Masters thesis of John Folkesson,
d98-jfo@nada.kth.se,
601025-8637
Advisor Örjan Ekeberg, SANS
Department of numerical analysis and computer science,
Royal Institute of Technology, Sweden

(Numerisk analys och datalogi
Kungliga Tekniska högskolan
100 44 Stockholm)

Abstract

In this work we have examined an application from the insurance industry. We first reformulate it into a problem of projecting a markov process. We then develop a method of carrying out the projection many steps into the future by using a combination of neural networks trained using a maximum entropy principle. This methodology improves on current industry standard solution in four key areas: variance, bias, confidence level estimation, and the use of inhomogeneous data.

The neural network aspects of the methodology include the use of a generalization error estimate that does not rely on a validation set. We also develop our own approximation to the hessian matrix, which seems to be significantly better than assuming it to be diagonal and much faster than calculating it exactly. This hessian is used in the network pruning algorithm. The parameters of a conditional probability distribution were generated by a neural network, which was trained to maximize the log-likelihood plus a regularization term.

In preparing the data for training the neural networks we have devised a scheme to decorrelate input dimensions completely, even non-linear correlations, which should be of general interest in its own right.

The results we found indicate that the bias inherent in the current industry-standard projection technique is very significant. This work may be the only accurate measurement made of this important source of error.

Projektion av en Markovprocess med Neurala nät

Sammanfattning

Jag har undersökt en tillämpning från försäkrings industrin. Först har jag omformulaerat den som en markovprocess. Sedan har jag utvecklat en metod för projektion av processen I flera steg med en kombination av neurala nät. Denna metod innebär en förbättring av den inom industrin vanliga använda metoder inom fyra områden: varians, bias, uppskattning av säkerhetsmarginalen, och möjligheten att använda inhomogena data.

De dataologiska aspekterna av metoden omfattar en feluppskattning som inte behöver en valideringsmängd. Jag har också använt ett sätt att skära bort grenar från nätverken.

Resultatet har visat att biasen i de standardmetoder som finns är stor. Denna undersökning är kanske den enda uppskattning som gjörts av denna viktiga felkälla.

Preface

I have been working as a consultant doing actuarial work, mostly with workers' compensation, for over ten years. My background before that was in electrical engineering and physics. After learning about the advances in neural networks through my studies at KTH, I began to think about applying them to the actuarial problems that I am so familiar with. I believe that the actuarial profession has not quite managed to keep up with the rapid changes to what can and can not be calculated. Modern computer methods and hardware have indeed made much progress in the last 20 years and these gains need to be applied. I hope this work will help move things in that direction.

I would like to thank my wonderful wife and four children for their patience and support while I carried out this project.

John Folkesson,
January 2001

Table of Contents:

Overview

The problem addressed in this work is that of predicting the outcome of a markov random process. The application is from the insurance industry. The problem is to predict the growth in individual workers' compensation claims over time. We describe the nature of these claims in the next section.

We have data on the past outcomes of the process for many claims over one-year intervals. From that data one would like to be able to project the process forwards many years.

Neural networks have been shown to do well at predicting the outcome of random processes. This prediction can be formulated as a function approximation where the process is modeled as a deterministic function of the input state plus a random variable having zero mean and a distribution about the mean that is conditionally dependent on the input state.

This deterministic function can be estimated with a neural network. If one has many examples of input-output state pairs, one can train a neural network to minimize the square error between the predicted and the actual output state. Neural networks are, in fact, able to approximate large classes of functions being so-called universal approximators. What this means is that given an arbitrarily large net one can come arbitrarily close to any function within certain classes (Scarselli 1998).

If one wants to project the process further out in time, more than one step that is, then one approach is to feed the one step projection back into the network's input to produce a two step projection. Unfortunately this simple recursive approach may be too much of an oversimplification of the process. In particular, if the process variance is large and the average input-output mapping is non-linear, then the two step estimate will contain a bias. This bias will become greater as the three-step estimate is made and so on. If one needs to go out many steps, a significant amount of bias could result.

One can not change the input-output mapping to make it linear but one can reduce the variance in the output states by breaking the state space up into regions. The variance in the output given that it is within one of the regions will be less than the variance without any restriction. In effect, we propose to form a discreet approximation to the continuous space of intermediate states.

Clearly, some estimate of the conditional probability is required to carry out this reduction in the variance and thus reduce the bias. Fortunately, neural networks are also practical tools for estimating conditional probability distributions (Husmeier 1998). Here the approach normally taken is to use a network to maximize the likelihood of the training data. The conditional probability distribution is first parameterized in some way. Then the parameters are estimated by a neural network. The network is then trained to maximize the likelihood of the training data.

This maximum likelihood approach is elegant and simple to implement. We have implemented it using the weibull distribution rather than the more conventional guassian distribution. This choice was made with consideration to the data.

Given the conditional probability distribution, we are able to break the output space up into regions with a mean and a probability for each. These means are used as inputs to the next projection step. The means for each region are feed back into the network to generate two step distributions. The various distributions obtained from projecting from each of the regions' means are combined into one using the probabilities for each region and then the output state space is broken up once again into regions for the next projection step and so on.

Several side issues come up in along the way, some of which are of interest in their own right. For instance, networks learn best from decorrelated inputs. Recently, the input and internal dimensions of a feedforward net were decorrelated using

additional layers of linear neurons trained using Sanger's learning rule (Schittenkopf 1997). They were not only able to improve the learning ability of the net, but were able to control the flow of real information through the net, thereby improving generalization.

In our case, we had even more reason to avoid correlation as we have made an approximation to the hessian matrix of the network weights that assumes that the all the nodes are decorrelated. This approximation was a great time saver when pruning the various networks.

For these reasons, we have devised a scheme whereby even non-linearly correlated inputs can be completely decorrelated, with all mutual information removed. The scheme involves training a series of neural networks to learn the conditional probabilities describing the correlations.

Another issue that often comes up in the context of neural networks is overfitting the data. One way to avoid overfitting and maintain good generalization ability is to keep the number of network parameters small relative to the amount of training data. In other cases, one needs a measure of the generalization error. Usually one simply tests the network on data that it hasn't been trained with and takes the test error as the generalization error.

Armed with a measure of the generalization error one can then decide two important questions. First, when to stop training a net. (See (Prechelt 1998) for a discussion of so called early stopping criteria.) And second, deciding which architecture of networks is best. In particular, the network with the smallest training error isn't always the best.

A new and interesting measure, the network information criterion, has been proposed (Murata 1994) and tested (Hintz-Madsen 1998) based on the hessian matrix of the regularized cost function of the network over the training data. This eliminates the need for validation data and allows more data to be used for training. We have tested this approach and found it to be advantageous.

To optimize the networks we have used the optimal brain surgery method (Hassibi 1993), which seem to be, out of all the types of pruning criteria, the one that prunes the right weights the most often. We have approximated the hessian as the sum of a block diagonal matrix and an outer product of the average gradient vector. This was neccasary as the hessian is very time consuming to calculate exactly.

Background Information

The Application

The data consists of workers' compensation insurance claims evaluated at 12 month intervals. These are workers who were injured on the job and are entitled to legislated benefits from the employer. A claims examiner estimates the ultimate amount of the claim based on the details of the injury. These estimates are included in the data along with other information. These claims tend to grow over time as the injured worker fails to return to work, becomes sicker etc. The task is to estimate this growth. Normally, on average, the claim examiner's estimate of the outstanding amount on the claim, (the *reserve*), is low by around 50-150% during the time the claim is open. One can also note that the distribution of the outstanding amount on a given claim has a standard deviation that is between 1 and 2 times its mean.

To help with estimating the outstanding losses, there are many samples of the process over one-year intervals going back three years. Care must be taken to not go too far back in time as the process is only approximately stationary. In other words the development ten years ago is a poor indication of the development today. On the other hand one would like to use as much data as possible.

The present standard approach to this problem only utilizes some of the available information on the claims. By using a neural net one should be able to reduce the uncertainty in the estimate (the process

variance). This is because much of what is taken random variation in the standard approach is, in fact, variation along unused dimensions in the input state space.

Another related issue is that the data are rather inhomogeneous, coming from many different sources. By coding the source of the data intelligently, one can add dimensions to the state space for the source. Thus, one can take advantage of the similarities in the data from the different sources while still allowing the net to discriminate between them. This will allow for a disciplined method of dealing with a mixture of data from different sources.

Another important problem with the standard approach is the bias that was mentioned in the introduction. This is particularly annoying in that a bias does not go to zero for larger and larger insurers. The bias is hard to estimate, being positive for some claims and negative for others. Using an approach based on neural networks we were able to estimate this bias and remove it. There is no other practical method for doing this that we are aware of.

An additional aspect of the problem is that the variability of the outstanding claims in aggregate can be calculated if one has an estimate of the distribution of individual claims about the mean. Traditional methods consist of making rather arbitrary guesses at this distribution. The neural network approach will give additional information allowing a distribution to be fit for each open claim. This will allow theoretically justifiable confidence intervals to be calculated.

The distributions of the state variables one step into the future are strongly functions of the *reserve*. The other state variables have a much smaller influence on the development of the claim. This observation is critical in understanding the choices described later in the methods section.

In summary, the neural network approach has the potential to eliminate the bias in the mean, reduce the variance in the estimated mean, incorporate inhomogeneous data into the estimate and calculate confidence intervals about the mean. In short, the risks associated with this type of insurance might be able to be dramatically reduced.

The Markov Process

In order to solve the problem outlined in the previous section we need to reformulate it as a markov process.

The principle assumption we make is that the future development of the claim is a random process, which depends only on the current state vector. This is the markov assumption.

The state vector consists of the information we have on the claim as of some date, such as the *paid* to date the amount of the *reserve*, the *age* of the claim and so on. We will refer to state transitions as steps of the process.

As we project the process one step, some of the variables of the state vector, $\mathbf{x}$, will not change. Others such as the *age* will change by a known amount, one year. Some of the variables will change by a random amount according to a distribution that is conditionally dependent on the input vector, $\mathbf{x}$. Let $f(\mathbf{x}^{(out)} \,|\, \mathbf{x}^{(in)})$ be this conditional probability distribution.

If we project two steps we find the distribution will be given by,

$$f(\mathbf{x}^{(2)} \,|\, \mathbf{x}^{(0)}) = \int f(\mathbf{x}^{(2)} \,|\, \mathbf{x}^{(1)})\, f(\mathbf{x}^{(1)} \,|\, \mathbf{x}^{(0)}) d\mathbf{x}^{(1)}$$

Here the superscripts on $\mathbf{x}^{(i)}$ refer to the number of the step that proceeded state $\mathbf{x}^{(i)}$. Now, one would like to be able to replace the integral over a continuum intermediate states with a single projection. If one were to choose the mean, $\boldsymbol{\mu}^{(1)}$, of $\mathbf{x}^{(1)}$ then at least the first order correction terms will vanish,

$$\begin{aligned} f(\mathbf{x}^{(2)} \,|\, \mathbf{x}^{(0)}) &\cong \int \Big\{ f(\mathbf{x}^{(2)} \,|\, \boldsymbol{\mu}^{(1)}) \\ &\quad + (\mathbf{x}^{(1)} - \boldsymbol{\mu}^{(1)}) \nabla_{\boldsymbol{\mu}} f(\mathbf{x}^{(2)} \,|\, \boldsymbol{\mu}^{(1)}) \Big\} \; f(\mathbf{x}^{(1)} \,|\, \mathbf{x}^{(0)}) d\mathbf{x}^{(1)} \\ &= f(\mathbf{x}^{(2)} \,|\, \boldsymbol{\mu}^{(1)}) \int f(\mathbf{x}^{(1)} \,|\, \mathbf{x}^{(0)}) d\mathbf{x}^{(1)} \\ &= f(\mathbf{x}^{(2)} \,|\, \boldsymbol{\mu}^{(1)}) \end{aligned}$$

The second order terms however will not vanish. This is what leads to the bias of the recursive procedure. By the recursive procedure we mean using the mean from one projection to project the mean one more step into the future.

As one can show by taking the expansion above out to the next term, the bias is proportional to the variance in $\mathbf{x}^{(1)}$. By reducing this variance we can reduce the bias. This is the heart of our approach.

One can also see our approach as the approximation of the above integral by expanding around several points each one being the mean within some region of the $\mathbf{x}^{(1)}$ space. The difficult part is to generate the mean values and the probabilities for each region.

If one breaks up the state space into fixed regions, the same for all input points $\mathbf{x}^{(0)}$, then for every $\mathbf{x}^{(0)}$ there will be some output regions with very sparse training data. This can make the method unstable. We need to make the regions vary with $\mathbf{x}^{(0)}$ in order to avoid this unstable behavior.

The aim is to make a machine, $\mathbf{M}(\mathbf{x}^{(k-1)})$, that is given a partition, $\{A^{(k)}_i\}$, of the state space one step into the future. The machine can then generate a set of mean output states, $\mathbf{x}^{(k)}_i$, given that it is within each region $A^{(k)}_i$ and the probability, $p^{(k)}_i$, of that region.

$$\{(\mathbf{x}^{(k)}_i, p^{(k)}_i)\} = \mathbf{M}(\mathbf{x}^{(k-1)})$$

Our machine is composed of various networks which used together realize the **M** above. The regions are just slices of the state space along lines of constant *reserve*. As pointed out earlier, for our application the process is most strongly a function of the *reserve* so that it is sufficient to reduce the variance in just this one dimension.

We are furthermore able to first calculate the probability density of the output *reserve* using our maximum likelihood distributions. Then we can chose the partition. Then we can calculate the means within each region for the random dimensions of our state space.

The number of regions at the current step can be chosen with regard to the importance of the current step to the ultimate quantity of interest. That would be for us the total mean payments of a group of claims over several years.

Thus, if the mean payment for the step is relatively small then we need not split the space up into a large number of regions. Alternatively if the claim is a very large one we may want to be extra precise with it and would create many regions. In this way we can apply our processing power judiciously.

In our case we chose to use the same number of regions for all the claims and just let the program run a bit longer.

Training a Multi-layer Perceptron

A very interesting review of learning theory is given in (Vapnik 1999). The primary principle in all the learning algorithms is the, so called, 'empirical risk minimization induction principle.' What it refers to is the validity of replacing the expected value of the risk, (also called cost), function with its average over the training set. After this replacement one can begin to devise algorithms to minimize the empirical risk rather than the true expected risk.

Training a multilayer perceptron to minimize this empirical risk can be done in a number of ways. The easiest learning algorithm to understand is probably back propagation in batch mode. In that learning mode one simply calculates the gradient of the cost function summed over all the training data points. Then one moves the weights in the direction of the steepest decent. The change in weights is given by a constant times the gradient.

In particular, if we define a cost function that we want to minimize as:

$$D(\mathbf{w}) = \Sigma D(\mathbf{w}, j)$$

Where the sum over j refers to the training data points and **w** is the weight vector. Then we update the weights according to:

$$\delta\mathbf{w} = -\eta \Sigma \nabla_w D(\mathbf{w}, j)$$

Then one recalculates the gradient with the new weights. Here η is a small parameter that defines the

step size. The problem with this approach is that it is quite slow.

A better method is to update the weights after calculating the contribution of each data point to the gradient.

$$\delta \mathbf{w} = -\eta \nabla_w D(\mathbf{w}, j)$$

Thus one doesn't wait till all the training data has been presented giving a more continuous improvement. This learning mode tends to converge much faster than batch mode.

A third learning algorithm is the quickprop method. (Fahlman 1992) Here one calculates the gradient as in batch mode but then compares it to the previous gradient to obtain some second derivative information. One then assumes one is moving towards the bottom of a parabolic bowl. One can then easily calculate where the bottom of the bowl is and move the weights there. There are some other small details to the algorithm for special situations like avoiding taking too large a step, convex regions of the weight space and getting started, but the main idea is to fit to a parabola. The update rule is given by:

$$\delta \mathbf{w}_n = \delta \mathbf{w}_{n-1} \Sigma \nabla_w D_n(\mathbf{w}, j) / (\Sigma \nabla_w D_{n-1}(\mathbf{w}, j) - \Sigma \nabla_w D_n(\mathbf{w}, j))$$

Here $\delta \mathbf{w}_n$ is the change in the weight vector and D_n is the cost function, both at the nth iteration. We have chosen to use this update rule in this work. It has the significant advantage of not being very sensitive to the choice of parameters. There are some parameters, not shown above, for the special situations but the learning rate is not so sensitive to them. This allows us to "set and forget them".

Additional algorithms typically use some type of mechanism to adaptively adjust the step size and fall under the name adaptive backpropagation. A good description is given in (Parlos 1994). Among the many variations demonstrated there they found that the adaptive rule:

$$\delta \mathbf{w}_n = -\eta \Sigma \nabla_w D_n(\mathbf{w}, j) / || \Sigma \nabla_w D_n(\mathbf{w}, j) ||^2$$

often gives faster convergence. This can be understood by noticing that the relative change in D is approximately held constant by the clever choice of the learning rate function.

Finally if one has the time or a small enough network one can do a full Gauss-Newton method update. The rule then becomes:

$$\delta \mathbf{w} = -\eta \, (\Sigma \nabla_w \nabla_w D(\mathbf{w}, j))^{-1} \, (\Sigma \nabla_w D(\mathbf{w}, j))$$

Here $\nabla_w \nabla_w D$ is the hessian matrix of the cost function. A general trend is that the faster algorithms tend to be less stable. They can go off in the wrong direction only to later start making progress again.

All the methods depend on the calculation of the gradient of the cost function. This calculation is preformed by the backprop algorithm, (Haykin 1999), which involves first passing the input data through the net, the forward pass. Then passing the error backward through the net to calculate the partial derivatives with respect to each weight. Each pass has a complexity of the order of the number of weights. Passing all the training data points through the net is referred to as one epoch.

Heuristics for Better Convergence

There are several ways to help the network to learn. (Haykin 1999, p.178-184) They include the choice of learning algorithm, activation function, choice of training samples, having the target values that are within reach of the network and initializing the weights with small random values

Those are the good practice and one should be familiar with them. More complex is the issue of normalization and decorrelation of the inputs. Having the initial weights and the various inputs all with different characteristic scales leads to longer training times as the network must first learn all the scaling factors and then the output function. Best is if all inputs have a variance of order 1 and a mean of 0.

In addition, correlated inputs tend to learn slowly as the network gets the same information from two sources. Ideally, each input should contain no information that is available at another input.

(Schittenkopf 1997) has shown that one can effectively control the flow of information through a network by inserting extra principle component analysis (PCA) layers between the normal perceptron layers.

A PCA layer can separate the linearly independent components in order of decreasing variance from each layer's outputs. Thus helping not only with decorrelating the inputs but also the internal representations of the inputs. The approach is however limited to linear correlations.

For our problem, we have made use of an approximation to the hessian matrix of the neural weights that relies on the assumption that the network's nodes are more or less decorrelated. Since this assumption will certainly fail if the input nodes are correlated, it was even more important for us to decorrelate the inputs.

We found that some of the input dimensions were strongly correlated. What's more, the correlations were non-linear. To solve this problem we devised a scheme to decorrelate the inputs completely using a simple neural network to maximize the conditional probability of one of the inputs given the values of the others. We then mapped the input into the value of its conditional cumulative probability, which becomes our new input variable having a uniform distribution from -1 to 1 and independent of the other variables.

Specifically let z_i be the i^{th} transformed input, then:

$$z_i = 2\, F\left(x_i \mid z_0, \ldots z_{i-1} \right) - 1$$

Where F is the conditional cumulative probability function. We start with one input dimension and then decorrelating an additional dimension with regard to the first dimension. Once that is done we add another, decorrelated it with regard to the first two and so on.

This approach is quite general and could be useful as a standard method of data preprocessing.

Generalization

Networks have the important ability to generalize. That is to say after having been trained on a sample of input-output pair examples, the network will be able to interpolate an input-output mapping for points it has not seen.

The accuracy of the generalization is a function of the number of training points and the number of parameters in the model. Not surprisingly the accuracy improves with more training data. The optimal number of weights is more difficult to determine. More weights improve the model's ability to match a complex input-output relationship. However increasing the number of weights beyond a certain point leads to overfitting.

Overfitting occurs when the network begins to memorize the training samples. The danger is that the network will adjust the 'extra weights' to bring down the error for outlying points. This improves the training error while worsening the generalization ability.

As it is the generalization ability that one wants to optimize, one must have some criterion to judge it by. The error on a validation set is one way to judge the generalization ability.

The use of a validation set has some problems however. The most obvious is the loss of potential training data to the validation set. One in effect settles for a worse final network in order to be able to measure the generalization. In cases where the data is plentiful and noise-free this is not a problem.

The other problem is how large a validation set should be. Too small a set with very noisy data can be problematic. A few randomly chosen outlying points that happen to fall into one set or the other can throw off the whole procedure. That was our experience at any rate.

For that reason we finally decided to use another measure of the generalization ability, the network information criteria, NIC (Murata 1994).

The Network Information Criteria

What the NIC does is to estimate the expected value of the cost function over the unknown distribution from which the training data was drawn. This is similar to the familiar division by (N-1) in estimating of the variance of a set of numbers. The 1 being due to the one parameter of the model, (i.e. the mean).

The NIC estimate is made by expanding the expected value of cost function about the optimal solution, keeping the terms up to 2^{nd} order. Then some assumptions are made that lead to conclusions concerning the statistics of the learned weights. These conclusions can then be used to express the expanded cost function in terms of the empirical cost and its derivatives. By the empirical cost we mean the average of the cost function over the training set.

If the cost function is $D(\mathbf{w})$, the NIC can be written as:

$$NIC = \langle D(\mathbf{w}) \rangle + tr(Var[\nabla D(\mathbf{w})] \langle \nabla\nabla D(\mathbf{w}) \rangle^{-1}) / N$$

Where $\nabla\nabla D(\mathbf{w})$ is the hessian matrix and N is the number of training points. The derivatives are with respect to the weights and we make the abbreviation

$$\langle \ldots \rangle = \langle \ldots \rangle_{empirical} = (1/N) \Sigma(\ldots)$$

for the average over the training set. The variance of the gradient term is also taken over the training set. It is this variance that carries the information on the variance of the underlying distribution.

The most important assumption used in deriving the NIC is that the network is in the neighborhood of the optimal network for the underlying distribution. One can say that the better the model, the better the NIC approximates the generalization error. So one can conclude that the NIC will perform badly for networks that are poorly designed or trained. One simply can not expect too much from either the NIC or a neural network.

One other cautionary note in using the NIC is that it assumes that one is comparing networks from the same basic model. One can not for instance use the NIC to compare an example from a multi-layer perceptron model to an example from a radial base function model. This is due to a quantity, which was dropped from the definition of the NIC since it is approximately constant for a given basic model. The interested reader is referred to the appendix for more details.

Coming back to the formula for the NIC, one sees that the trace becomes a generalization of the number of parameters, a sort of effective number of parameters. We will refer to it as the effective dimension of the network. In one important case it is exactly the number of weights.

In particular, if the cost function is the log-likelihood type then we can derive some simple relations. Setting $S(\mathbf{w})$ be the log-likelihood cost function and $p(\mathbf{y} | \mathbf{x}, \mathbf{w})$ be the conditional probability of output $\mathbf{y}$ given input $\mathbf{x}$ produced by our model with $\mathbf{w}$ being the parameters of the model.

$$\langle S(\mathbf{w}) \rangle = \langle -\ln [\, p(\mathbf{y} | \mathbf{x}, \mathbf{w}) \,] \rangle$$

In the appendix we show that for a log-likelihood cost function,

$$\langle \nabla\nabla S(\mathbf{w}) \rangle = \langle \nabla S(\mathbf{w}) \nabla S(\mathbf{w})^T \rangle$$

Also if the network is trained to a local minimum, the expected value of the gradient vanishes so we find that the hessian is equal to the variance of the gradient. Thus for this special case

$$NIC = \langle S(\mathbf{w}) \rangle + tr(I) / N$$

$$= \langle S(\mathbf{w}) \rangle + (\text{number of weights}) / N,$$

For S(w) trained to its minimum.

If now we introduce a regularization term as in (Hintz-Madsen 1998),

$$\langle D_R(\mathbf{w}) \rangle = \langle S(\mathbf{w}) \rangle + (\mathbf{w}R\mathbf{w})/2$$

where R is the regularization matrix. The same reasoning as above leads us to

$$\langle \nabla\nabla D_R(\mathbf{w}) \rangle = \langle \nabla S(\mathbf{w}) \nabla S(\mathbf{w})^T \rangle + R$$

$$Var[\nabla D(\mathbf{w})] = \langle \nabla S(\mathbf{w}) \nabla S(\mathbf{w})^T \rangle + \langle \nabla S(\mathbf{w}) \rangle \langle \nabla S(\mathbf{w})^T \rangle$$

Unfortunately the expected gradient no longer vanishes when the cost function is minimized, being equal to -R**w**.

So we have for the NIC with regularization,

$$NIC = \langle S(\mathbf{w})\rangle + (\mathbf{w}R\mathbf{w})/2$$

$$+ \mathrm{tr}(\mathrm{Var}[\nabla S(\mathbf{w})]\,[\langle \nabla S(\mathbf{w}) \nabla S(\mathbf{w})^T\rangle + R]^{-1}) /N$$

At the minimum of $\langle D_R(w)\rangle$.

Thus, the R matrix serves the purpose of reducing the effective number of dimensions to a number less than the number of weights.

Now how do we relate the NIC of $\langle D_R(\mathbf{w})\rangle$ to compare $\langle S(\mathbf{w})\rangle$? This issue seems to be absent from (Hintz-Madsen 1998) although it is fairly important. The issue is that we have a criteria that allows us to estimate which set of w's gives the lowest true, as opposed to empirical, expected value of D(**w,** R). We also know that,

$$\langle S(\mathbf{w})\rangle_{\text{true}} = \langle D_R(\mathbf{w})\rangle_{\text{true}} - (1/2)\ \mathbf{w}R\mathbf{w}.$$

The NIC differs from $\langle D_R(\mathbf{w})\rangle_{\text{true}}$ by a small constant so that we can use the following measure to compare two networks where the ultimate goal is to minimize $\langle S(\mathbf{w})\rangle_{\text{true}}$.

$$E(R) = NIC - (1/2)\ \mathbf{w}R\mathbf{w}$$

$$E(R) = \langle S(\mathbf{w})\rangle$$

$$+ \mathrm{tr}(\mathrm{Var}[\nabla S(\mathbf{w})]\,[\langle \nabla S(\mathbf{w}) \nabla S(\mathbf{w})^T\rangle + R]^{-1}) /N$$

At the minimum of $\langle D_R(\mathbf{w})\rangle$.

This is a bit subtle. It would seem that making R huge and thus the effective number of dimensions 0, will always minimize E(R) with respect to R, but no. The NIC must be calculated near the minimum of $\langle D_R(\mathbf{w})\rangle_{\text{true}}$ in order to be valid. Thus one must first train the network to the minimum of D for a given R. Then one can calculated E. Changing R will change the **w**.

Now one begins to understand the mechanism at work here. Choosing a large R will force us to solutions with small weights. That will make the training error larger than with unconstrained weights. The benefit will be that the effective dimension is now less. For small networks the benefit of a large R will be small but for larger networks it will be significant.

We now have a way to compare networks with many small weights to ones with fewer but less constrained weights.

In (Hintz-Madsen 1998) the above expression for E(R) was not derived. Instead they compared nets using the regularized NIC with the added assumption that $\langle \nabla S(\mathbf{w})\rangle = 0$. The assumption on the gradient leads to an error of order R^2, but the use of the NIC leads to and error of order R. They have also kept some terms of order R, in order not to remove the effects of regularization altogether. That the results they obtained were reasonable is due to the fact that these corrections are small if R is small. On the other hand, the added terms are not hard to calculate and are not always small.

The above results will be returned to later in the methods section when designing the generalization criteria for our network.

Network Pruning

As mentioned in the previous section having too many weights leads to worse generalization ability. It is therefore advantageous to remove weights that are less important. Exactly how one chooses a weight to cut is the subject of much effort. A good review of the existing methods is given in (Reed 1993). The optimal brain surgery method (Hassibi 1993) seems to have the best chance of not making an error and has the added benefit of adjusting the remaining weights so as to stay near the minimum of the cost function. Thus reducing the need for retraining.

The method makes use once again of the hessian matrix and the expansion of the cost function about the minimum. We write H for the hessian of the cost function in the section. It is straightforward to derive the following saliency measure, or more

precisely, the increase in the empirical cost resulting from cutting the i^{th} weight:

$s_i = (1/2)\, w_i^2 / \langle H \rangle^{-1}_{ii}$

And the optimal adjustment to the k^{th} weights after such a cut is given by:

$\delta w_k = - \langle H \rangle^{-1}_{ki} (2 s_i / w_i)$

The hessian is a rather difficult quantity to calculate as it involves updating M^2 values at each data point where M is the number of weights in the network. The complexity is thus $O(N M^2)$. (By the way, this is normally much harder than inverting the matrix, i.e. $O(M^3)$). For that reason one often assumes that the matrix is diagonal.

In the appendix we describe our approximation to the hessian that allows it to be calculated more quickly.

Methodology

Overview of the Projection Procedure

Our projection will be done one step at a time as shown schematically in diagram 1. The state space is partitioned into regions indexed by j = 0, …, n. The state vector, $\mathbf{x}^{(k)}_j$, describes the mean within the j^{th} region after k steps of the process. $p^{(k)}_j$ is the probability of the j^{th} region. The variable $\sigma^{(k)}_j$ is defined as:

$\sigma^{(k)}_j = [\langle (\mathit{paid})^2 \rangle - \langle \mathit{paid} \rangle^2]^{1/2}$.

In the above equation it is understood that the expectation values are conditional given that the state is within the j^{th} region after k steps.

By including this estimate of the standard deviation we can not only model the expected future payments, but also their uncertainty.

We reserve the index j=0 for the *closed* claim state. A claim is classified as *closed* when it is deemed that no more payments will be made. We will assume that *closed* claims remain closed.

The vector **x** will be defined as:

$\mathbf{x} = (\mathit{age}, \mathit{reserve}, \mathit{paid}, \mathit{medical\ raito}, x_4, \ldots, x_8)$

The *medical ratio*, and the last five variables will be explained later.

The single step projection is shown in finer detail in diagram 2.

The Basic Projection

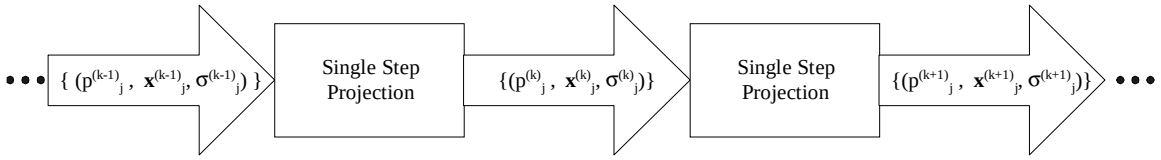

Diagram 1. The index j indicates the region of the state space for which $p^{(k)}_j$ is the probability and $x^{(k)}_j$ is the mean state vector given that it is within the jth region after k steps of the processs. This model is gives a discrete approximation to the continous state space, which becomes better with increasing n, the number of regions.

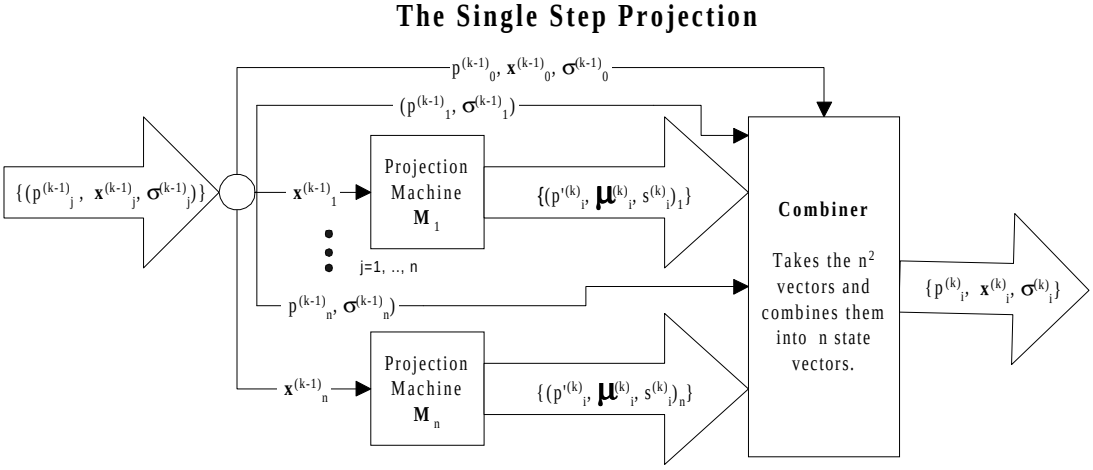

Diagram 2. The single step projection calculates a conditional probability distribution for each input and then partitions the output space. The individual $\mathbf{M}_j$ actually have some indirect communication in order to form a common partition, (indexed by i), that is best for just this set of input vectors.

The combiner sums up all the contributions from each of the input vectors:

$$\mathbf{x}^{(k)}_i = \Sigma\, p^{(k-1)}_j \,(p'^{(k)}_i\, \boldsymbol{\mu}^{(k)}_i)_j$$

$$p^{(k)}_i = \Sigma\, p^{(k-1)}_j \,(p'^{(k)}_i)_j.$$

The summations are over j. We will look at the projection machine **M** more closely in the following sections. For now we simply present, in diagram 3, the three blocks that make up the machine.

Data Adjustments

The first step of our work was to take a careful look at the data. The fact that the output state variables were mostly effected by the case reserve of the input state became apparent early. To try and find out which of the other possible input dimensions would likely help in prediction, we made a series of least square linear fits between the output *reserve* and *paid* verses the input *reserve*.

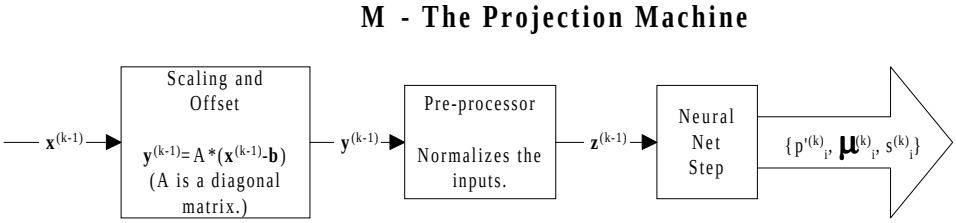

Diagram 3. In the first block the input variables are adjusted to have a variance of about 1. The second block removes most of the correlation between the input variables. The third block calculates the conditional probability distribution.

The data was partitioned along the various other dimensions one at a time and the correlation coefficients were used to estimate the reduction in the residual variance about the least square lines achieved by the partition.

In this way, we were able to narrow the input dimensions of interest down to four continuous and two discrete dimensions. The four continuous dimensions are the *age* of the claim, the *reserve*, the total *paid* to date, (simply referred to as the *paid*), the relative proportion of *medical* reserves to the total. The two discrete ones are the department that the claimant had been working in, (coded as four *classes* of departments) and the *group* code.

The last dimension was important. We had 31 sources of data. When looked at separately we found that it was possible to group the sources into six groups based on the similarity of the least square lines. If one is later presented with data from a new source and needed to use the neural network trained with this data to project it, then one could match the new data to one of the six groups based on its linear least square fit. This data is shown in appendix 1.

In order to avoid the problem of outlying points distorting the results we limited the data to *ages* from 100 days to 20 years. The very young claims are far too numerous, small and random to expect the net to deal with and the older claims were too few and had a very large variance. We also eliminated claims with *reserves* over $200,000, after having trended them for inflation at 1.5% per year.

The inflation trend was chosen after careful consideration of the data, which showed very little inflation over the four evaluations dates we had. We felt that some inflation must be incorporated into the model in order for it to be accurate going into the future. We could have argued for a trend of 0 to 3% based on the data and went with 1.5%.

The limiting of the data by *age* and size of claim is very much a standard practice in making estimates of claim development. Without doing so finding patterns becomes too difficult.

We had 14,326 pairs of data points over the three one-year intervals. We set aside 7,086 of the points as a test set. This data will be used when testing how well the net can be used to go out three steps with. From the claims in this test set, there were a total of 1,836 pairs of three-year interval test data points. We also used the one-year interval test data to confirm that our generalization error estimates were functioning properly. This left 7,240 pairs as the training set. Note that the claims used for the three-year interval test data were not in the one-year interval training data.

Nonlinear Rescaling of the Input Data - The Preprocessor

The problem we still faced was that the data was distributed very non-uniformly over the input space. There were lots of data for small young claims and much sparser data for older or larger claims. As we want the net to be able to generalize well especially in just those regions were the data was sparsest, we needed to help the net by making the input space more uniformly populated.

How we did this is shown schematically in diagram 4. We were, for example, able to remove all the mutual information between the *age* and *reserve* from z_1.

We start by normalizing the *age* of the claims to become more or less uniform over the interval -1 to 1. We did this by fitting the age data to a weibull distribution and then mapping each age into its cumulative probability. Thus the older aged claims were brought closer together. This helped the subsequent networks to learn in these regions.

The *paid* and *reserve* inputs were a bit trickier. The *paid* was very strongly correlated to the *age* and *reserves*. The correlation was non-linear so a linear decorrelation method would not help much. What was needed was a non-linear decorrelation method.

We started with the *reserves*. First, in the scaling and offset block, a small amount was added to each *reserve*. This was to avoid zero reserve values. Then the result was divided by the standard deviation. We then used back propagation to train $\mathbf{N}_0$, a simple net with one hidden node and two outputs. The normalized *age* was the sole input. We maximized the log-likelihood of a weibull distribution where the net generated the two weibull

parameters. The hidden node had a tanh activation and the 2 output nodes had exponential activations to avoid negative outputs. The network was straightforward to train as it always went to the same minimum point. We used quick-prop here and in all the training that follows.

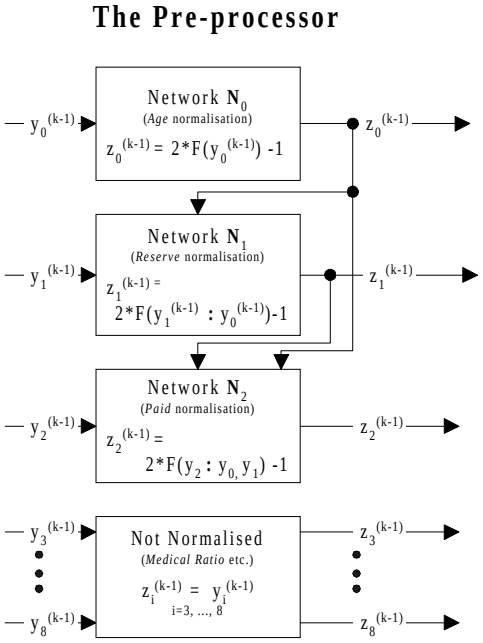

Diagram 4. Networks N_0 to N_2 are trained to maximize the log-likelihood and are then used to produce the conditional cumulative probability distributions F(). The result is that z_0, z_1 and z_2 are uniformly distributed and uncorrelated.

This distribution could then be used to map each reserve to its cumulative conditional probability given the age of the claim.

Normalized *reserve* = 2F(*reserve* | *age*) - 1

Or in terms of the notation of diagram 4:

$$z_1^{(k-1)} = 2F(y_1^{(k-1)} \mid y_0^{(k-1)}) - 1$$

We continued in this manner by fitting the *paid* to a distribution consisting of a weibull for *paids* greater than zero and a single parameter, 'p', as the probability of paids less than or equal to zero. A neural network with the normalized reserve and age as inputs then generated the three parameters. The *paids* were scaled by their standard deviation. The cost function was taken as the log-likelihood:

$$L=-\ln(p); \text{ for } y \le 0$$

$$L= -\ln(1-p)-\beta\ln(\lambda y) -\ln(\beta/y)+ (\lambda y)^{\beta}; \text{ for } y > 0$$

Where y is the *paid* for the claim, (we shall suppress the index 2 that identifies the *paid* variable here). The network had a single hidden layer with 5 nodes and sigmoid activation functions. The output node for p also had a sigmoid activation function, while the β and λ nodes simply used exponential activations to assure positive outputs. A lognormal distribution as well as more and fewer nodes was also tried but the above combination seemed to work best.

The *paids* were then mapped by:

$$z=2p-1; \text{ for } y \le 0$$

$$z =2[p + (1-p)\exp(-(\lambda y)^{\beta})]-1; \text{ for } y>0$$

The decorrelation is seen clearly by looking at the scatter plots in figures 1-6 or by the correlation coefficients that went from around 45% to 1%. The non-linear nature of the correlations is apparent in the plot of the mean of the distribution as a function of the two dependent variables, in figure 7.

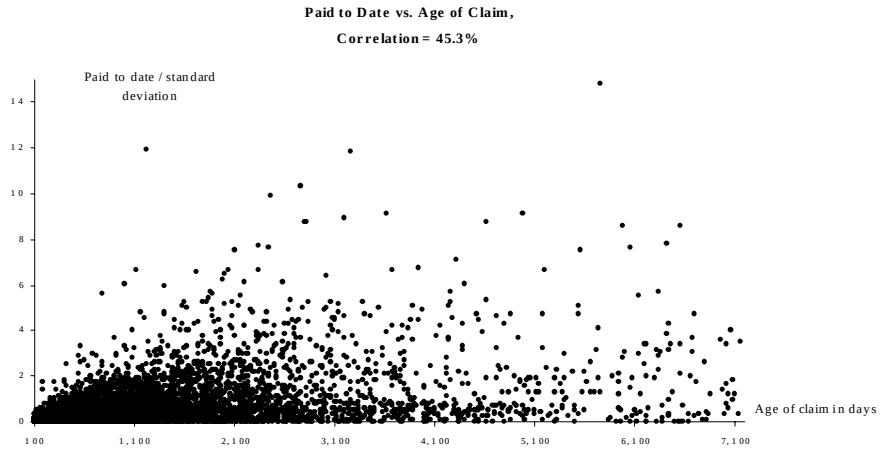

Figure 1. Paid to date is one of the input dimensions that was normalized and decorrelated using a simple network. Here is the raw input data with a high correlation between the age of the claim and the paid to date.

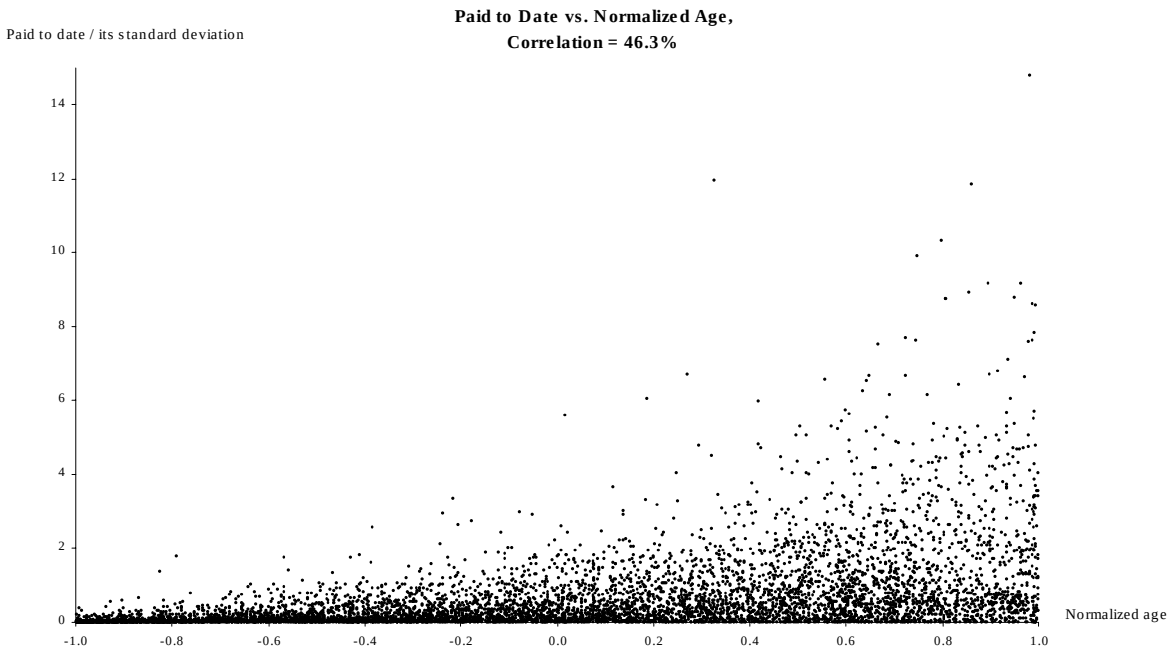

Figure 2. Here the age has been normalized to have a uniform distribution from -1 to +1. The correlation between the paid to date and the normalized age is still high.

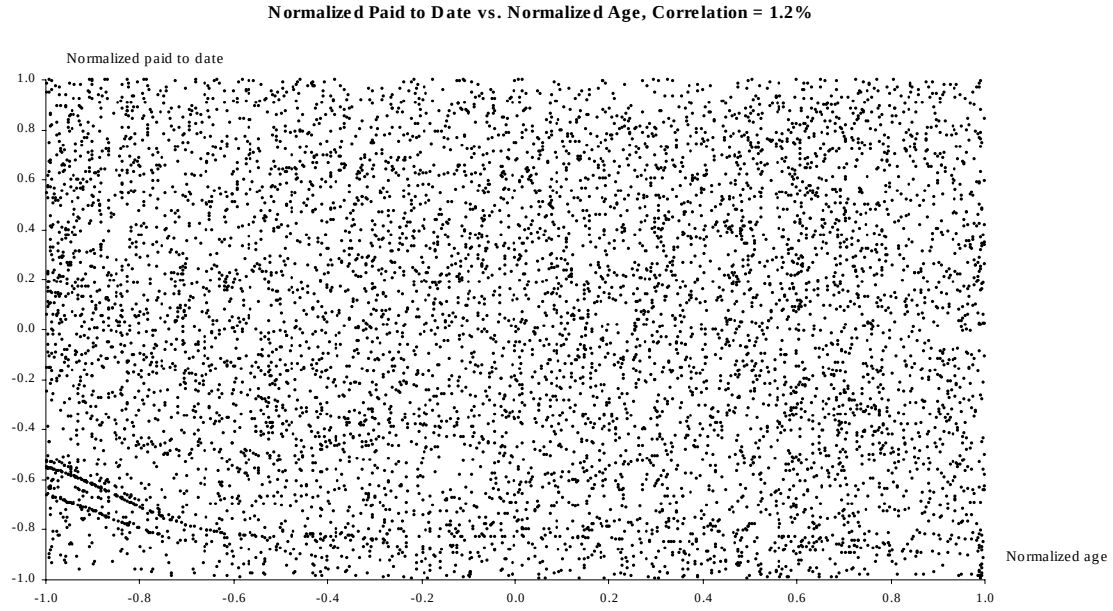

Figure 3. Here the paid to date has also been normalized to have a uniform distribution from -1 to +1. The correlation between the paid to date and the normalized age has been removed.

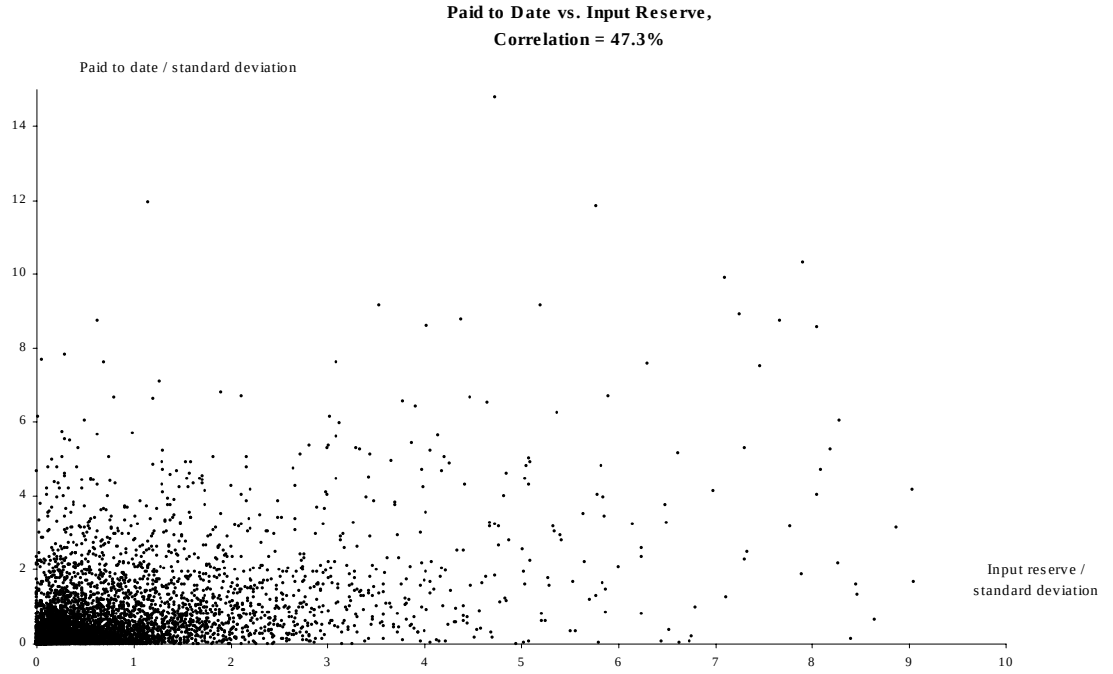

Figure 4. Paid to date is one of the input dimensions that was normalized and decorrelated using a simple network. Here is the raw input data with a high correlation between the reserve and the paid to date.

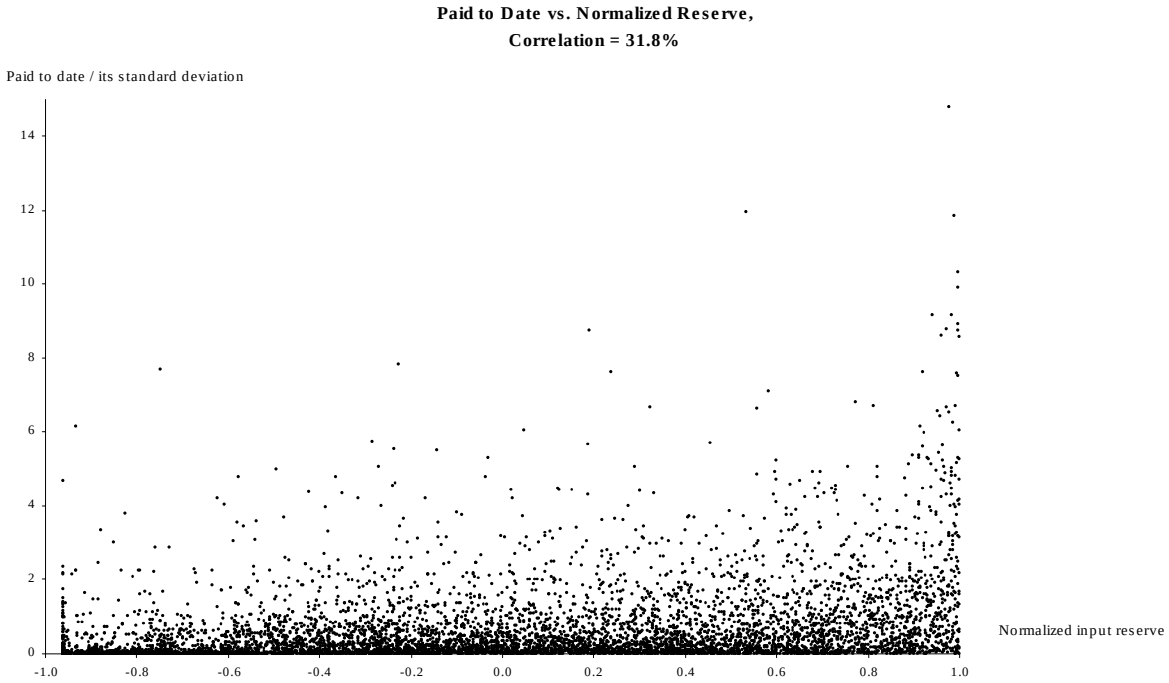

Figure 5. Here the reserves have been normalized and the correlation between the reserve and the paid is reduced.

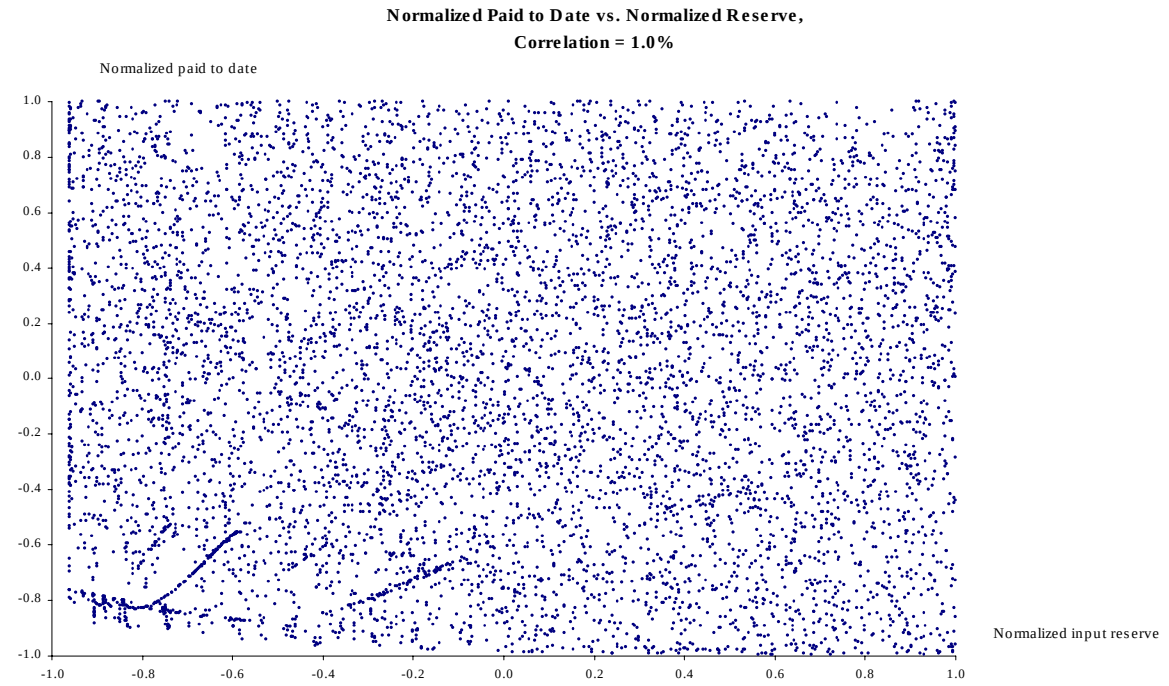

Figure 6. Here the paid has been normalized and correlation between the reserve and the paid has been eliminated

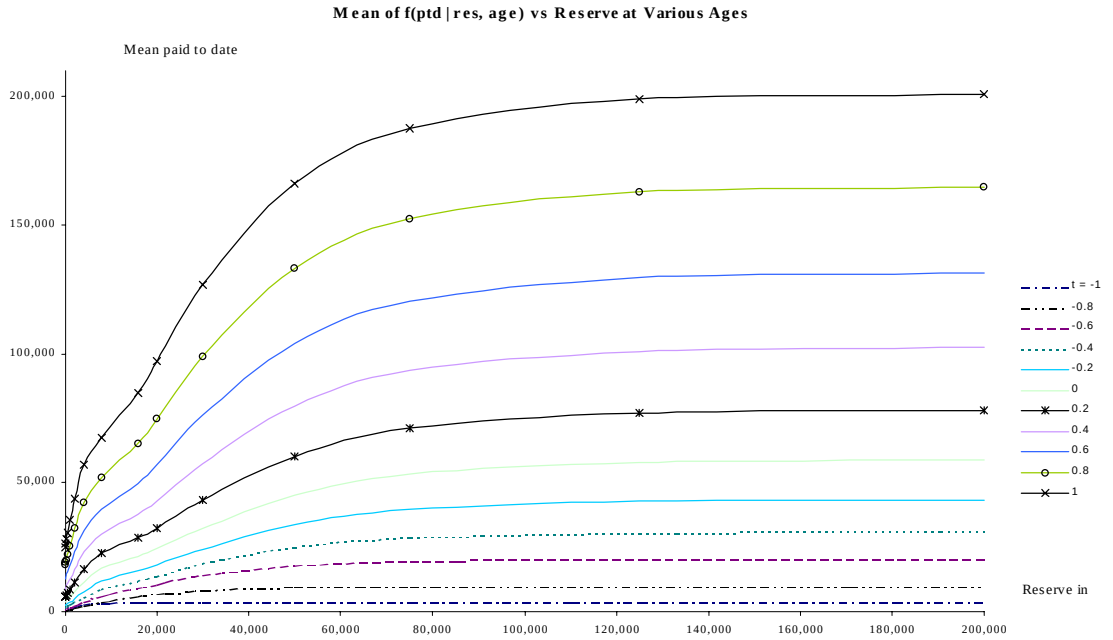

Figure 7. The mean of the maximum likelihood distribution for the paid to date given the reserve and age. This distribution was used to normalize the input dimensions. One can see that the correlations were not very linear.

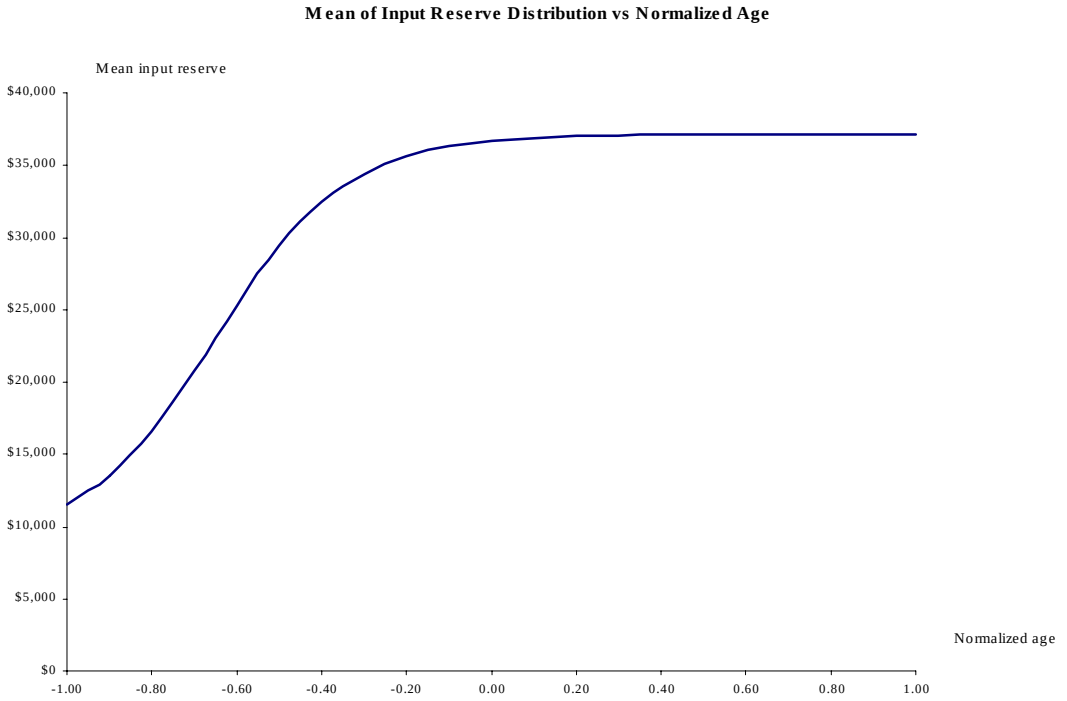

Figure 8. The mean of the maximum likelihood distribution for the input reserve given the age, which was used to normalize the input dimensions. One can again see that the correlations were not very linear.

The discrete inputs were coded as binary numbers. The six groups were coded into three binary inputs, x_4, x_5 and x_6 and the 4 classes into 2 binary inputs, x_7 and x_8. These inputs were either +1/2 or -1/2 depending on whether they represented a binary 1 or 0.

This then gave us our nine inputs to the network: three normalize/decorrelated inputs, (ie. *age*, *reserve* and *paid*), the *medical* ratio, and the five binary inputs.

Training the Networks - The Neural Network Step

We start this section by examining the last schematic diagram, diagram 5. It shows the four multilayer perceptron networks and how they work together to produce a complete conditional probability distribution of the output state.

The first projection network to be trained, $\mathbf{N}_5$ was the most critical. The net had to learn the conditional distribution of the *reserve* one step into the future. It was critical because the future development of the claim is mostly a function of the *reserve*.

The approach was very similar to that used for the decorrelation of the previous section. The network now had the extra inputs and the output cost function was based on a bit more general distribution. Also, the training data was split into two sets, those claims that remained open at the end of the step and those that were closed and therefore had no *reserve*. The set of open training data had 4,243 pairs while the test set had 4,121 pairs. The numbers for the closed claim sets were 2,997 and 2,965. The network was trained with the first set. We will train another network to learn the probability of remaining open later.

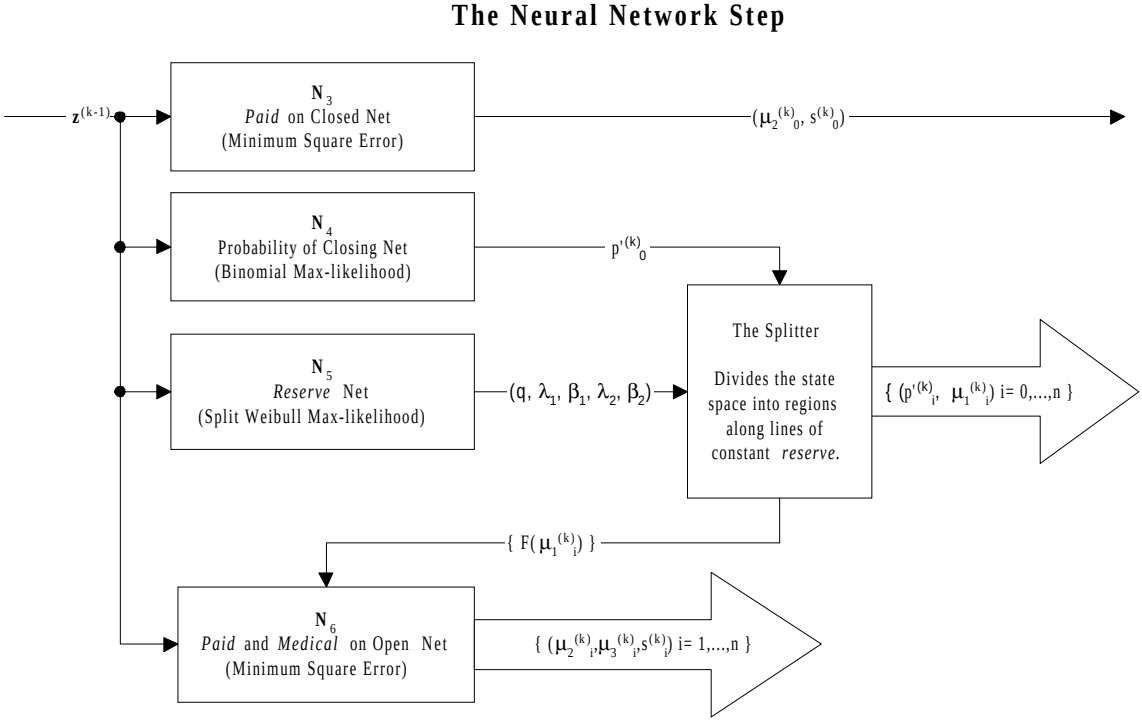

Diagram 5. The age of the claim, x_0, increases by 1 year and the static variables, (x_4 to x_9), do not change and so they need no network to project them. The Medical on the closed claims, $x_9{}^{(k)}{}_0$, is not needed.

The output *reserves* were scaled by their standard deviation after adding a small constant to them to avoid zero *reserves*.

The distribution taken was a pair of weibulls each covering a different range of output *reserves*. To set the ranges optimally for each input point we chose to use the linear least square fit between the input and output *reserve* as the dividing value. The first range was from 0 to the value of the least square fit and the second range was from the least square fit with no upper limit

The weibull probability density function is given by,

$$wb(\beta, \lambda, x) = (\beta / x)(\lambda x)^{\beta} \exp(-(\lambda x)^{\beta});$$

And we took a pair of these to form $f(x \mid \mathbf{z})$,

$$f(x \mid \mathbf{z}) = q\, wb(\beta_1, \lambda_1, x) / (1 - \exp(-(\lambda_1 t)^{\beta 1})); \; x \in (0, t]$$

$$f(x \mid \mathbf{z}) = (1-q)\, wb(\beta_2, \lambda_2, x)\; \exp((\lambda_2 t)^{\beta 2}); \quad x > t.$$

Here t = least square linear fit between the *reserve* out, x, and the *reserve* in. This linear fit was done once for all the training data and the slope and intercept was then used to calculate each t.

The cost function then became

$$S = -\ln(f(x \mid \mathbf{z}))$$

The network was then trained to learn the five parameters of the model, $(q, \beta_1, \lambda_1, \beta_2, \lambda_2)$. The output functions were, $(1+e^{-v})^{-1}$ for q, and e^{v} for the other parameters.

It is important to find a network that learns as much as possible from the training data without overfitting. Overfitting seems to occur when the training points per number of parameters in the model is too small. Now one approach to avoid overfitting would be to start with an over dimensioned network and then train it to the minimum training error. Now the network will presumably be overfitting the training data. Then one could prune the network until the overfitting is eliminated.

We have chosen another approach where we tried to avoid ever training a network to overfitting. This is reasonable given that our generalization measure is only accurate near the best generalization solution for the model.

We avoid overfitting by using regularization, which reduces the effective dimension to be significantly less of the amount of training data. The regularization matrix was chosen as the identity matrix times a parameter R. Choosing the R that gives the desired effective dimension is half guesswork half trial and error.

We first trained 22 networks minimum of the regularized log-likelihood cost function on the training set with R set to 0.01. These nets all had 9-12-7-5 architecture with tanh activation functions on the hidden nodes. The 9 being the number of input nodes and the 5 being the output. The two hidden layers were then 12 and 7 nodes each. Thus the networks had 251 weights. These training runs were done with the priority of trying as many networks as possible in a reasonable time. We therefore chose to stop the training of each network after 500 epochs even though the minimum hadn't yet been reached.

We then chose the network with the lowest training error and trained it further until the training error didn't improve anymore. This network was then our starting network for our pruning/regularization procedure described next.

Network Pruning with Adaptive Regularization

We had at this point trained a fully connected network to the minimum of the training error. The number of weights in the network was large considering of the amount of training data. We hope to have no over fitting, however, because the weights were being held small by the regularization term.

We now wanted to see if a network with larger weights but fewer of them could give a better generalization error.

Here we utilize the generalization measure defined earlier and repeated here,

$$E(R) = NIC - (1/2)\, \mathbf{w}^T R \mathbf{w}$$

$$E(R) = \langle S(\mathbf{w}) \rangle + \mathrm{tr}(\mathrm{Var}[\nabla S(\mathbf{w})]\, [\langle \nabla S(\mathbf{w}) \nabla S(\mathbf{w})^T \rangle + R]^{-1}) / N$$

At the minimum of $\langle D_R(\mathbf{w}) \rangle$.

We needed to minimized E(R) with regard to R and the number of pruned weights. We did this by calculating E(R) for two values of R. This involved training the network weights to a minimum of $\langle D_R(\mathbf{w}) \rangle$. The starting point for this training was the final network for the previous R, after first making a course adjustment to the weights for the change in R as described in the appendix.

We then picked the best E(R) of the two and pruned some of the weights. After re-training we calculated a new E(R). Here we chose a new R based on whether the previous two R values indicated that a smaller or larger R was best. We retrain again and chose the best E(R) network for pruning.

Now we just continue calculating two E(R)'s for each size network. In this way the R-value and the effective dimensions can change as the net gets smaller. The step in the R-values was calculated similarly to quick-prop. The gradient was found from the difference in the E(R). The previous gradient came from the previous size network. The size of the step in the R parameter was limited to 10%.

At first we pruned five weights using optimal brain surgery. Then we gradually reduced the number pruned each time so that the % change in the size of the network remained about constant. Thus at a size of 200 we pruned only 4, at 100 we pruned 2 and so on.

We continued this until the network no longer seemed to be able to model the process sufficiently well. We then have networks at two values of R for each size of network, and two sizes for each R.

The results are shown in figure 9. Here we show the E(R) and the log-likelihood cost of the test set for the networks with the best E(R) at each number of weights. The test error was simply calculated without being used for any training decisions. It then can be used as a control of the method. As one can see the test error followed our E(R) measure very well.

We chose the network with the minimum E(R) that had a test error just slightly higher than the minimum test error. If we had used the NIC directly as the generalization error estimate we would have chosen a smaller network with a somewhat larger test error.

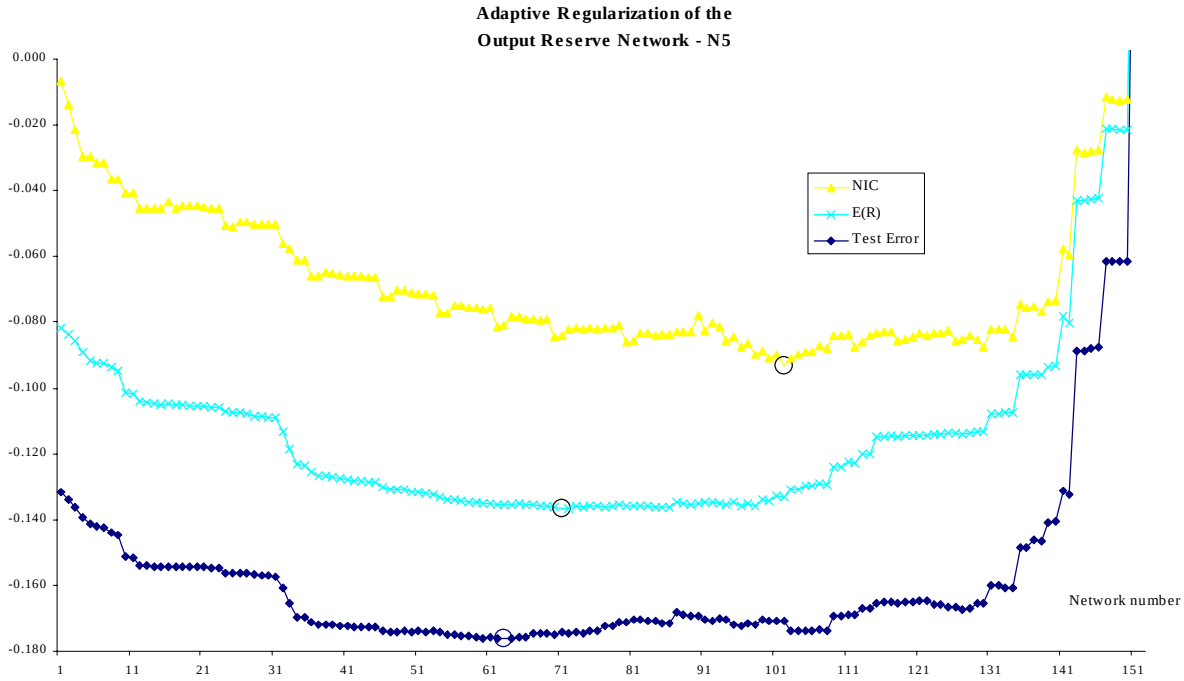

Figure 9. The result of pruning with adaptive regularization for the output reserve network showing the network information criteria, NIC, the E(R) generalization measure and the error on the test data as the pruning progressed. The circles show the minimum.

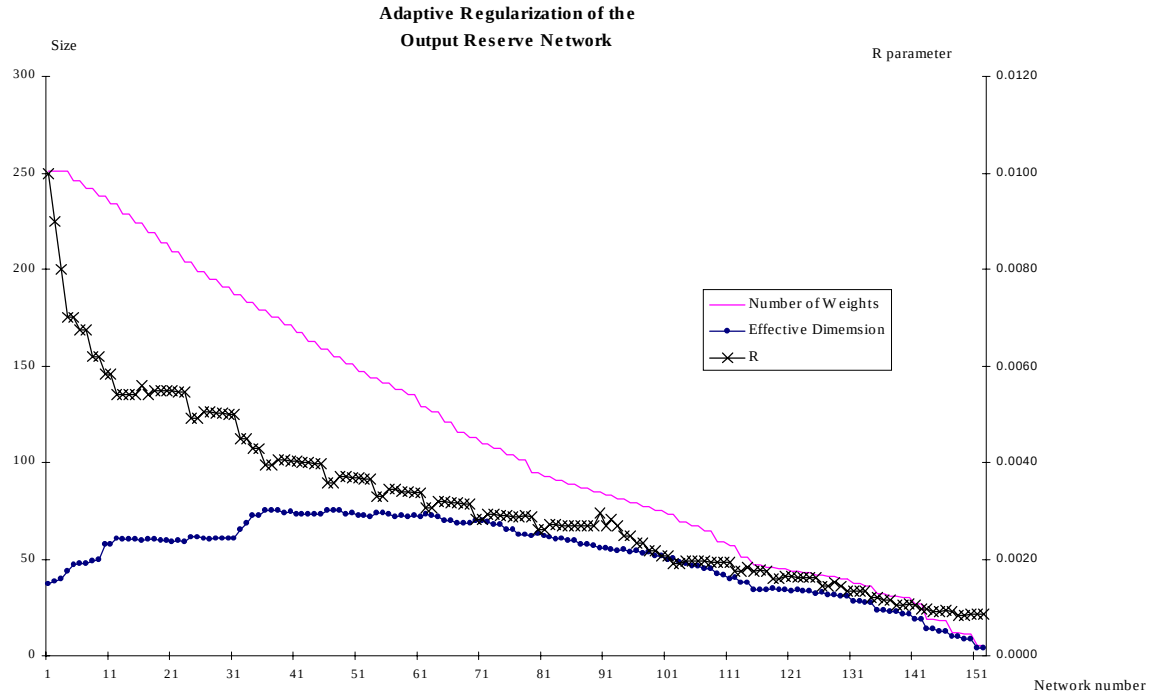

Figure 10. The result of the pruning with adaptive regularization for the output reserve network showing the number of weights, the effective dimension and the R-value as the pruning progressed.

More Networks

Having learned the distribution of the output *reserves*, we were now in a position to learn the mean payment (ie. change in *paid*) given the input state and the output *reserve*. In effect we treat the output *reserve* as just another input, normalized just the same way as the others using the network we just described.

Now the cost function for network $\mathbf{N}_6$ became a mean square error function.

$$\langle S(\mathbf{w}) \rangle = (1/2) \langle \Sigma (u_i(\mathbf{w})-d_i)^2 \rangle$$

We have written the outputs of the network as $u_i(\mathbf{w})$ and the targets as d_i. The subscript i refers to the payment, $(payment)^2$ and the *medical* ratio of the output state. We will need the means for all three of these when we project process further into the future. We also add a regularization term as before.

$$\langle D(\mathbf{w}) \rangle = \langle S(\mathbf{w}) \rangle + (1/2)\, \mathbf{w}^T R \mathbf{w}$$

Unlike the prior networks based on the weibull distribution, here we can have negative target values and need to set the mean target to 0 by first subtracting the mean payment and then dividing by the standard deviation.

We train the network in exactly the same way as for the reserve network. We first train a number of randomly initialized networks for 500 epochs. The networks all had a 10-10-7-3 architecture and R=.005. Then we take the best one and apply the network pruning with adaptive regularization algorithm to it.

We then select the network with the best E(R) for our final machine, (see figure 11). In this case the best E(R) also had the best test error. The best NIC was for a much smaller network with a much higher test error.

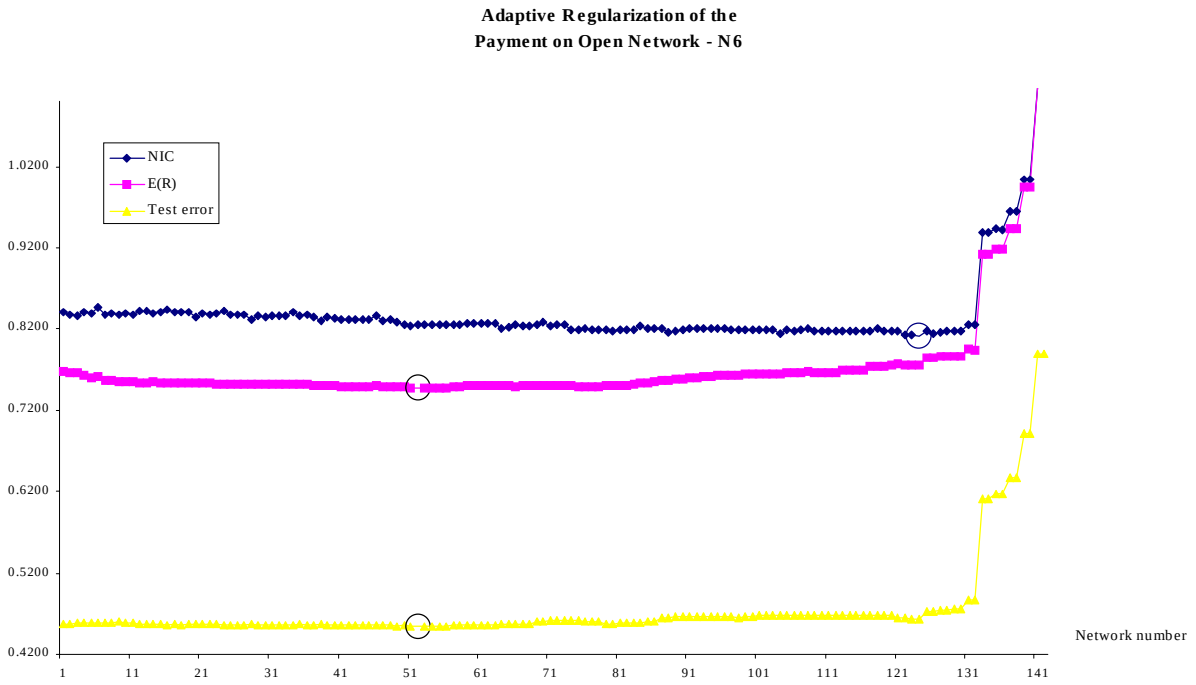

Figure 11. The result of pruning with adaptive regularization for the payment on open network showing the network information criteria, NIC, the E(R) generalization measure and the error on the test data as the pruning progressed. The circles show the minimum.

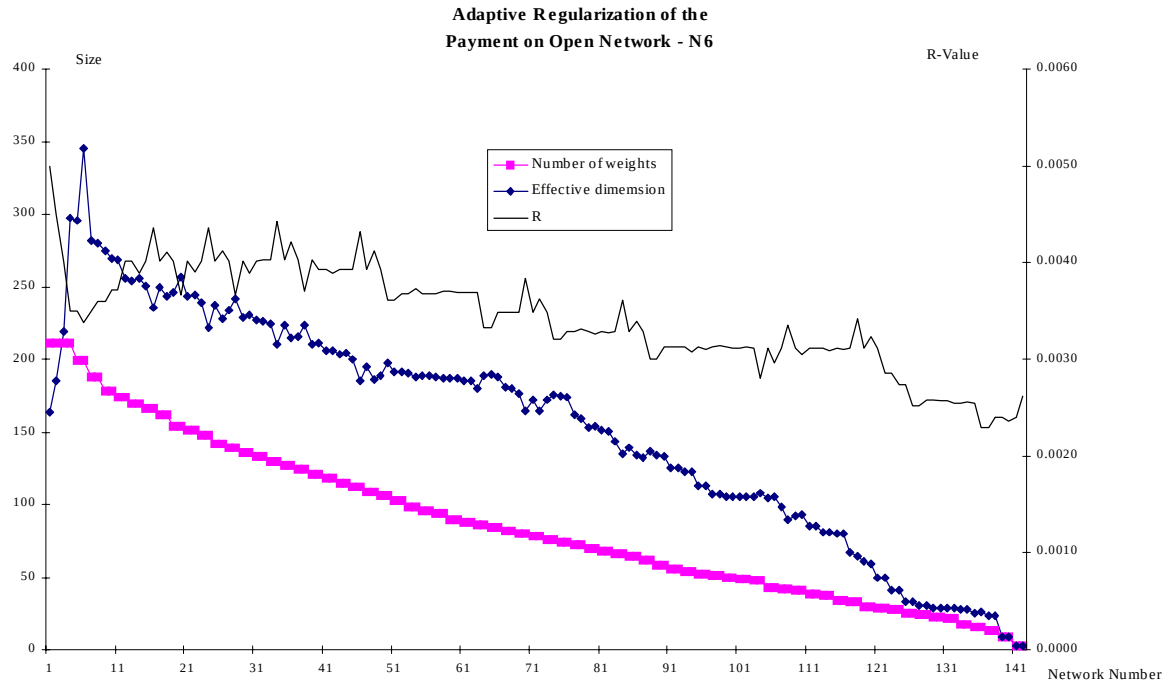

Figure 12. The result of the pruning with adaptive regularization for the payment on open network showing the number of weights, the effective dimension and the R-value as the pruning progressed

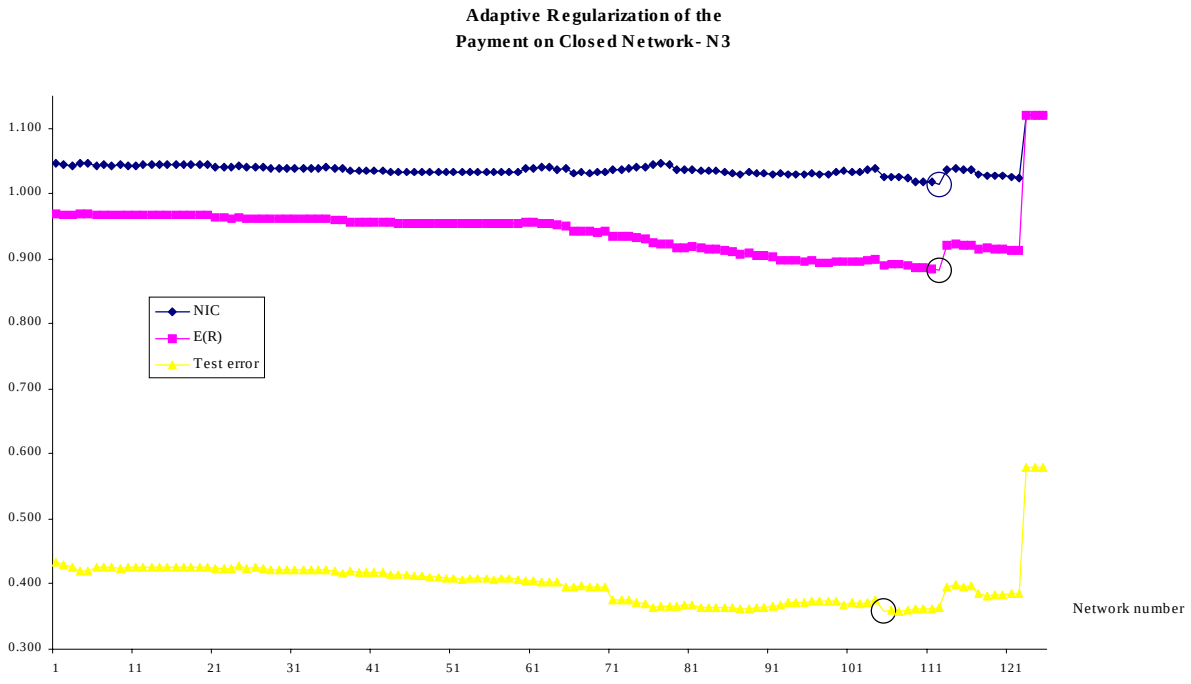

Figure 13. The result of pruning with adaptive regularization for the payment on closed network showing the network information criteria, NIC, the E(R) generalization measure and the error on the test data as the pruning progressed. The circles show the minimum.

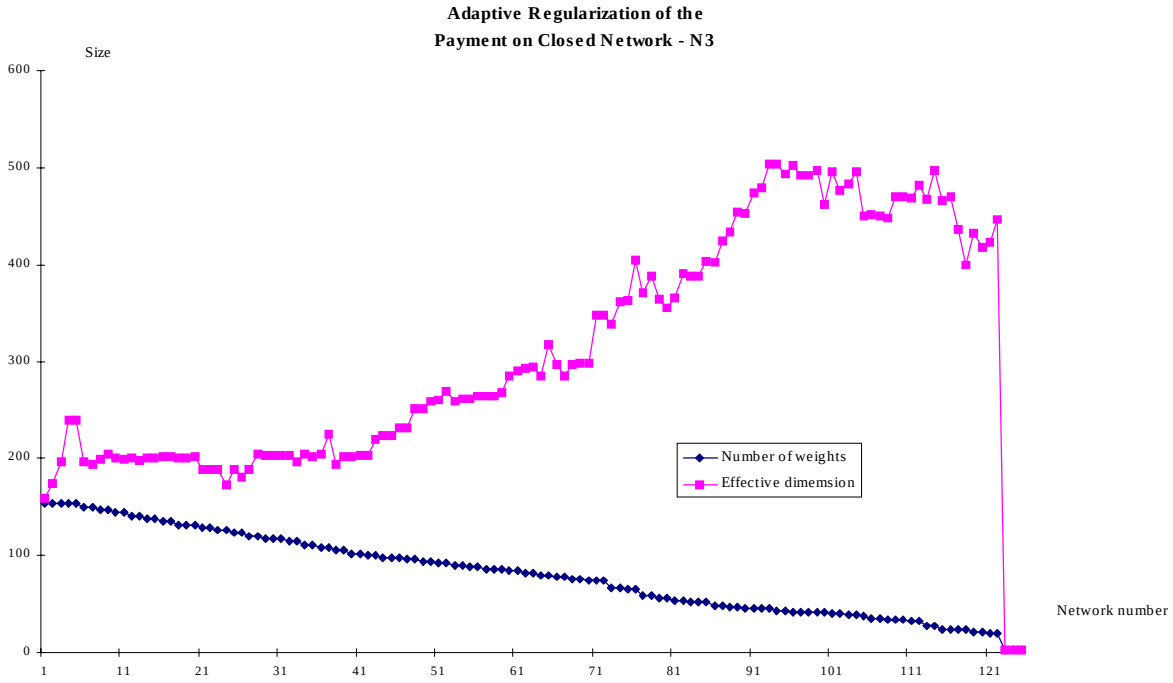

Figure 14. The result of the pruning with adaptive regularization for the payment on closed network showing the number of weights and the effective dimension as the pruning progressed.

We found that the square error cost function was much more time consuming to train and to calculate the hessian for. It would probably have been better to use a maximum likelihood cost function for the payments as well.

The payments on the closed claims were then learned by a similar procedure. Here we started from a 9-10-4-2 mean square error network with R=.01. The resulting minimum E(R) network had a test error somewhat higher than the minimum, figure 13.

To learn the probability of a claim closing we used a binomial log-likelihood network with the probability of closing being the sole network output. The activation function on the output was $(1+e^{-x})^{-1}$. The initial networks were chosen to have a 9-8-3-1 architecture and a R=.01.

The pruning/adaptive regularization procedure was then applied. Again the best E(R) network also had the best test error, figure 15.

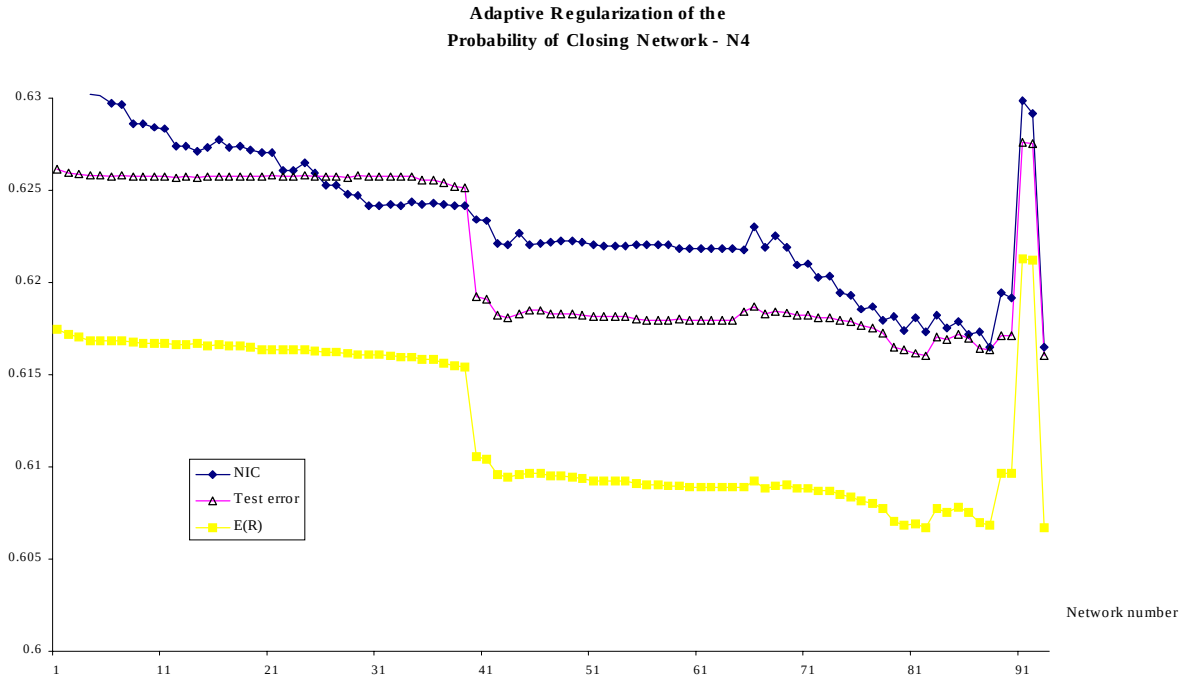

Figure 15. The result of pruning with adaptive regularization for the probability of closing network showing the network information criteria, NIC, the E(R) generalization measure and the error on the test data as the pruning progressed.

The Discrete Approximation

We now have trained a total of 7 networks, 3 for decorrelating and normalizing the inputs and 4 for projecting the output state distribution. These 7 networks then form the projection machine.

We first take the set of input state vectors, $\{ \mathbf{x}^{(k-1)}{}_j \}$ and normalize the input dimensions with the three input networks, $\mathbf{N}_0$, $\mathbf{N}_1$ and $\mathbf{N}_2$. We then calculate the probability of the claim closing, $(p'^{(k)}{}_0)_j$, using network $\mathbf{N}_4$. We can then calculate the probability distribution of the output *reserve* one year into the future using the *reserve* projection network $\mathbf{N}_5$. Then we split the output space up into a number of regions above and below the mean reserve. For example, we might split it into four reserve regions:

$A^{(k)}{}_0$ = (0, (mean reserve)/2],

$A^{(k)}{}_1$=((mean reserve)/2, (mean reserve)],

$A^{(k)}{}_2$=((mean reserve), 3(mean reserve)/2] and

$A^{(k)}{}_3$=(3(mean reserve)/2, ∞).

In diagram 2 the individual projection machines are shown as operating independently with no communication between them. That is not quite accurate as a higher level controller collects the information for determining the mean output *reserve* and making the partition which it then gives back to the individual machines..

We use the probability density function to calculate the mean reserve, $\mu_1{}^{(k)}{}_i$ and the probability, $p'^{(k)}{}_i$, for each region.

$$(p'^{(k)}{}_i)_j = (1- p'^{(k)}{}_0)_j \int_{A^{(k)}{}_i} f(u \mid \mathbf{x}^{(k-1)}{}_j) d\, u$$

$$(\mu_1{}^{(k)}{}_i)_j = \left[(1 - p'^{(k)}{}_0)_j / (p'^{(k)}{}_i)_j\right] \int_{A^{(k)}{}_i} u\, f(u \mid \mathbf{x}^{(k-1)})_j)\, d\, u$$

where $f(u \mid \mathbf{x}^{(k-1)})_j)$ is the probability density function generated by $\mathbf{N}_5$.

These $\mu_1{}^{(k)}{}_i$ are then normalized using the probability density function generated by $\mathbf{N}_5$ and then used as inputs to the payment on open network, $\mathbf{N}_6$. We can thus find the mean payments, $(\text{payments})^2$ and *medical* ratio for the four output regions. We then use $\mathbf{N}_3$ to get the mean payment and $(\text{payment})^2$ if it does close. Thus we have all the components of $\mathbf{x}^{(k)}{}_j$ for projecting one more step.

It is also of interest to calculate the expected value of the product of the output reserve and the output paid to date. This will allow us to estimate the covariance between the reserve and the payments. We need that to estimate the variance in the sum of the payments and reserves. This proved to be no problem to calculate using our machine.

Results

By applying the described method we projected the 1,836 claims from the test set three years into the future. We then compared the results with the actual development over the three years from 6/30/97 to 6/30/00.

We did this for several different numbers of output regions. We found that when increasing from 16 to 32 regions the estimates changed by about 0.1% so that further subdividing the output space was deemed unnecessary. The results are shown in table 1.

If we take the 32-region projection as our best estimate we find that our predicted payments were about 1.5 standard deviations higher than the actual payments, our reserves were about 0.7 standard deviations above the actual reserves and the sum was about 1.4 standard deviations above the actual sum.

We predicted that the expected number of closed claims after three years would be 1,306. The actual number was 1,310, which was quite close.

It is interesting to also compare our variance derived from the predicted expected payments and $(\text{payments})^2$ with the sum of the squared differences between our prediction and the actual payments, (shown in the last column of table 1). The square roots of these two quantities were $1.10 and $1.13 (million) respectively. Doing the same comparison for the reserves we find 0.74 and 0.84. And finally for the sum of the reserves and payments we find 1.54 and 1.64.

This seems to indicate that our estimated standard deviations were reasonably accurate. These standard deviations could then be used along with the means to fit the individual claims to a probability distribution. Those distributions could in turn be used to calculate the confidence intervals for the aggregate claim distribution using standard actuarial techniques. In our case we have so many claims that a normal distribution is probably fairly correct for the aggregate.

The bias was easily resolved using our model with such a large number of test claims. We did this by using only one output region and comparing to the 32-region estimate. With only one region we are projecting the second step into the future from the mean state after the first step. Then we project from the resulting biased estimated mean intermediate second state for the third step and so on. This corresponds to all the standard methods of projecting claim development. The common premise is that the 2 step projection can use the mean of the one step projection to project off of. We thus estimated the bias inherent in the standard approach.

When comparing the one region estimate to the 32 region estimate we see that the bias is about -17% for the payments over three years, 5% for the reserves and -11% for the sum of the reserves and payments.

Table 1. The results of projecting the claims three years using our projection machine. The projections were done for different numbers of output regions. The more regions the closer the results are to the true continuos state space projection. The estimated standard deviation is based on the projected mean and mean of the square.

	Projected Total	Actual Total	Error	Estimated Standard Deviation	Error as % of Standard Deviation	Square Root of Square Deviation From Projection
32 Regions						
Expected (Res. + Pay)	41,389,483	39,233,515	2,155,968	1,546,307	139%	1,644,567
Expected Payments	29,909,720	28,283,578	1,626,142	1,104,650	147%	1,136,350
Expected Reserve	11,479,762	10,949,937	529,825	741,164	71%	846,118
Number Closed	1,306	1,310	(4)	18.3	-22%	
16 Regions						
Expected (Res. + Pay)	41,438,618	39,233,515	2,205,103	1,538,919	143%	1,644,900
Expected Payments	29,948,056	28,283,578	1,664,478	1,107,280	150%	1,136,614
Expected Reserve	11,490,563	10,949,937	540,626	733,021	74%	846,148
Number Closed	1,305	1,310	(5)	18.3	-25%	
8 Regions						
Expected (Res. + Pay)	41,298,605	39,233,515	2,065,090	1,524,537	135%	1,644,916
Expected Payments	29,797,989	28,283,578	1,514,411	1,105,435	137%	1,136,625
Expected Reserve	11,500,616	10,949,937	550,679	724,097	76%	846,188
Number Closed	1,305	1,310	(5)	18.3	-28%	
4 Regions						
Expected (Res. + Pay)	40,785,232	39,233,515	1,551,717	1,494,283	104%	1,645,158
Expected Payments	29,262,291	28,283,578	978,713	1,094,999	89%	1,136,979
Expected Reserve	11,522,940	10,949,937	573,003	711,621	81%	846,255
Number Closed	1,304	1,310	(6)	18.3	-33%	
1 Region						
Expected (Res. + Pay)	36,915,162	39,233,515	(2,318,353)	1,021,033	-227%	1,649,911
Expected Payments	24,862,019	28,283,578	(3,421,559)	737,257	-464%	1,144,388
Expected Reserve	12,053,143	10,949,937	1,103,206	612,794	180%	848,784
Number Closed	1,278	1,310	(32)	18.6	-170%	

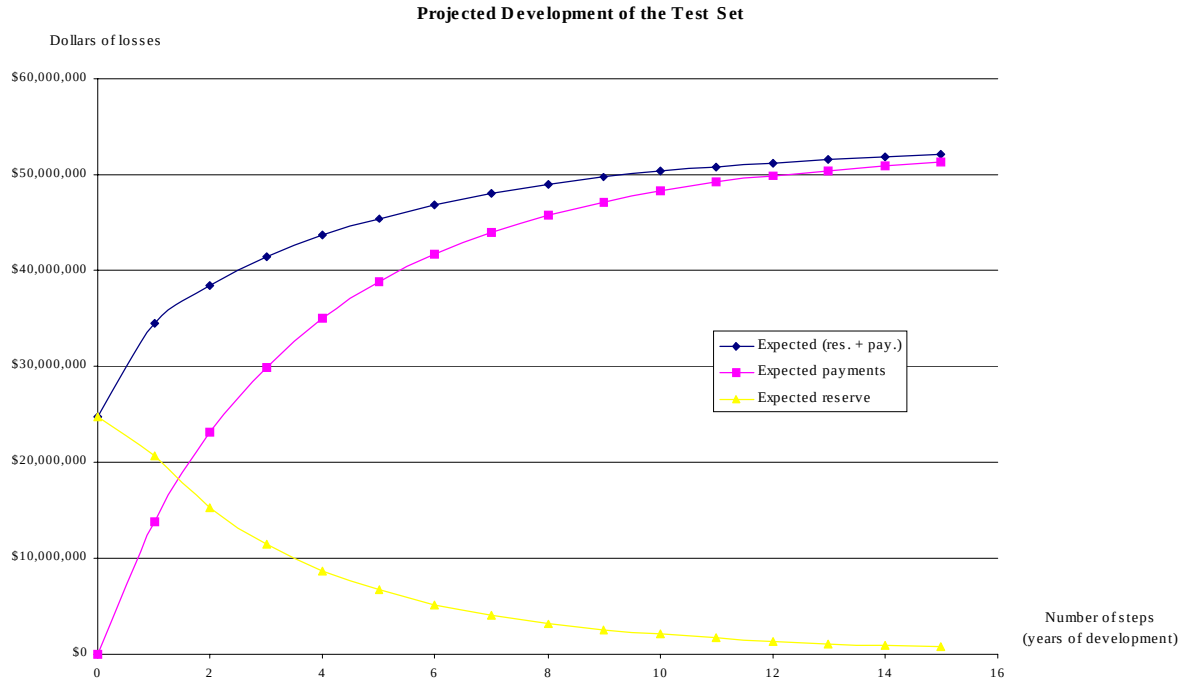

Figure 16. The Development of the open claims in the test set over 15 years as projected by our machine.

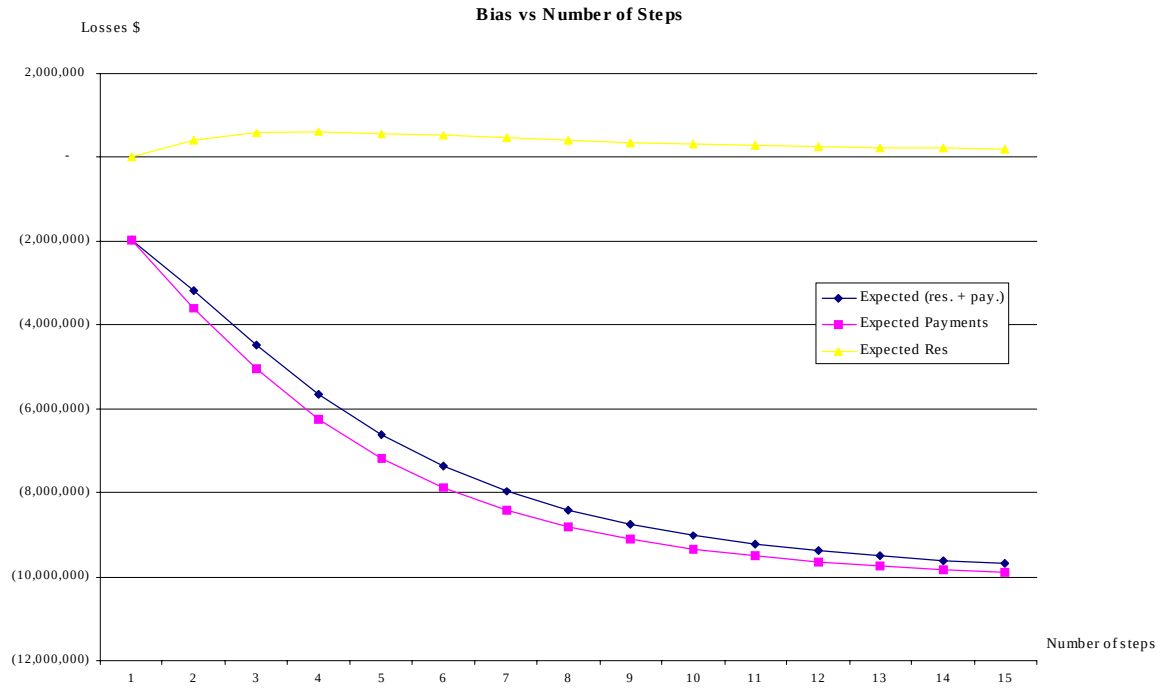

Figure 17. The bias as measured by our method shown from 1 to 15 steps of the projection.

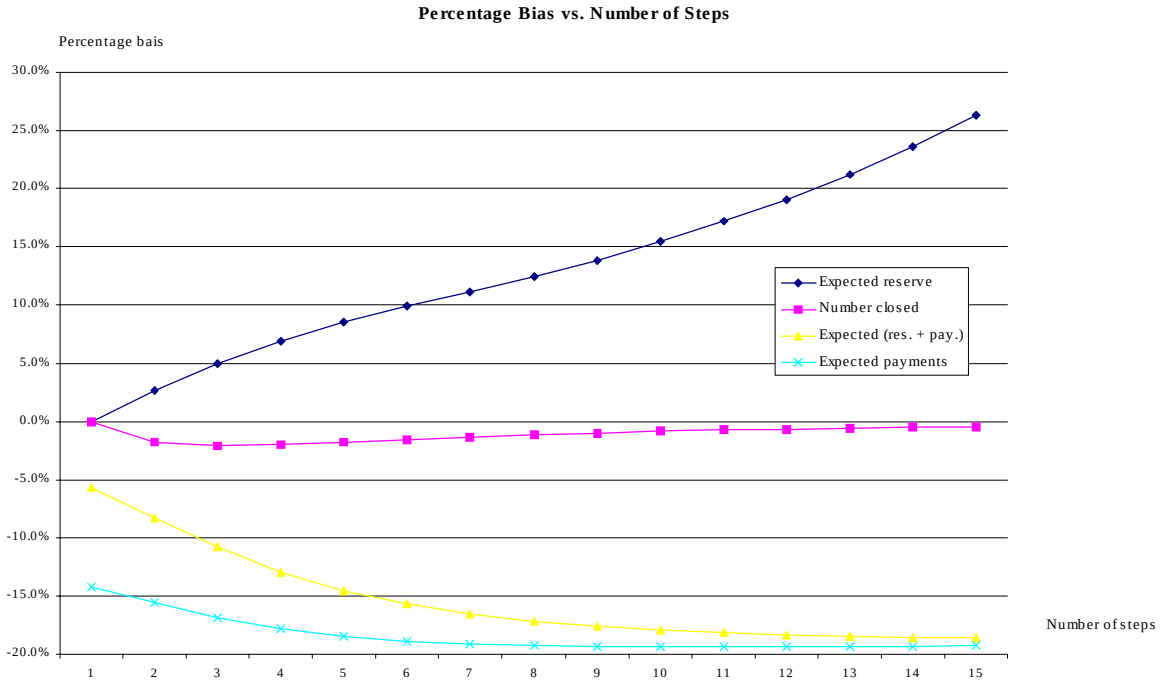

Figure 18. The bias as a percentage of the 32 region estimate.

The bias tends to grow the further out one projects the claims. We continued projecting the 32 region and one region estimates out to 15 steps. The results are shown in figures 16-18. One can see that the bias in the payments seems to be as high as -19%. The percentage bias in the reserves seem to grow considerable as one goes out further but the reserves themselves become much smaller so that the bias in dollars as opposed to a percentage, is becoming less.

One can conclude that the outstanding losses estimated by the standard methods would produce a 19% underestimate for these claims. This is frankly huge. It seems hard to accept that the bias is in fact that large but at the same time one can no longer simple ignore the bias as it clearly is significant.

One might ask how such a large bias could be ignored for so long. The answer is that the numbers being estimated are expected to be subject to large random variations. Also the bias doesn't show up at once but is very gradual. Each year the estimated ultimate loss value goes up a little on average. This increase is easily mistaken for bad luck.

We stated earlier that our method should reduce the variance in the estimate of the outstanding losses due to the fact that more information from individual claims is used in making the prediction. This seems to be very reasonable but can not be tested empirically from the results we have. One would need to set up a different sort of study to see this effect.

We also stated that inhomogeneous data could be better accommodated by our method. Again it is hard to quantify this assertion. In looking over the weights that the pruning algorithm did not remove one can say that the dimensions that were designed to model the inhomogenaity of the data were more or less ignored by the paid on closed and probability of closing networks. The paid on open and the reserve networks however did seem to benefit from the inclusion of these dimensions.

Discussion

Our results were very satisfying but there are some remaining issues and much room for further study.

It would seem that our reserve and closed claim projections were both accurate to within the statistical variation in the claims process. We were not entirely satisfied with the performance of the payment prediction being 1.5 standard deviations from the actual payments. That difference is not large enough to reject the projected expected payment from the model as the true mean, but it doesn't lend a lot of support to it either.

One must keep in mind that this data is from the real world and is subject to many forces that can not be taken account of by our model. Having followed these claims for over ten years one realizes that the assumption that the future development will be as the past is inaccurate. Political, social and economic changes have a large effect on the claims and as a result the nature of the claims changes from year to year.

Nevertheless we believe that we could do better. We would try using a log-likelihood network for the payments similar to what we used for the reserves. That may very well improve the payment prediction power and will certainly speed up the training process.

One important issue is the large losses or the claims with more than $200,000 in reserves. These claims were removed from our data as they would have caused us difficulty. There is nothing fundamentally different about those claims and it should be possible to project them as well but the network may need to be a separate one.

Another important part of the outstanding losses that was not estimated here is the claims that are not open at the time of the projection. These include claims that are closed but may reopen, as well as claims for injuries that have happened but no claim has been filed yet. The reopened claims are certainly very important and they make up a significant part of the total outstanding. Given data on past reopened claims one can make an estimate of this amount and add it in.

One needs to also estimate the claims that are open now but will close and then reopen. Our machine assumes that claims stay closed.

We would have liked to also investigate the common situation where there is much less data available to train the networks. Unfortunately time did not allow for this. One would have liked to get a feel for how the size of the networks should change with less data.

As we have already stated we would have liked to try a log-likelihood type network for the payments and compare to the mean square error estimate. The mean square error networks proved difficult to train. The hessian took much longer to calculate and the results were not as good as the log-likelihood networks for the reserves and closed claims.

We have successfully demonstrated that the practice of using the NIC as an estimate of the generalization error when one has a regulation term in the cost function leads to non-optimal networks. Instead, the E-measure that we have introduced seemed to work better and has a better theoretical support. It is also no harder to calculate.

The approximation to the hessian that we have also introduced seems to be of some value for pruning the networks. It speeds up the adaptive regulation procedure considerably. The estimate seems to be sufficiently good when the network has been optimally pruned. We would however recommend the full hessian calculation be used together with the approximation as we have done.

The entire adaptive regulation procedure worked very well and must be considered a good way to set the regulation parameter. That is otherwise rather difficult to do.

We also have used a method of decorrelation for the inputs that aided our final networks in learning. Although it may have seemed complicated it in fact

was rather easy to implement. The normalization networks were small enough that the training of them went quickly.

The most significant result of this work was the measurement of the bias in the standard methods for the projection. These methods all project multiple steps of the process from the mean state after each step. The bias found (-19%) is very alarming given the amount of money involved. Even if the bias were half that it would be a severe blow to the balance sheets of many businesses and institutions. Of course, such an accounting change will not happen overnight and we will have to work hard now to make these results known and accepted.

We plan to develop the program we have written in visual basic into an easy to use tool for actuaries and others that might be interested in making unbiased estimates of these workers' compensation losses.

The program is form around an object model and can be easily expanded by defining new objects for other probability distributions, network architectures, training algorithms and so on. We would like it to be able to take a set of raw data and generate a suggested machine architecture for projecting the claims. It should do all the boring details such as rescaling and normalization with little user input. The training of the machine could then proceed automatically with the user being able to interrupt and make changes to the parameters if he or she wants to. The current program is about 40% of the way there, needing a bit too much insight to use.

The time to train a machine is quite long, taking days or even weeks on our three year old PC. But computers are already much faster and getting faster all the time. It is important to be ready to use the full calculational power available as that happens. Unfortunately much analysis is still being done using methods that date back to hand calculators or earlier.

The fact that our machine gives an estimate of the ultimate value of individual claims is something of an advantage over other methods. One often need to have that detail of information in setting rates for different classes of insureds. Normally one is forced to allocate a large amount of future development on all the claims to individual claims. That allocation is done rather crudely in comparison to our method. We expect therefore that our method will be very useful in ratemaking.

Conclusion

We have successfully demonstrated a new method of projecting workers' compensation claims into the future. The method is unbiased, which can not be said of the other standard methods. The amount of the bias in those methods was measured to be -19%, which is something of a shock.

The method also was able to produce a variance about the mean for each claim. That level of detail should be of great help in setting confidence levels to estimated outstanding losses.

Several of our results are of general interest to computer scientist and others working with neural networks.

First we have devised a decorrelation scheme which is quite robust and can decorrelate non-linearly correlated data. This method utilizes a maximum likelihood estimate of the conditional probability distribution describing the correlations. This estimate is easily learned by simple networks.

Second we have introduced a generalization measure based on the hessian building on the results in (Murata 1994). This measure does not rely on a test or validation set. Instead an estimate is made based on the variability of the training data and the hessian of the error function. This measure was tested and it gave good results.

Third we have introduced an approximation to the hessian that can speed up calculations in some

situations. This approximation is particularly useful in network pruning giving a method that is somewhere between optimal brain damage and optimal brain surgery.

Forth and last we have used an adaptive regulation procedure that allows the regulation parameter to be learned as the network is pruned. This lessens the difficulty in choosing the regularization parameter.

The above four pieces of our methodology worked well. These pieces can be used separately or together in many other neural network applications.

References

(Falhman 1992) Scott E. Fahlman and Markus Hoehfeld, “Learning with Limited Numerical Precision Using Cascade-Correlation Algorithm, ” IEEE Transactions on Neural Networks, Vol 3, No. 4 July 1992.

(Hassibi 1993) Babak Hassibi, David G. Stork, and Gregory J. Wolff, “Optimal Brain Surgeon and General Network Pruning, ” IEEE Intnl. Conference on Neural Networks Vol 1, 1993.

(Haykin 1999) Simon Haykin, “Neural Networks a Comprehensive Foundation, ” Prentice-Hall 1999.

(Hintz-Madsen 1998) Mads Hintz-Madsen, Lars Kai Hansen, Jan Larsen, Morten With Pedersen, and Micheal Larsen, “Neural classifier construction using regularization, pruning and test error estimation, ” Neural Networks 11, 1998.

(Husmeier 1998) Dirk Husmeier and John G. Taylor, “Neural Networks for Predicting Conditional Probability Densities: Improved Training Scheme Combining EM and RVFL, ” Neural Networks, 11, No. 1 1998.

(Murata 1994) Noboro Murata, Shuji Yoshizawa, and Shun-ichi Amari, “Network Information Criterion - Determining the Number of Hidden Units for an Artificial Neural Network Model, ” IEEE Transactions on Neural Networks, Vol. 5, No.6, November 1994

(Parlos 1994) Alexander G. Parlos, Benito Fernandez, Amir F. Atiya, Jayakumar Muthusami, and Wei K. Tsai, “An Accelerated Learning Algorithm for Multilayer Perceptron Networks, ” IEEE Transactions on Neural Networks, Vol 5, No. 3 May1994 .

(Prechelt 1998) Lutz Preechelt, “Automatic early stopping using cross validation: quantifying the criteria, ” Neural Networks, 11, 1998.

(Reed 1993) Russell Reed, “Pruning Algorithms - A Survey, ” IEEE Transactions on Neural Networks, Vol 4, No. 5 September 1993.

(Schittenkopf 1997) Christian Schnittenkopf, Gustavo Deco, and Wilfred Brauer, “Two Strategies to Avoid Overfitting in Feedforward Networks, ” Neural Networks, Vol 10, No. 3, 1997.

(Scarselli 1998) Franco Scarselli and Ah Chung Tsoi, “Universal Approximation Using Feedforward Neural Networks: A Survey of Some Existing Methods, and Some New Results, ” Neural Networks, Vol. 11, No. 1, 1998.

(Vapnik 1999) Vladimir N. Vapnik, “An Overview of Statistical Learning Theory, ” IEEE Transactions on Neural Networks, Vol. 10, No.5 1999.

Appendix 1 - Grouping the Data Sources

Table 2 - The different programs represent different sources of data. As each source had its own characteristic mix of claim handling practices, they had different relationships between the input and output. These differences were examined by comparing the least square linear fit to the data. Thus the sources were grouped.

		Least Square Line Reserve out Vs Reserve In			Least Square Line Payments vs. Reserve in					
Group	Program	Slope	Intercept	Correlation	Slope	Intercept	Correlation	Res. In	Res. Out	Paid
5	2	1.11	1,213	67%	0.35	5,113	51%	14,421	17,196	10,115
5	14	0.91	7,412	43%	0.33	7,056	35%	23,133	28,467	14,762
5	29	0.98	1,341	69%	0.24	3,868	49%	22,406	23,322	9,275
5	3	0.94	855	81%	0.31	164	68%	30,611	29,587	9,645
5	18	0.83	4,012	68%	0.36	1,439	60%	29,367	28,438	11,878
4	6	0.64	12,154	43%	0.26	4,872	39%	24,301	27,588	11,079
4	17	0.73	8,059	60%	0.26	4,073	44%	17,851	21,125	8,763
4	21	0.78	6,715	69%	0.22	5,761	34%	23,219	24,907	10,847
3	9	0.81	5,702	63%	0.28	3,402	47%	19,603	21,588	8,821
3	25	0.79	4,860	72%	0.40	2,754	47%	20,293	20,985	10,861
3	7	0.79	6,084	43%	0.44	6,447	44%	22,627	23,848	16,373
3	23	0.74	5,367	55%	0.35	5,144	42%	22,397	21,918	12,879
2	15	0.69	6,828	56%	0.41	4,337	52%	19,774	20,544	12,474
2	27	0.59	8,759	68%	0.38	5,362	51%	24,050	22,930	14,472
2	28	0.64	5,139	67%	0.47	4,289	58%	16,972	15,961	12,314
2	22	0.54	5,935	61%	0.51	1,587	64%	21,363	17,422	12,531
1	24	0.69	7,221	51%	0.27	3,084	36%	12,833	16,092	6,593
1	16	0.66	3,926	64%	0.37	2,400	45%	13,362	12,783	7,369
1	4	0.70	3,608	60%	0.30	3,417	37%	10,836	11,155	6,697
0	30	1.45	(7,673)	75%	0.06	7,671	15%	22,493	24,908	8,959
0	19	0.81	9,508	59%	0.16	12,225	26%	23,907	28,915	16,070
0	12	0.75	19,243	9%	0.22	5,961	24%	12,860	28,938	8,743
0	11	0.77	2,765	77%	0.15	9,282	27%	24,563	21,585	13,029
0	13	0.72	4,528	75%	0.15	8,601	34%	29,016	25,306	12,920
0	8	0.89	(448)	87%	0.13	4,820	28%	20,972	18,230	7,469
0	5	0.81	3,819	74%	0.12	3,166	26%	14,100	15,178	4,870
0	10	0.44	12,254	28%	0.07	6,141	11%	11,373	17,310	6,930
0	1	0.60	4,338	81%	0.17	7,682	31%	19,765	16,204	11,079
0	26	0.41	8,865	46%	0.33	5,736	54%	21,209	17,614	12,754

Table 3 -Here we estimate the improvement to the variance obtained by partitioning the data into these 6 groups.

		Least Square Line Reserve out Vs Reserve In		Residual Variance		Least Square Line Payments vs. Reserve in		Residual Variance				
Group 5	Slope	Intercept	Correlation	(00,000)	Slope	Intercept	Correlation	(00,000)	Count	Res. In	Res. Out	Payments
Total	0.93	4,351	59%	13,199	0.26	6,287	43%	2,560	1,347	23,216	25,854	12,439
Interval 1	0.95	883			0.31	4,724			283	27,135	26,686	13,074
Interval 2	0.86	2,893			0.26	5,563			318	22,914	22,654	11,627
Interval 3	0.95	6,196			0.25	7,108			746	21,858	26,902	12,544
Group 4	Slope	Intercept			Slope	Intercept			Count	Res. In	Res. Out	Payments
Total	0.81	5,996	66%	5,810	0.25	4,412	42%	1,996	1,312	21,241	23,247	9,620
Interval 1	0.81	3,303			0.21	4,272			263	23,279	22,249	9,260
Interval 2	0.83	5,730			0.26	5,083			271	23,520	25,350	11,265
Interval 3	0.81	6,890			0.25	4,259			778	19,758	22,851	9,169
Group 3	Slope	Intercept			Slope	Intercept			Count	Res. In	Res. Out	Payments
Total	0.77	5,126	63%	5,068	0.38	3,775	45%	3,124	1,135	21,155	21,439	11,818
Interval 1	0.77	6,336			0.28	5,252			343	20,038	21,864	10,908
Interval 2	0.77	4,926			0.35	2,818			361	23,142	22,717	10,872
Interval 3	0.77	4,290			0.48	3,591			431	20,380	20,032	13,334
Group 2	Slope	Intercept			Slope	Intercept			Count	Res. In	Res. Out	Payments
Total	0.65	6,199	60%	4,282	0.44	4,213	55%	2,471	1,010	19,290	18,822	12,629
Interval 1	0.71	5,260			0.59	452			270	19,577	19,199	12,000
Interval 2	0.71	5,861			0.35	5,565			329	20,340	20,257	12,596
Interval 3	0.55	7,389			0.43	5,150			411	18,262	17,426	13,070
Group 1	Slope	Intercept			Slope	Intercept			Count	Res. In	Res. Out	Payments
Total	0.71	4,216	61%	2,618	0.28	3,645	37%	1,545	2,649	12,349	12,943	7,151
Interval 1	0.67	3,059			0.32	2,107			790	11,474	10,783	5,805
Interval 2	0.85	1,855			0.31	2,171			750	11,230	11,432	5,622
Interval 3	0.65	6,634			0.24	5,870			1,109	13,730	15,504	9,143
Group 0	Slope	Intercept			Slope	Intercept			Count	Res. In	Res. Out	Payments
Total	0.67	7,764	52%	6,214	0.23	4,709	37%	1,627	867	19,661	20,851	9,187
Interval 1	0.45	13,825			0.13	6,981			215	19,282	22,539	9,395
Interval 2	0.77	5,084			0.39	2,260			223	21,760	21,789	10,672
Interval 3	0.70	6,302			0.19	4,781			429	18,760	19,517	8,311
ALL	Slope	Intercept			Slope	Intercept			Count	Res. In	Res. Out	Payments
Total	0.79	4,874	61%	5,825	0.30	4,344	44%	2,159	8,320	18,316	19,355	9,910
Interval 1	0.78	4,125			0.32	3,297			2,164	18,101	18,231	9,114
Interval 2	0.81	3,729			0.32	3,462			2,252	18,642	18,815	9,510
Interval 3	0.79	5,940			0.28	5,399			3,904	18,248	20,289	10,583

Improvement in variance by partitioning: 1.4% 2.0%

Appendix 2 - Generalization Error of Neural Networks

The Network Information Criteria

The network information criteria, NIC, is an approximation to the generalization error. Let the cost function be given by $D(\mathbf{w})$, then the generalization error is given by $\langle D(\mathbf{w}) \rangle_{true}$. Where the expectation value is taken over the true distribution. We can, of course, only measure an expectation over training set, T, of N points drawn from the true distribution, which we refer to as the empirical distribution.

The NIC is derived by expanding $\langle D(\mathbf{w}) \rangle_{true}$ about the true optimal weights $\mathbf{w}^*$ of the model. The assumption is then that $\mathbf{w} - \mathbf{w}^*$ is small. One must therefore be careful to train the network to as small a training error as possible before applying the NIC. This point is vital for good results.

Furthermore, besides the error due to the $\mathbf{w} - \mathbf{w}^*$, there is an error due to the replacement of the true distribution with the empirical distribution. This leads to a term in the expansion of $\langle D(\mathbf{w}) \rangle_{true}$ as shown below:

$$U(T) = \langle D(\mathbf{w}^*) \rangle_{true} - \langle D(\mathbf{w}^*) \rangle$$

Where $\langle D(\mathbf{w}^*) \rangle = (1/N) \Sigma(D(\mathbf{w}^*, j)$ denotes the empirical expectation value and $D(\mathbf{w}^*, j)$ is the value of the cost function at the j^{th} point of set T. This U is a random variable, which depends on the model selected. U takes on different values depending on the particular training set T. The average of U over an ensemble of sets T is 0. However the ensemble variance of U is of order 1/N.

In this sense, the ensemble average of the NIC is equal to the generalization error, $\langle D(\mathbf{w}) \rangle_{true}$. It is off for a single set T by an amount U(T). This amount is generally larger than the effective dimension term of the NIC. However, if one is comparing two networks of the same basic type then one can assume that the values of $D(\mathbf{w}^*)$ for the two networks are about the same. One can then ignore U when comparing these networks. This argument breaks down when one of the networks is not able to model the system sufficiently well. In that case the training error will presumably be high.

An additional factor effecting the appropriateness of using the NIC for comparisons is that the effective number of dimensions should be much less than N.

Log-Likelihood Cost Functions and the NIC

With the cost function given by $D(\mathbf{w})$, the NIC can be written as:

$$NIC = \langle D(\mathbf{w}) \rangle + tr(Var[\nabla D(\mathbf{w})] \langle \nabla\nabla D(\mathbf{w}) \rangle^{-1}) / N$$

Where $\nabla\nabla D(\mathbf{w})$ is the hessian matrix, N is the number of training points. The actual risk has, once again, been replaced with the empirical risk, so that $\langle \ldots \rangle = (1/N) \Sigma(\ldots)$. The variance of the gradient term is also taken over the training set. For the special case of a log-likelihood cost function, (also called the entropy),

$$\langle S(\mathbf{w}) \rangle = \langle -\ln [p(\mathbf{y} \mid \mathbf{x}, \mathbf{w})] \rangle$$

We can then write,

$$\langle \nabla S(\mathbf{w}) \rangle = \langle -\nabla p(\mathbf{y} \mid \mathbf{x}, \mathbf{w}) / p(\mathbf{y} \mid \mathbf{x}, \mathbf{w}) \rangle$$

and

$$\langle \nabla\nabla S(\mathbf{w}) \rangle = \langle -\nabla\nabla p(\mathbf{y}|\mathbf{x}, \mathbf{w}) / p(\mathbf{y}|\mathbf{x}, \mathbf{w}) \rangle + \langle \nabla p(\mathbf{y}|\mathbf{x}, \mathbf{w}) / p(\mathbf{y}|\mathbf{x}, \mathbf{w}) \, (\nabla p(\mathbf{y}|\mathbf{x}, \mathbf{w}) / p(\mathbf{y}|\mathbf{x}, \mathbf{w}))^T \rangle$$

But taking only the first term on the right hand side for now,

$$-1^{st} \text{ term} = \langle \nabla\nabla p(\mathbf{y}|\mathbf{x}, \mathbf{w}) / p(\mathbf{y}|\mathbf{x}, \mathbf{w}) \rangle$$

$$= \int p(\mathbf{x}, \mathbf{y}) \, \nabla\nabla p(\mathbf{y}|\mathbf{x}, \mathbf{w}) / p(\mathbf{y}|\mathbf{x}, \mathbf{w}) \, d\mathbf{x}d\mathbf{y}$$

$$= \int p(\mathbf{x}) \, [p(\mathbf{y}|\mathbf{x}) / p(\mathbf{y}|\mathbf{x}, \mathbf{w})] \, \nabla\nabla p(\mathbf{y}|\mathbf{x}, \mathbf{w}) \, d\mathbf{x}d\mathbf{y}, \quad (\text{but } [p(\mathbf{y}|\mathbf{x}) / p(\mathbf{y}|\mathbf{x}, \mathbf{w})] \cong 1 \text{ so})$$

$$\cong \nabla\nabla \int p(\mathbf{x}) \, p(\mathbf{y}|\mathbf{x}, \mathbf{w}) \, d\mathbf{x}d\mathbf{y} = \nabla\nabla(1) = 0$$

So for log-likelihood we find,

$$\langle \nabla\nabla S(\mathbf{w}) \rangle = \langle \nabla S(\mathbf{w}) \, \nabla S(\mathbf{w})^T \rangle$$

Also if the network is trained to a local minimum, the expected value of the gradient vanishes so we find that the hessian is equal to the variance of the gradient. Thus for this special case

$$\text{NIC} = \langle S(\mathbf{w}) \rangle + \text{tr}(I) / N$$

$$= \langle S(\mathbf{w}) \rangle + M / N$$

Where M = the number of weights.

If now we introduce a regularization term as in (Hintz-Madsen 1998),

$$\langle D(\mathbf{w}) \rangle = \langle S(\mathbf{w}) \rangle + (1/2)\mathbf{w}R\mathbf{w}$$

where R is the regularization matrix. The same reasoning as above leads us to.

$$\langle \nabla\nabla D(\mathbf{w}) \rangle = \langle \nabla S(\mathbf{w}) \, \nabla S(\mathbf{w})^T \rangle + R$$

$$\text{Var}[\nabla D(\mathbf{w})] = \langle \nabla S(\mathbf{w}) \, \nabla S(\mathbf{w})^T \rangle - \langle \nabla S(\mathbf{w}) \rangle \langle \nabla S(\mathbf{w})^T \rangle$$

Unfortunately the expected gradient longer vanishes when the cost function is minimized, being equal to (-R**w**). This fact is not mentioned in (Hintz-Madsen 1998) and may not be significant if R is chosen small enough. In particular one can say that they have kept the first order terms in R and ignored the rest.

Approximation of the Hessian for Log-Likelihood Cost Functions

We let the cost function S(**w**) be a log-likelihood as in the previous section. Then we can write,

$$\langle \nabla\nabla S(\mathbf{w}) \rangle = \langle \nabla S(\mathbf{w}) \, \nabla S(\mathbf{w})^T \rangle$$

$$= \langle \nabla S(\mathbf{w}) \rangle \langle \nabla S(\mathbf{w})^T \rangle + \text{Var}[\nabla S(\mathbf{w})]$$

We note that the first term above is obtained by the back-prop algorithm. It is also equal to 0 if we are at a minimum of the cost function. With a regulation term added to S it will no longer vanish. As the numbers needed to calculate the $\langle \nabla S(\mathbf{w}) \rangle$ are available anyway, we always include it in our estimates. The second term, however, is rather difficult to compute as it has in principle M^2 terms to be computed at each data point. If we now focus on just one of those terms we find

$$V_{nn'} = \langle a^{(k)}_i \, a^{(k')}_{i'} \, \delta^{(k+1)}_j \, \delta^{(k'+1)}_{j'} \rangle - \langle a^{(k)}_i \, \delta^{(k+1)}_j \rangle \langle a^{(k')}_{i'} \, \delta^{(k'+1)}_{j'} \rangle$$

Where $V_{nn'}$ is the nn'^{th} element of $\mathrm{Var}[\nabla S(\mathbf{w})]$, $a^{(k)}_i$ is the activation of the i^{th} node of layer k and $\delta^{(k+1)}_j$ is the back propagated error on the j^{th} node of layer k+1. We reserve the index i=0 to refer to the bias for each layer. The index n corresponds to the $w^{(k)}_{ij}$ derivative. Now one can approximate V to varying degrees. One could set it to 0, assume it to be diagonal or assume it to be block diagonal. We have chosen the block diagonal approximation to V. In particular we assume that,

$$V_{nn'} = \mathrm{Covar}[a^{(k)}_i \, \delta^{(k+1)}_j \, , \, a^{(k')}_{i'} \, \delta^{(k'+1)}_{j'}] = 0 \text{ for } i \neq i', \, k \neq k'$$

That is to say that the output activations of different nodes are 'uncorrelated' in the sense that the above covariance vanishes. One could easily find other approximations, such as block diagonal by layers, just $k \neq k'$ above, or calculating the covariance all for the j=j' terms also.

For our purposes the above approximation is good enough. In particular, a small error in the saliency used for pruning decisions can be tolerated while still make a sensible pruning decision. We always had the exact hessian calculated when we start a set of prunings, as we needed to calculate it to get the effective dimension of the network. Thus we always prune the first weight using no approximation. It is when pruning more than one weight at a time that we use the approximate hessian for the second and later weights.

In calculating the updated weights after pruning one is more sensitive to errors in the hessian matrix. However we found that the updated weights using our approximation gave better training errors at the start of the retraining than not updating at all. It, in fact, made the whole pruning and regulation learning possible in reasonable time.

It is worth noting that we first tried using the approximate hessian to calculate the NIC and effective dimension in our pruning and adaptive regularization procedure. We found that for large networks with appreciable redundancy the approximation to the effective dimension was rather poor but as the redundancies were removed through pruning the approximation became better and better. This can be seen in figure 19 which shows the effective dimension and the E(R) calculated using the approximate hessian and the exact hessian for a network that was pruned and regulated using only the approximate hessian. We afterwards decided to redo the pruning procedure using the exact hessian to calculate the effective dimension. That the approximation becomes better as the redundancies are removed is easily understood since the redundancy will lead to correlations between the node's outputs. These correlations then make our approximation invalid.

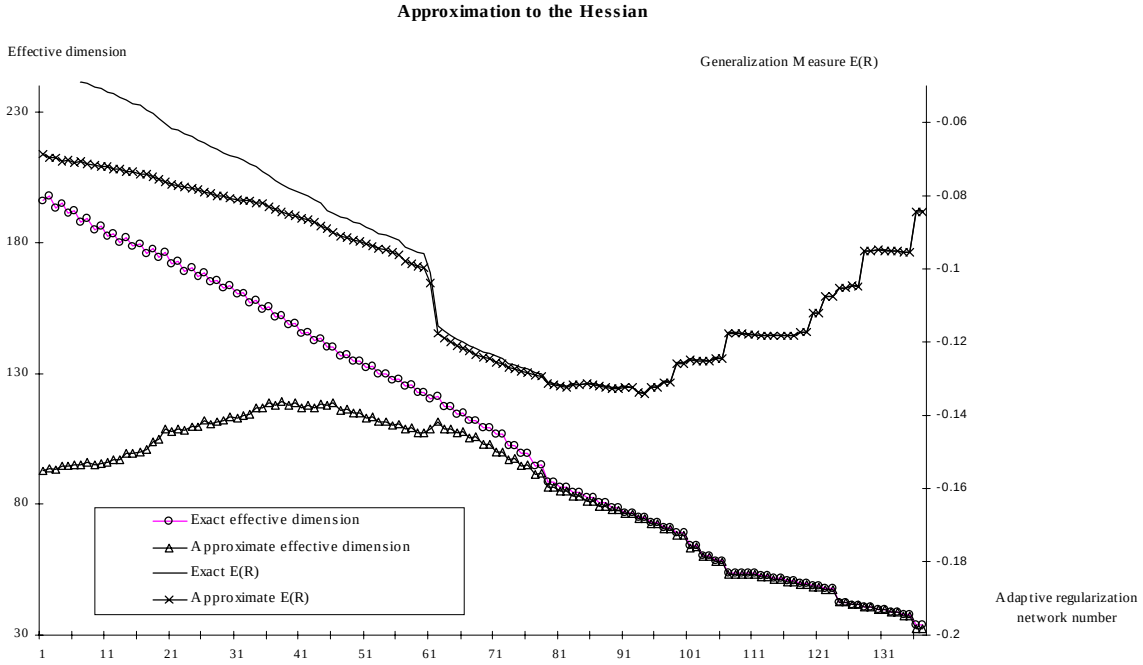

Figure 19. The effect of the approximation to the hessian on the generalization measure.

Approximation of the Hessian for Mean Square Error Cost Functions

We let C(**w**) be a mean square error cost function with one output variable.

$\langle C(\mathbf{w})\rangle = (1/2)\,\langle (y(\mathbf{w})-d)^2\rangle$

We have written the output of the network as y(**w**) and the target as d.

$\langle \nabla\nabla C(\mathbf{w})\rangle = \langle (y(\mathbf{w})-d)\,\nabla\nabla y(\mathbf{w})\rangle + \langle \nabla y(\mathbf{w})\,(\nabla y(\mathbf{w}))^T\rangle$

Now if y(**w**) is a good approximation then the 1st term should be small and we shall neglect it. The second term is similar to what we had in the previous section.

$\langle \nabla\nabla C(\mathbf{w})\rangle \cong \langle \nabla y(\mathbf{w})\,(\nabla y(\mathbf{w}))^T\rangle = \langle \nabla y(\mathbf{w})\rangle\,\langle \nabla y(\mathbf{w})^T\rangle + \mathrm{Var}[\nabla y(\mathbf{w})]$

and writing Var[∇y(**w**)] as simple V,

$V_{nn'} = \langle a^{(k)}_i\, a^{(k')}_{i'}\, \delta^{(k+1)}_j\, \delta^{(k'+1)}_{j'}\rangle - \langle a^{(k)}_i\, \delta^{(k+1)}_j\rangle\,\langle a^{(k')}_{i'}\, \delta^{(k'+1)}_{j'}\rangle$

Here the δ are from propagating an error of 1 on the output back through the net. We again assume this to be of the form

$V_{nn'} = \text{Covar}[a^{(k)}_i \delta^{(k+1)}_j , a^{(k')}_{i'} \delta^{(k'+1)}_{j'}] = 0$ for $i \neq i'$, $k \neq k'$

$\langle \nabla\nabla C(\mathbf{w}) \rangle_{nn'} \cong \langle \nabla y(\mathbf{w}) \rangle_n \langle \nabla y(\mathbf{w})^T \rangle_{n'} + V_{nn'}$

If there are several outputs then the hessian will be a sum of terms like the one above, one for each output. Calculating this would then require propagating an error of one back from each output in turn, one at a time.

$\langle C(\mathbf{w}) \rangle = \sum \langle C_t(\mathbf{w}) \rangle$

$\langle \nabla\nabla C(\mathbf{w}) \rangle_{nn'} = \sum \langle \nabla\nabla C_t(\mathbf{w}) \rangle_{nn'}$

$\cong \sum [\langle \nabla y_t(\mathbf{w}) \rangle_n \langle \nabla y_t(\mathbf{w})^T \rangle_{n'} + V_{t,\,nn}]$

Here the sum is over t the index of the outputs.

The NIC for Mean Square Error Cost Functions

Continuing from the one output cost function of the previous section, for the NIC, in addition to the hessian we need the $\text{Var}[\nabla C(\mathbf{w})]$.

$\text{Var}[\nabla C(\mathbf{w})] = \langle (y(\mathbf{w})-d)^2 \nabla y(\mathbf{w}) (\nabla y(\mathbf{w}))^T \rangle - \langle (y(\mathbf{w})-d) \nabla y(\mathbf{w}) \rangle \langle (y(\mathbf{w})-d) (\nabla y(\mathbf{w}))^T \rangle$

$= \langle (y(\mathbf{w})-d)^2 \rangle \langle \nabla y(\mathbf{w}) (\nabla y(\mathbf{w}))^T \rangle + \text{Cov}[(y(\mathbf{w})-d)^2, \nabla y(\mathbf{w}) (\nabla y(\mathbf{w}))^T] - \langle (y(\mathbf{w})-d) \nabla y(\mathbf{w}) \rangle \langle (y(\mathbf{w})-d) \nabla y(\mathbf{w}) \rangle^T$

$= \langle (y(\mathbf{w})-d)^2 \rangle \langle \nabla\nabla C(\mathbf{w}) \rangle + \text{Cov}[(y(\mathbf{w})-d)^2, \nabla y(\mathbf{w}) (\nabla y(\mathbf{w}))^T] - \langle \nabla C(\mathbf{w}) \rangle \langle \nabla C(\mathbf{w}) \rangle^T$

We might in certain instances be able to assume that the covariance term above is negligible. We also know that the errors $(y(\mathbf{w})-d)$ are uncorrelated with the $\nabla y(\mathbf{w})$ (that is to say we are at a local minimum of C). So,

$\text{Var}[\nabla C(\mathbf{w})] = \langle (y(\mathbf{w})-d)^2 \rangle \langle \nabla\nabla C(\mathbf{w}) \rangle$, provided that $\langle \nabla C(\mathbf{w}) \rangle = 0$ and $\text{Cov}[(y(\mathbf{w})-d)^2, \nabla y(\mathbf{w}) (\nabla y(\mathbf{w}))^T] = 0$.

If we plug this then into the formula for the effective dimension it will become the number of weights times the variance in the errors. We will however have more than one output, add a regulation term to the error function of the form $(1/2)\mathbf{w}^T R \mathbf{w}$ and not make the above assumptions. This changes things considerably. Specifically,

$\langle D(\mathbf{w}) \rangle = \langle C(\mathbf{w}) \rangle + (1/2)\mathbf{w}^T R \mathbf{w}$

where C is now the sum of several square errors terms.

$\text{NIC} = \langle D(\mathbf{w}) \rangle + \text{tr}(\text{var}[\nabla D(\mathbf{w})] \langle \nabla\nabla D(\mathbf{w}) \rangle^{-1}) / N$

$\text{NIC} = \langle D(\mathbf{w}) \rangle + \text{tr}(\text{var}[\nabla C(\mathbf{w})] \langle \nabla\nabla C(\mathbf{w}) + R \rangle^{-1}) / N$

$\text{Var}[\nabla C(\mathbf{w})] = \sum [\langle \nabla C_t(\mathbf{w}) \nabla C_{t'}(\mathbf{w})^T \rangle - \langle \nabla C_t(\mathbf{w}) \rangle \langle \nabla C_{t'}(\mathbf{w}) \rangle^T]$

$= \sum [\langle (y_t(\mathbf{w})-d_t) a^{(k)}_i \delta(t)^{(k+1)}_j (y_{t'}(\mathbf{w})-d_{t'}) a^{(k')}_{i'} \delta(t')^{(k'+1)}_{j'} \rangle$

$- \langle (y_t(\mathbf{w})-d_t) a^{(k)}_i \delta(t)^{(k+1)}_j \rangle \langle (y_{t'}(\mathbf{w})-d_{t'}) a^{(k')}_{i'} \delta(t')^{(k'+1)}_{j'} \rangle]$

The sum is over t and t', the subscripts of the output vectors. Here the $\delta(t)$ are gotten by propagating an error of 1 on output t back through the network. Note that the second term vanishes for R=0, but that is a minor simplification really.

Calculating the NIC for mean square error networks is significantly more time consuming than for log-likelihood networks. Furthermore, we found that the above approximation did not work well enough for these networks. We present this it in hopes that it may prove useful in other contexts. The approximate hessian was useful for pruning.

Inverting the Hessian Matrix

In the previous sections we derived approximations to the hessian matrix of the form:

$$\langle \nabla\nabla D(\mathbf{w}) \rangle_{nn'} \cong H_{nn'} \equiv G_n G_{n'} + V_{nn'}$$

Where G_n is a vector with M elements and $V_{nn'}$ is a MxM block diagonal matrix. We can then use a matrix inversion lemma known as Woodbury's equality. This gives us:

$$H^{-1} = V^{-1} - (V^{-1} G G^T V^{-1}) / d$$

Where $d = [1 + G^T V^{-1} G]$ is a scalar.

If V is block diagonal its inversion is much easier than inverting H. The above formula is then a manageable approach to the inversion of H. We found it to be stable for our applications.

For the case with a multiple output mean square error cost function, we must apply Woodbury's equality repeatedly, once for each output. The starting V is the sum of the V's for each output. Then the G's for each output are taken in turn.

Changing the regulation parameter

We have a need to change the regulation parameter and then re-train the network. In order to reduce the re-training times we have devised an update rule for the new weights after changing the regulation parameter based on the second order expansion of the training error. Specifically, the training error is,

$$\langle D_R(\mathbf{w}) \rangle = \langle S(\mathbf{w}) \rangle + (1/2)\mathbf{w}^T R \mathbf{w}$$

$$\cong \langle S(\mathbf{w_0}) \rangle + (1/2)\mathbf{w_0}^T R_0 \mathbf{w_0} + [\langle \nabla S(\mathbf{w_0}) \rangle^T + \mathbf{w_0}^T R_0]\Delta\mathbf{w} + (1/2)\,\Delta\mathbf{w}^T [\langle \nabla\nabla S(\mathbf{w_0}) \rangle + R_0]\,\Delta\mathbf{w}$$

$$+ (1/2)\mathbf{w_0}^T(\Delta R)\mathbf{w_0} + \mathbf{w_0}^T(\Delta R)\Delta\mathbf{w} + (1/2)\,\Delta\mathbf{w}^T(\Delta R)\Delta\mathbf{w}$$

Where we have written R as $R_0 + \Delta R$, and $\mathbf{w}$ as $\mathbf{w_0} + \Delta\mathbf{w}$. If $\mathbf{w_0}$ is a local minimum for R_0 then

$$\langle \nabla S(\mathbf{w_0}) \rangle^T + \mathbf{w_0}^T R_0 = 0.$$

So,

$$\langle D_R(\mathbf{w}) \rangle \cong \langle D_R(\mathbf{w_0}) \rangle + (1/2)\mathbf{w_0}^T(\Delta R)\mathbf{w_0} + \Delta\mathbf{w}^T(\Delta R)\,\mathbf{w_0} + (1/2)\,\Delta\mathbf{w}^T [\langle \nabla\nabla S(\mathbf{w_0}) \rangle + R_0 + \Delta R)]\,\Delta\mathbf{w}$$

For a given ΔR, minimizing the above gives us a $\Delta \mathbf{w}$ of,

$$\Delta \mathbf{w} = - [\langle \nabla\nabla S(\mathbf{w_0}) \rangle + R_0 + \Delta R)]^{-1} (\Delta R)\, \mathbf{w_0} = - \langle \nabla\nabla D_R(\mathbf{w_0})) \rangle^{-1} (\Delta R)\, \mathbf{w_0}$$

We use this after every change of regulation parameter.

Appendix 3 - The Weibull Distribution

We have made extensive use of the weibull distribution after finding that it gave a good fit to our data. The weibull density function looks like,

$$wb(\beta, \lambda, x) = (\beta /x)(\lambda x)^{\beta} \exp(-(\lambda x)^{\beta}) \quad x > 0;$$

It has the required property of being one-sided. It also can take on a bell like shape for $\beta > 1$. For $\beta = 1$ it is the exponential distribution. And for $\beta < 1$ it becomes more and more L shaped, being large near zero and having a long flat tail. Thus it has the ability to approximate a variety of distributions.

The cumulative probability function F(x) has the simple form,

$$F(\beta, \lambda, x) = 1-\exp(-(\lambda x)^{\beta}) \quad x > 0;$$

The mean and variance are given by,

$$\mu = (1/\lambda)\Gamma(1+1/\beta)$$

$$\sigma^2 = (1/\lambda)^2\, \Gamma(1+2/\beta) - \mu^2$$