Self Supervised Boosting

Max Welling, Richard S. Zemel and Geoffrey E. Hinton
Department of Computer Science
University of Toronto
10 King's College Road
Toronto, M5S 3G5 Canada

Abstract

Boosting algorithms and successful applications thereof abound for classification and regression learning problems, but not for unsupervised learning. We propose a sequential approach to adding features to a random field model by training them to improve classification performance between the data and an equal-sized sample of "non-data" generated from the model's current estimate of the data density. Training in each boosting round proceeds in three stages: first we sample non-data from the model's current Boltzmann distribution. Next, a feature is trained to improve classification performance between data and non-data. Finally, a coefficient is learned which determines the importance of this feature relative to ones already in the pool. Non-data only needs to be generated once to learn each new feature. The validity of the approach is demonstrated on binary digits and continuous synthetic data.

1 Introduction

While researchers have developed and successfully applied a myriad of boosting algorithms for classification and regression problems, boosting for density estimation has received relatively scant attention. Yet incremental, stage-wise fitting is an attractive model for density estimation. One can imagine that the initial *features*, or *weak learners*, could model the rough outlines of the data density, and more detailed carving of the density landscape could occur on each successive round. Ideally, the algorithm would achieve automatic model selection, determining the requisite number of weak learners on its own. It has proven difficult to formulate an objective for such a system, under which the weights on examples, and the objective for training a weak learner at each round have a natural gradient-descent interpretation as in standard boosting algorithms [10] [8]. In this paper we propose an algorithm that provides some progress towards this goal.

A key idea in our algorithm is that unsupervised learning can be converted into supervised learning by using the model's imperfect current estimate of the data to generate negative examples. A form of this idea was previously exploited in the *contrastive divergence* algorithm [5]. We take the idea a step further here by training a weak learner to discriminate between the positive examples from the original data and the negative examples generated by sampling from the current density estimate. This new weak learner minimizes a simple additive logistic loss function [3].

Our algorithm obtains an important advantage over sampling-based, unsupervised methods that learn features in parallel. Parallel-update methods require a new sample after each iteration of parameter changes, in order to reflect the current model's estimate of the data density. We improve on this by using one sample per boosting round, to fit one weak learner. The justification for this approach comes from the proposal that, for stagewise additive models, boosting can be considered as gradient-descent in function space, so the new learner can simply optimize its inner product with the gradient of the objective in function space [4].

Our algorithm incrementally constructs random fields from examples. As such, it bears some relation to maximum entropy models, which are popular in natural language processing [9]. In these applications, the features are typically not learned; instead the algorithms greedily select at each round the most informative feature from a large set of pre-enumerated features.

2 Additive Random Field Models and Products of Experts

The general form of the energy of an additive random field model is given by

$$E(\mathbf{s}; \alpha, \theta) = \sum_t \alpha_t \phi_t(\mathbf{s}, \theta_t) \tag{1}$$

Although similar in form, the main difference with a class of models known as "maximum entropy models" is the adaptive nature of the features $\phi(\cdot)$ through parameters θ. This same additive energy function with adaptive features underlies a "product of experts", for which Hinton developed a relatively efficient gradient descent algorithm to fit features in parallel [5]. This algorithm minimizes the contrastive divergence between the data and the model of the data based on the current energy function.

The main drawback of gradient descent algorithms for these models is that they need a new sample from the Boltzmann distribution after every change in the parameters α or θ,

$$P(\mathbf{s}) = \frac{1}{Z} \exp(-E(\mathbf{s}))) \qquad Z = \sum_{\mathbf{s}} \exp(-E(\mathbf{s})) \tag{2}$$

To see this we take the (functional) derivative of the log-likelihood w.r.t. the energy,

$$\frac{\partial L}{\partial E(\mathbf{s})} = -\left[\frac{1}{M} \sum_{m=1}^{M} \delta(\mathbf{s} - \mathbf{d}_m)\right] + P(\mathbf{s}) \tag{3}$$

where $\mathbf{d}_m$ denotes a data-vector. This equation makes explicit the main philosophy behind learning in random fields: *the energy of the states occupied by data is lowered (weighted by $\frac{1}{M}$) while the energy of* **all** *states is raised (weighted by $P(\mathbf{s})$).* Since there are usually an exponential number of states in the system, the second term is often approximated by a sample from $P(\mathbf{s})$. This is a very expensive operation, and it must occur each time we compute gradients.

Iterative scaling methods have been developed for models that do not include adaptive feature parameters θ but instead train only the coefficients α [9]. These methods make more efficient use of non-data than gradient descent, but they only minimize a loose bound on the cost function and their terminal convergence can be slow.

3 Self Supervised Boosting

In this section we propose an algorithm to add features sequentially, using just one sample of non-data per feature. The procedure was inspired by the interpretation of boosting as

functional gradient descent [8]. At each round of boosting a new feature is added to the energy landscape in three stages. In the first stage we sample non-data from the Boltzmann distribution (Equation 2). In the second stage we search for a feature that correlates best with the derivative of log-likelihood w.r.t. the energy function, and in the last stage we find the coefficient α that determines how much of that feature should be added to the pool.

Before we describe this procedure in more detail we first generalize the derivative (Equation 3) to incorporate a weighting function $w(\cdot)$ on the data and non-data,

$$\frac{\partial L_w}{\partial E(\mathbf{s})} = -\left[\frac{1}{M}\sum_{m=1}^{M} w(E(\mathbf{s}))\ \delta(\mathbf{s}-\mathbf{d}_m)\right] + w(-E(\mathbf{s}))\ P(\mathbf{s}) \tag{4}$$

In this way one can encourage the algorithm to focus more on data-vectors with large energy and non-data with low energy. The extra weights will incur a certain bias w.r.t. the maximum likelihood solution. However, it is expected that the extra effort on "hard cases" will cause the algorithm to converge faster to good density models.

In what follows, we will not need an explicit expression of the "weighted log-likelihood" L_w, but only its derivatives, given by Equation 4.

3.1 Stage One: Sampling the Non-Data

To represent the second term in the derivative (Equation 4) involving the Boltzmann distribution, we draw a representative sample. Sampling from a Boltzmann distribution can be done in a variety of ways. A quite general method for sampling from continuous models is Hybrid Monte Carlo, while a widely applicable method for discrete models is Gibbs sampling. But although these methods are very flexible, they are typically not very fast. One can also reverse the logic and start from a fast sampling method and subsequently derive the form of the weak learner from it. An approximate strategy is to sample from brief Markov chains that have not yet equilibrated and were initiated at the data-vectors. This approach is inspired by some successful experiments with contrastive divergence learning [7]. A certain bias is incurred but as our experiments will show, the method seems very robust against sampling from an approximate distribution.

3.2 Stage Two: Fitting a New Feature

After the sampling stage we have a set of data and a set of current non-data available. We now fit a new feature $\phi(\mathbf{s})$ by maximizing the inner product between the new feature and the functional derivative of the log-likelihood. This provides us with an uphill search direction on the log likelihood:

$$\langle \frac{\partial L_w(E_{1:t-1})}{\partial E(\mathbf{s})}, \phi_t(\mathbf{s}) \rangle = \sum_{\mathbf{s}} \frac{\partial L_w(E_{1:t-1})}{\partial E(\mathbf{s})} \phi_t(\mathbf{s}) \approx -\sum_{n=1}^{N} y_n w_n \phi_{t,n} \tag{5}$$

where the sum runs over data-vectors and non-data vectors, $y_n = 1$ for data and $y_n = -1$ for non-data, $w_n = w(y_n E(\mathbf{s}_n))$ and $\phi_{t,n} = \phi_t(\mathbf{s}_n)$. The energy $E_{1:t-1}$ is of the form of Equation 1. The new feature should thus maximally anti-correlate with the labels of the data and maximally correlate with the labels of the non-data. Adding it to the energy will lower the energy at locations where there are data and raise the energy at locations where there are non-data.

One could view this as fitting a new weak learner such that the classification of the data versus non-data is improved. In fact, a sensible choice for the weights:

$$w_n = \sigma(y_n E_n) \tag{6}$$

can be derived by observing that the derivative (Equation 4), with its second term represented by a fixed sample, is identical to the negative derivative of the "logistic loss" function, $Loss = \sum_n \log(1 + \exp(y_n E_n))$ w.r.t. the energy. Consequently, we can define $p(y = 1|\mathbf{s}) = \sigma(-E(\mathbf{s}))$ as the probability that a state $\mathbf{s}$ is occupied by a data-vector where $-E$ may be interpreted as the margin. Therefore, similar to "logit-boost" [4, 1], the new feature which is added to the energy will decrease the logistic loss function, where the novel aspect of our approach is that the negative examples are self-generated and change each round. The weights w_n are bounded between 0 and 1, and favor "badly classified" data and non-data. In the following we assume they are normalized to sum to 1.

We also incorporate an additive scalar a in the energy that is updated before fitting each feature. This term has an important effect on the weights: If we require that the log-likelihood is stationary w.r.t. a we find,

$$\frac{\partial L_w}{\partial a} = \sum_{\mathbf{s}} \frac{\partial L_w(E)}{\partial E(\mathbf{s})} \frac{\partial E(\mathbf{s})}{\partial a} \approx -\sum_n y_n w_n = 0 \tag{7}$$

Therefore updating a to maximize L_w[1] balances the total weight assigned to data and non-data. It is also important to bound the norm of the features, as otherwise, we can find an arbitrarily large inner product between a feature and the log-likelihood by scaling up its length.

3.3 Stage Three: Finding the Coefficient α_t

Once we have the new feature we have to decide what its importance is w.r.t. the other features, i.e. we need to find the coefficient α_t (also known as "maximum entropy weight"). One possibility is to apply the "improved iterative scaling" algorithm (ISS) proposed in [9]. Below, we will discuss a simple alternative.

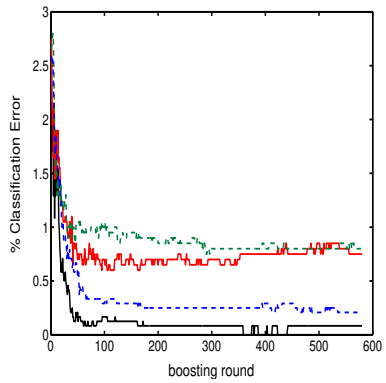
In the functional gradient interpretation we are faced with a line-search in the direction of the derivative approximated by the new feature. The derivative of L_w w.r.t. α_t is given by,

$$\frac{\partial L_w}{\partial \alpha_t} = \sum_{\mathbf{s}} \frac{\partial L_w(E)}{\partial E(\mathbf{s})} \frac{\partial E(\mathbf{s})}{\partial \alpha_t} \approx -\sum_n y_n w_n \phi_{t,n} \tag{8}$$

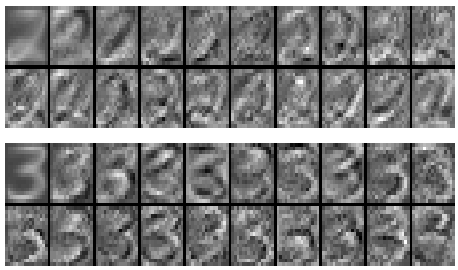
As mentioned in the previous section we always ensure that the weighted log-likelihood is maximized over a as well. This implies that we maximize over α_t and a simultaneously.

When iteratively updating α_t we not only change the weights w_n but also the Boltzmann distribution, which makes the non-data examples no longer representative of the current estimated data density. To correct for this we include importance weights g_n on the examples that are all $1/N$ at $\alpha_t = 0$. It is very easy to update these weights from iteration to iteration using $g_n \to g_n \exp(-\delta\alpha_t\ \phi_{t,n})$ and renormalizing. It is well known that in high dimensions the effective sample size of the weighted sample can rapidly become too small to be useful. We therefore monitor this effective sample size, given by $1/\sum_n g_n^2$, where the sum runs over the non-data only. If this effective sample size drops below a threshold we have two choices. We can obtain a new set of non-data from the updated Boltzmann distribution, reset the importance weights to $1/N$ and resume fitting α_t. Alternatively, we simply accept the current value of α_t and proceed to the next round of boosting. Because we initialize $\alpha_t = 0$ in the fitting procedure, the latter approach underestimates the importance of this particular feature, which is not a problem since a similar feature can be added in the next round.

[1] Since $P(\mathbf{s})$ is independent of a, it is easy to compute the second derivative $L''_w = \sum_n y_n w'_n$ and we can do Newton updates to compute the stationary point.



(a)



(b)

Figure 1: (a). Training error (lower 2 curves) and test error (higher 2 curves) for the weighted boosting algorithm (solid curves) and the un-weighted algorithm (dashed curves). (b). Features $\mathbf{v}_t$ found by stage 2 of the learning algorithm.

We also observed that the value of the α_t decreased as the model became more accurate over time. This behavior was encouraged by training α_t and $\phi_t(\cdot)$ on two separate samples of non-data, thereby reducing sensitivity to sample noise.

4 A Binary Example: The Generalized RBM

We propose a simple extension of the "restricted Boltzmann machine" (RBM) with (+1,-1)-units [2] as a model for binary data. Each feature is parametrized by weights $\mathbf{v}_t$ and a bias β_t:

$$\alpha_t \phi_t(\mathbf{s}) = -\alpha_t \log\cosh(\mathbf{v}_t^T \mathbf{s} + \beta_t) \tag{9}$$

where the RBM is obtained by setting all $\alpha_t = 1$. One can sample from the summed energy model using straightforward Gibbs sampling, where every visible unit is sampled given all the others. Alternatively, we can take advantage of efficiencies due to the RBM's layered architecture: by sampling all hidden units independently given the visible units and vice versa we have access to a fast mixing Markov chain. Unfortunately, by including the coefficients α_t this trick is no longer valid. But an approximate Markov chain can be designed

$$\alpha_t \log\cosh(\mathbf{v}_t^T \mathbf{s} + \beta_t) \approx \log\cosh(\alpha_t \mathbf{v}_t \mathbf{s} + \alpha_t \beta_t) \tag{10}$$

This approximate Gibbs sampling thus involves sampling from an RBM with scaled weights and biases,

$$P(h_t = 1|\mathbf{s}) = \sigma(2\alpha_t \mathbf{v}_t^T \mathbf{s} + 2\alpha_t \beta_t) \qquad P(s_i = 1|\mathbf{h}) = \sigma(2 \sum_t \alpha_t v_{it} h_t) \tag{11}$$

When using the above Markov chain, we will not wait until it has reached equilibrium but initialize it at the data-vectors and use it for a fixed number of steps, as is done in contrastive divergence learning.

When we fit a new feature in the second stage we need to make sure its norm is controlled. The appropriate value clearly depends on the number of dimensions in the problem; in the experiment described below we bounded the norm of the vector $[\mathbf{w}_t^T, \beta_t]$ to be no larger than 0.1. The updates in the second stage are thus given by $\mathbf{v}_t \rightarrow \mathbf{v}_t + \delta\mathbf{v}_t$ and $\beta_t \rightarrow \beta_t + \delta\beta_t$ with,

$$\delta\mathbf{v}_t \propto \sum_n w_n y_n \tanh(\mathbf{v}_t^T \mathbf{s}_n + \beta_t)\mathbf{s}_n \qquad \delta\beta_t \propto \sum_n w_n y_n \tanh(\mathbf{v}_t^T \mathbf{s}_n + \beta_t) \tag{12}$$

where the weights w_n are proportional to $\sigma(y_n E_n)$ (Equation 6). The coefficients α_t are fit by the procedure described in Section 3.3. We also recall that at all times the total weight on data and non-data is equal because L_w is stationary w.r.t. a.

To test whether we can learn good models of (fairly) high-dimensional, real-world data, we used the 16×16 real-valued digits from the "br" set on the CEDAR cdrom # 1. Models were learned on the first 600 training data of the binarized "2"s and "3"s, while 500 binarized digits of each class were used for testing. The minimum sample size for the coefficients α_t was set to $0.6M$. We used 2 different sets of non-data, 600 examples each, to fit $\phi_t(\cdot)$ and α_t. After a new feature was added, the total energies of all "2"s and "3"s were computed under both models. The energies of the training data (under both models) were used as two-dimensional features to compute a separation boundary using logistic regression, which was subsequently applied to the test data to compute the total misclassification. In Figure 1a we show the total error on both training data and test data as a function of the number of features in the model. For comparison we also plot the training and test error for the unweighted version of the algorithm ($w_n = 1, \ \forall \ \ n$). The classification error after 100 rounds of boosting for the weighted algorithm is about 0.65%, and only very gradually increases to about 0.75% after 600 rounds of boosting. This is good as compared to logistic regression (2.7%), k-nearest neighbors ($1.0\%, k = 1$ is optimal), while a parallel-trained RBM with $50, 100, 200$ hidden units achieves $0.9\%, 0.7\%, 0.85\%$ respectively. The unweighted learning algorithm converges much slower to a good solution, both on training and test data. In Figure 1b we show every 10^{th} feature $\mathbf{v}_t$ between rounds 1 and 200 for both digits. They may be interpreted as mean digits on the first round of boosting and as deformations of digits on later rounds.

5 A Continuous Example: The Dimples Model

For continuous data we propose a different form of feature, which we term a *dimple* because of its shape in the energy domain. A dimple is a mixture of a narrow Gaussian and a broad Gaussian, with a common mean:

$$\phi_t(\mathbf{s}) = -\log\left[\mathcal{N}(\mathbf{s}; \mu, \sigma_1) + \mathcal{N}(\mathbf{s}; \mu, \sigma_2)\right] \tag{13}$$

where the mixing proportion is constant and equal, and σ_2 is fixed and large. Each round of the algorithm fits μ and σ_1 for a new learner. A nice property of dimples is that they can reduce the entropy of an existing distribution by placing the dimple in a region that already has low energy, but they can also raise the entropy by putting the dimple in a high energy region [6].

Sampling is again simple if all $\alpha_t = 1$, since in that case we can use a Gibbs chain which first picks a narrow or broad Gaussian for every feature given the visible variables and then samples the visible variables from the resulting multivariate Gaussian. For general α the situation is less tractable, but a similar approximation as for the generalized RBM can be made,

$$\alpha \log\left[\mathcal{N}(\mathbf{s}; \mu, \sigma_1) + \mathcal{N}(\mathbf{s}; \mu, \sigma_2)\right] \approx \log\left[\mathcal{N}(\mathbf{s}; \mu, \sigma_1)^\alpha + \mathcal{N}(\mathbf{s}; \mu, \sigma_2)^\alpha\right] \tag{14}$$

This approximation will be accurate when one Gaussian is dominating the other, i.e., when the responsibilities are close to zero and one. This is expected to be the case in high-dimensional applications. In the low-dimensional example discussed below we implemented a simple MCMC chain with isotropic, normal proposal density which was initiated at the data-points and run for a fixed number of steps.

The type of dimple we used in the experiment below can adapt a common mean ($\boldsymbol{\mu}$) and the inverse-variance of the small Gaussian ($\boldsymbol{\tau}_1$) in each dimension separately. The update

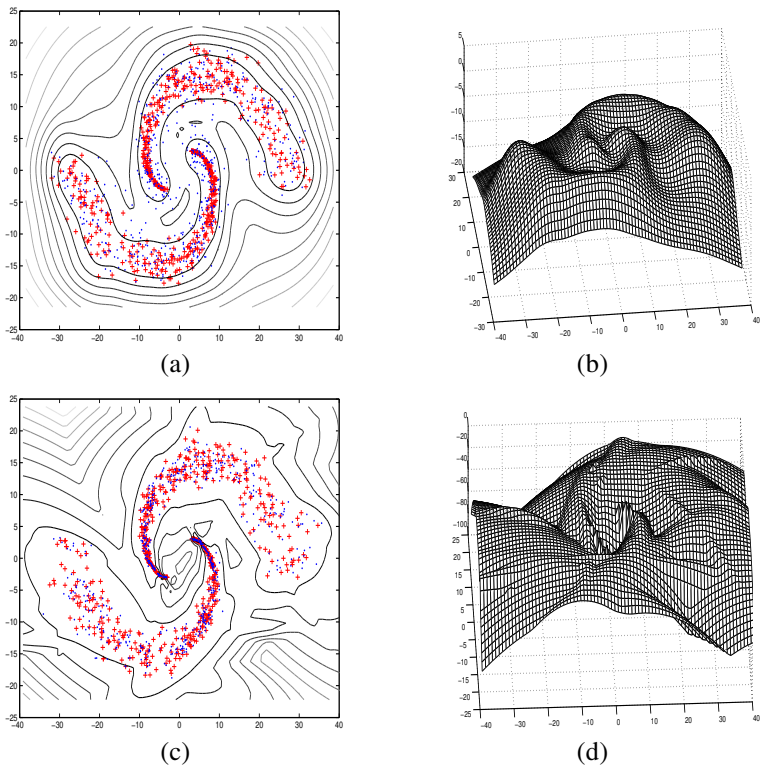


Figure 2: (a). Plot of iso-energy contours after 400 rounds of boosting. The crosses represent the data and the dots the non-data generated from the model. (b). Three dimensional plot of the negative energy surface. (c). Contour plot for a mixture of 30 Gaussians learned using EM. (d). Negative energy surface for the mixture of 30 Gaussians.

rules are given by, $\boldsymbol{\mu} \rightarrow \boldsymbol{\mu} + \delta\boldsymbol{\mu}$ and $\boldsymbol{\tau}_1 \rightarrow \boldsymbol{\tau}_1 + \delta\boldsymbol{\tau}_1$ with

$$\delta\mu_i \propto \sum_n w_n y_n (X_{i,n} - \mu_i)(r_{1,n}\tau_{1,i} + r_{2,n}\tau_{2,i}) \tag{15}$$

$$\delta\tau_{1,i} \propto -\sum_n w_n y_n r_{1,n} \left((X_{i,n} - \mu_i)^2 - 1/\tau_{1,i}\right) \tag{16}$$

where $r_{1,n} = \mathcal{N}_1/(\mathcal{N}_1 + \mathcal{N}_2)$ and $r_{2,n} = 1 - r_{1,n}$ are the responsibilities for the narrow and broad Gaussian respectively and the weights are given by Equation 6. Finally, the combination coefficients $\boldsymbol{\alpha}_t$ are computed as described in Section3.3.

To illustrate the proposed algorithm we fit the dimples model to the two-dimensional data (crosses) shown in figure 2a-c. The data were synthetically generated by defining angles $\varphi_\pm = \pm\pi u$ with u uniform between $[0, 1]$ and a radius $r_\pm = 10 + n\varphi_\pm$ with n standard normal, which were converted to Euclidean coordinates and mirrored and translated to produce the spirals. The first feature is an isotropic Gaussian with the mean and the variance of the data, while later features were dimples trained in the way described above. Figure 2a also shows the contours of equal energy after 400 rounds of boosting together with examples (dots) from the model. A 3-dimensional plot of the negative energy surface is shown in Figure 2b. For comparison, similar plots for a mixture of 30 Gaussians, trained in parallel with EM, are depicted in Figures 2c and 2d.

The main qualitative difference between the fits in Figures 2a-b (product of dimples) and

2c-d (mixture of Gaussians), is that the first seems to produce smoother energy surfaces, only creating structure where there is structure in the data. This can be understood by recalling that the role of the non-data is precisely to remove "dips" in the energy surface where there is no data. The philosophy of avoiding structure in the model that is not dictated by the data is consistent with the ideas behind maximum entropy modelling [11] and is thought to improve generalization.

6 Discussion

This paper discusses a sequential approach to training additive random fields to data. The philosophy is to view unsupervised learning as a sequence of classification problems where the aim is to discriminate between data-vectors and non-data vectors generated from the current model. The sampling step is usually the most time consuming operation, but it is also unavoidable since it informs the algorithm of the states whose energy is too low. The proposed algorithm uses just one sample of non-data to fit a new feature, which is very economical as compared to most non-sequential algorithms which must generate an entire new sample for every gradient update.

There are many interesting issues and variations that we have not addressed in this paper:
-What is the effect of using approximate, e.g. variational distributions for P?
-Can we improve the accuracy of the model by fitting the feature parameters and the coefficients α_t together?
-Does re-sampling the non-data more often during learning improve the final model?
-What is the effect of using different functions to weight the data and how do the weighting schemes interact with the dimensionality of the problem?

References

[1] M. Collins, R.E. Schapire, and Y. Singer. Logistic regression, adaboost and bregman distances. In *Computational Learning Theory*, pages 158–169, 2000.

[2] Y. Freund and D. Haussler. Unsupervised learning of distributions of binary vectors using 2-layer networks. In *Advances in Neural Information Processing Systems*, volume 4, pages 912–919, 1992.

[3] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. Technical report, Dept. of Statistics, Stanford University Technical Report., 1998.

[4] J.H. Friedman. Greedy function approximation: a gradient boosting machine. Technical report, Technical Report, Dept. of Statistics, Stanford University, 1999.

[5] G.E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14:1771–1800, 2002.

[6] G.E. Hinton and A. Brown. Spiking boltzmann machines. In *Neural Information Processing Systems*, volume 12, 2000.

[7] G.E. Hinton, M. Welling, Y.W. Teh, and K. Osindero. A new view of ICA. In *Int. Conf. on Independent Component Analysis and Blind Source Separation*, 2001.

[8] L. Mason, J. Baxter, P. Bartlett, and M. Frean. Boosting algorithms as gradient descent. In *Neural Information Processing Systems*, volume 12, pages 512–518, 2000.

[9] S. Della Pietra, V.J. Della Pietra, and J.D. Lafferty. Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):380–393, 1997.

[10] R.E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. In *Computational Learing Theory*, pages 80–91, 1998.

[11] S.C. Zhu, Z.N. Wu, and D. Mumford. Minimax entropy principle and its application to texture modeling. *Neural Computation*, 9(8):1627–1660, 1997.