Proceedings of SOUTHCON'94 (Orlando, March 1994), 60–64.

Learning in neural networks with Bayesian prototypes

Petri Myllymäki and Henry Tirri
University of Helsinki, Department of Computer Science
P.O.Box 26, FIN-00014 University of Helsinki, Finland

***Abstract*— Given a set of samples of a probability distribution on a set of discrete random variables, we study the problem of constructing a good approximative neural network model of the underlying probability distribution. Our approach is based on an unsupervised learning scheme where the samples are first divided into separate clusters, and each cluster is then coded as a single vector. These Bayesian prototype vectors consist of conditional probabilities representing the attribute-value distribution inside the corresponding cluster. Using these prototype vectors, it is possible to model the underlying joint probability distribution as a simple Bayesian network (a tree), which can be realized as a feedforward neural network capable of probabilistic reasoning. In this framework, learning means choosing the size of the prototype set, partitioning the samples into the corresponding clusters, and constructing the cluster prototypes. We describe how the prototypes can be determined, given a partition of the samples, and present a method for evaluating the likelihood of the corresponding Bayesian tree. We also present a greedy heuristic for searching through the space of different partition schemes with different numbers of clusters, aiming at an optimal approximation of the probability distribution.**

I. Introduction

This work aims at combining the ideas of memory-based reasoning (MBR) with neural network techniques, especially for developing massively parallel implementations for "shallow knowledge" domains such as the ones discussed in [4]. Given a problem, a memory-based reasoning system will search its prototype memory and match the input vector against the stored prototype vectors, providing them with a similarity rank according to some similarity measure (prototype matching). The prototypes are then modified according to the rankings, in order to provide a solution consistent with the input vector (adaptation). As there is nothing that resembles a shared memory, the memory-based reasoning architecture is inherently parallel, and each element can perform comparison of its input against the value stored in the interconnections independently from the others (see the general MBR structure in Figure 1). This offers a linear speed-up in the comparison process with respect to the number of computational elements available.

Neural network architectures are constructed from a large amount of highly connected elements, which makes them suitable for distortion tolerant storing of a large number of prototypes (represented by high dimensional vectors) by making single elements "sensitive" to a stored item. As a matter of fact, many existing neural network architectures [9, 17, 18, 8, 3] can be seen as realizations of the general memory-based reasoning scheme. These massively parallel models, and other kernel-based estimators (e.g., [13, 1]) assume that the domain probability distribution can be approximated as a finite mixture of particular types of kernel distributions (e.g., Gaussian). In our approach, we restrict ourselves to discrete problem domains, which allows us to drop all assumptions about the form of the underlying probability distribution. Modeling the probability distribution by a set of *Bayesian prototype* vectors, allows us to use the theory of *Bayesian networks* [14, 12] to construct a tree-structured Bayesian network representation of the probability distribution. The approach resembles the *simple Bayesian model* of [5] and [6], but unlike these models, our network can be used also for general regression tasks (adaptation), and not only for classification tasks (prototype matching). What is more, these tasks can be performed efficiently on a massively parallel neural network architecture, as there is a direct mapping from our Bayesian tree structure to a multi-layer feedforward neural network structure.

The accuracy of our model depends on how well the chosen Bayesian prototypes reflect the actual probability distribution. In [10, 11], the prototypes

This research was supported by the Technology Development Center (TEKES).

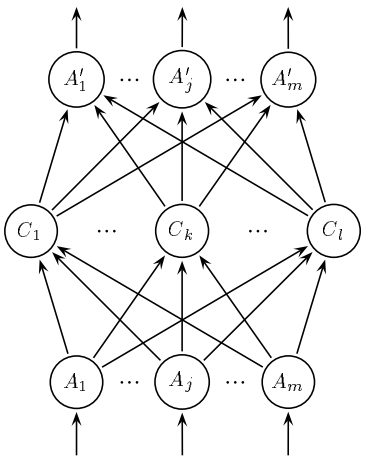

Figure 1: The general MBR scheme.

were assumed to be provided by a domain expert. However, although using experts is viable in some domains, in many cases such prototypes need to be inferred from raw data. In this paper we describe a probabilistic learning framework for selecting the proper number of Bayesian prototypes, and for determining the prototype vectors. We proceed first in Section *II* by showing how to construct a set of prototype vectors and the corresponding Bayesian prototype tree, given a data set and a clustering of the data vectors, and how the prototype tree is implemented as a feedforward neural network. A probabilistic criteria for evaluating a given Bayesian prototype tree is discussed in Section *III*. This criteria can be used to compare different Bayesian prototype trees with respect to the data set available. This selection criteria is used in the heuristic algorithm (presented in Section *IV*) for searching through the space of possible prototype trees. The algorithm tries to find an optimal Bayesian prototype tree structure for the problem domain distribution, which can then be used for constructing an optimal neural network structure.

II. Constructing Bayesian prototype sets through data clustering

Let us assume that the problem domain knowledge is represented using values of m discrete attributes (random variables) $A_1, \ldots, A_m$, where each attribute A_j has n_j possible values, $a_{j1}, \ldots, a_{jn_j}$. Our *training data set* $\mathcal{D}$ consists of N complete data vectors $\vec{d}_1, \ldots, \vec{d}_N$,

$$\vec{d}_i = (V_i(A_1), \ldots, V_i(A_m)),$$

where $V_i(A_j)$ is the value of attribute A_j,

$$V_i(A_j) \in \{a_{j1}, \ldots, a_{jn_j}\}.$$

In the general case, the data vectors are not uniformly distributed over the vector space, but form clusters of one or more similar vectors. In the presence of noise the best strategy is not necessary to store all the data vectors as separate prototypes, but to try to find representatives for a set of similar instances. Storing all the data vectors as separate prototypes would cause overfitting, i.e., fitting to the noise component also. As almost all real life data is noisy due to measurement errors or uncertainty, need for clustering is more a rule than an exception.

Let us cluster the N data vectors of the training set into l classes $c_1, \ldots, c_l$, and provide each data vector with the corresponding class label. We model each class c_k by a *Bayesian prototype vector*

$$\begin{aligned}\vec{c}_k &= (P_k(a_{11}), \ldots, P_k(a_{1n_1}), \ldots, \\ &\quad P_k(a_{m1}), \ldots, P_k(a_{mn_m})),\end{aligned}$$

where $P_k(a_{ij})$ expresses the likelihood for attribute A_i to have value a_{ij} in class c_k,

$$P_k(a_{ij}) = P(A_i = a_{ij} | C = c_k).$$

These conditional probabilities can be either estimated by simply using the relative frequencies, or as noted in [2, 5], approximated more accurately with the formula

$$P_k(a_{ij}) = \frac{F_k(a_{ij}) + 1}{|c_k| + n_i},$$

where $F_k(a_{ij})$ is the number of data vectors in cluster c_k with the property $A_i = a_{ij}$, and $|c_k|$ is the number of data vectors belonging to cluster c_k.

The class labels $c_1, \ldots, c_l$ can be regarded as mutually exclusive values of a hidden random variable, C. If the cluster structure chosen corresponds well to the true probability distribution of the problem domain, knowing membership in a cluster can be used for determining the attribute values. In the Bayesian framework this means that *all the variables A_i are conditionally independent of each other, given the value of the variable C* (observe that this does not mean that the attributes are independent). In other words, the Bayesian network corresponding to this representation is a tree, where a single variable C is the root of the tree, and variables A_i form the leaves (see Figure 2).

In addition to the conditional probabilities $P_k(a_{ij})$, we need for our Bayesian network model l prior probabilities, one for each of the classes c_k. These can be approximated by

$$P(c_k) = \frac{|c_k| + 1}{N + l}.$$

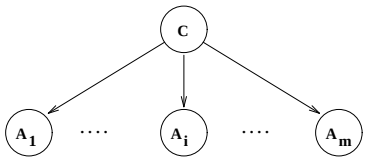

Figure 2: Bayesian prototype tree representation of the problem domain probability distribution.

The Bayesian prototype tree, together with the corresponding prototype vectors and prior probabilities, can be used for constructing a multi-layer feed-forward neural network which models the problem domain probability distribution (see Figure 3). The first three layers of the network perform prototype matching, corresponding to the simple Bayesian classifier model in [5, 6], and the last three layers perform Bayesian adaptation. For more details of the functioning of the model, see [10, 11].

III. Evaluating prototype trees

Let $\mathcal{M}$ denote a Bayesian prototype tree constructed as described in the previous section using a data set $\mathcal{D}$. Assuming that the training set $\mathcal{D}$ represents the problem domain distribution well, an optimal prototype tree is obtained by maximizing the probability

$$P(\mathcal{M} \mid \mathcal{D}) = \frac{P(\mathcal{M})P(\mathcal{D} \mid \mathcal{M})}{P(\mathcal{D})}.$$

As the probability $P(\mathcal{D})$ is constant for a given training set, we can compare the likelihood of different prototype sets by using as the criteria the formula

$$P(\mathcal{M} \mid \mathcal{D}) = cP(\mathcal{M})P(\mathcal{D} \mid \mathcal{M}), \qquad (1)$$

where c can be any positive constant. Since in our case we have a tight correspondence between the Bayesian prototype tree and the probabilistic neural network, we can use criteria (1), developed for the simpler Bayesian structure, for evaluating also probabilistic neural networks. Hence once we have illustrated how an optimal Bayesian tree structure can be learned, the resulting tree can then be trivially transformed into the corresponding optimal neural network.

In [7], criteria (1) was approximated using the Minimum Description Length [15] approach, where the goal is to minimize the sum

$$-\log P(\mathcal{M}) - \log P(\mathcal{D} \mid \mathcal{M}).$$

However, as their model is used for learning general multiply connected Bayesian networks, the encoding scheme is biased towards sparse networks with few values, and in our case it will give too high a penalty for introducing new values to the root node, i.e., for introducing new cases. In an alternative approach used by Cooper and Herskovitz [2], the probabilities needed for criteria (1) are approximated directly. However, unlike Lam and Bacchus in [7], they do not take into the account the prior probabilities $P(\mathcal{M})$, but assume all structures to be equally probable. We combine these two approaches and show how the approach of Cooper and Herskovitz can be improved by using the size of the prototype tree for determining the prior probabilities, introducing a bias towards simple structures.

Applying the theorems in [2] in our framework with m attributes and n_j values for each attribute A_j, we get for a Bayesian prototype tree with l prototypes the probability

$$P(\mathcal{D} \mid \mathcal{M}) = \frac{(l-1)!}{(N+l-1)!} \prod_{k=1}^{l} |c_k|! \cdot \prod_{j=1}^{m} \prod_{k=1}^{l} \left(\frac{(n_j-1)!}{(|c_k|+n_j-1)!} \prod_{i=1}^{n_j} F_k(a_{ji})! \right), \qquad (2)$$

where $|c_k|$ is the number of vectors partitioned in cluster c_k, and $F_k(a_{ij})$ is the number of vectors in c_k with the property $A_j = a_{ij}$. The parameters needed for applying this formula can be computed by using the partition of the data $\mathcal{D}$ used for creating the prototype set corresponding to the tree model $\mathcal{M}$.

To approximate the prior probability $P(\mathcal{M})$, let us first code all the different Bayesian trees using the information theoretic coding scheme in [15]. Using this coding scheme, we can approximate the prior probability as

$$P(\mathcal{M}) = 2^{-S(\mathcal{M})},$$

where $S(\mathcal{M})$ denotes the number of bits needed to represent Bayesian prototype tree $\mathcal{M}$. Our model contains l prototype vectors and l prior probabilities, one for each class c_k. In addition, to use our Bayesian prototype tree for Bayesian reasoning, we need one prototype vector for representing the initial attribute-value distribution and the outcome of the computation. Hence the total description length for the Bayesian tree structure is

$$S(\mathcal{M}) = l \sum_{j=1}^{m} n_j r + lr + \sum_{j=1}^{m} n_j r, \qquad (3)$$

where r is the number of bits needed to store a truncated real value. Observe that for a particular problem domain m, n_j and r will be constants, but l can vary from 1 to N (the size of the data set). This is not the only encoding scheme possible, but

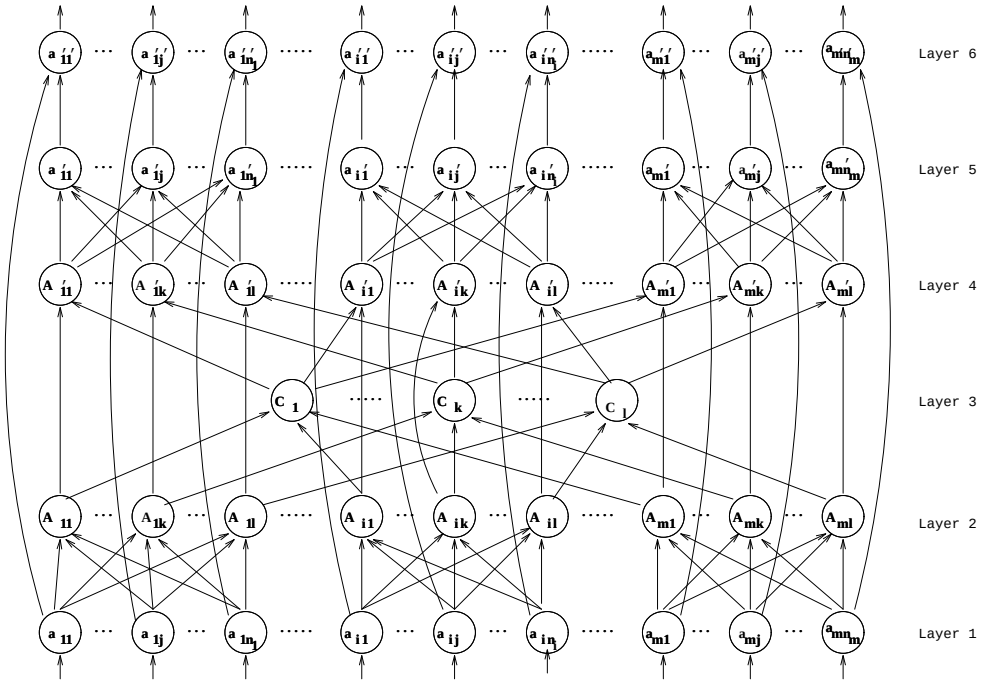

Figure 3: The probabilistic neural network implementing Bayesian MBR.

it is straightforward and reflects well the intuitive bias towards "simpler" trees, i.e., trees with the root node having less possible values. This translates in the neural network level to a probabilistic network with fewer nodes in the prototype layer.

Combining the equations (2) and (3), we get

$$P(\mathcal{M} \mid \mathcal{D}) = c2^{-S(\mathcal{M})}P(\mathcal{D} \mid \mathcal{M}), \qquad (4)$$

where $S(\mathcal{M})$ is the size of the Bayesian tree structure as defined in (3), $P(\mathcal{M} \mid \mathcal{D})$ is computed using formula (2), and c can be any positive constant.

IV. Searching for optimal prototype trees

Given a Bayesian tree with l prototypes, and a partition of the data D into these l classes, we can use the criteria (4) for computing the probability of the given Bayesian tree structure, given the data, and hence evaluate the goodness of the tree representation. Although in the general case the number of possible Bayesian network structures grows super-exponentially with the number of nodes [16], in our tree structure domain it is reasonable to assume that the number of prototypes does not exceed the size of the data set, so the number of tree structures to be evaluated grows only linearly with the number of data vectors. However, for a Bayesian tree with l prototypes, there are l^N ways to cluster the data, if the size of the data set is N, so it is clearly computationally infeasible to use the proposed scheme for evaluating all the possible tree structures with all the possible data partitions. We propose here a simple greedy heuristics, which starts by partitioning all the data vectors in separate clusters, and continues by combining the two prototypes which produce the greatest increase in the probability $P(\mathcal{M} \mid \mathcal{D})$.

Algorithm 1

1. l = n;
2. for i = 1 to N do
   (a) Assign data $\vec{d}_i$ to class c_i;
3. for i = N downto 1 do
   (a) $P_{\max} = P(\mathcal{M} \mid \mathcal{D})$; /* use eq. (4) */
   (b) $c = c_i$;
   (c) for k = 1 to l do
      i. If $(c_k = c_i)$ goto 3(c);
      ii. Assign data $\vec{d}_i$ to class c_k;
      iii. $P_k = P(\mathcal{M} \mid \mathcal{D})$; /* use eq. (4) */
      iv. If $(P_k \geq P_{\max})$
         A. $P_{\max} = P_k$;
         B. $c = c_k$;
   (d) if $(c = c_i)$ goto 3;
   (e) Assign data $\vec{d}_i$ to class c;
   (f) Destroy prototype c_i; $l = l - 1$;

The probability $P(\mathcal{M} \mid \mathcal{D})$ can be computed in $O(lmn)$ time, where $n = \max_j n_j$. Hence the complexity of the algorithm is $O(lmnN^2)$, and evaluating all the N possible tree structures with this greedy algorithm takes $O(lmnN^3)$ time steps.

V. Conclusion

We have presented a probabilistic framework for learning neural networks, by exploiting the direct correspondence between a tree structured Bayesian network representation and our probabilistic feedforward neural network model. As opposed to traditional neural network models, our network has a clear semantic interpretation: the prototypes are represented as a hidden layer in our network, and the activation values of the hidden nodes represent the similarity between the input vector and the corresponding prototype. The network is applicable not only to classification problems, but has properties of an interpolative associative memory: the output layer of a network gives the conditional probabilities for all the vector components, averaged over all the stored prototypes and weighted by the current activity values of the corresponding hidden nodes. We are currently experimenting with the proposed model by using data from a real medical diagnosis problem as the test set, and the initial results are very encouraging. Our next goal is to apply our system to public domain databases, in order to make comparisons to other existing methods possible.

References

[1] P. Cheeseman, J. Kelly, M. Self, J. Stutz, W. Taylor and D. Freeman, AutoClass: A Bayesian Classification System. Pp. 54–64 in *Proc. of the Fifth International Conference on Machine Learning*, Ann Arbor, June 12–14, 1988 (Morgan Kaufmann).

[2] G.F. Cooper and E. Herskovits, A Bayesian Method for Induction of Probabilistic Networks from Data. *Machine Learning*, vol. 9 , pp. 309–347, 1992.

[3] W. Hsu, L.S. Hsu and M.F. Tenorio, The Clusnet Algorithm and Time Series Prediction. *International Journal of Neural Systems*, vol. 4, no. 3, pp. 247–255, September 1993.

[4] H. Kitano, Challenges of Massive Parallelism. Pp. 813–834 in *Proc. of IJCAI-93, the Thirteenth International Joint Conference on Artificial Intelligence*, Chambéry, France, August 1993 (Morgan Kaufmann).

[5] I. Kononenko, Successive Naive Bayesian Classifier. *Informatica*, vol. 17, pp. 167–174, 1993.

[6] P. Langley, Induction of Recursive Bayesian Classifiers. Pp. 153–164 in P.B. Brazdil (ed.), *Proc. of ECML-93, European Conference on Machine Learning*, Vienna, Austria, April 5–7, 1993 (Springer-Verlag).

[7] W. Lam and F. Bacchus, Using Causal Information and Local Measures to Learn Bayesian Networks. Pp. 243–250 in D. Heckerman and A. Mamdani (eds.), *Proc. of the Ninth Conference on Uncertainty in Artificial Intelligence*, Washington, D.C., July 9–11, 1993 (Morgan Kaufmann).

[8] S. Lee and S. Shimoji, BAYESNET: Bayesian Classification Network Bsed on Biased Random Competition using Gaussian Kernels. Pp. 1354–1359 in *Proc. of the IEEE International Conf. on Neural Networks*, San Francisco, March 1993 (IEEE Press).

[9] J. Moody and C. Darken, Fast learning in networks of locally-tuned processing units. *Neural Computation*, vol. 1, 281–294, 1989.

[10] P. Myllymäki and H. Tirri, Bayesian Case-Based Reasoning with Neural Networks. Pp. 422–427 in *Proc. of the IEEE International Conf. on Neural Networks*, San Francisco, March 1993 (IEEE Press).

[11] P. Myllymäki and H. Tirri, Massively Parallel Case-Based Reasoning with Probabilistic Similarity Metrics. Pp. 48–53 in M.M. Richter, S. Wess, K.-D. Althoff and F. Maurer (eds.), *Proc. of the First European Workshop on Case-Based Reasoning*, University of Kaiserslautern, 1–5 November, 1993. SEKI Report SR-93-12 (SFB 314), University of Kaiserslautern.

[12] R. Neapolitan, **Probabilistic Reasoning in Expert Systems**. Wiley Interscience, 1990.

[13] E. Parzen, On estimation of a probability distribution and mode. *Annals of Mathematical Statistics*, vol. 33, pp. 1065–1076, 1962.

[14] J. Pearl, **Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference**. Morgan Kaufmann, 1988.

[15] J. Rissanen, **Stochastic Complexity in Statistical Inquiry**. World Scientific, 1989.

[16] W.R. Robinson, Counting unlabeled acyclic digraphs. In C.H.C. Little (ed.), **Lecture notes in mathematics 622: Combinatorial mathematics**. Springer-Verlag, 1977.

[17] D.F. Specht, Probabilistic Neural Networks. *Neural Networks*, vol. 3, pp. 109–118, 1990.

[18] D.F. Specht, A General Regression Neural Network. *IEEE Transactions on Neural Networks*, vol. 2, No. 6, pp. 568–576, November 1991.