Can Statistical Theory Help Us Use Neural Networks Better?

Brian D. Ripley

Professor of Applied Statistics
University of Oxford

ripley@stats.ox.ac.uk
http://stats.ox.ac.uk/∼ripley

Overview

Yes!

But how much?

- What is a neural network?
- How do we 'train' a network?
- Bumps in the likelihood
- Predictive inference
- Two case studies of multi-class classification

Main background reference is:

Ripley, B.D. (1996) *Pattern Recognition and Neural Networks.* Cambridge University Press.

S code for the examples in

Venables, W.N. and Ripley, B.D. (1997) *Modern Applied Statistics with S-Plus.* Second edition. Springer.

Types of Neural Network

One type of neural network predominates, known variously as

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

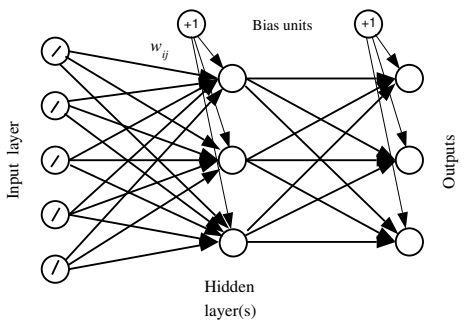
The other methods than seem to be used at all seriously are

- 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

What is a FFNN?



A generic feed-forward neural network.

In mathematics:

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

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:

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

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

Statisticians may like to think as this as a regression of logistic regressions plus the original variables.

Radial Basis functions

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 $\boldsymbol{x}_j$. They come from the field of approximation theory.

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 $g\colon \mathbb{R}^p \to \mathbb{R}^k$, of 'complexity' 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.

In other words, they can in principle model anything (noise as well as signal).

Training a FFNN

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

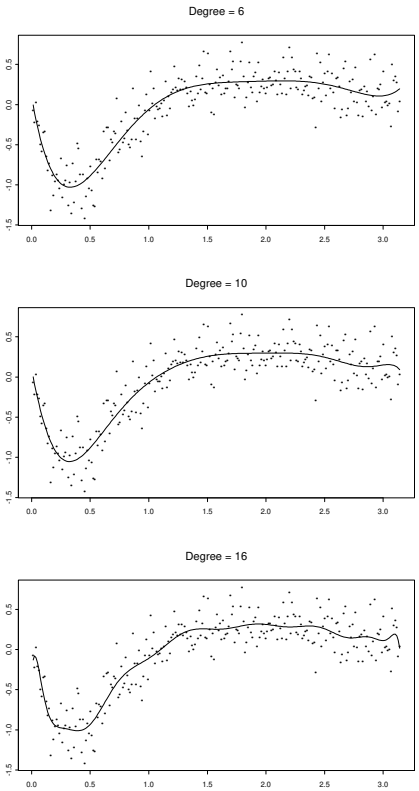
What's different about neural networks, compared to curve-fitting as done in statistics?

Main differences are:

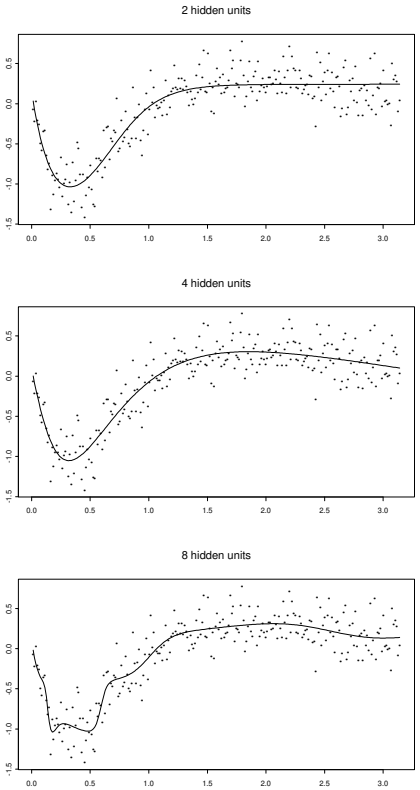
- Multiple local minima are common (especially for large models).
- Over-fitting is controlled by non-likelihood fitting.
- Sequence of possible architectures is not necessarily nested.

Consequence:

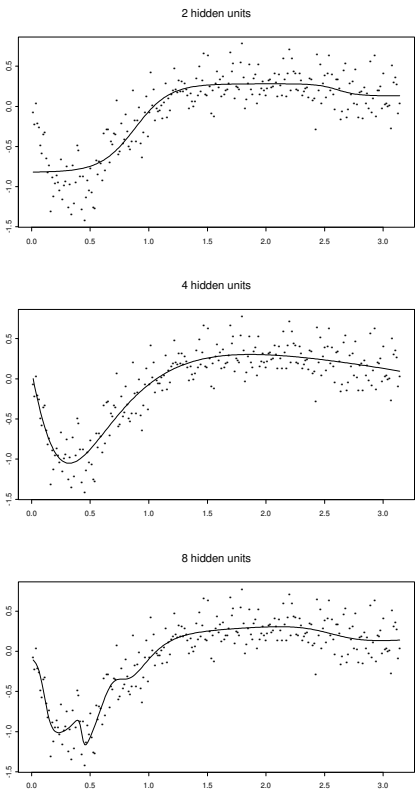
Everyone fitting polynomials will get the same answer, but almost no two people fitting neural networks will. (*Exception:* networks without hidden layers.)



Fits with polynomials.



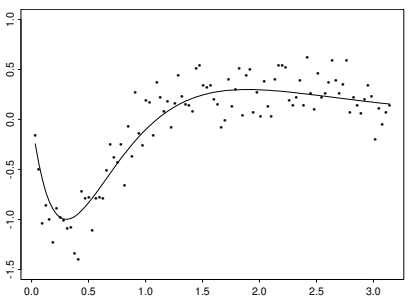
Fits with feed-forward neural networks.



More fits with feed-forward neural networks.

Demonstration example

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

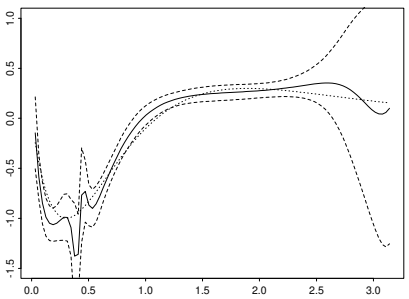


100 data points from a curve-fitting problem, with the true curve.

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

Regression Neural Networks

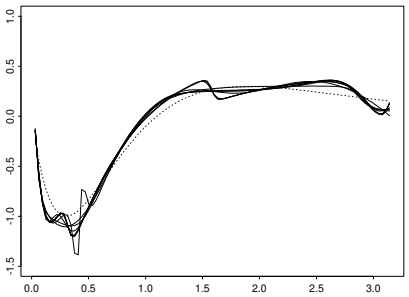
Model selection amounts to choosing the number of hidden units.



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

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

One problem is multiple local minima:



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

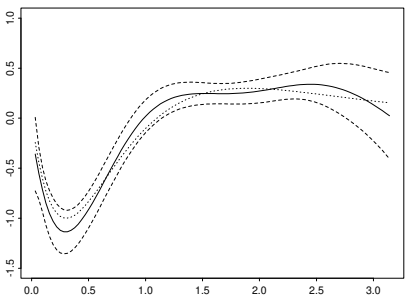
We clearly have over-fitting. Many ways to avoid this. Perhaps the best is to use *regularization* as used for smoothing splines. Somewhat easier for neural networks to use *weight decay*:

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

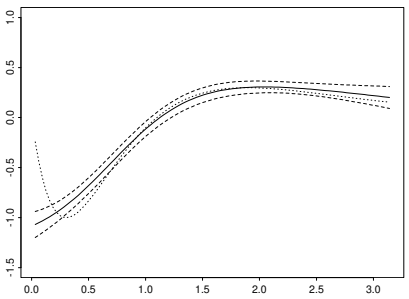
with the inputs and outputs suitably scaled.

Alternatively think of this as MAP with a Gaussian prior on the weights.

Effect of Weight Decay

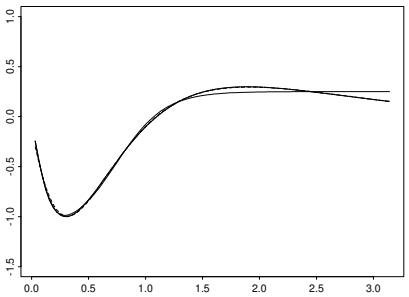


Curve fitted by 1-8-1 neural network, with ± 2 standard error bands and true curve $\lambda = 10^{-3}$ (above) and $\lambda = 0.1$ (below).



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.

Flexible Families for Discrimination

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

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

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'*]

(c) 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 $P(C = c \,|\, X = x)$, but should use our best estimate $P(C = c \,|\, X = x, \mathcal{T})$, where $\mathcal{T}$ denotes the training data. This is called the *predictive* approach, whereas using $P(C = c \,|\, X = x; \widehat{\theta})$ is called the *plug-in* approach.

We have

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

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

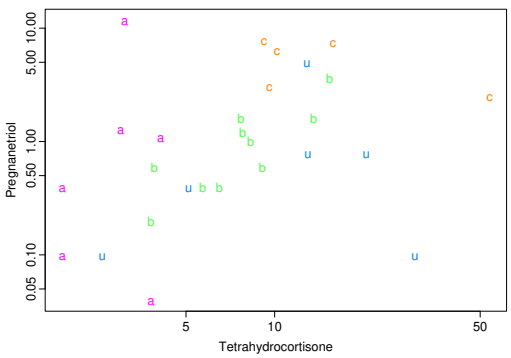
and normally assume that $p(x;\theta)$ does not in fact depend on θ (and hence $p(\theta \,|\, \mathcal{T}, x)$ does not).

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

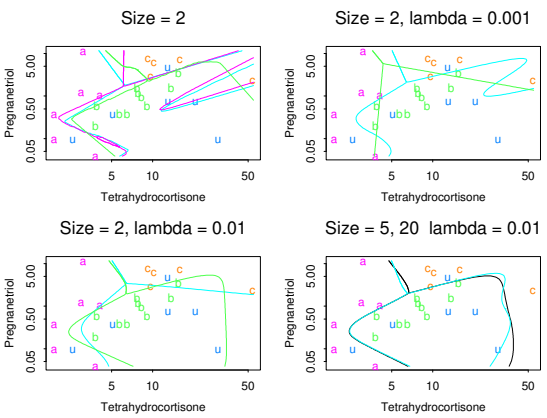
Cushing's Syndrome

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'.)

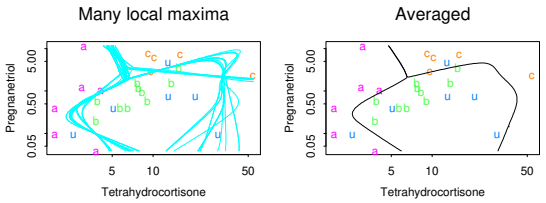


The observations are urinary excretion rates (mg/24h) of the tetrahydrocortisone and pregnanetriol, and are considered on log scale. Those marked u are future examples to be classified.



Two local minima are shown for each of the first three panels.

There are lots of local minima. Averaging across 20 fits is quite successful:



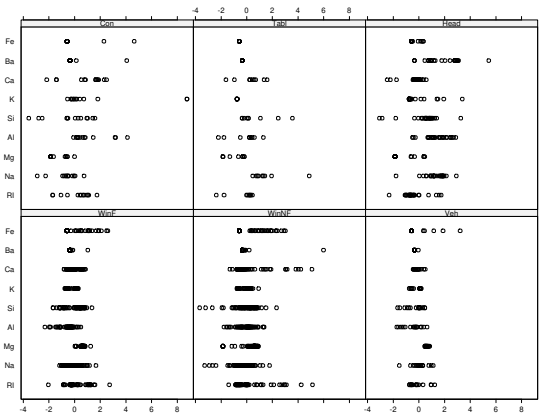
Neural networks of size 3 with $\lambda = 0.01$ applied to the Cushing's syndrome data. The right panel shows many local minima and their average (as the thick line).

Can we do better? Yes, a little, *if* we can do the integration. We currently break the integral up into regions about the local maxima of the posterior density, use a Gibbs sampler to sample around each peak and to correct a local Gaussian approximation, and to find a 'Bayes factor' for each region. By a similar process we can find Bayes factors over number of units and so on.

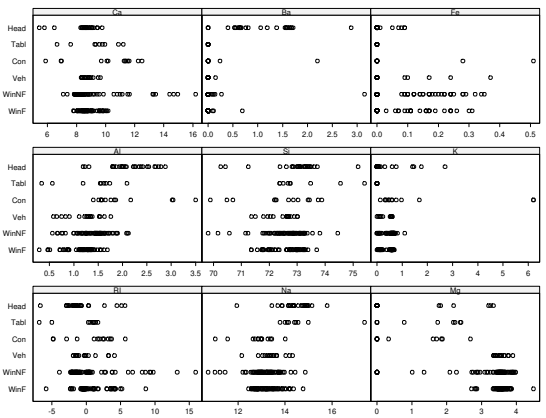
A Forensic Example

Data on 214 fragments of glass. Each has a measured refractive index and composition (weight percent of oxides of Na, Mg, Al, Si, K, Ca, Ba and Fe).

Grouped as window float glass (70), window non-float glass (76), vehicle window glass (17) and other (containers, tableware, headlamps) (22).



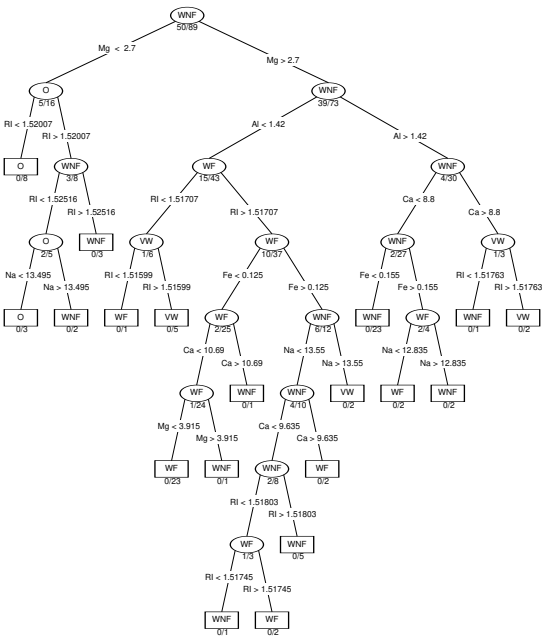
Strip plot by type of glass.



First used 89 to train, 96 to test.

methods	error rates %	
linear discriminant	41	22
nearest neighbour	26	17
neural network with 2 hidden units	38	16
ditto, averaged over fits	38	14
neural network with 6 hidden units	33	15
ditto, averaged over fits	28	12
classification tree	28	15

In the second column of error rates confusion between the two types of window glass is allowed.



Classification tree for the forensic glass example. At each node the majority classification is given, together with the error rate amongst training-set cases reaching that node.

On a 10-fold cross-validation experiment we found error rates (%) of

	# (hidden units)		
λ	2	4	8
0.0001	30.8	23.8	27.1
0.001	30.4	26.2	26.2
0.01	31.8	29.9	29.9

This shows that only a small amount of regularization by weight decay is needed in this dataset; this is related to the near-linear separation of some of the classes. The sharp variation in performance with λ and the number of hidden units in this example is unusual, and can be traced to the success in separating the rare classes.

The confusion matrix was

		Predicted					
		WinF	WinNF	Veh	Con	Tabl	Head
	WinF	56	11	3	0	0	0
T	WinNF	10	59	3	3	0	1
r	Veh	8	1	8	0	0	0
u	Con	0	4	0	8	0	1
e	Tabl	1	0	0	1	7	0
	Head	1	1	0	1	1	25