Open access • Journal Article • DOI:10.1162/NECO_A_00014

Tractable multivariate binary density estimation and the restricted boltzmann forest — Source link

Hugo Larochelle, Yoshua Bengio, Joseph Turian

Institutions: University of Toronto, Université de Montréal

Published on: 01 Sep 2010 - Neural Computation (MIT Press)

Topics: Boltzmann machine, Restricted Boltzmann machine, Binary data, Density estimation and Binary tree

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. We argue that, even in its tractable regime, 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 (RBForest), 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

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

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:

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

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:

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

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 (RBForest). In the RBForest, 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

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

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

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

where the normalization constant Z ensures that Equation 5 defines a valid distribution. It is straightforward to show that

$$p(\mathbf{x}|\mathbf{h}) = \prod_i p(x_i|\mathbf{h}) = \prod_i \mathrm{sigm}(b_i + \sum_j W_{ji}h_j) \tag{6}$$

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

$$p(\mathbf{h}|\mathbf{x}) = \prod_j p(h_j|\mathbf{x}) = \prod_j \mathrm{sigm}(c_j + \sum_i W_{ji}x_i). \tag{7}$$

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. However, the expression for the likelihood of an input $\mathbf{x}$ is in general computationally expensive:

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

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

$$\begin{aligned}
\sum_{\mathbf{h}\in\{0,1\}^l} e^{-E(\mathbf{x},\mathbf{h})} &= \sum_{h_1\in\{0,1\}} \cdots \sum_{h_l\in\{0,1\}} e^{\mathbf{b}^T\mathbf{x}+\mathbf{h}^T\mathbf{W}\mathbf{x}+\mathbf{c}^T\mathbf{h}} \\
&= e^{\mathbf{b}^T\mathbf{x}} \left(\sum_{h_1\in\{0,1\}} e^{h_1\mathbf{W}_{1,:}\mathbf{x}+c_1h_1}\right) \cdots \left(\sum_{h_l\in\{0,1\}} e^{h_l\mathbf{W}_{l,:}\mathbf{x}+c_lh_l}\right) \\
&= e^{\mathbf{b}^T\mathbf{x}} \prod_{i=1}^{l} \left(1+e^{\mathbf{W}_{i,:}\mathbf{x}+c_i}\right)
\end{aligned} \tag{9}$$

where $\mathbf{W}_{i,:}$ is the i^{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:

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

where $\mathbf{W}_{:,i}$ is the i^{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 Restricted Boltzmann Forests (RBForests)

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. In particular, we propose to introduce tree constraints in the RBM. These constraints will be 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.

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 define a path in the tree from the root to one of the leaves, which we will refer to as the

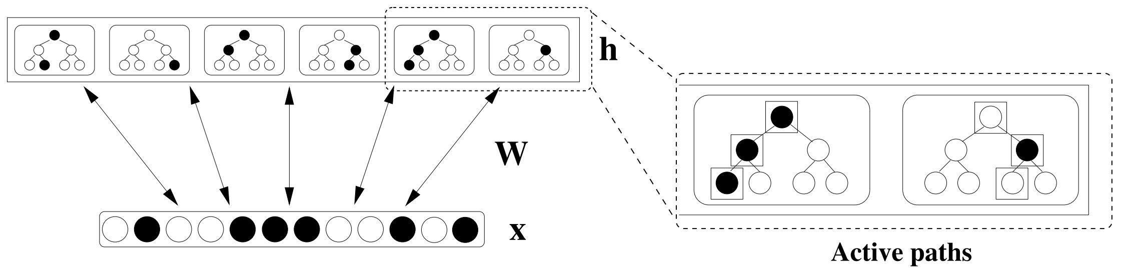


Figure 1: Illustration of a Restricted Boltzmann Forest for binary input vectors, where binary variables in the hidden layer are grouped in trees. When a hidden variable is inactive (white) or active (black), all hidden variables will accordingly be inactive in its left or right subtree. On the right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

active path[1].

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 (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).

3.1 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 constraints.

In an RBForest, the conditional distribution $p(\mathbf{x}|\mathbf{h})$ is unchanged and is as given

[1]Notice that for a path to be active, all hidden variables in the path need not be active.

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 and inactive as $L\left(i, N(t)\right)$ and $R\left(i, N(t)\right)$ respectively:

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

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

where $E(\mathbf{x}, \mathbf{h}_{N(t)})$ is the energy associated to the t^{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:

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

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

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

when $\mathbf{h}_{N(t)}$ respects the tree constraints (otherwise $p(\mathbf{h}_{N(t)}|\mathbf{x}) = 0$). Here, $\mathrm{path}(\mathbf{h}_{N(t)})$ is the set of hidden variables in the active path of the t^{th} tree with value $\mathbf{h}_{N(t)}$. To sample from $p(\mathbf{h}_{N(t)}|\mathbf{x})$, we first sample the root variable $h_{\mathrm{root}(t)}$ from $p(h_{\mathrm{root}(t)}|\mathbf{x})$ ($A(\mathrm{root}(t))$ 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 $p(h_i = 1|\mathbf{x})$ is also simple to compute:

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

which can be written in a recursive form as

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

Let's now consider the computations required by the terms $L\left(i, N(t)\right)$ and $R\left(i, N(t)\right)$ 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:

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

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

and for a leaf h_i we have $L\left(i, N(t)\right) = e^{\mathbf{W}_{i,:}\mathbf{x}}$ and $R\left(i, N(t)\right) = 1$. We can hence obtain all $L\left(i, N(t)\right)$ and $R\left(i, N(t)\right)$ 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

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

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

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

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 RBForest are given in the Appendix.

3.2 Learning in the RBForest

To train an RBForest, 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 θ

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

It 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 some input 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 RBForest 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 RBForests should also benefit.

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

Remains the question of how to compute $p(\mathbf{x})$ in an RBForest. 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 Equations 11 and 12 and the linearity of $E(\mathbf{x}, \mathbf{h})$ in $\mathbf{h}$, we have:

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

where each of the $L(\mathrm{root}(t), N(t))$ and $R(\mathrm{root}(t), N(t))$ 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 $\mathbf{h}|\mathbf{h}_{N(t)} \in S(t)\ \forall t \in \{1, \ldots, T\}$. 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 RBForest 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 RBForest will have 75 hidden variables, 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 RBForest generalization are competitive when compared to other tractable approaches.

5 Experiment: density estimation

We present here an experimental comparison of the RBM and the RBForest with a standard mixture of Bernoullis (MoB). Just like RBMs and RBForest, the "emission" probability distribution given the hidden state is also a product of independent Bernoullis. Hence, the only difference between the MoB, RBM and RBForest 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 RBForest, 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[2], with the exception of the

[2]See http://www.csie.ntu.edu.tw/∼cjlin/libsvmtools/datasets/.

ocr-letter dataset[3] and the *nips-0-12* dataset[4]. 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 η (in $\{0.005, 0.0005, 0.00005\}$), the number of CD steps[5] (in $\{10, 25\}$) and the number of iterations over the training set (in $\{100, 500, 1000\}$)[6]. The RBM had 23 hidden variables, and we considered RBForests 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[7]. 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.

The results are displayed in Table 1. The first surprising observation is that, even with only 23 hidden units, the RBM 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.

[3]See http://ai.stanford.edu/∼btaskar/ocr/.

[4]See http://www.cs.toronto.edu/∼roweis/data.html.

[5]For the *mushrooms* and *nips-0-12* datasets, which are smaller, we also considered 50 steps of CD

[6]No weight decay was used, since choosing an appropriate number of iterations seemed sufficient to avoid overfitting.

[7]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 RBM and RBForest (i.e. the RBM and/or RBForest outperforms the MoB if this difference is positive). Confidence interval estimates based on the test average NLL statistics of the different models are given.

Dataset	Number of inputs	Set sizes Train	Valid	Test	Test NLL diff. (RBM)	Test NLL diff. (RBForest)
adult	123	5000	1414	26147	**4.18 $\pm$ 0.11**	**4.12 $\pm$ 0.11**
connect-4	126	16000	4000	47557	**0.75 $\pm$ 0.04**	**0.59 $\pm$ 0.04**
dna	180	1400	600	1186	**1.29 $\pm$ 0.72**	**1.39 $\pm$ 0.73**
mushrooms	112	2000	500	5624	*-0.69 $\pm$ 0.13*	0.042 $\pm$ 0.12
nips-0-12	500	400	100	1240	**12.65 $\pm$ 1.55**	**12.61 $\pm$ 1.55**
ocr-letter	128	32152	10000	10000	*-2.49 $\pm$ 0.44*	**3.78 $\pm$ 0.43**
rcv1	150	40000	10000	150000	*-1.29 $\pm$ 0.16*	**0.56 $\pm$ 0.16**
web	300	14000	3188	32561	**0.78 $\pm$ 0.30**	-0.15 $\pm$ 0.31

5.1 Comparison with groups of multinomial hidden variables

To make the computation of Z tractable, instead of tree-structured groups of hidden variables, one could have used multinomial groups. In a multinomial group, the binary hidden variables are mutually exclusive, i.e. only one in each group can be active. It is possible to show that for any RBForest, 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 RBForest with only one tree and with parameters $\mathbf{W}$, $\mathbf{b}$ and $\mathbf{c}$. Then, consider a multinomial group version of that RBForest (with parameters $\mathbf{W}^*$, $\mathbf{b}^*$ and $\mathbf{c}^*$) by associating each bit h_i^* of a multinomial variable with a path $\mathrm{path}(\mathbf{h})$ in the RBForest 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 RBForest 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, there is a crucial difference between both models, which we hypothesize as being linked to differences in the difficulty of optimizing their non-convex training criteria. We observed that optimization is much easier with the RBForest than with multinomial units. In the case of the RBForest, we observe in practice that training progresses smoothly by first finding good values for the root node's weights and then slowly moving the optimization of 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 RBForest 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[8]. As in Table 1, we also ran experiments with models using multinomial groups of equivalent capacity as the trained RBForests. On half of the datasets, the RBForest was found 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.

5.2 Training a mixture of RBForests

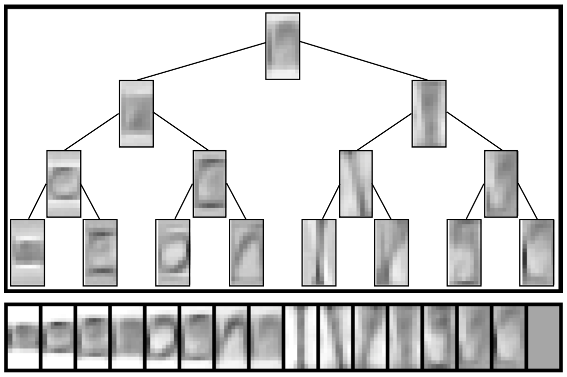
We mentioned previously that one advantage of having a tractable density estimator is that it can be used in a larger probabilistic model. For instance, consider a mixture of M RBForest components

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

where $p(\mathbf{x}|C = m)$ is given by the distribution of an RBForest with parameters $\mathbf{W}^m$, $\mathbf{b}^m$ and $\mathbf{c}^m$ (the different RBForest components have different parameters). Such a mix-

[8]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.

Dataset	Test NLL difference	Size of tree
adult	-0.03 ± 0.08	3
connect-4	**2.31 ± 0.03**	3
dna	-0.06 ± 0.64	7
mushrooms	**0.73 ± 0.09**	3
nips-0-12	1.35 ± 1.51	3
ocr-letter	**2.78 ± 0.40**	63
rcv1	**0.60 ± 0.16**	63
web	-0.17 ± 0.29	7


(a)

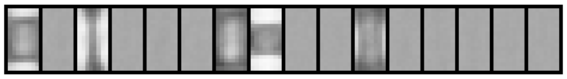

(b)

Figure 2: **(Left)** Comparison between using trees vs multinomial units. The NLL difference is the NLL obtained using multinomial units minus the NLL obtained using trees. The table also shows the size of the tree (number of nodes) with best validation average NLL. **(Right)** Illustration of training optimization a tree of hidden variables and a multinomial group. **(a)** On top, illustration for a single-tree RBForest 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.

ture can be trained using the Generalized EM algorithm. During the E step, we simply compute the posterior probabilities (or responsibilities) of each RBForest components having generated each training example

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

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

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:

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

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 RBForest components, optimizing Equation 22 is equivalent to training each RBForest components on a weighted version of the training set. The weight of example $\mathbf{x}_t$ for the m^{th} RBForest 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 $m^t h$ RBForest's parameters given example $\mathbf{x}_t$.

We trained a mixture of 8 RBForest using the Generalized EM algorithm, on the *ocr-letter* dataset. The parameters of each RBForest 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 RBForest 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 RBForest (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 RBForests, and the resulting density estimator would also be tractable, as long as each individual RBForest 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 RBForest is used need not correspond to an undirected graphical model (like the RBForest).

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 RBForest 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 RBForest 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 $\mathbf{x}_t$:

$$\widehat{\mathbf{h}}(\mathbf{x}_t) = \underset{\mathbf{h}|\mathbf{h}_{N(t)} \in S(t)\ \forall t \in \{1,\ldots,T\}}{\arg\max} p(\mathbf{h}|\mathbf{x}_t) = \underset{\mathbf{h}|\mathbf{h}_{N(t)} \in S(t)\ \forall t \in \{1,\ldots,T\}}{\arg\max} \prod_{t=1}^{T} p(\mathbf{h}_{N(t)}|\mathbf{x}_t)$$

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,

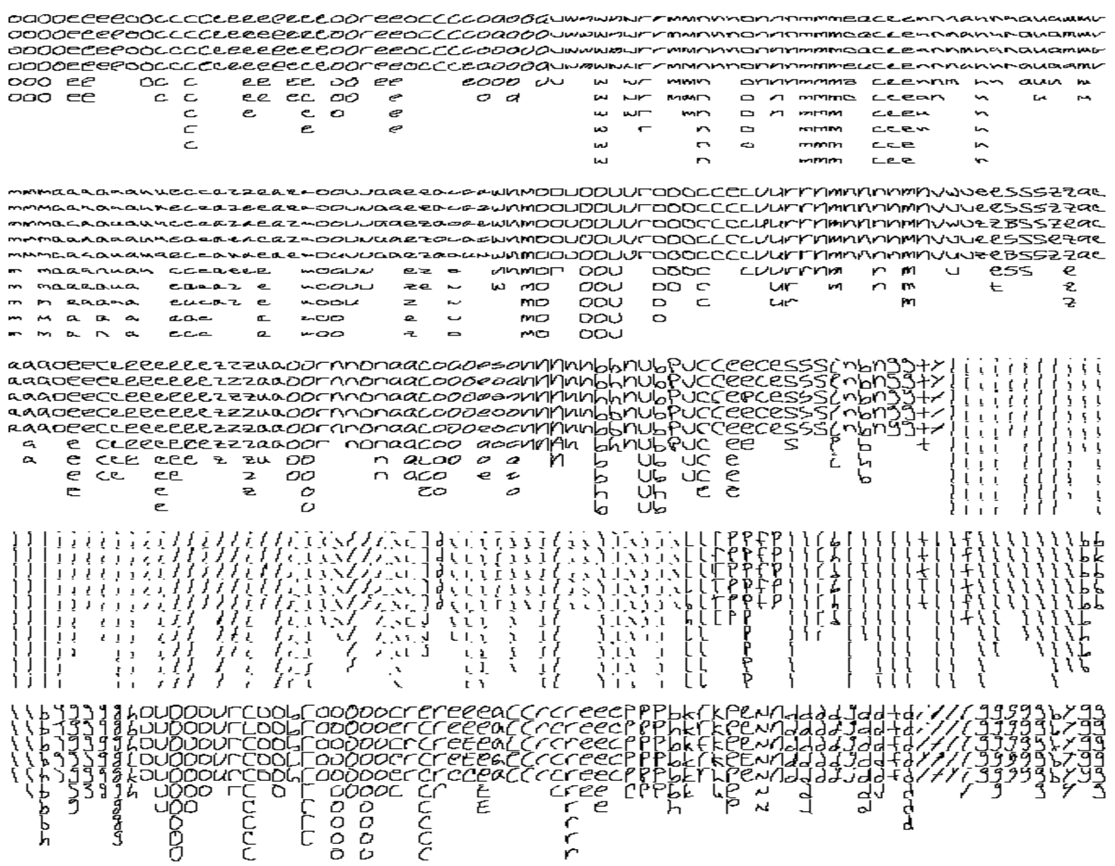


Figure 3: Visualization of 375 of the implicit mixture components (one per column) learned by the RBForest on the *ocr-letter* dataset.

finding the configuration with maximum probability can be done with an exhaustive search. Having found $\widehat{\mathbf{h}}(\mathbf{x}_t)$ for each test example $\mathbf{x}_t$, we then grouped together test examples sharing the same value for $\widehat{\mathbf{h}}(\mathbf{x}_t)$.

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 units the RBForest has. Figure 3 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 advantage of using trees was emphasized using a comparison with groups of multinomial units.

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
      $j \leftarrow$ index of left children of $h_i$
      $k \leftarrow$ index of right children of $h_i$
      $L(i, N(t)) \leftarrow e^{W_{i,:}\mathbf{x}} \left(L\left(j, N(t)\right) + R\left(j, N(t)\right)\right)$
      $R(i, N(t)) \leftarrow L\left(k, N(t)\right) + R\left(k, N(t)\right)$
```

```
        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
$L(\cdot,\cdot)$, $R(\cdot,\cdot)$ ←SUM-EXP-ENERGIES-RBF($\mathbf{x}$)
for $t = 1$ to $T$ do
    # Initialize top-down pass
    $p(h_{\text{root}(t)} = 1|\mathbf{x}) \leftarrow \frac{L(\mathbf{root}(t),N(t))}{L(\mathbf{root}(t),N(t))+R(\mathbf{root}(t),N(t))}$
    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
$L(\cdot,\cdot)$, $R(\cdot,\cdot)$ ←SUM-EXP-ENERGIES-RBF($\mathbf{x}$)
for $t = 1$ to $T$ do
  # Get root probability
  $p(h_{\mathrm{root}(t)} = 1|\mathbf{x}) \leftarrow \frac{L(\mathrm{root}(t),N(t))}{L(\mathrm{root}(t),N(t))+R(\mathrm{root}(t),N(t))}$
  $h_{\mathrm{root}(t)} \leftarrow$ sample from Bernoulli distribution with parameter $p(h_{\mathrm{root}(t)} = 1|\mathbf{x})$
  $i \leftarrow \mathrm{root}(t)$
  for $\delta = 1$ to D do
    if $h_i$ = 1 then
      $i \leftarrow$ index of left child of $h_i$
    else
      $i \leftarrow$ 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($\mathbf{h}$)
Input: value of the hidden units $\mathbf{h}$
Output: a sample $\mathbf{x}$ from $p(\mathbf{x}|\mathbf{h})$
```

```
**for** $i = 1$ to $l$ **do**
    $p(x_i = 1|\mathbf{h}) \leftarrow \mathrm{sigm}(b_i + \sum_j W_{ji} h_j)$
    $x_i \leftarrow$ 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 $p(h_i = 1|\mathbf{x}^{\text{pos}})$
$\widehat{h}^{\text{pos}} \leftarrow$ INFERENCE-RBF($\mathbf{x}^{\text{pos}}$)

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

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

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 $\mathbf{h}_{N(1)} \in S(1), \ldots, \mathbf{h}_{N(T)} \in S(T)$ 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 $p(\mathbf{x})$
$\mathrm{num} \leftarrow \prod_{t=1}^{T} \left(L(\mathrm{root}(t), N(t) + R(\mathrm{root}(t), N(t))\right)$
$p(\mathbf{x}) \leftarrow \mathrm{num}/Z$
```

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.