7 Neural Networks

7.1 Introduction

We have seen how we can consider classification and discrimination problems as a form of modelling the relationship between a categorical variable and various explanatory variables. We could make this more explicit and use, for example, logistic regression techniques. For example, suppose we have two categories, A and B, and explanatory variables $x_1,\ldots,x_k$ then we could model the probability that an object with values of $x_1,\ldots,x_k$ belongs to category A as a logistic function of the $x_1,\ldots,x_k$:

$$P[\text{belongs to A}] = \frac{\exp\{\alpha + \beta_1 x_1 + \ldots + \beta_k x_k\}}{1 + \exp\{\alpha + \beta_1 x_1 + \ldots + \beta_k x_k\}}$$

and then estimate the unknown parameters β_i from training data on objects with known classifications. New observations would be classified by classifying them as of type A if the estimated probability of belonging to A is > 0.5, otherwise classify them as of type B. The technique is widely used and is very effective, it is known as *logistic discrimination*. It can readily handle cases where the x_i are a mixture of continuous and binary variables. If there is an explanatory variable whish is categorical with $k>2$ levels then it needs to be replaced by $k-1$ dummy binary variables (though this step can be avoided with tree-based methods).

The idea could be extended to discrimination and classification with several categories, *multiple logistic discrimination.*

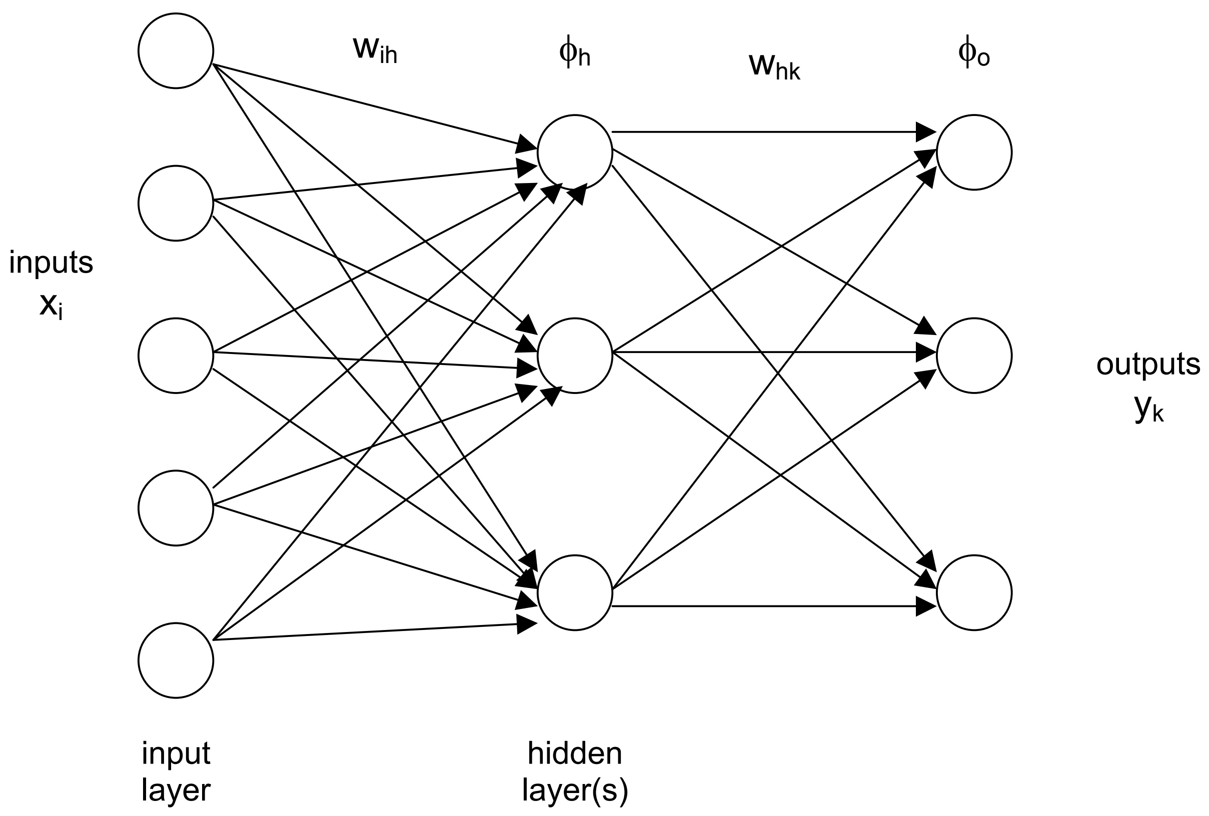
Neural networks, from a statistical point of view, can be thought of as a further extension of the idea and special cases of them are essentially non-linear logistic models. However, the technique is rather more general than just non-linear logistic modelling. It also has analogies with generalized additive modelling (mentioned very briefly on p97).

The full model for a feed-forward neural network with one hidden layer is

$$y_k = \phi_0\left(\alpha_k + \sum_h w_{hk}\phi_h(\alpha_h + \sum_i w_{ih}x_i)\right)$$

where the 'inputs' are x_i (i.e. values of explanatory variables), the 'outputs' are y_k (i.e. values of the dependent variable, and the α_j and w_{ij} are unknown parameters which have to be estimated (i.e. the network has to be 'trained') by minimising some fitting criterion, e.g. least squares or a measure of entropy. The functions ϕ_j are 'activation functions' and are often taken to be the logistic function $\phi(x)=\exp(x)/\{1+\exp(x)\}$. The w_{ij} are usually thought of as *weights* feeding forward input from the observations through a 'hidden layer' of units (ϕ_h) to output units which also consist of activation functions ϕ_o.

The model is often represented graphically as a set of inputs linked through a hidden layer to the outputs:



The number of inputs is the number of explanatory variables x_i, the number of outputs is the number of levels of y_k (if y_k is categorical), or the dimension of y_k (if y_k is continuous) and the number of 'hidden units' is open to choice. The greater the number of hidden units the larger the number of parameters to be estimated and (generally) the better will be the fit of the predicted y_k with the observed y_k.

7.2 Examples

These next two examples are taken from the `help(nnet)` output and are illustrations on the iris data yet again, this time the classification is based on (i.e. the neural network is trained on) a random 50% sample of the data and evaluated on the other 50%. In the first example the target values are taken to be the vectors (1,0,0), (0,1,0) and (0,0,1) for the three species (i.e. indicator variables) and we classify new data (i.e. with new values of the sepal and petal measurements) by which column has the maximum estimated value.

```
> library(nnet)
> data(iris3)
># use half the iris data
> ir <- rbind(iris3[,,1],iris3[,,2],iris3[,,3])
> targets <-class.ind(c(rep("s",50),rep("c",50),rep("v",50)))
> samp<-c(sample(1:50,25),sample(51:100,25),
+ sample(101:150,25))
>ir1 <- nnet(ir[samp,], targets[samp,], size=2, rang=0.1,
+                   decay=5e-4, maxit=200)
# weights:  19
initial  value 54.827508
iter  10 value 30.105123
iter  20 value 18.718125
… … … … … … … … … … … …
… … … … … … … … … … … …
iter 190 value 0.532753
iter 200 value 0.532392
final  value 0.532392
stopped after 200 iterations
>      test.cl <- function(true, pred){
+              true <- max.col(true)
+              cres <- max.col(pred)
+              table(true, cres)
+      }
>      test.cl(targets[-samp,], predict(ir1, ir[-samp,]))
    cres
true  1  2  3
   1 24  0  1
   2  0 25  0
   3  2  0 23
```

Thus, the classification rule only misclassifies 3 out of the 75 flowers which were not used in the analysis. If we used a net with only 1 unit in the hidden layer:

```
> ir1 <- nnet(ir[samp,], targets[samp,], size=1, rang=0.1,
+               decay=5e-4, maxit=200)
# weights:  11
initial  value 57.220735
iter  10 value 35.168339
…    …    …    …    …    …
iter  60 value 17.184611
final  value 17.167133
converged
>      test.cl <- function(true, pred){
+              true <- max.col(true)
+              cres <- max.col(pred)
+              table(true, cres)
+      }
>      test.cl(targets[-samp,], predict(ir1, ir[-samp,]))
    cres
true  1  2  3
   1 22  0  3
   2  0 25  0
   3  0  0 25
>
```

then it is still only 3, though a different 3 clearly. To see what the actual values of the predictions are we can print the first five rows of the estimated target values:

```
> predict(ir1, ir[-samp,])[1:5,]
             c         s v
[1,] 0.1795149 0.9778684 0
[2,] 0.1822938 0.9747983 0
[3,] 0.1785939 0.9788104 0
[4,] 0.1758644 0.9813966 0
[5,] 0.1850007 0.9714523 0
```

and we see that although it does not estimate the values as precisely (0,1,0) (or (1,0,0) or (0,0,1)) they are close. Hence the use of the `mac.col` function above.

We can find out more about the actual fitted (or trained) network, including the estimated weights with `summary()` etc:

```
> ir1
a 4-1-3 network with 11 weights
options were - decay=5e-04
> summary(ir1)
a 4-1-3 network with 11 weights
options were - decay=5e-04
 b->h1 i1->h1 i2->h1 i3->h1 i4->h1
 -0.15   0.41   0.74  -1.01  -1.18
 b->o1 h1->o1
 -0.06  -1.59
 b->o2 h1->o2
 -6.59  11.28
 b->o3 h1->o3
  3.75 -39.97
```

and we could draw a graphical representation putting in values of the weights along the arrows.

Another way of tackling the same problem is given by the following:

```
> ird <- data.frame(rbind(iris3[,,1], iris3[,,2], iris3[,,3]),
+          species=c(rep("s",50), rep("c", 50), rep("v", 50)))
>      ir.nn2 <- nnet(species ~ ., data=ird, subset=samp,
+ size=2, rang=0.1,  decay=5e-4, maxit=200)
# weights:  19
initial  value 82.614238
iter  10 value 27.381769
…    …    …    …    …
iter 200 value 0.481454
final  value 0.481454
stopped after 200 iterations
>      table(ird$species[-samp], predict(ir.nn2, ird[-samp,],
type="class"))

     c  s  v
  c 24  0  1
  s  0 25  0
  v  2  0 23
>
```

again, 3 of the new data are misclassified. However, if try a net with only one hidden unit we actually succeed slightly better:

```
>  ir.nn2 <- nnet(species ~ ., data=ird, subset=samp, size=1,
+ rang=0.1, decay=5e-4, maxit=200)
# weights:  11
initial  value 82.400908
final  value 3.270152
converged
>   table(ird$species[-samp], predict(ir.nn2, ird[-samp,],
+ type="class"))

     c  s  v
  c 24  0  1
  s  0 25  0
  v  1  0 24


> summary(ir.nn2)
a 4-1-3 network with 11 weights
options were - softmax modelling  decay=5e-04
 b->h1 i1->h1 i2->h1 i3->h1 i4->h1
 -1.79  -0.44  -0.91   1.05   1.65
 b->o1 h1->o1
  7.11  -0.99
 b->o2 h1->o2
 12.30 -36.31
 b->o3 h1->o3
-19.45  37.43
```

Exercise: Try modifying the **R** commands above to train a network on a much smaller sample, say 10 from each species, and the classifying the remainder. This can be done by changing the 25 to 10 in each of the three sample commands on P171. (I found that the misclassification rate on the new data was 6 out of 120 and even with training samples of 5 from each species it was 8 out of 135).

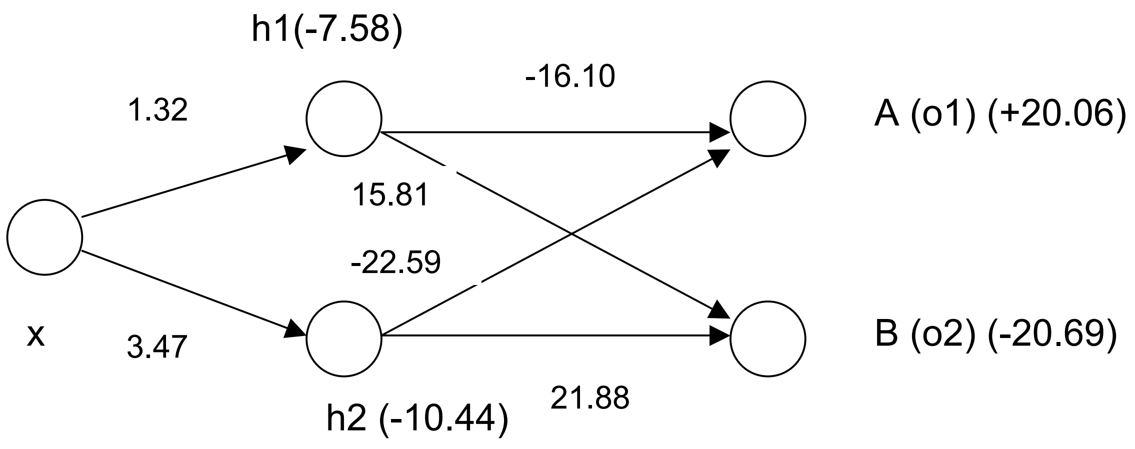
Simple Example:

This is an artificial example: the objective is to train a network with a hidden layer containing two units to return a value A for low numbers and a value B for high ones. The following code sets up a dataframe (`nick`) which has 8 rows and two columns. The first column has the values of `x` and the second the targets. The first five rows will be used for training the net and the last three will be fed into the trained net for classification, so the first 3 rows have low values of `x` and `target` value `A`, the next 2 rows have high values of `x` and the `target` value `B` and the final 3 rows have test values of `x` and unknown classifications.

```
> library(nnet) # open nnet library
> nick<-
+ data.frame(x=c(1.1,1.7,1.3,5.6,7.2,8.1,1.8,3.0),
+ targets=c(rep("A",3),rep("B",2),rep("U",3)))
> attach(nick)
# check dataframe is ok
> nick
     x   targets
1   1.1       A
2   1.7       A
3   1.3       A
4   5.6       B
5   7.2       B
6   8.1       U
7   1.8       U
8   3.0       U
> nick.net<-nnet(targets~.,data=nick[1:5,],size=2)
# weights:  10
initial  value 3.844981
final  value 0.039811
converged
Warning message:
group(s) U are empty in: nnet.formula(targets ~ ., data =
nick[1:5, ], size = 2)
```

```
# check predictions on training data
> predict(nick.net,nick[1:5,],type="class")
[1] "A" "A" "A" "B" "B"
# now classify new data
> predict(nick.net,nick[6:8,],type="class")
[1] "B" "A" "A"

# see what the predictions look like numerically
> predict(nick.net,nick[6:8,])
             A            B
6 1.364219e-15 1.000000e+00
7 1.000000e+00 4.659797e-18
8 1.000000e+00 1.769726e-08
> predict(nick.net,nick[1:5,])
             A            B
1 1.000000e+00 2.286416e-18
2 1.000000e+00 3.757951e-18
3 1.000000e+00 2.477393e-18
4 1.523690e-08 1.000000e+00
5 2.161339e-14 1.000000e+00
>
# look at estimates of weights.
> summary(nick.net)
a 1-2-2 network with 10 weights
options were - softmax modelling
 b->h1 i1->h1
 -7.58   1.32
 b->h2 i1->h2
-10.44   3.47
 b->o1 h1->o1 h2->o1
 20.06 -16.10 -22.59
 b->o2 h1->o2 h2->o2
-20.69  15.81  21.88
```



7.3 Summary

The above account is only a very brief introduction to simple neural networks, with particular reference to their use for classification. As with tree-based methods they can also be used for regression problems. Little has been said about the use and choice of activation functions, fitting criteria etc and examples have been given entirely in the context of the simple and basic facilities offered in the `nnet` library of **R**. To find out more then look at the reference Ripley (1996) given on p1, this is written with statistical terminology and largely from a statistical point of view. Another definitive reference is Chris Bishop (1995), *Neural Networks for Pattern Recognition,* Oxford, Clarendon Press.