Unsupervised Deep Learning: A Short Review

Juha Karhunen[1], Tapani Raiko[1], and KyungHyun Cho[2]

[1] Dept. of Information and Computer Science, Aalto University, Espoo, Finland.
[2] University of Montreal, Montreal, Canada
Email: juha.karhunen@aalto.fi, tapani.raiko@aalto.fi, kyunghyun.cho@umontreal.ca

Abstract. Deep neural networks with several layers have during the last years become a highly successful and popular research topic in machine learning due to their excellent performance in many benchmark problems and applications. A key idea in deep learning is to not only learn the nonlinear mapping between the inputs and outputs, but also the underlying structure of the data (input) vectors. In this chapter, we first consider problems with training deep networks using backpropagation type algorithms. After this, we consider various structures used in deep learning, including restricted Boltzmann machines, deep belief networks, deep Boltzmann machines, and nonlinear autoencoders. In the later part of this chapter we discuss in more detail the recently developed neural autoregressive distribution estimator (NADE) and its variants.

Keywords: Deep learning, neural networks, unsupervised learning, restricted Boltzmann machines, deep belief networks, deep Boltzmann machines, autoencoders, neural autoregressive distribution estimators.

1 Introduction

In late 1980's, neural networks became a hot topic in machine learning due to invention of several efficient learning methods and network structures. These new methods included multilayer perceptron networks trained by backpropagation type algorithms, self-organizing maps, and radial basis function networks [1, 2]. While neural networks are successfully used in many applications, interest in their research decreased later on. The emphasis in machine learning research moved to other areas, such as kernel methods and Bayesian graphical models.

Deep learning was introduced by Hinton and Salakhutdinov in 2006 [3]. Deep learning has then become a hot topic in machine learning, leading to a renaissance of neural networks research. This is because when trained properly, deep networks have achieved world-record results in many classification and regression problems.

Deep learning is quite an advanced topic. In this short review, we do not discuss most of their learning algorithms in detail. However, the NADE-k method introduced recently by two of the authors in [4] is discussed in more detail. There exist different types of reviews on deep learning containing more information. An older review is [5], and the doctoral theses [6, 7] are good introductions to deep learning. Schmidhuber lists in his recent review [8] over 700 references on deep learning, but the review itself is very short with no formulas. The book [9] in preparation will probably become a quite popular reference on deep learning, but it is still a draft, with some chapters lacking. In general, research on deep learning is advancing very rapidly, with new ideas and methods introduced all the time.

In the following, we first briefly discuss multilayer perceptron networks and restricted Boltzmann machines as starting points to deep learning, and move then to various deep networks.

2 Multilayer Perceptron Networks

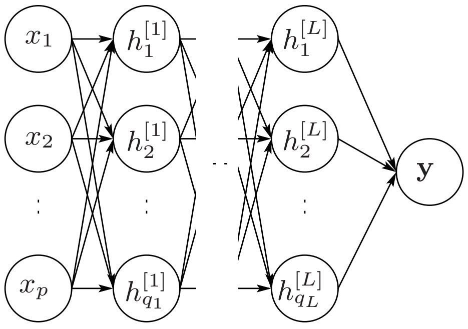

Fig. 1. The architecture of a multilayer perceptron network with L hidden layers.

Figure 1 shows a multilayer perceptron (MLP) network having an input layer, $L \geq 1$ hidden layers, and the output layer. In general, the numbers of neurons (nodes) in each layer can vary. Usually the processing in the hidden layers is nonlinear, while the output layer can be linear or nonlinear. In the input layer no computations take place, only the components of the input vector are inputted there, one component in each neuron.

The operation of neuron k in the l-th hidden layer is described by the equation

$$h_k^{[l]} = \phi \left(\sum_{j=1}^{m^{[l-1]}} w_{kj}^{[l]} h_j^{[l-1]} + b_k^{[l]} \right), \tag{1}$$

where $h_j^{[l-1]}$, $j = 1, \ldots, m^{[l-1]}$ are the $m^{[l-1]}$ input signals coming to the neuron k, and $w_{kj}^{[l]}$, $j = 1, \ldots, m^{[l-1]}$ are the respective weights multiplying the input signals. The number of neurons in the l:th layer is $m^{[l]}$. The input signals to the MLP network and to its first hidden layer are $x_1, \ldots, x_p$. The constant bias term b_k is added to the weighted sum. The components of the output vector $\mathbf{y}$ are computed similarly as the outputs of the l:th hidden layer in (1). The function $\phi(t)$ is the nonlinearity applied to the weighted sum. It is typically chosen to be the hyperbolic tangent $\phi(t) = \tanh(at)$ where a is constant or the logistic sigmoidal function $\phi(t) = 1/(1 + e^{-at})$. If the operation of a neuron is linear, $\phi(t) = at$ [1, 2].

Even though processing in a single neuron is simple, it is nonlinear. These distributed nonlinearities in each neuron of the hidden layers and possibly also in the output layer of the MLP network give to it a high representation power, but on the other hand make its exact mathematical analysis impossible and cause other problems such as local minima in the cost function. However, a MLP network having enough neurons in a single hidden layer can approximate any smooth enough nonlinear input-output mapping [1, 2].

With detailed notations the learning algorithms of MLP networks become quite complicated. We do not here go into details but just present an overall view, for details see the books [1, 2]. In general, MLP networks are trained in supervised manner using N known training pairs $\{\mathbf{x}_i, \mathbf{d}_i\}$ where $\mathbf{x}_i$ is i:th input vector and $\mathbf{d}_i$ is the corresponding desired response (output). The vector $\mathbf{x}_i$ is inputted to the MLP network and the corresponding output $\mathbf{y}_i$ is vector computed. The criterion used to learn the weights of the MLP network is typically the mean-square error $E = \mathrm{E}\{\| \mathbf{d}_i - \mathbf{y}_i \|^2\}$, which is minimized.

The steepest descent learning rule for a weight w_{ji} in any layer is given by

$$\Delta w_{ji} = -\mu \frac{\partial E}{\partial w_{ji}} \tag{2}$$

In practice, steepest descent is replaced by instantaneous gradient or a mini batch over 100-1000 training pairs. The required gradients are computed first for the neurons in the output layer using their local errors. These local errors are then propagated backwards to the previous layer, and the weights of its neurons can be updated, and so on. The name backpropagation of the basic learning algorithm for MLP networks comes from this. Usually numerous iterations and sweeps over the training data are required for convergence, especially if instantaneous stochastic gradient is used. Many variants of backpropagation learning and faster converging alternatives have been introduced [1, 2].

Usually MLP networks are designed to have only one or two hidden layers, because training more hidden layers using backpropagation type algorithms using steepest descent directions has turned out unsuccessful. The additional hidden layers do not learn useful features easily because the gradients with respect to them decay exponentially [10, 11]. A learning algorithm using only the steepest descent update directions often leads to poor local optima or saddle points [12], potentially due to its inability to break symmetries among multiple neurons in each hidden layer [13].

3 Deep learning

However, it would be desirable to have deep neural networks having several hidden layers. The idea is that the layer closest to the data vectors learns simple

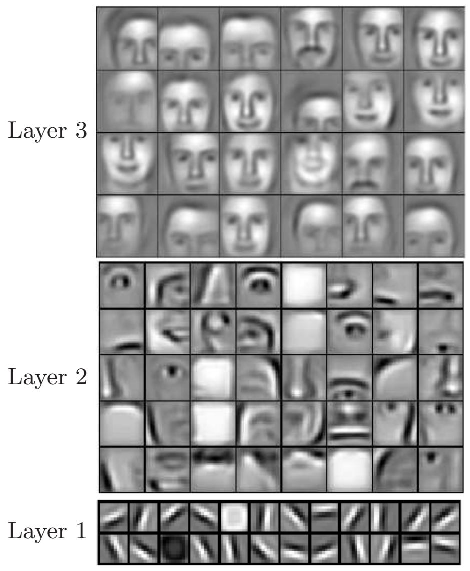

Fig. 2. Higher layers extract more general features of face images. The input to the network consists of pixels. Reprinted from [14].

features, while the higher layers should learn higher-level features. For example in digital images first low-level features such as edges and lines in different directions are learned in the first hidden layer. They are followed by shapes, objects, etc. in higher level layers. An example is shown in Figure 2. Human brains, especially cortex, contain deep biological neural networks working in this way. They are very efficient in tasks that are difficult for computers such as various applications of pattern recognition.

Deep learning addresses problems encountered when applying backpropagation type algorithms to deep networks with many layers. A key idea is to not only learn the nonlinear mapping between input and output vectors, but also the underlying structure of data (input) vectors. To achieve this goal, unsupervised pre-training is used. This is achieved in practice by using restricted Boltzmann machines or autoencoders in each hidden layer as building blocks in forming deep neural networks.

4 Restricted Boltzmann Machines

Boltzmann machines are a class of neural networks introduced already in late 1980's. They are based on statistical physics, and they use stochastic neurons contrary to most other neural network methods. Restricted Boltzmann Machines (RBMs) are simplified versions of Boltzmann machines; see Figure 3. In RBMs, the connections between the hidden neurons (top) and between the visible neu-

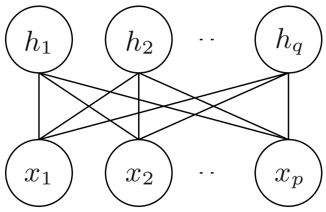

Fig. 3. Restricted Boltzmann machine.

rons (bottom) in the original Boltzmann machines are removed. Only the connections between the neurons in the visible layer and the hidden layer remain. Their weights are collected to the matrix $\mathbf{W}$. This simplification makes learning in RBMs tractable compared with Boltzmann machines, where it becomes soon intractable due to the many connections except for small-scale toy problems.

4.1 Modeling binary data

In an RBM, the top layer represents a vector of stochastic binary features $\mathbf{h}$. That is, the value of the state of each neuron can be either $+1$ or -1 with a certain probability. The bottom layer contains stochastic binary "visible" variables $\mathbf{x}$. Their joint Boltzmann distribution is [6]

$$p(\mathbf{x}, \mathbf{h}) = \frac{1}{Z} \exp(-E(\mathbf{x}, \mathbf{h}))$$

where $E(\mathbf{x}, \mathbf{h})$ is an energy term given by

$$E(\mathbf{x}, \mathbf{h}) = -\sum_i b_i x_i - \sum_j b_j h_j - \sum_{i,j} x_i h_j W_{ij},$$

and the normalization constant is

$$Z = \sum_x \sum_h \exp(-E(\mathbf{x}, \mathbf{h}))$$

From these equations, one can derive the conditional Bernoulli distributions

$$p(h_j = 1 \mid \mathbf{x}) = \sigma\left(b_j + \sum_i W_{ij} x_i\right)$$

$$p(x_i = 1 \mid \mathbf{h}) = \sigma\left(b_i + \sum_i W_{ij} h_j\right)$$

There $\sigma(z)$ is the logistic sigmoidal function

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

W_{ij} is a symmetric interaction term between input i and feature j, and b_i, b_j are bias terms. The marginal distribution over visible vector $\mathbf{x}$ is

$$p(\mathbf{x}) = \sum_{\mathbf{h}} \frac{\exp(-E(\mathbf{x}, \mathbf{h}))}{\sum_{\mathbf{u},\mathbf{g}} \exp(-E(\mathbf{u}, \mathbf{g}))} \tag{3}$$

The parameter update required to perform gradient ascent in the log-likelihood becomes ($\langle \cdot \rangle$ denotes expectation)

$$\varDelta W_{ij} = \epsilon(\langle x_i h_j \rangle_{data} - \langle x_i h_j \rangle_{model}) \tag{4}$$

In the data distribution $\mathbf{x}$ is taken from the data set and $\mathbf{h}$ from the conditional distribution $p(\mathbf{h} \mid \mathbf{x}, \boldsymbol{\theta})$ given by the model. In the model distribution both are taken from the joint distribution $p(\mathbf{x}, \mathbf{h})$ of the model. For the bias terms, one gets a similar but simpler equation. The expectations can be estimated using Gibbs sampling in which samples are generated from the respective probability distributions.

4.2 Modeling real-valued data

Restricted Bolzmann machines can be generalized to exponential family distributions [6, 15]. For example, digital images with real-valued pixels can be modeled by visible units that have a Gaussian distribution. The mean of this distribution is determined by the hidden units:

$$p(x_i \mid \mathbf{h}) = \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left(-\frac{(x - b_i - \sum_j h_j w_{ij})^2}{2\sigma_i^2} \right)$$

$$p(h_j = 1 \mid \mathbf{x}) = \sigma\left(b_j + \sum_i W_{ij} \frac{x_i}{\sigma_i^2} \right)$$

The marginal distribution over visible units $\mathbf{x}$ is given by Eq. (3) with an energy term

$$E(\mathbf{x}, \mathbf{h}) = \sum_i \frac{(x_i - b_i)^2}{2\sigma_i^2} - \sum_j b_j h_j - \sum_{i,j} h_j w_{ij} \frac{x_i}{\sigma_i^2}$$

If the variances are set to $\sigma_i^2 = 1$ for all visible units i, the parameter updates are the same as defined in Eq. (4).

5 Deep Belief Networks

Deep belief networks are generative models with many layers of hidden causal variables. Each layer of a deep belief network consists of a restricted Boltzmann machine. Hinton et al. derived a way to perform fast, greedy learning of deep belief networks one layer at a time [3]. When an RBM has learned, its feature activations are used as the "data" for training the next RBM in the deep belief networks, see Figure 4.

An important aspect of this layer-wise learning procedure is that each extra layer increases a lower bound on the log probability of the data, provided that the number of features per layer does not decrease. This layer-by-layer training can be repeated several times for learning a deep, hierarchical model of the data. Each layer of features captures strong high-order correlations between the activities of the features in the layer below.

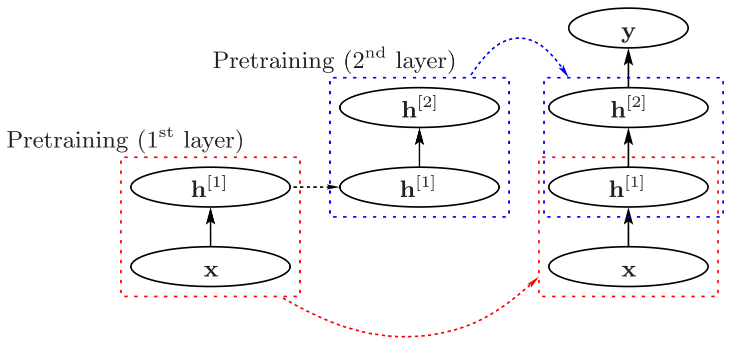

Fig. 4. Pretraining of a deep belief network.

This representation is more efficient than using a single hidden layer with many units. Using the greedy algorithm, one can learn a relatively good hierarchical representation of the data. But it is not yet an optimal representation, because the weights of each layer are learned independently of the weights of the next layers. Therefore, the representation found by the greedy learning algorithm can be improved using a fine tuning algorithm for the weights. To this end, one can use a variant of the standard backpropagation algorithm which is good at fine tuning.

Recursive learning of a deep generative model in this manner can be summarized as follows:

1. Learn the parameter vector W^1 of the first layer of a Bernoulli or Gaussian model.
2. Freeze the parameters of the lower level model. Use as the data for training the next layer of binary features the activation probabilities of the binary features, when they are driven by the training data.
3. Freeze the parameters W^2 that define the second layer of features, and use the activation probabilities of those features as data for training the third layer of features.
4. Proceed recursively for as many layers as desired.

6 Deep Boltzmann Machines

In the Deep Belief Networks (DBN) discussed above, the connections between the layers are directed except for the two upmost layers. However, for better flow of information, it would be desirable to have undirected connections everywhere. Salakhutdinov and Hinton introduced such Deep Boltzmann Machines (DBM) in 2009 [16, 6]. In Fig. 5, the vectors of the activities of the neurons in the visible and hidden layers are denoted respectively by $\mathbf{v}, \mathbf{h}^1, \ldots, \mathbf{h}^L$. The state vector of the DBM network having L hidden layers is denoted by $\mathbf{x} = [\mathbf{v}; \mathbf{h}^1; \ldots, \mathbf{h}^L] = [\mathbf{v}, \mathbf{h}]$. Note here the difference in notation: in RBMs $\mathbf{x}$ denoted the activities of visible units.

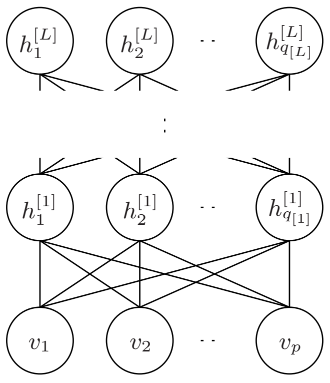

Fig. 5. Deep Boltzmann machine has undirected connections.

The Boltzmann distribution

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

is still used, where $\boldsymbol{\theta}$ denotes all the parameters. The energy function $-E(\mathbf{x} \mid \boldsymbol{\theta})$ is pretty complicated, consisting of products of activities multiplied by the respective weights. They are summed up together with activities multiplied by biases. Each parameter θ is updated according to the rule

$$\Delta\theta = \epsilon(\langle T_1 \rangle_{data} - \langle T_2 \rangle_{model})$$

The expectation $\langle T_1 \rangle_{data}$ of the term

$$T_1 = -\frac{\partial E(\mathbf{v}^{(n)}, \mathbf{h} \mid \boldsymbol{\theta})}{\partial \theta}$$

is computed over the data distribution $P(\mathbf{h} \mid \{\mathbf{v}^{(n)}\}, \boldsymbol{\theta})$. Here $\mathbf{v}^{(n)}$ denotes the n:th visible data vector. The expectation $\langle T_2 \rangle_{model}$ of the term

$$T_2 = -\frac{\partial E(\mathbf{v}, \mathbf{h} \mid \boldsymbol{\theta})}{\partial \theta}$$

is computed over the model distribution $P(\mathbf{v}, \mathbf{h} \mid \boldsymbol{\theta})$. The expectation over the data distribution can be estimated using a variational approximation, while the expectation over the model distribution can be computed using Gibbs sampling [16, 6].

6.1 Differences between Deep Belief Networks and Deep Boltzmann Machines

Thus far we have discussed only learning, but not using these networks for inference and generation of new samples. Considering first Deep Belief Network, it actually provides two networks that have common weights. In the recognition network, data vector is inputted to the visible layer, and information then proceeds upwards. One feedforward pass is required for inference. New data vectors resembling training vectors can be generated by sampling the uppermost hidden layer with MCMC (Markov Chain Monte Carlo) methods. Then the information proceeds downwards to the visible layer, and every layer tries to represent all the dependencies in the layer below it.

On the contrary, Deep Boltzmann machines have only one undirected network. Both recognition and generation of new samples require inference over the entire network. For recognition mean field methods are typically used, and MCMC for generation. The upper layers represent only such information that lower layers have not managed to represent.

7 Nonlinear Autoencoders

Deep belief networks can be used for training nonlinear autoencoders [7]. Autoencoder is a neural network (or mapping method) where the desired output is the input (data) vector itself. This is meaningful because in the middle of autoencoder, there is a data compressing bottleneck layer having fewer neurons than in the input and output layers. Therefore, the output vector of an autoencoder network is usually an approximation of the input vector only. Comparing it with the input vector provides the error vector needed in training the autoencoder network. Figure 6 shows a simple example of an autoencoder.

Autoencoders were first studied in 1990's for nonlinear data compression [17, 18] as a nonlinear extension of standard linear principal component analysis (PCA). Traditional autoencoders have five layers: a hidden layer between the input layer and the data compressing middle bottleneck layer, as well as a similar hidden layer with many neurons between the middle bottleneck layer and output layer [2]. Output vector of the middle bottleneck layer in autoencoders can be used for nonlinear data compression. They were trained using the backpropagation algorithm by minimizing the mean-square error, but this is difficult for multiple hidden layers with millions of parameters.

The greedy learning algorithm for restricted Boltzmann machines can be used to pre-train autoencoders also for large problems. It performs a global search for a good, sensible region in the parameter space. The fine-tuning of

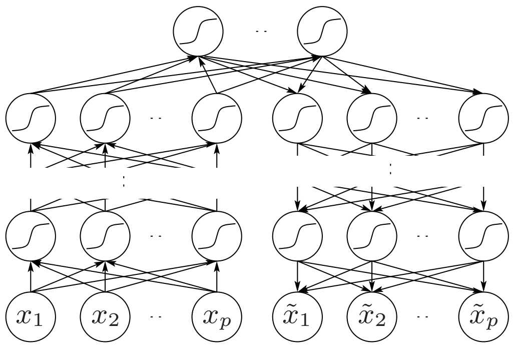

Fig. 6. An MLP network acting as an autoencoder.

model parameters is carried out using a variant of standard backpropagation. Generally speaking backpropagation is better at local fine-tuning of the model parameters than global search. So further training of the entire autoencoder using backpropagation will result in a good local optimum. Nonlinear autoencoders trained in this way perform considerably better than linear data compression methods such as PCA.

Autoencoders must be regularized for preventing them to learn identity mapping. Instead of a middle bottleneck layer, one can add noise to input vectors or put some of their components zero [19]. Or one can impose sparsity by penalizing hidden unit activations near zero. Still another possibility is to force the encoder to have small derivatives with respect to the inputs $\mathbf{x}$ (contractive constraint) [20, 21]. Discrete inputs can be handled by using a cross-entropy or log-likelihood reconstruction criterion.

Instead of stacking Restricted Boltzmann Machines, one can use a stack of shallow autoencoders to train deep belief networks, deep Boltzmann machines or deep autoencoders [22].

8 Neural Autoregressive Density Estimator (NADE)

8.1 Background

Traditional building blocks for deep learning have, however, some unsatisfactory properties. For example Boltzmann machines are difficult to train due to the intractability of computing the statistics of the model distribution. This may lead to potentially high-variance Monte Carlo Markov Chain (MCMC) estimators during training [23] and the computationally intractable objective function. Autoencoders have a simpler objective function such as denoising reconstruction error [19], which can be used for model selection but not for the important choice of the corruption function.

In [24], Larochelle and Murray introduced the so-called Neural Autoregressive Distribution Estimator (NADE), which specializes previous neural autoregressive density estimators [25] and was recently extended [26] to deeper architectures. It is appealing because both the training criterion (just log-likelihood) and its gradient can be computed tractably and used for model selection, and the model can be trained by stochastic gradient descent with backpropagation. However, it has been observed that the performance of NADE has still room for improvement.

Training of the neural autoregressive density estimator (NADE) can be viewed as performing one step of probabilistic inference on missing values in data for reconstructing missing values in data. The idea of using missing value imputation as a training criterion has appeared in three recent papers. This approach can be seen either as training an energy-based model to impute missing values well [27], as training a generative probabilistic model to maximize a generalized pseudo-log-likelihood [28], or as training a denoising autoencoder with a masking corruption function [26].

The NADE model involves an ordering over the components of the data vector. The core of the model is the reconstruction of the next component given all the previous ones. In the following, we describe a new model and method called NADE-k, proposed recently by two of the authors in [4]. It is an extension of the NADE model based on the reinterpretation of the reconstruction procedure as a single iteration in a variational inference algorithm.

8.2 Iterative NADE-k method

The NADE-k method extends the inference scheme of the original NADE method in [24] to multiple steps [4]. We argue that it is easier to learn to improve a reconstruction iteratively in k steps rather than to learn to reconstruct in a single inference step. The proposed model is an unsupervised building block for deep learning that combines the desirable properties of NADE and multi-prediction training: (1) Its test likelihood can be computed analytically, (2) it is easy to generate independent samples from it, and (3) it uses an inference engine that is a superset of variational inference for Boltzmann machines.

The NADE-k method is introduced and discussed in more detail in [4], with experiments on two datasets, MNIST handwritten numbers and Caltech-101 silhouettes. In these experiments, the NADE-k method outperformed the original NADE [24] as well as NADE trained with the order-agnostic training algorithm [26].

In the probabilistic NADE-k method D-dimensional binary data vectors $\mathbf{x}$ are considered. We start by defining conditional distribution $p_{\boldsymbol{\theta}}$ for imputing missing values using a fully factorial conditional distribution:

$$p_{\boldsymbol{\theta}}(\mathbf{x}_{mis} \mid \mathbf{x}_{obs}) = \prod_{i \in mis} p_{\boldsymbol{\theta}}(x_i \mid \mathbf{x}_{obs}), \tag{5}$$

where the subscripts mis and obs denote missing and observed components of $\mathbf{x}$. From the conditional distribution $p_{\boldsymbol{\theta}}$ we compute the joint probability distri-

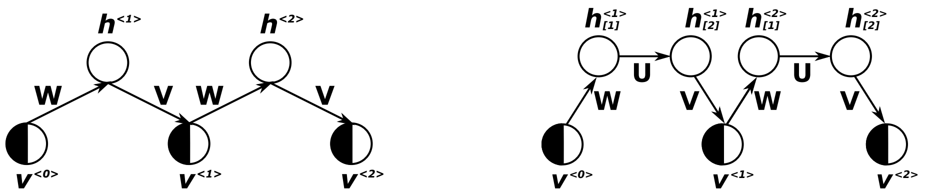

Fig. 7. The choice of a structure for NADE-k is very flexible. Left: Basic structure corresponding to Equations (10–11) with $n = 2$ and $k = 2$. Right: Depth added as in NADE by [26] with $n = 3$ and $k = 2$. These two structures are used in the experiments.

bution over $\mathbf{x}$ given an ordering o (a permutation of the integers from 1 to D) by

$$p_{\boldsymbol{\theta}}(\mathbf{x} \mid o) = \prod_{d=1}^{D} p_{\boldsymbol{\theta}}(x_{o_d} \mid \mathbf{x}_{o_{<d}}), \tag{6}$$

where $o_{<d}$ stands for indices $o_1 \dots o_{d-1}$.

The model is trained to minimize the negative log-likelihood averaged over all possible orderings o

$$\mathcal{L}(\boldsymbol{\theta}) = \mathrm{E}_{o \in D!}\left[\mathrm{E}_{\mathbf{x} \in \text{data}}\left[-\log p_{\boldsymbol{\theta}}(\mathbf{x} \mid o)\right]\right]. \tag{7}$$

using an unbiased, stochastic estimator of $\mathcal{L}(\boldsymbol{\theta})$

$$\hat{\mathcal{L}}(\boldsymbol{\theta}) = -\frac{D}{D-d+1} \log p_{\boldsymbol{\theta}}(\mathbf{x}_{o_{\geq d}} \mid \mathbf{x}_{o_{<d}}) \tag{8}$$

by drawing o uniformly from all $D!$ possible orderings and d uniformly from $1 \dots D$ [26]. Note that while the model definition in Eq. (6) is sequential in nature, the training criterion (8) involves reconstruction of all the missing values in parallel. In this way, training does not involve picking or following specific orders of indices.

We define the conditional model $p_{\boldsymbol{\theta}}(\mathbf{x}_{mis} \mid \mathbf{x}_{obs})$ using a deep feedforward neural network with nk layers, where we use n weight matrices k times. This can also be interpreted as running k successive inference steps with an n-layer neural network.

The input to the network is

$$\mathbf{v}^{\langle 0 \rangle} = \mathbf{m} \odot \mathrm{E}_{\mathbf{x} \in \text{data}}\left[\mathbf{x}\right] + (\mathbf{1} - \mathbf{m}) \odot \mathbf{x} \tag{9}$$

where $\mathbf{m}$ is a binary mask vector indicating missing components with 1, and $\odot$ is an element-wise multiplication. $\mathrm{E}_{\mathbf{x} \in \text{data}}\left[\mathbf{x}\right]$ is an empirical mean of the observations. For simplicity, we give equations for a simple structure with $n = 2$, illustrated in Fig. 7 (left).

In this case, the activations of the layers at the t-th step are

$$\mathbf{h}^{\langle t \rangle} = \phi(\mathbf{W}\mathbf{v}^{\langle t-1 \rangle} + \mathbf{c}) \tag{10}$$

$$\mathbf{v}^{\langle t \rangle} = \mathbf{m} \odot \sigma(\mathbf{V}\mathbf{h}^{\langle t \rangle} + \mathbf{b}) + (\mathbf{1} - \mathbf{m}) \odot \mathbf{x} \tag{11}$$

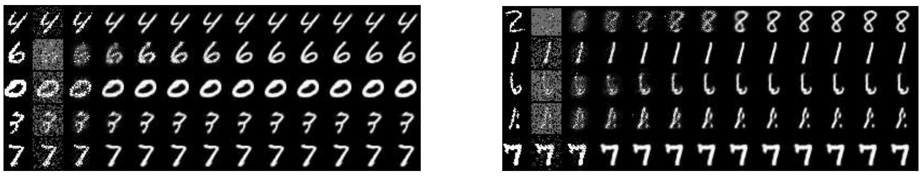

Fig. 8. The inner working mechanism of NADE-k. The left most column shows the data vectors $\mathbf{x}$, the second column shows their masked version and the subsequent columns show the reconstructions $\mathbf{v}^{\langle 0\rangle} \ldots \mathbf{v}^{\langle 10\rangle}$ (See Eq. (11)).

where ϕ is an element-wise nonlinearity, σ is a logistic sigmoid function, and the iteration index t runs from 1 to k. The conditional probabilities of the variables (see Eq. (5)) are read from the output $\mathbf{v}^{\langle k\rangle}$ as

$$p_{\boldsymbol{\theta}}(x_i = 1 \mid \mathbf{x}_{obs}) = v_i^{\langle k\rangle}. \tag{12}$$

Fig. 8 shows examples of how $\mathbf{v}^{\langle t\rangle}$ evolves over iterations, with the trained model.

The parameters $\boldsymbol{\theta} = \{\mathbf{W}, \mathbf{V}, \mathbf{c}, \mathbf{b}\}$ can be learned by stochastic gradient descent to minimize $-\mathcal{L}(\boldsymbol{\theta})$ in Eq. (7), or its stochastic approximation $-\hat{\mathcal{L}}(\boldsymbol{\theta})$ in Eq. (8), with the stochastic gradient computed by back-propagation.

Once the parameters $\boldsymbol{\theta}$ are learned, one can define a mixture model by using a uniform probability over a set of orderings O. The probability of a given vector $\mathbf{x}$ as a mixture model can be computed

$$p_{\mathrm{mixt}}(\mathbf{x} \mid \boldsymbol{\theta}, O) = \frac{1}{|O|} \sum_{o \in O} p_{\boldsymbol{\theta}}(\mathbf{x} \mid o) \tag{13}$$

with Eq. (6). One can draw independent samples from the mixture by first drawing an ordering o and then sequentially drawing each variable using $x_{o_d} \sim p_{\boldsymbol{\theta}}(x_{o_d} \mid \mathbf{x}_{o_{<d}})$. Furthermore, samples can be drawn from the conditional $p(\mathbf{x}_{mis} \mid \mathbf{x}_{obs})$ easily by considering only orderings where the observed indices appear before the missing ones.

It is well known that training deep networks is difficult without pretraining, and in the experiments described in more detail in [4], the networks are trained up to $kn = 7 \times 3 = 21$ layers. When pretraining, the model is trained to produce good reconstructions $\mathbf{v}^{\langle t\rangle}$ at each step $t = 1 \ldots k$. More formally, in the pretraining phase, the Equations (8) and (12) are replaced by

$$\hat{\mathcal{L}}_{\mathrm{pre}}(\boldsymbol{\theta}) = -\frac{D}{D-d+1} \frac{1}{k} \sum_{t=1}^{k} \log \prod_{i \in o_{\geq d}} p_{\boldsymbol{\theta}}^{\langle t\rangle}(x_i \mid \mathbf{x}_{o_{<d}}) \tag{14}$$

$$p_{\boldsymbol{\theta}}^{\langle t\rangle}(x_i = 1 \mid \mathbf{x}_{obs}) = v_i^{\langle t\rangle}. \tag{15}$$

8.3 Related Methods and Approaches

Order-agnostic NADE The proposed method follows closely the order-agnostic version of NADE [26], which may be considered as the special case of NADE-k with $k = 1$. On the other hand, NADE-k can be seen as a deep NADE with some specific weight sharing (matrices $\mathbf{W}$ and $\mathbf{V}$ are reused for different depths) and gating in the activations of some layers (See Equation (11)).

Additionally, in [26] it was found crucial to give the mask $\mathbf{m}$ as an auxiliary input to the network, and initialized missing values to zero instead of the empirical mean (See Eq. (9)). Due to these differences, the approach in [26] is called here NADE-mask. One should note that NADE-mask has more parameters due to using the mask as a separate input to the network, whereas NADE-k is roughly k times more expensive to compute.

Probabilistic Inference Consider the task of missing value imputation in a probabilistic latent variable model. The conditional probability of interest is obtained by marginalizing out the latent variables from the posterior distribution:

$$p(\mathbf{x}_{mis} \mid \mathbf{x}_{obs}) = \int_{\mathbf{h}} p(\mathbf{h}, \mathbf{x}_{mis} \mid \mathbf{x}_{obs}) \mathrm{d}\mathbf{h}.$$

Accessing the joint distribution $p(\mathbf{h}, \mathbf{x}_{mis} \mid \mathbf{x}_{obs})$ directly is often harder than alternatively updating $\mathbf{h}$ and $\mathbf{x}_{mis}$ based on the conditional distributions $p(\mathbf{h} \mid \mathbf{x}_{mis}, \mathbf{x}_{obs})$ and $p(\mathbf{x}_{mis} \mid \mathbf{h})$. Variational inference is one of the representative examples that exploit this.

In variational inference, a factorial distribution $q(\mathbf{h}, \mathbf{x}_{mis}) = q(\mathbf{h})q(\mathbf{x}_{mis})$ is iteratively fitted to $p(\mathbf{h}, \mathbf{x}_{mis} \mid \mathbf{x}_{obs})$ such that the KL-divergence between q and p

$$\mathrm{KL}[q(\mathbf{h}, \mathbf{x}_{mis}) || p(\mathbf{h}, \mathbf{x}_{mis} \mid \mathbf{x}_{obs})] = \\ - \int_{\mathbf{h}, \mathbf{x}_{mis}} q(\mathbf{h}, \mathbf{x}_{mis}) \log \left[\frac{p(\mathbf{h}, \mathbf{x}_{mis} \mid \mathbf{x}_{obs})}{q(\mathbf{h}, \mathbf{x}_{mis})} \right] \mathrm{d}\mathbf{h}\mathrm{d}\mathbf{x}_{mis}$$

is minimized. The algorithm alternates between updating $q(\mathbf{h})$ and $q(\mathbf{x}_{mis})$, while considering the other one fixed.

As an example, consider a restricted Boltzmann machine (RBM) defined by

$$p(\mathbf{v}, \mathbf{h}) \propto \exp(\mathbf{b}^\top \mathbf{v} + \mathbf{c}^\top \mathbf{h} + \mathbf{h}^\top \mathbf{W} \mathbf{v}).$$

One can fit an approximate posterior distribution parameterized as $q(v_i = 1) = \bar{v}_i$ and $q(h_j = 1) = \bar{h}_j$ to the true posterior distribution by iteratively computing

$$\begin{aligned} \bar{\mathbf{h}} &\leftarrow \sigma(\mathbf{W}\bar{\mathbf{v}} + \mathbf{c}) \\ \bar{\mathbf{v}} &\leftarrow \mathbf{m} \odot \sigma(\mathbf{W}^\top \mathbf{h} + \mathbf{b}) + (\mathbf{1} - \mathbf{m}) \odot \mathbf{v}. \end{aligned}$$

Notice the similarity to Eqs. (10)–(11): If one assumes $\phi = \sigma$ and $\mathbf{V} = \mathbf{W}^\top$, the inference in the NADE-k is equivalent to performing k iterations of variational inference on an RBM for the missing values [29].

Multi-Predictive Deep Boltzmann Machine Goodfellow et al. [28] and Brakel et al. [27] use backpropagation through variational inference steps to train a deep Boltzmann machine. This is very similar to NADE-k, except that they approach the problem from the view of maximizing the generalized pseudo-likelihood [30]. The deep Boltzmann machine also lacks the tractable probabilistic interpretation similar to NADE-k, see Eq. (6), that would allow to compute a probability or to generate independent samples without resorting to a Markov chain. NADE-k is also somewhat more flexible in the choice of model structures, as can be seen in Fig. 7. For instance, encoding and decoding weights do not have to be shared and any type of nonlinear activations, other than a logistic sigmoid function, can be used.

Product and Mixture of Experts One can ask what would happen if we would define an ensemble likelihood along the line of the training criterion in Eq. (7). That is,

$$-\log p_{\text{prod}}(\mathbf{x} \mid \boldsymbol{\theta}) \propto \mathrm{E}_{o \in D!}\left[-\log p(\mathbf{x} \mid \boldsymbol{\theta}, o)\right].$$

Maximizing this likelihood directly will correspond to training a product-of-experts model [31]. However, this requires evaluation of the intractable normalization constant during training as well as in the inference, making the model not tractable any more.

On the other hand, one can consider using the log-probability of a sample under the mixture-of-experts model as the training criterion

$$-\log p_{\text{mixt}}(\mathbf{x} \mid \boldsymbol{\theta}) = -\log \mathrm{E}_{o \in D!}\left[p(\mathbf{x} \mid \boldsymbol{\theta}, o)\right].$$

This criterion resembles clustering, where individual models may specialize in only a fraction of the data. In this case, however, the simple estimator such as in Eq. (8) would not be available.

8.4 Experimental Results

the NADE-k model has been studied with two datasets: binarized MNIST handwritten digits and Caltech 101 silhouettes. NADE-k was trained with one or two hidden layers (see Fig. 7 left and right) with a hyperbolic tangent as the activation function $\phi(\cdot)$. Stochastic gradient descent was used on the training set with a mini batch size fixed to 100. The AdaDelta method [32] was used to adaptively choose a learning rate for each parameter update on-the-fly. We used a validation set for early stopping and to select the hyperparameters. With the best model on the validation set, we report the log-probability computed on the test set.

In [4], the NADE-k method is tested extensively with the MNIST data and compared favorably with the NADE-mask method introduced in [26]. We mostly skip here these experiments, except for in Figure 8 we present how each iteration

$t = 1 \dots k$ improves the corrupted input $\mathbf{v}^{\langle t \rangle}$ from Eq. (9). We also investigated what happens with test time k being larger than the training $k = 5$. We can see that in all cases, the iteration – which is a fixed point update – seems to converge to a point that is in most cases close to the ground-truth sample. However, the experiments in [4] show that the generalization performance drops after $k = 5$ when training with $k = 5$. From Figure 8, we can see that the reconstruction continues to be sharper even after $k = 5$, which seems to be the underlying reason for this phenomenon.

We also evaluate the proposed NADE-k method on Caltech-101 Silhouettes [33], using the standard split of 4100 training samples, 2264 validation samples and 2307 test samples. We demonstrate the advantage of NADE-k compared with NADE-mask under the constraint that they have a matching number of parameters. In particular, we compare NADE-k with 1000 hidden units with NADE-mask with 670 hidden units. We also compare NADE-k with 4000 hidden units with NADE-mask with 2670 hidden units.

We optimized the hyper-parameter $k \in \{1, 2, \dots, 10\}$ in the case of NADE-k. In both NADE-k and NADE-mask, we experimented without regularizations, with weight decays, or with dropout. We did not use the pretraining scheme described in Eq. (14).

Table 1. Average log-probabilities of test samples of Caltech-101 Silhouettes. The results marked with $\star$ are from [34]. The terms in the parenthesis indicate the number of hidden units, the total number of parameters (M for million), and the L2 regularization coefficient. NADE-mask 670h achieves the best performance without any regularizations.

Model	Test LL	Model	Test LL
RBM* (2000h, 1.57M)	-108.98	RBM * (4000h, 3.14M)	**-107.78**
NADE-mask (670h, 1.58M)	-112.51	NADE-mask (2670h, 6.28M, L2=0.00106)	-110.95
NADE-2 (1000h, 1.57M, L2=0.0054)	-108.81	NADE-5 (4000h, 6.28M, L2=0.0068)	**-107.28**

As one can see from Table 1, NADE-k outperforms the NADE-mask regardless of the number of parameters. In addition, NADE-2 with 1000 hidden units matches the performance of an RBM with the same number of parameters. Furthermore, NADE-5 has outperformed the previous best result obtained with the RBMs in [34], achieving the state-of-art result on this dataset. We can see from the samples generated by the NADE-k shown in Figure 9 that the model has learned the data well.

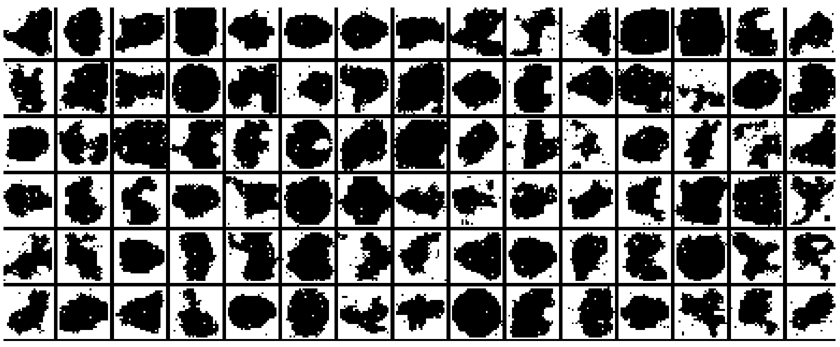
Fig. 9. Samples generated from NADE-k trained on Caltech-101 Silhouettes.

9 Conclusions

Deep learning has become a hot topic in machine learning, because it can provide world record results in different classification and regression problems and datasets. Many corporations including Google, Microsoft, Nokia etc. study it actively. Understanding deep learning well requires mathematical maturity and good knowledge of probabilistic modeling. Learning algorithms are complicated, and good initialization is important. The field is developing quite rapidly, with new structures and learning methods introduced all the time.

In this chapter, we have reviewed some of the most widely studied and used deep learning models for unsupervised learning tasks. Also, we have discussed in more detail a new model called iterative neural autoregressive distribution estimator NADE-k [4] that extends the conventional neural autoregressive distribution estimator (NADE) [26] and its training procedure. The proposed NADE-k method maintains the tractability of the original NADE while we showed that it outperforms the original NADE as well as similar, but intractable generative models such as restricted Boltzmann machines and deep belief networks.

The list of unsupervised models we have reviewed in this chapter is not exhaustive. During the last few years, a number of new deep learning models for unsupervised learning have been proposed. For instance, Kingma and Welling in [35] proposed a so-called variational autoencoder, where they proposed to train an autoencoder to maximize a variational lower-bound of a directed belief network. Bengio et al. [36] recently proposed a rather distinct deep learning-based framework for unsupervised learning, called generative stochastic network which aims to learn a Markov chain Monte Carlo transition operator instead of a full probability distribution.

References

1. S. Haykin, *Neural Networks and Learning Machines, 3rd ed.* Pearson, 2009.
2. K.-L. Du and M. Swamy, *Neural Networks and Statistical Learning*, Springer-Verlag, 2014.

3. G. Hinton, S. Osindero, and Y.-W. Teh, “A Fast Learning Algorithm for Deep Belief Nets”. Neural Computation, vol. 18, pp. 1527-1554, 2006.
4. T. Raiko, L. Yao, K. Cho, and Y. Bengio, “Iterative Neural Autoregressive Distribution Estimator (NADE-k)”. Accepted to Neural Information Processing Systems 2014 conference (NIPS 2014), Montreal, Canada, December 2014.
5. Y. Bengio, “Learning Deep Architectures for AI”, Foundations and Trends in Machine Learning, vol. 2, no. 1, 2009, pp. 1-127.
6. R. Salakhutdinov, “Learning Deep Generative Models”, Doctoral thesis, MIT, 2009. Available at http://www.mit.edu/∼rsalakhu/papers/Russ_thesis.pdf.
7. K. Cho, “Foundations and Advances in Deep Learning”. Doctoral thesis, Aalto University School of Science, Espoo, Finland, 2014.
8. J. Schmidhuber, “Deep Learning in Neural Networks: An Overview”, Technical Report IDSIA-03-14, Switzerland. arXiv:1404.7828 [cs.NE].
9. Y. Bengio, I. Goodfellow, and A. Courville, *Deep Learning.* An MIT Press book in preparation. Draft chapters available at http://www.iro.umontreal.ca/∼bengioy/dlbook/.
10. Y. Bengio, P. Simard, P. Frasconi, “Learning long-term dependencies with gradient descent is difficult”, IEEE Transactions on Neural Networks, vol. 5, no. 2, pp. 157–166, 1994.
11. X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks”. In Proc. of the Int. Conf. on Artificial Intelligence and Statistics (AISTATS 2010), pp. 249–256, 2010.
12. Y. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, Y. Bengio, “Identifying and attacking the saddle point problem in high-dimensional non-convex optimization”. In Advances in Neural Information Processing Systems (NIPS) 27, 2014.
13. 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), pp. 924–932, 2012.
14. H. Lee, R. Grosse, R. Ranganath and A. Ng, “Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations”. In Proc. of the 26th Int. Conf. on Machine Learning (ICML 2009), pp. 609–616, 2009.
15. K. Cho, A. Ilin, T.Raiko, “Improved Learning of Gaussian-Bernoulli Restricted Boltzmann Machines”. In Lecture Notes in Computer Science, Volume 6791, Artificial Neural Networks and Machine Learning (ICANN 2011), pp. 10–17, 2011.
16. R. Salakhutdinov and G. Hinton, “Deep Boltzmann Machines”, in *Proc. of 12th Int. Conf. on Artificial Intelligence and Statistics (AISTATS 2009)*, Clearwater Beach, Florida, USA, pp. 448–455, 2009.
17. M. Kramer, “Nonlinear Principal Component Analysis Using Autoassociative Neural Networks”. AIChE Journal, vol. 37, no. 2, pp. 233-243, 1991.
18. E. Oja, “Data Compression, Feature Extraction, and Autoassociation in Feedforward Neural Networks”. In Proc. of the Int. Conf. on Artificial Neural Networks (ICANN-91), Helsinki, Finland, June 1991, pp. 737–745, Elsevier.
19. P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, “Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion”. Journal of Machine Learning Research, vol. 11, pp. 3371–3408, 2010.
20. S. Rifai, P. Vincent, X. Muller, X. Glorot, Y. Bengio, “Contractive auto-encoders: Explicit invariance during feature extraction”. In Proc. of the 28th Int. Conf. on Machine Learning (ICML 2011), 2011.
21. H. Schulz, K. Cho, T. Raiko, S. Behnke, “Two-layer contractive encodings for learning stable nonlinear features” Neural Networks, 2014. (in press)

22. Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, "Greedy layer-wise training of deep networks". In Advances in Neural Information Processing Systems (NIPS), 2007.
23. Y. Bengio, G. Mesnil, Y. Dauphin, and S. Rifai, "Better Mixing via Deep Representations". In Proc. of the 30th Int. Conf. on Machine Learning (ICML 2013). arXiv preprint arXiv:1207.4404.
24. H. Larochelle and I. Murray, "The Neural Autoregressive Distribution Estimator". Journal of Machine Learning Research, vol. 15, pp. 29–37, 2011.
25. Y. Bengio and S. Bengio, "Modeling High-Dimensional Discrete Data with Multi-Layer Neural Networks". In Advances in Neural Information Processing Systems, pp. 400–406, 2000, MIT Press.
26. B. Uria, I. Murray, and H. Larochelle, "A Deep and Tractable Density Estimator". In Proc. of the 31st Int. Conf. on Machine Learning (ICML 2014), 2014. arXiv preprint arXiv:1310.1757.
27. P. Brakel, D. Stroobandt, and B. Schrauwen, "Training Energy-Based Models for Time-Series Imputation". Journal of Machine Learning Research, vol. 14, pp. 2771–2797, 2013.
28. I. Goodfellow, M. Mirza, A. Courville, and Y. Bengio, "Multi-Prediction Deep Boltzmann Machines". Advances in Neural Information Processing Systems, pp. 548–556, 2013.
29. C. Peterson and J. Anderson, "A Mean Field Theory Learning Algorithm for Neural Networks". Complex Systems, vol. 1, no. 5, pp. 995–1019, 1987.
30. F. Huang and Y. Ogata, "Generalized Pseudo-Likelihood Estimates for Markov Random Fields on Lattice". Annals of the Institute of Statistical Mathematics, vol. 54, no. 1, pp. 1–18, 2002.
31. G. Hinton, "Training Products of Experts by Minimizing Contrastive Divergence". Technical Report GCNU TR 2000-004, Gatsby Unit, University College London, 2000.
32. M. Zeiler, "ADADELTA: An Adaptive Learning Rate Method". Technical report, arXiv 1212.5701, 2012.
33. B. 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), pp. 509–516, 2010.
34. K. Cho, T. Raiko, and A. Ilin, " Enhanced Gradient for Rraining Restricted Boltzmann Machines". Neural Computation, vol. 25, no. 3, pp. 805–831, 2013.
35. D. P. Kingma, M. Welling, "Auto-Encoding Variational Bayes". In Proc. of the Int. Conf. on Learning Representations (ICLR 2014), 2014.
36. Y. Bengio, E. Thibodeau-Laufer, G. Alain, J. Yosinski, "Deep Generative Stochastic Networks Trainable by Backprop". In Proc. of the 31st Int. Conf. on Machine Learning (ICML 2014), 2014.