entre automates cellulaires n'est pas un simple artefact dû à l'hypothèse de continuité : comme on le montre à la fin de la section, le codage défini par les équations (5.1)–(5.4) est en fait un homéomorphisme.

Quelques notations : les configurations d'un automate cellulaires sont identifiées à des mots bi-infinis sur l'alphabet Q; uv dénote la concaténation des mots (finis ou infinis) u et v quand elle existe; $0^{-\infty}$ est le mot $(u_i)_{i\leq 0}$ tel que $\forall i, u_i = 0$; $Q^{+\infty}$ est l'ensemble des mots infinis à droite, de la forme $(u_i)_{i>0}$.

Théorème 5.2 *Il existe des automates cellulaires qui ne peuvent pas être simulés par une fonction de PL_2 avec une fonction d'encodage continue.*

Preuve. Soit $\mathcal{A}$ un automate cellulaire avec deux états «stables» distingués (0 et 1), tels que $\forall(q_l, q) \in \{0,1\}^2, \forall q_r \in Q, \delta(q_l, q, q_r) = q$. Les autres états sont dits «non-stables». Soit $f \in PL_2$ une fonction (hypothétique) capable de simuler $\mathcal{A}$.

Supposons d'abord que

$$\exists i, k \in \mathbb{N}, \phi(0^{-\infty}1^k0^{\infty}) \in \overset{\circ}{P_i} \; . \tag{5.5}$$

Puisque ϕ est continue, $\exists l \in \mathbb{N}, \forall x \in Q^{+\infty}, \phi(0^{-\infty}1^k0^l x) \in P_i$. Dans la suite de cette preuve, on note : $a = 0^{-\infty}1^k0^l$. Comme $f_i \circ \phi(ax) = \phi \circ T(ax)$ et $T(ax) \in aQ^{+\infty}$ (par définition des états stables), $\phi(aQ^{+\infty})$ est stable par f_i. Par conséquent

$$\forall n \geq 0, \forall x \in Q^{+\infty}, T^n(ax) = \phi^{-1} \circ f_i^n \circ \phi(ax).$$

La suite $(f_i^n(\phi(ax))_n$ a un nombre fini de points d'accumulation car f_i est affine. ϕ^{-1} is continue, donc $(T^n(ax))_n$ a également un nombre fini de points d'accumulation.

Supposons maintenant que (5.5) n'ait pas lieu. Puisque ϕ est injective, il existe nécessairement une arête E ayant au moins 3 antécédents distincts de la forme $0^{-\infty}1^p0^{+\infty}$ par ϕ. Au moins l'une de ces configurations (disons $0^{-\infty}1^k0^{+\infty}$) n'est pas envoyée par ϕ sur une extrémité de E. Si E est l'arête d'un unique polygone P_i (c'est-à-dire que E se trouve sur l'un des côtés de I^2), $\phi(aQ^{+\infty}) \subset P_i$ pour l suffisamment grand. Il découle du même argument que dans le premier cas que toute suite d'itérés $(T^n(ax))_n$ a un nombre fini de points d'accumulation. Considérons finalement le cas $E = P_i \cap P_j$ avec $i{\neq}j$. Par continuité, $\exists l, \phi(aQ^{+\infty}) \in P_i \cup P_j$, et $\phi(aQ^{+\infty})$ est stable par f. E est invariant par

f_i et f_j puisqu'il y a plus de deux points fixes sur E. Soit H_i (respectivement, H_j) le demi-plan délimité par E contenant P_i (respectivement, P_j). Etudions par exemple le cas d'une configuration $ax \in \phi^{-1}(P_i)$. Il faut distinguer plusieurs possibilités.

1. $f_i(H_i) \subset H_i$. Dans ce cas, $T^n(ax) = \phi^{-1} \circ f_i^n \circ \phi(ax)$.
2. $f_i(H_i) = H_j$: deux sous-cas se présentent.
   (a) $f_j(H_j) \subset H_j$, et $T^{n+1}(ax) = \phi^{-1} \circ f_j^n \circ f_i \circ \phi(ax)$.
   (b) $f_j(H_j) = H_i$, et $T^{2n}(ax) = \phi^{-1} \circ (f_j \circ f_i)^n \circ \phi(ax)$.

La fonction $f_j \circ f_i$ est également affine, donc dans tous les cas $T^n(ax)$ a un nombre fini de points d'accumulation. Soit $\mathcal{A}$ un automate cellulaire tel qu'il existe $x_0 \in Q^{+\infty}$ tel que $\forall k, l, T^n(0^{-\infty}1^k0^lx_0)$ a un nombre infini de points d'accumulation (par exemple, le décalage à gauche sur les états non-stables : $\delta(q_l, q, q_r) = q_r$ si $q \notin \{0, 1\}$). $\mathcal{A}$ ne peut pas être simulé par une fonction de PL_2. $\square$

L'image par la fonction de transition globale d'une configuration dans laquelle N cellules sont actives peut être calculée en temps $O(N)$ par une machine de Turing. Il découle donc du théorème 5.1 qu'un automate cellulaire travaillant sur une configuration initiale finie peut être simulé en temps quadratique par une fonction de PL_2 (on dit qu'une configuration est finie si l'ensemble des cellules qui ne sont pas dans état stable est fini). La définition 5.1 est donc peut-être plus pertinente pour des automates cellulaires travaillant sur des configurations infinies. Ces automates peuvent être considérés comme des modèles de machines massivement parallèles, et sont strictement plus puissants que les machines de Turing puisqu'ils peuvent travailler sur des entrées infinies (par exemple des représentations binaires de nombres réels [19]). Pour des automates travaillant sur des configurations finies, le théorème 5.2 doit être plutôt considéré comme un résultat de complexité que de calculabilité : il montre que les automates cellulaires cellulaires ne peuvent pas être simulés en temps réel par des fonctions de PL_2. Il n'est pas très surprenant que les simulations d'automates cellulaires soient plus lentes que les simulations de machines de Turing, puisque le premier modèle est parallèle alors que le deuxième est séquentiel. Remarquons que le théorème 5.2 s'applique effectivement même si on ne considère que des configurations finies. Pour montrer ceci, il suffit de remplacer le décalage à gauche utilisé à la fin de la preuve par un automate cellulaire admettant une trajectoire de configurations finies ayant un nombre infini de points d'accumulation. On peut prendre par exemple une machine de Turing énumérant toutes les suites finies sur un alphabet binaire, et la simuler par un automate cellulaire.

Montrons maintenant que les équations (5.1)–(5.4) définissent bien un homéomorphisme. Pour cela il faut d'abord définir une topologie sur l'ensemble des configurations d'une machine de Turing $\mathcal{T}$. Nous considérons $\mathcal{T}$ comme un automate cellulaire ayant pour ensemble d'états :

$$Q' = \{1, 3\} \cup (\{1, 3\} \times Q).$$

L'espace C' des configurations de $\mathcal{T}$ est un sous-ensemble strict de l'espace C des configurations d'un automate cellulaire habituel : pour toute configuration, une cellule et une seule doit se trouver dans un état de la forme (a, q). Si $c(i) = (a, q)$, ceci signifie que la tête de lecture de $\mathcal{T}$ se trouve sur la case i, et que $\mathcal{T}$ est dans l'état q. Dans le cas contraire, $c(i) \in \{1,3\}$. Soient $c, c' \in C'$ deux configurations telles que $c(h) = (a,q)$ et $c'(i) = c(i)$ si $i \in [-|h|-r, |h|+r]$, pour un certain $r \geq 0$. Posons $\phi(c') = (x'_1, x'_2)$. On a alors $|x'_1 - x_1| \leq \frac{1}{4^{p_1+2r}}$ et $|x'_2 - x_2| \leq \frac{1}{4^{p_2+2r+2}}$. L'encodage ϕ est donc continu en c puisque $||\phi(c') - \phi(c)||$ peut être rendu aussi petit que voulu en prenant r suffisamment grand. Réciproquement, pour un (x_1, x_2) donné, posons $c = \phi^{-1}(x_1, x_2)$ et $c(h) = (a,q)$. Si $|x'_1 - x_1| \leq \frac{1}{4^{p_1+2r}}$ et $|x'_2 - x_2| \leq \frac{1}{4^{p_2+2r+2}}$, alors $c'(i) = c(i)$ pour $i \in [h-r, h+r]$. Donc ϕ^{-1} est également continue.

5.4 Universalité en dimension un

Le principal résultat de cette section est que les machines de Turing ne peuvent pas être simulées par des fonctions affines par morceaux en dimension 1 au sens de la définition 5.1 (on peut par contre simuler les automates à une pile grâce à une adaptation immédiate du théorème 5.1). Nous montrons par contre que des fonctions plus complexes sont universelles. Les simulations d'automates cellulaires sont à nouveau comparées aux simulations de machines de Turing.

Théorème 5.3 *Il existe des machines de Turing qui ne peuvent pas être simulées par des fonctions de PL_1.*

La preuve de ce résultat est basée sur le lemme 5.1, qui fournit une propriété dynamique différenciant PL_1 de PL_2 (et de la classe de toutes les fonctions continues sur I).

Preuve du théorème 5.3. Soit $\mathcal{T}$ une machine de Turing ayant un nombre infini de cycles de longueur k pour un nombre infini de valeurs de k. Supposons que $\mathcal{T}$ puisse être simulé par $f : I \to I$. Il découle immédiatement de la définition 5.1 que des cycles de $\mathcal{T}$ distincts et de longueur k sont envoyés par ϕ sur des cycles de f distincts et de longueur k. D'après le lemme 5.1, f ne peut donc pas être affine par morceaux.

Une machine de Turing $\mathcal{T}$ répondant aux conditions voulues peut être facilement construite de la manière suivante. En partant d'une cellule blanche, la tête de lecture se déplace vers la droite jusqu'à arriver sur une autre cellule blanche. A ce moment elle repart vers la gauche jusqu'à la case blanche de départ, puis repart vers la droite, et ainsi de suite. S'il y a $k-2$ cases entre deux blancs consécutifs, $\mathcal{T}$ a donc un cycle de longueur k . Elle a en fait un nombre infini de cycles de longueur k (a condition que $|A| \geq 2$), puisque toutes les cellules du ruban qui ne sont pas situées entre les deux blancs peuvent contenir un symbole de A quelconque. □

Lemme 5.1 *Une fonction de PL_1 ne peut pas avoir un nombre infini de cycles de longueur k pour une infinité de valeurs de k.*

Preuve. Ce résultat est une conséquence très simple du lemme 5.2. Si ce lemme s'applique pour une infinité de valeurs de k, alors d'après le principe des tiroirs il existe deux intervalles L_{k_1} et L_{k_2} tels que $L_{k_1} \subset L_{k_2}$. Ceci est impossible par définition des L_k. □

Lemme 5.2 *Soit $f \in PL_1$ une fonction ayant une infinité de cycles de longueur k. Il existe un intervalle L_k de la forme $L_k = [c, c_i]$ ou $L_k = [c_i, c]$ tel que :*

(a) ${f^k}_{|L_k} = Id$;

(b) *Pour tout $k' < k$ et tout sous-intervalle $L' \subset L_k$, $f^{k'}{}_{|L'} \neq Id$.*

Preuve. L'ensemble F des points de période k de f a au moins un point d'accumulation x_∞. Par définition $f^k(x) = x$ pour $x \in F$, et f^k est affine par morceaux. Il existe donc un intervalle J d'extrémité x_∞ tel que $f^k{}_{|J} = \mathrm{Id}$, et $|F \cap J| = +\infty$. Soit $x_0 \in F \cap J$ un point tel que l'orbite de x_0 ne contienne aucun des c_i (un tel point existe puisque les c_i sont en nombre fini, et que chaque c_i ne peut pas appartenir à l'orbite de plus de k points de période k). Soit $(\alpha_j)_{0 \leq j \leq k-1}$ la suite telle que $f^j(x_0) \in I_{\alpha_j}$. Les fonctions affines $f_{|I_{\alpha_0}}, \ldots, f_{|I_{\alpha_{k-1}}}$ sont telles que $f_{|I_{\alpha_{k-1}}} \circ f_{|I_{\alpha_{k-2}}} \ldots \circ f_{|I_{\alpha_0}} = \mathrm{Id}$. Par conséquent pour tout intervalle K vérifiant

$$\forall j, 0 \leq j \leq k-1, f^j(K) \subset I_{\alpha_j} \tag{5.6}$$

on a $f^k{}_{|K} = \mathrm{Id}$. Par continuité, l'équation (5.6) est satisfaite sur $K = [x_0 - \epsilon, x_0 + \epsilon]$ pour un ϵ suffisamment petit. Soit K un intervalle maximal tel que (5.6) soit vérifiée et $x_0 \in K$. Il existe un entier l tel que c_{α_l} soit une extrémité de l'intervalle $f^l(K)$ (dans le cas contraire, K pourrait être étendu sans violer les contraintes $f^j(K) \subset I_{\alpha_j}$). Posons $L_k = f^l(K)$. La condition (a) est clairement vérifiée. Pour $k' < k$ et $x \in L_k$,

$$f^{k'}(x) = f_{|I_{\alpha_{k'+l-1}}} \circ f_{|I_{\alpha_{k'+l-2}}} \ldots f_{|I_{\alpha_{l+1}}} \circ f_{|I_{\alpha_l}}(x)$$

(les indices sont calculés modulo k). Le membre droit de cette équation ne peut pas être égal à Id sur un intervalle $L' \subset L_k$, sinon il serait égal à Id sur L_k tout entier, puisque les fonctions $f_{|I_j}$ sont affines. Ceci est impossible puisque x_0 serait un point de période au plus k'. □

Remarquons que ce résultat s'applique aussi à des fonctions affines par morceaux discontinues, et ses deux conséquences (le lemme 5.1 et le théorème 5.3) également. Il en va de même pour les fonctions analytiques par morceaux : L'argument d'«extension d'intervalle» fonctionne encore grâce à la propriété des zéros isolés.

Théorème 5.4 *Une machine de Turing quelconque peut être simulée en temps linéaire par une fonction dénombrablement affine par morceaux.*

Preuve. Soit $\mathcal{T}$ une machine de Turing, et soit q, h, r et s des entiers codant respectivement l'état de $\mathcal{T}$, la position de sa tête de lecture, les parties du ruban situées à gauche et à droite de sa tête de lecture. Une configuration de $\mathcal{T}$ est codée par le nombre rationnel $x_{q,h,r,s} = 1/(2^q 3^h 5^r 7^s)$. Puisque tous ces nombres sont distincts, ils forment une suite décroissante $(y_i)_{i \in \mathbb{N}}$. Soit f une fonction affine sur $[y_0, 1]$, sur chaque intervalle $[y_{i+1}, y_i]$, et envoyant chaque y_i sur le codage de la «configuration suivante». Par définition f simule $\mathcal{T}$ et est continue et dénombrablement affine par morceaux sur $]0, 1]$. Cette fonction peut être prolongée par continuité en 0 en posant $f(0) = 0$, puisque l'image d'une configuration ayant une «grande» valeur de h, r ou s a également une grande valeur de h, r ou s. □

Pour les automates cellulaires, la situation est à nouveau complètement différente.

Théorème 5.5 *Il existe des automates cellulaires qui ne peuvent pas être simulés avec un encodage continu par une fonction dénombrablement monotone par morceaux.*

Preuve. Soit f une fonction dénombrablement monotone par morceaux simulant un automate cellulaire avec deux états stables 0 and 1, comme dans la preuve du théorème 5.2. L'ensemble des configurations pour lesquelles certaines cellules ne sont pas dans un état stable est non dénombrable donc l'une de ces configurations est envoyée par ϕ dans un intervalle ouvert J sur lequel f est monotone. Une demi-ligne d'automates peut donc être simulée en itérant la fonction monotone f_J (même argument que pour le théorème 5.2). Ceci est impossible pour certains automates cellulaires, car avec une fonction monotone toute trajectoire a au plus un point d'accumulation. □

Pour les automates cellulaires le seul résultat positif que nous connaissions est qu'ils peuvent être simulés par des fonctions continues sur I. En effet, l'espace des configurations C est homéomorphe à l'ensemble de Cantor standard K, comme nous l'avons remarqué à la fin de la section 5.1. Un automate cellulaire peut donc être simulé par une fonction continue sur K. Mais K est compact, donc cette fonction peut être étendue en une fonction continue sur tout I, d'où le résultat.

Théorème 5.6 *Une machine de Turing quelconque peut être simulée en temps linéaire par une fonction continue et monotone par morceaux.*

Preuve. Mous allons montrer comment simuler une machine de Turing $\mathcal{T}$ travaillant sur un alphabet A muni d'un ordre total. On peut supposer sans perte de généralité que le caractère blanc B en est le plus petit élément, et que $\mathcal{T}$ n'écrit jamais cette lettre sur le ruban. Toutes les configurations pour lesquelles $\mathcal{T}$ est dans un état q donné et sa tête de lecture lit une lettre l donnée sont codées par un point du même intervalle $I_{q,l} = [a_{q,l}, b_{q,l}]$. Tous ces intervalles sont disjoints, et $I_{q,l} < I_{q,l'}$ (c'est-à-dire que $b_{q,l} < a_{q,l'}$) si $l < l'$. On va montrer que $\mathcal{T}$ peut être simulée par une fonction f croissante sur chaque $I_{q,l}$. Une extension monotone par morceaux de f sur tout I peut être obtenue en «bouchant les trous» entre les $I_{q,l}$.

Notre premier objectif est de construire pour chaque $I_{q,l}$ un ordre total sur les configurations de manière que si c et c' sont codées par des points du même intervalle $I_{q,l}$, alors $c < c' \Rightarrow T(c) < T(c')$. Pour un encodage ϕ, f simule $\mathcal{T}$ si et seulement si

$$f(\phi(c)) = \phi(T(c)). \tag{5.7}$$

Donc si ϕ préserve l'ordre sur les configurations, la fonction f définie par l'équation (5.7) sera croissante sur $I_{q,l} \cap \phi(C)$. Si c and c' sont respectivement encodés dans deux intervalles différents $I_{q,l}$ et $I_{q',l'}$, alors $c < c' \iff I_{q,l} < I_{q,l'}$. Construisons maintenant un tel ordre sur un ensemble de configurations de la forme $\phi^{-1}(I_{q,l})$. Soit $l(t)$ la lettre lue par $\mathcal{T}$ au temps t, et c la configuration initiale (donc par définition $l(0) = l$). Etant données deux configurations distinctes c et c', posons

$$t_C = \min\{t \geq 0; l(t) \neq l'(t)\}$$

($t_C \geq 1$ par définition des $I_{q,l}$). Deux cas peuvent être distingués.

1. $t_C < +\infty$. Dans ce cas, $c < c'$ si $l(t_C) < l'(t_C)$, et $c' < c$ sinon.

2. $t_C = +\infty$. Soient h et h' les positions des têtes de lectures dans c et c'.

   (a) c et c' sont ordonnées suivant l'ordre lexicographique $\prec$ sur le ruban. L'ordre lexicographique est défini par l'ordre suivant sur les cases du ruban :

   $$h > h+1 > \ldots > h+n > h+n+1 > \ldots > h-1 > \ldots > h-n > \ldots$$

Autrement dit, les cases situées à droite de la tête de lecture sont plus significatives pour $\prec$ que celles qui sont situées à gauche, et une case est d'autant plus significative qu'elle est plus proche de la tête de lecture.

(b) Si les configurations ne peuvent être départagées de cette manière (on note alors $c \simeq c'$), $c < c'$ si et seulement si $h < h'$.

Afin de montrer qu'un ordre total vient bient d'être défini, la seule propriété non triviale à vérifier est la transitivité. Soient $c, c', c'' \in \phi^{-1}(I_{q,l})$ trois configurations telles que $c < c'$ et $c' < c''$. Nous allons utiliser une notation particulière : le point 1–2 par exemple traite du cas où $c < c'$ par application du point 1 ci-dessus, et $c' < c''$ par application du point 2.

1–1. Soit $t'_C = \min\{t \geq 0; l'(t) \neq l''(t)\}$. Si $t'_C < t_C$, alors $l(t) = l''(t)$ pour $t < t'_C$, et $l(t'_C) = l'(t'_C) < l''(t'_C)$. Si $t'_C \geq t_C$, alors $l(t) = l''(t)$ pour $t < t_C$, et $l(t_C) < l'(t_C) \leq l''(t_C)$. Dans les deux cas, $c < c''$.

1–2. $l''(t) = l'(t) = l(t)$ pour $t < t_C$ et $l''(t_C) = l'(t_C) > l(t_C)$, donc $c'' > c$.

2–1. Même argument que pour 1–2.

2(a)–2(a). La relation $\prec$ est transitive puisque c'est une relation d'ordre.

2(a)–2(b). $l(t) = l'(t) = l''(t)$ pour $t \geq 0$ et $c \prec c' \simeq c''$, donc $c \prec c''$ et $c < c''$.

2(b)–2(a). Même argument que pour 2(a)–2(b).

2(b)–2(b). $l(t) = l'(t) = l''(t)$ pour $t \geq 0$, $c \simeq c' \simeq c''$ et $h < h' < h''$, donc $c < c''$.

Montrons maintenant que $c < c' \Rightarrow T(c) < T(c')$ pour $c, c' \in \phi^{-1}(I_{q,l})$. On considère à nouveau les trois cas 1, 2(a) et 2(b).

1. Si $t_C > 1$ (c'est-à-dire que l(1)=l'(1)), $T(c)$ et $T'(c)$ sont encodés dans le même intervalle $I_{q',l(1)}$. L'application de T diminue simplement t_C d'une unité, donc $T(c) < T(c')$. Si $t_C = 1$, alors $l(1) < l'(1)$, $T(c) \in I_{q',l(1)}$ et $T(c') \in I_{q',l'(1)}$. Dans ce cas $T(c) < T(c')$ puisque $I_{q',l(1)} < I_{q',l'(1)}$.
2. (a) Le point 2(a) s'applique également au couple $(T(c), T(c'))$, d'où $T(c) < T(c')$.

   (b) La tête de lecture de $\mathcal{T}$ se déplace dans la même direction pour c ou c', donc le point 2(b) s'applique également au couple $(T(c), T(c'))$. On en déduit que $T(c) < T(c')$.

L'ordre que nous venons de construire ne semble pas à première vue très constructif, puisque pour comparer deux configurations il faut observer le comportement de $\mathcal{T}$ pendant un temps non borné. En fait c'est le contraire qui est vrai, et la position d'une configuration c dans cet ordre peut être caractérisée par deux suites finies calculables $r(c), u(c) \in A^*$. Posons $r'(c) = l(0)l(1)\ldots l(t_B)$, avec t_B le plus petit instant tel que $l(t_B) = B$ ($t_B = +\infty$ éventuellement). La suite $r(c)$ est obtenue à partir de $r'(c)$ en effaçant la lettre $l(t)$ si à l'instant t la tête de lecture se trouve sur une case qu'elle a déjà visitée auparavant. Remarquons que $r(c)$ est bien finie et calculable, même si

$t_B = +\infty$. La suite $u(c)$ est composée des lettres non-blanches écrites sur le ruban à l'instant $t = 0$, selon l'ordre défini par $\prec$. On voit alors que l'ordre $<$ sur C est simplement l'ordre lexicographique sur $(r(c), u(c), h)$, c'est-à-dire que :

$$c < c' \iff \begin{cases} r(c) \prec r(c') \text{ ou} \\ r(c) = r'(c) \text{ et } u(c) \prec u'(c) \text{ ou} \\ r(c) = r'(c) \text{ et } u(c) = u'(c) \text{ et } h < h'. \end{cases}$$

Ici $\prec$ désigne l'ordre lexicographique sur les suites finies. Cette propriété vient du fait que si jamais les cases qui n'ont pas été atteintes par la tête de lecture entre $t = 0$ et $t = t_B$ sont atteintes ultérieurement, les cases les plus significatives pour $\prec$ seront atteintes en premier (mais il est impossible de décider si ces cases seront effectivement atteintes).

Il est maintenant facile de définir une fonction de codage qui préserve l'ordre sur C : soient $x, y > 0$ les rangs de $r(c)$ et $u(c)$ dans la suite des mots de A^* rangés suivant l'ordre lexicographique. On prend

$$\phi(c) = (b_{q,l} - a_{q,l})\theta(c),$$

avec par exemple

$$\theta(c) = 1 - \frac{1}{x - \frac{1}{3y + \tanh(h)}}.$$

Posons $f(\phi(c)) = \phi(T(c))$. Comme on l'a remarqué au début de la preuve, f simule $\mathcal{T}$ et est croissante sur $I_{q,l} \cap \phi(C)$. On peut prolonger f par continuité à tout $I_{q,l}$ puisque les points de $\phi(C)$ sont isolés. $\square$

Remarquons que dans cette preuve on peut en fait construire une fonction f dénombrablement affine par morceaux, ce qui redonne une preuve du théorème 5.4.

5.5 Des résultats négatifs plus généraux ?

Il serait intéressant d'obtenir des résultats négatifs pour une notion d'universalité moins restrictive que celle qui est utilisée dans ce chapitre. Avant de s'attaquer directement à cette question, il pourrait être utile d'étudier le problème suivant, qui est plus précis et peut-être plus simple : peut-on décider, étant donnés une fonction affine par morceaux et un point de départ, si la suite des itérés correspondante atteint un point fixe? Ce problème peut-être formulé précisément de la façon suivante.

Problème 5.1 *Est-il décidable de déterminer, étant donnés $f \in RPL_1$ et $x \in \mathbb{Q}$, s'il existe $t \in \mathbb{N}$ tel que $f^t(x) = f^{t+1}(x)$?*

Si ce problème se révèle indécidable, on pourra certainement obtenir un résultat d'universalité pour RPL_1, puisque la plupart des résultats d'indécidabilité sont basés sur des simulations de machines de Turing (ou d'autres machines universelles). Réciproquement, si ce problème est décidable, il semble peu probable que RPL_1 soit universel, puisque dans ce cas on aurait un algorithme pour résoudre le problème de la halte des machines de Turing (cet argument n'a de sens que si la fin d'un calcul de f est définie par stabilisation sur un point fixe). Démontrer qu'une famille de fonctions n'est pas universelle pour des classes d'encodages et de simulations très générales est probablement un problème difficile (ce type de problèmes est déjà difficile pour des machines discrètes). Pour des modèles discrets, nous proposons les définitions suivantes, qui sont cohérentes avec le point de vue de ce chapitre, inspiré des systèmes dynamiques.

Définition 5.2 *Une machine $\mathcal{M}$ est définie par une fonction récursive* totale $\phi_{\mathcal{M}}$: $\mathbb{N} \to \mathbb{N}$. *La fonction récursive partielle $f : \mathbb{N}^p \to \mathbb{N}^q$ calculée par $\mathcal{M}$ relativement aux fonctions récursives* totales $\phi_E : \mathbb{N}^p \to \mathbb{N}$ *(la fonction d'encodage) et $\phi_D : \mathbb{N} \to \mathbb{N}^q$ (la fonction de décodage) est définie de la manière suivante :*

Pour $n \in \mathbb{N}^p$, $f(n)$ est défini si et seulement si il existe $t \in \mathbb{N}$ tel que

$$\phi^t_{\mathcal{M}} \circ \phi_E(n) = \phi^{t+1}_{\mathcal{M}} \circ \phi_E(n).$$

Dans ce cas,

$$f(n) = \phi_D \circ \phi^t_{\mathcal{M}} \circ \phi_E(n).$$

Intuitivement, $\phi_{\mathcal{M}}$ représente la fonction de transition de $\mathcal{M}$, et $\phi^T_{\mathcal{M}} \circ \phi_E(n)$ représente l'état de $\mathcal{M}$ après T étapes de calculs sur l'entrée n.

Définition 5.3 *Une machine $\mathcal{M}$ est universelle si il existe un choix de ϕ_E et ϕ_D tel que $\mathcal{M}$ calcule une fonction $f : \mathbb{N}^2 \to \mathbb{N}$ universelle.*

Rappelons qu'une fonction $f : \mathbb{N}^2 \to \mathbb{N}$ est universelle s'il existe une numérotation récursive $\phi \mapsto \langle\phi\rangle$ des fonctions récursives partielles $\phi : \mathbb{N} \to \mathbb{N}$ telle que $f(\langle\phi\rangle, n) = \phi(n)$ [50].

La notion de machine universelle ne pose pas de problèmes pour les modèles discrets tels que les machines de Turing ou les machines RAM. Cependant nous n'en avons pas trouvé de définition formelle dans la littérature.

Remarquons qu'il est important de n'autoriser que des fonctions de décodage récursives *totales* si on veut éviter que des machines trop simples soient considérées comme universelles. En effet, si la fonction de décodage peut être une fonction récursive partielle quelconque, toute fonction récursive partielle est calculable en temps réel par une machine linéaire [51].

Nos définitions peuvent être adaptés au cas réellement intéressant des modèles de calcul continus en prenant $\phi_E : \mathbb{N}^p \to \mathbb{R}$, $\phi_{\mathcal{M}} : \mathbb{R} \to \mathbb{R}$, $\phi_D : \mathbb{R} \to \mathbb{N}^q$, et en définissant une notion de récursivité adéquate pour ces fonctions. Ceci peut être fait en utilisant l'une ou l'autre des théories de la récursivité proposées pour des domaines continus (voir par exemple [50] et [6]).

Il pourrait également être nécessaire de modifier la condition d'arrêt par point fixe de la définition 5.2. Cette condition est très naturelle pour les modèles discrets et est bien adaptée aux fonctions de PL_2, ainsi qu'on l'a montré dans la section 5.2. Elle convient également pour les réseaux de neurones récurrents de [44], mais risque de s'avérer trop restrictive dès qu'on ne considère plus des sigmoïdes affines par morceaux. Au chapitre 4 par exemple, l'ensemble d'arrêt H est un hyperplan de l'espace des états. Dans le cas général, on pourrait prendre pour H une union finie de parallépipèdes. Il faut en tout cas se garder d'autoriser des ensembles d'arrêts trop généraux. Par exemple on peut voir facilement que si H peut être un ensemble récursif quelconque, alors n'importe quelle machine gardant en mémoire au cours du calcul le temps écoulé t et l'entrée n est universelle (en effet tout le calcul peut alors être effectué par la fonction de décodage).

Chapitre 6

Itérations de fonctions analytiques réelles

Les simulations de machines de Turing du chapitre 4 ne s'appliquent qu'à une classe de sigmoïdes assez restreinte qui ne contient pas la tangente hyperbolique, ni d'ailleurs aucune fonction analytique (à cause de la propriété des zéros isolés). On peut donc se demander s'il est possible de simuler les machines de Turing en itérant des fonctions analytiques. Dans ce chapitre nous répondons par l'affirmative en décrivant trois simulations différentes. La première d'entre elles est basée sur un théorème d'existence d'analyse complexe, et a pour principal intérêt de montrer que la classe des fonctions analytiques est trop large pour qu'on puisse énoncer des résultats de simulation intéressants. Les deux autres simulations sont constructives et utilisent des fonctions usuelles (composées de polynômes et de fonctions trigonométriques). La deuxième simulation se fait en dimension 1 mais est très lente puisque basée sur une simulation d'automates à piles unaires. La troisième simulation se fait en temps linéaire, mais nécessite une dimension supplémentaire.

Nous ne redonnons pas de définition formelle d'une simulation de machine de Turing, car ce concept est suffisamment détaillé dans les chapitres précédents. Ici encore, il s'agit d'une correspondance entre les états successifs de la machine et les itérées de la fonction réelle. Dans les sections 6.1 et 6.3 une transition de la machine correspond à une application de la fonction comme au chapitre 5, par contre dans la section 6.2 elle correspond à plusieurs applications de cette fonction à cause du passage par les automates à piles unaires, comme dans le chapitre 4.

Il est à noter que toutes ces simulations se font sur $\mathbb{R}^n$. Si on se restreint au compact $[0,1]^n$, le problème est ouvert à notre connaissance.

6.1 Un résultat d'analyse complexe

Etant donné un ensemble A sans point d'accumulation, il existe une fonction holomorphe prenant des valeurs prescrites en chaque point de A. On peut même imposer les valeurs d'un nombre fini de dérivées en chaque point (voir [41], théorème 15.15) :

Théorème 6.1 *Soit Ω un ouvert du plan complexe et A un sous-ensemble de Ω sans point d'accumulation dans Ω. A chaque $\alpha \in A$ on associe un entier non négatif $m(\alpha)$*

et des nombres complexes $w_{n,\alpha}$ où $0 \leq n \leq m(\alpha)$. Il existe une fonction f holomorphe sur Ω telle que

$$f^{(n)}(\alpha) = n!w_{n,\alpha} \quad (\alpha \in A,\ 0 \leq n \leq m(\alpha)).$$

Ainsi énoncé ce résultat ne porte que sur les fonctions d'une variable complexe, mais il s'applique en fait également aux fonctions analytiques réelles.

On peut donc «simuler» n'importe quelle fonction de transition $\delta : C \to C$, récursive ou non, sur un ensemble dénombrable C en itérant une fonction analytique $f : \mathbb{R} \to \mathbb{R}$. Pour cela il suffit de prendre un codage injectif arbitraire $\phi : C \to \mathbb{N}$ puis de choisir f de manière que

$$\forall x \in C, f(\phi(x)) = \phi(\delta(x)).$$

Afin d'obtenir des simulations constructives, nous travaillons désormais avec la classe de fonctions $\mathcal{U}_n$ définie ci-dessous.

Définition 6.1 *$\mathcal{V}_n$ est la plus petite classe de fonctions de $\mathbb{R}^n$ dans $\mathbb{R}$ close par addition, multiplication et composition comprenant les projections $x \mapsto x_i$, les fonctions sinus $x \mapsto \sin x_i$ et les fonctions constantes.*

$\mathcal{U}_n$ est la classe des fonctions de $\mathbb{R}^n$ dans $\mathbb{R}^n$ dont chaque composante appartient à $\mathcal{V}_n$.

6.2 Simulation en dimension un

Le résultat suivant est basé sur une simulation d'automates à piles unaires, comme au chapitre 4.

Théorème 6.2 *Une machine Turing quelconque peut être simulée par une fonction de $\mathcal{U}_1$.*

Preuve. Soit $\mathcal{M}$ un automate à deux piles unaires dont l'ensemble d'états est $Q = \{0, 1, \ldots, S\}$. Une configuration dans laquelle $\mathcal{M}$ est dans l'état $r \in Q$ avec p éléments sur la première pile et q éléments sur la deuxième est codée par l'entier $x = 2^p 3^q 5^r$. On simule $\mathcal{M}$ par une fonction f de la forme $f(x) = a(x)b(x)c(x)x$ où

$$a(x) = 2^{\Delta p} \in \{1/2, 1, 2\}$$

(Δp est la variation de p lors d'une transition de $\mathcal{M}$), et de même

$$b(x) = 3^{\Delta q} \in \{1/3, 1, 3\},\ c(x) = 5^{\Delta r} \in \{5^{-S}, \ldots, 1, \ldots, 5^S\}.$$

Pour calculer le nouvel état de $\mathcal{M}$ et les opérations appliquées aux piles, il suffit de connaître l'état actuel r et de savoir pour chaque pile si elle est vide ou non. On voudrait donc écrire

$$a(x) = \alpha(e1(x), e2(x), s(x)),\ b(x) = \beta(e1(x), e2(x), s(x))$$

et

$$c(x) = \gamma(e1(x), e2(x), s(x)$$

où α, β, γ sont des polynômes, $e1, e2$ indiquent si les piles sont vides, et s caractérise l'état de $\mathcal{M}$. Etant donné un ensemble fini $E \subset \mathbb{R}^3$, on peut toujours trouver un polynôme prenant des valeurs prescrites sur E. Par conséquent on exige que :

– pour chaque état $r \in Q$, $E_r = \{s(x), x = 2^p 3^q 5^r\}$ est fini;

– $r \neq r' \Longrightarrow E_r \cap E_{r'} = \emptyset$.

Les fonctions e_1 and e_2 sont soumises aux mêmes conditions. Toutes ces contraintes sont satisfaites par les fonctions suivantes :

$$e_1(x) = \sin\frac{\pi x}{2}, e_2(x) = \sin\frac{\pi x}{3}, s(x) = \sin\frac{\pi x}{5^S},$$

ce qui donne la fonction f cherchée. En effet, $e_1(x) = \pm 1$ si $p = 0$, et $e_1(x) = 0$ sinon; $e_2(x) = \pm\sin\frac{\pi}{3}$ si $q = 0$, et $e_2(x) = 0$ sinon. Enfin les ensembles E_r sont finis puisque $s(x) \in \{\sin\frac{k\pi}{5^S}; -5^S < k < 5^S\}$. Pour voir qu'ils sont disjoints deux à deux, il suffit de remarquer que l'égalité $\sin t = \sin t'$ n'a lieu que si $t + t' = \pi + 2k\pi$ ou $t - t' = 2k\pi$, et que $\frac{2^p 3^q}{5^{S-r}} \pm \frac{2^{p'} 3^{q'}}{5^{S-r'}}$ ne peut pas être entier si $r \neq r'$. □

6.3 Simulation en dimension deux

Dans cette section on utilise un codage similaire à celui du chapitre 5. La situation est d'un certain côté plus simple : il n'est pas nécessaire d'utiliser un ensemble de Cantor car les configurations ne sont pas codées dans un ensemble borné.

Théorème 6.3 *Une machine de Turing peut être simulée en temps linéaire par une fonction de $f \in \mathcal{U}_2$.*

Preuve. On simule un automate $\mathcal{M}$ à deux piles binaires sur l'alphabet $\{0, 1\}$. Comme on l'a déjà remarqué au chapitre 5, il est possible de considérer que les piles sont de profondeur infinie mais ne contiennent qu'un nombre fini de 1 (ceci évite d'avoir à traiter le cas d'une pile vide). On code le contenu d'une pile par l'entier s_i $(i = 1, 2)$ dont elle est l'écriture binaire (le chiffre des unités de s_i représentant le sommet de pile). Soit $Q = \{0, 1, \ldots, S_1\} \times \{0, 1, \ldots, S_2\}$ l'ensemble d'états de $\mathcal{M}$. Une configuration pour laquelle $\mathcal{M}$ est dans l'état $(q_1, q_2) \in Q$ est codée par le point $x = (x_1, x_2)$ avec $x_i = q_i + 4S_i s_i$. Le nouvel état (q'_1, q'_2) et les nouvelles piles s'_1, s'_2 peuvent être calculés par des fonctions de $\mathcal{U}_2$ de la façon suivante.

Soient t_1 et t_2 les lettres sur les sommets de piles. Il existe des polynômes g_1 et g_2 tels $q'_i = g_i(q_1, q_2, t_1, t_2)$ puisque q'_i dépend uniquement de q_1, q_2, t_1 et t_2. Lors d'une transition s_i est soumis à une transformation affine : $s'_i = 2s_i + t'_i$ si t'_i est empilé, $s'_i = (s_i - t_i)/2$ si t_i est dépilé, et bien sûr $s'_i = s_i$ si la pile est inchangée. Par conséquent il existe des polynômes h_i et l_i tels que

$$s'_i = h_i(q_1, q_2, t_1, t_2)s_i + l_i(q_1, q_2, t_1, t_2).$$

Chaque composante q_i de l'état (q_1, q_2) est une fonction polynômiale de $\sin(\pi x_i/2S_i)$ puisque $\sin(\pi x_i/2S_i) = \sin(\pi q_i/2S_i)$ et que la fonction sinus est injective sur $[0, \frac{\pi}{2}]$. On peut enfin calculer s_i et t_i de la manière suivante :

$$s_i = \frac{x_i - q_i}{4S_i},\ t_i = \frac{1 - \cos\pi s_i}{2}.$$

□

Chapitre 7

Relations entre systèmes dynamiques et circuits booléens

Un nouveau modèle de calcul analogique a été récemment proposé par Hava Siegelmann & Eduardo Sontag ([45], résumé dans [47]). Dans ce modèle les calculs sont effectués en précision infinie par des réseaux de neurones récurrents à poids réels, alors que les poids étaient rationnels dans un article précédent [46]. Siegelmann & Sontag ont montré qu'un langage arbitraire (même non calculable) sur un alphabet fini peut être reconnu en temps exponentiel par leurs réseaux, et que la classe des langages reconnaissables en temps polynômial est exactement la classe P/poly des langages reconnaissables par des circuits booléens de taille polynômiale. Ils ont également montré qu'utiliser des classes de réseaux beaucoup générales, appelés *réseaux de processeurs généralisés* («generalized processor networks»), ne permet pas d'obtenir de gain en puissance de calcul.

Dans ce chapitre nous continuons l'étude de la puissance de calcul des systèmes dynamiques en basses dimensions entreprise au chapitre 5. Nous nous plaçons dans un cadre très semblable à celui de Siegelmann & Sontag : les systèmes dynamiques considérés sont des fonctions itérées sur $[0, 1]^n$; ces fonctions sont définies par un nombre fini de paramètres qui peuvent prendre des valeurs réelles arbitraires, éventuellement non calculables.

On montre dans la section 7.2 que les fonctions affines par morceaux itérées sur l'intervalle peuvent reconnaître un langage arbitraire en temps exponentiel. Rappelons que dans les cas de paramètres rationnels, deux dimensions sont nécessaires pour reconnaître des langages récursifs quelconques par itération de fonctions affines par morceaux (voir le chapitre 5). En dimension deux, la classe des langages reconnaissables est exactement P/poly. On montre ensuite que travailler en dimension plus élevée ou avec d'autres types de fonctions ne permet de reconnaître plus que P/poly en temps polynômial. Ce résultat s'applique pour la classe des fonctions approximables en temps polynômial, qui contient certainement la plupart des fonctions continues importantes en pratique (elle contient par exemple les fonctions affines par morceaux, comme on le montre dans la section 7.1, ainsi que les polynômes en plusieurs variables [45]).

Dans la section 7.3, nous étudions des systèmes en-ligne pour lesquels la donnée du calcul est lue bit par bit, au lieu d'être codée entièrement dans l'état initial. Ce mode de fonctionnement est plus proche de celui des réseaux réseaux récurrents de

Siegelmann & Sontag, et on peut en fait montrer que ces systèmes sont équivalents à leurs réseaux de processeurs généralisés. Nous pensons cependant que notre modèle est plus simple et plus naturel. On montre que les systèmes en-ligne sont des cas particuliers des systèmes hors-ligne de la section 7.2. Leur puissance de calcul est donc soumise à la même limitation.

Finalement nous comparons ces systèmes au modèle BSS de calcul sur les nombres réels (voir le chapitre 2). On montre que ce dernier est au moins aussi puissant. La réciproque est étudiée au chapitre suivant.

Pour tous les modèles de calcul analogiques étudiés dans ce chapitre ainsi que dans [45], on trouve donc que la classe des ensembles reconnaissables est P/poly. Il semble donc raisonnable d'affirmer que «calcul analogique efficace $\equiv$ P/poly».

7.1 Préliminaires

Les définitions et notations utilisées dans ce chapitre sont présentées ci-dessous.

L'intervalle unité est toujours $I = [0,1]$. La $i^{\text{ème}}$ composante de $x \in \mathbb{R}^d$ est notée $e_i(x)$. Pour $x, y \in \mathbb{R}^d$, $|x-y| = ||x-y||_\infty = \sup_{1 \leq i \leq d} |x_i - y_i|$.

L'ensemble des mots finis non vides sur un alphabet L est L^+.

On identifie un nombre naturel à son écriture en base 2, de sorte que $\boldsymbol{N}$ peut être identifié à $\{0,1\}^+$. La longueur de $x \in \{0,1\}^* = \mathbb{N}$ est $|x|$ (à ne pas confondre avec la valeur absolue !).

L'ensemble des nombres «décimaux» en base 2 est noté D : $x \in D_n$ si et seulement si $x = 0.x_1 \ldots x_n$ avec $x_i \in \{0,1\}$, et $D = \cup_{n \geq 1} D_n$.

On dit que les chiffres $x_1 \ldots x_n$ ($x_i \in \{0,1\}$) sont les n premiers chiffres de $x \in I$ si le nombre $y = 0.x_1 \ldots x_n$ est tel que $|x-y| \leq 2^{-n}$ (il n'y a bien sûr pas d'unicité). On note $y = \mathrm{Trunc}_n(x)$. De la même manière, on définit $\mathrm{Trunc}_n(x)$ pour $x \in I^d$ en prenant les n premiers chiffres de chaque composante.

Comme au chapitre 5, PL_d est l'ensemble des fonctions continues et affines par morceaux sur I^d. Rappelons enfin la notion classique de fonction réelle calculable en temps polynômial [26].

Définition 7.1 *Une fonction $f : I^d \to I^d$ est dans P_d s'il existe une machine de Turing $\mathcal{T}$ telle que*

1. *$\mathcal{T}$ reçoit pour entrée un entier p indiquant la précision demandée pour la sortie $f(x)$.*
2. *$\mathcal{T}$ calcule le nombre m de chiffres de x dont elle a besoin.*
3. *$\mathcal{T}$ fait appel à un oracle pour obtenir $\mathrm{Trunc}_m(x)$ (par définition cette étape se déroule en temps constant).*
4. *Enfin $\mathcal{T}$ calcule la sortie $\mathrm{Trunc}_p(f(x))$.*
5. *Il existe des constantes a et k telles que pour tout $x \in I^d$ et tout $p \in \mathbb{N}$, le calcul est effectué en temps au plus ap^k.*

Remarquons qu'il est possible de supposer sans perte de généralité que m est polynômial en p, puisqu'une machine de Turing ne peut lire qu'un nombre polynômial de bits en temps polynômial. Pour les fonctions d'encodage, nous utilisons la notion suivante de calculabilité en temps polynômial.

Définition 7.2 *Une fonction $\phi : \{0,1\}^+ \to I^d$ est dans* PE_d *s'il existe une machine de Turing $\mathcal{T}$ telle que*

1. *$\mathcal{T}$ reçoit en entrée $x \in \{0,1\}^+$ et un entier p indiquant la précision requise sur la sortie $\phi(x)$.*
2. *$\mathcal{T}$ calcule la sortie* $\mathrm{Trunc}_p(\phi(x))$.
3. *IL existe des constantes a et k telles que pour tout $x \in I^d$ et tout $p \in \mathbb{N}$, le calcul se déroule en temps au plus $a(p+|x|)^k$.*

P_d et PE_d sont en fait des sous-ensembles de la classe des fonctions de $\mathbb{R}^m$ to $\mathbb{R}^n$ calculables en temps polynômial [26].

Toutes les fonctions calculables en temps polynômial ne peuvent pas être utilisées comme fonction de transition de nos systèmes. (afin de montrer que des familles de circuits peuvent simuler ces systèmes, une propriété de Lipschitz est nécessaire). Nous n'utiliserons que la sous-classe suivante.

Définition 7.3 *Une fonction $f : I^d \to I^d$ est dans* LP_d *si $f \in \mathrm{P}_d$ et f est lipschitzienne (c'est-à-dire qu'il existe une constante C telle que pour tous les points x, y de I^d, $|f(x)-f(y)| \leq C|x-y|$).*

Il est facile de voir que $f \in \mathrm{LP}_d$ si et seulement si cette fonction est approximable en temps polynômial au sens de Siegelmann & Sontag [45]:

Lemme 7.1 *Une fonction $f : I^d \to I^d$ est dans* LP_d *si et seulement si*

- *f est lipschitzienne.*
- *Il existe une machine de Turing $\mathcal{T}$ qui sur l'entrée $x \in D_n^d$ calcule* $\mathrm{Trunc}_n(f(x))$ *en temps polynômial.*

Preuve. Supposons que f est approximable en temps polynômial. Etant donné $x \in I^d$ et $n \in \mathbb{N}$, nous devons calculer y tel que $|y - f(x)| \leq 2^{-n}$. Ceci peut être fait en prenant $y = \mathrm{Trunc}_p(f(\mathrm{Trunc}_p(x))$, pour p suffisamment grand. Le temps de calcul est polynômial en p par hypothèse. Puisque f est lipschitzienne,

$$\begin{aligned} |y - f(x)| &\leq |\mathrm{Trunc}_p(f(\mathrm{Trunc}_p(x)) - f(\mathrm{Trunc}_p(x))| + |f(\mathrm{Trunc}_p(x)) - f(x)| \\ &\leq 2^{-p} + 2^{k-p} \end{aligned}$$

où 2^k ($k \in \mathbb{N}$) est une borne supérieure de la constante de Lipschitz de f. Il suffit donc de prendre $p = k + n + 1$.

La réciproque est évidente. □

Pour pouvoir travailler avec des nombres réels arbitraires, on considère pour chacune des classes $\mathcal{C}$ définies ci-dessus la classe $\mathcal{C}/\mathrm{poly}$. Comme on l'explique au chapitre 2, les fonctions de ces nouvelles classes sont calculées par des machines de Turing utilisant des conseils de tailles polynômiales. Rappelons par exemple que $f \in \mathrm{PE}_d/\mathrm{poly}$ s'il existe une fonction $A : \mathbb{N} \rightarrow \{0,1\}^+$ (la fonction de conseil) et une machine de Turing $\mathcal{T}$ telle que sur l'entrée $(p, x, A(p+|x|))$, $\mathcal{T}$ calcule les p premiers chiffres de $f(x)$ en temps $O((p+|x|)^k)$. La longueur du conseil $A(p+|x|)$ doit également être polynômiale en $p+|x|$ (cette dernière exigence est en fait redondante, puisque la partie du ruban qui peut être parcourue par $\mathcal{T}$ en temps polynômial est de taille polynômiale).

Le résultat suivant est important car tous les exemples de systèmes présentés dans [45] ainsi que dans ce chapitre sont affines par morceaux.

Théorème 7.1 *$PL_d \subset \mathrm{LP}_d/\mathrm{poly}$.*

Preuve. Une fonction f de PL_d est clairement lipschitzienne. A l'intérieur d'un polyhèdre $P \in \mathcal{P}$, $e_j(f(x)) = \sum_{i=1}^{d} w_{ij}x_i + \theta_j$. La constante de Lipschitz de $f_{|P}$ est $L_P = \max_j \sum_{i=1}^{d} |w_{ij}|$, et celle de f est $\max_{P \in \mathcal{P}} L_P$.

Afin de montrer que $f \in \mathrm{P}_d$, nous allons d'abord montrer que f est une fonction lipschitzienne de ses paramètres. On écrira donc maintenant $f(W, x)$ au lieu de $f(x)$ pour les prendre en compte explicitement. Définissons d'abord une convention permettant d'associer à f un unique vecteur de paramètres $W \in \mathbb{R}^k$. On peut supposer que $\mathcal{P}$ est une triangulation, c'est-à-dire que chaque polygône P est un polytope (l'enveloppe convexe de $d+1$ points en position générale). f est uniquement définie par la liste W des sommets de ces polytopes et des valeurs de f à ces sommets, énumérés dans un certain ordre (fixé). On suppose sans perte de généralité que I^d est dans l'intérieur de $\cup_{P \in \mathcal{P}} P$, de sorte que f est définie, continue et affine par morceaux dans un voisinage de I^d. Certains vecteurs $W \subset \mathbb{R}^k$ ne correspondent à aucune triangulation de I^d (ceci arrive quand deux polytopes s'intersectent); d'autres sont associés à une triangulation qui ne recouvre pas I^d. Ces vecteurs sont appelés *invalides*. L'ensemble des vecteurs valides est ouvert: pour tout vecteur valide W_0, il existe un parallépipède $B(W_0, r)$ de vecteurs valides. Nous travaillons maintenant à l'intérieur d'un tel parallépipède, et considérons pour un $x \in I^d$ fixé la fonction

$$\begin{aligned} g \quad : \quad & B(W_0, r) \longrightarrow \mathbb{R}^d \\ & W \mapsto f(W, x). \end{aligned}$$

Cette fonction est continue, et affine par morceaux puisque les équations définissant le polytope dans lequel x est situé sont linéaires. Donc l'analyse de la fonction $x \mapsto f(W, x)$ effectuée au début de cette démonstration s'applique à g: g est lipschitzienne, et sa constante de Lipschitz est plus petite que $\sum_{i=1}^{d} x_i \leq d$.

Nous sommes maintenant prêts à montrer comment calculer $f(W_0, x)$ en temps polynômial. D'après le lemme 7.1, on peut supposer que $x \in D^d$. Une approximation de $f(W_0, x)$ peut être obtenue en calculant $f(\mathrm{Trunc}_q(W_0), x)$ pour q suffisamment grand (ceci a bien un sens, puisque l'ensemble des vecteurs valides est ouvert). Comme g est lipschitzienne, il suffit de prendre $q = n + C$ pour obtenir n chiffres de $f(W_0, x)$, pour une certaine constante C. Le calcul se fait clairement en temps polynômial avec $\mathrm{Trunc}_q(W_0)$ pour conseil. $\square$

7.2 Entrées hors-ligne

Dans cette section nous étudions un mode de calcul pour lequel l'entrée du système est codée dans son état initial.

Définition 7.4 *La fonction de décision $F : \{0,1\}^+ \to \{0,1\}$ est calculable hors-ligne si elle peut être calculée par un système hors-ligne $\mathcal{S} = (f, \phi, [a_0, b_0], [a_1, b_1])$. Ici $\phi : \{0,1\}^+ \to I^d$ est une fonction d'encodage, $[a_0, b_0]$ et $[a_1, b_1]$ ($a_1 > b_0$) sont deux intervalles de décision disjoints, et f est une fonction sur I^d.*

Sur une entrée $u \in \{0,1\}^+$, le calcul effectué par $\mathcal{S}$ est défini par la suite $(x(t))_{t\in\mathbb{N}}$ suivante :

- *$x(0) = \phi(u)$;*
- *$x(t+1) = f(x(t))$ pour $t \geq 0$.*

Soit $t(u) = \min\{t; x_1(t) \in [a_0, b_0] \cup [a_1, b_1]\}$ (on suppose que $t(u)$ existe effectivement). Le résultat du calcul est $F(u) = 0$ si $x_1(t(u)) \in [a_0, b_0]$, et $F(u) = 1$ sinon. La complexité en temps de ce calcul est la fonction T telle que $T(n) = \max_{|u|=n} t(u)$.

Théorème 7.2 *Toute fonction $F : \{0,1\}^+ \to \{0,1\}$ peut être calculée hors-ligne par $f \in PL_1$ en temps exponentiel. De plus la fonction d'encodage est dans* PE_1, *et est indépendante de F. Les intervalles de décision sont également indépendants de F, et sont tels que $a_0 = b_0 \in \mathbb{Q}$ et $a_1 = b_1 \in \mathbb{Q}$.*

Preuve. Il est clairement possible de remplacer I par l'intervalle $[0, 2]$. Nous allons travailler avec des écritures en base 4 utilisant uniquement les chiffres 1 et 3, comme au chapitre 5. Il sera donc utile d'utiliser à la place de F la fonction G telle que $G(N) = 1$ si $F(N) = 0$ et $G(N) = 3$ si $F(N) = 1$. Contrairement aux conventions utilisées dans le reste du chapitre, toutes les écritures «décimales» sont en base 4 dans cette preuve.

Soit $\phi(N) = 1.1(10)^N 3^{+\infty}$ (N est maintenant considéré comme un entier plutôt que comme un mot sur l'alphabet $\{0,1\}$). Sur $[1.1, 2]$, posons

$$f(x) = 2^{-(p+1)}[2(x - 1.1) + a] + q_0,$$

où $a = 0.0G(0)0G(1)\ldots 0G(n)0\ldots$ et $q_0 = 0.q_{01}\ldots q_{0p}$ est l'état initial d'un automate à pile $\mathcal{A}$ décrit ci-dessous. L'entier p est tel que $\mathcal{A}$ a moins de 2^p états. Par conséquent

$$f(\phi(N)) = 0.q_{01}\ldots q_{0p} 1G(0)1G(1)\ldots 1G(N-1)3G(N)3G(N+1)3\ldots.$$

Le travail de $\mathcal{A}$ consiste simplement à dépiler la suite de chiffres $1G(0)1G(1)\ldots 1G(N-1)3G(N)3G(N+1)3\ldots$ considérée comme une pile de profondeur infinie. Comme on l'a vu au chapitre 5, cette tâche peut être effectuée par une fonction de PL_1. Quand $\mathcal{A}$ lit un 3 à une position impaire dans cette suite, il sait que le chiffre suivant est $G(N)$. Après avoir lu $G(N)$, il efface la pile, et entre dans l'un des deux états finaux a_0, a_1. Enfin le «trou» $[1, 1.1]$ peut être rempli par une fonction affine de façon à obtenir une fonction continue et affine par morceaux $[0, 2]$.

La principale tâche à effectuer pour calculer le $n^{\text{ième}}$ chiffre de $\phi(N)$ est la comparaison de n et $2N+2$, ce qui peut être fait en temps polynômial en $\log n$ et $\log N$. Donc ϕ est dans PE_1. □

On a donc une différence importante avec les systèmes à poids rationnels étudiés au chapitre 5. Si les paramètres définissant $f \in PL_d$ sont rationnels, les itérations de f peuvent être calculées exactement sur une machine de Turing (on suppose que l'état initial est lui aussi rationnel); dans ce modèle on ne peut donc calculer ques des fonctions récursives (partielles). On a vu que toutes les fonctions récursives peuvent effectivement être calculées de cette manière, mais apparemment deux dimensions sont nécessaires. Par contre le théorème 7.2 montre qu'une seule dimension suffit dans le modèle réel. Le théorème 7.3 montre que même dans le modèle réel, il est apparemment plus efficace de calculer en dimension deux.

Théorème 7.3 *Soit $C = (C_n)_{n\in\mathbb{N}}$ une famille de circuits tels que la taille $S(n)$ de C_n est une fonction non décroissante de n. La famille C peut être simulée en temps $O(nS(n)^k)$ (pour un $k > 0$ fixé) par itération d'une fonction function de PL_2. De plus la fonction d'encodage ϕ est dans PE_2 et est indépendante de C. Les intervalles de décision sont également indépendants de C, et sont tels que $a_0 = b_0 \in \mathbb{Q}$ et $a_1 = b_1 \in \mathbb{Q}$.*

Preuve. Nous suivons la méthode de Siegelmann & Sontag, et travaillons sur l'intervalle $[0,2]^2$ comme dans la démonstration précédente. Il est à nouveau utile de travailler en base 4 et de considérer que F est définie sur $\{1,3\}^+$ au lieu de $\{0,1\}^+$. La fonction d'encodage est

$$\phi(x) = \phi(x_1 \ldots x_n) = (0.x_1 \ldots x_n, 1.1).$$

De ce point, on fait une transition en

$$f(\phi(x)) = (0.q_{01} \ldots q_{0p} x_1 \ldots x_n, c),$$

où $q_{o1} \ldots q_{0p}$ encode l'état initial d'une machine de Turing $\mathcal{M}$ définie ci-dessous, et où les chiffres de $c \in [0,1]$ encodent la suite $C = (C_n)_{n\in\mathbb{N}}$. D'après le chapitre 5, on sait qu'il existe une fonction de PL_2 qui simule $\mathcal{M}$ en temps réel. Le rôle de $\mathcal{M}$ est simplement de faire fonctionner C_n sur l'entrée $x_1 \ldots x_n$, en partant du ruban xc. Ceci est une instance du problème d'évaluation de circuit («Circuit Value Problem»), dont on sait qu'il est dans P [2]. Le temps de calcul total est $O(nS(n)^k)$ puisque l'encodage de C_n est à distance $O(nS(n)\log S(n))$ de la position initiale de la tête de lecture (un circuit de taille S peut être encodé par un mot de longueur $O(S \log S)$). $\square$

Corollaire 7.1 *Une fonction $F : \{0,1\}^+ \to \{0,1\}$ de* P/poly *peut être calculée hors-ligne en temps polynômial par $f \in PL_2$ avec une fonction d'encodage dans* PE_2.

Preuve. Une fonction de P/poly a des circuits de taille polynômiale [2]. $\square$

Puisque toute fonction admet des circuits de taille exponentielle (par exemple, sa forme normale disjonctive pour chaque taille d'entrée), on pourrait ajouter a ce corollaire qu'une fonction arbitraire peut être calculée en temps exponentiel, mais ceci est déjà connu d'après le théorème 7.2.

Théorème 7.4 *Si $F : \{0,1\}^+ \to \{0,1\}$ peut être calculée hors-ligne en temps polynômial par $f \in \mathrm{LP}_d$/poly avec une fonction d'encodage $\phi \in \mathrm{PE}_d$/poly, alors $F \in$* P/poly.

Ce résultat montre que les théorèmes 7.2 et 7.3 sont en un certain sens optimaux : les calculs en temps polynômial dans des dimensions supérieures ne sont pas plus puissants qu'en dimension deux, et le temps de calcul exponentiel du théorème 7.2 ne peut pas être remplacé par du temps polynômial, même en dimensions supérieures. Il est également inutile de calculer avec des classes de fonctions plus générales que PL_d. Remarquons enfin que le théorème 7.4 s'applique aux fonctions d'encodages de $\mathrm{PE}_d/\mathrm{poly}$, alors que PE_d suffirait pour les théorèmes 7.2 et 7.3. Cette plus grande généralité est justifiée car les fonctions de $\mathrm{PE}_d/\mathrm{poly}$ seront nécessaires dans la section suivante.

La preuve du théorème 7.4 est semblable à l'analyse des réseaux de processeurs généralisés de Siegelmann et Sontag. Nous avons d'abord besoin du résultat suivant.

Lemme 7.2 *Soit $(f_t)_{t\in\mathbb{N}}$ une famille de fonctions lipschitziennes sur I^d ayant leurs constantes de Lipschitz uniformément bornées par $C > 1$, et soit ϵ un réel strictement positif. Soient $(x(t))_{t\in\mathbb{N}}$ et $(\overline{x}(t))_{t\in\mathbb{N}}$ deux suites définies par $x(t+1) = f_t(x(t))$ et*

$$\overline{x}(t+1) = f_t(\overline{x}(t)) + \epsilon(t)$$

avec $\forall t \geq 0,\ |\epsilon(t)| \leq \epsilon$ et $|\overline{x}(0) - x(0)| \leq \epsilon$. Alors

$$\forall t \geq 0,\ |\overline{x}(t) - x(t)| \leq \frac{\epsilon}{C-1}(C^{t+1} - 1).$$

Preuve. Par récurrence. Le résultat est vrai pour $t = 0$. Supposons qu'il est vrai à l'instant t. Alors

$$\begin{aligned} |\overline{x}(t+1) - x(t+1)| &\leq C|\overline{x}(t) - x(t)| + \epsilon \\ &\leq \frac{C\epsilon}{C-1}(C^{t+1} - 1) + \epsilon \quad = \quad \frac{\epsilon}{C-1}(C^{t+2} - 1). \end{aligned}$$

□

Dans la section suivante nous appliquerons ce lemme à une famille de fonctions, mais pour le moment on va prendre $f_t = f$ pour tout t.

Preuve du théorème 7.4. Afin de pouvoir faire la distinction entre une sortie égale à 0 (intervalle $[a_0, b_0]$) et une sortie égale à 1 (intervalle $[a_1, b_1]$), on va calculer les vecteurs $x(t)$ avec une précision d'au moins $(a_1 - b_0)/3$ pour $0 \leq t \leq t(u)$ (rappelons que $t(u)$ est le temps de calcul sur l'entrée u, et que $n = |u|$). Soit $(\overline{x}(t))$ la suite définie par les relations $\overline{x}(0) = \mathrm{Trunc}_q(\phi(u))$ et $\overline{x}(t+1) = \mathrm{Trunc}_q(f(\overline{x}(t))$. Soit $C > 1$ un majorant de la constante de Lipschitz de f. D'après le lemme 7.2, il suffit d'avoir

$$\epsilon = 2^{-q} \leq \frac{(a_1 - b_0)(C-1)}{3(C^{T(n)} - 1)}.$$

On peut donc prendre $q = O(T(n)) = O(n^k)$, où k est une constante. Puisque $f \in \mathrm{P}_d$, $\overline{x}(t+1)$ peut être calculé à partir de $\overline{x}(t)$ en temps polynômial avec un conseil de taille polynômiale. Puisque $\phi \in \mathrm{PE}_d$, $\overline{x}(0)$ peut être calculé en temps polynômial en $q + n$ avec un conseil de taille polynômiale en $q + n$, donc polynômiale en n. □

7.3 Entrées en-ligne

Dans cette section, nous étudions un mode de calcul dans lequel la donnée du calcul n'est pas codée dans l'état initial du système, mais lui est au contraire fournie bit par bit en partant d'un état initial fixé. On peut montrer que ces systèmes en-ligne sont équivalents aux réseaux de processeurs généralisés de Siegelmann & Sontag. Cependant nous n'utiliserons pas ce modèle, car nous pensons que le notre est plus simple et plus naturel.

Définition 7.5 *La fonction de décision* $F : \{0,1\}^+ \to \{0,1\}$ *est calculable en ligne si elle peut être calculée par un système en ligne* $\mathcal{S} = (f, f_0, f_1, [a_0, b_0], [a_1, b_1])$*. Les fonctions* $f_0, f_1 : I^d \to I^d$ *sont appelées* fonctions d'encodage, $[a_0, b_0]$ *et* $[a_1, b_1]$ *(*$a_1 > b_0$*) sont deux intervalles de décision disjoints, et* f *est une fonction sur* I^d*.*

Sur une entrée $u \in \{0,1\}^+$*, le calcul effectué par* $\mathcal{S}$ *est défini par la suite* $(x(t))_{t\in\mathbb{N}}$ *suivante :*

- $x(0) = 0$;
- $x(t+1) = f_{u_{t+1}}(x(t))$ *pour* $0 \leq t \leq n-1$.
- $x(t+1) = f(x(t))$ *pour* $t \geq n$.

Soit $t(u) = \min\{t; x_1(t) \in [a_0, b_0] \cup [a_1, b_1]\}$ *(on suppose que* $t(u)$ *existe effectivement). Le résultat de ce calcul est* $F(u) = 0$ *si* $x_1(t(u)) \in [a_0, b_0]$*, et* $F(u) = 1$ *sinon.*

La complexité en temps de ce calcul est la fonction T *telle que* $T(n) = \max_{|u|=n} t(u)$*.*

On peut imaginer des définitions légèrement différentes de ces systèmes en ligne. On pourrait par exemple définir un système par deux fonctions f_0 et f_1 seulement, et prendre pour entrée un mot infini $u \in \{0,1\}^\omega$ avec $u_i = 0$ pour i suffisamment grand (cette définition est plus proche des réseaux de neurones de Siegelmann & Sontag).

Il est clair que les systèmes en ligne sont des cas particuliers des systèmes hors-ligne étudiés dans la section 7.2 avec pour fonction d'encodage

$$\phi(u_1 \ldots u_n) = f_{u_n} \circ \ldots f_{u_2} \circ f_{u_1}(0). \tag{7.1}$$

Théorème 7.5 *Si* $f_0, f_1 \in \mathrm{LP}_d/\mathrm{poly}$*, la fonction d'encodage de l'équation (7.1) est dans* $\mathrm{PE}_d/\mathrm{poly}$*.*

Preuve. Montrons que pour une entrée $u = u_1 \ldots u_n \in \{0,1\}^+$, on peut calculer $\phi(u)$ avec une précision 2^{-p} en temps $O((p+n)^k)$. En partant de $\overline{x}(0) = 0$, on calcule $\overline{x}(t+1) = \mathrm{Trunc}_q(f_{u_t}(\overline{x}(t))$, pour q suffisamment grand et $0 \leq t \leq n-1$. D'après le lemme 7.2, il suffit de prendre

$$\epsilon = 2^{-q} \leq \frac{2^{-p}(C-1)}{C^{n+1}-1},$$

où C majore les constantes de Lipschitz de f_0 et f_1 (on suppose sans perte de généralité que $C \geq 2$). On peut donc prendre $q = \lceil p + (n+1)\log C \rceil$. Puisque f_0 et f_1 sont dans $\mathrm{P}_d/\mathrm{poly}$, $\overline{x}(t+1)$ peut être calculé à partir de $\overline{x}(t)$ en temps polynômial avec un conseil de longueur polynômiale en q.

Le temps de calcul est clairement polynômial en la taille de l'entrée $s = n + p$, et se fait avec un conseil de taille polynômiale. Cependant ce conseil doit dépendre uniquement de s. Pour garantir ceci, il suffit de remplacer la valeur de q utilisée jusqu'à présent par $q = \lceil \log C \rceil (s+1)$. $\square$

On en déduit immédiatement que les systèmes en ligne ne sont pas plus puissants que les systèmes hors ligne.

Corollaire 7.2 *Si $F : \{0,1\}^+ \to \{0,1\}$ est calculable en temps polynômial par un système en ligne, alors $F \in \mathrm{P/poly}$.*

Preuve. Conséquence directe des théorèmes 7.4 et 7.5. □

On voit aussi facilement que la réciproque est vraie : les systèmes en ligne sont donc équivalents aux systèmes hors ligne pour le temps polynômial.

Théorème 7.6 *Toute fonction F de* P/poly *peut être calculée par un système en ligne avec f_0, f_1 et f dans PL_2.*

Preuve. Rappelons que la fonction d'encodage du théorème 7.3 est

$$\phi(x) = \phi(x_1 \ldots x_n) = (0.y_1 \ldots y_n, 1.1),$$

avec $x_i \in \{0,1\}$ and $y_i = 2x_i + 1 \in \{1,3\}$. Ici on prend plutôt $\phi(x) = (0.y_n \ldots y_1, 1.1)$ (cela revient au même, car une pile d'un automate à deux piles peut être renversée en temps constant; On peut ensuite poursuivre le calcul avec la même fonction f qu'au théorème 7.3). Il est clair que les deux fonctions d'encodage ci-dessous conviennent :

- $f_0(z_1, z_2) = (\frac{1+z_1}{4}, 1.1)$;
- $f_1(z_1, z_2) = (\frac{3+z_1}{4}, 1.1)$.

□

Les deux types de systèmes sont aussi équivalents pour le temps exponentiel.

Théorème 7.7 *Toute fonction $F : \{0,1\}^+ \to \{0,1\}$ peut être calculée par un système en ligne (f_0, f_1, f) tel que $f \in PL_1$ et $f_0, f_1 : \mathbb{R} \to \mathbb{R}$ soient des polynômes à coefficients rationnels.*

Preuve. Rappelons que la fonction d'encodage du théorème 7.2 est

$$\phi(N) = \phi(u_1 \ldots u_n) = 1.1(10)^N 3^{+\infty}.$$

Ici on prend plutôt $\psi(N) = \phi(0) - \phi(N)$, car l'état initial du système doit être 0. Une fois qu'on a calculé $\psi(N)$, il est clairement possible de calculer $\phi(N) = \phi(0) - \psi(N)$ puis d'appliquer la même méthode que pour la preuve du théorème 7.2.

Il ne reste plus qu'a trouver f_0 et f_1 tels que

$$f_{u_k} \circ \ldots \circ f_{u_1}(0) = \psi(N_k)$$

avec $N_k = u_1 \ldots u_k$. Il existe des constantes rationnelles non nulles a et b telles que $\psi(N) = a(1 - b^N)$. Puisque $N_{k+1} = 2N_k + u_{k+1}$, $\psi(N_{k+1}) = a\left[1 - b^{u_{k+1}}(b^{N_k})^2\right]$. Mais $b^{N_k} = 1 - \psi(N_k)/a$, donc $f_i(x) = a\left[1 - b^i(1 - x/a)^2\right]$. □

7.4 Comparaison avec le modèle BSS

Dans cette section nous montrons que le modèle BSS est au moins aussi puissant que les systèmes en ligne ou hors ligne. Pour cela nous considérons des machines de Turing réelles avec entrées et sorties discrètes. On définit donc les classes de complexité suivantes : P_R est l'ensemble des fonctions calculables en temps polynômial dans le modèle BSS; P_{RD} est la restriction de P_R aux entrées-sorties discrètes (en identifiant une fonction et le langage correspondant, $\mathrm{P}_{RD} = \mathrm{P}_R \cap \Sigma^+$). Pour des machines non déterministes, on définit les classes NP_R et NP_{RD} de la même manière.

Théorème 7.8 *Toute fonction $F : \{0,1\}^+ \to \{0,1\}$ peut être calculée en temps exponentiel dans le modèle BSS. En ce qui concerne le temps polynômial, on a :*

$$\mathrm{P/poly} \subset \mathrm{P}_{RD} \text{ et } \mathrm{NP/poly} \subset \mathrm{NP}_{RD}.$$

Preuve. Il suffit d'adapter les idées des preuves des théorèmes 7.2 et 7.3.

Afin de calculer une fonction $F : \{0,1\}^+ \to \{0,1\}$ quelconque, on considère de nouveau le réel $c = 0.F(0)F(1)\ldots F(N)\ldots$. Puisque $F(N) = \lfloor 2^{N+1}c \rfloor$, $F(N)$ peut être calculé en décalant c de $(N+1)$ positions vers la gauche (ce qui se fait avec des multiplications par 2) et en soustrayant à chaque fois le chiffre de poids fort (soustraction et comparaison). Ceci se fait clairement en temps exponentiel en $\log N$.

Afin de calculer $F \in \mathrm{P/poly}$, on simule un automate à deux piles en utilisant un registre pour chaque pile, comme pour la preuve du théorème 7.3.

Dans le cas de NP/poly on traite le conseil de la même manière. Il reste à transformer le choix non-déterministe $h \in \mathbb{R}^{p(n)}$ d'une machine de Turing réelle en un choix discret $g \in \{0,1\}^{p(n)}$ (p est une fonction polynômiale de la taille des entrées n). Ceci se fait de manière évidente avec le programme suivant [1].

```
for i:=1 to p(n) do
   if h[i]>0 then g[i]:=1 else g[i]:=0
```

□

1. Dans une version préliminaire de ce chapitre (rapport du LIP 93-01), j'ai également suggéré une solution basée sur une décomposition en base 2. Le programme proposé est hélas manifestement faux. L'erreur est réparable, mais la solution de ce chapitre me semble suffisament convaincante pour qu'on en reste là.

Chapitre 8

Une version affaiblie du modèle de Blum, Shub & Smale

Comme en complexité discrète, le problème P = NP est l'un des principaux problèmes ouverts dans la théorie des calculs sur les nombres réels à la Blum-Shub-Smale. Dans cette section nous proposons un modèle plus faible (le modèle BSS faible) pour lequel on peut montrer que P$\neq$NP modulo une hypothèse classique de complexité discrète (non-effondrement de la hiérarchie polynômiale).

Le modèle BSS faible est obtenu à partir du modèle habituel en abandonnant l'hypothèse que les opération arithmétiques se font en temps constant. Ceci conduit à une notion de calcul réel faisable probablement plus réaliste, puisqu'on retrouve la notion discrète de calculabilité en temps polynômial lorsqu'il se trouve que les paramètres d'une machine de Turing réelle sont entiers. Autrement dit, un algorithme qui fonctionne en temps polynômial dans le modèle faible est idéal au sens de Renegar, puisque «it seems that the ideal algorithm is one that provides a "nice" bound in the real number model and which when restricted to sentences with polynomials whose coefficients are rational numbers also provides a "nice" bound in the bit model» [36].

Afin de montrer que (probablement) P$\neq$NP dans notre modèle, nous étudions les capacités des machines réelles avec entrées-sorties discrètes, comme dans les chapitres précédents. Nous prouvons précisément que la classe des ensembles reconnaissables en temps polynômial est P/poly (voir le chapitre 7 pour une définition de P/poly). Le principal outil de cette preuve est un nouveau résultat sur les ensembles semi-algébriques intéressant par lui-même. Ce résultat affirme qu'un ensemble semi-algébrique d'intérieur non vide défini par des «petites» équations contient un «petit» point rationnel.

Nous décrivons finalement des applications de ces résultats aux réseaux de neurones récurrents et acycliques. On montre que la classe des fonctions calculables en temps polynômial par des réseaux récurrents utilisant des fonctions de sortie affines par morceaux (éventuellement non continues) est exactement P/poly. Ceci était connu jusqu'à présent uniquement pour des fonctions de sorties continues [45]. On définit pour des familles de réseaux acycliques réels une notion d'uniformité inspirée de la définition traditionnelle pour les circuits booléens. On montre ensuite que certaines familles de circuits réels peuvent être simulées efficacement par des familles de circuits à seuils. Ce problème a été étudié récemment par W. Maass [30] dans le cas plus général des familles non-uniformes de circuits réels, mais ses résultats s'appliquent uniquement à des circuits de profondeur constante. Cette restriction peut être relâchée pour des familles uniformes.

8.1 Préliminaires

Dans cette section nous précisons les notations utilisées dans ce chapitre, et rappelons quelques propriétés élémentaires des ensembles semi-algébriques (on peut trouver un traitement approfondi de ce sujet dans [5] et [8]).

8.1.1 Notations diverses

Comme d'habitude, $\mathbb{N}$ est identifié à $\{0,1\}^+$. La taille $s(x)$ d'un entier $x \in \mathbb{Z}$ est le nombre de chiffres de sa représentation en base 2. La taille d'un rationnel $x \in \mathbb{Q}$ est $s(x) = \min\{s(p)+s(q), x = p/q,\ p,q \in \mathbb{Z}\}$. La taille d'un point rationnel $x = (x_1,\ldots,x_p) \in \mathbb{Q}^p$ est $s(x) = \sum_{i=1}^{p} s(x_i)$.

Etant donné un anneau A, $A[X_1,\ldots,X_p]$ est l'anneau des polynômes en p variables à coefficients dans A. Lorsqu'il n'y a pas de risque de confusion, on écrit $A[X]$ au lieu de $A[X_1,\ldots,X_p]$. Le degré de $P \in A[X_1,\ldots,X_p]$ est noté $d(P)$, et le degré de la fraction rationnelle $R = P/Q$ est $d(R) = d(P) + d(Q)$.

8.1.2 Ensembles semi-algébriques

Définition 8.1 *Un ensemble semi-algébrique de base $\mathcal{S} \subset \mathbb{R}^p$ est défini par une conjonction d'égalités et d'inégalités polynômiales :*

$$\mathcal{S} = \{x \in \mathbb{R}^p;\ Q_1 > 0 \wedge \ldots \wedge Q_m > 0 \wedge R_1 = 0 \wedge \ldots \wedge R_n = 0\}$$

où $Q_i, R_i \in \mathbb{Z}[X_1,\ldots,X_p]$ sont des polynômes non constants (m et/ou n peuvent éventuellement être nuls). Un ensemble semi-algébrique est une union finie d'ensembles semi-algébriques de base.

On autorise habituellement toutes les conditions de signes ($P = 0$, $P \le 0$, $P < 0$, $P \ge 0$ and $P > 0$) dans cette définition, ce qui ne change évidemment rien.

Il est possible de prendre les P_i et les R_i dans $\mathbb{R}[X]$ au lieu de $\mathbb{Z}[X]$. Les ensembles correspondant seront encore appelés semi-algébriques. Il devrait être clair d'après le contexte si les polynômes considérés ont des coefficients entiers ou réels.

Théorème 8.1 (principe de Tarski-Seidenberg)
Soit $S' = \{x \in \mathbb{R}^n;\ \exists y \in \mathbb{R}\ (x,y) \in S\}$ la projection sur $\mathbb{R}^n$ d'un ensemble $S \subset \mathbb{R}^{n+1}$. Si S est semi-algébrique avec des coefficients dans $\mathbb{Z}$ (respectivement, dans $\mathbb{R}$), S' est semi-algébrique avec des coefficients dans $\mathbb{Z}$ (respectivement, dans $\mathbb{R}$).

L'élimination des quantificateurs est une conséquence importante du principe de Tarski-Seidenberg : un ensemble défini par une formule logique de la théorie du premier ordres des réels est semi-algébrique. Ces formules sont définies par induction de la manière suivante :

Définition 8.2 (formules du premier ordre)
Soit $P \in \mathbb{Z}[X_1,\ldots,X_n]$ et op $\in \{<,\le,=,\ge,>\}$ *:* P op 0 *est une formule du premier ordre.*

Si F_1 et F_2 sont des formules du premier ordre, $F_1 \vee F_2$, $F_1 \wedge F_2$, $\forall x \in \mathbb{R}\ F_1$ et $\exists x \in \mathbb{R}\ F_1$ sont des formules du premier ordre.

Si les variables libres d'une formule F sont $x_1, \ldots, x_n$, on peut écrire $F(x_1, \ldots, x_n)$ au lieu de F. Le sous-ensemble de $\mathbb{R}^n$ défini par F est alors

$$S = \{(x_1, \ldots, x_n) \in \mathbb{R}^n; F(x_1, \ldots, x_n) \text{ est vraie}\}.$$

Théorème 8.2 (élimination des quantificateurs) *L'ensemble $S \subset \mathbb{R}^n$ défini par une formule $F(x_1, \ldots, x_n)$ du premier ordre est semi-algébrique.*

Preuve. La classe des ensembles semi-algébriques est clairement close par opérations booléennes. Le principe de Tarski-Seidenberg montre qu'on peut éliminer un quantificateur existentiel. Un quantificateur universel peut être remplacé par un quantificateur existentiel puisque $\neg\exists x\ \neg F$ est équivalent à $\forall x\ F$. $\Box$

8.2 Présentation des résultats

8.2.1 Le modèle faible

Dans tout le chapitre, $\alpha_1, \ldots, \alpha_p$ sont les constantes non nulles apparaissant dans le programme d'une machine de Turing réelle. On peut associer un arbre de calcul à un programme de la manière suivante. Chaque nœud de l'arbre est étiqueté par une instruction. Un nœud correspondant à un test a deux fils correspondant aux deux issues possibles du test. Les autres nœuds internes ont un seul fils. En chaque nœud $\mathcal{N}$, le contenu d'une variable x est une fraction rationnelle $Q_{\mathcal{N},x}(u, \alpha)$ de l'entrée $u \in \mathbb{R}^n$ et des constantes $\alpha = (\alpha_1, \ldots, \alpha_p)$. Pour toute borne $T > 0$, un sous-ensemble de $\mathbb{R}^n$ reconnaissable en temps T par une MTR est donc semi-algébrique (cette propriété sera utile dans la section 8.3).

Dans le modèle faible le coût d'une opération arithmétique n'est pas constant.

Définition 8.3 *Si $x = Q(u, \alpha)$ après une opération arithmétique $x := y\ op\ z$, le coût de cette opération dans le modèle BSS faible est par définition égal au maximum de $d(Q)$ et de la taille maximum des coefficients de Q.*

Ce modèle est plus faible que le modèle standard puisque la classe des fonctions calculables en temps polynômial est plus petite. Il est par exemple possible de produire en temps polynômial des entiers de taille exponentielle avec un programme utilisant des constantes entières dans le modèle standard, alors que c'est clairement impossible dans le modèle faible (on peut voir grâce au théorème 8.4 que ceci reste vrai même si les constantes ne sont pas entières).

L'intérêt du modèle faible n'est pas uniquement qu'il permet de démontrer les théorèmes 8.3 et 8.6. Il constitue probablement un modèle de calcul réel plus réaliste que le modèle BSS standard, puisqu'on retrouve la notion habituelle de calcul en temps polynômial lorsque les constantes $\alpha_1, \ldots, \alpha_p$ sont entières (voir le lemme 8.3). Remarquons que la possibilité peut-être mal venue de créer en temps polynômial dans le modèle standard des entiers de taille exponentielle est un outil crucial dans la preuve récente que $\mathrm{P}_{\mathbb{R}} \neq \mathrm{NC}_{\mathbb{R}}$ [13]. Cette preuve ne fonctionne plus dans le modèle faible.

Voici une petite caractérisation bien utile du temps polynômial dans le modèle faible.

Lemme 8.1 *Une fonction est calculable en temps polynômial dans le modèle BSS faible si et seulement si elle est calculable en temps polynômial dans le modèle BSS standard et les fractions rationnelles $Q_{\mathcal{N},x}$ ont des degrés polynômiaux et des coefficients de taille polynômiale.*

Preuve. Si F est calculable en temps $T(n)$ dans le modèle faible, elle est aussi calculable en temps $T(n)$ dans le modèle standard et les degrés et coefficients des fractions rationnelles $Q_{\mathcal{N},x}$ sont bornés par $T(n)$.

Si F est calculable en temps $T(n)$ dans le modèle standard avec des degrés et des tailles de coefficients bornés par $S(n)$, F est calculable dans le modèle faible en temps $T(n)S(n)$. □

Nous montrons maintenant que le modèle faible contient le modèle dans lequel les divisions sont interdites et une variable ne peut être multipliée que par une constante (et pas par une autre variable). Ce cas particulier est suffisant pour traiter les réseaux de neurones de la section 8.5.

Lemme 8.2 *Soit F une fonction calculable en temps polynômial dans le modèle BSS standard par un programme sans divisions et sans multiplications de variables entre elles. Alors F est calculable en temps polynômial dans le modèle BSS faible.*

Preuve. Les fractions rationnelles $Q_{\mathcal{N},x}$ sont en fait des polynômes puisqu'on effectue pas de divisions. Montrons par récurrence qu'à l'instant t le contenu d'une variable x est $P(\alpha, u)$ avec $d(P) \leq t+1$ et les coefficients de P bornés par 2^t.

Le résultat est clairement vrai pour $t = 0$. Supposons qu'il soit vrai à l'instant t. Au temps $t+1$, il suffit de considérer le cas d'une multiplication par une constante $x := \alpha_i y$ ou une addition $x := y + z$. Soit $y = P(\alpha, u)$ et $z = Q(\alpha, u)$:

- $d(\alpha_i P) = d(P) + 1$ et les coefficients ne changent pas;
- $d(P + Q) \leq \max(d(P), d(Q))$ et les coefficients sont au plus doublés.

□

8.2.2 Principaux résultats

Nous nous intéressons principalement au calcul de fonctions discrètes $F : \{0,1\}^+ \rightarrow \{0,1\}$. On note I le registre contenant l'entrée $u \in \{0,1\}^+$ au début du calcul (on pourrait de manière équivalente utiliser n registres booléens, avec n la taille de l'entrée). Toutes les autres variables sont initialisées à 0 (ceci n'est bien sûr pas vrai pour les variables contenant un choix non-déterministe s'il y en a). On note R le registre dans lequel le résultat est lu à l'arrêt du programme: $F(u) = 0$ si $R < 0$, $F(u) = 1$ si $R \geq 0$.

Introduisons maintenant quelques notations. La classe des fonctions $F : \{0,1\}^+ \rightarrow \{0,1\}$ calculables en temps polynômial dans le modèle BSS faible est P_{WRD} (pour «weak real discrete»); P_{WR} est la classe correspondant à des entrées réelles plutôt que booléennes. Les analogues non-déterministes de P_{WR} et P_{WRD} sont NP_{WR} et NP_{WRD}.

Notons que pour NP_{WRD}, les entrées sont booléennes mais les choix non-déterministes sont réels. Afin de comparer le modèle faible avec le modèle classique de la machine de Turing, on considère aussi la classe P_{WID} obtenues à partir de P_{WRD} en exigeant que les constantes $\alpha_1, \ldots, \alpha_p$ soient entières.

Lemme 8.3 $\mathrm{P}_{WID} = \mathrm{P}$.

Preuve. Si $F \in \mathrm{P}_{WID}$, il découle du lemme 8.1 que $F \in \mathrm{P}$ puisqu'au cours d'un calcul chaque registre de la machine de Turing «réelle» contient un entier de taille polynômiale.

On peut en fait prouver un résultat plus fort que la simple réciproque $\mathrm{P} \subset \mathrm{P}_{WID}$: la simulation se fait en temps linéaire avec un nombre constant de registres, car le fonctionnement d'une machine de Turing peut être codé par des entiers opérations affines sur des entiers. Pour voir ceci il suffit de remplacer dans les simulations des chapitres 5 ou 7 les nombres rationnels par des nombres entiers. On codera donc le nombre «décimal» en base quatre $x/4^p$ (avec $x \in \mathbb{N}$) par le couple d'entiers $(x, 4^p)$. $\square$

En faisant varier les trois paramètres que sont le coût des opérations (coût unitaire ou modèle faible), la nature des entrées et des constantes (réelles ou discrètes) on peut en fait définir huit classes de complexité. En plus des modèles BSS standard et faible et des machines de Turing du lemme 8.3, on peut retrouver par exemple les machines RAM habituelles en prenant un coût unitaire ainsi que des entrées et des constantes discrètes.

Le lemme 3 est en fait un cas très particulier du théorème 8.3 qui est le principal résultat de ce chapitre.

Théorème 8.3 $\mathrm{P}_{WRD} = \mathrm{P/poly}$.

Preuve. Il découle du lemme 8.2 et des résultats du chapitre 7 (ou bien de [45]) que $\mathrm{P/poly} \subset \mathrm{P}_{WRD}$. On montre dans la section 8.4 que $\mathrm{P}_{WRD} \subset \mathrm{P/poly}$ à l'aide d'un résultat sur les ensembles semi-algébriques établi dans la section 8.3. $\square$

Corollaire 8.1 *Si* $\mathrm{P}_{WR} = \mathrm{NP}_{WR}$ *la hiérarchie polynômiale s'effondre au deuxième niveau.*

Preuve. D'après le théorème 8.3, $\mathrm{P}_{WR} = \mathrm{NP}_{WR} \Longrightarrow \mathrm{NP} \subset \mathrm{P/poly}$ puisque $\mathrm{NP} \subset \mathrm{NP}_{WRD}$. Il est connu que si $\mathrm{NP} \subset \mathrm{P/poly}$ la hiérarchie polynômiale s'effondre au deuxième niveau (voir le chapitre 2). $\square$

8.2.3 Résultats de faisabilité et de difficulté

Dans ce chapitre comme dans le précédent, nous avons mis l'accent sur la résolution de problèmes discrets avec des «boîtes noires» réelles. Tel n'était sûrement pas le point de vue de Blum, Shub & Smale quand ils ont proposé leur modèle. Ils étaient beaucoup plus intéressés par les problèmes continus. Cependant notre point de vue n'est pas trop restrictif pour les résultats de difficulté, puisqu'un algorithme résolvant un problème continu doit en particulier fournir la bonne réponse pour des entrées discrètes. On dispose donc d'indications de difficulté pour les généralisations réelles des problèmes NP-complets de l'optimisation combinatoire tels que le voyageur de commerce réel de [6] ou le problème sac-à-dos réel. Plus précisément, si l'un de ces problèmes peut être résolu en temps polynômial dans le modèle BSS faible alors NP $\subset$ P/poly d'après le théorème 8.3, et la hiérarchie polynômiale s'effondre au deuxième niveau. Remarquons au passage qu'aucun résultat de complétude pour le voyageur de commerce réel ou des problèmes similaires n'est connu, que ce soit dans le modèle BSS standard ou dans le modèle faible.

A notre avis, la plupart (voire la totalité) des problèmes naturels qui peuvent être résolus en temps polynômial dans le modèle BSS standard peuvent en fait être résolus en temps polynômial dans le modèle faible. Ceci est clairement vrai pour des problèmes très faciles tels que la multiplications de polynômes ou de matrices. Voici deux autres exemples non triviaux.

Le premier problème consiste en la détermination du signe d'un polynôme réel P aux racines d'un autre polynôme réel Q. Des algorithmes résolvant ce problème sont décrits dans [14] et [37], et on montre dans ces articles que les conditions du lemme 8.1 sont satisfaites. Ces algorithmes fonctionnent donc en temps polynômial dans le modèle faible. Remarquons que l'algorithme de [14] est utilisé dans le même article comme une brique de base pour le calcul de la topologie d'une courbe algébrique réelle plane.

Notre second exemple est tout simplement la résolutions des systèmes d'équations linéaires. Ce problème peut évidemment être résolu en temps polynômial dans le modèle BSS standard par l'algorithme d'élimination de Gauss. Un résultat remarquable et probablement méconnu est démontré dans [18] : l'élimination de Gauss fonctionne également en temps polynômial sur une machine de Turing. La preuve est basée sur un résultat intermédiaire affirmant que les coefficients du système apparaissant au cours de la procédure d'élimination ont une forme très particulière : ils sont quotients de déterminants extraits de la matrice du système de départ. Mais en temps que fonction polynômiale des entrées, le déterminant d'une matrice est de degré polynômial (en fait sous-linéaire), et ses coefficients de taille polynômiale (en fait constante). D'après le lemme 8.1, ceci montre que l'élimination de Gauss fonctionne en temps polynômial dans le modèle faible.

8.3 Un résultat sur les ensembles semi-algébriques

La preuve du théorème 8.3 est basée sur le résultat suivant.

Théorème 8.4 *Soit $\mathcal{S} \subset \mathbb{R}^p$ un ensemble défini par un système d'inégalités de la forme*

$$P_i(x) > 0, \;\; i = 1 \ldots N$$

avec $P_i \in \mathbb{Z}[X_1, \ldots, X_p]$.

Soit D le degré maximum des P_i et S la taille maximum de leurs coefficients. Si $\mathcal{S} \neq \emptyset$, il existe un point rationnel $x \in \mathcal{S}$ de taille $s(x) \leq aSD^b$, où a et b sont des constantes dépendant uniquement p.

Ce résultat affirme donc que si $\mathcal{S} \neq \emptyset$ est défini par des «petits» polynômes, il contient nécessairement un «petit» point rationnel. Remarquons que le nombre d'inéquations n'intervient pas dans cette borne.

Avant de démontrer le théorème 8.4, nous allons présenter quelques résultats de calcul formel. Ces outils sont nécessaires pour la démonstration du lemme 8.6, mais le reste de la section peut être lu indépendamment.

8.3.1 Outils de base

La borne suivante est bien connue.

Lemme 8.4 (Cauchy, 1829) *Soit* $P(x) = \sum_{i=0}^{D} a_i x^i$ $(a_D \neq 0)$ *un polynôme complexe, et*

$$m = \inf\{|a_i|, i = 0 \ldots D, a_i \neq 0\}, M = \sup\{|a_i|, i = 0 \ldots D\}.$$

Si $\alpha \in \mathbb{C}$, $\alpha \neq 0$ *est une racine de* P *alors*

$$\frac{\frac{m}{m+M} < |\alpha| < m + M}{m.}$$

Le vecteur de signes d'une suite de polynômes $P_1, \ldots, P_N \in \mathbb{R}[X]$ en un point $x \in \mathbb{R}$ est le vecteur $(\epsilon(P_i(x)))_{1 \leq i \leq N} \in \{-1, 0, 1\}^N$, avec, par définition, $\epsilon(y) = 1$ si $y > 0$, 0 si $y = 0$, -1 si $y < 0$. Habituellement on représente une racine réelle α d'un polynôme $P \in \mathbb{R}[X]$ par un intervalle $]a, b[$ contenant α et aucune autre racine de P. Un codage purement symbolique a été proposé dans [12]: les racines de P sont caractérisées par le vecteur de signes des dérivées de P. Plus précisément, si $d(P) \leq D$, α est caractérisé par le vecteur de signes

$$\epsilon(\alpha) = (\epsilon(P^{(i)}(\alpha))_{1 \leq i \leq D} \in \{-1, 0, 1\}^D.$$

D'après le lemme de Thom, une racine est uniquement déterminée par son vecteur de signe: si $P(\alpha) = P(\beta) = 0$ et $\epsilon(\alpha) = \epsilon(\beta)$, alors $\alpha = \beta$. Il est de plus possible de comparer deux racines α et β en comparant uniquement leurs vecteurs de signes. Plus formellement, il existe une fonction de comparaison $c_D : \{-1, 0, 1\}^D \times \{-1, 0, 1\}^D \rightarrow \{-1, 0, 1\}$ indépendante de P telle que $c_D(\epsilon(\alpha), \epsilon(\beta)) = -1$ si $\alpha < \beta$, 0 si $\alpha = \beta$, 1 if $\alpha > \beta$. Cette fonction peut d'ailleurs être calculée par un algorithme très simple (les détails sont dans [12]).

Pour $a \in \mathbb{R}^{D+1}$, soit P_a le polynôme $P_a(X) = \sum_{i=0}^{D} a_i X^i$, et $r(P_a)$ le nombre de racines réelles distinctes de P_a. Le résultat-clé suivant est un cas particulier du “quantifier elimination subproblem” de [37]. On pourra trouver des bornes similaires dans [14].

Lemme 8.5 (Renegar) *Soient* $N_1, \ldots, N_M$ *des entiers positifs, et* $\epsilon_1, \ldots, \epsilon_M \in \{-1, 0, 1\}^N$ *des vecteurs de signes. L'ensemble* $S_{a,b} \subset \mathbb{R}^{N(D+1)} \times \mathbb{R}^{D+1}$ *des coefficients* $(a, b) = (a_1, \ldots, a_N, b) \in \mathbb{R}^{D+1} \times \mathbb{R}^{D+1} \times \ldots \times \mathbb{R}^{D+1}$ *satisfaisant les deux propriétés suivantes est semi-algébrique :*

(i) $r(P_b) = N_1 + \ldots + N_M$*;*

(ii) Pour tout $j \in \{1, ..., M\}$, N_j *est le nombre de zéros réels distincts de* P_b *en lesquels le vecteur de signes de* $(P_{a_i})_{1 \leq i \leq N}$ *est* ϵ_j*.*

De plus, $S_{a,b}$ *peut être défini par des (in)égalités polynômiales de degré* $(ND)^{O(1)}$ *avec des coefficients de taille* $(ND)^{O(1)}$*.*

8.3.2 Preuve du théorème 8.4

On note $\alpha = (\alpha_i)_{0 \leq i \leq r(P)+1}$ la suite des racines réelles ordonnées de $P \in \mathbb{R}[X]$, avec la convention $\alpha_0 = -\infty$ et $\alpha_{r(P)+1} = +\infty$.

Lemme 8.6 *Pour $D \geq 1$ et $0 \leq k \leq D$, soit $S_k \subset \mathbb{R}^{(D+1)} \times \mathbb{R}^{(D+1)}$ l'ensemble des coefficients (a, b) tels que*

$$r(P_a) \geq k \text{ et } P_b(x) > 0 \text{ pour } x \in]\alpha_k, \alpha_{k+1}[\tag{8.1}$$

avec α la suite des racines de P_a. S_k est un ensemble semi-algébrique défini par des polynômes de degré $D^{O(1)}$ dont les coefficients sont de taille $D^{O(1)}$.

Preuve.[1] Supposons par exemple que $0 < k < r(P_a)$ (les autres cas sont très semblables). On va appliquer plusieurs fois le lemme 8.5 à P_a, P_b, P_aP_b et leur dérivées. Un ensemble semi-algébrique est dit «petit» s'il est défini par des polynômes de degré $D^{O(1)}$ dont les coefficients sont de taille $D^{O(1)}$.

Etant donnés $\epsilon, \epsilon' \in \{-,1,0,1\}^D$, soit $A_{k,\epsilon,\epsilon'}$ l'ensemble des coefficients $a \in \mathbb{R}^{D+1}$ tels que $\epsilon(\alpha_k) = \epsilon$ et $\epsilon(\alpha_{k+1}) = \epsilon'$. En appliquant le lemme 8.5 à P_a et ses dérivées on voit que $A_{k,\epsilon,\epsilon'}$ est un petit ensemble semi-algébrique, puisque $a \in A_{k,\epsilon,\epsilon'}$ si et seulement si

1. $r(P_a) = r > k$;
2. Les vecteurs de signes des zéros de P_a sont $\epsilon_1, \ldots, \epsilon_r \in \{-,1,0,1\}^D$;
3. $\epsilon = \epsilon_k$, $\epsilon' = \epsilon_{k+1}$ et $c_D(\epsilon_i, \epsilon_j) = -1$ pour $i < j$.

Soit $B_{\epsilon,\epsilon'}$ l'ensemble des coefficients $(a, b) \in \mathbb{R}^{2(D+1)}$ tels que $P_b \neq 0$ en deux racines consécutives α et β de P_a dont les vecteurs de signes sont $\epsilon(\alpha) = \epsilon$ et $\epsilon(\beta) = \epsilon'$. Afin de comparer les racines de P_a et P_b nous considérons le polynôme P_aP_b, car ses racines sont celles de P_a et P_b. D'après le lemme 8.5 appliqué à P_a, P_aP_b et leurs dérivées, $B_{\epsilon,\epsilon'}$ est un petit ensemble semi-algébrique puisque $(a, b) \in B_{\epsilon,\epsilon'}$ si

1. $r(P_aP_b) = r \geq 2$;
2. les vecteurs de signes de la suite des $2D + 1$ polynômes

   $$\left(\left((P_aP_b)^{(i)}\right)_{1 \leq i \leq D}, P_a, \left(P_a^{(i)}\right)_{1 \leq i \leq D}\right)$$

   aux zéros de P_aP_b sont

   $$(\sigma_1, \alpha_1, \epsilon_1), \ldots, (\sigma_r, \alpha_r, \epsilon_r) \in \{-1,0,1\}^D \times \{-1,0,1\} \times \{-1,0,1\}^D;$$

3. $c_{2D}(\sigma_i, \sigma_j) = -1$ pour tout $i < j$, et il existe $l < r$ tel que $\alpha_l = \alpha_{l+1} = 0$, $\epsilon_l = \epsilon$ et $\epsilon_{l+1} = \epsilon'$.

1. Merci à Felipe Cucker pour son aide.

On a donc obtenu une condition sur (a, b) équivalente à la propriété "$P_b \neq 0$ sur $]\alpha_k, \alpha_{k+1}[$" (il suffit de prendre $\alpha = \alpha_k$ et $\beta = \alpha_{k+1}$ dans la définition de $B_{\epsilon,\epsilon'}$). Si $P_b \neq 0$ sur $]\alpha_k, \alpha_{k+1}[$, P_b est soit positif soit négatif sur cet intervalle. Plus précisément, le signe de P_b est celui de sa première dérivée non nulle en α_k. On définit donc C_ϵ comme étant l'ensemble des coefficients (a, b) tels que la première dérivée non nulle de P_b en α soit positive, α étant la racine de P_a de vecteur de signes ϵ. D'après le lemme 8.5 appliqué à P_a, P_b et leurs dŕivées, C_ϵ est un petit ensemble semi-algébrique puisque $(a, b) \in C_\epsilon$ si

1. $r(P_a) = r \geq 1$;
2. les vecteurs de signes de la suite des $2D+1$ polynômes $((P_a^{(i)})_{1 \leq i \leq D}, (P_b^{(i)})_{0 \leq i \leq D})$ aux zéros de P_a sont $(\epsilon_1, \sigma_1), \ldots, (\epsilon_r, \sigma_r)$;
3. Il existe j tel que $\epsilon_j = \epsilon$ et la première composante non nulle de σ_j est positive.

Le résultat découle maintenant de la relation

$$S_k = \bigcup_{\epsilon,\epsilon'} (A_{k,\epsilon,\epsilon'} \times \mathbb{R}^{D+1}) \cap B_{\epsilon,\epsilon'} \cap C_\epsilon.$$

□

La semi-algébricité de S_k est une conséquence directe du principe d'élimination des quantificateurs, car S_k est définie par la formule du premier ordre suivante (pour $k > 0$; l'autre cas est très similaire) :

$$\begin{array}{lll} \exists \alpha_1, \alpha_2, \ldots, \alpha_k & & \alpha_1 < \alpha_2 < \ldots < \alpha_k \\ & \wedge & (P_a(\alpha_1) = 0 \wedge \ldots \wedge P_a(\alpha_k) = 0) \\ & \wedge & \forall x \ (P_a(x) = 0 \Rightarrow (x > \alpha_k \vee x = \alpha_1 \vee \ldots \vee x = \alpha_k)) \\ & \wedge & \forall x \ ((\forall y \ (\alpha_k < y \leq x \Rightarrow P_a(y) \neq 0)) \Rightarrow P_b(x) > 0). \end{array}$$

Cet argument ne fournit malheureusement pas les bornes nécessaires.

Lemme 8.7 *Etant donné $a \in \mathbb{R}^{N(D+1)}$, soit $(P_{a_i})_{1 \leq i \leq N}$ la suite des polynômes tels que $P_{a_i} = \sum_{j=0}^{D} a_{ij} X^j$, et soit*

$$S_a = \{x \in \mathbb{R}, P_{a_1}(x) > 0 \wedge P_{a_2}(x) > 0 \wedge \ldots \wedge P_{a_D}(x) > 0\}.$$

L'ensemble des coefficients

$$\mathcal{C} = \{a \in \mathbb{R}^{N(D+1)}, \mathcal{S}_a \neq \emptyset\}$$

est un ensemble semi-algébrique de $\mathbb{R}^{N(D+1)}$ défini par des polynômes de degré $D^{O(1)}$ dont les coefficients sont de taille $D^{O(1)}$. Dans chacun de ces polynômes apparaissent au plus $3(D+1)$ des $N(D+1)$ variables a_{ij}.

Preuve. Si $\mathcal{S}_a \neq \emptyset$ pour un certain $a \in \mathbb{R}^{N(D+1)}$, il existe un intervalle maximal non-vide $]\alpha, \beta[\subset \mathcal{S}_a$. Si α ou β sont finis, ils sont racines de deux polynômes P_{a_i} et P_{a_j} ($i = j$ éventuellement). Donc

$$\mathcal{C} = \bigcup_{i,j,k} \bigcap_{l} \mathcal{C}_{ijkl}$$

où
$$\mathcal{C}_{ijkl} = \left\{ a \in \mathbb{R}^{N(D+1)}, P_{a_l} > 0 \text{ sur }]\alpha_{i,j,k}, \alpha_{i,j,k+1}[\right\},$$
$\alpha_{i,j,k}$ étant la $k^{\text{ème}}$ racine de $P_{a_i}P_{a_j}$. La conclusion du lemme est valable pour chacun des $\mathcal{C}_{ijkl}$ d'après le lemme 8.6, donc elle valable pour $\mathcal{C}$ également. □

Preuve du théorème 8.4. Par récurrence sur D. Pour $D = 1$, comme on l'a remarqué dans la preuve du lemme 8.7, il existe un intervalle maximal non-vide $]\alpha, \beta[\subset \mathcal{S}$ dont les extrémités, quand elles sont finies, sont racines de deux polynômes P_i et P_j. Il découle du lemme 8.4 que α et β ne peuvent pas être trop grand (lorsqu'ils sont finis). De même $\beta - \alpha$ ne peut pas être trop petit puisque ce nombre est racine d'un polynôme de degré $D^{O(1)}$ avec des coefficients entiers de taille $O(SD^{O(1)})$ [29]. Il y a donc des petits points rationnels dans $]\alpha, \beta[$.

Supposons maintenant le résultat vrai en dimension $p - 1$. Afin de montrer qu'il est valable pour un ensemble $\mathcal{S} \subset \mathbb{R}^p = \mathbb{R}^{p-1} \times \mathbb{R}$, $S \neq \emptyset$ on considère la projection $\mathcal{S}'$ de $\mathcal{S}$ sur $\mathbb{R}^{p-1}$. On va montrer que $\mathcal{S}'$ est semi-algébrique (c'est juste le principe de Tarski-Seidenberg) et définissable par de petits polynômes. Par hypothèse de récurrence S' contient alors un petit point x_0. Les polynômes $x_p \mapsto P_i(x_0, x_p)$ ont des petits coefficients, donc il existe un petit x_p tel que $(x_0, x_p) \in \mathcal{S}$. Voici maintenant les détails de ce raisonnement.

On considère chaque P_i comme un polynôme en x_p uniquement, en écrivant
$$P_i(x) = \sum_{j=0}^{D} a_{ij}(x')x_p^j$$
avec $x' = (x_1, x_2, \ldots, x_{p-1})$. Avec les notations du lemme 8.7, $x' \in \mathcal{S}'$ si et seulement si $a(x') = (a_{ij}(x'))_{i,j} \in \mathcal{C}$. Soient c et d les constantes telles que les degrés des polynômes décrivant $\mathcal{C}$ et la taille de leurs coefficients soient plus petits que cD^d. Les a_{ij} sont des polynômes de degré au plus D, donc S' peut être défini par des polynômes de degré au plus cD^{d+1}. Un calcul facile montre aussi que la taille de leurs coefficients est bornée par $c'SD^{d'}$ où c' et d' sont de nouvelles constantes. La décomposition de S' en ensembles semi-algébriques de base est de la forme
$$S' = \bigcup_{i=1}^{m} S'_i$$
avec
$$S'_i = \{x' \in \mathbb{R}^{p-1};\ Q_{i1} > 0 \wedge \ldots \wedge Q_{ij_i} > 0 \wedge R_{i1} = 0 \wedge \ldots \wedge R_{ik_i} = 0\}.$$
Comme $\mathcal{S}$ est ouvert, $\mathcal{S}'$ est ouvert également, donc $k_i = 0$ pour au moins l'un des $\mathcal{S}'_i$, disons pour $\mathcal{S}'_1$. Par hypothèse de récurrence, il existe des constantes a' and b' et un point $x_0 \in \mathcal{S}'_1$ de taille
$$s(x_0) \leq a'(c'SD^{d'})(cD^{d+1})^{b'}.$$
Il y a donc clairement des constantes a et b telles que $s(x_0) \leq aSD^b$. On peut obtenir une borne similaire pour x_p en substituant x_0 à x' dans les polynômes $P_i(x', x_p)$, supprimant les dénominateurs, puis en appliquant le théorème 8.4 en dimension 1. □

Une caractéristique assez inhabituelle du théorème 8.4 (et cruciale dans la preuve du théorème 8.3) est que le nombre d'inéquations n'intervient pas dans cette borne. Un résultat récent voisin du notre fait intervenir le degré total du système [40] (c'est-à-dire la somme des degrés des polynômes). On connaît aussi une borne supérieure sur le diamètre d'un ensemble semi-algébrique borné [15] et une borne inférieure sur la distance entre deux composantes connexes d'un ensemble semi-algébrique [22]. Il serait intéressant de déterminer si le degré total peut être remplacé par le degré maximum dans ces trois bornes. D'un autre côté, ces bornes sont meilleures que la notre car leur dépendance par rapport à la dimension p du problème est simplement exponentielle. Il n'est pas difficile de voir que dans le théorème 8.4 cette dépendance est en fait doublement exponentielle. Il serait également intéressant de déterminer si elle peut être remplacée par une dépendance simplement exponentielle.

8.4 Preuve du résultat principal

Dans cette section nous terminons la démonstration du théorème 8.3. On peut supposer sans perte de généralité que toutes les comparaisons sont de la forme «$x > 0$», où x est une variable. Avant d'aborder le cas général, nous supposons d'abord que la condition suivante est vérifiée :

(H) A chaque fois qu'un test $x > 0$ est effectué, x est différent de zéro.

Pour le cas général nous fournirons deux solutions. Dans la première, nous traitons les cas d'égalité par des techniques de calcul formel (bases de Gröbner ou théorème de l'élément primitif). Dans la seconde, nous montrons qu'il est possible de supposer que (H) est satisfaite sans perte de généralité.

Remarquons que si (H) est faux, les α_i sont liés par une relation algébrique.

Théorème 8.5 *Si $F : \{0,1\}^+ \to \{0,1\}$ peut être calculée en temps polynômial sous l'hypothèse* (H)*, alors $F \in$* P/poly.

Preuve. L'idée principale est de remplacer les constantes $\alpha_1, \ldots, \alpha_p$ par des approximations rationnelles $\overline{\alpha_1}, \ldots, \overline{\alpha_p}$. D'après la définition des fonctions de conseil, ces constantes ne doivent pas dépendre de toute l'entrée u, mais uniquement de sa longueur $n = |u|$. On exécute ensuite le programme avec ces nouvelles constantes : à l'instant t, le contenu de la variable x est maintenant $\overline{x}(t)$ au lieu de $x(t)$. Il faut vérifier les deux points suivants :

1. Le résultat du calcul est inchangé;

2. Le calcul se fait toujours en temps polynômial.

La première condition sera satisfaite si à tout instant t et pour toute variable x, $x(t) < 0 \Rightarrow \overline{x}(t) < 0$ et $x(t) > 0 \Rightarrow \overline{x}(t) > 0$. En effet, ces deux résultats impliquent que les résultats de tous les tests (y compris le test final décidant du résultat) seront les mêmes dans les deux programmes. Soit $T(n) = O(n^l)$ le temps de calcul dans le pire des cas pour des entrées de taille n. D'après le lemme 8.1, on obtient un système d'inéquations polynômiales sur $\overline{\alpha} = (\overline{\alpha_1}, \ldots, \overline{\alpha_p})$ du type

$$\mathcal{E} = \{P_{t,u}(\overline{\alpha}) > 0\}_{t \leq T(n), |u|=n}.$$

Une équation $P_{t,u} > 0$ apparaît dans $\mathcal{E}$ si l'instruction exécutée au temps t sur l'entrée u est un test sur la variable x, et dans ce cas

$$P_{t,u} = \mathrm{sign}(x(t)).R_{\mathcal{N},x}.S_{\mathcal{N},x},$$

$R_{\mathcal{N},x}$ et $S_{\mathcal{N},x}$ étant les polynômes tels que $Q_{\mathcal{N},x} = R_{\mathcal{N},x}/S_{\mathcal{N},x}$ (les fractions rationnelles $Q_{\mathcal{N},x}$ sont définies dans la section 8.2). Le système $\mathcal{E}$ est satisfaisable car α en est une solution par construction. On peut maintenant invoquer le théorème 8.4 et le lemme 8.1 pour montrer que α peut être remplacé par une approximation rationnelle pas trop précise, ce qui assurera la seconde condition.

D'après le lemme 8.1, les $P_{t,u}$ sont des polynômes de degré polynômial et leurs coefficients sont de taille polynômiale. Il découle du théorème 8.4 que $\mathcal{E}$ a une solution rationnelle $\overline{\alpha}$ de taille polynômiale. Les approximations $\overline{x}(t)$ peuvent donc être calculées en temps polynômial. □

Il est à noter que $\mathcal{E}$ contient en général un nombre exponentiel d'inéquations. Il est donc crucial que ce nombre n'intervienne pas dans la borne du théorème 8.4. Si par exemple D était la somme des degrés des polynômes plutôt que le degré maximum, la preuve ci-dessus ne marcherait pas.

8.4.1 Comment traiter les cas d'égalité

A partir de maintenant nous ne nous plaçons plus sous l'hypothèse (H). En plus de $\mathcal{E}$, on a donc un second système d'équations

$$\mathcal{E}' = \{P_{t,u}(\alpha) = 0\}$$

correspondant au cas «$x = 0$» dans un test. Ce système va être traité d'une manière purement symbolique. Pour chaque test «$x > 0$» dans le programme de départ, on va effectuer le test $P_{t,u}(\alpha) = 0$ par un algorithme décrit ci-dessous. S'il se trouve que $P_{t,u}(\alpha) = 0$, on peut fournir tout de suite une réponse négative au test «$x > 0$». Si $P_{t,u}(\alpha) \neq 0$, le signe de $P_{t,u}(\alpha)$ est le signe $P_{t,u}(\overline{\alpha})$ ($\overline{\alpha}$ étant toujours une petite solution de $\mathcal{E}$).

Afin d'effectuer les tests $P_{t,u}(\alpha) = 0$, on calcule les polynômes $P_{t,u} \in \mathbb{Z}[X_1, \ldots, X_p]$ (ceci peut être fait en temps polynômial d'après le lemme 8.1). Remarquons que $\mathcal{I} = \{P \in \mathbb{Z}[X], P(\alpha) = 0\}$ est un idéal. Or il est bien connu que les idéaux de $\mathbb{Z}[X]$ sont finiment engendrés, c'est-à-dire que

$$\mathcal{I} = \left\{ \sum_{i=1}^{m} Q_i G_i, Q_1, \ldots, Q_m \in \mathbb{Z}[X] \right\}$$

pour un certain système fini de générateurs $\mathcal{G} = (G_1, \ldots, G_m) \in \mathbb{Z}[X]^m$. Ceci découle du théorème de Hilbert sur les anneaux nœthériens (voir par exemple [28], chapitre 6). On peut supposer que $\mathcal{G}$ est connu puisqu'il est indépendant de l'entrée. Il reste à déterminer en temps polynômial si $P_{t,u}$ appartient à l'idéal engendré par $\mathcal{G}$. Ceci ne pose pas de difficulté si on suppose de plus que $\mathcal{G}$ est une base de Gröbner de $\mathcal{I}$ (voir [16] pour une introduction très lisible aux bases de Gröbner). Par définition d'une base de Gröbner, un polynôme $P \in \mathcal{I}$ est dans la forme appelée «forme réduite» par rapport à $\mathcal{G}$ si et seulement si $P = 0$. Afin de décider si un polynôme Q appartient à $\mathcal{I}$, il suffit donc de calculer sa forme réduite par rapport à $\mathcal{G}$ (qui est unique), et $Q \in \mathcal{I} \Leftrightarrow P = 0$. Il est bien connu que Q peut être calculé en temps polynômial.

En fait il n'est pas nécessaire de recourir aux bases de Gröbner pour résoudre ce problème. Bruno Poizat a trouvé une solution assez simple basée sur le théorème de l'élément primitif (communication personnelle) : soit k le degré de transcendance sur $\mathbb{Q}$ du corps $K = \mathbb{Q}[\alpha_1, \dots, \alpha_p]$. On peut supposer sans perte de généralité que $\alpha_1, \dots, \alpha_k$ est une base de transcendance de K. Les autres constantes $\alpha_{k+1}, \dots, \alpha_p$ sont algébriques sur $\mathbb{Q}[\alpha_1, \dots, \alpha_k]$. D'après le théorème de l'élément primitif [28],

$$K = \mathbb{Q}[\alpha_1, \dots, \alpha_k][\alpha_{k+1}, \dots, \alpha_p] = \mathbb{Q}[\alpha_1, \dots, \alpha_k][\beta]$$

où l'élément primitif $\beta \in \mathbb{R}$ est algébrique sur $\mathbb{Q}[\alpha_1, \dots, \alpha_k]$. Il existe donc des fractions rationnelles Q_j $(k+1 \leq j \leq p)$ telles que $\alpha_j = Q_j(\alpha_1, \dots, \alpha_k, \beta)$.

Afin d'effectuer un test $P_{t,u}(\alpha) \stackrel{?}{=} 0$, on substitue d'abord les Q_j aux α_j dans $P_{t,u}$. Ceci donne un nouveau test équivalent, de la forme $P(\alpha_1, \dots, \alpha_k, \beta) \stackrel{?}{=} 0$. Cette expression ne s'annule que si en tant que polynôme en β, $P(\alpha_1, \dots, \alpha_k, \beta)$ est un multiple du polynôme minimum M de β sur $\mathbb{Q}[\alpha_1, \dots, \alpha_k]$. Cette condition est très facile à vérifier : il suffit d'effectuer la division de P par M, et puisque $\alpha_1, \dots, \alpha_k$ sont algébriquement indépendants, le reste R est égal à zéro si ses coefficients (qui sont dans $\mathbb{Q}[\alpha_1, \dots, \alpha_k]$) sont identiquement nuls. Tout ce calcul se fait clairement en temps polynômial.

8.4.2 Comment se débarrasser des cas d'égalité

Cette troisième solution est basée sur la remarque évidente suivante : quand un test "$x > 0$" est effectué au cours de l'exécution d'un programme $\mathcal{P}$ d'une MTR, et s'il se trouve que $x \neq 0$, alors il existe $\epsilon > 0$ tel que soit $x > \epsilon$, soit $x < -\epsilon$. A n fixé, le nombre de tests effectués durant les calculs sur toutes les entrées de taille n est fini, donc on peut choisir le même ϵ pour tous ces tests. Pour chaque n, définissons un nouveau système d'inéquations $\mathcal{E}''$ portant sur $\overline{\alpha_1}, \dots, \overline{\alpha_p}$ et sur une nouvelle variable $\overline{\epsilon}$. On obtient $\mathcal{E}''$ à partir de $\mathcal{E}$ par les transformations suivantes :

1. ajout de l'inéquation $\overline{\epsilon} > 0$;

2. remplacement de l'inéquation associée à un test «$x > 0$» par l'inéquation associée au test :

   (a) $x - \overline{\epsilon} > 0$ si $x > 0$;

   (b) $x - \overline{\epsilon} < 0$ si $x \leq 0$.

$\mathcal{E}''$ est satisfaisable puisque $(\alpha_1, \dots, \alpha_p, \epsilon)$ en est une solution, donc il existe une «petite» solution $(\overline{\alpha_1}, \dots, \overline{\alpha_p}, \overline{\epsilon})$ d'après le théorème 8.4. Le programme obtenu en remplaçant les α_i par les $\overline{\alpha}_i$ et les tests «$x > 0$» par «$x > \overline{\epsilon}$» fonctionne en temps polynômial et est équivalent à $\mathcal{P}$.

8.5 Application aux réseaux de neurones récurrents

On utilise le modèle de calcul analogique à base de réseaux récurrents proposé dans [45]. La dynamique d'un réseau de n neurones est définie par :

$$x_i(t+1) = f_i\left(\sum_{j=1}^{n} a_{ij}x_j + \sum_{j=1}^{p} b_{ij}u_j(t) + c_i\right)$$

avec $a_{ij}, b_{ij}, c_i \in \mathbb{R}$. Les u_j sont des lignes d'entrée binaires : $u_j(t) \in \{0,1\}$. On n'a besoin ici que de deux lignes d'entrée u_1 et u_2. On peut calculer des fonctions de $\{0,1\}^+$ dans $\{0,1\}$ avec ces réseaux de la façon suivante. L'entrée est lue séquentiellement sur u_1 en partant de l'état initial $x(0) = 0$; u_2 est une ligne de validation qui reste active (dans l'état 1) jusqu'à ce que l'entrée ait été lue entièrement. La sortie est lue sur deux processeurs désignés à l'avance, mettons x_1 et x_2. Ce dernier est également un processeur de validation, actif uniquement lorsque le calcul est terminé, et alors $x_1(t)$ en contient le résultat.

Afin de pouvoir appliquer les résultats des sections précédentes, on suppose que les fonctions de sortie f_i sont affines par morceaux (et éventuellement discontinues).

Théorème 8.6 *Si $F : \{0,1\}^+ \to \{0,1\}$ peut être calculée en temps polynômial par un réseau de neurones récurrent, alors $F \in \mathrm{P/poly}$.*

Preuve. C'est une conséquence immédiate du théorème 8.3, puisque les réseaux récurrents peuvent être simulés avec des additions, des comparaisons et des multiplication par des constantes uniquement. □

Ce théorème généralise des résultats de [45], qui traitait uniquement le cas des fonctions de sorties continues. Il est également démontré dans cet article que les fonctions calculables en temps polynômial restent dans P/poly si les f_i sont lipschtziennes et approximables en temps polynômial (voir le chapitre 7 pour plus de détails; la plupart des fonctions continues utiles en pratique vérifient cette propriété). Ici il n'est même pas possible de prendre des polynômes de degré 2 pour les f_i, car des polynômes de degré exponentiel apparaîtraient au cours du calcul.

Un résultat voisin est présenté dans [30]: une famille de circuits acycliques de profondeur constante et de taille polynômiale utilisant des fonctions de sorties polynômiales par morceaux (définies par des paramètres réels) peut être simulée par une famille de circuits à seuils qui reste de profondeur constante et de taille polynômiale. Ce résultat et le théorème 8.6 sont complémentaires, puisqu'un réseau récurrent peut être transformé en réseau acyclique en «déroulant» le calcul pendant un temps fini : le théorème 8.6 s'applique à des réseaux de profondeur polynômiale définis par un nombre constant de paramètres réels, alors que le résultat de Maass est valable pour des réseaux de profondeur constante définis par un nombre polynômial de paramètres réels.

8.6 Application aux réseaux acycliques

Dans cette section, nous considérons des familles de réseaux du même type que dans [30] et montrons qu'elles peuvent être simulées pour des entrées-sorties booléennes par des familles de circuits à seuils avec une augmentation raisonnable de la profondeur et de la taille des réseaux. Contrairement à [30], nous obtenons des résultats pour des réseaux de profondeur plus que constante ($\log n$ dans le pire des cas et $n^{O(1)}$ dans le meilleur) grâce à l'hypothèse supplémentaire que la famille est uniforme, c'est-à-dire qu'elle peut être générée par une MTR en temps polynômial dans le modèle BSS faible.

Définition 8.4 *Un réseau de neurones acyclique est formé de portes g calculant des fonctions de la forme*

$$y = f_g(\sum_{i=1}^{k} w_i x_i - \theta) \tag{8.2}$$

où $x_1, \ldots, x_k$ sont les sorties des prédécesseurs de g. La fonction de sortie f_g est polynômiale par morceaux, ce qui signifie qu'il existe une suite finie $c_0 = -\infty < c_1 < \ldots < c_{l-1} < c_l = +\infty$ telle que f est polynômiale sur chaque intervalle $]c_i, c_{i+1}[$.

Il y a n portes d'entrée booléennes et une porte de sortie, qu'on suppose être une porte à seuil.

Définition 8.5 *Une famille de circuits $C = (C_n)_{n\geq 1}$ (où C_n a n entrées) est* P-*uniforme si :*

1. *il existe $d \in \mathbb{N}$ tel que pour chaque n et pour chaque fonction de sortie f_g de C_n, les degrés des morceaux polynômiaux pieces de f_g sont bornés par d;*
2. *il existe une MTR qui sur l'entrée n produit une description complète de C_n en temps polynômial en n dans le modèle BSS faible.*

La seconde condition implique en particulier que C_n est de taille polynômiale, et que chaque fonction de sortie est composée d'un nombre polynômial de morceaux polynômiaux.

Dans une famille de circuits P-uniforme, chaque paramètre W_i d'un circuit C_n est une fraction rationnelle des constantes $\alpha_1, \ldots, \alpha_p$ de la MTR qui génère la famille (ces paramètres sont les poids et les seuils des portes, et les paramètres des fonctions de sortie). En réduisant ces fractions au même dénominateur, on peut écrire

$$W_i = \delta P_i(\alpha_1, \ldots, \alpha_p) \tag{8.3}$$

avec $\delta = 1/Q(\alpha_1, \ldots, \alpha_p)$ et $P_i, Q \in \mathbb{Z}[X_1, \ldots, X_p]$. Nous sommes maintenant en position d'exhiber des familles de circuits qui peuvent être simulées efficacement.

Définition 8.6 *Soit C une famille* P-*uniforme de circuits acycliques engendrée par une MTR $\mathcal{M}$. Pour chaque n, chaque porte g de C_n et chaque entrée $(u_1, \ldots, u_n) \in \{0,1\}^n$, la sortie y_g de g est une fonction polynômiale $y_g = Q_g(\alpha_1, \ldots, \alpha_p, \delta)$ des paramètres de $\mathcal{M}$ et du coefficient δ défini par l'équation (8.3). On dit que la famille C est* sympathique *si les Q_g ont des degrés polynomiaux et des coefficients de taille polynômiale (en n).*

Théorème 8.7 *Une famille sympathique de circuits acycliques de profondeur $D(n)$ peut être simulée par une famille non-uniforme $(T_n)_{n\geq 1}$ de circuits à seuils de taille polynômiale et de profondeur $O(D(n))$. De plus, les poids des portes de T_n sont binaires (dans $\{-1, 1\}$).*

Preuve. Comme dans la preuve du théorème 8.3, on remplace pour chaque circuit C_n les paramètres $\alpha_1, \ldots, \alpha_p, \delta$ par des approximations rationnelles $\overline{\alpha}_1, \ldots, \overline{\alpha}_p, \overline{\delta}$. Ces approximations doivent satisfaire la contrainte suivante : pour chaque entrée $u \in \{0,1\}^n$ et pour chaque porte g de C_n, le morceau polynômial de f_g sélectionné dans le calcul de C_n sur l'entrée u est le même qu'avec les paramètres réels $\alpha_1, \ldots, \alpha_p, \delta$. D'après la définition 8.6 et le théorème 8.4, il existe une solution de taille polynômiale. Il découle donc de la définition 8.6 que tous les nombres rationnels apparaissant dans un calcul de C_n sur toute entrée $u \in \{0,1\}^n$ sont de taille polynômiale.

On peut donc conclure comme dans [30]: C_n peut être simulé par un circuit à seuil T_n de taille polynômiale et de profondeur $O(D(n))$ car les opérations arithmétiques de base (addition multiple, comparaison, et multiplication d'entiers) peuvent être effectuées par des circuits à seuils (à poids binaires) de taille polynômiale et de profondeur constante [10] (nous utilisons ici le fait que les degrés des fonctions de sortie sont uniformément bornés).

Pour terminer la construction il faut traiter les cas d'égalité. Le plus simple est de s'en débarrasser comme dans la section 8.4.2, en introduisant un nouveau paramètre $\overline{\epsilon}$ qui s'ajoute à $\overline{\alpha_1}, \ldots, \overline{\alpha_p}$ et à $\overline{\delta}$. De cette manière on évite le problème d'avoir à effectuer les calculs algébriques de la section 8.4.1 avec des circuits de profondeur constante. □

Corollaire 8.2 *Une famille* P*-uniforme* $C = (C_n)_{n \geq 1}$ *de circuits acycliques de profondeur* $O(\log n)$ *peut être simulée par une famille non-uniforme* $(T_n)_{n \geq 1}$ *de circuits à seuils de taille polynômiale et de profondeur* $O(\log n)$*. De plus, les portes de* T_n *ont des poids binaires.*

Preuve. Montrons que C est sympathique. Dans (8.3), les polynômes P_i ont des degrés polynomiaux et des coefficients de taille polynômiale. Ceci est dû au fait que C_n peut être construit en temps polynômial dans le modèle BSS faible, et que les P_i sont obtenus en réduisant au même dénominateur un nombre polynômial de fractions rationnelles.

Soit D le degré maximum des P_i, et D_h le degré maximum des fractions Q_g correspondant à une porte g telle que tout chemin entre une unité d'entrée et g soit de longueur au plus h. D'après (8.2), $D_{h+1} \leq D + d(D_h + D)$. Comme $D = n^{O(1)}$, $d = O(1)$ et $h = O(\log n)$, on en déduit que $D_h = n^{O(1)}$. Il n'est pas difficile de voir que les coefficients des Q_g sont eux aussi de taille polynômiale. La famille C est donc sympathique, et on peut appliquer le théorème 8.7. □

Corollaire 8.3 *Une famille de circuits* P*-uniforme de profondeur* $D(n)$ *utilisant des fonctions de sortie affines par morceaux peut être simulée par une famille non-uniforme* $(T_n)_{n \geq 1}$ *de circuits à seuils de taille polynômiale et de profondeur* $O(D(n))$*. De plus, les portes de* T_n *ont des poids binaires.*

Preuve. On procède comme dans la preuve du corollaire 8.2. Pour des fonctions de sortie affines par morceaux, la formule de récurrence $D_{h+1} \leq D + d(D_h + D)$ devient $D_{h+1} \leq D_h + 2D$, ce qui donne $D_h = n^{O(1)}$ pour $h = n^{O(1)}$. □

Chapitre 9

Conclusion

Il est probablement trop tôt pour évaluer l'impact qu'auront les recherches sur la puissance de calcul des réseaux de neurones. J'espère cependant avoir contribué à démontrer que ces systèmes peuvent être effectivement analysés, et que leur étude donne lieu à des développements techniquement non triviaux. Je conclurai par une petite liste de problèmes que cette thèse laisse ouverts. On peut espérer qu'ils susciteront des recherches ultérieures.

1. Etant donné une fonction $f : [0,1] \to [0,1]$ affine par morceaux à coefficients rationnels et un rationnel $x \in [0,1]$, peut-on décider si

   $$\exists t \in \boldsymbol{N}, f^{t+1}(x) = f^t(x) \ ?$$

   Autrement dit, on demande si la trajectoire de x atteint un point fixe en temps fini.

2. Les réseaux de neurones récurrents utilisant la sigmoïde standard sont-ils universels? Autrement dit, on demande s'il est possible de simuler une machine de Turing quelconque avec des systèmes évoluant suivant la règle $x(t+1) = F(x(t))$ où les composantes de la fonction $F : [-1,1]^n \to [-1,1]^n$ sont de la forme

   $$F_i(x) = \tanh(\sum_{j=1}^{n} w_{ij} x_j - \theta_i).$$

   Même question si les F_i sont des polynômes.

3. Borner la puissance des calculs non-déterministes dans le modèle BSS faible : NP_{WRD} est-il inclus dans NP/poly, PSPACE/poly, EXPTIME/poly...? A défaut, montrer qu'il existe des langages de $\{0,1\}^+$ qui n'appartiennent pas à NP_{WRD}.

   Remarquons qu'il est possible de répondre à ces questions pour un modèle encore plus faible dans lequel multiplications et divisions sont complètement interdites. On trouve que l'ensemble des langages NPdans ce modèle est exactement NP/poly [27]. Les mathématiques de ce problème sont beaucoup plus simples, car les objets géométriques qui interviennent sont des polyèdres, et non plus des ensembles semi-algébriques généraux.

4. Même problème dans le modèle BSS déterministe usuel : P_{RD} est-il inclus dans P/poly, PSPACE/poly, EXPTIME/poly...? A défaut, montrer qu'il existe des langages de $\{0,1\}^+$ qui n'appartiennent pas à P_{RD}. Même problème pour le non-déterminisme.

Bibliographie

[1] N. Alon, A. K. Dewdney, and T. J. Ott. Efficient simulation of finite automata by neural nets. *Journal of the ACM*, 38(2):495–514, April 1991.

[2] J.L. Balcázar, J. Diáz, and J. Gabarró. *Structural Complexity I.* EATCS Monographs on Theoretical Computer Science. Springer-Verlag, 1988.

[3] J.L. Balcázar, J. Diáz, and J. Gabarró. *Structural Complexity II.* EATCS Monographs on Theoretical Computer Science. Springer-Verlag, 1990.

[4] E.B. Baum and D. Haussler. What size net gives valide generalization ? *Neural Computation*, 1:151–160, 1989.

[5] R. Benedetti and J.-J. Risler. *Real algebraic and semi-algebraic sets.* Hermann, Paris, 1990.

[6] L. Blum, M. Shub, and S. Smale. On a theory of computation and complexity over the real numbers : NP-completeness, recursive functions and universal machines. *Bulletin of the American Mathematical Society*, 21(1):1–46, July 1989.

[7] A. Blumer, A Ehrenfeucht, D. Haussler, and M. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM*, 36(4):929–965, 1990.

[8] J. Bochnak, M. Coste, and M.-F. Roy. *Géométrie algébrique réelle.* Springer-Verlag, 1987.

[9] F. Bothelho and M. Garzon. On dynamical properties of neural networks. *Complex Systems*, 5(4):401–403, 1991.

[10] J. Bruck and K.-Y. Siu. On the power of threshold circuits with small weights. *SIAM J. Disc. Math.*, 4(3):423–435, August 1991.

[11] P. Collet and J.P. Eckmann. *Iterated maps on the interval as dynamical systems*, volume I of *Progress in Physics*. Birkhäuser, Boston, 1980.

[12] M. Coste and M.-F. Roy. Thom's lemma, the coding of real algebraic numbers and the topology of semi-algebraic sets. *Journal of Symbolic Computation*, 5:121–129, 1988.

[13] F. Cucker. $P_{\mathbb{R}} \neq NC_{\mathbb{R}}$. *Journal of Complexity*, 8:230–238, 1992.

[14] F. Cucker, L. Gonzalez Vega, and F. Rosselló. On algorithms for real algebraic plane curves. In *Effective Methods in Algebraic Geometry*, volume 94 of *Progress in Mathematics*, pages 63–87. Birkhäuser, 1991.

[15] F. Cucker and F. Roselló. On the complexity of some problems for the Blum, Shub & Smale model. In *Proceedings of Latin'92*, volume 583 of *Lecture Notes in Computer Science*, pages 117–129. Springer-Verlag, 1992.

[16] J. Davenport, Y. Siret, and E. Tournier. *Calcul formel : Systèmes et algorithmes de manipulations algébriques.* Masson, 1987.

[17] A. K. Dewdney. Threshold matrices and the state assignment problem for neural nets. In *Proceedings of the 8th SouthEastern Conference on Combinatorics, Graph Theory and Computing*, pages 227–245. Louisiana State University, Baton Rouge, La.

[18] M. Dowd. Gaussian elimination runs in polynomial time. Technical Report LCSR-TM-3, Laboratory fo Computer Science Research, Rutgers University, October 1982.

[19] M. Garzon and S.P. Franklin. Neural computability II. In *Proc. 3rd Int. Joint Conf. on Neural Networks, Wash. D.C.*, volume 1, pages 631–637, 1989.

[20] P. Goldberg and M. Jerrum. Bounding the Vapnik-Chervonenkis dimension of concept classes parameterized by real numbers, 1993. Submitted to COLT'93.

[21] Eric Goles and Servet Martinez. *Neural and automata networks : dynamical behavior and applications.* Mathematics and its applications. Kluwer Academic Publishers, 1990.

[22] J. Heintz, T. Krick, M.-F. Roy, and P. Solerno. Single exponential time algorithms for basic constructions in elementary geometry. In *Proceedings of the* 10^{th} *Int. Conf. of the Chilean Comp. Sc. Soc.*, 1990.

[23] J.E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

[24] J.J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. In *Proc. Nat. Acad. Sci. USA*, volume 79, pages 2554–2558, 1982.

[25] R. Karp and R. Lipton. Turing machines that take advice. *L'Enseignement Mathématique*, 28:191–209, 1982.

[26] Ker-I Ko. *Complexity Theory of Real Functions.* Birkhäuser, 1991.

[27] P. Koiran. Computing over the reals with addition and order. in preparation.

[28] S. Lang. *Algebra.* Addison-Wesley, 1965.

[29] R. Loos. Computing in algebraic extensions. In Buchberger, Collins, and Loos, editors, *Computer Algebra, Symbolic and Algebraic Computation*, pages 173–187. Springer-Verlag, 1982.

[30] W. Maass. Bounds for the computational power and learning complexity of analog neural nets. To appear in Proc. STOC'93.

[31] M. L. Minsky. *Computation: Finite and Infinite Machines.* Prentice Hall, Engelwood Cliffs, 1967.

[32] C. Moore. Unpredictability and undecidability in dynamical systems. *Physical Review Letters*, 64(20):2354–2357, May 1990.

[33] C. Moore. Generalized shifts: unpredictability and undecidability in dynamical systems. *Nonlinearity*, 4:199–230, 1991.

[34] B. K. Natarajan. *Machine Learning: A Theoretical Approach.* Morgan Kaufmann, 1991.

[35] C. Preston. *Iterates of piecewise monotone mappings on an interval*, volume 1347 of *Lecture Notes in Mathematics.* Springer-Verlag, 1988.

[36] J. Renegar. On the computational complexity and geometry of the first-order theory of the reals. part I. *Journal of Symbolic Computation*, 13(3):255–299, March 1992.

[37] J. Renegar. On the computational complexity and geometry of the first-order theory of the reals. part III. *Journal of Symbolic Computation*, 13(3):329–352, March 1992.

[38] D. Richardson. Tessellation with local transformations. *Journal of Computer and System Sciences*, 6:373–388, 1972.

[39] Y. V. Rogozhin. Seven universal Turing machines. *Mat. Issled.*, 69:76–90, 1982. (Russian).

[40] F. Ronga. Recherche de solutions d'inéquations polynômiales. In C. Hayat-Legrand and F. Sergeraert, editors, *Algorithmique, topologie et géométrie algébriques*, Astérisque 192, pages 11–16. Société Mathématique de France, 1990.

[41] W. Rudin. *Analyse réelle et complexe.* Masson, 1987.

[42] D. E. Rumelhart and J. L. McClelland. *Parallel Distributed Processing*, volume 1. MIT Press, 1986.

[43] H. T. Siegelmann and E. D. Sontag. Neural nets are universal computing devices. SYCON Report 91-08, Rutgers University, May 1991.

[44] H. T. Siegelmann and E. D. Sontag. On the computational power of neural nets. SYCON Report 91-11, Rutgers University, November 1991.

[45] H. T. Siegelmann and E. D. Sontag. Neural networks with real weights: analog computational complexity. SYCON Report 92-05, Rutgers University, September 1992.

[46] H. T. Siegelmann and E. D. Sontag. On the computational power of neural nets. In *Proc. Fifth ACM Workshop on Computational Learning Theory*, July 1992.

[47] H. T. Siegelmann and E. D. Sontag. Analog computation via neural networks. In *Proc. 2^{nd} Israeli Symposium on Theory of Computing and Systems*, 1993. to appear.

[48] S. Smale. Some remarks on the foundations of numerical analysis. *SIAM Review*, 32(2):211–220, June 1990.

[49] E. D. Sontag. Feedforward nets for interpolation and classification. *J. Comp. Syst. Sci.*, 45:20–48, 1992.

[50] K. Weihrauch. *Computability*. EATCS Monographs on Theoretical Computer Science. Springer-Verlag, 1987.

[51] E. D. Zeigler. Every discrete input machine is linearly simulatable. *Journal of Computer and System Sciences*, pages 161–167, 1973.

Annexe A

Bibliographie personnelle

Revues internationales

[a] M. Cosnard, P. Koiran, and H. Paugam-Moisy. Bounds on the number of units for computing arbitrary dichotomies by multilayer perceptrons. 1991. à paraître dans *Journal of Complexity*.

[b] P. Koiran. On the complexity of approximating mappings using feedforward networks. 1991. à paraître dans *Neural Networks*.

Conférences internationales

[c] M. Cosnard, P. Koiran, and H. Paugam-Moisy. Complexity issues in neural network computations. In *Proceedings of Latin'92*, volume 583 of *Lecture Notes in Computer Science*, pages 530–545. Springer Verlag, 1992.

[d] P. Koiran. Approximating of mappings and application to translation invariant networks. In *Proceedings of IJCNN'91 Singapore*, pages 2292–2298, 1991.

[e] P. Koiran. Function approximation using a partition of the input space. In *Proc. of IJCNN'92 Baltimore*, volume I, pages 883–887, 1992.

[f] M. Cosnard, M. Garzon, and P. Koiran. Computability properties of low-dimensional dynamical systems. In *Proc.* 10^{th} *STACS*, volume 665 of *Lecture Notes in Computer Science*, pages 365–373. Springer-Verlag, 1993. version abrégée de [h].

[g] M. Cosnard and P. Koiran. Relations between models of parallel abstract machines. Heinz Nixdorf seminar, Paderborn, November 1992. à paraître dans les *Lectures Notes in Computer Science*, Springer-Verlag.

Rapports de recherche

[h] P. Koiran, M. Cosnard and M. Garzon. Computability with low-dimensional dynamical systems. Rapport de recherche du LIP 92-31. soumis à *TCS*.

[i] P. Koiran. On the relations between dynamical systems and boolean circuits. Rapport de recherche du LIP 93-01.

[j] P. Koiran. Dynamics of discrete time, continuous state Hopfield networks. Rapport de recherche du LIP en préparation. soumis à *Neural Computation*.