A New Learning Algorithm for Stochastic Feedforward Neural Nets

Yichuan Tang TANG@CS.TORONTO.EDU
Ruslan Salakhutdinov RSALAKHU@CS.TORONTO.EDU
Department of Computer Science, University of Toronto. Toronto, Ontario, Canada.

Abstract

Multilayer perceptrons (MLPs) or artificial neural nets are popular models used for nonlinear regression and classification tasks. As regressors, MLPs model the conditional distribution of the predictor variables $\mathbf{y}$ given the input variables $\mathbf{x}$. However, this predictive distribution is assumed to be unimodal (e.g. Normal distribution). For tasks such as structured prediction problems, the conditional distribution should be multi-modal, or one-to-many mappings. By turning the hidden variables in a MLP into stochastic nodes rather than deterministic ones, Sigmoid Belief Nets can induce a rich multimodal distribution in the output space. Learning Sigmoid Belief Nets is very slow and modeling real-valued data is difficult. In this paper, we propose a stochastic feedforward network where its hidden layers have both deterministic and stochastic variables. A new Generalized EM training procedure using importance sampling allows us to efficiently learn complicated conditional distributions. We demonstrate the superiority of our model to conditional Restricted Boltzmann Machines and Mixture Density Networks on 3 synthetic datasets and modeling facial expressions. Moreover, we show that latent features of our model improves classification and provide additional qualitative results on color images.

1. Introduction

Multilayer perceptrons (MLPs) are general purpose function approximators. The output of a MLP can be interpreted as the sufficient statistics (conditioned on the input X) of a member of the exponential family, thereby inducing a distribution over the output space Y. Since the nonlinear activations are all *deterministic*, MLPs model the conditional distribution $p(Y|X)$ with a unimodal assumption (e.g. an isotropic Gaussian).

For many structured prediction problems, we are interested in modeling a conditional distribution $p(Y|X)$ that is multimodal and may have complicated structure[1]. One way to model the multi-modality is to make the hidden variables stochastic. Given the same X, different hidden configurations leads to different output values of Y. A model that does exactly this is the Sigmoid Belief Net (SBN) (Neal, 1992). It is a stochastic feedforward neural network with binary hidden, input, and output variables. SBNs can be viewed as directed graphical models where the sigmoid function is used to compute the degree of "belief" of a child variable given the parent nodes.

Inference in such models is generally intractable. The original paper by Neal (1992) proposed a Gibbs sampler which cycles through the hidden nodes one at a time. However, Gibbs sampling can be very slow when learning large models or fitting moderately-sized datasets. In addition, slow mixing of the Gibbs chain would typically lead to a biased estimation of the gradient during learning. A variational learning algorithm based on the mean-field approximation was proposed in (Saul et al., 1996) to improve the learning of SBNs. A drawback of the variational approach is that, similar to Gibbs, it has to cycle through the hidden nodes one at a time. Moreover, beside the standard mean-field variational parameters, additional parameters must be introduced to lower-bound an intractable term that shows up in the expected free energy, making the lower-bound looser. Gaussian fields are used in (Barber & Sollich, 1999) for inference by making Gaussian approximations to units' input, but there is no longer a lower bound on the likelihood.

International Conference on Machine Learning 2013: Challenges in Representation Learning Workshop. Atlanta, Georgia, USA.

[1] An equivalent problem is learning one-to-many functions from $X \mapsto Y$.

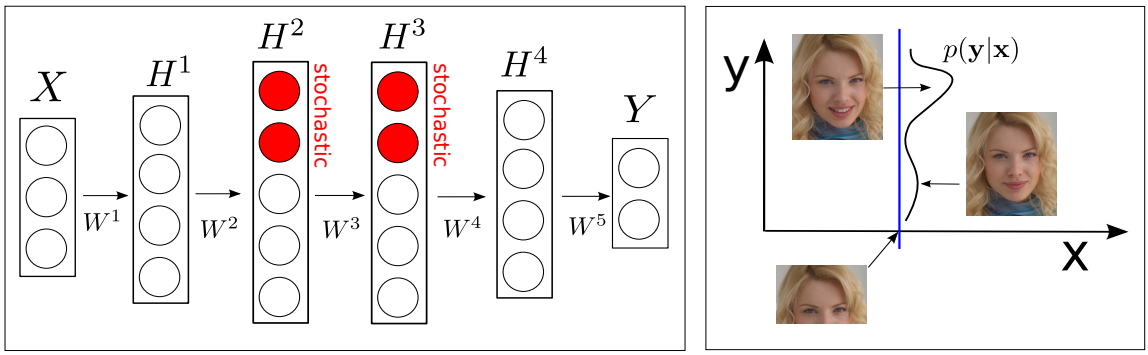


Figure 1. Stochastic Feedforward Neural Networks. Left: network diagram. Red nodes are stochastic and binary, while the rest of the hiddens are deterministic sigmoid nodes. Right panel: motivation of why multimodal outputs are needed. Given the top half of the face $\mathbf{x}$*, the mouth* $\mathbf{y}$ *can be different, leading to different expressions.*

In this paper, we introduce a Stochastic Feedforward Neural Network (SFNN), based on the Sigmoid Belief Net, for modeling conditional distributions $p(\mathbf{y}|\mathbf{x})$ over continuous real-valued Y output space. Unlike SBNs, to better model continuous data, SFNNs have hidden layers with *both* discrete stochastic *and* deterministic units. We also present a novel Monte Carlo variant of the Generalized Expectation Maximization algorithm for learning. Importance sampling is used for the E-step for inference, while error backpropagation is used by the M-step to improve a variational lower bound on the data log-likelihood.

SFNNs have several attractive properties, including:

- We can draw exact samples from the model without resorting to Markov chain Monte Carlo.
- Stochastic units form a distributed code to represent an exponential number of mixture components in output space.
- As a directed model, learning does not need to deal with a global partition function.
- Combination of stochastic and deterministic hidden units can be jointly trained using the backpropagation algorithm.

There have been several approaches to modeling structured distributions. The Conditional Gaussian Restricted Boltzmann Machines (C-GRBMs) (Taylor et al., 2006) is a popular model for modeling the conditional structured distribution $p(\mathbf{y}|\mathbf{x})$ (C-GRBM is a type of higher-order potential). While C-GRBMs have the advantage of exact inference, it is an energy based model that defines *different* partition functions for different input values of X. Learning also requires Gibbs sampling which is typically prone to poor mixing. Another model capable of learning multimodal output densities is a directed model known as Mixture Density Networks (MDNs) (Bishop, 1994). As the name of the model suggests, MDNs use a mixture of Gaussians to model output Y. The components' means, mixing proportions and the output variances are all modeled by a MLP that has input X. As with SFNNs, backpropagation algorithm can be used to train MDNs efficiently. However, the number of mixture components in the output Y space must be pre-specified and the number of parameters is linear in the number of mixture components. In contrast, SFNNs can model up to 2^{N_h} output components, where N_h is the number of stochastic hidden nodes.

We next describe new learning and inference algorithms for the proposed SFNN model. We then empirically demonstrate that SFNNs can learn good density models and are capable of extracting good representations for various classification tasks.

2. Stochastic feedforward neural networks

Standard multilayer perceptrons (MLPs), or feedforward neural networks, are entirely deterministic. Given an input vector $\mathbf{x}$, MLPs produce a distribution on the output variables $\mathbf{y}$ by defining the sufficient statistics of $\mathbf{y}$ (e.g. the mean) as a function of $\mathbf{x}$. For example, in a MLP with one input, one output and one hidden layer,

$$p(y|\mathbf{x}) \sim \mathcal{N}(y|\mu_y, \sigma_y^2)$$
$$\mu_y = \sigma\big(W_2\sigma(W_1\mathbf{x} + bias_1) + bias_2\big)$$

where $\sigma(a) = 1/(1+\exp(-a))$ is the sigmoid function.

When modeling a structured multimodal output distribution $p(y|\mathbf{x})$, one simple way is to use stochastic hidden variables instead of deterministic ones. This leads to a class of models called stochastic feedforward neural networks (Neal, 1992). The left panel of Fig. 1 shows a diagram of SFNNs with multiple hidden lay-

ers. Given an input vector $\mathbf{x}$, different states of the stochastic nodes can generates different modes in the output space Y.

Stochastic Feedforward Neural Networks (SFNNs), contain binary stochastic hidden variables $\mathbf{h} \in \{0,1\}^{N_h}$, where N_h is the number of hidden nodes. For clarity of presentation, we can construct a SFNN from a MLP with one hidden layer by replacing the sigmoid nodes with stochastic ones. The conditional distribution of interest, $p(y|\mathbf{x})$, is obtained by marginalizing out the latent stochastic hidden variables:

$$p(y|\mathbf{x}) = \sum_{\mathbf{h}} p(y, \mathbf{h}|\mathbf{x}) \tag{1}$$

SFNN has the same structure as a MLP, and it is also a directed graphical model where the generative process starts from $\mathbf{x}$, flows through $\mathbf{h}$, and then generates output y. Thus, we can factorize the joint distribution as:

$$p(y, \mathbf{h}|\mathbf{x}) = p(y|\mathbf{h})p(\mathbf{h}|\mathbf{x}) \tag{2}$$

For modeling real-valued y, we can have $p(y|\mathbf{h}) = \mathcal{N}(y|W_2\mathbf{h} + bias_2, \sigma_y^2)$ and $p(\mathbf{h}|\mathbf{x}) = \sigma(W_1\mathbf{x} + bias_1)$.

Since $\mathbf{h} \in \{0,1\}^{N_h}$ is a vector of Bernoulli random variables, $p(y|\mathbf{x})$ has *potentially* 2^{N_h} different modes[2], one for every possible binary configurations of $\mathbf{h}$. The fact that $\mathbf{h}$ can take on different states in SFNN is the reason why we can learn one-to-many mappings, which would be impossible with standard MLPs.

The modeling flexibility of SFNN comes with computational costs. Since we have a mixture model with potentially 2^{N_h} components conditioned on any $\mathbf{x}$, $p(y|\mathbf{x})$ does not have a closed-form expression. We can use Monte Carlo approximation with M samples for its estimation:

$$p(y|\mathbf{x}) \simeq \frac{1}{M}\sum_{m=1}^{M} p(y|\mathbf{h}^{(m)}) \qquad \mathbf{h}^{(m)} \sim p(\mathbf{h}|\mathbf{x}) \tag{3}$$

This estimator is unbiased and has relatively low variance. This is because the accuracy of the estimator does not depend on the dimensionality of $\mathbf{h}$ and that $p(\mathbf{h}|\mathbf{x})$ is factorial, meaning that we can draw *exact* samples.

If $\mathbf{y}$ is discrete, it is sufficient to have all of the hidden nodes be discrete. However, using only discrete hiddens is suboptimal when modeling real-valued output $\mathbf{y}$. This is due to the fact that while $\mathbf{y}$ is continuous, There are still a finite number of discrete hidden states, each one (e.g. $\mathbf{h}'$) is a Gaussian: $p(y|\mathbf{h}') = \mathcal{N}(y|\mu(\mathbf{h}'), \sigma_y^2)$, where the mean is a function of the hidden state: $\mu(\mathbf{h}') = W_2^{\mathsf{T}}\mathbf{h}' + bias_2$. When $\mathbf{x}$ varies, we only change the probability of *choosing* a specific hidden state $\mathbf{h}'$ via $p(\mathbf{h}'|\mathbf{x})$. However, if we allow $\mu(\mathbf{h}')$ to be a deterministic function of $\mathbf{x}$ as well, we can learn a smoother $p(y|\mathbf{x})$, even when it is desirable to learn small residual variances σ_y^2. This can be accomplished by allowing for both stochastic *and* deterministic units in a single SFNN hidden layer. This also allows the mean $\mu(\mathbf{h}', \mathbf{x})$ to have contributions from two components, one from the hidden state $\mathbf{h}'$, and another one from defining a deterministic mapping from $\mathbf{x}$. As we demonstrate in our experimental results, this is crucial for learning good density models of the real-valued $\mathbf{y}$.

In SFNNs with only one hidden layer, $p(\mathbf{h}|\mathbf{x})$ is a factorial Bernoulli distribution. If $p(\mathbf{h}|\mathbf{x})$ has low entropy, only a few discrete $\mathbf{h}$ states out of the 2^{N_h} would have any significant probability mass. We can increase the entropy over the stochastic hidden variables by adding a second hidden layer. The second hidden layer takes the discretized stochastic and any deterministic hidden nodes of the first layer as its input. This leads to our proposed SFNN model, shown in Fig. 1.

In our SFNNs, we assume a conditional diagonal Gaussian distribution on the output space:

$$\log p(\mathbf{y}|\mathbf{h}, \mathbf{x}) \propto -\frac{1}{2}\sum_i \log \sigma_i^2 - \frac{1}{2}\sum_i \frac{(y_i - \mu(\mathbf{h}, \mathbf{x}))^2}{\sigma_i^2}$$

We note that we can also use any other parameterized distribution (e.g. Student's t) for the output variables. This is a win compared to the Boltzmann Machine family of models, which require the output distribution to be from the exponential family.

2.1. Learning

We present a Monte Carlo variant of the Generalized EM algorithm (Neal & Hinton, 1998) for learning SFNNs. Specifically, importance sampling is used during the E-step to approximate the posterior $p(\mathbf{h}|y, \mathbf{x})$, while the Backprop algorithm is used during the M-step to calculate the derivatives of the parameters of both the stochastic and deterministic nodes. Gradient ascent using the derivatives will guarantee that the variational lower bound of the model log-likelihood will be improved. The drawback of our learning algorithm is that we require drawing M samples for the stochastic nodes for every weight update. However, as we will show in the experimental results, 20 samples is sufficient for learning very good SFNNs empirically.

The requirement of sampling is typical for models capable of structured learning. As a comparison, energy based models, such as conditional Restricted Boltzmann Machines, require MCMC sampling per weight

[2]In practice, due to weight sharing, we will not be able to have close to that many modes for a large N_h.

update to estimate the gradient of the log-partition function. These MCMC samples do not converge to the true distribution which in turn gives a biased estimate of the gradient.

For clarity, we provide the following derivations for SFNNs with one hidden layer containing only stochastic nodes[3]. For any approximating distribution $q(\mathbf{h})$, we can write down the following variational lower-bound on the data log-likelihood:

$$\begin{aligned}\log p(y|\mathbf{x}) &= \log \sum_{\mathbf{h}} p(y,\mathbf{h}|\mathbf{x}) \\ &= \sum_{\mathbf{h}} p(\mathbf{h}|y,\mathbf{x}) \log \frac{p(y,\mathbf{h}|\mathbf{x})}{p(\mathbf{h}|y,\mathbf{x})} \\ &= \sum_{\mathbf{h}} q(\mathbf{h}) \log \frac{p(y,\mathbf{h}|\mathbf{x};\theta)}{q(\mathbf{h})} + \mathrm{KL}(q(\mathbf{h})||p(\mathbf{h}|y,\mathbf{x})) \\ &\geq \sum_{\mathbf{h}} q(\mathbf{h}) \log \frac{p(y,\mathbf{h}|\mathbf{x};\theta)}{q(\mathbf{h})} \qquad (4)\end{aligned}$$

We arrived at the lower bound of the log-likelihood in Eq. 4 since the KL divergence must be greater than or equal to zero. $q(\mathbf{h})$ can be any arbitrary distribution. For the tightest lower-bound, $q(\mathbf{h})$ need to be the exact posterior $p(\mathbf{h}|y,\mathbf{x})$.

Since the posterior $p(\mathbf{h}|y,\mathbf{x})$ is hard to compute, but the "conditional prior" on $\mathbf{h}$ $p(\mathbf{h}|\mathbf{x})$ is easy (corresponding to a simple feedforward pass), we can set $q(\mathbf{h}) \triangleq p(\mathbf{h}|\mathbf{x})$. However, this would be a very bad approximation as learning proceeds, since the learning of the likelihood will increase the KL between the conditional prior and the posterior. Instead, we use importance sampling with the conditional prior as the proposal distribution.

Let Q be the expected complete data log-likelihood, which is a lower bound on the log-likelihood. We wish to maximize the lower bound:

$$\begin{aligned}Q(\theta,\theta_{old}) &= \sum_{\mathbf{h}} p(\mathbf{h}|y,\mathbf{x};\theta_{old}) \log p(y,\mathbf{h}|\mathbf{x};\theta) \qquad (5)\\ &= \sum_{\mathbf{h}} \frac{p(\mathbf{h}|y,\mathbf{x};\theta_{old})}{p(\mathbf{h}|\mathbf{x};\theta_{old})} p(\mathbf{h}|\mathbf{x};\theta_{old}) \log p(y,\mathbf{h}|\mathbf{x};\theta) \\ &\simeq \frac{1}{M}\sum_{m=1}^{M} w^{(m)} \log p(y,\mathbf{h}^{(m)}|\mathbf{x};\theta), \quad \mathbf{h}^{(m)} \sim p(\mathbf{h}|\mathbf{x};\theta_{old})\end{aligned}$$

where $w^{(m)}$ is the importance weight of the m-th sample from the proposal distribution $p(\mathbf{h}|\mathbf{x};\theta_{old})$. Using Bayes Theorem, we have

$$w^{(m)} = \frac{p(\mathbf{h}^{(m)}|y,\mathbf{x};\theta_{old})}{p(\mathbf{h}^{(m)}|\mathbf{x};\theta_{old})} = \frac{p(y|\mathbf{h}^{(m)},\mathbf{x};\theta_{old})}{p(y|\mathbf{x};\theta_{old})} \qquad (6)$$

The denominator of the RHS can be approximated using Eq. 3, therefore:

$$w^{(m)} \simeq \frac{p(y|\mathbf{h}^{(m)};\theta_{old})}{\frac{1}{M}\sum_{m=1}^{M} p(y|\mathbf{h}^{(m)};\theta_{old})} \qquad (7)$$

For convenience, we define the partial objective of the m-th sample as

$$Q^{(m)} \triangleq w^{(m)}\big(\log p(y|\mathbf{h}^{(m)};\theta) + \log p(\mathbf{h}^{(m)}|\mathbf{x};\theta)\big). \quad (8)$$

We can then approximate our objective function $Q(\theta,\theta_{old})$ with M samples from the proposal,

$$Q(\theta,\theta_{old}) \simeq \frac{1}{M}\sum_{m=1}^{M} Q^{(m)}(\theta,\theta_{old})$$

For our generalized M-step, we seek to perform gradient ascent on Q:

$$\begin{aligned}\frac{\partial Q}{\partial \theta} &\simeq \frac{1}{M}\sum_{m=1}^{M} \frac{\partial Q^{(m)}(\theta,\theta_{old})}{\partial \theta} \qquad (9)\\ &= \frac{1}{M}\sum_{m=1}^{M} w^{(m)} \frac{\partial}{\partial \theta}\Big\{ \log p(y|\mathbf{h}^{(m)};\theta) + \log p(\mathbf{h}^{(m)}|\mathbf{x};\theta)\Big\}\end{aligned}$$

Algorithm 1 EM algorithm for Learning Stochastic Feedforward Networks (See Fig. 1)

Given training D dimensional data pairs: $\{\mathbf{x}^{(n)},\mathbf{y}^{(n)}\}$, $n = 1 \ldots N$. Hidden layers $\mathbf{h}^1$&$\mathbf{h}^4$ are deterministic, $\mathbf{h}^2$&$\mathbf{h}^3$ are hybrid. $\theta = \{W^{1,2,3,4,5}, bias, \sigma_y^2\}$
repeat
//Approximate E-step:
Compute $p(\mathbf{h}^2|\mathbf{x}^{(n)}) = Bernoulli\big(\sigma(W^2\sigma(W^1\mathbf{x}^{(n)}))\big)$
$\mathbf{h}^2_{determ} \leftarrow p(\mathbf{h}^2_{determ}|\mathbf{x}^{(n)})$
for $m = 1$ **to** M (importance samples) **do**
  Sample: $\mathbf{h}^2_{stoch} \sim p(\mathbf{h}^2_{stoch}|\mathbf{x}^{(n)})$.
  let $\mathbf{h}^2$ be the concatenation of $\mathbf{h}^2_{stoch}$ and $\mathbf{h}^2_{determ}$.
  $p(\mathbf{h}^3|\mathbf{x}^{(n)}) = Bernoulli\big(\sigma(W^3\mathbf{h}^2)\big)$
  $\mathbf{h}^3_{determ} \leftarrow p(\mathbf{h}^3_{determ}|\mathbf{x}^{(n)})$
  Sample: $\mathbf{h}^3_{stoch} \sim p(\mathbf{h}^3_{stoch}|\mathbf{x}^{(n)})$
  let $\mathbf{h}^3$ be the concatenation of $\mathbf{h}^3_{stoch}$ and $\mathbf{h}^3_{determ}$.
  Compute $p(\mathbf{y}|\mathbf{x}^{(n)}) = \mathcal{N}\big(\sigma(W^5\sigma(W^4\mathbf{h}^3)); \sigma_y^2\big)$
end for
Compute $w^{(m)}$ for all m, using Eq. 7.

//M-step:
$\triangle\theta \leftarrow 0$
for $m = 1$ **to** M **do**
  Compute $\frac{\partial Q^{(m)}(\theta,\theta_{old})}{\partial \theta}$ by Backprop.
  $\triangle\theta = \triangle\theta + \partial Q^{(m)}/\partial\theta$
end for
$\theta_{new} = \theta_{old} + \frac{\alpha}{M}\triangle\theta$, *//*$\alpha$ is the learning rate.
until convergence

[3]It is straightforward to extend the model to multiple and hybid hidden layered SFNNs.

The terms $\frac{\partial}{\partial\theta}\{.\}$ can be calculated using error backpropagation of two parts. The first part, $\frac{\partial}{\partial\theta}\Big\{\log p(y|\mathbf{h}^{(m)};\theta)\Big\}$, treats y as the targets and $\mathbf{h}^{(m)}$ as the input data, while the second part, $\frac{\partial}{\partial\theta}\Big\{\log p(\mathbf{h}^{(m)}|\mathbf{x};\theta)\Big\}$, treats $\mathbf{h}^{(m)}$ as the targets and $\mathbf{x}$ as the input data. In SFNNs with hidden layers with mixture of deterministic and stochastic units, backprop will additionally propagate error information from the first part to the second part.

The full gradient is a weighted summation of the M partial derivatives, where the weighting comes from how well a particular state $\mathbf{h}^{(m)}$ can generate the data y. This is intuitively appealing, since learning adjusts both the "preferred" states' abilities to generate the data (first part in the braces), as well as increase their likelihood of being picked conditioning on $\mathbf{x}$ (second part in the braces). The detailed EM learning algorithm for SFNNs is listed in Alg. 1.

2.2. Cooperation during learning

We note that for importance sampling to work well in general, a key requirement is that the proposal distribution not to be small where the true distribution is significant. However, things are slightly different when using importance sampling for during learning. Our proposal distribution $p(\mathbf{h}|\mathbf{x})$ and the posterior $p(\mathbf{h}|y,\mathbf{x})$ are not fixed by rather governed by model parameters. Learning adapts these distribution in a synergistic and cooperative fashion.

Let's hypothesize that at a particular learning iteration, the conditional prior $p(\mathbf{h}|\mathbf{x})$ is small in certain regions where $p(\mathbf{h}|y,\mathbf{x})$ is large, which is bad for importance sampling. The E-step will draw M samples and weight them according to Eq. 7. While all samples $\mathbf{h}^{(m)}$ will have very low log-likelihood due to the bad conditional prior, there will be certain preferred state $\hat{\mathbf{h}}$ since the importance weighting is normalized to sum to 1. Learning in Eq. 9 will to two things. 1) it will adjust the generative weights so to allow preferred states to better generate the observed y; 2) it will make the conditional prior better by making it more likely to predict $\hat{\mathbf{h}}$ given $\mathbf{x}$. Since the generative weights are shared, the fact that $\hat{\mathbf{h}}$ generates y accurately will probably reduce the likelihood of y under another state $\mathbf{h}$. The updated conditional prior tends to be a proposal distribution for the updated model. The cooperative interaction between the conditional prior and posterior during learning provides some robustness to the importance sampler.

Empirically, we can see this effect as learning progress on Dataset A of Sec. 3.1 in Fig. 2. The plot shows the model log-likelihood given the training data as learning progresses until 3000 weight updates. 30 importance samples are used during learning with 2 hidden layers of 5 stochastic nodes. We chose 5 nodes because it is small enough that the true log-likelihood can be computed using brute-force integration. As learning progresses, the Monte Carlo approximation is very close to the true log-likelihood using only 30 samples. As expected, the KL from the posterior and prior diverges as the generative weights better models the multi-modalities around $x = 0.5$. We also compared the KL divergence between our empirical weighted importance sampled distribution and true posterior, which converges toward zero. This demonstrate that the prior distribution have learned to not be small in regions of large posterior. In another words this shows that the E-step in the learning of SFNNs is close to exact for this dataset and model.

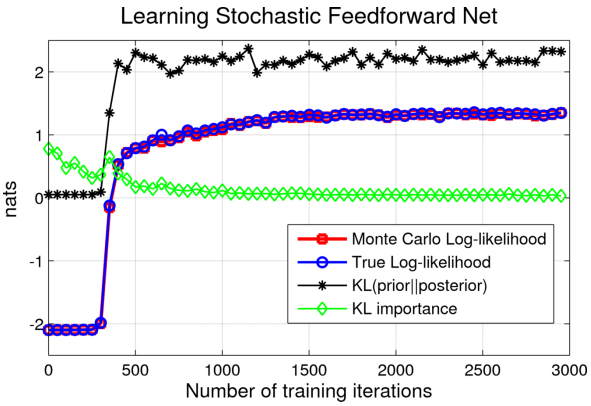


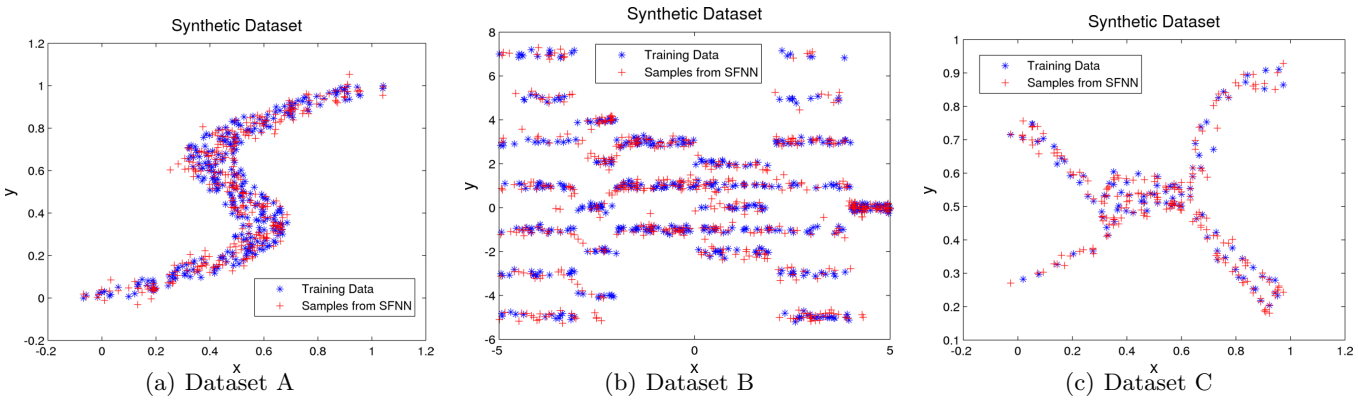
Figure 2. KL divergence and log-likelihoods. Best viewed in color.

3. Experiments

We first demonstrate the effectiveness of SFNN on synthetic one dimensional one-to-many mapping data. We then use SFNNs to model face images with varying facial expressions and emotions. SFNNs outperform other competing density models by a large margin. We also demonstrate the usefulness of latent features learned by SFNNs for expression classification. Finally, we train SFNNs on a dataset with in-depth head rotations, a database with colored objects, and a image segmentation database. By drawing samples from these trained SFNNs, we obtain qualitative results and insights into the modeling capacity of SFNNs.

3.1. Synthetic datasets

As a proof of concept, we used three one dimensional one-to-many mapping datasets, shown in Fig. 3.



(a) Dataset A (b) Dataset B (c) Dataset C

Figure 3. Three synthetic datasets of 1-dimensional one-to-many mappings. For any given x, multiple modes in y exist. Blue stars are the training data, red pluses are exact *samples from SFNNs. Best viewed in color.*

	Gaussian	MDN	C-GRBM	SBN	SFNN
A	0.078 ±0.02	1.05 ±0.02	0.57 ±0.01	0.79 ±0.03	1.04 ±0.03
B	-2.40 ±0.07	-1.58 ±0.11	-2.14 ±0.04	-1.33 ±0.10	-0.98 ±0.06
C	0.37 ±0.07	2.03 ±0.05	1.36 ±0.05	1.74 ±0.08	2.21 ±0.16

Table 1. Average test log-probability on synthetic 1D datasets.

Our goal is to model $p(y|x)$. Dataset A was used by (Bishop, 1994) to evaluate the performance of the Mixture Density Networks (MDNs). Dataset B has a large number of tight modes when x is negative, which is useful for testing a model's ability to learn many modes and a small residual variance. Dataset C is used for testing whether a model can learn modes that are far apart from each other.

We randomly split the data into a training, validation and a test set. We report the average test set log-probability averaged over 5 folds for different models in Table 1. The method called 'Gaussian' is a 2D Gaussian estimated on (x, y) jointly, and we report $\log p(y|x)$ which can be obtained easily in closed-form. C-GRBM is the Conditional Gaussian Restricted Boltzmann Machine where we used 25-step Contrastive Divergence (Hinton, 2002) (CD-25) to estimate the gradient of the log partition function. SBN is a Sigmoid Belief Net with three hidden stochastic binary layers between the input and the output layer. It is trained in the same way as SFNN, but there are no deterministic units. Finally, SFNN has four hidden layers with the inner two being hybrid stochastic/deterministic layers (See Fig. 1). We used 30 importance samples to approximate the posterior during the E-step. All other hyper-parameters for all of the models were chosen to maximize the validation performance.

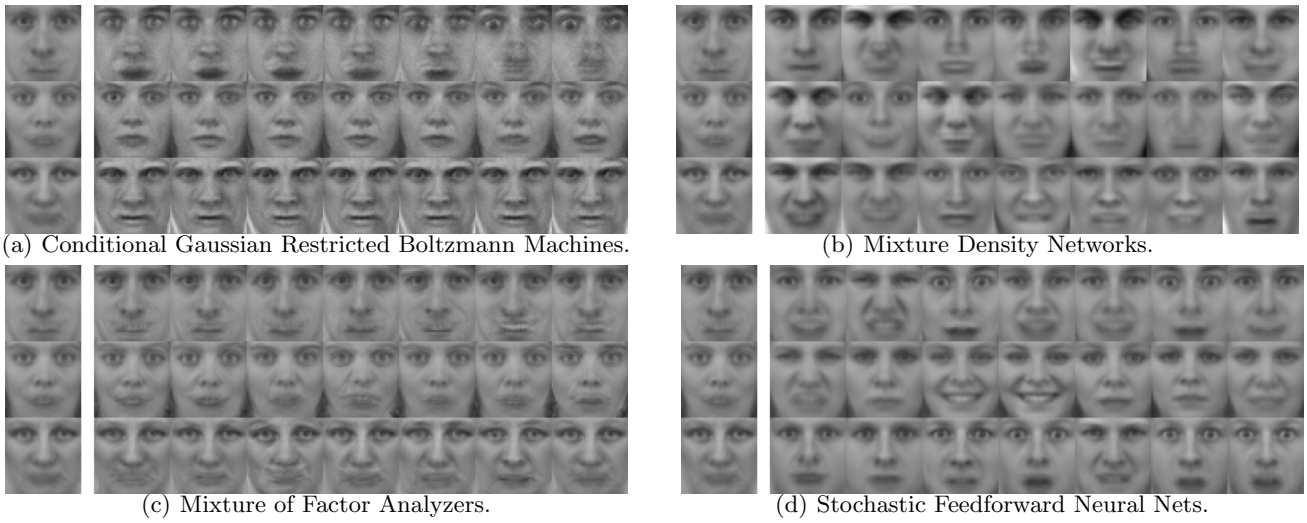
Table 1 reveals that SFNNs consistently outperform all other methods. Fig. 3 further shows samples drawn from SFNNs as red 'pluses'. Note that SFNNs can learn small residual variances to accurately model Dataset B. Comparing SBNs to SFNNs, it is clear that having deterministic hidden nodes is a huge win for modeling continuous y.

Figure 5. Facial Expression Training Data. Images are a subset of the Toronto Face Database.

3.2. Modeling Facial Expression

Conditioned on a subject's face with neutral expression, the distribution of all possible emotions or expressions of this particular individual is multimodal in pixel space. We learn SFNNs to model facial expressions in the Toronto Face Database (Susskind, 2011). The Toronto Face Database consist of 4000 images of 900 individuals with 7 different expressions. Of the 900 subjects, there are 124 with 10 or more images per subject, which we used as our data. We randomly selected 100 subjects with 1385 total images for training, while 24 subjects with a total of 344 images were selected as the test set. Figure 5 shows sample face images of the TFD dataset.

For each subject, we take the average of its face images as $\mathbf{x}$ (mean face), and learn to model this subject's varying expressions $\mathbf{y}$. Both $\mathbf{x}$ and $\mathbf{y}$ are grayscale and downsampled to a resolution of 48×48. We trained a SFNN with 4 hidden layers of size 128 on these facial expression images. The second and third "hybrid" hidden layers contained 32 stochastic binary and 96 de-



(a) Conditional Gaussian Restricted Boltzmann Machines. (b) Mixture Density Networks.

(c) Mixture of Factor Analyzers. (d) Stochastic Feedforward Neural Nets.

Figure 4. Samples generated from various models.

terministic hidden nodes, while the first and the fourth hidden layers consisted of only deterministic sigmoids. We refer this model as SFNN2. For comparisons, we also tested the same model but with only one hybrid hidden layer, that we call SFNN1. We used mini-batches of size 100 and and 30 importance samples for the E-step. A total of 2500 weight updates were performed. Weights were randomly initialized with standard deviation of 0.1, and the residual variance σ_y^2 was initialized to the variance of $\mathbf{y}$.

For comparisons with other models, we trained a Mixture of Factor Analyzers (MFA) (Ghahramani & Hinton, 1996), Mixture Density Networks (MDN), and Conditional Gaussian Restricted Boltzmann Machines (C-GRBM) on this task. For the Mixture of Factor Analyzers model, we trained a mixture with 100 components, one for each training individual. Given a new test face $\mathbf{x}_{test}$, we first find the training $\hat{\mathbf{x}}$ which is closest in Euclidean distance. We then take the parameters of the $\hat{\mathbf{x}}$'s FA component, while replacing the FA's mean with $\mathbf{x}_{test}$. Mixture Density Networks is trained using code provided by the NETLAB package (Nabney, 2002). The number of Gaussian mixture components and the number of hidden nodes were selected using a validation set. Optimization is performed using the scaled conjugate gradient algorithm until convergence. For C-GRBMs, we used CD-25 for training. The optimal number of hidden units, selected via validation, was 1024. A population sparsity objective on the hidden activations was also part of the objective (Nair & Hinton, 2009). The residual diagonal covariance matrix is also learned. Optimization used stochastic gradient descent with mini-batches of 100 samples each.

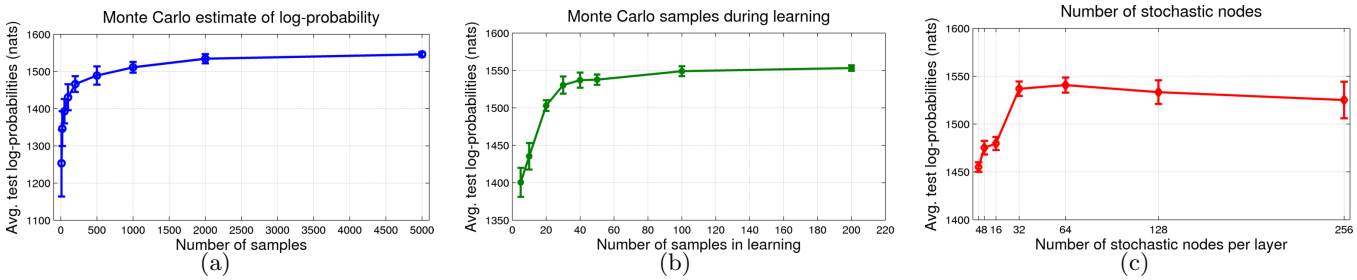
	MFA	MDN	C-GRBM	SFNN1	SFNN2
Nats	1406 $\pm$ 52	1321 $\pm$ 16	1146 $\pm$ 113	1488 $\pm$ 18	1534 $\pm$ 27
Time	10 secs.	6 mins.	158 mins.	112 secs.	113 secs.

Table 2. Average test log-probability and total training time on facial expression images.

Table 2 displays the average log-probabilities[4] along with standard errors of the 344 test images. We also recorded the total training time of each algorithm, although this depends on the number of weight updates and whether or not GPUs are used (see Sec. 3.4 for more details). For MFA and MDN, the log-probabilities were computed exactly. For C-GRBMs we used Annealed Importance Sampling (Neal, 2001; Salakhutdinov & Murray, 2008) with 50,000 intermediate temperatures to estimate the partition function for each test image $\mathbf{x}$. For SFNNs, we used Eq. 3 with 1000 samples. We can see that SFNNs substantially outperform all other models. Having two hybrid hidden layers (SFNN2) improves model performance over SFNN1, which has only one hybrid hidden layer.

Qualitatively, Fig. 4 shows samples drawn from the trained models. The leftmost column are the mean faces of 3 test subjects, followed by 7 samples from the distribution $p(\mathbf{y}|\mathbf{x})$. For C-GRBM, samples are generated from a Gibbs chain, where each successive image is taken after 1000 steps. For the other 3 models, displayed samples are *exact*. MFAs overfit on the training set, generating samples with significant artifacts. MDNs can model many modes, but the identity of the test subject is changed, which is undesirable. Samples produced by C-GRBMs suffer from poor mixing and get stuck at a local mode. SFNN samples show

[4]For continuous data, these are probabilities densities and can be positive.



(a) (b) (c)

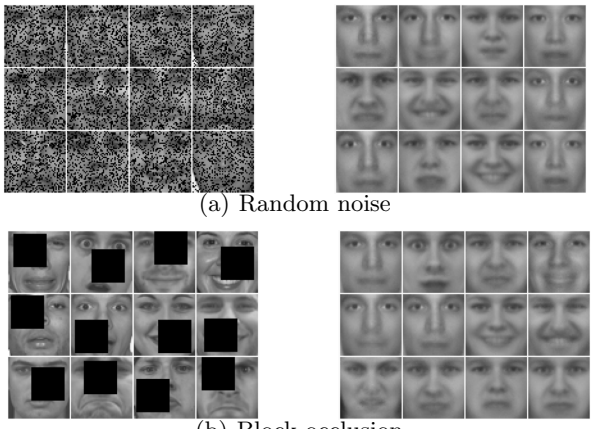
Figure 6. Plots demonstrate how hyperparameters affect the evaluation and learning of SFNNs.

	Linear	C-GRBM +Linear	SFNN +Linear	MLP	SFNN +MLP
clean	80.0%	81.4%	**82.4%**	83.2%	83.8 %
10% noise	78.9%	79.7%	80.8%	82.0%	81.7 %
50% noise	72.4%	**74.3%**	71.8%	79.1%	78.5%
75% noise	52.6%	58.1%	**59.8%**	71.9%	**73.1**%
10% occl.	76.2%	79.5%	80.1%	80.3%	**81.5**%
50% occl.	54.1%	59.9%	**62.5%**	58.5%	**63.4**%
75% occl.	28.2%	33.9%	**37.5%**	33.2%	**39.2**%

Table 3. Recognition accuracy over 5 folds. Bold numbers indicates that the difference in accuracy is statistically significant than the competitor models, for both the linear and nonlinear classifier groups.

that the model was able to capture a combination of mutli-modality and preserved much of the identity of the test subjects.

We further explored how different hyperparameters, such as the number of stochastic layers, and the number of Monte Carlo samples, can affect the learning and evaluation of SFNNs. We used face images and SFNN2 for these experiments. First, we wanted to know the number of M (Eq. 3) needed to give a reasonable estimate of the log-probabilities. Fig. 6(a) shows the estimates of the log-probability as a function of the number of samples. We can see that having about 500 samples is reasonable, but more samples provides a slightly better estimate. The general shape of the plot is similar for all other datasets and SFNN models. When M is small, we typically underestimate the true log-probabilities. While 500 or more samples are needed for accurate model *evaluation*, only 20 or 30 samples are sufficient for learning good models (as shown in Fig. 6(b). This is because while $M = 20$ gives sub-optimal approximation to the true posterior, learning still improves the variational lower-bound. In fact, we can see that the difference between using 30 and 200 samples during learning results in only about 20 nats of the final average test log-probability. In Fig. 6(c), we varied the number of binary stochastic hidden variables in the 2 inner hybrid layers. We did not observe significant improvements beyond more than 32 nodes.



(a) Random noise

(b) Block occlusion

Figure 7. SFNNs is robust to noise (a) and occlusions (b). Left panel shows a noisy test images $\mathbf{y}$*. Posterior inference in SFNN finds* $E_{p(\mathbf{h}|\mathbf{x},\mathbf{y})}[\mathbf{h}]$*. Right panel shows generated* $\mathbf{y}$ *images from the expected hidden activations.*

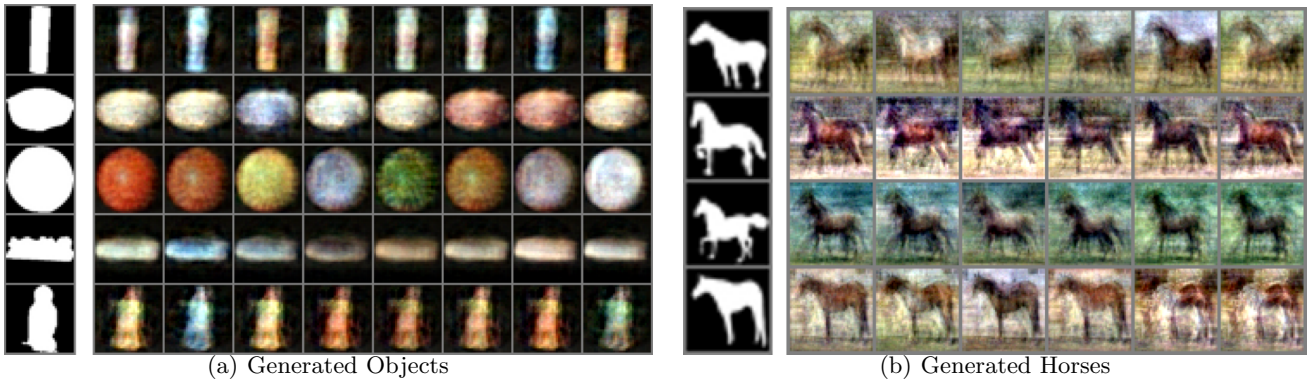
With more hidden nodes, over-fitting can also be a problem.

3.2.1. Expression Classification

The internal hidden representations learned by SFNNs are also useful for classification of facial expressions. For each $\{\mathbf{x}, \mathbf{y}\}$ image pair, there are 7 possible expression types: neutral, angry, happy, sad, surprised, fear, and disgust.

As baselines, we used regularized linear softmax classifiers and multilayer perceptron classifier taking pixels as input. The mean of every pixel across on cases was set to 0 and standard deviation was set to 1.0. We then append the learned hidden features of SFNNs and C-GRBMs to the image pixels and re-train the same classifiers. The results are shown in the first row of Table 3. Adding hidden features from the SFNN trained in an unsupervised manner (without expression labels) improves accuracy for both linear and nonlinear classifiers.

SFNNs are also useful when dealing with noise. As a generative model of $\mathbf{y}$, it is somewhat robust to noisy and occluded pixels. For example, the left panels of Fig. 7, show corrupted test $\mathbf{y}$ images. Using the im-



(a) Generated Objects (b) Generated Horses

Figure 8. Samples generated from a SFNN after training on an object database. Conditioned on a given foreground mask, the appearance is multimodal (different color and texture). Best viewed in color.

Figure 9. Samples from SFNN trained on rotated faces.

portance sampler described in Sec. 2.1 we can compute the expected values of the binary stochastic hidden variables given the corrupted test $\mathbf{y}$ images[5]. In the right panels of Fig. 7, we show the corresponding generated $\mathbf{y}$ from the inferred average hidden states. After this denoising process, we can then feed the denoised $\mathbf{y}$ and $E[\mathbf{h}]$ to the classifiers. This compares favorably to simply filling in the missing pixels with the average of that pixel from the training set. Classification accuracies under noise are also presented in Table 3. For example 10% noise means that 10 percent of the pixels of *both* $\mathbf{x}$ and $\mathbf{y}$ are corrupted, selected randomly. 50% occlusion means that a square block with 50% of the original area is randomly positioned in both $\mathbf{x}$ and $\mathbf{y}$. Gains in recognition performance from using SFNN are particularly pronounced when dealing with large amounts of random noise and occlusions.

3.3. Additional Qualitative Experiments

Not only SFNNs are capable of modeling facial expression of aligned face images, they can also model complex real-valued conditional distributions. In this section, we present some qualitative samples drawn from SFNNs trained on more complicated distributions. We start by learning the UMIST faces database (Graham & Allinson, 1998), which contains in-depth 3D rotation of heads. We then tested SFNNs on modeling generating colorful images of common objects from the ALOI dataset (Geusebroek et al., 2005), conditioned on the foreground masks. Finally, we tested on the Weizmann segmentation database (Borenstein & Ullman, 2002) of horses, learning a conditional distribution of horse appearances conditioned on the segmentation mask.

We trained the same SFNNs as described in the previous section on the 16 training subjects, using 4 subjects for testing. Conditioned on the profile view, we are modeling the distribution of rotations up to 90 degrees. Fig. 9 displays 3 test subjects' profile view along with seven exact samples drawn from the model (plotted on the right hand side). This is a particularly difficult task, as the model must learn to generate face parts, such as eyes and nose.

Amsterdam Library of Objects database (Geusebroek et al., 2005) is a database of 1000 everyday objects under various lighting, rotations, and viewpoints. Every object also comes with a foreground segmentation mask. For every object, we selected the image under frontal lighting without any rotations, and trained a SFNN conditioned on the foreground mask. Our goal is to model the appearance (color and texture) of these objects. We note that out of 1000 objects, there are many objects with similar foreground masks (e.g. round or rectangular). Conditioned on the test foreground masks, Fig. 8 shows random samples from the learned SFNN model.

3.4. Computation Time

Despite having to draw M samples during learning, Fig. 6 empirically demonstrated that 20 samples is often sufficient[6]. This is in part due to the fact that

[5]For this task we assume that we have knowledge of which pixels is corrupted.

[6]We note that this is still M times more expensive than standard backprop.

samples from the conditional prior are exact and in part due to the cooperation that occurs during learning (Sec. 2.2). Regarding hardware, our experiments are performed on nVidia GTX580 GPUs. This gives us over 10x speedup over CPUs. For example, a 4 hidden layer SFNN with 2304 input and output dimensions, 128 stochastic hidden nodes, and 50 samples per E-step, can update its parameters in 0.15 secs on a minibatch of 100 cases.

In Table 2, C-GRBM is also trained on the GPU, but is much slower due to its use of a large hidden layer and 25 CD steps. For example, the C-GRBM requires 1.16 secs per parameter update. MFA and MDNs are run on CPUs and we can also expect 10x speedup from moving to GPUs.

4. Discussions

In this paper we introduced a novel model with hybrid stochastic and deterministic hidden nodes. We have also proposed an efficient learning algorithm that allows us to learn rich multi-modal conditional distributions, supported by quantitative and qualitative empirical results.

The major drawback of SFNNs is that inference is not trivial and M samples are needed for the importance sampler. While this is sufficiently fast for our experiments we can accelerate inference by learning a separate recognition network to perform inference in one feedforward pass. These techniques have previously been used by (Hinton et al., 1995; Salakhutdinov & Larochelle, 2010) with success.

References

Barber, David and Sollich, Peter. Gaussian fields for approximate inference in layered sigmoid belief networks. In Solla, Sara A., Leen, Todd K., and Mller, Klaus-Robert (eds.), *NIPS*, pp. 393–399. The MIT Press, 1999. ISBN 0-262-19450-3.

Bishop, C. M. Mixture density networks. Technical Report NCRG/94/004, Aston University, 1994.

Borenstein, Eran and Ullman, Shimon. Class-specific, top-down segmentation. In *In ECCV*, pp. 109–124, 2002.

Geusebroek, J. M., Burghouts, G. J., and Smeulders, A. W. M. The amsterdam library of object images. *International Journal of Computer Vision*, 61(1), January 2005.

Ghahramani, Zoubin and Hinton, G. E. The EM algorithm for mixtures of factor analyzers. Technical Report CRG-TR-96-1, University of Toronto, 1996. URL `ftp://ftp.cs.toronto.edu/pub/zoubin/tr-96-1.ps.gz`.

Graham, Daniel B and Allinson, Nigel M. Characterizing virtual eigensignatures for general purpose face recognition. In *Face Recognition: From Theory to Applications*, pp. 446–456. 1998.

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

Hinton, G. E., Dayan, P., Frey, B. J., and Neal, R. M. The wake-sleep algorithm for unsupervised neural networks. *Science*, 268(5214):1158–1161, 1995.

Nabney, Ian. *NETLAB: algorithms for pattern recognitions*. Advances in pattern recognition. Springer-Verlag, 2002.

Nair, V. and Hinton, G. E. 3-D object recognition with deep belief nets. In *NIPS 22*, 2009.

Neal, R. M. Connectionist learning of belief networks. volume 56, pp. 71–113, July 1992.

Neal, R. M. Annealed importance sampling. *Statistics and Computing*, 11:125–139, 2001.

Neal, R. M. and Hinton, G. E. A new view of the EM algorithm that justifies incremental, sparse and other variants. In Jordan, M. I. (ed.), *Learning in Graphical Models*, pp. 355–368. 1998.

Salakhutdinov, R. and Larochelle, H. Efficient learning of deep boltzmann machines. *AISTATS*, 2010.

Salakhutdinov, R. and Murray, I. On the quantitative analysis of deep belief networks. In *Proceedings of the Intl. Conf. on Machine Learning*, volume 25, 2008.

Saul, Lawrence K., Jaakkola, Tommi, and Jordan, Michael I. Mean field theory for sigmoid belief networks. *Journal of Artificial Intelligence Research*, 4:61–76, 1996.

Susskind, J.M. The Toronto Face Database. Technical report, 2011. http://aclab.ca/users/josh/TFD.html.

Taylor, G., Hinton, G. E., and Roweis, S. Modeling human motion using binary latent variables. In *NIPS*, 2006.