

# Tractable Multivariate Binary Density Estimation and the Restricted Boltzmann Forest

**Hugo Larochelle**

*larocheh@cs.toronto.edu*

*Department of Computer Science, University of Toronto, Toronto, Canada M5S 3G4*

**Yoshua Bengio**

*yoshua.bengio@umontreal.ca*

*Dept. IRO, Université de Montréal, Montreal, Canada H3T 1J4*

**Joseph Turian**

*turian@iro.umontreal.ca*

*Dept. IRO, Université de Montréal, Montreal, Canada H3T 1J4*

**Keywords:** restricted Boltzmann machine, density estimation, mixture modeling

## Abstract

We investigate the problem of estimating the density function of multivariate binary data. In particular, we focus on models for which computing the estimated probability of any data point is tractable. In such a setting, previous work has mostly concentrated on mixture modeling approaches. We argue that, for the problem of tractable density estimation, the Restricted Boltzmann Machine (RBM) provides a competitive framework for multivariate binary density modeling. With this in mind, we also generalize the RBM framework and present the

Restricted Boltzmann Forest (RBFforest), which replaces the binary variables in the hidden layer of RBMs with groups of tree-structured binary variables. This extension allows us to obtain models that have more modeling capacity but that remain tractable. In experiments on several datasets, we demonstrate the competitiveness of this approach and study some of its properties.

## 1 Introduction

In this work, we consider the problem of learning densities for multivariate binary data. Such data can be found in various problems of pattern recognition: character recognition (Kassel, 1995), medical diagnosis (Aitchison & Aitken, 1976) and many natural language processing problems (Juan & Vidal, 2001). In particular, we focus on models which give a tractable estimate of the density function, i.e. models that can compute their estimated value of  $p(\mathbf{x})$  in a reasonable, practical amount of time. This constraint on a model is sometimes required, for instance if one wishes to use it as a module in a larger probabilistic model (e.g. a mixture model) or in a Bayes classifier, the later being a useful approach to classification for problems with scarce labeled data (Ng & Jordan, 2002). So far, the dominating approach to tractable density estimation has been mixture modeling, where a data point is assumed to have been generated from one out of  $m$  hidden components

$$p(\mathbf{x}) = \sum_{k=1}^m p(\mathbf{x}|k)p(k) . \quad (1)$$

Models falling in this category include mixtures of Bernoullis (Everitt & Hand, 1981; Carreira-Perpiñán & Renals, 2000; Juan & Vidal, 2001; Juan & Vidal, 2004; Lowd & Domingos, 2005) and non-parametric kernel estimators (Aitchison & Aitken, 1976). An alternative to mixture models are factor models, under which each data point is generated based on the value of a combination of individual factors:

$$p(\mathbf{x}) = \sum_{\mathbf{h} \in \mathcal{H}^l} p(\mathbf{x}|\mathbf{h})p(\mathbf{h}) = \sum_{h_1 \in \mathcal{H}} \cdots \sum_{h_l \in \mathcal{H}} p(\mathbf{x}|\mathbf{h})p(\mathbf{h}) \quad (2)$$

where  $\mathbf{h} = (h_1, h_2, \dots, h_l)$  is structured as a tuple of  $l$  factors. The number of different values it can take is exponential in  $l$ , even though the number of free parameters usually scales linearly in  $l$ . For example, in a logistic belief network with one hidden layer (Neal, 1992),  $p(\mathbf{x}|\mathbf{h})$  is factorized in several linear logistic regressions

$\prod_{i=1}^d p(x_i|\mathbf{h})$  and all binary variables (units) in the hidden layer are assumed to be independent (i.e.  $p(\mathbf{h}) = \prod_{j=1}^l p(h_j)$ ). Hence, another interpretation of a factor model is as a mixture model with an exponential number of components, where all components share parameters.

The computations required by Equation 2, which now involves an exponential sum, can be reduced if  $p(\mathbf{x})$  can be factorized in a numerator  $q(\mathbf{x})$  that is efficient to compute and a normalization constant  $Z$  that is more expensive:

$$p(\mathbf{x}) = \sum_{\mathbf{h} \in \mathcal{H}^l} p(\mathbf{x}, \mathbf{h}) = \frac{q(\mathbf{x})}{Z}. \quad (3)$$

This decomposition is generally not possible for directed graphical models (like a logistic belief network), but it is for some undirected or energy-based graphical models. To save computations, one can compute the normalization constant  $Z$  once, after training, so that the marginal cost of the computation of  $p(\mathbf{x})$  for any number of new data points will depend only on the cost of computing  $q(\mathbf{x})$ . One such energy-based model is the Restricted Boltzmann Machine (RBM) (Smolensky, 1986), for which the computation of  $q(\mathbf{x})$  is linear in the number of inputs and the number of hidden units.

Unfortunately, even with the factorization of the numerator, computing the normalization constant  $Z$  of an RBM quickly becomes intractable as we increase the number of hidden units. With only 20 hidden units, computing  $Z$  already requires a summation over about a million ( $2^{20}$ ) configurations of the hidden units. Such a number is reasonable in practice, however 20 hidden units is far from the regime in which RBMs are usually employed, e.g. to extract a new representation for the input (Hinton et al. (2006) use RBMs with 500 and 2000 hidden units). For this reason, small RBMs have never been considered a good alternative to mixture models for tractable density estimation.

The first contribution of this paper is an empirical demonstration that, even in its tractable regime, an RBM can often be a competitive tractable density estimator. However, there are some cases where the capacity of the RBM is too small in the tractable regime.

As a second contribution, this paper addresses this situation and presents a generalization of the RBM, which we call the Restricted Boltzmann Forest (RBFforest). In the RBFforest, we replace the binary hidden variables of the RBM with groups of tree-structured binary variables. By varying the size of the trees, the number of parameters

of the model can be increased while keeping the computations of  $p(\mathbf{x})$  tractable. We describe efficient algorithms for inference and training in the RBForest, and we study some of its properties.

## 2 Restricted Boltzmann Machines (RBMs)

An RBM defines a probability distribution over a binary input vector  $\mathbf{x}$  and a layer of binary hidden variables  $\mathbf{h}$ , through a bilinear energy function over these two vectors

$$E(\mathbf{x}, \mathbf{h}) = -\mathbf{b}^T \mathbf{x} - \mathbf{h}^T \mathbf{W} \mathbf{x} - \mathbf{c}^T \mathbf{h}. \quad (4)$$

The energy function is converted into a probability function as follows:

$$p(\mathbf{x}, \mathbf{h}) = \frac{e^{-E(\mathbf{x}, \mathbf{h})}}{Z} \quad (5)$$

where the normalization constant  $Z$  ensures that Equation 5 defines a valid distribution.

It is straightforward to show that

$$p(\mathbf{x}|\mathbf{h}) = \prod_i p(x_i|\mathbf{h}), \quad p(x_i = 1|\mathbf{h}) = \text{sigm}(b_i + \sum_j W_{ji}h_j) \quad (6)$$

where  $\text{sigm}(a) = 1/(1 + e^{-a})$ .  $p(\mathbf{h}|\mathbf{x})$  also has a similar form:

$$p(\mathbf{h}|\mathbf{x}) = \prod_j p(h_j|\mathbf{x}), \quad p(h_j = 1|\mathbf{x}) = \text{sigm}(c_j + \sum_i W_{ji}x_i). \quad (7)$$

Equation 7 implies that inference over the hidden variables given an input pattern is easy and simple to compute, as it can proceed independently for each hidden variable  $h_j$ . Consider the computation of the likelihood of an input  $\mathbf{x}$ :

$$p(\mathbf{x}) = \frac{\sum_{\mathbf{h} \in \{0,1\}^l} e^{-E(\mathbf{x}, \mathbf{h})}}{Z}. \quad (8)$$

Computing the numerator is not a problem. Because of the factorial aspect of the numerator, it can be computed in  $O(dl)$ :

$$\begin{aligned} \sum_{\mathbf{h} \in \{0,1\}^l} e^{-E(\mathbf{x}, \mathbf{h})} &= \sum_{h_1 \in \{0,1\}} \dots \sum_{h_l \in \{0,1\}} e^{\mathbf{b}^T \mathbf{x} + \mathbf{h}^T \mathbf{W} \mathbf{x} + \mathbf{c}^T \mathbf{h}} \\ &= e^{\mathbf{b}^T \mathbf{x}} \left( \sum_{h_1 \in \{0,1\}} e^{h_1 \mathbf{W}_{1,:} \mathbf{x} + c_1 h_1} \right) \dots \left( \sum_{h_l \in \{0,1\}} e^{h_l \mathbf{W}_{l,:} \mathbf{x} + c_l h_l} \right) \\ &= e^{\mathbf{b}^T \mathbf{x}} \prod_{i=1}^l (1 + e^{\mathbf{W}_{i,:} \mathbf{x} + c_i}) \end{aligned} \quad (9)$$

where  $\mathbf{W}_{i,:}$  is the  $i^{\text{th}}$  row of  $\mathbf{W}$ . However, computing the normalization constant is exponentially expensive, either in  $l$  or  $d$ , depending on whether the factorization is made according to the input or the hidden variables, respectively. Factorizing according to the inputs, we get:

$$Z = \sum_{\mathbf{h} \in \{0,1\}^l} \sum_{\mathbf{x} \in \{0,1\}^d} e^{\mathbf{b}^T \mathbf{x} + \mathbf{h}^T \mathbf{W} \mathbf{x} + \mathbf{c}^T \mathbf{h}} = \sum_{\mathbf{h} \in \{0,1\}^l} e^{\mathbf{c}^T \mathbf{h}} \prod_{i=1}^d \left( 1 + e^{\mathbf{W}_{:,i}^T \mathbf{h} + b_i} \right) \quad (10)$$

where  $\mathbf{W}_{:,i}$  is the  $i^{\text{th}}$  column of  $\mathbf{W}$ .  $Z$  does not depend on  $\mathbf{x}$  and has to be computed only once. So one option to make the computation of  $p(\mathbf{x})$  tractable is to limit the number of hidden units  $l$  enough to make the exponential summation required by  $Z$  computationally feasible. Doing so however also limits the number of parameters and the capacity of the RBM. Yet it is unclear how important the impact is on the general performance of the RBM. Indeed, while being certainly less powerful than a bigger RBM, an RBM with only 20 hidden variables is still an implicit mixture over a million components (with shared parameters). So even if only a small fraction of those were effectively used and had a significant probability of being picked, it might be sufficient to be competitive with standard mixture models.

### 3 Tractable RBMs with larger hidden layers

Instead of limiting the number of hidden variables of the RBM, another approach for making  $p(\mathbf{x})$  tractable would be to directly limit the number of possible configurations of the elements of  $\mathbf{h}$ , by introducing structural constraints on allowable configurations. Perhaps the simplest such constraint we could impose corresponds to having multinomial groups of units, that is subsets of hidden units within which only one unit can be active at a time. The hidden layer of the RBM could then contain several multinomial groups and it would be possible to have many more hidden units than in a standard binary layer for the same total number of possible configurations for the hidden layer  $\mathbf{h}$ .

Unfortunately, in practice, using such units can be difficult. Indeed, because the units within each group essentially compete with each other to explain the input, we often observe that several hidden variables take a while (if ever) to overcome the influence of other hidden variables and can remain essentially untrained for several iterations.

For this reason, we propose instead to introduce tree constraints in the hidden layer of the RBM. As we will see (see Section 5.1), this constraint will make learning much easier than with multinomial groups. These constraints are also applied to groups of hidden variables, yielding a hidden layer acting as a set of trees (*forest*). For this reason, we dub this new model the Restricted Boltzmann Forest.

### 3.1 Restricted Boltzmann Forests (RBForesTs)

The general idea is depicted in Figure 1. Hidden variables are grouped in  $T$  perfect (i.e. full and complete) binary trees of depth  $D$ . In a tree, hidden variables must respect the following constraints. When a hidden variable is inactive ( $h_i = 0$ ) all hidden variables in its left subtree must be inactive. Likewise, if a hidden variable is active ( $h_i = 1$ ), its right subtree variables must be inactive. So the activity of hidden variables defines a path in the tree from the root to one of the leaves, which we will refer to as the active path<sup>1</sup>.

It should be noticed that the RBForest can be seen as a generalization of the standard RBM. Indeed, an RBM is simply an RBForest with trees of depth 0. So, with the RBForest, we effectively add an additional degree of freedom in the architecture design (the tree depth) over which it can be advantageous to do model selection. Note also that both the RBM and the RBForest use exactly the same energy function. However, in the RBForest, we don't allow configurations of  $\mathbf{h}$  that do not respect the tree constraints (which would be equivalent to assigning an infinite energy to those configurations).

### 3.2 Inference in the RBForest

Let us first define some notation. Let  $N(t)$  be the set of indices of the hidden variables in the  $t^{th}$  tree,  $A(i)$  be the indices of the ancestors of the node  $h_i$  and  $P(i)$  be the index of the parent of  $h_i$ .  $C(i)$  will be the vector of values  $C(i)_k$  that  $h_k$  must take for the activation of node  $h_i$  to be allowed, for  $k \in A(i)$ . Also, for notational convenience, if  $M$  is a set of indices of hidden variables (as are  $N(t)$  and  $A(i)$ ), we consider that  $\mathbf{h}_M$  refers to the sub-vector of  $\mathbf{h}$  containing the hidden variables in  $M$ . Finally,  $S(t)$  will refer to the set of all configurations of variables  $\mathbf{h}_{N(t)}$  in the  $t^{th}$  tree that respect the tree

---

<sup>1</sup>Notice that for a path to be active, all hidden variables in the path need not be active.

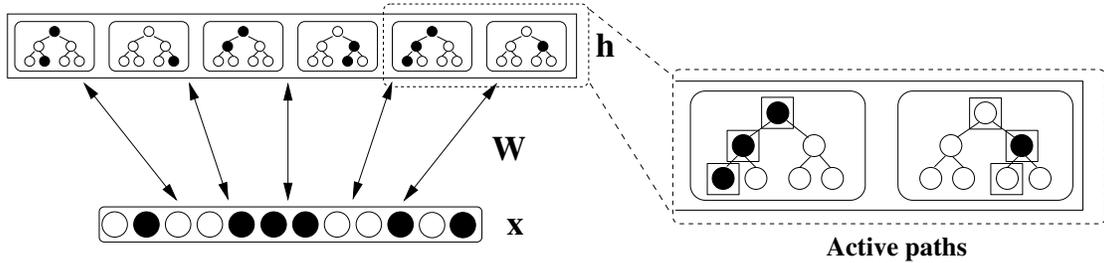


Figure 1: Illustration of a Restricted Boltzmann Forest for binary input vectors, where binary variables in the hidden layer are grouped in trees. When a hidden variable is inactive (white) or active (black), all hidden variables will accordingly be inactive in its left or right subtree. On the right (zoomed-in box), the variables in the active paths of two of the trees are identified.

constraints.

In an RBFforest, the conditional distribution  $p(\mathbf{x}|\mathbf{h})$  is unchanged and is as given in Equation 6. However,  $p(\mathbf{h}|\mathbf{x})$  is slightly more complex than in Equation 7. Since the tree constraints apply only within single trees, we obtain that  $p(\mathbf{h}|\mathbf{x})$  factorizes as  $p(\mathbf{h}|\mathbf{x}) = \prod_t p(\mathbf{h}_{N(t)}|\mathbf{x})$ . We can then develop the  $p(\mathbf{h}_{N(t)}|\mathbf{x})$  factors by taking advantage of the tree structure of  $\mathbf{h}_{N(t)}$ .

In particular, let us define the sums of exponentiated energies over all configurations compatible with  $h_i$  being active as  $L(i, N(t))$ ; similarly define  $R(i, N(t))$  for inactive ones:

$$L(i, N(t)) = \sum_{\mathbf{h}_{N(t)}|h_i=1, \mathbf{h}_{N(t)} \in S(t)} e^{-E(\mathbf{x}, \mathbf{h}_{N(t)})} \quad (11)$$

$$R(i, N(t)) = \sum_{\mathbf{h}_{N(t)}|h_i=0, \mathbf{h}_{N(t)} \in S(t)} e^{-E(\mathbf{x}, \mathbf{h}_{N(t)})} \quad (12)$$

where  $E(\mathbf{x}, \mathbf{h}_{N(t)})$  is the energy associated to the  $t^{\text{th}}$  tree for hidden variables  $\mathbf{h}_{N(t)}$  and input  $\mathbf{x}$ , i.e.,  $E(\mathbf{x}, \mathbf{h}_{N(t)})$  contains the terms of  $E(\mathbf{x}, \mathbf{h})$  in Equation 4 that are specific to  $\mathbf{x}$  and  $\mathbf{h}_{N(t)}$ .

For  $h_i$  in  $\mathbf{h}_{N(t)}$ , we have that the *tree-local distribution*  $p(h_i = 1|\mathbf{h}_{A(i)}, \mathbf{x})$  is simply:

$$p(h_i = 1|\mathbf{h}_{A(i)}, \mathbf{x}) = \frac{L(i, N(t))}{L(i, N(t)) + R(i, N(t))} \quad (13)$$

when  $\mathbf{h}_{A(i)} = C(i)_{A(i)}$  and 0 otherwise. Then, using this tree-local distribution we can

write

$$p(\mathbf{h}_{N(t)}|\mathbf{x}) = \prod_{i \in \text{path}(\mathbf{h}_{N(t)})} p(h_i|\mathbf{h}_{A(i)}, \mathbf{x}) \quad (14)$$

when  $\mathbf{h}_{N(t)}$  respects the tree constraints (otherwise  $p(\mathbf{h}_{N(t)}|\mathbf{x}) = 0$ ). Here,  $\text{path}(\mathbf{h}_{N(t)})$  is the set of hidden variables in the active path of the  $t^{\text{th}}$  tree with value  $\mathbf{h}_{N(t)}$ . To sample from  $p(\mathbf{h}_{N(t)}|\mathbf{x})$ , we first sample the root variable  $h_{\text{root}(t)}$  from  $p(h_{\text{root}(t)}|\mathbf{x})$  ( $A(\text{root}(t))$  is empty). We then move to the left or right subtree accordingly. We repeat this sampling procedure by using the tree-local distribution of Equation 13 until a leaf has been sampled. The marginal probability of each hidden variable being active,  $p(h_i = 1|\mathbf{x})$ , can be computed as follows:

$$p(h_i = 1|\mathbf{h}_{A(i)} = C(i)_{A(i)}, \mathbf{x}) \prod_{j \in A(i)} p(h_j = C(i)_j|\mathbf{h}_{A(j)} = C(i)_{A(j)}, \mathbf{x}) \quad (15)$$

or in a recursive form as follows:

$$p(h_i = 1|\mathbf{h}_{A(i)} = C(i)_{A(i)}, \mathbf{x}) p(h_{P(i)} = C(i)_{P(i)}|\mathbf{x}). \quad (16)$$

Let's now consider computing the terms  $L(i, N(t))$  and  $R(i, N(t))$  of Equation 11 and 12. These terms are required to compute the tree-local distributions. A naive computation of all these terms would be linear in the number of hidden variables (nodes in all trees) *and* in the depth of that tree. However, using the tree constraints, we have that for a non-leaf  $h_i$  and its two children  $h_j$  and  $h_k$ , the following holds:

$$L(i, N(t)) = e^{\mathbf{W}_{i,\cdot} \mathbf{x}} (L(j, N(t)) + R(j, N(t))) \quad (17)$$

$$R(i, N(t)) = L(k, N(t)) + R(k, N(t)) \quad (18)$$

and for a leaf  $h_i$  we have  $L(i, N(t)) = e^{\mathbf{W}_{i,\cdot} \mathbf{x}}$  and  $R(i, N(t)) = 1$ . We can hence obtain all  $L(i, N(t))$  and  $R(i, N(t))$  terms by proceeding level-wise, first assigning the value of these terms for the leaves and going upwards to compute all other terms. This bottom-up pass is then linear only in the number of hidden variables. The pseudocode for this procedure is given in the Appendix.

Once all terms have been computed, and using Equation 13 to rewrite Equation 16 as follows

$$p(h_i = 1|\mathbf{x}) = \frac{L(i, N(t)) p(h_{P(i)} = C(i)_{P(i)}|\mathbf{x})}{L(i, N(t)) + R(i, N(t))}, \quad (19)$$

we see that a top-down pass starting at the root with

$$p(h_{\text{root}(t)} = 1|\mathbf{x}) = \frac{L(\text{root}(t), N(t))}{L(\text{root}(t), N(t)) + R(\text{root}(t), N(t))} \quad (20)$$

can be used to compute all marginal probabilities of the hidden variables. This computation is also linear in the number of hidden variables in the tree. Pseudocodes of the sampling and inference procedures for the RBFforest are given in the Appendix.

### 3.3 Learning in the RBFforest

To train an RBFforest, Contrastive Divergence (CD) (Hinton, 2000) can be used just as in a regular RBM. CD provides an efficient approximation for the gradient of the negative log-likelihood (NLL) of some input  $\mathbf{x}_t$  with respect to any parameter  $\theta$

$$\frac{\partial -\log p(\mathbf{x}_t)}{\partial \theta} = \mathbf{E}_{\mathbf{h}|\mathbf{x}_t} \left[ \frac{\partial}{\partial \theta} E(\mathbf{x}_t, \mathbf{h}) \right] - \mathbf{E}_{\mathbf{x}, \mathbf{h}} \left[ \frac{\partial}{\partial \theta} E(\mathbf{x}, \mathbf{h}) \right]. \quad (21)$$

CD uses a short Gibbs chain of  $k$  steps starting at  $\mathbf{x}_t$  to obtain an approximate sample  $\mathbf{x}^{\text{neg}}$  from the model’s distribution and uses this sample to get a point estimate of the second expectation over  $\mathbf{x}$ . The conditional expectation over  $\mathbf{h}$  given  $\mathbf{x}^{\text{neg}}$  can then be done exactly and involves computing the conditional probabilities  $p(h_i = 1|\mathbf{x})$  of individual hidden variables being active. To train the model, one can then use this estimate of the gradient on the parameters to perform stochastic gradient descent. For more details, see the pseudocode in the Appendix. The only two differences between CD in a regular RBM and in an RBFforest are: (1) in the sampling procedure of the hidden layer given a value for the input layer and (2) in the computation of  $p(h_k = 1|\mathbf{x})$  for the positive and negative phase updates. Development of better learning algorithms for RBMs is currently an active area of research (see Tieleman (2008); Tieleman and Hinton (2009)) from which RBFforests should also benefit.

### 3.4 Computing $p(\mathbf{x})$

There remains the question of how to compute  $p(\mathbf{x})$  in an RBFforest. The formula is similar to that of Equation 8, with the sum over  $\mathbf{h} \in \{0, 1\}^l$  being replaced with a sum over values of  $\mathbf{h}$  that respect the tree constraints. More specifically, using Equ-

tions 11 and 12 and the linearity of  $E(\mathbf{x}, \mathbf{h})$  in  $\mathbf{h}$ , we have:

$$\begin{aligned} \sum_{\mathbf{h} | \mathbf{h}_{N(t)} \in S(t) \forall t} e^{-E(\mathbf{x}, \mathbf{h})} &= \left( \sum_{\mathbf{h}_{N(1)} \in S(1)} e^{-E(\mathbf{x}, \mathbf{h}_{N(1)})} \right) \dots \left( \sum_{\mathbf{h}_{N(T)} \in S(T)} e^{-E(\mathbf{x}, \mathbf{h}_{N(T)})} \right) \\ &= \prod_{t=1}^T (L(\text{root}(t), N(t)) + R(\text{root}(t), N(t))) \end{aligned}$$

where each of the  $L(\text{root}(t), N(t))$  and  $R(\text{root}(t), N(t))$  can efficiently be computed, as described previously. As for the computation of  $Z$ , its formula is that of Equation 10 where the sum over  $\mathbf{h} \in \{0, 1\}^l$  is replaced with a sum over  $\mathbf{h} | \mathbf{h}_{N(t)} \in S(t) \forall t \in \{1, \dots, T\}$ . This exponential sum must be done explicitly. However, it is still possible to keep it reasonably small while increasing the number of units  $l$ , by choosing appropriate values for the number of trees  $T$  and their depth  $D$ . For instance, an RBFforest with  $T = 5$  trees of depth  $D = 3$  will also have  $2^{(D+1)T} = 2^{20}$  terms in the sum over  $\mathbf{h}$ , just like in an RBM with 20 hidden variables. However, that RBFforest will have 75 hidden variables (and their corresponding parameters, weights and biases), more than three times as many. See the Appendix for pseudocodes computing  $Z$  and  $p(\mathbf{x})$ .

## 4 Related Work

Salakhutdinov and Murray (2008) have proposed a technique to approximate the value of the normalization constant  $Z$  for larger RBMs, making it possible to get an approximate estimate of the value of  $p(\mathbf{x})$ . We emphasize that this work and ours have different goals. Indeed, they were interested in evaluating the generative modeling capacity of large RBMs (i.e. in the regime they are usually used in), so having only an approximate estimate of  $p(\mathbf{x})$  was sufficient. Here, we specifically focus on the case where  $p(\mathbf{x})$  is tractable and exact (which is useful for using an RBM in a larger probabilistic model or in a Bayes classifier) and argue that even in that regime the RBM framework and its RBFforest generalization are competitive when compared to other tractable approaches.

## 5 Experiment: density estimation

We present here an experimental comparison of the RBM and the RBFforest with a standard mixture of Bernoullis (MoB). Just like RBMs and RBFforest, the “emission” prob-

ability distribution given the hidden state is also a product of independent Bernoullis. Hence, the only difference between the MoB, RBM and RBFforest lies in the nature of the prior distribution over the hidden state. In the MoB, this prior is explicit, and corresponds to a multinomial over the  $M$  possible mixture components, whereas in the RBM and RBFforest, the prior is implicit and more complex. This makes the MoB a perfect choice for experimental comparisons, which in essence will specifically evaluate the impact of changing the nature of the prior over the hidden state. Moreover, it has been argued previously that the MoB is a competitive density estimator in general (Lowd & Domingos, 2005).

The experiment evaluation was conducted on several datasets of multivariate binary data. These datasets vary in nature (text, image, biological and game related data) and in size (from a few hundred to many thousands of examples), while all being of relatively high dimensionality (between 100 and 500 inputs). The majority of these datasets were taken from the LIBSVM datasets web page<sup>2</sup>, with the exception of the *ocr-letter* dataset<sup>3</sup> and the *nips-0-12* dataset<sup>4</sup>. All datasets were divided in training, validation, and test sets. The validation set NLL was used as a criteria to select good values for the hyper-parameters, among combinations of values for the learning rate  $\eta$  (in  $\{0.005, 0.0005, 0.00005\}$ ), the number of CD steps<sup>5</sup> (in  $\{10, 25\}$ ) and the number of iterations over the training set (in  $\{100, 500, 1000\}$ )<sup>6</sup>. The RBM had 23 hidden variables, and we considered RBFforests with 3, 4, 5, 7 and 11 trees of depth 5, 4, 3, 2 and 1 respectively, so that the total number of hidden layer configurations would be less than 10 million. The mixtures of multivariate Bernoullis were trained with the EM algorithm, using the number of components (in  $\{32, 64, 128, 256, 512, 1024\}$ ) chosen based on the validation set NLL<sup>7</sup>. Early stopping based on the average NLL of the validation set was also used, with a look ahead of 5 iterations, and 2 random initializations of EM were always tested.

---

<sup>2</sup>See <http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/>.

<sup>3</sup>See <http://ai.stanford.edu/~btaskar/ocr/>.

<sup>4</sup>See <http://www.cs.toronto.edu/~roweis/data.html>.

<sup>5</sup>For the *mushrooms* and *nips-0-12* datasets, which are smaller, we also considered 50 steps of CD

<sup>6</sup>No weight decay was used, since choosing an appropriate number of iterations seemed sufficient to avoid overfitting.

<sup>7</sup>In our experiments, it turns out that 1024 was never chosen as the best number of components. Hence, bigger mixtures would certainly have led to overfitting.

Table 1: Density estimation experiment results, with details about the different datasets. The comparison is made by looking at the difference between the test set average NLL of the MoB with that of the standard RBM, the RBM with multinomial groups and RBForest (i.e. the RBM, RBM mult. and/or RBForest outperform the MoB if this difference is positive). The 95% confidence intervals were estimated based on a two-sample t-test statistic.

Dataset	Nb. inputs	Set sizes (Train,Valid,Test)	Diff. NLL RBM	Diff. NLL RBM mult.	Diff. NLL RBForest
<i>adult</i>	123	5000,1414,26147	<b>4.18 ± 0.11</b>	<b>4.15 ± 0.11</b>	<b>4.12 ± 0.11</b>
<i>connect-4</i>	126	16000,4000,47557	<b>0.75 ± 0.04</b>	-1.72 ± 0.05	<b>0.59 ± 0.04</b>
<i>dna</i>	180	1400,600,1186	<b>1.29 ± 0.72</b>	<b>1.45 ± 0.67</b>	<b>1.39 ± 0.73</b>
<i>mushrooms</i>	112	2000,500,5624	-0.69 ± 0.13	-0.69 ± 0.11	0.042 ± 0.12
<i>nips-0-12</i>	500	400,100,1240	<b>12.65 ± 1.55</b>	<b>11.25 ± 1.55</b>	<b>12.61 ± 1.55</b>
<i>ocr-letter</i>	128	32152,10000,10000	-2.49 ± 0.44	<b>0.99 ± 0.43</b>	<b>3.78 ± 0.43</b>
<i>rcv1</i>	150	40000,10000,150000	-1.29 ± 0.16	-0.044 ± 0.16	<b>0.56 ± 0.16</b>
<i>web</i>	300	14000,3188,32561	<b>0.78 ± 0.30</b>	0.018 ± 0.31	-0.15 ± 0.31

The results are displayed in Table 1, where the difference between the NLL achieved by the MoB and the other models are given. Confidence intervals at 95% confidence level are also given, based on the independent two-sample t-test statistic for equal sample sizes and equal variance<sup>8</sup>.

The first surprising observation is that, even with only 23 hidden units, the RBM

<sup>8</sup>While the assumption of equal variance is reasonable, the assumption of independence is of course clearly wrong. However, because we expect the correlation between the NLL of two models on the same example to be positive (e.g. the easy examples tend to be the same across models), this correlation would actually decrease the variance of the differences in average NLL. In other words, the assumption of independence actually makes the confidence intervals of Table 1 more conservative (i.e. wider) than they could be. Our approximation is better, but similar in the sense of being overconservative, to the common practice of computing NLLs and confidence intervals individually for each model and putting those in a table instead (hence ignoring correlations between individual example errors across models).

outperforms the MoB in more than half of the cases. Hence, while experiments in previous work on such small RBMs (Tieleman, 2008; Tieleman & Hinton, 2009) might have seemed of limited relevance to a practical application, it appears that this regime can still be competitive when compared to a standard mixture modeling approach.

Moreover, in the three cases where the RBM performs less well, its RBForest generalization allows us to reach and/or outperform the MoB. Notice that this experiment was designed to distinguish the performance of a standard RBM with the performance that is achievable when using the additional modeling degrees of freedom that the RBForest provides (i.e. the depth of a tree). In other words, in a case where the RBM performed better than the RBForest (e.g. on the *web* dataset), the RBForest would have reached the same performance if we had allowed the trees to be of depth 0.

We have found that for all datasets except *ocr-letter* and *rcv1*, the performance was worse when using only 3 trees (the smallest number of trees considered by RBForest). For *ocr-letter* and *rcv1*, we ran experiments with RBForests containing only one large tree (depth 8, i.e. 511 hidden units), but the performance was much worse and did not even reach that of the MoB. So it appears that the use of many trees is important.

In general, we expect that the RBM and RBForest should be particularly appropriate for datasets where the data is thought to have been generated from a combination of many individual factors that are not mutually exclusive. In the standard RBM, these factors are assumed to be binary, while in the RBForest each factor is assumed to take one value out of possibly many more (one for each tree path). Alternatively, these models can be viewed as mixture models with a large (exponential) number of components, but where the components share parameters. The *ocr-letter* dataset is a good example. While all 26 characters are different, they each can be decomposed into smaller patterns or pen strokes (i.e. the factors) and different characters can share some of those patterns (for instance, 'i' and 'l' both share a vertical center line in their bottom half). On the other hand, not all combinations of those smaller patterns make sense. In particular, we could easily define small sets of pen strokes (i.e. the possible values of the factors) such that only one of them can be found in any given character. The RBForest can capture both of these characteristics (composition of factors, where each factor can take only one of many different values). This might explain why RBForest is the best performing method on the *ocr-letter* dataset.

## 5.1 Comparison with groups of multinomial hidden variables

As previously mentioned, one could have used multinomial groups to make the computation of  $Z$  tractable, instead of tree-structured groups of hidden variables. Actually, it is possible to show that for any RBFforest, there exists an equivalent energy-based model with multinomial groups of hidden variables that computes exactly the same distribution (but with a different parametrization). We sketch the construction here: consider the case of an RBFforest with only one tree and with parameters  $\mathbf{W}$ ,  $\mathbf{b}$  and  $\mathbf{c}$ . Then, consider a multinomial group version of that RBFforest (with parameters  $\mathbf{W}^*$ ,  $\mathbf{b}^*$  and  $\mathbf{c}^*$ ) by associating each bit  $h_i^*$  of a multinomial variable with a path  $\text{path}(\mathbf{h})$  in the RBFforest tree and setting its weight vector to  $\mathbf{W}_{i,:}^* = \mathbf{h}^T \mathbf{W}$  and its bias to  $c_i^* = \mathbf{h}^T \mathbf{c}$ . Finally, use the same input bias vector  $\mathbf{b}^* = \mathbf{b}$ . Such a construction implies that each of these pairs of  $\mathbf{h}$  in the RBFforest and  $\mathbf{h}^*$  in the multinomial group version assign the same energy for any  $\mathbf{x}$ . So, these two models necessarily define the same distribution.

However, we motivate the use of trees by the empirical observation that hidden units in multinomial groups are often hard to train. Indeed, we observe that optimization is much easier with the RBFforest than with multinomial units. In the case of the RBFforest, we observe in practice that training progresses smoothly by first finding good values for the root node’s weights and then slowly moving towards optimizing the weights at the first level, and so forth. This is due mainly to the level-wise recursive nature of the value of  $p(h_k = 1|\mathbf{x})$  (see Equation 16), which is involved in the CD update of the weights  $\mathbf{W}$ . In the case of the multinomial groups, all the hidden variables are competing with each other to explain the input. So we observe that several hidden variables take a while (if ever) to overcome the influence of other hidden variables and can remain essentially untrained for several iterations. We illustrate this in Figure 2, where we show the weights of an RBFforest with one tree of depth 3 (i.e. 16 possible paths) and the weights of a model with one multinomial group of 16 (mutually exclusive) binary variables<sup>9</sup>.

We also designed an experiment in order to verify quantitatively that optimization is indeed easier when training the RBFforest. The idea is simple: we generated large

---

<sup>9</sup>Both models were trained on the *ocr-letter* training set for 10 iterations, with a learning rate of  $\eta = 0.005$  and using CD-10. We chose 10 iterations only to show the state of both models some way through training.

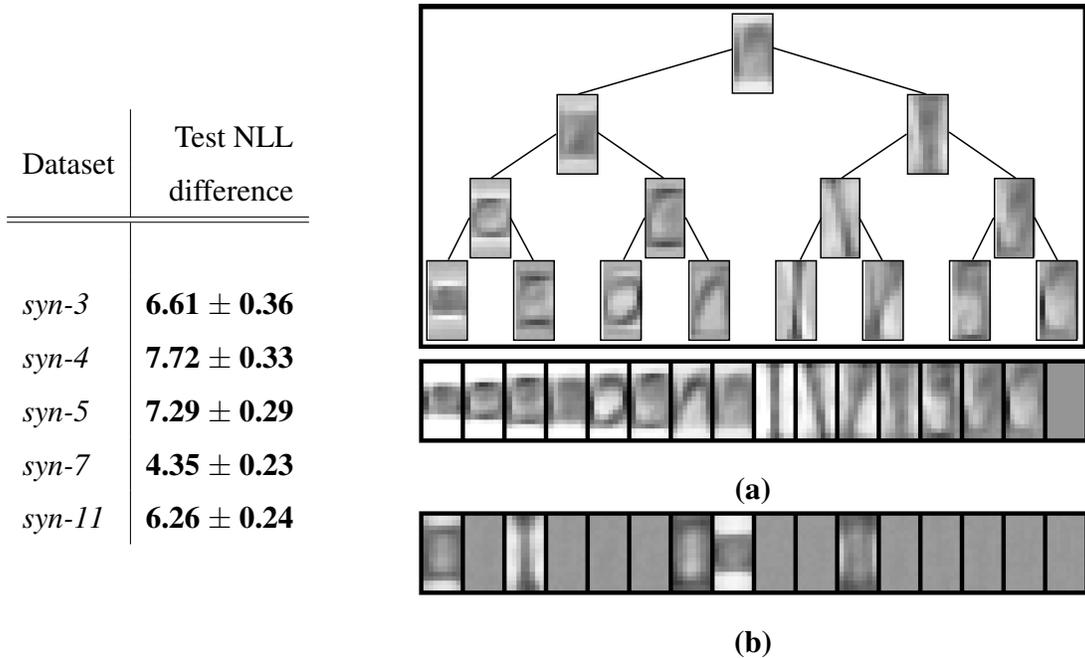


Figure 2: **(Left)** Comparison between using trees vs multinomial groups, on five synthetic datasets (referred to as *syn-T*, where  $T \in \{3, 4, 5, 7, 11\}$  is the number of multinomial groups used to generate the datasets). The NLL difference is the NLL obtained using multinomial groups minus the NLL obtained using trees. **(Right)** Illustration of training optimization for a tree of hidden variables and a multinomial group. **(a)** On top, illustration for a single-tree RBFforest of the weights  $\mathbf{W}_{i,:}$  for each hidden variable, and at bottom, illustration of the summed weights  $\mathbf{W}^T \mathbf{h}$  for all possible configurations of the hidden layer (i.e. the equivalent multinomial group weights). On the left of the tree root, weights captured different amounts of horizontal white backgrounds. On the right of the root, weights captured the structure of more vertically elongated characters. **(b)** Illustration of the weights learned using binary hidden variables forming a multinomial group. Most of them are still close to their initial near-zero value.

datasets from distributions that fall in the family of both the RBFforest and the RBM with multinomial groups, and quantitatively measured which of the two models achieved the best performance in terms of NLL. Because the generated training sets were large (50 000 examples), overfitting was not an issue and the best performing model was also the one which was most able to minimize its training error<sup>10</sup>. The distributions used to generate the datasets were obtained by randomly initializing the weights of RBMs with multinomial groups of equivalent sizes to the RBFforests considered in the experiment of Table 1 ( i.e. equivalent to RBFforests with 3, 4, 5, 7 and 11 trees of depth 5, 4, 3, 2 and 1 respectively). Each connection weight was sampled from a Normal distribution with mean 0 and a standard deviation of 10 (i.e. large enough to obtain a distribution with interesting structure). To ensure that there is an equivalent RBFforest corresponding to the same distribution, the connections to one hidden unit of each multinomial group was set to 0 (such a unit of a multinomial group would correspond to a right-most tree path in an RBFforest). While we could have instead initialized RBFforests and generated the datasets from them, we chose RBMs with multinomial groups instead to ensure that we were not favouring the RBFforest in any way. Training, validation and test sets of 50 000, 10 000 and 50 000 examples of dimensionality 200 were generated by sampling from those five RBMs with multinomial groups. The sampling procedure started at a vector sampled uniformly over all possible binary vectors, and used 500 steps of annealed Gibbs sampling<sup>11</sup> followed by 500 steps of normal Gibbs sampling. Finally, RBFforests and RBMs with multinomial groups were trained on each dataset, and the same values of learning hyper-parameters (i.e. learning rates, number of CD steps and number of iterations) as in the experiments of Table 1 were compared, with the best combination chosen based on the associated NLL error on the generated validation set. Before training, both the RBFforest and the RBM with multinomial groups had their connection weights initialized to small random values close to 0. For a given dataset, the number of multinomial groups and the size of the groups for the trained RBM as well as the number of trees and the size of the trees of the trained RBFforest were set

---

<sup>10</sup>This was verified empirically. In general, there were almost no difference between the training and test NLL errors for all synthetic datasets.

<sup>11</sup>By annealed Gibbs sampling, we mean Gibbs sampling where the conditional distributions used are those of the RBM but with its energy function divided by a temperature parameter. The temperature parameter was set at 100 for the first step and was linearly decreased to 1 until step 500.

to match (or, for the RBFforest, be equivalent to) those of the RBM that generated the dataset. Figure 2 shows the results. On all synthetic datasets, RBFforest clearly outperforms the RBM with multinomial groups, in agreement with the hypothesis of an easier optimization problem associated with RBFforests. Figure 3 also shows the progression of the test NLL errors for both the RBFforest and RBM with multinomial groups on one of the synthetic dataset (with the hyper-parameters chosen based on the validation error). We see that the superior performance of RBFforest is not due to a faster convergence and that it is indeed finding a better minima than the RBM with multinomial groups.

Table 1 also shows the performance of the RBM with multinomial groups on the 8 real world datasets. On half of the datasets, the RBFforest is found to be statistically better. On the others, no significant difference was detected. More importantly, improvements are found on the datasets where the RBM performs worse than the MoB.

So it appears that the contributions of RBFforests are two-fold. First, they extend the family of distributions that can be modeled in the energy-based framework relatively to standard RBMs (which are a particular case of RBFforests with trees of depth 0), while also improving the tractability of learning compared to structurally equivalent RBMs extended to have multinomial hidden units.

## 5.2 Training a mixture of RBFforests

We mentioned previously that one advantage of having a tractable density estimator is that it can be used in a larger probabilistic model. For instance, consider a mixture of  $M$  RBFforest components

$$p(\mathbf{x}) = \sum_{m=1}^M p(\mathbf{x}|C = m)p(C = m)$$

where  $p(\mathbf{x}|C = m)$  is given by the distribution of an RBFforest with parameters  $\mathbf{W}^m$ ,  $\mathbf{b}^m$  and  $\mathbf{c}^m$  (the different RBFforest components have different parameters). Such a mixture can be trained using the Generalized EM algorithm. During the E step, we simply compute the posterior probabilities (or responsibilities) of each RBFforest components having generated each training example

$$q(C = m|\mathbf{x}) = \frac{p(\mathbf{x}|C = m)p(C = m)}{\sum_{m=1}^M p(\mathbf{x}|C = m)p(C = m)}$$

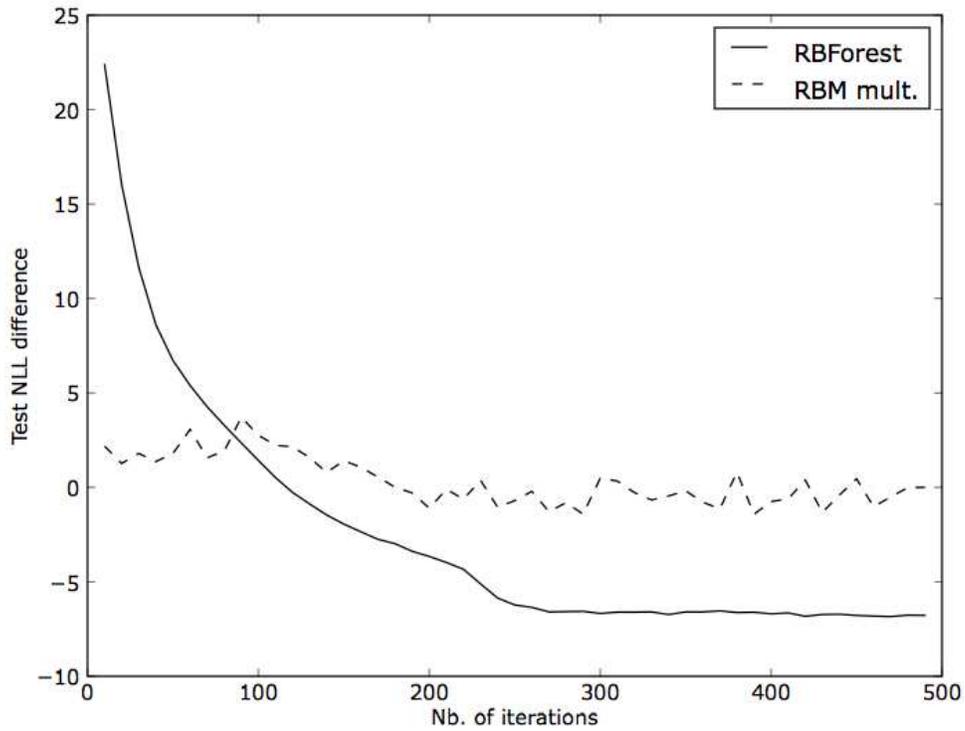


Figure 3: Learning curves for the RBFforest and the RBM with multinomial groups, on *syn-11* dataset. To normalize the curves, the value of the test NLL at the end of the curve for the RBM with multinomial groups was subtracted to all test NLL errors. We see that RBFforest is not simply converging faster, but is actually finding a better minima than RBM mult. The learning curve for RBM mult. is also much less stable. Using a smaller learning rate (not shown here) gives a more stable behavior for RBM mult., but worse performance.

which can be computed exactly since  $p(\mathbf{x}|C = m)$  is tractable in the RBFforest.

In the M step, we fix the values of the posterior probabilities computed in the E step and we minimize according to  $p(\mathbf{x}|C = m)$  and  $p(C = m)$  the following expected negative log-likelihood:

$$\begin{aligned} & - \sum_{\mathbf{x}_t \in \mathcal{D}} \sum_{m=1}^M q(C = m|\mathbf{x}_t) \log(p(\mathbf{x}_t|C = m)p(C = m)) \\ & = - \sum_{\mathbf{x}_t \in \mathcal{D}} \left( \sum_{m=1}^M q(C = m|\mathbf{x}_t) \log p(\mathbf{x}_t|C = m) - \sum_{m=1}^M q(C = m|\mathbf{x}_t) \log p(C = m) \right) \end{aligned} \quad (22)$$

where  $\mathcal{D}$  is the training set. The prior distribution  $p(C = m)$  minimizing Equation 22 is simply

$$p(C = m) = \frac{\sum_{\mathbf{x}_t \in \mathcal{D}} q(C = m|\mathbf{x}_t)}{|\mathcal{D}|}.$$

As for the RBFforest components, optimizing Equation 22 is equivalent to training each RBFforest components on a weighted version of the training set. The weight of example  $\mathbf{x}_t$  for the  $m^{\text{th}}$  RBFforest component is simply  $q(C = m|\mathbf{x}_t)$ , meaning that the learning rate is multiplied by  $q(C = m|\mathbf{x}_t)$  when updating the  $m^{\text{th}}$  RBFforest's parameters given example  $\mathbf{x}_t$ .

We trained a mixture of 8 RBFforest using the Generalized EM algorithm, on the *ocr-letter* dataset. The parameters of each RBFforest are initialized randomly. However, to break the initial symmetry, we also perform  $K$ -means on the training set (with  $K = M$ ) and perform 10 000 training updates for each RBFforest on different subsets of data as given by  $K$ -means. Then we proceed with the Generalized EM algorithm on the full training set. This larger model reached a test NLL difference of 6.27 with the MoB, improving on a single RBFforest (3.78).

Another approach to training mixtures of RBM-inspired models was also presented by Nair and Hinton (2009). Their framework for so-called implicit mixtures of RBMs is directly applicable to RBFforests, and the resulting density estimator would also be tractable, as long as each individual RBFforest is also tractable. In this work, the choice of training an explicit mixture (derived from a directed graphical model) instead of an implicit mixture (derived from an undirected graphical model) was made in order to show that the larger probabilistic system in which the RBFforest is used need not correspond to an undirected graphical model (like the RBFforest).

On a different but related topic, we also mention that the implicit mixtures of Nair and Hinton (2009) use multinomial units to index the mixture components and, as we have seen in Section 5.1, using such units can create difficulties during training. Nair and Hinton (2009) actually mention such difficulties, which forced them to introduce a temperature parameter when sampling the multinomial units. This parameter can be fixed to some tuned value, and can vary during training using some annealing schedule. The experiment of Figure 2 suggests that using tree-structured units might be another way to facilitate training in such implicit mixtures, maybe avoiding the need for tuning a temperature parameter.

### 5.3 Visualizing the implicit mixture components

A possible explanation for why the RBM and the RBFforest are able to perform well with relatively few parameters is that, because of their factorial nature, they implicitly learn a mixture of exponentially many components (see Equation 2). However, it is not necessarily clear whether they actually take advantage of this situation (for instance, only a handful of those components could end up having a significant probability of being picked). To verify this, we took the best RBFforest trained on the *ocr-letter* dataset (3 trees of depth 5) and grouped the test samples into different groups based on which implicit mixture components had the highest posterior probability (or responsibility). More precisely, for each test example  $\mathbf{x}_t$ , we found which of the  $64^3 = 262144$  implicit mixture component (i.e. which value of  $\mathbf{h}$ ) had the largest probability given  $\mathbf{x}_t$ :

$$\hat{\mathbf{h}}(\mathbf{x}_t) = \arg \max_{\mathbf{h} | \mathbf{h}_{N(t)} \in S(t) \forall t \in \{1, \dots, T\}} p(\mathbf{h} | \mathbf{x}_t) = \arg \max_{\mathbf{h} | \mathbf{h}_{N(t)} \in S(t) \forall t \in \{1, \dots, T\}} \prod_{t=1}^T p(\mathbf{h}_{N(t)} | \mathbf{x}_t)$$

Since  $p(\mathbf{h} | \mathbf{x}_t)$  factorizes according to each tree, finding  $\mathbf{h}$  with the largest probability simply requires finding the configuration  $\mathbf{h}_{N(t)}$  of each tree that maximizes  $p(\mathbf{h}_{N(t)} | \mathbf{x}_t)$  separately. Since each tree only has  $2^{D+1} = 2^6 = 64$  possible configurations of its units, finding the configuration with maximum probability can be done with an exhaustive search. Having found  $\hat{\mathbf{h}}(\mathbf{x}_t)$  for each test example  $\mathbf{x}_t$ , we then grouped together test examples sharing the same value for  $\hat{\mathbf{h}}(\mathbf{x}_t)$ .

Among the  $64^3 = 262144$  implicit mixture components, about 4000 were “responsible” for at least one test sample, hence much more than the  $63 \times 3 = 189$  hidden

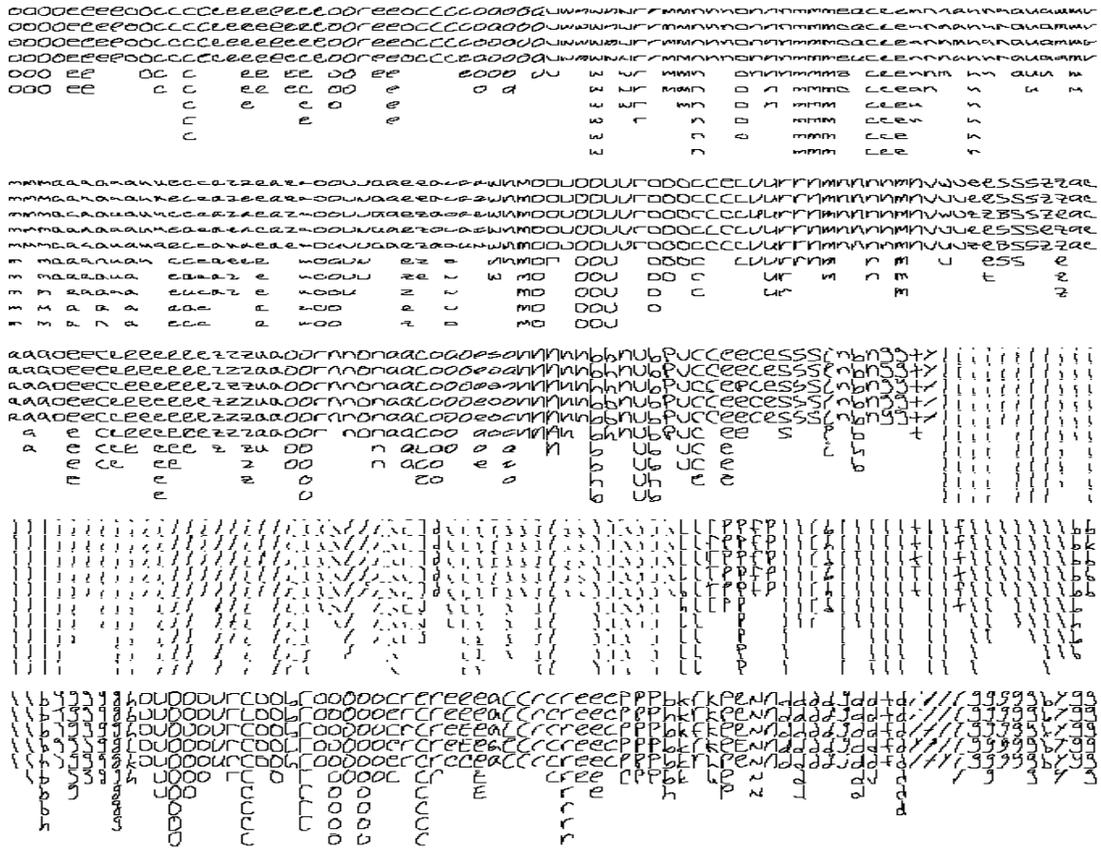


Figure 4: Visualization of 375 of the implicit mixture components (one per column) learned by the RBForest on the *ocr-letter* dataset.

units the RBForest has. Figure 4 illustrates a small subset of all groups of test samples sharing the same implicit component. We see that many implicit components captured some meaningful structure from the data, with groups often being class-specific, but with data from the same class being also split across different groups that capture specific variations on a character’s size or angle.

## Conclusion

We presented experimental evidence that, even in its tractable regime, the RBM is often a competitive model for multivariate binary density modeling. For cases where it lacks capacity, we proposed the Restricted Boltzmann Forest, a generalization of the RBM. Efficient inference and training algorithms were proposed for the RBForest, and the ad-

vantage of using trees was emphasized using a comparison with groups of multinomial units.

## Acknowledgments

We thank James Bergstra and anonymous reviewers for their helpful comments.

## Appendix

### Pseudocode for Inference and Sampling in the RBF

To do inference, we first need a function which computes the sum of exponentiated energies  $L(i, N(t))$  and  $R(i, N(t))$ , with a bottom-up pass.

**Algorithm:** SUM-EXP-ENERGIES-RBF( $\mathbf{x}$ )

**Input:** input  $\mathbf{x}$

**Output:** all sums of exponentiated energies  $L(i, N(t))$  and  $R(i, N(t))$

**for**  $t = 1$  to  $T$  **do**

  # Initialize energies

**for**  $i$  such that  $h_i$  is a leaf in tree  $t$  **do**

$$L(i, N(t)) \leftarrow e^{W_{i,:}\mathbf{x}}$$

$$R(i, N(t)) \leftarrow 1$$

**end for**

  # Bottom-up pass

**for**  $\delta = D - 1$  to  $0$  **do**

**for**  $i$  such that  $h_i$  is a node at level  $\delta$  of tree  $t$  **do**

$$j \leftarrow \text{index of left children of } h_i$$

$$k \leftarrow \text{index of right children of } h_i$$

$$L(i, N(t)) \leftarrow e^{W_{i,:}\mathbf{x}} (L(j, N(t)) + R(j, N(t)))$$

$$R(i, N(t)) \leftarrow L(k, N(t)) + R(k, N(t))$$

**end for**

**end for**

**end for**

From the values of  $L(i, N(t))$  and  $R(i, N(t))$ , we can then infer all the hidden variables' marginal probabilities given some input  $p(h_i = 1|\mathbf{x})$  with a top-down pass.

**Algorithm:** INFERENCE-RBF( $\mathbf{x}$ )

**Input:** input  $\mathbf{x}$

**Output:** marginal probabilities  $p(h_i = 1|\mathbf{x})$

# Get sum of exponentiated energies

$L(\cdot, \cdot), R(\cdot, \cdot) \leftarrow \text{SUM-EXP-ENERGIES-RBF}(\mathbf{x})$

**for**  $t = 1$  to  $T$  **do**

# Initialize top-down pass

$$p(h_{\text{root}(t)} = 1|\mathbf{x}) \leftarrow \frac{L(\text{root}(t), N(t))}{L(\text{root}(t), N(t)) + R(\text{root}(t), N(t))}$$

**for**  $\delta = 1$  to  $D$  **do**

**for**  $i$  such that  $h_i$  is a node at level  $\delta$  of tree  $t$  **do**

$$p(h_i = 1|\mathbf{x}) = \frac{L(i, N(t))p(h_{P(i)} = C(i)P(i)|\mathbf{x})}{L(i, N(t)) + R(i, N(t))}$$

**end for**

**end for**

**end for**

As for sampling hidden layer variables given some input, the procedure is very similar.

**Algorithm:** SAMPLE-HIDDEN-RBF( $\mathbf{x}$ )

**Input:** input  $\mathbf{x}$

**Output:** a sample  $\mathbf{h}$  from  $p(\mathbf{h}|\mathbf{x})$

# Get sum of exponentiated energies

$L(\cdot, \cdot), R(\cdot, \cdot) \leftarrow \text{SUM-EXP-ENERGIES-RBF}(\mathbf{x})$

**for**  $t = 1$  to  $T$  **do**

# Get root probability

$$p(h_{\text{root}(t)} = 1|\mathbf{x}) \leftarrow \frac{L(\text{root}(t), N(t))}{L(\text{root}(t), N(t)) + R(\text{root}(t), N(t))}$$

$h_{\text{root}(t)} \leftarrow$  sample from Bernoulli distribution with parameter  $p(h_{\text{root}(t)} = 1|\mathbf{x})$

```

i ← root(t)
for  $\delta = 1$  to D do
  if  $h_i = 1$  then
    i ← index of left child of  $h_i$ 
  else
    i ← index of right child of  $h_i$ 
  end if
  # Get tree local probability
   $p(h_i = 1 | \mathbf{h}_{A(i)} = C(i)_{A(i)}, \mathbf{x}) \leftarrow \frac{L(i, N(t))}{L(i, N(t)) + R(i, N(t))}$ 
   $h_i \leftarrow$  sample from Bernoulli distribution with
    parameter  $p(h_i = 1 | \mathbf{h}_{A(i)} = C(i)_{A(i)}, \mathbf{x})$ 
  end for
end for

```

Sampling the input (visible) units is exactly the same as in a standard RBM.

**Algorithm:** SAMPLE-VISIBLE-RBF(**h**)

**Input:** value of the hidden units **h**

**Output:** a sample **x** from  $p(\mathbf{x} | \mathbf{h})$

```

for i = 1 to l do
   $p(x_i = 1 | \mathbf{h}) \leftarrow \text{sigm}(b_i + \sum_j W_{ji} h_j)$ 
   $x_i \leftarrow$  sample from Bernoulli distribution with parameter  $p(x_i = 1 | \mathbf{h})$ 
end for

```

## Pseudocode Contrastive Divergence training of the RBF

Training in an RBF is based on stochastic gradient descent. When cycling over training examples, the parameters of the RBF are updated for a given example  $\mathbf{x}_t$  according to the following algorithm:

**Algorithm:** TRAIN-RBF-CD( $\mathbf{x}_t, \eta, k$ )

**Input:** training example  $\mathbf{x}_t$ , learning rate  $\eta$ , number of Gibbs steps  $k$

```

# Positive phase
 $\mathbf{x}^{\text{pos}} \leftarrow \mathbf{x}_t$ 
# Set positive hidden statistics to the vector of probabilities  $p(h_i = 1 | \mathbf{x}^{\text{pos}})$ 
 $\hat{\mathbf{h}}^{\text{pos}} \leftarrow \text{INFERENCE-RBF}(\mathbf{x}^{\text{pos}})$ 

# Negative phase
 $\mathbf{x}^{\text{neg}} \leftarrow \mathbf{x}^{\text{pos}}$ 
while  $k > 0$  do
     $\mathbf{h}^{\text{neg}} \leftarrow \text{SAMPLE-HIDDEN-RBF}(\mathbf{x}^{\text{neg}})$ 
     $\mathbf{x}^{\text{neg}} \leftarrow \text{SAMPLE-VISIBLE-RBF}(\mathbf{h}^{\text{neg}})$ 
     $k \leftarrow k - 1$ 
end while
# Set negative hidden statistics to the vector of probabilities  $p(h_i = 1 | \mathbf{x}^{\text{neg}})$ 
 $\hat{\mathbf{h}}^{\text{neg}} \leftarrow \text{INFERENCE-RBF}(\mathbf{x}^{\text{neg}})$ 

# Update parameters
 $\mathbf{b} \leftarrow \mathbf{b} + \eta (\mathbf{x}^{\text{pos}} - \mathbf{x}^{\text{neg}})$ 
 $\mathbf{c} \leftarrow \mathbf{c} + \eta (\hat{\mathbf{h}}^{\text{pos}} - \hat{\mathbf{h}}^{\text{neg}})$ 
 $\mathbf{W} \leftarrow \mathbf{W} + \eta (\hat{\mathbf{h}}^{\text{pos}} \mathbf{x}^{\text{pos}T} - \hat{\mathbf{h}}^{\text{neg}} \mathbf{x}^{\text{neg}T})$ 

```

## Computing $p(\mathbf{x})$

To compute  $p(\mathbf{x})$ , we first need to compute the normalization constant  $Z$ , with the following procedure:

**Algorithm:** COMPUTE-Z-RBF()

**Output:** normalization constant  $Z$  of the RBF

```

# Initialize  $Z$ 
 $Z \leftarrow 0$ 
# Sum over all possible values of  $\mathbf{h}$ , compatible with the tree constraints
for  $\mathbf{h}$  such that  $\mathbf{h}_{N(1)} \in S(1), \dots, \mathbf{h}_{N(T)} \in S(T)$  do

```

$$Z \leftarrow Z + \prod_{j=1}^d \left( 1 + e^{\mathbf{w}_{:,j}^T \mathbf{h} + b_j} \right)$$

**end for**

With a procedure to compute  $Z$ , we can then compute  $p(\mathbf{x})$  as follows:

**Algorithm:** COMPUTE-PROB-RBF( $\mathbf{x}$ )

**Input:** input  $\mathbf{x}$

**Output:** probability  $p(\mathbf{x})$  under the RBF

# Get  $Z$

$Z \leftarrow$  COMPUTE-Z-RBF()

# Get sum of exponentiated energies

$L(\cdot, \cdot), R(\cdot, \cdot) \leftarrow$  SUM-EXP-ENERGIES-RBF( $\mathbf{x}$ )

# Get numerator of  $p(\mathbf{x})$

$\text{num} \leftarrow \prod_{t=1}^T (L(\text{root}(t), N(t)) + R(\text{root}(t), N(t)))$

$p(\mathbf{x}) \leftarrow \text{num}/Z$

## References

- Aitchison, J., & Aitken, C. (1976). Multivariate binary discrimination by the kernel method. *Biometrika*, *63*, 413–420.
- Carreira-Perpiñán, M. A., & Renals, S. A. (2000). Practical identifiability of finite mixtures of multivariate bernoulli distributions. *Neural Computation*, *12*, 141–152.
- Everitt, B. S., & Hand, D. J. (1981). *Finite mixture distributions*. Monographs on Statistics and Applied Probability. London: Chapman and Hall.
- Hinton, G. E. (2000). *Training products of experts by minimizing contrastive divergence* (Technical Report GCNU TR 2000-004). Gatsby Unit, University College London.
- Hinton, G. E., Osindero, S., & Teh, Y. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, *18*, 1527–1554.

- Juan, A., & Vidal, E. (2001). On the use of bernoulli mixture models for text classification. *PRIS '01: Proceedings of the 1st International Workshop on Pattern Recognition in Information Systems* (pp. 118–126). ICEIS Press.
- Juan, A., & Vidal, E. (2004). Bernoulli mixture models for binary images. *ICPR '04: Proceedings of the Pattern Recognition, 17th International Conference on (ICPR'04) Volume 3* (pp. 367–370). Washington, DC, USA: IEEE Computer Society.
- Kassel, R. (1995). *A comparison of approaches to on-line handwritten character recognition*. Doctoral dissertation, MIT Spoken Language Systems Group.
- Lowd, D., & Domingos, P. (2005). Naive bayes models for probability estimation. *Proceedings of the Twenty-second International Conference on Machine Learning (ICML'05)* (pp. 529–536). New York, NY, USA: ACM.
- Nair, V., & Hinton, G. E. (2009). Implicit mixtures of restricted boltzmann machines. In D. Koller, D. Schuurmans, Y. Bengio and L. Bottou (Eds.), *Advances in neural information processing systems 21 (nips'08)*, 1145–1152.
- Neal, R. M. (1992). Connectionist learning of belief networks. *Artificial Intelligence*, 56, 71–113.
- Ng, A. Y., & Jordan, M. I. (2002). On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. *Advances in Neural Information Processing Systems 14 (NIPS'01)* (pp. 841–848).
- Salakhutdinov, R., & Murray, I. (2008). On the quantitative analysis of deep belief networks. *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML'08)* (pp. 872–879). ACM.
- Smolensky, P. (1986). Information processing in dynamical systems: Foundations of harmony theory. In D. E. Rumelhart and J. L. McClelland (Eds.), *Parallel distributed processing*, vol. 1, chapter 6, 194–281. Cambridge: MIT Press.
- Tieleman, T. (2008). Training restricted boltzmann machines using approximations to the likelihood gradient. *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML'08)* (pp. 1064–1071). Helsinki, Finland: ACM.

Tieleman, T., & Hinton, G. (2009). Using fast weights to improve persistent contrastive divergence. *Proceedings of the Twenty-sixth International Conference on Machine Learning (ICML'09)* (pp. 1033–1040). New York, NY, USA: ACM.