Chapitre 2
Réseaux neuronaux et machine learning

Denis Priou – v2.0 (10/04/2018)

Dans le chapitre précédent, nous avons découvert le fonctionnement élémentaire d'un neurone formel et d'un petit réseau de neurones. Nous avons aussi vu une formulation simplifiée des méthodes de descente de gradient et de rétropropagation des erreurs. Ces concepts ont été mis en œuvre sur un exemple très simple, celui de la reconnaissance d'un type d'iris parmi les trois possibles (setosa, versicolor, virginica), dans une population de 150 échantillons.

L'objectif de ce chapitre est de formuler ces concepts de façon plus générale et de les mettre en œuvre sur des données et sur des réseaux de neurones plus complexes. Il s'agit d'un premier passage à l'échelle. Nous manipulerons environ un millier de neurones répartis dans trois couches (entrée, cachée, sortie). Nous appliquerons les méthodes du machine learning à la reconnaissance de caractères et utiliserons des jeux de données contenant plusieurs dizaines de milliers d'échantillons. Pour plus de détail sur les méthodes employées, on pourra consulter l'article intitulé « Gradient-Based Learning Applied to Document Recognition » (Yann LeCun et al., Proc. of the IEEE, nov. 1998).

1 – Reconnaissance de caractères

Nous allons contruire un réseau de neurones capable de reconnaître des caractères manuscrits ou typographiés (plus précisément, des chiffres : 0, 1, 2, ..., 9), fournis au réseau sous la forme d'une image (un bitmap). Nous allons tester ce réseau sur un jeu de données de référence, le « MNIST dataset » (voir http://yann.lecun.com/exdb/mnist/).

Nous testerons préalablement notre réseau de neurones sur un jeu de données simplifié de quelques milliers de caractères, construit par nos soins. Ce jeu simplifié, que nous avons baptisé « miniMNIST », ne contient que des chiffres impairs typographiés (1, 3, 5, 7, 9) : il nous permettra de monter en puissance progressivement et d'avoir une première approche des problèmes de confusion entre des chiffres graphiquement proches (3 et 5, par exemple).

1.1 – Le jeu de données miniMNIST

Le jeu de données miniMNIST est fait de caractères représentés sous la forme de bitmaps de 8 pixels de largeur par 11 pixels de hauteur, selon le modèle du tableau 1.

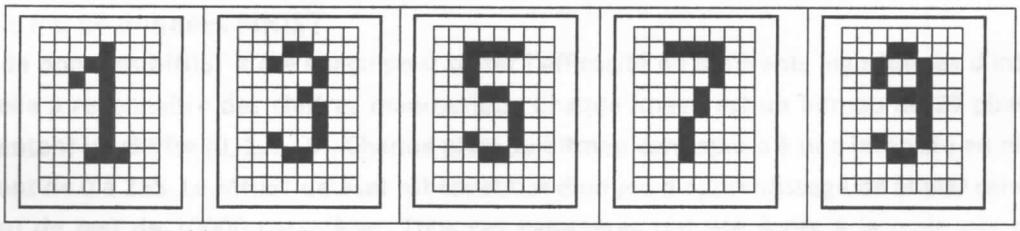

Tableau 1 – les caractères du jeu de données « miniMNIST »

La taille d'un bitmap est de 88 octets, chaque pixel étant caractérisé par une intensité en niveaux de gris (de 0 à 255). Nous avons généré un jeu d'apprentissage de 5000 caractères et un jeu de test de 1000 caractères, à raison de 20% de caractères pour chaque chiffre. Les caractères originels sont supposés constitués de pixels noirs (0) ou blancs (255). Pour introduire des éléments d'aléa dans ces derniers, nous avons pratiqué les tranformations suivantes (voir tableau 2) pour dégrader leur qualité :

- Un ou deux pixels « blancs » passent au « noir » et réciproquement (effet « grain de poussière ») ;
- Le chiffre a subi une translation par rapport à la position originelle ;
- Le chiffre a été partiellement « effacé », i.e. l'intensité de certains pixels « blancs » est diminuée plus ou moins fortement ;
- Le chiffre est déformé par translation d'une ou plusieurs colonnes.

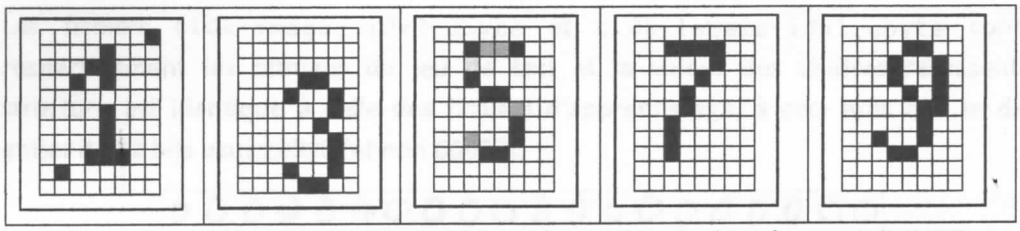

Tableau 2 – Quelques caractères « dégradés » du jeu de données « miniMNIST »

Quatre fichiers ont été créés :

- Le fichier `miniMNIST_images_appr.dat`, contenant les bitmaps du jeu d'apprentissage : ce fichier débute par une clé « magique » de 32 bits (`0x00001112`), suivie de 3 entiers de 32 bits indiquant le nombre de bitmaps (5000), leur largeur (8 colonnes) et leur hauteur (11 lignes) ; on trouve enfin 5000 séquences de 11 x 8 octets non signés (les 5000 bitmaps, stockés ligne par ligne) ;
- Le fichier `miniMNIST_labels_appr.dat`, contenant la liste des chiffres représentés dans les bitmaps du jeu d'apprentissage (dans le même ordre) : ce fichier débute par une clé magique de 4 octets (`0x00001110`), suivie du nombre de valeurs sur 4 octets (5000) ; on trouve enfin la valeur de ces 5000 chiffres (octets non signés) ;
- Les fichiers `miniMNIST_images_test.dat` et `miniMNIST_labels_test.dat` contiennent respectivement les bitmaps du jeu de test et la valeur des chiffres représentés : leur structure est identique à celle des fichiers d'apprentissage, à ceci près que le deuxième entier de 32 bits vaut 1000 (et non 5000).

Cette structuration des données en fichiers correspond exactement à celle retenue par le MNIST pour ses propres fichiers d'apprentissage et de test (voir paragraphe 1.2). Ceci nous permettra d'appeler plus facilement, à partir d'un même code, l'un ou l'autre des jeux de données.

1.2 – Le jeu de données MNIST

Le jeu de données MNIST a été conçu pour tester l'efficacité de différents algorithmes d'intelligence artificielle à reconnaître des chiffres manuscripts. Chaque image est un bitmap de 28 pixels par 28 représentant un chiffre (0, 1, ..., 9). Chaque pixel du bitmap correspond à une intensité en niveaux de gris allant de 0 à 255. Le MNIST dataset est constitué d'un jeu d'apprentissage de 60000 caractères et d'un jeu de test de 10000 caractères. Tous ces caractères ont été écrits à la main par différents « cobayes » humains et ont été numérisés et pré-traités. La diversité graphologique et l'aspect « données réelles » font de ce jeu de données un test standard pour les algorithmes de reconnaissance de caractères (voir figure 1).

Le jeu de données MNIST est fourni sous la forme de quatre fichiers :

- Le fichier `train-images-idx3-ubyte`, qui contient les bitmaps du jeu d'apprentissage : ce fichier débute par une clé « magique » de 32 bits (`0x00000803`), suivie de 3 entiers de 32

bits indiquant le nombre de bitmaps (60000), leur largeur (28 colonnes) et leur hauteur (28 lignes) ; on trouve ensuite 60000 séquences de 28 x 28 octets non signés (les 60000 bitmaps, stockés ligne par ligne) ;

- Le fichier `train-labels-idx1-ubyte`, qui contient la liste des chiffres représentés dans les bitmaps du jeu d'apprentissage (dans le même ordre) : ce fichier débute par une clé magique de 4 octets (`0x00000801`), suivie du nombre de valeurs sur 4 octets (60000) ; on trouve ensuite la valeur de ces 60000 chiffres (octets non signés).
- Les fichiers `t10k-images-idx3-ubyte` et `t10k-labels-idx1-ubyte` contiennent respectivement les bitmaps du jeu de test et la valeur des chiffres représentés : leur structure est identique à celle des fichiers d'apprentissage, à ceci près que le deuxième entier de 32 bits vaut 10000 (et non 60000).

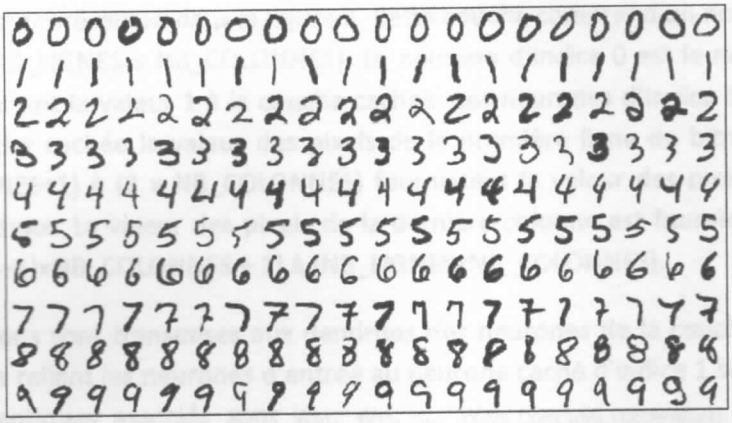

Figure 1 – Quelques exemples de caractères tirés du jeu de données MNIST

2 – Réseau de neurones retenu pour traiter les jeux de caractères

Définissons la structure et le fonctionnement du réseau neuronal qui traitera les données miniMNIST et MNIST : nombre de couches, nombre de neurones par couche, méthodes de traitement, etc. Nous en profiterons pour reformuler en toute généralité les concepts fondamentaux présentés dans le chapitre précédent : neurone formel, couche et réseau de neurones, algorithme de rétro-propagation des erreurs, algorithme de descente de gradient. Nous définirons au passage quelques notations utiles dans la suite de ce chapitre.

2.1 – Structure retenue pour le réseau neuronal

Notre réseau de neurones prend en entrée les bitmaps du jeu de données, l'un après l'autre, et formule pour chacun une prédiction (est-ce un zéro ? est-ce un chiffre 1 ? etc.). Ce réseau comporte trois couches : la couche d'entrée, la couche cachée et la couche de sortie.

La couche d'entrée reçoit les données d'entrée, à savoir le bitmap à analyser : chaque neurone de cette couche prend en entrée la valeur de l'un des pixels du bitmap. La couche d'entrée comporte donc (NB_LIGNES x NB_COLONNES) neurones, où NB_LIGNES et NB_COLONNES désignent respectivement le nombre de lignes et le nombre de colonnes d'un bitmap. Nous complétons cette couche par un neurone de biais : la couche d'entrée comprend donc (1 + NB_LIGNES x NB_COLONNES) neurones. Le neurone de biais a pour indice 0 ; les neurones d'indice 1, 2, ..., (NB_LIGNES x NB_COLONNES) reçoivent la valeur de chacun des (NB_LIGNES x NB_COLONNES) pixels. On note $N_{entrée}$ le nombre total de neurones de la couche d'entrée. Pour le jeu de données miniMNIST, $N_{entrée} = 89$; pour le jeu de données MNIST, $N_{entrée} = 785$.

La couche de sortie produit dix valeurs, à savoir la probabilité pour le bitmap fourni en entrée du réseau représente le chiffre 0, le chiffre 1, ..., le chiffre 9. Cette couche comporte donc 10 neurones. On note N_{sortie} le nombre de neurones de la couche de sortie, pour conserver aux explications leur caractère de généralité.

La couche cachée n'a pas de taille contrainte, à la différence des deux autres couches. Le nombre de neurones cachés, que l'on note $N_{caché}$, sera compris entre $N_{entrée}$ et N_{sortie}. On testera l'efficacité prédictive du réseau en faisant varier ce nombre. La couche cachée contiendra un neurone de biais, d'indice 0, et ($N_{caché}$–1) neurones « actifs » (c'est-à-dire recevant les informations produites par la couche d'entrée).

2.2 – Couche d'entrée

La couche d'entrée est représentée dans la figure 2. Cette couche comprend un nombre de neurones égal à $N_{entrée}$ = (1 + NB_LIGNES x NB_COLONNES). Le neurone d'indice 0 est le neurone de biais : il fournit systématiquement la valeur 1 à la couche cachée. Les neurones d'indice 1 à NB_COLONNES fournissent à la couche cachée la valeur des pixels de la première ligne du bitmap. Les neurones d'indice (NB_COLONNES+1) à (2 x NB_COLONNES) fournissent la valeur des pixels de la deuxième colonne, et ainsi de suite. La valeur des pixels de la dernière colonne est fournie par les neurones d'indice [(NB_LIGNES–1)xNB_COLONNES + 1] à (NB_LIGNESxNB_COLONNES).

Toutes ces informations sont transmises aux dendrites des neurones de la couche cachée. Dans la figure 2, les dendrites reliant les neurones d'entrée au neurone caché d'indice 1 sont représentés en bleu. Les poids synaptiques associés sont w_{01}, w_{11}, ..., $w_{(NB\_LIGNESxNB\_COLONNES)(1)}$. Les dendrites du neurone caché d'indice j sont représentés en rouge. Les poids synaptiques associés sont w_{0j}, w_{1j}, ..., $w_{(NB\_LIGNESxNB\_COLONNES)(j)}$. Enfin, les dendrites du dernier neurone d'entrée, d'indice $N_{entrée} - 1$ = (NB_LIGNES x NB_COLONNES), figurent en vert. Les poids synaptiques associés sont $w_{(0)(Ncaché-1)}$, $w_{(1)(Ncaché-1)}$, ..., $w_{(NB\_LIGNESxNB\_COLONNES)(Ncaché-1)}$. De façon générale, le neurone d'entrée « i » transmet ses informations au neurone caché « j » via sa dendrite de poids synaptique w_{ij}.

Convention de notation ! Dans la suite de ce chapitre, on utilisera systématiquement la lettre « i » pour désigner l'indice d'un neurone de la couche d'entrée. La valeur de « i » pourra donc varier de 0 à ($N_{entrée}$–1). On notera systématiquement X_i la valeur transmise par le neurone d'entrée d'indice i à la couche cachée. On utilisera systématiquement le lettre « j » pour désigner l'indice d'un neurone de la couche cachée. La valeur de « j » pourra donc varier de 0 à ($N_{caché}$–1).

Pour le jeu de données miniMNIST, NB_LIGNES vaut 8 et NB_COLONNES vaut 11 : la couche d'entrée comporte donc 1 + 8 x 11 = 89 neurones. Dans le cas du jeu de données MNIST, NB_LIGNES et NB_COLONNES sont égaux à 28 : la couche d'entrée comporte donc 1 + 28 x 28 = 785 neurones.

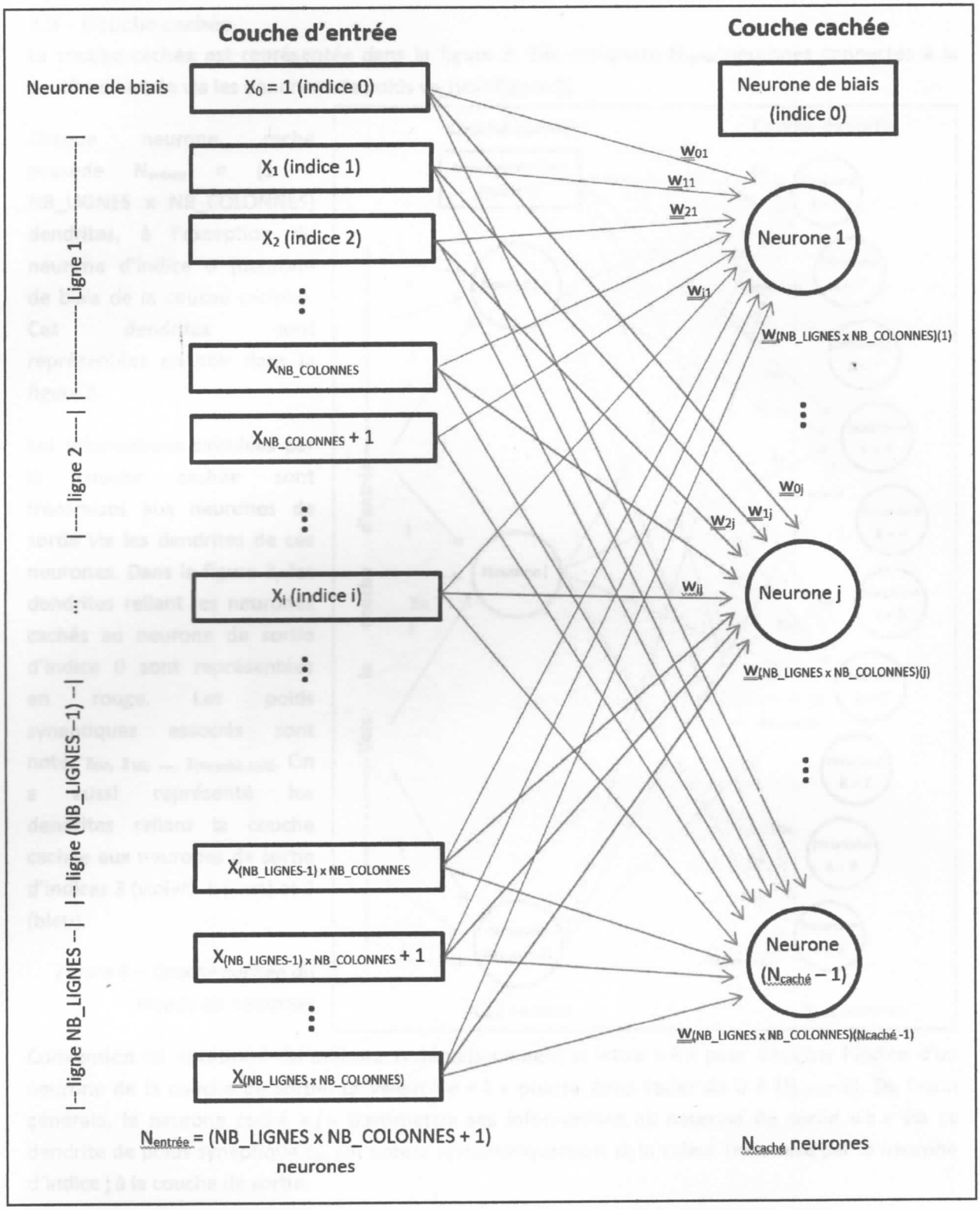

Figure 2 – Couche d'entrée du réseau neuronal

2.3 – Couche cachée

La couche cachée est représentée dans la figure 3. Elle comporte $N_{caché}$ neurones connectés à la couche d'entrée via les dendrites de poids w_{ij} (voir figure 2).

Chaque neurone caché possède $N_{entrée}$ = (1 + NB_LIGNES x NB_COLONNES) dendrites, à l'exception du neurone d'indice 0 (neurone de biais de la couche cachée). Ces dendrites sont représentées en noir dans la figure 3.

Les informations calculées par la couche cachée sont transmises aux neurones de sortie via les dendrites de ces neurones. Dans la figure 3, les dendrites reliant les neurones cachés au neurone de sortie d'indice 0 sont représentées en rouge. Les poids synaptiques associés sont notés z_{00}, z_{10}, ..., $z_{(Ncaché-1)(0)}$. On a aussi représenté les dendrites reliant la couche cachée aux neurones de sortie d'indices 3 (violet), 6 (vert) et 9 (bleu).

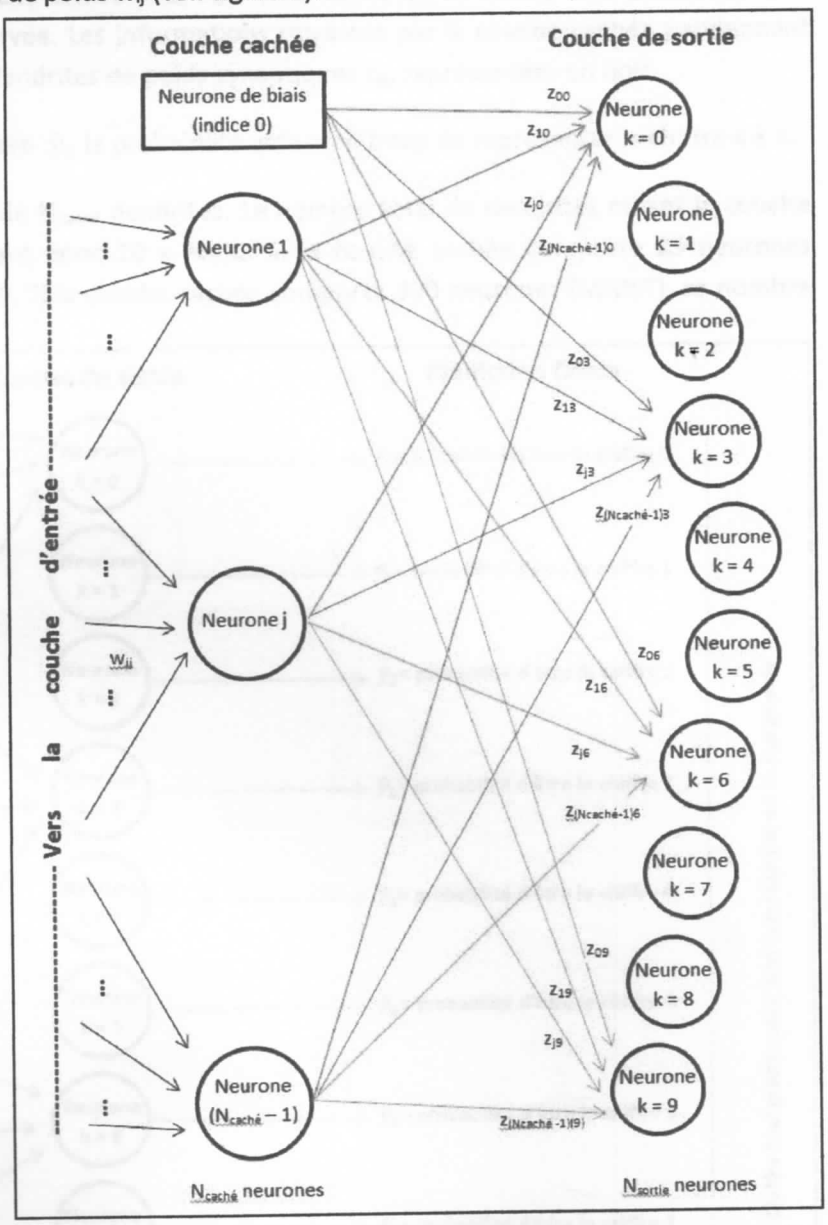

Figure 3 – Couche cachée du réseau de neurones

Convention de notation ! On utilisera systématiquement la lettre « k » pour désigner l'indice d'un neurone de la couche de sortie. La valeur de « k » pourra donc varier de 0 à (N_{sortie}–1). De façon générale, le neurone caché « j » transmettra ses informations au neurone de sortie « k » via sa dendrite de poids synaptique z_{jk}. On notera systématiquement U_j la valeur transmise par le neurone d'indice j à la couche de sortie.

Pour le jeu de données miniMNIST, le nombre total de dendrites reliant la couche d'entrée à la couche cachée vaut 89 x $N_{caché}$. Si la couche cachée comporte 25 neurones, ce nombre vaut 2225. Dans le cas du jeu de données MNIST, le nombre de dendrites est nettement plus important et vaut 785 x $N_{caché}$. Si la couche cachée comporte 300 neurones, ce nombre vaut 235500. On peut donc apprécier les volumes de calcul respectifs dans ces deux cas de figure.

2.4 – Couche de sortie

La couche de sortie comporte N_{sortie} (= 10) neurones qui produisent, pour le bitmap fourni en entrée, la probabilité $\hat{y}_0$, $\hat{y}_1$, ..., $\hat{y}_9$ d'être le chiffre 0, 1, ..., 9 (voir figure 4). Le chiffre final prédit est celui dont la probabilité est la plus élevée. Les informations calculées par la couche cachée parviennent aux neurones de sortie via leurs dendrites de poids synaptiques z_{jk}, représentées en noir.

Convention de notation ! On notera $\hat{y}_k$ la probabilité qu'a un bitmap de représenter le chiffre « k ».

Chaque neurone de sortie possède $N_{caché}$ dendrites. Le nombre total de dendrites reliant la couche cachée à la couche de sortie vaut donc 10 x $N_{caché}$. Si la couche cachée comporte 25 neurones (miniMNIST), ce nombre vaut 250. Si la couche cachée comporte 300 neurones (MNIST), ce nombre vaut 3000.

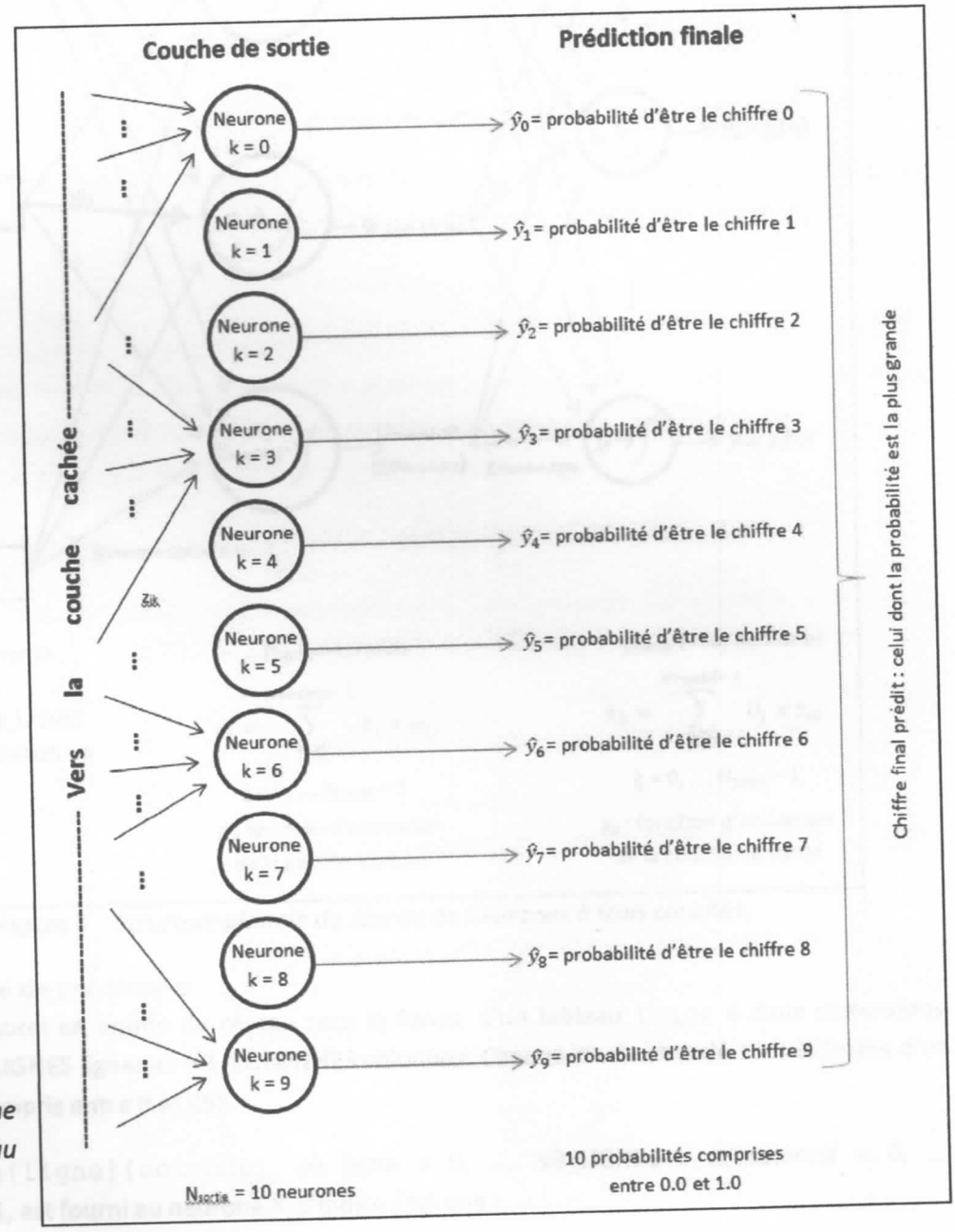

Figure 4 – Couche de sortie du réseau de neurones.

La structure globale du réseau de neurones est résumée dans la figure 5.

3 – Algorithmes mis en oeuvre

La figure 5 reprend l'ensemble des notations introduites plus haut. Elle permet de décrire les algorithmes appliqués pour reconnaître les caractères des jeux de données miniMNIST et MNIST.

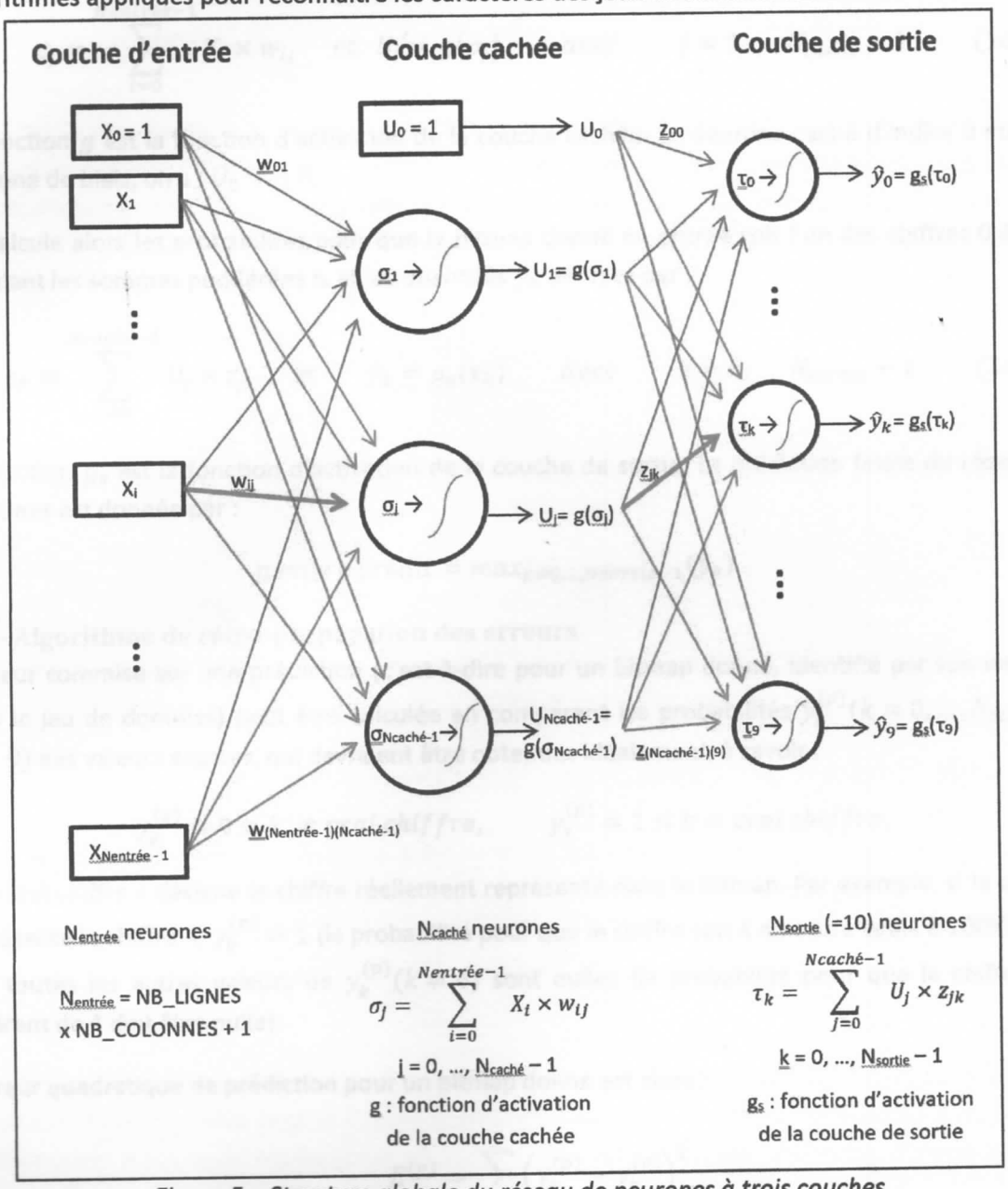

Figure 5 – Structure globale du réseau de neurones à trois couches.

3.2 – Algorithme de prédiction

Un bitmap est fourni en entrée du réseau sous la forme d'un tableau `Image` à deux dimensions, comportant NB_LIGNES lignes et NB_COLONNES colonnes. Chaque pixel est codé sous la forme d'un nombre entier compris entre 0 et 255.

Le pixel `Image[ligne][colonne]`, où ligne = 0, ..., NB_LIGNES–1 et colonne = 0, ..., NB_COLONNES–1, est fourni au neurone X_i d'indice i tel que :

$$i = ligne \times NB\_COLONNES + colonne$$

Une fois initialisée la couche d'entrée X_i, on met en action les neurones de la couche cachée en calculant les moyennes pondérées σ_j et les quantités U_j, définies par (j = 1, ..., $N_{caché}$–1) :

$$\sigma_j = \sum_{i=0}^{Nentrée-1} X_i \times w_{ij} \quad et \quad U_j = g(\sigma_j), \qquad avec \qquad j = 1, \dots, N_{caché} - 1 \qquad (1a/1b)$$

La fonction g est la fonction d'activation de la couche cachée. Le neurone caché d'indice 0 étant le neurone de biais, on a : $U_0 = 1.0$.

On calcule alors les probabilités pour que le bitmap donné en entrée soit l'un des chiffres 0 à 9 en calculant les sommes pondérées τ_k et les quantités $\hat{y}_k$ définies par :

$$\tau_k = \sum_{j=0}^{Ncaché-1} U_j \times z_{jk} \quad et \quad \hat{y}_k = g_s(\tau_k) \qquad avec \qquad k = 0, \dots, N_{sortie} - 1 \qquad (2a/2b)$$

La fonction g_s est la fonction d'activation de la couche de sortie. La prédiction finale du réseau de neurones est donnée par :

$$nombre\ prédit = max_{k=0,\dots,Nsortie-1}(\hat{y}_k) \qquad (3)$$

3.2 – Algorithme de rétro-propagation des erreurs

L'erreur commise sur une prédiction (c'est-à-dire pour un bitmap donné, identifié par son indice p dans le jeu de données) peut être calculée en comparant les probabilités $\hat{y}_k^{(p)}(k = 0, \dots, N_{sortie} - 1 = 9)$ aux valeurs exactes, qui devraient être obtenues idéalement, à savoir :

$$y_k^{(p)} = 0\ si\ k \neq vrai\ chiffre, \qquad y_k^{(p)} = 1\ si\ k = vrai\ chiffre,$$

où « vrai chiffre » désigne le chiffre réellement représenté dans le bitmap. Par exemple, si le bitmap représente le chiffre 4, $y_k^{(p)} = 1$ (la probabilité pour que le chiffre soit 4 doit être égale à 100%) alors que toutes les autres valeurs de $y_k^{(p)}(k \neq 4)$ sont nulles (la probabilité pour que le chiffre soit différent de 4 doit être nulle).

L'erreur quadratique de prédiction pour un bitmap donné est alors :

$$E^{(p)} = \sum_{k=0}^{9} \left(\hat{y}_k^{(p)} - y_k^{(p)}\right)^2 \qquad (4)$$

où p désigne le numéro d'ordre du bitmap dans le jeu de données fourni en entrée du réseau de neurones. L'erreur quadratique moyenne sur l'ensemble du jeu de données est alors :

$$E = \frac{1}{P}\sum_{p=0}^{P-1} E^{(p)} = \frac{1}{P} \sum_{\substack{0 \le p \le P-1 \\ 0 \le k \le 9}} \left(\hat{y}_k^{(p)} - y_k^{(p)}\right)^2 \qquad (5)$$

où P est le nombre total de bitmaps dans le jeu de données.

L'erreur $\delta_k^{sortie(p)}$ commise en sortie du neurone de sortie k, pour l'échantillon d'indice p, est donnée par :

$$\delta_k^{sortie(p)} = \frac{\partial E^{(p)}}{\partial \tau_k} = 2 \times \left(\hat{y}_k^{(p)} - y_k^{(p)}\right) \times g_s'(\tau_k), \qquad k = 0, \dots, 9 \tag{6}$$

où g_s' est la dérivée de la fonction d'activation des neurones de la couche de sortie. Si la fonction d'activation retenue est la fonction identité, sa dérivée g_s' est une fonction constante : $g_s'(x) = 1$. Si l'on utilise la fonction sigmoïde, le calcul de sa dérivée est réalisé au moyen de la formule (9) ci-dessous. L'erreur globale moyenne, sur l'ensemble du jeu de données, pour la couche de sortie, est donnée par :

$$\delta_k^{sortie} = \frac{1}{P} \sum_{p=0}^{P-1} \delta_k^{sortie(p)}, \qquad k = 0, \dots, 9 \tag{7}$$

On peut déduire de la formule (6) l'erreur $\delta_j^{caché(p)}$ commise par le neurone j de la couche cachée :

$$\delta_j^{caché(p)} = \frac{\partial E^{(p)}}{\partial \sigma_j} = \sum_{k=0}^{9} \frac{\partial E^{(p)}}{\partial \tau_k} \times \frac{\partial \tau_k}{\partial \sigma_j} = \sum_{k=0}^{9} \delta_k^{sortie(p)} \times g'(\sigma_j) \times z_{jk}, \quad j = 1, \dots, N_{caché} - 1 \tag{8}$$

où g' est la dérivée de la fonction d'activation des neurones cachés. Si la fonction d'activation retenue est la fonction sigmoïde, sa dérivée g' vérifie la propriété suivante :

$$g'(\sigma_j) = g(U_j) \times \left(1 - g(U_j)\right) \qquad avec \qquad U_j = g(\sigma_j) \tag{9}$$

Pour $j = 0$ (neurone de biais caché), aucune erreur n'est commise : $\delta_0^{caché(p)} = 0$. L'erreur globale moyenne, sur l'ensemble du jeu de données, pour la couche cachée, est donné par :

$$\delta_j^{caché} = \frac{1}{P} \sum_{p=0}^{P-1} \delta_j^{caché(p)}, \qquad j = 1, \dots, N_{caché} - 1 \tag{10}$$

3.3 ~~3.4~~ – **Algorithme correction des poids synaptiques : méthode de descente de gradient**

Les erreurs étant calculées pour la couche cachée et la couche de sortie, il est possible d'en déduire les corrections à appliquer sur les poids synaptiques w_{ij} et z_{jk}. Pour un échantillon d'indice p du jeu de données, le gradient de l'erreur quadratique $E^{(p)}$ est obtenu comme suit :

$$\frac{\partial E^{(p)}}{\partial w_{ij}} = \delta_j^{caché(p)} \times X_i \qquad (j \neq 0) \tag{11}$$

$$\frac{\partial E^{(p)}}{\partial z_{jk}} = \delta_k^{sortie(p)} \times U_j \tag{12}$$

Le gradient global moyen, pour tout le jeu de données, est donné par :

$$\frac{\partial E}{\partial w_{ij}} = \frac{1}{P} \sum_{p=0}^{P-1} \delta_j^{caché(p)} \times X_i \tag{13}$$

$$\frac{\partial E}{\partial z_{jk}} = \frac{1}{P}\sum_{p=0}^{P-1} \delta_k^{sortie(p)} \times U_j \qquad (14)$$

Les corrections globales à effectuer sur les poids synaptiques sont alors :

$$\Delta w_{ij} = -\alpha \times \frac{\partial E}{\partial w_{ij}} \qquad (15)$$

$$\Delta z_{jk} = -\alpha \times \frac{\partial E}{\partial z_{jk}} \qquad (16)$$

où α est le coefficient d'apprentissage.

3.4 ~~2.5~~ – **Algorithme d'apprentissage : méthode générale**

L'algorithme général d'apprentissage comporte trois étapes.

La première étape consiste à initialiser les corrections de poids synaptique Δw_{ij} et Δz_{jk} à zéro ($i = 0, \dots, N_{entrée} - 1$; $j = 0, \dots, N_{caché} - 1$; $k = 0, \dots, N_{sortie} - 1$).

La deuxième étape consiste à calculer, pour chaque échantillon p du jeu de données ($p = 0, \dots, P - 1$), sa contribution $\Delta w_{ij}^{(p)}$ et $\Delta z_{jk}^{(p)}$ aux corrections de poids synaptique. Pour cela, il faut :

a) Calculer les valeurs $X_i^{(p)}$ de la couche d'entrée ($i = 0, \dots, N_{entrée} - 1$) à partir de l'échantillon d'indice p (voir paragraphe 2.2) ;

b) Calculer les prédictions $\hat{y}_k^{(p)}$ concernant cet échantillon ($k = 0, \dots, N_{sortie} - 1$) en utilisant les formules (1a) et (1b) pour calculer $\sigma_j^{(p)}$ et $U_j^{(p)}$, $j = 0, \dots, N_{caché} - 1$, puis (2a) et (2b) pour calculer $\tau_k^{(p)}$ et $\hat{y}_k^{(p)}$, $k = 0, \dots, N_{sortie} - 1$, enfin le nombre prédit via la formule (3) ;

c) Calculer la contribution $\delta_k^{sortie(p)}$ de l'échantillon p à l'erreur commise en sortie ($k = 0, \dots, N_{sortie} - 1$), via la formule (6), puis la contribution $\delta_j^{cachée(p)}$ de l'échantillon p à l'erreur commise par la couche cachée ($j = 1, \dots, N_{cachée} - 1$), via la formule (8) ;

d) Calculer les contributions $\partial E^{(p)}/\partial w_{ij}$ et $\partial E^{(p)}/\partial z_{jk}$ de l'échantillon p au gradient ($i = 0, \dots, N_{entrée} - 1$; $j = 0, \dots, N_{cachée} - 1$; $k = 0, \dots, N_{sortie} - 1$), via les formules (11) et (12) ;

e) Ajouter aux corrections globales Δw_{ij} et Δz_{jk} les contributions $\Delta w_{ij}^{(p)} = -(\alpha/P) \times \left(\partial E^{(p)}/\partial w_{ij}\right)$ et $\Delta z_{jk}^{(p)} = -(\alpha/P) \times \left(\partial E^{(p)}/\partial z_{jk}\right)$ de l'échantillon d'indice p ; ceci permet d'obtenir, une fois l'échantillon p traité, la somme cumulée (sur les échantillons $0, \dots, p$) des corrections à effectuer ; il est alors inutile de conserver en mémoire les valeurs intermédiaires calculées dans les étapes précédentes.

La troisième étape consiste à calculer la norme des corrections globales moyennes $\|\Delta w_{ij}\|$ et $\|\Delta z_{jk}\|$, pour estimer s'il est nécessaire ou non de procéder à une nouvelle itération :

$$\|\Delta w_{ij}\| = \sqrt{\sum_{i,j} (w_{ij})^2} \quad et \quad \|\Delta z_{jk}\| = \sqrt{\sum_{j,k} (z_{jk})^2} \qquad (17a/17b)$$

Si ces normes sont inférieures à une certaine valeur seuil, que l'on s'est donnée initialement, alors on considère avoir atteint un niveau de précision suffisant. Sinon, on procède à une nouvelle itération.