Table des matières

I Introduction **3**

II Stimulus **5**
0.1 Objectif 6
0.2 Utilisation 6

III Génération de données **9**
0.3 Introduction 10
0.4 Spikes reçus jusqu'alors 10

IV Additivité **11**
0.5 Théorie(à finir) 12
0.6 Utilisation du script 13
0.7 Résultats 13

V Propagation des spikes **14**
0.8 Utilisation du script 15
0.9 Résultats 15
0.10 Réflexions 15

VI Probabilistic flooding et I/E **16**
0.11 Utilisation du script 17
0.12 Résultats 18
0.13 Application sur dnfsim 18
0.14 Variance pour {20,40,60,80}_asend 18
0.14.1 Utilisation du script 18
0.14.2 Résultats 19

VII Annexe **20**

VIII TODO **21**

Première partie

Introduction

Ce rapport s'inscrit dans une étude des champs neuronaux dynamiques. Jusqu'alors nous travaillons sur le modèle CNFT (INSERT REF HERE) . Rappelons tout d'abord en quoi consiste ce modèle. Nous avons un réseau de neurones. Chaque neurone est connecté avec l'ensemble des autres neurones. Chaque neurone dispose d'un seuil d'activation. Après avoir reçu une charge suffisamment importante en tension, le neurone s'active et envoie un signal aux neurones afférents. Cet envoie suit un routage bien particulier. L'idée est d'envoyer des tensions plus faibles aux neurones les plus éloignés en suivant une courbe type différence de deux gaussiennes. Le problème de ce modèle est qu'il requière une interconnexion de neurones globale. Chaque neurone est lié aux autres neurones. Cet état de fait implique qu'il sera difficile d'implémenter ce routage sur une carte FPGA par exemple.

Afin d'alléger le modèle dans son ensemble, on va s'intéresser à des type de routage qui ne nécessite pas autant de connexions neurales. Ce rapport a donc pour objectif d'exposer les différents outils mis à disposition afin de tester différents routages dans le cadre de ce modèle. On va voir comment générer des données relatifs à ce routage et comment les analyser. La mise en pratique sera faite avec le routage "Probabilistic flooding" (INSERT REF HERE)

Deuxième partie

Stimulus

0.1 Objectif

On peut exciter chacun des nœuds du graphe à différents moments. L'objectif est de voir le comportement du graphe (et du routage associé) selon différents événements. Ces événements sont caractérisés par l'envoie de stimulus, i.e l'ajout de paquet à la file d'attente d'un ou plusieurs nœuds à un ou plusieurs moments de la simulation.

0.2 Utilisation

Le framework pour la programmation d'envoi de stimulus se base sur le format xml. Tout les fichiers de programmation doivent se trouver sous le dossier statistiques/stimulis avec l'extension .stimulis (et non .xml). Cette extension est nécessaire aux fichiers de scripts notamment.

Commençons par un exemple. Admettons que l'on veuille envoyer un paquet de type spike au nœud zéro du graphe à l'instant t=0 :

asend.stimulis

```
<?xml version="1.0" encoding="UTF-8"?>
<programation class="main.java.network.generic.packet.Spike">
        <time t="0.0">
                <add indice="0"/>
        </time>
</programation>
```

On retrouve ici trois balise :

- <programation> : On stipule qu'on va commencer un programme d'envoi stimulis (il y a une faute à "programation" mais j'ai codé ça comme ça). L'attribut "class" est le nom de la class Java des paquets que l'on veut envoyer en tant que stimulis.
- <time> : Pour spécifier les envois à un temps t donné. A noté que

$$dtStimulis = k * dtGraphe \ \forall k \epsilon \mathbb{N}^*$$

  i.e on ne peut pas envoyer des stimulis entre deux computations.
- <add> : On envoie un paquet à l'indice indice. Notez que les indices sont en représentation 1D du graphe.

Un exemple d'envoi de paquets ipv4 sur un réseau :

ipv4.stimulis

```
<?xml version="1.0" encoding="UTF-8"?>
<programation class="main.java.network.generic.packet.IPv4Datagramme">
        <time t="0.0">
                <add indice="0">
                        <params> A </params>
```

```
                        <params> yes </params>
                        <params> ca fonctionne ! </params>
                </add>
        </time>
        <time t="0.1">
                <add indice="80">
                        <params> B </params>
                        <params> yes </params>
                        <params> ca fonctionne ! </params>
                </add>
        </time>
</programation>
```

On découvre la balise <params>. L'utilisation de cette balise est directement liée à la structure du constructeur de la class spécifiée dans la balise programation. Dans l'exemple précédent on utilisait des spikes, objet simple ne prenant aucun paramètre lors de la construction, on a pas eu besoin d'utiliser la balise <params>. Par contre un paquet ipv4 transporte un message. Ce message est "A". Les valeurs "yes" et "ça fonctionne !" n'ont aucun effet sur la construction d'un objet ipv4 vu qu'il ne prend qu'un string en params.

Il faut regarder de plus près le code java.

Packet.java

```java
public class Packet {

    private int size;

    protected Packet(Object ... params){
    }

    //...
```

Chaque classe fille de Packet doit implémenter le constructeur protected.(ça doit être le premier constructeur dans le code pour des raisons d'introspection, même si il y a moyen de faire mieux).

Spike.java

```java
public final class Spike extends Packet {

    public Spike(Object ... params) {
        setSize(1);
    }

}
```

On voit que Spike n'utilise aucun de ses paramètres.

IPv4Datagramme.java

```
1  public final class IPv4Datagramme extends Packet{
2
3          private String message;
4
5          public static final int MESSAGE_INDEX = 0;
6
7          public IPv4Datagramme(Object ... params) {
8          setSize(16);
9          setMessage((String)params[MESSAGE_INDEX]);
10         }
11
12         //...
```

IPv4 utilise un seul paramètre. A noté que les objets récupérés seront de type String.

Troisième partie

Génération de données

0.3 Introduction

Afin de tester certaines propriétés du graphe, il est nécessaire de générer un jeu de donnée conséquent concernant différents caractéristiques du routage.

0.4 Spikes reçus jusqu'alors

Dans cette section, nous allons nous intéressé à la génération du jeu de donnée représentant la quantité de spike reçu par chaque nœud du graphe, de l'instant 0 à l'instant t. Pour ce faire, il faut utiliser le script generateData.py. Ce script s'occupe de lancer cette expérience :

J'ai un graphe de routage type probabilistic flooding model, j'envoie des stimulis à un ou plusieurs moments de la propagation. Ce graphe a une taille et un poids (probabilité de détruire un spike sortant d'un nœud). Je laisse tourner la propagation pendant t secondes.Je récupère la matrice représentant le nombre total de spikes reçus par chaque nœuds jusqu'à l'instant t

On lance cette expérience plusieurs fois (itérations) et selon différents paramètres (taille, poids, programmation des stimulis).

Un exemple d'appel de cette commande :

```
python generateData.py --weigths 0.0 0.1 0.2 0.3 0.4 --times 1 2
--packet_initialisation twoa twob twoab --iterations 100
--tailles_grilles 9 --forcerewrite
```

Ici on génère les données pour les combinaisons de poids temps. On stipule que ce sera de taille 9 mais on aurait pu écrire

```
--taille_grilles 1 2 3 4 5 6 7
```

On veut que l'expérience soit réalisée selon plusieurs programmations (twoa twob et twoab). Les dites programmation doivent se trouver dans le dossier statistiques/stimulis sous la forme nom.stimulis (exemple : twoab.stimulis).

Les données générées se trouvent sous le dossier statistiques/data (avec une arborescence dépendante des paramètres).

A noter que ce script n'est utilisable qu'avec le PFModel pour le moment mais il est facilement extensible à d'autre modèles.

Faire

```
python generateData.py --help
```

pour plus d'informations

Quatrième partie

Additivité

Il est nécessaire de conserver la propriété d'additivité du réseau lors de l'utilisation de nouveaux routages. Pour cela, posons une définition théorique de l'additivité.

0.5 Théorie(à finir)

A partir de maintenant, on considère un graphe G(N,E) avec N l'ensemble de ses nœuds et E l'ensemble de ses arêtes. Chaque nœud possède une file d'attente FIFO. On peut lancer une propagation sur ce graphe.

Definition 1. *(Paquet) (PAS ASSEZ RIGOUREUX)*
Un ***paquet*** $\mathtt{p}$ *est un objet. Il contient un certain nombre d'informations. Il se déplace sur le graphe de nœuds en nœuds au fil du temps (lors de la propagation du graphe). Si le* ***paquet*** *ne contient aucune information, on parle de* ***spike****.*

Definition 2. *(Stimulis)*
Un ***stimulis*** $\mathtt{S}_{p,n,t}$ *est le placement du paquet* $\mathtt{p}$ *dans la file d'attente du nœud n à l'instant t. Si t=0 et si p est un spike, on parle alors d'* ***activation****. Une* ***activation*** $\mathtt{A}_n$ *est le* ***stimulis*** $\mathtt{S}_{s,n,0}$

Definition 3. *(Configuration)*
Une ***configuration*** $\mathtt{C}$ *est un ensemble de stimulis. On parle d'initialisation* $\mathtt{I}$ *si c'est un ensemble d'activation.*

Definition 4. *(Additivité)*
Soit $\mathtt{A}^0, \mathtt{A}^1, .., \mathtt{A}^K$ *des activations où* $\mathtt{A}^i = \mathtt{A}_{n_i} \forall i$.
On pose l'initialisation $\mathtt{I} = \{\mathtt{A}^k \mid n \epsilon [0, K] \text{ et } K > 0\}$
On pose $\mathtt{I}_k = \{\mathtt{A}^k\}\ \forall n \epsilon [0, K]$

Soit $M_{t,n,\mathtt{I}}$ *la variable aléatoire qui représente l'ensemble des paquets reçus par le nœud n depuis le début de la propagation pour chaque nœud du graphe à l'instant t selon l'initialisation* $\mathtt{I}$.

On pose $M_{t,\mathtt{I}} = (M_{t,0,\mathtt{I}}, M_{t,1,\mathtt{I}}, ..., M_{t,card(N),\mathtt{I}})$ *et* $M_{t,\mathtt{I}_i} = (M_{t,0,\mathtt{I}_i}, M_{t,1,\mathtt{I}_i}, ..., M_{t,card(N),\mathtt{I}_i})$

On dit que le routage est ***additif pour l'initialisation*** $\mathtt{I}$ *si* $\forall t$

$$\mathbb{E}(M_{t,\mathtt{I}}) = \sum_{i=1}^{n} (\mathbb{E}(M_{t,\mathtt{I}_i}))$$

On dit qu'un routage est ***additif*** *si il l'est pour n'importe quelle initialisation.*

Note : On pourrait parler d'additivité pour une configuration qui n'utilise que des spikes, et pas forcement une initialisation.

0.6 Utilisation du script

Afin de tester cette propriété d'additivité sur des résultats expérimentaux, nous avons crée un script en Scilab. Ce dernier lit un ensemble de donnée de plusieurs répétition d'expériences afin d'extraire une moyenne de ces données, puis il va calculer à quel point le routage est additif sur ces résultats moyens.

Par exemple

```
scilab -nw -f Additivite.sce -args '9' '1 2'
'0.0 0.1 0.2 0.3 0.4' '5 20 50 75 100'
'twoa twob' twoab 0
```

vérifie l'additivité twoa+twob=twoab sur les combinaisons possibles entre

- tailles : 9
- times : 1 2 (secondes)
- poids : 0.0 0.1 0.2 0.3 0.4
- tailles échantillons : 5 20 50 75 100 (nombre de répétition de l'expérience)

0.7 Résultats

On lance le script avec taille 9 dt 0.1 et time 5s. Sur l'image suivante on remarque qu'a partir d'un certain temps de computation, on a une additivité maximale autour de 0.7. On remarque aussi que l'activité fluctue bien moins quand on augmente le nombre d'expériences. Cette dernière observation pourrait nous amener à la conclusion qu'il faille lancer plusieurs spikes en même temps par activation de neurone (on simulerait en quelque sorte la répétition d'expérience même si on a aucune garantie que le comportement soit similaire. C'est à tester).

{TODO pourquoi c'est meilleurs en 0.7 et pas en 0.0 ?}

En zoomant autour de 0.7 on trouve une valeur minimale de finesse en 0.72. A noter que cette valeur n'est valable que pour cette configuration (taille 9 time 5s dt 0.1)

Cinquième partie

Propagation des spikes

Pour qu'un nouveau model soit viable, en plus d'avoir un comportement additif, il doit avec une répartition voltaïque de type "différence de deux gaussienne". Dans cette partie on va voir comment vérifier cette propriété et si notre modèle "Probabilistic flooding" permet d'obtenir la répartition désirée.

0.8 Utilisation du script

Le script MeanAndVariance.sce permet d'obtenir les statistiques moyennes variances et écart type sur un échantillon d'expérience et ce en utilisant différents paramètres. Un exemple de cette commande :

```
scilab -nw -f MeanAndVariance.sce -args
9 '1 3 5' '0.7' 1000 a_send_20 temp/
```

avec respectivement comme arguments :
— 9 : la taille du réseau (la racine)
— '1 3 5' : les temps de computations
— '0.3 0.7' : les poids
— 1000 :le nombre d'expérience qu'on utilise pour calculer nos statistiques
— a_send_20 : le scénario des expérience (a _send _20.stimulis doit de trouver sous le dossier stimulis/)
— temp/ : le dossier sous lequel les images vont être enregistrer
Cette commande va générer sous le dossier temp un ensemble de graphique représentant les 3 statistiques sur l'ensemble du graphe et sur la diagonale.

0.9 Résultats

On voit sur l'image ci contre que c'est avec un poids de 0.7 et un temps de computation de 3s qu'on obtient une courbe proche de ce que l'on cherche.
Le problème c'est que l'écart type est élevé comme on peut le voir sur cette image.

0.10 Réflexions

Afin de voir si ce modèle est fiable et pas trop aléatoire on se penche sur le calcul d'un intervalle de confiance autour de la moyenne. Le seul problème c'est que la distribution ne suit pas une loi de probabilité connu,comme on peut le voir sur cette image, du coup on pourrait s'orienter vers des méthodes bootstrap.

Sixième partie

Probabilistic flooding et I/E

On rappelle que l'objectif est de trouver un modèle de réseau neural simulant le comportement de DNF tout en consommant moins de ressource. Pour cela on va utiliser le routage probabilitic flooding. Ce routage comme vu précédemment s'avère additif et suit une distribution exponentielle/gaussienne selon certains paramètre. On a donc le candidat idéal pour simuler le routage originel de DNF. On va donc mettre en place de couche de graphes utilisant le routage PF (probabilitic flooding). Une couche avec des signaux excitateurs et une autre avec des signaux inhibiteurs. On applique une transformation affine a ce deux couches puis on soustrait l'une et l'autre. On obtient à la fin la distribution du potentiel des neurones en fonction de leur éloignement aux stimulis. On veut que cette distribution ressemble à la différence de gaussienne du routage de DNF. Pour ça on doit trouver les paramètres des transformations affines ainsi que les poids et les temps de computations des deux graphes. C'est le script InhibiteurExcitateur.sce qui nous permet d'observer l'effet de tel ou tel paramètre sur le comportement voltaïque du réseau de neurones.

0.11 Utilisation du script

Je n'ai pas fais de mode bash sur le script InhibiteurExcitateur.sce (todo ?), donc pour l'utiliser il faudra modifier directement le fichier scilab. Pour cela on a plusieurs paramètres :

```
w_inhib = 0.5;
w_excit = 0.7;
t_inhib = 3;
t_excit = 5;
taille = 19;
aE = 1;
aI = 1;
bE = 0;
bI = 0;
initialisation_packet = 'a_send_20';
maxiteration = 1000;
```

- w_inhib : le poids des nœuds du réseau inhibiteur (RI)
- w_excit : le poids des nœuds du réseau excitateur (RE)
- t_inhib : le temps de computation du RI
- t_excit : le temps de computation du RE
- taille : la taille du réseau de neurone
- aE : coefficient directeur de la transformation affine sur le RE
- aI : coefficient directeur de la transformation affine sur le RI
- bE : base de la transformation affine sur le RE
- bI : base de la transformation affine sur le RI
- initialisation_packet : configuration des stimulis
- maxiteration : nombre de répétition de l'expérience

0.12 Résultats

On veut obtenir une courbe de cette forme. Pour ça on pose comme paramètres :

— w_inhib = 0.5
— w_excit = 0.7
— t_inhib = 3
— t_excit = 5
— aE = 0.1714286
— aI = 0.0428571
— bE = -0.5
— bI = 0.0

Et on récupère une courbe de cette forme. La correspondance nous a semblé suffisante pour des tests en conditions réelles.

0.13 Application sur dnfsim

On va donc utiliser cette combinaison linéaire de Probabilistic flooding graphes afin de simuler DNF. Pour cela, on incorpore les différents graphes au logiciel dnfsim. On run un scénario avec deux bulles à des positions opposées qui tournent. Ça fonctionne très mal, la variance est très élevée, on a rarement le focus d'une bulle. Pour palier à ce problème de variance on s'oriente sur l'envoi de {20,40,60,80} spikes par activation de neurone pour voir ce que ça donne. On part de l'idée qu'envoyer plusieurs spikes en même temps permettrait la diminution de variance, en effet c'est comme si on répétait l'expérience plusieurs fois. Afin de voir si notre hypothèse est plausible, on va faire quelques simulations dans cette configuration.

0.14 Variance pour {20,40,60,80}_asend

On va supposer que notre scénario de départ permet l'envoi de {20,40,60,80} en A (en haut à gauche du réseau). Puis on va observer la répartition des spikes sur la diagonale du réseau (de A à B - en bas à droite -). On en déduit une variance ainsi qu'un intervalle de confiance pour cette variance.

0.14.1 Utilisation du script

Il suffit de run

```
scilab -nw -f MultiBootstrap.sce
```

en ligne de commande. Ce script affiche la variance selon les configurations 20/40/60/80 ainsi que l'intervalle bootstrap à 95

0.14.2 Résultats

Pour commencer sous la contrainte de 5s de computation, on ne voit pas de grosses différences en terme de variance entre les 4 scénarios (20 40 60 80) comme on peut le constater sur cette image. Pour voir vraiment une différence, on voudrait augmenter le temps de computation. Le problème c'est que l'on change totalement les paramètres de départ, du coup on pourrait perdre des propriétés. En l'occurrence, on perd la forme désirée de la courbe (une demi gaussienne). Afin de trouver un moyen de diminuer la variance du modèle, il faut définir quelques contraintes. Plus précisément on doit garder la propriété d'additivité ainsi que la même forme de courbe.

{TODO trouver un set de paramètre qui permettent de diminuer la variance.Si j'arrive pas avec paramètre jouer sur l'activation des neurones. Pourquoi on a ce comportement là (très varié) comment on peut l'améliorer ? Y réfléchir. Quand j'ai trouvé un bon set de paramètre, vérifier si il y a bien additivité.}

Pistes pour diminuer la variance du routage :
- wrapper les maps (essayer d'abord avec un stimulis en milieu de map)
- connexion 8-connexe
- superposer plusieurs maps E et plusieurs map I et faire la moyenne
- rendre certains le transfert des spike aux quatres premiers voisins
- rendre des connexions obsolètes avec une ou deux computations suite au transfert d'un spike

Septième partie

Annexe

Huitième partie

TODO

TODO expliquer les résultats pour avoir une meilleur compréhension des paramètres
TODO g1 additif,g2 additif => ? a*g1+b*g2+c additif =