Rapport de Stage
Apprentissage des paramètres pour réseaux de neurones modélisés en automates temporisés

Laetitia Laversa
Lieu : i3s - Equipe MDSC
Encadrants : Elisabetta De Maria, Cinzia Di Giusto

1er Septembre 2017

Table des matières

1 Introduction

Notre cerveau et tout notre système nerveux contient des neurones. Ces cellules spécifiques communiquent et permettent de transmettre des messages dans notre corps grâce à des courants électriques ou des composés chimiques appelés neurotransmetteurs. De façon simplifiée, les neurones sont reliés entre eux par des synapses, chaque neurone accumule de l'énergie envoyée par les neurones précédents dans le réseau et une fois un certain seuil atteint, il envoie à son tour un message aux suivants. Il existe deux types de synapses, les synapses excitatrices et les synapses inhibitrices. Lorsque le message provient d'une synapse excitatrice, le potentiel de membrane des neurones voisins augmente, tandis qu'une synapse inhibitrice le diminue. Nous nous interressons ici à la modélisation de neurones et, plus précisément, de réseaux de neurones, afin d'y étudier l'apprentissage de paramètres, et ce de différentes manières.

Pour ce faire, nous nous appuirons sur le logiciel Uppaal, choisi pour ses différentes fonctions, il contient en effet un outil de modélisation d'automates temporisés, de simulation ainsi qu'un model checker.

2 Modélisation d'un neurone et d'un réseau de neurones

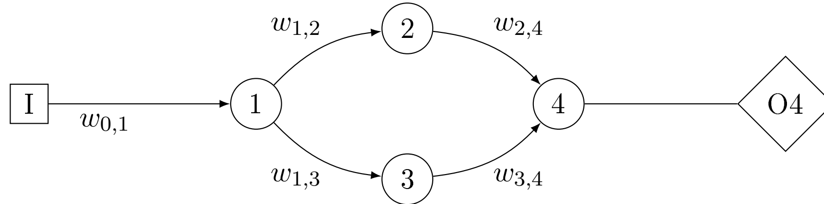
On représente un réseau de neurones par un graphe orienté et pondéré, où chaque noeud représente un neurone et chaque arc une synapse. Pour les synapses excitatrices, le poids est positif tandis que pour les synapses inhibitrices, le poids est négatif. Notons qu'une synapse inhibitrice ne peut pas devenir excitatrice et inversement. On retrouve aussi une ou plusieurs entrées qui génèrent des messages de façon aléatoires ou non, ainsi que des sorties qui facilitent la visualisation des émissions des neurones.

Lorsqu'un neurone envoie un message, on dit qu'il émet un spike. A ce moment-là, il envoie un signal aux neurones voisins, i.e., à travers tous les arcs sortants de ce neurone. Selon leur état, les neurones voisins pourront recevoir ce signal et accumuler l'énergie envoyée, qui correspond au potentiel de membrane de la synapse, sinon cette énergie sera perdue.

Nous nous appuirons sur deux systèmes à titre d'exemples. Notons que dans chaque système, les entrées sont représentées par des carrés, les neurones par des cercles et les sorties par des losanges. Les neurones et leur sortie sont numérotés, nous les nommerons donc N et O suivi de leur numéro. Quant aux synapses, un trait plein indique qu'elles sont excitatrices, et un trait en pointillé indique qu'elles sont inhibitrices.

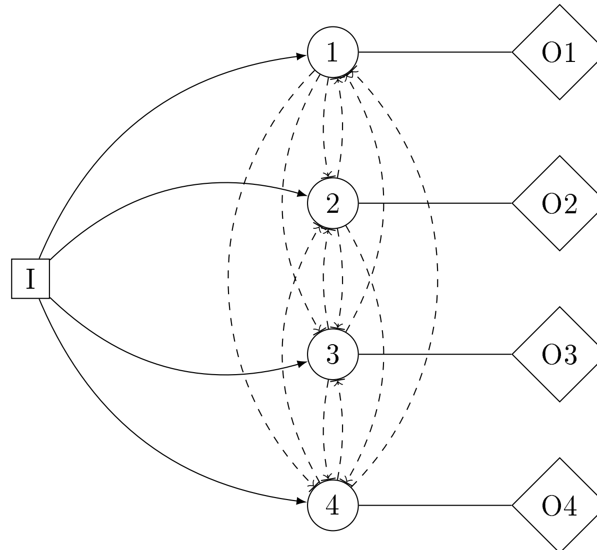
Exemple 2.1 (Diamant). *Dans ce système, nous avons une entrée I, des synapses excitatrices et une sortie $O4$ qui correspond aux émissions du $N4$. On retrouve sur chaque arc le poids associé.*

Figure 1 – Graphe représentant le système neuronal en diamant



Exemple 2.2 (Winner-takes-all). *Dans ce système, seules les synapses sortantes de l'entrée I sont excitatrices, les autres sont inhibitrices, et chaque sortie correspond aux emissions du neurone associé.*

Figure 2 – Graphe représentant le système neuronal "Winner-takes-all"



Interressons nous maintenant au fonctionnement du neurone.

Definition 1 (Neurone). *Un neurone est un défini par un tuple (a, p, T, θ, τ) tel que :*

— *T est le temps d'accumulation pendant lequel le neurone reçoit les signaux et à la fin duquel il regarde si l'énergie accumulée est suffisante*
— *a est la somme d'énergie accumulée pendant le temps d'accumulation*
— *p est le potentiel de membrane tel que :*

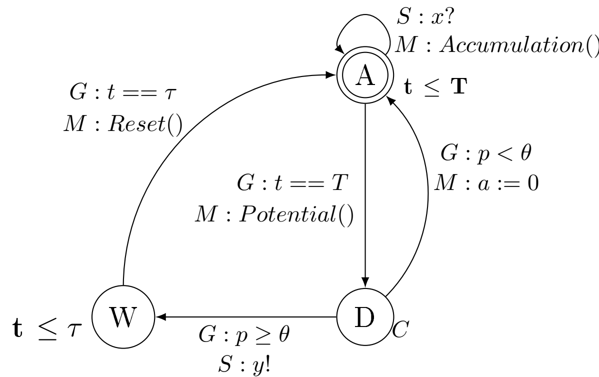
$$p_i = a + (p_{i-1} \cdot \lambda)$$

avec p_i le potentiel à la fin d'une période d'accumulation et p_{i-1} celui à la fin de la période précédente.
— *θ est le seuil que le potentiel doit atteindre pour émettre un spike*
— *τ est le temps de la période réfractaire qui correspond au temps après un spike où le neurone ne peut n'y émettre ni recevoir*

On modélise un neurone avec un automate temporisé.

Definition 2 (Automate d'un neurone)**.** *Soit N un automate temporisé représentant un neurone, $N = (L, l_0, X, \Sigma, Arcs, Inv)$ tel que :*
— *$L = \{A, D, W\}$: l'ensemble des états*
— *$l_0 = \{A\}$: l'état initial*
— *$X = \{t\}$: l'ensemble des horloges*
— *$\Sigma = \{x, y\}$: l'ensemble des signaux*
— *Arcs : l'ensemble des arcs*
— *Inv : l'ensemble des invariants des états*

FIGURE 3 – Automate temporisé du neurone



Sur la figure **??**, on retrouve les invariants en gras, ainsi que les arcs. Notons que les états "urgent" sont notés par un U, les états "committed" par un C et les états initiaux par deux cercles. Lorsqu'un signal est émis par un neurone il est suivi d'un !, tandis que lorsqu'il est reçu il est suivi d'un ?.

Sur les arcs on peut retrouver 3 éléments :

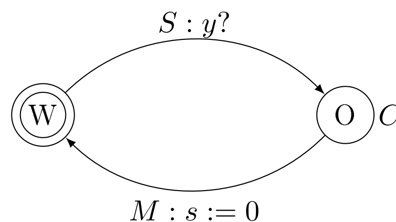
— une garde G qui doit être respectée pour pouvoir empreinter cet arc
— une synchronisation S qui signifie qu'un signal doit être envoyé lorsque cet arc est empreinté s'il est suivi d'un ! ou qu'il doit être reçu s'il est suivi d'un ?
— une mise à jour M qui doit être appliquée si l'arc est empreinté

Dans l'état initial A, le neurone accumule l'énergie reçue grâce aux neurones voisins. Puis avec une périodicité T, il va dans l'état D dans lequel on regarde si son potentiel de membrane a atteint son seuil θ. Si son potentiel est supérieur ou égal à θ, il émet un spike et passe dans l'état W dans lequel il doit attendre une période réfractaire τ pour retouner dans l'état A. Si son potentiel est inférieur à θ, il retourne en A et reviendra en D dans un temps T en ajoutant son nouveau cumul a à son précédent potentiel p qui aura été diminué, étant multiplié par λ.

A chaque neurone nous associons un automate temporisé qui représente sa sortie, et nous permet de visualiser ses émissions.

Definition 3. *Soit O un automate temporisé représentant la sortie d'un neurone, $O = (L, l_0, X, \Sigma, Arcs, Inv)$ tel que :*
— $L = \{W, O\}$ *: l'ensemble des états*
— $l_0 = \{W\}$ *: l'état initial*
— $X = \{s\}$ *: l'ensemble des horloges*
— $\Sigma = \{y\}$ *: l'ensemble des signaux*
— *$Arcs$: l'ensemble des arcs*
— *Inv : l'ensemble des invariants des états*

FIGURE 4 – Automate temporisé de la sortie d'un neurone



La sortie passe dans l'état O quand son neurone émet un spike, puis revient immédiatement dans l'état W en remettant son horloge à zéro.

3 Algorithmes de rétropropagation

Notre but est l'apprentissage de paramètres, c'est-à-dire l'obtention de valeurs pour les paramètres qui permettent à notre système d'avoir le comportement attendu. Ainsi, les algorithmes de rétropropagation ont pour but de modifier les valeurs des paramètres, ici les poids des synapses, jusqu'à l'obtention du résultat attendu. Nous en avons deux à notre disposition, ShouldHaveFired() si un neurone aurait dû émettre et ShouldNotHaveFired() si un neurone n'aurait pas dû émettre.

Le comportement attendu est exprimé en une version de CTL adpatée à Uppaal, celui-ci indique si un ou plusieurs neurones doivent émettre, ou non, un spike a un moment donné. Quelque soit l'algorithme, il change les poids des synapses entrantes et, par récursion, remonte dans le réseau en appliquant l'un ou l'autre algorithme selon le cas, et ce jusqu'aux neurones qui ont eu le comportement attendu.

Voici les deux algorithmes exprimés en pseudo-code.

Algorithme 1 : ShouldHaveFired(neurone)

si *visité(neurone)* **alors**
    stop;
fin
visité($neurone$) = vrai;
pour $p \in predecesseurs(neurone)$ **faire**
    $w_p \leftarrow w_{p,neurone}$;
    $f_p \leftarrow emis_recemment(p)$;
    $w_{p,neurone} \leftarrow w_{p,neurone} + \Delta$;
    si $w_p \geq 0$ **alors**
        si $\neg f_p$ **alors**
            ShouldHaveFired(p);
        fin
    sinon
        si f_p **alors**
            ShouldNotHaveFired(p);
        fin
    fin
fin

Algorithme 2 : ShouldNotHaveFired(neurone)

```
si visité(neurone) alors
    stop;
fin
visité(neurone) = vrai;
pour p ∈ predecesseurs(neurone) faire
    w_p ← w_{p,neurone} ;
    f_p ← emis_recemment(p) ;
    w_{p,neurone} ← w_{p,neurone} − Δ ;
    si w_p ≥ 0 alors
        si  f_p alors
            ShouldNotHaveFired(p);
        fin
    sinon
        si ¬f_p alors
            ShouldHaveFired(p);
        fin
    fin
fin
```

3.1 Apprentissage des paramètres par model checking

Dans un premier temps, on cherche les poids satisfaisant une formule de la manière suivante :

1. on demande au model checker d'Uppaal si les poids actuels permettent de satisfaire la formule
2. s'il répond non, on regarde quels neurones n'ont pas le comportement attendu
3. on applique l'algorithme, ce qui change certains ou tous les poids du système
4. on recommence en 1 jusqu'à ce que le model checker valide la formule

Voici un exemple de cette méthode.

Exemple 3.1 (Par model checking)**.** *Le système utilisé est celui en diamant et la formule CTL étudiée est*

$$EG(O4.O \Rightarrow O4.s \leqslant 20)$$

ce qui signifie que les spikes du N4 doivent être espacés au plus de 20 unités de temps, on appelle cela une pseudo-périodicité. Etant donné que le model checker ne peut pas vérifier la propriété à l'infini, on va modifier cette formule pour s'assurer que nous avons cette pseudo-périodicité au moins pour les 100 premiers spikes. On obtient donc :

$$EG(((O4.nbS4 < 100) \wedge (O4.O)) \Rightarrow (O4.s \leqslant 20))$$

où $O4.nbS4$ est le nombre de spikes émis par $N4$. Cette formule pose un problème, le model checker va nous répondre vrai si $N4$ n'émet aucun signal. Pour contrer cela, on va, en plus de cette formule, en vérifier une autre :

$$EF(O4.O)$$

qui nous certifiera que $N4$ a bien émis un spike.
Les poids de départ du système sont :

$w_{0,1}$	$w_{1,2}$	$w_{1,3}$	$w_{2,4}$	$w_{3,4}$
0.1	0.1	0.1	0.1	0.1

Les variables pour chaque neurone sont :

Neurone	T	θ	τ	λ
N1	2	0.35	3	7/9
N2	2	0.35	3	7/9
N3	2	0.35	3	7/9
N4	2	0.55	3	1/2

On a $\Delta = 0.1$, et ce pour toutes les expériences à venir.

Etape	Résultats	Modification	Appel
Etape 1	Formule fausse		
Etape 2	N4 n'a pas eu comportement attendu		
Etape 3	ShouldHaveFired(N4)	$w_{3,4} = 0.2$	SHF(N3)
		$w_{2,4} = 0.2$	SHF(N2)
	ShouldHaveFired(N3)	$w_{1,3} = 0.2$	SHF(N1)
	ShouldHaveFired(N1)	$w_{0,1} = 0.2$	
	ShouldHaveFired(N2)	$w_{1,2} = 0.2$	
Etape 1	Formule fausse		
Etape 2	N4 n'a pas eu comportement attendu		
Etape 3	ShouldHaveFired(N4)	$w_{3,4} = 0.3$	SHF(N3)
		$w_{2,4} = 0.3$	SHF(N2)
	ShouldHaveFired(N3)	$w_{1,3} = 0.3$	SHF(N1)
	ShouldHaveFired(N1)	$w_{0,1} = 0.3$	
	ShouldHaveFired(N2)	$w_{1,2} = 0.3$	
Etape 1	Formule satisfaite		

TABLE 1 – Déroulement d'un apprentissage par model checking

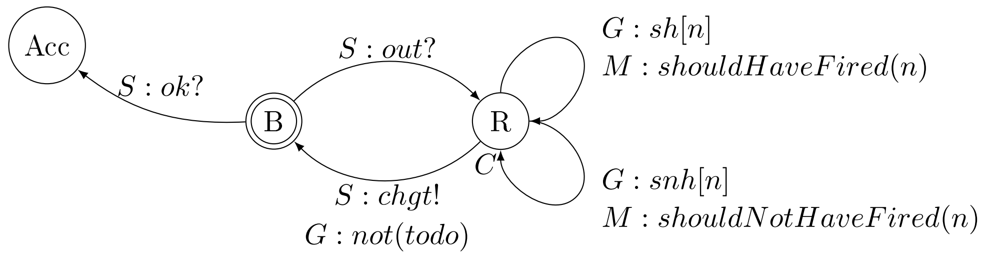
On sait donc que pour satisfaire cette formule dans ce système, il nous faut des poids tels que :

$w_{0,1}$	$w_{1,2}$	$w_{1,3}$	$w_{2,4}$	$w_{3,4}$
0.3	0.3	0.3	0.3	0.3

3.2 Apprentissage des paramètres par simulation

Dans un deuxième temps, l'idée est de faire appliquer l'algorithme par le réseau lui-même jusqu'à une stabilisation du modèle. Il s'agit donc d'apprentissage des paramètres par simulation. Pour ce faire, les sorties des neurones envoient des signaux à un nouvel élément, un automate représentant l'exécution de l'algorithme, que l'on nommera *algo*. Ainsi, une simulation est lancée et lorsqu'un neurone émet un spike non attendu, ou si au contraire un spike attendu n'a pas lieu, sa sortie envoie un signal à *algo* qui, grâce à des variables internes qui indiquent quel algorithme appliquer à quel neurone, applique l'algorithme et simule la récursion.

FIGURE 5 – Automate gérant l'application des algorithmes : *algo*



Détaillons les variables nécessaires à son fonctionnement.

- n, entier indiquant à quel neurone il faut appliquer l'algorithme
- *sh*[ ], tableau de booléen, indiquant à quels neurones il faut appliquer ShouldHaveFired()
- *snh*[ ], tableau de booléen indiquant à quels neurones il faut appliquer ShouldNotHaveFired()
- *todo*, booléen à vrai s'il reste au moins un neurone à qui il faut appliquer un des deux algorithmes

algo reste dans l'état R tant qu'un neurone doit appliquer l'un des deux algorithmes. Quand les tableaux *sh* et *snh* sont entièrement à $false$, *todo* est lui aussi à $false$ et *algo* retourne dans l'état B, en indiquant aux neurones, grâce au signal *chgt* , de remettre toutes leurs variables à zéro, de façon à recommencer une nouvelle simulation. Quand la formule est satisfaite, *algo* reçoit le signal *ok* qui le fait passer dans un état d'acceptation dans lequel il ne changera plus les poids qui ont permis la stabilisation.

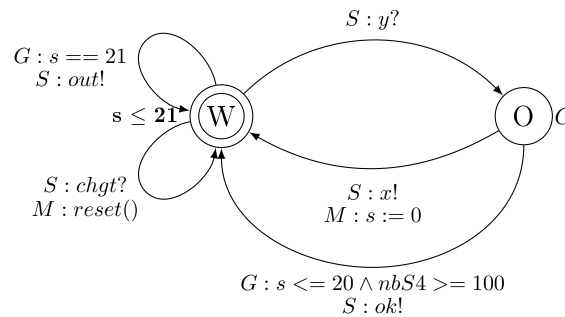
Il est aussi nécessaire de modifier quelque peu les automates des neurones et des sorties. Nous observerons comment dans l'exemple.

Exemple 3.2 (Par simulation). *Cet exemple s'appuie sur le système en diamant.*

Modification du système

Commençons par les modifications faites sur les automates. On attend du N4 qu'il émette un signal toutes les 20 unités de temps maximum. Il faut donc modifier sa sortie pour qu'elle puisse indiquer à algo que N4 aurait dû émettre. De plus, lorsque algo fini la récursion il doit prévenir tous les neurones et leur sortie que les paramètres ont été changés et qu'il faut mettre les horloges et autres variables à zéro, avec la fonction Reset(), par le signal chgt. On obtient pour la sortie O4 l'automate suivant.

FIGURE 6 – Automate temporisé de la sortie O4



On se doit de modifier l'automate du neurone, afin de recevoir les signaux de algo, qui devient le suivant. (voir figure **??***)*

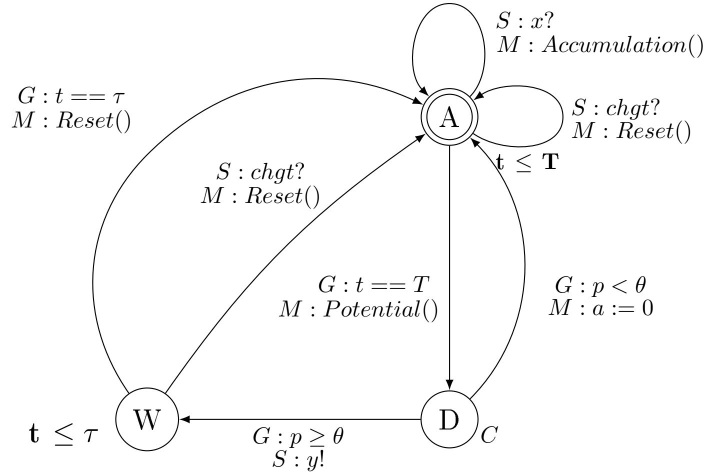
Expérimentation

Les paramètres et poids de départ sont les mêmes que précédemment. Nous observerons les changements d'états importants de nos automates (on ignore les allers-retour de A-D-A, les retours en A pour le neurone et W pour la sortie après un spike), le changement de certaines variables ainsi que l'application des algorithmes. On observe le résultat dans la table **??** *à la page* **??**.

Une fois l'état d'acceptation atteint, on peut regarder les poids obtenus.

$w_{0,1}$	$w_{1,2}$	$w_{1,3}$	$w_{2,4}$	$w_{3,4}$
0.2	*0.3*	*0.3*	*0.3*	*0.3*

FIGURE 7 – Automate temporisé des neurones N1, N2, N3 et N4



Temps	N1	O1	N2	O2	N3	O3	N4	O4	O4.s	nbS4	algo	Algorithme appliqué
18	W	O										
21								W	21		R	SHF(N4)
												SHF(N3)
												SHF(N2)
											B	
31	W	O										
42								W	21		R	SHF(N4)
												SHF(N2)
												SHF(N1)
												SHF(N3)
											B	
46	W	O										
53	W	O										
54			W	O	W	O						
56							W	O	14	1		
					...							
875							W	O	7	100	Acc	

TABLE 2 – Déroulement d'un apprentissage par simulation

On constate des résultats différents d'avec la première méthode, puisqu'en effet plusieurs jeux de poids peuvent permettre à notre formule d'être vraie pour un même système.

4 Ajout d'une priorité

Il est évident qu'un neurone dont on attend un certain comportement ne doit pas appliquer l'algorithme le poussant à avoir un comportement inverse. Admettons qu'on l'on attende d'un certain neurone qu'il émette un spike, il n'y a pas de raison qu'il applique l'algorithme ShoudNotHaveFired(), cela ne fera que ralentir le processus de stabilisation. Pour l'accélérer, on ajoute alors une nouvelle variable indiquant ce que l'on attend de chaque neurone et l'automate qui applique l'algorithme doit vérifier avant chaque application s'il est utile d'appliquer cet algorithme à ce neurone. On nommera "priorité" cette vérification puisqu'elle donnera priorité à ce que l'on attend du neurone sur ce que le système veut lui appliquer comme algorithme.

Le système en diamant n'est en réalité par concerné par ce problème, puisqu'à aucun moment on ne demande à un neurone de faire quelque chose de contraire au comportement attendu. En effet, on souhaite que N4 émette un signal, on ne souhaite donc lui appliquer que ShouldHaveFired(). Ainsi la priorité va empêcher l'application de ShouldNotHaveFired(). Mais il n'y a pas de raison pour laquelle le système pourrait demander l'applicatiion de ShouldNotHaveFired() à N4, la sortie O4 ne le demandant pas et N4 étant le prédécesseur d'aucun neurone. Cet exemple n'est donc pas interressant pour démontrer l'efficacité de cette priorité.

Dans le système Winner-takes-all, la formule à satisfaire est :

$$EG((O1.O \Rightarrow O1.s \leqslant 10) \wedge (\neg(O2.O) \wedge \neg(O3.O) \wedge \neg(O4.O)))$$

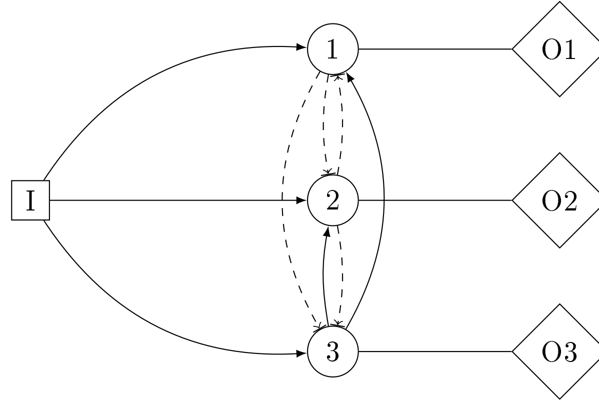
On souhaite donc que deux spikes consécutifs du N1 soient séparés de maximum 10 unités de temps, et que les neurones N2, N3, et N4 n'émettent pas.

Ainsi, lorsque l'on souhaite que N1 émette et qu'il ne le fait pas, on lui applique ShoudHaveFired() et comme les neurones prédécesseurs sont inhibiteurs, on ne leur applique pas ShouldHaveFired(). Cependant, si c'est un autre neurone qui émet, on lui applique ShouldNotHaveFired() et par récursion, on applique ShouldHaveFired() à tout les autres neurones. Cela ne pose pas problème pour le N1 dont on attend un spike, mais pose problème pour les deux autres prédécesseurs. La priorité devient donc interressante si on a des poids pour les synapses qui permettent aux neurones N2, N3, et N4 d'envoyer des spikes.

Un troisième exemple plus complexe nous permet de voir d'autant mieux l'utilité de cette priorité.

Exemple 4.1 (Système Winner-takes-all 2). *Il s'agit d'un système Winner-takes-all, mais avec des propriétés différentes pour les neurones. On a ici 3 neurones, dont 2 ont des synapses inhibitrices et 1 des synapses excitatrices.*

Figure 8 – Graphe représentant le système neuronal "Winner-takes-all 2"



La formule à satisfaire correspond à l'émission de spike toutes les 10 unités de temps maximum par le neurone 1 et l'absence d'émission par les neurones 2 et 3. On traduit cela en CTL par :

$$EG((O1.O \Rightarrow O1.s \leqslant 10) \wedge (\neg(O2.O) \wedge \neg(O3.O)))$$

Avec ce nouveau système Winner-takes-all 2, on attend un spike de N1 et s'il n'a pas lieu on lui appliquera alors ShouldHaveFired() ce qui impliquera l'application d'un ShouldHaveFired() à N3, ce qui est contradictoire avec le comportement attendu. La priorité aura donc un rôle important dans ce système avec cette formule.

4.1 Expériences

On va donc s'interresser aux systèmes Winner-takes-all 1 et 2. Pour le premier, on démarrera les expériences avec les poids de la matrice 1 de façon à s'assurer que les neurones N2, N3 et N4 puissent émettre, et pour le second, on démarrera avec les poids de la matrice 2.

Matrice 1

$$\begin{pmatrix} 0 & 0.9 & 0.9 & 0.9 & 0.9 \\ 0 & 0 & -0.1 & -0.1 & -0.1 \\ 0 & -0.1 & 0 & -0.1 & -0.1 \\ 0 & -0.1 & -0.1 & 0 & -0.1 \\ 0 & -0.1 & -0.1 & -0.1 & 0 \end{pmatrix}$$

Matrice 2

$$\begin{pmatrix} 0 & 0.1 & 0.1 & 0.1 \\ 0 & 0 & -0.1 & -0.1 \\ 0 & -0.1 & 0 & -0.1 \\ 0 & 0.1 & 0.1 & 0 \end{pmatrix}$$

Les paramètres de chaque neurone dans chaque système sont tels que :

Système Winner-takes-all 1

Neurone	T	θ	τ	λ
N1	2	0.75	3	1/2
N2	2	0.75	3	1/2
N3	2	0.75	3	1/2
N4	2	0.75	3	1/2

Système Winner-takes-all 2

Neurone	T	θ	τ	λ
N1	2	0.75	3	1/2
N2	2	0.75	3	1/2
N3	2	0.75	3	1/2

On lancera 2 expériences par jeux de données (système et présence ou non de la priorité).

4.2 Comparaison des résultats

On comparera, pour chaque système avec et sans la priorité, le temps pour atteindre les poids permettant la stabilisation (noté "Temps 1"), le temps pour atteindre les 100 spikes avec les propriétés attendues (noté "Temps 2"), le nombre de fois où l'automate *algo* est allé de l'état W à l'état B, ce qui correspond au nombre de fois où on a dû appliquer l'algorithme (sans compter les récursions), ainsi que les matrices de poids qui permettent la stabilisation. Voici les résultats des expériences :

Système	Priorité	Temps 1	Temps 2	Etat B	Poids
Winner 1	Sans	>1360		>500	
		>1406		>500	
	Avec	280	801	37	Matrice A
		708	1221	40	Matrice A'
Winner 2	Sans	342	873	16	Matrice B
		1158	1679	39	Matrice B'
	Avec	427	1104	5	Matrice C
		241	944	5	Matrice C

Table 3 – Temps et algorithmes nécessaires à une stabilisation pour les systèmes Winner-takes-all 1 et 2, avec et sans priorité

Matrice A

$$\begin{pmatrix} 0 & 0.9 & 0.3 & 0.3 & 0.2 \\ 0 & 0 & -0.9 & -0.9 & -0.9 \\ 0 & -0.1 & 0 & -0.9 & -0.9 \\ 0 & -0.1 & -0.9 & 0 & -0.9 \\ 0 & -0.1 & -0.9 & -0.9 & 0 \end{pmatrix}$$

Matrice A'

$$\begin{pmatrix} 0 & 0.9 & 0.3 & 0.3 & 0.3 \\ 0 & 0 & -0.9 & -0.9 & -0.9 \\ 0 & -0.1 & 0 & -0.9 & -0.9 \\ 0 & -0.1 & -0.9 & 0 & -0.9 \\ 0 & -0.1 & -0.9 & -0.9 & 0 \end{pmatrix}$$

Matrice B

$$\begin{pmatrix} 0 & 0.8 & 0.2 & 0.2 \\ 0 & 0 & -0.2 & -0.2 \\ 0 & -0.2 & 0 & -0.2 \\ 0 & 0.8 & 0.2 & 0 \end{pmatrix}$$

Matrice B'

$$\begin{pmatrix} 0 & 0.8 & 0.2 & 0.2 \\ 0 & 0 & -0.2 & -0.3 \\ 0 & -0.2 & 0 & -0.3 \\ 0 & 0.8 & 0.2 & 0 \end{pmatrix}$$

Matrice C

$$\begin{pmatrix} 0 & 0.6 & 0.1 & 0.1 \\ 0 & 0 & -0.1 & -0.1 \\ 0 & -0.1 & 0 & -0.1 \\ 0 & 0.6 & 0.1 & 0 \end{pmatrix}$$

Certaines expériences étant trop longues, on se permet d'arrêter la résolution une fois 500 tours d'algorithmes passés, ce qui nous informe déjà de la quantité de travail nécessaire à la stabilisation. On constate que l'ajout de la priorité accélère dans la plupart des cas le temps d'obtention de poids permettant un système stable, mais permet aussi de ne pas appliquer d'algorithmes inutiles, ce qui diminue donc la charge de travail. Les poids obtenus ne nous apportent, quant à eux, aucune information intéressante. L'ajout de cette priorité a donc les effets positifs attendus.

5 Amélioration du modèle du neurone

Notre modèle de neurone actuel simplifie amplement le fonctionnement d'un véritable neurone, nous allons alors le complexifier afin de s'approcher au plus près du comportement réel d'un neurone.

5.1 Emettre pendant la période d'accumulation

Pour commencer, le neurone n'attend pas que la période d'accumulation se termine pour émettre un spike. En effet, le potentiel est comparé au seuil θ à chaque signal reçu et dès que le potentiel dépasse ce seuil, le spike est émis. Cependant, la période d'accumulation T est concervée mais n'a plus le même objectif, elle ne détermine à présent que la fréquence à laquelle le potentiel est atténué, car, pour des raisons logicielles, une atténuation avant chaque accumulation, qui simulerait parfaitement la perte continue de potentiel, n'est pas possible, il nous faut donc une perte de potentiel discrète.

5.2 Recevoir pendant la période réfractaire

Un neurone peut aussi recevoir des signaux et accumuler de l'énergie pendant la période réfractaire. Ce qui implique aussi que le potentiel de membrane doit pouvoir être atténué avec toujours la même fréquence, que le neurone soit dans l'état A ou dans l'état W. Nous avons donc besoin de 2 horloges pour la modélisation, $t1$ sera utile pour la période d'accumulation, tandis que $t2$ sera utile pour la période réfractaire.

5.3 Emettre pendant la période réfractaire

La période réfractaire, d'une durée τ, est en fait divisée en deux parties, d'abord la période réfractaire absolue, d'une durée $\tau' < \tau$, pendant laquelle

le neurone peut seulement recevoir des signaux, puis la période réfratatire relative, d'une durée de $\tau - \tau'$ pendant laquelle le neurone peut recevoir mais aussi émettre des signaux, à condition que le potentiel de membrane soit très supérieur au seuil θ, et atteigne un nouveau seuil θ. Ce seuil θ' n'est pas constant et décroit de façon logarithmique après le spike, jusqu'à atteindre θ à la fin de la période réfractaire.

5.4 Nouveau modèle de neurone

Les modifications qui vont être apportées au modèle nécessitent de nouvelles fonctions, dont voici les explicitations.

— $Po()$ atténue le potentiel de membrane et met l'horloge $t1$ à zéro
— $Accu()$ ajoute le poids de la synapse du signal entrant au potentiel de membrane
— $AccuP()$ applique $Po()$ puis $Accu()$
— $PRA()$ remet à zéro l'horloge $t2$ et le potentiel p pour entrer dans la période réfractaire absolue
— $PRR()$ remet le seuil θ' à son maximum et met l'horloge $t3$ à zéro pour entrer dans la période réfractaire relative
— $Seuil()$ fait décroître le seuil θ' et met l'horloge $t3$ à zéro

On obtient alors le modèle de neurone suivant. (figure **??**)

Definition 4 (Automate d'un neurone, version 2). *Soit N un automate temporisé repésentant un neurone, $N = (L, l_0, X, \Sigma, Arcs, Inv)$ tel que :*

— $L = \{A, B, C, D, E, F, G, H, I\}$ *: l'ensemble des états*
— $l_0 = \{A\}$ *: l'état initial*
— $X = \{t1, t2, t3\}$ *: l'ensemble des horloges*
— $\Sigma = \{x, y\}$ *: l'ensemble des signaux*
— $Arcs$ *: l'ensemble des arcs*
— Inv *: l'ensemble des invariants des états*

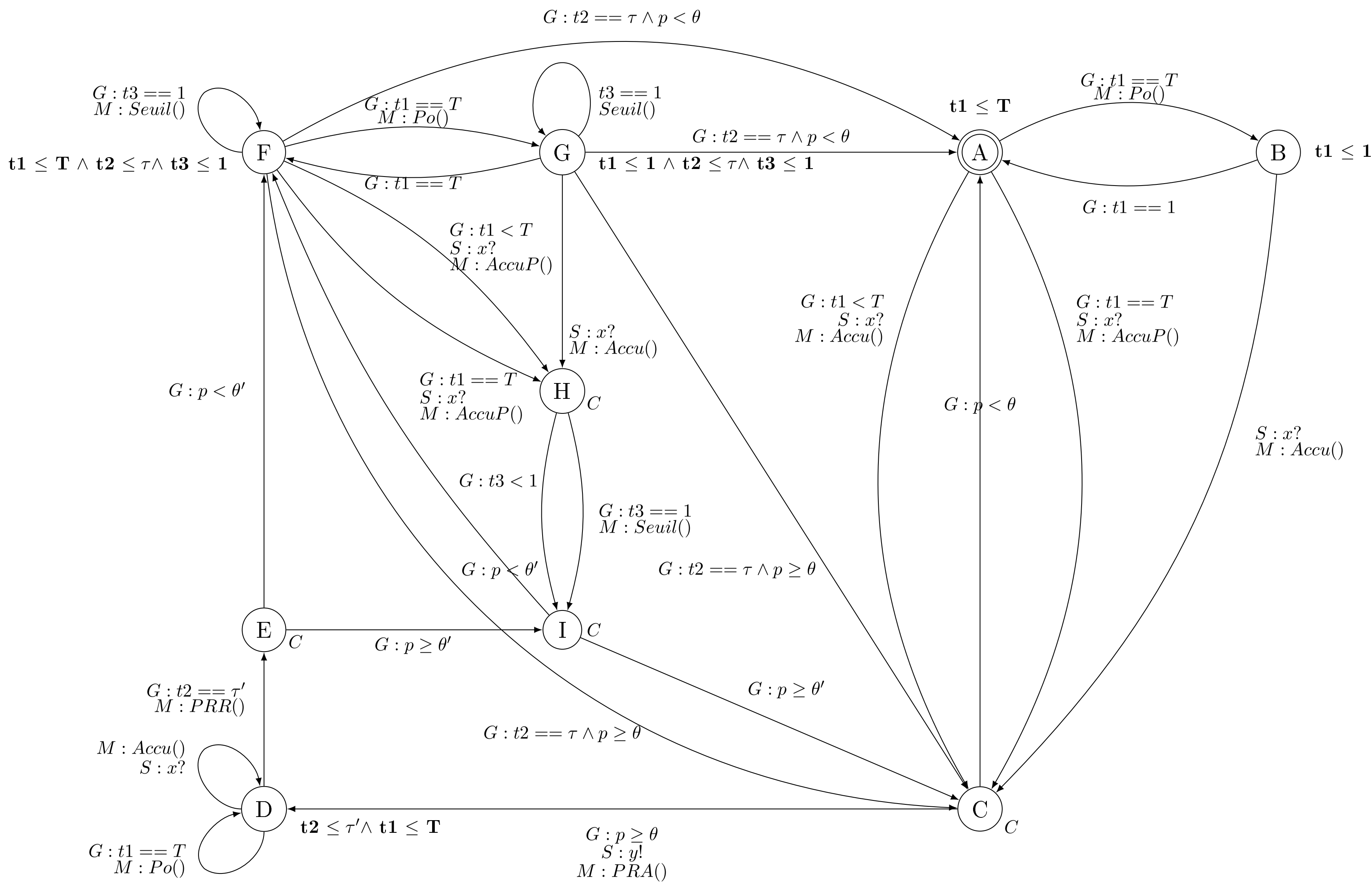


Figure 9 – Automate temporisé du neurone, version 2

Avec ce nouveau modèle, dans l'état initial A, le neurone atténue le potentiel de membrane avec une fréquence T en passant dans l'état B. On suppose que $T > 1$, et que donc le potentiel n'aura pas besoin d'être atténuer l'unité de temps suivant l'atténuation, et si ce n'est pas le cas, on pourra modifier les variables telles que $T = T \times 2$ et $\lambda = \lambda \times \lambda$. Si on reçoit un signal au moment où l'automate devait passer dans l'état B, on applique la fonction $AccuP()$. Si on reçoit un signal dans l'état B ou dans l'état A et que $t1 < T$, on applique la fonction $Accu()$. Lorsqu'une unité de temps s'est écoulée dans l'état B, on retourne dans l'état initial.

On peut à présent recevoir les signaux entrants et atténuer le potentiel dans l'état D (anciennement W), qui correspond à la période réfractaire absolue. A la fin de celle-ci, on passe dans l'état F qui correspond à la période réfractaire relative. On passe cependant d'abord par l'état "urgent" E dans lequel on regarde si le potentiel de membrane atteint le seuil θ'. L'état G a le même but que l'état B, c'est-à-dire nous permettre d'atténuer le potentiel correctement. Dans ces deux états, F et G, on peut donc recevoir des signaux et le seuil θ' décroit à chaque unité de temps pendant la période réfractaire grâce aux boucles. A chaque signaux reçus, on commence par accumuler l'énergie reçue en allant dans l'état H, puis on diminue le seuil si besoin en allant dans l'état I, et enfin on compare, la valeur du potentiel au seuil θ'. Si il est atteint, on passe dans l'état D pour émettre un spike. Enfin, quand $t2 = \tau$, on va en D pour émettre un spike si $p \geq \theta$, sinon à l'état initial A.

6 Comparaison des modèles de neurones

L'objectif est à présent de savoir quelle méthode d'apprentissage associée à quel modèle de neurone est le plus efficace. Pour cela, plusieurs expériences sont nécessaires.

6.1 Apprentissage par model checking

Comparons dans un premier temps la méthode de résolution par model checking sur deux réseaux en diamant, avec chacun un modèle de neurone différent. On a vu plus tôt le déroulement de la résolution pour le premier modèle (table **??**, page **??**), voyons à présent celui pour le second modèle (table **??**, page **??**).

Les paramètres et poids de départ sont les suivants. Notons que la valeur affichée pour θ' est celle au début de la période réfractaire relative et qu'elle decroît par la suite.

Neurone	T	θ	θ'	τ	τ'	λ
N1	2	0.35	1.75	3	1	7/9
N2	2	0.35	1.75	3	1	7/9
N3	2	0.35	1.75	3	1	7/9
N4	2	0.55	2.75	3	1	1/2

Etape	Résultats	Modification	Appel
Etape 1	Formule fausse		
Etape 2	N4 n'a pas eu comportement attendu		
Etape 3	ShouldHaveFired(N4)	$w_{3,4} = 0.2$	SHF(N3)
		$w_{2,4} = 0.2$	SHF(N2)
	ShouldHaveFired(N3)	$w_{1,3} = 0.2$	SHF(N1)
	ShouldHaveFired(N1)	$w_{0,1} = 0.2$	
	ShouldHaveFired(N2)	$w_{1,2} = 0.2$	
Etape 1	Formule fausse		
Etape 2	N4 n'a pas eu comportement attendu		
Etape 3	ShouldHaveFired(N4)	$w_{3,4} = 0.3$	SHF(N3)
		$w_{2,4} = 0.3$	SHF(N2)
	ShouldHaveFired(N3)	$w_{1,3} = 0.3$	
	ShouldHaveFired(N2)	$w_{1,2} = 0.3$	
Etape 1	Formule fausse		
Etape 2	N4 n'a pas eu comportement attendu		
Etape 3	ShouldHaveFired(N4)	$w_{3,4} = 0.4$	SHF(N3)
		$w_{2,4} = 0.4$	SHF(N2)
	ShouldHaveFired(N3)	$w_{1,3} = 0.4$	
	ShouldHaveFired(N2)	$w_{1,2} = 0.4$	
Etape 1	Formule satisfaite		

TABLE 4 – Déroulement d'un apprentissage par model checking avec le nouveau modèle de neurone

On sait donc que pour satisfaire cette formule dans ce système, il nous faut des poids tels que :

$w_{0,1}$	$w_{1,2}$	$w_{1,3}$	$w_{2,4}$	$w_{3,4}$
0.2	0.4	0.4	0.4	0.4

On constate que les poids sont différents de ceux obtenus avec le modèle de neurone précédent, ce qui est normal puisque leur fonctionnement est lui aussi différent.

6.2 Apprentissage par simulation

Interressons-nous à présent à la méthode de résolution par simulation. Nous avons vu précédemment cette méthode de résolution avec le système en

diamant(table **??**, page **??**). Concervons donc ce système, que nous modifions afin qu'il contienne notre nouveau modèle de neurone. Nous obtenons la table **??** de la page **??**.

On sait donc que pour satisfaire cette formule dans ce système, il nous faut des poids tels que :

$w_{0,1}$	$w_{1,2}$	$w_{1,3}$	$w_{2,4}$	$w_{3,4}$
0.2	0.4	0.4	0.5	0.5

Encore une fois, on constate des poids différents, mais, comme dit plus tôt, plusieurs poids peuvent satisfaire une même propriété.

6.3 Ajout de la priorité

Voyons à présent si l'ajout de la priorité a, comme sur le neurone précédent, un effet accélérateur sur la résolution. On peut imaginer que oui, puisque le fonctionnement du neurone ne rentre pas en jeu dans cette priorité qui permet seulement de ne pas appliquer d'algorithmes inutiles. Reprenons les mêmes conditions que dans l'expérience de la section 4.2 et comparons nos résultats à ceux de la table **??**, de la page **??**. On a donc comme paramètres et poids de départ :

Poids du système Winner 1

$$\begin{pmatrix} 0 & 0.9 & 0.9 & 0.9 & 0.9 \\ 0 & 0 & -0.1 & -0.1 & -0.1 \\ 0 & -0.1 & 0 & -0.1 & -0.1 \\ 0 & -0.1 & -0.1 & 0 & -0.1 \\ 0 & -0.1 & -0.1 & -0.1 & 0 \end{pmatrix}$$

Poids du système Winner 2

$$\begin{pmatrix} 0 & 0.1 & 0.1 & 0.1 \\ 0 & 0 & -0.1 & -0.1 \\ 0 & -0.1 & 0 & -0.1 \\ 0 & 0.1 & 0.1 & 0 \end{pmatrix}$$

Système Winner-takes-all 1

Neurone	T	θ	θ'	τ	τ'	λ
N1	2	0.75	3.75	3	1	1/2
N2	2	0.75	3.75	3	1	1/2
N3	2	0.75	3.75	3	1	1/2
N4	2	0.75	3.75	3	1	1/2

Système Winner-takes-all 2

Neurone	T	θ	θ'	τ	τ'	λ
N1	2	0.75	3.75	3	1	1/2
N2	2	0.75	3.75	3	1	1/2
N3	2	0.75	3.75	3	1	1/2

Temps	N1	O1	N2	O2	N3	O3	N4	O4	O4.s	nbS4	algo	Algorithme appliqué
14	W	O										
21								W	21		R	SHF(N4)
												SHF(N2)
												SHF(N1)
												SHF(N3)
											B	
23	W	O										
27	W	O										
31	W	O	W	O	W	O						
35	W	O										
39	W	O										
42								W	21		R	SHF(N4)
												SHF(N2)
												SHF(N3)
											B	
46	W	O										
50	W	O	W	O	W	O						
54	W	O										
58	W	O	W	O	W	O						
62	W	O										
63								W	21		R	SHF(N4)
												SHF(N2)
												SHF(N3)
											B	
67	W	O	W	O	W	O						
71	W	O	W	O	W	O						
75	W	O	W	O	W	O						
79	W	O	W	O	W	O						
83	W	O	W	O	W	O						
84								W	21		R	SHF(N4)
											B	
88	W	O	W	O	W	O	W	O	4	1		
					...							
488	W	O			W	O	W	O	4	100	Acc	

TABLE 5 – Déroulement d'un apprentissage par simulation avec le nouveau modèle de neurone

Système	Priorité	Temps 1	Temps 2	Etat B	Poids
Winner 1	Sans	424	721	96	Matrice D
		1996	2293	467	Matrice D'
	Avec	89	386	12	Matrice E
		50	346	20	Matrice E
Winner 2	Sans	218	614	27	Matrice F
		58	454	6	Matrice F'
	Avec	38	830	3	Matrice G
		36	828	3	Matrice G

TABLE 6 – Temps et algorithmes nécessaires à une stabilisation pour les systèmes Winner-takes-all 1 et 2 avec le nouveau modèle de neurone

Matrice D
$$\begin{pmatrix} 0 & 0.6 & 0.3 & 0.3 & 0.3 \\ 0 & 0 & -0.7 & -0.7 & -0.7 \\ 0 & -0.4 & 0 & -0.7 & -0.7 \\ 0 & -0.4 & -0.7 & 0 & -0.7 \\ 0 & -0.4 & -0.7 & -0.7 & 0 \end{pmatrix}$$

Matrice D'
$$\begin{pmatrix} 0 & 0.6 & 0.2 & 0.2 & 0.3 \\ 0 & 0 & -0.8 & -0.8 & -0.7 \\ 0 & -0.4 & 0 & -0.8 & -0.7 \\ 0 & -0.4 & -0.8 & 0 & -0.7 \\ 0 & -0.4 & -0.8 & -0.8 & 0 \end{pmatrix}$$

Matrice E
$$\begin{pmatrix} 0 & 0.9 & 0.3 & 0.3 & 0.3 \\ 0 & 0 & -0.7 & -0.7 & -0.7 \\ 0 & -0.1 & 0 & -0.7 & -0.7 \\ 0 & -0.1 & -0.7 & 0 & -0.7 \\ 0 & -0.1 & -0.7 & -0.7 & 0 \end{pmatrix}$$

Matrice F
$$\begin{pmatrix} 0 & 0.5 & 0.3 & 0.3 \\ 0 & 0 & -0.2 & -0.2 \\ 0 & -0.2 & 0 & -0.2 \\ 0 & 0.5 & 0.3 & 0 \end{pmatrix}$$

Matrice F'
$$\begin{pmatrix} 0 & 0.5 & 0.2 & 0.3 \\ 0 & 0 & -0.1 & -0.2 \\ 0 & -0.1 & 0 & -0.2 \\ 0 & 0.5 & 0.2 & 0 \end{pmatrix}$$

Matrice G
$$\begin{pmatrix} 0 & 0.4 & 0.1 & 0.1 \\ 0 & 0 & -0.1 & -0.1 \\ 0 & -0.1 & 0 & -0.1 \\ 0 & 0.4 & 0.1 & 0 \end{pmatrix}$$

On obtient comme résultats la table **??**.

Comme avec les résultats précendents, il est facile de constater que cette priorité accélère l'obtention de poids rendant le système stable et permet de diminuer le nombre de fois où l'algorithme a été appliqué. Cependant, on constate ici qu'avec ce nouveau modèle de neurone et la priorité, les poids obtenus sont toujours les mêmes pour un système donné (Matrices E et G). Pour terminer, une comparaison avec les résultats de la table **??** est possible. Dans un premier temps, on observe que les poids obtenus sont différents, et, comme dit plus tôt, cela s'explique par le fonctionnement différent du neurone. Dans un second temps, on constate qu'avec le nouveau modèle de neurone, l'obtention de ces poids est toujours plus rapide qu'avec l'an-

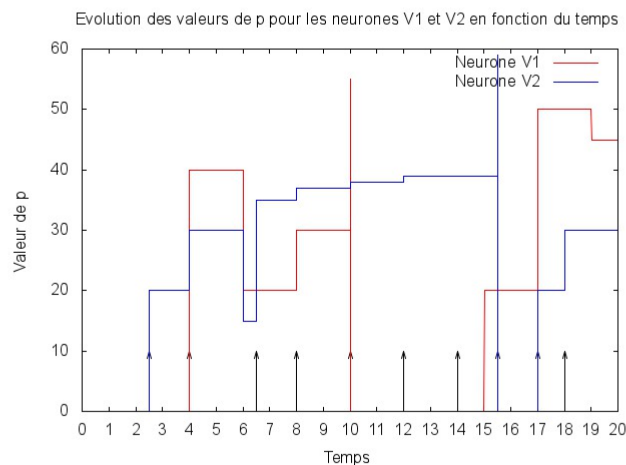
cien modèle. De même, le nombre d'algorithmes appliqués est aussi toujours inférieur.

6.4 Poids différents

On a observé lors de toutes nos expériences que les poids obtenus sont différents selon le neurone utilisé, et notamment, dans le système en diamant, que le neurone avait besoin de poids plus hauts pour atteindre son seuil. Il est alors légitime de se demander pourquoi. Pour répondre à cette question, on observe la valeur du potentiel de membrane de deux neurones, l'un correspondant au premier modèle, l'autre au second, reliés à la même entrée, qui envoie des messages aléatoirement. Les neurones ont les mêmes paramètres, afin de pouvoir être comparés, et les poids des synapses sont eux aussi identiques. On a en effet pour les deux neurones (θ' et τ' ne concernant que le deuxième neurone) :

T	λ	θ	θ'	τ	τ'	Poids
2	1/2	0.55	2.75	3	1	0.2

On obtient la courbe suivante.



Les flèches représentent les moments où l'entrée a envoyé un message. On comprends alors que le premier modèle de neurone, en rouge, a un décallage entre le moment où il reçoit le message et le moment où il accumule l'énergie, dû à la période d'accumulation. De ce fait, le moment où il aurait dû perdre l'énergie est lui aussi décallé, et il peut alors atteindre un potentiel qu'il n'aurait pas dû atteindre. Le nouveau modèle de neurone, en bleu, est plus proche de la réalité et perd son énergie de façon plus régulière. Ce qui explique donc la nécessité de poids plus importants pour atteindre le seuil

attendu pour le nouveau neurone.

Pour conclure cette partie, on peut donc dire que ce nouveau modèle de neurone nous permet donc d'avoir un apprentissage de paramètres plus performant que le précédent, même s'il peut nécessiter des poids plus importants, ce que l'on remarque notamment grâce à la méthode par simulation.

7 Conclusion

Pour conclure ce rapport, la méthode d'application par simulation de l'algorithme de rétropropagation est plus simple à appliquer par l'utilisateur, cependant si la rédaction d'un script faisant le travail de l'utilisateur (demandes au model checker si la propriété est vraie et changements des poids) pourrait palier à cela, elle ne serait pas suffisante étant donné que certaines formules ne sont parfois ni validées ni infirmées par le model checker, demandant trop de mémoire. De plus, il est facile de constater que l'évolution du modèle de neurone permet d'accélérer et de simplifier l'obtention de poids permettant de satisfaire une propriété donnée. Le travail à effectuer par le suite pourrait être d'améliorer la résolution par model checker et l'analyse de l'impact des valeurs des paramètres du neurone sur l'apprentissage, tels que λ, θ ou τ.