Journal of Machine Learning Research 80:1–16, 2018 ACML 2018

Batch Normalized Deep Boltzmann Machines

Hung Vu† HUNGV@DEAKIN.EDU.AU
Tu Dinh Nguyen‡ TU.DINH.NGUYEN@MONASH.EDU
Trung Le‡ TRUNGLM@MONASH.EDU
Wei Luo† WEI.LUO@DEAKIN.EDU.AU
Dinh Phung‡ DINH.PHUNG@MONASH.EDU
† *Center for Pattern Recognition and Data Analytics, Deakin University, Geelong, Australia*
‡ *Monash University Clayton, VIC 3800, Australia*

Abstract

Training Deep Boltzmann Machines (DBMs) is a challenging task in deep generative model studies. The careless training usually leads to a divergence or a useless model. We discover that this phenomenon is due to the change of DBM layers' input signals during model parameter updates, similar to other deterministic deep networks such as Convolutional Neural Networks (CNNs) or Recurrent Neural Networks (RNNs). The change of layers' input distributions not only complicates the learning process but also causes redundant neurons that simply imitate the others' behaviors. Although this phenomenon can be coped using batch normalization in deep learning, integrating this technique into the probabilistic network of DBMs is a challenging problem since it has to satisfy two conditions of energy function and conditional probabilities. In this paper, we introduce Batch Normalized Deep Boltzmann Machines (BNDBMs) that meet both aforementioned conditions and successfully combine batch normalization and DBMs into the same framework. However, unlike CNNs, due to the probabilistic nature of DBMs, training DBMs with batch normalization has some differences: i) fixing shift parameters β but learning scale parameters γ; ii) avoiding normalizing the first hidden layer and iii) maintaining multiple pairs of population means and variances per neuron rather than one pair in CNNs. We observe that our proposed BNDBMs can stabilize the input signals of network layers and facilitate the training process as well as improve the model quality. More interestingly, BNDBMs can be trained successfully without pretraining, which is usually a mandatory step in most existing DBMs. The experimental results in MNIST, Fashion-MNIST and Caltech 101 Silhouette datasets show that our BNDBMs outperform DBMs and centered DBMs in terms of feature representation and classification accuracy (3.98% and 5.84% average improvement for pretraining and no pretraining respectively).

1. Introduction

Deep Boltzmann Machines (DBMs) Salakhutdinov and Hinton (2009) are powerful hierarchical generative models that are widely-used in machine learning community as probabilistic modeling methods Salakhutdinov and Hinton (2009) and feature extractors Montavon et al. (2012). However, training DBM is notoriously non-trivial Melchior et al. (2016); Cho et al. (2013b). DBM's training algorithms, e.g., Persistent Contrastive Divergence (PCD) Tieleman (2008) or Parallel Tempering Melchior et al. (2016); Cho et al. (2013a), require additional tricks such as layer-wise pretraining Salakhutdinov and Hinton (2009); Cho et al.

(2013b,a), centering trick Melchior et al. (2016), careful parameter initialization and low learning rates. Otherwise, the training tends to diverge or converge to a bad solution with meaningless neurons that are always being active or inactive or independent of data.

To understand why training multilayer DBMs is challenging, let us consider the case of binary DBMs where all units are Bernoulli random variables. In this network, the state of i-th hidden unit $h_i^{(l)}$ in l-th hidden layer $\boldsymbol{h}^{(l)}$ is sampled from a Bernoulli distribution $h_i^{(l)} \sim \text{Bernoulli}(\mu_i^{(l)})$ with the mean $\mu_i^{(l)} = \sigma(\text{b}_i^{(l)} + \boldsymbol{h}^{(l-1)\top}\boldsymbol{w}_{\cdot i}^{(l)} + \boldsymbol{w}_{i\cdot}^{(l+1)}\boldsymbol{h}^{(l+1)})$ where $\sigma(\cdot)$ is the sigmoid function of the signals from two adjacent layers. The sigmoid function produces the extreme output that is close to the boundary 0 or 1 whenever its input is outside $[-4, 4]$ (Fig. 1). Since this interval is fairly small, it is highly possible that the input $\text{b}_i^{(l)} + \boldsymbol{h}^{(l-1)\top}\boldsymbol{w}_{\cdot i}^{(l)} + \boldsymbol{w}_{i\cdot}^{(l+1)}\boldsymbol{h}^{(l+1)}$ jumps out if biases or weight matrices consist of large magnitude numbers. This results in $\mu_i^{(l)} \approx 1$ or 0 and $h_i^{(l)} = 1$ or 0, rendering an "always-on" or "always-off" neuron. According to Salakhutdinov and Hinton (2012), the parameter gradients are proportional to the difference between two expectations with respect to these terms μ and h, and therefore the gradients are close to 0 when these terms always reach one of the boundary values 1 or 0. As a result, no learning happens and the neuron has no chance to escape from the dead state. To avoid the explosion in the sigmoid input, zero-biases, small weight initialization and a small learning rate are always recommended for DBMs. Layer-wise pretraining is another solution to initialize DBM parameters into a good region in the parameter space. Overall, the stability of layer's input has a great impact on the success of DBM learning.

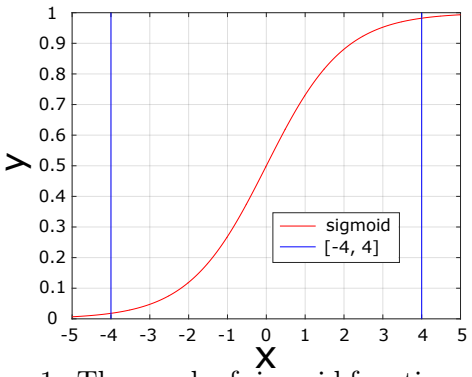

Figure 1: The graph of sigmoid function. The input value x outside $[-4, 4]$ has y close to 0 or 1.

Ioffe and Szegedy (2015) introduced batch normalization to cope with the change of the distributions of neurons during CNN training, also known as the internal covariate shift phenomenon. This motivates us to adopt this idea to handle the instability of layer's input in DBM training. However, improving the probabilistic network of DBM with deep learning techniques is a non-trivial problem. In common deep networks such as CNNs, advanced techniques or transformations, e.g., convolutional layer, pooling, drop-out, batch normalization, can be inserted into each layer easily. This is because deep networks can be considered as a deterministic composite function of elementary mappings, each of which is specified by the transformations between layers, and adding a new transformation is equivalent to inserting a mapping between layers. By contrast, all layers in DBMs define a joint distribution by admitting an energy function and the transformations between layers are defined by conditional distributions derived from the joint distribution. As a result, when applying a new transformation to a layer in DBMs, we need to guarantee that: (i) there exists an energy function that can adapt to this transformation; and (ii) this energy function (if exists) must result in a tractable form for the conditional distributions, which are important for propagating the information between layers.

It is challenging to use new transformation in DBMs without violating these conditions. An example is Deep Energy-based Model (DEMs) Ngiam et al. (2011), which is a RBM with two random variable layers of visible and hidden units and many deterministic intermediate layers between them. This structure allows to represent the posterior $p(\boldsymbol{h}|\boldsymbol{v})$ through a powerful deterministic function $g(\boldsymbol{v})$. However, this network sacrifices the opposite direction of $p(\boldsymbol{v}|\boldsymbol{h})$, which cannot be computed explicitly as in the vanilla RBMs. A seminal work, which preserves all properties of energy-based models, is Convolutional Restricted Boltzmann Machines (CRBM) Norouzi et al. (2018), where the connections between visible and hidden units are reorganized to bring the convolutional structure into RBMs. Max-pooling is also introduced in RBMs Lee et al. (2009) but it needs to be modified to accept the probabilistic property of RBMs. Although both convolutional and max-pooling transformations are available in RBMs, combining them into a probabilistic generative model is not straightforward. For example, Lee et al. (2009) proposed to stack pairs of convolutional RBMs and probabilistic max-pooling RBMs together to form a network with hierarchical feature representation. However, this architecture is a Deep Belief Net (DBN) Hinton et al. (2006), a hybrid network of both directed and undirected connections, and it is not an energy-based model as DBMs. By adding both convolutional and pooling parameters to the same energy function, CssCDBMs Xiaojun and Haibo (2018) introduce an effective way to include both techniques in DBMs. However, this results in a highly complicated model.

These limited studies not only reveal the challenges to combine deep learning techniques with energy-based models, but also point out that energy-based model studies have fallen behind the development of deep learning. As an effort to fill this gap, we introduce a novel model, named Batch Normalized DBM, that combines one of the best deep learning tricks - batch normalization and the probabilistic framework of DBMs. To that end, we design a *novel* energy function with normalization scale and shift parameters, which not only satisfies two aforementioned conditions, but also mathematically introduces the batch normalization formula to the conditional probabilities and therefore completely tackles the problem of internal covariate shift. More importantly, our proposed BNDBMs require no step of pretraining and therefore they have more advantages in training than many existing DBMs. Our research also finds that, compared with deterministic networks, e.g., CNNs, applying batch normalization to the probabilistic model of DBMs needs more careful investigation including: i) fixing shift parameters β but learning scale parameters γ and ii) avoiding normalizing the first hidden layer and iii) maintaining multiple pairs of population mean and variance per neuron rather than one pair in most deterministic networks. We refer to Sec. 4.3 for more discussion. Experiments on MNIST, Fashion-MNIST and Caltech 101 Silhouette datasets indicate that our BNDBM has a significant improvement in training efficiency (less dependent on pretraining steps) and classification accuracy (3.98% and 5.84% on average for training with/without pretraining).

2. Deep Boltzmann Machines

2.1. Model representation

Deep Boltzmann Machines (DBM) Salakhutdinov and Hinton (2009) are simplified variants of Boltzmann Machines where neurons are arranged in a multilayer architecture and there is no intra-layer connection. Let consider a network of one visible layer $\boldsymbol{v}$ with the bias

vector $\boldsymbol{a}$ and two hidden layers $\boldsymbol{h} = \{\boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)}\}$ with biases $\boldsymbol{b}^{(1)}$ and $\boldsymbol{b}^{(2)}$. The number of visible units is represented by M and the numbers of hidden units are K_1 and K_2 respectively. The interactions between each layer and its upper and lower layers are encoded in a visible-to-hidden matrix $\boldsymbol{W}^{(1)}$ and a hidden-to-hidden matrix $\boldsymbol{W}^{(2)}$. We use $\boldsymbol{\Psi} = \{\boldsymbol{a}, \boldsymbol{b}^{(1)}, \boldsymbol{b}^{(2)}, \boldsymbol{W}^{(1)}, \boldsymbol{W}^{(2)}\}$ to denote the set of all parameters. As an energy-based model, each state $(\boldsymbol{v}, \boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)}) \in \{0,1\}^{M+K_1+K_2}$ is associated with an energy value:

$$E\left(\boldsymbol{v}, \boldsymbol{h}; \boldsymbol{\Psi}\right) = -\boldsymbol{a}^\top \boldsymbol{v} - \boldsymbol{b}^{(1)\top} \boldsymbol{h}^{(1)} - \boldsymbol{b}^{(2)\top} \boldsymbol{h}^{(2)} - \boldsymbol{v}^\top \boldsymbol{W}^{(1)} \boldsymbol{h}^{(1)} - \boldsymbol{h}^{(1)\top} \boldsymbol{W}^{(2)} \boldsymbol{h}^{(2)}$$

A probability is assigned to each state as $p\left(\boldsymbol{v}, \boldsymbol{h}; \boldsymbol{\Psi}\right) = e^{-E(\boldsymbol{v}, \boldsymbol{h}; \boldsymbol{\Psi})} / \mathcal{Z}\left(\boldsymbol{\Psi}\right)$, wherein $\mathcal{Z}\left(\boldsymbol{\Psi}\right)$ is a normalizing constant, also known as partition function, and it is dependent on the model parameters. The restrictions on no connections between two neurons at the same layer provide a nice property of their conditional independence given the adjacent upper and lower layers. In the case of DBM with Bernoulli units, the conditional probability of a neuron to be 1 is represented via a logistic sigmoid function $\sigma\left(\cdot\right)$:

$$p\left(v_m = 1 | \boldsymbol{h}^{(1)}; \boldsymbol{\Psi}\right) = \sigma\left(a_m + \boldsymbol{w}_{m\cdot}^{(1)} \boldsymbol{h}^{(1)}\right) \qquad p\left(h_n^{(2)} = 1 | \boldsymbol{h}^{(1)}; \boldsymbol{\Psi}\right) = \sigma\left(\mathrm{b}_n^{(2)} + \boldsymbol{h}^{(1)\top} \boldsymbol{w}_{\cdot n}^{(2)}\right)$$

$$p\left(h_n^{(1)} = 1 | \boldsymbol{v}, \boldsymbol{h}^{(2)}; \boldsymbol{\Psi}\right) = \sigma\left(\mathrm{b}_n^{(1)} + \boldsymbol{v}^\top \boldsymbol{w}_{\cdot n}^{(1)} + \boldsymbol{w}_{n\cdot}^{(2)} \boldsymbol{h}^{(2)}\right)$$

2.2. Parameter training

The DBM parameters $\boldsymbol{\Psi}$ can be learned by maximizing the likelihood function $\mathcal{L}\left(\boldsymbol{v}; \boldsymbol{\Psi}\right) = \sum_{\boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)}} p\left(\boldsymbol{v}, \boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)}; \boldsymbol{\Psi}\right)$ from data $\mathcal{D} = \left\{\boldsymbol{v}^{[i]}\right\}_{i=1}^{N}$, where N is the number of training data points. The gradient of the log-likelihood is estimated as the partial derivatives of $\log \mathcal{L}$ with respect to each parameter:

$$\nabla_{\boldsymbol{\Psi}} \log \mathcal{L} = \mathbb{E}_{\text{data}}\left[-\frac{\partial E}{\partial \boldsymbol{\Psi}}\right] - \mathbb{E}_{\text{model}}\left[-\frac{\partial E}{\partial \boldsymbol{\Psi}}\right] \tag{1}$$

wherein $\mathbb{E}_{\text{data}}$ and $\mathbb{E}_{\text{model}}$ are expectations over the data distribution and the model distribution respectively. Since estimating two expectations in DBMs is computationally demanding, approximation approach was proposed to work around. The first term is usually approximated via a factorial distribution:

$$q\left(\boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)}; \tilde{\boldsymbol{\Psi}}\right) = \prod_{l=1}^{2} \prod_{i=1}^{K_l} \left(\tilde{\mu}_i^{(l)}\right)^{h_i^{(l)}} \left(1 - \tilde{\mu}_i^{(l)}\right)^{1 - h_i^{(l)}} \tag{2}$$

The variational parameters $\tilde{\mu}$ are trained by updating the corresponding fixed-point equations until convergence:

$$\tilde{\mu}_n^{(1)} = \sigma\left(\mathrm{b}_n^{(1)} + \boldsymbol{v}^\top \boldsymbol{w}_{\cdot n}^{(1)} + \boldsymbol{w}_{n\cdot}^{(2)} \tilde{\boldsymbol{\mu}}^{(2)}\right) \qquad \tilde{\mu}_n^{(2)} = \sigma\left(\mathrm{b}_n^{(2)} + \tilde{\boldsymbol{\mu}}^{(1)\top} \boldsymbol{w}_{\cdot n}^{(2)}\right)$$

Meanwhile, the second expectation can be estimated using Gibbs sampling. This method, however, requires to run Gibbs chains until convergence (hundreds or thousands of iterations) in every epoch. To shorten the training time, we can maintain persistent chains over training as Persistent Contrastive Divergence (PCD) approach does in Tieleman (2008). The combination of variational approximation and PCD results in a success of DBM training with a pretraining step Salakhutdinov and Hinton (2009).

2.3. Layer-wise pretraining

Training DBMs directly from randomly initialized weights generally does not work Salakhutdinov and Hinton (2009); Cho et al. (2013a,b). The existing DBMs Salakhutdinov and Hinton (2012); Cho et al. (2013a,b); Xiaojun and Haibo (2018) require good initialization of parameters via a pretraining step. The pretraining procedure for DBMs was originally introduced by Salakhutdinov and Hinton (2009). This method views two consecutive layers as a RBM and then stacks trained RBMs to initialize DBM's parameters before triggering the main training procedure.

3. Batch Normalized Deep Boltzmann Machines

Training deep networks is challenging since the distribution of signals passing between layers changes dramatically and continuously during training. This phenomenon is known as "internal covariate shift" and can be eliminated by the batch normalization Ioffe and Szegedy (2015) in CNNs. Interestingly, we observe the same phenomenon of unstable input distributions during DBM training and this motivates us to apply this technique to DBM.

3.1. Batch normalization

A batch normalization operator, described in Ioffe and Szegedy (2015), normalizes every dimension i of an input vector $\boldsymbol{x}$ independently to make a normalized vector $\bar{\boldsymbol{x}}$ with the mean of 0 and standard deviation of 1 across its dimensions.

$$\bar{\boldsymbol{x}}_i = \frac{\boldsymbol{x}_i - \mathbb{E}\left[\boldsymbol{x}_i\right]}{\sqrt{\mathrm{Var}\left(\boldsymbol{x}_i\right) + \epsilon}}$$

where ϵ is a small positive constant to prevent a division-by-zero issue. To preserve the capacity of deep networks, a scale parameter γ_i and a shift parameter β_i were introduced to obtain an output signal $\boldsymbol{y}_i = \gamma_i \bar{\boldsymbol{x}}_i + \beta_i$. As a result, the batch normalization can be expressed as an affine transform $\mathcal{B}_{\gamma_i,\beta_i}\left(\boldsymbol{x}_i\right) = \bar{\gamma}\boldsymbol{x}_i + \bar{\beta}$, wherein $\bar{\gamma}_i = \gamma_i / \sqrt{\mathrm{Var}\left(\boldsymbol{x}_i\right) + \epsilon}$ and $\bar{\beta}_i = \beta_i - \gamma_i \mathbb{E}[\boldsymbol{x}_i] / \sqrt{\mathrm{Var}(\boldsymbol{x}_i) + \epsilon}$.

3.2. DBM with batch normalization

For the sake of simplicity, we build our Batch Normalized Deep Boltzmann Machines (BNDBMs) based on the DBM network mentioned in Sec. 2. By adding batch normalization parameter pairs $\left(\boldsymbol{\gamma}^{(l)}, \boldsymbol{\beta}^{(l)}\right)$ to the l^{th} hidden layer to normalize its input signals $\boldsymbol{x}^{(l)}$, a BNDBM is specified by a parameter set $\boldsymbol{\Gamma}=\boldsymbol{\Psi}\cup\{\gamma_i^{(1)}, \beta_i^{(1)}\}_{i=1}^{K_1} \cup \{\gamma_j^{(2)}, \beta_j^{(2)}\}_{j=1}^{K_2}$. The energy function over the network is redefined as:

$$\begin{aligned} E\left(\boldsymbol{v}, \boldsymbol{h}; \boldsymbol{\Gamma}\right) \quad = \quad & -\sum_{i=1}^{M} a_i \mathrm{v}_i - \sum_{l=1}^{2}\sum_{j=1}^{K_l}\left(\bar{\gamma}_j^{(l)} \mathrm{b}_j^{(l)} + \bar{\beta}_j^{(l)}\right) h_j^{(l)} \\ & -\sum_{i=1}^{M}\sum_{j=1}^{K_1} \bar{\gamma}_j^{(1)} \mathrm{v}_i w_{ij}^{(1)} h_j^{(1)} - \sum_{i=1}^{K_1}\sum_{j=1}^{K_2} \bar{\gamma}_i^{(1)} \bar{\gamma}_j^{(2)} h_i^{(1)} w_{ij}^{(2)} h_j^{(2)} \end{aligned} \tag{3}$$

Similar to DBMs, we can obtain the formula of the joint probability distribution, $p\left(\boldsymbol{v}, \boldsymbol{h}; \boldsymbol{\Gamma}\right)$, the marginal distributions, $\sum_{\boldsymbol{v}} p\left(\boldsymbol{v}, \boldsymbol{h}; \boldsymbol{\Gamma}\right)$, $\sum_{\boldsymbol{h}^{(1)}} p\left(\boldsymbol{v}, \boldsymbol{h}; \boldsymbol{\Gamma}\right)$ and

$\sum_{\boldsymbol{h}^{(2)}} p(\boldsymbol{v}, \boldsymbol{h}; \boldsymbol{\Gamma})$ (we refer the supplementary material for more details), and finally the conditional probabilities (Eqs. 4, 5 and 6) from the energy function (Eq. 3). As we expected, the conditional distributions between layers are in the sigmoid form as DBMs but with batch normalization operators added to hidden layers:

$$p\left(v_m = 1|\boldsymbol{h}^{(1)}\right) = \sigma\left(a_m + \sum_{j=1}^{K_1} \bar{\gamma}_j^{(1)} w_{mj}^{(1)} h_j^{(1)}\right) \quad (4) \qquad p\left(h_n^{(1)} = 1|\boldsymbol{v}, \boldsymbol{h}^{(2)}\right) = \sigma\left(\mathcal{B}_1\left(t_n^{(1)}\right)\right) \quad (6)$$

$$p\left(h_n^{(2)} = 1|\boldsymbol{h}^{(1)}\right) = \sigma\left(\mathcal{B}_2\left(t_n^{(2)}\right)\right) \quad (5)$$

$$t_n^{(1)}\left(\boldsymbol{v}, \boldsymbol{h}^{(2)}\right) = \mathrm{b}_n^{(1)} + \sum_{j=1}^{M} \mathrm{v}_j w_{jn}^{(1)} + \sum_{j=1}^{K_2} \bar{\gamma}_j^{(2)} w_{nj}^{(2)} h_j^{(2)} \qquad t_n^{(2)}\left(\boldsymbol{h}^{(1)}\right) = \mathrm{b}_n^{(2)} + \sum_{j=1}^{K_1} \bar{\gamma}_j^{(1)} h_j^{(1)} w_{jn}^{(2)} \quad (7)$$

wherein $t_n^{(1)}$ and $t_n^{(2)}$ are the non-normalized input signals of the sigmoid functions and we write $\mathcal{B}_1$ and $\mathcal{B}_2$ instead of $\mathcal{B}_{\gamma_n^{(1)}, \beta_n^{(1)}}$ and $\mathcal{B}_{\gamma_n^{(2)}, \beta_n^{(2)}}$ to shorten notations.

In DBM models, the conditional distribution equations have an important role to explain how signals are transferred between layers. Compared with CNNs where layer-to-layer transformations (e.g., batch normalization, max-pooling and convolutional operators) can be freely designed and integrated between layers, the conditional distributions in DBMs are deterministically derived using probability rules from the energy equation. Therefore, any modification of such distributions should come from an appropriate energy equation. The introduction of the energy function above (Eq. 3) naturally adds batch normalization transform into the conditional probabilities without breaking the probabilistic characteristics of DBMs. As a result, DBM is improved significantly with the power of batch normalization in our proposal.

3.3. Model training

BNDBM training by optimizing data log-likelihood function also requires the approximation of the intractable posterior using variational distribution and the model distribution estimation using efficient sampling. The variational distribution keeps the same form as Eq. 2 but the fixed-point update equations become:

$$\tilde{\mu}_n^{(1)} = \sigma\left(\mathcal{B}_1\left(\mathrm{b}_n^{(1)} + \sum_{j=1}^{M} \mathrm{v}_j w_{jn}^{(1)} + \sum_{j=1}^{K_2} \bar{\gamma}_j^{(2)} w_{nj}^{(2)} \tilde{\mu}_j^{(2)}\right)\right) \quad \tilde{\mu}_n^{(2)} = \sigma\left(\mathcal{B}_2\left(\mathrm{b}_n^{(2)} + \sum_{j=1}^{K_1} \bar{\gamma}_j^{(1)} \tilde{\mu}_j^{(1)} w_{jn}^{(2)}\right)\right)$$

Suppose that the model distribution is estimated by N_c persistent MCMC chains and the model is trained with stochastic batch gradient ascent of batch size N_s, BNDBM uses the following equations to update its biases and weights

$$\Delta a_m = \eta\left(\sum_{i=1}^{N_s} \frac{\mathrm{v}_m^{[i]}}{N_s} - \sum_{i=1}^{N_c} \frac{\hat{\mathrm{v}}_m^{\langle i \rangle}}{N_c}\right) \quad (8) \qquad \Delta b_n^{(l)} = \eta \bar{\gamma}_n^{(l)}\left(\sum_{i=1}^{N_s} \frac{\tilde{\mu}_n^{(l)[i]}}{N_s} - \sum_{i=1}^{N_c} \frac{\hat{h}_n^{(l)\langle i \rangle}}{N_c}\right) \quad (9)$$

$$\Delta w_{mn}^{(1)} = \eta \bar{\gamma}_n^{(1)} \left(\sum_{i=1}^{N_s} \frac{\mathrm{v}_m^{[i]} \tilde{\mu}_n^{(1)[i]}}{N_s} - \sum_{i=1}^{N_c} \frac{\hat{\mathrm{v}}_m^{\langle i \rangle} \hat{h}_n^{(1)\langle i \rangle}}{N_c} \right) \tag{10}$$

$$\Delta w_{mn}^{(2)} = \eta \bar{\gamma}_m^{(1)} \bar{\gamma}_n^{(2)} \left(\frac{\sum_{i=1}^{N_s} \tilde{\mu}_m^{(1)[i]} \tilde{\mu}_n^{(2)[i]}}{N_s} - \frac{\sum_{i=1}^{N_c} \hat{h}_m^{(1)\langle i \rangle} \hat{h}_n^{(2)\langle i \rangle}}{N_c} \right) \tag{11}$$

and its normalization parameters:

$$\Delta \gamma_n^{(1)} = \eta \left(\sum_{i=1}^{N_s} \tilde{\mu}_n^{(1)[i]} \frac{t_n^{(1)} \left(\boldsymbol{v}^{[i]}, \tilde{\boldsymbol{\mu}}^{(2)[i]} \right) - \mathbb{E}_b[t_n^{(1)}]}{N_s \sqrt{\mathrm{Var}\left[t_n^{(1)}\right] + \epsilon}} - \sum_{i=1}^{N_c} \hat{h}_n^{(1)\langle i \rangle} \frac{t_n^{(1)} \left(\hat{\boldsymbol{v}}^{\langle i \rangle}, \hat{\boldsymbol{h}}^{(2)\langle i \rangle} \right) - \mathbb{E}_b[t_n^{(1)}]}{N_c \sqrt{\mathrm{Var}\left[t_n^{(1)}\right] + \epsilon}} \right) \tag{12}$$

$$\Delta \gamma_n^{(2)} = \eta \left(\sum_{i=1}^{N_s} \tilde{\mu}_n^{(2)[i]} \frac{t_n^{(2)} \left(\tilde{\boldsymbol{\mu}}^{(1)[i]} \right) - \mathbb{E}_b[t_n^{(2)}]}{N_s \sqrt{\mathrm{Var}\left[t_n^{(2)}\right] + \epsilon}} - \sum_{i=1}^{N_c} \hat{h}_n^{(2)\langle i \rangle} \frac{t_n^{(2)} \left(\hat{\boldsymbol{h}}^{(1)\langle i \rangle} \right) - \mathbb{E}_b[t_n^{(2)}]}{N_c \sqrt{\mathrm{Var}\left[t_n^{(2)}\right] + \epsilon}} \right) \tag{13}$$

$$\Delta \beta_n^{(l)} = \eta \left(\sum_{i=1}^{N_s} \frac{\tilde{\mu}_n^{(l)[i]}}{N_s} - \sum_{i=1}^{N_c} \frac{\hat{h}_n^{(l)\langle i \rangle}}{N_c} \right) \tag{14}$$

wherein $\boldsymbol{v}^{[i]}$ and $\tilde{\boldsymbol{\mu}}^{(l)[i]}$ are the i^{th} data vector and its corresponding mean-field vector, $\hat{\boldsymbol{v}}^{\langle i \rangle}$and $\hat{\boldsymbol{h}}^{(l)\langle i \rangle}$ are layer states on the i^{th} Gibbs chain and $\mathbb{E}_b[\cdot]$ is batch mean.

We follow Ioffe and Szegedy (2015) to use separate statistics for batch normalization in training and inference modes. The normalization should only depend on batch statistics for efficient training, which are batch mean $\mathbb{E}_b$ and batch variance Var_b as described in Alg. 1. By contrast, the inference process demands fixed population statistical vectors, $\mathbb{E}_p$ and Var_p, which are evaluated over the training set as soon as the training ends. An unbiased estimate is also introduced to compute the population variance. It is noteworthy that, unlike batch normalization in CNNs where we only need one pair of population mean and variance per neuron, BNDBM requires $\max(N_{\text{mf}}, N_{\text{gs}})$ pairs because the input distribution of each neuron develops over N_{mf} mean-field steps and N_{gs} Gibbs sampling steps. The distribution of a layer usually varies much at the first steps but less at the last steps when the mean-field or Gibbs sampling converges. For this reason, separate population mean and variance are preferable for each step to handle this change.

Although the gradient update in BNDBMs is well specified by Eqs. 8-14, model learning in practice is difficult to converge because of the change of parameters with respect to normalization scale parameters γ (Eqs 9-10 and 14). After each batch update, the new scale values affect both other parameters and layers' input signals ($t_i^{(1)}$ and $t_i^{(2)}$) and make the model evolve in an unstable and fluctuating way.

To facilitate the learning process, we propose some strategies to lessen the effects of this dependence.

- **Simplified update equations**: We observe that the scale parameters have the same roles as learning rate η. This motivates to integrate theminto η. In other words, we can simply discard all scale parameters in gradient update equations (Eqs. 9-11).

- **Modified sigmoid inputs**: It can be seen that the sigmoid inputs (i.e., $t_i^{(1)}$ and $t_i^{(2)}$ in Eqs. 5 and 6) are functions of parameters γ, biases and weights. However, values of γ, e.g., 1, 1.2, 2, are many times larger than the unit biases and the connection weights, which are encouraged to be small numbers around zeros, e.g., 0.01, 0.002. When γ varies over training batches, it changes the layer's input distributions $t_i^{(1)}$ and $t_i^{(2)}$ dramatically and causes the training to fluctuate. Again, we include γ into weights and therefore freely remove it from the signal input formula (Eq. 7).

Algorithm 1 BNDBM training mode

Input: Training data $\mathcal{D} = \left\{\boldsymbol{v}^{[i]}\right\}_{i=1}^{N}$, N_b batches, N_{mf} mean-field steps and N_{gs} Gibbs steps
Output: BNDBM parameters $\boldsymbol{\Gamma}$, population means $\mathbb{E}_p$ and population variances Var_p
begin
  // Training parameters with instantaneous batch mean $\mathbb{E}_b$ and batch variance Var_b
  for $k \leftarrow 1, \ldots, N_b$ **do**
    $\mathcal{D}_k \leftarrow$ get batch k^{th}
    Run mean-field with batch normalization
    Run Gibbs sampling with batch normalization
    Update $\boldsymbol{\Gamma}$ (Eqs. 8-14)
  end
  // Updating the population mean and variance ($\mathbb{E}_p$ and Var_p) with all training data
  for $k \leftarrow 1, \ldots, N_b$ **do**
    $\mathcal{D}_k \leftarrow$ get batch k^{th}
    Compute $t_n^{(1)}$ and $t_n^{(2)}$ from $\mathcal{D}_k$
    Evaluate batch mean $\mathbb{E}_b^k$ and variance Var_b^k
  end
  for $i \leftarrow 1, \ldots, \max(N_{mf}, N_{gs})$ **do**
    $\mathbb{E}_p^i\left[t_n^{(l)}\right] \leftarrow \frac{\sum_{k=1}^{N_b} \mathbb{E}_b^k\left[t_n^{(l)\langle i\rangle}\right]}{N_b}$; $\text{Var}_p^i\left[t_n^{(l)}\right] \leftarrow \frac{\sum_{k=1}^{N_b} \text{Var}_b^k\left[t_n^{(l)\langle i\rangle}\right]}{N_b - 1}$
  end
  $\mathbb{E}_p \leftarrow \left[\mathbb{E}_p^i\right]_{i=1}^{\max(N_{\text{mf}}, N_{\text{gs}})}$; $\text{Var}_p \leftarrow \left[\text{Var}_p^i\right]_{i=1}^{\max(N_{\text{mf}}, N_{\text{gs}})}$
end

3.4. Pre-normalization vs post-normalization

In this section, we provide a principal comparison between BNDBM and its closely related work of centered DBM (cDBM) Montavon and Muller (2012), one of the state-of-the-art DBMs. This not only helps to understand the difference between these methods but also explains how our BNDBM can achieve better results in the experiment section. cDBMs were proposed to improve original DBMs by centering visible and hidden states. More specifically, given a cDBM with 2 hidden layers, its energy function is shown in Eq. 15 whilst Eqs. 16 and 18 are two examples of conditional distributions corresponding to hidden layers :

$$E\left(\boldsymbol{v}, \boldsymbol{h}\right) = -\sum_{i=1}^{M} a_i \bar{\bar{\text{v}}}_i - \sum_{l=1}^{2}\sum_{j=1}^{K_l} \text{b}_j^{(l)} \bar{\bar{h}}_j^{(l)} - \sum_{i=1}^{M}\sum_{j=1}^{K_1} \bar{\bar{\text{v}}}_i w_{ij}^{(1)} \bar{\bar{h}}_j^{(1)} - \sum_{i=1}^{K_1}\sum_{j=1}^{K_2} \bar{\bar{h}}_i^{(1)} w_{ij}^{(2)} \bar{\bar{h}}_j^{(2)} \tag{15}$$

$$p\left(h_n^{(1)}=1|\boldsymbol{v},\boldsymbol{h}^{(2)}\right)=\sigma\left(\mathrm{b}_n^{(1)}+\sum_{j=1}^{M}\bar{\bar{\mathrm{v}}}_j w_{jn}^{(1)}+\sum_{j=1}^{K_2}w_{nj}^{(2)}\bar{\bar{h}}_j^{(2)}\right) \tag{16}$$

$$\bar{\bar{h}}_n^{(1)}=h_n^{(1)}-d_n^{(1)},\qquad \text{where } h_n^{(1)}\sim\text{Bernoulli}\left(p\left(h_n^{(1)}=1|\boldsymbol{v},\boldsymbol{h}^{(2)}\right)\right) \tag{17}$$

$$p\left(h_n^{(2)}=1|\boldsymbol{h}^{(1)}\right)=\sigma\left(\mathrm{b}_n^{(2)}+\sum_{j=1}^{K_1}\bar{\bar{h}}_j^{(1)}w_{jn}^{(2)}\right) \tag{18}$$

wherein c_i and $d_j^{(l)}$ are the offsets of units and are set to the activation mean of the corresponding neuron state. The terms $\bar{\bar{\mathrm{v}}}_i=\mathrm{v}_i-c_i$ and $\bar{\bar{h}}_j^{(l)}=h_j^{(l)}-d_j^{(l)}$ are centered states and they are computed after the sigmoid transformation is applied to the layer's input signals. For example, the centered state $\bar{\bar{h}}_j^{(1)}$ (Eq. 17) is computed using the conditional probability in Eq. 16. Therefore, cDBM can be viewed as a post-normalization of layer signals. This post-normalization does not immediately correct the layer distribution of the current layer, e.g., $\boldsymbol{h}^{(1)}$, but it holds the states of the adjacent layers, e.g., $\boldsymbol{v}$ and $\boldsymbol{h}^{(2)}$, in a proper range and then stabilizes the current layer's inputs in the next propagation. By contrast, BNDBM does a pre-normalization of layer's inputs, before the sigmoid transformation. It can be seen that the post-normalization strategy of cDBM does not guarantee that the sigmoid inputs of network layers do not vary during training while the pre-normalization allows us to control the input distributions at a desired mean and variance. As a result, cDBM cannot solve the internal covariate shift problem as effectively as BNDBM.

4. Experiments and Analysis

In this section, we conduct our experiments on three datasets of MNIST LeCun et al. (1998), Fashion-MNIST Xiao et al. (2017) and Caltech 101 Silhouette Marlin et al. (2010) which are widely-used in DBM's literature Cho et al. (2013b); Salakhutdinov and Hinton (2012); Xiaojun and Haibo (2018); Sankaran et al. (2017). It is noteworthy that, due to the high complexity of the models, most unsupervised DBMs including BNDBMs only work efficiently on binary or grey-scale datasets. Several studies Sankaran et al. (2017) propose to train DBMs on Cifar, but these are supervised DBMs with labels added to the models.

We compare BNDBM with original DBM Salakhutdinov and Hinton (2009), and centered DBM (cDBM) Melchior et al. (2016). With the purpose of comparison with the post-normalization approach of cDBM, we follow its network structures Montavon and Muller (2012); Melchior et al. (2016) and use the networks of $784-500-100$ and $784-500-500$ in all experiments. All DBMs are also trained with PCD algorithm Tieleman (2008) using $N_s=N_c=100$ and learning rate of 10^{-2} over 500 epochs. The numbers of mean-field and Gibbs sampling iterations are set to 10. The weights of all methods are randomly initialized from normal distribution with zero-mean and 0.1-standard deviation whilst the biases are set to 0. Although both scale parameters γ and shift parameters β are learnable in theory, we fix $\beta=0$ to ensure that the input signals of neurons are not moved too far outside the range of $[-4, 4]$. We train the models in two settings with or without pretraining. For DBM and BNDBM, we follow the pretraining process proposed in Salakhutdinov and

Classification ↑	**PCD (500-100)**			**PCD (500-500)**			**CD^{prob}(500-100)**		
Pretraining	**DBM**	**cDBM**	**BNDBM**	**DBM**	**cDBM**	**BNDBM**	**DBM**	**cDBM**	**BNDBM**
MNIST	94.81 ±0.58	93.89 ±0.19	**95.37** ±0.19	96.68 ±0.36	96.63 ±0.15	**96.80** ±0.11	11.35 ±0.00	85.56 ±3.25	**90.31** ±0.002
Fashion-MNIST	81.27 ±1.07	73.60 ±2.7	**81.66** ±0.76	69.88 ±8.79	83.07 ±0.40	**84.70** ±0.38	15.92 ±6.60	71.97 ±2.12	**72.67** ±2.59
Caltech 101 Silhouette	65.27 ±0.24	58.39 ±0.58	**65.63** ±0.25	66.54 ±0.65	65.96 ±0.42	**69.20** ±0.45	28.08 ±3.13	54.40 ±0.99	**62.94** ±0.19
No pretraining	**DBM***	**cDBM***	**BNDBM***	**DBM***	**cDBM***	**BNDBM***	**DBM***	**cDBM***	**BNDBM***
MNIST	11.35 ±0.00	84.35 ±2.13	**93.81** ±0.58	12.22 ±2.74	94.05 ±0.08	**96.50** ±0.11	25.32 ±11.84	61.84 ±3.42	**85.73** ±0.81
Fashion-MNIST	16.14 ±9.91	71.64 ±1.15	**76.26** ±1.20	35.66 ±8.39	**82.27** ±0.27	78.75 ±0.99	12.31 ±0.66	68.25 ±3.19	**71.92** ±1.09
Caltech 101 Silhouette	19.77 ±4.78	54.70 ±0.78	**59.45** ±1.20	23.65 ±1.20	64.63 ±8.12	**66.80** ±0.69	35.34 ±3.98	55.95 ±0.79	**60.99** ±0.31
Our average improvement	**vs DBM**	**vs cDBM**		**vs DBM**	**vs cDBM**		**vs DBM**	**vs cDBM**	
Pretraining	0.44	5.60		5.86	1.67		56.95	4.76	
No pretraining	60.75	6.28		56.86	0.38		48.56	10.87	

Table 1: The means and standard deviations of all methods in classification accuracy (%) over 10 runs. The methods marked with * are trained with no pretraining. The best number is in bold while the next best is underlined. The last two rows show the average of *improvements* over datasets, where an *improvement* of BNDBM vs a method A is their difference. The positive signs of *improvement* values show BNDBMs outperform A whilst negative signs indicate A is better.

Hinton (2009) with 100 epochs to initialize the model parameters. For cDBM, we adopt the implementation[1] with two-stage pretraining technique Cho et al. (2013b). To examine models quantitatively, we use the last hidden layer as feature representation and compare their classification accuracy (using Logistic Regression classifiers) and reconstruction error.

Besides using PCD Tieleman (2008) to train the models, we employ CD method Hinton (2002), which is well-known for its high reconstruction capacity. Furthermore, to minimize the reconstruction error in CD training, when computing the state of a layer, we ignore sampling steps and instead pass the conditional probability values directly as input signals to the next layers (we name this CD version CD ^{prob}). In our experiments, we observe that the methods trained with CD ^{prob} obtain lower reconstruction error than with PCD. However, due to no sampling steps, models using this training scheme become deterministic networks and they do not follow the probabilistic framework mentioned in Sec. 2. As a result, this strategy indeed works as a feature extractor, where data information is transferred from the visible layer to the higher layers. Two-way interactions between adjacent layers guarantee that the next layer is the higher representation of the current layer rather than just a meaningless mapping. Since sigmoid function tends to compress values out of $[-4, 4]$ into outputs around 0 or 1 (Fig. 1), we maintain the input signal distribution with the mean of $\beta = 0$ and the standard deviation of $\gamma = 1$ to fit it into [-4, 4] and therefore preserve the information between layers. The setting is widely-used in CNNs Ioffe and Szegedy (2015).

Since we aim to learn data representations under a completely unsupervised manner, we do not use labels to train models in all experiments. In particular, we first train DBMs to

1. https://github.com/kyunghyuncho/deepmat

Reconstruction	**PCD (500-100)**		**PCD (500-500)**		**CD^{prob}(500-100)**	
Average improvement	**vs DBM**	**vs cDBM**	**vs DBM**	**vs cDBM**	**vs DBM**	**vs cDBM**
Pretraining	-1.13	-1.50	-0.59	-0.64	**0.29**	**1.55**
No pretraining	-0.98	-1.86	**1.02**	-1.34	**0.06**	**3.57**

Table 2: The average reconstruction error *improvement* of BNDBM versus DBM and cDBM over all datasets. The bold numbers (positive values) indicate improvement in reconstruction error.

obtain new representations for data, i.e., the activations of last hidden layer, and then use a simple classifier (i.e., Logistic Regression) to *evaluate* the quality of these representations. These two steps are performed separately. We also do not fine-tune the networks Salakhutdinov and Hinton (2009) because it will update the model parameters, hence update the representations, towards minimizing the classification error. As a result, the learned representations are driven by the labels, hindering us from evaluating their quality accurately. It is worthy to note that our unsupervised setting is different from supervised training for classification in Salakhutdinov and Hinton (2009) where supervised DBMs are trained with label variables on top and then discriminatively fine-tuned towards low classification error. Similarly, our experimental setting is also different from semi-supervised learning in Sankaran et al. (2017) where DBMs are trained with class labels for classification tasks.

Table 1 and 2 report the classification and reconstruction results over 10 runs. Overall, for PCD training, BNDBM is the best method for classification and has comparable quality in reconstruction task. When trained with CD ^{prob}, high classification accuracy (at least 4.76% and 10.87% better than DBM and cDBM in average) shows that batch normalization helps preserve the information between the layers very well. Higher layers in BNDBM try to interpret the data exactly but at more abstract levels. As we expected, BNDBM also achieves excellent reconstruction error, at least 0.06 and 0.29 smaller than DBM and cDBM on average (Table 2). For computation time, since BNDBM requires extra normalization cost in every mean-field and Gibbs sampling step, DBM with normalization is usually 3.13 times slower than DBM in training phase.

Compared with the supervised DBM Salakhutdinov and Hinton (2012), our accuracy of 95.37% ($500-100$ units) and 96.80% ($500-500$ units) on MNIST is lower than supervised DBM using 100% (60,000) labeled training data. However, if 1% labeled training data is used for fine-tuning, supervised DBM with fine-tuning only obtains 95.18% in Salakhutdinov and Hinton (2012) that is lower than our unsupervised BNDBM without any fine-tuning step. This interesting result shows the better capacity of BNDBM in unsupervised feature learning over DBM. This comparison between supervised and unsupervised training also reveals how batch normalization improves DBM's performance in unsupervised training. More specifically, when data labels are added, the highest hidden layer has to align with the label layer during training and therefore it has meaningful representation. Without the label layer, it works lazily and has many dead neurons. Batch normalization helps to motivate it and encourage it to response actively to the data.

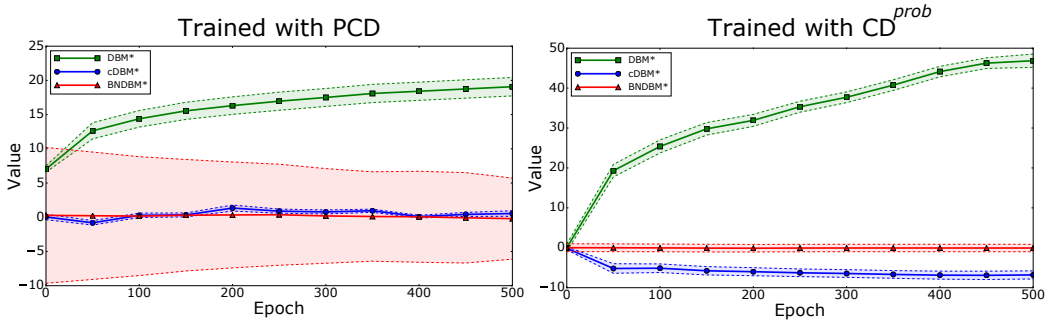

Figure 2: Sigmoid input distributions of a random unit (unit 63, layer 2, network $500-100$) of three methods without pretraining over epochs. The distribution means are drawn as a solid lines while the dash lines with the same color represents the standard deviations. The experimental numbers were recorded in MNIST.

4.1. Training facilitation

Training DBMs is very challenging and most DBM variants depend on "easy-to-train" models such RBMs or AEs in a pretraining step to initialize their parameters. Table 1 also shows the performance of DBM, cDBM and BNDBM without pretraining (the methods with *). We are unable to train DBM successfully without pretraining and this agrees with the observations in Cho et al. (2013b); Melchior et al. (2016). By contrast, our BNDBM with batch normalization still obtains high classification accuracy even if the pretraining phase is not applied. Compared with post-normalization in cDBM, BNDBM* shows 5.84% better than cDBM* on average over all datasets and learning methods. Training without pretraining is a very nice capacity since it proves that our BNDBM* is "easy-to-train" enough to stand alone without the help of pretraining methods.

To understand the effect of normalization in BNDBM*, for every 50 epochs, we run a mean-field approximation to estimate $p(\boldsymbol{h}|\boldsymbol{v})$ on the MNIST testing set and observe the inputs of a randomly picked hidden unit. Fig. 2 demonstrates the change of the input distributions during DBM*, cDBM* and BNDBM* training. It can be seen that these signals change dramatically in DBM* and cDBM* but they are more stable in BNDBM*. The more unchanged signals are, the less dependence between layers is. The movement of input distributions in DBM* toward large positive values reveals that DBM* learning gets stuck in a poor local optimum as described in Sec. 1. For cDBM*, its post-normalization indirectly controls input signals via normalizing the states of adjacent layers and then reducing the variations of sigmoid inputs. However, this unlikely guarantees to produce stable layer signals and therefore there still exists a fluctuation (in PCD) or a slight shift (in CD ^{prob}) of input distributions during cDBM* training (Fig. 2). By contrast, BNDBM* allows these distributions to be consistent and controllable during training. As a result, we can easily trained BNDBM from randomly initialized parameters and still obtain a high quality model.

4.2. Feature representation improvement

BNDBM shows an excellent capacity of unsupervised feature learning by producing good representation, which wins classification task in most datasets and training methods. These

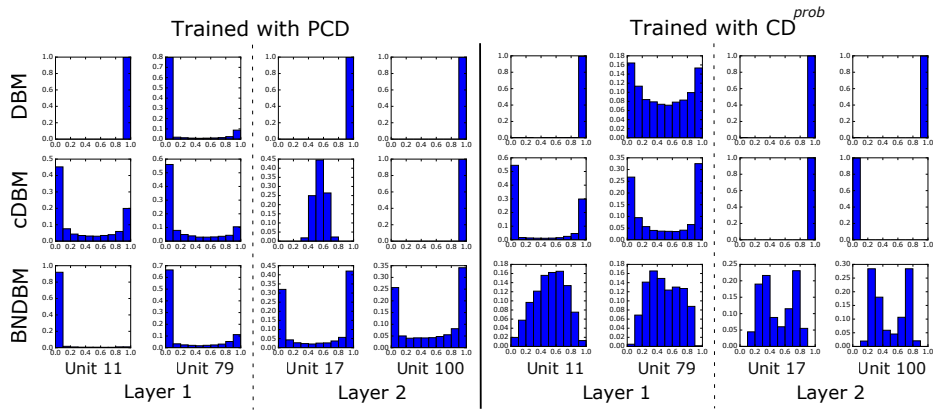

Figure 3: Sigmoid output distributions of some hidden units in DBM, cDBM and BNDBM.

features can be obtained via running a mean-field procedure, where the sigmoid outputs at the last layer are considered as the new representation of the data.

Fig. 4 visualizes the activations (sigmoid outputs) at the 1^{st} and 2^{nd} hidden layers of the network $500 - 100$ in forms of $10,000 \times 500$ and $10,000 \times 100$ (#data points $\times$ #neurons) matrices respectively corresponding to all $10,000$ MNIST testing samples. For the visualization purpose, we resize these matrices into the same image size to best fit in the figure. Each element (i, j) in a matrix is the activation of the j^{th} neuron with respect to the i^{th} data point. Vertical homogeneous strips in each matrix, which indicate the corresponding neurons respond to all test data in a similar way, appear frequently in the non-normalized hidden layers of DBM and cDBM, whilst the normalized layers of BNDBM produce more random and heterogeneous patterns. The randomness and diversity in these patterns reflect that neurons in BNDBM are trying to distinguish data samples and cooperating with the others (rather than copying their behaviors). As a result, BNDBM, by diversifying layer signals, can produce richer feature representation of raw data and more distinctive states at the hidden layer than DBM and cDBM.

By taking the histogram of a matrix column in Fig. 4, we can observe the distribution of hidden activations at a particular unit. Fig. 3 visualizes the output of the sigmoid functions observed at four hidden neurons after training. Unlike DBM that tends to produce output values in a narrow interval (around 0 or 1), by controlling the sigmoid input signals, BNDBM guarantees that the distributions cover the majority of $[0, 1]$ and thus it enhances the distinction between class representation. Meanwhile, although the post-normalization strategy in cDBM can also make sigmoid outputs diverse (e.g., unit 17, layer 2, PCD training in Fig. 3), it does not solve the problem of the layer input change completely (e.g., unit 100, layer 2, PCD training or units 17 and 100, layer 2, CD training in Fig. 3). This explains why post-normalization in cDBM cannot obtain as high accuracy as pre-normalization in BNDBM. The visualization in Fig. 3 also points out that the phenomenon of distribution collapse happens more severely in higher layers than in lower layers (i.e. the first hidden layers). This is because the first hidden layer directly connects with data and thus aligns with the good distribution of data.

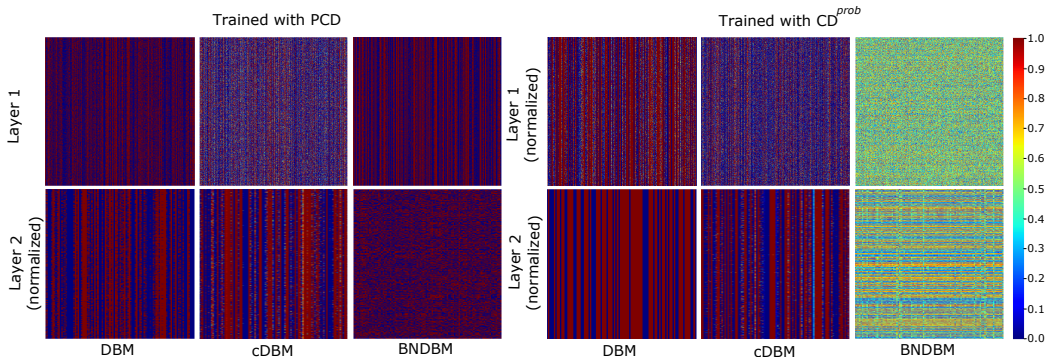

Figure 4: Activations of 10,000 MNIST test samples over hidden layers. Each image represents a matrix of 10,000 rows and (# hidden units) columns in the corresponding layers. These matrices are resized into square form for better visualization.

4.3. Batch normalization in DBMs vs CNNs

After our experiments, we realize the significant differences in the use of batch normalization in deterministic networks such as CNNs, RNNs or BNDBMs trained with CD ^{prob}, where sampling steps are removed, and probabilistic networks such as BNDBMs. Therefore, this section highlights these differences and provides a guideline to train BNDBMs.

Non-normalized $\boldsymbol{h}^{(1)}$: In the former, it is possible to normalize the first hidden layer. However, for DBMs, we cannot obtain good results when $\boldsymbol{h}^{(1)}$ is normalized. Table 3 shows the classification accuracy of BNDBM over different settings of normalizing $\boldsymbol{h}^{(1)}$, $\boldsymbol{h}^{(2)}$ and both $\boldsymbol{h}^{(1)}$and $\boldsymbol{h}^{(2)}$. To clearly observe the impact of normalization, the network is trained without the pretraining step. It is interesting that normalizing $\boldsymbol{h}^{(2)}$ boosts the performance whilst normalizing $\boldsymbol{h}^{(1)}$ causes negative effect. We believe that this is because the first hidden layer in probabilistic networks, as the only hidden layer communicating directly to the visible layer, has more important role to model the data distribution than higher hidden layers. An example is Restricted Boltzmann Machines that require only the first hidden layer to learn data distribution successfully. When more constraints are added to $\boldsymbol{h}^{(1)}$ via batch normalization, the model becomes less flexible to learn the data distribution and shows a decline in performance. However, further research should be conducted to understand the role of each hidden layer comprehensively. It is noteworthy that when training BNDBM with CD ^{prob}, where the capacity of distribution modeling is ignored, normalizing the first hidden layer is recommended. This explains why we avoid normalizing $\boldsymbol{h}^{(1)}$ for BNDBM trained with PCD but normalize all layers in CD ^{prob} in all experiments (of Tables 1 and 2).

Parameters γ and β: In deterministic networks, these parameters are usually fixed ($\gamma = 1$ and $\beta = 0$) to make the layer's input distributions unchanged between iterations. Meanwhile, since DBMs usually include sampling steps, i.e., $h_i^{(l)} \sim \text{Bernoulli}(\mu_i^{(l)})$ where $\mu_i^{(l)} = \sigma\left(\mathcal{B}_l\left(t_i^{(l)}\right)\right)$, γ and β do not only control the layer's input distributions but also specify the parameters $\mu_i^{(l)}$ of Bernoulli distributions, where the states of neurons are drawn from. If $\gamma = 1$ and $\beta = 0$, the Bernoulli parameter $\mu_i^{(l)}$ of a hidden unit $h_i^{(l)}$ has 50% to be < 0.5 and 50% to be > 0.5. It means that all hidden units always have equal probabilities to

	$\boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)}$	$\boldsymbol{h}^{(1)}$ **only**	$\boldsymbol{h}^{(2)}$ **only**
MNIST	82.48 ±0.64	45.67 ±6.91	**96.50** ±0.11
Fashion-MNIST	66.00 ±0.88	54.64 ±7.17	**78.75** ±0.99
Caltech 101 SH	51.54 ±0.61	24.84 ±3.62	**66.80** ±0.69

Table 3: Accuracy (%) when normalizing all layers and each layer in the BNDBM* network of $500-500$.

Learnable γ	**False**		**True**	
Learnable β	**False**	**True**	**False**	**True**
MNIST	86.47 ±4.00	85.73 ±1.12	**96.50** ±0.11	85.11 ±0.11
Fashion-MNIST	77.16 ±2.09	78.73 ±1.27	**78.75** ±0.99	78.28 ±1.43
Caltech 101 SH	64.50 ±1.53	65.28 ±0.92	**66.80** ±0.69	63.52 ±1.07

Table 4: Accuracy (%) when training the BNDBM* network ($500-500$) with/without learnable γ and β.

be on and off. This is not a good idea because some hidden states corresponding to dominant data points should be activated more frequently than other states. For this reason, γ should be learned during PCD training whilst we fix $\beta = 0$ to ensure that a unit has both on and off states and therefore, it does not become an "always-on" or "always-off" neuron. The experimental results in Table 4 confirm the importance of choosing a right training setting for normalization parameters in probabilistic networks, where the best accuracy is obtained using learnable γ and fixed $\beta = 0$.

Population means and variances: Since signals are passed many times between layers during mean-field or Gibbs sampling in BNDBMs, each step of mean-field or Gibbs sampling requires a pair of population mean and variance to normalize layer distributions. As a result, BNDBMs need to store $\max(N_{\text{mf}}, N_{\text{gs}})$ pairs per neuron. This contrasts with one pair of mean and variance per unit in most deterministic networks.

Overall, these differences suggest that we should pay attention to the probabilistic nature of DBMs whenever applying any deep learning technique to DBMs.

5. Conclusion

In this paper, we discussed the difficulty of DBM training and the underlying reasons. Due to bi-directional connections in DBMs, layers' input signals change significantly and unpredictably between training epochs. The deeper layer is, the more its input varies and more serious this problem is. This results in more tips and tricks proposed in the literature in order to train DBMs. To address this problem, we proposed to standardize layers' inputs by integrating batch normalization operators into DBMs. This is done by introducing a novel energy function to obtain the normalization form of layers' input signals. Extensive experiments on MNIST, Fashion-MNIST and Caltech 101 Silhouette confirm that our proposed BNDBM method can be trained easily without any pretraining step and has more powerful feature representation than DBM and cDBM via controlling the layers' input distributions.

References

KyungHyun Cho, Tapani Raiko, and Alexander Ilin. Gaussian-Bernoulli Deep Boltzmann Machine. In *IJCNN*, pages 1–7, Dallas, TX, USA, August 4-9 2013a.

KyungHyun Cho, Tapani Raiko, Alexander Ilin, and Juha Karhunen. A Two-Stage Pretraining Algorithm for Deep Boltzmann Machines. In *ICANN*, September 10-13 2013b.

G.E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800, 2002.

Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for Deep Belief Nets. *Neural Computation*, 18(7):1527–1554, 2006.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, number 11, 1998.

Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y. Ng. Convolutional Deep Belief Networks for scalable unsupervised learning of hierarchical representations. In *ICML*, pages 609–616, New York, NY, USA, 2009. ISBN 978-1-60558-516-1.

Benjamin M. Marlin, Kevin Swersky, Bo Chen, and Nando de Freitas. Inductive principles for Restricted Boltzmann Machine learning. In *AISTATS*, Italy, May 13-15 2010. PMLR.

Jan Melchior, Asja Fischer, and Laurenz Wiskott. How to center Deep Boltzmann Machines. *Journal of Machine Learning Research*, 17(99):1–61, 2016.

Gregoire Montavon and Klaus Robert Muller. Deep Boltzmann Machines and the centering trick. In *Neural Networks: Tricks of the Trade - Second Edition*, pages 621–637. 2012.

Gregoire Montavon, Mikio Braun, and Klaus-Robert Muller. Deep Boltzmann Machines as feed-forward hierarchies. In *AISTATS*, 21–23 Apr 2012.

Jiquan Ngiam, Zhenghao Chen, Pang Wei Koh, and Andrew Y. Ng. Learning Deep Energy Models. In *ICML*, pages 1105–1112, 2011.

M. Norouzi, M. Ranjbar, and G. Mori. Stacks of convolutional Restricted Boltzmann Machines for shift-invariant feature learning. In *CVPR*, June 2018.

Ruslan Salakhutdinov and Geoffrey Hinton. Deep Boltzmann Machines. In *AISTATS*, volume 5, pages 448–455, 2009.

Ruslan Salakhutdinov and Geoffrey Hinton. An efficient learning procedure for Deep Boltzmann Machines. *Neural Computation*, 24(8):1967–2006, Aug 2012.

Anush Sankaran, Gaurav Goswami, Mayank Vatsa, Richa Singh, and Angshul Majumdar. Class sparsity signature based Restricted Boltzmann Machine. *Pattern Recognition*, 2017.

Tijmen Tieleman. Training Restricted Boltzmann Machines using approximations to the likelihood gradient. In *ICML*, pages 1064–1071, New York, NY, USA, 2008.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. 2017.

Bi Xiaojun and Wang Haibo. Contractive Slab and Spike Convolutional Deep Boltzmann Machine. *Neurocomputing*, 290:208 – 228, 2018.