

Artificial Neural Networks

Roger Barlow



CODATA School - Roger Barlow - Artificial Neural Networks

The main use of the internet is to share cute pictures of cats and dogs



The human brain is very good at recognising which is which



Classification

We recognise and classify objects -
quickly
robustly
reliably

and we don't use conventional logic (i.e. flow charts)

This attacks a very general statistics/data problem:

Physicist: is this event signal or background
is the track a muon or a pion?

Astronomer: is this blob a star or a galaxy?

Doctor: is this patient sick or well?

Banker: is this company a sound investment or junk?

Employer: is this applicant employable or a liability?



Neural Networks

The brain is made of ~100,000,000,000 neurons.

Each neuron has MANY inputs. From external sources (eyes, ears...) or from other neurons.

Each neuron has one output connected to MANY externals (muscles or other neurons).

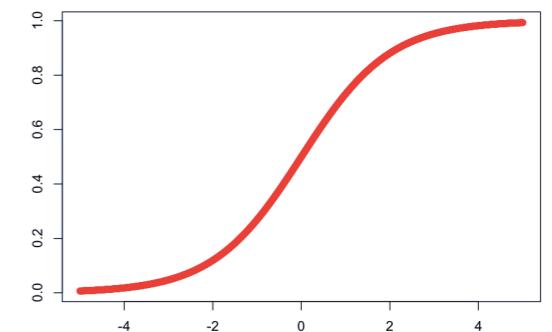
The neuron forms a function of the inputs and presents it to all the outputs.

Artificial Neural Networks

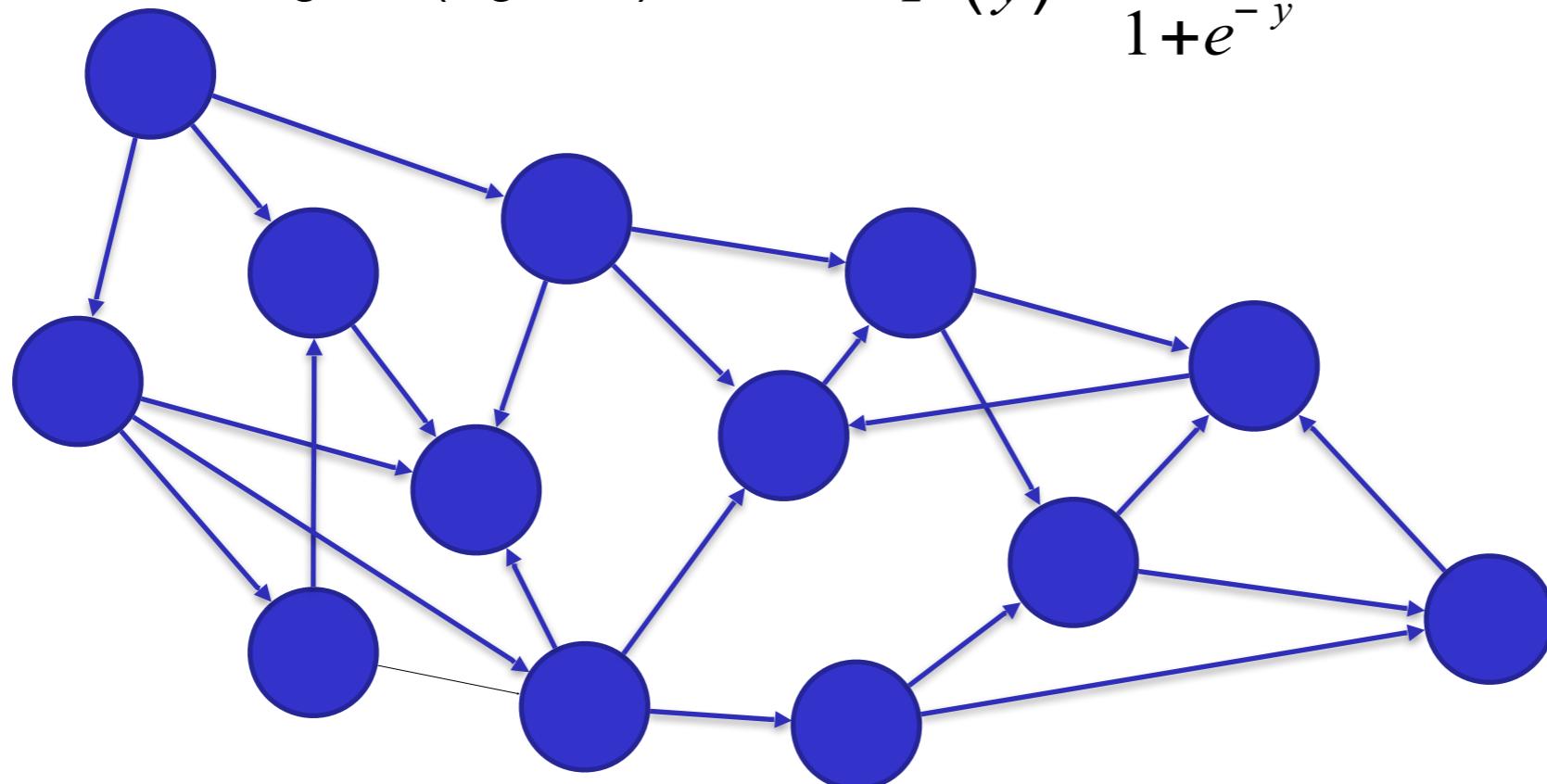
Duplicate the working of brain neurons in software

Neuron/node i has many inputs U_j . Apply weights, form $y_i = \sum w_{ij} U_j$ and generate output $U_i = F(y_i) = F(\sum w_{ij} U_j)$

F is thresholding function. Output increases monotonically from 0 to 1.
Linear central region but saturates at extremes.



Often use logistic (sigmoid) function $F(y) = \frac{1}{1+e^{-y}}$



Sometimes use
 $F(y) = \tanh(y)$

Can simulate
networks with
various topologies

The Multilayer Perceptron

A network architecture for binary classification:
recognise data ‘events’ (all of the same format)
as belonging to one of 2 classes.
e.g. Signal or Background? (S or B?)

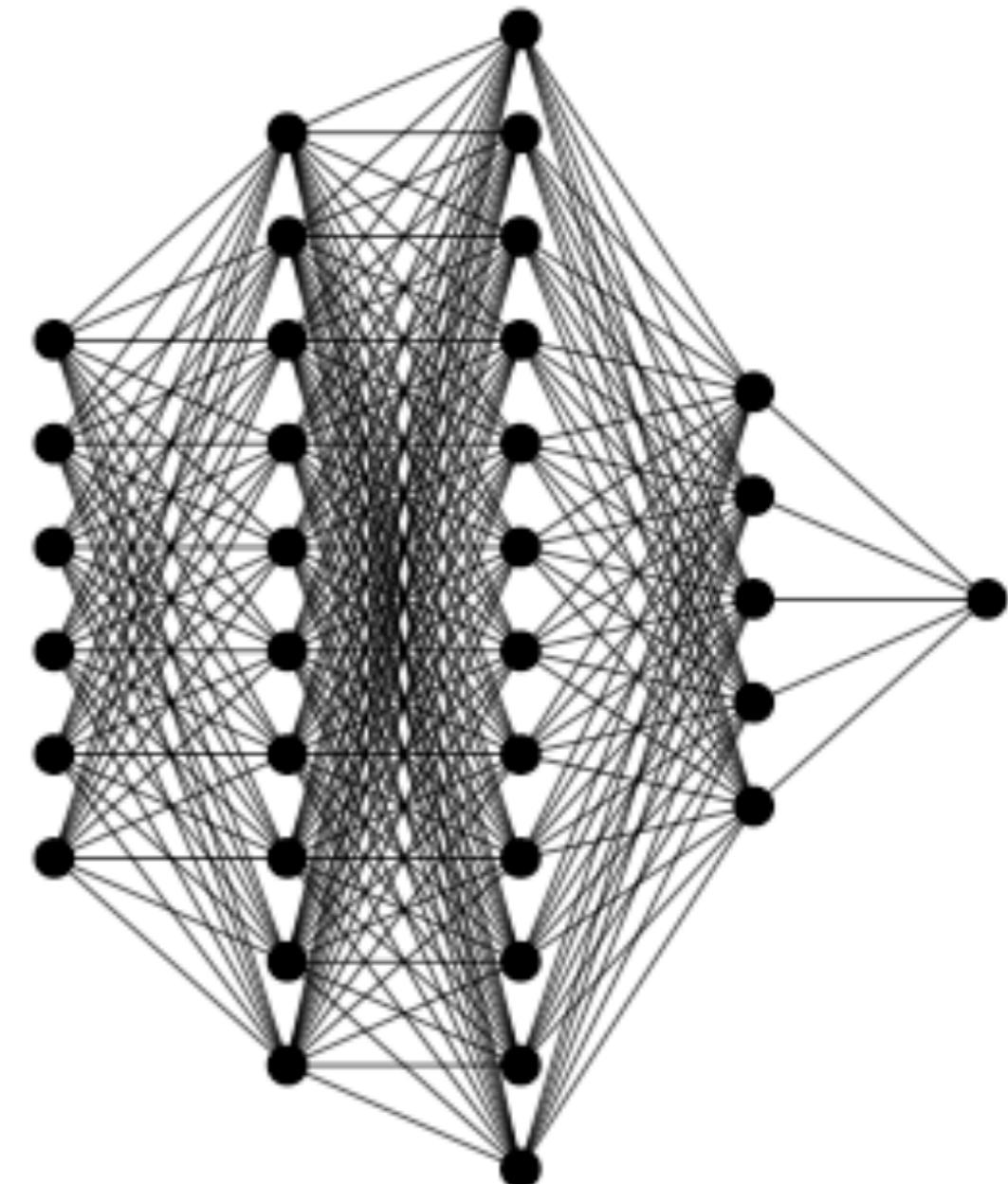
Nodes arranged in layers.

First layer – input

Last layer – single output, ideally 1
(for S) or 0 (for B)

In between - ‘hidden’ layers

Action is synchronised: all of first layer
effects the second (effectively)
simultaneously, then second layer
effects third, etc



How do we set the weights?

By Training: using samples of known events

Present events whose type is known: has a desired output T, which is 0 or 1. Call the actual output U.

Define ‘Badness’ $B = \frac{1}{2} (U-T)^2$. “Training the net” means adjusting the weights to reduce total (or average) B.

Strategy: change each weight w_{ij} by step proportional to $-dB/dw_{ij}$.

Do this event by event (or in batches, for efficiency).

All we need do is calculate those differentials... start with final layer and work backwards ('back-propagation')

For a single event, let U be the output given by the net and T be the desired outcome.

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

Strategy: adjust all network weights w by an amount $-\alpha \frac{\partial B}{\partial w}$

α is a small fixed number

Consider a weight in the final layer, w_{1l} (1st index 1 as only 1 output node).

$$U = U(\sum_l w_{1l} U_l) \quad U(y) = \frac{1}{1 + e^{-y}} \quad U'(y) = U(1 - U)$$

$$\frac{\partial B}{\partial w_{1l}} = (U - T) \frac{\partial U}{\partial w_{1l}} = (U - T)U(1 - U)U_l$$

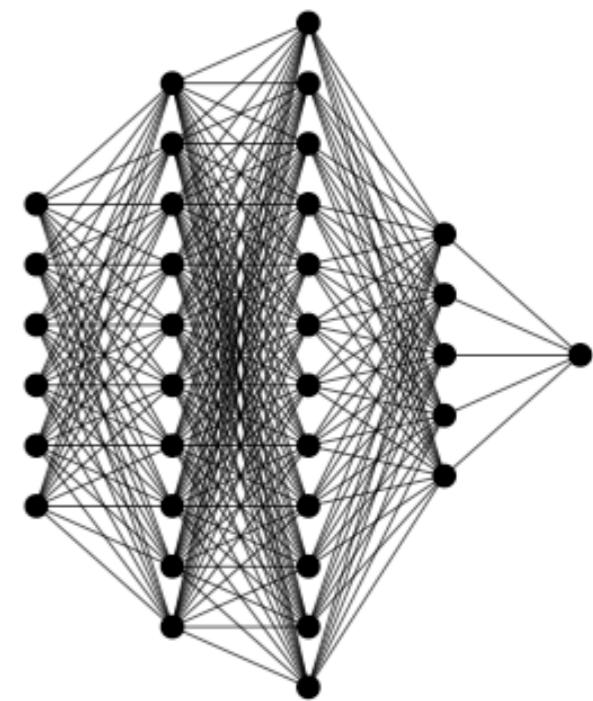
Now consider the penultimate layer, l'

$$\frac{\partial B}{\partial w_{ll'}} = (U - T) \frac{\partial U}{\partial w_{ll'}} = (U - T)U(1 - U)w_{1l} \frac{\partial U_l}{\partial w_{ll'}}$$

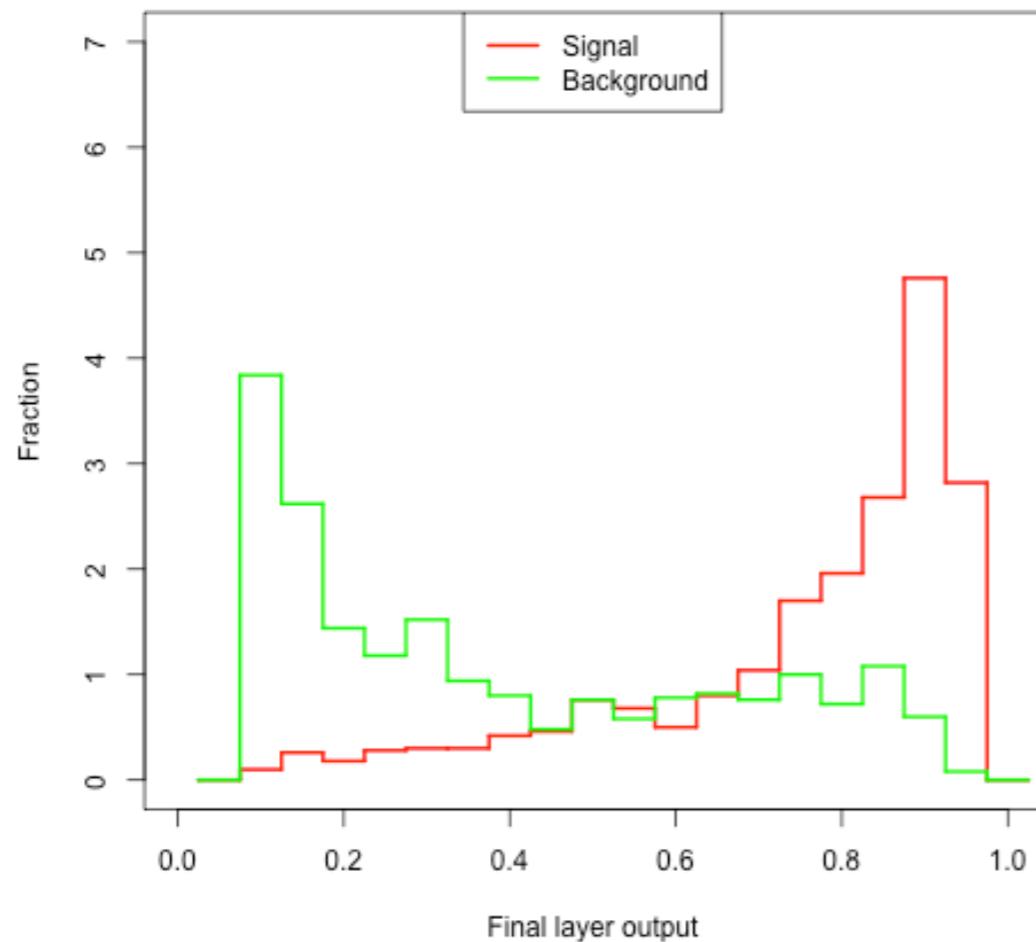
$$= (U - T)U(1 - U)w_{1l}U_l(1 - U_l)U_{l'}$$

Now l''

$$\frac{\partial B}{\partial w_{l'l''}} = (U - T)U(1 - U) \sum_l w_{1l}U_l(1 - U_l)w_{ll'}U_{l'}(1 - U_{l'})U_{l''}$$



Performance: Output histograms



After training - over the whole training sample many times - the outputs from the S and B samples will look something like this

Select signal by requiring $U > \text{cut}$

Small cut value: high efficiency but high background

Large cut value: low background but low efficiency

Exactly where to put the cut depends on
(i) The penalties for Type I and Type II errors
(ii) The prior probabilities of S and B

Note the actual shape of the histograms means nothing. Any transformation of the x-axis does not affect the results

Reminder:

Type I error: excluding a signal event

Type II error: including a background event

Performance: ROC* plots

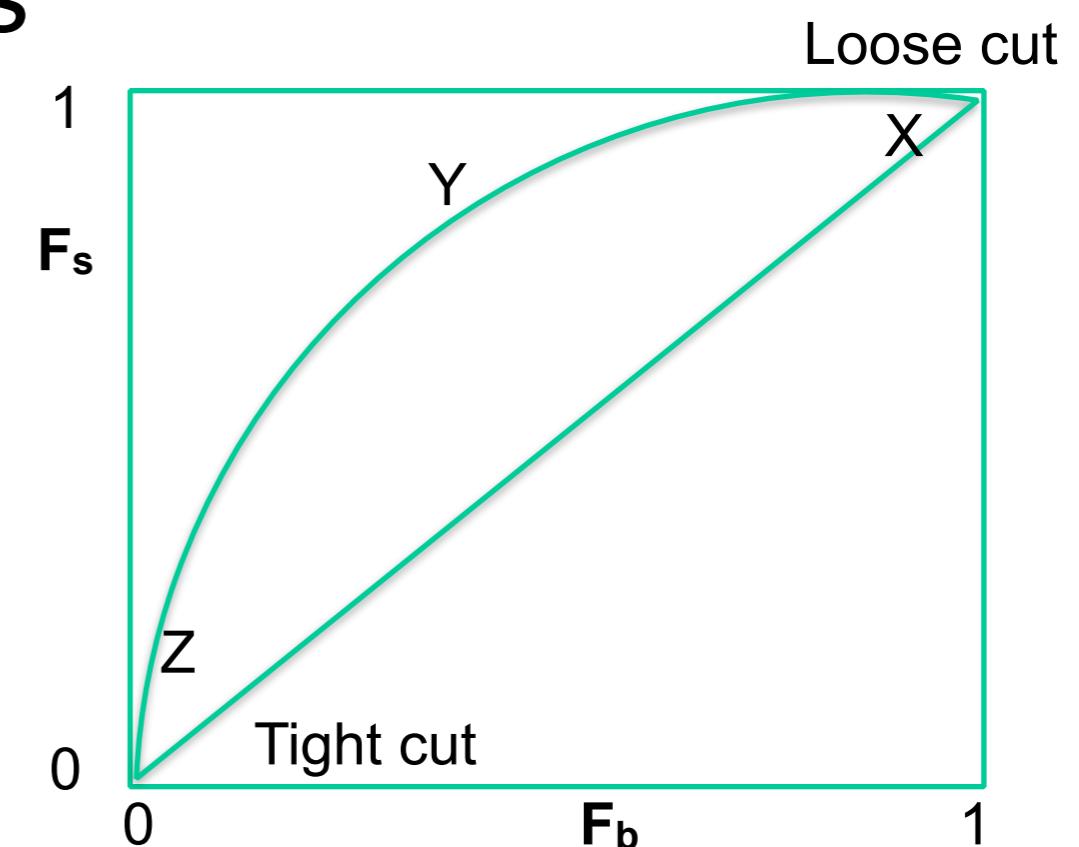
Plot fraction of background accepted against fraction of signal accepted, sliding the cut from 0 (nothing) to 1 (everything)

(Note that conventions vary on how to do this)

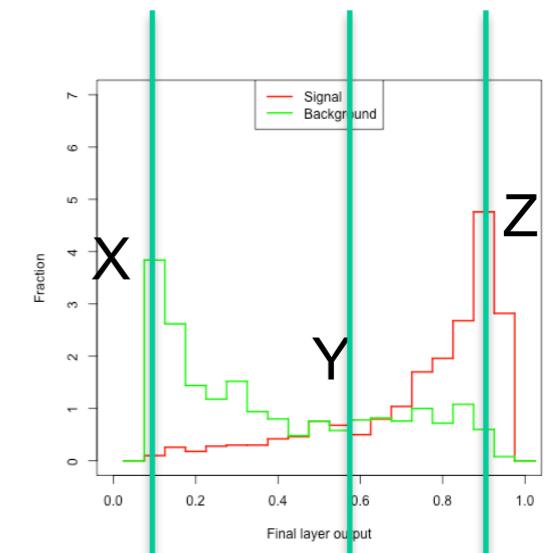
If net is working, background falls faster than efficiency

No discrimination gives 45 degree line

The bigger the bulge, the better



To draw ROC plot can use histograms, or go back to raw data, rank it according to the output (use R function order), and step through it



*Receiver Operating Characteristic

Training, over-training, testing, validating

Network is trained on the sample, and then re-trained, and then re-re-trained...getting better all the time, as measured by $\sum(T_i - U_i)^2$

An ‘over-trained’ network will select peculiarities of individual events in the sample. Improved performance on training sample but worse performance on other samples

Recommended procedure: have separate training sample (about 80% of data) and testing sample (remaining 20%). Train on training sample until performance on testing sample stops improving

Easy to do if you have lots of samples - which is generally the case for large Monte Carlo samples but not for real data

Validating. Given output X, what can you say about probability of S or B? (i.e. those histograms) Separate sample needed for validation.

Or cross-validation. For each event, train on the rest of the sample and compare truth and prediction, avoiding bias. (If too slow, use sub-samples ‘K-fold cross validation’)

Warning! Language ambiguities

- Signal Efficiency

Fraction of signal events remaining after the cut

- Background Efficiency

- (i) *Fraction of background events remaining after the cut, OR*
(ii) *fraction of background events removed by cut*

- Contamination (or Contamination probability)

- (i) *Fraction of background events remaining after cut*
OR (ii) *fraction of selected events which are background*

- Purity

Fraction of selected events which are signal

- True positive rate

Same as signal efficiency - not purity

- False positive rate

Same as background efficiency (i) - not Contamination

Neural Network Regression

Not considered here but trivial extension -

Desired output not simple true/false but numeric

Examples:

- House price from location, no. of rooms, etc
- Pupil progress from past performance+background

Train to minimise $\frac{1}{2} (T-U)^2$, test, predict as before, but T is a (scaled) number, not just 0 or 1.

NN classification is just a subset of NN regression

Problem



Tell a camel from a dromedary:

ro

Given 5 inputs, and events of 2 types:

either 1-2-3-2-1 (+ noise) or 0-4-1-4-0 (+noise)

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

Ogden Nash

3 samples to work on:

Download from <http://barlow.web.cern.ch/barlow/Sample1.txt> etc

First
column is
0 or 1 for C
or D

```
sample1
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
...
```

Small
added
noise

```
sample2
0 1.587052 4.715568 -0.8595715 1.504009 2.145417
1 2.52062 2.682234 3.909693 0.2611399 0.3924642
1 -0.5450664 -1.449915 -0.2813677 4.057942 0.9299015
0 -1.047951 4.223808 3.068302 9.673196 3.915838
1 -2.863264 1.250906 0.293735 -0.2080808 -0.6673748
1 -0.2963963 2.988054 1.449716 2.326187 -0.5594592
1 4.581936 6.263028 5.522227 3.473845 -2.042601
...
```

Large
added
noise

```
sample3
0 -0.7064082 3.266121 0.2208592 4.825086 0
0 0.912854 3.48706 0.3057296 4.402847 -0.07224356
0 0.2116067 4.659067 0.9210807 4.95437 -0.7723788
1 0.7854812 2.079436 1.336324 2.16746 0.5728526
0 0.1380971 0 1.143737 4.632105 0.2767737
0 0.4398898 4.436032 1.55822 3.477277 0.3308824
1 0 1.320041 3.46353 1.087296 1.499402
...
```

Medium
added
noise plus some
losses

*This page intentionally left blank
as a reminder to organise work groups*

Write your own ANN -

```
ALPHA=0.05 # learning parameter
```

```
nodes=c(5,7,10,1)    # 5 inputs, 2 hidden layers, with 7 and 10 nodes , 1 output
nlayers=length(nodes) -1      # 3 sets of weights
```

```
net=list() # set up empty list
# net[[ j ]] holds weight matrix feeding nodes of layer j+1 from nodes in layer j
```

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

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

```
backprop <- function(layer,n1,n2,factor){ # recursive function used for back-propagation
  if(layer>1) for(n in 1:nodes[layer-1])
    backprop(layer-1,n2,n,factor*net[[layer]][n1,n2]*r[[layer]][n2]*(1-r[[layer]][n2]))
  net[[layer]][n1,n2] <-> net[[layer]][n1,n2] - ALPHA*factor*r[[layer]][n2]
}
```

```
netlearns <- function(x,truth) { # like netsays but changes weights
  r <- list() # to contain the outputs of all nodes in all layers
  r[[1]] <- x # the input layer
  for(layer in 1:nlayers) r[[layer+1]] <- as.vector(1/(1+exp(-net[[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))
}
```

Or download Fritsch & Günther's package

<https://cran.r-project.org/web/packages/neuralnet/neuralnet.pdf>

```
install.packages('neuralnet')
```

Do this once. It asks you to choose a mirror. Tip - don't choose an https site

```
library(neuralnet)
```

Do this once per session

```
help(neuralnet)
```

Just do this! and read it all very carefully, twice

```
df <- data.frame(truth,input1,input2)  
nnet<-neuralnet(truth~input1+input2,df,c(4,5))
```

Very basic example

```
nnet<-neuralnet(V1~V2+V3+V4+V5+V6,df,c(4,5),
```

less basic example

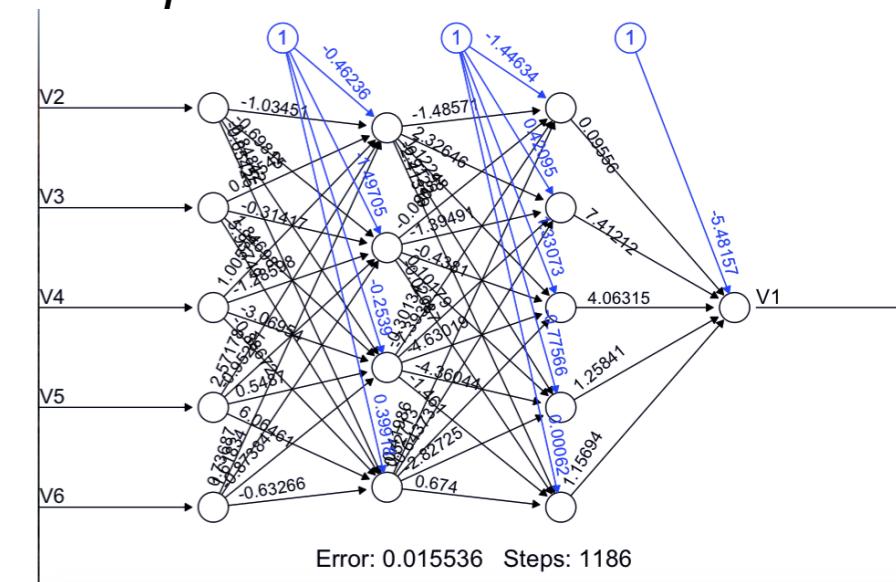
```
lifesign='full',  
algorithm='backprop',  
learningrate=0.05,  
linear.output=FALSE  
)
```

```
plot(nnet)
```

Nice picture of net

```
test=compute(nnet,t(c(1,2,3,2,1)))  
test$net.result
```

how it's used



Lab Session

6 Questions

1. What is the effect of varying the learning parameter α ?
2. What is the effect of using more, or fewer, nodes in the hidden layers?
3. What is the effect of using more, or fewer, hidden layers?
4. What is the effect of pre-processing the input data to give each data input mean zero and standard deviation 1? If you feel strong enough, also try Principal Component Analysis
5. What is the effect of using a tanh function rather than a sigmoid ? (Use different differential)
6. What happens if a network trained on one sample is applied to another sample?

The ‘what is the effect of...’ questions, refer to both the eventual separation and the training time. Sample 2 and sample 3 can be used for this - sample 1 is too easy.

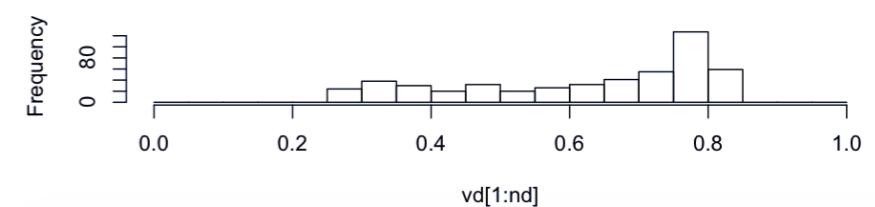
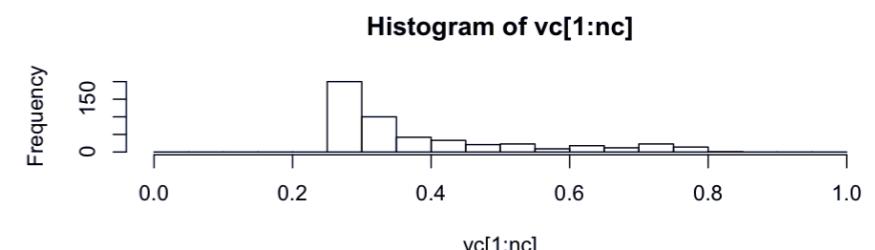
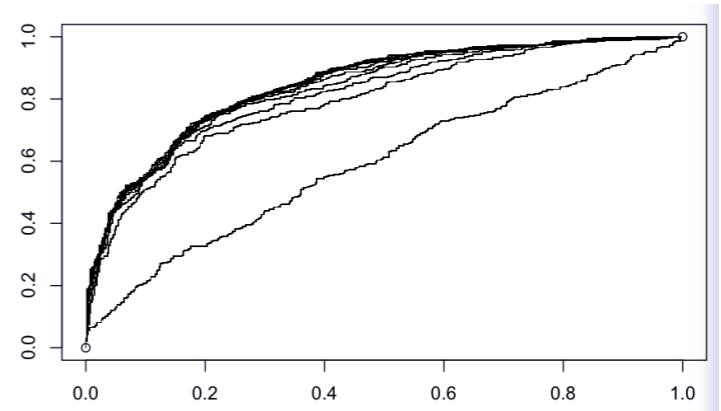
Some (possibly) useful R stuff

```
sample <- read.table("Sample1.txt", header=FALSE)
Nsample <- dim(sample)[1]
print(head(sample))
for (i in 1:Nsample) {print(sample[i,1]); print(sample[i,-1])}

plot(c(0,1),c(0,1))
v <- netsays(t(sample[,-1]))
p <- sample[order(v),1]
nc <- sum(sample[,1]==0)
nd <- Nsample-nc
nnc <- nc
nnd <- nd
for (i in 1:length(p)) {if(p[i]==1) {nd <- nd-1} else {nc <- nc-1}}
points(nc/nnc,nd/nnd,pch='.')

vc <- rep(0,nnc)
vd <- rep(0,nnd)
nc <- 0
nd <- 0
for (i in 1:Nsample){
  itype <- sample[i,1]
  isay <- netsays(as.numeric(sample[i,-1]))
  if(itype==0) {nc <- nc+1;vc[nc] <- isay} else {nd<- nd+1;vd[nd] <- isay}
}

hc <- hist(vc,breaks=seq(0,1,.05))
hd <- hist(vd,breaks=seq(0,1,.05))
```



Lab Session

Either write your own code, or download the neuralnet package, as directed

Set up a network with 2 hidden layers, with 8 and 5 nodes

Train and test with the file sample1. It should achieve perfect separation. If not, keep trying till you do.

Train and test with sample2. Draw ROC plots to show the performance. Make sure you are not over-training.

Now try sample3 in the same way.

Tackle your allocated question. Prepare a couple of slides to show your results, for presentation in the round-up session.

When you're done, if you've time, tackle any of the other problems that look interesting.