Laboratoire 2
Cours : réseaux de neurones 6INF911

Réseau de Hopfield discret

Le réseau de Hopfield est une mémoire auto-associative qui régénère en sortie la forme présentée en entrée. L'intérêt pratique de ce réseau réside dans sa capacité à reconstruire un signal bruité ou incomplet.

Un réseau de Hopfield comporte une seule couche de neurones, dont la sortie est reliée à l'entrée de tous les autres neurones par le **biais d'éléments de délai**.

Dans sa version discrète, le réseau de Hopfield manipule uniquement des valeurs binaires. Les neurones sont activés par la fonction *Hardlimit()*, dont le déclenchement est soumis à un seuil Ui spécifique à chaque neurone.

Selon sa définition, la sortie demeure inchangée si la valeur nette est égale à la valeur de seuil. En pratique, le seuil T est généralement nul.

À l'inverse des réseaux statiques qui fournissent une réponse finale en une seule passe de propagation du signal, un réseau de Hopfield requiert plusieurs cycles avant de converger vers son état d'équilibre. Ce comportement s'apparente à celui d'une machine à états finis qui, une fois amorcée, tend par elle-même vers un état stable.

A. MÉMORISATION DE FORMES SIMPLES

Apprentissage incrémental

Soit les trois formes suivantes que l'on désire mémoriser dans un réseau de Hopfield :

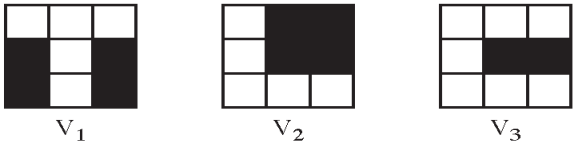


FIGURE 1

Les vecteurs bipolaires correspondant à chacune de ces formes sont : *V1, V2* et *V3.* Puisque chaque forme à mémoriser comprend 9 bits, le réseau doit avoir 9 neurones, disposés de la même façon qu'à la figure 1.

1. Dans Matlab, définissez et initialisez les variables *V1, V2* et *V3 :*

```
V1=[1 1 1 -1 1 -1 -1 1 -1];
V2=[1 -1 -1 1 -1 -1 1 1 1];
V3=[1 1 1 1 -1 -1 1 1 1];
```

2. À la différence des autres types de réseaux neuroniques, il est possible de calculer dès le départ les poids dans un réseau de Hopfield. Calculez les poids W du réseau pour mémoriser les formes *V1* et *V2* selon la règle de Hebb.

3. Vérifiez votre matrice de poids en injectant dans le réseau les formes mémorisées *V1* et *V2* avec la fonction :

```
out,energy= my_hopfield(W,V1,10,3,3,1)
out,energy= my_hopfield(W,V2,10,3,3,1)
```

4. Injectez la forme *V3* qui n'a pas encore été mémorisée, et vérifiez le résultat. Ceci était-il prévisible?

```
out,energy= my_hopfield(W,V3,10,3,3,1)
```

Mémorisez la forme *V3*

1. ***Par apprentissage incrémental, mémoriser la forme*** à la matrice actuelle de poids mémorisant les formes ***V1*** et ***V2.*** Confirmez la validité de la matrice résultante en réinjectant la forme *V3* dans le réseau

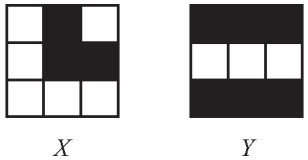
2. Vérifier le cas de désapprendre une forme par soustraction de sa matrice de poids. Confirmez la validité de la matrice résultante en retirant la forme *V3* dans le réseau.

3. Dans votre rapport de lab, discutez des avantages de l'apprentissage incrémental par rapport aux autres formes d'apprentissage.

États stables

Définissez et initialisez les formes suivantes :

```
>>X=[1 -1 1 1 -1 -1 1 1 1];
>>Y=[-1 -1 -1 1 1 1 -1 -1 -1];
```



X *Y*

FIGURE 2

1. Injectez la forme *X* dans le réseau; réalisez quelques essais. Expliquez pourquoi si le résultat est différent d'une fois à l'autre.
2. Sachant que la distance de Hamming entre deux vecteurs binaires de même longueur est égale au nombre de bit, à la même position, qui diffère. On vous demande d'écrire une fonction **`hd()`** qui prend en argument deux vecteurs binaires et calcule et retourne la distance de Hamming.
3. Calculer W0 pour mémoriser les formes *V1*, *V2* et *V3*, alors vers quoi devrait converger le réseau si on lui fournissait la forme *Y* en entrée? (note : utiliser la distance de Hamming pour répondre)

```
>> hd(Y,V1),hd(Y,V2),hd(Y,V3)
```

4. Injectez *Y* dans le réseau W0 et verifier votre résultat avec `my_hopfield()` . Comparez attentivement le résultat avec les formes mémorisées.
5. Calculer la matrice de poids pour mémoriser les formes suivantes : *V1*, *V2* et *le complément de V3* . Vers quoi converge le réseau si on lui fournissait la forme *Y* en entrée? Comparer votre résultat de la question 4?

États parasites

Les caractères suivants proviennent de la fonte OCR-A, couramment utilisée dans les années 1970 pour la reconnaissance optique de caractères imprimés :

Échantillons de caractères de la fonte OCR-A
FIGURE 3

1. Chargez ces caractères dans Matlab avec la commande `load ocra`. Vous aurez alors dans votre espace de travail les variables `zero`, `un`, `sept` et `huit`. Vous disposez aussi du caractère `noisy`.

```
xsize=12
ysize=10
ocra = scipy.io.loadmat('ocra.mat')
zero=ocra.get('zero')
un=ocra.get('un')
sept=ocra.get('sept')
huit=ocra.get('huit')
noisy=ocra.get('noisy')
data=np.vstack((zero,un,sept,huit))
v=sign(zero + un + sept)
v=np.where(v>=0,1,v)
v=np.where(v<-1,-1,v)
noisy=v
 my_show(noisy,10,12)
```

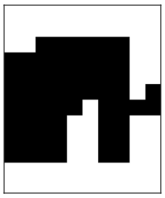


FIGURE 4

2. Calculez les poids requis pour mémoriser les formes `zero`, `un`, `sept` et `huit` de la figure 3 et placez le résultat dans la variable `poids`.
3. Vers quel caractère devrait converger le réseau si on lui fournissait la forme `noisy` en entrée?
4. Injectez `noisy` dans le réseau via l'appel de la fonction `my_hopfsolv()` Quel est le résultat?

Note : Vous saviez déjà que ce réseau a huit états stables : les quatre caractères mémorisés et leurs compléments. Pour les réseaux de grande taille, ***toute combinaison linéaire d'un nombre impair de formes génère un état stable parasite****, sauf lorsque les formes sont orthogonales entre elles (produit scalaire nul). Cette dernière condition est difficilement réalisable en pratique.*
Ces mémoires parasites sont généralement situées à proximité des formes mémorisées, et ont une énergie supérieure aux « bonnes » formes; elles correspondent à des minima locaux de la fonction d'énergie.

5. Combien d'états stables existe-t-il au maximum dans le réseau? Notez qu'une forme ne peut engendrer d'état parasite avec son complément;
6. Il est parfois possible de faire désapprendre au réseau certains états parasites. Essayez de réinjecter `noisy` en enlevant au préalable ce minimum local de la mémoire du réseau, par *dilution neuronale*. Le principe est d'enlever une fraction de la matrice de poids de l'état parasite :

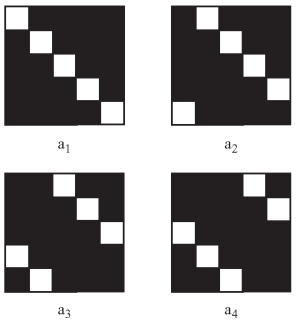
$$W_0 = W_0 formes - \alpha W_0 parasites$$

Avec alpha $\alpha \subseteq [0,1]$ la portion d'état parasite à soustraire de la matrice des poids. Le paramètre αest déterminé par essai/erreur. S'il est toujours possible de désapprendre l'état parasite, il est cependant fréquent que les autres états stables soient aussi affectés.

7. Indiquez dans votre rapport les résultats obtenus dans votre tentative d'éliminer l'état parasite `noisy`. N'oubliez pas de tester les états stables `zero`, `un`, `sept` et `huit` par la suite.

Capacité et résistance au bruit

Soit les quatre formes suivantes, séparées les unes des autres par une distance de Hamming de 10 bits :



Formes séparées les unes des autres d'une distance de Hammin de 10 bits.

FIGURE 5

Chargez ces formes dans votre espace de travail avec la commande `load diag5x5`

8. Calculez la matrice de poids W0 requise pour mémoriser seulement la forme a1.
9. Déterminez par essai/erreur le taux de bruit maximum toléré par le réseau :
   ```
   a1noise =noise(a1, 10);
   ```
   %Le 2ieme argument de la fonction noise 10 correspond au pourcentage d'erreur (10%).
   ```
   my_hopfield(W0,a1noise,15,xsize,ysize,1)
   ```
10. Déterminez similairement le taux de bruit maximum lorsqu'il y a 2, 3 et 4 formes en mémoire. Bien que ce soit statistiquement insuffisant, considérez que le taux obtenu est bon si au moins trois essais successifs permettent de récupérer la forme d'origine.

Il est communément admis que le nombre de formes qui peuvent être mémorisées avec un réseau de M neurones est de :

$$p < \frac{N}{2log_2(N)}$$

Cette dernière équation suppose que toutes les formes mémorisées peuvent être récupérées correctement avec un niveau de bruit « acceptable ».

11. De quels facteurs dépend la récupération correcte d'un signal bruité.
12. Commentez l'équation précédente à la lumière de vos résultats. Présentez vos résultats sous forme graphique.

RECONNAISSANCE DE CARACTÈRES MANUSCRITS

Nous utiliserons un réseau de Hopfield pour essayer de reconnaître des caractères manuscrits.

Description des données

Tel qu'illustré, vous disposez d'un ensemble de 100 chiffres, à raison de 10 échantillons par classe. Ces échantillons sont regroupés dans dix fichiers Matlab, `d0` à `d9`, qui peuvent être lus avec la commande `load`.

Script Python à adapter pour visualiser une matrice chiffres :

```
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import mnist
(X_train, Y_train), (X_test, Y_test) = mnist.load_data()

fig, ax = plt.subplots(10,10)
for i, ax in enumerate(ax.flatten()):
    ax.imshow(X_train[i,:,:], cmap='gray_r')
    ax.axis('off')
plt.plot()
```

Les échantillons sont des images binaires de 10x10, transformées pour les besoins en vecteurs de 1x100. Chaque fichier `d0` à `d9` contient deux variables :

a) La matrice `samples`, qui contient tous les échantillons de la classe, à raison d'un échantillon par ligne de la matrice. Par exemple, samples(1,:) correspond à l'échantillon `s1` de la classe, `samples(2,:)` à l'échantillon `s2`, et ainsi de suite.

b) Un scalaire `classe`, qui indique la classe d'appartenance des échantillons (chiffre de 0 à 9). Cette variable est utile lorsque vous travaillez sur plusieurs fichiers différents : sa valeur vous indique à tout moment le contenu de `samples`.

Choix des prototypes

Votre principal travail sera de choisir vos prototypes, c'est-à-dire les formes à mémoriser;

1. Combien le réseau doit-il avoir de neurones? Si on tient compte du bruit, combien de formes pouvez-vous mémoriser de façon réaliste avec ce réseau.

2. Chargez en mémoire les échantillons de chiffre zéro avec `load d0`. En examinant la figure des chiffres, choisissez l'échantillon qui représenterait le mieux l'ensemble des chiffres « zéro ». Par exemple, si vous choisissez le 7e échantillon, initialisez : `proto0 = samples(7,:);` Vous pouvez visualiser votre échantillon avec `my_show(proto0,10,10)`.
3. Calculez les poids du réseau pour mémoriser la forme `proto0` selon la règle de Hebb.

4. Vérifiez la convergence de votre réseau du `proto0` en injectant les 10 échantillons du chiffre 0 cad `samples(i,:) avec i=1 :10. Donner le taux de reconnaissance sur les 10` échantillons.

5. Chargez en mémoire les échantillons de chiffre zéro avec `load d1`. En examinant la figure des chiffres, choisissez l'échantillon qui représenterait le mieux l'ensemble des chiffres « un » soit `proto1 = samples(xx,:);` Vous pouvez visualiser votre échantillon avec `my_show(proto1,10,10)`.
6. Calculez les poids du réseau pour mémoriser la forme `proto1` selon la règle de Hebb.
7. Vérifiez la convergence de votre réseau du `proto1` en injectant les 10 échantillons du chiffre 1 et donner le taux de reconnaissance sur les 10 échantillons.
8. Chargez en mémoire les échantillons de chiffre zéro avec `load d2`. En examinant la figure des chiffres, choisissez l'échantillon qui représenterait le mieux l'ensemble des chiffres « un » soit `proto2 = samples(xx,:);` Vous pouvez visualiser votre échantillon avec `my_show(proto2,10,10)`.
9. Calculez les poids du réseau pour mémoriser la forme `proto2` selon la règle de Hebb.
10. Vérifiez la convergence de votre réseau du `proto2` en injectant les 10 échantillons du chiffre 2 et d`onner le taux de reconnaissance sur les 10` échantillons.
11. Calculez les poids du réseau pour mémoriser la forme `proto= [proto0; proto1 ; proto2]` selon la règle de Hebb.
12. Vérifiez la convergence de votre réseau du `proto` en injectant les 10 échantillons des chiffres 0, 1 et 2 et d`onner le taux de reconnaissance sur les 30` échantillons.
13. Commenter vos résultats.