ML Exercises - neural networks

Jan-Philipp Kolb

27 Mai, 2019

Exercise neural networks (I)

Create example data

1) Set the seed to 42, draw 200 numbers from a unified random distribution between -10 and 10 and store them in a vector named `x`.
2) Create a vector named `y` containing the value of `sin(x)`.

Randomize initial weights

3) Create a vector of 10 random values, picked in the interval [-1,1].

Solution neuralnet exercises (I)

Create example data

```
set.seed(42)
x<-runif(200, -10, 10) #1)
y<-sin(x) #2)
```

Randomize the initial weights

```
weight<-runif(10, -1, 1) #3)
```

Exercise neural networks (II)

Split the dataset

4) 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.

Create a first model

5) Load the `nnet` package and use the function `nnet` to create a model. Pass your weights via the `Wts` argument and set `maxit=50`. Set `linout=T` and take some time to look at the structure of your model.

Solution neuralnet exercises (II)

Split the dataset

```
set.seed(42) #4)
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])
```

Create a first model

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

```
## # weights:  10
## initial  value 110.954499
## iter  10 value 70.129237
## iter  20 value 49.540887
## iter  30 value 37.185115
```

Solution neuralnet exercises (III)

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

```
## List of 15
##  $ n            : num [1:3] 1 3 1
##  $ nunits       : int 6
##  $ nconn        : num [1:7] 0 0 0 2 4 6 10
##  $ conn         : num [1:10] 0 1 0 1 0 1 0 2 3 4
##  $ nsunits      : num 5
##  $ decay        : num 0
##  $ entropy      : logi FALSE
##  $ softmax      : logi FALSE
##  $ censored     : logi FALSE
##  $ value        : num 29.9
##  $ wts          : num [1:10] -5.43 1.36 6.59 -1.32 1.64 ...
##  $ convergence  : int 1
##  $ fitted.values: num [1:150, 1] 0.6488 0.3885 -0.4967 -0.239
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
```

Exercise neural networks (III)

Prediction

6) Predict the output for the test set and compute the RMSE of your predictions. Plot the function sin(x) and then plot your predictions.

Repeat with different parameters

7) The number of neurons in the hidden layer, and the number of hidden layer used, has a great influence on the effectiveness of your model. Repeat the exercises four to six, but this time use a hidden layer with seven neurons and initiate randomly 22 weights.

Solution neuralnet exercises (IV)

Prediction

```
# 6)
predict.model.1<-predict(reg.model.1,data.frame(X=reg.test$X))
str(predict.model.1)
```

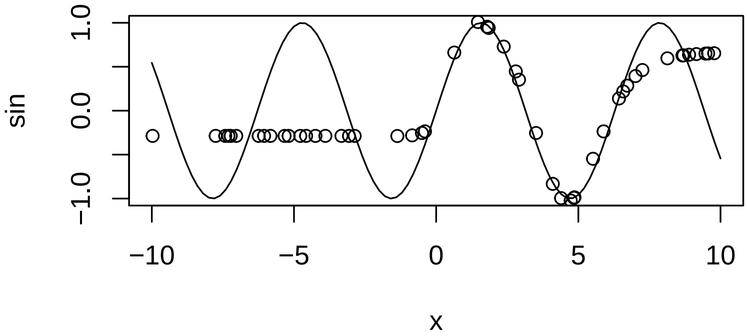
```
##  num [1:50, 1] -1.022 -0.287 -0.833 -0.281 0.651 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : NULL
```

```
# 6)
rmse.reg<-sqrt(sum((reg.test$Y-predict.model.1)^2))
rmse.reg
```

```
## [1] 3.894301
```

Solution neuralnet exercises (V)

```
# 6)
plot(sin, -10, 10)
points(reg.test$X,predict.model.1)
```



Solution neuralnet exercises (V)

Repeat with different parameters

```
# 7)
set.seed(42)
reg.model.2<-nnet(reg.train$X,reg.train$Y,size=7,
                  maxit=50,Wts=runif(22, -1, 1),linout=TRUE)
```

```
## # weights:  22
## initial  value 372.098787
## iter  10 value 58.093395
## iter  20 value 38.898448
## iter  30 value 22.575492
## iter  40 value 20.900687
## iter  50 value 18.497097
## final  value 18.497097
## stopped after 50 iterations
```

Solution neuralnet exercises (VI)

```
# 7)
str(reg.model.2)
```

```
## List of 15
##  $ n            : num [1:3] 1 7 1
##  $ nunits       : int 10
##  $ nconn        : num [1:11] 0 0 0 2 4 6 8 10 12 14 ...
##  $ conn         : num [1:22] 0 1 0 1 0 1 0 1 0 1 ...
##  $ nsunits      : num 9
##  $ decay        : num 0
##  $ entropy      : logi FALSE
##  $ softmax      : logi FALSE
##  $ censored     : logi FALSE
##  $ value        : num 18.5
##  $ wts          : num [1:22] 1.266 0.853 10.438 3.841 -0.382
##  $ convergence  : int 1
##  $ fitted.values: num [1:150, 1] 0.561 0.548 -0.692 -0.518 0.
##   ..- attr(*, "dimnames")=List of 2
```

Solution neuralnet exercises (VII)

```
# 7)
predict.model.2<-predict(reg.model.2,data.frame(X=reg.test$X))
str(predict.model.2)
```

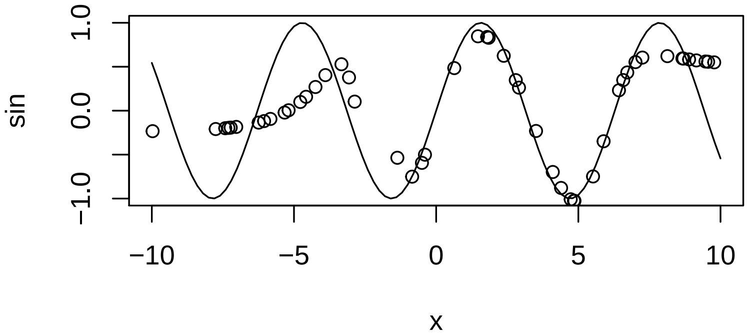
```
##  num [1:50, 1] -1.008 -0.196 -0.698 -0.751 0.556 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : NULL
```

```
# 7)
rmse.reg<-sqrt(sum((reg.test$Y-predict.model.2)^2))
rmse.reg
```

```
## [1] 3.064144
```

Solution neuralnet exercises (VIII)

```
# 7)
plot(sin, -10, 10)
points(reg.test$X,predict.model.2)
```



Exercise neural networks (IV)

Normilze

8) 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.

Create model with `nnet`

9) 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.

Solution Exercise 8

```r
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)),
              replace=FALSE)
clust.train<-scale.data[index,]
clust.test<-scale.data[-index,]
```

Solution Exercise 9

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

```
## # weights:  83
## initial  value 191.254326
## iter  10 value 5.268334
## iter  20 value 2.433467
## iter  30 value 0.009669
## final  value 0.000071
## converged
```

Exercise neural networks (V)

prediction and confusion table

10) Make prediction with the values of the test set.
11) Create the confusion table of your prediction and compute the accuracy of the model.

Solution Exercise 10 and 11

Prediction

```
# 10)
predict.model.clust<-predict(clust.model,clust.test[,1:4],
                              type="class")
```

Confusion matrix

```
# 11)
(Table<-table(clust.test$Species ,predict.model.clust))
```

```
##             predict.model.clust
##              setosa versicolor virginica
##   setosa         15          0         0
##   versicolor      0         10         1
##   virginica       0          2        10
```

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

```
## [1] 0.9210526
```