Compte rendu

Projet : Classification par réseau de neurones

1 Introduction

L'objectif de ce projet était de faire de la classification sur des données en utilisant des perceptrons reliés ensemble pour former un réseau de neurones. Les données à classifiées sont de deux types : des données en deux dimensions (pouvant être représentées comme des points dans un plan) et des images de 20 par 20 pixels représentant des chiffres manuscrits. Pour cela, nous avions à implémenter plusieurs solutions techniques : un unique perceptron, un perceptron monocouche et, si le temps le permettait, un perceptron multicouches (à une couche cachée).

2 Unique perceptron

2.1 Analyse théorique et calcul du gradient

Pour N données d'entrainement, avec $y_p(\mathbf{x}_n)$ la sortie du neurone comprise entre 0 et 1 (à cause de la sigmoïde) pour la donnée d'entrée $\mathbf{x}_n$ et avec $t_p(c_n)$ qui vaut 0 si la données $\mathbf{x}_n$ appartient à la classe p, 0 sinon, le critère à minimiser pour la classe p est donné par :

$$f_p(\mathbf{W}) = \frac{1}{2N} \sum_{n=1}^{N} \left(y(\mathbf{x}_n) - t_p(c_n)\right)^2$$

En utilisant l'algorithme de rétro-propagation du gradient, on peut calculer le gradient du critère par rapport aux poids que l'on cherche à optimiser :

$$\boxed{\frac{\partial f}{\partial \mathbf{w}} = \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}_n \left(y(\mathbf{x}_n) - t_p(c_n)\right) y(\mathbf{x}_n)(1 - y(\mathbf{x}_n))}$$

2.2 Unique perceptron pour classification de données 2D

Une fois l'étude théorique du problème réalisée, et le calcul du gradient effectué, nous avons pu implémenter un unique perceptron sur Matlab. Nous avons tout d'abord travaillé avec les données en 2D fournies dans les fichiers `DataX_2Clases_Perceptron.mat`.

2.3 Unique perceptron pour classification d'images de chiffres manuscrits

Il y a eu deux phase dans l'utilisation d'un perceptron pour classifier des images de chiffres. Nous avons d'abord commencé par une classification binaire avec deux classes comme pour les données en deux dimensions, puis nous avons fait une classification du type "un contre tous", où tous les chiffres sont utilisés pour entraîner le perceptron à classifier une classe en particulier.
NB : après plusieurs essais d'initialisation aléatoires ou à 1, c'est finalement l'initialisation des poids à 0 qui nous a semblé la plus efficace dans ce projet.

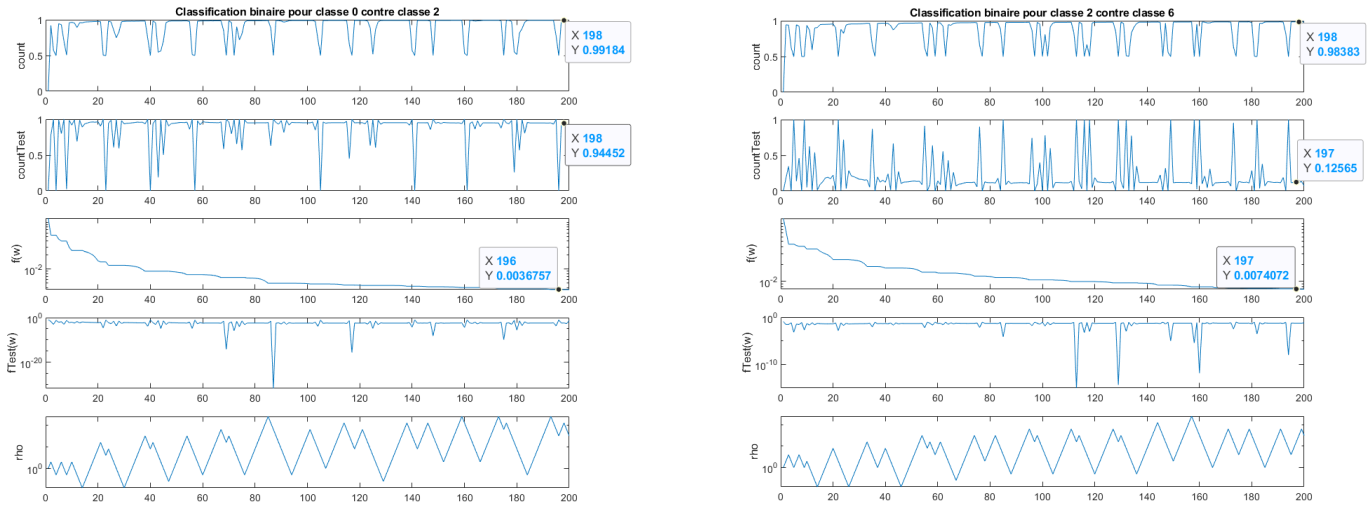

FIGURE 1 – Exemple de phase d'apprentissage en classification binaire

La figure 1 présente deux exemple de phase d'apprentissage en classification binaire : classe 0 (classe cible) contre 2 et classe 2 contre 6. C'est deux exemples sont extrêmes car on voit dans le premier cas que la classification fonctionne très bien (0,94% d'images tests bien classifiés) alors que dans le deuxième cas la classification éprouve des difficultés (0,13% d'images tests bien classifiés). Deux conjectures nous sont apparues :

— l'initialisation des poids $\mathbf{w}$ à zéro n'est peut être pas judicieuse dans certains cas (même si elle semble tout à fait convenable dans la plupart des cas) ;

— certains chiffres sont plus difficiles à différencier.

Nous avons testé cette classification binaire sur plusieurs paires de chiffres, et en changeant la classe cible. Les résultats sont présentés sous forme de matrice de confusion.

Deuxième classe

Classe cible

	1	2
1	0,8568	0,1432
2	0,3262	0,6738

	1	6
1	0,7203	0,2797
6	0,1304	0,8696

	1	8
1	0,9882	0,0118
8	0,012	0,988

	6	8
6	0,9905	0,0095
8	0,5391	0,4609

	2	8
2	0,8483	0,1517
8	0,3812	0,6188

	2	6
2	0,1328	0,8672
6	0,1219	0,8781

FIGURE 2 – Classification binaire pour plusieurs classes différentes

Plus la matrice de transition se rapproche de l'identité et plus on peut considérer que la classification effectuée est efficace. On voit avec ces résultats que l'efficacité de la classification dépend des classes mises en jeu mais également de la classe cible, qui dans le dernier exemple entre les classes 2 et 6 joue énormément sur le résultat. Nous avons essayé de trouver un lien entre l'efficacité d'une classification de deux classes et la forme des chiffres mais les résultats de la figure 2 ne nous permettent pas de trouver un quelconque lien.

Remarque sur la classification binaire : nous trouvons que la classification n'est pas très stable, dans le sens où il y a de grandes variations ponctuelles du nombre de données bien classifiées. Une amélioration de l'algorithme pourrait être de mettre en jeu une notion de "marge" à maximiser, qui serait la distance entre l'hyperplan séparateur défini par le perceptron et les points les plus proches (dans un espace vectoriel de dimension 20×20), notions justement utiliser dans les SVM...

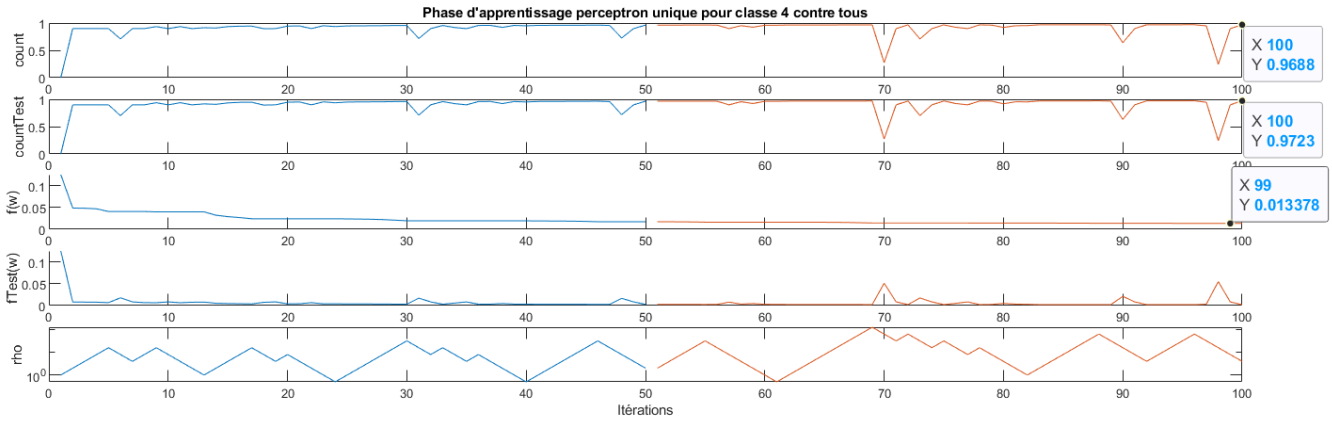

FIGURE 3 – Exemple de phase d'apprentissage en classification 4 contre tous

La classification en mode "un contre tous" semble avoir de meilleurs résultats que la classification binaire et semble également moins instable (vis-à-vis du nombre de points bien classifiés aux différentes itérations). Nous pensons que les problèmes qu'il pourrait y avoir comme entre la classe 2 et la classe 6 sont "moyennés" avec les autres classification binaires ce qui donne globalement un meilleur résultat. Une autre piste est qu'ayant plus d'exemple à disposition, et devant trouver une séparation entre un chiffre et tous les autres, l'algorithme arrive à trouver des éléments discriminants propre à chaque chiffre et qui permet de le différencier des autres chiffres, plus que ne le fait l'algorithme en classification binaire.

3 Perceptron monocouche

Dans le cas particulier d'un réseau de neurones avec un seule couche, on se rend compte qu'un perceptron monocouche est équivalent à P perceptrons en parallèles. En effet, les sorties de chaque perceptron dépendent des poids associés au perceptron et des données d'entrée, il n'y a donc aucune interaction entre les perceptrons et il sont donc indépendants les uns des autres. La seule chose que change est qu'on ne fait plus une classification binaire mais plutôt une classification du type "un contre tous" (sous entendu la classe voulu ou une autre classe).

Ainsi, pour traiter le cas du perceptron monocouche nous avons réutilisé les résultats précédents pour un unique perceptron que nous avons répété sur P perceptrons. La seule différence entre les perceptrons est qu'ils sont associés à des classes différentes et donc que leur critère à optimiser, et donc le gradient associé est paramétré sur la classe en question (i.e le p qui change dans $t_p(c_n)$).

Pour améliorer les performances de l'algorithme (en terme de nombre d'itérations), nous avons préféré garder un critère pour chaque perceptron que nous optimisons indépendamment les uns des autres avec des pas différents. Ceci est possibles car les perceptrons sont indépendants et que le critère global du réseau de neurones est donc la sommes des critères pour chaque neurone. Cela permet une diminution du nombre d'itérations.

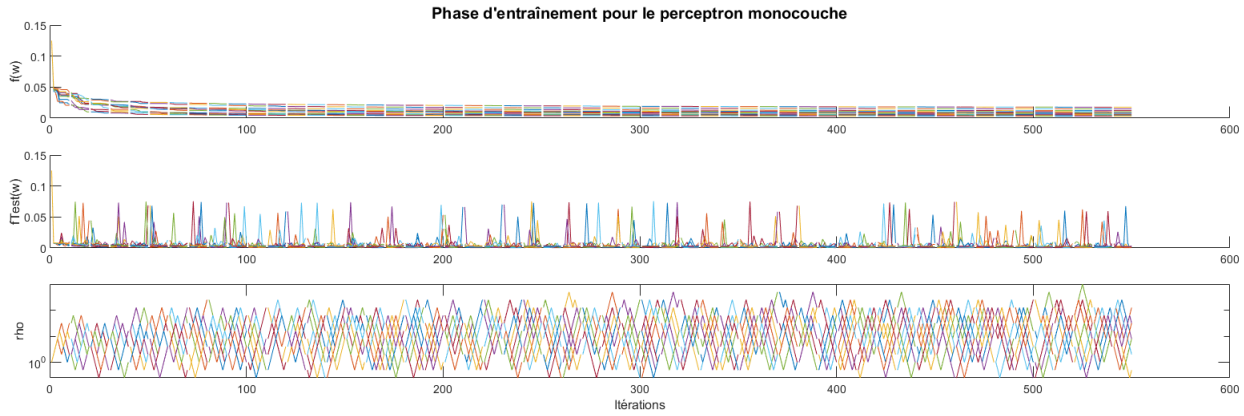

FIGURE 4 – Phase d'apprentissage du perceptron monocouche

Nous avons réalisé le perceptron monocouche et l'avons entraîné sur les données d'entraînement sur 550 itérations (durée du calcul : environ 30 min). On observe bien une diminution des critères pour chaque perceptron ainsi que la

diminution en moyenne du critère sur les données de test.

Nous avons également calculé la matrice de transition du réseau de neurone avec les poids optimisés. On remarque que la matrice est "proche" de la matrice identité ce qui suggère une bonne performance de la classification obtenue. En moyenne, nous obtenons 0,9153% d'images de test bien classifiées, ce qui est très satisfaisant compte tenu du nombres de perceptrons mis en jeu (résultat à relativiser à la simplicité des données mise en jeu, qui ne sont pas des images complexes à discriminer sur 2000 classes comme c'est la cas pour le projet ImageNet d'un tout autre niveau...).

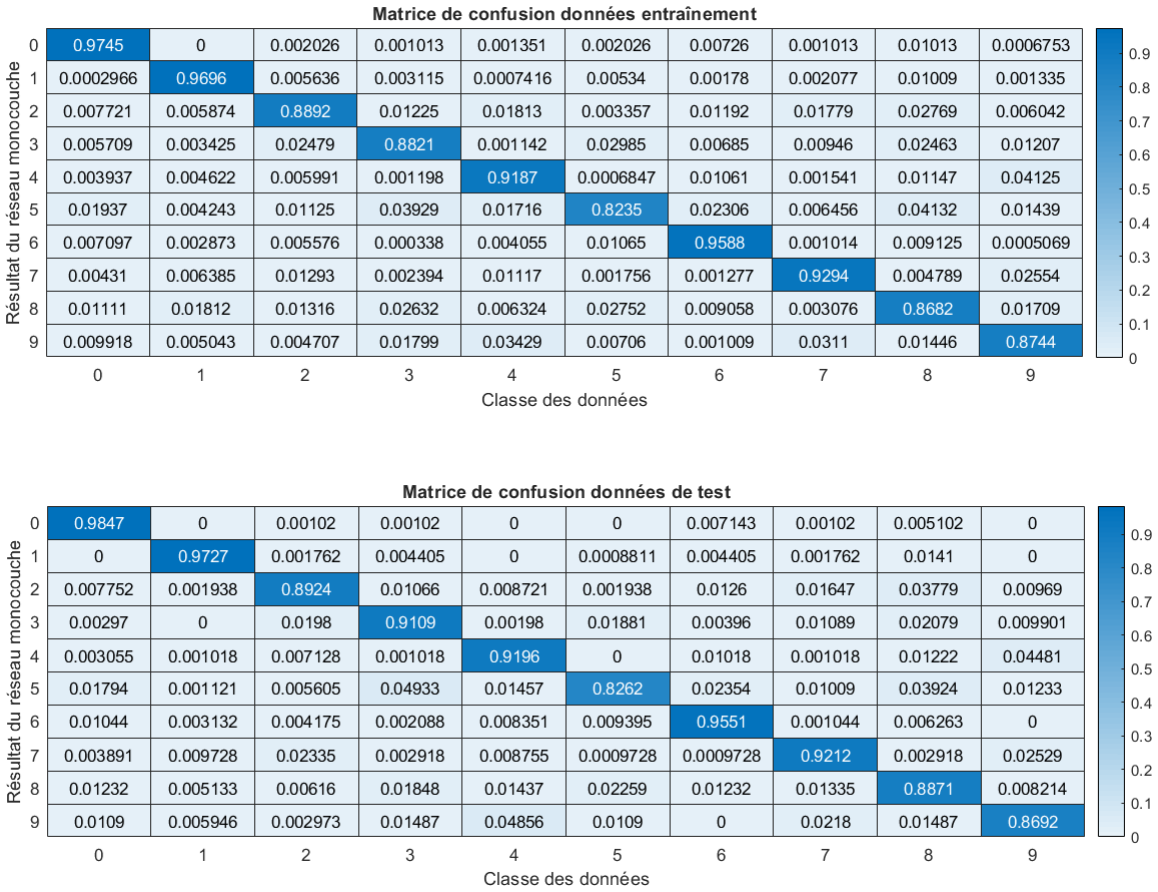

FIGURE 5 – Matrice de confusion pour les données d'entraînement et de test

4 Perceptron multicouche à une couche cachée

Dans le cas d'un perceptron multicouche, il n'est plus possible de séparer les perceptrons car il y des intéractions entre eux entre les couches, ce qui nous permet entre autre d'approximer des fonctions plus complexes que précédemment. Il faut donc recalculer les gradients pour la mise à jour des poids des deux couches.

$$f(\mathbf{W}^{(1)},...,\mathbf{W}^{(L)}) = \frac{1}{2N}\sum_{n=1}^{N}\sum_{p=1}^{P}(y(\mathbf{x}_n) - t_p(c_n))^2$$

4.1 Cas particulier pour la classification des données en 2D

Nous avons essayé de mettre en place le perceptron multicouche dans le cadre des données en 2D. Dans de cas particulier, nous avons $L = 2$ et $P = 2$. Voici les gradients que nous avons calculé :

$$\frac{\partial f}{\partial w_{i,j}^{(1)}} = \frac{1}{N}\sum_{l_2=1}^{2} w_{l_2,i}^{(2)}\sum_{n=1}^{N}\left[(y_1(\mathbf{x}_n) - t_1(c_n)) + (y_2(\mathbf{x}_n) - t_2(c_n))\right]\mathbf{x}_j y_i^{(1)}\left(1 - y_i^{(1)}\right) y_{l_2}^{(2)}\left(1 - y_{l_2}^{(2)}\right)$$

$$\frac{\partial f}{\partial w_{i,j}^{(2)}} = \frac{1}{N}\sum_{n=1}^{N}\left[(y_1(\mathbf{x}_n) - t_1(c_n)) + (y_2(\mathbf{x}_n) - t_2(c_n))\right] y_j^{(1)} y_i^{(2)}\left(1 - y_i^{(2)}\right)$$