Licence 3
INFO-MIAGE

Apprentissage artificielle

THOMAS Johann

KHALED Sara

I) Structure du projet

Dans le cadre de la conception d'un réseau de neurones, nous avons opté pour un modèle de programmation objet. Ce modèle définit des classes permettant d'organiser la structure de notre réseau de neurones. Elles sont au nombre de six, il y a deux classes représentant les neurones, trois classes représentant les couches et une représentant le réseau dans son ensemble. Les classes neurones se décomposent entre la classe de neuronePremier et la classe neurone. Les classes de couches se décomposent entre les classes coucheInitiale, coucheFinale et couche.

L'organisation générale de ces couches est représentée en annexe (fig 1).

Classes Neurones

Les classes Neurones sont au nombre de 2. La classe générale neurone qui définit le comportement général d'un neurone et la classe neuronePremier qui hérite de la première mais qui se distingue par le fait que les entrées des neuronePremier sont des valeurs entre 0 et 1.

Chaque neurone contient une liste de ses prédécesseurs associée à des poids. La classe neurone contient ses prédécesseurs dans une liste associative où l'index correspond au neurone et la valeur au poids. La classe neurone premier contient une liste de tableaux où l'index correspond à l'entrée et la valeur un tableau ou la première case est la valeur de l'entrée et la deuxième le poids. De ce fait, la mise à jour des poids durant l'apprentissage et le calcul de la valeur de sortie s'en trouve simplifiée à l'aide de méthodes.

Les méthodes et attributs sont détaillés en annexe (fig 2).

Classes Couches

Les classes couches sont au nombre de 3. La classe générale couche définit le comportement d'une couche intermédiaire, les classes coucheInitiale et coucheFinale héritent de cette classe couche.

Chaque couche contient sa couche précédente dans le réseau sauf la coucheInitiale qui contient une la liste d'entrée du réseau. Elles contiennent aussi une liste de neurones. Dans le cas de la coucheFinale, il s'agit d'un seul neurone. Ainsi, la couche est capable d'induire les opérations de modification à ses neurones et aux couches précédentes à l'aide de méthodes.

Les méthodes et attributs sont détaillés en annexe (fig 3).

Classe Réseau

Il s'agit de la classe finale de notre projet, elle contient toutes les classes précédemment citées. Elle dispose d'une couche initiale, d'une couche finale et d'une liste de couches.

La classe réseau dispose dans ses attributs d'éléments permettant de paramètrer et de faire varier l'apprentissage du réseau.

Les méthodes et attributs sont détaillés en annexe (fig 4).

Autres classes

Dans le but de tester le programme, nous avons créé une classe apprentissage, une autre apprentissage Batch et une dernière classe test permettant soit d'entraîner soit de tester un réseau de neurones.

II) Modalités d'apprentissages

Dans la réalisation de ce projet, nous avons choisi d'implémenter un programme permettant l'apprentissage en batch et un autre permettant l'apprentissage en stochastique, ainsi il est possible de réaliser les deux et donc de les comparer. En ce qui concerne le processus de validation des données, nous utilisons une partie des données d'entraînement afin d'observer le comportement du réseau.

Dans le but d'entraîner notre réseau de neurones, nous avons défini trois jeux de données de difficultés croissante. Le premier concerne une simple fonction logique XOR, le deuxième concerne une association de deux fonctions logique XOR et NAND et enfin la dernière concerne un dataset répertoriant le résultat obtenue par des étudiant au SAT selon le niveau d'éducation des parents, le revenus des parents, leur GPA au lycée et à l'université ainsi que le nombre d'année qu'il le reste à étudier (les données ont été normalisées en divisant les valeurs numérique et par substitution du niveau d'éducation des parents par une valeur numérique).

III) Résultats des tests

Au cours de ces tests, nous avons utilisé plusieurs types de données et nous avons fait varier les paramètres du réseau de neurones. Ainsi nous présenterons les configurations qui ont permis d'obtenir le plus rapidement un résultat satisfaisant.

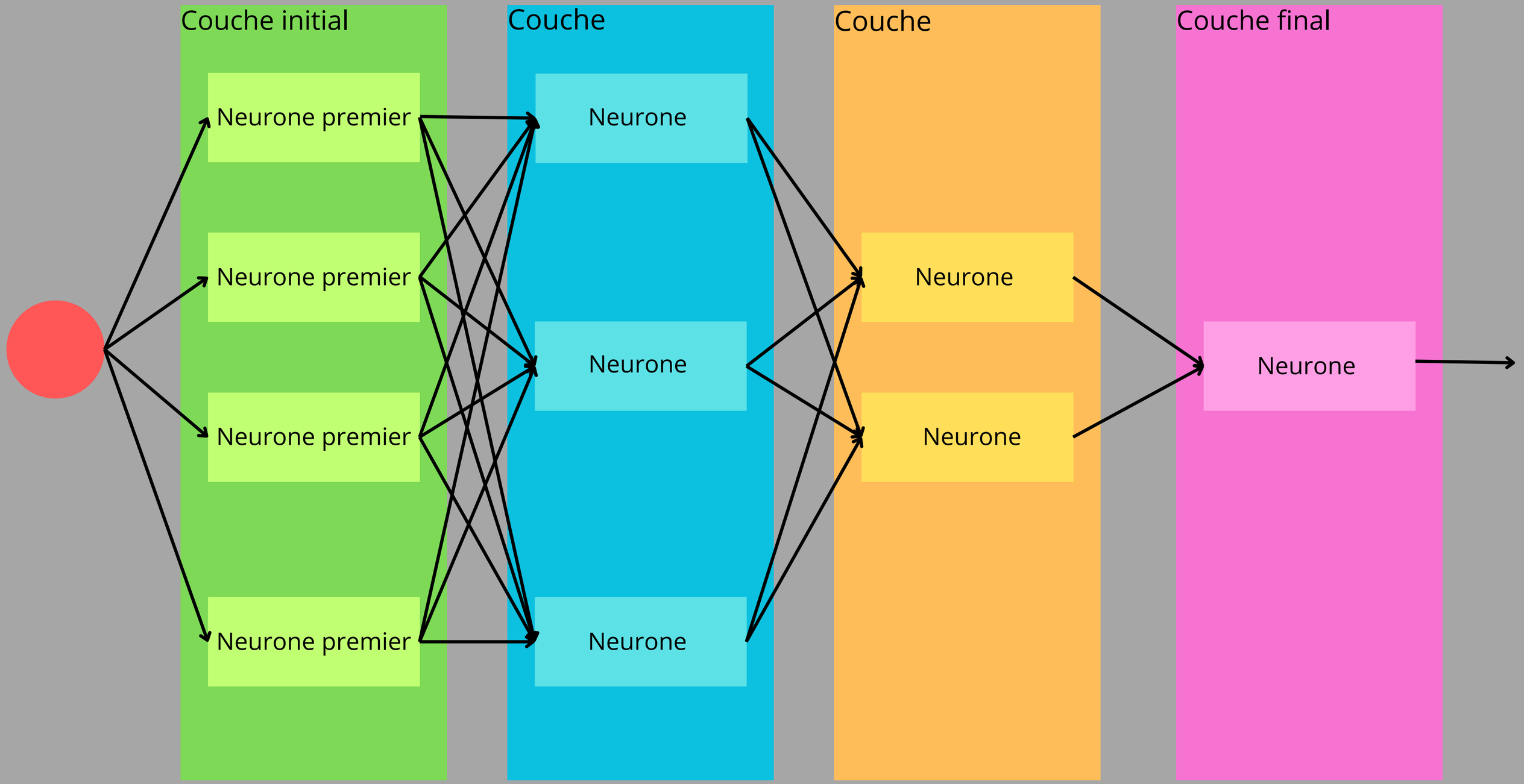
Dans un premier temps, nous avons réalisé le test sur la fonction XOR en mode batch et stochastique. En ce qui concerne le mode stochastique, les tests ont révélé que la meilleure configuration était sur un réseau de neurone à 3 couches, avec une marge d'erreur et un paramètre d'apprentissage à 0.1 et un paramètre de Sigmoïde à 4. En ce qui concerne le mode batch, les tests ont révélé que la meilleure configuration était sur un réseau de neurone à 4 couches, avec une marge d'erreur et un paramètre d'apprentissage à 0.2 et un paramètre de Sigmoïde à 5.

Au cours de nos tests sur la fonction XOR, le mode batch s'est révélé plus fiable que le mode stochastique puisque sur 6 essais le mode batch a réussi à deux reprises à apprendre le modèle alors que le mode stochastique ne l'a appris qu'une fois.

Dans un second temps, nous avons testé une fonction logique XORNAND, l'apprentissage en mode batch s'est révélé être plus efficace avec des réseaux de 3 couches et une marge d'erreur basse à 0.01 et un taux d'apprentissage à 0.3, le paramètre de la sigmoïde était de 5. En ce qui concerne le stochastique, il était beaucoup plus lent que le batch, les paramètres optimaux étaient un réseau de 3 couches, 0.1 de marge d'erreur et un taux d'apprentissage à 0.1, le paramètre de la sigmoïde était de 4. Finalement, il a fallu trois essais pour l'apprentissage batch et six pour le stochastique.

Enfin, nous avons testé notre réseau de neurones sur les données concernant les résultats au SAT. L'apprentissage en stochastique a été le plus efficace, avec un réseau de 5 couches, une marge d'erreur de 0.003, un taux d'apprentissage de 0.05 avec un paramètre de la sigmoïde à 4. Dans le cas du batch, il y eut de nombreux minimum locaux, il est difficile d'établir qu'une configuration a vraiment fonctionné mais on peut noter que le réseau disposait de 5 couches, d'un paramètre de sigmoïde à 4 et d'un taux d'apprentissage à 0.7. Ainsi l'apprentissage stochastique semble être plus adapté dans une situation de nombreuse données puisque le batch se retrouvait régulièrement dans des minimum locaux.

Réseau


Couche initial
Couche
Couche
Couche final
Neurone premier
Neurone premier
Neurone premier
Neurone premier
Neurone
Neurone
Neurone
Neurone
Neurone
Neurone

Neurone

listeEntree : dict - Tableau associatif indexé par l'entrée et dont la valeur est le poids
Val: float - Valeur de sortie du neurone
Erreur: float - Valeur erratique du neurone

- getValeur()
- getErreur()
- addEntree(val,poids) - Ajout d'une entrée
- removeEntree() - Retrait d'une entrée
- afficherListe() - Affichage des entrées
- calculSigmoide(paramSigmoïde) - Initiation du calcul de val
- calculErreurAttendue(valAttendue) - Calcul de l'erreur selon une valeur passée en paramètre
- calculErreur(listeSuivant,paramSigmoïde) - Calcul de l'erreur en mode stochastique
- calculErreurBatch(listeSuivant,paramSigmoïde) - Calcul de l'erreur en mode batch
- correctionPoids(coeffApp) - Correction des poids menant à ce neurone

Couche

preCouche : couche - Couche précédente
liste_neurone : liste_neurone - Liste des neurones composant la couche

- afficherListe() - Affichage des neurones de la couche
- initierCalcul(paramSigmoïde) - Appel de la méthode calculSigmoide() pour les neurones
- propagationErreur(liste,paramSigmoïde) - Appel de calculErreur() pour les neurones
- propagationErreurBatch(liste,paramSigmoïde) - Appel de calculErreurBatch() pour les neurones
- propagationErreurCorrection(coeffApp) - Appel de correctionPoids() pour les neurones

Réseau

valAttendue : float - valeur de sortie
tauxApprentissage : float - taux d'apprentissage
paramSigmoide : int - paramètre de la fonction sigmoïde pour accélérer l'apprentissage
margeErreur : float - marge d'erreur acceptable
listeDonnee : liste de float - liste des entrées du réseau
listeCouche : liste de couche - liste des couches composant le réseau
coucheInitiale : couche - première couche du réseau
coucheFinale: couche - dernière couche du réseau

- modifierListeDeDonnee(listeDonee:list) - modification de la liste par remplacement des valeurs
- modifierTauxApp(tauxApp:float) - modification du taux d'apprentissage
- modifierValAtt(valAtt:float) - modification de la valeur de sortie
- modifierMargeErreur(margeErreur:float) - modification de la marge d'erreur
- modifierParamSigmoide(paramSigmoide:float)
- getErreur() - récupération de l'erreur de sortie du réseau c'est à dire l'erreur du neurone de la couche de sortie
- afficherListeDonnee() - affichage des entrées
- afficherReseau() - affichages des couches du réseau
- initierCalcul() - initiation du calcul de la valeur de sortie du réseau
- initierCorrection() - initiation de l'apprentissage par descente de gradient
- initierCorrectionBatch(correct:bool) - initiation de l'apprentissage par descente de gradient en mode batch, le nombre de données à valider est indiquer en dehors du réseau