République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université des Sciences et de la Technologie Houari Boumediene

Faculté d'Electronique et d'Informatique
Département informatique

Projet AARN

Conception et réalisation d'un réseau de neurones pour l'apprentissage automatique d'un diagnostic médical.

Réalisé par :
AYAD Wafa
BELKACEMI Imene

Introduction :

L'utilisation des réseaux de neurones est l'une des techniques les plus efficaces pour effectuer l'apprentissage automatique sur des données appartenant à un domaine quelconque. Lors de ce projet, nous allons montrer le schéma conceptuel que nous avons mis en place afin de défendre certaines architectures que nous jugeons efficaces pour réaliser un meilleur apprentissage sur les données relatives au domaine du diagnostic médical.
Plus précisément, nous allons établir plusieurs réseaux de neurones et leurs appliquer un apprentissage automatique supervisé dans le but d'apprendre la présence ou non d'un cancer tumeur, tout en utilisant le jeu de données que nous possédons et qui décrit les variations des SNPs (Single Nucleotide Polymorphism) chez un ensemble d'individus donné. Ces réseaux, une fois réalisés, doivent être capables d'affirmer la maladie ou non d'une personne donnée.
Sur cette base, nous aborderons deux parties dans ce rapport. Une partie pour expliquer notre conception et une autre pour présenter les résultats obtenus. La première partie est de son côté subdivisée en deux sous parties, nous discutons d'abord nos données d'apprentissage, puis présentons le script développé pour obtenir les résultats les plus performants.

I. Conception :

1. Données d'apprentissage :

Nous disposons de deux fichiers .txt :

- 'NNInPut' : contenant 111 colonnes sur 42 attributs, l'information se présente comme ceci : 00, 01, 10 ou 11.
  La valeur 00 signifie que la donnée est manquante. Mais ça ne pose pas problème car nous savons que les réseaux de neurones sont robustes contre le bruit.
- 'NNTarget' : contenant 111 lignes sur un seul attribut (0 : cancer / 1 : pas de cancer). Cette matrice doit être transposée pour qu'elle soit cohérente avec la matrice contenant les données d'entrée.

1.1. Codage des données :

Nous avons normalisé les informations contenues dans le fichier NNInPut pour avoir des valeurs comprises entre 0 et 1 afin d'obtenir des résultats améliorés.
Le programme qui permet de normaliser les valeurs, développé en Java, est disponible sur le CD.

1.2. Sélection et préparation des données :

1- Test de l'ensemble d'apprentissage initial :

Après un certain nombre d'essais, nous avons constaté que le sur-apprentissage se produit même avec des réseaux de neurones ayant une très simple architecture (disons une seule couche cachée contenant deux neurones), c'est dû au nombre d'instances très petit (111) par rapport au nombre d'attributs (42).

En effet, le sur-apprentissage se caractérise par une erreur très petite lors de l'apprentissage, par contre quand de nouvelles données sont présentées devant lui, l'erreur est trop grandes (les courbes validation/test et train divergent). En d'autres termes, le réseau mémorise les exemples de test mais il n'apprend pas à généraliser pour toutes les nouvelles situations.

2- Amélioration des données de test :

Pour pallier au problème du sur-apprentissage rencontré ci-dessus, nous devons avoir beaucoup plus d'instances dans les données d'entrée, distinctes de préférence pour qu'il puisse généraliser à partir d'exemples (Principe d'abduction).

Nous n'avons malheureusement pas pu obtenir de nouvelles instances, bien que ça représente la solution la plus adéquate. Donc, nous proposons de rajouter les mêmes instances 3 fois en respectant les indices entre Input et Target en exécutant la commande suivante deux fois sur les deux matrices (NNInPut et NNTarget (transposée)) :

```
Input = [Input Input] ;
```

En effet, nous avons utilisé la fonction 'dividerand' pour assurer l'association correcte des indices entre l'Input et l'Output lors de la division de l'ensemble d'apprentissage entre données d'apprentissage, de test et de validation.

2. Présentation du script proposé :

Nous avons fait en sorte de varier plusieurs métriques dans le but de déduire la configuration la plus adéquate à la résolution de ce problème.

Le tableau suivant montre ces paramètres et les valeurs qu'elles prennent :

Paramètre	Valeurs
Fonction d'apprentissage	trainlm, trainrc, trainscg
Fonction de performance	sse, mse
Nombre de neurones (une seule couche cachée)	De 1 à 10
Nombre de neurones (deux couches cachées)	De 1 à 5
Pourcentage de partitionnement (trainRatio)	De 50 à 90 (pas de 10)

La variation s'effectue sur la fonction de performance, la fonction d'apprentissage et le taux de partitionnement à chaque fois. Une portion des données est dédiée à l'apprentissage, cette valeur correspond au pourcentage de partitionnement (trainRatio), le reste est distribué équitablement entre l'ensemble de test et l'ensemble de validation.
Nous avons exécuté cela non seulement pour une seule couche cachée ayant un nombre de neurones allant de un à dix, mais encore, pour deux couches cachées avec de un à cinq neurones dans chacune des deux couches.

La fonction 'ntrain' (disponible sur le fichier ntrain.m) refait l'apprentissage sur le même réseau de neurones dix fois tout en retournant le meilleur.
Nous tenons aussi à simuler chaque meilleur réseau de neurones dans la fonction ntrain. La différence entre les résultats et NNTarget est au voisinage de zéro, ce qui signifie que le taux d'erreur est minime.

II. Résultats obtenus :

Les affichages obtenus lors de l'exécution du script représentent nos résultats. Deux fichiers les contenant sont disponibles sur le CD. (resultats 1cc.txt et resultats 2cc.txt).
Le meilleur résultat est obtenu avec la fonction 'sse' bien que au cours des premières itérations, nous remarquons que la fonction 'mse' donne des résultats plus performants. Et aussi, nous l'avons eu avec une seule couche cachée.

Le tableau suivant récapitule brièvement ce que nous avons obtenu par la fonction 'sse' dans le cas d'une seule couche cachée à l'itération de l'obtention du meilleur résultat (une performance de l'ordre e-28) :

TrainRatio	70%			80%		
TrainFcn	trainrp(10nr)	trainscg(10nr)	trainlm(08nr)	trainrp(10nr)	trainscg(10nr)	trainlm(09nr)
Performance	0.3064	6.9253e-06	3.5631e-28	0.2955	2.9162e-04	2.8409e-28
Regression	0.9980	1.0000	1.0000	0.9979	1.0000	1.0000

Tableau 1 : Tableau récapitulatif des meilleurs résultats obtenus.

Un des graphes visualisés lors de l'exécution est disponible sur l'annexe.

Constatations :
Meilleur résultat obtenu :

- Performance : 2.8409e-28.
- Régression : 1.

D'après ces résultats expérimentaux, nous jugeons l'architecture suivante étant la plus adéquate pour la résolution de ce problème (diagnostic médical) :

- Fonction de performance : 'sse'.
- Fonction d'apprentissage : 'trainlm'.
- Architecture du réseau de neurones : 'Une seule couche cachée avec dix neurones'.
- Partitionnement de l'ensemble d'apprentissage : 'TrainRatio' = 80%, 'valRatio' = 10%, 'testRatio' = 10%.

Conclusion :

Après l'étude que nous avons faite sur l'ensemble de données (NNInPut et NNTarget) et à travers les modifications effectuées pour obtenir les meilleurs résultats possibles (Performance et Régression) à cause de la *quantité insuffisante* des exemple que le réseaux utilise pour apprendre, nous avons pallier aux problèmes posés d'une façon à dupliquer les données pour lui permettre de voir beaucoup plus d'instances afin qu'il puisse prendre une bonne décision face aux nouvelles interrogations.
Pour finir, nous concluons que la sureté d'un bon apprentissage automatique est relative à la taille de l'ensemble d'apprentissage, et dans le cas d'insuffisance de cette importante ressource, les décisions prises par les meilleures architectures des réseaux de neurones qui peuvent être conçues sont de plus en plus moins fiables, sinon nous procédons à l'apprentissage par cœur qui se fait par la duplication des données d'apprentissage.

Annexe :

A. Annexe 1 : Un des graphes visualisés lors de l'exécution du script.

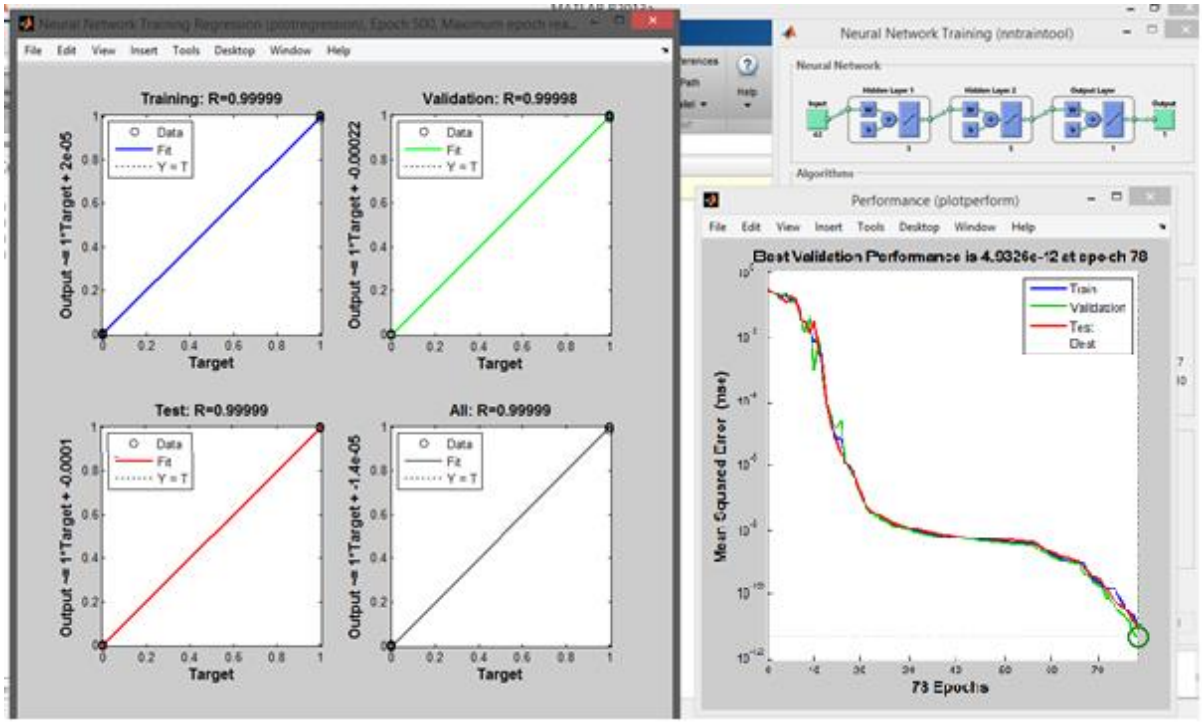

Figure 1 : Performance et régression obtenus par une architecture de deux couches cachées ayant cinq neurones dans chacune.

Répartition des tâches :

- Conception et développement du script : fait ensemble.
- Rédaction du rapport : fait ensemble.
- Finition : fait ensemble.