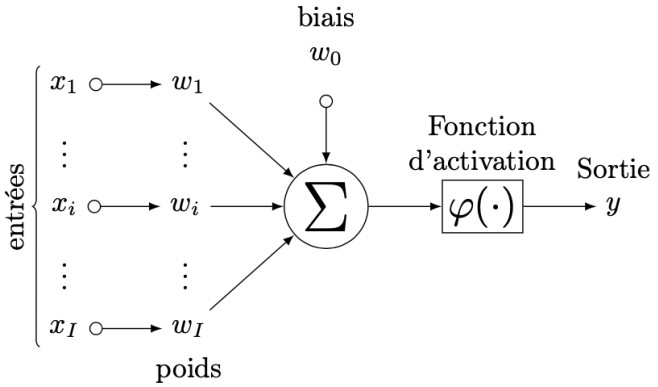
MAALOUL RAYAN
EL KHANOUSSI YASSINE

RAPPORT : PROJET RÉSEAU DE NEURONES

Objectif : L'objectif est de programmer une méthode de reconnaissance automatique de nombres manuscrits à partir de réseaux de neurones artificiels. On se limitera ici à des méthodes basiques.

1) Le perceptron simple (classification binaire).

Nous allons nous intéresser plus particulièrement au cas du simple perceptron.
A partir de plusieurs informations en entrée le perceptron retourne une information en sortie.Cette sortie est calculée grâce aux différents poids associés aux connexions et a une fonction d'activation φ qui permet d'avoir une sortie y comprise entre 0 et 1.



$$z = \sum_{i=0}^{I} w_i x_i,$$

$$y = \varphi(z),$$

$$\varphi(z) = \frac{1}{1 + \exp(-z)}.$$

Le principe est donc de considérer le neurone inactif si la sortie y est proche de 0 et actif si il est proche de 1.

Nous allons utiliser le réseau de neurones pour classifier des données d'entrée pour cela nous définissons la classe Cn de l'entrée Xn qui vaut 0 ou 1.
Nous définissons aussi la fonction de cible qui indique si la donnée d'entrée (Xn,Cn) appartient bien à la classe p.

$$t_p(c_n) = \begin{cases} 1, & \text{si } c_n = p, \\ 0, & \text{sinon} \end{cases}$$

Dans le cas du perceptron simple il existe uniquement deux classes.
Le cas de plusieurs classes sera traité plus tard pour le cas de plusieurs perceptrons en parallèles.

Critère d'apprentissage :

Les données d'entrée sont les (Xn,Cn).
Nous devons donc trouver les poids qui minimisent l'erreur possible du réseau de neurones.
Pour cela nous devons minimiser cette fonction de coût :

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

Dans notre cas a nous il n'y a qu'une seule couche et un seul perceptrons donc P=1 et L=1.
Entraîner un réseau de neurones consiste donc à trouver les poids tels que la valeur de sortie du perceptron **y(Xn)** soit aussi proche que possible de la valeur attendue **t(Cn).**

Pour optimiser ces poids nous avons décidé d'utiliser la méthode du gradient à pas fixe.

Le gradient de la fonction de coût pour un perceptrons simple est :

$$\frac{dJ}{dw}=\frac{1}{N}\sum_{n=0}^{N}(y(x_n)-t_p(c_n)(y(x_n)-y(x_n)^2)x_n$$

Nous pouvons voir que pour le premier cas ((2 entrées + biais)*1000) nous pouvons en extraire différents taux d'erreur en fonction du pas choisi ainsi que du nombre d'itérations.

pas \ nombre d'itérations	i=10	i=100	i=1000
p=1*10e-0	Tau= 5.7%	Tau= 0.3 %	Tau= 0.03%
p=1*10e-2	Tau = 48.8%	Tau = 23.4%	Tau =4.5%

On remarque qu'il faut un coefficient rho assez élevé (de l'ordre de 1) afin de pouvoir converger assez rapidement vers les valeurs des poids w optimisées. Tout en faisant attention à ce que les valeurs w convergent effectivement (c.f figure 2) et que le gradient converge vers le vecteur nul également (c.f figure 3). Il faudra 1000 itérations pour avoir un taux d'erreur minimal, l'expérience nous montre qu'au-delà de 1000 itérations nous ne pouvons pas en tirer un meilleur taux de réussite.

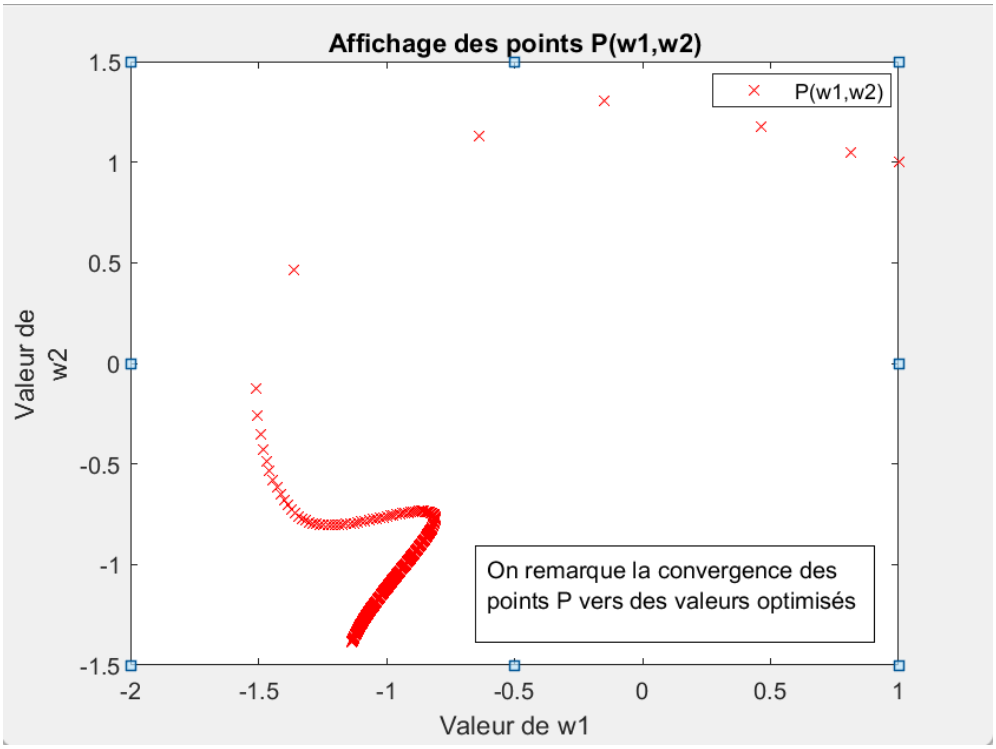


Figure 1: Affichage des points P(w1,w2) pour 1000 itérations

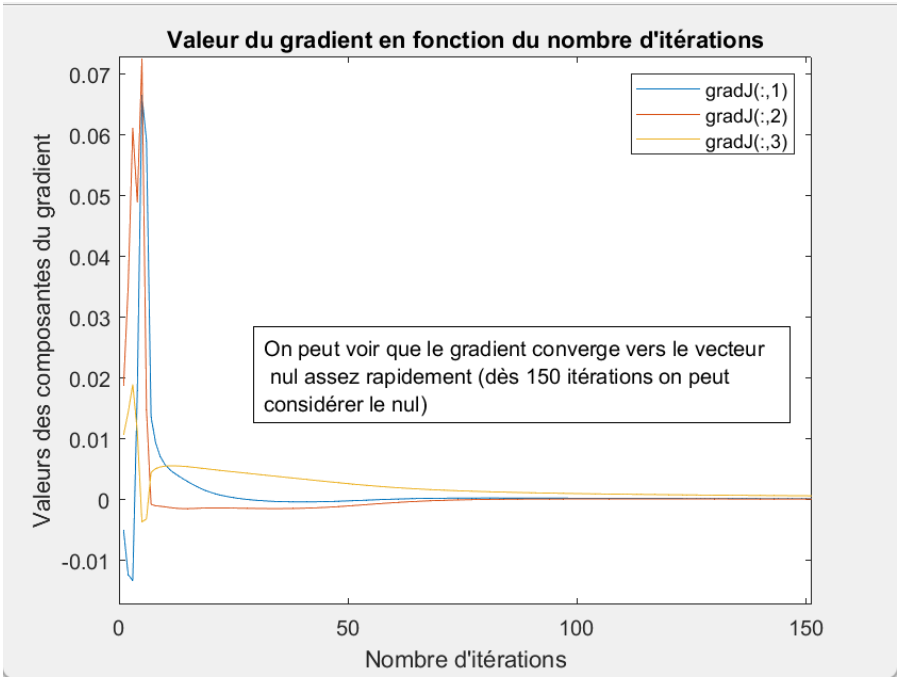


Figure 2: Valeur du gradient de J en fonction du nombre d'itérations

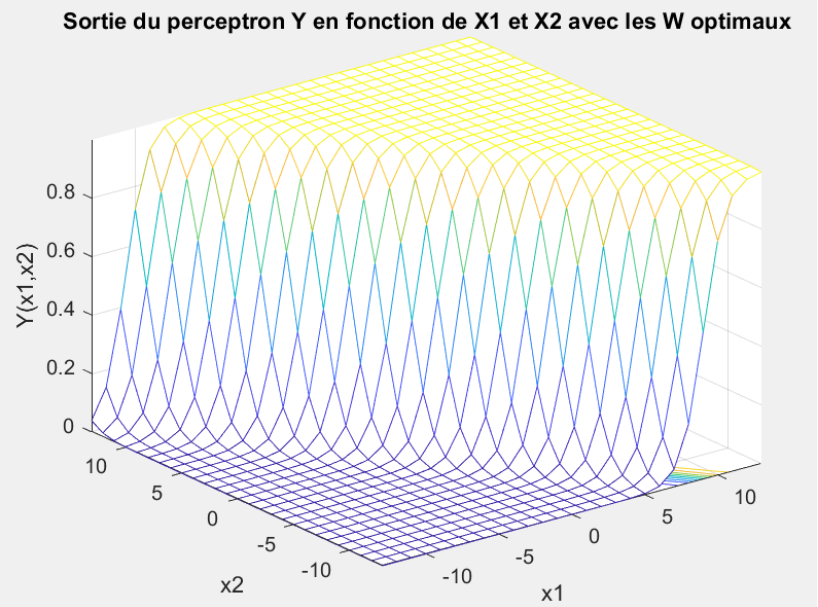


Figure 3; Evaluation de Y en fonction de x1 et x2 avec les W optimaux

Cas de figure avec 2000 entrées:

Résultats: Nous pouvons voir que pour le deuxième cas (2000 entrées) on trouve un taux d'erreur d'environ 24,95%.

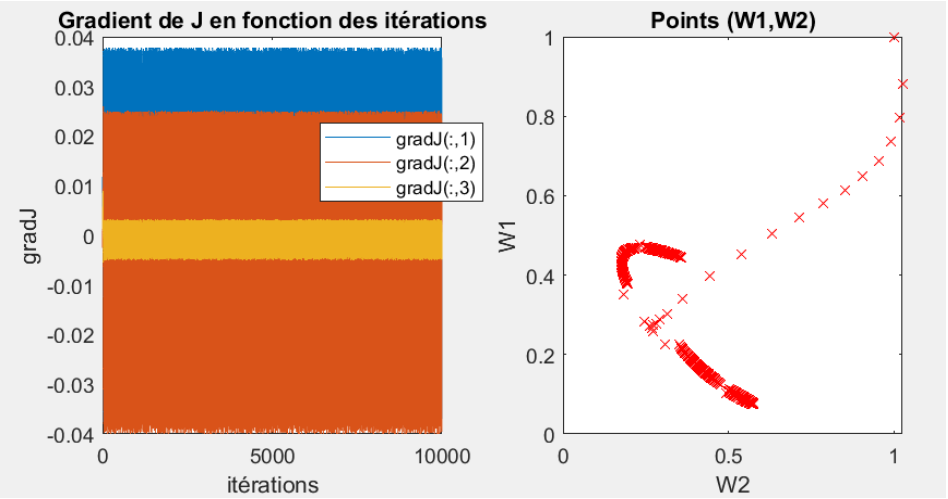


Figure 4: Valeur des poids W1 et W2 et évolution de gradJ en fonction des itérations

Analyse des résultats:

On remarque que plus on augmente le nombre d'itérations plus le taux d'erreur diminue ce qui s'explique par le faite que les W sont plus proche de la valeur optimal.Mais à partir de 150 itérations le taux d'erreur stagne car les W ont atteint leurs valeurs optimal (le gradient est presque nul) .

De plus on remarque dans le deuxième cas que si on augmente le nombre d'entrées le taux d'erreur augmente c'est le phénomène de surentraînement en effet il y a trop de données d'entrée par rapport à la complexité du problème. Le gradient ne converge pas vers 0 et par conséquent les poids W ne convergent pas vers des points optimaux. En réalité on ne trouve jamais des points optimaux, c'est peut-être les limites du modèle à 2 entrées et un perceptron.

Classification manuscrite :

Dans le cas du perceptron simple il n'y a que deux classes possibles donc on comparera deux chiffres.

Le principe ici est d'utiliser le perceptron simple pour des couples de chiffres manuscrits et ainsi voir si le perceptron reconnaît assez bien les images des différents chiffres manuscrits.

Le principe est le même que précédemment la différence est que nous créons nous même les sorties attendues en associant la classe 0 a un chiffre et la classe 1 a l'autre chiffre.

On a donc effectué différents calculs de taux d'erreurs pour différents couples de chiffres avec un entraînement sur 300 itérations et rho=10e-2 :

couple	(1,2)	(3,4)	(4,7)	(1,7)	(5,6)	(8,9)	(6,8)	(1,9)
taux d'erreur	4.7%	3,4%	6.1%	3,1%	7.5%	6.7%	4.7	2.2%

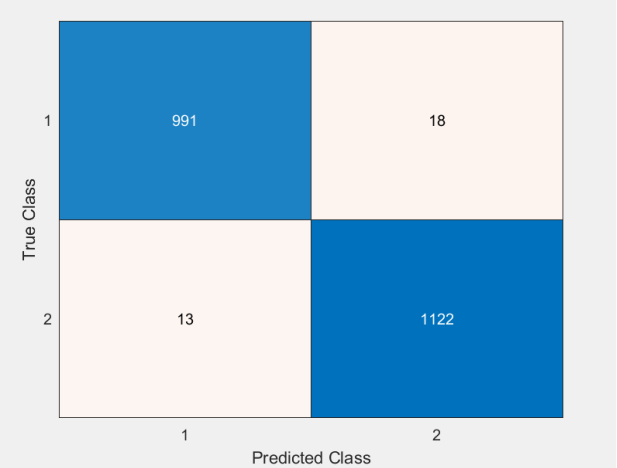


Figure 5: Matrice de confusion

Analyse des résultats :

On constate que le taux d'erreur dépend du couple choisi, en effet, nous avons un taux d'erreur qui semble être petit pour des chiffres qui ne se ressemblent pas, comme par exemple le couple (1,9) ou le couple (1,7). À priori on pourrait croire que 1 et 7 se ressemblent, mais tout dépend de comment on écrit 1 (une barre verticale ou "1"...). En revanche, on constate que le taux d'erreur est élevé pour des chiffres qui sont à priori semblables. En effet, le couple (5,6) présente le taux d'erreur le plus élevé car les formes se ressemblent fortement, suivi par le couple (8,9).

2) Classification multiclasse par monocouche

Ici nous allons faire la classification multiclasse par monocouche. Lorsqu'on voit comment les perceptrons sont calculés à partir des entrées X, on remarque qu'il y a indépendance entre les différentes sorties Y. En effet, pour calculer la sortie d'un perceptron, nous avons

besoin des poids Wij correspondant à l'entrée i et au perceptron j, donc ce poids est unique pour un seul perceptron.
À partir de là, on peut faire de façon analogue avec la méthode un seul perceptron en considérant désormais qu'un perceptron ne s'active qu'avec un seul chiffre et ne s'active pas pour le reste. Donc au lieu d'avoir 2 chiffres, on peut le faire avec 10 chiffres, 1 contre tous. Il nous faudra donc 10 perceptrons afin de pouvoir
reconnaître l'ensemble des chiffres de 0 à 9. Un seul perceptron ne s'entraîne à s'activer qu'avec un seul chiffre.

Les résultats:

chiffre	0	1	2	3	4	5	6	7	8	9
taux d'erreur en %	3.0	3.2	5.8	9.1	5.1	8.8	9.1	3.4	9.9	6.9

Tableau 1: Taux d'erreur pour 500 itérations et rho=10e-2

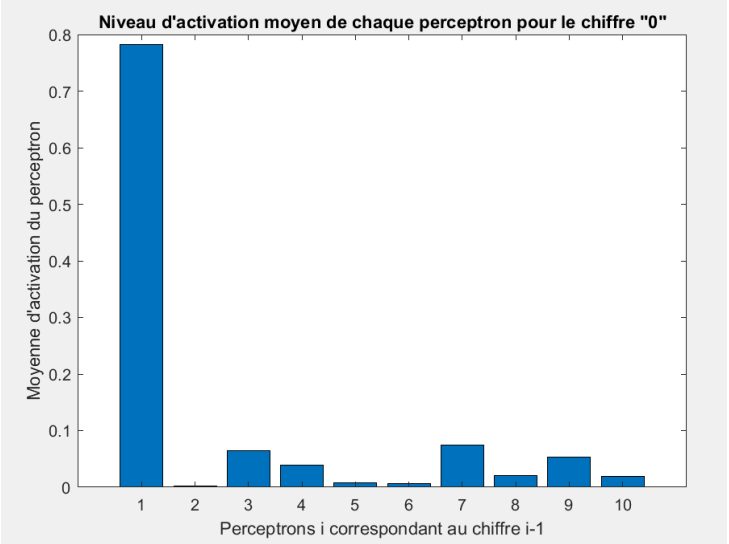


Figure 6: Niveau d'activation moyen de chaque perceptron pour le chiffre "0".

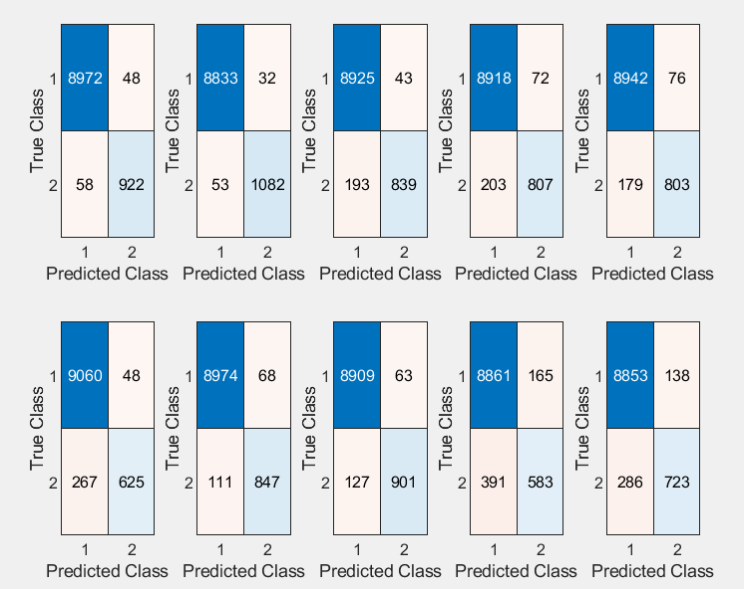


Figure 7: Matrices de confusion pour chaque chiffre/perceptron dans l'ordre croissant (gauche à droite puis haut bas).

Analyse des résultats : On remarque dans le tableau 1, de manière similaire à l'étude des couples de chiffres, que les chiffres qui se ressemblent le plus entre eux ont des taux d'erreur les plus élevés parmi les 10 chiffres. Nous nous sommes ensuite intéressés au niveau d'activation des différents perceptrons sur des tests d'un seul chiffre afin de confirmer cette tendance. On peut constater figure 4, qu'après le perceptron correspondant au chiffre "0" c'est ceux des chiffres "6", "3" et "8" qui ont des valeurs élevées par rapport au reste. Cela confirme notre hypothèse car ces 4 chiffres se ressemblent dans les formes qui les constituent.
Nous nous sommes ensuite intéressés à la matrice de confusion pour chaque perceptron. Dans la figure 6, on remarque que quand il s'agit de reconnaître le chiffre, alors le taux de réussite n'est pas assez élevé comparé au taux de réussite pour reconnaître que ce n'est pas le bon chiffre. Il est peut-être assez difficile de faire la différence lorsqu'il y a des chiffres assez semblables tels que le 5 et le 8 qui présentent un taux bas de réussite pour reconnaître le bon chiffre.

Conclusion:

Nous avons vu que pour un réseau de neurones composé de 2 entrées et une sortie (un seul perceptron) nous pouvons trouver des poids optimaux pour un certain nombre de données pouvant donner un taux de réussite satisfaisant (de l'ordre de 3%), cependant, lorsque l'on augmente les données auxquelles le réseau s'entraîne cela peut être néfaste et impliquer une augmentation du taux d'erreur sans possibilité de le réduire malgré la recherche de poids optimaux (que l'algorithme ne peut pas trouver d'ailleurs). Nous avons vu ensuite la classification pour des chiffres manuscrits, nous avons remarqué que le taux d'erreur dépendait des chiffres considérés, plus ils se ressemblent et plus le taux d'erreur est susceptible d'être élevé. Finalement nous en avons déduit, à partir de ce modèle, un autre modèle permettant de réaliser une multi-classification avec plusieurs perceptrons, les mêmes observations ont été relevées concernant le taux d'erreur et la ressemblance.