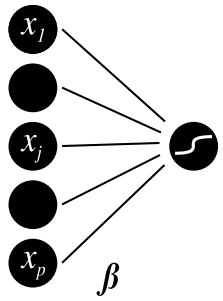
Outline

Neural Networks

The term *Neural Network* has evolved to encompass a large class of models and learning methods. We describe the most widely used neural network called the *single hidden layer back-propagation network*.
Initially motivated by biological processes, NN are simply another nonlinear method which can be used to find good predictions for classification and regression problems.

Each node of the network receives input from other nodes.

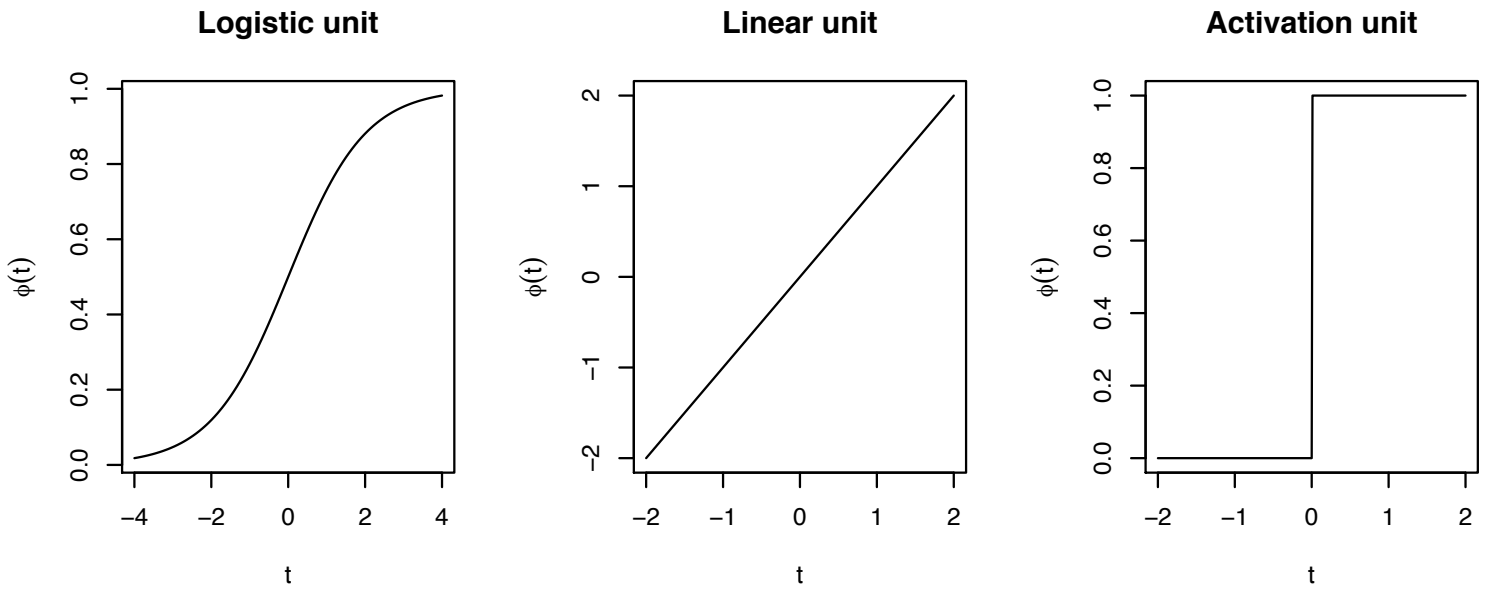


will pass a a signal along itself to other nodes in the network. For some so-called activation function f,

$$\text{total inputs to node } j: \qquad x_j = \sum_{i \to j} w_{ij} y_i$$

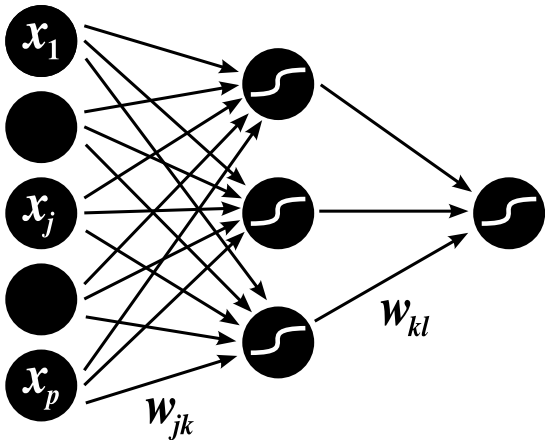
$$\text{output from node } j: \qquad y_j = f(x_i).$$

Many *activation functions* f are possible.



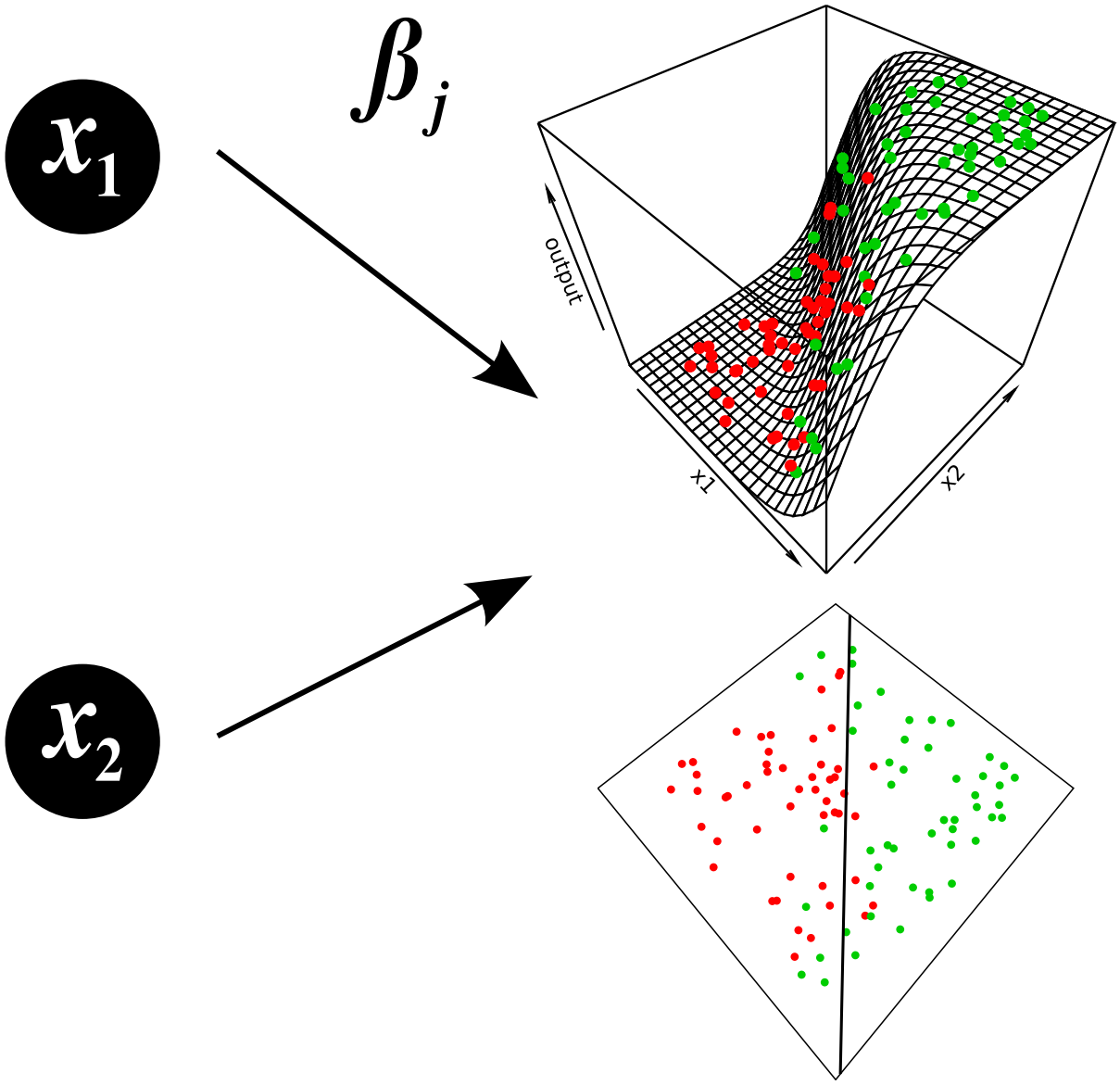
The ‘activation unit’ is not differentiable rendering it difficult to use for modelling. The linear unit is uninteresting in a network.

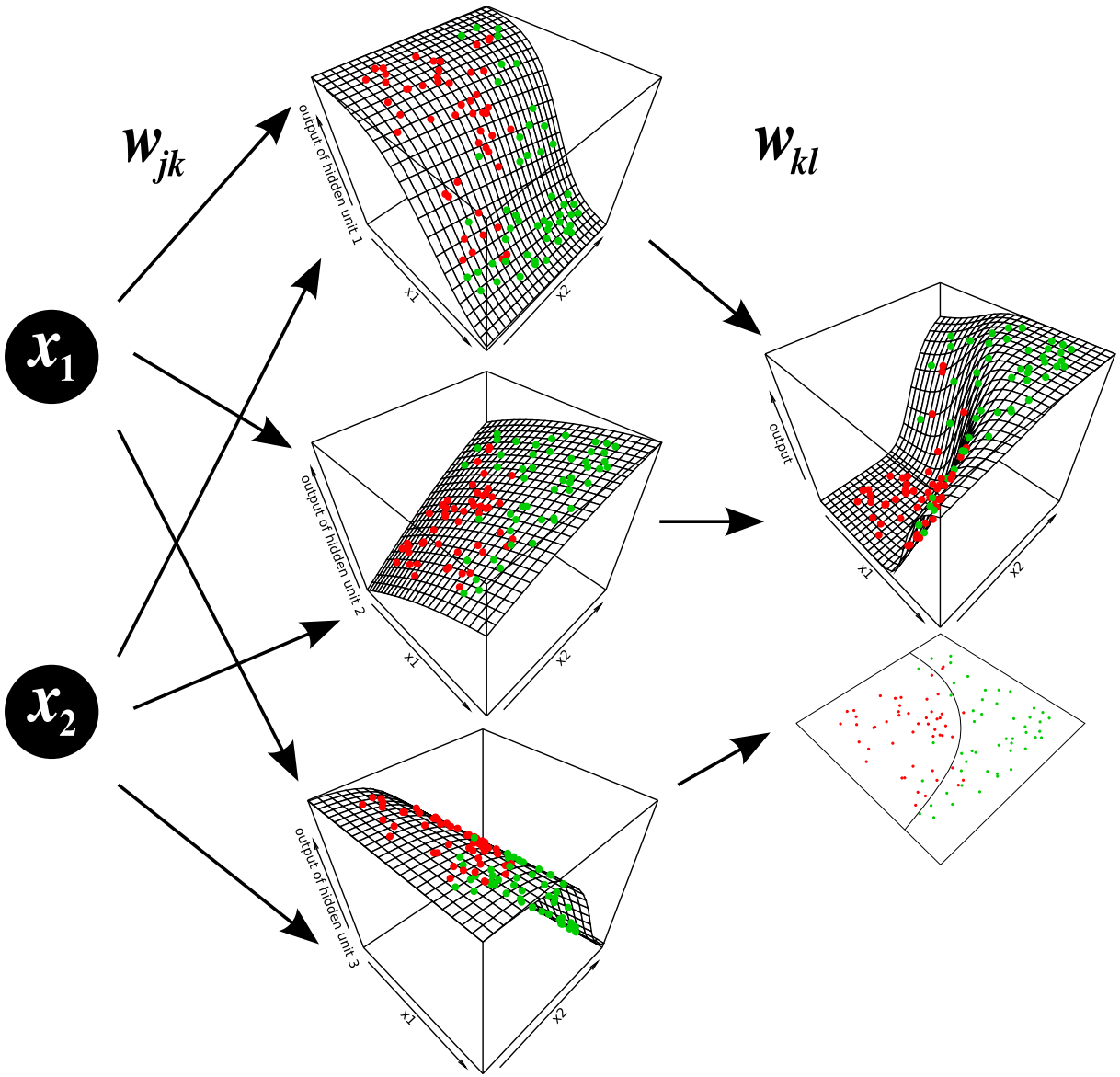
Neural networks combine many layers of nodes to allow for a very flexible model, a single hidden layer neural network in particular consists of 3 layers of nodes, below is an example of one.



A bias term is also incorporated at each neuron which provides a constant output of 1, so at node j

$$x_j = \sum_{i \to j} w_{ij} y_i = w_0 + \sum_{i \to j, i \neq 0} w_{ij} y_i.$$


x_1
β_j
x_2
output
x1
x2


w_{jk}
w_{kl}
x_1
x_2
output of hidden unit 1
output of hidden unit 2
output of hidden unit 3
output
x1
x2

There is nothing to stop us from adding more hidden layers.
It can be shown that single hidden layer neural networks can model arbitrarily complex surfaces (if the number of nodes in the hidden layer is large enough).
But more layers (“deep” networks) can sometimes model complex surfaces more easily.
When using linear activation units, the neural network collapses to a linear model. Logistic activation units are preferred as they are nonlinear and differentiable.

For inputs x_j, the output of the node l in the final layer can be expressed explicitly as

$$y_l = f\Big(\sum_{k\to l} w_{kl} f(\sum_{j\to k} w_{jk} x_j)\Big) \quad =: y_l(x,w)$$

for a neural network with a single hidden layer.
The statistical part is to find good weights w, given some training data.

Neural Networks can be used for both regression and classification. In the regression framework of Rumelhart and McClelland, for observations (X_i, Y_i), $i = 1, \ldots, n$, we train neural networks with a single output node y_1 by seeking weights w to minimize

$$E(w) = \sum_{i=1}^{n} |Y_i - y_1(X_i, w)|^2.$$

A generic drawback of Neural Networks is that $E(w)$ is a non-convex function and can have multiple minima. It is thus not easy to find a good solution.

In a classification setting, we now have K output nodes $y_1, \ldots, y_K$, each representing one of the classes. Let $Y_{i,k} := 1\{Y_i = k\}$ for $k = 1, \ldots, K$.
By augmenting the final outputs, it is straightforward to enforce that each output node returns probability predictions $p_{i,k} := P(Y = k|X_i)$ via the *softmax* transformation

$$p_{i,k} = \frac{\exp y_k(X_i)}{\sum_{l=1}^{K} \exp y_l(X_i)}.$$

A measure of fit is via likelihoods, using

$$L(w) \propto \prod_{i=1}^{n} \prod_{\text{outputs } k} (p_{i,k})^{Y_{i,k}}.$$

We can seek weights to maximise the log-likelihood

$$\ell(w) \propto \sum_{i=1}^{n} \sum_{k} Y_{i,k} \log p_{i,k}.$$

As the log-likelihood attains a maximum at

$$\sum_i \sum_k Y_{i,k} \log Y_{i,k},$$

it is conventional to consider finding weights to minimize the cross-entropy

$$E(w) = \sum_p \sum_k Y_{i,k} \log \frac{Y_{i,k}}{p_{i,k}} = \sum_p \sum_k Y_{i,k} \log Y_{i,k} - \sum_p \sum_k Y_{i,k} \log p_{i,k},$$

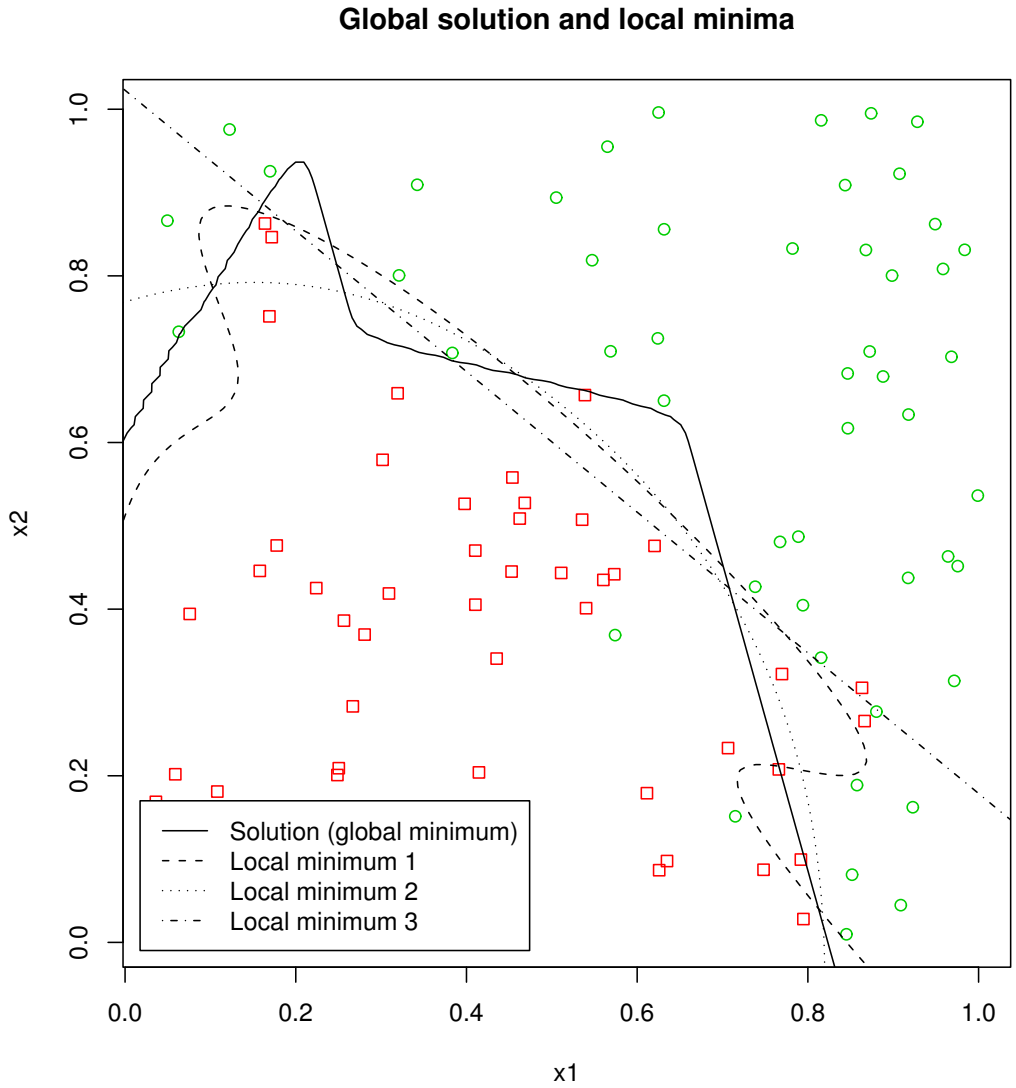
so $E(w) \geq 0$ with equality iff we can find w so that $p_{i,k} = Y_{i,k}$, i.e. a perfect fit (on the training data).
For both criteria, E can be minimised via gradient descent with update rule

$$w_{ij} \leftarrow w_{ij} - \eta \frac{\partial E}{\partial w_{ij}}.$$

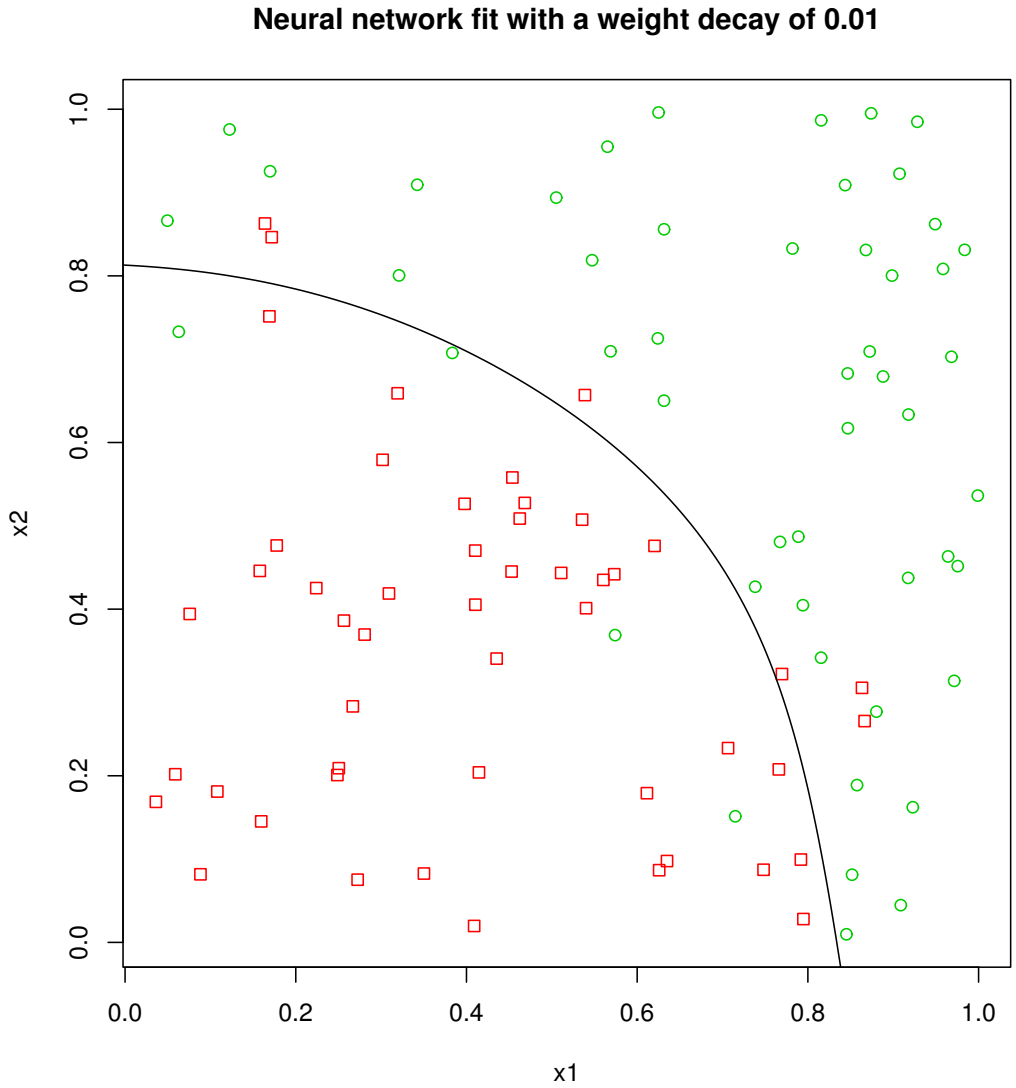
Corresponding algorithm called *back-propagation*.

- Due to the modular nature of the nodes, partial derivatives are easily calculated via the chain rule, which leads to an algorithm called *back-propagation*.
- In back-propagation, predictions are made in a “forward-pass” through the network, while derivatives are computed in a “backward-pass” propagating error information towards earlier layers of network.
- As the high-dimensional likelihood surface need not be convex, we often find suboptimal maxima.
- With large numbers of nodes in the network, we have to be careful not to overfit. Regularisation is obtained bys a combination of
  - not fitting until convergence
  - Using weight decay, a regularization penalty on the weights w.
  - Choosing a simple but suitable network structure.

Neural Networks



Neural Networks



Cushings data (load with `data(Cushings)` in package `MASS`).

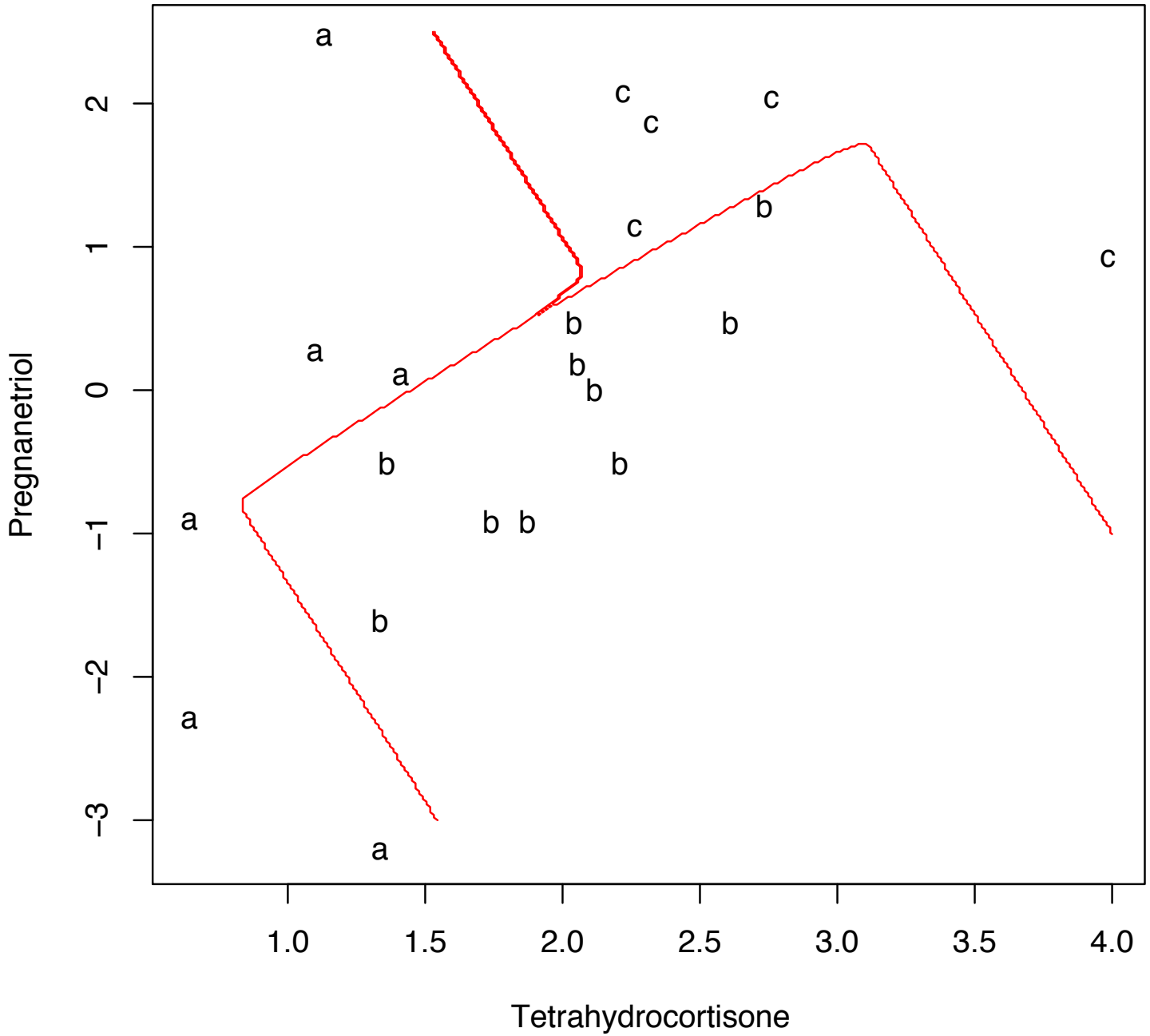
```
> ?Cushings
Cushings                  package:MASS                  R Documentation

Diagnostic Tests on Patients with Cushing's Syndrome

Description:
     Cushing's syndrome is a hypertensive disorder associated with
     over-secretion of cortisol by the adrenal gland. The observations
     are urinary excretion rates of two steroid metabolites.

Format:
     The 'Cushings' data frame has 27 rows and 3 columns:

     'Tetrahydrocortisone' urinary excretion rate (mg/24hr) of
          Tetrahydrocortisone.

     'Pregnanetriol' urinary excretion rate (mg/24hr) of
          Pregnanetriol.

     'Type' underlying type of syndrome, coded 'a' (adenoma) , 'b'
          (bilateral hyperplasia), 'c' (carcinoma) or 'u' for unknown.
```

```
## we will not deal with the untyped data here
cush <- Cushings[Cushings$Type!="u",]
cush[,1:2] <- log(cush[,1:2])

## plot
pairs(cush)


## fit neural network with 5 nodes in the hidden layer
cush.nnet <- nnet(Type ~ . , data=cush, size=5)
```

Tetrahydrocortisone
Pregnanetriol
Type
−3
−2
−1
0
1
2
1.0
2.0
3.0
4.0
1.5
2.5

Display the decision boundaries.

```
## take a lattice of points
## 100 by 100 lattice
m <- 100
x <- seq(0,4,length.out=m)
y <- seq(-3,2.5,length.out=m)
z <- data.frame(expand.grid(
              Tetrahydrocortisone=x,
              Pregnanetriol=y))
cush.nnp <- predict(cush.nnet,z)

## plot the data and decision boundaries
## classes are a,b,c =1,2,3 so set contours at 1.5 and 2.5
plot(cush[,1:2], pch=as.character(cush$Type))
contour(x, y, matrix(max.col(cush.nnp),m,m), levels=c(1.5,2
        add=TRUE, d=FALSE, lty=1, col=2)
```


Pregnanetriol
Tetrahydrocortisone
2
1
0
−1
−2
−3
1.0
1.5
2.0
2.5
3.0
3.5
4.0
a
c
b