Learning Deep Energy Models

Jiquan Ngiam JNGIAM@CS.STANFORD.EDU
Zhenghao Chen ZHENGHAO@CS.STANFORD.EDU
Pang Wei Koh PANGWEI@CS.STANFORD.EDU
Andrew Y. Ng ANG@CS.STANFORD.EDU
Computer Science Department, Stanford University, Stanford, CA 94305, USA

Abstract

Deep generative models with multiple hidden layers have been shown to be able to learn meaningful and compact representations of data. In this work we propose deep energy models, which use deep feedforward neural networks to model the energy landscapes that define probabilistic models. We are able to efficiently train all layers of our model simultaneously, allowing the lower layers of the model to adapt to the training of the higher layers, and thereby producing better generative models. We evaluate the generative performance of our models on natural images and demonstrate that this joint training of multiple layers yields qualitative and quantitative improvements over greedy layerwise training. We further generalize our models beyond the commonly used sigmoidal neural networks and show how a deep extension of the product of Student-t distributions model achieves good generative performance. Finally, we introduce a discriminative extension of our model and demonstrate that it outperforms other fully-connected models on object recognition on the NORB dataset.

1. Introduction

Deep networks are able to learn rich and complex models of data, making them well suited as generative models of images and other natural data (Bengio, 2007). However, having multiple layers of stochastic hidden units makes inference and learning challenging.

To overcome this problem, we propose using deep energy models. This model can be viewed as having a feedforward neural network that deterministically transforms the input and subsequently models the output of the feedforward network with a layer of stochastic hidden units. Intuitively, the feedforward neural network extracts features from the input which are more easily modeled with a single stochastic layer. This formulation allows us to efficiently train all layers of the network jointly, leading to better generative models.

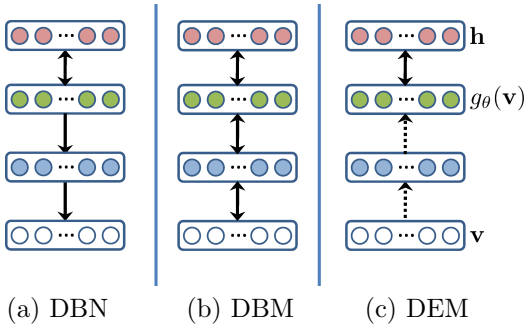


Figure 1. Comparison of (a) deep belief networks (DBNs), (b) deep Boltzmann machines (DBMs) and (c) deep energy models (DEMs). DBNs have undirected connections at the top two layers which form a RBM and directed connections to the lower layers. DBMs have undirected connections through all layers. DEMs can viewed as having deterministic hidden units for the lower layers and stochastic hidden units at the top hidden layer. Here, dotted arrows represent deterministic relationships.

Our work builds on Hinton et al. (2006b), which introduced the contrastive backpropagation algorithm, and also the work of Mnih & Hinton (2005), who first showed how to stack a product of Student-t model. We defer to Section 3 for a further discussion of these connections.

1.1. Related Work

We first review two popular deep generative models, deep belief networks (DBNs) and deep Boltzmann ma-

Appearing in *Proceedings of the 28th International Conference on Machine Learning*, Bellevue, WA, USA, 2011.

chines (DBMs). In their seminal work, Hinton et al. (2006a) demonstrated how to train deep belief networks with multiple layers of hidden units efficiently. A deep belief network is a graphical model with undirected connections at the top hidden layers and directed connections in the lower layers (Fig. 1a). Their learning algorithm uses greedy layerwise training by stacking restricted Boltzmann machines (RBMs), each of which models the posterior distribution of the previous layer. Hinton et al. (2006a) showed that a variational bound on the data likelihood can always be improved by adding sufficiently large layers. We note that it is possible, but computationally expensive, to train all layers of the DBN jointly using a contrastive wake-sleep algorithm.

A closely related model, deep Boltzmann machines, was recently proposed by Salakhutdinov & Hinton (2009); Salakhutdinov & Larochelle (2010). In contrast to DBNs, DBMs have undirected connections between all layers of the network (Fig. 1b). A similar layerwise training algorithm using RBMs is used to initialize the DBM and all layers are jointly trained thereafter. In recent developments, Salakhutdinov & Larochelle (2010) demonstrate a new method of training DBMs more efficiently using approximate inference.

DBNs and DBMs share the property of having multiple stochastic hidden layers; this makes inference and learning difficult as computing the conditional posterior over the hidden units is intractable. In contrast, inference and learning are more tractable in models with only a single stochastic hidden layer (e.g., an RBM, which exploits the conditional independence of the hidden variables given the visible variables). Our model builds upon the benefits of having only a single layer of stochastic hidden units for efficient training and inference.

2. Deep Energy Models

We first motivate deep energy models (DEMs) by considering models which combine multiple deterministic hidden layers with a single stochastic hidden layer. These models deterministically transform the input into a new representation using a feedforward neural network, before modeling the output of this feedforward network with a single layer of stochastic hidden units (Fig. 1c). Using deterministic hidden units instead of stochastic hidden units allows us to perform efficient learning and inference (conditioned on a visible state). This joint learning significantly improves generative performance and also changes the representations learned at each level.

We consider models which are parameterized by an energy function $F(\mathbf{v})$ such that the probability of a (continuous) data vector $\mathbf{v}$ is $p(\mathbf{v}) = \frac{1}{Z}\exp(-F(\mathbf{v}))$, where Z is the partition function.

Let us denote by $g_\theta(\mathbf{v})$ the feedforward output of a neural network g_θ. Analogous to RBMs, the undirected connections between $\mathbf{g}_\theta(\mathbf{v})$ and the set of binary stochastic hidden units $\mathbf{h}$ (Fig. 1c) define an energy function

$$E(\mathbf{v}, \mathbf{h}) = \tfrac{1}{\sigma^2}\mathbf{v}^T\mathbf{v} + \mathbf{h}^T W \mathbf{g}_\theta(\mathbf{v}) - \mathbf{c}^T\mathbf{h} - \mathbf{b}^T\mathbf{v}. \quad (1)$$

Integrating out the binary hidden units gives us the free energy on $\mathbf{v}$

$$\begin{aligned} F(\mathbf{v}) &= \log \sum_{\mathbf{h}} p(\mathbf{v}, \mathbf{h}) \\ &= \tfrac{1}{\sigma^2}\mathbf{v}^T\mathbf{v} - \sum_i \log(1 + e^{w_i^T \mathbf{g}_\theta(\mathbf{v}) + c_i}) - \mathbf{b}^T\mathbf{v}. \quad (2) \end{aligned}$$

This model can be viewed as using a feedforward neural network to model the energy landscape that defines the probability distribution. Furthermore, by adding more layers to the feedforward neural network, we can potentially increase the representational power of the model, allowing it to model the input distribution more accurately. Notice that the conditional posteriors of the hidden variables are still easy to compute exactly: $p(h_i|\mathbf{v}) = \text{sigmoid}(\mathbf{w_i}^T\mathbf{g}_\theta(\mathbf{v}) + c_i)$. In particular, if we assume that g_θ is a sigmoidal neural network, then the *exact* conditional posterior of the network corresponds exactly to the form for *approximate* inference in a DBN.

More generally, this formulation provides an alternative method for training feedforward neural networks of arbitrary activation functions. Unsupervised training methods for neural networks have primarily used reconstruction as a learning objective (Bengio, 2007). In particular, stacked autoencoder models are trained layerwise by learning an autoencoder that reconstructs the previous layer. Training all layers jointly is also possible by treating the deep network as a single autoencoder; this consists of an encoder that computes a "hidden" representation and a decoder that reconstructs the data.

Unlike the autoencoder approach, we are able to train a deep feedforward neural network to optimize for the data likelihood directly. Furthermore, training using a reconstruction objective requires the learning of both an encoder and an explicit decoder. In comparison, our model has no requirements for an explicit decoder (decoding is done *implicitly* through inference). This potentially allows the model to discover representations that are highly invariant, as having invariant

features (e.g. to transformations of the input) might make the decoding ambiguous and could therefore be harder to achieve under the constraint of having to reconstruct the input.

2.1. Learning

We train our models by maximizing the log-likelihood of a training set (e.g. samples of natural image patches) using stochastic gradient ascent. Specifically, we learn the parameters of the feedforward network g_θ, variance parameter σ, weights W and biases $\mathbf{c}$. The derivative of the log-likelihood of the model over a training set D is given by:

$$\triangle\theta = \mathbf{E}_M\left[\frac{\partial F(\mathbf{v})}{\partial\theta}\right] - \mathbf{E}_D\left[\frac{\partial F(\mathbf{v})}{\partial\theta}\right] \tag{3}$$

where the first term represents an expectation of the partial derivative over the model distribution and the second an expectation over the data.

While the second term is straightforward to compute, the first term is usually difficult since it is often intractable to integrate over the model distribution. Thus, we approximate the first term using samples from the model distribution. More explicitly, we construct persistent Markov chains with "fast weights" (Tieleman & Hinton, 2009) from which we sample a new visible state after every parameter update step (Tieleman, 2008); this corresponds to a stochastic approximation for maximum likelihood training.

To sample from the model distribution, we employ the Hybrid Monte Carlo (HMC) sampler (Neal, 1993). HMC provides an efficient method to draw samples from the model distribution by performing a physical simulation of an energy-conserving system to generate proposal moves. In detail, we add "kinetic energy" variables, $\mathbf{p}$ for each visible variable in our model, while the model's energy function specifies a potential energy over the visible variables. Each HMC step involves sampling $\mathbf{p}$[1] and carrying out a physics-based simulation using "leap-frog" discretization. The final state of the simulation is accepted or rejected based on the Metropolis-Hastings algorithm.

This method of using the HMC sampler to estimate the gradient of the data log-likelihood is known as *contrastive backpropagation* (Teh et al., 2003; Hinton et al., 2004; 2006b).

2.2. Greedy layerwise with a deep objective

In this section, we introduce a variant of the standard greedy layerwise training approach that effectively allows pretraining of our model. Conventionally, greedy layerwise training proceeds by training additional layers to model the posteriors of the layer before. For DBNs, this was achieved by training an RBM to model the posteriors of the hidden units in the previous layer. We suggest an alternative approach: train the next layer to optimize for the *data* likelihood, but freeze the parameters of the earlier layers. Hence, even though only the parameters of the current layer are being modified during training, the learning objective is the data likelihood of the entire deep model.

This training procedure works well with the learning method outlined in Section 2.1; the same learning algorithm can be applied in a greedy layerwise fashion without significant change. Concretely, we consider greedy layerwise training for models where the feedforward network g_θ is parameterized by a network with sigmoidal activation functions. After training a network with l layers, we can "fold" the top hidden layer into the model by letting $\tilde{g}_\theta(\mathbf{v}) = p(\mathbf{h}|\mathbf{v})$, since the posteriors of the hidden units is computed in the same form as the other layers in the network. A next layer can then be trained using $\tilde{g}_\theta$ as the feedforward network.

2.3. Joint training for multiple layers

After greedy layerwise training, it is also easy to jointly train all layers of the model (at comparable computational cost). One simply unfreezes the weights of the previous layers while optimizing for the same objective function. This is in contrast to DBNs and DBMs where training all layers jointly usually involves an additional cost since it requires sampling all the hidden layers of the network. Our deep energy models do not require sampling the hidden units of intermediate layers since we use deterministic hidden units.

In practice, we find that mixing greedy layerwise steps together with joint training steps performed well in terms of the model convergence and quality of the resulting model. Concretely, for three layer networks, we suggest training the second layer with greedy layerwise training followed by joint training before stacking the third layer. We found that this training protocol results in DEM models that converge to better solutions.

3. General Deep Energy Models

While networks based on binary hidden units show promising results on natural images, using nonlinearities other than the sigmoid function in the feedforward neural network can potentially result in better generative models. In this section, we will describe the general form of DEMs and show how models such as the

[1] We sample $\mathbf{p} \sim N(0, I)$ as we used whitened data in our experiments.

product of Student-t (PoT) distributions and covariance RBMs (cRBMs) can be viewed as special cases of DEMs and be extended to have multiple layers of nonlinearities.

Let us revisit the energy function of our deep energy models and view it in a more general context:

$$F(\mathbf{v}) = \frac{1}{\sigma^2}\mathbf{v}^T\mathbf{v} + H(\mathbf{v}) - \mathbf{b}^T\mathbf{v} \tag{4}$$

The original deep energy models can be viewed as having $H(\mathbf{v}) = -\sum_i \log(1 + \exp(\mathbf{w_i}^T\mathbf{g}_\theta(\mathbf{v}) + c_i))$, where $g_\theta(\mathbf{v})$ is the output of a feedforward neural network. By relaxing this form and allowing functions other than soft-rectification, one can recover models such as the PoT model which corresponds to a choice of $H_{PoT}(\mathbf{v}) = \sum_i \alpha_i \log(1 + (\mathbf{w_i}^T\mathbf{v})^2)$.

As an another example, the cRBM corresponds to $H_{cRBM}(\mathbf{v}) = -\sum_i \log(1+\exp(\sum_f P_{if}(\mathbf{C_f}^T\mathbf{v})^2 + c_i))$, which can also be viewed as a two layer network where one first computes the squared responses of linear filters followed by a soft-rectification. Each of these choices of $H(\mathbf{v})$ give an $F(\mathbf{v})$ that is asymptotically dominated by the quadratic term $\mathbf{v}^T\mathbf{v}$, and therefore represents a normalizable probability distribution.

Recent work in extending these models have focused on linear combinations of different models. The mean-covariance RBM (mcRBM), for example, linearly combines the cRBM and RBM to jointly model both the mean and covariance of the inputs. Others have also tried stacking a DBN (Ranzato & Hinton, 2010; Dahl et al., 2010) over the learned features. These models and their extensions have been successfully applied to various modalities such as natural images and audio (Dahl et al., 2010), showing good results in denoising images (Welling et al., 2003) and in generating large realistic samples (Ranzato et al., 2010). However, further improving these models has not been straightforward.

We propose building deep versions of these models as an alternative to linear summations of existing models. The PoT and cRBM models can be viewed as models of the energy landscape using shallow feedforward networks with specific choices of activation functions. To create more expressive models, we can build deep networks in which each layer is essentially a replica of the PoT or cRBM models. In particular, we can parameterize the neural network $g_\theta(\mathbf{v})$ to have specific activation functions, e.g., $\log(1 + z^2)$ for a "Stacked PoT" (SPoT) model. In this way, a single layer SPoT model is equivalent to the PoT model. While we present the SPoT model as a specific instantiation of DEMs, we note that Mnih & Hinton (2005) had previously showed that learning a two layer SPoT model was possible on a small synthetic dataset.

The SPoT models can be learned using the same method presented in Section 2.1. The greedy layerwise training procedure (Section 2.2) remains the same, except that each new layer might no longer have the interpretation of being the posteriors of the hidden units.

4. Evaluation of Generative Models

We evaluate our generative models and training schemes by visually inspecting samples from the models, and by estimating the log-likelihood of the test data under each model. For the latter, we use a feedforward pass through the network to compute the unnormalized probability for the test data, and estimate the partition function Z with annealed importance sampling (AIS) (Neal, 1998; Salakhutdinov & Murray, 2008).

In our implementation of AIS, we successively draw samples from a sequence of "progressively harder" distributions $P_0, P_1, \ldots, P_K$, with P_K being our target distribution and P_0 a Gaussian baseline that we can easily sample from. We construct the interpolating distributions $P_1, \ldots, P_{K-1}$ by setting

$$F_k(\mathbf{v}) = (1 - \beta_k)F_0(\mathbf{v}) + \beta_k F_k(\mathbf{v}), \tag{5}$$

using HMC to draw samples from the canonical, unnormalized distribution $\tilde{p}_k(\mathbf{v}) = e^{-F_k(\mathbf{v})}$.[2]

5. Experiments on Natural Images

We trained our models on 200,000 randomly sampled natural images patches (16x16) (van Hateren & van der Schaaf, 1998), and randomly sampled another 10,000 image patches from a held-out set to use as a test set. PCA-whitening was used to reduce each example to a 142 dimension vector.

For these experiments, we considered the standard DEM network with sigmoidal units (Sigmoid-DEM) and the SPoT models. The models were trained with greedy-layerwise mixed with joint optimization. We denote models that were trained with greedy layerwise stacking as M1, M2 and M3. Similarly, the models that were trained with jointly are denoted as M12 and M123. To distinguish between training methods, we further denote the models based on their training

[2] In our experiments, we fit the baseline Gaussian P_0 to 10,000 samples from the model and let $K = 28,000$, with $\beta_1, \ldots, \beta_{3000}$ uniformly spaced in $[0, 0.5]$, $\beta_{3001}, \ldots, \beta_{12000}$ in $[0.5, 0.9]$, $\beta_{12001}, \ldots, \beta_{20000}$ in $[0.9, 0.98]$, and $\beta_{20001} \ldots, \beta_{28000}$ in $[0.98, 1.0]$. Z is then estimated by taking the average of 3000 AIS runs.

Table 1. Generative performance of models on natural image patches. Joint training of the models significantly improves the performance of the models. AIS was used to estimate the partition function $\hat{Z}$ of each model; the standard deviations of these estimates are provided (left value shows $(\hat{Z} - 3\hat{\sigma})$, right shows $(\hat{Z} + 3\hat{\sigma})$).

Model	AIS Log-likelihood	
	Test	$\log(\hat{\mathbf{Z}} \pm \mathbf{3}\hat{\sigma})$
SPoT M1	-8.7	-196.9, -196.9
SPoT M1-M2	2.2	-681.4, -681.4
SPoT M1-M2-M12	7.2	-1056.5, -1056.4
SPoT M1-M2-M3	1.13	-2257.2, -2257.2
SPoT M1-M2-M12-M3	5.3	-3618.4, -3618.4
Sigmoid-DEM M1	-49.2	339.3, 339.4
Sigmoid-DEM M1-M2	-57.5	322.6, 322.6
Sigmoid-DEM M1-M2-M12	-2.3	372.3, 372.9
mcRBM	-5.5	378.7, 378.7

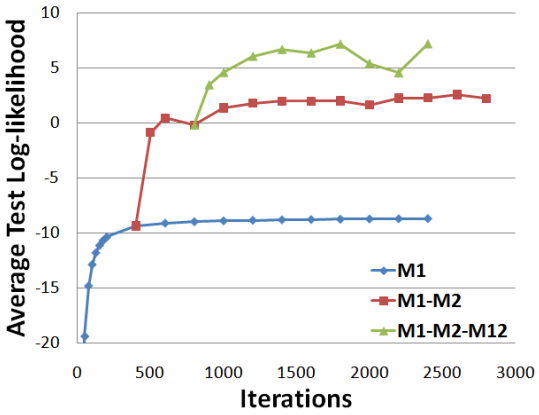


Figure 2. This graph shows the test log-likelihood of the SPoT model over iterations. The first layer model quickly plateaus and adding the second layer improves performance slightly. Joint training resulted in a further improvement. A similar result holds for the Sigmoid-DEM networks.

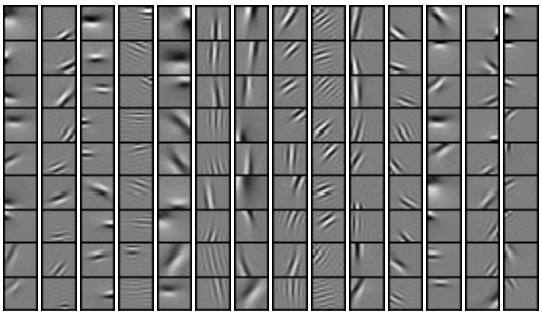


Figure 3. Visualization of second layer in a SPoT M1-M2-M12 model. Each sorted column shows the most strongly connected first layer units to a particular second layer unit. We find units that pool together similar first layer features. Some units are selective for particular frequencies while others are selective for orientation and position.

sequence; for instance, a model denoted by M1-M2-M12-M3 refers to a model that was trained greedy-layerwise for two layers, followed by joint training of the two layers and finally greedy-layerwise stacking of a third layer on top. We trained all models with the same learning parameters.[3]

Table 1 shows the average test log-likelihoods of each model at convergence. We omit the results for three-layered models with joint training (e.g. M1-M2-M3-M123), as AIS returned variable and unreliable estimates for these highly multimodal models. Fig. 2 shows how SPoT model training progresses stably over iterations. From this graph, we can see that joint training of the first two layers can result in a significant improvement in performance over a purely greedy-layerwise strategy, even when the latter is run for a large number of iterations. Furthermore, training a two layer SPoT model jointly results in a better generative model than simply adding a third layer with greedy layerwise training.

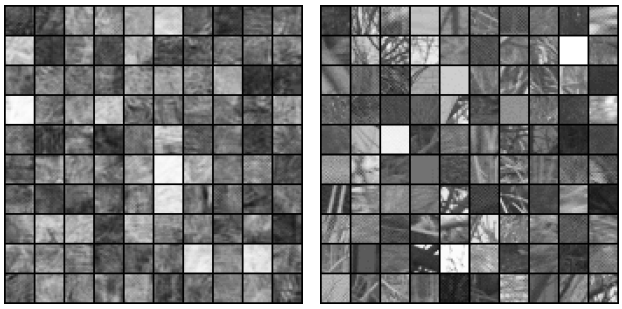
Compared to the mcRBM (Ranzato & Hinton, 2010), the one layer SPoT model (M1) does worse but the deeper models outperform the mcRBM. This shows that adding additional layers can be more beneficial than simply adding mean units to a covariance model.[4]

To understand the effects of the second layer in the M12 model, we visualized the most strongly connected first layer units to the second layer (Fig. 3). We found a diverse set of second layer including units that were selective and invariant. Some units were selective to location and orientation but not frequency, while there were also others which were only selective to frequency.

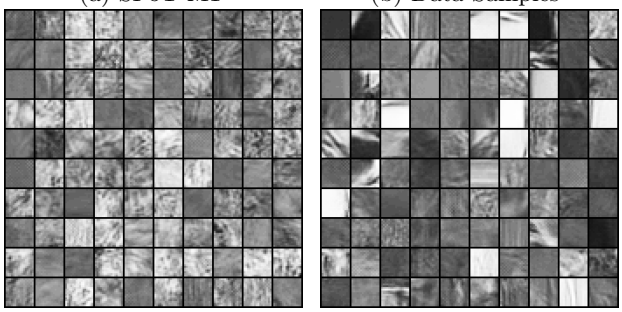
We also sampled from the models to compare them as suggested by Ranzato et al. (2010). The visual quality of the samples improved with the deeper models, showing more structured samples that appear qualitatively closer to natural image patches (Fig. 4).

[3]We used stochastic gradient ascent with mini-batches of 200 examples, a learning rate of 0.001 and momentum that was slowly increased from 0.5 to 0.9. When running the HMC sampler, we dynamically adapt the step-size of the "leap-frog" discretization to maintain a rejection rate of 10% and used 20 "leap-frog" steps for each HMC step.

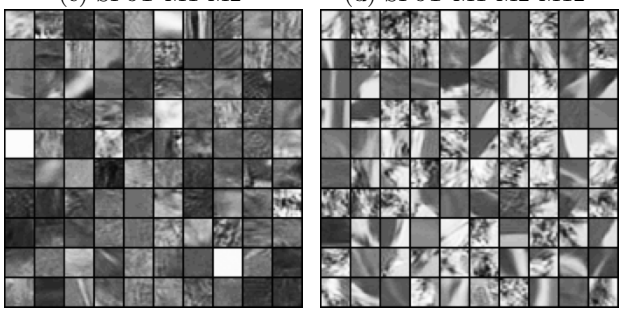
[4]The SPoT M1 model is similar to a cRBM (which is a mcRBM without the mean units), in the sense that both can be viewed as models of the covariance structure with hidden units from different distributions.



(a) SPoT M1 (b) Data Samples



(c) SPoT M1-M2 (d) SPoT M1-M2-M12



(e) SPoT M1-M2-M12-M3 (f) SPoT M1-M2-M12-M3-M123

Figure 4. Samples from the generative models. After training the model, we ran the HMC sampler for 500 burn-in iterations with random initialization to obtain samples from the model distribution. The sample quality closely follows the log-likelihood estimates from Table 1.

6. Discriminative Deep Energy Models

One application of learning a generative model of data is to find features that are useful for tasks such as image classification (Hinton et al., 2006a; Ranzato & Hinton, 2010; Larochelle & Bengio, 2008). In this section, we present an discriminative extension of our model that is trained using a hybrid generative-discriminative learning scheme. Specifically, we use the activations of *every layer* in g_θ as features which are used to learn a linear classifier for some associated labels $\mathbf{y}$, with weights U. These features specify a joint energy between the inputs $\mathbf{v}$ and labels $\mathbf{y}$:

$$E(\mathbf{v},\mathbf{y}) = \frac{1}{\sigma^2}\mathbf{v}^T\mathbf{v} + H(\mathbf{v}) - \mathbf{b}^T\mathbf{v} - \mathbf{d}^T\mathbf{y} - \sum_l \mathbf{a}_l^T U_l \mathbf{y} \tag{6}$$

where $\mathbf{a}_l$ are the activations of the l^{th} layer of g_θ and $\mathbf{y}$ is a one-hot vector of image labels. Analogously, this energy function defines a probability distribution $p(\mathbf{v},\mathbf{y}) \propto \exp(-E(\mathbf{v},\mathbf{y}))$ over $\mathbf{v}$ and $\mathbf{y}$.

We are able to train this model generatively by integrating out $\mathbf{y}$ when sampling $\mathbf{v}$: $F(\mathbf{v}) = -\log(\sum_{\mathbf{y}} \exp^{E(\mathbf{v},\mathbf{y})})$, analogous to implicit mixtures of RBMs (Nair & Hinton, 2008).

We train the model using a hybrid generative-discriminative objective:

$$\arg\max_\theta \left((1-\alpha)\ell(y|x;\theta) + \alpha\ell(y,x;\theta)\right) \tag{7}$$

where $\ell(y|x;\theta)$ is the log-likelihood of the labels given the data (discriminative cost) and $\ell(y,x;\theta)$ is the log-likelihood of the joint distribution of the labels and data (generative cost).

The gradient for the generative cost is computed in a similar fashion to Equation 3. For the discriminative cost, we are able to compute the gradients exactly since the model corresponds to a (short-circuited) feed-forward neural network with softmax classification.

7. Experiments on Object Recognition

In this section, we evaluate our models on object recognition. We used the *normalized-uniform* set of NORB (LeCun et al., 2004) which has 24,300 training and 24,300 testing examples of 5 object categories (cars, planes, people, animals and trucks). Each example contained two 96x96 images corresponding to a stereoscopic view of the object. We downsampled each image to 32x32 and used PCA-whitening to further reduce each example to a 176 dimensional vector.

We trained the generative SPoT models with 600 hidden units on each layer. Similar to the results in Section 5, we found that the deep models out perform the shallow models the samples generated from the model (Fig. 5) also closely resemble the objects from the dataset. We also trained the hybrid discriminative-generative models on NORB, cross validating on a subset of the training data to find the optimal α parameter that weights the discriminative-generative costs.[5]

For the hybrid discriminative-generative SPoT model, we obtained a test accuracy of 93.8% with $\alpha = 0.25$. In comparison, the fully discriminative SPoT model overfits and only achieves a test accuracy of 85.7%. This shows that regularizing the model to be a generative model significantly helps the model to generalize

[5]In practice, we found that using an SVM with the learned features performed slightly better and we report these results.

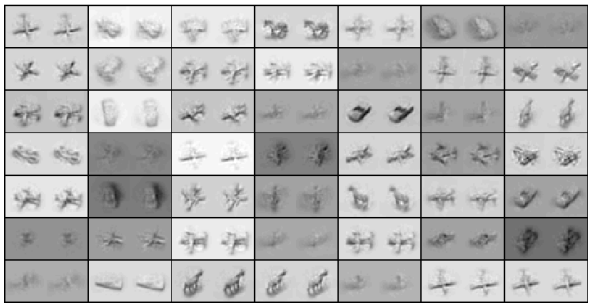


Figure 5. Samples from SPoT M12 model trained on the NORB dataset.

beyond the dataset. We also performed a control experiment where we used random initializations as network parameters, as suggested by Saxe et al. (2011). This gives a test accuracy of $74.4\% \pm 2.0\%$, showing that the learning algorithm is responsible for the good performance. We highlight that our model is a fully connected model which does not assume any structure in the input data, unlike convolution based approaches which hold the current state-of-the-art performances on classification (97.2%, Huang & LeCun (2006); Le et al. (2010); Coates et al. (2011)).

8. Discussion

8.1. Evaluation of generative models

While AIS has had success in recent related work (Salakhutdinov & Murray, 2008; Salakhutdinov & Larochelle, 2010), it is typically hard to determine how accurate AIS estimates are. Instead, it is common to gauge the consistency of AIS estimation by finding the variance of $\hat{Z}$ across different AIS estimates. However, consistency does not necessarily imply accuracy (Neal, 1998). Fortunately, our results (Fig. 2) suggest that AIS estimation is functioning as intended. First, the estimated data log-likelihood consistently increases with the number of training iterations, and converges reasonably well. Second, data likelihood increases with the addition of a second layer, and with the commencement of joint training of the first and second layers. Third, the relative changes in AIS log-likelihood estimates across models match up with the visual quality of the filters (Fig. 6) and samples (Fig. 4) from each model.

Another evaluation method used in the literature is Parzen window density estimation. We implemented Parzen estimation using the protocol described by Desjardins et al. (2010), with 10,000 samples per model. However, Parzen estimates for the average test log-likelihood were lower than AIS estimates by ~ 100 across all models. Moreover, while AIS returned log-likelihoods that increased smoothly with training iterations, Parzen estimates were largely inconsistent across iterations. These discrepancies can be attributed to the fact that accurate Parzen estimation in this high-dimensional space needs a prohibitively large number of samples.

Table 2. Classification test accuracies using fully-connected methods on the NORB object recognition dataset.

Method	Accuracy
SPoT M123 Discriminative Only	85.7%
SPoT M123 Hybrid Training	93.8%
3D DBNs (Nair & Hinton, 2009)	93.5%
DBMs (Salakhutdinov & Hinton, 2009)	92.8%
SVMs (Bengio & LeCun, 2007)	88.4%

8.2. Scaling up

One obstacle to scaling the SPoT model to realistic datasets is the computational cost involved in using the HMC sampler. HMC allows us to sample from arbitrary energy functions. However, the HMC sampler can be considerably slower than Gibbs sampling methods (e.g., for training RBMs) since the HMC sampler has to evaluate the gradient of the energy function multiple times for a single Monte Carlo step. Fortunately, we were able to use GPUs with Jacket[6] to speed up our computations significantly. The use of GPUs for training deep models was proposed by Raina et al. (2009). A potential alternative to sampling-based learning is to use score matching to train our models; for example, Kingma & LeCun (2010) showed how score matching can be applied to arbitary energy functions while Köster & Hyvärinen (2010) showed how to train both layers of a two layer ICA-based model.

8.3. Effect of Joint Training

In our quantitative results, we found that joint training of multiple layers significantly improved the performance of the model. To examine the qualitative aspects of training, we first trained a single layer Sigmoid-DEM and visualized the learned representations (Fig. 6-Left). We then used the single layer model to initialize a two layer model, which was trained jointly. By stacking another layer and learning both layers together, we see a dramatic change in the first layer weights of the model (Fig. 6-Right). The first layer weights appear to be blob-like initially but change to Gabor-like filters when jointly trained with a second layer. This indicates that generative training for multiple layers at a time can be crucial for learning meaningful representations in all layers.

[6] http://www.accelereyes.com/

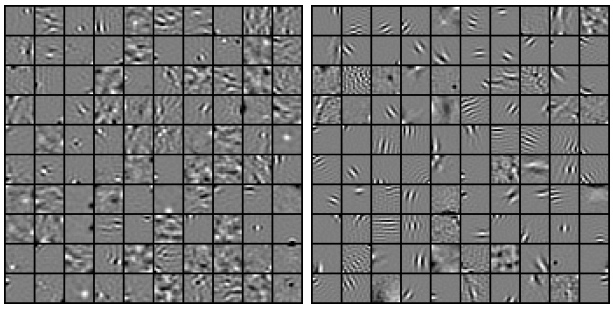


Figure 6. Left-Layer 1 filters from a single layer Sigmoid-DEM model (equivalent to an RBM). Right-Same filters after joint training as part of a two layer model. While the initial filters from the single layer model appear to be mostly blob-like, learning two layers jointly result in filters that appear more Gabor-like. Training the single layer model for more iterations did not significantly change the qualitative appearance of the filters.

Our results suggest that it is possible to obtain significantly better models by jointly training all layers for deep models, instead of only relying on greedy-layerwise stacking.

Acknowledgments

This work was supported by the DARPA Deep Learning program under contract number FA8650-10-C-7020.

References

Bengio, Y. Learning deep architectures for AI. Technical report, Université de Montréal, 2007.

Bengio, Y. and LeCun, Y. Scaling learning algorithms towards ai. In *Large-Scale Kernel Machines*, 2007.

Coates, A., Lee, H., and Ng, A. Y. An analysis of single-layer networks in unsupervised feature learning. In *AISTATS*, 2011.

Dahl, G. E., Ranzato, M., Mohamed, A., and Hinton, G. E. Phone recognition with the mean-covariance restricted Boltzmann machine. In *NIPS*. 2010.

Desjardins, G., Courville, A., and Bengio, Y. Parallel tempering for training of restricted boltzmann machines. In *AISTATS*, 2010.

Hinton, G. E., Welling, M., and Mnih, A. Wormholes improve contrastive divergence. In *NIPS*, 2004.

Hinton, G. E., Osindero, S., and Teh, Y. W. A Fast Learning Algorithm for Deep Belief Nets. *Neural Computation*, 18(7):1527–1554, July 2006a.

Hinton, G. E., Osindero, S., Welling, M., and Teh, Y. W. Unsupervised Discovery of Non-linear Structure using Contrastive Backpropagation. *Cognitive Science*, 30(4): 725–731, 2006b.

Huang, F. J. and LeCun, Y. Large-Scale Learning with SVM and Convolutional Nets for Generic Object Categorization. In *CVPR*, 2006.

Kingma, D. P. and LeCun, Y. Regularized estimation of image statistics by score matching. In *NIPS*, 2010.

Köster, U. and Hyvärinen, A. A two-layer model of natural stimuli estimated with score matching. *Neural Computation*, 22(9):2308–2333, 2010.

Larochelle, H. and Bengio, Y. Classification using discriminative restricted boltzmann machines. In *ICML*, 2008.

Le, Q. V., Ngiam, J., Chen, Z., Chia, D., Koh, P. W., and Ng, A. Y. Tiled convolutional neural networks. In *NIPS*, 2010.

LeCun, Y., Huang, F.J., and Bottou, L. Learning methods for generic object recognition with invariance to pose and lighting. In *CVPR*, 2004.

Mnih, A. and Hinton, G. E. Learning nonlinear constraints with contrastive backpropagation. In *IJCNN*, 2005.

Nair, V. and Hinton, G. E. Implicit mixtures of restricted boltzmann machines. In *NIPS*, 2008.

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

Neal, R. M. Probabilistic inference using markov chain Monte Carlo methods. Technical report, University of Toronto, 1993.

Neal, R. M. Annealed Importance Sampling. *Statistics and Computing*, 11:125–139, 1998.

Raina, R., Madhavan, A., and Ng, A. Y. Largescale deep unsupervised learning using graphics processors. In *ICML*, 2009.

Ranzato, M. and Hinton, G. E. Modeling Pixel Means and Covariances Using Factorized Third-Order Boltzmann Machines. In *CVPR*, 2010.

Ranzato, M., Mnih, V., and Hinton, G. E. Generating more realistic images using gated mrf's. In *NIPS*. 2010.

Salakhutdinov, R. and Hinton, G. E. Deep Boltzmann machines. In *AISTATS*, 2009.

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

Salakhutdinov, R. and Murray, I. On the quantitative analysis of deep belief networks. In *ICML*, 2008.

Saxe, A., Koh, P.W., Chen, Z., Bhand, M., Suresh, B., and Ng, A. On random weights and unsupervised feature learning. In *ICML*, 2011.

Teh, Y., Welling, M., and Osindero, S. Energy-based models for sparse overcomplete representations. *JMLR*, 4: 1235–1260, 2003.

Tieleman, T. Training Restricted Boltzmann Machines using Approximations to the Likelihood Gradient. In *ICML*, 2008.

Tieleman, T. and Hinton, G. E. Using fast weights to improve persistent contrastive divergence. In *ICML*, 2009.

van Hateren, J. H. and van der Schaaf, A. Independent component filters of natural images compared with simple cells in primary visual cortex. *Proceedings: Biological Sciences*, 265(1394):359–366, 1998.

Welling, M., Hinton, G. E., and Osindero, S. Learning sparse topographic representations with products of student-t distributions. In *NIPS*, 2003.