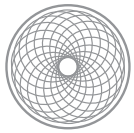


Informatique - Orléans

Réseau de neurone

Groupe :

- BELMEKNASSI Abdelhakim
- HUBERT Simon
- CHAKIR Hatim
- BOULEALF Soufiane

Table des matières

INTRODUCTION 1

Approche 1

ANALYSE DU TRAVAIL 1

Première Question ? 1

Stockage des chiffres manuscrit 1

Fonctionnement du perceptron multicouche 2

Codage et Décodage des Images 3

Codage 3

Décodage 4

Les Commandes 5

Les Packages 5

Test et Courbes 6

MODE D'EMPLOI 7

ANNEXE : UML 8

Pack lecture UMLPack analyseChiffre UML 8

Pack reseauNeurone UML 9

INTRODUCTION

Approche

De nos jours, la reconnaissance est devenu un domaine très important, notamment dans différents domaines :

- Contrôle des véhicule : reconnaissance des plaques d'immatriculation

- Sécurité : reconnaissance des empreinte digitales, visages, paroles ...

Dans ce rapport, nous allons nous concentrer sur la reconnaissance des chiffres manuscrits, grâce à un réseau de neurones artificielles, qui va bien évidemment apprendre avant de pouvoir reconnaitre des chiffres.

ANALYSE DU TRAVAIL

Première Question ?

Dans le sujet, et heureusement, on nous demande pas d'inventer ou de chercher une équation de calcule d'activation, la démarche à suivre est bien expliquer dans le sujet, reste à savoir maintenant comment la programmer ?

Stockage des chiffres manuscrit

Nous avons tous simplement créer une classe «CodeNumber» dans la package «analyseChiffres» qui contient :

- un champ tableau : qui contient tout les pixels de l'image

- un champ nombre : qui contient le nombre désigné par les pixels

pour l'apprentissage, puisque on a besoin de la réponse, on utilisera des codeNumber pour l'apprentissage vu qu'elle contient un champ nombre qui contient le nombre désigné. Mais pour la reconnaissance, on peut reconnaitre des CodeNumber, ou juste des tableaux de float représentant des nombres quelconques.

Fonctionnement du perceptron multicouche

Nous avons penser à plusieurs structures pour notre réseau, par exemple : créer une classe «Neurone» qui contiendra plusieurs champs désignant par exemples le résultat en sortie du neurones ... Mais instancier un nombre très important de neurones, nous coutera cher.

Finalement nous avons décider de créer une classe «Réseau» qui contient notre réseau et qui est caractériser par plusieurs champs différents :

- w1 : une matrice contiendra les poids de la première partie du réseau de neurones (entre la couche d'entrée et la couche cachée).

- w2 : pareil que w1, mais pour la deuxième partie du réseau (entre la couche cachée et la couche de sortie».

- activation1 : résultat de la fonction d'activation de la couche caché

- activation2 : résultat de la fonction d'activation de la couche de sortie

- delta1 : delta calculé lors de la rétro-propagation au niveau de la couche caché

- delta2 : pareil que delta1 mais pour la couche de sortie

- stat : un tableau de deux cases qui contient les statistiques lors de la reconnaissance. La case «0» contient le nombre d'échecs, la case «1» contient le nombre de réussite.

- historique : un champ «StringBuffer» qui contient l'historique des statistiques tout au long d'une expérience.

Les matrices des poids sont définis de la manière suivante : chaque ligne contient tous les poids connectés au neurone x, par exemple :
w1 [0][1] = le poids entre le neurone 0 de la couche intermédiaire et le neurone 1 de la couche d'entrée.
w2 [2][5] = le poids entre le neurone 2 de la couche de sortie et le neurone 5 de la couche intermédiaire .

Nous avons définit deux constructeurs, un constructeur qui initialise, grâce au paramètre donné, le nombre des neurones de la couche cachée et initialise tous les poids avec des valeurs aléatoire. Un autre constructeur qui prend un paramètre de type booléen en plus et qui décide de si les poids doivent être défini d'une manière aléatoire, ou si on doit les chargé du fichier «lesPoids.txt». Nous avons décider d'ajouter cette fonctionnalité pour faire plus de tests ...

Dans notre Réseau il n'y a qu'une seule méthode pour l'apprentissage, qui prend en paramètre la liste des images d'apprentissage, et le nombre d'itération de la liste, on mélange la liste pour chaque itération pour mieux apprendre aux réseau. Et on a plusieurs méthodes pour reconnaitre, ces différentes méthodes effectue le même traitement, mais ne donne pas le même résultat, par exemple : on a une méthode qui retourne un String qui est le rapport de reconnaissance, une méthode qui calcule les statistiques lors de la reconnaissance...

Remarque :

Notre classe «Reseau», utilise une autres classe «Traitement» qui contient que des méthodes «static», nécessaire au calcule de la fonction d'activation, rétro-propagation ... etc. toutes ces fonction sont la traduction exacte des données fournit sur le sujet pour le fonctionnement du perceptron.

cette classe contient aussi deux variables static, qui définissent le seuil d'activation et de désactivation d'un neurone.

Nous avons fixé le seuil d'activation à 0.9 et le seuil de désactivation à 0.1 .

Codage et Décodage des Images

1. Codage

Nous avons décidé pour le codage des images, de créer une classe «Codeur» qui traitera les images, pour cela nous avons créer une méthode «chiffrer_image» qui prend une image et la traite pixel par pixel grâce au BufferedImage et qui récupère le code de tous les pixels de l'image sous niveau de gris en utilisant la méthode «getGris», pour les simplifier par la suite avec la méthode convertisseur qui nous donne un nombre entre -1 et 1. Le code récupéré est stocké dans un tableau de flloat et retourné par la fin du traitement de chaque image.

Avec la méthode «capture_pixel» on stocke tous les tableaux de pixels de chaque image dans une «ArrayListe» de «CodeNumber» , cette Liste serra utilisé pour écrire le code des pixels dans le fichier avec la méthode «écriture», qui prendra en paramètre cette liste de «CodeNumber» et le nom du fichier ou on veut écrire les codes pixels, et cette méthode parcourra la liste et grâce à une méthode «toString», on écrira chaque pixel de l'image de la

manière souhaité et qui nous faciliteras la récupérations de ces dernier, de manière qui nous facilitera le traitement par la suite.

La méthode «chiffrage» prend en paramètre une image qui contient plusieurs petites images de chiffres, cette méthode découpe l'image de manière a récupérer les petites images et les enregistre dans un dossier, et toutes ces images serrons traiter par la suite avec une méthode «remplir_tableau_simple» qui vas nous retourner un ArrayListe de tableau de float, chaque tableau contiendra les codes pixel d'une image du dossier

Cette liste seras utilisé pour l'apprentissage dans le réseau de neurone.

Nous avons eu des inconvénients avec la taille des images, ce qui nous a mené à penser à une méthode «redimensionner» et comme son nom l'indique, redimensionne ces dernière à une taille 16x16 et qui contient précisément 256 pixels.

2. Décodage

Pour le Décodage ou autrement dit, la création d'images, nous avons créé une classe «CréerImage» qui permet à l'aide d'une base de chiffres manuscrits de créer un répertoire d'images correspondant à cette base. Cette classe est tout d'abord constitué de trois attributs qui sont «*Hauteur*», «*Largeur*» et «*image*».

- «*Hauteur*» : permet de définir la hauteur de l'image en pixels, on la fixe automatiquement à 16 pour le projet tout comme «*Largeur*» qui lui défini la largeur.

- «*image*» : représente l'image créé par cette classe, c'est une «BufferedImage».

Pour effectuer cela nous avons tout d'abord trois sous-programme important qui permettent la gestion de l'image :

- «CreerImage» : qui initialise la «BufferedImage», *image,* en fonction de «*Hauteur*» et «*Largeur*». L'attribut *image* est donc créé, c'est une image blanche de taille «*Hauteur*» *X* «*Largeur*».

- «DessinerImage» : qui prend en paramètre un tableau de pixels (représentés par des floats) et qui retourne rien. Ce sous-programme colore une image pixel par pixel pour obtenir une image représentant un chiffre. Pour cela il utilise pour chaque pixel une valeur du tableau

Remarque :

Il faut donc que la taille du tableau en paramètre soit égale au nombre de pixels de l'image.

- placé en paramètre, sauf que les valeurs dans le tableau correspondent à des niveaux de gris. Donc on convertit cette valeur pour obtenir une couleur que l'on introduit ensuite dans l'image à l'aide de la méthode d'instance *setRGB.*

- «SauverImage» : prend en paramètre le chemin, le nom et le format de l'image sous forme de String. Ce sous programme permet donc tout simplement la sauvegarde de l'image au bon endroit et sous le format choisi.

- «CreerRepertoire » : prend en paramètre un Iterator de «CodeNumber», le nom du Répertoire et le nom de l'Image. Ce sous-programme crée donc un répertoire en fonction du nom donnée en paramètre, et crée ensuite à l'aide des méthode ci-dessus toutes les images de l'Iterator dans le répertoire qui vient d'être créer. Cette méthode n'a donc plus qu'a être appeler par le «Main» pour la création d'un répertoire d'images.

Les Commandes

Nous avons classifié les commandes par rapport à la taille du tableau d'argument. Donc on a des commandes de taille 1 (-h, -name), de taille 3 (-b, -i), de taille 4 (-r, -ri...). Ce qui nous a permis d'avoir une gestion d'erreur plutôt simple. Ensuite pour chaque type de taille nous avons créer une fonction qui selon le cas (selon la commande) exécute chaque commande tout simplement.

La gestion des erreurs est effectuer grâce à des exceptions nous avons créer deux types d'exceptions :

- CommandeException : les erreurs concernant les commandes.

- ConversionException : les erreurs concernant les conversion d'images ... etc.

Nous avons aussi créer une classe «Utile» avec que des fonctions static, qui nous permettent d'avoir des fonctions utile comme par exemple : remplir un tableau avec des valeurs aléatoires, définir si le chemin correspond à un fichier ou un dossier, copie de fichier, vérification du format de string ... etc.

Les Packages

Nous avons divisé l'ensemble des classes en trois packages :

- «analyseChiffres» : ce packages contient toutes les classes nécessaires au traitement des images, manuscrit ou en forme de fichier images.

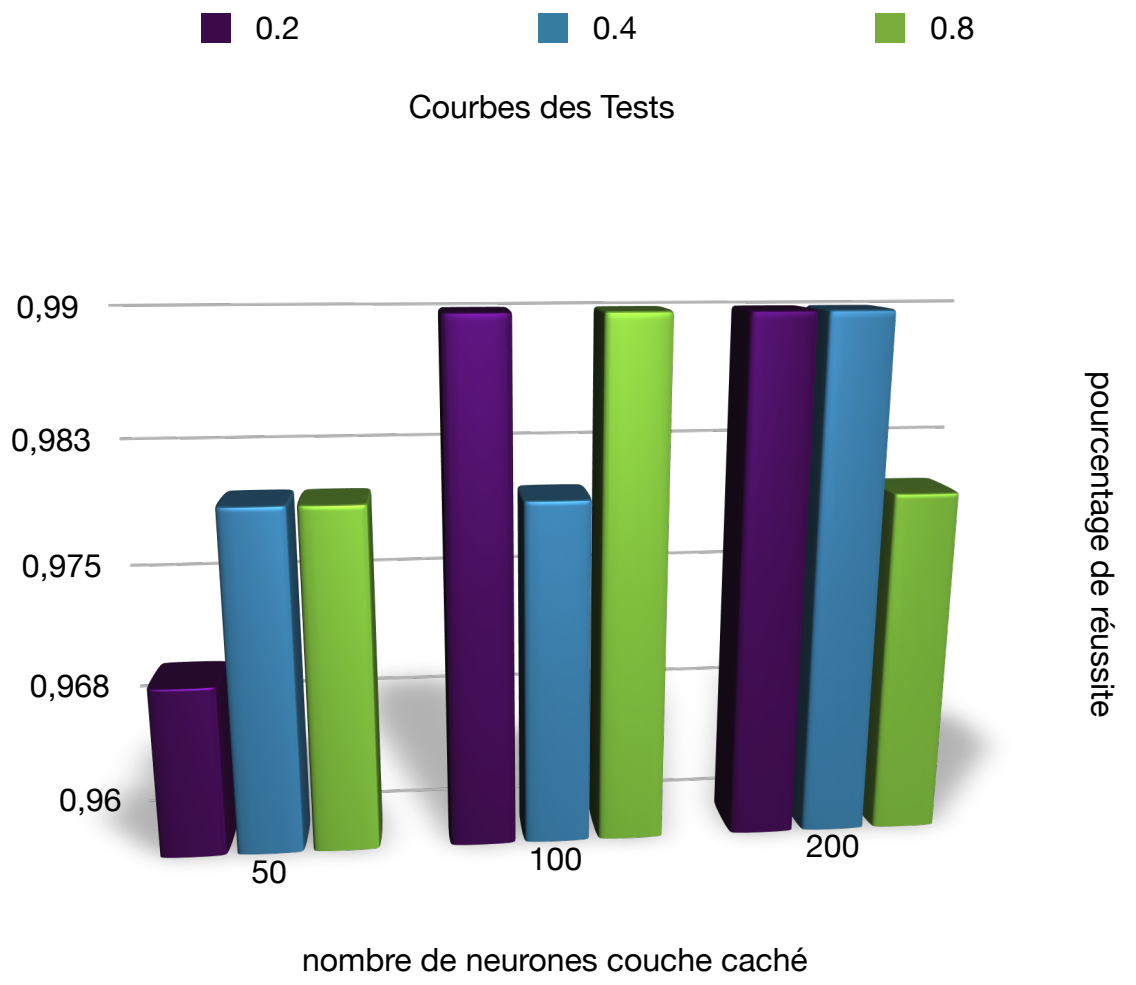
- «reseauNeurone» : comme son nom l'indique, ce package, contient toutes les classes nécessaires pour créer notre réseau de neurone artificielle.

- «lecture» : contient les classes qui traite les arguments passé au programme, donc la classe «Main» et et la classe commande.

Test et Courbes

Nous avons effectué quelques tests au niveau du pourcentage de réussite de notre perceptron, sachant que notre pour que le seuil d'activation est fixé à 0.9 et de désactivation à 0.1 .

le graphe suivant nous donne les statistiques des relation par rapport au pas d'erreur compris entre 0 et 1, et puis le nombre de neurones de la couche caché



MODE D'EMPLOI

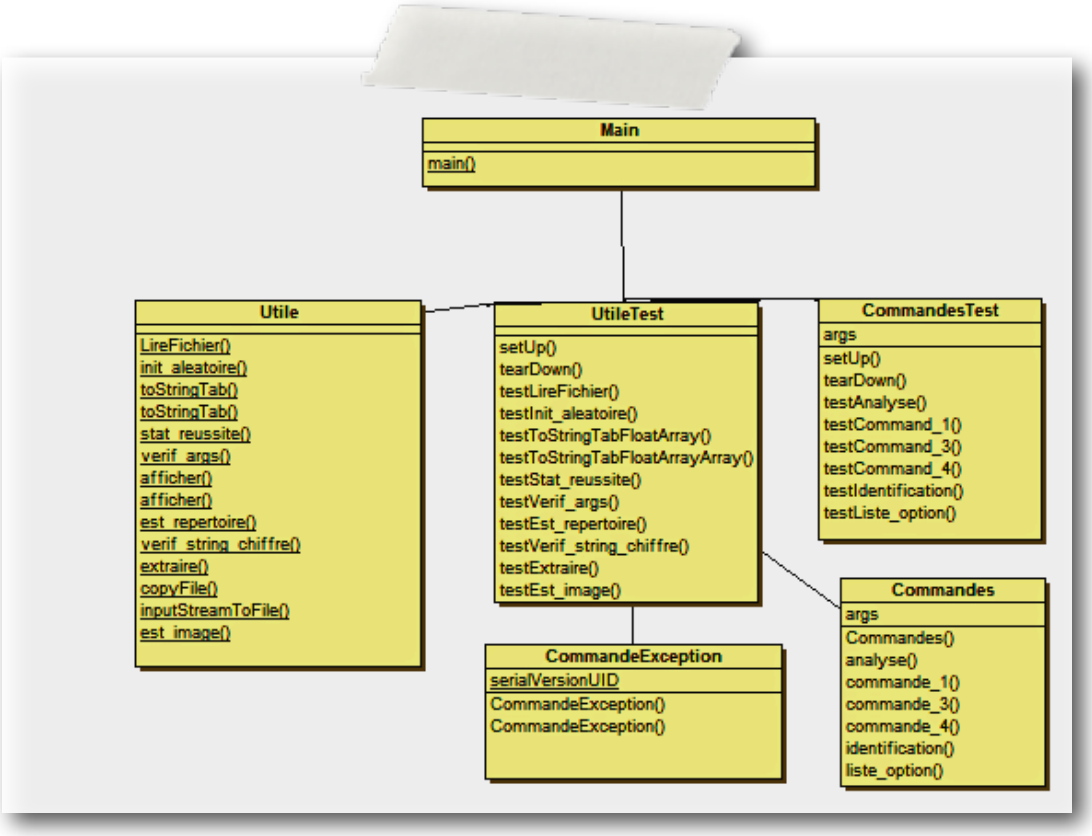
Le programme est donc sous forme d'un fichier «neurone.jar», ce fichier qui faudra exécuter par la suite sous le terminal avec la commande «java -jar» comme suit :

```
$  java -jar neurone.jar -h

-------------------------

REGLES ET FORMAT

-------------------------

Les codes des chiffres manuscrit enregistrés dans le fichier "base.txt" est de la forme => x1,x2,...,x256,leNombre.Le 1er caractère du nom d image doit contenir le nombre représenté par l image.

Les caractéristiques "param" écrit de la forme suivante => "nombre_de_neurone_couche_cache"-"teta"

teta : est le pas d'erreur compris entre 0 et 1 nécessaire pour le calcule et l'apprentissage.

------------------------------

LISTE DES COMMANDES

------------------------------

-name : pour afficher les noms des membres du groupe.

-h : pour afficher la liste des options.

-b base.txt image : lit une base de chiffres manuscrits et crée un dossier "image" contenant les images des chiffres de la "base". le numéro que caractérise l'image sera inscrit sur le premier caractére du nom de l image.

-i image base.txt : construit une base de chiffres manuscrits ? partir des fichiers contenus dans le dossier "image".

-r param base.txt donnees.txt : renvoie le taux de réussite des chiffres reconnus contenu dans "donnees.txt" par le réseaux de neurones ayant comme caractéristiques "param" et pour base d'apprentissage "base.txt".

-ri param base image : renvoi les valeurs des chiffres contenu dans les images du fichier "image" en effectuant une reconnaissance sur l'ensemble des sous images représentant des chiffres, après avoir fais un apprentissage avec les images du dossier "base".

-w base.txt donnees.txt resultat.html : construit une page html contenant le taux de réussite des chiffres reconnus contenu dans "donnees.txt" par le réseaux de neurones ayant comme base d'apprentissage "base.txt" en faisant varier les caractéristiques du réseaux.
```

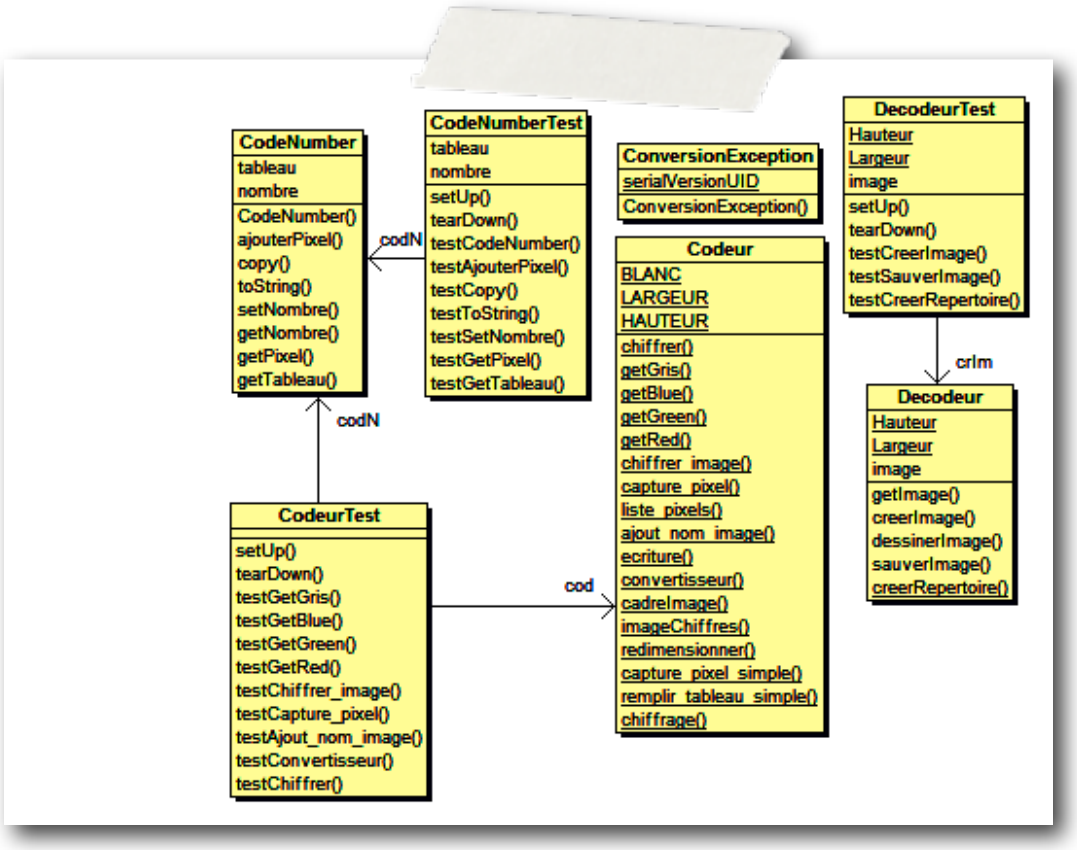
Pour pouvoir visualiser la liste des options il faut exécuter la commande -h comme dans l'exemple.

ANNEXE : UML

Pack lecture UML



Pack analyseChiffre UML



Pack reseauNeurone UML