COMPTE RENDU BE4

Réseaux de neurones

I – Analyse d'un réseau de neuronnes

On utilise le code fourni dans l'énoncé :

On mémorise les valeurs de couche_sortie pour la première itération et pour la dernière. On obtient alors :

- ⇨ 1ère itération : [[0.2689864], [0.36375058], [0.23762817], [0.3262757]]
- ⇨ Dernière itération : [[0.00966449], [0.00786506], [0.99358898], [0.99211957]]

On remarque que les résultats obtenus à la première itération sont nettement plus proches des résultats attendus [0, 0, 1, 1]

On modifie la fonction proposée pour obtenir une trace de l'évolution des valeurs des erreurs sur chacune des valeurs du vecteur de la synapse au fur et à mesure des itérations.

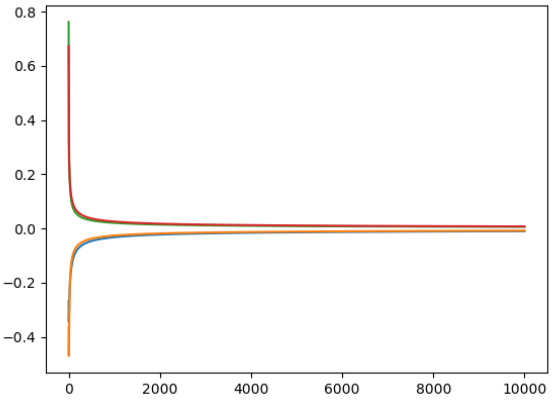

Fig 1 : Evolution de l'erreur sur 10000 itérations

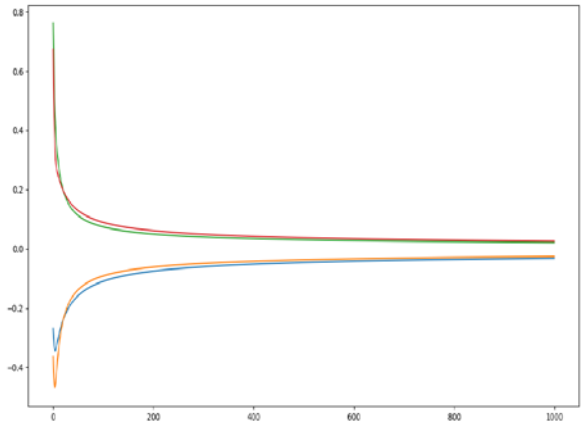

Fig 2 : Evolution de l'erreur sur 1000 itérations

On observe la décroissance (en valeur absolue) extrêmement rapide, qui semble tendre vers 0 de manière asymptotique. Si le réseau semble avoir un apprentissage rapide au début, son rythme d'apprentissage diminue au fil des itérations.

Pour la suite, on construit le tableau binaire des différentes combinaisons possibles pour les trois entrées (table de vérité) que l'on pose à la page suivante.

Table de vérité correspondant à Y=(x1 or (x1 and x3))

X1	X2	X3	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

On implante ces vecteurs d'entrée dans le code python, et on regarde le comportement du réseau vis-à-vis de ces nouveaux signaux. Là encore, en visualisant la courbe d'apprentissage (erreur en fonction des itérations), on observe deux phénomènes :

- ⇨ Le réseau se comporte comme précédemment pour presque toutes les entrées. Au bout d'un certain nombre d'itérations, la sortie converge vers le résultat attendu.
- ⇨ Dans le cas de l'entrée qui vaut le vecteur nul, le réseau n'arrive pas à corriger pour apprendre (étant donné que le produit scalaire est toujours nul, l'erreur ne varie pas).

Finalement, on peut bien affirmer que le réseau apprends en fonction de la corrélation entre les entrées et les sorties (ici unique)

II – Avec une couche cachée

On considère maintenant le cas où le réseau de neurones dispose d'une couche cachée afin d'analyser les caractéristiques de l'entrée. Là aussi, on peut afficher les courbes d'évolution de l'erreur :

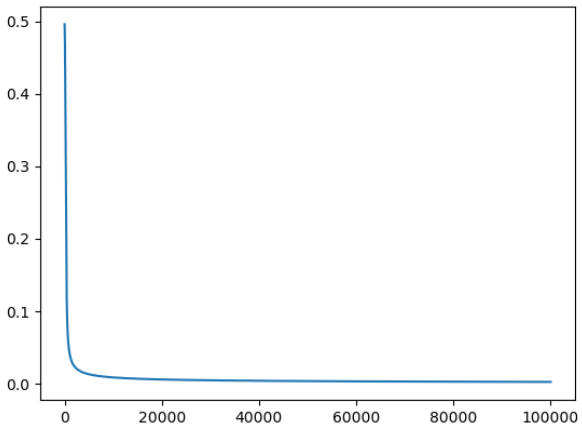

Fig. 3 : Evolution de l'erreur moyenne en fonction des itérations sur un réseau muni d'une couche cachée

On veut maintenant tester le réseau précédemment entrainé sur la table de vérité suivante :

X1	X2	X3	Y
1	0	0	1
0	0	0	0
0	1	0	1

On définit pour cela la fonction **test**(vecteur_entree) qui effectue le calcul du produit scalaire de synapse1 avec le vecteur d'entrée, puis utilise la fonction sigmoïde pour traduire le résultat

```
>>> test([[1,0,0],[0,0,0],[0,1,0]])
array([[ 1.        ],
       [ 0.5       ],
       [ 0.99999983]])

>>>
```

On constate que l'apprentissage à permit d'obtenir les résultats voulus, sauf pour le cas de l'entrée nulle, car il n'y a alors aucune corrélation entre les entrées et la sortie. C'est le même résultat que précédemment.

Table de vérité correspondant à Y=(X1 ^ X2 ^ /X3)

X1	X2	X3	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

On essaie également de comparer différentes fonctions indicatrices. On note ici le résultat obtenu à l'aide de la fonction tangente hyperbolique, qui est à mettre en regard avec la fig. 3. On remarque notamment que la convergence est moins rapide et plus chaotique.

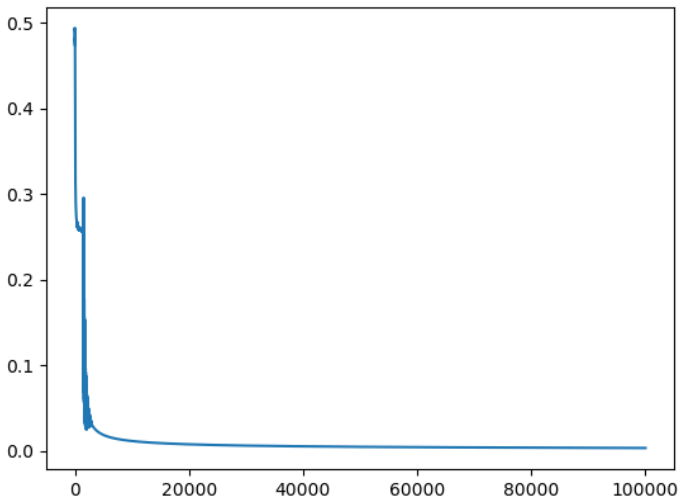

Fig. 4 : Courbe d'evolution de l'erreur pour la fonction tangeante hyperbolique

Si on s'intéresse à la convergence à l'aide de la fonction arctan(), on obtient :

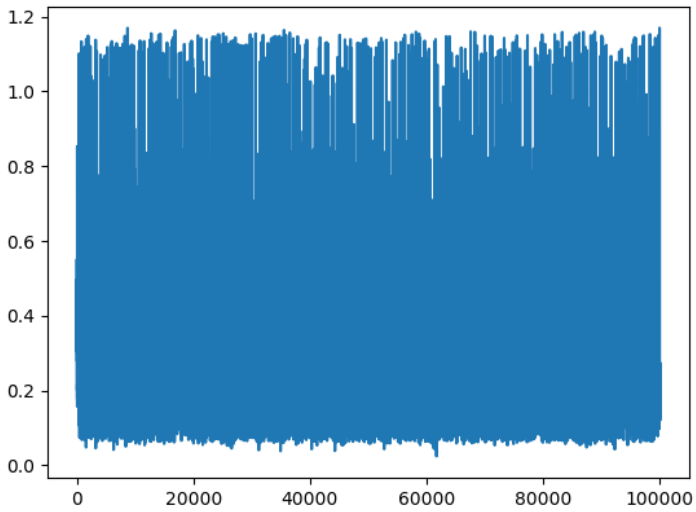

Fig. 5 : Courbe d'evolution de l'erreur pour la fonction arctan

Ce qui est le résultat visible d'un problème lié à la descente de gradient. Pour pallier ce phénomène, on implante un taux d'apprentissage alpha, pour diminuer la vitesse de descente de gradient et ainsi éviter les instabilités. On obtient alors :

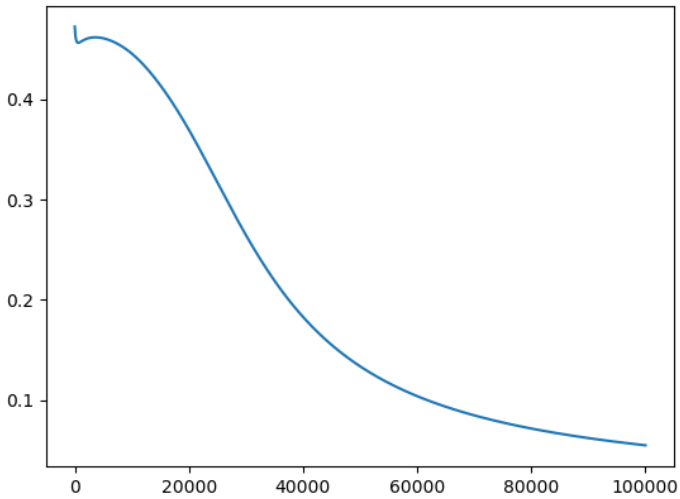

Fig. 6 : Courbe d'evolution de l'erreur pour la fonction arctan avec alpha=0.01

Ensuite, on essaie d'ajouter une couche supplémentaire dans le réseau neuronal. Pour cela, on rajoute un niveau composé de 3 neurones, chacun prenant 5 entrées.

Pour entrainer le réseau, on choisit de réaliser la table de l'addition de deux nombres en binaire naturel sur deux bits.

II – Construction d'un RN de reconnaissance des chiffres manuscrits

On crée un RN à 3 couches, prenant comme entrée un vecteur de 48 éléments (une matrice de 6x8 pixels aplatie) et donnant une réponse en sortie dans {0,1,…,9}.

- ⇨ Pour pouvoir réaliser un tel RN, on change la fonction d'activation
- ⇨ On lit les données nécessaires à l'entrainement dans le fichier train.data
- ⇨ On commence par un réseau avec 1 couche cachée

Pour commencer, il faut préparer les données. Pour cela on lit le fichier ligne a ligne, on convertit la ligne de donnée dans le format souhaité, en isolant la valeur de la solution qui est le dernier élément de la ligne.

On exécute ensuite le RN, et on s'aperçoit que celui-ci diverge. Il faut donc modifier le taux d'apprentissage afin d'obtenir un code fonctionnel.

On utilise dans un premier temps le facteur *alpha*, pour limiter la vitesse d'apprentissage. On obtient les résultats suivants :

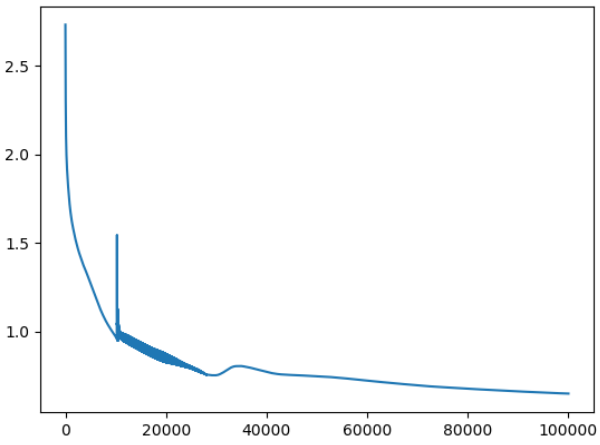

Fig. 7 : Evolution de l'erreur pour alpha=10^{-4}

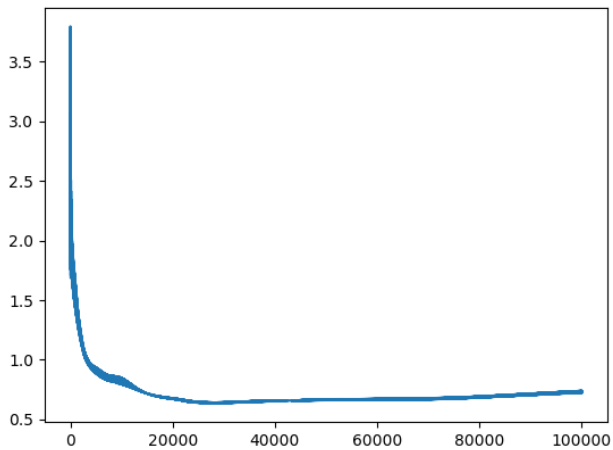

Fig. 8 : Evolution de l'erreur pour alpha=10^{-3}

On peut alors réaliser un balayage sur le paramètre alpha, et essayer de déterminer la variation de l'erreur finale en fonction de celui-ci. On reste pour l'instant dans le cadre d'un apprentissage sur 100000 itérations.

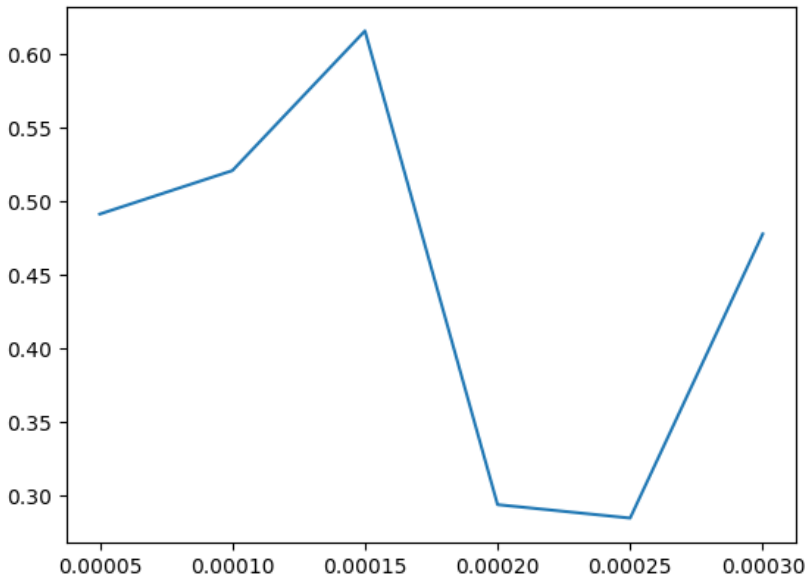

Fig. 9 : Evolution de la valeur de l'erreur finale d'apprentissage en fonction d'alpha