Artifical Neural networks

Roger Barlow
Huddersfield University

Aachen Online Statistics School

14^{th} March 2023

Kittens and puppies, cats and dogs

What the internet is used for, mostly

You can tell the difference

Quickly
Accurately
Reliably
Robustly

Profound

Impressive!

Not an algorithm

The Classification Problem

Examples from all over science. And beyond.

- Is this a quark jet or a gluon jet?
- Is this blob a star or a galaxy?
- Is this a stone implement or just a rock?
- Is this email genuine or spam?
- Is this patient sick or well?
- Is that one of our tanks or one of their tanks?

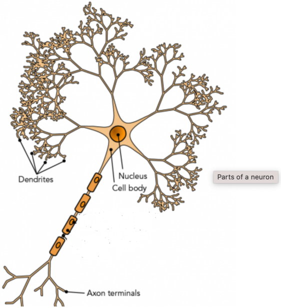
Let's generalise this to

- Is this signal or background?

Classic response to this problem is to devise a set of rules, flowcharts, if-then-else algorithms...
Wouldn't it be nice to just look and know? Like we do with cats and dogs...

Let's try and copy the human brain

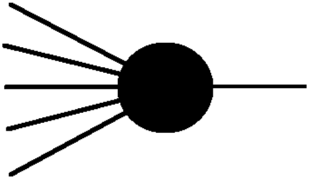
Massively over-simplified anatomy lesson:



The brain comprises 100,000,000,000 neurons
Each combines many inputs
(from eyes, ears, touch... and other neurons)
into a single value
which it outputs to many places
(to muscles ... and other neurons)

Let's emulate that in software - not very difficult

From neurons to nodes



A node (or neuron) combines many inputs into one output

Simplest choice: add them all up.

$R = \sum r_i$ Too simple to be useful

Better choice: do a weighted sum.

$R = \sum w_i r_i$ Sign & size of weight depend on importance and effect of input

Best choice: feed this weighted sum through a thresholding (or activation) function. $R = S(\sum w_i r_i)$

E.g. Logistic function

$S(x) = \frac{1}{1+e^{-x}}$

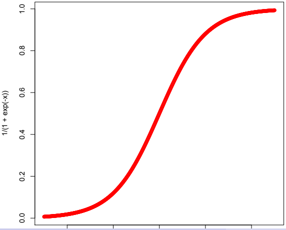
People also use $tanh(x)$ and

$RELU(x) = max(0, x)$ and

$Swish(x) = xS(x)$

Will need differential later:

$S'(x) = S(1 - S)$



From nodes to networks

Can combine nodes in various ways, including multilayer feed-forward network, or Perceptron

Nodes are in layers

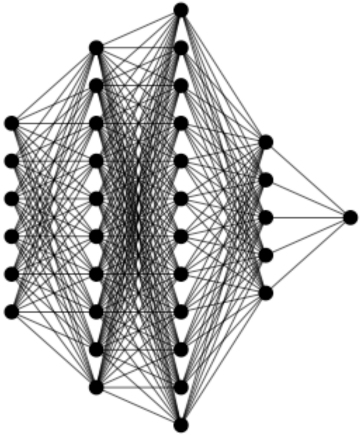
First layer corresponds to the data values

Final layer has single node (usually) that will give 1 for signal and 0 for background

In between are hidden layers where the work is done

Each node takes an input from all nodes in the previous layer, and sends an output to all nodes in the next layer. But thats all.

Topology makes timing straightforward. Present data. Calculate all values in 1st hidden layer. Then next layer. And so on to final yes/no layer



Note: each line corresponds to a weight w_{ij}^{ℓ} : from node j in layer ℓ to node i in layer $\ell+1$

How to determine the weights

The weights are found by training

Take a sample of events for which the identities are known. Call these true values T (they will be 0 or 1) and the final network output U, or U_1^{L+1}

Define Badness $B = \frac{1}{2}(U - T)^2$. of an event

Clearly we want to minimise B

So adjust each weight w_{ij}^ℓ by an amount proportional to $\frac{\partial B}{\partial w_{ij}^\ell}$

$$w_{ij}^\ell \to w_{ij}^\ell - \alpha \frac{\partial B}{\partial w_{ij}^\ell}$$

where α is the 'learning parameter'.

Do this for all events in the training sample - and then repeat (but not too many times)

Back propagation

Differentials evaluated through back propagation
In the final layer: $U = S(\sum w^L_{1j} R^L_j)$
where R^L_j is the output from node j in layer L

$$\frac{\partial B}{\partial w^L_{1j}} = (U - T)\frac{\partial U}{\partial w^L_{1j}} = (U - T)U(1 - U)R^L_j$$

The weights in the previous layer, w^{L-1}_{jk}, affect B through the R^L_j

$$\frac{\partial B}{\partial w^{L-1}_{jk}} = (U - T)U(1 - U)w^L_{1j}\frac{\partial R^L_j}{\partial w^{L-1}_{jk}}$$

$$\text{and } \frac{\partial R^L_j}{\partial w^{L-1}_{jk}} = R^L_j(1 - R^L_j)R^{L-1}_k$$

and so on backwards to the first layer, picking up factors of $R(1 - R)$ and w as we go

From algebra to code

This is a complete neural network program - small but perfectly formed

Aachen — more Aachen.R — 80×29

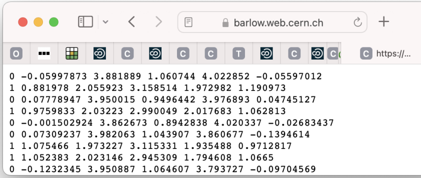
```
ALPHA=0.1   #  learning parameter

net <- c(5,7,10,1)   #  5 inputs, 2 hidden layers with 7 and 10 nodes

N <- length(net)-1  # number of sets of weights. 3 in this case

w <- list() # create list of random weights. Note double brackets
for (i in 1:N) w[[i]] <- matrix(runif(net[i]*net[i+1],-1,1), net[i+1], net[i])

netsays <- function(x){  # get answer from the network
   for(i in 1:N) x <- 1/(1+exp( -w[[i]] %*% x))
   return(x)
   }

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

netlearns <- function(x,truth){ # train network on an event
  r <- list()  # like netsays but save node values in list
  r[[1]] <- x
  for(L in 1:N) r[[L+1]] <- 1/(1+exp(-w[[L]] %*% r[[L]]))
  u <- r[[N+1]]
  for (i in 1:net[N]) backprop(N,1,i,(u-truth)*u*(1-u),r)
  }
Aachen.R
```

Type it in. Or maybe type it into a file and then `source` it. You can leave out the comments. And you can change the names, if you want. And the network topology. And ALPHA.

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 comprising 5 numbers
Ideal 'camel' is 0-4-1-4-0
Ideal 'dromedary' is 1-2-3-2-1
We have some smeared events
labelled 0 for camel, 1 for dromedary



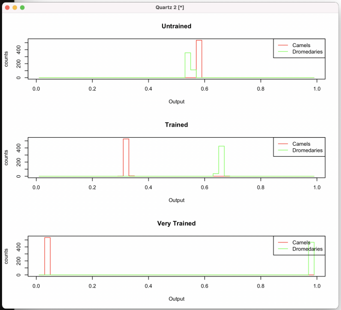

```
0 -0.05997873 3.881889 1.060744 4.022852 -0.05597012
1 0.881978 2.055923 3.158514 1.972982 1.190973
0 0.07778947 3.950015 0.9496442 3.976893 0.04745127
1 0.9759833 2.03223 2.990049 2.017683 1.062813
0 -0.001502924 3.862673 0.8942838 4.020337 -0.02683437
0 0.07309237 3.982063 1.043907 3.860677 -0.1394614
1 1.075466 1.973227 3.115331 1.935488 0.9712817
1 1.052383 2.023146 2.945309 1.794608 1.0665
0 -0.1232345 3.950887 1.064607 3.793727 -0.09704569
```

Download sample file from
https://barlow.web.cern.ch/barlow/NN/Sample1.txt

Try it out

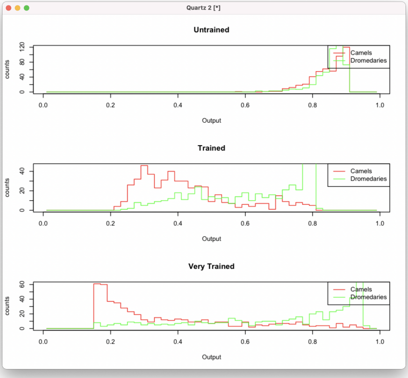
Aachen — vi Aachen.R — 84×30

```
# read sample
cdtable <- read.table("https://barlow.web.cern.ch/barlow/NN/Sample1.txt")
cdsample <- data.matrix(cdtable)
nevents <- dim(cdsample)[1] # number of events in the sample
truth <- cdsample[,1]    # separate the answers
events <- cdsample[,-1]  # and the data
results <- rep(0,nevents)

showresults <- function(title){
for(i in 1:nevents) results[i] <- netsays(events[i,])
hc=hist(results[truth==0],breaks=seq(0,1,.02),plot=FALSE)
hd=hist(results[truth==1],breaks=seq(0,1,.02),plot=FALSE)
plot(hc$mids,hc$counts,type='s',col='red',xlab='Output',ylab='counts',main=title)
lines(hd$mids,hd$counts,type='s',col='green')
legend('topright',col=c('red','green'),lwd=1,legend=c('Camels','Dromedaries'))
}

par(mfrow=c(3,1))
showresults("Untrained")

for(i in 1:nevents) netlearns(events[i,],truth[i])
showresults("Trained")

for(j in 1:10) for(i in 1:nevents) netlearns(events[i,],truth[i])
showresults("Very Trained")


"Aachen.R" 67L, 2141B written
```



A realistic example

Bit more of a challenge...



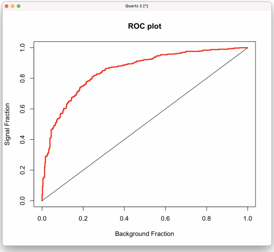
In `Sample1.txt` the C and D patterns were smeared by a little
In `Sample2.txt` they have been smeared by a lot

How do you quantify the performance?
Where should you put a cut?

Evaluating performance: ROC plots

"Receiver Operating Characteristic". Don't ask.

Select 'signal' with output≈ 1 and reject 'background' with output ≈ 0.
ROC Plot: background fraction accepted versus signal fraction accepted



Aachen — vi Aachen.R — 84×11

```
perf=truth[order(results)]
nc=sum(truth==0)
nd=sum(truth==1)
plot(1-cumsum(perf==0)/nc,1-cumsum(perf==1)/nd,
    type='l',col='red',lwd=3,
    xlab="Background Fraction",ylab="Signal Fraction",main="ROC plot")
lines(c(0,1),c(0,1))
```

Start with cut at $U = 0$. All events accepted.
Top right corner.
Increase cut. Lose signal and background, move down and left, but more left than down.
Eventually reach bottom left corner, cutting at $U = 1$.

Straight line shows effect of cut with no discrimination.
The further the curve gets from that, the better.

Where to put a cut?

You do not have enough information

You need to know the signal-to-background ratio in the real data. (In the training sample it has to be 50:50)

You also need to know the relative cost of a Type I error (excluding a signal event) and a Type II error (including a background event)

Be very careful of any probability. Does $p = 0.85$ mean that there is an 85% probability that this is signal, or that 85% of the signal is here?

Training and over-training

Optimise performance (through ROC plot) by changing network topology, selection of input variables, etc
BUT
Repeated training cycles lead to `over-training.` The network gets to recognise individual events.
Serious work must use separate training and testing samples. (Maybe 80:20 or 90:10 split). Train on large sample until performance (measured on small test sample) stops improving.
If you really need to know the performance, need further division into training - testing - validation
Cross-validation can save you the final split. It takes time and effort, and if you have a large sample you don't really need it.

Conclusions

Neural Networks have become part of the basic toolkit.

You should use them. Some of you probably already are.

You should treat them with familiarity, not reverence. Don't be afraid to write your own. Use a package - but don't just use one package, there are lots of excellent ones out there.

Lots of cool stuff (GANs, CNNs) follow on from the basic ANN ideas. Other ML tools (BDTs, SVMs) uses similar concepts.

Do stuff and have fun!