TP R sur les réseaux neuronaux

Emmanuel Rachelson and Matthieu Vignes

10 janvier 2014, SupAero - ISAE

1 Un tutoriel simple: le calcul de la racine carrée

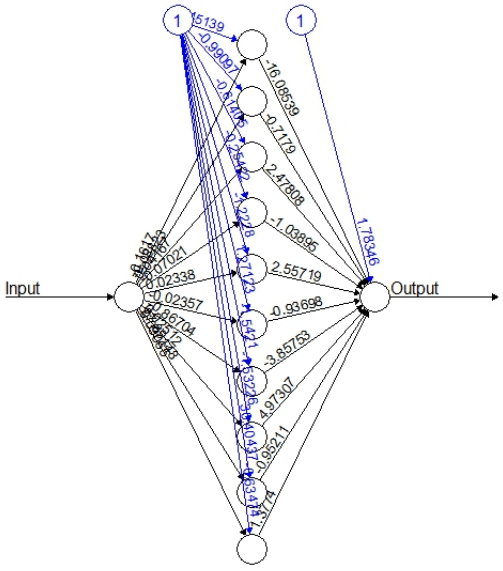
Dans ce tutoriel, un réseau neuronal va être construit pour prendre en entrée un nombre réel positif et calculer sa racine carrée. Ou plutôt une approximation aussi précise que possible. Cela est un exemple (simple) de ce que l'on peut faire ensuite pour faire de la prévision dans des situations beaucoup plus complexes.

Nous utiliserons la librairie `neuralnet`. Cela permettra de produire la Figure 1 où l'on voit les 10 neurones de la couche cachée. La sortie de notre script pourrait ressembler à (que pensez vous des erreurs ?):

Entrée	Sortie attendue	Sortie du réseau neuronal
1	1	0.9623402772
4	2	2.0083461217
9	3	2.9958221776
16	4	4.0009548085
25	5	4.0009548085
36	6	5.9975810435
49	7	6.9968278722
64	8	8.0070028670
81	9	9.0019220736
...	...	...

Et le code: (penser à `?neuralnet` pour des infos sur la librairie)

```
install.packages('neuralnet')
library(neuralnet)
# un va creer un reseau neuronal pour faire l'operation racine carree
# on genere 50 nombres aleatoires uniformement entre 0 et 100 et
# on stocke
traininginput <- as.data.frame(runif(50, min=0, max=100))
trainingoutput <- sqrt(traininginput)
trainingdata <- cbind(traininginput,trainingoutput)
colnames(trainingdata) <- c("Input","Output")
# on entraine le reseau avec 10 neurones caches
```




```
# le seuil est celui des derivees partielles pour la fonction
# d'erreur (critere d'arret)
net.sqrt <- neuralnet(Output~Input,trainingdata, hidden=10, threshold=0.01)
print(net.sqrt)
# a quoi ressemble notre reseau:
plot(net.sqrt)
# test du reseau du des donnees d'apprentissage
testdata <- as.data.frame((1:10)^2) # on genere des carres
net.results <- compute(net.sqrt, testdata) # et on fait mouliner
# c'est quoi les propretes de 'net.sqrt' ?
ls(net.results)
# Les resultats ?
print(net.results$net.result)
# Une presentation plus jolie des resultats
cleanoutput <- cbind(testdata,sqrt(testdata), as.data.frame(net.results$net.result)
colnames(cleanoutput) <- c("Input","Expected Output","Neural Net
Output")
print(cleanoutput)
```

2 Une représentation améliorée sous R

Bon vous avez compris que les réseaux neuronaux sont un outil puissant pour apprendre des relations entre les variables. Pas vraiment besoin des hypothèses classiques des outils statistiques standards. Ils ne marchent pas toujours mieux mais on peut leur trouve rune utilité là où on les critique souvent...Leur côté "boîte noire !

Dans le package `nnet`, on trouve tt ce qu'il faut pour entraîner un modèle de perceptron multicouche avec des fonctions flexibles pour optimiser le choix des paramètres. Mais côté visualisation, bof.

Özesmi and Özesmi (1999) décrivent des diagrammes d'interprétation neuronal pour visualiser les paramètres d'un réseau appris. L'importance (le poids) de chaque variable explicatrice *via* leur influence -supposée causale- sur les variables réponses peut être représentée. Le diagramme illustrera les connexions entre les couches et ses éléments grâce à des couleurs et des épaisseurs d'arcs.

Un blogueur proposait une fonction pour une telle visualisation qui a est déposée dans le fichier `plotNet.R`. Chargez le (`source()`) et chargez aussi les paquets `nnet, clusterGeneration`. Vous me direz si vous préférez le `plot` du paquet `neuralnet` que nous avons utilisé dans la section précédente.

On commence en créant un jeu de données artificiel pour construire le réseau neuronal dessus: 8 variables aléatoires avec une structure arbitraire de corrélation sont utilisées pour créer une réponse (combinaison linéaire des 8 variables auquel on ajouter un bruit).

```
set.seed(2)
num.vars<-8
num.obs<-1000
# matrice de correlation arbitraire et variable aleatoires generees
cov.mat<-genPositiveDefMat(num.vars,covMethod=c("unifcorrmat"))$Sigma
rand.vars<-data.frame(mvrnorm(num.obs,rep(0,num.vars),Sigma=cov.mat))
parms<-runif(num.vars,-10,10)
# la reponse est une c.l.  des variables ci-avant a laquelle
# on rajoute un terme d'erreur
y<-as.matrix(rand.vars) %*% matrix(parms) + rnorm(num.obs,sd=20)
```

Maintenant, on peut entraîner le réseau neuronal sur notre jeu de données. La fonction `nnet` prend en entrée une formule ou 2 arguments séparés. La réponse est convertie pour avoir ses valeurs entre 0 et 1 (argument `linout`, voir la doc).

```
require(nnet)
y<-data.frame((y-min(y))/(max(y)-min(y)))
names(y)<-'y'
nnet.mod1<-nnet(rand.vars,y,size=10,linout=T)
```

On a un réseau à 10 noeuds dans la couche cachée et une fonction de transfert linéaire pour la variable réponse. Tous les autres arguments sont

laissés à leurs valeurs par défaut. Ca n'est pas forcément malin et c'est d'ailleurs là que réside le secret dans la création d'un réseau "optimal" par le bon mélange des paramètres du modèle. Regardons ce que donne le `plot`:

```
par(mar=numeric(4),mfrow=c(1,2),family='serif')
plot.net(nnet.mod1,nid=F)
plot.net(nnet.mod1)
```

On crée 2 images. L'illustration standard des réseau neuronaux et celle du diagramme d'interprétation neuronal. Les arcs en noir sont les poids positifs, ceux en gris les négatifs. L'épaisseur est proportionnelle à l'amplitude des poids par rapport à la somme de tous les autres. Les 8 variables aléatoires explicatrices sont représentées dans la première couche (X1-X8) et la réponse apparaît à droite (`'y'`). Les neurones de la couche cachée sont les `H1, ..., H10`, (10 comme l'argument `size` de la fonction `nnet`). `B1` et `B2` sont les biais.

En jouant avec les arguments, on peut faire de jolies choses:

```
ir<-rbind(iris3[,,1],iris3[,,2],iris3[,,3])
targets<-class.ind( c(rep("s", 50), rep("c", 50),
+ rep("v", 50)) )
samp<-c(sample(1:50,25), sample(51:100,25), sample(101:150,25))
ir1<-nnet(ir[samp,], targets[samp,], size = 2, rang = 0.1,
+ decay = 5e-4, maxit = 200)
# un plot du modele avec des parametres differents
par(mar=numeric(4),family='serif')
plot.net(ir1,pos.col='darkgreen',neg.col='darkblue',alpha.val=0.7,
+ rel.rsc=15,circle.cex=10,cex=1.4,circle.col='brown')
```

On voit qu'on peut représenter des sorties multiples. On peut avoir les poids à partir de l'objet `nnet`. Essayez !

Et si on utilise le `plot` (argument `wts.only`) :

```
plot.net(ir1,wts.only=T)
```

Vous comprenez les sorties ?

On peut aussi représenter (arguments `all.in` et `all.out`) des connexions pour des variables spécifiques. Par exemple si on ne veut représenter que les poids pour la largeur de sépale (`'Sepal W.'`) et la variété *virginica sp.* (`'v'`):

```
plot.net(ir1,pos.col='darkgreen',neg.col='darkblue',
+ alpha.val=0.7,rel.rsc=15, circle.cex=10,cex=1.4,circle.col='brown',
+ all.in='Sepal W.',all.out='v')
```

3 Application des réseaux neuronaux à des données sur la concentration en ozone

On utilisera ici aussi le paquet R `nnet`. On aura aussi besoin des paquets `MASS` et `e1071`.

On rappelle que les données (identiques à celles du TP sur les SVM) ont été extraites et mises en forme par MétéoFrance; elles contiennent les variables suivantes :

- `JOUR`: le type de jour ; férié (1) ou pas (0),
- `O3obs`: la concentration d'ozone effectivement observée le lendemain à 17h locales correspondant souvent au maximum de pollution observée,
- `MOCAGE`: prévision de cette pollution obtenue par un modèle déterministe de mécanique des fluides (équation de Navier-Stockes),
- `TEMPE`: température prévue par MétéoFrance pour le lendemain 17h,
- `RMH2O`: rapport d'humidité,
- `NO2`: concentration en dioxyde d'azote,
- `NO`: concentration en monoxyde d'azote,
- `STATION`: lieu de l'observation : Aix-en-Provence, Rambouillet, Munchhausen, Cadarache ou Plan de Cuques,
- `VentMOD`: force du vent et
- `VentANG`: orientation du vent

1. Charger les données `ozone.dat` (dispos là `http://carlit.toulouse.inra.fr/wikiz/images/0/02/Ozone.dat.rar`). Passer la variable `JOUR` en facteur. Faire une transformation carrée pour la variable `RMH2O` et une transformation log pour les variables `NO2` et `NO` (pourquoi ?). Retirer les variables initiales.
2. Séparer les échantillons en un échantillon d'apprentissage `datappr` (sur lequel le modèle sera entraîné) et un échantillon de test `datestr` (sur lequel on mesurera les écarts entre prévisions et valeurs réllement observées). Les tailles respectives de ces deux jeux seront de 80 et 20%.

3.1 Régression

Que pensez vous de:

```
nnet.reg=nnet(O3obs∼.,data=datappr,size=5,decay=1, linout=TRUE,
+ maxit=500)
summary(nnet.reg)
```

Optimiser les paramètres nécessite la validation croisée. La fonction `tune.nnet()` de la librairie `e1071` est adaptée:

```
plot(tune.nnet(O3obs∼.,data=datappr,size=c(2,3,4), decay=c(1,2,3),
+ maxit=200,linout=TRUE))
plot(tune.nnet(O3obs∼.,data=datappr,size=4:5,decay=1:10))
```

Noter taille et `decay` optimaux. Faire varier le nombre d'itérations serait fastidieux. Noter aussi que chaque exécution donne des résultats différents...C'est rapide ?

Réestimer le modèle supposé optimal avant de tracer le graphe des résidus.

3.2 Discrimination

L'apprentissage donne :

```
nnet.dis=nnet(DepSeuil∼.,data=datappq,size=5,decay=1)
summary(nnet.reg)
```

On peut calculer l'erreur apparente ou par re-substitution mesurer la qualité de l'ajustement:

```
table(nnet.dis$fitted.values>0.5,datappq$DepSeuil)
```

La validation croisée est toujours nécessaire pour otpimiser les choix en présence: nombre de neurones, `decay` et éventuellement le nombre maximum d'itérations. Mais la fonction `tune.net()` pose problème dans le cas de la discrimination. En utilisant la fonction du fichier `CVnnBIS.R` fourni, ça ira mieux !

`niter` permet de répliuqer l'apprentissage et de moyenner les résultats pour réduire la variance de l'estimation de l'erreur. Vue que R n'est pas un foudre de guerre pour toutes ces boucles, on laissera la valuer par défaut (1).

Tester pour `size` parmi $(5, 6, 7)$ et `decay` dans $(0, 1, 2)$. Il semble plus simple de fixer le nombre de neurones à 7 et de ne varier que `decay` de 0 à 5 avec plusiuers éxécutions (initialisation de l'apprentissage du réseau neuronal et tirage pour la validation croisée sont aléatoires !). On pourrait construire un plan d'expériences...

```
CVnn(DepSeuil .,data=datappq,size=7, decay=0)
```

Noter taille et `decay` optimaux et réestimer le modèle pour ces valeurs.

3.3 Prévision sur l'échatillon test

Différentes prévisions sont considérées assorties des erreurs estimées sur l'échantillon test. Prévision quantitative de la concentration, prévision de dépassement à partir de la prévision quantitative ou directement.

On (i) calculera les prévisions, (ii) calculera l'erreur quadratique moyenne de prévision, formera la matrice de confusion pour la prévision de dépassement du seuil et on fera la même chose pour la discrimination.

Noter les taux d'erreur. Si on compare aux SVM ?

3.4 Comparaison des courbes ROC

Un peu comme dans le TP SVM...Une méthode de prévision d'occurence de dépasseemnt de pic de pollution est-elle globalement meilleure ?