Récapitulatif des discussions

## Elaboration de la carte de Kohonen

Tout d’abord, le projet a débuté par la mise en place d’une carte auto-organisatrice par l’utilisation d’un algorithme de Kohonen, sous Python.

Le principe de cet algorithme est de donner en entrée un échantillon de stimuli (des vecteurs de dimension n en pratique, nN) à la carte, qui apprendra de manière non-supervisée afin de reproduire au mieux le comportement des données d’entrée.

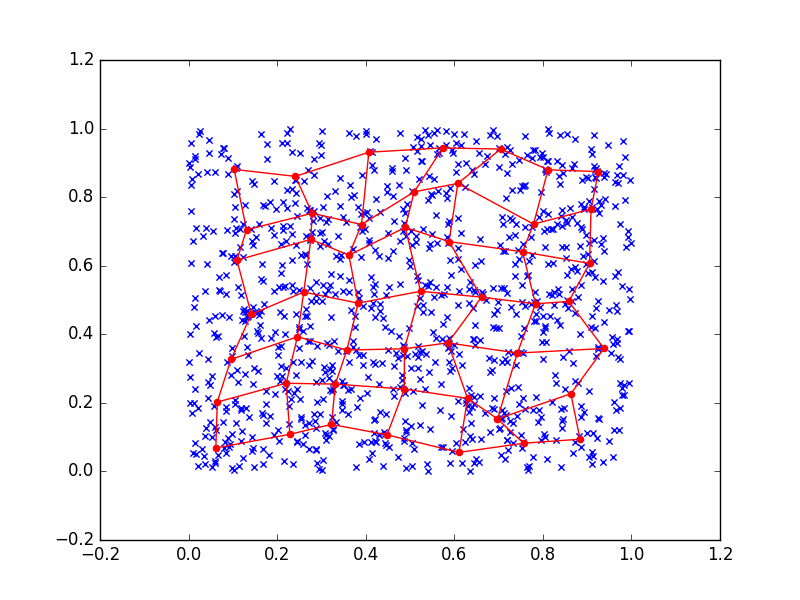
Tout d’abord, les neurones sont initialisés avec des poids aléatoires de la dimension des vecteurs d’entrée. Parmi les stimuli d’entrée, un vecteur est choisi aléatoirement. Ainsi, il est possible de déterminer le vecteur gagnant étant le vecteur le plus proche du vecteur choisi auparavant. Son apprentissage s’effectue selon l’équation suivante :

avec et des fonctions du temps

Par ailleurs, tous les voisins du vecteur gagnant subissent aussi un apprentissage de manière exponentielle (ceux qui sont les plus éloignés sont peu modifiés et le rayon d’apprentissage est exponentiel) tel que :

L’apprentissage s’effectue durant un nombre d’itération de l’ordre de 10 fois le nombre d’entrée en donnée.

Les tests se sont principalement concentrés sur le déploiement d’une grille sur des échantillons pris aléatoirement dans [0,1]. Une interface graphique permettant la visualisation des résultats pas à pas ou avec un plus grand pas ou jusqu’à la dernière itération serait intéressante pour la seconde étape du projet. Pour l’instant nous obtenons le résultat suivant :

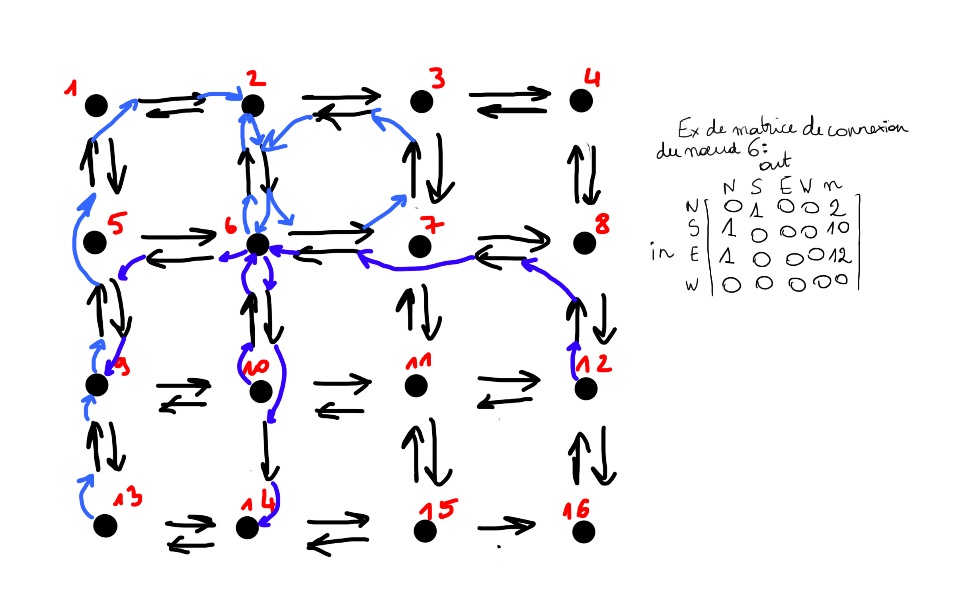
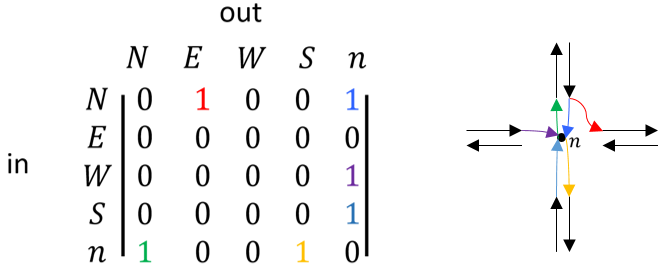


D’autres tests devraient s’effectuer sur d’autres bases de données telles que la base iris (avec 4 dimensions), la base sonar-seijnowski, la base proben. Néanmoins, il serait aussi intéressant d’utiliser cette carte sur de la compression d’image. Cela permettrait une visualisation plus concrète des résultats. L’interface graphique à implanter devrait pouvoir visualiser une imagette échantillon voulue servant à la compression puis la visualisation de l’image complète compressée.

## Le projet en lui-même

Le but est de transformer une SOM de Kohonen en implantant un routage type FPNA avec des nœuds et des liens (opérateurs) de manière distincte, cela pour représenter l Ainsi, cela permettrait de définir soi-même quels neurones seraient reliés entre eux et dans quel sens le message aurait la possibilité de circuler. Ainsi, le réseau est orienté, la distance entre deux neurones n’est pas forcément la même, dépendant de quel neurone est celui de départ ou d’arrivée.

Les neurones pourraient être représentés par une matrice de connexion North, East, West, South et le numéro du neurone relié, représentant les connexions directes entre neurones. (Les flèches bleues représentent les connexions directes entre les opérateurs et les neurones).



out

in

Il faut aussi pouvoir définir une nouvelle distance, comme tous les nœuds ne sont pas reliés ensemble, cela redéfinit la notion de voisinage. A terme, le but serait que la carte se réorganise en définissant elle-même quels liens à rompre ou à créer.

Tout d’abord, les liens seraient fixes et ne changeraient pas, cela permettrait de voir le comportement de la carte avec la nouvelle définition de la distance et des voisins.

La distance seraient calculée par le biais d’un algorithme de calcul du plus court chemin type Dijkstra (en effet, ce qui est intéressant est d’optimiser les liens et le chemin parcourus).

Plusieurs questions sont soulevées grâce à cette nouvelle approche. Comment seraient gérés les cycles ? Quel serait le chemin le plus efficace entre deux chemins de même longueur ? Quelle serait la complexité d’une telle approche ?

## Implémentation d’une nouvelle distance dans la SOMA

Le but est de pouvoir créer une matrice d’adjacence des opérateurs à partir des matrices locales de connexion. L’idée serait de coder chaque opérateur sortant sur 10 bits répartis de la manière suivante : 4 bits pour la colonne, 4 bits pour la ligne du neurone considéré puis 2 bits selon le fait que l’opérateur va au nord (00), est (01), ouest (10) ou sud (11).

Ex : neurone 3, 12 : 1100.0011 sera l’identifiant du neurone

Le lien sortant Nord aura pour identifiant 1100.0011.00

Le lien sortant Est aura pour identifiant 1100.0011.01

Admettons que l’opérateur sortant Nord du neurone en exemple est connecté à l’opérateur entrant Est du neurone 3,12. Néanmoins, ce dernier opérateur est équivalent à l’opérateur sortant Ouest du neurone 3,13. Ainsi, pour remplir la matrice d’adjacence il suffit de remplir d’un 1 à la colonne 1100.0011.00 et la ligne 1101.0011.01 (les lignes correspondant aux opérateurs recevant l’information et les colonnes aux opérateurs émettant l’information)

On étudie des matrices avec un nombre de neurone étant une puissance de 2.

La matrice d’adjacence est calculée à travers l’algorithme suivant :

**indice(i,j,Card)=Card+4\*j+64\*I (Card {N:0,E:1,W:2,S:3,n:8})**

**n représente le nombre de neurone sur une ligne ou colonne du réseau**

**ꓯi [0,n], adji est la matrice de connexion locale du neurone i**

**On appelle A la matrice d’adjacence.**

**A[n\*n\*8][n\*n\*8] est initialisée à 0**

**Pour i de 0 à n\*n\*8**

**Pour j de 0 à n\*n\*8**

**Pour k dans N,E,W,S**

**Switch(k) :**

**N : indi = indice(i-1,j,S)**

**Pour h dans N,E,W,S:**

**Si ajdi(i,j)[k,h] = 1**

**Alors A[indi][indice(i,j,h)]=1**

**E : indi = indice(i,j-1,W)**

**Pour h dans N,E,W,S:**

**Si ajdi(i,j)[k,h] = 1**

**Alors A[indi][indice(i,j,h)]=1**

**W : indi = indice(i,j+1,E)**

**Pour h dans N,E,W,S:**

**Si ajdi(i,j)[k,h] = 1**

**Alors A[indi][indice(i,j,h)]=1**

**S : indi = indice(i+1,j,N)**

**Pour h dans N,E,W,S:**

**Si ajdi(i,j)[k,h] = 1**

**Alors A[indi][indice(i,j,h)]=1**

Une fois la matrice d’adjacence calculée, la matrice de distance est calculée de la manière suivante :

**dist[n²][n²] est initialisée à -1**

**P 🡨 M**

**Pour p de 1 à n²**

**P 🡨 P\*M**

**Pour i0 de 1 à n**

**Pour j0 de 1 à n**

**Pour i1 de 1 à n**

**Pour j1 de 1 à n**

**Si dist[indice(i0 ,j0,n)][indice(i1 ,j1,n)] = -1 et**

**P[indice(i0 ,j0,n)][indice(i1 ,j1,n)] ≠ 0**

**Alors dist[indice(i0 ,j0,n)][indice(i1 ,j1,n)] = p**

Des tests doivent être effectués pour vérifier si l’algorithme permet bien de trouver les distances au travers de configuration « modèle » telle que la configuration type Kohonen : si les résultats obtenus sont identiques à la carte de Kohonen avec une distance L1 cela signifie que le calcul de distance devrait être le bon. Par ailleurs, un autre test intéressant serait d’imposer un chemin dans les connexions des neurones

## Ce qu’il reste à faire

* D’autres tests moins communs de la SOM
* Créer une interface graphique pas à pas pour l’exemple dans [0,1]
* Réflexions sur la manière dont on pourrait calculer la distance entre 2 neurones dans la « nouvelle carte auto-organisatrice »
* Prog python : optimisation
* Réglage paramètres (compression d’images)
* Programmer calcul distance via routage FPNA