Cartes de Kohonen:
Apprentissage auto-suppervisé

Définition du problème:
Le problème que nous adressons ici, concerne l'apprentissage *auto-organisé*: comment un système plastique s'auto modifie pour s'adapter aux données présentées en entrée. Un tel type de système est très souvent illustré par les réseaux de neurones dits de **Kohonen**. Ce type de réseau apprend à minimiser la distance entre ses poids et les entrées, sans qu'aucune consigne sur la solution à priori ne soit donnée au système. Dans ce TP, nous illustrerons ce cas d'apprentissage **non supervisé** par la programmation en langage C d'un réseau de Kohonen.

La carte de Kohonen:

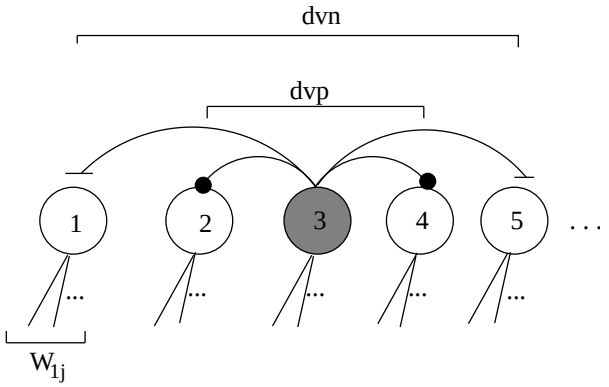

FIG. 1 – *Représentation d'un réseau de neurone Kohonen 1D. Une notion de topologie est garantie par les interconnections excitatrices ou inhibitrices latérales. Dvp est la distance au voisinage positif, c'est a dire la distance max des connections excitatrices. Dvn est la distance au voisinage négatif, c'est a dire la distance max des connections inhibitrices. Les Entrées du réseau sont apprises par les connections montantes (W_{ij}).*

Algorithme :

Soit E l'ensemble des vecteurs d'entrée e_1, e_2, ...,e_m de dimension n. Soit ϵ une constante d'apprentissage.

- Choisir aléatoirement un vecteur d'entrée e_j dans E.

- Présenter e_j à chaque neurone i de la carte dont on calcule le potentiel (distance des poids aux entrées).

$$Pot^i = ||e_j, W_{i.}|| \tag{1}$$

- Recherche du neurone gagnant: neurone dont l'activité est maximum sur toute la carte:

$$Act^i = \frac{1}{1 + Pot^i} \tag{2}$$

$$Winner = argmax_i(Act^i) \tag{3}$$

– Mise à jour des poids du neurone gagnant ainsi que des neurones voisins:

$$W_{ij} = W_{ij} + (\epsilon \cdot |e_j - w_{ij}| \cdot \phi(winner,i)) \quad (4)$$

avec $\phi(winner,i)$ un coefficient associé au voisinage du neurone avec le gagnant.

Conventions:
Dans ce TP, nous supposerons le réseau de neurones de taille fixe: 20 neurones. Nous supposerons que l'ensemble des données est composées de 20 vecteurs de deux dimensions. Chaque neurone possède par conséquence deux poids en entrée. Dvp = 1 et dvn =2.

Gestion des données:

– Identifier la structure de données nécessaire à la représentation d'un vecteur d'entrée, que l'on appellera Data. Créer un tableau de 20 Datas que l'on appellera DataSet.

– Ecrire la procédure InitialiseSet () qui initialise les vecteurs d'entrée par des valeurs appartenant à [0,200].

– Ecrire la fonction Data SortData() qui retourne un vecteur d'entré tiré aléatoirement dans DataSet (attention, on veillera à ce que qu'un donnée déjà sélectionnée ne puisse être de nouveau tirée).

apprentissage:

– Identifier la structure de données nécessaire à la représentation d'un neurone (W_ij y compris), on l'appellera Neuron, créer un tableau de 20 neurones que l'on appellera NeuronSet.

– Ecrire la fonction qui calcule le potentiel de tous les neurones de la carte en fonction d'un vecteur d'entrée donné (eq.1).

– Ecrire la fonction qui calcule l'activité de tous les neurones de la carte (eq.2).

– Ecrire la fonction qui retourne l'indice du neurone gagnant (eq.3).

– Ecrire la fonction de mise à jour des poids (la distance des neurones est calculée grâce à leur indice dans NeuronSet) (eq.4).

– Tester l'apprentissage du réseau de neurones.

facultatif:

– Etendre l'IHM du programme grâce aux librairies graphiques fournies.

– Faire varier ϵ au cours du temps.

– Considérer une carte toroide.