Bidirectional vision through Deep Boltzmann Machine boosted with conjugate neural networks

Shaobo Guan
Department of Neurocience
Brown University
Providence, RI, 02906
`Shaobo_Guan@brown.edu`

Abstract

We created a hierarchical MRF (deep Boltzmann machine, or DBM) that does both discriminative classification (bottom-up) and generative sampling (top-down) tasks, as the bidirectional biological visual system. Compared with the origianl DBM, we used two novel techniques to improve the performance: 1) the important weightening effectively balanced the layers and help to implement a sufficient amount of top-down influence; 2) We boosted the DBM learning using the response of a conjugate feed-forward neural network, the improved efficiency makes it possible to be applied to very deep networks.

1 Introduction

1.1 Motivation and Goal

The biological visual system is a powerful computing machine, which effortlessly comprehends complicated visual scenes, surpassing any artificial system. The recent surge of the feed-forward deep convolutional neural networks makes exciting progress in object classification, partially approximates the feed-ward "sweep" of information processing in the brain. However, the brain does much more than just the feed-forward sweep: there are as many feedback connections as the feed-forward ones. To make a better artificial visual system and a better model of the biological visual system, I plan to incorporate both feed-forward and feedback processing into one single model, so that it does not only perform classification, but also work in the following three generative tasks that the discriminative feedforward networks can not achieve: (1) to apply inductive bias (top-down influence), (2) recovers occlusion and remove noise (pattern completion) and (3) generates samples (imagination).

1.2 General Approaches

The undirected probabilistic graphical model, or Markov Random Field (MRF), is a promising candidate for implementing the bidirectional visual processing. If we can construct a hierarchical MRF that models the joint distribution of the image and the label, the three listed tasks can be naturally solved. Because MRF is a generative model, the three tasks can be respectively computed as (1) the probability distribution of image pixels conditioned on the label, (2) the probability distribution of the occluded pixels conditioned on the visible pixels, and (3) the distribution of the image pixels marginalized over the labels.

The question now is to select structure of MRF and determine the learning algorithm

1.3 Inspiration and Hypothesis

Restricted Boltzmann Machine (RBM) [3] is special case of MRF: RBM can be viewed either as a neural network or a structured MRF of an exponential family distribution. Models composed several layers of RBM, become deep Boltzmann machine (DBM)[4], or deep belief net (DBN)[3], depending on the way they are stacked together. They are able learn complex features though unsupervised learning. In the inference time, the model is capable of both classification and the three generative tasks.

Despite these advantages, the learning of DBM/DBN can be very challenging, making it difficult to apply to large scale network/data. This proposed project to explore the potential of boosting the learning using a feedforward neural network (FNN), which can be efficiently learned through back propagation.

Although the feedforward neural network is discriminatively trained to fit the output label, the learn features of the hidden nodes turn out to be reasonable basis for image reconstruction and they approximates the biological neurons' preference [1]. Given such success, I plan to construct a MRF that shares the same structure of such FNN and test the hypothesis whether we can take directed weights learned from FNN and use it to construct the undirected weights of MRF. The detailed method is decribed in section three.

2 Related work

2.1 Use feedforward neural network (FNN) to construct images

Using feedforwad neural network to construct images have been carried out for different purposes: (1) to visualize the preference of a hidden node (2) to understand how the network works and (3) to generate new images. Among which, Gatys el. al. presented a state-of-the-art texture synthesis algorithm using FNN [2]: to generate a texture, they first apply a learned FNN on a seed texture image and get the response of all nodes up to a certain layer, select a set of summary statistics of the FNN's response, and then initialized a random noise image and use gradient decent to adjust the noise image to match the summery statics of the seed image.

This work and similar lines of work directly took advantage of FNN and use gradient decent based method to infer the input based on the output. There are two problems here: (1) The generated images are not true samples. (2) It is easy to get the "fooling images", i.e. images that are completely unrecognizable to humans but FNN believe to be recognizable objects with high confidence.

We hope our energy based MRF can overcome these two problems

2.2 unsupervised structured MRF

Hinton's group has made several thrusts into learning structured MRF via unsupervised learning. e.g. restricted Boltzmann machine (RBM) and deep belief net (DBN) [3]. Those are true generative models, and in principle, can perform the three generative tasks. The problem here is that every the training step requires Gibbs sampling and thus every computationally expensive and hard to parallelize. Therefore it is difficult to apply them to a complicated deep network structure. What is more, because the model is trained in the unsupervised manner, the model only learns a representation of the data, but not able to do classification. It would be great if a model could do generative and discriminative tasks.

3 Graphical Model formulation, Learning and Inference

3.1 DBM graph structure

In this project, we propose a modified version of deep Boltzmann machine (DBM), as is shown in Figure 1.a. The graph is a pairwise MRF with Bernoulli stochastic nodes. The nodes forms layered structure where edges only exist between neighboring layers, just like the FNN . While the original DBM model only contains one visible layer coding the input (e.g., image), our model contains two

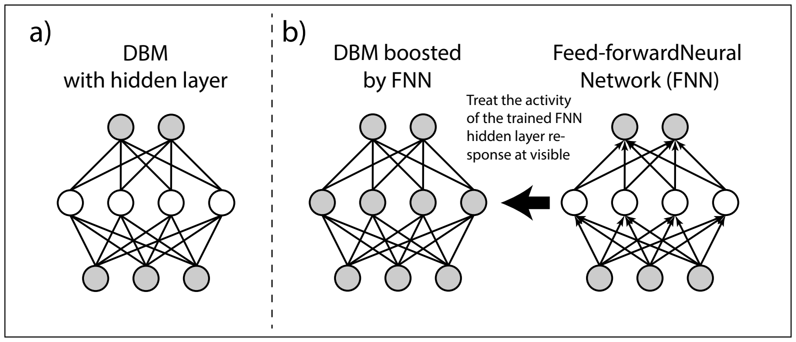

Figure 1: Graph structure: a) deep Boltzmann machine with hidden layers, b) deep Boltzmann machine boosted with a conjugate feed forward neural network

visible layers: the first layer codes the input (e.g. image) and last layer codes the the class label, and the other layer(s) in between are hidden layers.

3.1.1 DBM model description

Suppose the graph contains H layers, and every layer h contains M_h nodes. In layer h, the response of the node i is denoted as $X_{h,i} \in \{0,1\}$, and the response of all nodes of layer h is denoted as $X_h \in \{0,1\}^{M_h}$. Every node $X_{h,i}$ has a unitary bias parameter $b_{h,i}$ and every pair of connected nodes $X_{h_i,i}, X_{h_j,j}$ has a pairwise parameter $W_{(h_i,i),(h_j,j)}$; thus $b_h \in \mathbb{R}^{M_h}$ represents all the bias terms of layer h, and matrix $W_{h_i,h_j} \in \mathbb{R}^{(M_{h_j},M_{h_i})}$ represents all the pair wise connection weights between layer h_i and h_j. All the parameters of the model are denoted as $\theta = (b, W)$

The energy of MRF is determined by

$$E(X;\, \theta) = -\sum_h b_h^T X_h - \sum_{h_j - h_i = 1} X_{h_j}^T W_{h_i,h_j} X_{h_i} \tag{1}$$

The joint probability of the MRF is governed by:

$$p(X;\, \theta) = \frac{\exp^{-E(X;\, \theta)}}{Z(\theta)}, \quad \text{where } Z(\theta) = \sum_X \exp^{-E(X;\, \theta)} \tag{2}$$

The joint probability of the network is intractable to compute exactly. This because of the normalization term $Z(\theta)$ needs $\mathcal{O}(2^N)$ steps to compute, where N is number of nodes in the network, which is typically very large. However, due to the layered design, the nodes in the same layer are conditional independent given the neighboring layers, making the conditional provability easy to compute. Let p_h be the vector of conditional probability:

$$p_h = \Big[p(X_{h,1} = 1|\, X_{h-1}, X_{h+1}),\ p(X_{h,2} = 1|\, X_{h-1}, X_{h+1}), \ldots \quad p(X_{h,M_h} = 1|\, X_{h-1}, X_{h+1})\Big]$$

The conditional probability (given X_{h-1} and X_{h-1}) has a simple form:

$$p_h = \sigma\Big(b_h + W_{h-1,h} X_{h-1} + W_{h,h+1}^T X_{h+1}\Big) \quad \text{where } \sigma \text{ is the logistic function} \tag{3}$$

Based on this, the true likelihood $p(X)$ can be approximated using pseudo likelihood $\tilde{p}(X)$:

$$p(X) \approx \tilde{p}(X) = \prod_h p(X_h | X_{h-1}, X_{h+1}) \tag{4}$$

Where the conditional probability is given by p_h

3.1.2 Importance weighted iDBM model

The above described DBM is the standard DBM model except that we make the highest layer code class label, whereas previous applications of DBM treats the highest layer as hidden feature extractors.

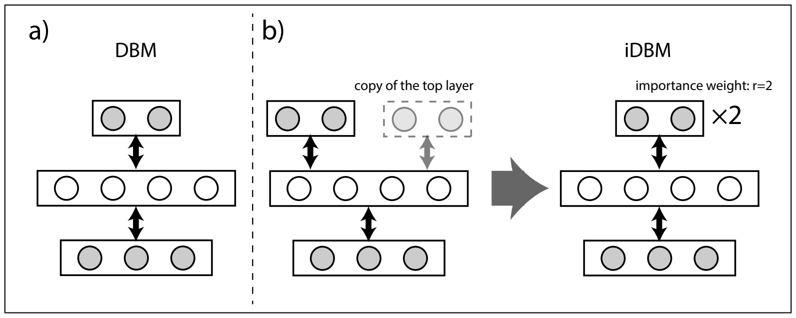

Figure 2: importance weighted iDBM: a) regular DBM, b) iDBM as a convenient equivalent form of DBM with multiple copies of nodes at some layers

However, we noticed a problem of using the top layer as class labels. There are typically much more nodes in the bottom layer (e.g. 784 pixels of images) compared with the top layer (10 class labels of images). Intuitively, flipping all ten class label nodes and flipping ten pixel nodes will cause a comparable amount of perturbation to the energy function, however the former case should be much more severe compared with the latter case.

To deal with this problem caused by uneven layer size, we proposed a novel variation of the original model – importance weighted DBM, or iDBM

One direct solution could be to add more copies of nodes to those layers with fewer nodes, where the added copies share the exact same parameters as the original copy. This is equivalent as to introduce a importance-weight factor r_h for layer h to the energy function, the new energy function becomes:

$$E(X;\,\theta) = -\sum_h r_h b_h^T X_h - \sum_{h_j - h_i = 1} r_{h_i} r_{h_i} X_{h_j}^T W_{h_i,h_j} X_{h_i} \tag{5}$$

and the conditional probability becomes:

$$p_h = \sigma\Big(b_h + r_{h-1} W_{h-1,h} X_{h-1} + r_{h+1} W_{h,h+1}^T X_{h+1}\Big) \tag{6}$$

This simple trick turns out to be very effective in our experiments

3.2 Inference Algorithm

In this project, we use Gibbs sampling for inference. Gibbs sampling is usually very slow to converge; making it not practical to apply to a model as large as the proposed graph. However, because of the layered structure and the conditional independence, we can do blocked Gibbs sampling one layer at a time, enabling to us sample very efficiently in parallel. The inference algorithm is as follows:

```
for i in num_Gibbs_cycles:          % number of Gibbs cycles
  for h in a randoms order in H:    % pick a layer
    if X_h is visible:
      pass
    elif X_h is hidden:
      calculate p_h given_neighboring_layers
      sample binary variables X_h using p_h
```

Based on different scenarios of visible hidden nodes, the model could use the exact same inference algorithm to perform a variety of tasks:

1. **recognition / classification**
   Only the bottom layer (pixel) is visible; infer the top layer (class label) response
2. **generate class-specific samples**
   Only the top layer (class label) is visible; infer the bottom layer (pixel label) response
3. **generate random samples**
   All layers are hidden

4. **pattern completion**
   Only a fraction of the bottom layer (pixel) is visible; infer the rest of the bottom layer

5. **pattern completion under top-down influence**
   The top layer and a fraction of the bottom layer (pixel) is visible; infer the rest of the bottom layer

3.3 DBM, Learning Algorithm

Note that the proposed model belongs to the exponential family, therefore the gradient of the log-likelihood has a simple form:

Suppose we have L observations, let $X_v^{(l)}$ denote the visible/observed nodes of data l

$$\begin{aligned}\frac{\partial \sum_l \log p(X^{(l)};\theta)}{\partial b_{h,i}} &= \sum_l \mathbb{E}_{X_v^{(l)},\theta}[X_{h,i}] - L\mathbb{E}_\theta[X_{h,i}] \\ \frac{\partial \sum_l \log p(X^{(l)};\theta)}{\partial W_{(h_i,i),(h_j,j)}} &= \sum_l \mathbb{E}_{X_v^{(l)},\theta}[X_{h_i,i}X_{h_j,j}] - L\mathbb{E}_\theta[X_{h_i,i}X_{h_j,j}]\end{aligned} \tag{7}$$

In the above equation, $\mathbb{E}_{X_v^{(l)},\theta}[.]$ represents the conditioned expectation of the sufficient statistics given observed nodes $X_v^{(l)}$, whereas $\mathbb{E}_\theta[.]$ represents the unconditioned expectation of the sufficient statistics of the model.

In practice, the learning is usually done using mini-batch stochastic gradient decent methods, to efficiently handle the computational cost of a single mini-batch.

In practice, both expectations can be computed using the contrastive divergence algorithm [5], which is an efficient Gibbs sampling procedure for this type of graphical model. Basically, the network will compute the conditional and unconditional expectation using only a few (less than ten) Gibbs iterations.

3.4 bDBM: FNN-boosted DBM and Learning Algorithms

The challenge to learn a DBM is that both the hidden nodes and the model parameters are to be determined. The learning would be much easier if the hidden nodes become visible nodes. This is just like the Gaussian-mixture EM learning: if the label assignment becomes known, the learning becomes straight-forward.

In order to obtain a reasonable response of hidden layer nodes, we turn to the feed-forward neural network, FNN. We have the very efficient back-propagation algorithms to train a FNN, and the hidden nodes in the successfully trained FNN is shown to be good feature extractors. Here we would explore the possibility of using these feature extractors to boost the learning of DBM.

Our proposed method is as follows (Figure 1, b):

1. construct a FNN that shares the same structure (using logistic activation function) as the DBM, and train it using back-propagation

2. Evaluate the FNN with input images and get the response of the hidden nodes as values between [0,1], and then produce iid. Bernoulli samples using the response

3. Treat these samples as the observed data for training the DBM. With fully observed data, the conditional expectation term of Equation (7) becomes the mean sufficient statistics of the data.

Actually, the conjugate FNN could be viewed as a layer-wise greedy mean-field approximation of the scholastic DBM model.

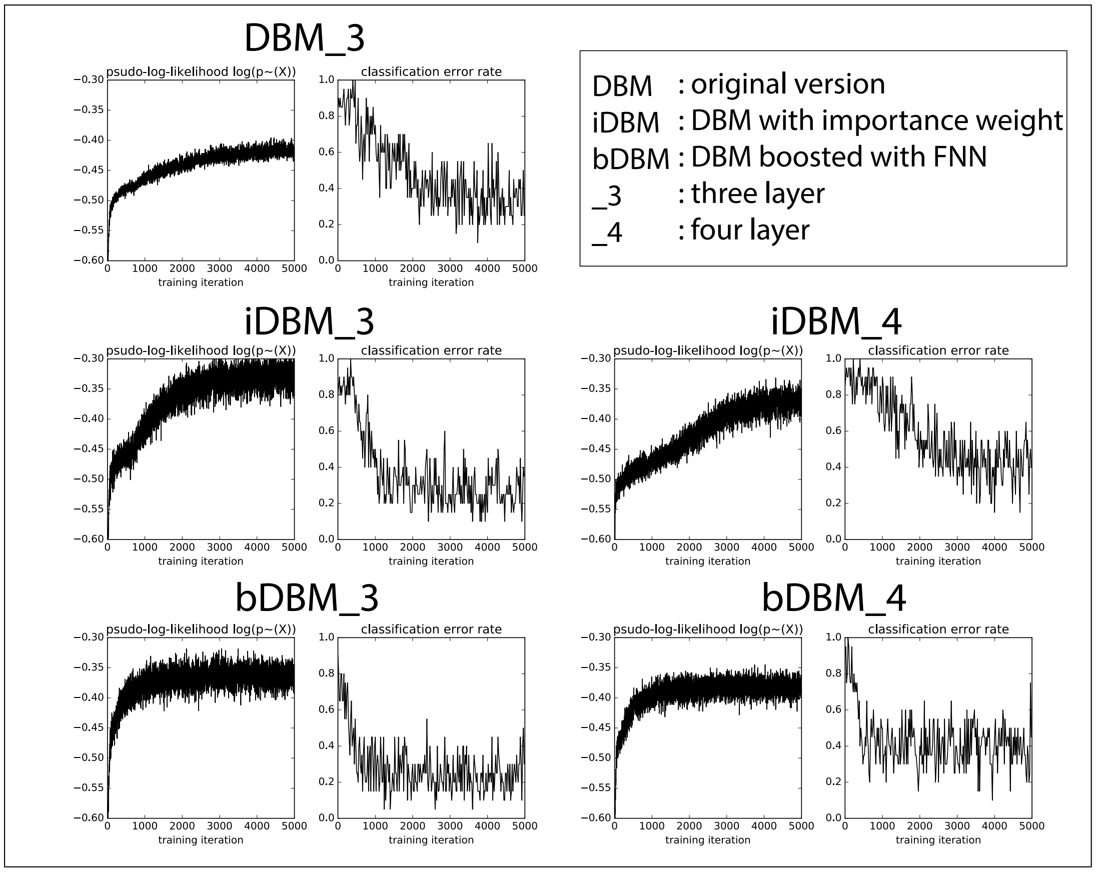

Figure 3: **Psudo log likelihood and classification error during training.** The five panels corresponds to the five experimental conditions in the text (each shows one run). The psudo log likelihood is calculated as in Equation (4)

4 Experiments on MNIST dataset

4.1 Experimental conditions

Data set: The original NMIST dataset contains hand-written 0 9 digits, where every image contains 28*28 pixels of gray-scale variables. To make it compatible with DBM, the gray-scale values are binarized with threshold 0.3; to speed up the experiments, we down-sampled the image to 14*14 pixels. The class label is coded as binary arrays by the top layer, e.g. class 2 is coded as [0,0,1,0,0,0,0,0,0,0]

Coding environment: Python with numpy package. The DBM model, the inference and the learning algorithms are all implemented by the author. The FNN training is done using sklearn package.

We did experiments under the following conditions and compared the results

1. **DBM** with 3 layers (1-hidden), no important weights, layer size [196,196,10]
2. **iDBM** with 3 layers (1-hidden), with important weights of the top layer $r = 10.0$, layer size [196,196,10]
3. **bDBM** with 3 layers (1-hidden), booted with FNN, with important weights of the top layer $r = 10.0$, layer size [196,196,10]
4. **iDBM** with 4 layers (2-hidden), with important weights of the top layer $r = 10.0$, layer size [196,196,64,10]
5. **bDBM** with 4 layers (2-hidden), booted with FNN, with important weights of the top layer $r = 10.0$, layer size [196,196,64,10]

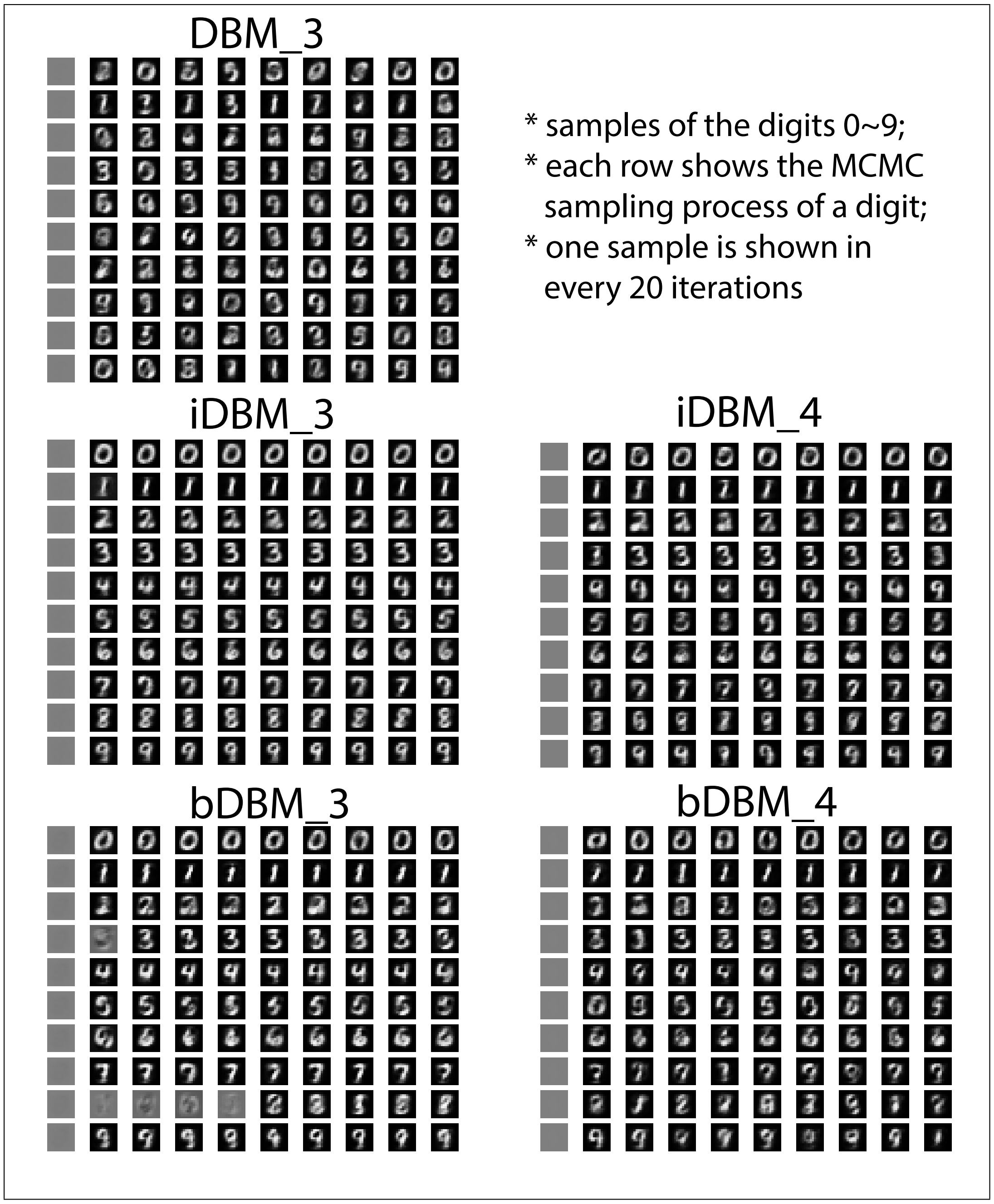
DBM_3
* samples of the digits 0~9;
* each row shows the MCMC sampling process of a digit;
* one sample is shown in every 20 iterations
iDBM_3
iDBM_4
bDBM_3
bDBM_4

Figure 4: **Class-specific samples generated in MCMC process** The top layer was given as specific class label arrays, and the the bottom layers are sampled through MCMC.

The gradient descent step size, L2 regularization coefficient and the minibatch size are chosen empirically and kept the same across the five experimental conditions.

All the training process takes less then 5 min on MacBook Pro running Python2.7

4.2 Training convergence and classification (bottom-up) performance

In order to tell how fast the training converges and how the model does bottom-up classification, I plotted the pseudo log likelihood and the the classification error rate along the training iterations, as is shown in Figure (3). We find:

- Importance weighted iDBM converges faster and do classification better than the original DBM. This suggest that the importance weight is an efficient way to balance the layer size and produce better results.
- FNN boosted bDBM converges faster then iDBM, especially in the four-layer model. The benefit of the boosting could be bigger in a deeper network.
- In the end, the FNN boosted bDBM does not produces a higher pseudo likelihood, suggesting the boosting benefit is more about the speed but not the final results

4.3 Class-specific sampling (top-down) performance

Other than the convergence and the classification performance, we examined how well the network generates class-specific samples, as a way to measure top-down performance. We plot the MCMC sampling process in Figure 4. We find:

- The original DBM, generates a variety of digits-like samples, but the samples are not well restricted by the class labels, suggesting that the top-down influence is not enough to constrain the samples of the fist layer
- The importance weighted iDBM, compared with the original DBM, generates much more reasonable samples, confined by the class labels. This suggest that the importance weight is an efficient way to implement a sufficient amount of top-down influence.
- The FNN-boosted bDBM also generates reasonable samples, but seems slightly worth than the non-boosted DBM, this might be because the boosted hidden nodes are trained in the bottom-up FNN, therefore they tend to code features optimized for the bottom-up task, but no top-down task.

4.4 visualization of the learned pairwise connection weights between the lowest two layers

The W_{h_1,h_2} of non-boosted DBM and boosted DBM are shown in Figure 5. We find that the non-boosted model learned a more homogeneous filters, whereas the boosted model learned a more diverse set of weights that focused on local contrast. This might explain some of the difference in the performance

5 Conclusion

In this project, we implemented a deep Boltzmann machine that incorporates bottom-up classification (recognition) and top-down sampling (imagination) of visual system. We used two novel techniques to improve the performance: 1) the important weightening effectively balanced the layers and help to implement a sufficient amount of top-down influence; 2) FNN-boosting accelerates the training process, making if possible to be applied to very deep networks.

We noticed that both the classification and sampling was not perfect, this may be due to the stochastic gradient descent stage. If change my own implementation with other established packages, the performance could be much improved.

I did not use the model to implement pattern completion and denoising in the project, which are two other inference scenarios as is suggested in section 3.2

References

[1] Krizhevsky, A., Sutskever, I. & and Hinton, G. E. (2012) ImageNet Classification with Deep Convolutional Neural Networks *Advances in Neural Information Processing Systems, Lake Tahoe, Nevada*:

[2] L. A. Gatys, A. S. Ecker, & M. Bethge (2015) Texture Synthesis Using Convolutional Neural Networks *Advances in Neural Information Processing Systems 28*

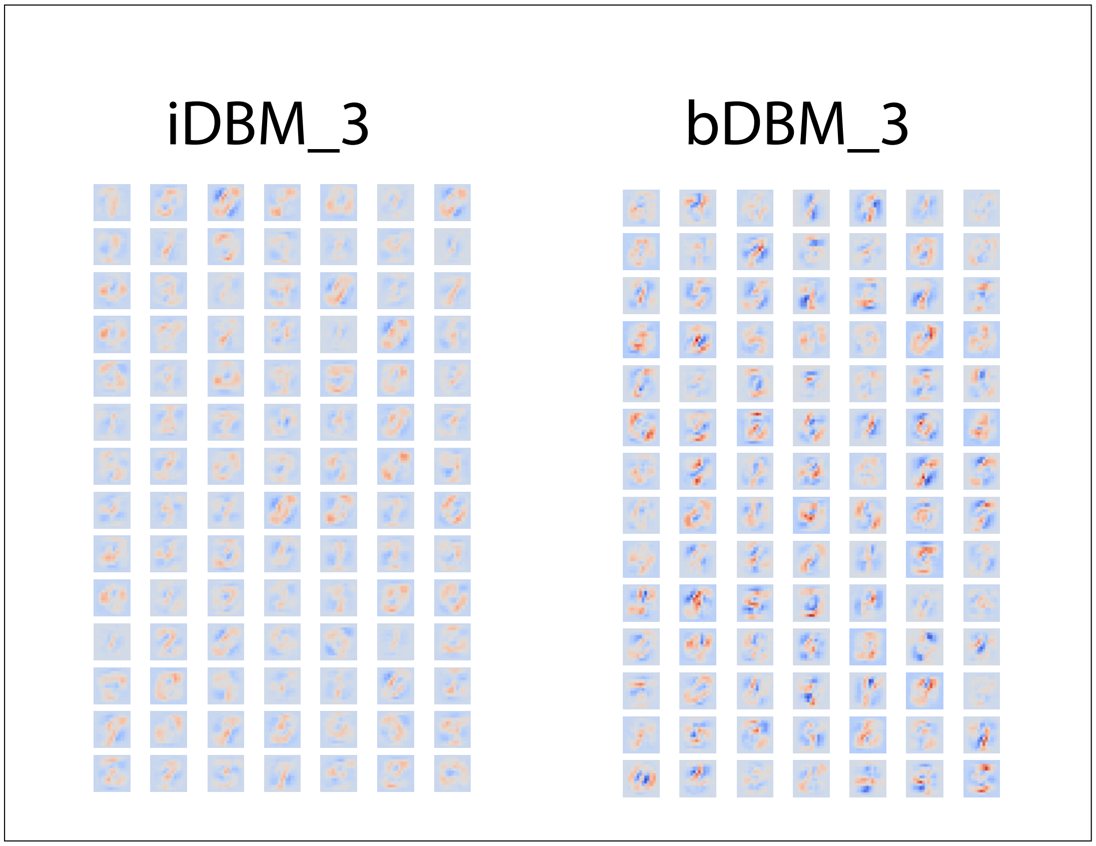

Figure 5: **Learned pairwise connection parameter** W_{h_1,h_2} We only show the comparison of the non-boosted model with the boosted model, because the important weights does not make any visible difference here

[3] Hinton, G. E., Osindero, S. & Teh, Y. (2006) A fast learning algorithm for deep belief nets. *Neural Computation*, 18, pp 1527-1554

[4] Srivastava, N., Salakhutdinov, R. R. and Hinton, G. E. (2013) Modeling Documents with a Deep Boltzmann Machine *arXiv preprint arXiv:1309.6865*

[5] Hinton, G. E. (2002), Training Products of Experts by Minimizing Contrastive Divergence, *Neural Computation* 14.8 pp 1771-1800