Neural Networks v. GLMs in pricing general insurance

Workshop to be presented by:

Julian Lowe (Chairman)
Louise Pryor

1. Executive Summary

Neural Networks are often referred to, with awe, as some mysterious representation of the human brain that can "solve" problems. They have also been referred to in previous GISG papers as having potential applications to general insurance pricing or reserving. The purpose of this paper is to remove some of this awe by explaining what Neural Networks are, how they compare with traditional statistical models, and consider what scope there is for their use in general insurance.

The paper is in three main sections. The first section describes what Neural Networks are. The second section briefly describes GLMs, and makes a few observations on the practical nitty gritty of using such models. The third section compares the two approaches from a theoretical perspective and with some practical examples based on real general insurance data. Finally, some references to Neural Network software and publications are given for anyone interested in pursuing the topic further.

The practical examples presented in this paper are only half finished. The University of Edinburgh's Artificial Intelligence department will be training some Neural Networks with the data during the summer. The results will be compared with some PC-based Neural Network models and the results of traditional modelling techniques in a further paper to be presented at the conference.

Contents

Section

2. Neural Networks

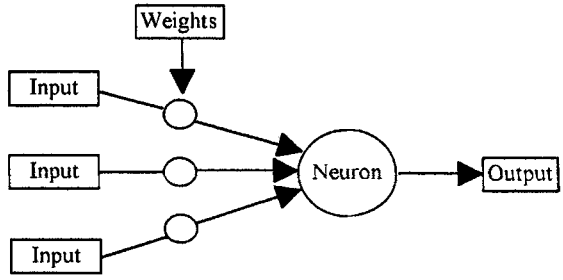
2.1 What are they?

Neural Networks ("NNs") are computer systems that are often said to operate in a similar fashion to the human brain (or most other brains come to that). A NN consists of a series of units called "neurons". These are usually simple processing units which take one or more inputs and produce an output. Each input to a neuron has an associated "weight" which modifies the strength of the input. When a NN is trained, these weights are adjusted to bring the output as close as possible to that desired.

Early NNs typically consisted of n inputs x_1, x_2, x_3, ..., x_n, each x_i being either 0 or 1. Each x_i is multiplied by a weight w_i, and the NN would output either a 0 or a 1 depending on whether the sum, S, below, was more or less than a certain amount:

$$S = \sum_{i=1}^{n} w_i \cdot x_i$$

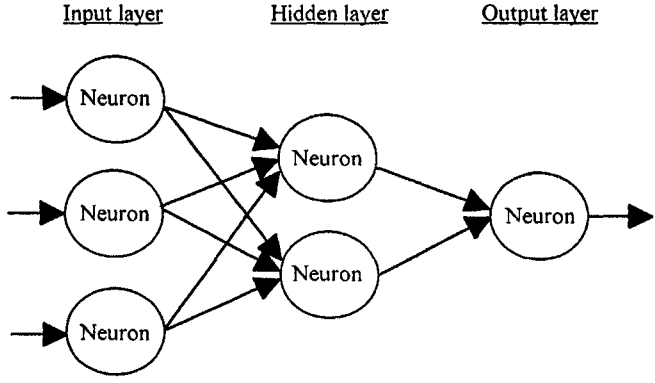
Neurons whose output depends on some function exceeding a certain amount are known as thresholding output neurons. A simple representation of this basic NN is given below:



Although the example above is very simple, more complicated NNs are just variations on this basic theme. The variations stem from the way the component parts are connected (the NN topology), how the calculations of each neuron are translated to a suitable output (the transfer, or activation, function) and how the weights are derived ("training" the NN). These different aspects of NNs are considered in the following sections.

2.2 NN topologies

The most common structure for a NN consists of an Input layer, one or more intermediate "Hidden" layers and an Output layer. The inputs of a given neuron are fed from the outputs of neurons in the previous layer. Information flows from the Input layer, through the Hidden layer(s) and finally out through the Output layer. This is known as a Feed-forward network. Because there is no feedback, the NN produces a result in a single operation and is stable - that is it happily arrives at a single value given a certain set of inputs.



As one might guess, there are many other possible structures for NNs. These can include connections back to previous layers or even back to the neuron itself. NNs for which inputs to a given neuron are taken from outputs from its own or subsequent layers are called, not unnaturally, Feedback networks.

A network which has neurons that compete with each other, so that only one neuron responds to a particular input pattern, is known as a Competitive network. Simple systems of thresholding output neurons, such as those described in 2.1, are called perceptrons, a term coined by Rosenblatt[1].

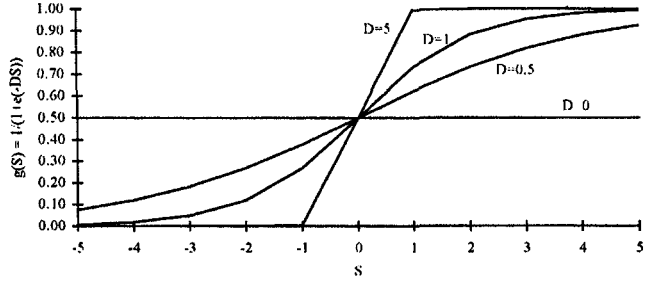
2.3 NN transfer functions

The transfer, or activation, function is applied to the weighted sum of the inputs of a neuron to translate the inputs to an output. Good candidates for transfer functions are bounded, monotonic, continuous and differentiable everywhere. A commonly used function is the sigmoid function, g(S):

$$g(S) = \frac{1}{1 + e^{-DS}}$$

As D gets larger, g(S) becomes more and more like a simple step function - that is a function that just switches between two levels. For the hard of thought, a quick summary of different sigmoid functions for different levels of D is given below:

Comparison of Sigmoid functions



In practice, different values of D simply lead to the weights being rescaled, so D is usually taken to be 1.

Why bother with this transfer function, when we could just have the threshold function, giving an output if S was above a certain level? Well, Minsky & Papert[2] proved that perceptrons, the systems of thresholding output neurons, could never represent the exclusive-or function. That is, a function that outputs "0" from two "0" inputs or two "1" inputs, and "1" from a combination of "0" and "1" inputs. They also showed that perceptrons couldn't represent a variety of other functions. So, different types of transfer function are needed so that NNs can represent certain types of function.

2.4 Training NNs

The essence of NNs is to "learn" the weights that best replicate the desired output given a series of inputs. The most commonly used way of deriving the weights is known as "back-propagation", or "the generalized delta rule", versions of which were first described by Bryson & Ho[3] in 1969 and Werbos[4] in 1974. These techniques were then popularized by Rumelhart, Hinton & Williams[5] in 1986.

For given input values, the NN looks at the difference between the calculated output and the desired output, and then, starting with the Output layer and working back to the Input layer, adjusts the weights according to their contribution to this difference. Usually all the weights are adjusted together - that is the changes to all the weights are calculated and then implemented simultaneously, rather than changing each weight one at a time.

In a similar fashion to many types of optimization, the weights are adjusted by the derivative of the "error" (the difference between desired and calculated outputs) with respect to the weight multiplied by a constant:

$$\text{Error, } E = \frac{1}{2} \cdot \sum_{p} \left(d_p - g_p(S) \right)^2 ,$$

where d_p is the desired output, $g_p(S)$ is the fitted output, p is the number of training patterns being run through the NN Then:

$$w_{ij} \rightarrow w_{ij} - \eta \cdot \sum_{p} \frac{\partial E}{\partial w_{ij}},$$

where η is an "arbitrary" constant, that needs to be chosen empirically. η might typically be chosen to be between 0.1/p and 10/p, for inputs and outputs of the order of 1.

A "momentum" term can be used to smooth the changes in the weights. This can be done by adding back part of the previous change in a given weight, so jumps from one weight to another do not fluctuate too much.

As with other optimization techniques, there is the potential problem that the algorithm might reach a local minimum but not a global minimum. Attempts to correct this can be made by "jiggling" the optimum weights, and seeing if the algorithm finds its way back to the same spot. There are lots of other optimization techniques that can be and are applied.

2.5 So what?

NNs sound quite interesting, but what can they be used to represent? Quite a lot, as has been demonstrated by the following proofs:

(i) Any continuous function can be represented by a NN with one Hidden layer and sigmoid transfer functions to any degree of accuracy (Cybenko[6], Funahashi[7]).

(ii) Any discontinuous function can be represented by a NN with at most two Hidden layers to any degree of accuracy (Cybenko[8]).

(iii) Any Boolean (logical) function can be represented by a NN with one Hidden layer with a sigmoid transfer function and an Output layer with thresholding output neurons (Hertz, Krogh & Palmer[9]). So, the non-linear transfer function in the Hidden layer overcomes the problem of the exclusive-or function described in 2.3.

Suddenly, NNs sound rather more appealing. However, the proofs do not tell you whether it is possible for the NN to learn the weights, and they require that the number of neurons in the Hidden layer(s) tends to infinity as the degree of accuracy tends to zero.

2.6 Using NNs in practice

There are a variety of software packages for PCs and mainframes that can be used to run NNs. The DTI have launched a NN information pack[10] which includes a list of 38 companies that provide NN services and 27 different software packages. Most of the software packages are designed to run on a mainframe, and quite a few of them are designed for specific applications, such as gas chromatography recognition or muzzle velocity prediction (!!). These are probably not terribly useful for general insurance applications. Appendix I lists some currently available packages that can be used on a PC running Windows via various spreadsheet packages.

The Working Party has been analysing some real general insurance data, kindly donated by a mystery UK composite company. The data consists of 75,000 individual claim records. The aim of the modelling has been to arrive at a rating structure. This has been done using a program called NeuralWare on a Unix workstation, a program called Braincel on a PC and, for comparison, GLIM using a PC. Details of how the NN modelling was performed and how the results compare to analysing the data using "traditional" statistical techniques will be presented at the conference in October.

2.7 Using NNs in general insurance

NNs can be used in any context in which one wants to represent a set of data by a model for making some sort of prediction. One possible application is in underwriting. Here one has many characteristics of a set of risks as a series of inputs, and one wants to know as an output what the propensity to claim is, or the likely level of claim. This can be used as a model to suggest accepting or rejecting a risk, much as is done in a banking context when considering accepting or rejecting a loan. Alternatively, one can try and output an indication of a suitable rate for a risk.

NNs have been widely used in credit-rating. Although there are obvious similarities with premium rating, which might suggest that it would be relatively easy to adopt existing NN credit-rating techniques, there are some fundamental differences that complicate matters. Banks usually charge everyone in a given broad class the same rate of interest, and so are primarily interested in trying to pinpoint "bad" risks. Insurers do not want to just reject 5% of all risks and charge a blanket rate for the rest, they want to arrive at a rate reflecting the riskiness of the policy. Recasting existing banking-type NN applications into an insurance context is not therefore as easy as one might think.

The required gradation of risk in insurance rating applications, rather than an outright acceptance or rejection, makes the formulation of a rating model for NNs potentially tricky. For a NN to "learn" whether a risk is a certain level of risk (a £300, £400 or £500 risk premium), ideally the NN needs to have as training outputs what the actual level of risk is. This means it can't necessarily just be presented with a series of individual policy details, with either a £5,000 claim or no claim at all, since that does not provide any indication of the level of risk associated with that policy with which to train the NN.

NNs can be trained to estimate severities, as we have a history of the actual claim sizes to train the NN with. At an individual policy level, however, one cannot easily teach a NN to predict frequencies unless one knows what the theoretical frequency for each risk is to train the NN with. The Working Party has been pondering the best way to present such problems to train a NN and we will present our thoughts on how best to achieve this at the October conference.

Reserving is another potential NN application. One could envisage a NN system to assess the likely claims cost for claims below some sensible maximum, based on a variety of different claim characteristics. One could then check such a system by sampling a certain number of the claims at a given point in time, and monitoring the overall accuracy of the system over time. This would just be a replacement for existing types of "formula" reserving methods for claims below a certain amount.

Like any organization, insurers want to forecast the state of the world in which they operate: competitors' pricing levels or movements in asset classes for example. Any statistical modelling done currently should be capable of being modelled using NNs.

3. Generalized Linear Models

3.1 What are they?

Generalized linear models are just an extension of classical linear models. Most people have at least a general awareness that linear models are of the form:

$$y_i = \sum_1^n \alpha_j \cdot x_{ij}$$, give or take some arbitrary constants.

For example, when fitting a line y = a.x. If one were given lots of y's and x's and asked to fit a line through them, most people would do so by least squares regression, which minimizes the expression:

$$D = \sum (y - \mu)^2$$, where μ is the fitted value of y for a given x.

In fact, this minimization is equivalent to assuming that the y's are realizations of a vector of independently distributed Normal random variables with means u and constant variance. The fitted μ's for a given x are just "a.x" for the fitted a. To formalize this structure, one can define classical linear models in three parts:

(i) A random component. A vector y has an independent Normal distribution with constant variance σ^2 and $E(y) = \mu$.

(ii) A systematic component. The covariates (the x's in the simple example above), x_j, produce a "linear predictor", η, given by:

$$\eta = \sum_1^n \alpha_j \cdot x_j$$

(iii). The systematic component, (ii), is linked to the random component (i):

$$\mu = \eta$$

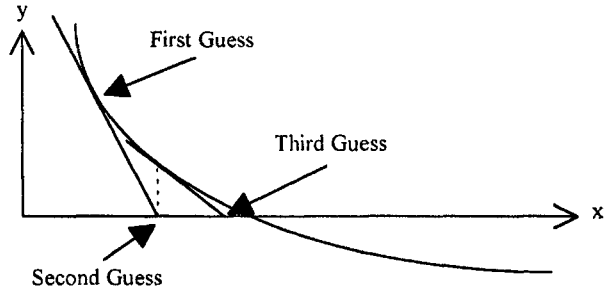
This relationship is called a Link function, $\eta = g(\mu)$. In this case the Link function, g() is just "identity", so the μ's are equal to the linear predictor.

All GLMs do is to extend this definition. The first generalization is to allow the random component to come from the exponential family of distributions. This includes the Normal, Poisson, Binomial, Gamma and Inverse Gamma distributions. The second generalization is to allow the Link function to be any monotonic differentiable function. This means that the μ's can be some function of the linear predictor, rather than just equal to it. For example the Link function may be a log or a power function, which would mean:

$$u = \exp(\sum_{1}^{n} \alpha_j \cdot x_j), \text{ or } \left((\sum_{1}^{n} \alpha_j \cdot x_j)\right)^{-2} \text{ and so on.}$$

The traditional measure of discrepancy, D, described earlier, is now no longer appropriate for models that do not have an identity Link function and a Normal random component. This is because each points' contribution to the fit of the model is no longer just defined as the difference between actual and fitted, squared. Instead, measures of discrepancy are formed from the logarithm of a ratio of likelihoods, called "deviance".

The parameters of the model, α, in the linear predictor can be estimated by an iterative process. This can be done in a similar way to the Newton-Raphson technique. This is used to find, iteratively, the roots of an equation, that is, where the function crosses the x-axis. This is done by making an initial estimate, then looking at the gradient of the curve at that point, and drawing a line at a tangent to the curve until that lines touches the x-axis and starting again:



Products such as GLIM perform a similar type of iteration to solve for the parameters of a GLM, stopping when changes to the estimates are sufficiently small.

3.2 Using GLMs in practice

The GLM modelling approach applied to the data is outlined in the 1992 Wright & Brockman paper[11]. In summary, a "traditional" statistical model was fitted to the data using the software package GLIM. This involved fitting multiplicative models for frequency and severity, assuming Poisson and Gamma error structures respectively, for two different types of claim within the data. Parameters were fitted for each of the levels within a given rating factor, and then a variety of statistical tests used to consider eliminating parameters that did not appear significant, grouping parameters that did not appear significantly different from each other and reducing the number of parameters where this achieved a better balance between the number of parameters and the fit of the model.

It is not the intention of this paper to dwell at any length on the niceties of using GLIM to model general insurance data for rating purposes. For those that are interested however, the GLIM code used to perform the modelling is given in Appendix II.

We will however comment on one small aspect of the traditional modelling approach that does not seem to have received too much attention in previous actuarial papers on the subject of GLMs. This is the use of residuals as diagnostic tools in assessing such models. In the discussion following the Wright & Brockman[11] paper, reference was made to the use of deviance residuals rather than traditional residuals for considering the fit of a model. This is a complete diversion from the subject of NNs but we thought it might be of use to people using GLMs in practice.

As mentioned in 3.1, if we are using non-Normal error structures, we cannot compare models by measures of standard deviation alone. Instead, a measure of "deviance" is used, whereby each data points' contribution to the overall fit of the model is determined by the error structure adopted. It may not be immediately clear to the reader of the Wright & Brockman[11] paper exactly what each points contribution to the "deviance" is, so we thought it would be useful to define them below. They are:

Poisson: $2.(y.\log (y/\mu) - (y - \mu))$ multiplied by the weight (exposure)
Gamma: $2.(- \log(y/\mu) + (y - \mu)/\mu)$ multiplied by the weight (no. of claims)

Where y is the actual data, μ is the fitted data. A useful check if one wants to calculate and plot deviance residuals using GLIM, is to check that the sum of all the deviances calculated as defined above adds up to the figure for deviance output by GLIM. GLIM may output the scaled deviance, which is just the actual deviance divided by the degrees of freedom of the model (number of data points less the number of parameters)

To look at deviance residuals, the deviances are simply plotted against fitted values. The most sensible scale for plotting the fitted values is $\log(\mu)$ for a Gamma error structure and $\mu^{1/2}$ for Poisson. This should give the line of constant y as a 45% slope on the residual plot which is a pleasant property for a residual plot to have.

If we just looked at normal residual plots, we would expect them to be skew when modelling frequencies and severities. The point of any residual plot is to look for evidence of non-randomness, which is obviously made tricky if the residuals are bunched near the axis with a few scattered further away in a distinctly non-random fashion. Plots of deviance residuals however should be broadly symmetric and hence easier to interpret.

Having said that, when modelling claim frequencies, the nature of the model is such that one still gets a systematic non-random effect. This is because one is modelling a discrete phenomenon with a continuous model. That is, there are lots of risks with no claims, a 0% actual frequency, and some risks with an actual claim. There is not a smooth gradation of frequencies from 0 to 0.5, say, instead there is a jump in frequency from nothing to something. The risks with an actual frequency of zero will have deviances that are just equal to the expected number of claims for the exposures in a given data cell. A deviance residual plot at first sight will therefore be a bit disconcerting, as, depending on the level one has grouped the data, for all the cells with zero actual claim frequencies there will be a concentration of small positive residuals at the bottom of the plot. Anyone trying to interpret deviance residuals should not be put off or worried by this effect - it is just a function of the nature of the model.

4. A comparison of NNs and GLMs

4.1 Why use NNs?

Part of the rationale for looking at NNs is the observation that the human brain achieves immense computing power by operating lots of neurons quite slowly but in parallel. Human brains consist of a series of neurons, about 10^{10} of them. Each one receives signals from between 10 and 10,000 other neurons. A single neuron takes between 0.01 and 0.001 of a second to compute - pretty slow compared to even a PC let alone a mainframe. Faced with a barrage of complex stimuli however, the human brain can work out in about 0.1 of a second whether the thing leaping at him around the corner is his wife or a tiger, and take the appropriate action (shoot the tiger but not the wife). This only allows between 10 and 100 or so connections between neurons, yet the structure of the interactions between neurons allows this incredibly fast sifting of information.

The allure of NNs is to try and mimic this structure and emulate the power of the brain's massively parallel computing system with computers that, whilst they can't have as many as 10^{10} transistors each connecting with 10,000 others, they can make up for this by passing signals between each other about 1,000,000 times faster than the brain. Having said that, one needs to be wary of drawing too strong a parallel between NNs and the human brain, so as not to raise unrealistic expectations. Obviously there is more to the human brain than lots of little transistors passing 0's and 1's between each other and in practice lots of realizations of NNs are implemented in serial rather than in parallel, with hundreds of neurons rather than millions of them.

NNs are of interest from two viewpoints. Firstly from a "biological" point of view, in trying to understand how the brain works. Secondly from a mathematical / engineering / physicist's point of view, in trying to take some tips from nature in realizing efficient ways of recognizing patterns in data, which is effectively what any statistical modelling is trying to do.

The most important reason for using NNs is that they work. They have been successfully used to recognize patterns in data in a variety of applications including credit rating, fraud detection and investment management. They are especially useful in areas in which there is no clear theory of how the patterns arise: rather than depending on a model, they are driven solely by the data on which they are trained.

4.2 Why use GLMs?

Why do we use any model? Consider for example a set of data on, say, motor claims. This may consist of tens of thousands of policies, each one with several dozen different attributes (age of car, location and so on), and details of the premiums received and claims paid for each one. If simply represented as tens of thousands of numbers on reams of paper, the human mind simply cannot grasp the essential characteristics of the data, or discern any pattern, let alone use the data to make sensible predictions. To understand the data in any meaningful way requires the formulation of a pattern that in some way represents the data, and allows the important characteristics to be represented by a limited number of terms that can be easily understood.

Further, when considering any set of data, there will be some systematic influences affecting the claims experience, such as the inflation in the cost of car repairs say, and random influences, such as the frequency of claims. To understand the data effectively, one needs to differentiate between the systematic influences and the random variations. It is the combination of reducing complexity and separating systematic influences from random variations that leads to a statistical model. A further aspect of using any statistical model is that one can test the assumptions underlying the model, allowing the modeller to gain a greater understanding of the characteristics of the data.

GLMs form a very broad category of statistical models, and there are a number of commercially available software packages available to implement them: GLIM, SAS and S-Plus amongst others. They therefore form an obvious way to perform many types of statistical modelling and are widely used in general insurance reserving and rating applications.

4.3 How do NNs and GLMs compare?

They are similar in many ways. They both try to discern some pattern in a set of data. They both work essentially by iteration, refitting "parameters" of the model until some optimum fit is reached. The end result in both cases is a model that one can then use to make predictions about future events. One of the main differences is that GLMs start with a model, and then see how good a fit the model is. NNs effectively finish with a model, being largely driven by the data on which they are trained.

GLMs are much more transparent than NNs in their operation. The lack of transparency is both a strength and a weakness of NNs. It is a strength in that one does not have to specify in too much detail the topology of the NN or the details of the transfer functions and learning methods. Some problems defy easy definition - a program that recognizes faces or fingerprints for example Others have so many possible parameters that traditional models are overwhelmed, a model for the state of the economy for example. On the other hand, whilst the lack of a model makes NNs potentially easy to apply, it also limits the usefulness of the output and the confidence one may have in it, since one is not always sure how the answers were arrived at!

GLMs limit you to certain categories of models whereas theoretically the range of functions that can be represented by NNs is unlimited. It is certainly true that lots of real world phenomena are distinctly non-linear and so not strictly amenable to modelling by GLMs. However, a great many such phenomena can be broken down and approximated by linear models within the GLM family. No such statistical models are "right", they are hopefully just useful representations of reality.

In terms of computer time, NNs are likely to take longer to fit than GLMs, as the fitting algorithms are usually more general and less refined than commercially available GLM packages. In terms of human time however, GLMs are likely to take rather more time by way of definition, programming and subsequent analysis.

There are of course more types of statistical models than just GLMs and there are a great many non-linear modelling techniques. It is hard to see, other than ease of application, in what sense NNs can be said to be superior to GLMs or more general non-linear statistical models. They do, however, have a certain appeal and with increasingly powerful PCs may sometimes provide a mechanism of substituting computing power for brain power by avoiding the need to come up with a sophisticated model.

4.4 Comparison of results

The data analysed consisted of approximately 75,000 individual claims records. Each record had 8 rating factor fields (age of policyholder, location and so on) with between two and around one hundred "levels", details of the period of exposure of each claim (one month or twelve months for example) and a record of the number and amount of any claims. Allowing for part-years of exposure, the data represented about 50,000 policy years in total. The total number of claims was approximately 10,000, the average claim size being 500 "units", the maximum being about 150,000 units.

The type of GLM fitted is described briefly in 3.2. The final model is of the form:

Rate = Base level x F_1 x F_2 x F_3 x F_4 x F_5 x F_6 x F_7 x F_8, where the F_i factors are:

	Levels within each rating factor									
	1	2	3	4	5	6	7	8	9	10
Factor 1	1.00	0.80	0.74	0.65	0.65	0.65	0.57	0.53	0.47	0.43
Factor 2	1.00	1.14								
Factor 3	1.00	1.13								
Factor 4	1.00	1.29	1.54	1.83	2.07					
Factor 5	1.00	0.82	0.78	0.73	0.70	0.60	0.55			
Factor 6	1.00	0.95	0.89	0.82						
Factor 7	1.00	1.11	1.23							
Factor 8	1.00	0.75	0.61	0.50	0.41	0.29	0.20			

So, for example, a risk in level 1 of the first seven rating factors, but level 2 of the eighth rating factor would have a risk premium of Base level x 0.75, and so on.

Can NNs come up with a "better" or more useful representation of the data, either by way of the level of risk premium for each claim or an indication of risks that are particularly likely to claim? Hopefully we will have a better idea when we describe the work done to train NNs at the conference in October. We will also be scratching our heads trying to come up with a meaningful way of comparing the competing models.

5. Bibliography

Texts referred to in the paper

(1) "The Perceptron - a perceiving and recognizing automaton", Report 85-460-1 Cornell Aeronautical Laboratory, Rosenblatt F. (1962)

(2) "Perceptrons. Expanded edition", MIT Press, Minsky M.L. & Papert S.A. (1969) (reprinted 1988)

(3) "Applied Optimal Control", Blaisdell New York, Bryson A.E. & Ho Y.C. (1969)

(4) "Beyond regression: new tools for prediction and analysis in behavioural science", Ph.D. thesis Harvard University, Werbos P. (1974)

(5) "Learning representations by back-propagating errors", Nature Vol. 323 pp. 533-6, Rumelhart D.E., Hinton G.E. & Williams R.J. (1986)

(6) "Approximation by superpositions of a sigmoidal function", Mathematical Control Systems Signals Vol. 2 pp. 303-14, Cybenko G. (1989)

(7) "On the approximate realization of continuous mappings by neural networks", Neural Networks Vol. 2 pp183-92, Funahashi K. (1989)

(8) "Continuous valued neural networks with two layers are sufficient", Dept. of Science Tufts University, Cybenko G. (1988)

(9) "Introduction to the theory of neural computation", Addison-Wesley, Hertz J., Krogh A. & Palmer R.G. (1991)

(10) "Neural computing: learning solutions", D.T.I., (1993)

(11) "Statistical motor rating: making effective use of your data", JIA Vol. 119 pp. 457-543, Wright T.S. & Brockman M.J. (1992)

Other publications of interest

"Networks and chaos - statistical and probabilistic aspects", Chapman & Hall - Monographs on statistics and applied probability, Barndorff-Nielsen O.E., Jensen J.L. & Kendall W.S. (1993)

"Theory of the backpropagation neural network", International Joint Conference on Neural Networks Vol.1 pp 593-611, Hecht-Nielsen R. (1989)

"Generalized Linear Models", Chapman & Hall - Monographs on statistics and applied probability, McCullagh P. & Nelder J.A. (1983)

Appendix I

NN software

The following are all PC based NN applications which are add-ons for, or interface with, spreadsheets. The approximate prices are correct to the best of our knowledge and are simply intended as a guide to anyone interested in buying such a package.

Package	Supplier	Approx. cost
4Thought	Right Information Systems Ltd	£5,000 +
Braincel	Analytical Power Tools, Palisade	< £500
IBM NN Utility	IBM UK Ltd	n/a
NeuDesk	Neural Computer Sciences	< £500
NeuralWorks	Scientific Computers Ltd	< £500
NNetSheet-C	Expert Systems Ltd	£500 - £5,000
NNetSheet-1.2	Expert Systems Ltd	< £500
Neuralyst	Analytical Power Tools, Palisade	< £500

Appendix II

GLIM code for fitting a rating model

```
\tra o\!limits info sent to GLIM.LOG
\units 2560\!number of values in each data vector

\data c d p a g n\! labels for data
\dinput 10 100\!assign a channel number for data, with max width

\fac d 8 \!declare d, p, a, g as different levels
\fac p 8 \!declare d, p, a, g as different levels
\fac a 5 \!declare d, p, a, g as different levels
\fac g 8 \!declare d, p, a, g as different levels

\yvar c\
\error G\! G defines a Gamma error structure, P a Poisson
\link L\! defines a Log link function
\weight n\!declare weights
\cyc 30 %prt 1.e-6 1.e-8\!specify settings for iter  n control
\acc 10\!set max accuracy for output
\cal %lp=%log(%if(%yv<0,0.5,%yv+0.5))\!initialise linear predictor
\fit  d + p + a + g \disp l e\print ::\

\stop\!end batch mode and exit GLIM
```

"c" and "n" are the amounts and number of claims, "d", "p", "a" and "g" are just rating factors, each one being at a certain level (8 levels for all except "a" with 5).

The sample code above fits claims severities ("c") with a Gamma error structure and a Log Link function, weighted by the number of claims ("n") in each cell.