Generalizing the Noisy Or Model to $n$ary Variables

Sampath Srinivas
srinivas@rpal.rockwell.com
Rockwell International Science Center, Palo Alto Laboratory
444 High Street
Palo Alto, CA 94301
Technical Memorandum No. 79

April 1992

Abstract

The Noisy Or model is convenient for describing a frequently occurring class of uncertain relationships in belief networks [1]. Pearl describes the Noisy Or model for binary variables. Here we generalize the model to $n$ary input and output variables and to arbitrary functions other than the Boolean Or function. The resulting generalization models the uncertain relationship which results from introducing possible failures in the inputs of any discrete function.

1 The Model

Let $F : U_1 \times U_2 \times U_3 \ldots U_n \rightarrow X$ be a discrete valued function from n discrete random variables $U_1, U_2, U_3 \ldots U_n$ to another discrete random variable X.

We will use the following notations: We will refer to a possible state of the variable U with the lower case u. So for example using this notation, we can say that there exists x such that $x = F(u_1, u_2 \ldots u_n)$ for any possible $u_1, u_2 \ldots u_n$. We will refer to a vector of variables with bold face. In specific, $\mathbf{U}$ denotes the vector $[U_1, U_2, U_3 \ldots U_n]$. Similarly, we use bold face notation for vectors of states: $\mathbf{u}$ denotes the vector of states $[u_1, u_2, u_3 \ldots u_n]$.

Consider a situation where each input U_i feeds to a noise function $\mathcal{N}_i : U_i \rightarrow U_i^{'}$. $U_i^{'}$ is a discrete random variable which has the same states as U_i.

The noise function $\mathcal{N}_i$ is modeled as a box that does the following: The box can fail in any particular state u_i with probability $P_{U_i}^{inh}(u_i)$ – read this as the inhibitor probability for the state u_i of variable U_i.

When failure occurs in state u_i then the output of the box, i.e., the value of $U_i^{'}$ is u_i irrespective of the input U_i. The probability that no failure occurs is given by $P_{U_i}^{normal} = 1 - \sum_{u_i} P_{U_i}^{inh}(u_i)$.

Now consider feeding the outputs $U_i^{'}$ of the noise functions as inputs to the function F to yield an output X (See Fig 1). We see that we now have an uncertain relationship $P(X|\mathbf{U})$ between the actual inputs $\mathbf{U}$ to the output F. This uncertain relationship constitutes the generalized Noisy Or model.

A typical example might be a situation where an aggregated sensor reading X is derived from the readings of a set of individual sensors U_i. Each sensor U_i may fail with probability $P_{U_i}^{inh}(u_i)$ in state u_i, in which case its output is always u_i. The aggregation scheme is described by function F. As an example, F might be a simple average.

In the specific case where all the variables U_i are binary (with states $true$ and $false$) with

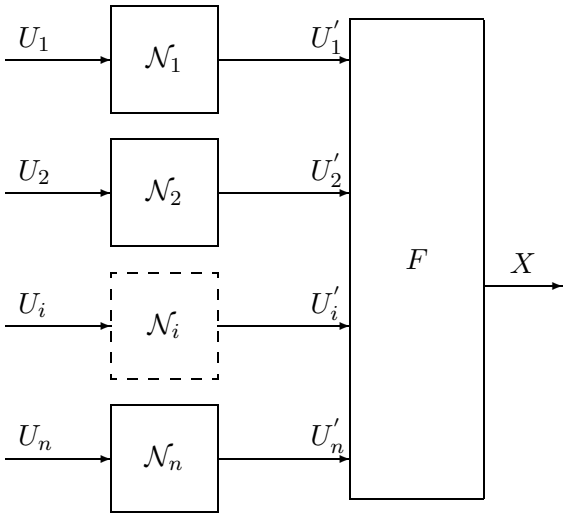

Figure 1: The generalized Noisy Or model

$P_{U_i}^{inh}(true) = 0$, X is binary and F is the Boolean Or function, we see that this generalized model collapses to the binary Noisy Or model described in [1].

2 Characterizing $P(X|\mathbf{U})$

Each noise function $\mathcal{N}_i$ defines a probability distribution $P_i(U_i^{'}|U_i)$ relating $U_i^{'}$ and U_i. From the model for $\mathcal{N}_i$ we see that the distribution P_i is calculated as:

$$P_i(u_i^{'}|u_i) = \begin{cases} P_{U_i}^{normal} + P_{U_i}^{inh}(u_i^{'}) & \text{if } u_i^{'} = u_i \\ P_{U_i}^{inh}(u_i^{'}) & \text{otherwise} \end{cases} \quad (1)$$

We now characterize the distribution $P(X|\mathbf{U})$ in terms of the inhibitor probabilities for each U_i and the function F.

We note that:

$$P(x|\mathbf{u}) = \sum_{\mathbf{u}'} P(x|\mathbf{u}^{'}, \mathbf{u}) P(\mathbf{u}^{'}|\mathbf{u})$$

Noting that the value of X is independent of $\mathbf{U}$ once $\mathbf{U}^{'}$ is known we simplify the equation to:

$$P(x|\mathbf{u}) = \sum_{\mathbf{u}'} P(x|\mathbf{u}^{'}) P(\mathbf{u}^{'}|\mathbf{u})$$

We note that $P(x|\mathbf{u}^{'}) = 1$ when $x = F(\mathbf{u}^{'})$ and $P(x|\mathbf{u}^{'}) = 0$ when $x \neq F(\mathbf{u}^{'})$. This simplifies the defining equation to:

$$P(x|\mathbf{u}) = \sum_{x=F(\mathbf{u}')} P(\mathbf{u}^{'}|\mathbf{u})$$

Now we note that the dependence of $\mathbf{u}^{'} = [u_1, u_2 \ldots u_n]$ on $\mathbf{u} = [u_1, u_2 \ldots u_n]$ can be split into n pairwise dependencies of $u_i^{'}$ on u_i. This is because the value of a variable $U_i{'}$ depends solely on U_i and not on any other variable U_j where $i \neq j$.

Thus we can simplify the equation to:

$$P(x|\mathbf{u}) = \sum_{x=F(\mathbf{u}')} P(\mathbf{u}^{'}|\mathbf{u}) = \sum_{x=F(\mathbf{u}')} \prod_{\mathbf{u}'} P_i(u_i^{'}|u_i) \quad (2)$$

We note that we have already defined $P(u_i^{'}|u_i)$ in terms of the inhibitor probabilities.

The above equation is easily converted to an algorithm to generate a conditional probability table given the inhibitor probabilities and the deterministic underlying function F (described later).

2.1 Binary Noisy Or as a special case

In the case of the Binary Noisy Or the inputs U_i and the output X are boolean with states t and f. Consider the following assignment of inhibitor probabilities for input U_i:

$$\begin{aligned} P_{U_i}^{inh}(f) &= c_i \\ P_{U_i}^{inh}(t) &= 0 \end{aligned}$$

In other words, an input U_i can fail with probability c_i in the f state and it *cannot* fail in the t state.

For the case $u_i^{'} = f$ this leads to the following values for $P_i(u_i^{'}|u_i)$:

$$\begin{aligned} P_i(f|f) &= P_{U_i}^{normal} + P_{U_i}^{inh}(f) \\ &= (1 - c_i) + c_i = 1 \\ P_i(f|t) &= P_{U_i}^{inh}(t) \\ &= c_i \end{aligned}$$

In other words we have:

$$P_i(f|u_i) = \begin{cases} 1 & \text{when } u_i = f \\ c_i & \text{when } u_i = t \end{cases}$$

Now consider $P(f|\mathbf{u})$. We introduce some notation here and refer to the state $[f, f, \ldots f]$ of $\mathbf{U_i^{'}}$,i.e., the state where are all the variables are in the false state, as $\mathbf{f}^{'}$. Similarly we refer to the state $[f, f, \ldots f]$ of $\mathbf{U_i}$ as $\mathbf{f}$.

For $x = f$, we note that the only case $\mathbf{u}^{'}$ satisfying $x = f = F(\mathbf{u}^{'})$ is $\mathbf{u}^{'} = \mathbf{f}^{'}$. This is because of the nature of the Boolean Or function.

So we have:

$$\begin{aligned} P(f|\mathbf{u}) &= \sum_{f=F(\mathbf{u}')} P(\mathbf{u}^{'}|\mathbf{u}) \\ &= P(\mathbf{f}^{'}|\mathbf{u}) \\ &= \prod_i P_i(f_i|u_i) \\ &= \prod_{u_i=t} c_i \prod_{u_i=f} 1 \\ &= \prod_{u_i=t} c_i \end{aligned}$$

It follows that:

$$P(t|\mathbf{u}) = 1 - \prod_{u_i=t} c_i$$

We thus see that the Binary Noisy Or described in [1] is a special case of our formulation.

2.2 Choice of a $n$ary function F

To make the above formulation attractive as a generalization of the Noisy Or paradigm to multi-valued inputs and outputs, a good intuitive function $F : U_1 \times U_2 \times U_3 \ldots U_n \rightarrow X$ has to be defined such that in the special case of binary inputs and binary outputs F collapses to the normal Boolean Or function.

For each U_i define an increasing ordering for the states and number them starting at 0 (In the binary case the state f is numbered 0 and the state t is numbered 1). Similarly, order and number the states of X.

Let m_i be the number of states of variable U_i. Let h_i be the highest state number of variable U_i, i.e., $h_i = m_i - 1$. Similarly define the number of states m_x and the highest state number h_x for the variable X.

Taking compatibility with the binary case and equal weighting of all inputs as criteria we suggest the following function[1][2]:

$$F(u_1, u_2 \ldots, u_n) = \left\lceil h_x \left(\frac{1}{n} \sum_i \frac{u_i}{h_i} \right) \right\rceil$$

Basically, we are finding the fraction of each input's state number over the maximum possible state number of that input, averaging these fractions, scaling to the number of states of the output, and mapping back to an actual state of the output after converting the scaled result to an integer.

This additive function will have the characteristic that as any input goes 'higher' it will tend to drive the output 'higher'. Further, the inputs are 'equally weighted' regardless of their arity. So, for example, a change from state 0 to state 1 in a binary input will have just the same effect as a change from 0 to 5 in an input with 6 states. Finally, the output is 0 if and only if all the inputs are 0.

[1] We use the syntax $\lceil \rceil$ for the Ceiling function. For a real number x, $\lceil x \rceil$ is the smallest integer i that satisfies $i \geq x$.

[2] In the following equation, the state u of a variable U is used synonymously with its state number.

We note that this function reduces to the Boolean Or function in the case where all inputs are binary and the output is binary.

2.3 Case of Binary output and $n$ary inputs

If the output X is a binary variable and the inputs U_i are $n$ary we note that the function F is as follows: When all the inputs are in state u_i^0 (i.e, the state numbered 0) the output X is in x^0, i.e the 'false' state of the output. When any of the inputs are in a state other than u_i^0 the output is x^1, i.e., the 'true' state of the output.

If further, we define $P_{U_i}^{inh}(u_i^0) = c_i$ and $P_{U_i}^{inh}(u_i^j) = 0$ for $j \neq 0$ we see that we have a generalization of the binary Noisy Or where there is only one possible failure state u_i^0 for any input U_i since the failure probability for the other states is 0. We can consider the state u_i^0 as a distinguished *false* state for the input U_i.

Thus the output X is false if and only if all the inputs are false or when the ones that are true fail in the false state – otherwise the output is true.

2.4 Obtaining strictly positive distributions

In some situations one requires a probability distribution for a belief net node X with predecessors $\mathbf{U}$ to be strictly positive, i.e., $\forall x \forall \mathbf{u} P(x|\mathbf{u}) > 0$.

For the generalized Noisy Or model, the definition of $P(x|\mathbf{u})$ is in Equation 2. From this definition we note that the following condition is necessary to ensure a strictly positive distribution:

> For all states x of X, the set $Invert(x) = \{\mathbf{u}' : x = F(\mathbf{u}')\}$ is not empty. In other words, F should be a function that maps *onto* X.

This condition is a natural restriction – if F does not satisfy this condition, the variable X, in effect, has superfluous states. For example, the function defined in Section 2.2 satisfies this restriction.

Assuming that the above condition is satisfied, the following condition is sufficient to ensure a strictly positive distribution:

> For any $\mathbf{u}'$ and $\mathbf{u}$, $P(\mathbf{u}'|\mathbf{u}) > 0$, i.e., $\prod_{\mathbf{u}'} P_i(u_i'|u_i) > 0$.

This second condition is a stronger restriction. From Equation 1 we note that this restriction is equivalent to requiring that all inhibitor probabilities are strictly positive, i.e., that $P_{U_i}^{inh}(u_i') > 0$ for all U_i and u_i. In other words, every input node should have a non-zero probability of failing in every one of its states.

Finally, we note that the canonical Binary Noisy Or formulation of [1] and its generalization to $n$ary inputs described in Section 2.3 always result in a distribution which is *not* strictly positive since $P(t|\mathbf{f}) = 0$.

3 Computing $P(X|\mathbf{U})$

We consider the complexity of generating the probabilities in the table $P(X|\mathbf{U})$.

Let $S = \prod_i m_i$ be the size of the joint state space of all the inputs U_i.

We first note that $P_i(u_i'|u_i)$ can be computed in $\Theta(1)$ time from the inhibitor probabilities. This leads to:

$$P(\mathbf{u}'|\mathbf{u}) = \prod_i P_i(u_i'|u_i) = \Theta(n)$$

Therefore:

$$P(x|\mathbf{u}) = \sum_{x=F(\mathbf{u}')} P(\mathbf{u}'|\mathbf{u}) = \Theta(Sn)$$

This is because, for a given x and $\mathbf{u}$ we have to traverse the entire state space of $\mathbf{U}'$ to check which $\mathbf{u}'$ satisfy $x = F(\mathbf{u}')$.

To compute the entire table we can naively compute each entry independently in which case we have:

$$P(X|\mathbf{U}) = m_x S\Theta(Sn) = \Theta(m_x n S^2)$$

However the following algorithm computes the table in $\Theta(nS^2)$:

Begin Algorithm
For each $\mathbf{u_i}$:

- For all states x of X set $P(x|\mathbf{u_i})$ to 0.
- For each $\mathbf{u_i'}$:
  - Set $x = F(\mathbf{u_i'})$.
  - Increment $P(x|\mathbf{u_i})$ by $P(\mathbf{u_i'}|\mathbf{u_i})$.

End Algorithm

3.0.1 Binary Noisy Or

In the case of the Binary Noisy Or, all U_i and X are binary variables. We see from Sec 2.1 that:

$$\begin{aligned} P(f|\mathbf{u}) &= \prod_{u_i=t} c_i = \Theta(n) \\ P(t|\mathbf{u}) &= 1 - \prod_{u_i=t} c_i = \Theta(n) \end{aligned}$$

For computing the table, we see that since $P(t|\mathbf{u}) = 1 - P(f|\mathbf{u})$, we can compute both probabilities for a particular $\mathbf{u}$ in $\Theta(n)$ time. So the time required to calculate the entire table $P(X|\mathbf{U})$ is $\Theta(Sn)$.

We see that in the case of the Binary Noisy Or there is a substantial saving over the general case in computing probabilities. This saving is achieved by taking into account the special characteristics of the Boolean Or function when computing probabilities.

3.0.2 Binary output and $n$ary inputs

From an analysis similar to the previous section we note that computation of $P(X|\mathbf{U})$ takes $\Theta(Sn)$ time in this case too.

3.1 Storage complexity

For the general case we need to store m_i inhibitor probabilities per predecessor. Therefore in this case $O(nm_{max})$ storage is required where $m_{max} = \max_i(m_i)$. This contrasts with $O(m_x m_{max}^n)$ for storing the whole probability table.

For a Binary Noisy Or we need to store one inhibitor probability per predecessor and this is $\Theta(n)$. Using tables instead would cost $\Theta(2 \times 2^n) = \Theta(2^n)$.

In the case of $n$ary inputs and binary output (as described above) one inhibitor probability per predecessor is stored. Thus storage requirement is $\Theta(n)$. Using a table would cost $O(m_{max}^n)$.

3.2 Reducing computation complexity

In general, one could reduce the complexity of computing $P(x|\mathbf{u})$ if one could take advantage of special properties of the function F to efficiently generate those $\mathbf{u}'$ that satisfy $x = F(\mathbf{u}')$ for a particular x.

Given a function F, we thus need an efficient algorithm $Invert$ such that $Invert(x) = \{\mathbf{u} : x = F(\mathbf{u})\}$. By choosing F carefully one can devise efficient $Invert$ algorithms. However, to be useful as a modeling device, the choice of F has also to be guided by the more important criterion of whether F does indeed model a frequently occurring class of phenomena.

This Noisy Or generalization has high complexity for computing probability tables from the inhibitor probabilities[3]. If the generalization is seen mostly as a useful modeling paradigm, then this complexity is not a problem, since the inhibitor probabilities can be pre-compiled into probability tables before inference takes place. Inference can be then performed with standard

[3]However, the binary Noisy Or does not suffer from this problem since the special structure of the F function and the fact that the inputs and outputs are binary reduce the complexity dramatically by a factor of S.

belief net propagation algorithms.

If this generalization, however, is seen as a method of saving storage by restricting the models to a specific kind of interaction, the cost of computing the probabilities on the fly may outweigh the gains of saving space.

4 Implementation in IDEAL

This generalized Noisy Or model has been implemented in the IDEAL [2] system.

Chance nodes in IDEAL can be declared to be Noisy Or nodes. The user provides the inhibitor probabilities and if required, the deterministic function F (see below). Noisy Or nodes support all belief net editing operations (like adding arcs) with appropriate defaults.

IDEAL ensures that belief networks or influence diagrams containing Noisy Or nodes can be solved with any of the implemented inference and evaluation algorithms. This is achieved by appropriate conversion to chance nodes when necessary.

For user convenience, IDEAL supports three kinds of Noisy Or nodes in increasing order or generality:

- *Binary* Noisy Or nodes are binary nodes as described in Section 2.3[4].
- *Nary* nodes can have any number of states. The deterministic function F for *Nary* nodes is the standard function described in Section 2.2. The user specifies inhibitor probabilities. *Nary* nodes are a generalization of *Binary* Nodes.
- *Generic* nodes can have any number of states. The user specifies both the deterministic function F and the inhibitor probabilities. *Generic* nodes are a generalization of *Nary* nodes and are an implementation of the most general form of the model described in this article.

[4]Note that the predecessors of *Binary* nodes need not be binary.

The implementation is documented in [3].

References

[1] Pearl, J. (1988) *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann Publishers, Inc., San Mateo, Calif.

[2] Srinivas, S. and Breese, J. (1990) IDEAL: A software package for analysis of influence diagrams. Proc. of 6th Conf. on Uncertainty in AI, Cambridge, MA.

[3] Srinivas, S. and Breese, J. (1989) IDEAL: Influence Diagram Evaluation and Analysis in Lisp – Documentation and User's Guide. Technical Memorandum No. 23, Rockwell International Science Center, Palo Alto, CA 94301.