Rapport de projet
Classification à l'aide d'un réseau de neurones

Théo Villette - Cléa Humbert

November 7, 2021

1 Présentation

1.1 Problématique

Ce projet consiste à effectuer une classification de chiffres manuscrits de 0 à 9 représentés par des images de taille $20 * 20$ en niveaux de gris. Nous passerons pour cela par un réseau de neurones composé de $20 * 20 = 400$ entrées et de 10 sorties correpondant au nombre de chiffres manuscrits.

Nous étudierons d'abord une classification binaire à 2 entrées et 1 sortie puis nous appliquerons cette classification à 400 entrées correspondant aux matrices $20 * 20$ vectorisées pour classer deux chiffres manuscrits. Puis nous mettrons en place la classification multiclasse par perceptron monocouche.

1.2 Principe

Dans tout les cas, la démarche suit un même principe : trouver les poids des différents neurones en entrainant notre modèle pour ensuite pouvoir tester le réseau sur un jeu de données test.
Nous avons exploité la méthode du gradient à pas fixe mais celle-ci ne convergeait pas assez vite pour le cas de la classification multiclasse. Nous avons alors implémenté la méthode du gradient à pas variable en procédant comme suit :

- calculer le gradient de la fonction de coût.

- itérer $\underline{w}^{(k+1)} = \underline{w}^{(k)} - \rho^{(k)} * \underline{grad}(f)^{(k)}$

  Avec $\rho^{(k)} = 2 * \rho^{(k-1)}$ si $f^{(k)} \leq f^{(k-1)}$ et $\rho^{(k)} = \rho^{(k-1)}/2$ sinon.

- vérifier la précision avec un jeu de données test.

2 Classification binaire par perceptron

Dans le cadre d'une classification binaire, la fonction de coût quadratique devient :

$$f(\underline{w}) = \frac{1}{2N}\sum_{n=1}^{N}\Big(y(\underline{x}_n) - t(c_n)\Big)^2$$

avec :
$\underline{w}$ le vecteur contenant les poids $w_j^{(k)}$ de taille (1, nbit) .
$\underline{x}_n$ le vecteur contenant les nièmes valeurs d'entrée.
$y(\underline{x}_n)$ la nième valeur de sortie de la sigmoide.
$t(c_n)$ la nième valeur attendue en sortie.

Après calcul, on trouve le gradient sous la forme suivante :

$$\frac{\partial f}{\partial \underline{w}} = \frac{\partial f}{\partial y(\underline{x}_n)} * \frac{\partial y(\underline{x}_n)}{\partial z_n} * \frac{\partial z_n}{\partial \underline{w}}$$

C'est-à-dire

$$\Big(\frac{\partial f}{\partial \underline{w}}\Big)_i = \frac{1}{N}\sum_{n=1}^{N}\Big(y(\underline{x}_n) - t(c_n)\Big)\Big(y(\underline{x}_n) - y(\underline{x}_n)^2\Big)x_{in}$$

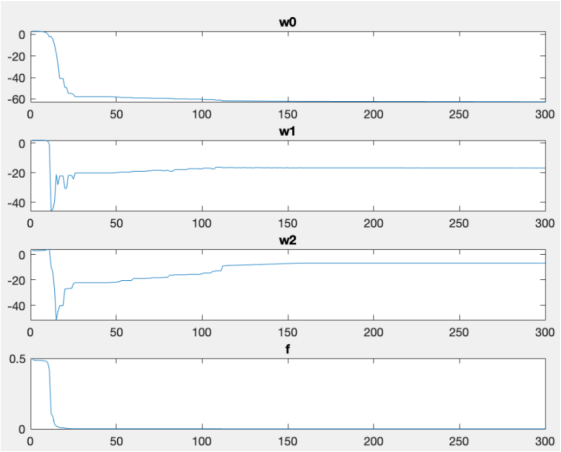
avec $y(\underline{x}_n) = \frac{1}{1+e^{-z_n}}$ et $\underline{z} = \underline{x}^T\underline{w}$

2.1 Avec 2 entrées

Pour 2 entrées, le vecteur poids devient $\underline{w} = \begin{pmatrix} w_0 \\ w_1 \\ w_2 \end{pmatrix}$ avec w_0 le biais.

En choisissant $\rho = 1$, $nbit = 300$ et $\underline{w_0} = \begin{pmatrix} 3 \\ 2 \\ 3 \end{pmatrix}$, on obtient :

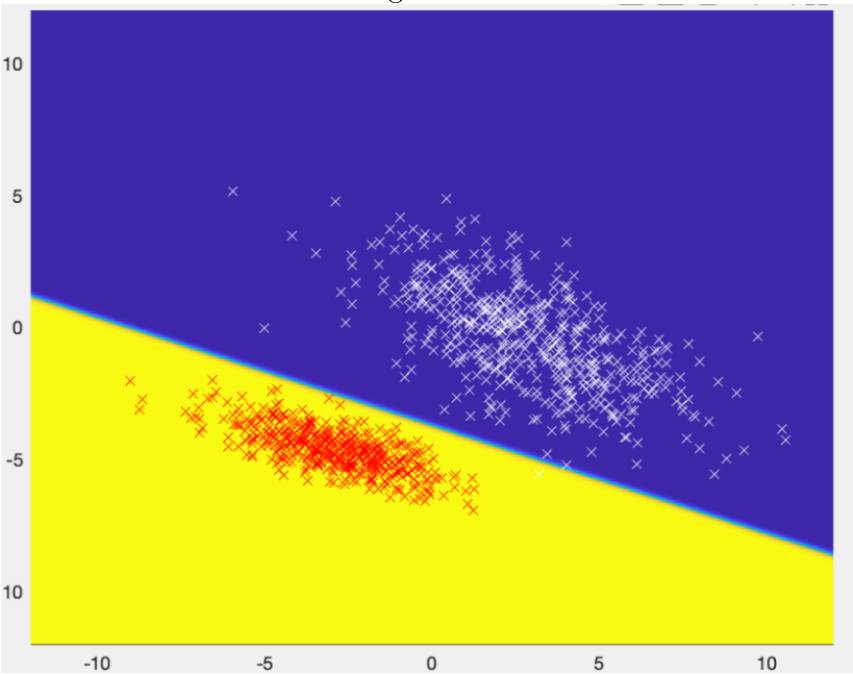
Figure 1:



Choix des paramètres :
- $\rho^{(0)}$ n'a pas ou très peu d'influence sur la rapidité de convergence puisque nous utilisons la méthode du gradient à pas variable.
- nbit a été choisi en fonction de la vitesse de convergence. Au bout de 300 itérations, les poids semblent avoir atteint leur valeur de convergence.
- $\underline{w_0}$ a été choisi aléatoirement.

En effectuant un test de précision sur les données issues de dataTest1 on trouve $99,9\%$ de réussite.
On obtient également le graphe suivant où on voit bien la démarcation entre les deux classes :

Figure 2:



Ici on a affiché les valeurs de sortie du perceptron dans le plan (x1, x2). On remarque la transition entre 0 et 1 au niveau de la frontière qui représente la fonction sigmoïde.

On ne peut pas utiliser cette méthode pour le dataTest2 car il n'existe pas de frontière définie comme une droite.

2.2 Avec 2 chiffres manuscrits

Ici l'entrée est un vecteur colonne de taille 401. Ainsi $\underline{w} = \begin{pmatrix} w_0 \\ \vdots \\ w_{400} \end{pmatrix}$.

En prenant les paramètres suivants : $\rho = 1$, $w_{i0} = 0, \forall i \in [|1, I+1|]$ et $nbit = 500$, voici la précision pour quelques paires de chiffres :

chiffre 1	chiffre 2	précision
0	1	$99,91\%$
0	2	$99,11\%$
1	2	$99,26\%$
3	8	$96,93\%$
7	1	$99,45\%$
4	2	$98,26\%$
9	2	$98,82\%$
5	6	$97,95\%$

On obtient la matrice de corrélation suivante pour les chiffres 0 et 1:

979 (vrais positifs)	1 (faux négatifs)
1 (faux positifs)	1134 (vrais négatifs)

3 Classification multiclasse par perceptron monocouche

Dans le cas présent nous suivrons le même procédé que l'étape précédente. Nous avons trouver les poids comme s'il était question d'une classification binaire "1 contre tous" pour chaque perceptron.
Après avoir chargé et mis en forme (vectorisation, concaténation, ...) toutes les données, nous avons entrainer notre modèle à pas variable avec les mêmes valeurs de paramètres que dans la partie précédente.

Ainsi, on obtient les valeurs suivantes pour une image choisie au hasard dans les données de test respectiviement de 0 et et de 1.

$0 \to 0.970983$ $0 \to 0.000001$
$1 \to 0.000000$ $1 \to 0.981660$
$2 \to 0.020538$ $2 \to 0.000528$
$3 \to 0.095811$ $3 \to 0.004869$
$4 \to 0.000000$ $4 \to 0.000000$
$5 \to 0.000024$ $5 \to 0.019244$
$6 \to 0.000000$ $6 \to 0.000399$
$7 \to 0.000014$ $7 \to 0.000707$
$8 \to 0.000020$ $8 \to 0.039794$
$9 \to 0.000000$ $9 \to 0.000332$

4 Classification multiclasse par perceptron multicouche

La relation d'entrée sortie d'un neurone α de la couche λ est :

$$z_{\alpha}^{(\lambda)} = \sum_{\beta=0}^{I_{\lambda}} w_{\alpha,\beta}^{(\lambda)} * y_{\beta}^{(\lambda-1)}$$

On a également les relations suivantes :

$$\frac{\partial y_p^{(2)}}{\partial y_i^{(2)}} = \frac{\partial y_p^{(2)}}{\partial z_p^{(2)}} * \frac{\partial z_p^{(2)}}{\partial y_i^{(1)}}$$
$$\frac{\partial y_i^{(1)}}{\partial w_{i,j}^{(1)}} = \frac{\partial y_i^{(1)}}{\partial z_i^{(1)}} * \frac{\partial z_i^{(1)}}{\partial w_{i,j}^{(1)}}$$
$$\frac{\partial y_p^{(2)}}{\partial w_{p,i}^{(2)}} = \frac{\partial y_p^{(2)}}{\partial z_p^{(2)}} * \frac{\partial z_p^{(12}}{\partial w_{p,i}^{(2)}}$$

5 Conclusion

On peut conclure en disant que les résultats des différents tests sont plutôt satisfaisants (précision autour de 98%). Cependant on remarque que l'entrainement du réseau prend au total environ 142 secondes, soit plus de deux minutes. Notre modèle demande beaucoup de ressources. On aurait pu réduire ce temps de calcul en utilisant une autre fonction de coût, une autre méthode d'apprentissage comme la méthode de Newton ou une autre fonction d'activation comme la fonction ReLU par exemple.

PS : pour récupérer les script utilisés pour ce compte rendu, voici le lien du dépot git : https://gitlab.ensimag.fr/villetth/perceptron.git