Workshop réseaux de neurones: reconnaissance automatique de chiffres manuscrits

O. Bernard, F. Varray

19 mars 2018

L'objectif de ce projet est de vous familiariser avec les concepts fondamentaux des méthodes dites par réseaux de neurones au travers de l'étude d'une application phare des années 1990 : la reconnaissance automatique de chiffres manuscrits (digit). Le contexte mathématique est introduit dans cet énoncé et différentes implémentations sous le langage de programmation `Matlab` sont à faire afin de répondre aux questions.

1 Introduction

Le problème de reconnaissance automatique des chiffres tracés à la main a été grandement résolu dans les années 1990 par le français Yann LeCun qui à l'époque avait proposé une méthode révolutionnaire appelée les réseaux de neurones convolutionels profonds afin d'apporter une solution fiable et extremement robuste à ce problème. Yann LeCun est actuellement directeur du département intelligence artificelle chez Facebook. De nombreuses videos traitant ce sujet sont accessibles sur internet [1]

2 Les réseaux de neurones

2.1 Apprentissage par l'exemple

La méthode dite par réseaux de neurones fait partie des techniques d'apprentissage par ordinateur (en anglais *machine learning*). La plupart des méthodes d'apprentissage par ordinateur sont basées sur deux phases : une étape d'apprentissage et une étape de test. La première étape nécessite la mise en place d'une base de données à partir de laquelle l'algorithme va apprendre à reproduire une tâche spécifique (ici la reconnaissance automatique de chiffres manuscripts). La seconde étape a pour but d'évaluer la précision de la méthode à partir de données non apprises par l'algorithme. Dans le cadre de ce projet, la base de données utilisée est inspirée d'une base de données publique (`http://yann.lecun.com/exdb/mnist/`) nommée mnist (Mixed National Institute of Standards and Technology) largement utilisée au sein de la communauté scientifique internationale afin de continuer à améliorer (si besoin est) la performance des méthodes de reconnaissance de chiffres manuscripts. Un exemple de chiffres extraits de cette base de données est fourni dans la figure 1.

1. `https://www.youtube.com/watch?v=FwFduRA_L6Q`

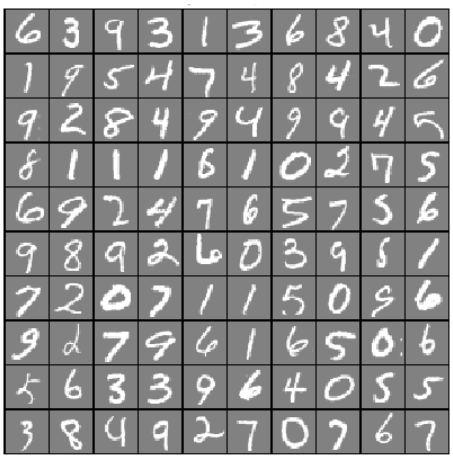
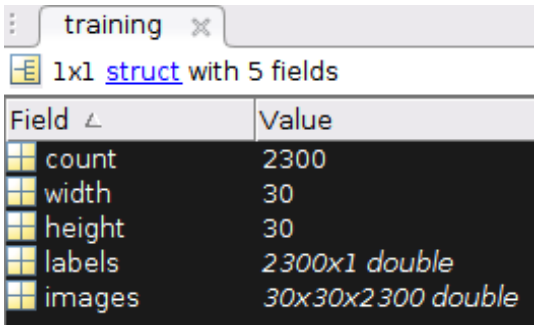
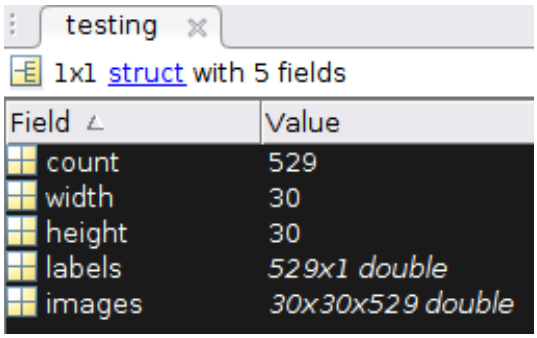


FIGURE 1 – Exemple de chiffres manuscrits extraits de la base de données mnist



(a) Variable `training`



(b) Variable `testing`

FIGURE 2 – Organisation interne des deux variables `Matlab` de type structure `training` et `testing`

A partir du code fourni dans le projet, la base de données est téléchargée automatiquement et placée dans le repertoire *data* à la racine du projet. Une fois téléchargée, l'instruction `load` de `Matlab` permet de lire la base de données et les variables `training` et `testing` sont chargées en mémoire. Ces deux variables sont des structures qui permettent d'accéder à des sous-variables associées. La figure 2 permet d'avoir un aperçu de l'organisation interne de telles structures. A partir de cette figure, nous pouvons voir que la base d'entrainement est constituée de 2300 imagettes (cette information est accessible via l'instruction `training.count`). Chaque chiffre est stocké sous forme d'une imagette de taille 30×30 dont les valeurs sont comprises entre 0 et 1. L'ensemble des imagettes est stocké au sein de la variable `training.images`. A partir de cette variable, il est possible de récupérer la $i^{\text{ème}}$ imagette via la commande `Matlab` suivante : `img = training.images(:,:,i)`. A chaque imagette de la variable `training.images` est associé le chiffre correspondant via la variable `training.label`. Par exemple, si la $i^{\text{ème}}$ imagette de la variable `training.images` correspond au chiffre 3, l'instruction `training.labels(i)` retournera la valeur 3.

2.2 Fonction de décision

La plupart des méthodes d'apprentissage par ordinateur correspondent à des fonctions de décision $h_\Theta(x)$, où Θ est une matrice dont les coefficients sont les inconnues à déterminer et x correspond à un vecteur de données que l'on fournit en entrée de l'algorithme. Dans le cadre de ce projet, la méthode de réseau de neurones renvoie un vecteur de probabilité appartenant à $\mathbb{R}^{10\times 1}$ où chaque composante correspond à la probabilité (valeur comprise entre 0 et 1) de détection d'un unique

chiffre ($1^{\text{ère}}$ composante : probabilité de détection du chiffre 1, $2^{\text{ème}}$ composante : probabilité de détection du chiffre 2, $\cdots$, $10^{\text{ème}}$ composante : probabilité de détection du chiffre 0). Ainsi, chaque chiffre correspond à une classe k et la fonction de décision peut être modélisée de la façon suivante :

$$(h_\Theta(x))_k = p(y = k|x; \Theta)\,, \tag{1}$$

où le second membre de cette équation correspond à la probabilité que la sortie y appartienne à la classe k connaissant le vecteur d'entrée x et la paramétrisation Θ.

2.3 Phase d'apprentissage

Lors de la phase d'apprentissage, l'ensemble des m imagettes connues (ici $m = 2115$) de taille 30×30 est utilisé. Chaque imagette est représentée sous forme d'un vecteur de taille $[1 \times n]$ (avec $n = 900$) composé des valeurs des pixels de chaque ligne de l'imagette mise bout-à-bout. Ainsi, au début du code fourni dans le projet, l'ensemble des données d'apprentissage est stocké au sein d'une matrice X de taille $[m \times n]$. En parallèle de la matrice X, un vecteur y de taille $[m \times 1]$ est créé. Chaque élément de y possède la valeur de la classe de l'imagette correspondante. La classe 1 correspond au chiffre 1, la classe 2 au chiffre 2, $\cdots$, la classe 9 au chiffre 9 et la classe 10 au chiffre 0. Ainsi, si la $i^{\text{ème}}$ imagette de la base d'entrainement correspond au chiffre 4, alors la $i^{\text{ème}}$ ligne de la matrice X correspond aux valeurs des pixels de l'imagette et le $i^{\text{ème}}$ élément du vecteur y, nommé $y^{(i)}$ dans la suite de l'énoncé, sera égale à 4.

Le vecteur de paramètres Θ contient les inconnues à déterminer. Ceci est réalisé lors de la phase d'apprentissage à partir d'une mise en équation qui est développée dans la section 4.

2.4 Phase de test

Une fois le vecteur de paramètres Θ connu, la phase de test s'effectue de la façon suivante. Pour une nouvelle imagette, le vecteur x associé de taille $[1 \times n]$ est créé (mise bout-à-bout des lignes de l'imagette). Dans un second temps, la fonction de décision $h_\Theta(x)$ est calculée. Cette fonction renvoie un vecteur de taille $[10 \times 1]$ où chaque composante k correspond à la probabilité d'appartenance à la classe k (*cf.* équation 1). Classiquement, la probabilité la plus importante permet de déterminer la valeur du chiffre de l'imagette. La vraie valeur manuscrite peut être alors utilisée afin de mesurer la fiabilité de l'approche.

3 Modélisation de la fonction de décision des réseaux de neurones

Les réseaux de neurones ont la particularité d'exploiter des non-linéarités afin de représenter la fonction de décision. Cette non-linéarité se justifie par la volonter de modéliser des fonctions de décision complexes adaptées à l'analyse de données de très grande taille.

3.1 L'unité logique

L'élément de base qui constitue un réseau de neurones correspond à l'unité logique dont un schéma explicatif est fourni dans la figure 3.

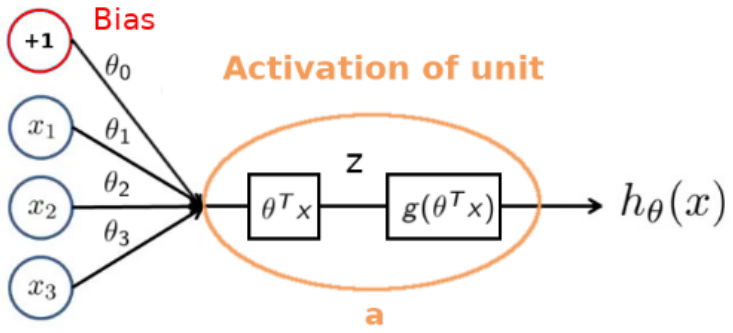


FIGURE 3 – Illustration du fonctionnement d'une unité logique

A partir de la représentation de la figure 3, $x = [1, x_1, x_2, x_3]^T$ correspond au vecteur de données (dont la valeur 1 rajoutée en début de vecteur modélise un biais), $\Theta = [\theta_0, \theta_1, \theta_2, \theta_3]$ correspond au vecteur de paramètres, $h_\Theta(x) = g(z) = g\left(\Theta^T x\right)$ correspond à la fonction de décision avec $g(\cdot)$ une fonction d'activation pré-définie. De nombreuses fonctions d'activation existent dans la littérature parmis lesquels les plus connues sont la fonction sigmoïde et la fonction RELU. Dans ce projet, nous allons utiliser la fonction d'activation sigmoïde dont l'expression est $g(z) = \frac{1}{1+e^{-z}}$.

3.2 Les réseaux de neurones : ensemble d'unités logiques

L'utilisation d'une seule unité logique afin de représenter une fonction de décision complexe n'est bien évidement pas suffisant. Il apparait donc naturel de combiner un ensemble d'unités logiques afin de complexifier la fonction de décision sous-jacente, ce qui correspond à la famille des réseaux de neurones. La figure 4 fournit un exemple d'architecture de réseau de neurones (généralement les différents biais n'apparaissent pas dans les schémas d'architecture même s'ils sont toujours présents). Le terme architecture correspond à la façon dont sont combinées les unités logiques entre elles.

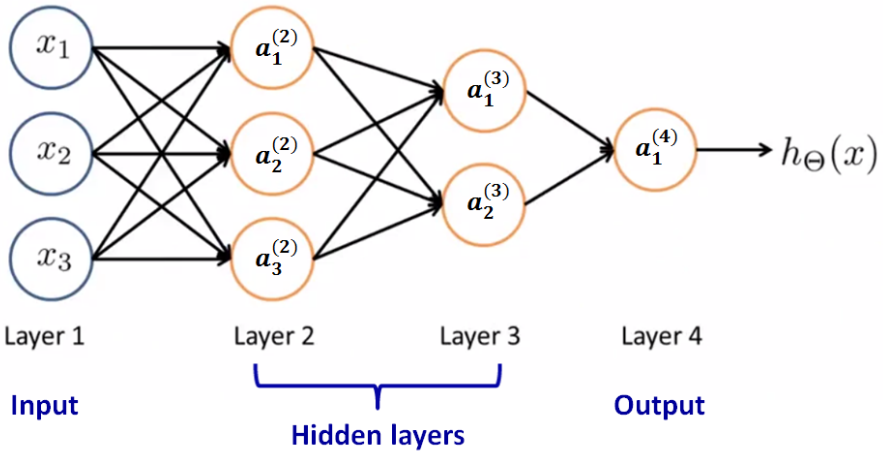


FIGURE 4 – Illustration de l'architecture d'un réseau de neurones

Un réseau de neurones est caractérisé par

- le nombre L de couches utilisées ($L = 4$ dans l'exemple de la figure 4)
- le nombre s_l d'unités logiques déployées par couche l ($s_1 = 3$, $s_2 = 3$, $s_3 = 2$ et $s_4 = 1$ dans l'exemple de la figure 4)
- le nombre K d'unités logiques de sortie ($K = s_4 = 1$ dans l'exemple de la figure 4)

D'un point de vu sémantique, la première couche correspond aux données d'entrée, la dernière couche correspond aux données de sortie (fonction de décision $h_\Theta(x)$) et les couches entre l'entrée et la sortie sont appelées les couches cachées. Dans l'exemple de la figure 4, étant donné que chaque unité logique est connectée à l'ensemble des unités de la couche précédente, on dit que le réseau est pleinement connecté (en anglais on parle de ***fully connected network***).

Dans la représentation des réseaux de neurones, à chaque flèche qui relie une unité logique à une autre correspond un poids (paramètre) à déterminer. Il est d'usage d'introduire la notion de matrice $\Theta^{(l)}$ qui regroupe l'ensemble des poids qui permettent de passer de la couche l à la couche $l+1$. Un exemple de défintion d'une telle matrice est fourni dans la figure 5.

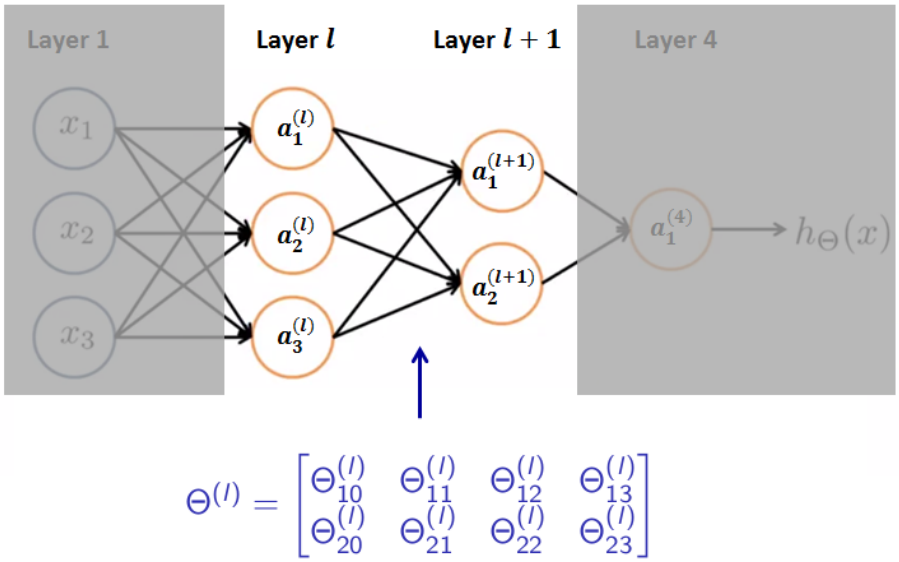


FIGURE 5 – Illustration de la définition de matrice $\Theta^{(l)}$ modélisant le passage de la couche l à la couche $l+1$.

A partir de l'exemple de la figure 5, le passage de la couche l à la couche $l+1$ peut se modéliser de la façon suivante :

$$a^{(l+1)} = g\left(z^{(l+1)}\right) = g\left(\Theta^{(l)}\, a^{(l)}\right),$$

avec

$$a^{(l+1)} = \begin{bmatrix} a_1^{(l+1)} \\ a_2^{(l+1)} \end{bmatrix}, \; z^{(l+1)} = \begin{bmatrix} z_1^{(l+1)} \\ z_2^{(l+1)} \end{bmatrix}, \; a^{(l)} = \begin{bmatrix} 1 \\ a_1^{(l)} \\ a_2^{(l)} \\ a_3^{(l)} \end{bmatrix}$$

et

$$\begin{cases} a_1^{(l+1)} &= g\left(\Theta_{10}^{(l)} + \Theta_{11}^{(l)} a_1^{(l)} + \Theta_{12}^{(l)} a_2^{(l)} + \Theta_{13}^{(l)} a_3^{(l)}\right) \\ a_2^{(l+1)} &= g\left(\Theta_{20}^{(l)} + \Theta_{21}^{(l)} a_1^{(l)} + \Theta_{22}^{(l)} a_2^{(l)} + \Theta_{23}^{(l)} a_3^{(l)}\right) \end{cases}$$

Il est à noter que

- $a^{(1)} = x$
- $a_j^{(l)}$ correspond à l'activation de l'unité j de la couche l
- dans le cas des réseaux pleinement connectés, si le réseau possède s_l unités pour la couche l, s_{l+1} unités pour la couche $l+1$, alors $\Theta^{(l)}$ sera de dimension $s_{l+1} \times (s_l + 1)$
- le réseau de la figure 4 est entièrement caractérisé par les trois matrices $\Theta^{(1)} \in \mathbb{R}^{3\times 4}$, $\Theta^{(2)} \in \mathbb{R}^{2\times 4}$ et $\Theta^{(3)} \in \mathbb{R}^{1\times 3}$

3.3 Algorithme de propagation avant

Une fois les matrices $\{\Theta^{(l)}\}_{l\in[1,L-1]}$ (caractérisant un réseau) définies, pour tout nouvel échantillon de donnée x, une décision associée sera dérivée au travers de l'algorithme suivant (appelé ***algorithme de propagation avant***) :

1) $z^{(2)} = \Theta^{(1)}x \quad \rightarrow \quad a^{(2)} = g(z^{(2)})$
   (avec $z^{(2)}$ et $a^{(2)} \in \mathbb{R}^{3\times 1}$ dans l'exemple de la figure 4)
2) Ajout de $a_0^{(2)} = 1$ au vecteur $a^{(2)} \quad \rightarrow \quad a^{(2)} \in \mathbb{R}^{4\times 1}$
3) $z^{(3)} = \Theta^{(2)}a^{(2)} \quad \rightarrow \quad a^{(3)} = g(z^{(3)})$
   (avec $z^{(3)}$ et $a^{(3)} \in \mathbb{R}^{2\times 1}$ dans l'exemple de la figure 4)
4) Ajout de $a_0^{(3)} = 1$ au vecteur $a^{(3)} \quad \rightarrow \quad a^{(3)} \in \mathbb{R}^{3\times 1}$
5) $z^{(4)} = \Theta^{(3)}a^{(3)} \quad \rightarrow \quad h_\Theta(x) = a^{(4)} = g(z^{(4)})$

4 Comment faire apprendre un réseau de neurones ?

4.1 Formulation énergétique

Le but de la formulation énergétique est de permettre d'estimer une fonction de décision (au travers de la détermination des matrices $\{\Theta^{(l)}\}_{l\in[1,L-1]}$) représentant le plus fidèlement possible les données d'entrainement, c'est à dire quelque soit la donnée $x^{(i)}$ issues de la base de données, on souhaite que $h_\Theta(x^{(i)}) = y^{(i)}$. Il existe plusieurs formulation énergétique dans la littérature, parmis les plus connues étant la fonction ***d'entropie mutuelle*** dont l'équation est donnée ci-dessous :

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)}\,\log\left((h_\Theta(x^{(i)}))_k\right) + \left(1-y_k^{(i)}\right)\log\left(1-(h_\Theta(x^{(i)}))_k\right)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{q=1}^{s_l}\sum_{p=1}^{s_{l+1}}\left(\Theta_{pq}^{(l)}\right)^2 \tag{2}$$

où

- K représente le nombre de sorties du réseau de neurones. Dans le cadre de notre projet, K correspond au nombre de chiffres à reconnaitre, c'est à dire $K = 10$;
- m correspond au nombre de données en entrée ;
- $\Theta_{pq}^{(l)}$ correspond au poids entre la $q^{\text{ème}}$ unité de la couche l et la $p^{\text{ème}}$ unité de la couche $l+1$;
- λ correspond à un hyperparamètre de pondération du terme de régularisation (2^{nd} terme de l'équation 2) ;
- $(h_\Theta(x^{(i)}))_k$ correspond à la $k^{\text{ème}}$ composante du vecteur de fonction de décision ayant $x^{(i)}$ pour entrée ;
- $y_k^{(i)}$ est un scalaire défini par :

$$y_k^{(i)} = \begin{cases} 1 & \text{si } y^{(i)} = k \\ 0 & \text{sinon,} \end{cases} \tag{3}$$

où $y^{(i)}$ correspond à la $i^{\text{ème}}$ case du vecteur y. En d'autres termes, les scalaire $y_c^{(i)}$ s'identifie à 1 si la $i^{\text{ème}}$ imagette est associée au chiffre k et 0 si ce n'est pas le cas.

Etant donné que chaque composante du vecteur de décision $h_\Theta(\cdot)$ appartient à l'intervalle $[0, 1]$, la fonction d'énergie $J(\Theta)$ est positive et atteint sa valeur minimale égale à 0 s'il existe un ensemble de matrices Θ tel que $(h_\Theta(x^{(i)}))_k = 1$ si $y^{(i)} = k$ et $(h_\Theta(x^{(i)}))_k = 0$ si $y^{(i)} \neq k$. En partique, ce cas idéal arrive rarement, cependant l'idée est bien d'optimiser un ensemble de matrices Θ de telle sorte que :

- la valeur de $(h_\Theta(x^{(i)}))_k$ soit proche de 1 si $y^{(i)} = k$
- la valeur de $(h_\Theta(x^{(i)}))_k$ soit proche de 0 si $y^{(i)} \neq k$.

Ainsi, le vecteur de fonction de décision sous-jacent $h_\Theta(\cdot)$ permettra de reconnaître au mieux l'ensemble des imagettes $\{x^{(i)}\}_{i\in[1,m]}$ associées aux chiffres $\{y^{(i)}\}_{i\in[1,m]}$.

4.2 Minimisation de la fonction d'énergie $J(\Theta)$ par descente de gradient

La minimisation de la fonction d'énergie dans le cadre des réseaux de neurones est fondamentale et nécessite le recours à des outils d'analyse fonctionnelle avancés. Aussi la compréhension détaillée de cette partie n'est pas requise pour réaliser le projet dans son ensemble. Néanmoins, il nous est apparu important dans vous fournir des étapes clées afin d'acquérir une compréhension globale des méthodes par réseaux de neurones.

La fonction d'énergie $J(\Theta)$ est relativement complexe et sa minimisation s'effectue généralement en utilisant un algorithme itératif de descente de gradient optimisé. Dans le cadre de ce projet, nous utiliserons une méthode dérivée de l'algorithme du ***gradient conjugué de type Polack-Ribiere***. Ce type de méthode est particulièrement bien adapté pour la minimisation de fonctions continues et différentiables à plusieurs variables mais nécessite de pouvoir calculer la valeur de la fonction ainsi que la valeur de ses dérivées en tout point de l'espace. Dans le cadre des réseaux de neurones, il est donc nécessaire de savoir calculer la dérivée de l'énergie $J(\Theta)$ en fonction des paramètres du réseau, *i.e.* savoir calculer l'expression suivante

$$\frac{\partial}{\partial \Theta_{pq}^{(l)}} J(\Theta) \tag{4}$$

4.3 Algortihme de rétropropagation

L'une des méthodes les plus connues et utilisée en réseau de neurones afin d'estimer la dérivée de la fonction d'énergie en fonction des paramètres du réseau s'appelle la méthode de la rétropropagation (en anglais on parle de ***back propagation***). Partant d'un couple de données $(x^{(i)}, y^{(i)})$, cette méthode se base sur la modélisation de l'erreur faite à la sortie de chaque couche via les expressions suivantes :

$$\begin{cases} \delta^{(L)} = a^{(L)} - Y^{(i)} & (\in \mathbb{R}^{s_L \times 1}) \\ \delta^{(L-1)} = \left(\Theta^{(L-1)}\right)^T \delta^{(L)} .* g'\left(z^{(L-1)}\right) & (\in \mathbb{R}^{s_{L-1} \times 1}) \\ \quad \vdots & \\ \delta^{(l)} = \left(\Theta^{(l)}\right)^T \delta^{(l+1)} .* g'\left(z^{(l)}\right) & (\in \mathbb{R}^{s_l \times 1}) \\ \quad \vdots & \\ \delta^{(2)} = \left(\Theta^{(2)}\right)^T \delta^{(3)} .* g'\left(z^{(2)}\right) & (\in \mathbb{R}^{s_2 \times 1}) \end{cases} \tag{5}$$

où $g'(\cdot)$ correspond à la dérivée de la fonction sigmoïde et $.*$ représente l'opérateur de multiplication élément par élément. La première égalité de l'équation 5 est triviale : l'erreur réalisée à la dernière couche est obtenue par soustraction du vecteur de décision finale, *i.e.* $a^{(L)} \in \mathbb{R}^{10\times 1}$, avec le vecteur

correspondant au chiffre réel, *i.e.* $Y^{(i)} \in \mathbb{R}^{10\times 1}$ ayant pour $k^{\text{ème}}$ composante $y_k^{(i)}$ (*cf.* équation 3). L'obtention des autres égalités de l'équation 5 ne fait pas partie du cadre de ce projet. Il est cependant interessant d'observer que l'erreur obtenue à la couche l se calculera à partir de l'erreur de la couche $l+1$ (d'où le terme algorithme de rétropropagation). En ignorant dans un premier temps le terme de régularisation, il est possible de montrer que la dérivée de la fonction d'énergie $J(\Theta)$ en fonction des paramètres $\Theta_{pq}^{(l)}$ du réseau peut s'exprimer de la façon suivante :

$$\frac{\partial}{\partial \Theta_{pq}^{(l)}} J(\Theta) = a_q^{(l)} \delta_p^{(l+1)} \tag{6}$$

où $\delta_p^{(l+1)}$ correspond à l'erreur de l'unité p de la couche $l+1$ et $a_q^{(l)}$ correspond à l'activation de l'unité q de la couche l. Ainsi, à partir des équations 5 et 6, l'algorithme décrit ci-dessous est génaralement appliqué afin de calculer le gradient de la fonction d'énergie sur l'ensemble du réseau.

1) Initialisation de $\Delta_{pq}^{(l)} = 0$ (pour tout l, p, q)

2) Pour $i = 1$ à m

   Application de l'algortihme de propagation avant afin de calculer $a^{(l)}$ pour $l = 2, \cdots, L$

   A partir de $y^{(i)}$, calcul de $\delta^{(L)} = a^{(L)} - Y^{(i)}$

   Calcul de $\delta^{(L-1)}, \delta^{(L-2)}, \cdots, \delta^{(2)}$

   Calcul de $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} \left(a^{(l)}\right)^T \in \mathbb{R}^{s_{l+1}\times s_l+1}$ pour $l = 1, \cdots, L-1$

3) A partir des valeurs de $\Delta^{(l)}$ calcul des dérivées via l'expression suivante

   $\frac{\partial}{\partial \Theta_{pq}^{(l)}} J(\Theta) := \frac{1}{m}\Delta_{pq}^{(l)} + \frac{\lambda}{m}\Theta_{pq}^{(l)}$ si $q \neq 0$

   $\frac{\partial}{\partial \Theta_{pq}^{(l)}} J(\Theta) := \frac{1}{m}\Delta_{pq}^{(l)}$ si $q = 0$

4.4 Algorithme d'apprentissage final

L'algorithme décrit à la page suivante est finalement appliqué afin d'apprendre une fonction de décision permettant de représenter le plus fidèlement possible les données d'entainement $\{x^{(i)}, y^{(i)}\}_{i\in[1,m]}$.

1. Initialisation aléatoire des poids du réseau (valeurs aléatoires comprises entre $[-\epsilon, \epsilon]$)

2. Application de l'algortihme de propagation avant afin d'obtenir $h_\Theta(x^{(i)})$ pour tout $x^{(i)}$

3. Implementation du code permettant de calculer la fonction d'énergie $J(\Theta)$

4. Implementation du code de rétropropagation permettant de calculer $\frac{\partial}{\partial \Theta_{pq}^{(l)}} J(\Theta)$

   Pour $i = 1$ à m {

   Application de la propagation avant et de la rétropropagation à partir de la donnée $(x^{(i)}, y^{(i)})$ afin d'obtenir $a^{(l)}$ et $\delta^{(l)}$ pour $l = 2, \cdots, L$

   Calcul de $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)}(a^{(l)})^T$ pour $l = 1, \cdots, L-1$

   }

   Calcul de $\frac{\partial}{\partial \Theta_{pq}^{(l)}} J(\Theta)$

5. Utilisation d'une méthode de descente de gradient avancée afin de minimiser $J(\Theta)$ vis-à-vis des paramètres $\Theta_{pq}^{(l)}$

5 Liste des questions liées au projet

Dans le cadre de ce projet, le réseau de neurones dont l'architecture est décrite dans la figure 6 est mis en place afin de reconnaitre des chiffres manuscrits. Ce réseau comporte 3 couches ($s_1 = 900$, $s_2 = 25$ et $s_3 = 10$) et est pleinement connecté.

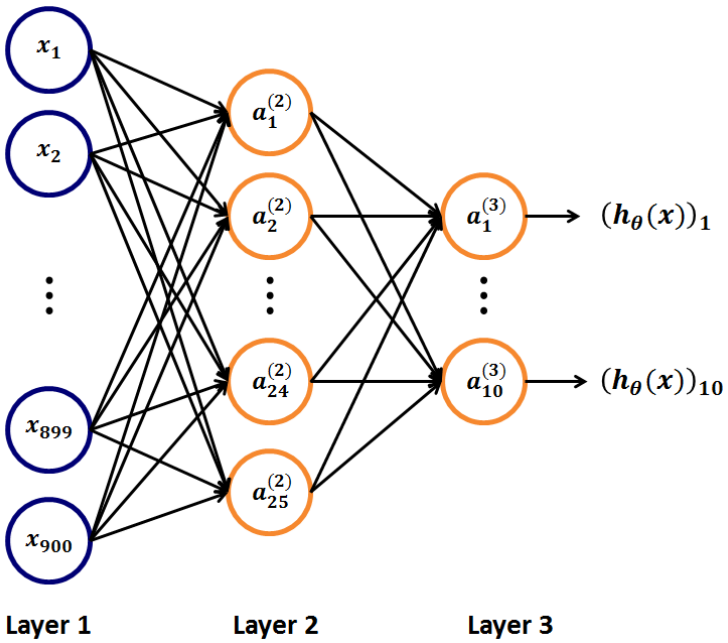


FIGURE 6 – Illustration de l'architecture du réseau étudié dans ce projet afin de reconnaitre des chiffres manuscrits.

5.1 Modélisation mathématique

On se place dans le cas où le réseau de neurones de la figure 6 a été correctement entrainé afin de reconnaitre des chiffres manuscrits.

1. Combien de matrices $\Theta^{(l)}$ doivent être apprises afin de caractériser entièrement le réseau de neurones de la figure 6 ? Quelles sont leur tailles ?
2. Combien de fonctions de décision sont modélisées en sortie du réseau de neurones de la figure 6 ? Que modélise la fonction de décision $(h_\Theta(x))_{10}$
3. A quoi sert le terme de régularisation de la fonction d'énergie $J(\Theta)$ (2^{nd} terme de l'équation 2) ? Le but étant de minimiser la fonction d'énergie $J(\Theta)$, quel est l'impact de ce terme sur l'évolution de la valeur des paramtètres $\Theta_{pq}^{(l)}$ lors de l'application de l'algorithme de descente de gradient ?
4. Quelle métrique proposez-vous de mettre en place afin d'évaluer la précision de votre réseau de neurones vis-à-vis de la base de test ?
5. Lors de la phase de test, l'imagette fournit en entrée de la fonction de décision $h_\Theta(x)$ correspond au chiffre manuscrit 3. Dans l'hypothèse où cette imagette est correctement détectée, quelle composante du vecteur $h_\Theta(x)$ possèdera la plus grande valeur ? Est ce que cette valeur sera forcément égale à 1 ?

5.2 Implémentation `Matlab`

Vous devez télécharger le projet `Matlab` via le lien suivant `https://www.creatis.insa-lyon.fr/~bernard/workshop/code.zip`. Le projet est structuré de la façon suivante :

`data`	Dossier utilisé pour stocker la base de données mnist modifiée
[∗] `exercise`	Dossier contenant des scripts pour la prise en main du formalisme mathématique
[∗] `+nn`	Dossier contenant les fichiers relatifs à la méthode de réseau de neurones
`param`	Dossier utilisé pour stocker les matrices $\{\Theta^{(j)}\}$ sous forme d'un fichier '.mat'
[∗] `scripts`	Dossier contenant les scripts principaux du projet
`+tools`	Dossier contenant des fonctions d'usage générale
`+visu`	Dossier contenant des fonctions utilisées pour des aspects de visualisation avancés

∗ indique que le dossier contient des fichiers à modifier.

Il est a noté que les fonctions présentent dans un dossier dont le nom commence par le symbole `+` sont utilisable au travers du nom du dossier associé. Par exemple, la fonction `sigmoid` présente dans le dossier `+nn` peut être appelée au sein d'un script afin de calculer la valeur de la fonction sigmoïde pour une donnée z passée en argument d'entrée via l'instruction suivante :

```
val = nn.sigmoid(z)
```

Le but de cette partie est de compendre dans son ensemble l'implémentation du réseau de neurones présenté dans la figure 6. Afin de vous rendre compte de ses performances, vous pouvez dans un premier temps jouer avec l'interface gui correspondante. Pour ce faire, il vous suffit de lancer via `Matlab` le script `scriptGuiNN.m` présent dans le dossier `scripts`.

1. Compléter le code du script `script_ex1.m` présent dans le dossier `exercise` afin d'estimer le vecteur de fonction de décision pour le premier chiffre issu de la base de données d'entrainement et d'un jeu de paramètres $\Theta^{(1)}$ et $\Theta^{(2)}$ pré-calculés. Si vous avez bien programmé le calcul du vecteur de décision, le produit scalaire $h_\Theta(x^{(1)}) \cdot h_\Theta(x^{(1)})^T$ doit vous retourner la valeur 0.9897 pour le cas étudié. Est ce que les valeurs obtenues pour le vecteur de décision $h_\Theta(x^{(1)})$ sont en accord avec le chiffre réel ?

2. Compléter le code du script `script_ex2.m` présent dans le dossier `exercise` afin de calculer la valeur de l'energie $J(\Theta)$ (équation 2) à partir de la base de données d'entrainement et d'un jeu de paramètres $\Theta^{(1)}$ et $\Theta^{(2)}$ pré-calculés. Vous programmerez la fonction d'énergie tout d'abord sans puis avec le terme de régularisation. Si vous avez bien codé le calcul de l'énergie, vous devez obtenir la valeur 0.065407 sans terme de régularisation et la valeur 0.24002 avec le terme de régularisation pour le cas étudié.

3. Compléter le code du script `script_ex3.m` présent dans le dossier `exercise` afin de calculer la dérivée de la fonction d'énergie $J(\Theta)$ en fonction des paramètres $\Theta_{pq}^{(l)}$ (*cf.* équation(6)), et ce, pour l'ensemble des paramètres du réseau. Pour ce faire, vous devrez implémenter les différentes étapes de l'algorithme fourni à la page 8. Si vous avez bien programmé le calcul du gradient, l'exécution des trois lignes ci-dessous doit vous retourner la valeur $4.3053e-05$ pour le cas étudié.

```
grad = [Theta1_grad(:);Theta2_grad(:)]';        %– Unroll gradients
val = (grad*grad');                             %– Apply scalar product
disp(['Scalar product = ',num2str(val)])        %– display result
```

   Les questions 1) 2) et 3) peuvent bloquer certains d'entre vous, notamment ceux qui ne maitrisent pas suffisament le langage `Matlab`. Afin de vous permettre d'avancer malgré tout, sachez que le code que vous devez réaliser lors de ces trois questions est présent dans la fonction `costFunction.m` présente dans le dossier `+nn`. Cependant, nous vous invitons à jouer le jeu en essayant au maximum de répondre aux questions par vous même afin de progresser au travers des différentes questions.

4. Exécuter et analyser les deux fichiers suivants présents dans le dossier `scripts` dans l'ordre suivant :

`scriptTrainingNN.m`	script qui permet d'exécuter la phase d'entrainement
`scriptTestingNN.m`	script qui permet d'exécuter la phase de tests

   Analyser, entre autre, où et comment a été programmé l'algorithme finale décrit à la page 9.

5. Etudier l'influence du choix de la méthode de descente de gradient (paramètre `gradient_descent_scheme` dans le fichier `scriptTrainingNN.m`) vis-à-vis de la performance de la méthode estimée à partir de la ***base de données de test***.

6. Coder un nouveau script nommé `scriptInfluenceMaxIteration.m` au sein du dossier `scripts` afin de tracer sur un même graphique l'évolution de la mesure de précision calculée à partir de la ***base de données de test*** pour un nombre d'iterations variant de 20 à 200 avec 10 points de mesures (on gardera les valeurs par défaut des autres paramètres pour l'ensemble des experiences) lors de la phase d'apprentissage à partir de la ***base de données d'entraintement***. Commenter les résultats ainsi obtenus et conclure sur la valeur du nombre d'itérations à retenir.

7. Lors de l'exécution du code correspondant à l'interface gui, vous vous renderez compte que de temps en temps, l'estimation du chiffre manuscrit est érronée. Quelles pistes d'amélioration proposez-vous ?