LANSAR Mohammed
BADIANE Omar

Rapport du Projet : Perceptron Multi-classe

I) Introduction et motivation

Un perceptron est par définition un algorithme d'apprentissage supervisé pouvant séparer plusieurs classes. Après avoir programmé un perceptron permettant la séparation binaire pour des chiffres (autrement dit savoir si une image est un chiffre précis ou non) nous avons décidé d'étendre « la portée » de notre perceptron en le codant de façon à séparer plusieurs classes en même temps. En effet la classification binaire est très limitée pour un jeu de donnée contenant plus de deux classes puisque ce dernier peut éprouver d'énormes difficultés à correctement séparé les données avec un taux d'erreur raisonnable. Ainsi nous avons décidé d'implanter un modèle de perceptron multi-classe s'appuyant sur un aspect probabiliste de la chose (plus exactement on utilisera la fonction *softmax* pour déterminer nos probabilités à l'appartenance d'une classe).

II) Organisation du code et jeu de donnée utilisé

Le projet Java contenu dans le dossier présent est divisé en plusieurs classes :

- La classe **Cerveau** contient les procédures utilisées dans la réalisation du perceptron (et la procédure **perceptron** elle-même) ainsi que deux procédures : l'une permettant de rajouter un chiffre dans nos tableaux de données (cette procédure **rajoutCase** sera explicitée plus tard) et l'autre permettant de tester notre perceptron.
- La classe **ReconaissanceImage** contient notre **main** qui lui-même contient l'algorithme permettant de charger les données ainsi que l'appel des procédures contenus dans la classe **Cerveau**.
- Enfin la classe **Traitement** contient les fonctions qui nous ont permis de réaliser l'analyse des résultats.

Toute la documentation des procédures utilisés se trouve dans le code Java.

Pour ce projet, nous avons décidé de nous concentrer sur le jeu de donnée appelée *Caltech 101 silhouettes* au détriment des chiffres. Ce choix s'explique en la quantité de donnée d'entrainement qui est quatre fois plus importante que celle contenant les chiffres et le nombre de classes est dix fois plus importants que cette dernière. Ainsi en plus d'avoir un grand nombre de classe rendant la tâche plus difficile, mais non moins intéressante, chaque classe contient des exemples plus ou moins différents au sein d'une même classe ce qui est encore plus intéressant pour l'observation de la phase d'apprentissage. Ensuite on précise que nous avons délibérément augmenté d'une case les tableaux des données pour prendre en compte le fait que notre **w** prend le biais : on utilise la procédure **rajoutCase**. Ensuite on choisit un **Eta** (ou taux d'apprentissage) pour une valeur de 0.1 puisque cette valeur nous semble optimale en terme de rapidité d'apprentissage ainsi que pour la convergence de notre taux d'erreur que nous analyserons dans la partie III) Résultat (et plus en détail en a)Taux erreur).

III) Résultat

Ainsi en exécutant la procédure **perceptron**, tout en joignant nos données d'entrainements adaptés, notre tableau 2D de poids **w** est mis à jour après 88 itérations (itération maximal fixé à cette valeur qui semble donner de meilleurs résultats après avoir testé différentes valeurs) et un **Eta** = 0.1 (une valeur pour laquelle le nombre d'erreur converge bien vers de petites valeurs). Nous soulignons le fait que par la suite nous évoquerons le pourcentage de réussite du perceptron qui représentera le pourcentage d'image bien classé. On évalue donc le pourcentage de réussite grâce à la fonction **perceptronTest**. Dans un premier temps on effectue cette évaluation sur notre ensemble d'entrainement pour vérifier que le perceptron à bien convergé vers un nombre d'erreur peu important (vue le nombre de classe et la complexité de différenciation de certaine image même pour un humain, on établira arbitrairement que 10% d'erreur est une bonne valeur).

On obtient donc un pourcentage d'environ 96% de réussite (soit 4% d'erreur) pour l'ensemble d'entrainement (cf. Figure 1 – Capture d'écran du terminal d'Eclipse).
Puis dans un second temps, on teste notre perceptron pour l'ensemble dit de test et on obtient environ 60% de réussite soit environ 40% d'erreur. On étudiera les erreurs en détail dans la partie suivante (cf. IV)Étude des résultats).

```
On test notre Perceptron pour notre ensemble d'entrainement :
/* Il y a 96.439026% de reussite. */
On test notre Perceptron pour notre ensemble de test :
/* Il y a 60.251408% de reussite. */
```

Figure 1 - Capture d'écran du terminal d'Eclipse

IV) Étude des résultats

Désormais nous allons étudier en détail les résultats obtenus et effectuer des interprétations quant aux erreurs trouvées.

a) Taux erreur

On va observer comment varie le taux d'erreur pendant l'apprentissage du perceptron avec l'ensemble d'entrainement donc. On utilise une procédure **perceptronErr** qui va écrire dans un fichier texte le nombre d'erreur en fonction des époques. On trace la fonction $y = f(x)$ avec x le nombre d'époque et y le nombre d'erreur. On obtient ainsi le graphique suivant :

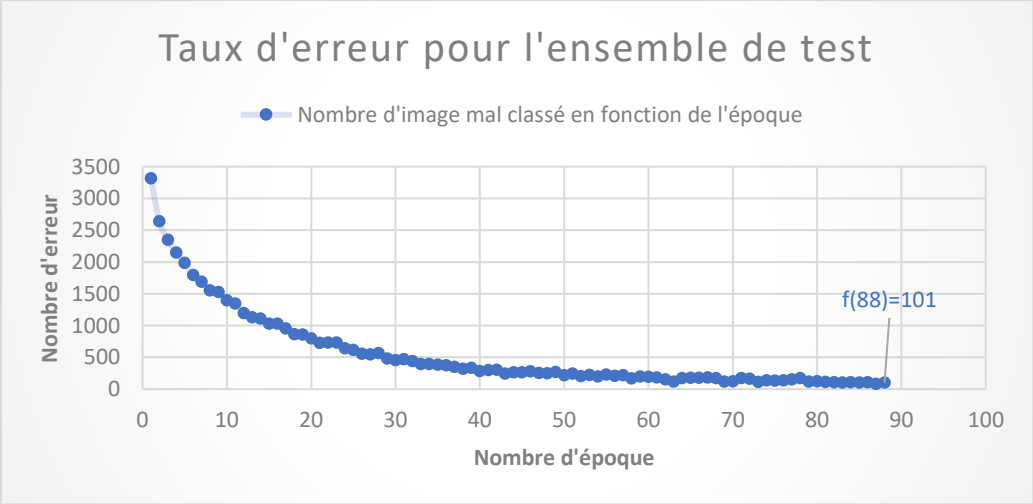

Figure 2 - Graphique représentant l'évolution du nombre d'erreur en fonction des époques

On remarque que notre fonction est décroissante et que pour 88 itérations ont a $f(88) = 101$. Ainsi notre perceptron avec un Eta = 0,1 et 88 itérations converge bien vers un nombre d'erreur très correct puisqu'il ne représente que 4% des données (cf. Figure 1 – Capture d'écran de la terminale d'Eclipse). Ce qui est une preuve de bon fonctionnement de notre algorithme d'une part et surtout un chiffre montrant que notre perceptron est plutôt performant puisque le nombre d'erreur converge bien. Malgré ce pourcentage on obtient environ 60% de réussite pour le test. Cela implique que certaines classes ou images sont problématiques.

b) Classe problématique

Un taux d'erreur aussi élevé implique forcement que soit certaines classes sont problématiques (dans le sens où elles se ressemblent) soit que les images sont difficilement différentiables. Dans cette partie nous

évoquerons uniquement le problème des classes. En effet, on obtient 40% de mauvaise réponse avec les données de test : on a donc environ 922 images mal classées. On va donc étudier la phase d'apprentissage car c'est ici que nos poids sont « calibrés » et surtout étudier la phase de test. Ainsi si on se concentre sur les classes, on utilise une procédure **reperageErreur** qui va compter combien de fois on aurait dû prédire tel ou tel classe. Ainsi on obtient les diagrammes suivants :

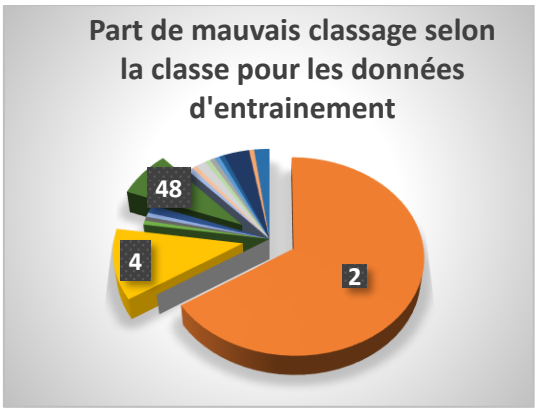

Figure 3 - Diagramme de la part des classes qui n'ont pas bien été estimé pour l'apprentissage

Figure 4 - Diagramme de la part des classes qui n'ont pas bien été estimé pour l'apprentissage

Ainsi grâce à ces deux diagrammes on peut voir que : que ce soit pendant l'apprentissage ou même l'évaluation des données de test, on a la Classe 2 qui semble fortement problématique ainsi que la classe 1, 4 et 48. Pour être bref, on peut expliquer que la deuxième classe est la plus problématique puisque à vue d'œil (en observant les images de cette classe sur le site où se trouve les données) on remarque que cette classe fait référence aux têtes « faces 2 » et qu'elle occupe un grand espace de l'image d'où l'éventuelle « marge d'erreur » puisque ne l'oublions pas, nos images sont en noir et blanc : ainsi le perceptron peut difficilement différencier deux classes où les images ont les mêmes formes sans la composante RGB de chaque pixel. On peut donc supposer que pour ce cas, un jeu de donnée en couleur aurait permis une meilleure différenciation des formes.

c) Visualisation des poids et des images moyennes

Nous allons désormais afficher deux exemples de séparatrices (ou neuronnes) grâce à une procédure nommé **impressionClasse**. On va donc stocker dans un fichier texte (pour chaque classe) les valeurs des pixels (les valeurs des poids de chaque neurone donc) en ayant au préalable additionné toutes les valeurs de chaque pixel par la valeur absolue du poids minimum pour ainsi avoir des valeurs strictement positives. L'extension du fichier sera PGM pour Portable GrayMap qui est un format de donnée (très pratique) utilisé pour représenter une image en niveaux de gris (Page Wikipedia des PPM). Voici les deux neuronnes obtenus après avoir retiré le poids **w[classe][0]** (le biais) de chaque classe et convertis en .jpg les fichier .pgm :

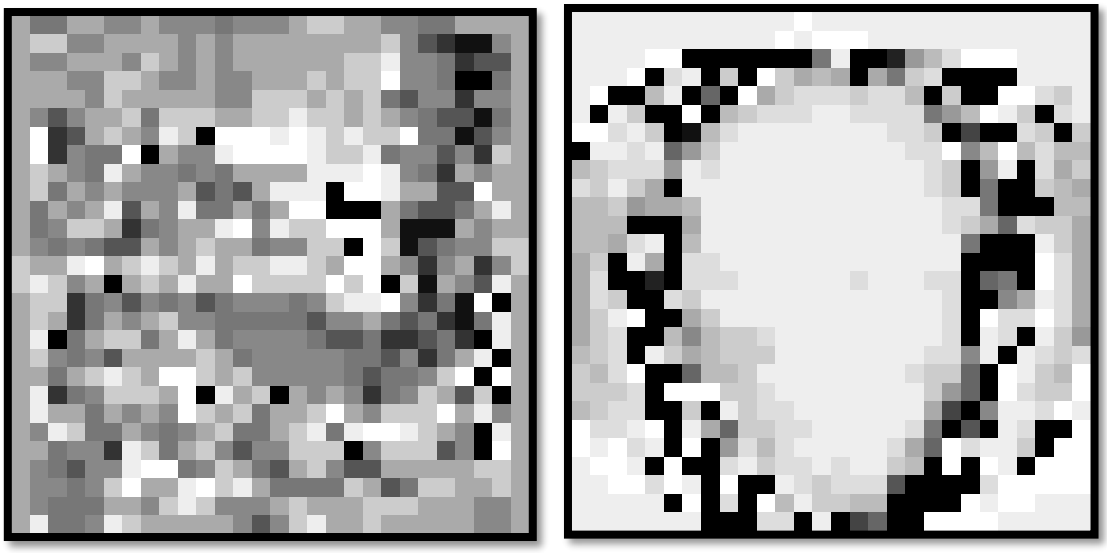

Figure 5 - Représentation des neurones en image nuancé de gris (à gauche les neurones de la classe 1 contenant "Airplanes" et à droite celle de la classe 2 contenant "Faces 2")

On réutilise la même procédure en incluant la moyenne des pixels et les images triés pour faire les moyennes des images. On obtient pour les classes 1 et 2 toujours :

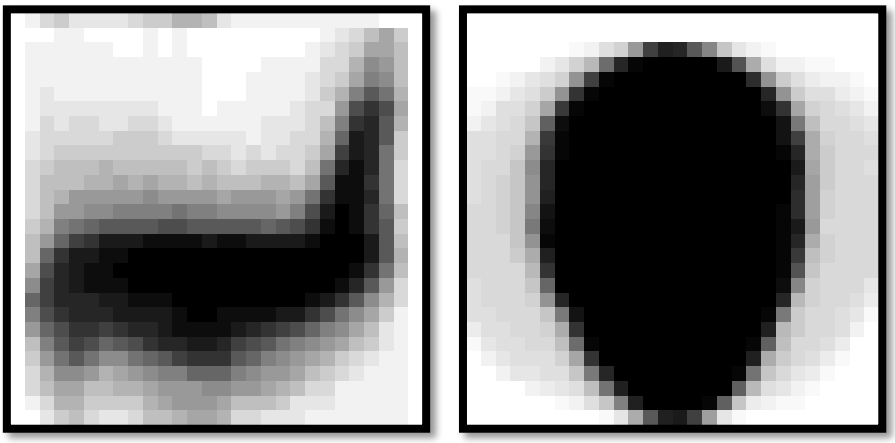

Figure 6 - Représentation de la moyenne des images nuancé de gris (à gauche les poids w de la classe 1 contenant "Airplanes" et à droite celle de la classe 2 contenant "Faces 2")

Ici on reconnait parfaitement la forme de la classe en question en comparant avec le tableau de donnée du site où ces données ont été prise. On y remarque beaucoup de similitude avec nos séparatrices **w** (cf. Figure 5 - Représentation des neurones en image nuancé de gris (à gauche les neurones de la classe 1 contenant "Airplanes" et à droite celle de la classe 2 contenant "Faces 2")) mais malheureusement nous n'avons pas les connaissances nécessaires pour interpréter ce phénomène.