A Two-stage Pretraining Algorithm for Deep Boltzmann Machines

KyungHyun Cho, Tapani Raiko, Alexander Ilin and Juha Karhunen
Aalto University School of Science
Department of Information and Computer Science
Espoo, Finland
firstname.lastname@aalto.fi

Abstract

A deep Boltzmann machine (DBM) is a recently introduced Markov random field model that has multiple layers of hidden units. It has been shown empirically that it is difficult to train a DBM with approximate maximum-likelihood learning using the stochastic gradient unlike its simpler special case, restricted Boltzmann machines (RBM). In this paper, we propose a novel pretraining algorithm that consists of two stages; obtaining approximate posterior distributions over hidden units from a simpler model and maximizing the variational lower-bound given the fixed hidden posterior distributions. We show empirically that the proposed method overcomes the difficulty in training DBMs from randomly initialized parameters and results in a better, or comparable, generative model when compared to the conventional pretraining algorithm.

1 Introduction

Deep Boltzmann machine (DBM), proposed by Salakhutdinov and Hinton [2009], is a recently introduced variant of Boltzmann machines which extends widely used restricted Boltzmann machines (RBM) to a model that has multiple hidden layers. It differs from the popular deep belief network (DBN) which is built by stacking multiple layers of RBMs [Hinton and Salakhutdinov, 2006] in that every edge in the DBM model is undirected. In this way, DBMs facilitate propagating uncertainties across multiple layers of hidden variables.

Although it is straightforward to derive a learning algorithm for DBMs using a variational approximation and stochastic maximum likelihood method, recent research [see for example Salakhutdinov and Hinton, 2009] has shown that learning the parameters of DBMs is not trivial. Especially the generative performance of the trained model, commonly measured by the variational lower-bound of log-probabilities of test samples, tends to degrade as more hidden layers are added.

Salakhutdinov and Hinton [2009] proposed a greedy layer-wise pretraining algorithm that could be used to initialize parameters of DBMs, and showed that it largely overcomes the difficulty of learning a good generative model.

Along this line of research, we propose another way to approach pretraining DBMs in this paper. The proposed scheme is based on an observation that training DBMs consists of two separate stages; approximating a posterior distribution over the states of hidden units and updating parameters to maximize the lower-bound of log-likelihood given those states.

Based on this observation, our proposed method pretrains a DBM in two stages. During the first stage we train a simpler, directed deep model such as DBNs or stacked denoising autoencoders (sDAE) to obtain an approximate posterior distribution over hidden units. With this fixed approximate posterior distribution, we train an RBM that learns a distribution over a combination of data

samples and their corresponding posterior distributions of hidden units. Finetuning the model is then trivial as one only needs to free hidden variables from the approximate posterior distribution computed during the first stage.

We show that the proposed algorithm helps learning a good generative model which is empirically comparable to the pretraining method proposed by Salakhutdinov and Hinton [2009]. Furthermore, we discuss the potential degrees of freedom in extending the proposed approach.

2 Deep Boltzmann Machines

We start by describing deep Boltzmann machines (DBM) [Salakhutdinov and Hinton, 2009]. A DBM with L layers of hidden neurons is defined by the following energy function:

$$-E(\mathbf{v}, \mathbf{h} \mid \boldsymbol{\theta}) = \sum_i v_i b_i + \sum_{i,j} v_i h_j^{(1)} w_{i,j} + \sum_j h_j^{(1)} c_j^{(1)} + \sum_{l=2}^{L} \left(\sum_j h_j^{(l)} c_j^{(l)} + \sum_{j,k} h_j^{(l)} h_k^{(l+1)} u_{j,k}^{(l)} \right), \tag{1}$$

where $\mathbf{v} = [v_i]_{i=1\ldots N_v}$ and $\mathbf{h}^{(l)} = \left[h_j^{(l)}\right]_{j=1\ldots N_l}$ are N_v binary visible units and N_l binary hidden units in the l-th hidden layer. $\mathbf{W} = [w_{i,j}]$ is the set of weights between the visible neurons and the first layer hidden neurons, while $\mathbf{U}^{(l)} = \left[u_{j,k}^{(l)}\right]$ is the set of weights between the l-th and $l+1$-th hidden neurons.

With the energy function, a DBM can assign a probability to each state vector $\mathbf{x} = [\mathbf{v}; \mathbf{h}^{(1)}; \cdots; \mathbf{h}^{(L)}]$ using a Boltzmann distribution:

$$p(\mathbf{x} \mid \boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \exp\left\{-E(\mathbf{x} \mid \boldsymbol{\theta})\right\}.$$

Based on this property the parameters can be learned by maximizing the log-likelihood $\mathcal{L} = \sum_{n=1}^{N} \log \sum_{\mathbf{h}} p(\mathbf{v}^{(n)}, \mathbf{h} \mid \boldsymbol{\theta})$ given N training samples $\{\mathbf{v}^{(n)}\}_{n=1,\ldots,N}$, where $\mathbf{h} = \left[\mathbf{h}^{(1)}; \cdots; \mathbf{h}^{(L)}\right]$.

The gradient computed by taking the partial derivative of the log-likelihood function with respect to each parameter is used in most cases with a mini-batch per update. It is then used to update the parameters, effectively forming a stochastic gradient ascent method. A standard way of computing gradient results in the following update rule for each parameter θ:

$$\nabla\theta = \left\langle \frac{\partial E(\mathbf{v}^{(n)}, \mathbf{h} \mid \boldsymbol{\theta})}{\partial \theta} \right\rangle_{\mathrm{m}} - \left\langle \frac{\partial E(\mathbf{v}, \mathbf{h} \mid \boldsymbol{\theta})}{\partial \theta} \right\rangle_{\mathrm{d}}, \tag{2}$$

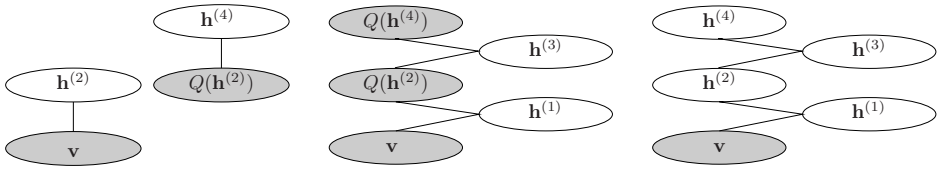
where $\langle\cdot\rangle_{\mathrm{d}}$ and $\langle\cdot\rangle_{\mathrm{m}}$ denote the expectation over the data distribution $P(\mathbf{h} \mid \{\mathbf{v}^{(t)}\}, \boldsymbol{\theta})$ and the model distribution $P(\mathbf{v}, \mathbf{h} \mid \boldsymbol{\theta})$, respectively.

3 Training Deep Boltzmann Machines

Although the update rules (2) are well defined, it is intractable to exactly compute them. Hence, an approach that uses variational approximation together with Markov chain Monte Carlo (MCMC) sampling was proposed by Salakhutdinov and Hinton [2009].

First, the variational approximation is used to compute the expectation over the data distribution. It starts by approximating $p(\mathbf{h} \mid \mathbf{v}, \boldsymbol{\theta})$, which is intractable unless $L = 1$, by a factorial distribution $Q(\mathbf{h}) = \prod_{l=1}^{L} \prod_{j=1}^{N_l} \mu_j^{(l)}$. The variational parameters $\mu_j^{(l)}$ can then be estimated by the following fixed-point equation:

$$\mu_j^{(l)} \leftarrow f\left(\sum_{i=1}^{N_{l-1}} \mu_i^{(l-1)} w_{ij}^{(l-1)} + \sum_{k=1}^{N_{l+1}} \mu_k^{(l+1)} w_{kj}^{(l)} + c_j^{(l)} \right), \tag{3}$$



(a) Stage 1 (b) Stage 2 (c) Finetuning

Figure 1: Illustration of the two-stage pretraining algorithm followed by finetuning of all parameters.

where $f(x) = \frac{1}{1+\exp\{-x\}}$. Note that $\mu_i^{(0)} = v_i$ and the update rule for the top layer does not contain the second summation term, that is $N_{l+1} = 0$.

This variational approximation provides the values of variational parameters that maximize the following lower-bound with respect to the current parameters:

$$p(\mathbf{v} \mid \boldsymbol{\theta}) \geq \mathbb{E}_{Q(\mathbf{h})} \left[-E(\mathbf{v}, \mathbf{h})\right] - \mathcal{H}(Q) - \log Z(\boldsymbol{\theta}), \tag{4}$$

where

$$\mathcal{H}(Q) = \sum_{l=1}^{L} \sum_{j=1}^{N_l} \left(\mu_j^{(l)} \log \mu_j^{(l)} + (1 - \mu_j^{(l)}) \log(1 - \mu_j^{(l)}) \right)$$

is an entropy functional. Hence, each gradient update step does not increase the exact log-likelihood but its variational lower-bound. A tighter lower-bound could be used by explicitly summing out the hidden units in the odd numbered hidden layers.

Second, the expectation over the model distribution is computed by persistent sampling. By persistent sampling, we mean that we do not wait for a sampling chain to converge before computing each update direction, but run the chain for only a few steps and continue using the same chain over consecutive updates. The most naive approach is to use Gibbs sampling, while there have been some work in applying more advanced sampling methods [Salakhutdinov, 2010, 2009, Desjardins et al., 2010, Cho et al., 2010]. In this paper, we use coupled adaptive simulated tempering (CAST) recently proposed in [Salakhutdinov, 2010].

This approach closely resembles variational expectation-maximization (EM) algorithm [see for example Bishop, 2007]. Learning proceeds by alternating between finding the variational parameters $\boldsymbol{\mu}$ and updating the DBM parameters to maximize the given variational lower-bound using the stochastic gradient method. However, it has been known and will be shown in the experiments in this paper that training a DBM using this approach starting from randomly initialized parameters is not trivial [Salakhutdinov and Hinton, 2009, Desjardins et al., 2012, Cho et al., 2011a].

Hence, Salakhutdinov and Hinton [2009] proposed a pretraining algorithm to initialize the parameters of DBMs. The pretraining algorithm greedily trains each layer of a DBM by considering each layer as an RBM, following a pretraining approach used for training deep belief networks (DBN) [Hinton and Salakhutdinov, 2006]. However, due to the undirectedness of edges in DBMs it has been proposed to use the first layer RBM with two duplicate copies of visible units with tied weights and the last layer RBM with two duplicate copies of hidden units with tied weights. Once one layer has been trained, another layer can be trained on the aggregate prior distribution of the hidden units of the lower layer to extend the depth. After the pretraining, learned weights are used as initializations of weights of DBMs.

4 Restricted Boltzmann Machines and Denoising Autoencoders

Here we briefly discuss restricted Boltzmann machines (RBM) and single-layer denoising autoencoders (DAE) which will constitute an important part of the two-stage pretraining algorithm that will be described in the next section.

An RBM is a special case of DBMs, where the number of hidden layers is restricted to one, $L = 1$. Due to this restriction it is possible to compute the posterior distribution over the hidden units conditioned on the visible neurons exactly and tractably. The conditional probability of each hidden

unit $h_j = h_j^{(1)}$ is

$$p(h_j = 1 \mid \mathbf{v}, \boldsymbol{\theta}) = f\left(\sum_i w_{ij} v_i + c_j\right). \tag{5}$$

Hence, the positive part of the gradient in (2) can be computed exactly and efficiently. However, the negative part, which is computed over the model distribution, still relies on persistent sampling, or more approximate methods such as contrastive divergence (CD) [Hinton, 2002].

A single-layer DAE is a special form of multi-layer perception network [Haykin, 2009] with a single hidden layer and a tied set of weights. A DAE tries to learn a network that reconstructs an input vector optimally by minimizing the following cost function:

$$\sum_{n=1}^{N} \left\| g_v\left(\mathbf{W}^\top g_h\left(\mathbf{W}\phi(\mathbf{v}^{(n)})\right)\right) - \mathbf{v}^{(n)} \right\|^2, \tag{6}$$

where $g_h(\cdot)$ and $g_v(\cdot)$ are component-wise nonlinear functions, usually logistic sigmoid functions. Unlike an ordinary autoencoder, a DAE explicitly sets some of the components of an input vector randomly to zero during learning via $\phi(\cdot)$ which explicitly adds noise to an input vector.

There is an important difference in training DAEs compared with training RBMs. In DAEs, the objective of learning is not to learn a distribution but to minimize the reconstruction error. It usually leads to easier learning.

These two models are significant because they can be stacked to form more powerful models that learn hierarchical features. A deep belief network (DBN) is constructed by stacking multiple layers of RBMs, and a stacked denoising autoencoder (sDAE) can be built by stacking DAEs on top of each other. With probabilistic interpretation, one may consider these stacked models as having multiple layers of latent random variables where their posterior distributions can be computed by iteratively obtaining (approximate) posterior means of the hidden units layerwise.

5 A Two-stage Pretraining Algorithm

In this paper, we propose an alternative way of initializing parameters of a DBM compared with the one described at the end of Sec. 3. Unlike the conventional pretraining strategy described at the end of the previous section, we employ an approach that separates obtaining (approximate) posterior distributions over hidden units and initializing parameters.

Before proceeding to the description of the proposed algorithm, we first divide the hidden layers of a DBM into two sets. Let us denote a vector of hidden units in the odd-numbered layers as $\mathbf{h}_+$ and the respective vector in the even-numbered layers as $\mathbf{h}_-$. Note that due to the structure of DBMs, it is possible to explicitly sum out $\mathbf{h}_+$, which reduces the space of variational parameters to half. In this sense we may define $\boldsymbol{\mu}_-$ and $\boldsymbol{\mu}_+$ as variational parameters of the hidden units in the odd-numbered layers and the even-number layers, respectively.

5.1 Stage 1

We focus on finding a good set of variational parameters $\boldsymbol{\mu}_-$ of $Q(\mathbf{h}_-)$ that has a potential to give a reasonably high variational lower-bound in Eq. (4). In other words, we propose to first find a good posterior distribution over hidden units given a visible vector regardless of parameter values of a DBM. Although it might sound unreasonable to find a good set of variational parameters without any fixed parameter values, we can do this by *borrowing* (approximate) posterior distributions over latent variables from other models.

DBNs and sDAEs described in Sec. 4 are natural choice to find a good approximate posterior distribution over units in the even-numbered hidden layers. One justification for using either of them is that they can be trained efficiently and well [see, e.g. Bengio et al., 2012, and references therein]. It becomes a trivial task as one can iteratively train each even-numbered layer as either an RBM or a DAE on top of each other, as is a common practice when a DBN or a sDAE is trained.

5.2 Stage 2

Once a set of initial variational parameters $\boldsymbol{\mu}_-$ is found from a DBN or an sDAE, we train a model that has predictive power of the variational parameters given a visible vector. It can be simply done by learning a joint distribution of $\mathbf{v}$ and $\boldsymbol{\mu}_-$ using an RBM.

The structure of the RBM can be directly derived from the DBM such that its visible layer corresponds to the visible layer and the even-numbered hidden layers of the DBM and its hidden layer to the odd-numbered hidden layers of the DBM. The connections between them can also follow those of the DBM. This corresponds to finding a set of DBM parameters that *fit* the variational parameters obtained in the first stage.

Another way to look at what happens during the second stage is to consider what an RBM has been trained for. It is easy to see that training the RBM at the second stage maximizes

$$\log \sum_{\mathbf{h}_+} \exp\left\{-E(\mathbf{v}, \boldsymbol{\mu}_-, \mathbf{h}_+)\right\} - \log Z(\boldsymbol{\theta}).$$

which corresponds to maximizing the variational lower-bound of a DBM given in (4) given the variational parameters $\boldsymbol{\mu}_-$ which were found in the first stage.

Once an RBM has been trained, we can use the learned parameters as initializations for training the DBM, which corresponds to freeing $\mathbf{h}_-$ from its variational posterior distribution obtained in the first stage. Finetuning of the initialized parameters can be performed according to the standard procedure given in Sec. 3.

A simple illustration of the two-stage pretraining algorithm is given in Fig. 1.

	DBM	$\text{DBM}^{\text{sDAE}}_{\text{RBM}}$	$\text{DBM}^{\text{DBN}}_{\text{RBM}}$	$\text{DBM}^{\text{S\&H}}$	$\text{DBM}^{\text{S\&H}}_{\text{RBM}}$	$\text{DBM}^{\text{S\&H}}_{\text{FVBM}}$
L2s	-	-90.60	-90.49	-259.06	-305.49	-301.78
L2	-	-91.60	-92.06	-237.61	-296.88	-269.46
L3	-	-90.91	-93.50	-230.00	-291.14	-264.68
L4	-	-101.61	-103.35	-230.46	-302.05	-265.37

Table 1: Log-probabilities of the test samples of MNIST after pretraining.

	DBM	$\text{DBM}^{\text{sDAE}}_{\text{RBM}}$	$\text{DBM}^{\text{DBN}}_{\text{RBM}}$	$\text{DBM}^{\text{S\&H}}$	$\text{DBM}^{\text{S\&H}}_{\text{RBM}}$	$\text{DBM}^{\text{S\&H}}_{\text{FVBM}}$	(S)
L2s	-131.07	-84.49	-87.58	-93.98	**-79.28**	-80.28	
L2	-143.59	-85.44	-86.17	-92.03	**-79.03**	-80.45	-84.62
L3	-128.72	**-81.84**	-86.03	-93.83	-82.28	-83.86	-85.10
L4	-128.70	**-83.25**	-91.19	-96.20	-85.23	-86.44	

Table 2: Log-probabilities of the test samples of MNIST after finetuning. (S) - reported by Salakhutdinov and Hinton [2010].

5.3 Discussion

It is quite easy to see that the proposed algorithm has high degree of freedom to plug-in alternative algorithms and models in both stages.

The most noticeable flexibility can be found in stage 1. Any other machine learning model that gives reasonable posterior distributions over multiple layers of binary hidden units can be used instead of RBMs or DAEs. For instance, recently proposed variants of RBMs such as spike-and-slab RBMs [Courville et al., 2011] could directly be applied. Also, instead of stacking each layer at a time, one could opt to train a deep autoencoders at once using advanced backpropagation algorithms [Raiko et al., 2012, Martens, 2010, Schulz and Behnke, 2012].

In stage 2, one may opt to use a DAE instead of an RBM. It will make learning faster and therefore leaves more time for finetuning the model afterward. Also, the use of different kinds of algorithms for training an RBM can be considered. For quicker pretraining, one may use CD with only a small number of Gibbs sampling steps per update, or for better initial models, tempering-based MCMC

sampling methods such as parallel tempering [Desjardins et al., 2010, Cho et al., 2010] or tempered transition [Salakhutdinov, 2009] could be used.

Another obvious possibility is to utilize the conventional pretraining algorithm proposed by Salakhutdinov and Hinton [2009] during the first stage. This approach gives approximate posterior distributions over all hidden units $[\mathbf{h}_-; \mathbf{h}_+]$ as well as initial values of the parameters that can be used during the second stage. In this way, one may use either an RBM or a fully visible Boltzmann machine (FVBM) during the second stage starting from the parameter values obtained during the first stage. When an RBM is used in the second stage, one could simply discard $\boldsymbol{\mu}_+$.

Another important point of the proposed algorithm is that it provides another research perspective in training DBMs. The existing pretraining scheme developed by Salakhutdinov and Hinton [2009] was based on the observation that with some assumptions the variational lower-bound could be increased by learning weight parameters layerwise. However, the success of the proposed scheme suggests that it may not be the set of parameters that need to be directly pretrained, but the set of variational parameters that determine how tight the variational lower-bound is and their corresponding parameters. This way of approaching the problem of training DBMs enables us to potentially find another explanation on why training large DBMs without pretraining is not trivial, if not impossible.

6 Experiments

	DBM	$\text{DBM}^{\text{sDAE}}_{\text{RBM}}$	$\text{DBM}^{\text{DBN}}_{\text{RBM}}$	$\text{DBM}^{\text{S\&H}}$	$\text{DBM}^{\text{S\&H}}_{\text{RBM}}$	$\text{DBM}^{\text{S\&H}}_{\text{FVBM}}$
L2s	-	-117.94	-118.46	-2731.18	-2471.42	-2525.74
L2	-	-117.78	-130.04	-2878.03	-2571.00	-2614.64
L3	-	-118.16	-132.91	-2943.63	-2618.30	-2648.74
L4	-	-121.04	-142.86	-2937.42	-2620.88	-2650.02

Table 3: Log-probabilities of the test samples of Caltech-101 Silhouettes after pretraining.

	DBM	$\text{DBM}^{\text{sDAE}}_{\text{RBM}}$	$\text{DBM}^{\text{DBN}}_{\text{RBM}}$	$\text{DBM}^{\text{S\&H}}$	$\text{DBM}^{\text{S\&H}}_{\text{RBM}}$	$\text{DBM}^{\text{S\&H}}_{\text{FVBM}}$
L2s	-161.26	**-115.47**	-115.48	-129.47	-124.36	-125.55
L2	-156.97	**-113.32**	-120.11	-130.57	-128.84	-129.52
L3	-158.82	**-110.85**	-120.69	-131.50	-133.68	-137.61
L4	-156.68	**-111.22**	-130.01	-126.60	-130.68	-130.81

Table 4: Log-probabilities of the test samples of Caltech-101 Silhouettes after finetuning.

In the experiments, we train DBMs on two datasets which are a handwritten digit dataset (MNIST) [LeCun et al.] and Caltech-101 Silhouettes dataset [Marlin et al., 2010]. Despite the critique that MNIST has been overly misused we have decided once again to use MNIST, since the experimental result of using DBMs for MNIST is readily available for direct comparison [Salakhutdinov and Hinton, 2010, Salakhutdinov, 2010, Montavon and Müller, 2012]. Caltech-101 Silhouettes is another binary dataset that is more difficult to learn by Boltzmann machines, which could be used to analyze the effect of the depth of DBMs.

We train DBMs with varying numbers of units in the hidden layers; 500-1000, 500-500-1000, 500-500-500-1000 (denoted by *L2*, *L3* and *L4*, respectively). We also trained a smaller DBM with 400-100 hidden units (denoted by *L2s*) which was the one that was used by Montavon and Müller [2012] without any pretraining algorithm.

For learning algorithms, we extensively tried various combinations. They are presented in Tab. 5. In summary, a DBM^{stage1}_{stage2} denotes a deep Boltzmann machine in which its superscript and subscript denote the algorithms used during the first and second stages, respectively.

During the first stage of the proposed algorithm, we learned the parameters of RBMs by CD and the enhanced gradient [Cho et al., 2011b]. DAEs were trained by setting 20% of input pixels to 0 on every update and regularizing the sparsity of hidden units to be close to 0.1. Large RBMs in stage 2 were trained using the stochastic approximation procedure [SAP, Tieleman and Hinton, 2009] with

	Stage 1	Stage 2	Finetuning
DBM	$\times$	$\times$	$\circ$
$\text{DBM}^{\text{S\&H}}$	(S)	$\times$	$\circ$
$\text{DBM}^{\text{sDAE}}_{\text{RBM}}$	sDAE	RBM	$\circ$
$\text{DBM}^{\text{DBN}}_{\text{RBM}}$	DBN	RBM	$\circ$
$\text{DBM}^{\text{S\&H}}_{\text{RBM}}$	(S)	RBM	$\circ$
$\text{DBM}^{\text{S\&H}}_{\text{FVBM}}$	(S)	FVBM	$\circ$

Table 5: Algorithms used in the experiment. (S) denotes the pretraining algorithm proposed by Salakhutdinov and Hinton [2009].

CAST and the enhanced gradient, except for when the conventional pretraining algorithm was used during the first stage. Then we used the traditional gradient during the second stage.

We trained each model for 200 epochs in the case of MNIST and 2000 epochs in the case of Caltech-101 Silhouettes with a learning rate scheduled by $\frac{\eta_0}{1+\frac{n}{5000}}$ where n is the number of updates. η_0 was set to 0.01 and 0.0005 for pretraining and finetuning, respectively. When we did not pretrain a DBM, we trained each DBM for twice more epochs and set η_0 to 0.001. In all cases, we used a minibatch of size 128 to compute a stochastic gradient update direction.

When training RBMs with CAST, we formed equally spaced 21 tempered fast chains from 0.9 to 1 with a single Gibbs sampling step on each update. When DAEs were trained, at every update we dropped off a randomly chosen set of hidden units with probability 0.1 [Hinton et al., 2012].

For finetuning, we again used the SAP with CAST with the same 21 tempered fast chains equally spaced between 0.9 and 1. At each update we iterated at most 30 fixed-point updates starting from uniform random values to compute variational parameters and performed 5-step Gibbs sampling for sampling.

We evaluated the resulting models with the variational lower-bound of log-probabilities of test samples. In order to estimate the normalizing constant $Z(\boldsymbol{\theta})$, annealed importance sampling (AIS) has been used with 20001 equally spaced temperatures from 0 to 1 with 128 independent runs. This enables us to directly compare our result with that reported by Salakhutdinov and Hinton [2010] and Salakhutdinov [2010].

Additionally, we checked the effect of pretraining on the discriminative property of the approximate posterior distribution $Q(\mathbf{h})$. For each final model of each algorithm and structure, we trained a simple multinomial logistic regression classifier for each hidden layer l using the variational parameters $\boldsymbol{\mu}_l$ as features. This is expected to show how much information about input samples is captured by each hidden layer of the model.

We trained each model with each learning algorithm five times starting from different random initializations. All the performance values including log-probabilities and classification accuracies presented are medians over those random trials.

6.1 Result and Analysis

Tables 1-2 present the lower-bound of log-probabilities of the test samples of MNIST computed from the trained DBMs after pretraining and after finetuning are presented. It is clear that initializing the parameters of the DBMs using the proposed pretraining scheme significantly facilitates finetuning. It is observed by the much higher log-probabilities achieved by the models that were initialized with the proposed two-stage algorithm. Furthermore, it is notable that no good generative model could be learned without pretraining[1].

A similar trend can also be found in Tab. 3-4 where the lower-bounds computed from the DBMs trained on Caltech-101 Silhouettes are presented. It is clear that the proposed algorithm, especially when DAEs are used during the first stage, outperforms the DBMs trained without any pretrain-

[1]The result reported by Salakhutdinov and Hinton [2010] is different from the performance obtained by $\text{DBM}^{\text{S\&H}}$ in this paper due to the difference in training procedures. For instance, in this paper, we have trained the model for fewer epochs.

	μ_1	μ_2	μ_3	μ_4	μ_{all}
L2s	95.42	71.08	-	-	95.45
L2	93.81	76.82	-	-	93.09
L3	95.75	66.79	11.35	-	95.12
L4	95.73	83.12	18.62	11.35	95.24

(a) DBM

	μ_1	μ_2	μ_3	μ_4	μ_{all}
L2s	96.19	94.38	-	-	96.82
L2	96.97	97.90	-	-	97.98
L3	96.62	96.42	84.47	-	97.31
L4	96.64	95.83	91.56	84.25	97.29

(b) $\text{DBM}^{\text{sDAE}}_{\text{RBM}}$

	μ_1	μ_2	μ_3	μ_4	μ_{all}
L2s	95.91	94.15	-	-	96.59
L2	96.63	97.88	-	-	**98.06**
L3	96.28	96.79	92.81	-	97.45
L4	96.12	96.28	92.69	85.09	97.14

(c) $\text{DBM}^{\text{DBN}}_{\text{RBM}}$

	μ_1	μ_2	μ_3	μ_4	μ_{all}
L2s	97.14	92.60	-	-	97.29
L2	97.55	95.44	-	-	97.79
L3	97.65	97.22	95.09	-	97.92
L4	97.62	97.24	96.49	94.47	98.07

(d) $\text{DBM}^{\text{S\&H}}$

	μ_1	μ_2	μ_3	μ_4	μ_{all}
L2s	97.34	93.33	-	-	97.50
L2	97.70	95.57	-	-	97.93
L3	97.86	97.68	96.23	-	**98.24**
L4	97.80	97.49	96.33	91.94	**98.21**

(e) $\text{DBM}^{\text{S\&H}}_{\text{RBM}}$

	μ_1	μ_2	μ_3	μ_4	μ_{all}
L2s	97.26	93.60	-	-	**97.57**
L2	97.70	95.86	-	-	98.00
L3	97.85	97.42	95.25	-	98.14
L4	97.84	97.38	96.54	94.50	98.15

(f) $\text{DBM}^{\text{S\&H}}_{\text{FVBM}}$

Table 6: Classification accuracy on MNIST. For each structure, the best accuracies are bold-faced.

ing as well as those with the conventional pretraining scheme or any other approach considered significantly.

Table 6 shows the layer-wise classification accuracies of test samples of MNIST. It is clear from the significantly lower accuracies in the higher hidden layers of the DBMs trained without pretraining that pretraining is essential to allow upper layers to capture discriminative structures of data samples. This phenomenon could be observed regardless of the configuration used for the proposed two-stage algorithm[2]. It is noticeable that the DBMs trained with the proposed two-stage algorithm using the conventional pretraining during the first stage consistently outperform those trained with the conventional pretraining only consistently over all structures, especially when the RBMs were used for the second stage. A similar trend was observed for Caltech-101 Silhouettes, but the results are not shown here due to the space limitation.

7 Conclusion

The experimental success of the proposed two-stage pretraining algorithm in training DBMs suggests that the difficulty of DBM learning might be due to the fact that the estimated variational lower-bound at the initial stage of learning is too crude, or too loose. Once one initializes the variational parameters well enough by utilizing another deep hierarchical model, the parameters of a DBM can be fitted to give a tighter variational lower-bound which facilitates jointly estimating all parameters. From then on, one can learn a good generative model by following the standard DBM training procedure.

The proposed two-stage pretraining algorithm provides a general framework in which many of previously successful hierarchical deep learning models such as DBNs and sDAEs can be used. It even makes possible to include the conventional pretraining algorithm as a part of the proposed algorithm and improve upon it. This is a significant step in developing and improving a training algorithm for DBMs, as it allows us to fully utilize other learning algorithms that have been extensively studied previously.

[2] It should be noted that these accuracies were computed purely to illustrate the effect of generative training of DBMs, and they are lower than other state-of-the-art accuracies (see, [Hinton et al., 2012]).

References

Y. Bengio, A. C. Courville, and P. Vincent. Unsupervised feature learning and deep learning: A review and new perspectives. *CoRR (Cornell Univ. Computing Research Repository)*, abs/1206.5538, 2012.

C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, corrected 2nd printing edition, 2007.

K. Cho, T. Raiko, and A. Ilin. Parallel tempering is efficient for learning restricted Boltzmann machines. In *Proc. of the Int. Joint Conf. on Neural Networks (IJCNN 2010)*, Barcelona, Spain, July 2010.

K. Cho, T. Raiko, and A. Ilin. Gaussian-Bernoulli deep Boltzmann machine. In *NIPS 2011 Workshop on Deep Learning and Unsupervised Feature Learning*, Sierra Nevada, Spain, December 2011a.

K. Cho, T. Raiko, and A. Ilin. Enhanced gradient and adaptive learning rate for training restricted Boltzmann machines. In *Proc. of the 28th Int. Conf. on Machine Learning (ICML 2011)*, pages 105–112, New York, NY, USA, June 2011b. ACM.

A. Courville, J. Bergstra, and Y. Bengio. A spike and slab restricted Boltzmann machine. In *Proc. of the 14th Int. Conf. on Artificial Intelligence and Statistics (AISTATS 2011)*, 2011.

G. Desjardins, A. Courville, Y. Bengio, P. Vincent, and O. Delalleau. Parallel tempering for training of restricted Boltzmann machines. In *Proc. of the 13th Int. Conf. on Artificial Intelligence and Statistics (AISTATS 2010)*, pages 145–152, 2010.

G. Desjardins, A. C. Courville, and Y. Bengio. On training deep Boltzmann machines. *CoRR (Cornell Univ. Computing Research Repository)*, abs/1203.4416, 2012.

S. Haykin. *Neural Networks and Learning Machines*. Pearson Education, 3rd ed. edition, 2009.

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

G. Hinton and R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313 (5786):504–507, July 2006.

G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *ArXiv e-prints 1207.0580*, July 2012.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, number 11, pages 2278–2324.

B. M. Marlin, K. Swersky, B. Chen, and N. de Freitas. Inductive principles for restricted Boltzmann machine learning. In *Proc. of the 13th Int. Conf. on Artificial Intelligence and Statistics (AISTATS 2010)*, pages 509–516, 2010.

J. Martens. Deep learning via Hessian-free optimization. In J. Fürnkranz and T. Joachims, editors, *Proc. of the 27th Int. Conf. on Machine Learning (ICML 2010)*, pages 735–742, Haifa, Israel, June 2010. Omnipress.

G. Montavon and K.-R. Müller. Learning feature hierarchies with centered deep Boltzmann machines. *ArXiv e-prints 1203.3783*, March 2012.

T. Raiko, H. Valpola, and Y. LeCun. Deep learning made easier by linear transformations in perceptrons. In *Proc. of the 15th Int. Conf. on Artificial Intelligence and Statistics (AISTATS 2012)*, La Palma, Canary Islands, Spain, April 2012.

R. Salakhutdinov. Learning in Markov random fields using tempered transitions. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 1598–1606. 2009.

R. Salakhutdinov. Learning deep Boltzmann machines using adaptive MCMC. In J. Fürnkranz and T. Joachims, editors, *Proc. of the 27th Int. Conf. on Machine Learning (ICML 2010)*, pages 943–950, Haifa, Israel, June 2010. Omnipress.

R. Salakhutdinov and G. Hinton. An efficient learning procedure for deep Boltzmann machines. Technical Report MIT-CSAIL-TR-2010-037, MIT, August 2010.

R. Salakhutdinov and G. E. Hinton. Deep Boltzmann machines. In *Proc. of the Int. Conf. on Artificial Intelligence and Statistics (AISTATS 2009)*, pages 448–455, 2009.

H. Schulz and S. Behnke. Learning two-layer contractive encodings. In *Proc. of the 21st Int. Conf. on Artificial Neural Networks (ICANN 2012)*, September 2012.

T. Tieleman and G. E. Hinton. Using fast weights to improve persistent contrastive divergence. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 1033–1040, New York, NY, USA, 2009. ACM.