GOUSSEAUD Gaëtan 13402498 L3 CDVA
Email: gousseaud.gaetan@gmail.com 2016/2017

Rapport SOM sur Iris data:

Compilation:
gcc alg.c -lm -Wall

Utilisation:
./a.out database.txt random_fork(float()) alpha_init(phase1) starting_neigh(phase2) random_seed iteration_ratio_divider

-**database.txt** : your file with your database inside , this version only supports Iris-data
-**random_fork(float())**: random range for -itself +itself used for neuron's values initialization
-**alpha_init(phase1)**: alpha initialization for phase 1 , phase 2 will have 10 percent of it
-**starting_neigh(phase2)**: value representing the neighbouring_arc at the beginning of phase 2
-**random_seed**: seed used for srand48
-**iteration_ratio_divider**: divider used for phase 1 / phase 2 proportions such as phase 1 : 1/divider & phase 2 : (1/divider)*(divider-1)\n\n");

Format de la base de donnée:

- 150 vecteurs de taille 4 float/double
- 3 type d'étiquettes
- 50 vecteurs de chaque type.

Sortie du programme:

Une carte rectangulaire(Choisie dynamiquement suivant le nombre d'or utilisant un parcours de diviseurs en tenaille) 7*10 de neurones étiquetés . Mon implémentation permets des modifications simples afin de l'adapter à d'autre base de données en float/double voir même int mais malheureusement qui doivent se faire en interne.

Initialiser avec une fourchette paramétrable mais recommandée de 0.005
[vec_moy - 0.005, vec_moy + 0.005]

L'apprentissage se faire sur **500 * nb_vec itération** mais **peux se changer internalement** et fonctionnera comme prévu. De même pour **son nombre de phase** qui **peut être augmenté** en copiant et en collant la fonction d'apprentissage en changeant quelques paramètres dans le main mais encore une fois **internalement**. Je peux vous envoyer une version avec plus de phase si vous le souhaitez.
La fonction d'apprentissage est tel:
wij(t+1) = wij(t) +(alpha * (xi - wij))

J'ai choisis d'avoir **plusieurs fonctions** d'apprentissage **pour différencier** le choix du **rayon de voisinage** mais il pourrait se faire facilement dans une fonction avec un paramètre de type de fonction.

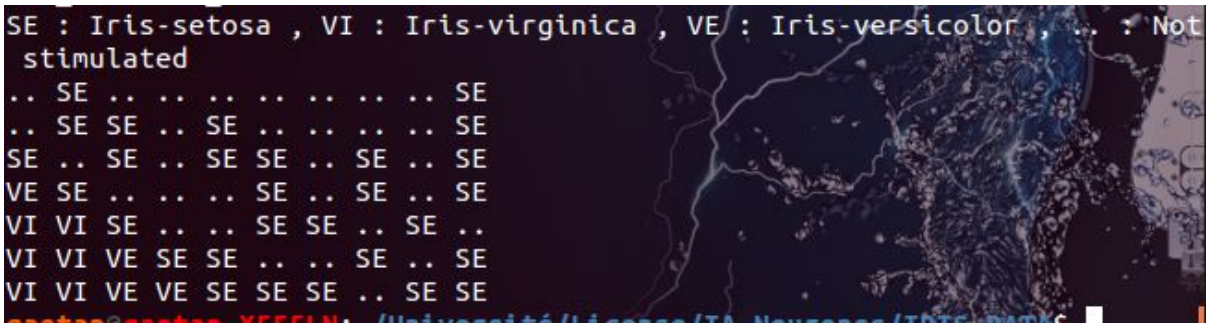
Le **voisinage** décrémente dynamiquement (ex: pour la phase 1 si il y a 20000 itération et que sa valeur optimale est de 4 alors il décrémentera toutes les 20000/4 itérations), de même que pour le reste, **il peut se changer** facilement **de façon interne** pour la première phase (j'ai laissé une valeur **optimale** trouvée **dynamiquement**). Un voisinage de 1 correspond au bmu et ses 8 voisins..

Le programme est commenté et codé de façon clair et lisible en anglais.

Quelques cartes avec les paramètres suivant:
random_fork : 0.005 , **alpha_init** : 0.7 , **rayon(phase2)** : 1 , **random_seed** : 1, **divider**: 4

random_fork : 0.005 , **alpha_init** : 0.7 , **rayon(phase2)** : 2 , **random_seed** : 1, **divider**: 4

random_fork : 0.005 , **alpha_init** : 0.8 , **rayon(phase2)** : 1 , **random_seed** : 2, **divider**: 4



random_fork : 0.005 , **alpha_init** : 0.8 , **rayon(phase2)** : 1 , **random_seed** : 2, **divider**: 3

Exemple de ligne pour avoir cette carte: