Exercises - neural networks

Jan-Philipp Kolb

27 Mai, 2019

Exercise neural networks (I)

1. We'll start by creating the data set on which we want to do a simple regression. Set the seed to 42, generate 200 random points between -10 and 10 and store them in a vector named X. Then, create a vector named Y containing the value of sin(x). Neural network are a lot more flexible than most regression algorithms and can fit complex function with ease. The biggest challenge is to find the appropriate network function appropriate to the situation.
2. A network function is made of three components: the network of neurons, the weight of each connection between neuron and the activation function of each neuron. For this example, we'll use a feed-forward neural network and the logistic activation which are the defaults for the package `nnet`. We take one number as input of our neural network and we want one number as the output so the size of the input and output layer are both of one. For the hidden layer, we'll start with three neurons. It's good practice to randomize the initial weights, so create a vector of 10 random values, picked in the interval [-1,1].

Exercise neural networks (II)

3. Neural networks have a strong tendency of overfitting data, they become really good at describing the relationship between the values in the data set, but are not effective with data that wasn't used to train your model. As a consequence, we need to cross-validate our model. Set the seed to 42, then create a training set containing 75% of the values in your initial data set and a test set containing the rest of your data.
4. Load the `nnet` package and use the function `nnet` to create a model. Pass your weights via the `Wts` argument and set `maxit=50`. We fit a function which can have multiple possible output values. We set `linout=T`. Take the time to look at the structure of your model.
5. Predict the output for the test set and compute the RMSE of your predictions. Plot the function sin(x) and then plot your predictions.
6. The number of neurons in the hidden layer, as well as the number of hidden layer used, has a great influence on the effectiveness of your model. Repeat the exercises three to five, but this time use a hidden layer with seven neurons and initiate randomly 22 weights.

Exercise neural networks (III)

7. Now let us use neural networks to solve a classification problem, so let's load the iris data set! It is good practice to normalize your input data to uniformize the behavior of your model over different range of value and have a faster training. Normalize each factor so that they have a mean of zero and a standard deviation of 1, then create your train and test set.
8. Use the `nnet()` and use a hidden layer of ten neurons to create your model. We want to fit a function which have a finite amount of value as output. To do so, set the linout argument to true. Look at the structure of your model. With classification problem, the output is usually a factor that is coded as multiple dummy variables, instead of a single numeric value. As a consequence, the output layer have as one less neuron than the number of levels of the output factor.
9. Make prediction with the values of the test set.
10. Create the confusion table of your prediction and compute the accuracy of the model.

Solution Exercise 1

```
set.seed(42)
x<-runif(200, -10, 10)
y<-sin(x)
```

Solution Exercise 2

```
weight<-runif(10, -1, 1)
```

Exercise 3

```
set.seed(42)
index<-sample(1:length(x),round(0.75*length(x)),replace=FALSE)
reg.train<-data.frame(X=x[index],Y=y[index])
reg.test<-data.frame(X=x[-index],Y=y[-index])
```

Exercise 4

```
library(nnet)
set.seed(42)
reg.model.1<-nnet(reg.train$X,reg.train$Y,size=3,maxit=50,Wts=we
```

```
## # weights:  10
## initial  value 103.169943
## iter  10 value 70.636986
## iter  20 value 69.759785
## iter  30 value 63.215384
## iter  40 value 45.634297
## iter  50 value 39.876476
## final  value 39.876476
## stopped after 50 iterations
```

```
str(reg.model.1)
```

```
## List of 15
##  $ n            : num [1:3] 1 3 1
```

Exercise 5

```r
predict.model.1<-predict(reg.model.1,data.frame(X=reg.test$X))
str(predict.model.1)
```

```
##  num [1:50, 1] -0.201 0.184 -0.873 -0.981 0.598 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : NULL
```

```r
rmse.reg<-sqrt(sum((reg.test$Y-predict.model.1)^2))
rmse.reg
```

```
## [1] 3.41651
```

```r
plot(sin, -10, 10)
points(reg.test$X,predict.model.1)
```

Exercise 6

```
set.seed(42)
reg.model.2<-nnet(reg.train$X,reg.train$Y,size=7,maxit=50,Wts=ru
```

```
## # weights:  22
## initial  value 353.642846
## iter  10 value 55.906010
## iter  20 value 42.700328
## iter  30 value 22.757713
## iter  40 value 16.910492
## iter  50 value 12.770497
## final  value 12.770497
## stopped after 50 iterations
```

```
str(reg.model.2)
```

```
## List of 15
##  $ n             : num [1:3] 1 7 1
##  $ nunits        : int 10
```

Exercise 7

```
data<-iris

scale.data<-data.frame(lapply(data[,1:4], function(x) scale(x)))
scale.data$Species<-data$Species

index<-sample(1:nrow(scale.data),round(0.75*nrow(scale.data)),re
clust.train<-scale.data[index,]
clust.test<-scale.data[-index,]
```

Exercise 8

```
set.seed(42)
clust.model<-nnet(Species~.,size=10,Wts=runif(83, -1, 1),data=cl
```

```
## # weights:  83
## initial  value 187.294915
## iter  10 value 10.386561
## iter  20 value 5.337510
## iter  30 value 2.311922
## iter  40 value 1.426508
## iter  50 value 1.387440
## iter  60 value 1.386324
## final  value 1.386294
## converged
```

Exercise 9

```
predict.model.clust<-predict(clust.model,clust.test[,1:4],type="
```

Exercise 10

```
(Table<-table(clust.test$Species ,predict.model.clust))
```

```
##             predict.model.clust
##              setosa versicolor virginica
##   setosa         16          0         0
##   versicolor      0          9         0
##   virginica       0          0        13
```

```
(accuracy<-sum(diag(Table))/sum(Table))
```

```
## [1] 1
```