épreuve d'informatique **module LI332** **Université Pierre et Marie Curie**

Examen final du 5 janvier 2015

Documents autorisés - Engins électroniques interdits

Problème - Réseaux de neurones en OCaml (16pts)

Les réseaux de neurones sont une structure d'intelligence artificielle tentant d'imiter le comportement des neurones biologiques. Chaque neurone est modélisé suivant le modèle de McCulloch et Pitts datant de 1943. Le comportement (très simplifié) d'un neurone est le suivant, il reçoit un ensemble de signaux entrants et si la somme de ces signaux est suffisament forte (supérieure à un seuil fixé), il émet un signal en sortie (1), sinon il n'émet rien (0). Ce fonctionnement peut donc être modélisé par la formule suivante et est présenté dans la figure 1

$$\sum_{i \in in_x} w_i.x_i \geq T$$

Avec x_i le signal de l'entrée i, w_i des poids affectés à chaque entrée et T le seuil correspondant au neurone.

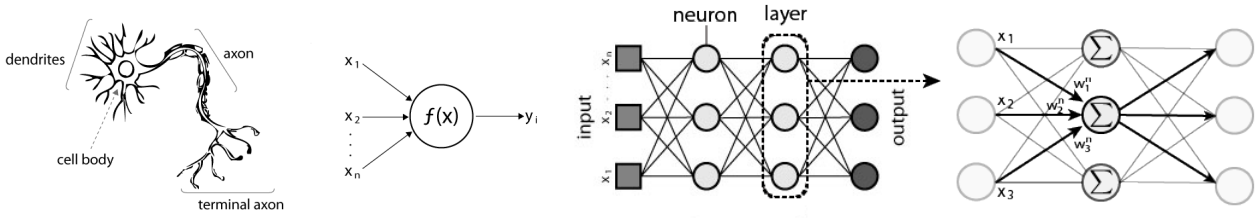

FIGURE 1 – (Gauche) Neurone biologique et modélisation par McCulloch et Pitts (1943). (Droite) Un réseau de neurones est représenté par un treillis de neurones, c'est à dire un ensemble de couches connectées les unes aux autres, chaque couche contenant un ensemble de neurones. Lors de la propagation, chaque neurone reçoit la valeur de tous les neurones de la couche précédente et transmet son résultat à la couche suivante.

Q.1 **(3pts)** On souhaite écrire la fonction `neuronReaction : float list -> float list -> float -> boolean` prenant dans l'ordre la liste des signaux, la liste des poids et le seuil et retournant 1.0 ou 0.0. Si les tailles des listes sont inconsistantes, la fonction devra lever l'exception `NeuronIncoherentInput`.

1. Ecrire cette fonction en style fonctionnel (récursion et matching)
2. Ecrire cette fonction en utilisant des fonctions du module `List`
3. Ecrire cette fonction en style impératif (boucles)

Un neurone seul ne permet pas d'apprentissage très poussé. On souhaite ainsi développer un module complet de réseaux de neurones. Dans un réseau multicouche, chaque neurone d'une couche reçoit en entrée le signal de *tous* les neurones de la couche précédente (chaque lien ayant son propre poids) et calcule sa sortie qu'il envoie à tous les neurones de la couche suivante. Lorsque l'on donne le vecteur $X = (x_1, x_2, ..., x_n)$ à la première couche, l'opération de la question 1 est appliquée à chaque neurone, couche par couche, jusqu'à la dernière couche produisant le vecteur $S = (s_1, ..., s_p)$ (sortie de chaque neurone). Cette opération est appellée *propagation*. On considère qu'on dispose d'un module `Valeur` qui permet d'encapsuler différents types (entiers, flottants ou autre), qui suit la signature suivante

```
module type Valeur =
sig
  type t
  val ( + ) : t -> t -> t
  val ( * ) : t -> t -> t
  val ( - ) : t -> t -> t
  val ( / ) : t -> t -> t
  val zero : t
  val one : t
end
```

Q.2 **(4pts)** On va commencer par écrire les modules `Vecteur` et `Matrice`, *cette fois en utilisant les types Array d'OCaml* et qui permettront de représenter les poids et les entrées/sorties du système.

1. Ecrire le module `Vecteur` **fonctorisé par une** `Valeur` (attention au type) et contenant les fonctions

(a) `init n v` permettant d'initialiser un vecteur de `n` entrées contenant la valeur `v`
(b) `dim x` permettant de rendre la taille du vecteur
(c) `neuron a b` permettant d'effectuer la somme de la multiplication des vecteurs a et b

2. Ecrire le module `Matrice` contenant un tableau de vecteurs **(toujours fonctorisé par une `Valeur`)** contenant
(a) `init n p` permettant d'initialiser une matrice de `n` lignes et `p` colonnes.
(b) `identity n` permettant d'initialiser une matrice identité de taille `n`.
(c) `transpose m` permettant d'effectuer la transposition de `m` (inversion des positions lignes, colonnes).

Nous allons commencer par représenter une couche du réseau en considérant uniquement les poids w comme étant une `Matrice` de flottants, les seuils T comme étant un `Vecteur` de flottants et la sortie de tous les neurones conservés dans un `Vecteur` de flottants

Q.**3 (3pts)** Implémenter le module `perceptronLayer` en précisant bien les types de données utilisés et contenant

1. `init n r` permettant d'initialiser une couche de `n` neurones connectés à `v` entrées.
2. `eval l e` evaluant le résultat de toute la couche `l` en réponse au vecteur d'entrées `e` (suivant la formule de la première question)

Q.4 **(2pts)** Ecrire la fonction `init N` permettant de créer un réseau complet à N couches (chacune de type `perceptronLayer`). Les poids seront initialisés avec une valeur aléatoire entre 0 et 1.

Q.5 **(2pts)** Ecrire la fonction `propagate` permettant à partir d'un vecteur X donné en entrée de calculer le résultat S de tout le réseau.

Q.5 **(3pts)** On remarque que la fonction `neuronReaction` nous limite à utiliser une seule fonction de seuil. On veut donc permettre à l'utilisateur de choisir sa fonction avec un `type activation = Linear | Sigmoid | Custom of (float->float)`. Ecrire la fonction `threshold` prenant un argument de ce type et qui rend une fonction anonyme permettant soit de renvoyer la même valeur (Linear), soit un calcul de sigmoïde ($1/1+e^{-x}$), soit d'utiliser sa propre fonction (Custom)

Exercice - Calculatrice « infinie » (7 pts)

Par nature, les types numériques d'un ordinateur sont bornés (32 ou 64 bits), ce qui fait que les opérations mathématiques traditionnelles sont limitées. Pour des raisons évidentes, nous voulons tel Chuck Norris être capable de compter jusqu'à l'infini 2 fois. Pour cela nous allons utiliser *des listes d'entiers* pouvant ainsi contenir un nombre de taille quelconque. Les techniques d'addition, soustraction et multiplication seront ainsi tout simplement celles qu'on effectue traditionnellement sur un papier. Pour vous simplifier la tâche nous considèrerons uniquement les entiers positifs, vous n'aurons donc ni à gérer les flottants, ni les problèmes de signes. On definit le type `chuckNumber` (qui ne servira que pour la question 4).

```
chuckNumber = Val of int list
| Add of int list * int list
| Sub of int list * int list
| Mult of int list * int list
```

Q.1 **(3pts)** La fonction `add_infinite` prend en argument deux listes et rend la liste correspondant à l'addition des deux nombres.

1. Ecrire cette fonction en style fonctionnel OCaml
2. Ecrire cette fonction en utilisant une interopérabilité avec C (on ne s'intéressera qu'au code C)

Q.2 (**3pts**) Ecrire la fonction `mult_infinite` prenant en entrée deux chaînes de caractères et effectuant la multiplication des deux arguments.

1. Ecrire cette fonction en style fonctionnel OCaml
2. Ecrire cette fonction en utilisant une interopérabilité avec C (on ne s'intéressera qu'au code C)

Q.3 (**1pt**) Ecrire la fonction `eval_infinite` prenant en entrée un `chuckNumber` et évaluant le résultat **en utilisant les appels d'interopérabilité avec C.**