Fouad Badran, Cécile Mallet, Awa Niang, Charles Sorror, Sylvie Thiria

TPC01 : Cartes topologiques (CT) : Données simulées et reconnaissance de chiffres manuscrits

I - Les Objectifs

Les 2 parties de ce TP ont pour but de commencer à familiariser l'auditeur, avec le maniement des cartes topologiques et leur exploitation.

Partie 1 : Dans cette 1ère partie, nous travaillerons avec des données simulées qui représentent soit la lettre **Z** soit la lettre **F**. Nous chercherons à trouver une taille de carte topologique optimum d'abord en 1D puis en 2D. Dans le cas 2D nous complèterons les résultats avec classification par labellisation.

Partie 2 : Mise en œuvre des cartes topologiques pour la classification de chiffres manuscrits pour lesquels différents codages seront utilisés. On s'intéressera à trouver des paramètres d'apprentissage optimum et à montrer une représentation interne de la carte.

==========

Le rapport de TP devra être synthétique. Il doit montrer la démarche suivie, et ne faire apparaître que les résultats nécessaires. Il s'agit de quantifier les résultats tout en rédigeant un rapport qui les analyse et les commente. Les paramètres utilisés devront être indiqués, Les graphiques des expériences doivent être insérés dans le rapport. Les résultats présentés devront être analysés et commentés.

II - Les Données

•) Pour la 1ère partie du TP, les fonctions **Zcreadata.m** et **Fcreatdata.m**, mises à votre disposition servent à la génération des données simulées en 2D qui représentent respectivement la lettre Z et la lettre F, dont les points sont répartis en 3 classes. Deux paramètres sont utilisés par ces fonctions :

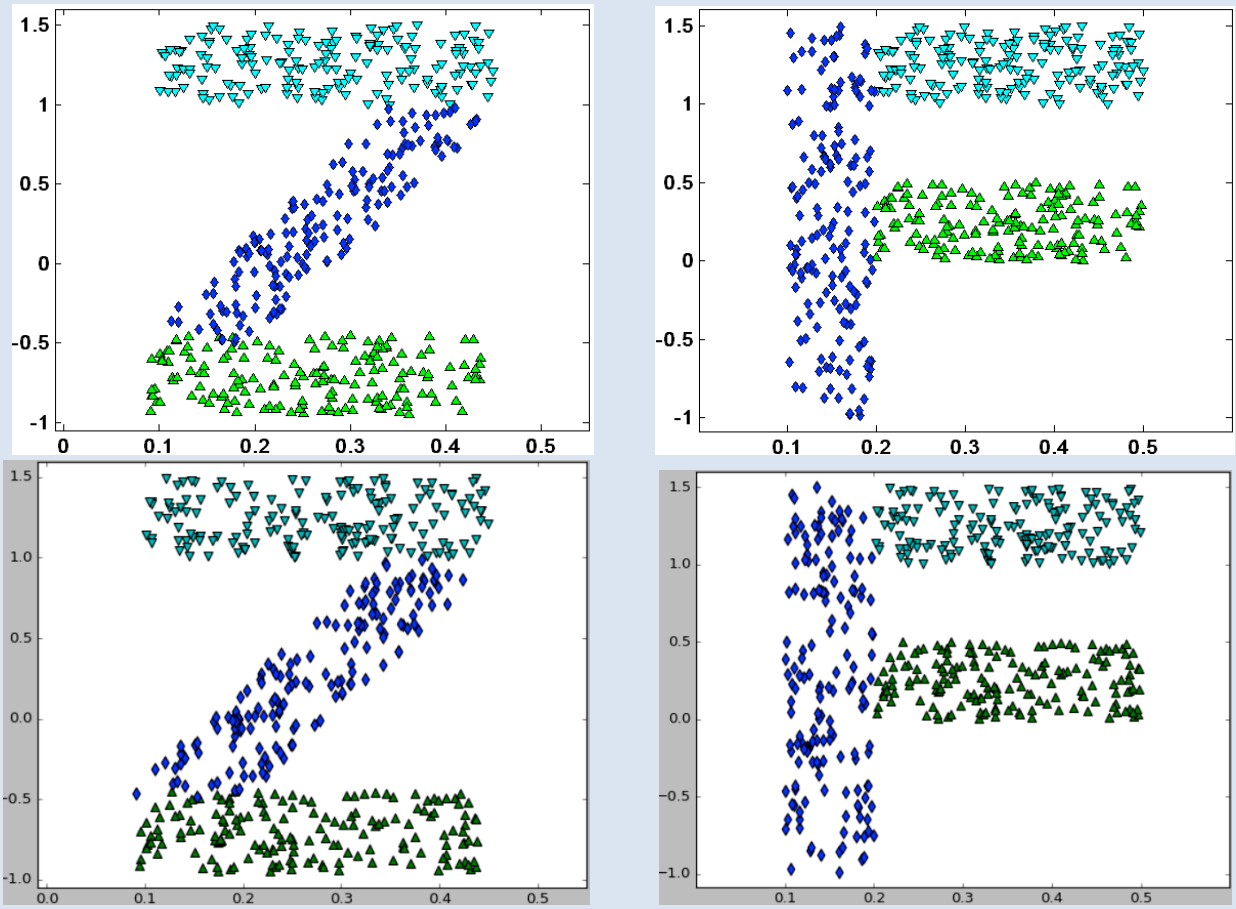
- N : pour indiquer le nombre total de données.
- classnames : (optionnel) pour renseigner le nom des classes qui serviront à la labellisation des données (valeur par défaut : {'A', 'B', 'C'}).

En retour on obtient :

- X : l'ensemble des données.
- labs : les labels de classe associés aux données.
- cnames = {'X', 'Y'} : les noms des composantes.

...

La fonction **lettreplot.m** nous permet de visualiser les données tirées, en représentant, par des formes différentes, les points des différentes classes :



Les 3 classes sont représentées par des points de formes différentes qui sont placées sur chacune des barres qui forment la lettre :

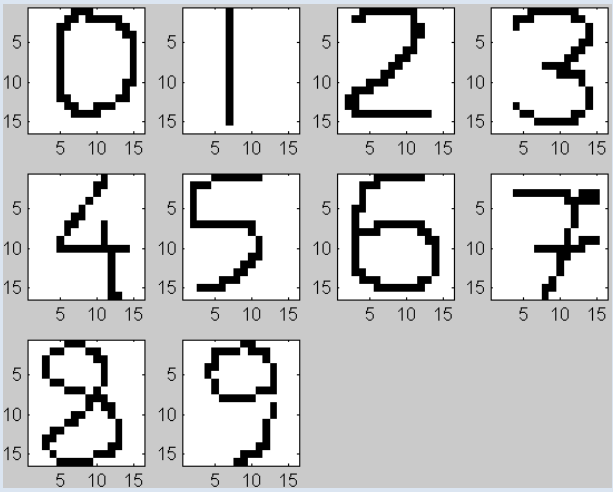
pour le Z
- classe « top » : triangles dirigés vers le bas,
- classe « bottom » : triangles dirigés vers le haut
- classe « diag » : losanges

pour le F
- classe « top » : triangles dirigés vers le bas,
- classe « middle » : triangles dirigés vers le haut
- classe « left » : losanges

•) Pour la 2ème partie du TP nous disposons du fichier **x.txt** qui est la base de données de chiffres manuscrits. Elle est composée de 480 chiffres codés en binaire (±1), dans une matrice 256x480. Cela signifie que chaque image binaire 16x16 a été transformée en un vecteur de dimension 256 qui, à son tour, correspond à une colonne de la matrice du fichier **x.txt**. Dans ce fichier, les pixels sont codés par les valeurs -1 et +1.

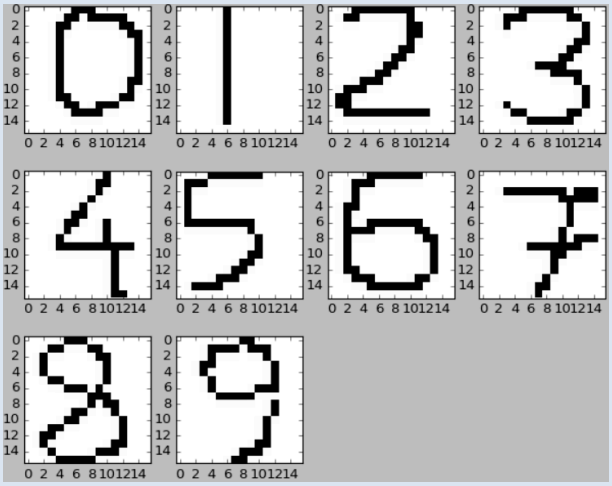
Le programme **display_pat.m** permet la visualisation de cette base de données. Par exemple, pour voir les 10 premières formes :

>> load x.txt
>> display_pat(x,1,10)



Chiffres codés par une grille de 16x16 pixels :
-1 : pixel blanc,
+1 : pixel noir

En python3 :
x = np.loadtxt("x.txt");
TPC01_methodes.display_pat(x,1,10);



Ces données ont par ailleurs été codées selon les différentes conventions suivantes :

HX : Histogramme des projections du chiffre sur l'axe des x : dans chaque colonne on calcule le nombre de pixels noir – le nombre de pixels blancs. HX conduit à un vecteur de 16 composantes.

HY : Histogramme des projections du chiffre sur l'axe des y : dans chaque ligne on calcule le nombre de pixels noir – le nombre de pixels blancs. HY conduit aussi à un vecteur de 16 composantes.

PH : Profil Haut – pour chaque colonne, on code la coordonnée de la première transition blanc/noir en partant du haut. PH est un vecteur de 16 composantes.

PB : Profil Bas – pour chaque colonne, on code la coordonnée de la première transition blanc/noir en partant du bas. PB est un vecteur de 16 composantes.

PG : Profil Gauche – pour chaque ligne, on code la coordonnée de la première transition blanc/noir en partant de la gauche. PG est un vecteur de 16 composantes.

PD : Profil Droit – pour chaque ligne, on code la coordonnée de la première transition blanc/noir en partant de la droite. PD est un vecteur de 16 composantes.

(Il est à noter que les coordonnées sont indicées de la gauche vers la droite en ligne et de haut en bas pour les colonnes)

Ces conventions de codage peuvent être combinées pour former différents fichiers d'apprentissage. Nous disposons des cas suivants :
1 : codage HX seul ; vecteur de 16 composantes. Fichier d'entrée : **hx.txt**
2 : codage HX,HY ; vecteur de 32 composantes. Fichier d'entrée : **hx_hy.txt**
3 : codage PG,PD ; vecteur de 32 composantes. Fichier d'entrée : **pg_pd.txt**
4 : codage HX,HY,PG,PD ; vecteur de 64 composantes. Fichier d'entrée : **hx_hy_pg_pd.txt**
5 : codage PB,PH ; vecteur de 32 composantes. Fichier d'entrée : **pb_ph.txt**
6 : codage HX,HY,PB,PH ; vecteur de 64 composantes. Fichier d'entrée : **hx_hy_pb_ph.txt**
On précise que toutes les données de ces fichiers ont été de surcroît « normalisées » dans l'intervalle [-1, 1], SAUF **hx.txt**

III - Eléments pour le déroulement du TP

Pour réaliser ce TP, nous utiliserons la librairie Matlab « SOM_Toolbox version 2.0beta », qui implémente le logiciel d'apprentissage par cartes topologiques, développé par l'équipe de Kohonen. C'est un produit du domaine public que vous pouvez par ailleurs trouver sur le site Web de l'Université de Helsinky à l'adresse http://www.cis.hut.fi/research/software .

Principales fonctions de la Som_toolbox :

- **som_data_struct** : Création d'une structure de donnée qui, en plus des données elles-mêmes, comporte, entre autre, un champ de labellisation. Passer par ce type de structure, ce qui n'est pas toujours nécessaire, permet l'utilisation de certaines fonctions comme, par exemple, celles qui se rapportent à la labellisation.
- **som_map_struct** : crée une structure de carte topologique. On y trouve entre autre sa taille, les vecteurs référents, sa topologie (taille, connexion, forme), des informations sur les paramètres d'apprentissage par étape (algorithme, température initiale et finale, nombre d'itération, ...). On distingue éventuellement une étape d'initialisation, et 2 étapes d'apprentissage la seconde étant un affinement de la 1ère utilise normalement une température plus faible.
- **som_lininit** : Crée (ou modifie) une CT avec une initialisation linéaire des vecteurs référents (il y a aussi **som_randinit** qui le fait de façon aléatoire)
- **som_grid** : Visualisation de la grille de la carte topologique qui peut aussi être projeté dans l'espace de coordonnées des référents.
- **som_batchtrain** : Algorithme batch d'apprentissage de la CT, et **som_seqtrain** pour la version séquentielle.
- **som_quality** : calcule et renvoie l'erreur de quantifiquation (distance moyenne des données à leurs référents) et l'erreur topographique (proportion des données dont les 2 référents les plus proches ne sont pas adjacents sur la carte)
  Avec sompy-python 3 : l'erreur de quantification est donnée à la fin du run. L'erreur topographique (dans le cas d'un maillage rectangulaire uniquement) peut être obtenue, par exemple, et à l'aide du module triedctk as ctk de la façon suivante :

```
bmus2 = ctk.mbmus(sMap, Data=Xapp, narg=2);
TE    = ctk.errtopo(sMap, bmus2);
print("final topologique error  = %.4f" %TE)
```

- **som_label** : Permet de mettre à jour les labels des données d'une structure de données ou des référents d'une structure de CT.

- **som_autolabel :** Mise à jour automatique des labels selon un mode spécifié (fréquence ou vote majoritaire par exemple).
- **som_hits :** Renvoie un vecteur qui contient les nombres de données d'un ensemble associées à chaque neurone.
- **som_cplane :** visualisation d'une carte topologique (2D) à laquelle on peut associer différentes couleurs aux référents.
- **som_show** : Affichage de la matrice **U** et de la CT variable par variable.

...

Fonctions pour la 1ère partie :

► Fonctions de la Som_Toolbox :

som_data_struct, som_map_struct , som_lininit, som_grid, som_batchtrain, som_quality, som_label, som_autolabel, som_hits, som_cplane, som_show

► Fonctions ad hoc :

app_lettre : Exemple de script pour l'apprentissage de données en forme de lettre (Z ou F) avec une carte topologique.
Zcreadata et Fcreatata : Génération des données en forme de **Z** pour l'un et de **F** pour l'autre.
lettreplot : Affichage des données
classifperf : Performance de classification avec labellisation par vote majoritaire
ctk_confus : Etablit la matrice de confusion pour une carte topologique et des données, les deux devant être déjà labellisées.
ctk_label2num : Traduit les labels en chiffres, comme **som_label2num**, mais en faisant correspondre la numérotation avec l'ordre des noms de label passés.
ctk_showindice.m : Fait apparaitre les indices des référents sur une carte topologique dans l'espace des données.
ctclassif_Umat : Exemple de script pour la présentation :
- d'une carte topologique (labellisée par vote majoritaire) avec :
  - une taille des neurones proportionnelle au nombre d'éléments qu'ils captent (hits)
  - les indices des neurones
  - le décompte, par neurone, des données qu'il capte par label
  - l'affectation du label de classe des neurones selon le vote majoritaire
  - une couleur des neurones différente selon leur label (ou classe)
- et de la matrice U

Fonctions pour la 2ème partie :

► Fonctions de la Som_Toolbox :

som_data_struct, som_label, som_lininit, som_batchtrain, som_autolabel, som_cplane, som_grid, som_hits, som_bmus, som_map_struct, som_show

► Fonctions ad hoc :

set_sdata : Création des structures de données avec labellisation pour l'apprentissage et le test.
app_chiffres : Exemple de script de définition et d'apprentissage d'une carte topologique.
classifperf, ctk_label2num, **ctk_confus :** Voir la liste de la 1ère partie où ces fonctions sont déjà énumérées.
show_refpat : Visualisation de chiffres manuscrits de la base des donnée brute par référent (c'est à dire) sur la carte topologique.
display_pat : Visualisation de chiffres manuscrits de la base des donnée brute.

...

show_refactiv : Visualisation des niveaux d'activation des neurones d'une carte topologique en réaction à la présentation de différentes formes en entrée de la carte. Cette fonction utilise la fonction **euclidist** qui permet de calculer les distances (Delta) entre les vecteurs de 2 matrices ; en l'occurrence, celle des référents et une matrice de données. Pour une donnée d'entrée, l'activation d'un neurone doit être d'autant plus élevée que son référent est proche de la donnée. Cela est visuellement présenté par la fonction **show_refactiv** en multipliant les distances par -1.
Remarque : une autre possibilité consiste à prendre l'exponentielle comme fonction d'activation :
s = exp(-lambda*Delta) où lambda est un paramètre à choisir.

Sachez que toutes les fonctions de la SOM_Toolbox sont documentées et qu'elles sont par ailleurs recensées dans le document « Contents ». Les scripts de démonstration (**som_demo1, 2, 3, 4**) sont un bon point de départ pour découvrir les possibilités de la SOM_Toolbox. Qu'il s'agisse des fonctions de la SOM_Toolbox ou des fonctions ad hoc vous pouvez vous aider des commandes **help** ou **type** pour obtenir des informations les concernant.

Préambule

La réalisation d'une carte topologique fait intervenir un grand nombre de méta paramètres qui en complique la mise en œuvre. Ces métas paramètres portent aussi bien sur l'architecture de la carte que sur l'algorithme d'apprentissage.

Pour simplifier l'approche des CT, nous avons fait le choix d'une certaine configuration pour l'ensemble de nos TP. Rien n'empêchera bien sur le lecteur curieux d'en essayer d'autres. En particulier, nous avons toujours retenu des CT à connexions hexagonales, de forme rectangulaire à voisinage gaussien et initialisées de façon linéaire. Par ailleurs nous utiliserons deux étapes d'entrainement de la CT pour lesquelles nous avons retenu la version batch de l'algorithme d'apprentissage. La première étape utilise (en principe) une température initiale plus élevée permettant ainsi la prise en compte d'un voisinage élargi. La seconde étape, avec des températures plus petites permet un raffinement plus local de la quantification vectorielle.

La configuration que nous venons de décrire correspond d'ailleurs à celle proposée par défaut par la fonction d'apprentissage automatique (som_make) de la SOM_Toolbox.
Les principaux paramètres sur lesquels nous jouerons, selon les TP, sont la taille de la carte, les nombres d'itérations d'apprentissage et les températures initiales et finales.

Nous venons d'évoquer un premier niveau de difficulté que sont les aspects algorithmiques et techniques nécessaires à notre domaine d'expertise. Un second niveau concerne le choix, la pertinence des représentations utilisées pour l'étude d'un problème mais également et surtout notre capacité à les expliquer et à les interpréter. C'est souvent sur ce point de compétence que doit se concentrer l'attention et que l'effort doit être porté.

IV - Compléments ou rappels de cours

Matrice de confusion

La matrice de confusion permet de comparer les résultats de la classification faite par un modèle (classe estimée) par rapport à la bonne classification (classe réelle).

Les lignes i de la matrice correspondent aux classes réelles, et les colonnes j aux classes estimées. L'élément M(i,j) de la matrice indique le nombre d'individus appartenant à la classe Ci que le modèle classe dans la classe Cj. On peut tirer plusieurs pourcentages de cette matrice qui permettent d'apprécier la qualité de classification faite par le modèle.

Considérons par exemple la matrice de confusion à 3 classes suivante :

		Classes estimées C1	C2	C3		
	C1	30			2	1
Classes réelles	C2	1	29	4		
	C3	2	0	31		

Celle-ci nous indique par exemple que 3 éléments qui appartiennent à la classe C1 sont classés à tord par le classifieur dans la classe C2 pour 2 d'entre eux et dans la classe C3 pour le 3ème. On peut calculer des pourcentages en ligne et en colonne. Pour la 1ère ligne, 30/33=90.91% d'élément de la classe C1 sont correctement classés, pour la 2ème colonne, 29/31=93.55% des éléments classés en C2 le sont avec raison. Les termes de la diagonale de la matrice indiquent les nombres d'éléments bien classés. Si l'on fait la somme des termes en ligne ou en colonne, on trouve qu'il y a 100 données. On peut calculer une performance globale de bonne classification en rapportant le nombre des éléments de la trace (diagonale de la matrice) à celui des données : (30+29+31)/100 = 90%.

V - 1ère Partie : Données simulées en forme de lettre Z et F

L'un des choix important dans la définition d'une CT est celui du nombre de neurones. C'est plus particulièrement ce paramètre que nous voulons mettre à l'épreuve dans l'exercice qui suit. Pour cela nous nous appuierons sur 2 jeux de données simulées qui représentent les lettres **Z** et **F**. On utilisera pour chaque lettre un seul ensemble d'apprentissage (Xapp) de 500 points et on se limitera dans tous les cas à des cartes ne dépassant pas 50 neurones maximum (soit 10 formes en moyenne par neurone).

Pour ce travail, vous devez ouvrir le script **app_lettre.m** et l'étudier. Utiliser le ensuite pour

- Positionner la taille de l'ensemble d'apprentissage (**Napp=500**)
- Instancier la fonction la **Zcreatdata.m** ou **Fcreadata.m** selon la lettre sur laquelle vous souhaitez travailler
- Choisir la dimension de la carte topologique (variables **nlmap** et **ncmap** qui correspondent aux nombre de lignes et de colonnes de la carte).

Remarque : dans le script **app_lettre.m**, nous avons retenu les paramètres d'apprentissage suivants :

	nombre d'itérations	Température initiale	Température finale
1ère étape	20	5	1.25
2ème étape	50	1.25	0.10

Pour éviter de démultiplier les nombres d'expériences, nous vous proposons de garder toujours ces mêmes paramètres (ce qui n'est pas une obligation).

En Python3 les définitions spécifiques au TP sont dans le module TPC01_methodes.
Une définition app_lettre s'y trouve qui pourra être appelée à partir de votre propre code en récupérant les variables de sortie qui permettront de poursuivre le TP.

Travail à faire

1°) Travail sur des cartes de dimension 1.

Pour chacune des 2 lettres :

- Essayer de trouver une carte ayant un nombre de neurones minimal (≤50) et ayant une performance en classification optimale. Pour le calcul de la performance, vous pouvez utiliser la fonction **classifperf.m** à lancer après **app_lettre.m** (ou l'intégrer dans app_lettre).
  (En python3 la perf peut être obtenue en utilisant ~~TPC01_methodes.confus~~ ctk.classifperf)
- Pour la carte que vous aurez retenue, indiquer les erreurs de quantification et topographique (fonction à utiliser : **som_quality.m**) et représenter : (py3 : QE est donnée à la fin du run, pour TE : bmus2=ctk.mbmus(sMap, Data=Xapp, narg=2); TE= ctk.errtopo(sMap, bmus2);)
  - L'ensemble des données différenciées selon leur label et la carte en projection. Pour cela vous pouvez utiliser la fonction **lettreplot.m** suivie de la fonction **som_grid.m**. Exemple :

```
figure;  hold on;
lettreplot(Xapp);
som_grid(sMap,'Coord',sMap.codebook,'marker','o','markercolor','r',…
                'markersize',4, 'linecolor','k', 'linewidth',1.5);
ctk_showindice(sMap);    % Pour faire apparaitre les indices des référents
```

version python 3 :

```
plt.figure();
TPC01_methodes.lettreplot(Xapp);
ctk.showmapping(sMap, Xapp, bmus=[], seecellid=1, subp=False, override=True);
```

  - Les histogrammes en x et en y des données et des référents (fonction à utiliser : plt.**hist.m**).

…

2°) Travail sur des cartes de dimension 2.

On veut voir ce qu'apporte le passage à une carte 2D (toujours sous la contrainte d'un nombre maximum de 50 neurones). Reprendre, dans ce contexte et pour chaque lettre, les mêmes demandes que celles formulées au point 1°.

- Pour la carte que vous aurez retenue, vous devrez présenter de plus :
  - La matrice de confusion : utilisation de la fonction **ctk_confus.m** qui nécessite, avant d'être appelée, que les labels des données et des référents soient renseignés dans leurs structures respectives. Exemple :

```
sMap = som_label(sMap, 'clear','all');          % raz
sMap = som_autolabel(sMap, sDapp, 'vote');      % vote majoritaire
MC   = ctk_confus(sMap, sDapp, classnames,1);   % matrice de confusion
```

    Où sMap est la structure de la carte topologique dont les labels ont déjà été renseignés par **app_lettre**, sData est la structure pour les données, et classnames contient les noms des classes.
    En python3 la matrice de confusion ~~est sensée avoir été déjà établie (c.f. plus haut)~~ peut être obtenue en utilisant TPC01_methodes.confus
  - La carte topologique avec les nombres d'éléments captés par classe pour chaque neurone et la comparer à la U-matrice. Pour obtenir les figures nécessaires, vous pouvez prendre exemple sur le script **ctclassif_Umat.m**. ou vous en servir directement.
    En python3 dans TPC01_methodes cette fonction existe aussi, mais il faut lui passer des paramètres.

V - 2ème Partie : Reconnaissance de chiffres manuscrits

A la différence de la 1ère partie, les données utilisées pour la reconnaissance de chiffres sont des données réelles et non pas simulées. Par ailleurs, les données dont on dispose devront être partitionnées entre un ensemble d'apprentissage et un ensemble de test. Pour ne pas trop démultiplier les expériences on vous propose d'utiliser une carte de 12x12 neurones et de garder les autres paramètres de la carte par défaut. Par contre vous devrez choisir vous-même les nombres d'itérations et les températures. A titre indicatif, dans certains cas, quelques dizaines d'itérations peuvent suffire, et en tout état de cause, il ne devrait pas être nécessaire d'aller au delà d'une centaine d'itérations, Les températures quant à elles doivent pouvoir se situer entre 20 pour la plus élevée et 0.10 pour la plus basse. Le script **app_chiffres.m** est un exemple de définition de la carte topologique et de son apprentissage en 2 étapes.

Travail à faire

1°) Vous devez comparer les résultats d'apprentissage obtenus selon les différents fichiers de données suivants : **x.txt, hx_hy.txt, pg_pd.txt et hx_hy_pg_pd.txt**.
Les 340 premiers exemples devront servir à l'apprentissage (Napp=340), et les derniers chiffres restant constitueront l'ensemble de test. Le script **set_sdata.m** peut être utilisé pour charger les données et construire les structures nécessaires à l'apprentissage et au test, labellisation incluse.
Les résultats obtenus pourront s'apprécier en présentant :

- les performances sur les ensembles d'apprentissage et de test (fonction **classifperf**) ce qui demande d'avoir effectué au préalable une labellisation des référents par vote majoritaire. Fonctions à utiliser : **som_label** avec le paramètre '**clear**' et **som_autolabel** avec le paramètre '**vote**'.
  En python3, ~~j'ai procédé différemment (why ?) en utilisant la matrice de confusion :~~
  ~~Tfreq,Ulab = ctk.reflabfreq(sMap,Xapp,Xapplabels); # fréquence des labels par référents~~
  ~~CBlabmaj   = ctk.cblabvmaj(Tfreq,Ulab); # labellisation des référents par vote majoritaire~~
  ~~Xappbmus = ctk.mbmus(sMap, Data=Xapp, narg=1); # bmus de l'ensemble d'apprentissage~~
  ~~MCapp, Perfapp = ctk.confus(sMap,Xapp,Xapplabels,classnames,~~
  ~~CBlabels=CBlabmaj,Databmus=Xappbmus,visu=0);~~

  ```
  Perfapp = ctk.classifperf(sMap, Xapp, Xapplabels)
  Perftest= ctk.classifperf(sMap, Xapp, Xapplabels, Xtest, Xtestlabels)
  print('Papp=%f Ptest=%f'%(Perfapp,Perftest));
  ```
- la visualisation de la classification sur la carte, par vote majoritaire : Vous aurez besoin des 3 fonctions **ctk_label2num**, **som_cplane** et **som_grid** ;

  Exemple (à étudier) :
  ```
  classnames = {'0','1','2','3','4','5','6','7','8','9'};  % labels des classes
  class_ref = ctk_label2num(sMap.labels, classnames); % Classes des referents
  figure; hold on;
  som_cplane(sMap, class_ref);
  som_grid(sMap,'Label',sMap.labels,'Line','none','Marker','none','Labelcolor','k');
  ```
  En python3 :
  ```
  Tfreq,Ulab = ctk.reflabfreq(sMap,Xapp,Xapplabels);
  CBlabmaj   = ctk.cblabvmaj(Tfreq,Ulab);
  CBilabmaj = ctk.label2ind(CBlabmaj, classnames); # transformation des labels en int
  ctk.showcarte(sMap,figlarg=12, fighaut=12,shape='s',shapescale=600,\
                colcell=CBilabmaj,text=CBlabmaj,\
                sztext=16,cmap=cm.jet,showcellid=False);
  ```

2°) Pour le cas optimal, vous devrez de plus présenter les éléments suivants :

- La carte avec les fréquences d'affectation des données par neurone pour l'ensemble d'apprentissage et de test et un résumé des erreurs de classification après avoir examiné de près ces cartes.
- Fonctions à utiliser : **som_label** avec le paramètre '**clear**', **som_autolabel** avec le paramètre 'freq', **som_cplane** (avec la classification obtenue précédemment sur l'ensemble d'apprentissage), et **som_grid**.

  En python3 les méthodes à utiliser sont ctk.cblabfreq (Tfreq et Ulab) et ctk.showcarte (Pour l'ensemble de test il faudra préalablement utiliser ctk.reflabfreq (chose déjà faite pour l'ensemble d'apprentissage))

  La visualisation des chiffres manuscrits (i.e. en données brutes) par référents de la carte topologique (fonction **show_refpat**). (Il y a aussi une méthode ctk.showrefpat en python3 avec un passage de paramètre peut être un peu différent, puisqu'on aurait besoin de passer, entre autre, pour l'ensemble d'apprentissage, les bmus et les hist (fonction respective : ctk.mbmus, ctk.findhits)
- Les états d' «activation» de la carte pour les 10 premiers chiffres. Fonction à utiliser : **show_refactiv ;** on pourra également utiliser la fonction **display_pat** pour mettre les chiffres en regard de leur activation sur la carte). (En pythonb3, on retrouve des fonctions +|- similaires)

3°) Toujours avec le cas optimal retenu en 2° (et en gardant les mêmes paramètres d'apprentissage), on veut voir l'impact de la taille des répartitions des données entre l'ensemble d''apprentissaage et celui de test. Vous devrez donc faire varier l'ensemble d'apprentissage selon les valeurs suivantes : Napp = {50, 100, 150, 200, 250, 300, 350, 400, 450}.

Présenter :

- dans un tableau, les performances en apprentissage et en test
- la classification sur la carte, par vote majoritaire pour chacun des cas