Artificial Neural Networks 3

Roger Barlow
The University of Huddersfield

CODATA School on Data Science

Autumn 2021

Second lecture: You ran this program

```
ALPHA=1.0 # learning parameter

nodes <- c(5,7,10,1) # 5 inputs, 2 hidden layers with 7 and 10 nodes, 1 output
nlayers  <- length(nodes) -1 # will give 3

w <- list() #  set up empty list of weight matrices

# make weights and fill with random numbers
for(j in 1:nlayers) w[[j]] <- matrix( runif(nodes[j]*nodes[j+1],-1,1),nodes[j+1],nodes[j])

netsays <- function(x) { # returns net output for some vector x
  for ( j in 1:nlayers)  x <- 1/(1+exp(-w[[j]] %*% x ))
  return(x)
  }

backprop <- function(layer,n1,n2,factor){ # recursive function for back propagation
#   from node n2 in layer to node n1 in layer+1
  if(layer>1) for( n in 1:nodes[layer-1]) backprop(layer-1,n2,n,factor*w[[layer]][n1,n2]*r[[layer]][n2]*(1-r[[layer]][n2]))
  w[[layer]][n1,n2] <<- w[[layer]][n1,n2] - ALPHA * factor * r[[layer]][n2]
  }

netlearns <- function(x,truth) { # like netsays but changes weights
  r <<- list()  # list of vectors containing results of all nodes in all layers
  r[[1]] <<- x # the input layer
  for(layer in 1:nlayers) r[[layer+1]] <<- 1/(1+exp(-w[[layer]] %*% r[[layer]]))
  u <- r[[nlayers+1]] # final answer for convenience
  for (n in 1:nodes[nlayers]) backprop(nlayers,1,n,(u-truth)*u*(1-u))
}
```

More realistic training samples

Camels and Dromedaries

The camel has a single hump,
The dromedary, two.
Or else the other way around.
I'm never sure. Are you?
– Ogden Nash

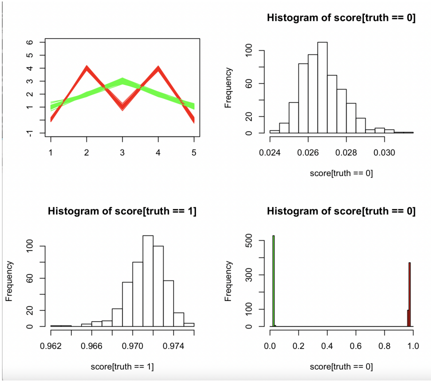
Data samples comprising 5 numbers
Ideal 'camel' is 0-4-1-4-0
Ideal 'dromedary' is 1-2-3-2-1
We have some non-ideal labelled samples
Use them to train the network to tell the difference

Look at data on Sample1.txt

Get it from the school moodle page. Or

`http://barlow.web.cern.ch/barlow/NN/Sample1.txt`

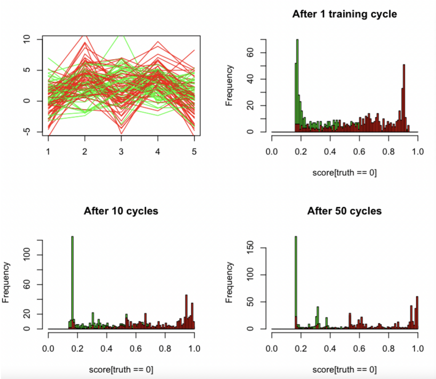
```
> #        Read sample. Change filename if necessary
> df <- data.matrix(read.table("Sample1.txt"))

> print(head(df,3)) #        have  QUICK LOOK
     V1          V2       V3        V4       V5          V6
[1,]  0 -0.05997873 3.881889 1.0607440 4.022852 -0.05597012
[2,]  1  0.88197800 2.055923 3.1585140 1.972982  1.19097300
[3,]  0  0.07778947 3.950015 0.9496442 3.976893  0.04745127

> #       Separate the true values, 0 or 1 for each
> truth <- df[,1]

> #       Separate the input data
> sample <- df[,2:6]

> #        draw an empty plot
> plot(0,0,type='n',ann=FALSE,xlim=c(1,5),ylim=c(-1,6))

> #        and draw in some of the events, coloured appropriately
> colours=c('red','green')

> for(i in 1:100) lines(1:5,sample[i,],col=colours[1+truth[i]])

> N <- dim(df)[1] #          How many events are there?

> score <- rep(0,N) #          empty vector for storing results

> #         find the scores.  (No training yet so meaningless)
> for( i in 1:N) score[i] <- netsays(sample[i,])

> print(paste(" Scores before training ",mean(score[truth==1]), mean(score[truth==0])))
[1] " Scores before training  0.576956931302425 0.525675192970615"

> #         train - using every event in the sample once
> for( i in 1:N) netlearns(sample[i,],truth[i])

> #         now find the scores after the training
> for( i in 1:N) score[i]=netsays(sample[i,])

> print(paste(" Scores after training ",mean(score[truth==1]),mean(score[truth==0])))
[1] " Scores after training  0.969494092436271 0.0231471679893081"

> #        plot histograms to give more information than the mean
> hist(score[truth==0])

> hist(score[truth==1])

> #        Now put both histograms on the same plot
> h1 <- hist(score[truth==0],breaks=seq(0,1,.01),plot=FALSE)

> h2 <- hist(score[truth==1],breaks=seq(0,1,.01),plot=FALSE)

> plot(h1,col='green')

> lines(h2,type='s',col='red')
```



Look at data on Sample2.txt

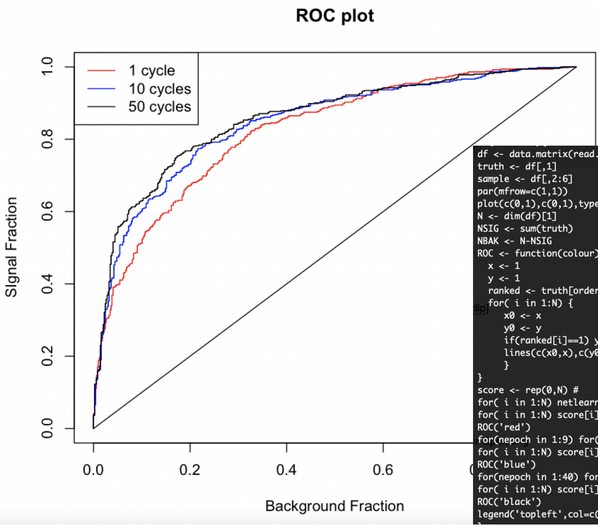
More of a challenge

```
> df <- data.matrix(read.table("Sample2.txt"))
> truth <- df[,1]
> sample <- df[,2:6]
> plot(0,0,type='n',ann=FALSE,xlim=c(1,5),ylim=c(-5,10))
> for(i in 1:100) lines(1:5,sample[i,],col=colours[1+truth[i]])
> N <- dim(df)[1] #          How many events are there?
> score <- rep(0,N) #          empty vector for storing results
> for( i in 1:N) netlearns(sample[i,],truth[i])
> for( i in 1:N) score[i]=netsays(as.numeric(sample[i,]))
> h1 <- hist(score[truth==0],breaks=seq(0,1,.01),plot=FALSE)
> h2 <- hist(score[truth==1],breaks=seq(0,1,.01),plot=FALSE)
> plot(h1,col='green',main="After 1 training cycle")
> lines(h2,type='s',col='red')
> for(nepoch in 1:9) for( i in 1:N) netlearns(sample[i,],truth[i])
> for( i in 1:N) score[i]=netsays(sample[i,])
> h1 <- hist(score[truth==0],breaks=seq(0,1,.01),plot=FALSE)
> h2 <- hist(score[truth==1],breaks=seq(0,1,.01),plot=FALSE)
> plot(h1,col='green',main="After 10 cycles")
> lines(h2,type='s',col='red')
> for(nepoch in 1:40) for( i in 1:N) netlearns(sample[i,],truth[i])
> for( i in 1:N) score[i]=netsays(sample[i,])
> h1 <- hist(score[truth==0],breaks=seq(0,1,.01),plot=FALSE)
> h2 <- hist(score[truth==1],breaks=seq(0,1,.01),plot=FALSE)
> plot(h1,col='green',main="After 50 cycles")
> lines(h2,type='s',col='red')
>
```



Network performance: ROC plots

Stands for "Receiver Operating Characteristics", for reasons lost in history




```
df <- data.matrix(read.table("Sample2.txt"))
truth <- df[,1]
sample <- df[,2:6]
par(mfrow=c(1,1))
plot(c(0,1),c(0,1),type='l',main='ROC plot',xlab="Background Fraction",ylab="Signal Fra
N <- dim(df)[1]
NSIG <- sum(truth)
NBAK <- N-NSIG
ROC <- function(colour){
  x <- 1
  y <- 1
  ranked <- truth[order(score)]
  for( i in 1:N) {
     x0 <- x
     y0 <- y
     if(ranked[i]==1) y <- y - 1/NSIG else x = x - 1/NBAK
     lines(c(x0,x),c(y0,y),col=colour)
     }
}
score <- rep(0,N) #          empty vector for storing results
for( i in 1:N) netlearns(sample[i,],truth[i])
for( i in 1:N) score[i] <- netsays(sample[i,])
ROC('red')
for(nepoch in 1:9) for( i in 1:N) netlearns(sample[i,],truth[i])
for( i in 1:N) score[i] <- netsays(sample[i,])
ROC('blue')
for(nepoch in 1:40) for( i in 1:N) netlearns(sample[i,],truth[i])
for( i in 1:N) score[i] <- netsays(sample[i,])
ROC('black')
legend('topleft',col=c('red','blue','black'),legend=c('1 cycle','10 cycles','50 cycles'
```

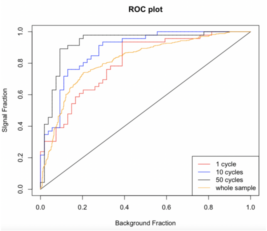
Using ROC plots

To decide where to put any signal/background cut you *also* need to know

- The fraction of signal events in the real data
- The relative cost of mistakenly excluding a signal or including a background event

If you have this data you can use it to find the best place to put a cut, and give a figure-of-merit or score for the network If not, then use something like: the background fraction corresponding to 90% signal fraction.
Note: signal fraction is not the same as purity. Be very wary of the expression "background efficiency"

Overtraining



Use part of the sample. `df <- df[1:100,]`
Performance *looks* much better....
But when you try these weights on the whole sample, it's worse.
This is Overtraining. The network gets to recognise individual events.

Remedy:

Separate data into training ($\sim 90\%$) and testing ($\sim 10\%$) samples. Train on training sample until score on test sample stops improving.

Extension (1). Train-validate-test. $\sim 80\% :\sim 10\% :\sim 10\%$ When you really need an unbiassed value for the score to compare models
Extension (2). Cross-validation. Repeat training-testing split several times (typically 10) and get properties. Avoid sacrificing $\sim 10\%$
Warning: 'Validation' and 'testing' are interchanged in the literature.

Conclusion

If your network can't achieve perfect separation, you need to draw a ROC plot

Always keep separate samples for training and testing

Activity for you

Have a look at `Sample3.txt`. It is smeared - more than sample 1 but less than sample 2 - and also has a 10% chance of any value giving zero (due to faulty apparatus?).

What can you make of it?

Present some result(s) as a plot and put on Nextcloud for the class zoom session