Data Mining 2
TP1 & 2 - Toolbox réseaux de neurones

Thomas Robert

1 Codage et test des fonctions

Afin de vérifier le fonctionnement de la toolbox, j'ai testé les diverses fonctions avec deux problèmes très simple : un problème de régression de la fonction $y = 2 \times x$, et un problème de classification de dimension 1.

Le réseau testé est un réseau MLP avec une couche cachée, 1 entrée et 1 ou 2 sorties selon les cas.

En utilisant la toolbox, on se rend compte que le choix des fonctions d'activation à un impact fort sur les performances du modèle. Par ailleurs, le fait que la méthode utilise un pas fixe et un nombre d'itération fixé fait que le choix de ces paramètres est très important : un pas trop faible rend la convergence très lente, un pas trop grand fait diverger au lieu de converger.

Une bonne amélioration serait de fixer un critère de fin en plus d'un nombre d'itération maximum, et d'utiliser une méthode à pas variable pour accélérer la convergence et s'assurer de ne pas faire augmenter le coût, comme ça peut être le cas actuellement si le pas est choisi trop grand.

Notons également que pour augmenter légèrement la rapidité des calculs et surtout le confort d'utilisation, j'ai ajouté un paramètre à la fonction `onlinegrad` qui possède désormais une option `verbose` indiquant si on doit ou non afficher toutes les itérations.

2 Générer le jeu de test

On génère un jeu de données que l'on découpe en apprentissage (10%) et en validation (90%) (voir figure 1).

3 Création d'un réseau

On se propose de tester un réseau avec 2 fonctions d'activation `tanh`. Nous avons 2 entrées qui sont les 2 dimensions de chaque donnée x, et une sortie (la classe).

On teste donc ce réseau avec entre 1 et 10 neurones dans la couche cachée.

Notons qu'a partir des données initiales, il est important de construire le bon vecteur cible à passer à la toolbox. Dans le cas d'une sortie de `tanh`, la cible contient des 0 et des 1.

On notera que le choix d'un critère type MSE se justifie par le fait qu'il faut pouvoir dériver la fonction, mais la vraie mesure de qualité du réseau est le taux de bonne classification, que l'on obtient en arrondissant la sortie afin d'obtenir des 0 et des 1, et de comparer à la cible.

En effet, il faut distinguer le fait que la sortie soit réelle alors que l'on réalise en réalité de la classification et donc que l'ensemble cible est un ensemble de valeurs de cardinalité finie. Il n'est donc pas "grave" d'être un peu éloigné de la valeur cible tant que la valeur la plus proche est la bonne.

Les résultats sont visibles sur le graphe 2.

On constate qu'on obtient des résultats satisfaisants et environ identiques dès que l'on a au moins 3 neurones dans la couche cachée. Ceci est peut-être dû au fait que le problème n'est pas linéaire, ce qui augmente la complexité du problème et demande sans doute plus de neurones qu'un problème de séparation linéairement séparable. Un problème plus simple aurait sans doute nécessité moins de neurones pour avoir des résultats satisfaisants. Il est tout de même assez impressionnant de remarquer que 3 neurones dans la couche cachée suffisent à résoudre un problème de classification quadratique.

On constate tout de même que l'ajout de neurones ne semble pas améliorer la résolution du problème, comme on pourrait le penser naïvement.

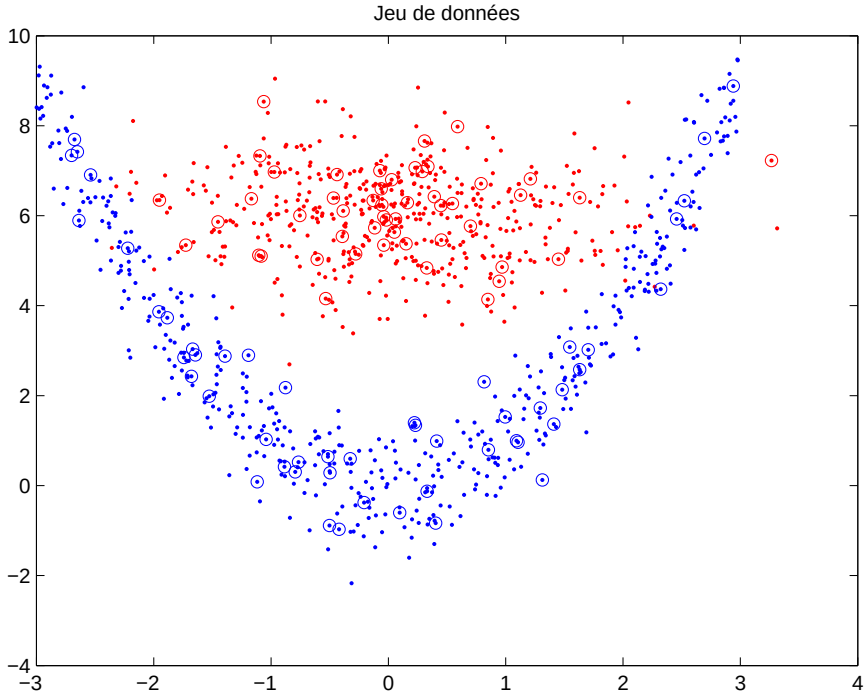

FIGURE 1 – Encerclés, les points du jeu d'apprentissage. Les autres sont le jeu de test.

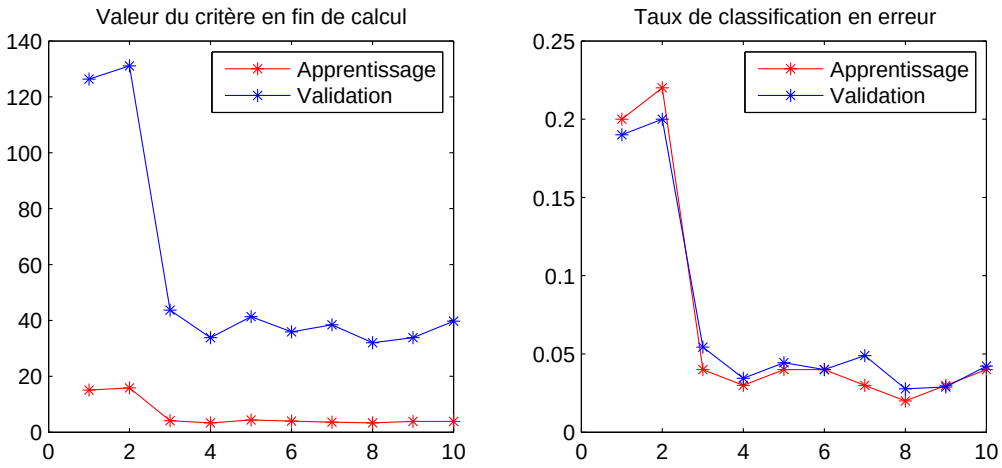

FIGURE 2 – Synthèse des résultats (en abscisse le nombre de neurones dans la couche cachée)

4 Code Matlab

4.1 Test des fonctions

```
% Exemple de problème de régression

clear all

net=ASINETfactory(1,[3 1],{'linear','linear'});

X = [1 2 3 4 5 6]';
Y = [2 4 6 8 10 12]';

[netout,learningErr,valError]=ASINETonlinegrad(net,X,Y,0.01,100,'mse', false);
YE=ASINETforward(netout,X);

%YE =
%    2.0510
%    4.0419
%    6.0328
%    8.0237
%   10.0146
%   12.0055
```

```
% Exemple de problème de classification

clear all

net=ASINETfactory(1,[5 2],{'tanh','softmax'});

X = [1 2 3 7 8 9]';
Y = [0 0 0 1 1 1; 1 1 1 0 0 0]';

[netout,learningErr,valError]=ASINETonlinegrad(net,X,Y,0.001,1000,'nll', false);
YE=ASINETforward(netout,X)

%YE =
%    0.0095    0.9905
%    0.0185    0.9815
%    0.0518    0.9482
%    0.9666    0.0334
%    0.9761    0.0239
%    0.9791    0.0209
```

4.2 Génération du jeu de test

```
clear all

nX=1000;
X = zeros(nX,2);

X(1:nX/2  ,:) = randn(nX/2,2) + repmat([0 6], nX/2, 1);
X(nX/2+1:nX,1) = (rand(nX/2,1) - 0.5) * 6;
X(nX/2+1:nX,2) = X(nX/2+1:nX,1).^2 + 0.7*randn(nX/2,1);

Y = [ones(nX/2,1) ; zeros(nX/2,1)];

plot(X(Y==1,1), X(Y==1,2), '*r'); hold on;
plot(X(Y==0,1), X(Y==0,2), '*b');

% découpage en apprentissage et test
[Xapp, Yapp, Xtest, Ytest] = splitdata(X, Y, 0.1);
```

4.3 Création du réseau

```
% valeurs à tester
n_vals = 1:10;

% Stockage pour affichage
errApp = zeros(length(n_vals),1);
errVal = zeros(length(n_vals),1);
nbErrApp = zeros(length(n_vals),1);
nbErrVal = zeros(length(n_vals),1);

% Pour chaque nb de neurones dans la couche cachée
for i = 1:length(n_vals)

    n = n_vals(i);

    net=ASINETfactory(2,[n 1],{'tanh','tanh'})
    [netout,learningErr,valError]=ASINETonlinegrad(net,Xapp,Yapp,0.01,250,'mse', true, Xtest, Ytest);
    YE=ASINETforward(netout,Xapp);
    YE2=ASINETforward(netout,Xtest);

    errApp(i) = learningErr(end);
    errVal(i) = valError(end);
    nbErrApp(i) = sum(round(YE) ~= Yapp);
    nbErrVal(i) = sum(round(YE2) ~= Ytest);

end

figure;
subplot(1,2,1);
plot(n_vals, errApp, '*-r'); hold on
plot(n_vals, errVal, '*-b');
title('Valeur du critère en fin de calcul');
legend('Apprentissage', 'Validation');
subplot(1,2,2);
plot(n_vals, nbErrApp/length(Xapp), '*-r'); hold on
plot(n_vals, nbErrVal/length(Xtest), '*-b');
title('Taux de classification en erreur');
legend('Apprentissage', 'Validation');
```