Rapport Intelligence Artificielle Embarquée

Oscar Gigon - ING3 INEM

gigonoscar@cy-tech.fr

4 mars 2023

Table des matières

Première partie

Introduction

Dans le cadre du cours d'Intelligence artificielle embarquée, nous avons programmé en langage C un réseau de neurones capable de différencier les motifs du chiffre 0 et ceux du chiffre 1. Ces motifs sont représentés par des grilles de 6x8 pixels, où chaque pixel peut être soit blanc (représenté par un caractère '.') soit noir (représenté par un caractère '*'). Chaque motif est associé à une sortie désirée fixée arbitrairement à 0 pour le chiffre 0 et à 1 pour le chiffre 1.

Nous allons implémenter un algorithme appelé perceptron simple, qui permet d'ajuster les poids des neurones du réseau en fonction de l'erreur commise sur les sorties désirées. Nous allons également proposer une variante efficace de ce modèle appelée "neurones à biais", qui permet d'ajouter un terme de biais aux entrées du réseau.

Nous allons ensuite évaluer la qualité de l'apprentissage en construisant des courbes de généralisation pour les deux motifs. Ces courbes permettent de mesurer la capacité du réseau à reconnaître des entrées bruitées, c'est-à-dire des motifs où une partie des pixels a été inversée de manière aléatoire.

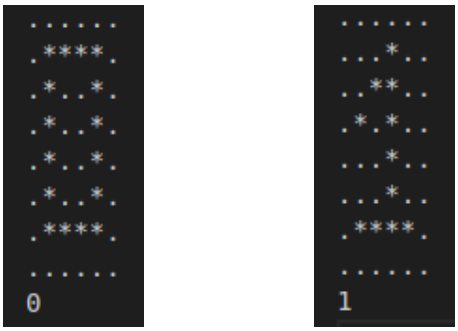
Deuxième partie

Développement

1 Séparation de deux classes, règle du perceptron simple

1.1 Au préalable

Avant de développer l'algorithme du perceptron, nous avons commencé par créer deux fichiers : zero.txt et un.txt. En pensant à ajouter la valeur de classe Y_d^k en fin de fichier.



Nous créons maintenant deux tableaux inputs[48] et weights[48] pour les données d'entrée et les poids du motif, nous avons défini la taille de nos tableaux à 48 car nos motifs ont en tout 48 "pixels". Puis avec la fonction $init\_weights()$, on initialise les poids du motif en utilisant une distribution uniforme aléatoire centrée sur zéro.

Nous avons ensuite programmé la fonction $read\_file()$ qui nous permet de lire un fichier texte contenant l'un des motifs et de le retranscrire dans le tableau des entrées en remplaçant '.' par 0 et '*' par 1.

Nous définissons également deux constantes, $\theta = 0,5$ et $\varepsilon = 0,01$, utilisées plus tard dans le calcul du potentiel du neurone et dans l'apprentissage du modèle.

1.2 Algorithme

En premier, nous avons ajouté au programme une fonction $rand01()$ qui va nous permettre de tirer un entier entre 0 et 1, elle nous permettra de lire aléatoirement l'un des

deux fichiers txt (zero.txt ou un.txt). Nous avons également créé une fonction nous permettant de mémoriser la valeur de classe Y_d^k qui se trouvait à la fin du fichier txt. Une fois que nous avons lu l'un des fichiers grâce à la fonction $read\_file()$, comme expliqué précédemment, cette fonction lit le fichier zero.txt ou un.txt en fonction de la valeur de rd (0 pour zero.txt, 1 pour un.txt) mais elle nous permet aussi de propager le motif lu sur la rétine. Cette fonction retranscrit dans le tableau des entrées en remplaçant '.' par 0 et '*' par 1.

Nous calculons maintenant le potentiel du neurone de sortie avec la fonction $neural\_pot()$ qui prend en paramètre les entrées du réseau et les poids des connexions initialisés avec $init\_weights()$. Elle utilise la somme pondérée des entrées et des poids, puis soustrait la valeur du seuil θ.

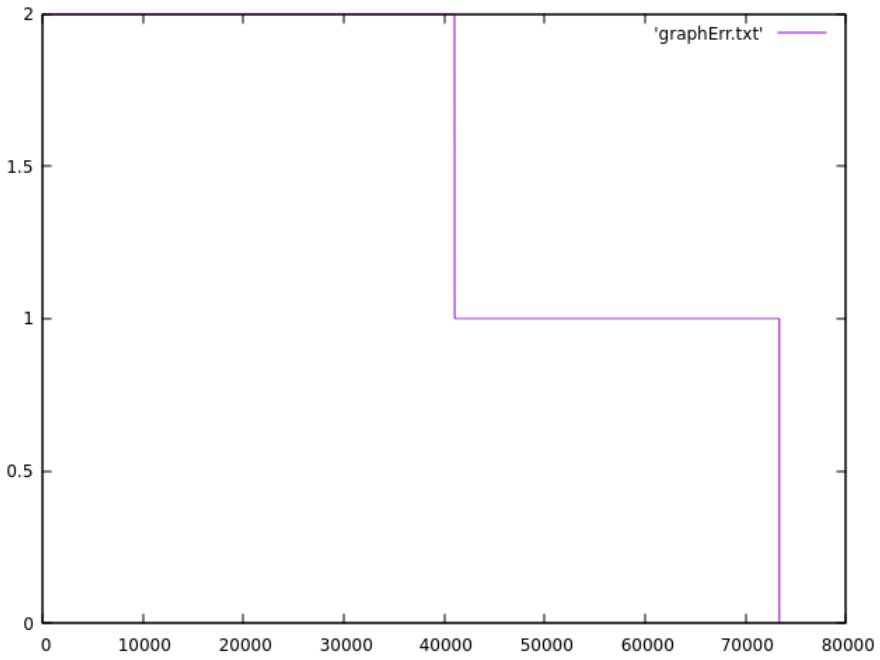
Nous utilisons maintenant le potentiel du neurone de sortie obtenu comme paramètre pour calculer la réponse du neurone de sortie. Cette fonction vérifie si le potentiel de neurone de sortie est supérieur à 0. Si oui, la fonction retourne 1 et sinon elle retourne 0.

Ensuite, nous calculons l'erreur entre la sortie attendue et la sortie réelle avec la fonction $error\_calc()$ ou X est la sortie réelle et Y la sortie attendue. Cette fonction soustrait la sortie réelle de la sortie attendue et retourne ensuite l'erreur sous forme d'entier.

Après cette fonction, nous ajustons les poids du réseau de neurones en fonction de l'erreur commise avec la fonction $learn()$. Les poids sont mis à jour en ajoutant un petit facteur (eps) multiplié par l'erreur et les entrées correspondantes. Elle prend en entrée l'erreur err et les entrées inputs.

Enfin, nous calculons l'erreur totale commise par le réseau de neurones en utilisant toutes les images de chiffres manuscrits disponibles (soit les fichiers "zero.txt" et "un.txt") avec la fonction $error\_tot()$. Elle initialise les poids du réseau de neurones de manière aléatoire pour chaque image, puis calcule l'erreur commise par le neurone de sortie pour cette image. Elle renvoie la somme de ces erreurs pour toutes les images.

Voilà un graphe qui nous montre l'évolution de l'erreur totale au cours de l'apprentissage :



Vous pouvez voir ici l'erreur totale sur l'axe des ordonnées et le nombre d'itération qui correspond à la durée de l'apprentissage sur l'axe des abscisses.

1.3 Généralisation

Dans cette partie sur la généralisation, la première étape a été de créer une fonction capable d'ajouter du bruit aléatoirement sur un motif. Cette fonction, nommée *invert_pixels*() prend en entrée un tableau de pixels et un pourcentage de bruit *noise_percent*(), et inverse aléatoirement la valeur de *noise_percent* des pixels dans le tableau. Cela simule l'ajout de bruit dans les données d'entrée pour le réseau neuronal. Elle utilise la fonction *rand*() pour sélectionner des pixels au hasard à inverser.

Ensuite, nous avons la fonction *count_errors*() qui calcule le nombre d'erreurs de classification commises par le réseau neuronal pour 50 motifs générés avec le même niveau de bruit. Elle prend en entrée un tableau avec des motifs et un pourcentage de bruit

noise_percentage. Pour chaque motif, elle applique la fonction *invert_pixels*() pour ajouter du bruit aux données d'entrée, puis elle calcule la sortie du réseau neuronal pour ce motif et compare la sortie à la vérité terrain pour déterminer s'il y a une erreur de classification. Le nombre total d'erreurs de classification est ensuite retourné.

Enfin, pour pouvoir construire la courbe de généralisation des motifs, nous avions besoin de la fonction *generalization_curve*(). Elle prend en entrée un tableau avec des motifs et l'indice du motif à tracer $motif\_index$. Pour chaque niveau de bruit de 0 à 100% par incréments de 10%, elle appelle la fonction *count_errors*() pour calculer le nombre d'erreurs de classification commises par le réseau neuronal pour 50 motifs générés avec ce niveau de bruit. Elle affiche également le taux d'erreur correspondant pour chaque niveau de bruit et écrit les résultats dans un fichier texte graphErrBruit0.txt ou graphErrBruit1.txt en fonction de l'indice du motif.

Pour le bon fonctionnement des fonctions précédentes, nous avions besoin d'une fonction qui remplit un tableau avec plusieurs motifs à l'intérieur. Cette fonction, $generate\_motifs()$, génère des motifs en créant un tableau à deux dimensions motifs de taille 50x48, où chaque ligne correspond à un motif. Les 25 premières lignes contiennent des copies du premier motif motif0, et les 25 suivantes contiennent des copies du deuxième motif motif1.

Enfin, nous avons ajouté la fonction $free\_motifs()$, cette fonction est utilisée pour libérer la mémoire allouée pour le tableau avec des motifs. Elle parcourt chaque ligne du tableau et utilise la fonction $free()$ pour libérer la mémoire de chaque ligne, puis libère la mémoire du tableau principal.

Voici les courbes de généralisation des motifs 0 et 1 avec en abscisse le pourcentage d'entrées bruitées et en ordonnée le taux d'erreur.

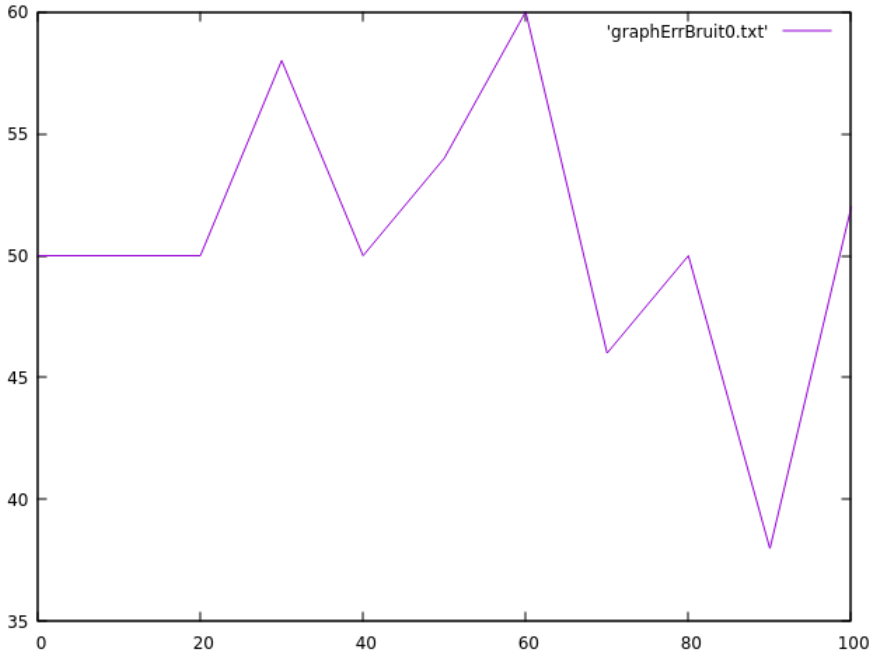


FIGURE 1 – Courbe de généralisation du motif 0

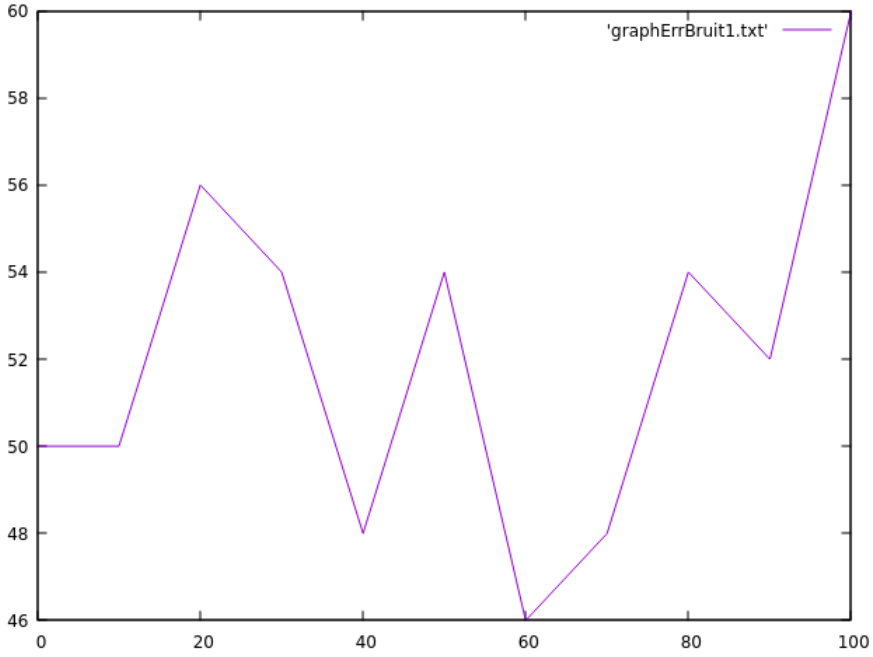


FIGURE 2 – Courbe de généralisation du motif 1

Troisième partie

Conclusion

En conclusion, dans ce devoir nous avons implémenté un réseau de neurones en langage C pour différencier les motifs du chiffre 0 et du chiffre 1 représentés par des grilles de 6x8 pixels. Nous avons utilisé l'algorithme du perceptron simple. Nous avons évalué la qualité de l'apprentissage en construisant des courbes de généralisation pour les deux motifs, afin de mesurer la capacité du réseau à reconnaître des entrées bruitées. Ce cours nous aura finalement permis de concevoir et de comprendre le principe de perceptron. Nous avons ainsi acquis des compétences en Intelligence artificielle nous permettant de comprendre les réseaux modernes.