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

TPB03 : Utilisation d'un MLP pour la classification de Chiffres Manuscrits

Pour une réalisation en Python de ce TP, utiliser le document « triedpython », et consulter la table d'équivalence des codages Matlab/Python.

I - Les Objectifs

Le but de ce TP est de reconnaître des chiffres manuscrits à l'aide de réseaux de neurones. Le TP Comprend trois parties :

• La première partie **(§ IV)** consiste à comparer différentes architectures du réseau. On étudiera successivement

- un réseau linéaire sans couche cachée,
- un réseau avec une fonction d'activation non linéaire sans couche cachée
- un réseau avec une fonction d'activation non linéaire et une couche cachée

On comparera les performances de ces différentes architectures

• Dans la seconde partie **(§ V)** on étudiera l'importance du choix du codage.

• La 3ème partie **(§ VI)** abordera, de façon simplifiée, la technique des masques et des poids partagés. Il s'agit de nouveau d'une problématique d'architecture.

==========

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.

Complément de cours : On présente le plus souvent la technique de l'arrêt prématuré (ou « early stopping ») en s'appuyant sur la courbe d'erreur en validation. En fait, on peut tout aussi bien s'appuyer sur la courbe de performance; ce qui est plus adapté dans le cas de la classification. C'est ce qui sera pratiqué dans ce TP comme cela pourra être observé.

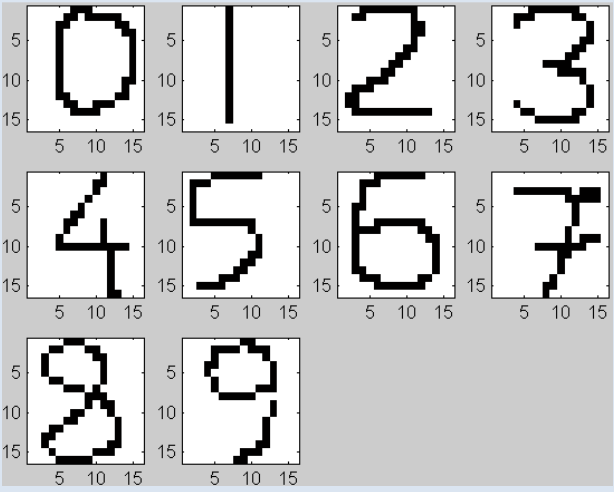
II - Les Données Brutes

x.txt : Est la base de données de chiffres manuscrits en entrée du réseau. Elle est composée de 480 chiffres codés en binaire (±1), dans une matrice 256x480. Cela signifie que chaque «bitmap» 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** .

t.txt : Contient les sorties désirées qui sont associées à chaque chiffre de **x.txt** dans le même ordre. Chaque réponse désirée est constituée d'un vecteur colonne de longueur 10, rempli de -1 partout sauf à la position dont l'indice correspond au chiffre manuscrit. À cette position là, le composant du vecteur vaut +1. L'ensemble de tous ces vecteurs est organisé dans la matrice sous forme d'une matrice 10x480 stockée dans le fichier **t.txt**

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)
```



III - Généralités sur le déroulement du TP

On procédera en deux temps :
- Une phase d'apprentissage qui permet de définir le réseau optimal (architecture et poids du réseau)
- Un calcul des résultats sur les données de Test qui n'ont pas été utilisées lors de l'apprentissage, dont le but est d'évaluer les performances du réseau obtenu.

Lors de la phase d'apprentissage les données de validation sont utilisées pour éventuellement interrompre prématurément (early stopping) l'apprentissage lorsque le dernier minimum de la performance en validation n'a pas été amélioré depuis un certain nombre d'itérations (variable **nb_it_addpmin** dans les programmes d'apprentissage).

A) Répartition des données

On va donc diviser l'ensemble de données en 3 parties de la façon suivante :

- Ensemble d'apprentissage : formes de 1 à 200 ou 300. Il sert à calculer les poids du réseau. On vérifiera que la qualité des résultats s'améliore lorsque l'on prend plus d'exemples en apprentissage mais le temps de calcul augmente. Il est conseillé dans un premier temps de travailler avec un nombre limité de d'exemples (200) dans l'ensemble d'apprentissage pour limiter au maximum les temps de calcul. Dans un second temps on pourra utiliser plus de données (300) pour confirmer les conclusions obtenues.

- Ensemble de validation : formes de 301 à 400. Il permet de vérifier que les performances du réseau obtenu peuvent se généraliser, il sert à vérifier qu'il n'y a pas de sur-apprentissage, et à déterminer si l'apprentissage est terminé. Ce sont les performances sur cet ensemble qui permettent de définir l'architecture optimale du réseau.

- Ensemble de test : formes de 401 à 480. L'ensemble de test ne doit pas intervenir dans la détermination du réseau de quelque manière que se soit ni pour la détermination des poids (apprentissage) ni pour la détermination de l'architecture (validation). Il sert à évaluer les performances du réseau obtenu.

B) Utilisation des programmes fournis pour le déroulement du TP

- **Apprentissage et Validation**

exempleTPB03P1P2 (pour les parties 1 et 2) et **exempleTPB03P3** (pour la partie 3) permettent de déclencher l'apprentissage et la validation en ayant défini différents paramètres nécessaires aux simulations (architecture, pas d'apprentissage, nombre d'itérations, nombre de d'exemples dans les ensembles de d'apprentissage, et de validation**)**. Ces programmes appellent un programme d'apprentissage (**trainingter.m** pour les parties 1 et 2 et **trainshared.m** pour la partie 3) qui effectue automatiquement tous les traitements nécessaires comme la minimisation, les calculs d'erreur et de performance en apprentissage et en validation.

La fréquence de calcul des résultats est déterminée par le paramètre **ecf**. On fixera **ecf=10**.
La fréquence de l'affichage des courbes graphiques de l'apprentissage est contrôlée par le paramètre **gdf**. Pour gagner du temps d'exécution, il est conseillé d'affecter une valeur élevée à ce paramètre.
On précise entre autres que le nombre d'itérations est défini par la variable **n_epochs** sachant que : 1 epoch = 1 passage de la base d'apprentissage.

- **Test**

Pour calculer l'erreur et la performance du réseau sur l'ensemble de test, les étudiants devront modifier ou compléter les fichiers **exempleTPB03P1P2** et **exempleTPB03P3** en s'inspirant de la façon dont les résultats en validation sont calculés à l'aide des différents programmes fournis.

IV - 1ère Partie du TP : Optimisation de l'architecture du classifieur MLP

L'Objectif est de déterminer l'architecture optimale du réseau et de calculer ses performances en apprentissage validation et test. Trois types d'architecture sont possibles :

A) **Réseau linéaire entièrement connecté sans couche cachée** : (paramètre **struct1** égal à 1)
- Correspond à un réseau sans couche cachée avec une fonction de transfert linéaire (pour les cellules de sortie). L'initialisation des poids et l'adaptation des pas d'apprentissage se fait automatiquement en tenant compte du nombre d'entrée de chaque neurone.

B) **Réseau Sigmoïdal entièrement connecté sans couche cachée** (paramètre **struct1** égal à 2):
- Réseau identique au précédent mais avec une fonction d'activation tangente hyperbolique (tanh) sur les cellules de sorties

C) **Réseau entièrement connecté à une couche cachée** (paramètre **struct1** égal à 3) :
- Il utilise la fonction d'activation tangente hyperbolique (tanh) pour les cellules cachées et une fonction linéaire pour celles de sortie

Travail à faire :
Donnez les lignes de code qui permettent d'effectuer les adaptations des poids et des pas
Vous devez comparer les 3 architectures proposées et les analyser en terme de :
- performances sur la base de validation ;
- nombre d'itérations nécessaires pour observer la convergence

Il faut réaliser différentes expériences car selon l'initialisation des poids, les résultats sont différents, et jouer aussi sur le pas d'apprentissage (**lr**). Il conviendra de mettre un nombre d'itérations (**n_epochs**) suffisant (de façon à ce que l'«early stopping» puisse se produire).
Pour le réseau à couche cachée, il faudra de plus faire varier le nombre de cellules cachées (**hidden_layer**).

Une fois le réseau optimal obtenu vous devez, pour chaque architecture de réseau, calculer ses résultats (erreur et performance) sur l'ensemble de Test. Il vous appartiendra d'intégrer les lignes de codes nécessaires pour cela.

Pour chaque cas de type de réseau, vous devrez inclure dans votre rapport le graphe des courbes d'apprentissage correspondant à la meilleure performance obtenue sur l'ensemble de Test.

remarques
x.txt et **t.txt** sont les fichiers de données à utiliser pour cette 1ère partie de TP.

V - 2ème Partie du TP : Optimisation du codage des données

L'objectif de cette seconde partie est de comparer les performances d'un classifieur MLP en fonction des différents types de codage que l'on va faire sur les chiffres, et qui serviront d'entrée au réseau de neurones.

Description des codages utilisés :
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 est donc 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 est aussi 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 6 conventions de codage, peuvent être combinés pour former différents fichiers d'apprentissage. Nous vous soumettons les différents 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**
Le fichier des sorties désirées est le fichier **t.txt**

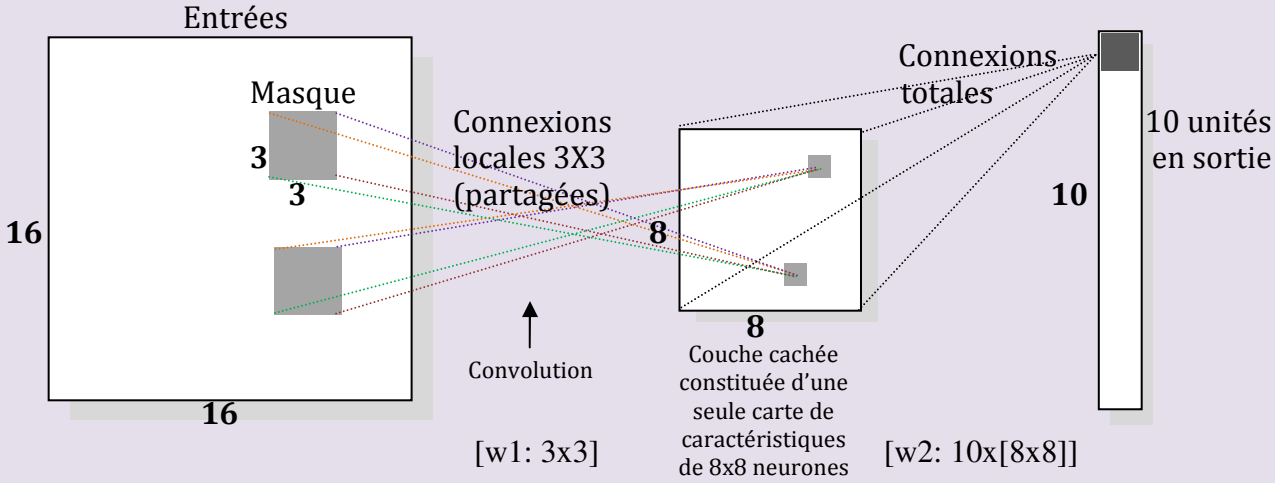
Travail à faire :
Dans chacun des 6 cas de codage, vous devrez entraîner un réseau MLP avec une couche cachée et chercher un nombre « optimal » de neurones dans cette couche cachée (entre 10 et 20, en général).

Vous devrez comparer ces essais et les analyser en terme de :
A) performance et erreur sur la base de test ;
B) nombre d'itérations pour observer une saturation ;

Quel codage allez vous choisir ?

VI - 3ème Partie du TP : Réseau à masques et poids partagés

Le propos de cette 3ème partie de TP est la mise en œuvre d'un réseau à masque et poids partagés tel qu'il est présenté ci-après :



C'est en particulier entre l'entrée et la couche cachée que s'opère la convolution par la mise en œuvre des connexions locales par masque et des poids partagés. L'architecture proposée a une seule couche cachée de taille (8*8) fixée : il s'agit de l'unique carte des caractéristiques. Elle traduit la représentation interne des entrées encodées par le réseau de neurones. L'un des buts de ce TP est de visualiser cette représentation interne.

► Travail à réaliser

A partir du script **exempleTPB03P3**, paramétrer puis lancer l'apprentissage du réseau. Parmi les paramètres à utiliser :
- choisir la fonction d'activation de la couche cachée (linéaire ou tangente hyperbolique).
- retenir différentes tailles de masque (3x3, 5x5, 8x8, 10x10, 12x12)

Il est (fortement) suggéré de répéter une même expérience avec des conditions initiales différentes.

Les algorithmes qui ont été développés étant gourmant en temps d'exécution on pourra se limiter à une centaine d'itérations par exemple.

n_pattern_showing.m est une fonction qui permet d'afficher, pour plusieurs chiffres, l'état de la carte de caractéristique : il s'agit de la représentation interne du réseau.

Pour les différentes expériences que vous retiendrez, il ne sera pas utile de produire les graphiques des courbes, par contre, vous devrez produire, pour les 12 premiers chiffres les représentations internes du réseau ainsi que les performances sur la base de test (en prenant exemple sur **trainshared.m**). Ces différents éléments devront faire l'objet d'un commentaire qui synthétise les différentes expériences. Les paramètres utilisés de chaque expérience présentée devront être indiqués.

En se référents au deux premières parties du TP, il conviendra de comparer les performances en test et de faire apparaître l'intérêt d'utiliser des architectures à masques par rapport à l'utilisation d'un bon codage.