TR-IT-0028

On the Applicability of Bayesian Belief Networks to Language Modeling in Speech Recognition

Helmut Lucke

1993.11

Baysian Belief Networks are a powerful tool for combining different knowledge sources with various degrees of uncertainty in a mathematical sound and computationally efficient way. Surprisingly they have not yet found their way into the speech processing field, despite the fact that it in this science multiple unreliable information sources exists. The present paper provides an introduction to the theory of Bayesian Networks. It also proposes several extensions to the classic theory as described by Pearl by describing mechanisms for dealing with statistical dependence among daughter nodes (usually assumed to be marginally independent) and by providing a learning algorithm based on the EM-algorithm with which the probabilities of link matrices can be learned from example data. Using these ideas a possible language model for speech recognition is constructed. It is evaluated over a text data base.

1 Introduction – dealing with uncertainties

Modeling language for speech recognition inevitably involves dealing with uncertainty. This is for two reasons. Firstly the very observations made by the speech recognizer are of probabilistic nature. A speech recognizer rarely outputs just one word candidate with high certainty but rather a list of candidates with relative rankings. A language model parsing such an output needs to take information contained in the relative rankings and scores into account to produce the most likely sequence of words. Secondly language models specified as a set of crisp rules not involving probabilities or other kinds of scoring are not very efficient in describing the language. Either the coverage is too low, forbidding sentences which are commonly uttered or else if the rules are less strict many exceptional and unlikely to occur sentences are classed as grammatical.

For this reason, the use of probabilistic models such as stochastic grammars has been proposed. These models replace the grammatical/ungrammatical classification by a graded scale. Structurally very simple models such as the bigram/trigram models [2] have been observed to be extremely effective, purely on grounds of their stochastic nature.

Recently a similar trend from rule-based to probability-based methods has been observed in the field of artificial intelligence. A mathematical tool known as Bayesian networks [4] has been developed to incorporate probabilities into expert systems and to allow them to make plausible deductions on the basis of insufficient information [5].

Bayesian networks provide a mathematically sound foundation for making plausible inferences under uncertainty. What is more, the calculation scheme implied by these networks turns out to be surprisingly simple and computationally efficient. Although a number of mathematical problems have to be solved in order to apply these models to expert systems and the like, the known properties of Bayesian networks make them far too attractive to be overlooked.

We attempt to show in this paper that Bayesian networks, which were primarily developed for use in probabilistic expert systems, can be applied (on a somewhat lower level) to the parsing of natural language particularly when uncertainty is involved.

The paper can roughly be divided into two parts. In the first part we will present some key results of the theory of Bayesian networks, working from a simple example. We will then generalize the network propagation equation and provide a mechanism for inferring the conditional probabilities of the model. This learning algorithm which is based on the EM algorithm is computationally efficient and guaranteed to increase the likelihood of the observations. In the latter half of this paper we will discuss how syntactic parse trees of a sentence can be viewed as a causal structure describing the 'causal' development of this sentence and how the Bayesian network propagation techniques developed in the earlier part can be used to infer syntactic structure. The learning algorithm can be used to update the probabilities of the model in an iterative fashion. In this way grammatical rules can be learned purely by observing unlabeled example text.

2 Bayesian Belief Propagation

In this section we will summarize and extend the theory of Bayesian Belief Propagation. The theory as proposed by Pearl [4] represents knowledge in the form of a qualitative graph with probabilities associated to each link. Each node in the graph corresponds to an event and a directed link between two nodes indicates a causal relationship between the corresponding events.

We will illustrate Pearl's approach by quoting one of his own examples:

> **Example [Pearl]:** *Mr. Holmes receives a telephone call from his neighbour Dr. Watson, who states that he hears the sound of a burglar alarm from the direction of Mr. Holmes's house. While preparing to rush home, Mr. Holmes recalls that Dr. Watson is known to be a tasteless practical joker, and he decides to first call another neighbour, Mrs. Gibbon, who, despite occasional drinking problems is far more reliable.*

A graphical structure representing the causal influences of this story is shown in Figure 1. Mr. Holmes receives two possibly different accounts about an alarm sound at his house. He tries to infer the probability of a burglary given the above evidence.

Before discussing this example in detail, let us define some notation. Let H denote the random variable describing whether there is a burglary, S the one describing the state of the alarm system (either ringing or not ringing) and G and W the ones describing the results of Gibbon's and Watson's testimony. H can in fact take two values: Burglary or not Burglary which we may denote h_1 and h_2. The same is true for the other random variables. In general however a random variable at a node may take more than two values.

Further we denote by $\Pr(S|H)$ the matrix

$$\Pr(S|H) = \begin{pmatrix} \Pr(S = s_1|H = h_1) & \Pr(S = s_1|H = h_2) \\ \Pr(S = s_2|H = h_1) & \Pr(S = s_2|H = h_2) \end{pmatrix} \tag{1}$$

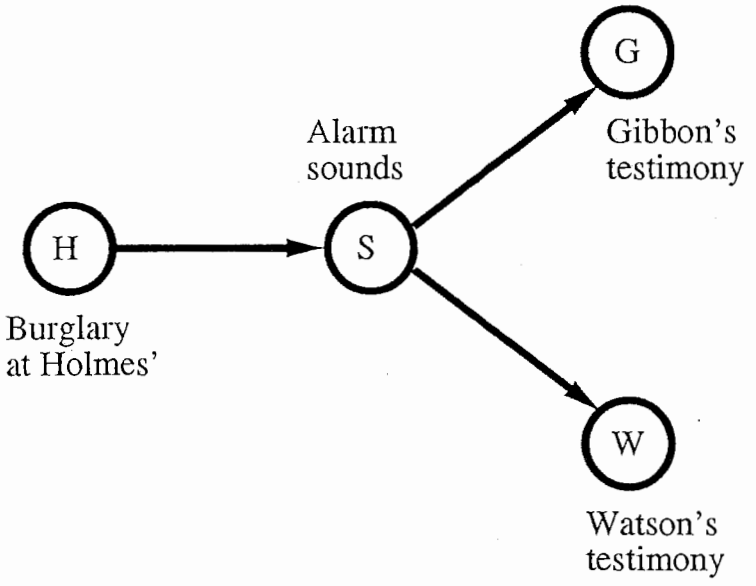

図 1:

Now, according to Pearl's theory, Holmes can reasonably be expected to have at least estimates of the matrices $\Pr(S|H)$, $\Pr(G|S)$ and $\Pr(W|S)$ stored in his knowledge base. He can also be expected to know the prior probability $\Pr(H)$ of there being a burglary on any given day of the year without any additional evidence. However he is unlikely to know more complicated conditional probabilities such as $\Pr(H|W,G)$, the quantity that he is interested in. It is therefore argued that Holmes need to compute these probabilities on the fly in order to make a decision on whether he should rush back home.

Fortunately there is a simple computational scheme available which allows one to calculate the $\Pr(H|W,G)$ from the elementary quantities $\Pr(S|H)$, $\Pr(G|S)$, $\Pr(W|S)$ and $\Pr(H)$. These calculations can be performed on a local scale following the graphical structure of Figure 1.

2.1 Belief propagation equations

The calculating scheme developed by Pearl is simple in that it only involves a few fundamental operations. It is also local in the sense that we could assign a simple processor to each node in the graphical representation. Each processor communicates with its neighbouring processors by exchanging simple messages via the links of the graph. It performs simple operations on its inputs received from the other nodes and mediates the results to its neighbours. The propagation equations are true for scenarios, where the graphical structure is of the form of a tree, i.e. does not contain any cycles. This is the case in the example above. Moreover the graph has the property that every node has at most one parent, i.e. at most one cause. This is not required by the theory in general, but in this paper for the sake of simplicity we will only consider such graphs. Directed trees in which nodes may have more than one parent are known as poly-trees and Pearl discusses such structures.

To describe the propagation equations we need to define some more notation: The evidence e stands for the total observed information i.e. the two testimonies in the example. Further for a node X let e_X^- denote the part of the evidence that is connected to one of the descendants of X and e_X^+ the remaining evidence. For the graph in Figure 1 we have

$$\begin{array}{ll} e_W^- = \text{Watson's testimony} & e_W^+ = \text{Gibbon's testimony} \\ e_G^- = \text{Gibbon's testimony} & e_G^+ = \text{Watson's testimony} \\ e_S^- = e_H^- = e & e_S^+ = e_H^+ = \emptyset \end{array} \tag{2}$$

Furthermore Pearl defines for each node X of the diagram two vectors:

$$\lambda(X) = \Pr(e_X^-|X) = \begin{pmatrix} \Pr(e_X^-|X{=}x_1) \\ \Pr(e_X^-|X{=}x_2) \\ \vdots \\ \Pr(e_X^-|X{=}x_n) \end{pmatrix} \qquad \pi(X) = \Pr(X|e_X^+) = \begin{pmatrix} \Pr(X{=}x_1|e_X^+) \\ \Pr(X{=}x_2|e_X^+) \\ \vdots \\ \Pr(X{=}x_n|e_X^+) \end{pmatrix} \tag{3}$$

The essence of the belief propagation theory lies in the fact that these quantities can be 'propagated' through the underlying graph and thereby calculated recursively. It turns out that the propagation rules are surprisingly simple. However we need to define two more auxiliary vectors: In a general tree, if V is a parent of nodes $U_1, \ldots, U_k$ (see Figure 3), we define the auxiliary vectors:

Conditional probabilities: The derivations in this paper make use of conditional probabilities of the form $\Pr(A|B)$. We will use capital letters A, B, C to indicate nodes in the networks. A node A can be thought of as a discrete random variable taking values $a_1, a_2, \ldots$. If one or more of these node variables enters the argument of the probability function P the resulting expression is to be interpreted as a tensor. For example the expression $\Pr(e_U^- \mid U)$ is a vector. It has as many components as U can take values. The i'th component is $\Pr(e_U^- \mid U = u_i)$. Similarly $\Pr(e_U^-, U \mid V, X, e_X^+)$ is a tensor of rank 3 whose ijk'th component is $\Pr(e_U^-, U = u_i \mid V = v_j, X = x_k, e_X^+)$.

Important vectors and there definitions:

$\lambda(U)$	$= \Pr(e_U^- \mid U)$	(evidential support vector)
$\pi(U)$	$= \Pr(U \mid e_U^+)$	(causal support vector)
$\mathrm{BEL}(U)$	$= \Pr(U \mid e)$	(Belief vector)

Vector products:

ab	componentwise vector product: $(ab)_i = a_i b_i$
$a \cdot b$	familiar dot product: $a \cdot b = \sum_i a_i b_i$
$\lambda(U)\Pr(U \mid V)$	vector-matrix product. Vector matrix product of this form are performed in the only sensible way, i.e. identifying states of the same variable. So if i indices states of the variable U and j indices states of the variable V then $\lambda(U)\Pr(U\|V) = (\sum_i \lambda(U)_i \Pr(U = i\|V = j))$. In general we will not make transpositions of vectors or matrices explicit.

Normalizing constants: When the letter α appears in front of a vector it represents a real number normalizing that vector. Occasionally α occurs more than once in an equation. In this case they usually represent different normalization factors.

図 2: Notation used in this paper

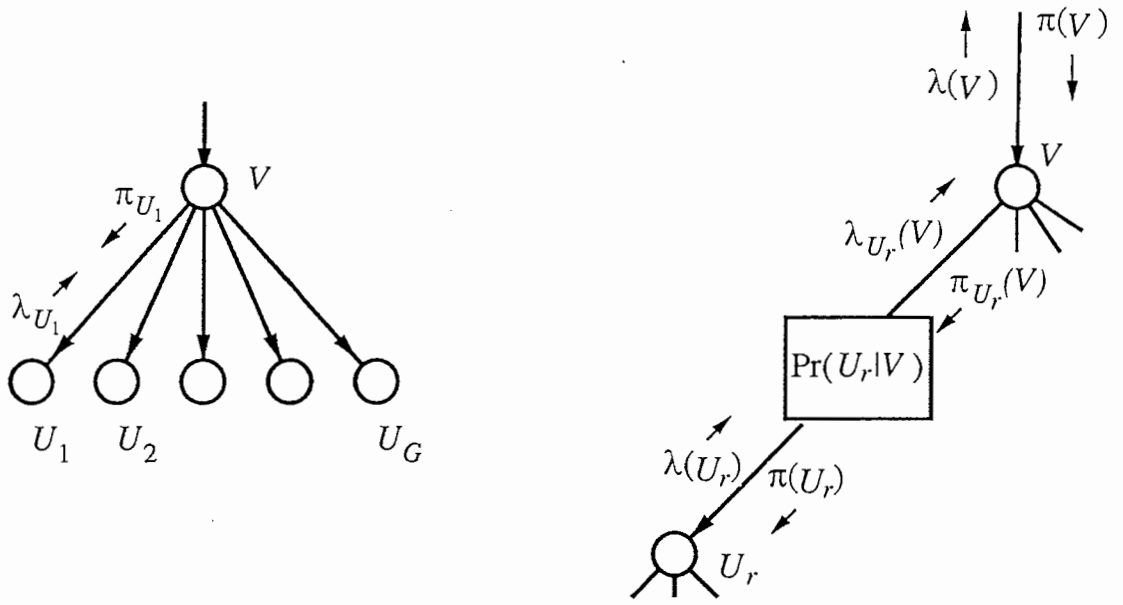

図 3:

$$\lambda_{U_r}(V) = \Pr(e^-_{U_r}|V) \tag{4}$$
$$\pi_{U_r}(V) = \Pr(V|e^+_{U_r}) \tag{5}$$

One can now show (for a derivation see Pearl [4] or section 4.1 where we derive more general versions of these equations):

$$\lambda_{U_r}(V) = \lambda(U_r)\Pr(U_r|V) \tag{6}$$
$$\lambda(V) = \prod_{r=1}^{k} \lambda_{U_r}(V) \tag{7}$$
$$\pi_{U_r}(V) = \alpha\pi(V)\prod_{l\neq r} \lambda_{U_l}(V) \tag{8}$$
$$\pi(U_r) = \Pr(U_r|V)\pi_{U_r}(V) \tag{9}$$

Here the products on the right-hand-side of equation 6 and 9 are familiar vector-matrix products. The vector product involved in equations 7 and 8 is the component-wise vector product (i.e. vector $\times$ vector = vector). The coefficient α in equation 8 is a scalar chosen such that the vector $\pi_{U_r}(V)$ on the left-hand-side is normalized.

Using equation 7 and 9 it is thus possible to calculate the λ and π vectors of all nodes in the network from the π vector of the root node (node H in our example) and the λ vectors of the other leaf nodes. Further at each node U the so called belief vector $\mathrm{BEL}(U)$ defined as $\Pr(U|e)$ can be calculated as a componentwise product of the λ and π vectors:

$$\mathrm{BEL}(U) = \Pr(U|e) = \alpha\lambda(U)\pi(U) = \frac{\lambda(U)\pi(U)}{\lambda(U)\cdot\pi(U)}, \tag{10}$$

where again the scalar α is chosen such as to normalize the resulting vector. The belief of a node is sometimes referred to as the posterior distribution of this event.

Going back to the example, the π vector of the root-node H is the prior distribution of H, $\Pr(H)$, describing the probability of there being a burglary at on any given day which

Holmes can be expected to have at least an estimate for. Further if the possible values of the variable (node) G are "Gibbon hears the alarm ringing" versus "Gibbon does not hear the alarm ringing", then the $\lambda(G)$ is either equal to $(0,1)$ or $(1,0)$ depending on Gibbon's testimony. The same is true for $\lambda(W)$ however depending on Watson's testimony. So from equations 7, 9 and 10 Holmes can calculate the belief of any node in the network, based on his knowledge of the testimonies, the probabilities $\Pr(H), \Pr(S|H), \Pr(G|S)$ and $\Pr(W|S)$. In particular he can calculate the belief of node H, which is the information he is interested in.

3 Analysis of Bayesian belief propagation

In the previous section we gave a brief introduction to the theory of Bayesian belief propagation using one of Pearl's examples. Even though the tree considered was of extreme simplicity, it should be clear that the belief propagation equations will hold for a tree of arbitrary complexity. Further more since the calculations are local at each node it is clear that even complex dependencies can be calculated easily and accurately by following the structure of the tree. The theory as described above is only applicable to simple trees, i.e. causal structures in which each event can have at most one cause, however a generalization to poly-trees, where multiple causes exist for an event, is also available, but not discussed in this paper.

How could an expert system make use of this approach for automated reasoning? According to the ideas put forward in the previous section, the knowledge base of such a system would consist of probability matrices of the form $\Pr(Y|X)$ for events X and Y known or observed to be in a causal relationship and prior distributions $\Pr(X)$. For a given problem it would then be the task of the system to propose a plausible network structure connecting the observed events with other unobserved but relevant events. After a network is constructed the belief propagation equations can be used to calculate the posterior probability distribution of the unobserved nodes of interest.

Thus the expert system has to cope with two different problems: a qualitative one, consisting of the construction of the graphical structure and a quantitative one propagating the λ and π vectors through this network.

Expert systems implementing the quantitative part have already been proposed by Lauritzen and Spiegelhalter [3] and others, however these systems are not able to do solve the first problem: the assignment of the graphical structure. Instead this structure is given in advance and hence forms part of the knowledge base.

While the studies on such expert systems are interesting and help to understand propagation mechanism they are of limited use in practise. It is infeasible for an expert system to maintain a very large tree connecting all possible (observed and unobserved) events. Such a network would connect seemingly unrelated events. Moreover it would require the propagation of the λ and π vectors from parts of the network that are only marginally related to the events of interest. It would also be extremely difficult to establish a large network connecting all nodes of interest and yet have it loop-free as required by the theory.

Instead one should look for an approach that constructs a network 'on the fly' connecting nodes when they become relevant in the light of observed events. This would entail establishing the relevance of various observed and unobserved events on the basis of the known $\Pr(Y|X)$ matrices and constructing a network which connects the unobserved events of interest to the relevant observed events, possibly creating new (previously unknown) unobserved events in the process. To the best of our knowledge we are at the moment at a lack of such an algorithm, but it is not infeasible that such an algorithm exists.

In section 6 of this paper we will use the theory of Belief trees to stochastically parse a sentence. In this paradigm the words of the sentence form the observed events. Unobserved events are grammatical markers such as "Noun phrase", "Prepositional phrase", etc. It is the object of the parser to connect the possible unobserved events to the observed events in the 'best possible way'. This will create a tree structure (the parse tree). The theory of Bayesian

networks will give us the quantitative tool for calculating the beliefs of all unobserved events and also the likelihood of the chosen tree structure.

4 Extensions to the basic belief propagation equations

With regard to section 6.2, we will now present a few extensions to the belief propagation equations. We will also provide the necessary derivations of all results. Since the equations quoted in the previous section are special cases of the results developed here, the proof apply to the previous section as well.

4.1 Dependence of daughter nodes

In the example discussed above, the daughters of a given parent node were assumed to be statistically independent. Thus the nature of Gibbon's testimony only depends marginally on Watson's testimony or expressed mathematically:

$$\Pr(G, W|S = s) = \Pr(G|S = s)\Pr(W|S = s), \tag{11}$$

for each possible s (state of the alarm system).

The belief propagation equations can be modified to apply in situations when equation 11 does not hold. This will be relevant in the second part of this paper, when we apply the theory to language acquisition.

If there are dependency relations among the daughter nodes, it is not sufficient to specify each parent-daughter relation separately, but rather jointly using a tensor of rank 3 or higher. This tensor expresses the joint conditional probabilities between daughter nodes $U_1, \ldots, U_n$ and parent V, viz $\Pr(U_1, \ldots, U_n|V)$ instead of separate link matrices $\Pr(U_1|V), \ldots, \Pr(U_n|V)$. In the most general case, the daughters of V can be divided into, say, G groups. Nodes within one group are regarded as statistically independent from the nodes of any other group, so there is no need to specify the parent-daughter relation by one big tensor. However the nodes within each group are considered to be statistically dependent and their relation to the parent is expressed by a tensor for this group. Thus if the nodes in group g are labelled $U_1^g, \ldots, U_{n_g}^g$ the overall conditional probability tensor has the form

$$\Pr(U_1^1, \ldots, U_{n_1}^1, U_1^2, \ldots, U_{n_2}^2, \ldots, U_1^G, \ldots, U_{n_G}^G|V) = \prod_{g=1}^{G} \Pr(U_1^g, U_2^g, \ldots, U_{n_g}^g|V) \tag{12}$$

Figure 4 displays such a scenario graphically.

The belief propagation equations can be adopted in a straight forward way to the new situation. For convenience we define additional notation. We denote the group consisting of nodes $U_1^g, \ldots, U_{n_g}^g$ by V^g. Further we define the evidence $e_{V^g}^-$ as the combined evidence $e_{U_1^g}^-, \ldots, e_{U_{n_g}^g}^-$ and likewise $e_{V^g}^+$ as the remaining evidence $e \backslash e_{V^g}^-$, i.e. the entire evidence e less $e_{V^g}^-$. Furthermore we define

$$\lambda(V^g) = \Pr(e_{V^g}^-|V) \tag{13}$$

$$\pi(V^g) = \Pr(V|e_{V^g}^+), \tag{14}$$

and we write A^g to indicate the tensor for group g, i.e.

$$A^g_{i,j_1,\ldots,j_{n_g}} = \Pr(U_1^g{=}j_1, \ldots, U_{n_g}^g{=}j_{n_g}|V{=}i). \tag{15}$$

To keep the derivation readable we will make a further simplification from now on: We will identify a possible value x_i of a random variable X with the index i itself. Thus we

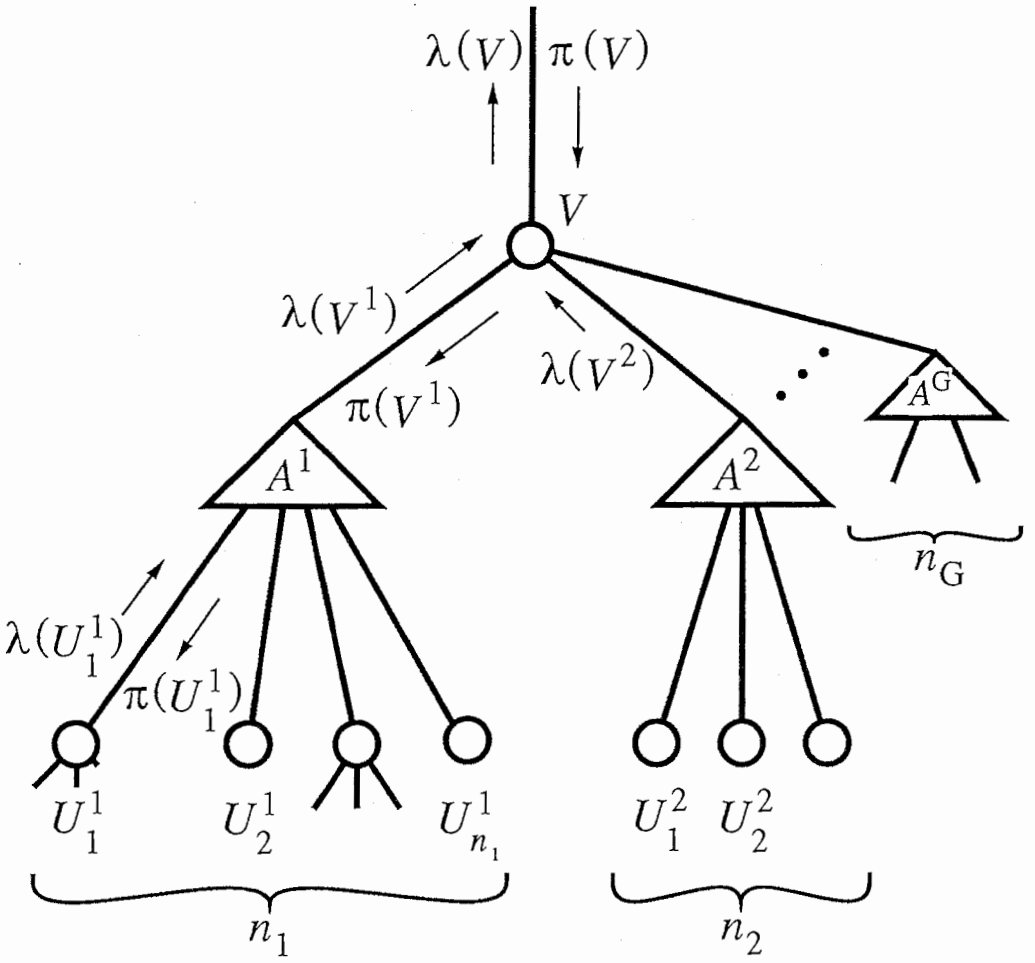

図 4: This figure shows a node V with daughters $U_1^1, U_2^1, \ldots, U_{n_1}^1, U_1^2, \ldots, U_{n_2}^2, \ldots, U_{n_G}^G$. The daughter nodes are divided into G groups as shown. The groups are regarded as statistically independent, but the nodes within each group are dependent. Hence the conditional probability distribution of the daughters given the parent has the product form shown in equation 12.

will write $\Pr(X = i)$ instead of $\Pr(X = x_i)$. In fact, in equation 15, we already used this simplification.

4.2 Applicable laws of probability

In order to derive equations 6 to 9 and their generalizations we require three laws of probability theory. These are descibed below:

(4.2.1) Bayes' law

Bayes' law states that for events a, b, c we have

$$\Pr(a \mid b, c) = \frac{\Pr(b \mid a, c)\Pr(a \mid c)}{\Pr(b \mid c)}. \tag{16}$$

Now, let U be a random variable corresponding to a node of the network, so that in our notation $\Pr(U \mid b, c)$ is a vector. This vector must be normalized. We can then write Bayes' law in the following form:

$$\Pr(U \mid b, c) = \alpha \Pr(b \mid U, c)\Pr(U \mid c), \tag{17}$$

where $\alpha = 1/\Pr(b \mid c)$ can be determined simply as a normalizing constant.

(4.2.2) Conditioning

Let a and b be events and U a random variable that can take a finite number of values $1, \ldots, n$. The following equation holds:

$$\Pr(a \mid b) = \sum_i \Pr(a \mid U, b)_i \Pr(U \mid b)_i \tag{18}$$

(4.2.3) Separation

To illustrate the law of separation consider the probability vector $\Pr(e_U^- \mid e_U^+, U)$. Since the graphical structure is assumed to be a tree, the value of e_U^+ can effect the value of e_U^- only via the value of U. We say e_U^+ and e_U^- are marginally independent, i.e independent when conditioned on the value of U. Hence

$$\Pr(e_U^- \mid e_U^+, U) = \Pr(e_U^- \mid U). \tag{19}$$

We also say "U separates e_U^+ from e_U^-".

4.3 Derivation of the propagation equations

We will begin by proving a generalization of equation 6. The captial letters B, C S and D on top of the equal sign (=) indicate applications of Bayes' Law, the laws of Conditioning and Separation and a definition respectively. Finally a bracketed numeral indicates an application of an earlier equation.

Now, as for the λs we have:

$$\begin{aligned}
\lambda(V^g)_i &\stackrel{D}{=} \Pr(e^-_{V^g}|V=i) && (20)\\
&\stackrel{D}{=} \Pr(e^-_{U^g_1},\ldots,e^-_{U^g_{n_g}}|V=i) && (21)\\
&\stackrel{C}{=} \sum_{j_1,\ldots,j_{n_g}} \Pr(e^-_{U^g_1},\ldots,e^-_{U^g_{n_g}}|V=i, U^g_1=j_1,\ldots U^g_{n_g}=j_{n_g}) A^g_{i,j_1,\ldots,j_{n_g}} && (22)\\
&\stackrel{S}{=} \sum_{j_1,\ldots,j_{n_g}} \prod_{k=1}^{n_g} \Pr(e^-_{U^g_k}|U^g_k=j_k) A^g_{i,j_1,\ldots,j_{n_g}} && (23)\\
&\stackrel{D}{=} \sum_{j_1,\ldots,j_{n_g}} \prod_{k=1}^{n_g} \lambda(U^g_k)_{j_k} A^g_{i,j_1,\ldots,j_{n_g}} && (24)
\end{aligned}$$

$$\begin{aligned}
\lambda(V)_i &\stackrel{D}{=} \Pr(e^-_V|V=i) && (25)\\
&\stackrel{D}{=} \Pr(e^-_{V^1},\ldots,e^-_{V^G}|V=i) && (26)\\
&= \prod_{g=1}^{G} \Pr(e^-_{V^g}|V=i) && (27)\\
&\stackrel{D}{=} \prod_{g=1}^{G} \lambda(V^g)_i && (28)
\end{aligned}$$

Let us reflect on this result for a moment. Equation 24 expresses the vector $\lambda(V^g)$ as a simple tensor-vector product between the tensor A^g and the λ-vectors $\lambda(U^g_k)$ $(k=1,\ldots,n_g)$. Regarding Figure 4 one could say that the vectors $\lambda(U^g_k)$ enter the triangle (tensor) A^g from the bottom. Here the tensor-vector product is formed and the λ vector representing the group $\lambda(V^g)$ is emitted at the top. Equation 28 shows how the λ vectors from the various groups need to be combined to give the overall λ vector: by component-wise vector product. One could write these two results in the vector form:

$$\begin{aligned}
\lambda(V^g) &= A^g_{i,j_1,\ldots,j_{n_g}} \lambda(V^g_1)\cdots\lambda(V^g_{n_g}) && (29)\\
\lambda(V) &= \prod_{g=1}^{G} \lambda(V^g), && (30)
\end{aligned}$$

where the multiplication in equation 29 is a tensor-vector product (like a generalized matrix-vector product) and the multiplication in equation 30 is a componentwise vector product. However since it is difficult to distinguish between the two forms of multiplication in this notation and it is also not easy to see which indices of the tensor A^g identify with which λ vector in equation 29, we will always use the more explicit component-wise description shown in equations 24 and 28.

For the πs we derive

$$\begin{aligned}
\pi(V^g)_i &\stackrel{D}{=} \Pr(V=i|e^+_{V^g}) && (31)\\
&\stackrel{D}{=} \Pr(V=i|e^+_V, e^-_{V^1},\ldots,\not{e}^-_{V^g},\ldots,e^-_{V^G}) && (32)\\
&\stackrel{B}{=} \alpha\Pr(V=i, e^-_{V^1},\ldots,\not{e}^-_{V^g},\ldots,e^-_{V^G}|e^+_V) && (33)\\
&\stackrel{C}{=} \alpha\Pr(V=i|e^+_V)\Pr(V=i, e^-_{V^1},\ldots,\not{e}^-_{V^g},\ldots,e^-_{V^G}|e^+_V, V=i) && (34)\\
&\stackrel{S}{=} \alpha\Pr(V=i|e^+_V) \prod_{g'=1,\ldots,\not{g},\ldots,G} \Pr(e^-_{V^{g'}}|V=i) && (35)
\end{aligned}$$

$$\stackrel{D}{=} \alpha\pi(V)_i \prod_{g'=1,\ldots,\not{g},\ldots,G} \lambda(V^{g'})_i \tag{36}$$

$$\pi(U_k^g)_{j_k} \stackrel{D}{=} \Pr(U_k^g{=}j_k | e^+_{U_k^g}) \tag{37}$$

$$\stackrel{D}{=} \Pr\left(U_k^g{=}j_k \middle| e^+_{V^g}, \bigwedge_{k'\neq k} e^-_{U_{k'}^g}\right) \tag{38}$$

$$\stackrel{C}{=} \sum_{i,j_1,\ldots,\not{j_k},\ldots,j_{n_g}} \left\{ \begin{array}{r} \Pr\left(U_k^g{=}j_k \middle| V{=}i, \left(\bigwedge_{k'\neq k} U_{k'}^g{=}j_{k'}\right), e^+_{V^g}, \bigwedge_{k'\neq k} e^-_{U_{k'}^g}\right) \\ \Pr\left(V{=}i, \bigwedge_{k'\neq k} U_{k'}^g{=}j_{k'} \middle| e^+_{V^g}, \bigwedge_{k'\neq k} e^-_{U_{k'}^g}\right) \end{array} \right\} \tag{39}$$

$$\stackrel{S,B}{=} \alpha_1 \sum_{i,j_1,\ldots,\not{j_k},\ldots,j_{n_g}} \left\{ \begin{array}{r} \Pr\left(U_k^g{=}j_k \middle| V{=}i, \bigwedge_{k'\neq k} U_{k'}^g{=}j_{k'}\right) \Pr(V{=}i) \\ \Pr\left(\bigwedge_{k'\neq k} U_{k'}^g{=}j_{k'}, e^+_{V^g}, \bigwedge_{k'\neq k} e^-_{U_{k'}^g} \middle| V{=}i\right) \end{array} \right\} \tag{40}$$

$$\stackrel{B}{=} \alpha_1 \sum_{i,j_1,\ldots,\not{j_k},\ldots,j_{n_g}} \left\{ \begin{array}{r} \dfrac{\Pr\left(\bigwedge_{k'=1,\ldots,n_g} U_{k'}^g{=}j_{k'} \middle| V{=}i\right)}{\Pr\left(\bigwedge_{k'\neq k} U_{k'}^g{=}j_{k'} \middle| V{=}i\right)} \Pr(V{=}i) \\ \Pr\left(\bigwedge_{k'\neq k} U_{k'}^g{=}j_{k'}, e^+_{V^g}, \bigwedge_{k'\neq k} e^-_{U_{k'}^g} \middle| V{=}i\right) \end{array} \right\} \tag{41}$$

$$\stackrel{D,B}{=} \alpha_1 \sum_{i,j_1,\ldots,\not{j_k},\ldots,j_{n_g}} A^g_{i,j_1,\ldots,j_{n_g}} \Pr(V{=}i) \Pr\left(e^+_{V^g}, \bigwedge_{k'\neq k} e^-_{U_{k'}^g} \middle| \bigwedge_{k'\neq k} U_{k'}^g{=}j_{k'}, V{=}i\right) \tag{42}$$

$$\stackrel{S}{=} \alpha_1 \sum_{i,j_1,\ldots,\not{j_k},\ldots,j_{n_g}} A^g_{i,j_1,\ldots,j_{n_g}} \Pr(V{=}i) \Pr(e^+_{V^g}|V{=}i) \prod_{k'=1,\ldots,\not{k},\ldots,n_g} \Pr\left(e^-_{U_{k'}^g} \middle| U_{k'}^g{=}j_{k'}\right) \tag{43}$$

$$\stackrel{D}{=} \alpha_2 \sum_{i,j_1,\ldots,\not{j_k},\ldots,j_{n_g}} A^g_{i,j_1,\ldots,j_{n_g}} \pi(V^g)_i \prod_{k'=1,\ldots,\not{k},\ldots,n_g} \lambda(U_{k'}^g)_{j_{k'}} \tag{44}$$

Here $1,\ldots\not{k}\ldots,n_g$ stands for the integers from 1 to n_g with the exception of k. The index k' generally ranges over these integers except in the numerator of equation 41 where it covers the full range $1,\ldots,n_g$. The constants α_1 and α_2 are given by

$$\alpha_1 = \frac{1}{\Pr\left(e^+_{V^g}, \bigwedge_{k'\neq k} e^-_{U_{k'}^g}\right)} \quad \text{and} \quad \alpha_2 = \frac{\Pr(e^+_{V^g})}{\Pr\left(e^+_{V^g}, \bigwedge_{k'\neq k} e^-_{U_{k'}^g}\right)}, \tag{45}$$

but since these constants are independent of j_k and since we know that the vector $\pi(U_k^g)$ is normalized, they can simply be determined as normalizing constants.

Thus it is clear that even in the presence of statistical dependence among the daughter nodes, the λ and π vectors can be propagated in a fairly straight forward fashion.

4.4 Learning the conditional probabilities

In this section we will discuss a method of how the conditional probabilities stored in the matrices can be learned from a set of training samples. A training sample is an instantiation of the observed nodes of a network. The training problem is that of choosing the link matrices such that the overall probability of the training material (i.e. the product of probabilities of each sample given the link matrices) is maximized. We will distinguish two cases: The simple case in which all nodes are observed and the more complicated case involving matrices between unobserved nodes.

(4.4.1) Case 1: all nodes are observed

Suppose we wish to find the link matrix $\Pr(Y|X)$ between two nodes X and Y. In this case we simply instantiate a matrix of counters $(C(X,Y)_{ij})$ which counts the number of times node X is in state i and node Y is in state j. After processing the entire training data and updating the respective counters, the maximum likelihood estimate of the matrix $\Pr(Y|X)$ is then given by

$$\Pr(Y = y_j|X = x_i) = \frac{C(X,Y)_{ij}}{\sum_{j'} C(X,Y)_{ij'}} \tag{46}$$

Now consider the slightly more complicated case in which a node V has n daughters structured in G groups. $U_1^1, U_2^1, \ldots, U_{n_1}^1, U_1^2, \ldots, U_{n_2}^2, \ldots, U_{n_G}^G$ that was discussed in section 4.1. Here the relationship between the parent V and the daughters $U_1^g, U_2^g, \ldots, U_{n_g}^g$ in group g is expressed in the tensor shown in equation 15. If the nodes $V, U_1^g, U_2^g, \ldots, U_{n_g}^g$ are all observed we can again simply instantiate a tensor of counters $C(V, U_1^g, \ldots, U_{n_g}^g)_{ij_1\ldots j_{n_g}}$ and count the number of occurances of each event over the training data. The maximum likelihood estimate for A^g is then

$$A^g_{i,j_1,\ldots,j_{n_g}} = \frac{C(V, U_1^g, \ldots, U_n^g)_{ij_1\ldots j_{n_g}}}{\sum_{j_1',\ldots,j_{n_g}'} C(V, U_1^g, \ldots, U_n^g)_{ij_1'\ldots j_{n_g}'}} \tag{47}$$

(4.4.2) Case 2: some nodes are unobserved

If the nodes involved are unobserved, we can no longer simply count simultaneous occurances. However we can ask for the expected number of simultaneous occurances. For a single sample e this expected number equals the probability

$$\Pr(V{=}i, U_1^g{=}j_1, \ldots, U_n^g{=}j_{n_g}|e) \tag{48}$$

Here again, we split the daughter nodes into different groups according to their dependencies and only consider one group as it is independent of all the others. Fortunately, there is an easy way to calculate this quantity using intermediate results from the belief propagation. We have

$$\begin{aligned}
&\Pr(V{=}i, U_1^g{=}j_1, \ldots, U_n^g{=}j_{n_g}|e) \\
&\quad \stackrel{B}{=} \Pr(V{=}i|e)\Pr(U_1^g{=}j_1, \ldots, U_n^g{=}j_{n_g}|V{=}i, e) && (49)\\
&\quad \stackrel{D}{=} \mathrm{BEL}(V)_i \Pr(U_1^g{=}j_1, \ldots, U_n^g{=}j_{n_g}|V{=}i, e_{\bar{V}^g}) && (50)
\end{aligned}$$

$$\overset{(10),B}{=} \alpha\lambda(V)_i\pi(V)_i\frac{\Pr(e_{V^g}^-|V=i,U_1^g=j_1,\ldots,U_n^g=j_{n_g})\Pr(U_1^g=j_1,\ldots,U_n^g=j_{n_g}|V=i)}{\Pr(e_{V^g}^-|V=i)} \quad (51)$$

$$\overset{(28),D}{=} \alpha\prod_{g'=1}^{G}\lambda(V^{g'})_i\pi(V)_i\frac{\Pr(e_{V^g}^-|U_1^g=j_1,\ldots,U_n^g=j_{n_g})A^g_{i,j_1,\ldots,j_{n_g}}}{\lambda(V^g)_i} \quad (52)$$

$$= \alpha\prod_{g'=1,\ldots,\not{g},\ldots,G}\lambda(V^{g'})_i\pi(V)_i\Pr(e_{U_1^g}^-,\ldots,e_{U_{n_g}^g}^-|U_1^g=j_1,\ldots,U_n^g=j_{n_g})A^g_{i,j_1,\ldots,j_{n_g}} \quad (53)$$

$$\overset{D,S}{=} \alpha\pi(V^g)_i\prod_{k=1}^{n_g}\Pr(e_{U_k^g}^-|U_k^g=j_k)A^g_{i,j_1,\ldots,j_{n_g}} \quad (54)$$

$$\overset{D}{=} \alpha\pi(V^g)_i\prod_{k=1}^{n_g}\lambda(U_k^g)_{j_k}A^g_{i,j_1,\ldots,j_{n_g}} \quad (55)$$

Instead of counting joint occurances, we now add the expected number of joint occurances in the tensor $C(V,U_1^g,\ldots,U_{n_g}^g)$. Hence after processing all samples e in the data we obtain

$$C(V,U_1^g,\ldots,U_{n_g}^g) = \sum_e \Pr(V,U_1^g,\ldots,U_{n_g}^g|e), \quad (56)$$

where the sum denotes tensor addition. An estimate for the conditional probability tensor A^g can now be obtained using equation 47. It should be noted that since equation 48 represents the expected number of joint occurances for one sample, equation 55 represents the expected number of joint occurances over the entire training data.

Note however that a previous estimate of A^g is required in equation 55. Hence, unlike the 'all nodes observed' case the tensors can only be learned iteratively. We therefore have the following learning algorithm:

1. Choose initial (perhaps random) connection tensors.

2. Process the training data once, accumulating the tensor $(\Pr(V=i,U_1^g=j_1,\ldots,U_n^g=j_{n_g}|e))$ in a 'counting' tensor $C(V,U_1^g,\ldots,U_n^g)$, for each parameter tensor in the network.

3. Normalize the C tensors and obtain the new estimates for the A tensors viz

$$A^g_{i,j_1,\ldots,j_{n_g}} = \frac{C(V,U_1^g,\ldots,U_n^g)_{ij_1\ldots j_{n_g}}}{\sum_{j_1',\ldots,j_{n_g}'}C(V,U_1^g,\ldots,U_n^g)_{ij_1'\ldots j_{n_g}'}} \quad (57)$$

4. Go back to step 2 until convergence is achieved.

We will show in the next section that re-estimating the parameters in this way is guaranteed to increase the overall likelihood of the training data. Hence this iterative technique converges. However it may not converge to the maximum likelihood estimate of the parameter tensors. Instead it may converge to a 'local maximum likelihood' estimator, i.e. a point in the parameter space where likelihood is locally maximal.

5 A theorem about convergence of the iterative training method

In this section we will show that the previously described method for updating the probability estimates of the link tensors produces always improves the overall likelihood of the training data. In order to state the theorem we need to do some preliminary work:

Figure 5 shows three causal trees. The nodes of the trees are represented as circles and squares. The squares represent *evidence nodes*, i.e. nodes for which the state can be observed. The circles represent unobserved nodes. The triangles represent connection tensors. As can be seen, the daughters of a given parent are sometimes assumed to be marginally independent (when each link has its own tensor) and sometimes exhibit dependencies (when certain links share tensors). The trees are said to model the observed nodes. The parameters of the model consist of the components of each of the connection tensors as well as the prior vectors of the root nodes (π_1, π_2 and π_3 in Figure 5).

5.1 Some definitions

Samples A sample is an assignment values to the evidence nodes in the tree. We consider two different kinds of samples:

Deterministic samples provide a specific value for each of the evidence nodes.

Non-deterministic samples assign to each evidence node a likelihood vector. This is a vector of non-negative real values describing the likelihood of some event given the value of the evidence node.

Deterministic events are a subclass of non-deterministic events, namely those in which each likelihood vector has the form $(0,\ldots,0,1,0,\ldots,0)$.

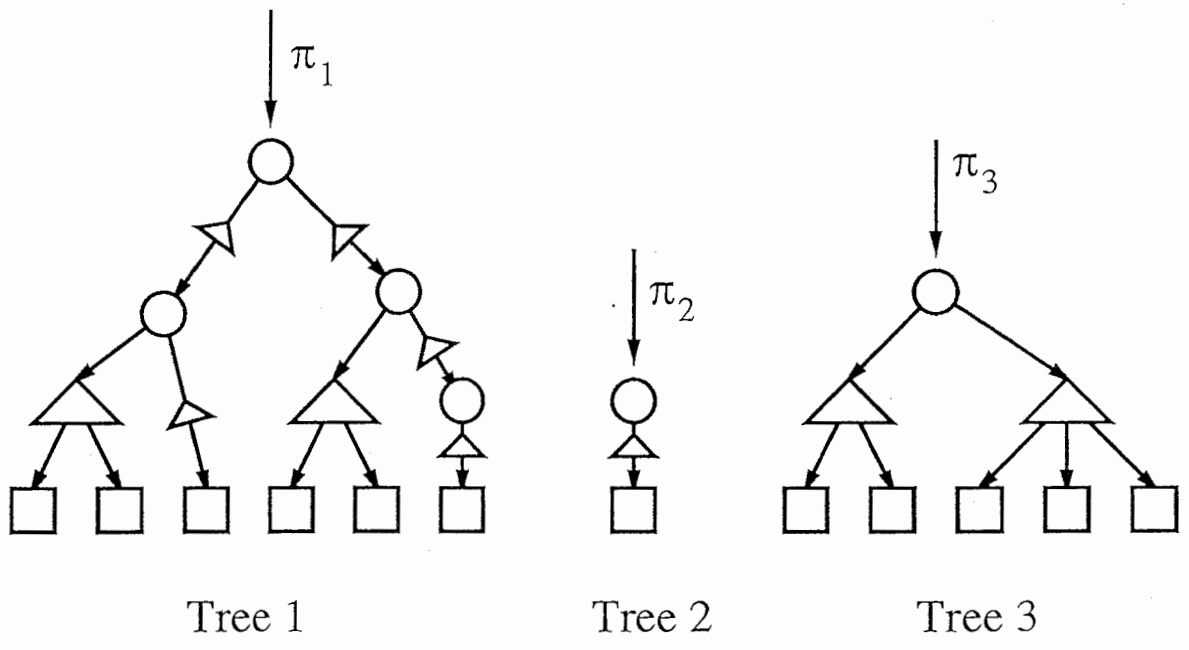

図 5:

Entropy of a sample For a given sample, we can calculate the probability of this sample given the model parameters. To do this recall that the λ and π vectors at a node X were defined

$$\lambda(X) = \Pr(e_X^-|X) \tag{58}$$
$$\pi(X) = \Pr(X|e_X^+) \tag{59}$$

Thus

$$\lambda(X) \cdot \pi(X) = \sum_i \Pr(e_X^-|X{=}i) \Pr(X{=}i|e_X^+) \tag{60}$$
$$= \sum_i \Pr(e_X^-|X{=}i, e_X^+) \Pr(X{=}i|e_X^+) \tag{61}$$
$$= \Pr(e_X^-|e_X^+) \tag{62}$$

Line 61 is justified, for the node X separates e_X^- from e_X^+ and so the probability of e_X^- only depends on the state of X. At the root node of a tree, $e_X^+ = \emptyset$, so $\lambda \cdot \pi = \Pr(e)$, where e stands for the part of the evidence spanned by the tree. When there are multiple trees as in Figure 5, the fact that these are not connected implies that they span independent parts of the sample. Thus the overall probability of the sample is obtained by multiplying the terms $\lambda(R) \cdot \pi(R)$ over all root nodes R.

It is usually more convenient however to use the negative logarithm of this quantity. This is known as the entropy of the sample. We define the entropy of a tree T with root node R as

$$E(T) = -\log(\lambda(R) \cdot \pi(R)) \tag{63}$$

and the entropy of the sample as

$$E(T) = -\sum_{\text{root nodes } R} \log(\lambda(R) \cdot \pi(R)) \tag{64}$$

Training data The training data is a set of samples. The entropy of the training data given the current model is defined as the sum of the entropies of the samples in the training data.

The training problem The training problem is that of finding model parameters such that the overall entropy of the training data is minimized. We will show that the iterative algorithm developed in the previous section always decreases the entropy of the training data.

5.2 Parameterization of the model

Figure 6 shows, how the isolated trees of Figure 5 can be converted into a single equivalent tree. This is done by introducing a new root node R and connecting the previous root nodes as daughters to this node. The new root node represents a random variable that can only take one value (thus it is not random at all). The components of the connection matrix connecting R to one of the old root nodes R_i are set identical to the components of the prior π_i of R_i in Figure 5 (Since R can only take one value the connection matrix is really just a

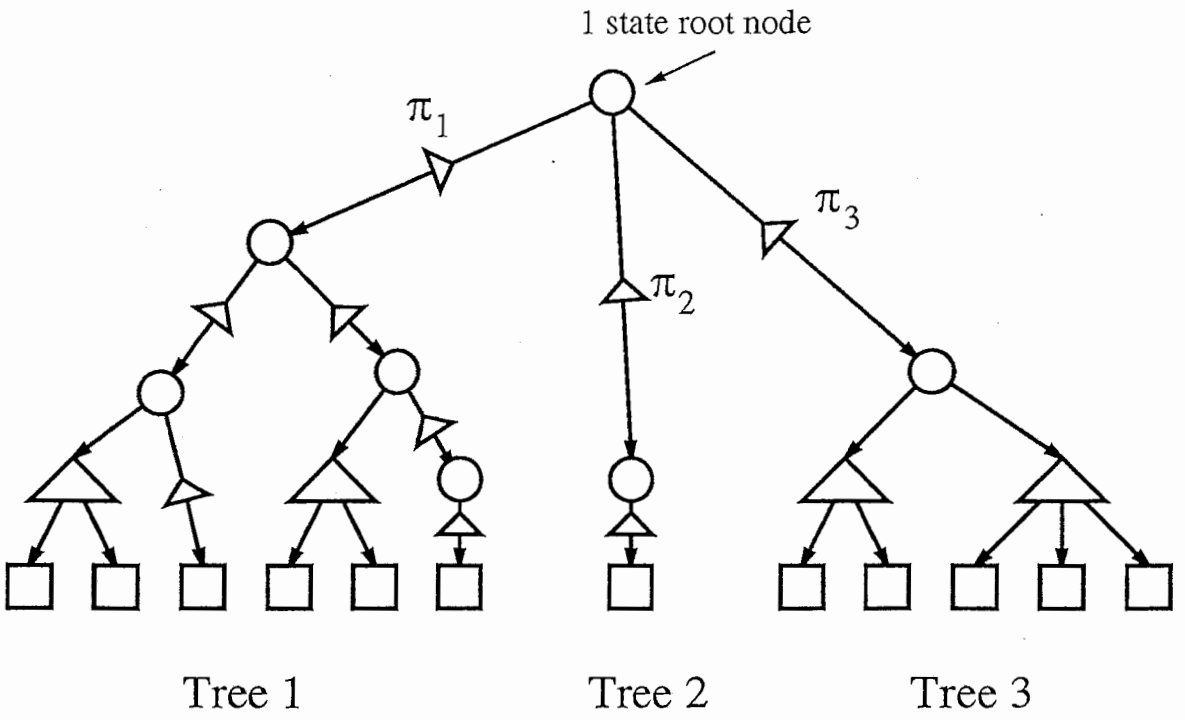

図 6:

vector.). We do not need to consider the prior to R, for since R only takes one value this prior is always equal to the unit vector (1).

It should thus be clear that Figure 6 is equivalent to Figure 5. The advantage of this transformation is that we only need to consider one tree and that prior vectors are represented as connection tensors, so a re-estimation formula for the connection tensors will automatically apply to the priors as well.

The total set of parameters to be estimated thus consists of the components of the various connection tensors in Figure 6.

Weight linkage Since the latter part of this paper makes extensive use of a concept known as weight linkage we will introduce this concept here: We may impose additional constraints on a model by forcing certain connection tensors to have identical components. Thus if two tensors A^x and A^y of the model have the same rank (number of indices) and corresponding indices have the same dimension we can impose the additional constraint:

$$A^x_{ij_1\ldots j_n} = A^y_{ij_1\ldots j_n} \qquad \text{for all } i, j_1, j_2, \ldots, j_n \tag{65}$$

This reduces the number of free parameters of the model.

In the following we will consider different models, which although having the same geometrical structure (or topology) differ in the actual parameter values. To differentiate between two such models we will use the letter Θ to indicate one parameter set and $\bar{\Theta}$ to indicate another. Thus the probability of a sample e under model Θ will be written as $\Pr(e|\Theta)$ and that of e under model $\bar{\Theta}$ will be written as $\Pr(e|\bar{\Theta})$. More specifically Θ is defined as the set of all connection tensors of the model and we will write $A \in \Theta$ to indicate

that a given tensor A is part of model Θ. Naturally Θ can be partitioned into a set of subsets

$$\Theta = \bigcup_h \Theta^h, \tag{66}$$

where each subset Θ^h contains all tensors that are linked. (Linking of tensors really defines an equivalence relation on Θ and Θ^h are the equivalence classes.) In slight abuse of notation we will also write

$$(A : V, U_1, \ldots, U_n) \in \Theta^h \tag{67}$$

to indicate that tensor A, which links the parent node V to the daughter nodes U_1 to U_n belongs to the tensors in Θ^h.

We can now state the main result of this section:

Theorem 1 *Let Θ be a parameter set for a model describing samples e of a training data set* Train. *Let E_Θ be the entropy of the training data given the parameter set Θ. Then if we choose a new parameter set $\bar{\Theta}$ for the same model such that for any tensor $\bar{A} \in \bar{\Theta}^h$ we have*

$$\bar{A}_{ij_1\ldots j_n} = \frac{C^h_{ij_1\ldots j_n}}{\sum\limits_{j'_1,\ldots,j'_n} C^h_{ij'_1\ldots j'_n}}, \tag{68}$$

where

$$C^h_{ij_1\ldots j_n} = \sum_{e\in\text{Train}} \sum_{(A:V,U_1,\ldots,U_n)\in\Theta^h} \Pr\left(V{=}i, \bigwedge_k U_k{=}j_k \middle| e, \Theta\right) \tag{69}$$

and denote $E_{\bar{\Theta}}$ the entropy of the training data given $\bar{\Theta}$ we have

$$E_{\bar{\Theta}} \leq E_\Theta \tag{70}$$

Proof: We will first consider the case, in which the training data Train consists of just one sample e. Let $\mathcal{N}$ denote the total set of nodes in the model and let

$$S : \mathcal{N} \mapsto \mathbb{N} \tag{71}$$

be a function that assigns a particular value to each node in the model and let $\mathcal{S}$ be the set of all such functions. Further for the sample e and a value allocation S we denote

$$\Pr(e, S|\Theta) = \Pr\left(e, \bigwedge_{U\in\mathcal{N}} U{=}S(U) \middle| \Theta\right). \tag{72}$$

A moments thought will reveal that

$$\Pr(e, S|\Theta) = \prod_{(A:V,U_1,\ldots,U_n)\in\Theta} A_{S(V),S(U_1),\ldots,S(U_n)} \prod_{\text{leaf nodes } X} \lambda(X)_{S(X)}. \tag{73}$$

Now for two different sets of parameters Θ and $\bar{\Theta}$ define the function

$$Q(\Theta, \bar{\Theta}) = \frac{1}{\Pr(e|\Theta)} \sum_S \Pr(e, S|\Theta) \log \Pr(e, S|\bar{\Theta}) \tag{74}$$

and consider the term $E_\Theta - E_{\bar{\Theta}}$. By the concavity of the log function we have:

$$\begin{aligned} E_\Theta - E_{\bar{\Theta}} &= \log \frac{\Pr(e|\bar{\Theta})}{\Pr(e|\Theta)} && (75)\\ &= \log\left(\sum_S \frac{\Pr(e,S|\Theta)}{\Pr(e|\Theta)} \frac{\Pr(e,S|\bar{\Theta})}{\Pr(e,S|\Theta)}\right) && (76)\\ &\geq \sum_S \frac{\Pr(e,S|\Theta)}{\Pr(e|\Theta)} \log \frac{\Pr(e,S|\bar{\Theta})}{\Pr(e,S|\Theta)} && (77)\\ &= Q(\Theta,\bar{\Theta}) - Q(\Theta,\Theta) && (78) \end{aligned}$$

It follows that a sufficient condition for $E_{\bar{\Theta}} \leq E_\Theta$ is that $Q(\Theta,\bar{\Theta}) \geq Q(\Theta,\Theta)$. This latter condition is clearly satisfied for a $\bar{\Theta}$ that maximizes $Q(\Theta,\cdot)$. In fact we have strict inequality:

$$E_{\bar{\Theta}} < E_\Theta \tag{79}$$

unless Θ itself maximizes $Q(\Theta,\cdot)$. We therefore ask for the set of parameters $\bar{\Theta}$ that maximizes $Q(\Theta,\cdot)$ for a given Θ. From equation 73 we have

$$\log \Pr(e,S|\bar{\Theta}) = \sum_{(\bar{A}:V,U_1,\ldots,U_n)\in\bar{\Theta}} \log \bar{A}_{S(V),S(U_1),\ldots,S(U_n)} + \sum_{\text{leaf nodes } X} \log \lambda(X)_{S(X)}. \tag{80}$$

Substituting this into equation 74 and changing the order of summation gives:

$$Q(\Theta,\bar{\Theta}) = \sum_{\bar{\Theta}^h\subset\bar{\Theta}} \sum_{i,j_1,\ldots,j_n} C^h_{ij_1\ldots j_n} \log \bar{A}_{ij_1\ldots j_n} + \sum_{\text{leaf nodes } X} \sum_i D(X)_i \log \lambda(X)_i, \tag{81}$$

where the tensor $\bar{A}_{ij_1\ldots j_n}$ is the representative tensor of the class $\bar{\Theta}^h$ and

$$\begin{aligned} C^h_{ij_1\ldots j_n} &= \sum_S \sum_{(A:V,U_1,\ldots,U_n)\in\Theta^h} I(S(V)=i) \prod_{k=1}^n I(S(U_k)=j_k) \frac{\Pr(e,S|\Theta)}{\Pr(e|\Theta)} && (82)\\ &= \sum_{(A:V,U_1,\ldots,U_n)\in\Theta^h} \sum_{S|S(V)=i,\bigwedge_k S(U_k)=j_k} \Pr(S|e,\Theta) && (83)\\ &= \sum_{(A:V,U_1,\ldots,U_n)\in\Theta^h} \Pr\left(V=i, \bigwedge_{k=1}^n U_k=j_k \,\middle|\, e,\Theta\right) && (84) \end{aligned}$$

and

$$\begin{aligned} D(X)_i &= \sum_S I(S(X)=i) \Pr(S|e,\Theta) && (85)\\ &= \Pr(X=i|e,\Theta) && (86)\\ &= \mathrm{BEL}_\Theta(X)_i. && (87) \end{aligned}$$

The second term in equation 81 is independent of $\bar{\Theta}$ so it remains to maximize

$$\sum_{\bar{\Theta}^h\subset\bar{\Theta}} \sum_{i,j_1,\ldots,j_n} C^h_{ij_1\ldots j_n} \log \bar{A}_{ij_1\ldots j_n} \tag{88}$$

with respect to the components of the representative tensor $\bar{A}_{ij_1\ldots j_n}$ of $\bar{\Theta}^h$ subject to the constraints

$$\sum_{j_1,\ldots,j_n} \bar{A}_{ij_1\ldots j_n} = 1 \tag{89}$$

It is easy to show (for example by using Lagrangian multipliers) that this maximization is in fact solved by:

$$\bar{A}_{ij_1\ldots j_n} = \frac{C^h_{ij_1\ldots j_n}}{\sum_{j'_1,\ldots,j'_n} C^h_{ij'_1\ldots j'_n}}, \tag{90}$$

This completes the proof of the Theorem for the special case of there being just one sample e in the training set. In the case of n samples, we simply create one big sample by writing all n samples one after each other. We use a model which consists simply of n copies of the model for one sample, with the corresponding tensors linked across all copies. This big model applied to the big sample is equivalent to the original model applied repeatedly to all samples in the training data. Carrying out the same analysis as above for the big model leads to equations 68 and 69 as claimed. ■

The usefulness of Theorem 1 should be clear. The partial contributions

$$\Pr\left(V{=}i, \bigwedge_{k=1}^{n} U_k{=}j_k \,\middle|\, e, \Theta\right) \tag{91}$$

to the C^h tensor can readily be calculated using equation 55. After processing the training data once, one merely needs to re-normalize the C^h tensors to obtain a new estimate of the model parameters. The theorem guarantees that the new set of parameters models the training data better than the previous one.

6 Learning the hidden structure of language

In this section we will study whether and how it is possible to employ the above ideas in order to learn the structure of natural language. Basically the problem we wish to study is this: We are given a stream of symbols (e.g. words), which exemplifies the language we wish to study. We are seeking an algorithm that will learn a probability distribution that most closely describes the observed symbol sequence.

6.1 Production rules seen as causalities

We begin with an informal discussion of grammars. Traditionally grammars of a language (e.g. the English language) are written in the form of production rules. A typical rule could be of the form

$$\text{NP} \rightarrow \text{Art Noun}, \tag{92}$$

meaning a grammar symbol NP (which stands for noun phrase) can be replaced by an article (Art) followed by a noun. Typically a grammar consists of many such rules. The symbols involved are either abstract grammatical terms such as "NP," "Art," "Noun" or specific words of the language. For instance we might have the additional rules

$$\text{Art} \rightarrow \text{the} \tag{93}$$

$$\text{Noun} \rightarrow \text{book} \tag{94}$$

In linguistic terminology, words of the language such as "the" and "book" are referred to as terminal symbols of the grammar and abstract units like "NP", "Noun" are referred to as non-terminal symbols, because these units need to be replaced by strings of terminal symbols to make a valid sentence.

This paper takes the approach that a rule such as

$$\text{NP} \rightarrow \text{Art Noun}, \tag{95}$$

can be viewed as a causal relation, i.e. we have and abstract cause "NP" that causes the creation of two abstract units "Art" and "Noun", which in turn will eventually cause the creation of an article and a noun as terminal symbols.

In the problem of language learning we are thus anticipating a hidden causal structure that led to words (or terminal symbols) we are able to observe. The problem is that of discovering this structure (or rather of finding the most suitable structure) and learning the probability matrices involved in the productions.

We are thus faced with two problems: A qualitative one of assigning a causal tree (or in the language of grammarians a parse tree) to the string of terminal symbols observed and that of finding the probabilities associated with each link in the graph. The former will be referred to as the *Structure problem*, the latter as the *Assignment problem*.

Of these two problems, the Assignment problem can essentially be solved using the belief propagation and probability learning equations introduced in sections 2 and 4. The Structure problem is however more difficult. A variety of approaches to solve this problem will be discussed.

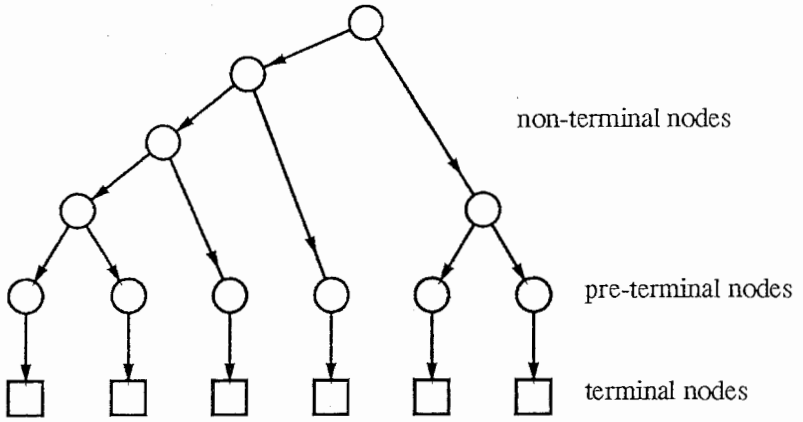

図 7:

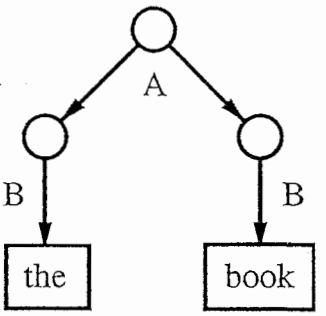

図 8:

6.2 A specific grammar model

In order to simplify the Structure problem, we will consider a representation of a Grammar in a special form. This imposes constraints on the possible structures that a parse tree can take and reduces the search space for finding an appropriate one. The particular form adopted is that of a stochastic context-free grammar in Chomsky Normal form. In short such a grammar consists of a set of terminal symbols (the words of the language) a set of non-terminal symbols and a set of production rules. The production rules are constrained to be of one of two forms:

$$\begin{aligned} a &\rightarrow bc \\ a &\rightarrow \alpha, \end{aligned}$$

where a, b and c are non-terminal symbols and α is a terminal symbol. Thus a non-terminal symbol may either re-write as a string of two non-terminal symbols or as a single terminal symbol. It has been shown that any context-free grammar can be written in Chomsky Normal form (by increasing the number of non-terminal symbols if necessary), so the Chomsky normality constraint does not impose an structural constraints on the languages that can be modeled by context-free grammars, but only on the structure of the parse trees. In fact the parse trees must be of the form depicted in Figure 7, where each node has two daughter nodes except the so-called pre-terminal nodes, the ones leading to terminal symbols, which have only one. It also follows that once the number of non-terminal symbols are determined the grammar can effectively be represented by three quantities: a tensor of rank 3, a matrix and a vector: We write

A_{ijk} to mean the probability of non-terminal symbol i re-writing as the two non-terminals j and k

B_{im} to indicate the probability of the non-terminal i re-writing as the terminal symbol m and

P_i to indicate the prior probability of the first symbol in a derivation (the root).

These three quantities must satisfy the stochastic constraints:

$$\sum_{jk} A_{ijk} + \sum_{m} B_{im} = 1 \qquad \text{for all } i \tag{96}$$

and $$\sum_i P_i = 1 \tag{97}$$

These three quantities specify our knowledge base, or in other words parameterize the language model.

For a given parse tree, such as the one shown in Figure 7 we can now apply the belief propagation algorithm. The tensors A and B serve as connection tensors and P provides the prior to the root node of the tree. In fact a tree such as the one shown in Figure 7 will contain many instances of the A and B tensors, one for each terminal and non-terminal production respectively, all of which are linked.

The quantities A and B can be used as the weight matrices for the belief propagation algorithm. For example, suppose we observe the two words "the book" in succession in the observation sequence and a tree of the form shown in Figure 8 is constructed. Now, remember that each non-terminal node represents a random variable that can take a number of discrete values. In the case of Mr. Holmes inference about burglary each node could have two values, like "alarm sounds" versus "alarm does not sound". In our case, each non-terminal node can range over all non-terminal symbols of the grammar. Similarly each terminal node can range over all terminal symbols. Thus if the grammar consists of N_{nt} non-terminal symbols and N_t terminal symbols then each non-terminal node represents a random variable which can range over N_{nt} values and each terminal node represents a random variable which can range over N_t values. Now in the present example the identity of the terminal symbols is known: "the" and "book". This knowledge is expressed by assigning these nodes a λ vector of the form $(0,\ldots,0,1,0,\ldots,0)$. Recall that each component of a λ vector represents the likelihood of a symbol given the evidence. So the 1 appears at the position corresponding to the word in question.

It should be noted that in some applications the identity of the word may not be known with certainty. For example in connection with a speech recognizer only a set of scores for different word candidates may be available. Indeed, a speech recognizer based on Hidden Markov Models for instance outputs the likelihoods $\Pr(O|M)$, the probability of the acoustic evidence O having resulted from the Markov model M. This quantity corresponds precisely to the definition of the components of the λ vectors for these are defined as $\Pr(e_X^-|X)$, X being the terminal symbol (or Markov Model) and e_X^- the underlying evidence. Hence, in this case we can use the likelihoods output by the recognizer as the components of the λ vector. It is the naturalness at which uncertainty can be incorporated into the Bayesian belief networks which make them so attractive as a language model for speech recognition.

The instantiated λ vectors at the leaf nodes together with the P vector of equation 97 at the root node allow us now the calculation of the belief of each node and also the summands of equation 69. This in turn allows us to update the model parameters after a whole set of training samples has been processed.

This clearly works, no matter how large the parse trees involved are. The key question that remains to be answered is how the trees may be constructed in the first place. This will question will be addressed in the remainder of this section.

6.3 The π vectors at the leaf nodes – a symbol 'predictor'

The π vector of a leaf node is also a very useful quantity. It describes the probabilities of word candidates given the entire context of the word (within the current tree). Therefore the quantity $-\log(\lambda \cdot \pi)$ at a leaf node measures the amount by which the current symbol was expected in the given context.

Recall that the belief of a leaf node could be calculated as

$$\mathrm{BEL}(X) = \frac{\lambda(X)\pi(X)}{\lambda(X) \cdot \pi(X)}, \tag{98}$$

where the numerator is the component-wise vector product and the denominator is the familiar dot-product. Hence if the λ vector of a leaf node is deterministic (i.e. of the form $(0, \ldots, 0, 1, 0, \ldots, 0)$) the belief vector will not be different from this no matter what the π vector is at this node, but if the λ vector represents likelihoods from an underlying acoustic observation, as is the case in a speech recognition task, the belief vector represents the posterior probability distribution of the symbol in question, given the entire acoustic evidence.

6.4 Finding the best tree coverage of an observation sequence

The previous section showed that propagating λ and π vectors through a tree structure is a computationally efficient method for calculating beliefs of terminal and non-terminal symbols especially when uncertainty is involved (for instance when several word hypotheses are produced by a speech recognizer). We will now address the underlying problem of finding a possible parse tree to use for the propagation algorithm.

(6.4.1) The language units

Traditionally, linguistic theory uses the sentence as the basic unit and defines a language as a set of (grammatically correct) sentences. In this framework it would be most natural to assign a single tree structure to each sentence (the parse tree of this sentence). This implies that we need to be told the end points of the sentences in order to process them. In written language these end points are usually available, through punctuation, but in the spoken language accurate segmentation points may be lacking.

For this and other reasons, the tree construction algorithms presented below will not make use of the sentence unit. Instead trees will be constructed over adjacent segments of the observation sequence, where the boundary points are also optimized. The reasons for following this approach are described as:

1. The sentence boundaries may not be known (see above)

2. The sentence unit may not be the optimal unit. By this is meant that if the training data is limited it may be better to learn a "sub-grammar" consisting of smaller-than-sentence units as opposed to attempting to learn the full grammar.

Hence we allow the parse trees to span any adjacent sequence of symbols from the observation sequence that seems to form a suitable unit. Depending on the modeling ability of the grammar, these units may be larger or smaller than the actual sentence. Thus the

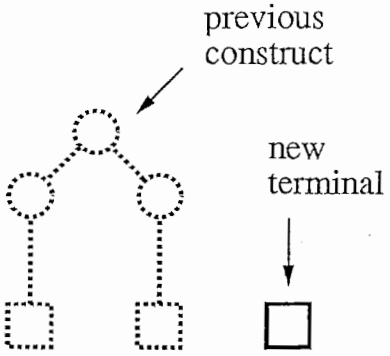

図 9:

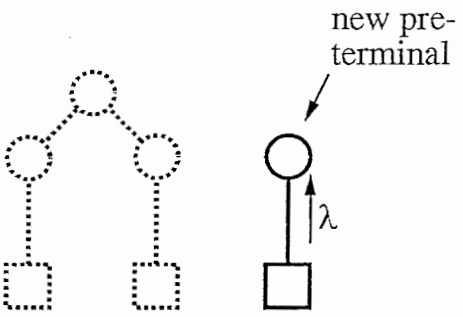

図 10:

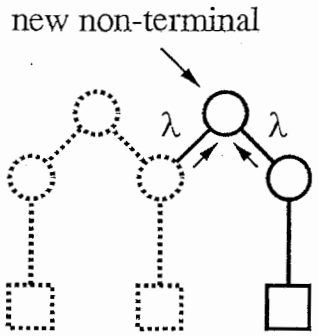

図 11:

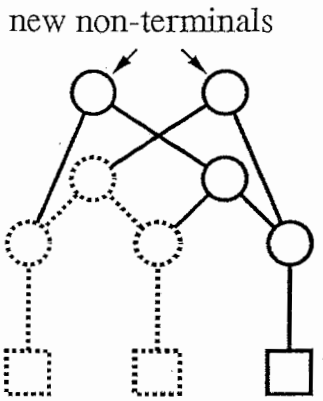

図 12:

tree boundaries are not a priori aligned to the sentence boundaries. After learning of the language they may turn out to be aligned but only if such an alignment is advantageous for the reduction of total entropy. We will now consider several tree construction algorithms in turn.

6.5 Tree construction: version 1

Since the grammar model is constraint to be in Chomsky Normal Form the topology of the trees are fairly constraint. In particular each terminal symbol has a unique 'pre-terminal' symbol as its parent. These nodes may be created immediately, as soon as the terminal symbol has been observed (Figures 9 and 10). Next the λ vector, which has been instantiated for the terminal node can be propagated up to the pre-terminal node. The general propagation equations were given in section 4 (equations 20 to 28), but in this simple case these collapse to a simple matrix-vector multiplication:

$$\lambda(U) = B\lambda(X), \tag{99}$$

where U is the pre-terminal node and X is the terminal node. The new vector $\lambda(U)$ is independent of the shape of the eventual parse tree, so it can be calculated at this stage once and for all (Figure 10).

The newly created pre-terminal node, can be regarded as a tree of size one, and we can calculate the entropy of this tree using equation 63. Next, for two neighbouring pre-terminal nodes, a non-terminal node is proposed (Figure 11), having the two pre-terminals as daughters. Again, the λ vector may be passed up this new node. The propagation equation

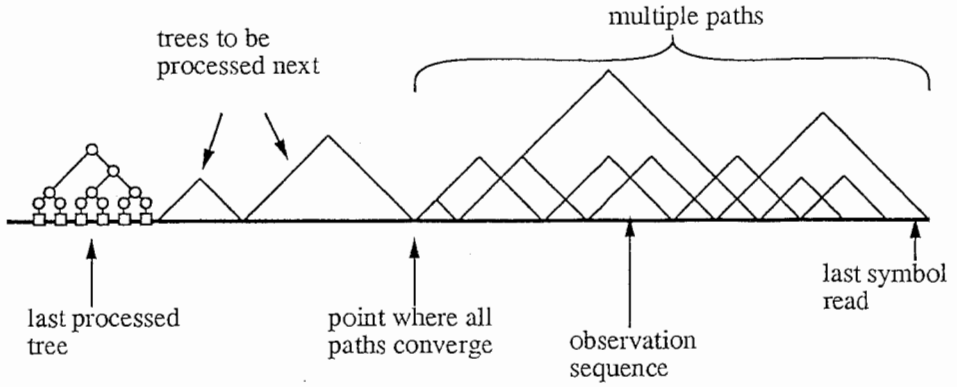

図 13: This diagram illustrates the parsing algorithm. When a new symbol is read all (newly) possible parse trees are constructed bottom up. This defines a lattice of trees. The possible paths through the lattice are traced and pruned such that only the best (i.e. minimal entropy) path for each end point is kept. At some earlier point (relative to the symbol just read) all these paths converge, identifying a unique sequence of trees. These trees are then chosen for the propagation and re-estimation algorithm.

follows again from equations 20 and 28 and in this case read:

$$\lambda(V)_i = \sum_{jk} A_{ijk}\lambda(U_1)_j\lambda(U_2)_k \tag{100}$$

where V is parent node and U_1 and U_2 are the two daughters. It is not yet clear whether this non-terminal will be part of the final tree, but if it is then again the its λ vector will not change as it depends only on its daughters. Again we can calculate the entropy of this tree (of size 2) using equation 63. If the entropy calculated is larger than the sum of the entropies calculated previously for the two daughter nodes, the new node is rejected and removed from memory. Otherwise it is kept as a potential node for the final tree. The process continues now in the same fashion by constructing new non-terminal nodes between any pair of existing non-terminal nodes which span adjacent sequences of the observation sequence (Figure 12).

Continuing this procedure leads to a lattice of nodes. Each non-terminal node defines a potential tree, but these trees are partially overlapping. To obtain a single coverage of the observation sequence, the lattice of trees needs to be pruned. This can be achieved by using a dynamic programming technique. One simple selects the path through the lattice which minimizes the overall entropy.

After the trees have been isolated in this way, the π vectors are passed down from the root-node. At this time the beliefs of each node and the partial contributions to the weight re-estimation terms (equation 69) can be calculated.

At the end of a training epoch the quantities A, B and P may be updated using equation 68.

6.6 Convergence of the grammar learning algorithm

As was pointed out previously, the grammar learning algorithm consists of two parts: finding the structure of the parse trees and calculating the belief of the nodes. We will now

show that under suitable conditions the algorithm will converge.

We will consider the case of finite training data. Parameters are updated each time the entire training material has been processed. We will show that after each update the total entropy calculated over the training data will be reduced. Since the entropy is always positive it follows that the algorithm converges.

In general it cannot be assumed that the segmentation of the training data into parse trees and the topology of these trees is identical from iteration to iteration. Rather we expect that with the changing model parameters the topology of the parse trees will also change. However if the topology of the pare trees in one iteration is exactly identical to the topology of the previous iteration then Theorem 1 ensures that the overall entropy has been reduced.

In the general case, the entropy of the training data given the parameter set will still monotonically decrease from iteration to iteration, if the tree structures are selected on the principle of maximizing this probability with the given parameter set. To make this clearer, let $\mathcal{T}$ denote a tree structure constructed over the training data and $\Pr(e|\mathcal{T},\Theta)$ the probability of the training data given the tree structure and the parameter set Θ. Theorem 1 proves that

$$\Pr(e|\mathcal{T},\bar{\Theta}) \geq \Pr(e|\mathcal{T},\Theta), \tag{101}$$

if $\bar{\Theta}$ is derived as was explained in the previous section. Now, if we use a new tree structure $\bar{\mathcal{T}}$, a sufficient condition for

$$\Pr(e|\bar{\mathcal{T}},\bar{\Theta}) \geq \Pr(e|\mathcal{T},\Theta) \tag{102}$$

is

$$\Pr(e|\bar{\mathcal{T}},\bar{\Theta}) \geq \Pr(e|\mathcal{T},\bar{\Theta}). \tag{103}$$

This condition is trivially satisfied if we choose the $\bar{\mathcal{T}}$ which maximizes $\Pr(e|\bar{\mathcal{T}},\bar{\Theta})$ and hence the grammar learning algorithm will necessarily converge in this case.

A $\bar{\mathcal{T}}$ which maximizes $\Pr(e|\bar{\mathcal{T}},\bar{\Theta})$ can be found by searching through all possible tree structures. This is computationally less expensive as it first seems, because the independence of neighbouring trees can be used to discard non-optimal structures at a fairly early stage. The algorithm that was proposed in section 6.4 uses heuristic principles to restrict the search space. Unfortunately it is not guaranteed to produce the optimal tree coverage. In practice this does not pose a problem as the following argument will make clear:

If the algorithm finds the tree coverage $\mathcal{T}$ in one iteration, the change from parameter set Θ to $\bar{\Theta}$ will only make it more likely that this structure will be found again and will be part of the tree lattice constructed. If the finally selected tree structure $\bar{\mathcal{T}}$ does not equal $\mathcal{T}$, it must be because this structure lead to a higher probability of the training data. Even if $\bar{\mathcal{T}}$ is not optimal, the fact that it is better than $\mathcal{T}$ under the new parameter set is sufficient to guarantee equation 102 and hence convergence.

6.7 Problems with version 1 of the tree construction algorithm

Section 6.5 introduced a simple algorithm for finding a tree structure suitable for the observation sequence. It uses heuristic pruning techniques, based on minimal entropy selection, to keep the total number of trees to be considered low. However the algorithm, as it stands, has a serious deficiency that calls for its modification: It is not capable of learning the succession of two non-terminal symbols unless they occur in the same tree.

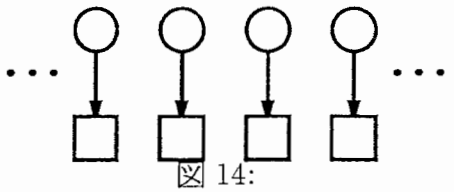

図 14:

To explain this point further, let us consider what happens during the early stages of training. Initially all rule probabilities are randomized and during the first sweep through the training data each tree has size 1 as shown in Figure 14.

Now, the relationship between any two neighbouring non-terminal symbols must be learned. However the present algorithm, as it stands, does not do this, for the parameter re-estimation algorithm is only applied within each tree. This means that only the parameters of the B matrix are learned, so the resulting grammar is in fact equivalent to a uni-gram grammar.

By modifying the tree building algorithm we can however avoid this problem.

6.8 Tree construction: version 2

In version 2 of the tree construction algorithm trees are constructed and isolated in the same way as in version 1. However, prior to calculating the partial contributions to the weight re-estimation term (equation 56) two neighbouring trees are combined into a big tree by hypothesizing a common root node. Figure (6.8.1) illustrates this, during the first iteration, when all constructed trees have size 1. After all weight contributions have been calculated, the hypothetical root node is removed and the next two trees are combined to a new tree (Figure (6.8.1)). (The tree who formed the right branch of the previous construct now forms the left branch.) Training is repeated on this new construct. In this way, the information about neighbouring trees is learned by virtue of the hypothetical nodes, and hence 'tree-growth' is possible during training.

(6.8.1) Allowing multiple priors

Version 2 of the tree building algorithm depends upon the initial symbol distribution vector P. An additional reduction in entropy can be achieved, by allowing this vector to vary according to size of the tree to which it is applied.

In this scenario, rather than storing just one vector P, we use an array of vectors `prior`$[s]$. For a tree of size s (i.e. which spans s symbols of the observation sequence) the vector `prior`$[s]$ is used as the prior for that tree.

6.9 Tree construction: version 3

Version 3 of the tree building algorithm is very similar to version 2. To understand it let us first consider an extension to the stochastic context free grammar formalism.

Up to now the grammar was described by three quantities: B_{im} describing the terminal productions, A_{ijk} describing the non-terminal productions and P_i describing the prior

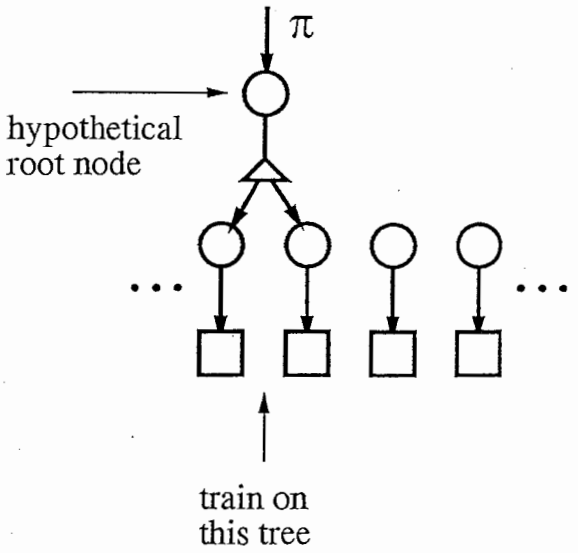

図 15:

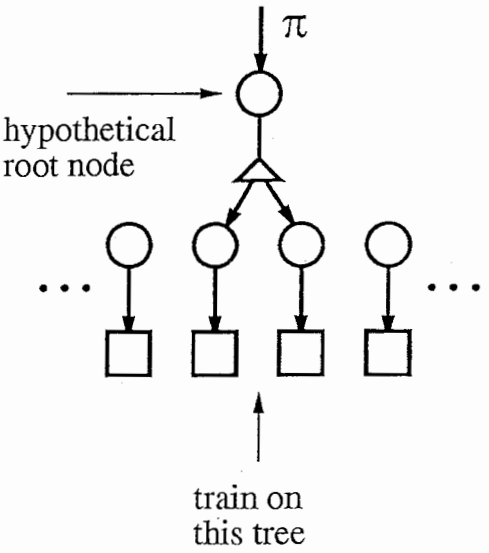

図 16:

probability of the root symbol. We now replace the prior P_i by a matrix of conditional probabilities D_{ij}, describing the value of the current root node given the value of the previous root node. To describe this more precisely, if we have a sequence of trees $T_1, \ldots$ spanning the observation sequence, with root nodes $R_1, \ldots$ then

$$D_{ij} = \Pr(R_t{=}j | R_{t-1}{=}i). \tag{104}$$

One way of looking at this is to say that we provide a bigram grammar for the root nodes of the trees. We now no longer have isolated trees, but rather one big structure. The λ and π vectors can be propagated through this structures just as the theory dictates (section 4.1). Each root node has two groups of daughter nodes. One group consists of the two immediate daughters within the same tree. Its relation to the parent is described by the tensor A. The other group consists of a single node, the root node of the next tree, and the relation is described by the matrix D.

Figure 17 illustrates how the λ and π vectors are propagated through the network.

If this architecture is used for training, the D matrix learns the 'root node bigrams'. However this knowledge has to be transferred into the A tensor, if the trees are to grow in subsequent iterations. Otherwise we are in a situation similar to the one of version 1 of the tree construction algorithm.

In principal this problem can be solved by linking the parameters of the D matrix with those of the A tensor. But since they have different dimensions this cannot be achieved in quite such a straight forward way. However the entries of the D matrix can be calculated from the entries of the A tensor by re-introducing the initial symbol vector P.

In version 2 of the tree building algorithm we hypothesized a new root node H for any pair of adjacent trees with root nodes R_1 and R_2. Thus the tensor A learns the conditional probabilities $\Pr(R_1, R_2|H)$. Since the prior of H is given by the initial symbol distribution

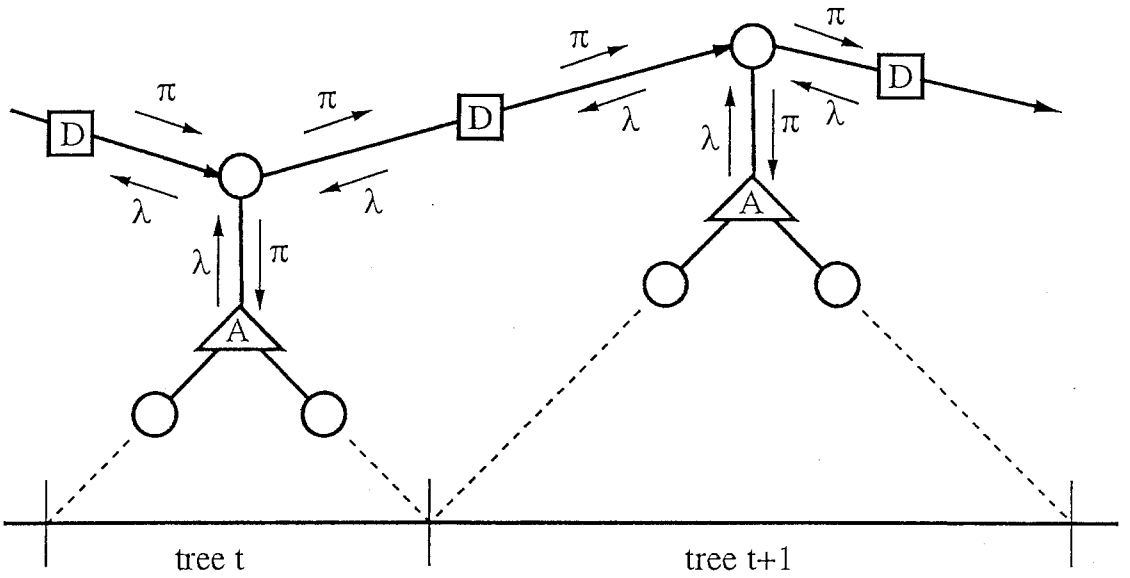

図 17:

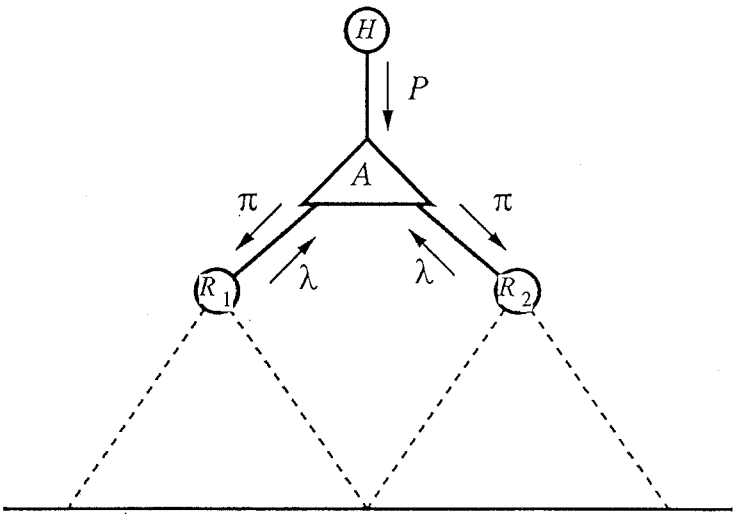

図 18:

vector P, we can get the joint distribution of $\Pr(R_1, R_2)$ by multiplying A by P:

$$\Pr(R_1{=}j, R_2{=}k) = \sum_i A_{ijk} P_i \tag{105}$$

and from this we can calculate the conditional probabilities by simple normalization:

$$\Pr(R_1{=}j|R_2{=}k) = \frac{\Pr(R_1{=}j|R_2{=}k)}{\sum_{j'} \Pr(R_1{=}j'|R_2{=}k)} \tag{106}$$

Hence we can calculate D from A by setting

$$D_{jk} = \frac{\sum_i A_{ijk} P_i}{\sum_{i,j'} A_{ij'k} P_i} \tag{107}$$

The tree-building and learning algorithm can now be described as follows: At the beginning of the iteration the D matrix is calculated from the A tensor and the P vector according to equation 107 above. Trees are then constructed using version 1 of the tree building algorithm. The root nodes are then linked using the D matrices as shown in Figure 17 and the λ and π vectors are propagated through the entire network. When it comes to calculating the contributions to the re-estimation terms (equation 48) the D matrices are again replaced by the A tensors and the λ vectors from the two root nodes plus the P vector are used to calculate the contribution to the A tensor at this point of the structure.

6.10 Discussion on tree building algorithms

In the previous subsections we have presented three closely related algorithms for parsing the observation sequence. Of these version 1 does not work during the training period, because the tree are not encouraged to grow. (It is however a useful algorithm once the grammar parameters have been learned). The main reason for including it here was its simplicity and the fact that it forms the basis for the other two algorithms.

Version 2 is a simple extension of version 1, that does allow trees to grow from iteration to iteration. The drawback of this algorithm is that training is no longer performed on the structure exhibiting minimal entropy, but rather on a modification of this structure. The fact that the minimal entropy structure has been modified implies that the convergence argument given in section 6.6 does no longer strictly apply. So the overall entropy calculated over the observation data may no longer be strictly decreasing from iteration to iteration. However experiments have confirmed that this is not a problem in practice.

Version 3 is a neat extension to version 2 that on top of providing a justification for the hypothetical root nodes also extends the grammar model by a bigram grammar between root nodes, so that the trees are no longer independent of each other.

はい	もしもし	。	えーっと	そちら	第	1	回	の	国際会議	の	事務	局	でしょ	う	か	。
感動詞	感動詞	記号	間投詞	代名詞	接頭語	数詞	接尾語	格助詞	固有名詞	格助詞	普通名詞	普通名詞	助動詞	助動詞	終助詞	記号
													未然	終止		
はい	そう	です	。	えーっと	ちょっと	その	会議	の	こと	で	ね	あのー	登録	の	こと	...
感動詞	副詞	助動詞	記号	間投詞	副詞	間投詞	普通名詞	格助詞	普通名詞	格助詞	終助詞	間投詞	普通名詞	格助詞	普通名詞	

図 19: Example sentences from the ATR dialogue database with part of speech labels.

7 Experimental work

Some preliminary experiments have been carried out will be described in the remainder of this paper.

7.1 Database

The experiments were based on the ATR Dialogue database. This is a text database consisting of some 8500 sentences of telephone conversations together with their part of speech labels [1]. A few example sentences are shown in Fig. 19. The database contains a total of about 6500 different words with 51 different parts of speech. The part of speech labels were judged as a suitable input unit for the BLI model since their number is limited ensuring enough training samples for each part of speech. Ideally words should have been used as input unit.

For the purpose of the experiment the data base was divided into two equal portions, one for training and one for testing.

7.2 Segmentation of the training data

In HMM training, it is customary to update the HMM probabilities each time the entire training data has been processed. In the experiments reported herein we update more frequently. Thus successive training epochs are carried out on different part of the training data. This is motivated by the following observation: The grammar inference mechanism described here only requires an unlabeled source of symbols to operate. Even though training is performed on a finite amount of training data for the experiments described herein, it is feasible that the algorithm is used on an infinite symbol source such as a news broadcasting station. At present when the training data is used up the algorithm starts again at the beginning of the training data. In the future it could indefinitely make incremental improvements to its current weights by continuously processing an infinite source.

Initially when the model parameters are still random, only trees of size 1 are constructed (see experimental section and Fig. 22 below). During this phase essentially the uni-gram statistics of the data base are learned. It follows that it is not necessary to have a very long update period during this time of learning. As the sizes of the trees increases, the model parameters can only be accurately estimated by taking more and more data into account during one training epoch. Thus it seems advantageous to start with a relatively small update period and increase this gradually. This was done in the experiments, starting with a update period of 100 symbols, which was increased by 30 after each update.

7.3 Preliminary experiments

In the first experiment the three parsing algorithms were compared. For this purpose we trained models with 30 non-terminal symbols. The results are shown in the following table:

method	tree size	train	test
version 1	1.0	4.48	4.46
version 2	2.81	2.81	2.80
version 3	2.94	3.88	3.87

The table shows the average size of the trees constructed by the algorithm and the entropy over the test and training data. As was pointed out earlier version 1 of the parsing algorithm does not lead to tree growth and was included as a control.

Somewhat surprisingly version 2 performed significantly better than version 3, despite the fact that version 3 provides a bigram type grammar for the root nodes of the trees.

7.4 Evaluation of the BLI model

In order to evaluate the performance of the BLI model a model with 50 non-terminal symbols were trained. The number 50 was chosen for two reasons: Firstly it roughly agrees with the number of non-terminal symbols (51) and secondly it was the largest number that could be trained in reasonable time on a work station. Following the results of the previous section we used version 2 of the tree building algorithm. Other model parameters were chosen as follows:

number of terminals (N_t)	51
number of non-terminals (N_{nt})	50
initial update period	100
update period increase	30
maximum tree size	6

Fig. 21 shows the development of the entropy when estimated during training and Fig. 22 shows the average tree size during each training epoch. Initially the entropy was close to 6, but dropped sharply to the "unigram-level" of about 4.5 after the first few iterations. During this period the tree size was 1. When trees started to be constructed the entropy dropped further. Both curves are quite noisy. This is due to the fact that different training epochs were carried out on different parts of the training material.

Training was discontinued after 300,000 symbols had been processed. This took about two weeks on a DEC station 5000. The trained model was subsequently evaluated on the test data and an entropy of about 2.7 bits was observed. This was compared to the entropies obtained from a bigram or trigram grammar. The overall results are summarized in the following table

model entropy	bigram	trigram	BLI
training data	2.83	2.39	2.60
test data	2.84	2.76	2.70

As can be seen, the BLI model outperforms the bigram model, but its entropy is similar to the trigram entropy. On the training data, the BLI entropy is considerable higher than the

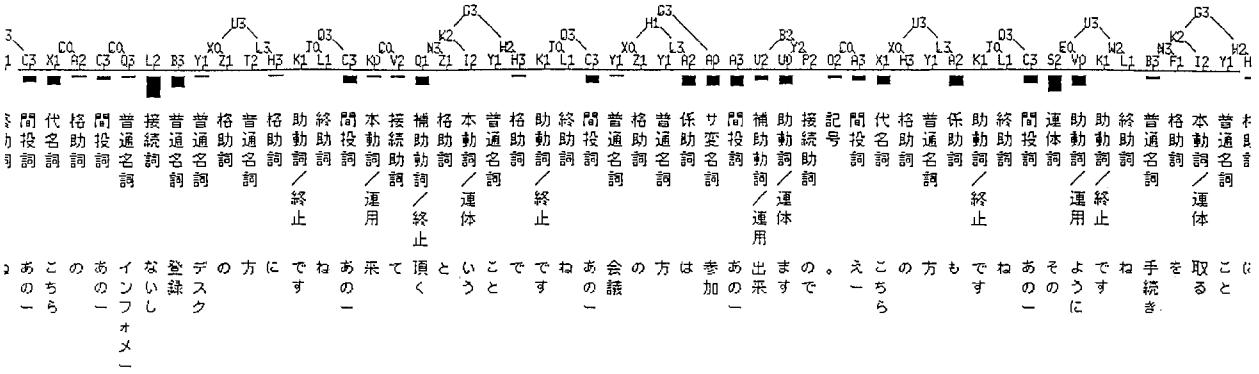

図 20: Example screen dump obtained during training

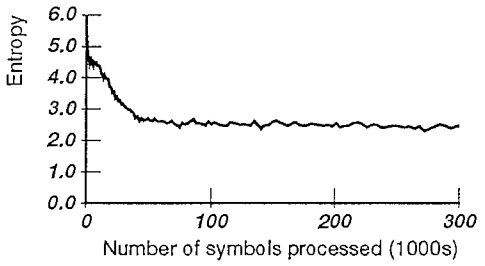

図 21: Development of the entropy during training

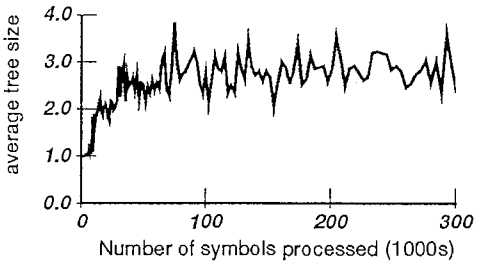

図 22: The average tree size during training

trigram entropy and much closer the test data entropy. This means that the generalization is much higher for the BLI model. In other words the BLI model seems to rely less on the coverage between the training and the test data. This means that the BLI model will probably scale better to larger tasks once the problems concerning training time have been overcome.

Fig. 20 shows how the BLI algorithm processed part of the training material. It shows the part of speech sequence that was used as input to the model together with the parse trees constructed. The letters labeling the nodes of the trees refer to the non-terminal symbols with the highest belief component at that node. The bar chart above the part of speech represents the entropy of the terminal node ($E(s_t)$). The underlying word sequence was included as a reference.

It is evident that the parse trees roughly correspond to Japanese grammatical units such as bunsetsu. Since the sentence end marker ($_{\circ}$) is processed just like all other symbols, it is also incorporated in the tree structures. For example, since a sentence often starts with a filler (間投詞) a tree connecting period (記号) with (間投詞) is constructed.

8 Discussion

In this report we have shown that parse trees may be regarded as Bayesian network. This framework enables one to integrate rule probabilities of stochastic grammars and observational uncertainties (such as acoustic match likelihoods) in a mathematically sound and computationally efficient way. Moreover we have shown that the causal and evidential support vectors that arise as intermediate quantities in the Bayesian formalism may be used, in a natural way, to calculate new grammar parameter estimates. The parameter re-estimation is guaranteed to increase the likelihood of the observed data, so long as the parse trees are selected on a maximum likelihood principle.

While the propagation of support vectors has a well-developed theory with many nice mathematical properties, the construction of suitable belief networks connecting the observed events is a problem for which to date no fully satisfactory solution exists. For this reason, the usage of Bayesian networks in AI has been limited to applications where the network structure was fixed could be decided in advance. While this might be adequate in some applications there is clearly a need in larger applications with many different and partially unrelated nodes to construct Bayesian networks 'on the fly' by connecting only the relevant nodes.

In the context of natural language processing that we are concerned here the problem of constructing suitable networks describing the observation sequence is well known under the name of parsing. A great many parsing algorithms exists, but unfortunately most are not of stochastic nature.

We described three different simple methods for parsing the observation sequence based on the CYK parsing algorithm. We found that during training an algorithm that simply selects the tree structure with the highest likelihood may not work, because neighbouring trees may never be combined, so trees do not 'grow' from iteration to iteration. Therefore there is the need for compromising the maximum likelihood selection principle in order to learn the 'neighbouring-tree' statistics during training. Two of the algorithm proposed have this property and we compared them experimentally.

Finally we compared the model experimentally with the common bigram and trigram model. We showed that on the given task the BLI model did not only perform superior to both n-gram models, but also showed better generalization ability.

参考文献

[1] T. Ehara, N. Inoue, H. Kohyama, T. Hasegawa, F. Shohyama, and T. Morimoto. Contents of the ATR Dialogue Database. Technical Report TR-I-0186, ATR Interpreting Telephony Research Laboratories, 1990.

[2] Frederick Jelinek. Up from trigrams! In *Proc. Eurospeech*, pages 1037–1040, Genova, Italy, September 1991.

[3] S. L. Lauritzen and D. J. Spiegelhalter. Local computationswith probabilities on grphical structures and their application to expert systems. *Statistical Society*, pages 157–224, 1988.

[4] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems – Networks of plausible inference.* Morgan and Kaufmann, 2929 Campus Drive, Suite 260, San Mateo, CA 94403, 1987.

[5] D. J. Spiegelhaleter and S. L. Lauritzen. Sequential updating of conditional propabilities on directed graphical structures. *Networks*, 20:579–605, 1990.