REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mohamed Khider – BISKRA
Faculté des Sciences Exactes, des Sciences de la Nature et de la Vie
Département d'informatique

N° d'ordre : SIOD1/M2/2021

Mémoire

Présenté pour obtenir le diplôme de master académique en

Informatique

Parcours : **Système d'Information Optimisation et Décision (SIOD)**

Modélisation des réseaux de neurones impulsionnels par des automates temporisés

---

Par :

BERDOUD SOFIANE

Soutenu le ../../…. devant le jury composé de :

Nom Prénom	grade	Président
Nom Prénom : DR. Djaber Khaled	grade	Rapporteur
Nom Prénom	grade	Examinateur

Année universitaire 2020-2021

Remerciements

Avant tout, notre sincère louange à ALLAH le tout puissant qui nous a donné la fois, la volonté, la sante et la patience afin d'accomplir ce mémoire.

Nous remerciements les plus distingués à Mr. djaber khaled qui nous a honorés par son encadrement, par sa présence toujours avec nous, pour sa direction, son orientation, sa modestie, ses conseils et toutes ces remarques constructives pour le bon déroulement de notre projet.

Nous remercions beaucoup l'ensemble des enseignements du département d'informatique pour la formation qu'ils nous ont assurée tout le long de notre cursus universitaire.

Dédicace

A

Mes chers parents,

Mes frères et sœurs,

Mes enseignants et mes chers amis,

Les Mots Clés

RN : Réseau de neurone

NN : Neural network

RNA : Réseau de neurone artificiel

ANN : Artificial neural network

RNN : Réseau de neurone récurrent

RNC : Réseau de neurone à convolution

RNI : Réseau de neurone impulsionnel

SNN : Spiking neural network

IF : Integrate and fire

LIF : Leaky integrate and fire

HH : Hodgkin-huxley

CTL : computation Tree Logic

TCTL : Timed Computation Tree Logic

AT : automate temporisé

Résumé

Les réseaux de neurones artificiels RNA sont de plus en plus utilisés dans le monde de l'informatique. Inspirés du cerveau humain, les RNA permettent la compréhension de quelques phénomènes naturels. Les réseaux de neurones impultionnels RNI sont la troisième génération des RNA. Ils sont dotés d'un pouvoir expressif plus large qui nous permet de modéliser et simuler plus de phénomènes. Leur inconvénient est le taux modéré d'exactitude. Comparé aux autres techniques, le Model Checking est une technique exhaustive et complètement automatique. Cette technique repose sur les automates temporisés AT et la logique temporelle TCTL. L'inconvénient de cette technique est l'explosion de l'espace d'états qui nécessite beaucoup de ressources à savoir l'espace mémoire et les opérations de calcul. Les deux méthodes adoptent un modèle discret de temps. Le but de ce travail est de combiner les deux méthodes pour palier aux problèmes lié à chacune d'elles. La modélisation des RNI par les automates temporisés nous permet de profiter du pouvoir expressif des RNI et l'exactitude de Model Checking. UPPAAL est l'un des Model Checkers qui utilise les ATs et la logique TCTL. Cet outil sera utilisé pour une telle modélisation.

Mots clés: Réseaux de neurone artificiel, Réseau de neurone impulsionnel, automates temporisés, modélisation, simulation, Model Checking, Model Checker, UPPAAL.

Abstract

Artificial Neural Networks ANN are increasingly used in the computing sciences. Inspired by the human brain, ANNs allow the understanding of some natural phenomena. SNN Spiking Neural Networks are the third generation of ANNs. They are endowed with a broader expressive power that allows us to model and simulate more phenomena. Their downside is the moderate rate of accuracy. Compared to other techniques, Model Checking is a exhaustive and completely automatic technique. This technique is based on Timed Automata TA and TCTL timed logic. The disadvantage of this technique is the explosion of state space which requires a lot of resources namely memory space and calculation operations. Both methods adopt a discrete time model. The goal of this work is to combine the two methods to overcome the problems associated with each one of them. The modeling of RNI by Timed Automata allows us to take advantage of the expressive power of SNNs and the accuracy of Model Checking. UPPAAL is one of the Model Checkers that uses ATs and TCTL logic. This tool will be used for such a modeling.

Keywords: Artificial Neural Networks, Spiking Neural Network, Timed automata, modeling, simulation, Model Checking, Model Checker, UPPAAL.

الملخص

تستخدم الشبكات العصبية الاصطناعية ANN بشكل متزايد في علوم الحوسبة. مستوحاة من الدماغ البشري، تسمح الشبكات العصبية الاصطناعية بفهم بعض الظواهر الطبيعية. شبكات SNN العصبية هي الجيل الثالث من شبكات ANN لقد تم منحهم قوة تعبيرية أوسع تسمح لنا بنمذجة ومحاكاة المزيد من الظواهر. عيبها هو معدل الدقة المعتدل. مقارنة بالتقنيات الأخرى ، تعد تقنية Model Checking أسلوبًا شاملاً آليا تمامًا. تعتمد هذه التقنية على Timed Automata و TCTL المنطق الزمني. عيب هذه التقنية هو الانفجار في عدد الحالات التي تتطلب الكثير من الموارد مثل مساحة الذاكرة وعمليات الحساب. تعتمد كلتا الطريقتين نموذجًا زمنيًا منفصلًا. الهدف من هذا العمل هو الجمع بين الطريقتين للتغلب على المشاكل المرتبطة بكل واحدة منهما. تتيح لنا نمذجة SNN بواسطة Timed Automata الاستفادة من القوة التعبيرية لشبكات SNN ودقة Model Checking. UPPAAL هو أحد Model Checkers التي تستخدم ATs و المنطق الزمني TCTL. سيتم استخدام هذه الأداة في هذه النمذجة.

الكلمات الرئيسية: الشبكات العصبية الاصطناعية ، الشبكة العصبية SNN ، Timed Automata ، النمذجة ، المحاكاة ، Model Checking ، Model Checker ، UPPAAL

Sommaire

Table des figures

Introduction générale

Les réseaux de neurones (RN) artificiels s'inspirent des neurones du système nerveux. Leurs domaines d'application sont la modélisation biologique et la réalisation de machines destinées à effectuer des tâches auxquelles les ordinateurs et les outils traditionnels semblent moins bien adaptés que les êtres vivants, telles que la reconnaissance des formes, la classification et à l'identification des systèmes.

Leur origine vient de l'essai de modélisation mathématique du cerveau humain. Les premiers travaux sont ceux de W.M. Culloch et W. Pitts. Ils supposent que l'impulsion nerveuse est le résultat d'un calcul simple effectué par chaque neurone et que la pensée née grâce à l'effet collectif d'un réseau de neurone interconnecté.

Un réseau de neurones est un assemblage de constituants élémentaires interconnectés (appelés « neurones » en hommage à leur modèle biologique), qui réalisent chacun un traitement simple mais dont l'ensemble en interaction fait émerger des propriétés globales complexes.

Les réseaux de neurones impulsionnels sont considérés comme la troisième génération de réseaux de neurones artificiels (ANN). Les réseaux de neurones impulsionnels prennent en compte le fait que la sortie des neurones réels n'est pas caractérisée par une fonction continue, mais par une séquence discrète d'éléments indivisibles appelés impulsions (spikes). Ce type de réseau de neurones appelé ainsi « Spiking Neura

Network » (SNN) sont des modèles qui tiennent compte explicitement de la synchronisation des entrées.

Ce réseau ressemble à un réseaux d'automates temporisés communiquant entre eux via des canaux de communication, car un automate temporisé adopte un modèle discret de temps. Ces automates sont introduites par Allur et Dill, et ils sont utilisés dans la vérification formelle, notamment la technique de Model Checking, cette technique est une technique exhaustive et complètement automatique, ce qui nous permet de mieux comprendre le système modélisé d'un côté et d'assurer son bon fonctionnement à travers la vérification des propriétés attendues de lui.

L'objectif de ce travail est de modéliser les SNNs en utilisant les ATs et de vérifier quelques propriétés en utilisant le Model Checker UPPAAL. Pour cela, ce mémoire est composé comme suit :

Chapitre 1 : Ce chapitre est dédié aux réseaux de neurones, où nous allons introduit les neurones artificiels et ces modèles, parmi lesquels, on trouve les SNNs qui représentent la troisième génération des RN.

Chapitre 2 : La technique de Model Checking est le sujet de ce chapitre. Nous allons introduit en premier lieu les automates temporisés, ensuite nous donnerons la définition de la logique temporelle TCTL et en fin nous présenteront l'algorithme du Model Checking TCTL.

Chapitre 3 : Dans ce chapitre, nous allons introduire le Model Checker UPPAAL et nous proposerons des algorithmes qui expliquent le fonctionnement des SNNs ainsi qu'un algorithme pour transformer un réseaux SNNs en un réseaux d'automates temporisés. Nous allons ensuite appliquer le Model Checker pour vérifier et valider notre travail.

chapitre 1 : Réseaux de neurones

1.1 Introduction

Les réseaux de neurones (RN) formels sont des systèmes de traitement de l'information dont la structure s'inspire de celle du système nerveux. Leurs deux grands domaines d'application sont d'une part la modélisation biologique, dont il ne sera pas question ici, et d'autre part, la réalisation de machines destinées à effectuer des tâches auxquelles les ordinateurs et les outils traditionnels semblent moins bien adaptés que les êtres vivants, telles que des tâches perceptives et motrices, ainsi, qu'à la reconnaissance de formes, la classification et à l'identification des systèmes.

1.2 Définition

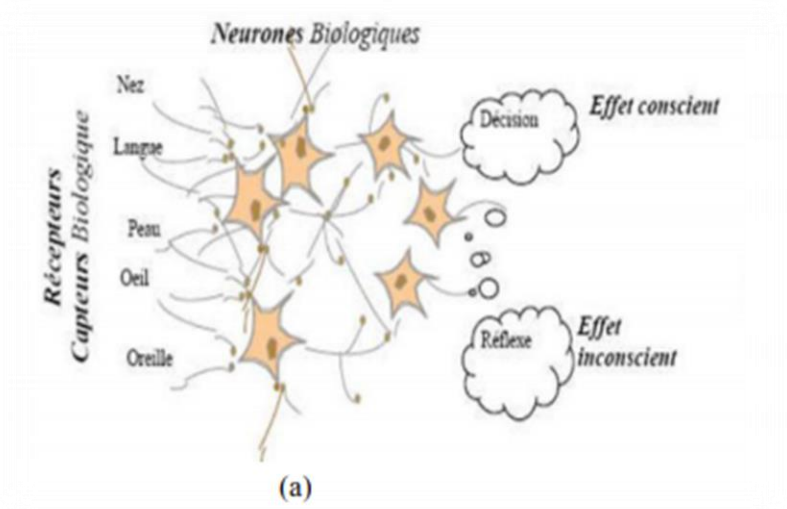


Figure 1- *structure d'un réseau de neurone biologique et le cerveau humain*

L'origine des réseaux de neurones vient de l'essai de modélisation mathématique du cerveau humain. Les premiers travaux datent de 1943 et sont l'œuvre de W.M. Culloch et W. Pitts. Ils supposent que l'impulsion nerveuse est le résultat d'un calcul simple effectué par chaque neurone et que la pensée née grâce à l'effet collectif d'un réseau de neurone interconnecté(voir la figure 1) [1].

Un réseau de neurones est un assemblage de constituants élémentaires interconnectés (appelés « neurones » en hommage à leur modèle biologique), qui réalisent chacun un traitement simple mais dont l'ensemble en interaction fait émerger des propriétés globales complexes. Chaque neurone fonctionne indépendamment des autres de telle sorte que l'ensemble forme un système massivement parallèle. L'information est stockée de manière distribuée dans le réseau sous forme de coefficients synaptiques ou de fonctions d'activation, il n'y a donc pas de zone de mémoire et de zone de calcul, l'une et l'autre sont intimement liés. [1]

Les réseaux de neurones artificiels sont des réseaux fortement connectés de processeurs élémentaires fonctionnant en parallèle. Chaque processeur élémentaire calcule une sortie unique sur la base des informations qu'il reçoit. Les neurones artificiels sont souvent utilisés sous forme de réseaux qui diffèrent selon le type de connections entre les neurones [4].

Ces types constitués d'un nombre fini de neurones qui sont arrangés sous forme de couches. Les neurones de deux couches adjacentes sont interconnectés par des poids. L'information dans le réseau se propage d'une couche à l'autre, on dit qu'ils sont de type « feed-forward ». Nous distinguons trois types de couches [4] :

Couche d'entrée : les neurones de cette couche reçoivent les valeurs d'entrée du réseau et les transmettent aux neurones cachés. Chaque neurone reçoit une valeur, il ne fait pas donc de sommation.

Couches cachées : chaque neurone de cette couche reçoit l'information de plusieurs couches précédentes, effectue la sommation pondérée par les poids, puis la transforme selon sa fonction d'activation qui est en général une fonction sigmoïde. Par la suite, il envoie cette réponse aux neurones de la couche suivante.

Couche de sortie : elle joue le même rôle que les couches cachées, la seule différence entre ces deux types de couches est que la sortie des neurones de la couche de sortie n'est liée à aucun autre neurone.

La capacité de traitement d'un réseau de neurones est stockée sous forme de poids d'interconnexions obtenus par un processus d'apprentissage. On distingue trois types d'apprentissage [2] :

- L'apprentissage dit supervisé dans lequel on présente simultanément au réseau une entrée et une sortie désirée de façon à pouvoir calculer la différence entre la sortie produite et la sortie désirée et minimiser l'erreur commise.
- L'apprentissage dit guidé ou renforcé dans lequel on indique au réseau si la sortie produite est correcte ou pas, ce qui entraîne une variation de poids de connexions.
- L'apprentissage non supervisé qui consiste à détecter de manière automatique les similitudes dans les exemples traités et à modifier les poids de connexions entre neurones pour que des exemples ayant les mêmes caractéristiques fournissent les mêmes sorties. Les connexions entre les neurones qui composent le réseau décrivent la topologie du modèle, selon laquelle, on peut classifier plusieurs modèles de réseaux de neurones.

1.3 Le modèle mathématique

Les réseaux de neurones sont fondés sur des théories mathématiques. Dans [3], on trouve un modèle mathématique, une fonction de transfert et un modèle mathématique pour un réseau de neurones :

1.3.1 Modèle d'un neurone

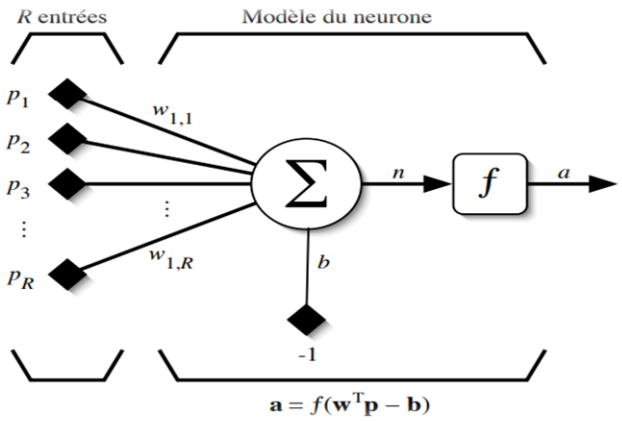


Figure 2 - *modèle d'un neurone artificiel*

Le modèle mathématique d'un neurone artificiel est illustré à la figure 2. Un neurone est essentiellement constitue d'un intégrateur qui effectue la somme pondérée de ses entrées. Le résultat *n* de cette somme est ensuite transformée par une fonction de transfert f qui produit la sortie a du neurone. En suivant les notations présentées à la section précédente, les R entrées du neurone correspondent au vecteur p= [P_1 P_2··P_R] T, alors que W = [$w_{1,1}$ $w_{1,2}$ · · · $w_{1,R}$] T représente le vecteur des poids du neurone. La sortie n de l'intégrateur est donnée par l'équation suivante :

$$n = \sum_{j=1}^{R} w_{1,j} p_j - b$$

$$= w_{1,1}p_1 + w_{1,2}p_2 + \cdots + w_{1,R}p_R - b, \qquad (1.1)$$

Que l'on peut aussi écrire sous forme matricielle :

$$n = w^T p - b. \qquad (1.2)$$

Cette sortie correspond à une somme pondérée des poids et des entrées moins ce qu'on nomme le biais *b* du neurone. Le résultat *n* de la somme pondérée s'appelle le niveau d'activation du neurone. Le biais *b* s'appelle aussi le seuil d'activation du neurone. Lorsque le niveau d'activation atteint ou dépasse le seuil *b*, alors l'argument de f devient positif (ou nul). Sinon, il est négatif.

Un autre facteur important dans le modèle que nous nous sommes donné concerne son caractère discret. En effet, pour pouvoir simuler un réseau de neurones, nous allons rendre le temps discret dans nos équations. Autrement dit, nous allons supposer que tous les neurones sont synchrones, c'est-à-dire qu'à chaque temps *t*, ils vont simultanément calculer leur somme pondérée

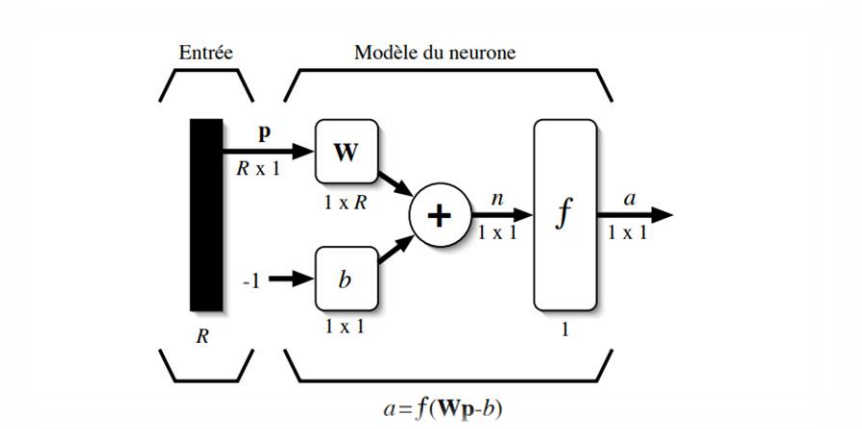


Figure 3 - *représentation matricielle du modèle d'un neurone artificiel*

Et produire une sortie $a(t) = f(n(t))$. Dans les réseaux biologiques, tous les neurones sont en fait asynchrones. Revenons donc à notre modèle tel que formulé par l'équation (1.2) et ajoutons la fonction d'activation *f* pour obtenir la sortie du neurone :

$$a = f(n) = f(w^T p - b). \qquad (1.3)$$

En remplaçant w_T par une matrice W = w_T d'une seule ligne, on obtient une forme générale que nous adopterons tout au long de ce mémoire :

$$a = f(Wp - b). \qquad (1.4)$$

On y représente les *R* entrées comme un rectangle noir (le nombre d'entrées est indiqué sous le rectangle). De ce rectangle sort le vecteur *p* dont la dimension matricielle est *R* × *1*. Ce vecteur est multiplié par une matrice *W* qui contient les poids (synaptiques) du neurone. Dans le cas d'un neurone simple, cette matrice possède la dimension *1*×*R*. Le résultat de la multiplication correspond au niveau d'activation qui est ensuite comparé au seuil *b* (un scalaire) par soustraction. Finalement, la sortie du neurone est calculée par la fonction d'activation f. La sortie d'un neurone est toujours un scalaire.

1.3.2 La fonction de transfert

Nom de la fonction	Relation d'entrée/sortie
seuil	$a = 0$ si $n < 0$ $a = 1$ si $n \geq 0$
seuil symétrique	$a = -1$ si $n < 0$ $a = 1$ si $n \geq 0$
linéaire	$a = n$
linéaire saturée	$a = 0$ si $n < 0$ $a = n$ si $0 \leq n \leq 1$ $a = 1$ si $n > 1$
linéaire saturée symétrique	$a = -1$ si $n < -1$ $a = n$ si $-1 \leq n \leq 1$ $a = 1$ si $n > 1$
linéaire positive	$a = 0$ si $n < 0$ $a = n$ si $n \geq 0$
sigmoïde	$a = \frac{1}{1+\exp^{-n}}$
tangente hyperbolique	$a = \frac{e^n - e^{-n}}{e^n + e^{-n}}$
compétitive	$a = 1$ si n maximum $a = 0$ autrement

Figure 4 - *les fonctions de transfert a=f(n)*

La fonction de transfert est en général une fonction non linéaire monotone croissante ; par ailleurs les fonctions de transfert ont des propriétés diverses : elles peuvent être déterministes, continues, discontinues ou aléatoires. [20]

Différentes fonctions de transfert pouvant être utilisées comme fonction d'activation du neurone (voir la figure 4).

Les trois les plus utilisées sont les fonctions : « seuil » (en anglais « hard limit »), «linéaire » et « sigmoïde ».

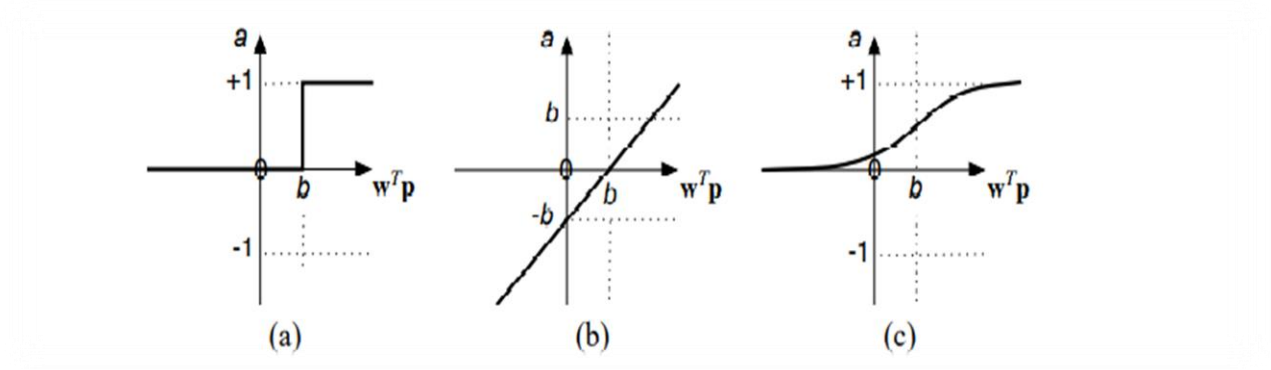


Figure 5 - *fonction de transfert : (a) du neurone seuil ; (b) du neurone linéaire, et (c) du neurone sigmoïde*

1.3.3 Modèle d'un réseau de neurone

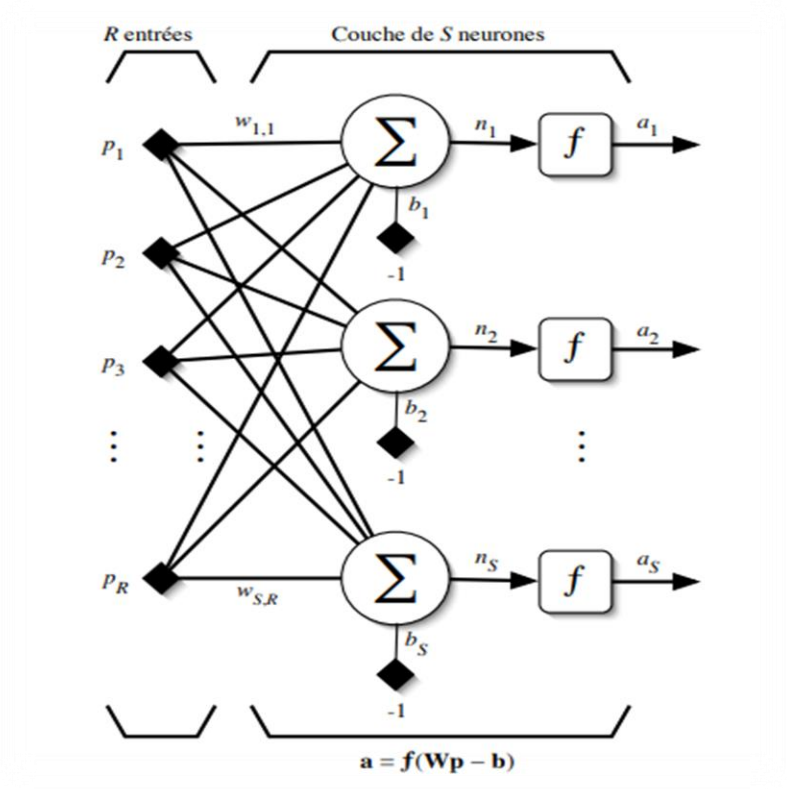


Figure 6 - *couche de S neurones*

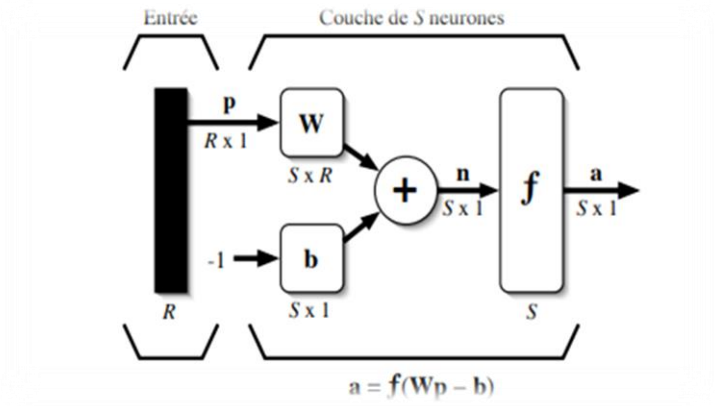


Figure 7 - *représentation matricielle d'une couche de S neurones*

Un réseau de neurones est un maillage de plusieurs neurones, généralement organisé en couches. Pour construire une couche de S neurones, il s'agit simplement de les assembler comme à la figure 6. Les *S* neurones d'une même couche sont tous

branchés aux *R* entrées. On dit alors que la couche est totalement connectée. Un poids *w (i, j)* est associe à chacune des connexions. Nous noterons toujours le premier indice par i et le deuxième par *j* (jamais l'inverse). Le premier indice (rangée) désigne toujours le numéro de neurone sur la couche, alors que le deuxième indice (colonne) spécifie le numéro de l'entrée. Ainsi, *w (i, j)* désigne le poids de la connexion qui relie le neurone *I* à son entrée *j*. L'ensemble des poids d'une couche forme donc une matrice *W* de dimension *S* × *R* : [3]

$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,R} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,R} \\ \vdots & \vdots & \ddots & \vdots \\ w_{S,1} & w_{S,2} & \cdots & w_{S,R} \end{bmatrix} \qquad (1.5)$$

Notez bien que *S ! = R*, dans le cas général (les nombres de neurones et d'entrées sont indépendants). Si l'on considère que les *S* neurones forment un vecteur de neurones, alors on peut créer les vecteurs *b* = [b_1 b_2 · · · b_s] T, n = [n_1 n_2 · · · n_s] T et a = [a_1 a_2 · · · a_s] *T*.

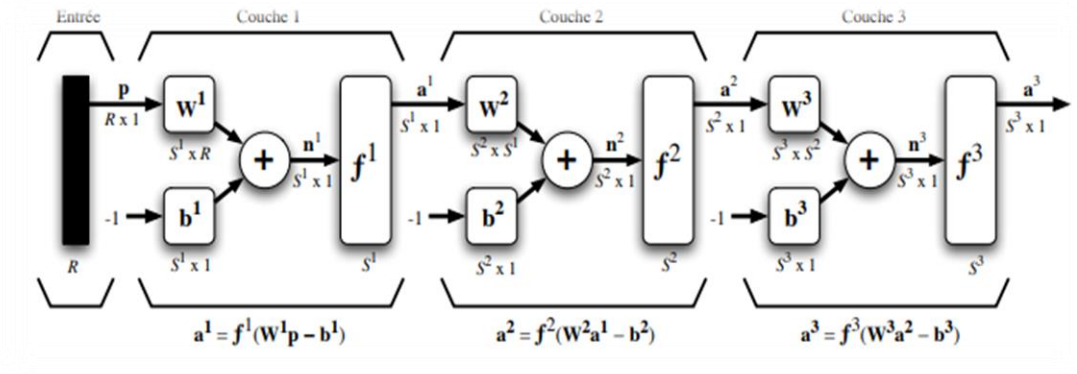


Figure 8 - *représentation matricielle d'un réseau de trois couches*

1.4 Les types

Il existe plusieurs types de réseaux de neurones dans la littérature, nous présentons dessous les plus connus :

1.4.1 Feed-forward

Les RN feed-forward étaient le premier type et sans doute le plus simple de réseaux neurones artificiels conçu. [13] il y a deux architecture de feed-forward qu'il sont :

1.4.1.1 Single layer feed-forward :

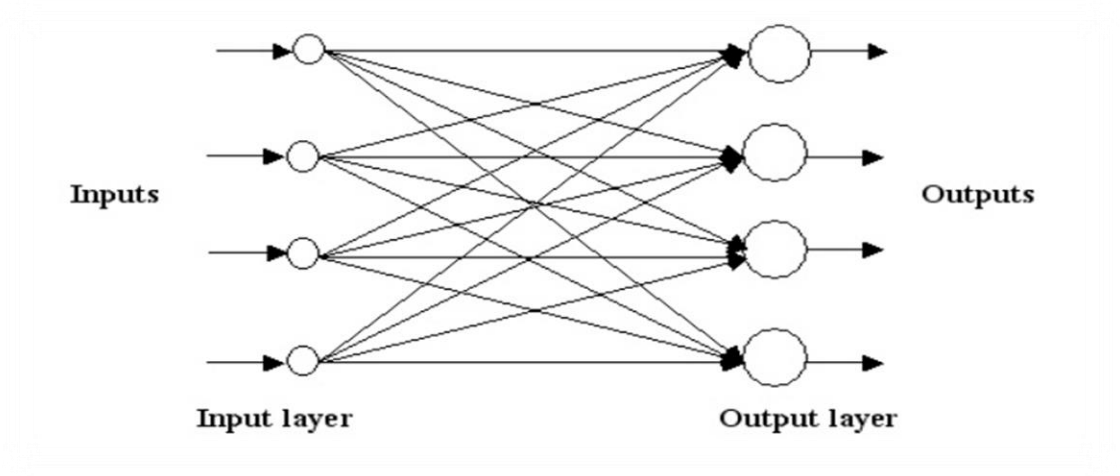


Figure 9 - *réseau de neurones « feed-forward » à une seule couche*

Dans la figure 9, un réseau neuronal feed-forward d'une seul couche (entièrement connecté) est montrée. Y compris la couche d'entrée, il y a deux couches dans cette structure. Cependant, la couche d'entrée ne compte pas car aucun calcul n'est effectué dans cette couche. Les signaux d'entrée sont transmis à la couche de sortie via les poids et les neurones de la couche de sortie calcule les signaux de sortie. [12]

1.4.1.2 Multi-layer feed-forward :

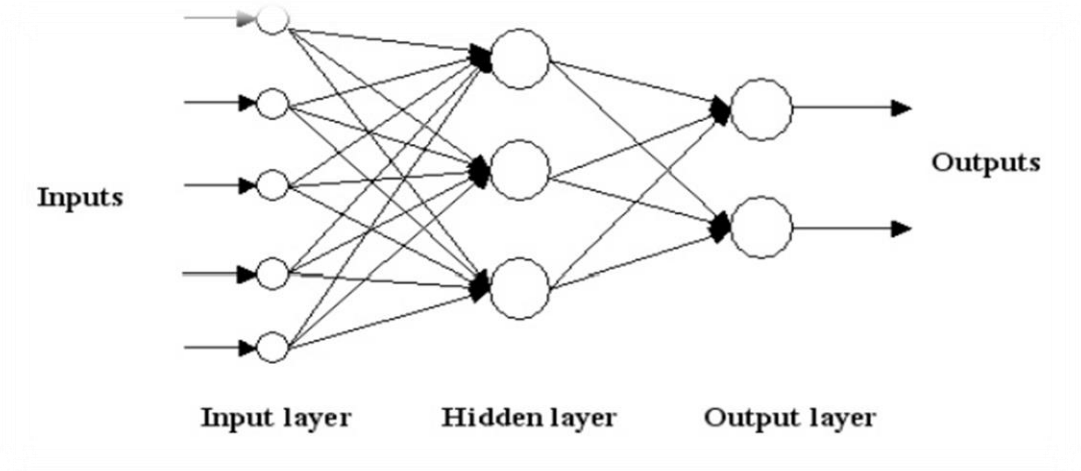


Figure 10 - *réseau de neurones « feed-forward » multicouche*

Dans la figure 10, un réseau de neurones feed-forward multicouche avec un " couche caché " est illustré. Contrairement à un réseau monocouche, il y a (au moins) une couche de « neurones cachés » entre les couches d'entrée et de sortie. La fonction des neurones cachés est d'intervenir entre l'entrée externe et la sortie du réseau d'une manière utile. L'existence d'une ou plusieurs couches cachées permet au réseau d'extraire des statistiques d'ordre supérieur. Pour l'exemple donné dans la figure *9*. Il n'y a qu'une seule couche cachée et le réseau est appelé réseau *5-3-2* car il y a *5* neurones d'entrée, *3* neurones cachés et *2* neurones de sortie. [12]

Dans les figures 9 et 10, les réseaux sont « entièrement connectés » car chaque neurone de chaque couche est connecté à tous les autres neurones de l'avant suivant couche. Si certaines des connexions synaptiques manquaient, le réseau serait appelé « partiellement connecté ». [12]

1.4.2 Réseau de neurone Récurrent

Le Réseau Neuronal Récurrent (RNR) est un réseau avec des boucles, qui permet à l'information de persister dans le réseau. RNR dispose d'une connexion feed-back au réseau lui-même, ce qui permet aux activations de revenir en boucle, d'apprendre des séquences et des informations à persister. RNR sont extrêmement puissants dans la modélisation des données séquentielles, de la parole ou du texte et appliqués sur des données non séquentielles pour former d'une manière non séquentielle. [14]

Au fil des ans, de nombreuses variétés de RNR ont été recherchées et développées [18] :

Bidirectional RNR : La sortie de ce type de RNR dépend non seulement des résultats passés mais également des résultats futurs.

Deep RNR: Dans ce type de RNR, il y a plusieurs couches présentes par étape, permettant un plus grand taux d'apprentissage et plus de précision.

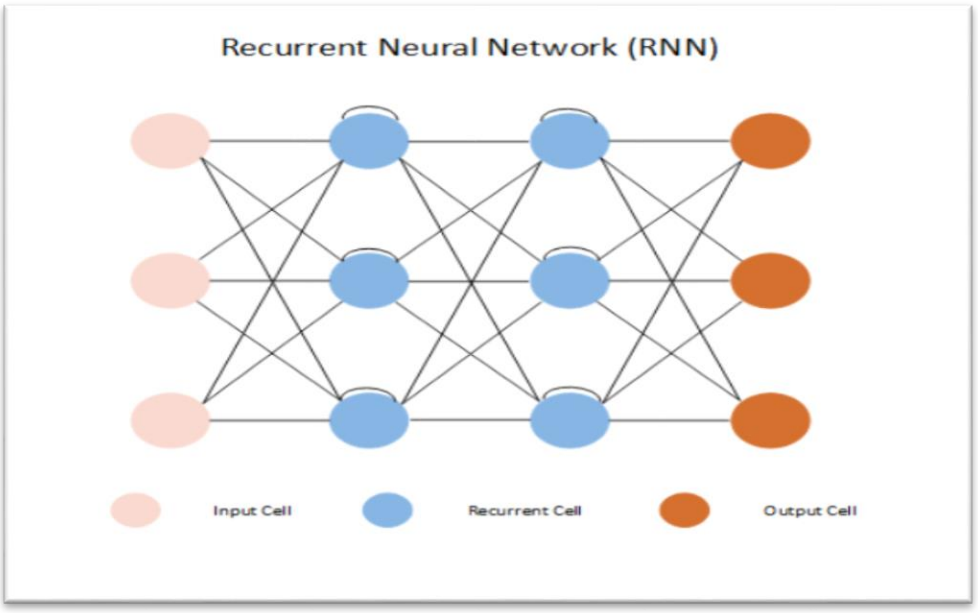


Figure 11 - *l'architecture du réseau de neurone récurent*

RNR peut être utilisé pour l'image, le sous-titrage vidéo, la prédiction de mots, la traduction de mots, le traitement d'images, la reconnaissance vocale, le traitement de la parole, le traitement du langage naturel, les applications de traitement de la musique, etc. [14].

1.4.3 Réseau de neurone à convolution (RNC)

Les réseaux de neurones à convolution sont analogues aux RNA traditionnels dans la mesure où ils sont constitués de neurones qui s'auto optimisent grâce à l'apprentissage. Chaque neurone recevra toujours une entrée et effectuera une opération (comme un produit scalaire suivi d'une fonction non linéaire), la base d'innombrables ANN. Des vecteurs d'image brute d'entrée à la sortie finale du score de classe, l'ensemble du réseau exprimera toujours une seule fonction de score perceptif (le poids). La dernière couche contiendra les fonctions de perte associées aux classes, et tous les trucs et astuces réguliers développés pour les ANN traditionnels s'appliquent toujours [16].

L'architecture d'un RNC est modélisée pour tirer avantage de la structure 2D d'une image d'entrée (ou de diverses entrées 2D similaires à un signal de parole) qui est souvent réalisée avec des connexions voisines et des poids liés suivis d'un regroupement qui provoque des caractéristiques invariantes de traduction. Un RNC se compose d'une gamme de couches convolutives et de sous-échantillonnage éventuellement accompagnées à travers des couches entièrement connectées, c'est-à-dire qu'une ou plusieurs couches entièrement connectées sont présentes après de nombreuses convolutions et couches de mise en commun. L'entrée et la sortie de chaque étape sont des ensembles de tableaux appelés cartes de caractéristiques. Dans le cas d'une image colorée, chaque carte de caractéristiques serait un tableau 2D contenant un canal de couleur de l'image d'entrée, un 3D pour une vidéo et un 1D pour une entrée audio. L'étage de sortie représente les caractéristiques extraites de tous les emplacements des données [17].

Les RNC sont utiles dans de nombreuses applications, en particulier dans ceux liées aux images. Les applications de RNC incluent la classification d'image, la segmentation sémantique d'image, la détection des objets dans des images, etc. [15].

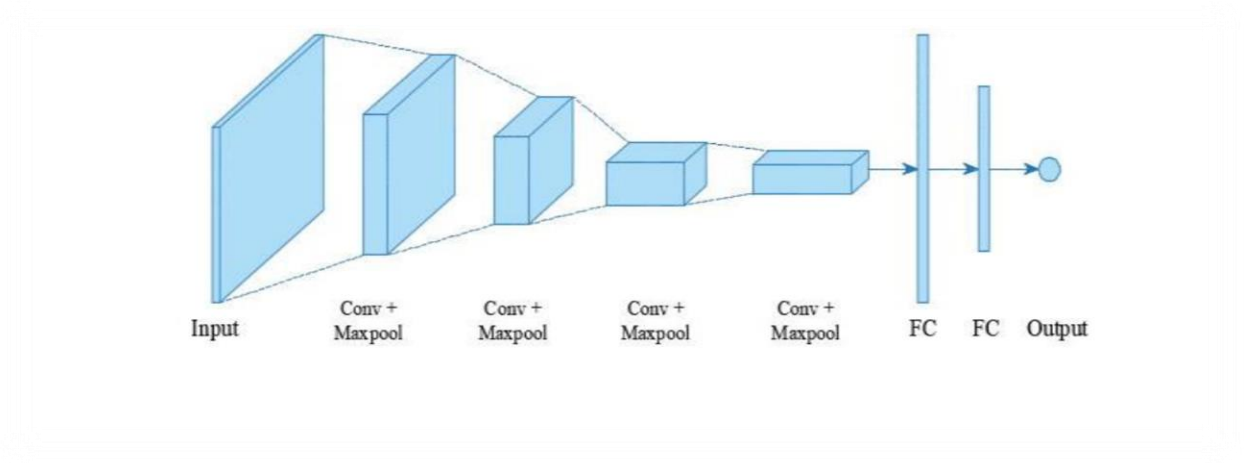


Figure 12 - une architecture de base d'un réseau de neurones à convolution

1.5 Réseaux de neurones impulsionnels

1.5.1 Définition

Au cours de ces dernières années, l'objectif de la recherche sur les réseaux de neurones s'est déplacé vers des modèles plus plausibles sur le plan biologique, tels que les réseaux de neurones impulsionnels [6].

Les réseaux de neurones impulsionnels sont considérés comme la troisième génération de réseaux de neurones artificiels (ANN) [5].

Les réseaux de neurones impulsionnels à l'encontre des réseaux de neurones classiques, prennent en compte le fait que la sortie des neurones réels n'est pas caractérisée par une fonction continue, mais par une séquence discrète d'éléments indivisibles appelés impulsion. Ce type de réseau de neurones appelé ainsi « Spiking Neural Network » (SNN) sont des modèles qui tiennent compte explicitement de la synchronisation des entrées. L'entrée et la sortie du réseau sont habituellement

représentées par des séries de temps de décharges appelées les trains d'impulsions. [21]

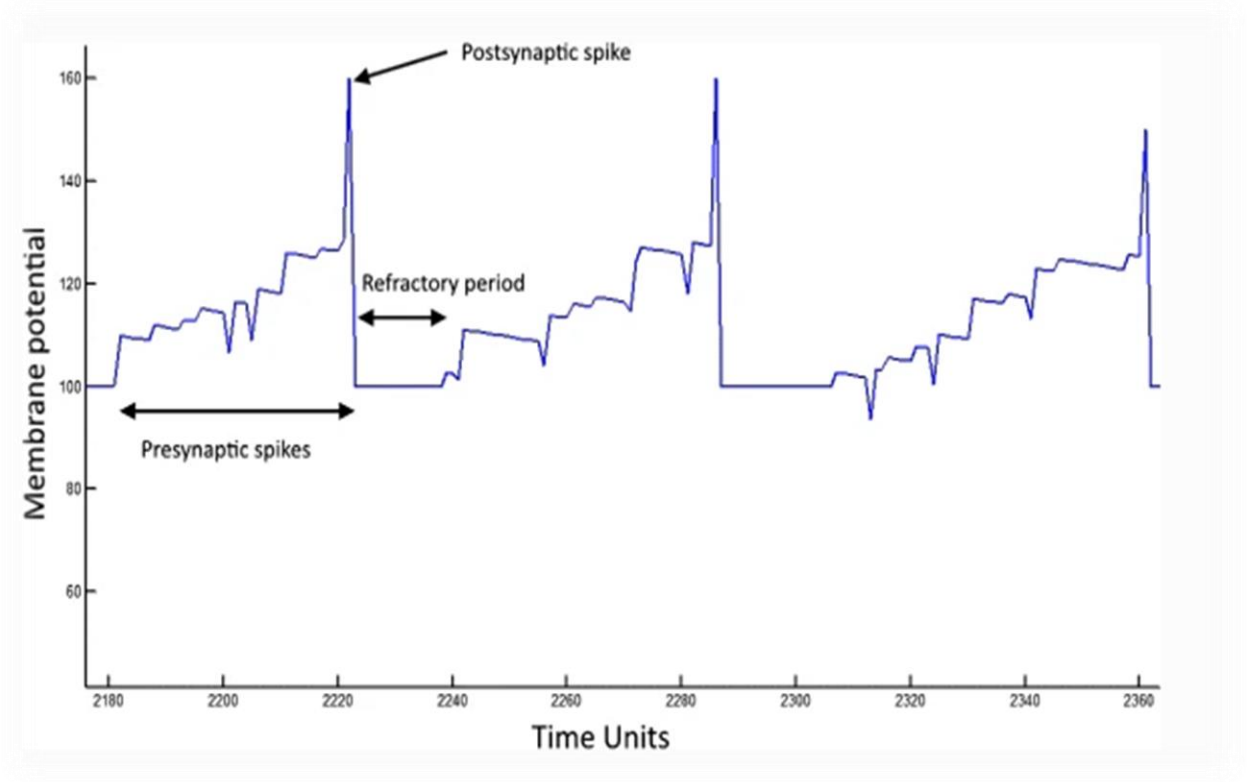


Figure 13 - *Dynamique potentielle membranaire d'un neurone unique avec modèle membranaire simplifié*

Ces SNN modélisent le temps précis des impulse tirés par un neurone, contrairement aux réseaux de neurones conventionnels, qui ne modélisent que le taux de tir moyen des neurones. En raison de ce codage temporel, les SNN sont considérés comme particulièrement utiles pour le traitement de modèles temporels, tels que la reconnaissance vocale. Ces schémas temporels consistent à corréler des événements temporels, comme une série de phonèmes dans le cas de la parole. Le fait que ces événements temporels peuvent se produire plus d'une fois, ce qui rend les modèles

temporels réellement différents des modèles statiques. Donc, pour traiter de tels modèles avec un SNN, ils doivent être codés en plusieurs impulse par neurone d'entrée et les neurones doivent pouvoir piquer plus d'une fois. Bien qu'il y ait eu diverses études introduisant des règles d'apprentissage pour les réseaux de neurones impulsionnels qui n'impulser qu'une seule fois. Aucun algorithme supervisé pratique n'a été développé qui puisse faire face aux neurones qui déclenchent plusieurs impulse. Dans une méthode d'apprentissage est conçue pour un impulse par neurone, qui se révèle être extensible pour faire face à des neurones d'entrée augmentant plusieurs fois. Cependant, en utilisant cette extension, la classification est plus influencée par les premiers impulse que par les impulse ultérieurs, ce qui rend la méthode moins pratique. Un principe souvent utilisé pour classer plusieurs impulse par neurone d'entrée consiste à regrouper d'abord ces impulse multiples en impulse simples en utilisant un SNN en combinaison avec une règle d'apprentissage non supervisée, puis en alimentant ces impulse uniques en un deuxième SNN qui utilise une règle d'apprentissage supervisé. Cette méthode de classification est loin d'être optimale car la méthode de clustering ne fait pas de distinction entre les informations saillantes et invariantes dans les données pour une tâche de classification particulière. [6]

1.5.2 Modèle mathématique

Les SNN prennent plusieurs formes selon le modèle adéquat. D'après [11] Nous pouvons citer quatre (4) modèles qui sont :

1.5.2.1 Modèle IF:

L'un des modèles les plus simples est appelé integrate-and-fire (IF) Ce modèle intègre des pointes d'entrée à son potentiel membranaire v. Si v dépasse un seuil défini v_{th}, un pic de sortie est déclenché et v est remis à son potentiel de repos v_{rest}. Après la mise à feu, le neurone entre en mode réfractaire pendant la durée de t_{ref}. Aucun pic n'est intégré pendant cette période (voir Figure 14). Le modèle est défini par la formule suivante:

$$\mathrm{c_m}\ \frac{\partial \mathrm{v}}{\partial t} = \mathrm{z(t)}\ , \mathrm{v} \leftarrow \mathrm{v_{rest}}\ \text{when}\ \mathrm{v} \geq \mathrm{v_{th}} \qquad (1.6)$$

Avec c_m la capacité de la membrane.

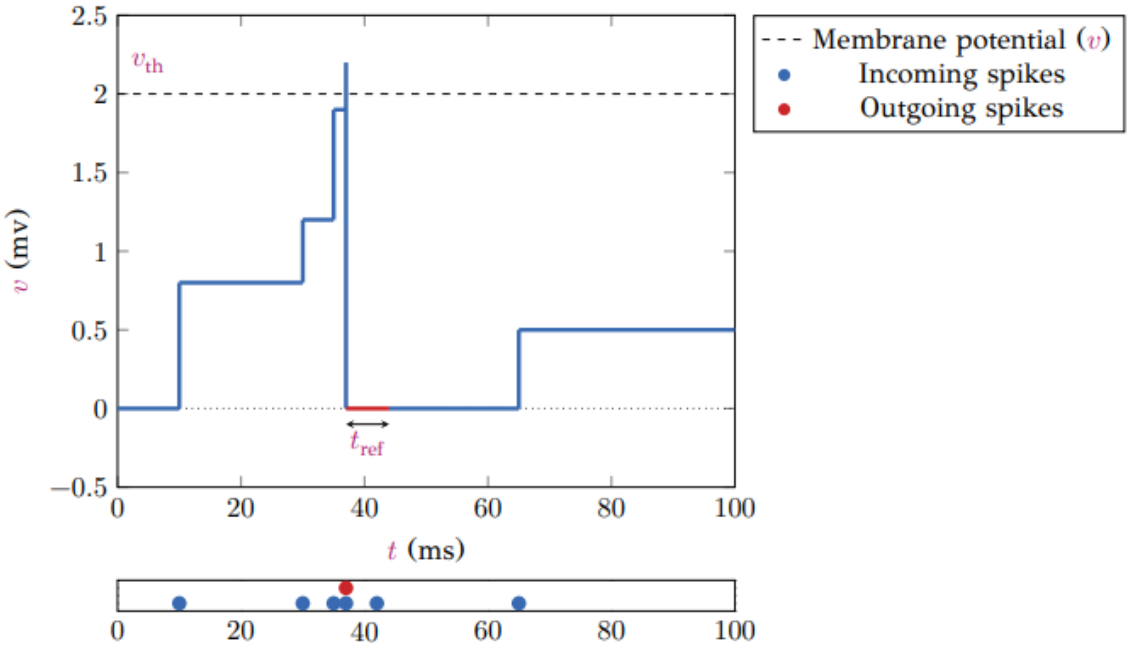


Figure 14 - *évolution du potentiel membranaire d'un neurone IF en fonction d'un train d'impulsions entrant*

1.5.2.2 Leaky integrate and fire model:

Les modèles Leaky Integr-and-Fire (LIF) sont un peu plus proches de la biologie, en ajoutant une fuite au potentiel de membrane v. Cette fuite permet aux neurones de revenir à l'état de repos en l'absence d'activité (voir figure 15). Le LIF peut être exprimé comme suit:

$$\tau_{leak} \frac{\partial v}{\partial t} = [v(t) - v_{rest}] + r_m z(t) , v \leftarrow v_{rest} \text{ when } v \geq v_{th} \qquad (1.7)$$

Avec $\tau_{leak} = r_m \ c_m$ la constante de temps qui contrôle la forme de la fuite et r_m la résistance de la membrane.

Il existe des modèles plus complexes appartenant à la famille des neurones IF, tels que les modèles de neurones d'intégration et de feu exponentiels (EIF), quadratiques d'intégration et de feu (QIF) et de neurones adaptatifs exponentiels d'intégration et de

feu (LIF) qui permettent d'obtenir des comportements plus proche des observations biologiques.

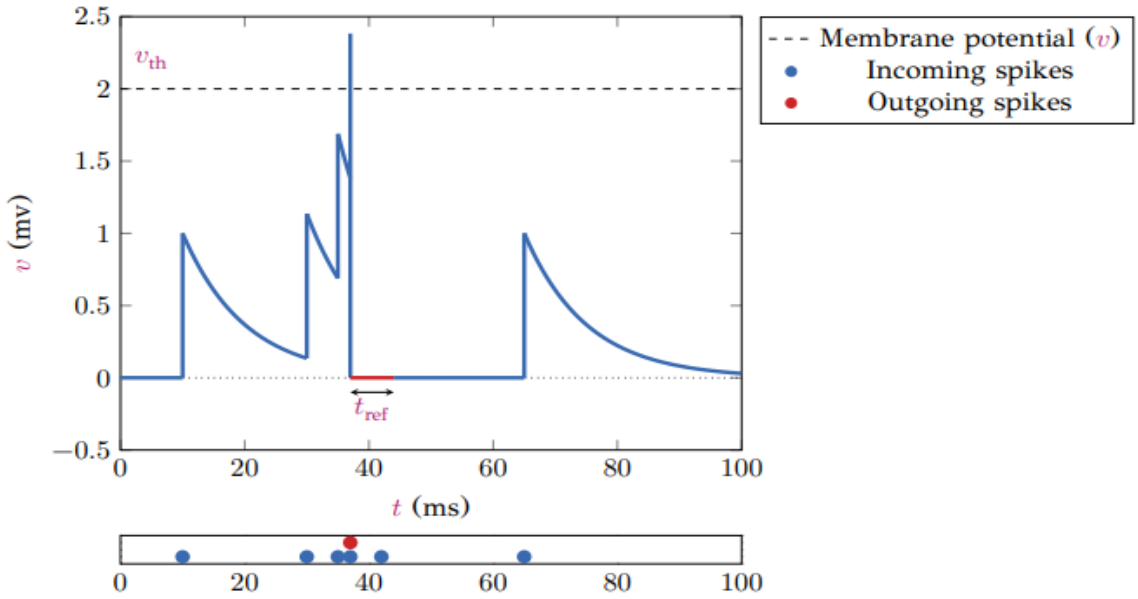


Figure 15- *évolution du potentiel membranaire d'un neurone « LIF » en fonction d'un train d'impulsions entrant*

1.5.2.3 Izhikevich's model:

Il existe des neurones à pointes plus complexes, comme ceux d'Izhikevich. Ce modèle a l'avantage d'être relativement simple, mais permet de reproduire plusieurs des modes de tir observés in vivo.

$$\frac{\partial \mathrm{v}}{\partial t} = 0.04\mathrm{v}^2 + 5\mathrm{v} + 140 - \mathrm{U} + \mathrm{z(t)}, \mathrm{v} \leftarrow \mathrm{c}, \mathrm{U} \leftarrow \mathrm{U} + \mathrm{d} \ \text{ when } \ \mathrm{v} \geq 30\ \mathrm{mV}$$

$$\frac{\partial \mathrm{U}}{\partial t} = \mathrm{a(bv - U)} \qquad (1.8)$$

Avec a, b, c, d les paramètres qui définissent le mode de déclenchement du neurone

1.5.2.4 Hodgkin-huxley model:

Le modèle de Hodgkin-Huxley est important en neurosciences, car il est très proche de la biologie. Il utilise quatre équations et des dizaines de paramètres qui reproduisent le comportement de différents canaux ioniques dans les neurones naturels. Cependant, ce modèle est l'un des plus complexes à simuler et nécessite un nombre élevé d'opérations, ce qui empêche son utilisation dans les SNN à grande échelle.

1.5.3 Domaine d'utilisation

Les neurones impulsionnels sont capables de reproduire avec exactitude le fonctionnement des neurones classiques. D'autre part du fait de leur construction plus proche des neurones biologiques, ils permettent d'étudier aussi bien les fonctionnements unitaires que les interactions entre populations dans les systèmes perceptifs tels que la vision, l'audition ou l'olfaction ou même pour l'étude de concepts fonctionnels tels que le liage, le « contrôle attentionnel », les oscillations, la pensée ou la conscience. Ainsi la compréhension et la maîtrise du fonctionnement des neurones impulsionnels représentent aujourd'hui un enjeu majeur de l'évolution des neurosciences computationnelles. [22]

1.6 Conclusion

Nous avons présenté dans ce chapitre les réseaux de neurones naturels et artificiels pour mieux comprendre les ressemblances, et mieux comprendre les modélisations et les simulations faites sur ces réseaux, afin de trouver des solutions à des problèmes complexes. Nous avons présenté en particulier les réseaux de neurones impulsionnels qui sont le sujet de notre mémoire, et qui vont être modélisés par des automates temporisés dans le troisième chapitre. Dans le chapitre suivant, nous allons présenter ces derniers ainsi que la technique de Model Checking.

chapitre 2 : Model Checking

2.1 Introduction

Dans le but de palier aux inconvénients des tests sur les logiciels, on a créé une méthode exhaustive pour vérifier formellement les différents logiciels et/ou systèmes, surtout ceux dites critiques. Cette méthode est connue sous le nom : Model Checking. Le principe de base de cette technique repose sur un modèle qui représente le système ou le logiciel et une propriété à vérifier. L'algorithme du Model Checking répond par oui si la propriété est satisfaite et par non autrement, avec la possibilité de préciser un contre-exemple. Le modèle du système doit être une machine à état/transition, nous parlons ici sur : la structure de Kripke, les automates et les RDPs…etc. Il existe quelques Model Checker (Un logiciel qui utilise la technique Model Checking) parmi lesquels on trouve UPPAAL. Ce dernier modélise les systèmes à l'aide des automates temporisés et exprime les propriétés à vérifier dans une logique temporelle qui est TCTL. Dans ce chapitre, nous allons introduire les automates temporisés, la logique TCTL et le Model Checker UPPAAL.

2.2 Les automates temporisés

Les automates Temporisés ont été introduits par Alur et Dill [8] pour modéliser explicitement le temps. Cette modélisation est effectuée par l'intermédiaire de variables horloges à valeurs réelles (temps dense). La progression de telles horloges est implicite. Un automate temporisé est constitué de deux parties principale :

- Un automate fini qui décrit les états de contrôle du système, et les transitions supposées instantanées entre les états.
- Un nombre fini d'horloges utilisées pour spécifier les contraintes de temps associées aux transitions de l'automate fini. Initialement, les horloges ont des valeurs nulles puis elles évoluent à la même vitesse d'une façon synchrone avec le temps.

Chaque état *q* est étiqueté par une expression logique *Inv(q)* bâtie sur les variables d'horloges. *Inv(q)* est appelé invariant de place et il doit être vérifié tant que l'automate reste dans l'état *q*. Il sert en particulier à assurer que l'on ne reste pas indéfiniment dans un état donné. Chaque transition est étiquetée par un triplet composé:

- d'une expression logique portant sur les valeurs des horloges, appelée *garde* ou condition de franchissement.
- d'une étiquette.
- de remises à zéro d'horloges.

Avant de présenter la syntaxe et la sémantique des automates temporisés avec invariants, on introduit les notions de contraintes et de valuations d'horloges.

2.2.1 Contraintes d'horloges

L'ensemble des contraintes sur les horloges, noté *G(X)* avec *X* un ensemble fini d'horloges, est composé des formules ψ syntaxiquement définies par :

$$\psi := \top \mid x \sim c \mid \psi \wedge \psi \mid \neg\psi$$

Où *c* est une constante entière et $\sim \in \{<, \leq\}$

Une valuation ou interprétation d'horloges, notée *vh*, est une fonction de *X* dans $R+$. La valeur *vh(x)* est une valuation de l'horloge *x*. La notation *vh* + δ pour tout $\delta \in R+$ exprime la valuation associant à chaque horloge x la valeur *vh(x)* + δ.

Soit λ un ensemble d'horloges tel que $\lambda \subseteq X$. On note $[\lambda := 0]$ *vh* une nouvelle valuation vh' où $vh'(x) = vh(x)$ si $x \in X$ et $x \notin \lambda$ et $vh'(x) = 0$ si $x \in \lambda$. Pour une horloge x et une contrainte ψ, la relation de satisfaction, vh satisfait ψ notée par vh ψ, est définie par:

- $vh \models \top$.
- $vh \models x \sim c$ si $v_h(x) \sim c$.
- $vh \models \neg\psi$ si $v_h \not\models \psi$.
- $vh \models \psi1 \wedge \psi2$ si $v_h \models \psi1$ et $v\ v_h \models \psi2$.

2.2.2 Syntaxe des automates temporisés

Formellement, un automate temporisé est représenté par un 7-uplet (Σ, Q, q_0, X, Tr, Inv, ρ) où

- Σ est un ensemble fini d'étiquettes de transitions.
- Q est un ensemble fini d'états dont l'état initial est q_0.
- X est un ensemble fini d'horloges.
- Tr est un ensemble fini de transitions, chaque transition tr $\in$ Tr étant définie par le tuple (q, l, ψ, λ, q') avec:
    - q, q' $\in$ Q respectivement la source et la cible de la transition.
    - l l'étiquette de la transition.
    - ψ la garde de la transition qui doit être satisfaite pour que la transition soit déclenchable. – λ l'ensemble des horloges réinitialisées à zéro.
- Inv est une fonction qui associe un invariant à chaque état.
- ρ est une fonction associant à chaque état un ensemble de variables propositionnelles.

On s'intéresse aux automates temporisés satisfaisant la propriété Non Zéno. Cette propriété indique que l'automate temporisé doit toujours donner la possibilité au temps de diverger, c'est à dire que l'on ne doit pas appliquer un nombre infini de transitions dans un intervalle de temps fini.

2.2.3 Sémantique des automates temporisés

Selon [8], la sémantique de l'automate temporisé At = (Σ,Q,q_0, X, Tr, Inv, ρ) est définie par l'ensemble des exécutions du système de transitions ST = (Σ, Q_s, q_{0s}, Tr_s, ρ). Chaque état de Q_s est un couple (q, vh) où q $\in$ Q et vh : X $\rightarrow$ R + est une valuation d'horloge tel vh satisfait l'invariant Inv(q). L'état initial q_{0s} est le couple (q_0, vh_0) où $\forall x \in$ X, $vh_0(x) = 0$

. On distingue deux types de transitions dans ST :

- *Transition de progression du temps*

  Pour un état (q, vh) et une durée $\delta \geq 0$, la transition de progression du temps est définie par: (q, vh) $\xrightarrow{\delta}$ (q, vh + δ), si pour tout $0 \leq \delta' \leq \delta$, vh + δ' satisfait l'invariant Inv(q).

- *Transition de changement d'états*

  Pour un état (q, vh) et une transition (q ,l, ψ, λ, q′) tel que vh satisfait ψ, on a (q, vh) $\xrightarrow{l}$ (q ′ ,vh[λ := 0]), avec vh l'interprétation d'horloges, ψ les contraintes d'horloges et λ ⊆ X un ensemble d'horloges.

Une exécution de l'automate At est une exécution $\sigma_t = (q_0,vh_0) \xrightarrow{l1} (q1,vh1) \xrightarrow{l2} ... \xrightarrow{li} (q_i ,vh_i)$... du système ST, où (q_0,vh_0) est l'état initial q_{0s} et $l_i \in \Sigma$ est soit un événement de changement d'état, soit d'écoulement du temps. On va noter l'ensemble des exécutions de l'automate At par Exec(A_t), et la séquence $l_1 l_2 l_3 ... l_i ...$ par Trace(σ_t). [8]

2.3 Logique temporelle (TCTL)

La logique CTL[1] ne permet de décrire que des propriétés qualitatives. Une façon d'introduire le temps dans cette logique est de borner la portée des opérateurs temporels. TCTL (Timed Computation Tree Logic) est une logique temporisée, extension de CTL, qui permet d'exprimer des propriétés sur des systèmes temps réels dits temporisés c.à.d. des systèmes dont le comportement est conditionné par le passage du temps. [9]

Cette logique est très souvent associée aux automates temporisés.

[1] Computation Tree Logic

- Etant donné un système décrit sous la forme d'un réseau d'automates temporisés, on souhaite pouvoir énoncer (et ensuite vérifier) des propriétés sur ce système. Parmi ces propriétés, on trouvera des propriétés simplement temporelles, mais aussi des propriétés temps réel qui mettent en jeu l'information quantitative sur les délais séparant les actions.
- Pour énoncer formellement de telles propriétés temps réel, plusieurs approches sont possibles. Le plus simple est d'exprimer la propriété en termes d'atteignabilité (ou de non-atteignabilité) de certains ensembles de configurations de l'automate.
- Pour des propriétés plus compliquées, on peut aussi utiliser la technique des automates observateurs : étant donné une propriété p et un réseau R, on ajoute à R un nouvel automate Ap se synchronisant sur certaines actions de R. Vérifier p se ramène alors à tester l'atteignabilité de certains états du produit R || Ap. cette méthode est souvent utilisée dans les outils UPPAAL et HYTECH.

Une autre possibilité est d'utiliser une logique temporelle temporisée, c'est à dire l'extension ou l'adaptation d'une logique temporelle par des primitives permettant d'exprimer des conditions sur les durées et les dates. Cette approche est sûrement plus satisfaisante dans la mesure où elle fournit un langage logique pour spécifier les propriétés [7]

2.3.1 Syntaxe de TCTL

Soit AP un ensemble de propositions atomiques; les formules de TCTL sont définies par les règles suivantes [9] :

R1 : chaque proposition atomique P est une formule de TCTL.

R2 : si f et g sont des formules de TCTL alors (f ∨ g) et (¬f) sont des formules de TCTL.

R3 : si f est une formule de chemin alors E f U ∼ c g et A f U ∼ c g sont des formules de TCTL, avec ∼ ∈ {<, <=, =,>,>=} et c ∈ N. c peut être un intervalle de temps.

2.3.2 Sémantique TCTL

Pour le modèle M(G) du système de transition $(Q, q_0, -\rightarrow)$ de l'automate temporisé G, on définit la relation de satisfaction d'une formule TCTL en un état $q = (s, v)$, $q \in Q$ par induction:

- – $q \models p$ si et seulement si $p \in P(s)$ i.e. la proposition p est associée à s.
- $q \models x \prec c$ si et seulement si $v(x) \prec c$ i.e. la valuation de x respecte cette contrainte d'horloge.
- $q \models x-y \prec c$ si et seulement si $v(x)-v(y) \prec c$ i.e. la valuation de x moins celle de y respecte cette contrainte d'horloge.
- $q \models \neg\varphi$ si et seulement si $\neg (s \models \varphi)$ i.e. s'il est faut que le sommet s puisse vérifier la formule φ.
- – $q \models \varphi_1 \vee \varphi_2$ si et seulement si $(s \models \varphi_1) \vee (s \models \varphi_2)$.
- $q \models \exists\varphi_1 \cup_{\prec c} \varphi_2$ si et seulement si il existe au moins une séquence $(q_0,q_1,...)$ avec $q = q_0$ tel que quel que soit (i, j), $0 \leq i \leq j$, $q_i \models \varphi_1$ et $q_j \models \varphi_2$ avec un temps entre q_0 et q_j satisfaisant la relation $\prec c$. Autrement dit, s'il existe une séquence où φ_1 est toujours vraie jusqu'à un état, situé dans un temps respectant la contrainte $\prec c$ où φ_2 est vraie.
- $q \models \forall\varphi_1 \cup_{\prec c} \varphi_2$ si et seulement si pour toutes les séquences $(q_0,q_1,...)$ avec $q = q_0$ tel que quel que soit (i, j), $0 \leq i \leq j$, $q_i \models \varphi_1$ et $q_j \models \varphi_2$ avec un temps entre q_0 et q_j satisfaisant la relation $\prec c$. Autrement dit, si pour toutes les séquences φ_1 est toujours vraie jusqu'à un état, situé dans un temps respectant la contrainte $\prec c$ où φ_2 est vraie.

Remarque : Ici, l'état q d'un automate temporisé est un couple (s, v) alors que dans la logique CTL l'état d'un automate se réduisait à un sommet de son graphe.

Remarque : L'opérateur (suivant) n'existe pas en TCTL. En effet, cette logique travaillant sur du temps borné et réel, il peut y avoir une infinité d'état entre deux états d'une séquence. De ce fait la notion de suivant n'a plus de sens. [7]

2.4 Model Checking

L'idée de base de ce que l'on appelle la vérification des modèles est d'utiliser des algorithmes, exécutés par des outils informatiques, pour vérifier l'exactitude des systèmes. L'utilisateur entre une description d'un modèle du système (le comportement possible) et une description de la spécification des exigences (le comportement souhaitable) et laisse la vérification à la machine. Si une erreur est reconnue, l'outil fournit un contre-exemple montrant dans quelles circonstances l'erreur peut être générée. Le contre-exemple consiste en un scénario dans lequel le modèle se comporte de manière indésirable, ainsi le contre-exemple fournit la preuve que le modèle est défectueux et doit être révisé. Cela permet à l'utilisateur de localiser l'erreur et de réparer la spécification du modèle avant de continuer. Si aucune erreur n'est trouvée, l'utilisateur peut affiner sa description de modèle et peut redémarrer le processus de vérification. [26]

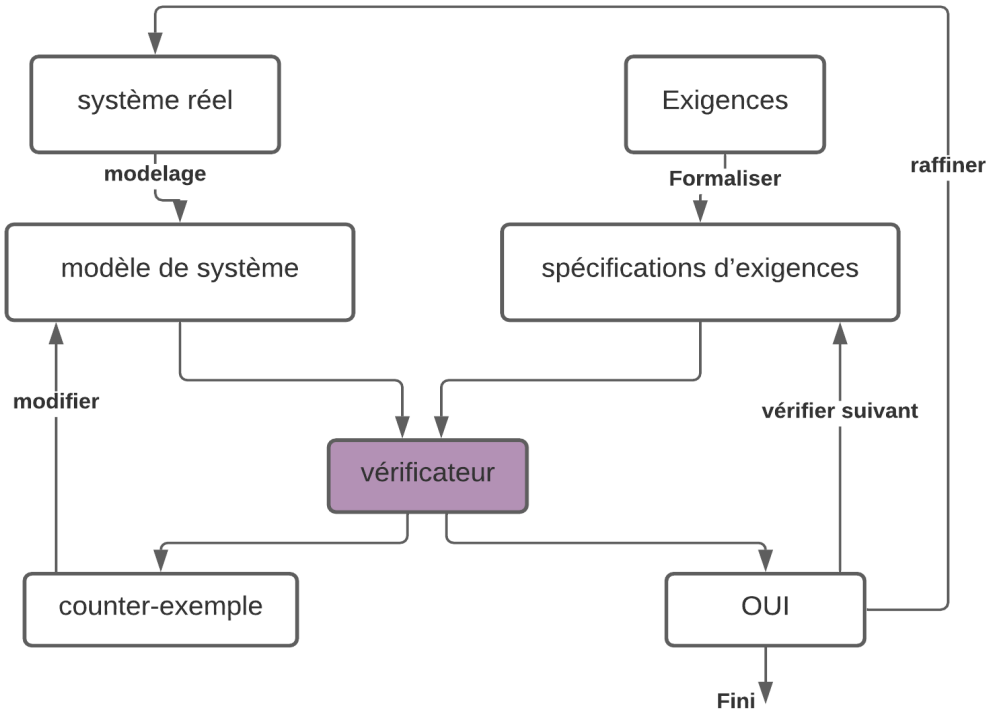


Figure 16 -*Méthode de vérification du Checking Model*

2.4.1 Model Checking TCTL

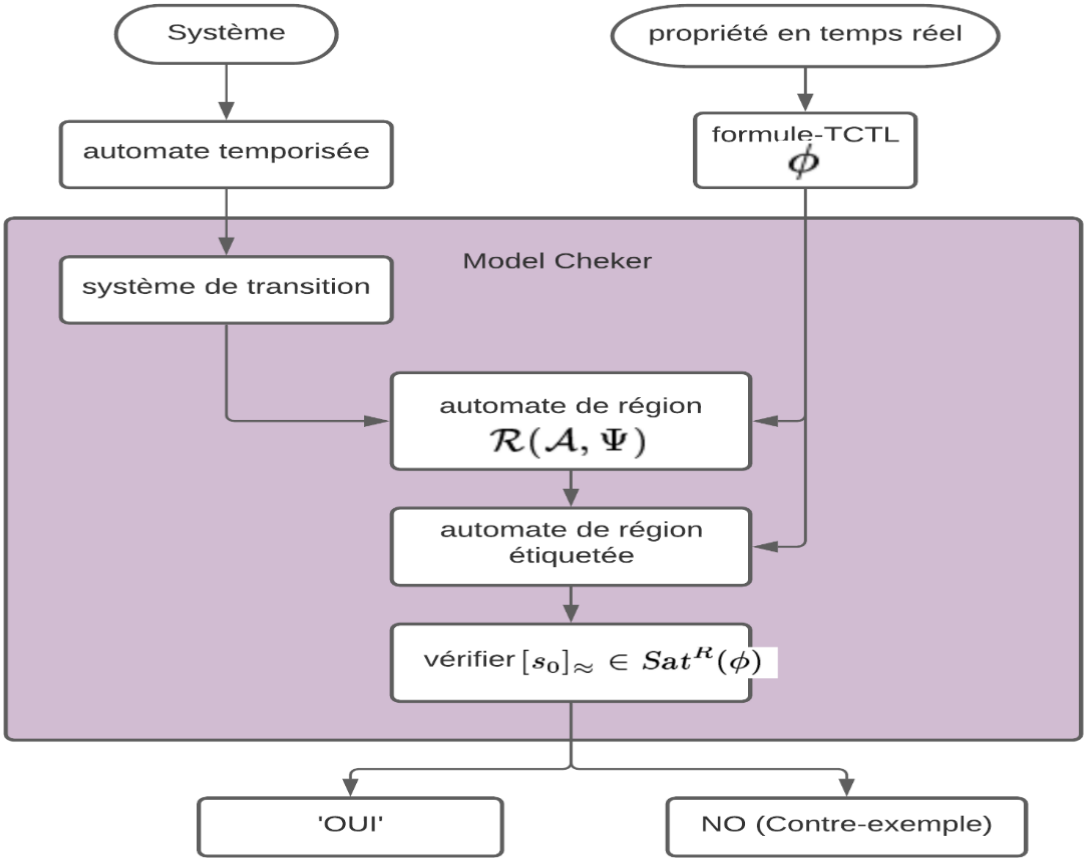


Figure 17 - *Vue d'ensemble de model Checking TCTL sur un automate temporisé*

La création de l'automate de région est détaillée dans [26].

2.4.2 L'algorithme de model Checking TCTL

Dans [26] l'auteur a proposé l'algorithme suivant :

function $Sat^R(\phi : Formula)$: **set of** $Region$;
(* precondition: true *)
begin
if $\phi = \text{true} \longrightarrow$ **return** $S/_{\approx}$
[] $\phi = \text{false} \longrightarrow$ **return** $\varnothing$
[] $\phi \in AP \longrightarrow$ **return** $\{ [s,w]_{\approx} \mid \phi \in Label(s) \}$
[] $\phi = \alpha \longrightarrow$ **return** $\{ [s,w]_{\approx} \mid s = (l,v) \ \wedge \ v \cup w \models \alpha \}$
[] $\phi = \neg \phi_1 \longrightarrow$ **return** $S/_{\approx} - Sat^R(\phi_1)$
[] $\phi = \phi_1 \ \vee \ \phi_2 \longrightarrow$ **return** $(Sat^R(\phi_1) \cup Sat^R(\phi_2))$
[] $\phi = z.\phi_1 \longrightarrow$ **return** $\{ [s,w]_{\approx} \mid (s,[z]w) \in Sat^R(\phi_1) \}$
[] $\phi = \mathsf{E}[\phi_1 \,\mathsf{U}\, \phi_2] \longrightarrow$ **return** $Sat^R_{EU}(\phi_1,\phi_2)$
[] $\phi = \mathsf{A}[\phi_1 \,\mathsf{U}\, \phi_2] \longrightarrow$ **return** $Sat^R_{AU}(\phi_1,\phi_2)$
fi
(* postcondition: $Sat^R(\phi) = \{ [s,w]_{\approx} \mid \mathcal{M},(s,w) \models \phi \}$ *)
end

2.5 Conclusion

Nous avons présenté le modèle des automates temporisés à savoir leur syntaxe et leur sémantique pour donner une idée précise sur le pouvoir expressif de ces automates. Nous avons par la suite introduit la logique TCTL, qui est une logique temporelle adéquate à l'expression des propriétés dites temporelles. Nous avons donné une idée sur la technique Model Checking ainsi que le Model Checker UPPAAL. Le chapitre suivant est consacré à trouver le lien entre le modèle des automates temporisés et le

modèle des SNNs abordés dans le premier chapitre et aussi à utiliser le Model Checker UPPAAL pour vérifier certaines propriétés liées aux SNNs.

chapitre 3 : Implémentation et Conception

3.1 UPPAAL

UPPAAL a été créé par l'université Uppsala en Suède et l'université Aalborg au Danemark. Les fondements théoriques ont été publiés en 1994 par W. Yi, P. Pettersson et M. Daniels. La première version a été distribuée en 1995. [10]

UPPAAL est un outil de modélisation et de vérification de systèmes temps-réel qui peuvent être représentés sous forme de processus non-déterministes et interactifs ayant une structure de contrôle finie et des horloges avec des valeurs réelles. Un modèle d'un système dans UPPAAL se compose d'un réseau de processus décrits par des automates temporisés étendus communiquant par des canaux ou des structures de données partagées. La conception de l'outil UPPAAL est basée sur l'architecture Client/serveur, comme le montre la Figure 16.

Le serveur est représenté par la machine UPPAAL développée en C++ et le client est représenté par l'interface GUI développée en JAVA TM. La communication entre la machine UPPAAL et son interface graphique est établie à travers des protocoles internes.

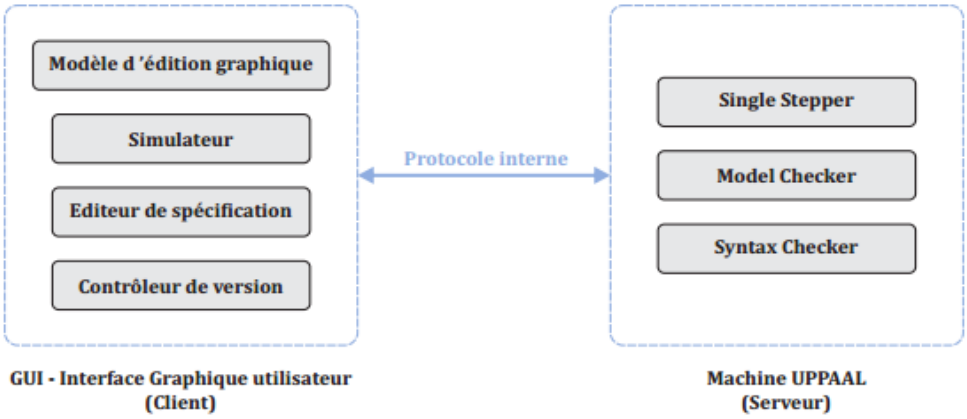


Figure 18 - Vu d'ensemble d'UPPAAL

3.1.1 Justification du choix d'UPPAAL :

Nous allons travailler par l'outil UPPAAL, vu que ce dernier utilise les automates temporisés comme un modèle pour modéliser les systèmes temps réel. Ce choix donc est motivé par la simplicité des AT (des machines états-transitions) et leurs extensions temporelles qui facilite la modélisation du passage du temps ou les autres contraintes temporelles (qui caractérisent les systèmes temps réel).

UPPAAL est doté d'un simulateur qui nous permet de capter le comportement de notre système, ainsi que les valeurs qui le caractérisent.

UPPAAL, utilise pour la vérification un model checker, ce qui facilite la vérification de notre système, le vérifier d'une manière formelle, automatique et exhaustive.

3.1.2 Description :

Uppaal se compose dans [23] de trois parties principales:

- *Un langage de description* : est un langage de commande surveillé non déterministe avec des types de donnée (par exemple tableaux, entiers délimités, etc..) il sert de langage de modélisation ou de conception pour décrire le comportement du système comme un réseau d'automates étendus avec horloges et des données variables.
- *Le simulateur* : est un outil de validation qui permet l'examen de possibles exécutions dynamiques d'un système au début de la conception (ou de modélisation). Il offre ainsi un moyen peu couteux de détection des défauts avant vérification par le Model Checker exhaustif qui couvre le comportement dynamique du système.
- *Le Model Checker* : peut vérifier l'accessibilité et l'invariabilité par l'exploration des propriétés de l'espace d'états d'un système, c'est-à-dire l'accessibilité de l'analyse en termes symboliques représentés par des contraintes.

Un autre élément important pour l'efficacité est l'application d'une technique qui réduit symboliquement la vérification des problèmes à celle de la manipulation efficace et la résolution de contrainte.
Pour faciliter la modélisation et le débogage, le vérificateur du modèle Uppaal génère automatiquement une trace de diagnostic qui explique pourquoi la propriété est (ou n'est pas) satisfaite par un modèle du système.

3.1.3 Caractéristiques de UPPAAL :

L'éditeur graphique permet la description graphique des systèmes. Une simulation graphique qui fournit des graphiques de visualisation et d'enregistrements de la dynamique des comportements possibles du système. Il est également utilisé pour visualiser les traces générées par le model Checking. [24]

3.1.4 Syntaxe :

Uppaal permet d'analyser les réseaux d'automates temporisés communiquant par des synchronisations binaires et utilisant des canaux du type émission/réception. Ainsi, sur un canal, un émetteur envoie le signal et un récepteur se synchronise avec lui par le signal complémentaire. Les automates temporisés d'UPPAAL sont donc une extension des automates temporisés originaux, auxquels on a ajouté des variables entières, des tableaux pour les horloges, …etc. [8]. La description d'un modèle se compose de trois parties :

a. Les déclarations globales et locales.
b. Les modèles d'automates.
c. La définition du système.

Déclarations : Les déclarations peuvent être locales ou globales. UPPAAL permet de déclarer des horloges (type clock), des entiers (type int), des booléens (type bool), des canaux de synchronisation (type chan), des tableaux, des structures de données (type struct), et de définir des types de données (typedef). [25]

Locations [25]: UPPAAL est un Model-Checker intégrant une interface graphique permettant la construction d'automates temporisés [8]. Un automate temporisé est considéré comme un graphe à états. Les locations représentent les états dans ce graphe. Elles se communiquent via les transitions. Un identifiant optionnel peut être associé à chaque location afin de l'identifier. Elles peuvent être étiquetées par des invariants. Un invariant peut être une expression composée d'un ensemble de contraintes sur des horloges, une différence entre des horloges ou des expressions booléennes ne faisant pas intervenir des horloges. UPPAAL distingue trois types de locations :

- *Location initiale* : Chaque automate doit avoir une location initiale représentée pas un double cercle.
- *Location urgente* : La location urgente ne permet pas de faire passer le temps quand le processus s'y trouve [25]. Le système est donc obligé de faire des transitions instantanées, d'un état du système à un autre, tant que le système est dans un état urgent [8]. Sémantiquement, une location urgente est équivalente à une location dont l'invariant est $x <= 0$ ou x est une horloge remise à zéro sur toutes les transitions entrantes de la location en question.
- *Location engagée* : La location engagée ne permet pas l'écoulement du temps quand un processus s'y trouve. Elles sont utiles pour assurer l'atomicité de la synchronisation entre trois processus où plus. Elles sont plus prioritaires que les locations urgentes.

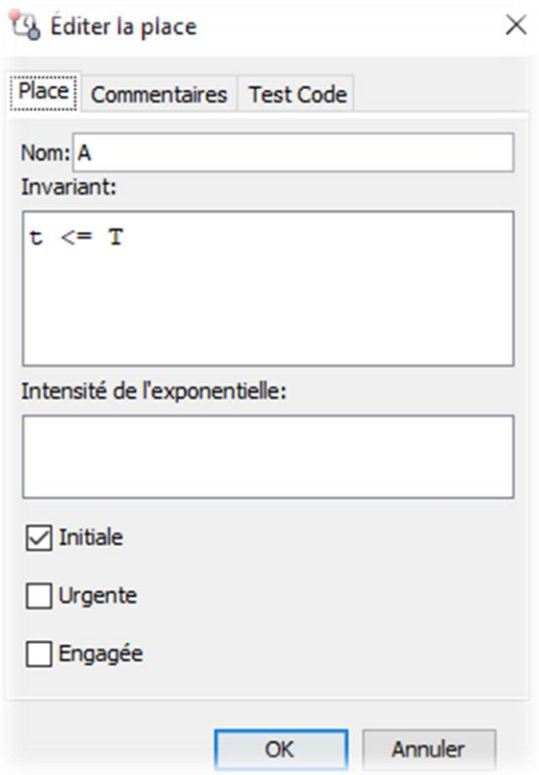


Figure 19 - *Interface des Locations pour Uppaal.*

Transitions [25]: Les locations sont connectées entre elles par les transitions (edges). Ces dernières sont étiquetées par des sélections, des gardes, des synchronisations et des mises à jour des variables et des fonctions

- *Sélection* : Cette option permet d'associer, d'une façon indéterminée, à une variable donnée une valeur dans un intervalle bien déterminé. Les gardes, la synchronisation et les mises à jour peuvent être spécifiées en fonction de la valeur sélectionnée.
- *Une garde* : qui exprime une condition sur les valeurs des variables (true par défaut). Cette condition doit généralement être compatible avec l'invariant de l'état origine de la transition. [8]. Une transition est franchie si et seulement si la garde spécifiée est satisfaite. Les gardes expriment des contraintes d'horloges ou de données nécessaires pour franchir une transition.

- *Une synchronisation* : Les différents automates dans un modèle UPPAAL peuvent se synchroniser via des canaux de synchronisation permettant l'envoi et la réception d'un message. Deux transitions dans deux processus différents peuvent se synchroniser si et seulement si les gardes des deux transitions sont satisfaites et elles admettent deux étiquettes de synchronisation X_1 ! (Envoi) et X_2 ? (Réception) où X_1 et X_2 représentent le même canal de synchronisation. Quand deux processus se synchronisent, les deux transitions sont franchies au même temps. Les mises à jour de la transition portant l'étiquette X_1! se font avant celles de la transition portant l'étiquette X_1? Il existe deux types de canaux de synchronisation :

    - Les canaux de synchronisation binaires qui permettent de synchroniser deux processus via un seul canal de synchronisation a (au moyen de deux actions X! et X?).
    - Les canaux de synchronisation de diffusion (broadcast) qui permettent de synchroniser un processus avec plusieurs autres processus. Une transition avec une étiquette de synchronisation X! avec ‘‘X’’ est un canal broadcast, peut synchroniser avec chaque transition ayant une étiquette de synchronisation X? et dont les gardes sont satisfaites.

    L’absence de synchronisation indiquant une action interne de l’automate. [8]

- *Mise à jour* : L'exécution de la mise à jour sur une transition permet de changer l'état du système. Une mise à jour est une liste d'expressions séparées par des virgules et exécutées dans un ordre séquentiel. L'opérateur d'assignement est ‘‘=‘‘. [25]

Une remise à zéro de certaines horloges et une mise à jour de certaines variables entières. [8]

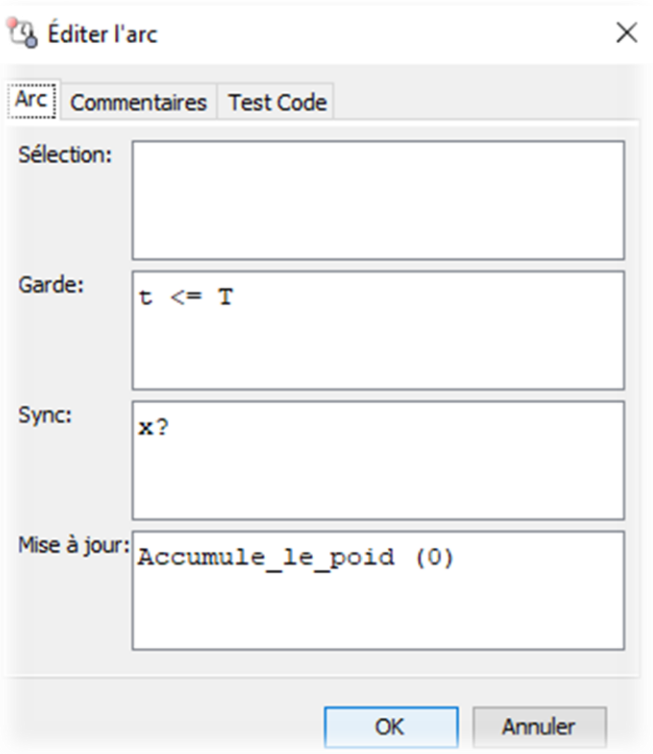


Figure 20 - *Interface des Transitions pour UPPAAL*

Nous pouvons également définir les horloges locales (propre à un automate) ou globales (communes à tous les automates). UPPAAL permet de vérifier les formules suivantes :

- ∃♦p : il existe un chemin le long duquel p est vrai un jour.
- ∃■p : il existe un chemin le long duquel p est toujours vrai.
- ∀♦p : le long de tout chemin p est vrai un jour.
- ∀■p : le long de tout chemin p est toujours vrai.
- p ⇒ q : quand p est vrai, alors q sera forcément vrai un jour.[10]

Les avantages de l'outil UPPAAL sont : son interface graphique et son module de simulation. Le module de simulation permet de détecter des erreurs sur les modèles dans la phase de modélisation. Pour plus de détails sur l'outil UPPAAL [8].

3.2 Transformation et vérification

Dans cette section, nous allons expliquer notre contribution. Pour cela, nous expliquons le principe de fonctionnement pour chaque neurone de chaque couche d'un SNNs. A travers des algorithmes qui vont nous servir dans l'étape suivante qui est la transformation elle-même. Cette transformation est décrite sous forme d'un algorithme pour chaque type de neurone.

3.2.1 Principe de fonctionnement

L'algorithme de neurone de l'input : nous avons deux types de neurones pour la couche d'entrée qui sont : Fix_rate_input et Non_Deterministic_Input

Algorithme De Neurone Fix_rate_input

```
Var :       temps: entier ;
            Retardement , Window : constant
Débute :
            E 1 :
                  Tant que ( temps < Retardement) alors
                        Incrémenter le temps par une unité de temps
                  Fin Tant que

                  réinitialiser le temps par 0
            E 2 :

                  Tant que ( temps <= D) faire
                        Incrémenter le temps par une unité de temps
                  Fin Tant que


                  SI (temps > 0 ET Le temps < Window) faire
                      Envoie Xi ;
                  FIN SI

            E 3
                  Tant que ( temps < window) alors
                        Incrémenter le temps par une unité de temps
                  Fin Tant que
                  T = 0
```

GO to E 2 ;

FIN

Le rôle principal de l'état initial est l'arbitrage, où il incrémente le temps jusqu'à un certain délai D (la garde ***t = = D***). Après l'écoulement de ce délai, il réinitialise l'horloge à 0 et il passe à l'état suivant. Dans ce dernier, il entre dans une boucle infinie d'envoi de Spikes (***X !***) et de pauses (l'invariant ***t <= Win***) en passant vers l'état S et revenant vers cet état lui-même. Cet envoi est cadré par une fenêtre Win d'où la (***garde t > 0 && t < Win***).

Algorithme De Neurone Non_Deterministic_Input

```
Var :     temps: entier ;
          Retardement ,Tmin : constant
Débute :
          E 1 :
                    Tant que ( temps < Retardement) alors
                              Incrémenter le temps par une unité de temps
                    Fin Tant que
                    Envoie Xi ;
                    réinitialiser le temps par 0

          E 2 :
          E 3 :
                    Tant que ( temps < Tmin) alors
                              Incrémenter le temps par une unité de temps
                    Fin Tant que
                    Envoie Xi ;
                    Réinitialiser le temps par 0
                    GO to E 2 ;
FIN
```

Comme dans le cas du neurone *Fixe_Rate*, l'état initial ici est dédié pour à l'arbitrage où on attend un délai D, la différence entre les deux réside dans l'envoie systématique d'un Spike (***X !***) en passant de l'état initial vers l'état S en réinitialisant l'horloge à 0. L'état S dans ce cas-là est Committed pour exprimer le non déterminisme. A partir de l'état R, on attend une temps Tmin qui sépare l'envoi du Spike (***X !***) suivant et on réinitialise le temps à 0.

L'algorithme du neurone intermédiaire :

Algorithme De Neurone Intermédiaire

```
Var :     Potentiel, temps, som_poid : entier ;
          Periode_d'accumulation , period_refractaire , threshold : constant
Débute :
          E 1 :
                    Tant que ( temps < periode_d'acumulation) faire
                         Accumule_le_poid () ;
                    Fin Tant que

                     Fin_de_l'accumulation () ;
          // E 2 :
                    SI (Potentiel < threshold) faire
                         Debut_de_l'acuumulation () ;
                         Go to E 1 ;
                    SINON
                          Envoie_Spike ;
                         Debut_De_period_réfractaire () ;
                    FIN SI
          // E 3 :
                    // Retardes jusqu'à la fin de la period_refractaire ;
                     SI (temps == Période_refractaire) faire
                        Début_de_l'accumulation () ;
                         GO to E 1 ;
                    FIN SI

FIN
```

La procédure *Accumule_le_poid( )* incrémente les poids des entrées réceptionnés dans une période donnée. Cette période est nommée Periode_d'accumulation, elle est calculée par la formule suivante :

Poids = Poids + w[i] avec w[i] le poids de l'entrée X_i.

La procédure *Fin_de_l'accumulation()* est appelée sur chaque fin de période d'accumulation pour calculer le potentiel en respectant la formule suivante :

Potentiel = poids + lambda * Potentiel (t – 1)

La procédure *Debut_de_l'acuumulation()* est appelée si le potentiel est inférieur à un seuil donné (threshold). Cette procédure réinitialise à zéro le temps et la somme des poids (***Temps= 0 et poids=0***)

On fait appel à la procédure *Debut_De_period_réfractaire()* si le neurone envoie le Spike. Après l'envoi de Spike le neurone entre dans la période réfractaire. Cette procédure initialise le potentiel et le temps à zéro (***temps =0 et potentiel =0***).

La troisième étape représente la période réfractaire. Cette période est fixée par une valeur donnée Si on atteint la fin de la période réfractaire (***temps = période réfractaire***) on fait appel à la procédure *Début_de_l'accumulation ().*

L'algorithme du neurone de sortie :

Algorithme De Neurone De Sortie

```
Var :    Laste_Spike : entier ;
Débute :
        E 1 :   Tant que (temps < window) alors
                        Incrémenter le temps par une unité de temps
                Fin Tant que
                Recevoir Yi ;
                réinitialiser le Laste_Spike par 0
        E 2 :
                  Go to E 1 ;
FIN
```

Le consommateur attend les Spike de sortie correspondantes sur le canal y (y ?) Et, dès qu'il reçoit le Spike il réinitialise le temps à 0.

3.2.2 Algorithme de transformation

Dans ce qui suit, nous allons décrire comment nous avons pu faire la transformation d'un SNN vers un réseau d'automates temporisés.

Déclaration Générale :

Déclaration :

Typedef int [-1, 1] poid_S;
Broadcast chanel Y_O ;
Broadcast chanel X_I ;

Neurones input :

Il y a deux types de neurone d'entrée : ***Fix_rate_Input*** et ***NON_Deterministique_Input***

- *Fix_rate_Input* : est une séquence infinie composée de deux parties : un préfix fini et arbitraire, et une séquence infinie, donc Fix_rate_Input = (préfix U séquence) infini avec l'état initial de la séquence infini est l'état final de préfix.

Le préfix fini est modélisé par deux états :

1. Création des états :
    - L'état B : représente l'état initial, c'est un état arbitraire pour une durée de temps avec un invariant ***t ≤ Delay***
    - L'état W : représente l'état final de préfix. Son invariant ***t < Window***
2. Création des transitions :
    - La transition B → W : sa garde est ***t = Delay*** pour passer à l'état W après le Delay. Sur cette transition, on remit l'horloge à 0.

La séquence infinie est modélisée par deux états :

1. Création des états :
   - L'état W : représente l'état initial (représente l'état final de préfix) avec l'invariant ***t < Window.***
   - L'état S : Son invariant ***t ≤ Window***.
2. Création des transitions :
   - La transition W → S : cette transition a pour rôle de faire une synchronisation ***X !***, avec une garde ***t > 0 && t < Window.***
   - La transition S → W : Sa garde ***t = Window***. L'horloge remis à 0.

Paramètre :

broadcast chanel : X ;

Déclaration :

Constant Int : Delay, Window ;

Clock :t ;

Début :

```
Création_Etat_B ( ) ;

Création_Etat_W ( ) ;
Création_Etat_S ( ) ;

Création_Transition (B, W) {
      Gard ← t = Delay ;
      t ← 0 ;
 }
Création_Transition (W, S) {
      Gard ← t > 0 && t < Window ;
      Synchronisation ← X !;
 }
Création _Transition (S, W) {

    Gard ← t = Window ;
    t ← 0 ;
}
```

FIN

Les Procédure :

```
Création_Etat_B ( ) {

      ID ←B ;
      Type ← INITIAL ;
       Invariant ← t ≤ Delay
}
 Création_Etat_W ( ) {
      ID ←W ;
      Invariant ← t < Delay
}
Création_Etat_S ( ) {
       ID ←S ;
       Invariant ← t ≤ Delay
}
```

- *Non_deterministic_input* : c'est une séquence infinie composée de sous séquences séparées par une durée de temps égale à $\mathbf{T_{min}}$ unités de temps. Le Non_deterministic_input est modélisé par trois états :

1. Création des états : on doit créer trois états : B, S, W
   - L'état B : représente l'état initial et arbitraire avec un invariant ***t ≤ Delay.***
   - L'état S : une état URGENT.
   - L'état W.
2. Création des transitions :
   - La transition B → S : avec une garde ***t = Delay*** pour passer à l'état S après le Delay initial et une synchronisation ***X !*** pour envoyer les Spikes. L'horloge est remise à 0.
   - La transition S → W : puis que l'état URGENT l'horloge n'incrémente pas dans cet état et passe directement à l'état W.
   - La transition W → S : sur cette transition, une garde $t \geq T_{min}$ pour faire une synchronisation après le minimum du temps qui sépare chaque sous séquences de celle qui suit. Il y a une synchronisation ***X !*** pour envoyer le Spike. L'horloge est remise à 0.

<table>
<tr><td colspan="2">Paramètre :

Broadcast chanel : X ;</td></tr>
<tr><td colspan="2">Déclaration :

Clock : t ;

Constant Int : Delay, T_{min} ;</td></tr>
<tr><td>Début :

Création_Etat_B ( ) ;
Création_Etat_S ( ) ;

Création_Etat_W ( ) ;

Création_Transition (B, S) {
Gard ← t = Delay ;
Synchronisation ← X !;</td><td>Les Procédure :

Création_Etat_B ( ) {
ID ←B ;
Type ← INITIAL ;
Invariant ← t ≤ Delay
}

Création_Etat_S ( ) {
ID ←S ;
Type ← URGENT</td></tr>
</table>

t ← 0 ;
}
 	
Création_Transition (S, W) {
}
 	
Création _Transition (W, S) {
Gard ← t ≥ T_{min};
Synchronisation ← X !;
 	
t ← 0 ;
}
FIN
	}
	
	Création_Etat_W ( ) {
	ID ←W ;
	}

Le Neurone Intermédiaire :

1. Création des états : on doit créer trois états : A, D, W
   - L'état A : représente l'état initial. Son invariant est ***t ≤ T*** (***T*** : la période d'accumulation).
   - L'état D : est un état committed, pour passer à la période réfractaire.
   - L'état W : Son invariant est ***t ≤ tau*** (***tau*** : la période réfractaire).
2. Création des transitions :
   - La transition A → A : Une boucle pour accumuler les poids des entrées. Sur cette transition une synchronisation ***X_i ?*** pour recevoir les entrées.
   - La transition A → D : SA garde est ***t = T*** pour passer à l'état D après la période d'accumulation. Sur cette transition on calcule le potentiel. L'horloge est remise à 0.
   - La transition D → A : Cette transition a pour rôle de commencer une nouvelle période d'accumulation. Elle est conditionnée par la condition ***potentiel < seuil***. L'horloge ***t*** doit être remise à 0 et la somme des poids (***som_poid***) est remise à zéro pour recalculer la nouvelle somme des nouvelles entrées.
   - La transition D → W : Elle est affranchie si ***potentiel ≥ seuil***. Dans ce cas-là, le neurone envoi un SPIKE. Cet envoi est fait sur un broadcast chanel ***Y !***. L'horloge est remise à 0, et le potentiel aussi.

- La transition W → A : Sa garde est ***t ≤ tau*** qui représente la période réfractaire. L'horloge, le potentiel et le ***som_poid*** sont remise à 0.

Paramètre:

Broadcast chanel : X_i, Y;

poid_S poid[i] ;

Déclaration :

Clock : t ;

Constant Int : T = « période_d'accuulation **», seuil** = « threashold », **tau**= « période_réfractaire », **lambda**= « lambda (X_i) » ;

Int : potentiel = 0, **som_poid** = 0 ;

Début

```
Création_Etat_A () ;
Création_Etat_D () ;
Création_Etat_W () ;

Création_Transition (A, A) {
        garde ← t ≤ T ;
        Recevoir_poids () ;
}

Création_Tranasition (A, D) {
        garde ← t = T ;
        potentiel ← som_poid + lambda *
potentiel ;
}

Création_Transition (D, A) {
        garde ← potentiel ≤ seuil ;
        som_poid ← 0 ;

        t ← 0 ;
}
```

Les Procédures :

```
Création_Etat_A () {
        ID ← A ;
        Type ← INITIAL ;
        Invariant ← t ≤ T ;

}

Création_Etat_D () {
        ID ← D ;
        Type ← committed ;
}
Création_Etat_W () {
        ID ← W ;
        Invariant ← t ≤ tau ;
}
Recevoir_poids () {
        Pour (1 à m) faire
        {Xi !}
}
```

Création_Transition (D, W) {
garde ← Potentiel > seuil ;
Synchronisation ← Y !;
potentiel ← 0
t ← 0 ;
}
 	
Création_Transition (W, A) {
garde ← t = tau ;
potentiel ← 0 ;
t = 0 ;
}
 	
FIN	

Le Neurone de sortie :

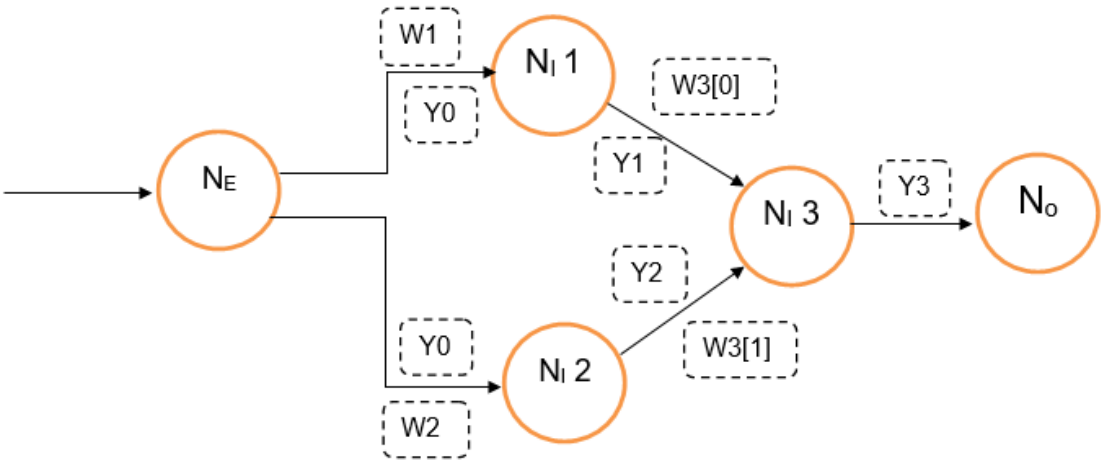
1. Création des états : on doit créer deux états W, O ;
   - L'état W : représente l'état INITIAL
   - L'état O : est un état URGENT

2. Création des Transition :
   - La transition W → O : Sur cette Transition une synchronisation ***Y ?*** pour recevoir les entée (les sorties de neurone intermédiaire)
   - La transition O → W : Sur cette transition on remise ***d_s*** (le temps depuis dernier sortie) la a 0.

Paramètre : ***broadcast chanel** : Y ;*
Déclaration : ***Clock** : d_s ;*

Début :	***Les Procédure :***
Création_Etat_W () ; *Création_Etat_O () ;* *Création_Transition (W, O) {* *Recevoi_Potentiel () ;* *}* *Création_Transition (O, W) {* *d_s ← 0 ;* *}* ***FIN***	*Création_Etat_W () {* *ID ←W ;* *Type ← INITIAL ;* *}* *Création_Etat_O () {* *ID ←O ;* *Type ← URGENT ;* *}* *Recevoi_Potentiel () {* *Y ?;* *}*

Exemple :



On a :

N_E c'est un neurone d'entrée (FIX_RATE_INPUT) ou (Non_Deterministic_Input).

N_I 1 et N_I 2 des neurones intermédiaires avec S=1 (S est le nombre des synapses).

N_I 3 c'est un neurone intermédiaire avec S=2.

N_O c'est le neurone de sortie.

Y broadcasté chanel.

Si Le neurones N_E est un FIX_RATE input donc Tmin= , Delay= avec l'algorithme Fix_rate_Input la modélisation de l'automate temporisée de ce neurone dans UPPAAL (voir la figure 21).

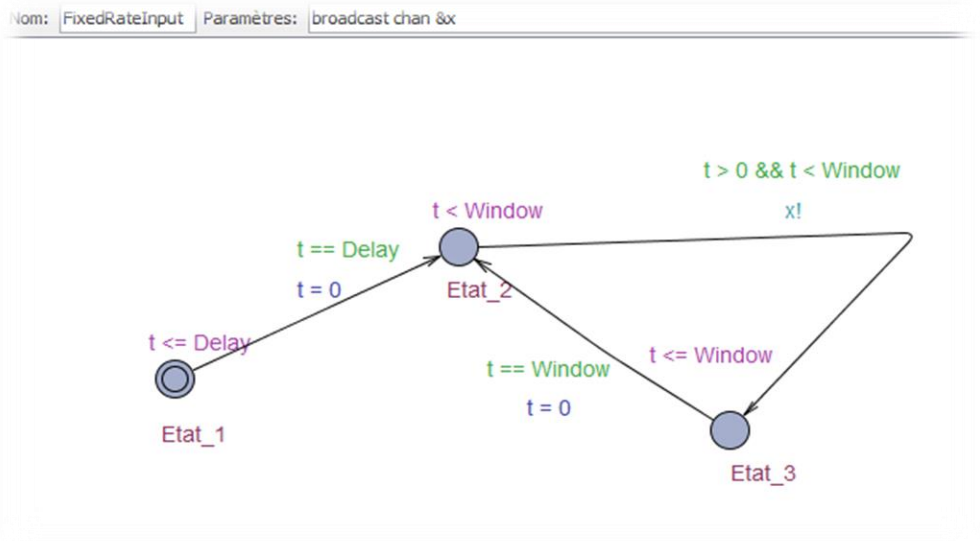


Figure 21 – *modèle de FIX RATE INPUT*

Si le neurones N_E est un NON_DETEMINISTIC input donc Window=, Delay= ,avec l'algorithme Non_deterministic_input la modélisation de Template de l'automate temporisé de ce neurone dans UPPAAL (voir la figure 22).

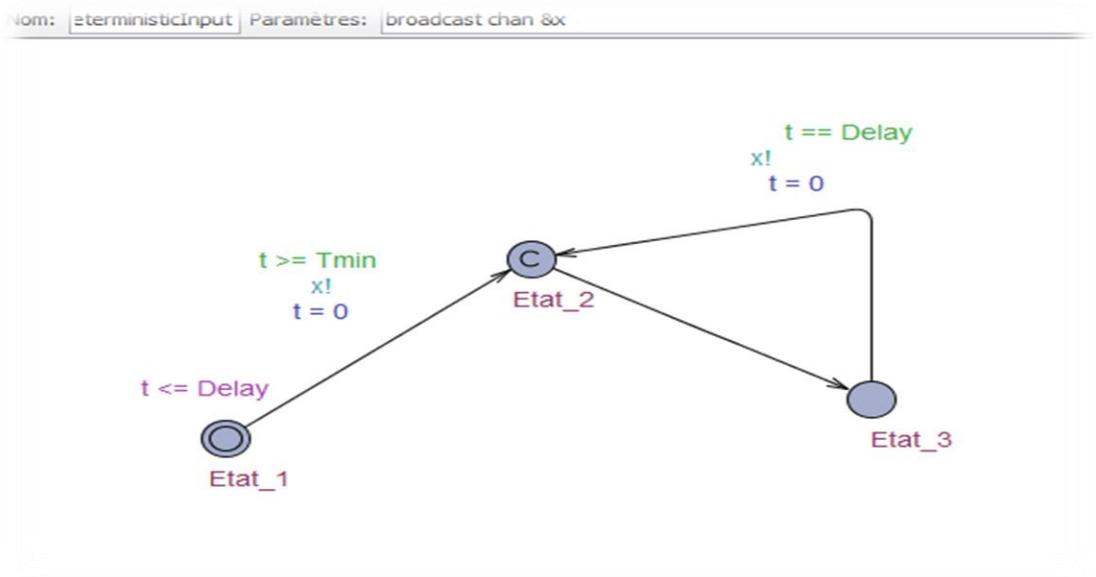


Figure 22 –*modèle de Non Deterministic Input*

Le neurone N_I 1 est N_i 2 avec S=1 (S : numéro des synapse d'entrée) est neurone intermédiaire donc avec l'algorithme Neurone_Intermédiaire la modélisation de Template de l'automate temporisée de se neurone dans UPPAAL (voir la figure 23).

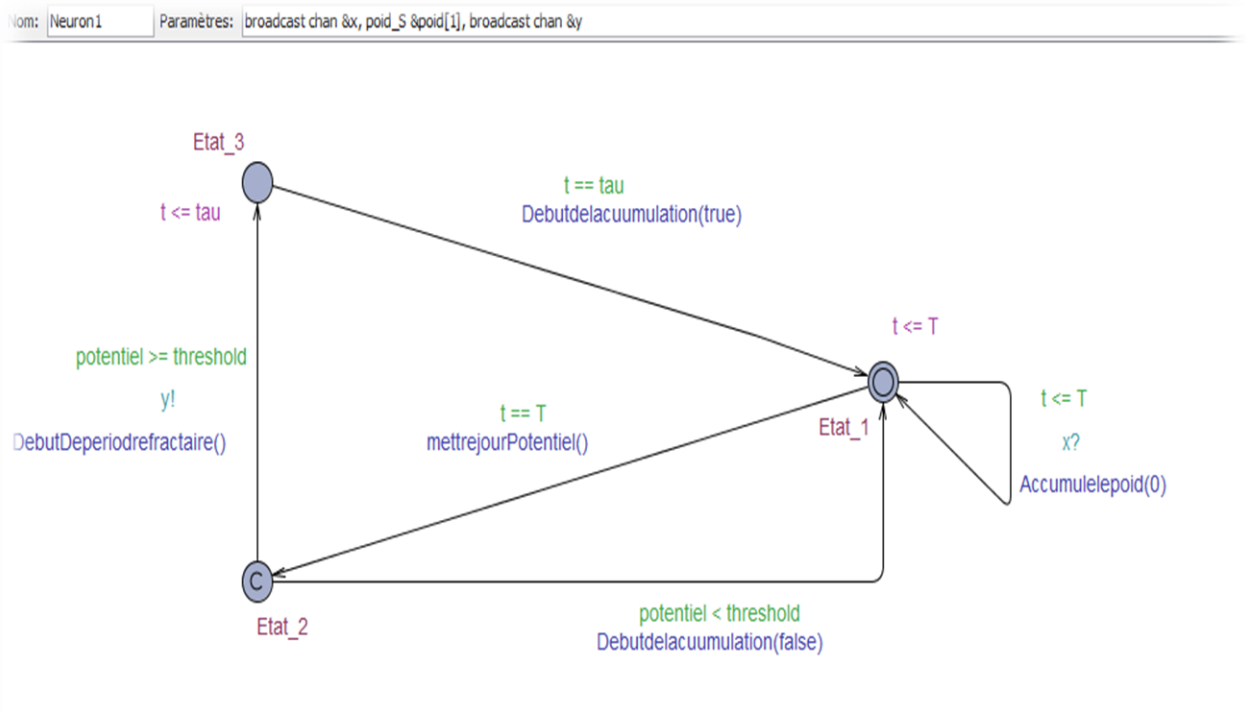


Figure 23 – *modèle de Neurone Intermédiaire avec un synapse*

Le neurone N_I 3 avec S=2 (S : numéro des synapse d'entrée) est neurone intermédiaire donc avec l'algorithme Neurone_Intermédiaire la modélisation de Template de l'automate temporisée de ce neurone dans UPPAAL (voir la figure 24).

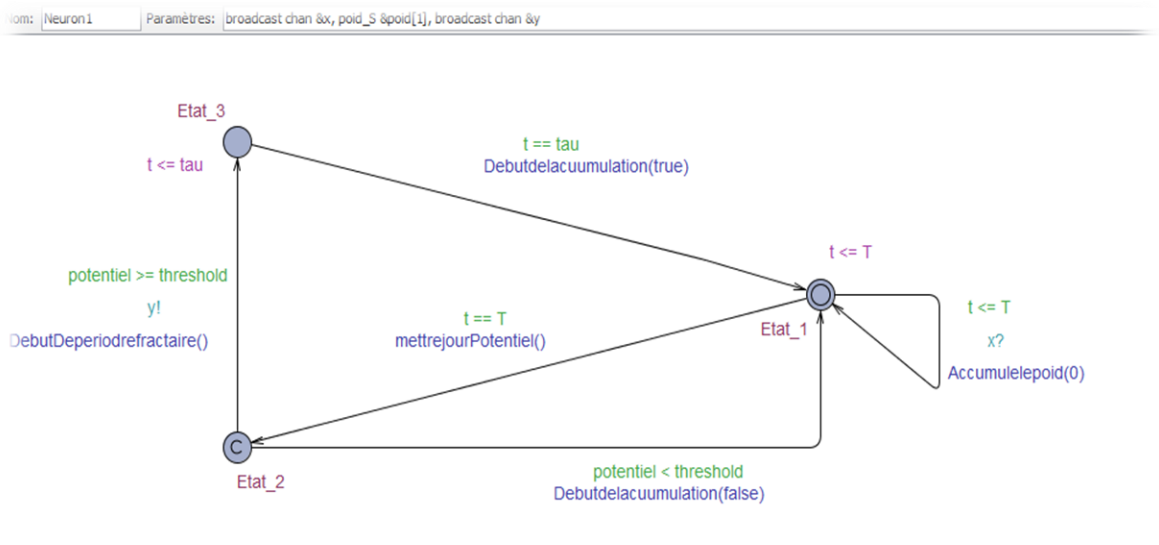


Figure 24 – *modèle de Neurone Intermédiaire avec deux Synapses*

Le neurone N_o est neurone de sortie donc avec l'algorithme Neurone_Sortie la modélisation de Template de l'automate temporisée de se neurone dans UPPAAL (voir la figure 25).

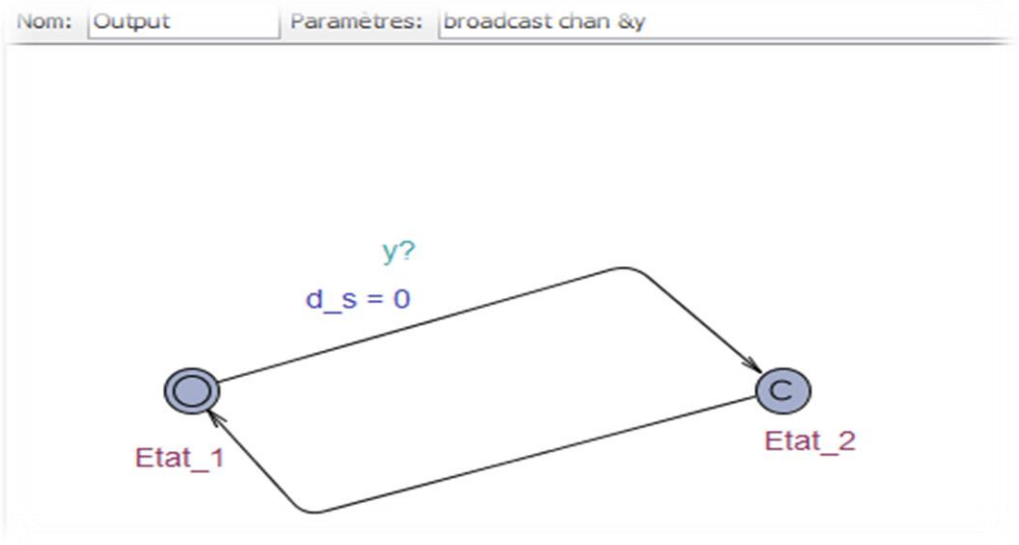
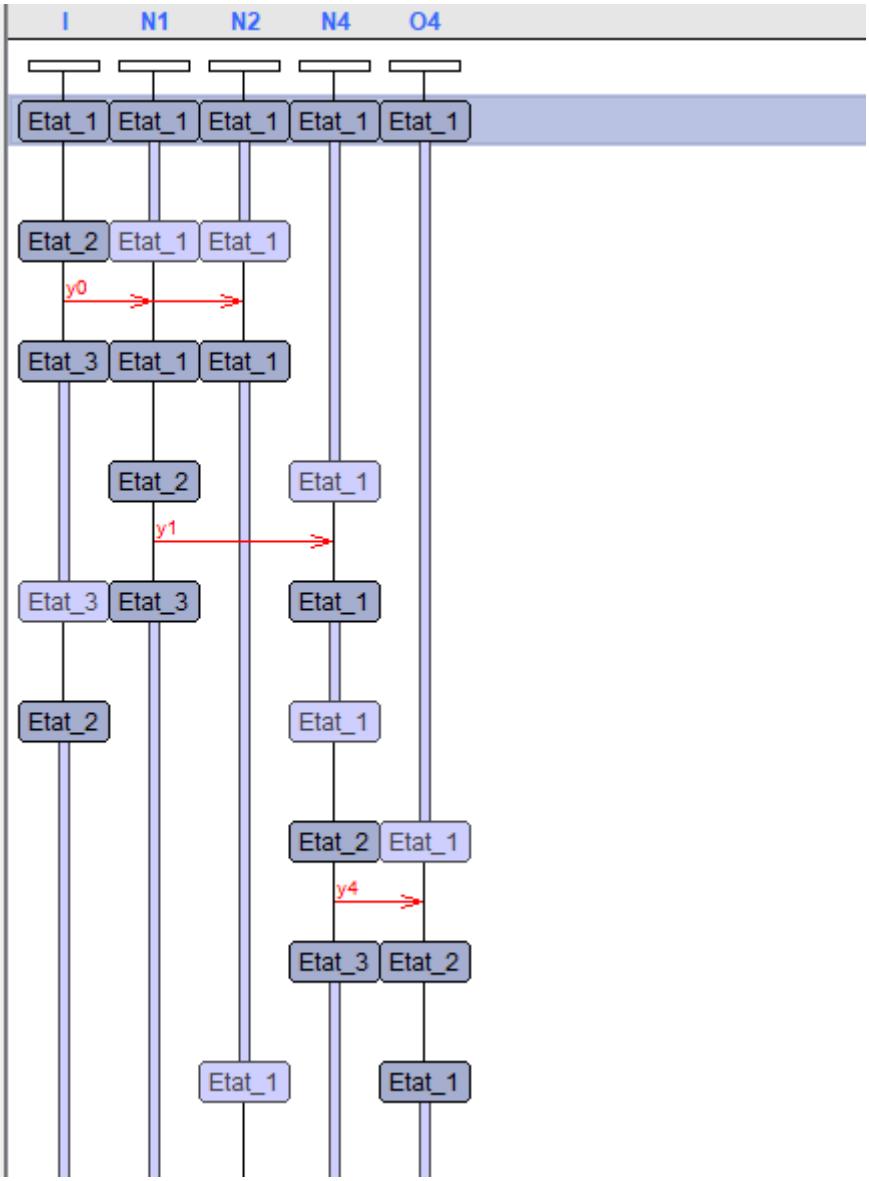


Figure 25 – *modèle de Neurone de Sortie*

La Simulation:

Cette figure représente un fragment de la simulation faite.



Le neurone Input envoie Input a partir de broadcast chanel Y_0 a N_1 et N_2

Le neurone N_1 dans l'etat_2 envoie à neurone N_4 à partir le broadcast chanel Y1

Le neurone N4 dans Etat_2 envoie à neurone de sortie O à partir le broadcaste chanel Y_4.

Cette opération de simulation nous permettre de mieux comprendre le fonctionnement de ce type de réseaux ainsi de détecter les anomalies possibles. La figure précédente

représente un diagramme de séquence issu de l'outil UPPAAL comme un résultat de la simulation.

La vérification :

L'objectif initial de ce travail était d'arriver à vérifier quelques propriétés attendues en utilisant UPPAAL. Malheureusement, cette étape n'a pas était complètement faite. L'obstacle majeur était le temps écoulé dans la modélisation, et le deuxième obstacle c'était le mauvais départ sur le choix du modèle lui-même, c à d les automates temporisés. Ces derniers sont étendus par les probabilités pour construire les automates temporisés probabilistiques. Cette extension est utilisée par UPPAAL SMC qui nous permettre de tracer des courbes pour vérifier le taux d'exactitude et de le comparer avec celui des RNI. Nous comptons continuer à travailler sur le sujet pour atteindre cet objectif.

Conclusion Générale

Notre travail rentre dans le cadre de la vérification formelle précisément la technique de Model Checking. Nous avons choisi de travailler sur les réseaux de neurones impulsionnels qui sont un modèle très répondu dans la technologie de nos jours, et dans tous les domaines de l'industrie. Les SNNs sont utilisés même dans le domaine de la biologie et l'agriculture. L'objectif de notre travail est de tester le bon fonctionnement des systèmes modélisés par les SNNs. Pour cela, nous avons fait appel à un Model Checker qui est UPPAAL. L'utilisation d'UPPAAL est motivé par la technique de model Checking qui est une technique automatique et exhaustive pour vérifier tels systèmes, ainsi que les outils nécessaires pour modéliser avec des automates temporisés et de simuler les systèmes en question. Il est clair que les deux modèles, SNNs et automates temporisés ont des ressemblances. Ces deux modèles reposent sur un modèle discret de temps. La validation de notre travail est faite sous forme d'une comparaison entre les résultats obtenus par les deux modèles. Nous avons conclu que les SNNs ont un avantage par rapport aux automates temporisés car ils ne nécessitent pas un très grand espace mémoire ni un temps de calcul énorme, contrairement aux ATs qui ont le fameux problème d'explosion combinatoire. Le problème pour les SNNs est le taux d′exactitude qui est généralement modéré, contrairement à la technique de Model Checking qui nous offre un taux de vérification élevé et qui surtout nous offre des contres exemples qui nous permettent de mieux comprendre le système en question. Un autre point fort pour la technique de Model Checking est la facilité d'exprimer des propriétés. Ces dernières sont exprimées en logique temporelle TCTL qui est facile à maitriser même pour les non spécialistes. Notre modélisation peut être améliorée par l'ajout de quelques exceptions tel que le cas où les synapses envoient plusieurs données dans la même période. Nous projetant même à travailler sur d'autres modèles des SNNs tel que les modèle H&H.

Bibliographie

[1] :URL :https://www.researchgate.net/publication/319939107_Les_Reseaux_de_Neurones_Artificiels.

[2] : Contreras, Ivan, and Josep Vehi. "Artificial intelligence for diabetes management and decision support: literature review." *Journal of medical Internet research* 20.5 (2018): e10775.

[3]: Parizeau, Marc. "Réseaux de neurones." *GIF-21140 et GIF-64326* 124 (2004).

[4] : Ammar, Mohamed Yessin. *Mise en œuvre de réseaux de neurones pour la modélisation de cinétiques réactionnelles en vue de la transposition batch/continu*. Diss. 2007.

[5]: Iakymchuk, Taras, et al. "Simplified spiking neural network architecture and STDP learning algorithm applied to image classification." *EURASIP Journal on Image and Video Processing* 2015.1 (2015): 1-11.

[6]: Booij, Olaf, and Hieu tat Nguyen. "A gradient descent rule for spiking neurons emitting multiple spikes." *Information Processing Letters* 95.6 (2005): 552-558.

[7] : MILI, Soufyane. Spécification et vérification des systèmes temps réel. Diss. 2017.

[8] : Rached, Miloud. *Spécification et vérification des systèmes temps réel réactifs en B*. Diss. Thèse de doctorat, Université Paul Sabatier, Toulouse, France, 2007.

[9] : Ouazar, Fatiha, and M. C. Boukala. *Vérification distribuée des systèmes temps réel*. Diss. 2008.

[10] : Hernandez, Florent, Tuteur LIRMM, and Rodolphe Giroudeau. "Mémoire de stage de Master." (2007).

[11]: Falez, Pierre. Improving Spiking Neural Networks Trained with Spike Timing Dependent

Plasticity for Image Recognition. Diss. Université de Lille, 2019.

[12] : Sazlı, Murat Hüsnü. "A brief review of feed-forward neural networks." (2006).

[13]: Poznyak, Tatyana, Jorge Isaac Chairez Oria, and Alex Poznyak. *Ozonation and biodegradation in environmental engineering: Dynamic neural network approach*. Elsevier, 2018.

[14]: Katte, Trupti. "Recurrent neural network and its various architecture types." *International Journal of Research and Scientific Innovation (IJRSI)* 5 (2018): 124-129.

[15]: Wu, Jianxin. "Introduction to convolutional neural networks." *National Key Lab for Novel Software Technology. Nanjing University. China* 5 (2017): 23.

[16]: O'Shea, Keiron, and Ryan Nash. "An introduction to convolutional neural networks." *arXiv preprint arXiv:1511.08458* (2015).

[17]: Sakib, Shadman, et al. "An Overview of Convolutional Neural Network: Its Architecture and Applications." (2019).

[18]: Mandic, Danilo, and Jonathon Chambers. *Recurrent neural networks for prediction: learning algorithms, architectures and stability*. Wiley, 2001.

[19]:URL: https://www.edrawsoft.com/neural-network-examples.html.

[20] : Houssou, Mohammed. Optimisation de la structure des réseaux de neurones par algorithmes génétiques. Diss. Boumerdes, 2005.

[21]:https://www.researchgate.net/publication/311949848_Reseaux_de_Neurones_Impulsionnels_pour_la_Segmentation_des_Images_et_la_Detection_des_Contours.

[22] : Mouraud, Anthony. "Approche distribuée pour la simulation événementielle de réseaux de neurones impulsionnels." Université des Antilles et de la Guyane, le 25 (2009).

[23] : Behrmann, Gerd, Alexandre David, and Kim G. Larsen. "A tutorial on UPPAAL 4.0 (Updated November 28, 2006)."

[24] : CHARFI, Faez. Une approche dinterfaage de CoD UPPAAL pour la spcification et la vrification des systmes temps rel. Diss. Thesis, September, 2003.

[25] : Moussa, Majda. Vérification et configuration automatiques de pare-feux par Model Checking et synthèse de contrôleur. Diss. École Polytechnique de Montréal, 2014.

[26] : Katoen, Joost-Pieter. Concepts, algorithms, and tools for model checking. Erlangen: IMMD, 1999.