Neural Network Discriminant Analysis: Statistical Aspects

Brian D. Ripley

Professor of Applied Statistics
University of Oxford

Programme

- General philosophy of neural networks
- Feed-forward neural networks
- Selecting and combining network architectures
- Estimating the performance of a network

For simplicity, all the examples are feed-forward neural networks (MLPs), but the principles apply much more widely, including to radial basis function networks.

What is a Neural Network?

Opinions differ. Here's three:

Haykin (1994) (after Aleksander & Morton (1990)):

'A neural network is a massively parallel distributed processor that has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects:

1. Knowledge is acquired by the network through a learning process.
2. Interneuron connection strengths known as synaptic weights are used to store the knowledge.'

[Many commonly accepted methods are excluded.]

Ripley (1996):

'This book is not about the impact of the study of neural networks as models of animal brains, but discusses what are more accurately (but rarely) called *artificial* neural networks which have been developed by a community which was originally biologically motivated (although many 'neural network' methods were not). Thus for the purposes of this book, a neural network is a method which arose or was popularized by the neural network community and has been or could be used for pattern recognition.'

Current FAQ (10/3/96) of `comp.ai.neural.nets`:

When we are talking about a neural network, we *should* usually better say "artificial neural network" (ANN). Biological neural networks are much more complicated in their elementary structures than the mathematical models we use for ANNs.

An ANN is a network of many simple processors ("units"), each possibly having a (small amount of) local memory. The units are connected by unidirectional communication channels ("connections"), which carry numeric (as opposed to symbolic) data. The units operate only on their local data and on the inputs they receive via the connections.

The design motivation is what distinguishes neural networks from other mathematical techniques:

A neural network is a processing device, either an algorithm, or actual hardware, whose design was motivated by the design and functioning of human brains and components thereof.

Most neural networks have some sort of "training" rule whereby the weights of connections are adjusted on the basis of presented patterns. In other words, neural networks "learn" from examples, just like children learn to recognize dogs from examples of dogs, and exhibit some structural capability for generalization.

Neural networks normally have great potential for parallelism, since the computations of the components are independent of each other.

It is a large field, with *lots* of new journals, including:

- Neural Computation
- Neural Networks
- IEEE Transactions on Neural Networks
- International Journal of Neurocomputing
- International Journal of Neural Networks
- International Journal of Neural Systems
- Neural Processing Letters
- Network: Computation in Neural Systems
- Sixth Generation Systems (formerly Neurocomputers)
- Connection Science: Journal of Neural Computing, Artificial Intelligence and Cognitive Research

Types of Neural Network

One type of neural network predominates, known variously as

- multi-layer perceptron
- backpropagation network
- feed-forward network (FFNN)

The other methods that seem to be used at all seriously are

- Radial basis function networks (RBFs)
- Kohonen's Self Organizing Maps

RBFs are used in very similar ways to FFNNs.

SOMs are a variant of cluster analysis / multidimensional scaling (See Ripley, 1996, chapter 9.)

'On-line' *vs* 'Batch'

Emphasis is placed on *'on-line'* methods, in which the data are not stored except through the changes the learning algorithm has made.

The theory of such algorithms is studied for a very long stream of examples, but the practical distinction is less clear, as this stream is made up by

- repeatedly cycling through the training set or
- sampling the training examples (with replacement).

In contrast, methods which use all the examples together are called *'batch'* methods.

It is often forgotten that there are intermediate positions, such as using small batches chosen from the training set.

Controversy

Anderson, Pellionisz and Rosenfeld (1990, p. 541):

'Neural networks are statistics for amateurs. A properly designed network, when learning and responding, performs good statistical inference, based on what is saw when it learned and what it sees when it responds. Most networks conceal the statistics from the user ....'

Hecht-Nielsen (1990) *'Neurocomputing'*:

The main advantage of neurocomputing over statistics seems to be neurocomputing's ability to draw from many more sources of inspiration (...) for its methods than are exploited by workers in statistics.

In essence, in terms of its everyday practice, there has only been modest progress in regression analysis since the days of Gauss.

Scott Fahlman (CMU) 27 Jan 1993 on `comp.ai.neural-nets`:

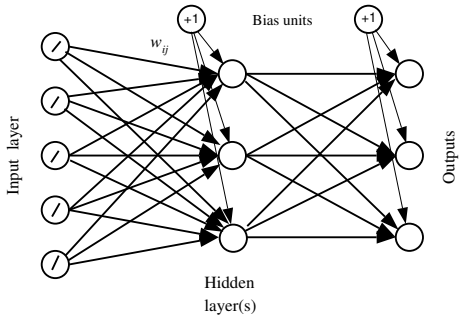
The real difference is that these neural net models are able to work with a different (richer, generally more powerful) set of architectures than the statisticians have been able to use in the past. ... Furthermore, the number, type, and topology of the basis functions and the number of parameters in the model might themselves be variable and subject to learning. (A few statistical methods, such as MARS, also explore the space of model sizes dynamically.)

... So the parameter estimation methods used in neural nets will look familiar and perhaps rather primitive to a statistician. It is the richness of the available set of models that gives us some new power.

E. Robert Tisdale (UCLA) 6 Nov 1994 on `comp.ai.neural-nets`:

There is nothing new or magical about neural networks. Artificial neural networks are sometimes used as "black box models" with lots of parameters which can be adjusted to approximate the response of any system provided that there are sufficient data to estimate the parameters accurately. The methods used to adjust these parameters have barely changed at all since they were invented by Karl Friedrich Gauss about 200 years ago. Neural network research seems to have contributed almost nothing at all save for jargon and confusion.

What is a FFNN?



A generic feed-forward neural network.

In mathematics:

$$y_k = \phi_o\left(\alpha_k + \sum_j w_{jk}\,\phi_h\left(\alpha_j + \sum_i w_{ij}\,x_i\right)\right)$$

where

- ϕ_h is usually the logistic $\phi_h(x) = e^x/(1+e^x)$
- ϕ_o is linear, logistic or threshold
- the 'biases' α_i can be replaced by weights to $+1$
- the number of *hidden units* can vary
- sometimes weights are set to zero or equal.

Extensions

Include 'skip-layer' connections from input to output to allow linear part:

$$y_k = \phi_o\left(\alpha_k + \sum_i w_{ik}x_i + \sum_j w_{jk}\,\phi_j\left(\alpha_j + \sum_i w_{ij}\,x_i\right)\right)$$

Equivalent to scaling down input weights and up output weights for one hidden unit, but can be set initially from linear fit.

RBFs are approximations of the form

$$f(\boldsymbol{x};\theta) = \alpha + \sum_j \beta_j G(\|\boldsymbol{x} - \boldsymbol{x}_j\|)$$

for (pre-specified) centres $\boldsymbol{x}_j$.

Examples of G proposed include

- the Gaussian $G(r) = \exp -r^2/2$
- the multiquadric $G(r) = \sqrt{(c^2 + r^2)}$ (Hardy, 1971)
- the thin-plate-spline function $G(r) = r^2 \log r$.

Universal Approximation

Both FFNNs and RBFs provide functions $g\colon \mathbb{R}^p \to \mathbb{R}^k$, parametrized by a 'size' N and some parameters θ. Both are *universal approximators* in the sense that the family of functions is dense in the topologies of uniform convergence on compacta and in $L_q(\mu)$ for any probability measure μ on $\mathbb{R}^p$, and any q.

There are even results on the rate of convergence in $L_2(\mu)$, $O(1/\sqrt{N})$, but they apply to quite restricted classes of target functions (Barron, Girosi, Jones).

Using a Flexible Family

Suppose we are given a family $f(x;\theta)$, how do we use it to do discrimination? How do we use linear functions?

(a) We model the log posterior probabilities

$$\log P(C = c \mid X = x) = g_c(x;\theta)$$

and fit by maximum (conditional) likelihood. Allocate future examples to class with highest $g_c(x;\widehat{\theta})$.
[*Logistic discrimination*]

(b) Fit the regression

$$E(I(C = c) \mid X = x) = g_c(x;\theta),$$

allocate to class with largest $g_c(x;\widehat{\theta})$, equivalent to the class with nearest 'target' $\boldsymbol{t}_c = (0,\ldots,0,1,0,\ldots)$ to $\boldsymbol{g}(x;\widehat{\theta})$.
[*'Minimum (mean) square classifier'*]

It has been known since Fisher that (b) is equivalent to LDA for two classes, and so mode efficient under a normal model with equal within-class covariances.

This is not so for more than two classes, and it has been suggested (Breiman & Ihaka) to generalize the following characterization of LDA:

(i) Fit the regression

$$E(I(C = c) \mid X = x) = g_c(x;\theta)$$

(ii) Apply standard LDA in the lower-dimensional space of fitted values.

Dietterich & Bakiri (1991, 1995) have suggested choosing higher-dimensional targets which are well-spaced in m dimensions for $m > K - 1$.

'Plug-in' *vs* 'Predictive'

Statistical decision theory tells us that as we don't know $P(C = c \mid X = x)$, we should use our best estimate $P(C = c \mid X = x, \mathcal{T})$, where $\mathcal{T}$ denotes the training data. This is called the *predictive* approach, whereas using $P(C = c \mid X = x;\widehat{\theta})$ is called the *plug-in* approach.

We have

$$P(c \mid x, \mathcal{T}) = \int P(c \mid x;\theta)p(\theta \mid \mathcal{T}, x)\,\mathrm{d}\theta$$

$$p(\theta \mid \mathcal{T}, x) \propto \prod_{i=1}^{n} P(c_i \mid x_i;\theta)p(x_i;\theta)\ p(\theta)p(x;\theta)$$

and normally assume that $p(x;\theta)$ does not in fact depend on θ (and hence $p(\theta \mid \mathcal{T}, x)$ does not).

There will be appreciable differences only if $p(c \mid x;\theta)$ is not highly peaked *and* $p(c \mid x;\theta)$ is not linear around the posterior mode for θ. The differences are small in LDA and logistic discrimination, but can be large in LDA and non-linear logistic methods.

Training a FFNN

Fitting a FFNN is normally done by least-squares or maximum likelihood, although unconventional optimization algorithms are used.

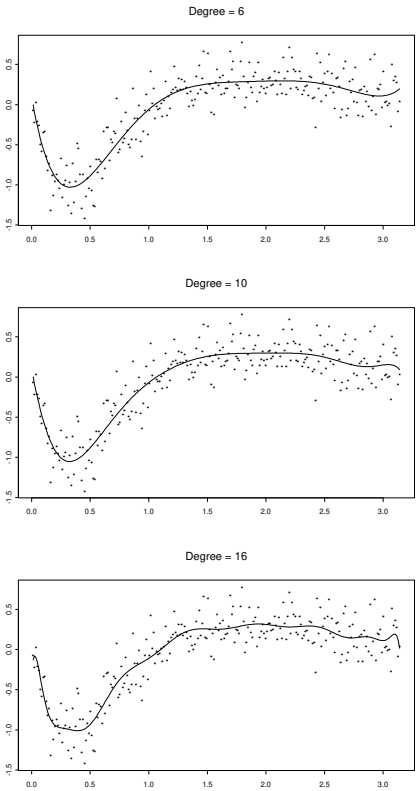
What's different about neural networks, compared to curve-fitting as done in statistics?

Main differences are:

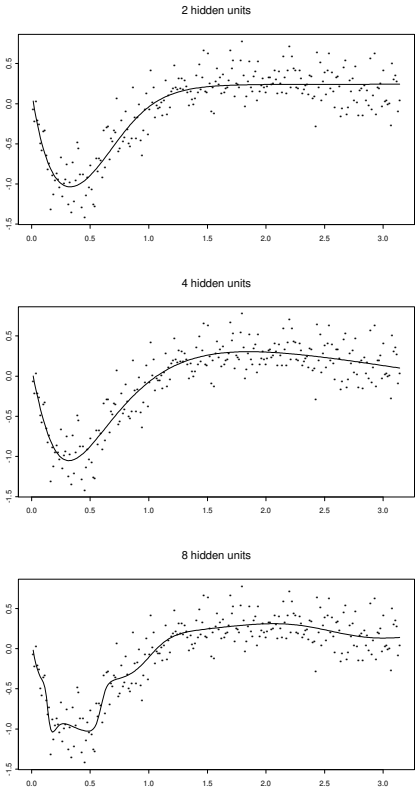
- Multiple local minima are common (especially for large models).
- Over-fitting is controlled by non-likelihood fitting.
- Sequence of possible architectures is not necessarily nested.

Consequence:

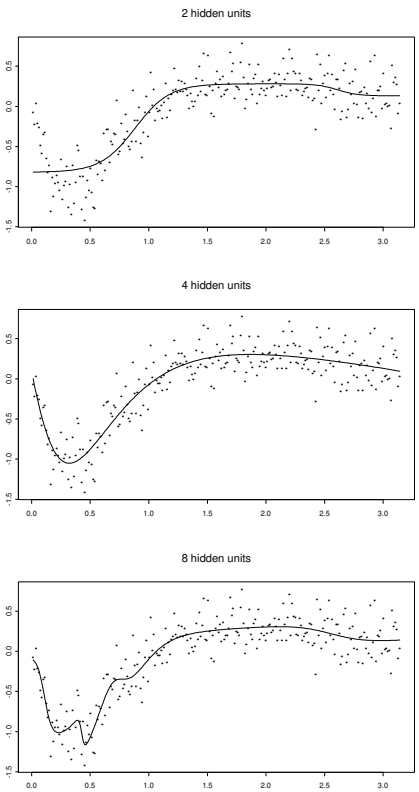
Everyone fitting polynomials will get the same answer, but almost no two people fitting neural networks will. (*Exception:* networks without hidden layers.)



Fits with polynomials.



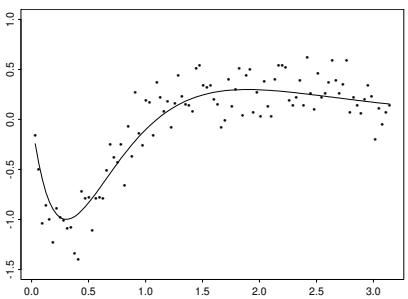
Fits with feed-forward neural networks.



More fits with feed-forward neural networks.

Demonstration example

Taken from Wahba & Wold (1975) where it illustrates smoothing splines.

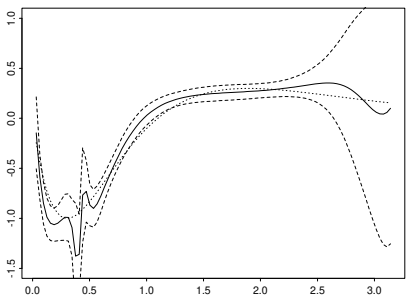


100 data points from a curve-fitting problem, with the true curve.

Throughout we assume that the true noise variance, $(0.2)^2$, is known.

Regression Neural Networks

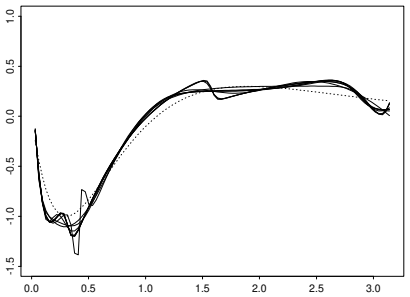
Model selection amounts to choosing the number of hidden units.



Curve fitted by 1-8-1 neural network, with ± 2 standard error bands and true curve (dotted).

The error limits come form standard regression theory, and are based on local linearization.

One problem is multiple local minima:



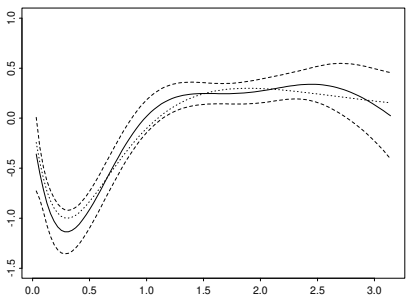
1-8-1 neural networks with different starting weights.

We clearly have over-fitting. Many ways to avoid this. Perhaps the best is to use *regularization* as used for smoothing splines. Somewhat easier for neural networks to use *weight decay*:

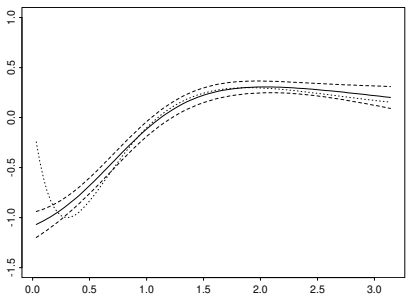
$$\sum_{i=1}^{n}[y_i - f(x_i; \mathbf{w})]^2 + \lambda \sum_{ij} w_{ij}^2$$

with the inputs and outputs suitably scaled.

Effect of Weight Decay

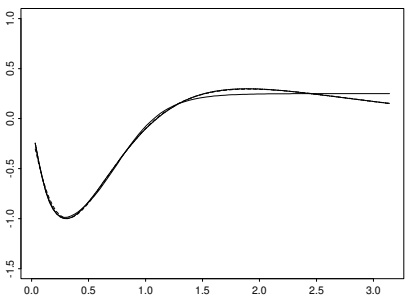


Curve fitted by 1-8-1 neural network, with ± 2 standard error bands and true curve $\lambda = 10^{-3}$ (above) and $\lambda = 0.1$ (below).



Bias/variance Compromises

We now have two ways to control the fit, the number of hidden units and λ, the degree of regularization. The true curve is not exactly representable by our model, so we have some bias.



Fitting to the true curve with 2 and 3 hidden units, and 8 units with $\lambda = 10^{-3}$.

Choosing fewer hidden units leads to more bias, as does adding a regularizer. But it also reduces their variability, so reduces the mean-square error. Can manipulate λ and the number of hidden units as a function of n to get the right compromise for very large n. Doesn't help in real problems!

Selection Techniques

Many classical techniques work with *nested* families of models, e.g. degree of a polynomial regression, number of hidden units in a 'raw' FFNN.

- *Forwards selection* Add terms until the fit is good enough (cascade correlation).
- *Backwards selection* Build a large model then drop terms (projection pursuit regression, MARS, ...).
- *Stepwise selection* Try both adding and dropping at each step.

Larger models will fit better, so need either significance tests or penalty for complexity. Many such

(a) *AIC* Akaike's 'An Information Criterion'

(b) *BIC* 'Bayesian Information Criterion'

(c) *MDL/MML* 'Minimum Description (or Message) Length— Rissanen and Wallace schools.

These work for non-nested models, but a search in model space is needed.

Cross-Validation

Suppose we have:

- A training set $\mathcal{T}$ used to fit weights.
- A validation set used to select architectures (models).
- A test set used to estimate performance of selected model.

We can then use the performance on the *validation* set to choose the best model. ('Hold-out' method of Highleyman, 1962.)

Wasteful of data, so split the training set into V parts. Use $(V-1)$ of these to train, and test the performance on the Vth part. Do this for all V parts. This creates a pseudo test set of size n. The performance measure is (almost) unbiased, since the parts are (almost) independent, but it is for performance at size $(1-1/V)n$.

Idea from the mid-1970s (M. Stone, Geisser, ...).

How do we choose V? Traditionally *leave-one-out*, so $V = n$. This gives very variable estimates of performance, and can be seriously wrong. Often better to choose $V \approx \sqrt{n}$ or 10, and much cheaper computationally.

This estimates the effect of over-fitting by $O(1/\sqrt{(}n))$ perturbations of the problem, the order of the effects of variation between training sets. The *Bootstrap* is based on same notion.

Caveat

(Cross-)validation applies to a well-defined procedure, so troubles with multiple local minima, early stopping and so on.

AIC, NIC, p_{eff}

Akaike's (1973, 4) 'An Information Criterion'.
ML measure of fit is the deviance

$$2\sum \log p(X_i, Y_i)/p(X_i, Y_i; \widehat{\theta})$$

on the training set (the supplied data). Consider a test set, that is an independent sample from the same process of the same size. We would expect the deviance for the test set to be larger.

AIC corrects this bias:

$$E\left[\text{deviance}_{\text{test}} - \text{deviance}_{\text{train}}\right] \sim 2p$$

for p parameters and for large n *under the model*. Choose model(s) which have small values of

$$\text{deviance}_{\text{train}} + 2p$$

Since AIC is based on maximum likelihood estimations, it has nothing to say in our example if regularization is used.

If we drop the requirements that the model is true, and that fitting is by ML, we find

$$E\left[\text{deviance}_{\text{test}} - \text{deviance}_{\text{train}}\right] \sim 2p_{\text{eff}}$$

where $p_{\text{eff}} = \text{tr}[KJ^{-1}]$ and

$$J = -E\frac{\partial^2 \log p(X_i,Y_i;\theta_0)}{\partial\theta\,\partial\theta^T},\ K = \text{Var}\frac{\partial \log p(X_i,Y_i;\theta_0)}{\partial\theta}$$

and θ_0 is the 'least false' value of θ, that is minimizes

$$-E\log p(X, Y; \theta)$$

This is due to Moody (1991, 1992) and Murata, Yoshizawa & Amari (1991,3,4) (hence NIC). Derived (but not stated) by Stone (1977). Proof is easy, based on second-order Taylor expansions of the log-likelihood, *about a single local maximum*.

Note that AIC, NIC only correct the bias as an estimate of the expected test-set deviance. We have to consider the fluctuations. For large n, AIC-like procedures never choose too small a model, but may choose too large a one.

Asymptotically LOO CV is the same as NIC (Stone, 1977), and often accurately so in small samples too. NIC is much less work!

BIC and Bayes Factors

'BIC' often refers to the penalty $p \log n$. An approximation to the use of *Bayes factors*.

The predictive Bayesian approach assesses models via $P(\text{model} \mid \mathcal{T})$, integrating out 'weights' θ. Now

$$P(\text{model} \mid \mathcal{T}) \propto P(\text{model})P(\mathcal{T} \mid \text{model})$$

Strictly we average predictive distributions with weights $P(\text{model} \mid \mathcal{T})$. But if we choose, a crucial factor is

$$B_{st} = \frac{P(\mathcal{T} \mid \text{model}_s)}{P(\mathcal{T} \mid \text{model}_t)}$$

called the Bayes factor, since

$$\frac{P(\text{model}_s \mid \mathcal{T})}{P(\text{model}_t \mid \mathcal{T})} = B_{st}\frac{P(\text{model}_s)}{P(\text{model}_t)}$$

Often

$$\begin{aligned}\log P(\mathcal{T} \mid \text{model}) \;\approx\; & \log \ell(\widehat{\theta}; \mathcal{T}) + \log p(\widehat{\theta}) \\ & + \tfrac{p}{2}\log 2\pi - \tfrac{1}{2}\log|H|\end{aligned}$$

for H the Hessian of the sum of the first two terms, so can be approximated.

The last approximation depends on a single local maximum, but we can sum up over multiple local maxima.

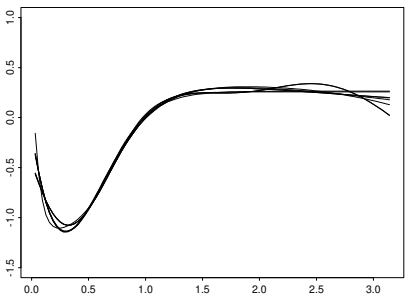
In example we look at $\sigma^2 \times \log P(\mathcal{T} \mid \text{model})$, since $\sigma^2 \times \log \ell(\widehat{\theta}; \mathcal{T})$ is the residual sum of squares.

Results for Example

Size	λ	SSq	SSq_{CV}	p_{eff}	NIC	Bayes
2	0	3.47	4.29	7	4.03	—
2	10^{-3}	3.57	4.78	5	3.98	3.30
3	0	4.12	4.81	10	4.92	—
3	10^{-3}	3.40	4.54	6	3.93	3.30
3	10^{-2}	3.61	4.91	6	4.11	3.72
3	10^{-1}	5.40	6.20	6	5.94	6.08
5	0	3.14	4.57	16	4.42	—
5	10^{-3}	3.21	4.51	17	4.59	3.38
5	10^{-2}	3.61	4.91	6	4.11	4.11
5	10^{-1}	5.38	6.18	6	5.92	5.98
8	0	2.89	5.35	25	4.89	—
8	10^{-3}	3.21	4.54	18	4.67	4.74
8	10^{-2}	3.58	4.94	7	4.16	4.63
8	10^{-1}	5.36	6.14	7	5.90	5.83

Column SSq_{CV} is based on 10-fold cross-validation, the same sets for all the models.

Model Uncertainty



The fitted curves for all the well-supported models.

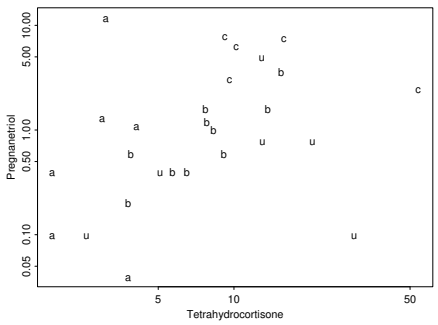
Finding the Bayes factors and weighted average is very time-consuming, *and* assumes the models are true. Stone (1974) suggested using LOO cross-validation to choose the weights in a weighted average, as have many people since.

Clearly we should take a weighted average over local minima, and it *may* suffice to take an unweighted average. In the Bayes approach we extend this to averaging over *models*: Wolpert's *stacked generalization*.

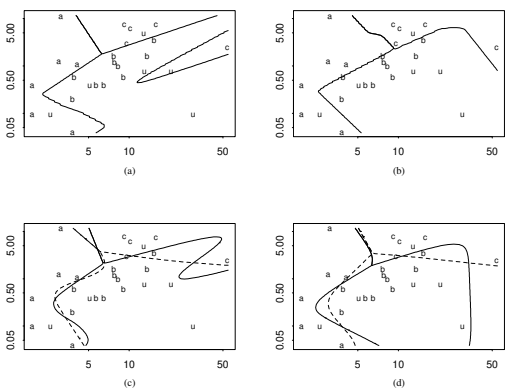
Cushing's Syndrome

Data on diagnostic tests on patients with Cushing's syndrome, a hypertensive disorder associated with over-secretion of cortisol by the adrenal gland.

This dataset has three recognised types of the syndrome represented as a, b, c. (These encode 'adenoma', 'bilateral hyperplasia' and 'carcinoma'.)



The observations are urinary excretion rates (mg/24h) of the tetrahydrocortisone and pregnanetriol, and are considered on log scale. Those marked u are future examples to be classified.

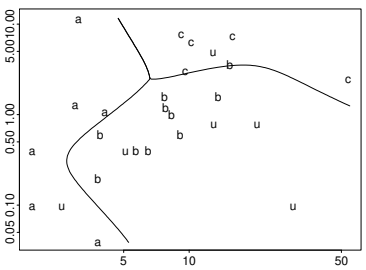


Neural network fits with two hidden units and connections from the input to output layer. Figures (a) and (b) are two solutions without weight decay which fit perfectly. Figures (c) and (d) show two local minima each, for $\lambda = 0.001$ and $\lambda = 0.01$ respectively. The dashed lines correspond to the local minimum that fits less well.

We can compare the seventeen local minima found with $\lambda = 0.01$. Six of them carry 5% or more of the total mass, with fitted values and weights

value	4.544	5.928	6.198	4.502	4.269	4.180
%	24.4	21.1	17.5	6.9	6.1	5.7

The second and sixth are shown at (d).



Predictive neural network fit to the data on Cushing's syndrome with two hidden units and $\lambda = 0.01$.