Obtaining the correspondence between Bayesian and Neural Networks

A. Stassopoulou & M. Petrou
Department of Electronic and Electrical Engineering
University of Surrey
Guildford, Surrey GU2 5XH, U.K.

Running head: Correspondence between Bayesian and Neural Networks.

Abstract

We present in this paper a novel method for eliciting the conditional probability matrices needed for a Bayesian network with the help of a neural network. We demonstrate how we can obtain a correspondence between the two networks by deriving a closed-form solution so that the outputs of the two networks are similar in the least square error sense, not only when determining the discriminant function, but for the full range of their outputs. For this purpose we take into consideration the probability density functions of the independent variables of the problem when we compute the least square error approximation. Our methodology is demonstrated with the help of some real data concerning the problem of risk of desertification assessment for some burned forests in Attica, Greece where the parameters of the Bayesian network constructed for this task are successfully estimated given a neural network trained with a set of data.

Keywords: Bayesian networks, Neural networks, conditional probability matrices, least square approximation, risk assessment.

1 Introduction

Bayesian and neural networks (in particular multi-layer perceptrons) are often used in many applications. Bayesian networks have many advantages[6,7]:

- They are bidirectional, allowing the flow of information from causes to effects and from effects to causes.
- They allow input data to be inserted at any one of their nodes.
- They can cope with incomplete and uncertain data.
- They can cope with uncertain rules of reasoning.
- They assign degrees of confidence in the resultant classifications.

To achieve all these, the networks make use of probability theory and prior knowledge expressed in the form of conditional probabilities between causes and effects. The elicitation of these conditional probabilities is the weakest point of these networks. In this paper we are solving this problem by making use of the major advantage neural networks have, namely their ability to be trained, often with the help of a small number of data (e.g Duin[4]).

In particular, we find a mathematical function which relates the parameters used by a neural network with the parameters of the output function of a Bayesian network. Effectively a closed-form solution is obtained for determining the free parameters of a neural network which optimally approximates (in the least square error sense) a given Bayesian network. The advantage of our method is that no training is required in order to obtain the weights of the neural network and moreover we theoretically prove that there exists a correspondence between the two networks.

Another important contribution of the mathematical function obtained is that we can inversely determine the conditional probability matrix elements of the Bayesian network given the weights obtained by training the neural network with sufficient training patterns.

The basic idea of this work was presented in brief in Stassopoulou et al[9]. Here, however, we expand upon it by solving the correspondence equations between the two networks taking into consideration the density of the training patterns.

Our methodology is demonstrated in conjunction with the problem of assessing the risk of desertification of burned forests in Attica, Greece.

2 Motivation

The output of a Pearl-Bayes network consists of a linear superposition of the confidences in the values of the conditioning variables and multiplicative products of them, all weighted with the appropriate combinations of the elements of the conditional probability matrix. One can linearize this function in terms of all the variables by creating extra input nodes in the network, one for each non-linear combination of the confidences of the conditioning variables that appear in the output function. Thus, a function of the form $f(x, y) = a_1x + a_2y + a_3xy$ can be thought of as linear in terms of x, y and z where $z \equiv xy$. The output of a perceptron on the other hand is given by the linear superposition of all the input variables to the output node, fed into a decision function, usually a sigmoid. The use of a sigmoid, however, is not necessary. One can use instead, the identity function as a decision function. Then there is a direct correspondence between the linearized Pearl-Bayes network and the perceptron architecture, and one can easily work out the relationship between the weights of the neural network and the elements of the conditional probability matrix of the Pearl-Bayes network.

Once we have established such a correspondence, it is easy to consider that we can use a set of training data to train the perceptron with the identity decision function and from the correspondence between the two networks infer the elements of the conditional probability matrix of the Bayesian network.

This idea however, although simple in its conception, does not work in practice when the problem we are tackling is that of an expert inference system. In a Bayesian network we handle uncertainty in the inference rules through the probabilities given in the conditional probability matrix. The inference rules are usually supplied by an expert and have the form “IF..AND..THEN..”. The experts very seldomly put confidences in the verdict of a rule and if they do, this is not quantitative. The only way to judge the validity of a rule is to make a historical study of input data and output results to compare the resultant classification with what actually happened in reality. Such an example is a rule about erosion, which is an issue addressed in the application section of this paper: “IF slope is steep AND rocks are permeable AND the soil depth is zero THEN the risk of erosion is high”. To check how good this rule is, we must do statistics on a set of regions which, for example, some years earlier were classified by an expert as running high risk of erosion, and compare this prediction with the state of those regions today. This cannot be easily done and thus we have to handle the situation in a different way.

It is clear that the classification an expert does, with the help of a set of rules he/she uses, is really a hard classification. Thus, it is best represented by a sigmoid function which is a function that allows a softened up hard classification. Motivated, therefore, by the need to handle the expert rules in our particular application, we had to use as an output function of the neural network implemented, a sigmoid rather than the identity function. This means that is is not easy to find the correspondence between the conditional probability matrix of the Bayesian network and the weights of the perceptron.

What follows is a detailed description of how we obtained this correspondence. For simplicity but without loss of generality, we demonstrate first the approach using simple network structures.

3 Problem Formulation

In this section we present the two network models in order to establish our notation. We describe first a small Bayesian network consisting of three two valued variables and then present the neural network constructed to represent this Bayesian network.

3.1 Bayesian Network

Assume that we have a Bayesian network with 3 two-valued variables labeled X, Y and Z shown in figure 1. Let us call the two possible values of each variable low and $high$.

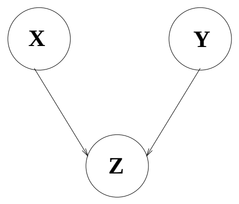


Figure 1: A Bayesian network with two-valued variables

Assume that the conditional probability matrix M, which relates Z to X and Y, is given by:

$$M = \begin{pmatrix} m_1 & 1-m_1 \\ m_2 & 1-m_2 \\ m_3 & 1-m_3 \\ m_4 & 1-m_4 \end{pmatrix}$$

The elements of this matrix are the values of $P(Z|X,Y)$ that correspond to the following cases:

$$M = \begin{pmatrix} P(Z=low|X=low,Y=low) & P(Z=high|X=low,Y=low) \\ P(Z=low|X=low,Y=high) & P(Z=high|X=low,Y=high) \\ P(Z=low|X=high,Y=low) & P(Z=high|X=high,Y=low) \\ P(Z=low|X=high,Y=high) & P(Z=high|X=high,Y=high) \end{pmatrix}$$

Then the beliefs in the first and second state of Z are equal to:

$$BEL(Z=low) = m_1AC + m_2AD + m_3BC + m_4BD \qquad (1)$$

and

$$BEL(Z=high) = (1-m_1)AC + (1-m_2)AD + (1-m_3)BC + (1-m_4)BD$$

where we denote by A and B the elements of the belief vector of X (i.e. $BEL(X = low) = A$ and $BEL(X=high) = B$) and by C and D the elements of the belief vector of Y.

Consider the belief in the first state of node Z only, i.e. $BEL(Z = low)$. By setting $B = 1-A$ and $D = 1-C$, equation (1) becomes:

$$BEL(Z=low) = (m_1 - m2 - m3 + m4)AC + (m_2 - m4)A + (m_3 - m4)C + m_4 \quad (2)$$

A and C range from 0 to 1 since they represent probabilities. It can be proven that in this range of input variables, the Bayesian output is bounded (see Appendix A).

Furthermore it can be shown that the function achieves its minimum and maximum values (min and max respectively) at two of the four corners of the square defined by the A and C coordinates (see Appendix A). The values at these four corners do in fact represent the four independent entries of the conditional probability matrix i.e. m_1, m_2, m_3 and m_4. We can therefore find the bounds of the Bayesian output function by obtaining simply the min and max values of the matrix elements: m_1, m_2, m_3 and m_4. The significance of being able to obtain these bounds is justified in section 4.

3.2 Neural Network constructed

Assume now that we have a neural network consisting of two input units labeled X_1 and Y_1, two units in the hidden layer (E and F) and one unit in the output layer (Z_1) as shown in figure 2.

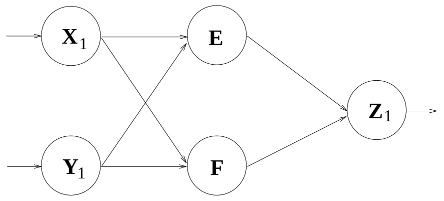


Figure 2: A neural network constructed

This neural network has been chosen to model the Bayesian network of figure 1. Each root is represented by one unit in the input layer (since the second is a complement in the case of two-valued variables). The units X_1 and Y_1 in the neural network represent the first states of nodes X and Y in the Bayesian network and Z_1 represents the first state of Z. Thus, the input values A and C are fed into units X_1 and Y_1 respectively. The units E and F do not correspond to any node in the Bayesian network.

The output of unit Z_1 is found using the following procedure: First units E and F will perform a weighted sum of their input values which will be given by:

$$S_E = AW_{X_1E} + CW_{Y_1E} \tag{3}$$

and

$$S_F = AW_{X_1F} + CW_{Y_1F} \tag{4}$$

where the notation W_{ij} means the weight of the connection from i to j. After these sums are computed, the sigmoid function is used to compute the outputs of the hidden units. These are given by:

$$f_E = \frac{1}{1 + e^{-S_E - t_1}} \tag{5}$$

and

$$f_F = \frac{1}{1 + e^{-S_F - t_2}} \tag{6}$$

where t_1 and t_2 are some thresholds.

Now, f_E and f_F will be the input values to Z_1 and the weighted sum of Z_1 will be given by:

$$S_{Z_1} = f_E W_{EZ_1} + f_F W_{FZ_1} = \frac{W_{EZ_1}}{1 + e^{-S_E - t_1}} + \frac{W_{FZ_1}}{1 + e^{-S_F - t_2}} \tag{7}$$

Finally, the output of unit Z_1 is given by:

$$f_{Z_1} = \frac{1}{1 + e^{-S_{Z_1} - t_3}} \tag{8}$$

where t_3 is again a threshold parameter.

The objective is to determine the weights and thresholds in terms of the conditional probability matrix elements so that the Bayesian network and the neural network give the same output.

4 The basic idea for obtaining the correspondence

We first find the correspondence of a special case Bayesian network in which the output function is linear in X_1 and Y_1 and then we consider the general case of the quadratic Bayesian output function, as the one given by equation 2.

4.1 Linear output case

Assume that the Bayesian network was constructed in such a way that the output function was linear in A and C. For this to happen the coefficient of the AC term in equation 2 must be zero. In other words, the following restriction on the conditional probability matrix elements must apply:

$$m_1 = m2 + m3 - m4 \tag{9}$$

Then, the resulting Bayesian output function becomes:

$$f_B = (m_2 - m4)A + (m_3 - m4)C + m_4 \tag{10}$$

It is well known[2,1] that a neural network with no hidden layer is capable of implementing the above linear function.

Therefore the neural network output function reduces to one with the same number of degrees of freedom (i.e. we are free to choose values for $W_{X_1Z_1}$, $W_{Y_1Z_1}$ and t) as the Bayesian output function (where we can choose the values of m_2, m_3 and m_4) due to the linearity condition. So, the problem reduces to finding a way so that the neural network output:

$$f_N = \frac{1}{1+e^{-x}} \tag{11}$$

where $x = W_{X_1Z_1}A + W_{Y_1Z_1}C + t$ behaves like the Bayesian network output, i.e. linearly, in the specified range $[min, max]$ of the Bayesian network output.

A lot of research in the past has been directed towards approximating the discriminant function that separates the two classes $Z = low$ and $Z = high$ in the feature space. The optimal discriminant function between two classes is that which minimizes the Bayesian classification error[3]. It is known that a multilayer perceptron can approximate this discriminant function by piecewise linear segments[5]. The Bayesian network however, is usually used for probabilistic reasoning and fusion of information coming from uncertain sources. The purpose of using it is not only to assign a class to the output variable, but also to have a reliable estimate of the confidence with which this class is assigned. If we were only interested in the two networks (the neural and the Bayesian) to get the hard classification error right, we would have to try to match their behaviour as closely as possible near the threshold value (i.e. near the value $S_{Z_1} + t_3 = 0$ which is the discriminant function). As we are interested, however, in the overall agreement between the two networks, we must choose a criterion of correspondence that reflects this requirement.

This can be achieved by linearizing the sigmoid function $y = 1/(1 + e^{-x})$ over the range $[min, max]$ of its possible values, with $x = W_{X_1Z_1}A + W_{Y_1Z_1}C + t$.

The line of approximating the sigmoid function over the finite $[min, max]$ range can be chosen, so that the total least square error of the approximation over the full range of the values of x is minimized.

From the correspondence of the parameters of the linearized sigmoid function and the linear Bayesian output function, we can derive a set of equations that relate the parameters of the neural network, with those of the Bayesian.

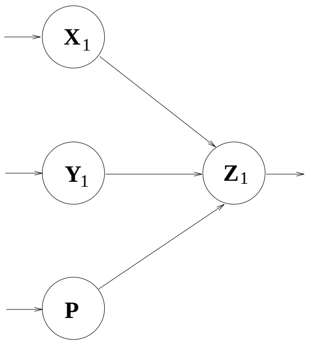


Figure 3: The new product input node is introduced

4.2 Generalised case

Assume now that the conditional probability matrix is such that the restriction of linearity, given by equation 9, does not hold. In fact, the Bayesian network output is a function with four degrees of freedom given by:

$$f_B = (m_1 - m_2 - m_3 + m_4)AC + (m_2 - m4)A + (m_3 - m4)C + m_4 \tag{12}$$

We therefore have a quadratic function in terms of A and C which cannot be implemented by the simple perceptron with no hidden layers. However, an architecture with one hidden layer as the one shown in figure 2 would result a neural output function with nine degrees of freedom compared with the Bayesian output function of four degrees. This difference in the number of degrees of freedom is a serious drawback when a direct correspondence has to be obtained. This problem, however, can be overcome by reducing it to the previous case of no hidden layer. For this reason we introduce the cross product term AC as a new feature alongside A and C. This new feature will be calculated by a special processor. We therefore end up with the Bayesian output being a linear function in terms of AC, A and C which can easily be implemented with a neural network of the same architecture as before, as shown in figure 3 where the additional unit in the input layer, labeled P, represents the product of the values A and C of the input units X_1 and Y_1 respectively.

This way the neural network output has the same number of degrees of freedom as the Bayesian output i.e. four. By following the same method as the one adopted in the previous section, we can determine the line that best fits the sigmoid in the finite range of its possible output values determined by the conditional probability matrix elements. Then we can obtain a direct correspondence with the Bayesian output of equation 12, from which, by equating the coefficients, we can obtain the relationship between the parameters of the two networks.

5 Defining and solving the correspondence problem

The linearization of the sigmoid function has to happen over the whole range of its definition. For optimal performance, we shall define first an over all error and then choose the linearization parameters so that this error is minimized. The independent variables of the problem are A and C and ideally, we would like to minimize the square difference between the output of the Bayesian and the neural networks over all combinations of values of A and C. If we call the square difference in the two outputs $F(A, C)$, then clearly, the error function we want to minimize is given by:

$$f = \int_0^1 \int_0^1 F(A, C) dAdC \tag{13}$$

where

$$F(A, C) \equiv [\frac{1}{1 + e^{-x(A,C)}} - (sx(A, C) + \delta)]^2 \tag{14}$$

$x(A, C) \equiv W_{PZ_1} AC + W_{X_1Z_1} A + W_{Y_1Z_1} C + t$, and σ and δ are the parameters of the line that approximates the sigmoid function.

Upon changing variables of integration from (A, C) to (A, x), we obtain:

$$f = \int_0^1 \int_{x_1(A)}^{x_2(A)} F(x) \frac{\partial(A, C)}{\partial(A, x)} dxdA \tag{15}$$

where $x_1(A)$ and $x_2(A)$ are the limits of x, and the Jacobean is given by:

$$\frac{\partial(A, C)}{\partial(A, x)} = \frac{1}{W_{PZ_1} A + W_{Y_1Z_1}} \tag{16}$$

It turns out that the analytic calculation of this integral is not possible. A more convenient form then for its numerical evaluation turns out to be:

$$f = \int_{xmin}^{xmax} F(x) \int_{A_1(x)}^{A_2(x)} \frac{\partial(A, C)}{\partial(A, x)} dAdx \tag{17}$$

where $A_1(x)$ and $A_2(x)$ are the limits of A which are functions of x, and $xmin$ and $xmax$ are the values of x for which the sigmoid function obtains its min and max values respectively.

We define

$$p(x) \equiv \int_{A_1(x)}^{A_2(x)} \frac{1}{W_{PZ_1}A + W_{Y_1Z_1}} dA \tag{18}$$

The total error function then is given by:

$$f = \int_{xmin}^{xmax} p(x)[\frac{1}{1+e^{-x}} - (sx + \delta)]^2 dx \tag{19}$$

In this expression $p(x)$ plays the role of the importance of each error according to the number of combinations of values A and C that can give rise to the particular x value. Thus, when we choose the approximating linear function by minimizing the error, more importance will be given to errors that are expected to arise more frequently than less frequent errors. To illustrate this point, assume that we have a Bayesian network as above with the independent entries of the matrix being $m_1 = 0.2$, $m_2 = 0.7$, $m_3 = 0.25$ and $m_4 = 0.4$. Figure 4 shows the output of the Bayesian network as a function of A and C which are measured along the two axes. Each band of grey represents a particular range of output values. For example, the black band starting on the top right corner gives all the A-C combinations which give an output between 0.2 and 0.3. Continuing with steps of 0.1 we reach the bottom left corner which has an output value of 0.7. From this plot we can clearly see that some outputs (and hence some x's due to the one-to-one mapping) are given by a greater number of combinations of values of A's and C's than others.

The integral of equation 18 can easily be calculated analytically, but special care should be taken for its limits. Figure 5 shows diagrammatically the area of integration. The relative orientation of the limiting lines $x_1(A) = 0$ and $x_2(A) = 0$ depends on the weights of the neural network. Thus, the exact shape of the area over which we have to integrate depends on the relative location along the x axis of points P_1, P_2, P_3 and P_4. In general there are 24 possible sequences by which these points may appear along the x axis and each of them would correspond to a different formula for $p(x)$. Even if $p(x)$ was given in its closed form, integral 19 (or its first derivatives with respect to s and δ) could not be calculated analytically. So, as one has to resort to numerical methods anyway, instead of giving the analytic evaluation of integral 18, it would be more instructive and more relevant to the general case, to use a numeric approach from the beginning.

In the following two subsections, we shall describe how equation 19 should be used for the case when one knows the elements of the conditional probability matrix and wants to derive the weights of the neural network, and for the case when one knows the weights of the neural network and wants to derive the elements of the conditional probability matrix of the Bayesian network.

It can be shown (see Appendix B) that in this general case s and δ are given by:

$$s = \frac{S_3 - \delta S_2}{S_1} \tag{20}$$

and

$$\delta = \frac{S_1 S_5 - S_3 S_2}{S_1 S_4 - S_2^2} \tag{21}$$

where $S_1, ..., S_5$ are as defined in Appendix B.

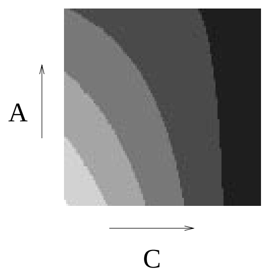


Figure 4: Contours of constant values of f_B (given by eq.12) as functions of the independent variables A and C

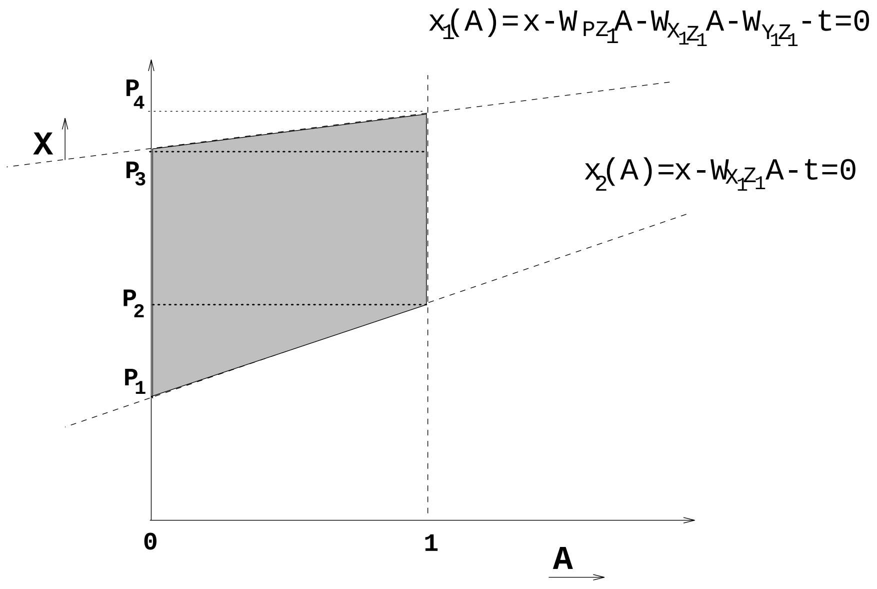


Figure 5: The area representing the term $p(x)$

5.1 From the Neural network to the Bayesian network

When the weights of the neural network are known from the training phase, it is not difficult to calculate the integrals that appear in the expressions for s and δ given by equations 20 and 21. First we have to calculate $p(x)$ from equation 18. The integrand of (18) is effectively the ratio $\Delta A\Delta C/(\Delta A\Delta x)$ where ΔA, ΔC and Δx are elementary ranges of the values of the corresponding variables. To compute this numerically, we need to sample the area (A, C) with a regular grid and count how many sample points are inside rectangle $\Delta A\Delta x$ given that there is one, say, sample point inside rectangle $\Delta A\Delta C$. As ΔA is common to both, we only need to use sample points along the C axis and from the definition of x in terms of A, C and the (known) neural net weights, count the number of sample points we have inside each interval Δx, as a function of A. This then has to be integrated over A to calculate (18). This integration is done by sampling the A axis and for each range ΔA access and accumulate the corresponding values of the integrand.

The above process is equivalent to saying that to compute $p(x)$ we consider a regular grid of points in the (A, C) space, for each point we calculate the value of x and then count how many (A, C) points are mapped in the interval Δx.

The integrals that appear in formulae (25) are then calculated numerically using the trapezium rule and the tabulated values of $p(x)$. As some of the integrands are exponential functions, the trapezium rule may not be adequate for their calculation. Thus, the whole process is performed twice, the second time with the grid of points chosen in the (A, C) space, twice as dense as the first time. The result is assumed correct only if all the corresponding integrals calculated in both steps agree with each other within a pre-specified accuracy.

If one has high confidence in the training data as being representative of the true classes that the network will be called to identify in the testing phase, and one does not rely on the generalization capabilities of the network, then the accuracy of the approximation can be further improved: instead of minimizing the error uniformly over the whole (A, C) space, we weigh the values of x by function $p(x)$ that reflects the frequency by which values of the (A, C) pair actually are expected to appear in practice. This is not difficult to be achieved: After the network has been trained and its weights are fixed, we calculate the histogram of x values by using the (A, C) values of the training patterns themselves.

Thus, in this case a modified version of equation (18) is implicitly used:

$$p(x) \equiv \int_{A_1(x)}^{A_2(x)} \frac{1}{-W_{PZ_1}A - W_{Y_1Z_1}} \tilde{g}(A, x) dA \tag{22}$$

where $\tilde{g}(A, x)$ is the prior probability density function of the data we are dealing with, expressed in terms of (A, x) and derived from the corresponding function $g(A, C)$, say, that applies to the (A, C) space. This option allows us to reduce the error of our classifier exploiting prior knowledge concerning our problem.

5.2 From the Bayesian network to the Neural network

When the weights of the neural networks are not known, the calculation of $p(x)$ is not straightforward. One needs to calculate first the histograms of the output values f_B of the Bayesian network as given by equation 12 by considering a regular sampling grid in the (A, C) space. The histogram of the $p(x)$ values then can be derived from them, assuming that the Bayesian and the neural outputs should be the same. Thus, we use the equation $x = lnf_B - ln(1 - f_B)$. This histogram is corrected to have bins of equal width by interpolation and then it is used in the numerical calculation of the integrals that appear in (30).

6 Correspondence in singly connected networks with intermediary nodes

In this section we describe how to obtain correspondence in singly connected Bayesian networks where the input nodes are connected to the output nodes through some intermediary nodes[a]. See figure 6.

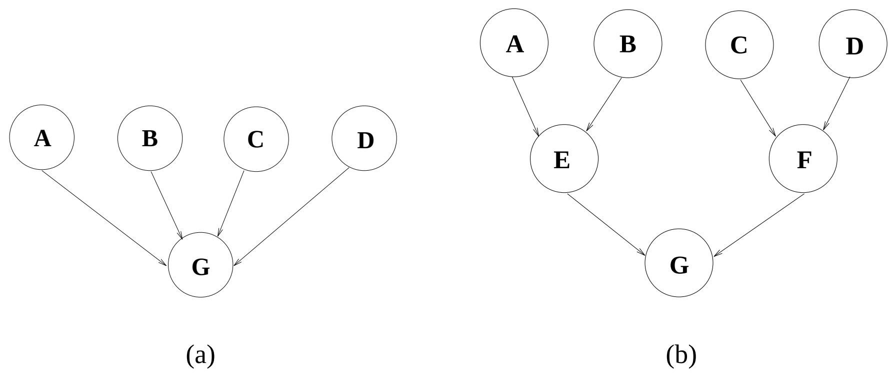


Figure 6: Example Bayesian networks (a) without intermediary nodes and (b) with intermediary nodes

Usually the intermediary nodes are introduced in a Bayesian network in the first place to aid the construction of conditional probability matrices. It is more often the case that rules connecting directly input to output may not be available by the experts but instead rules are only provided between the nodes of interest and some other intermediary nodes whose state at any point in time is usually of less interest. Moreover even if rules did exist which connected input to output, the conditional probability matrix between them could be of high dimensionality with further difficulties in deriving its elements. If we now follow our suggested method of obtaining the correspondence, the number of degrees of freedom of the Bayesian network will be different from that of the suggested neural network.

The method to overcome the above problem is to reduce the singly connected network of figure 6b to the network of figure 6a and apply the method of correspondence as before. In order to do this we need to calculate the conditional probability matrix $P(G|A, B, C, D)$ which will now define the new *reduced* network. This probability can be found in terms of

[a]Singly connected is a Bayesian network with only one path between any two nodes

the conditional probabilities $P(E|A, B)$, $P(F|C, D)$ and $P(G|E, F)$ which now define the existing network. This formula is given by:

$$P(G|A, B, C, D) = \sum_j \sum_i P(G|E_j, F_i)P(E_j|A, B)P(F_i|C, D)$$

where the sums range over all states of variables E and F.

We have now created a conditional probability matrix which will be used in the network shown in figure 6a but performs effectively the same inference given the same data as the one of figure 6b since the information of the intermediary nodes is incorporated in the *unified* matrix $P(G|A, B, C, D)$.

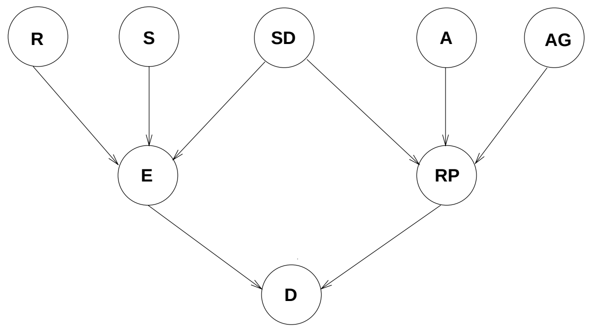
In summary, when the Bayesian network has intermediary nodes, we obtain correspondence with the neural network by reducing it to a simpler architecture with fewer degrees of freedom.

7 Application to a Geographic Information System

In this section we present an application of the method presented in this paper for obtaining the elements of the conditional probability matrix of a Bayesian network from the weights of the corresponding trained neural network.

The application we consider is that of assessing the degree of risk of desertification of burned forest areas in the Mediterranean region, given some information (evidence) on the factors that influence desertification.

Burned stands in the Mediterranean forests usually regenerate naturally between 2 to 5 years after a forest fire. However, several factors can prevent this natural regeneration. The degree of land degradation (desertification) varies between different areas and depends on the topography of the area, surface geology, soil depth and human factors.



R	:	Rock Type (permeable, semi-permeable, impermeable)
S	:	Ground Slope (gentle, middle, steep)
SD	:	Soil Depth (bare, shallow, deep)
A	:	Ground Aspect (south, west/east, north)
AG	:	Animal Grazing (slightly, moderately, heavily grazed)
E	:	Risk of Erosion (low, medium, high)
RP	:	Regeneration Potential (low, medium, high)
D	:	Risk of Desertification (no/slight, low, medium, high,very high)

Figure 7: Bayesian network constructed

We have created the Bayesian network shown in figure 7 for the purpose of taking into consideration all these factors in order to assess the risk of desertification of a burned forest. All variables and their possible states are also shown in figure 7. We represent all variables of interest by nodes and draw arcs form causes to effects using the information provided by experts. These links are then quantified by the conditional probability matrices. We demonstrate in this section how we derive the three conditional probability matrices $P(E|R, S, SD)$, $P(RP|SD, A, AG)$ and $P(D|E, RP)$ using the proposed algorithm. Experts have supplied us with rules with which we can infer (by hard reasoning) the state of the three derivative nodes, namely E, RP and D, given the state of their corresponding parents. We can therefore consider the above network as consisting of three simple subnetworks, namely one with inputs R, S and SD, one with inputs SD, A and AG, and one with inputs E and RP.

We can then construct the corresponding neural network for each of these three networks and train it, each one with its own training data provided.

We shall now demonstrate how we can obtain the conditional probability matrix for the first sub-network. As described in previous sections, we construct a single layer neural network shown in figure 8 which consists of 26 input and 2 output nodes. The two output nodes return the confidences of a region being in the two independent states of erosion, low and $medium$. The 26 input nodes are determined as follows: For each independent state of each parent variable, we assign one input unit in the corresponding neural network representing the confidence in that state. This gives a total of 6 inputs for the 3, 3-valued parent variables namely R, S and SD. The remaining 20 input units are all the possible product combinations of these 6 confidences, excluding products of confidences in states of the same variable. For example, let us denote the confidence in the ith state of variable R by R_i etc, i.e. let subscripts indicate which state and capital letters which variable of the network. Then we have in the input layer of the corresponding neural network the units representing: R_1, R_2, S_1, S_2, SD_1, SD_2, R_1S_1, R_1S_2, R_1SD_1,...,$R_2S_2SD_2$. There are 26 such units. The confidence with which the output node belongs to class $high$ can be inferred from the outputs in nodes E_1 and E_2, ie the confidences with which it belongs to the classes low and $medium$.

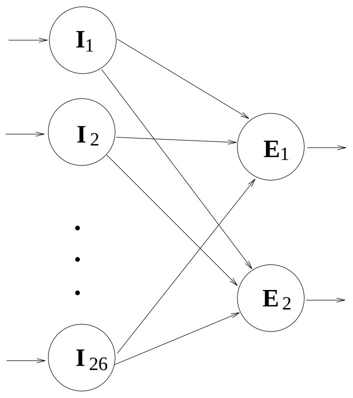


Figure 8: A neural network which corresponds to the first Bayesian sub-network of figure 7

We then use as training patterns the results and data provided by experts for the risk of erosion for 39 sites. After training the neural network, we obtain its weights. As we have observed from the test sites, the 39 training patterns seemed representative of the true classes that the network would need to identify. Therefore we derived $p(x)$ that appears in equation 18 by calculating the frequency with which each output occurs when tested on the training patterns. After deriving the parameters of the approximating line, using equations 20 and 21, we calculated the first column of the matrix using the weights on the links connected to the state low. This column gives the probability of $erosion$ being low given all the possible combinations of the parents. The remaining 26 weights are used to derive the second column of independent matrix elements corresponding to $erosion$ being $medium$. The other conditional probability matrices are derived in a similar fashion.

Table 1 shows the root-mean square (RMS) error for each output, for each of the variables i.e. the quadratic difference between the neural network output and the Bayesian network output over all 39 training patterns. Their average RMS error shown in the last column is computed using the individual RMS errors for each output.

		RMS				
		State 1	State 2	State 3	State 4	Average
$Output$	Erosion	0.015	0.012	N/A	N/A	0.0135
	Regeneration	0.010	0.009	N/A	N/A	0.0095
	Desertification	0.022	0.023	0.011	0.021	0.0192

Table 1: The RMS error resulting by calculating $p(x)$ using the training pattern file.

		RMS				
		State 1	State 2	State 3	State 4	Average
$Output$	Erosion	0.039	0.019	N/A	N/A	0.0290
	Regeneration	0.029	0.014	N/A	N/A	0.0215
	Desertification	0.043	0.042	0.035	0.024	0.0360

Table 2: The RMS error resulting by calculating $p(x)$ using uniformly distributed random inputs.

Table 2 was derived by calculating the correspondence between the two networks assuming uniform distributions of the input variables in the range (0,1).

As we can see from the tables, the method of uniform random input has higher error than the method of obtaining $p(x)$ using the pattern file outputs. This is as expected since the probability distribution of each output as derived using the pattern file, is closer to the "true" probability distribution, assuming that we consider as "true" the distribution which best models the training data.

This is because the method of using uniformly distributed random inputs neglects the information given by the pattern file, thus produces the highest RMS error taken over all *training* patterns.

Table 3 shows the results of the risk of $erosion$, $regeneration\, potential$ and $desertification$ obtained using the Bayesian network with derived conditional probability matrices using correspondence as indicated by the RMS of table 1. The results are for the 39 sites used for training and each column gives the belief in the risk of each of the variables. The class with the highest probability is assigned as the final classification label. The " $\surd$" in each column indicates agreement with the expert. Out of the 39 sites, 37 sites agreed on the risk of erosion, 38 agreed for the risk of regeneration potential and 35 agreed on the desertification risk.

Table 4 shows the results obtained by the Bayesian network for the three variables on 14 test sites which were not used for the derivation of the conditional probability matrices. Again the "$\surd$" is to indicate agreement with experts. Out of 14 sites, 12 agreed on the erosion, 11 agreed on regeneration potential and 11 agreed on desertification risk.

8 Summary and Conclusions

We have presented in this paper a novel method for eliciting the conditional probability matrices needed for a Bayesian network with the help of a neural network. We demonstrated how we can obtain a correspondence between the two networks by deriving a closed-form solution so that the outputs of the two networks are similar in the least square error sense, not only when determining the discriminant function, but for the full range of their outputs. The probability density functions of the independent variables of the problem is taken into consideration when we compute the least square error approximation.

We have shown that any singly connected Bayesian network, where input is represented by the root nodes (i.e. nodes with no parents) and the output by the leaf nodes (i.e. nodes with no children), can be reduced to one with direct links from input to output by maintaining the same information as the complete structure. This method guarantees exact correspondence and can be extended to any number of inputs. The total number of nodes in the input layer of the neural network constructed to represent a Bayesian network consisting of M nodes, each with N_i (for $i = 1 \ldots M$) states, is $\prod_{i=1}^{M} N_i - 1$.

Site	BEL(E)	BEL(RP)	BEL(D)
lavrio1	(0.13, 0.26, 0.61) √	(0.09, 0.26, 0.65) √	(0.10, 0.15, 0.43, 0.23, 0.09) √
lavrio2	(0.42, 0.45, 0.13)	(0.16, 0.15, 0.69) √	(0.14, 0.26, 0.38, 0.13, 0.09) √
lavrio3	(0.42, 0.45, 0.13) √	(0.15, 0.16, 0.69) √	(0.14, 0.26, 0.38, 0.13, 0.09) √
lavrio4	(0.63, 0.26, 0.11) √	(0.10, 0.15, 0.75) √	(0.16, 0,32, 0.27, 0.14, 0.11) √
lavrio5	(0.63, 0.26, 0.11) √	(0.59, 0.27, 0.14) √	(0.10, 0.20, 0.43, 0.15, 0.12) √
lavrio6	(0.63, 0.26, 0.11) √	(0.15, 0.16, 0.69) √	(0.15, 0.31, 0.29, 0.14, 0.11) √
pateras1	(0.49, 0.38, 0.13) √	(0.59, 0.27, 0.14) √	(0.10, 0.18, 0.47, 0.14, 0.11) √
pateras2	(0.49, 0.38, 0.13) √	(0.59, 0.27, 0.14) √	(0.10, 0.18, 0.47, 0.14, 0.11) √
pateras3	(0.49, 0.38, 0.13) √	(0.59, 0.27, 0.14) √	(0.10, 0.18, 0.47, 0.14, 0.11) √
pateras4	(0.49, 0.38, 0.13) √	(0.59, 0.27, 0.14) √	(0.10, 0.18, 0.47, 0.14, 0.11) √
pateras5	(0.74, 0.17, 0.09) √	(0.04, 0.15, 0.81) √	(0.17, 0.36, 0.21, 0.14, 0.12) √
pateras6	(0.49, 0.38, 0.13) √	(0.59, 0.27, 0.14) √	(0.10, 0.18, 0.47, 0.14, 0.11) √
pateras7	(0.27, 0.64, 0.09) √	(0.14, 0.39, 0.47) √	(0.12, 0.21, 0.41, 0.17, 0.09) √
pateras8	(0.27, 0.64, 0.09) √	(0.16, 0.15, 0.69) √	(0.13, 0.23, 0.44, 0.12, 0.08) √
pateras9	(0.27, 0.64, 0.09) √	(0.14, 0.43, 0.43) √	(0.12, 0.20, 0.40, 0.18, 0.10) √
pateras10	(0.74, 0.17, 0.09) √	(0.04, 0.15, 0.81) √	(0.17, 0.36, 0.21, 0.14, 0.12) √
pateras11	(0.74, 0.17, 0.09) √	(0.16, 0.15, 0.69) √	(0.16, 0.34, 0.25, 0.13, 0.12) √
pateras12	(0.87, 0.12, 0.01) √	(0.16, 0.15, 0.69) √	(0.17, 0.38, 0.20, 0.12, 0.13) √
pateras13	(0.87, 0.12, 0.01) √	(0.16, 0.15, 0.69) √	(0.17, 0.38, 0.20, 0.12, 0.13) √
pateras14	(0.27, 0.64, 0.09) √	(0.16, 0.15, 0.69)	(0.13, 0.23, 0.44, 0.12, 0.08) √
pateras15	(0.27, 0.64, 0.09) √	(0.10, 0.15, 0.75) √	(0.14, 0.24, 0.42, 0.13, 0.07) √
pateras16	(0.27, 0.64, 0.09) √	(0.10, 0.15, 0.75) √	(0.14, 0.24, 0.42, 0.13, 0.07) √
pendeli1-1	(0.42, 0.45, 0.13)) √	(0.04, 0.15, 0.81) √	(0.15, 0.28, 0.34, 0.14, 0.09)
pendeli1-2	(0.13, 0.26, 0.61) √	(0.10, 0.15, 0.75) √	(0.11, 0.15, 0.47, 0.20, 0.07) √
pendeli1-3	(0.14, 0.25, 0.61) √	(0.14, 0.39, 0.47) √	(0.09, 0.14, 0.40, 0.25, 0.12) √
pendeli1-4	(0.42, 0.45, 0.13) √	(0.16, 0.15, 0.69) √	(0.14, 0.26, 0.38, 0.13, 0.09) √
pendeli1-5	(0.42, 0.45, 0.13) √	(0.04, 0.15, 0.81) √	(0.15, 0.28, 0.34, 0.14, 0.09)
pendeli1-6	(0.42, 0.45, 0.13) √	(0.16, 0.15, 0.69) √	(0.14, 0.26, 0.38, 0.13, 0.09) √
pendeli1-7	(0.14, 0.25, 0.61) √	(0.14, 0.39, 0.47) √	(0.09, 0.14, 0.40, 0.25, 0.12) √
pendeli2-1	(0.13, 0.26, 0.61) √	(0.16, 0.15, 0.69) √	(0.11, 0.15, 0.48, 0.19, 0.07) √
pendeli2-2	(0.42, 0.45, 0.13)	(0.16, 0.15, 0.69) √	(0.14, 0.26, 0.38, 0.13, 0.09) √
pendeli2-3	(0.13, 0.26, 0.61) √	(0.09, 0.26, 0.65) √	(0.10, 0.15, 0.43, 0.23, 0.09) √
pendeli2-4	(0.13, 0.26, 0.61) √	(0.09, 0.26, 0.65) √	(0.10, 0.15, 0.43, 0.23, 0.09) √
barnavas1	(0.14, 0.25, 0.61) √	(0.14, 0.39, 0.47) √	(0.09, 0.14, 0.40, 0.25, 0.12)
barnavas2	(0.14, 0.25, 0.61) √	(0.16, 0.15, 0.69) √	(0.11, 0.15, 0.48, 0.19, 0.07) √
barnavas3	(0.14, 0.25, 0.61) √	(0.14, 0.39, 0.47) √	(0.09, 0.14, 0.40, 0.25, 0.12) √
barnavas4	(0.14, 0.25, 0.61) √	(0.16, 0.15, 0.69) √	(0.11, 0.15, 0.48, 0.19, 0.07) √
barnavas5	(0.14, 0.25, 0.61) √	(0.16, 0.15, 0.69) √	(0.11, 0.15, 0.48, 0.19, 0.07) √
barnavas6	(0.42, 0.45, 0.13) √	(0.15, 0.16, 0.69) √	(0.14, 0.26, 0.38, 0.13, 0.09)

Table 3: Results of erosion, regeneration potential and desertification of the 39 training sites using the matrices derived

Site	BEL(E)		BEL(RP)		BEL(D)	
Tlavrio1	(0.13, 0.26, 0.61)	$\surd$	(0.16, 0.19, 0.65)	$\surd$	(0.10, 0.15, 0.47, 0.20, 0.08)	$\surd$
Tlavrio2	(0.42, 0.45, 0.13)	$\surd$	(0.11, 0.24, 0.65)	$\surd$	(0.15, 0.37, 0.17, 0.15, 0.16)	$\surd$
Tpateras1	(0.87, 0.12, 0.01)	$\surd$	(0.11, 0.24, 0.65)	$\surd$	(0.16, 0.34, 0.25, 0.13, 0.12)	$\surd$
Tpateras2	(0.74, 0.17, 0.09)	$\surd$	(0.16, 0.15, 0.69)	$\surd$	(0.16, 0.34, 0.25, 0.13, 0.12)	$\surd$
Tpateras3	(0.27, 0.64, 0.09)	$\surd$	(0.16, 0.19, 0.65)	$\surd$	(0.13, 0.23, 0.43, 0.13, 0.08)	$\surd$
Tpateras4	(0.74, 0.17, 0.09)	$\surd$	(0.11, 0.24, 0.65)	$\surd$	(0.15, 0.33, 0.22, 0.16, 0.14)	$\surd$
Tpateras5	(0.27, 0.64, 0.09)	$\surd$	(0.16, 0.19, 0.65)	$\surd$	(0.13, 0.23, 0.43, 0.13, 0.08)	$\surd$
Tpateras6	(0.27, 0.64, 0.09)	$\surd$	(0.15, 0.26, 0.59)		(0.13, 0.22, 0.42, 0.15, 0.08)	$\surd$
Tpendeli1-1	(0.13, 0.26, 0.61))	$\surd$	(0.16, 0.19, 0.65)	$\surd$	(0.10, 0.15, 0.47, 0.20, 0.08)	$\surd$
Tpendeli1-2	(0.36, 0.40, 0.24)		(0.15, 0.26, 0.59)		(0.12, 0.23, 0.37, 0.17, 0.11)	
Tpendeli1-3	(0.13, 0.26, 0.61)	$\surd$	(0.15, 0.26, 0.59)		(0.10, 0.15, 0.44, 0.22, 0.09)	
Tpendeli2-1	(0.30, 0.31, 0.39)		(0.64, 0.29, 0.07)	$\surd$	(0.07, 0.13, 0.51, 0.17, 0.12)	$\surd$
Tbarnavas1	(0.42, 0.45, 0.13)	$\surd$	(0.11, 0.24, 0.65)	$\surd$	(0.14, 0.26, 0.35, 0.15, 0.10)	$\surd$
Tbarnavas2	(0.42, 0.45, 0.13)	$\surd$	(0.11, 0.24, 0.65)	$\surd$	(0.14, 0.26, 0.35, 0.15, 0.10)	

Table 4: Results of erosion, regeneration potential and desertification of the 14 test sites using the matrices derived

The additional features are $\prod_{i=1}^{M} N_i - 1 - M$. This indicates that the number of new features in the neural network increases exponentially with the addition of further nodes in the Bayesian network. However, the required neural network would, otherwise, need the additional units to be added in hidden layers if the method of introducing new features were not adopted. Moreover, by reducing any N-th order function to a linear function, we automatically have a structure for the neural network i.e. a simple two layered perceptron, which will guarantee to model the function. Otherwise a suitable number of hidden layers and of hidden units in these layers would have to be defined.

The proposed algorithm provides a theoretical study specifying the link between the two models, i.e. the Bayesian networks and the neural networks. Moreover, it provides a way of determining the weights for the neural network without training if the elements of the conditional probability matrix of the Bayesian network were available. Alternatively, the elements of the conditional probability matrix can be determined in terms of the weights and thresholds of the corresponding trainable neural network. This is particularly useful since the method of deriving these matrix elements so far was by quantifying expert rules which had the risk of inconsistencies and ambiguities.

A Obtaining the bounds of the Bayesian output function

In this section we will prove that the Bayesian output function given by (12), is bounded and attains its maximum and minimum values at the corners of the unit square defined by the variables A and C which both range from 0 to 1. The four corners are indeed the four independent elements of the conditional probability matrix i.e. m_1, m_2, m_3 and m_4.

For notational simplicity, assume a general notation of the function as:

$$f = \alpha AC + \beta A + \gamma C + \psi \tag{23}$$

The stationary point of the function is (A, C)=$(-\frac{\gamma}{\alpha}, -\frac{\beta}{\alpha})$.

The Hessian matrix of f is:

$$H = \begin{pmatrix} 0 & \alpha \\ \alpha & 0 \end{pmatrix}$$

Using the eigenvalues α and $-\alpha$ of this matrix, we can determine the nature of the stationary point (see Claycombe and Sullivan[2]). Since one eigenvalue is positive and the other one negative, then the stationary point is neither a maximum nor a minimum. Therefore the function does not have a maximum or minimum in the interior of the permitted region and since it is a continuous function, it therefore attains its maximum and minimum on the boundary of the region (see Beveridge and Schechter[1]) i.e. on the lines $A = 0$, $A = 1$, $C = 0$ or $C = 1$.

Along the boundary defined by $A = 0$ the function f is linear in C and therefore the maximum and minimum are attained at the extreme values of C i.e. 0 or 1 according to the values of γ and ψ and similarly for the other lines defining the permitted region. For global maximum and minimum we choose the corner with the highest and lowest values respectively. We therefore proved that on the boundary, the maximum and minimum values lie on the corners of the unit square. These corners i.e. $(A = 0, C = 0)$, $(A = 0, C = 1)$, $(A = 1, C = 0)$ and $(A = 1, C = 1)$ represent the conditional probability matrix elements m_4, m_3, m_2 and m_1 respectively.

So for obtaining the extrema we simply choose the maximum and minimum values of the conditional probability matrix.

The above proof extends to any number of parent nodes in the Bayesian network.

B Local linear approximation of the sigmoid function

In this appendix we give the derivation of the formulae determining the parameters of the line that approximates in the least square error sense the sigmoid function over a finite range. The function that has to be minimized, taking into consideration the probability density function of the independent variables, is given by (19).

Taking the derivatives with respect to the parameters of the line and equating them to zero, we obtain:

$$\frac{\partial f}{\partial s} = s\int_{xmin}^{xmax} p(x)x^2dx + \delta\int_{xmin}^{xmax} p(x)xdx - \int_{xmin}^{xmax} \frac{xp(x)}{1+e^{-x}}dx = 0 \qquad (24)$$

Introducing S_1, S_2 and S_3 as:

$$\begin{aligned} S_1 &\equiv \int_{xmin}^{xmax} x^2p(x)dx \\ S_2 &\equiv \int_{xmin}^{xmax} xp(x)dx \\ S_3 &\equiv \int_{xmin}^{xmax} \frac{x}{1+e^{-x}}p(x)dx \end{aligned} \qquad (25)$$

we get:

$$sS_1 + \delta S_2 = S_3 \qquad (26)$$

Similarly, differentiating with respect to δ and equating to zero we obtain:

$$\frac{\partial f}{\partial \delta} = s\int_{xmin}^{xmax} p(x)xdx + \delta\int_{xmin}^{xmax} p(x)dx - \int_{xmin}^{xmax} \frac{p(x)}{1+e^{-x}}dx = 0 \qquad (27)$$

Introducing S_4 and S_5 as:

$$\begin{aligned} S_4 &\equiv \int_{xmin}^{xmax} p(x)dx \\ S_5 &\equiv \int_{xmin}^{xmax} \frac{1}{1+e^{-x}}p(x)dx \end{aligned} \qquad (28)$$

we get:

$$sS_2 + \delta S_4 = S_5 \qquad (29)$$

Solving equations (26) and (29) we obtain:

$$s = \frac{S_3 - \delta S_2}{S_1} \tag{30}$$

and

$$\delta = \frac{S_1 S_5 - S_3 S_2}{S_1 S_4 - S_2^2} \tag{31}$$

The integrals S_1, S_2, S_3, S_4 and S_5 are then solved numerically to derive the values of the parameters s and δ of the line.

References

1. Beveridge, G. S. G. and Schechter, R. S., *Optimization: Theory and Practice*, McGraw-Hill, Inc., 1970.

2. Claycombe, W. W. and Sullivan, W. G., *Foundations of Mathematical Programming*, Reston Publishing Company, Inc., 1975.

3. Devijver, P. A. and Kittler, J., *Pattern recognition: A statistical approach*, Prentice-Hall International Inc. London, 1982.

4. Duin, R. P. W., "Superlearning Capabilities of Neural Networks?," in *Proceedings of the 8th Scandinavian Conference on Image Analysis*, 1993, pp. 547–554.

5. Longstaff, I. D. and Cross, J. F., "A pattern recognition approach to understand the multi-layer perceptron," *Pattern Recognition Letters*, **5**, 1987, pp. 315–319.

6. Nabhan, T. M. and Zomaya, A. Y., "Toward generating neural network structures for function approximation," *Neural Networks*, **7**(1), 1994, pp. 89–99.

7. Pearl, J., "Fusion, propagation, and structuring in Belief Networks," *Artificial Intelligence*, **29**, 1986, pp. 241–288.

8. Pearl, J., *Probabilistic reasoning in intelligent systems: Networks of plausible inference*, Morgan Kaufmann Publishers Inc., 1988.

9. Stassopoulou, A., Petrou, M. and Kittler, J., "Bayesian and neural networks for geographic information processing," *Pattern Recognition Letters*, **17**, 1996, pp. 1325–1330.