Session : S7

Projet de spécialité GIN950 et GIN955

# Projet de recherche sur les réseaux de neurones à décharges

**Rapport**

**Par :**

**Samir Lechekhab** **LECS2813**

**Signature :** ..... **Date :** .....

**Département de Génie électrique et de Génie informatique**

Université de Sherbrooke

# Table des matières

# 1 Introduction aux réseaux de neurones à décharge

L'objectif du projet est, dans un premier temps, de se familiariser avec les réseaux de neurones à décharge(principe de fonctionnement, architecture, simulations,...). Dans un second temps l'objectif est de réussir à créer un réseau de neurone capable d'effectuer la reconnaissance de note de musique jouée à la guitare, pour à terme, réécrire la partition d'un fichier audio passé en entrée. Pour tenter d'atteindre cette objectif différentes approches ont été testées et détaillées dans ce document.

# 2 Le neurone en biologie

## 2.1 Les généralités

Un neurone est une des principales cellules du système nerveux central et du système nerveux périphérique, il s'agit d'une cellule excitable qui, lors d'un stimulus, entraîne la formation d'un signal électrique qui libérera des neurotransmetteurs pour communiquer avec d'autres neurones ou cellules et ainsi traiter de l'information. Il y en a environ 86 milliards dans le cerveau.

## 2.2 Structure

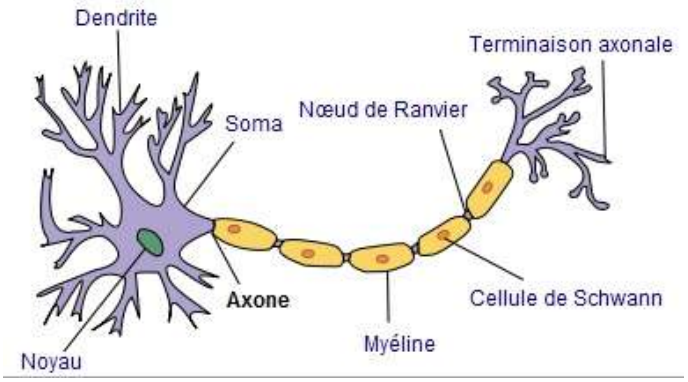


FIGURE 1 – Schéma d'un neurone et de ses différents composants. Tirée de futura-sciences.com

### 2.2.1 Le soma

Le soma est le corps de la cellule, en son centre se trouve le noyau. Le soma sert à produire les molécules qui servent à faire fonctionner la cellule ainsi que les neurotransmetteurs qui servent quant à eux à transmettre chimiquement l'information vers d'autres neurones.

### 2.2.2 Les dendrites

les dendrites sont les prolongements filamenteux qui partent du soma, leur fonction est de recevoir l'information en provenance des autres neurones.

### 2.2.3 l'axone

L'axone est un long prolongement de la cellule, son rôle est de transmettre l'information sous forme électrique aux terminaisons axonales.

### 2.2.4 La couche de myéline

La couche de myéline est une sorte de gaine isolante se trouvant autour de l'axone pour la protéger et aussi améliorer la vitesse de propagation de l'influx nerveux.

### 2.2.5 Les terminaisons axonales

C'est ici que l'information sous forme d'un courant électrique laisse place à des neurotransmetteurs qui voyageront dans les synapses (zone d'échange des neurotransmetteurs) pour transmettre l'information au prochain neurone.

# 3 Les réseaux de neurones à décharges

Il existe différent type de réseaux de neurones, dans notre cas nous allons explorer uniquement les réseaux de neurones à décharges abrégés SNN pour *spiking neural network*.
Les SNN sont la 3ème générations de réseau de neurones et sont beaucoup plus proche du fonctionnement d'un réseau de neurones biologique. Contrairement aux réseaux de neurones couramment utilisés pour le "Machine learning", les SNNs fonctionnent en utilisant des impulsions qui sont des événements discrets qui ont lieu à des moments précis dans le temps. Ces impulsions sont déterminées par des équations différentielles représentant le potentiel de la membrane du neurone. Quand un neurone reçoit une impulsion, il augmente son potentiel jusqu'à atteindre un seuil et revenir à son niveau de repos. Pour plus de réalisme il est possible de rajouter une période de "refactoring" directement après l'émission d'une impulsion pour empêcher toute modification du potentiel de la membrane.

## 3.1 Domaine d'utilisation des réseaux de neurones à décharges

Comme dit précédemment, les réseaux de neurones à décharges sont principalement utilisés par des neuroscientifiques pour leur grande ressemblance aux neurones biologiques et donc faire des simulations. Les réseaux de neurones à décharge sont aussi utilisés en informatique, mais cependant le fait de simuler un comportement analogique consomme énormément de ressource.

## 3.2 Leaky integrate-and-fire

Le modèle le plus couramment utilisé, de par sa simplicité de simulation et d'analyse pour représenter ce phénomène est le modèle "Leaky Integrate-and-fire" abrégé LIF. L'information n'est donc pas contenue dans une valeur continue, mais dans la composante temporelle des impulsions délivrées par chaque neurones.

### 3.2.1 Modèle mathématique

$$\tau_m \frac{dv}{dt} = -v(t) + RI(t)$$

Où *v(t)* correspond au potentiel de la membrane à un instant donné, $\tau_m$ est la constante de temps de la membrane, où $R$ est la résistance de la membrane. Le seuil n'est pas visible dans cette équation, mais il consiste en une valeur, qui une fois atteinte, émet une impulsion et remet la valeur de la membrane à sa valeur de *reset*.

### 3.2.2 Premier test avec un neurone de type LIF

Dans ce premier exemple nous allons stimuler un neurone de type LIF avec un courant d'entrée continu pour augmenter de manière constante le potentiel de la membrane, le faire atteindre son seuil et ainsi de suite, on devrait alors observer une émission de décharge à intervalle régulier.

### 3.2.3 Résultat

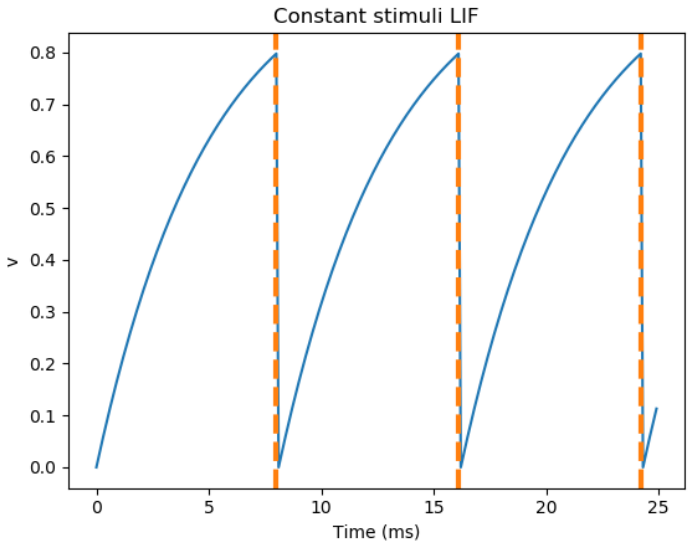


FIGURE 2 – Graphe du potentiel de la membrane dans le temps. On peut y observer les impulsions représentées par les traits orange. Généré par le code Pyhton *1 HelloWorldLIF.py*

## 3.3 Code

```
from brian2 import *

tau = 10*ms
v_th = 'v>0.8'
v_rst = 'v=0'
eqs = '''
dv/dt = (1-v)/tau : 1
'''

LIF = NeuronGroup(1, eqs, threshold=v_th, reset=v_rst, method='linear')
Monitor = StateMonitor(LIF, 'v', record=0)

run(50*ms)
plot(Monitor.t/ms, Monitor.v[0])
xlabel('Time (ms)')
ylabel('v')
title('Constant stimuli LIF');
show()
```

### 3.3.1 Analyse du code

Dans un premier temps nous définissons l'équation différentielle du comportement du neurone de type *LIF* avec un courant constant d'entrée constant de 1. Finalement nous utilisons un objet *StateMonitor* pour enregistrer, tout au long de la simulation, une variable.

# 4 Les synapses

En biologie, les synapses sont les zones se trouvant entre deux neurones qui permettent l'échange d'informations entre un neurone émetteur, via des neurotransmetteurs, pour stimuler un neurone récepteur. Avec Brian2 le fonctionnement est le même pour faire communiquer deux neurones entres eux on va instancier une synapse pour connecter deux neurones d'un même groupe ou d'un groupe différent. De plus il est possible de définir un poids pour chaque synapse afin de stimuler plus ou moins le neurone récepteur.

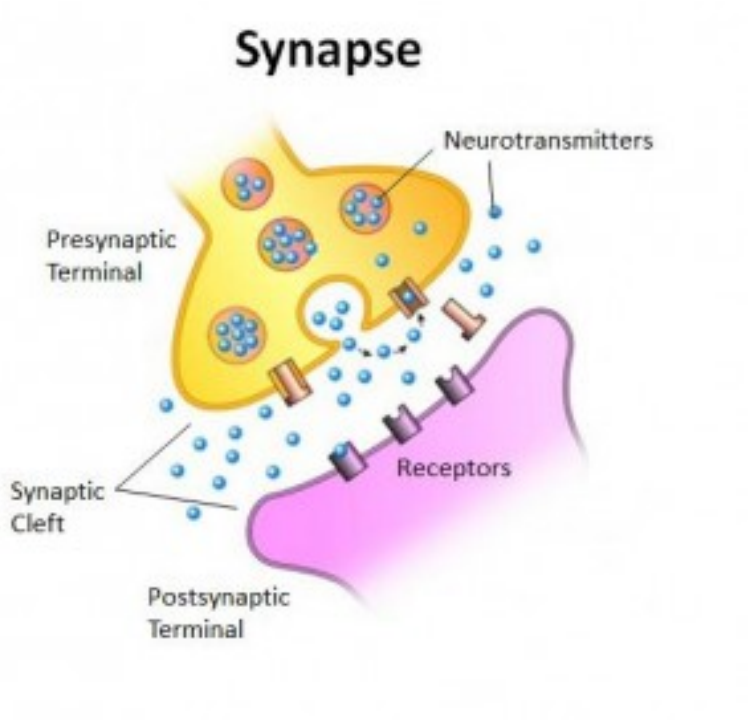


FIGURE 3 – Représentation du fonctionnement d'une synapse. Tirée de https : www.pinterest.fr

## 4.1 Exemple

Dans cet exemple nous allons créer un groupe de 4 neurones de type LIF dont un, qui servira de neurone transmetteur, sera constamment stimulé et sera connecté via des synapses à des neurones récepteurs. De plus chaque synapse aura un poids différent qui sera incrémenté de 0.2 selon l'indice du neurone récepteur. On peut donc s'attendre à voir le neurone émetteur envoyer une impulsion à intervalle régulier et les neurones récepteurs envoyer une impulsion en différé, respectivement pour le deuxième neurone enverra des impulsions à une fréquence deux fois moins élevée que le premier récepteur et le troisième à une fréquence trois fois moins élevée que le premier.

### 4.1.1 Résultat

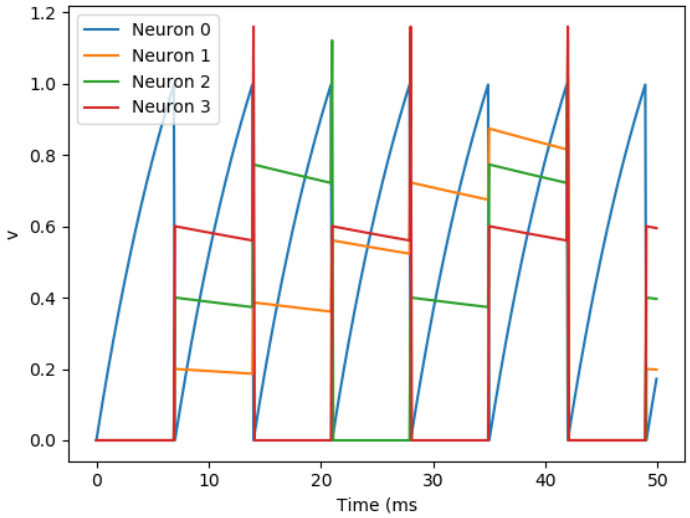


FIGURE 4 – Graphe du potentiel de la membrane dans le temps pour chaque neurone. Généré par le code Pyhton *5 SimpleSynapses.py*

### 4.1.2 Code

```
#Simple demonstration de l'utilisation de synapses pour
#connecter 4 neurones entre eux.
from brian2 import *

eqs = '''
dv/dt = (I-v)/tau : 1
I : 1
tau : second
'''

G = NeuronGroup(4, eqs, threshold='v>1', reset='v = 0', method='exact')
G.I = [2, 0, 0, 0]
G.tau = [10, 100, 100, 100]*ms

S = Synapses(G, G, 'w : 1', on_pre='v_post += w')
S.connect(i=0, j=[1,2,3])
S.w = 'j*0.2'

M = StateMonitor(G, 'v', record=True)

run(50*ms)

plot(M.t/ms, M.v[0], label='Neuron 0')
plot(M.t/ms, M.v[1], label='Neuron 1')
plot(M.t/ms, M.v[2], label='Neuron 2')
plot(M.t/ms, M.v[3], label='Neuron 3')
xlabel('Time (ms')
ylabel('v')
legend()
show()
```

### 4.1.3 Analyse du code

Dans un premier temps nous définissons l'équation différentielle du comportement du neurone, une fois ceci fait nous créons un groupe de 4 neurones avec une valeur de threshold de 1 et une valeur de reset de 0. La ligne suivante applique un courant constant de 2 à l'entrée du premier neurone et un courant nul à l'entrée des trois autres. Ensuite nous créons les synapses pour connecter le premier neurone aux trois autres en appliquant des poids synaptiques différents à chaque synapse.

# 5 Encodage de l'information

Lorsque l'on programme des SNNs il est important de définir la manière dans laquelle on va encoder l'information, passer de valeurs analogiques en impulsion. Pour ceci il existe plusieurs méthodes :

## 5.1 Codage fréquentiel

La première méthode est d'encoder l'information dans la fréquence d'émission des impulsions.

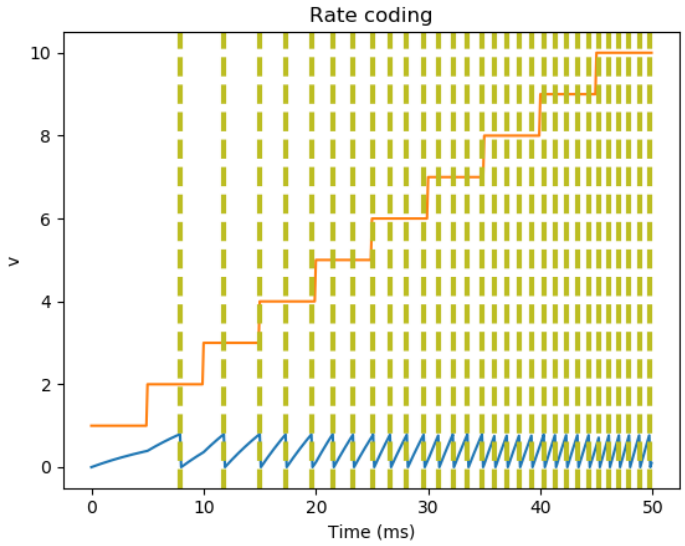


FIGURE 5 – Spike times : [ 7.9 11.8 15. 17.3 19.6 21.5 23.3 25.1 26.6 28.1 29.6 30.9 32.2 33.5 34.8 35.9 37. 38.1 39.2 40.3 41.3 42.3 43.3 44.3 45.2 46.1 47. 47.9 48.8 49.7] ms. Généré par le code Pyhton *4 Rate coding.py*

Dans cet exemple nous augmentons par palier la valeur d'entrée (en orange) toutes les 5ms et l'on peut remarquer que l'intervalle entre deux impulsions (traits verts pointillés) diminue en fonction de cela.

1. 4 mV : 434.78Hz
2. 5 mV : 555.55Hz
3. 6 mV : 666.66Hz
4. 7 mV : 769.23Hz
5. 8 mV : 909.09Hz
6. 9 mV : 1000.00Hz
7. 10 mV : 1111.11Hz

## 5.2 Codage temporel

La deuxième est de représenter l'information dans l'intervalle de temps avant la première impulsion par exemple pour une valeur d'entrée de 0 à 10, le temps avant la première impulsion varie entre 0 et 10 ms

# 6 Projet de détection de note de musique à l'aide d'un réseau de neurones à décharge

Le but de ce projet de créer un réseau de neurones capable d'apprendre à reconnaître des notes de musique jouées à la guitare pour ensuite pouvoir réécrire la partition. Étant donné que le principe de fonctionnement des neurones à décharges est très proche du principe de fonctionnement des neurones biologiques, il me semble approprié d'utiliser un mécanisme d'apprentissage biologique appelé le principe de plasticité synaptique en fonction du temps d'occurrence des impulsions ou abrégé STDP en anglais.

# 7 Principe de plasticité synaptique en fonction du temps d'occurrence des impulsions

STDP est un processus biologique qui va s'occuper de modifier la force des connexions entre les neurones du cerveau dans un processus d'apprentissage en créant soit une potentialisation à long terme (PLT) qui aura pour effet d'augmenter de manière durable le taux de transmission d'une synapse. Ou au contraire, créer une dépression synaptique à long terme (DLT) qui aura pour effet de diminuer le taux de transmission d'une synapse. Le choix de créer une PLT ou une DLT est basé sur la règle de Hebbian qui dit la chose suivante :

> « Faisons l'hypothèse qu'une activité persistante et répétée d'une activité avec réverbération (ou trace) tend à induire un changement cellulaire persistant qui augmente sa stabilité. Quand un axone d'une cellule A est assez proche pour exciter une cellule B de manière répétée et persistante, une croissance ou des changements métaboliques prennent place dans l'une ou les deux cellules ce qui entraîne une augmentation de l'efficacité de A comme cellule stimulant B. »
> Donald Hebb, 1949.

Que l'on peut résumer par « cells that fire together, wire together ». Ce qui est assez simple a comprendre si un neurone récepteur envoie une impulsion peu de temps après qu'un neurone $A$ en ai envoyé une cela signifie que $A$ a une grande influence sur celui-ci donc leur lien doit se renforcer, dans le cas inverse où le neurone récepteur envoie une impulsion avant que le neurone $A$ lui en ai envoyé une cela signifie donc que le neurone $A$ n'a qu'une influence faible sur le neurone récepteur.

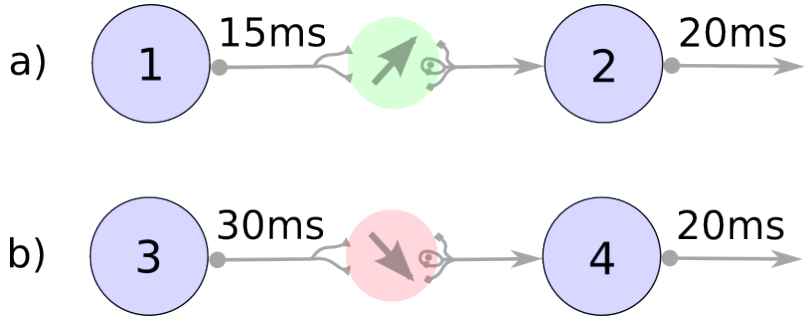


FIGURE 6 – Exemple du fonctionnement du principe de plasticité synaptique en fonction du temps d'occurrence des impulsions. Triée de neurons.readthedocs.io

La courbe suivante représente la manière avec laquelle une synapse adapte son poids en fonction de la différence de temps entre deux impulsions.

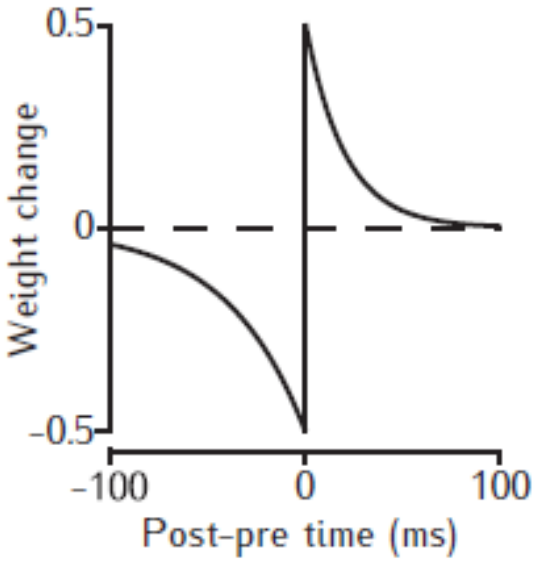


FIGURE 7 – Graphique de l'augmentation du poids synaptique en fonction de la différence de temps d'émission d'impulsion entre un neurone pré synaptique et d'un neurone post-synaptique. Tirée de www.compneuroprinciples.org

# 8 Implémentation avec Brian2

La première étape est de créer un groupe de neurones identique de type *LIF* que nous allons connecter les uns aux autres avec des synapses dont la force des connexions est une valeur aléatoire à l'état initial et on décrie le principe de STDP en donnant le comportement suivant aux synapses :

```
...

S = Synapses(input, neurons,
                '''w : 1
                   dApre/dt = -Apre / taupre : 1 (event-driven)
                   dApost/dt = -Apost / taupost : 1 (event-driven)''',
               on_pre='''ge += w
                         Apre += dApre
                         w = clip(w + Apost, 0, gmax)''',
               on_post='''Apost += dApost
                          w = clip(w + Apre, 0, gmax)''',
               )
S.connect(p=1.0)
S.w = 'rand()*gmax'

...
```

Dans cette partie de code nous définissons le comportement de la variable $w$ qui correspond au poids synaptique des connexions, lorsque deux neurones envoient une impulsion en même temps le poids synaptique augmentera et dans le cas contraire il diminuera, ce poids sera compris dans l'intervalle de 0 à *gmax*.

# 9 Solution de détection de fréquence en fonction de la constante de temps du neurone

Dans cette première solution, nous utilisons un neurone d'entrée avec une constante de temps spécifique pour envoyer un nombre d'impulsions correspondant à la fréquence d'entrée du signal, ce neurone est connecté via des synapses à d'autres neurones envoyant des impulsions en fonction du nombre d'impulsions reçu. Cependant les neurones détectant les basses fréquences émettent aussi lors de plus haute fréquence, c'est pour ça que les neurones détectant les hautes fréquences servent aussi d'inhibiteur pour les neurones de plus basses fréquences.

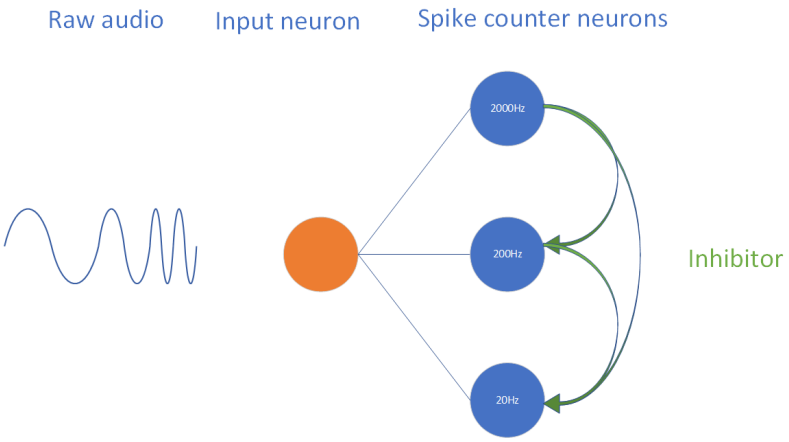


FIGURE 8 – Un premier neurone convertit la fréquence du signal d'entrée en un certain nombre d'impulsions. Les neurones suivants comptent le nombre d'impulsions pour déterminer la fréquence et inhibent les neurones de plus basse fréquence.

# 10 Différentes solutions envisageables avec STDP

Dans les différentes approches que nous allons voir, quelques bases sont communes, comme le fait d'utiliser les mécanismes de STDP pour l'apprentissage, d'avoir une phase d'entraînement où les poids synaptiques vont changer leurs valeurs pour ensuite être enregistrées dans un fichier pour pouvoir être réutilisées plus tard lors de la détection de notes, ceci se fait via la librairie Pickle.

## 10.1 Apprentissage avec en entrée un audio brut

Ceci est la première solution qui a été testée, il s'agit de prendre des fichiers audio au format ".wav" convertis en un tableau amplitude en fonction du temps pour stimuler un neurone d'entrée connecté à chaque neurone de la couche cachée.

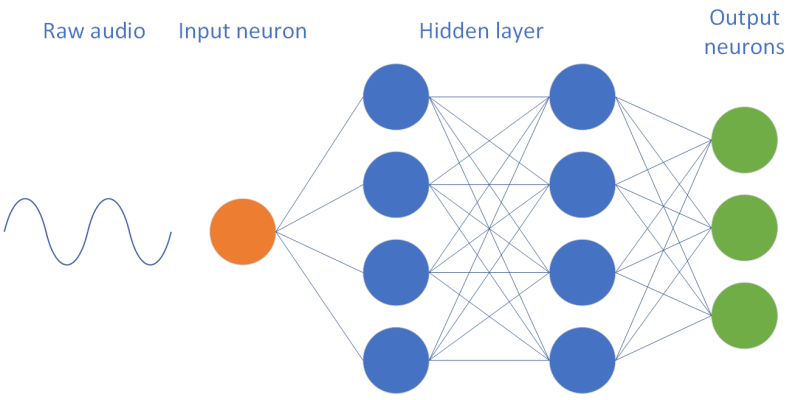


FIGURE 9 – Conversion analogique vers impulsions d'une entrée audio.

### 10.1.1 Conversion analogique vers impulsions

Une fois le tableau créé il sert à stimuler un neurone unique d'entrée qui sert de convertisseur "Analogique vers Impulsions" pour ensuite envoyer à chaque neurone de la couche cachée "Hidden layer" le train d'impulsion.

```
...
signal = numpy.array(read("SweetHomeAlabama.wav")[1],dtype=float)
ta = TimedArray(signal/500, dt=0.1*ms)

tau = 5*ms
eqs = '''
dv/dt = (I-v)/tau : 1
I = ta(t, i) : 1
'''
G1 = NeuronGroup(1, eqs, threshold='v>1', reset='v=0', method='euler')
...
```

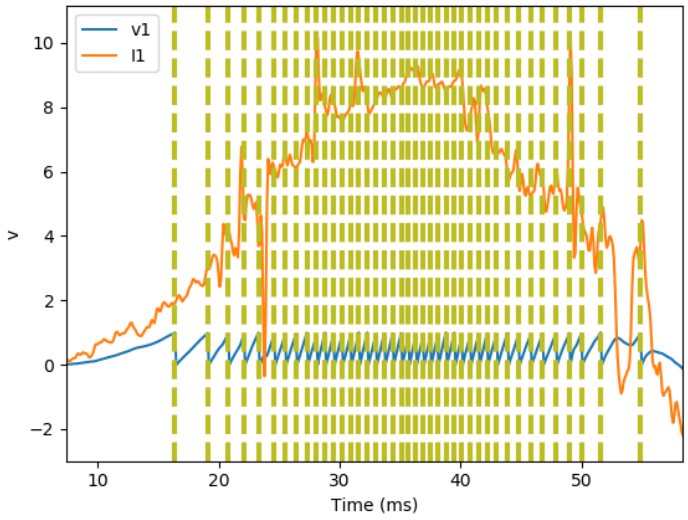


FIGURE 10 – La ligne orange représente le signal audio en entrée. La ligne bleue correspond au potentiel de la membrane et les traits verts représentent les impulsions. Généré par le code Pyhton *3 music input.py*

## 10.2 Apprentissage avec en entrée une série de Fourier

La deuxième solution ressemble à la première à la différence que le signal audio subit dans un premier temps une transformation de Fourier pour stimuler chaque neurone d'entrée avec l'amplitude d'une fréquence spécifique.

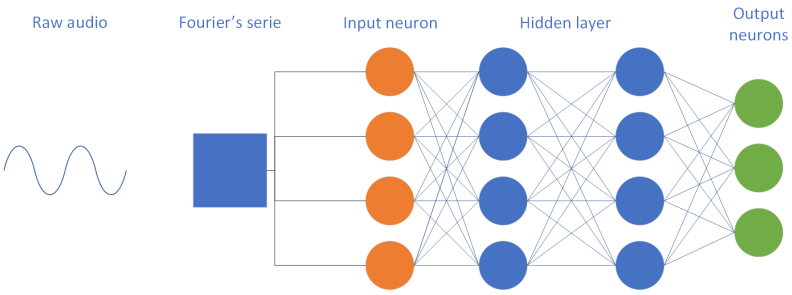


FIGURE 11 – Schéma du réseau de neurones de la deuxième solution.

## 10.3 Apprentissage par reconnaissance de pattern

Cette méthode semble être la plus prometteuse, elle est basée sur une expérience réalisée par une équipe de chercheur constitué de Peter U. Diehl et Matthew Cook en 2014, le principe est d'apprendre à un réseau de neurones à reconnaître et classifier des nombres écrits à la main grâce à la bibliothèque MNIST (cette bibliothèque contient plus de 60000 nombres manuscrits de 28x28 pixels). D'après le rapport intitulé *Unsupervised Learning of Digit Recognition Using Spike-Timing-Dependent Plasticity* ils ont pu atteindre un taux de réussite de plus de 91.56%. L'idée de cette solution est de se baser sur leur réseau de neurones pour reconnaître des spectrogrammes de note de musique. Cependant avec la taille des spectrographes largement supérieure à 28x28 l'apprentissage risque d'être extrêmement lent.

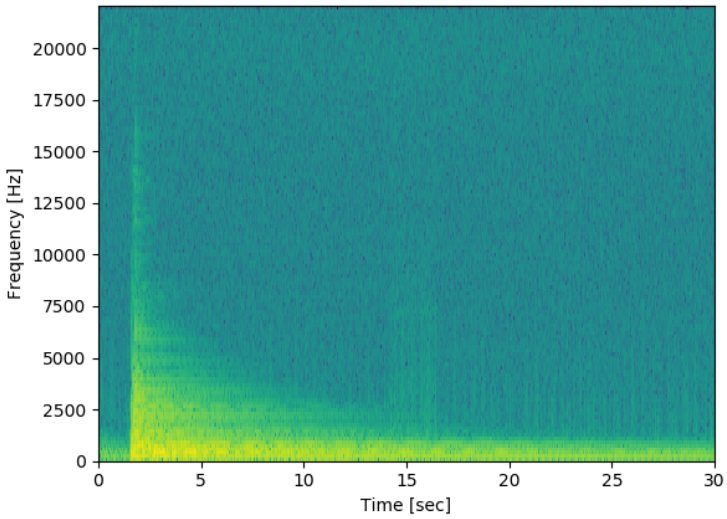


FIGURE 12 – Spectrogramme de la note C majeur jouée à la guitare électrique. Généré par le code Pyhton *7 Spectrograms.py*

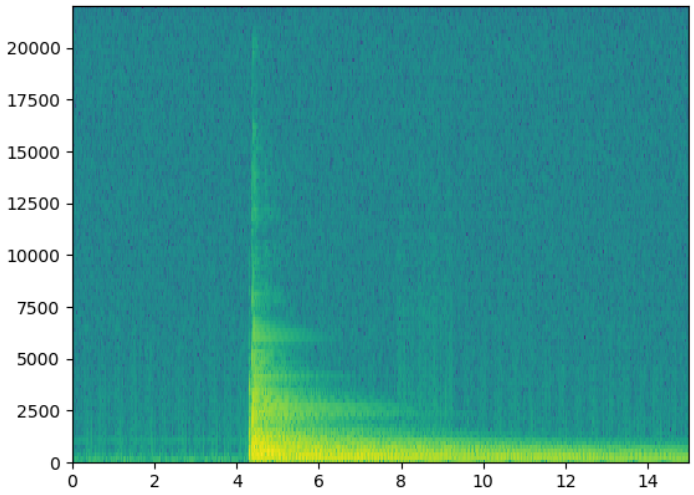


FIGURE 13 – Spectrogramme de la note D majeur jouée à la guitare électrique.

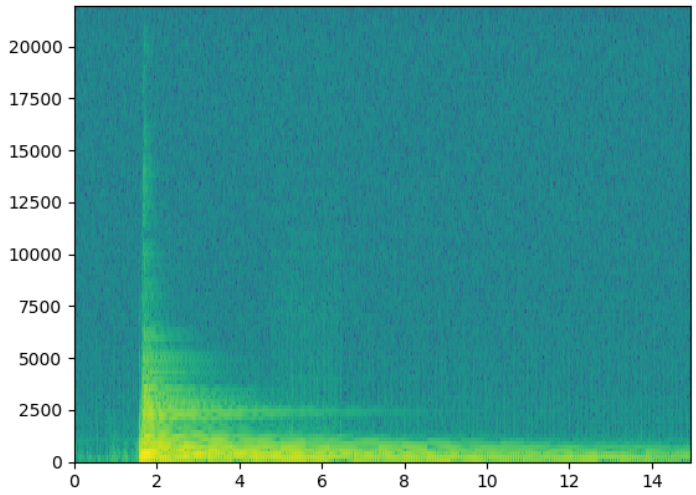


FIGURE 14 – Spectrogramme de la note G majeur jouée à la guitare électrique.

### 10.3.1 Code pour l'extraction des spectrographes

```
import scipy
import matplotlib.pyplot as plt
import scipy.io.wavfile
sample_rate, X = scipy.io.wavfile.read('Guitar/C-major.wav')
print (sample_rate, X.shape )
plt.specgram(X[:,0], Fs=sample_rate, xextent=(0,30))
plt.ylabel('Frequency [Hz]')
plt.xlabel('Time [sec]')
plt.show()
```

### 10.4 Apprentissage par tableaux de fréquences

Cette méthode consiste à effectuer l'apprentissage à l'aide d'un spectrogramme, chaque neurone de la première couche reçois un tableau d'amplitude pour une fréquence spécifique, cette couche est directement connectée à un réseau de neurone avec des liens synaptique de type STDP, après un certain temps d'apprentissage il est possible de voir certaine zone du réseau s'activer dépendamment de la note de musique détectée, une fois ceci fait des neurones de sortis correspondant à chaque note sera connecté aux neurones correspondant à la partie du réseau active.

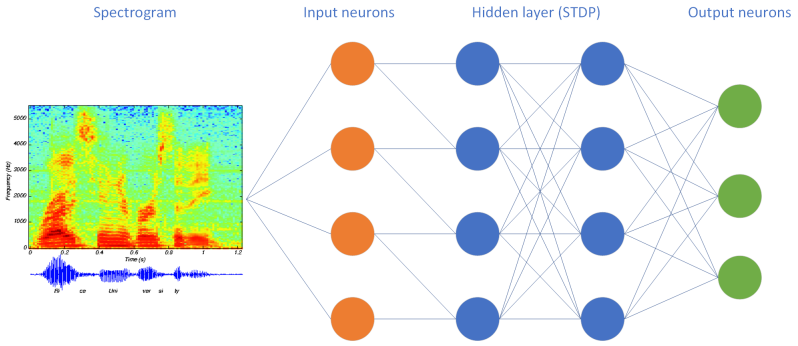


FIGURE 15 –

## 11 Conclusion

Pour conclure, ce projet m'a permis de travailler pour la première fois avec des réseaux de neurones et plus spécifiquement à décharges, lors de ces recherche j'ai pu acquérir des connaissances dans les domaines de la biologie, la neuroscience, l'informatique et de l'audio. Malgré les recherches effectuée et les nombreuses tentatives je n'ai pas pu fournir un code capable de résoudre la problématique que je m'étais fixé. Ceci n'est pas dû à l'architecture des réseaux développer mais à la difficulté de régler les nombreuse variables. car, individuellement, tout les éléments fonctionnaient mais une fois regroupé les réseaux n'avait pas le comportement attendu.

## 12 Lexique

— **ANN** : Artificial Neural Network, réseau de neurones artificiels.
— **SNN** : Spiking Neural Networks, réseau de neurones à décharges.
— **LIF** : Leaky Integrate-and-fire.

## 13 Références

— Figure 1, source :
https://www.futura-sciences.com/sante/definitions/biologie-neurone-209/
— Figure 3, source :
https://www.pinterest.fr/pin/500603314807986934/?autologin=true
— Figure 6, source :
http://neurons.readthedocs.io/en/latest/intro/learn.html
— Figure 7, source :
http://www.compneuroprinciples.org/code-examples/all/all?page=1
— Introduction aux réseaux de neurones à décharge
https://towardsdatascience.com/spiking-neural-networks-the-next-generation-of-machine-learni
— http://neurons.readthedocs.io/en/latest/index.html
— Site web de Brian
http://briansimulator.org/
— Guide d'utilisation de brian2tools
http://brian2tools.readthedocs.io/en/stable/
— Documentation sur les "neurones poisson"
https://www.tu-chemnitz.de/informatik/KI/scripts/ws0910/Neuron_Poisson.pdf
— Apprentissage non-supervisé
https://en.wikipedia.org/wiki/Unsupervised_learning
— Règle de Hebb
https://fr.wikipedia.org/wiki/R%C3%A8gle_de_Hebb
— Document sur STDP
http://isn.ucsd.edu/classes/beng260/2016/reports/2016_Group4.pdf
— Projet de Peter U. Diehl et Matthew Cook
https://github.com/zxzhijia/Brian2STDPMNIST
— Leaky Integrate-and-Fire
http://hopf.cns.nyu.edu/~eorhan/notes/lif-neuron.pdf