Pattern Recognition Letters 102 (2018) 1–7

Contents lists available at ScienceDirect

Pattern Recognition Letters

journal homepage: www.elsevier.com/locate/patrec

Population-Contrastive-Divergence: Does consistency help with RBM training?

Oswin Krause[a,*], Asja Fischer[b], Christian Igel[a]

[a] Department of Computer Science, University of Copenhagen, Denmark
[b] Department of Computer Science, University of Bonn, Germany

ARTICLE INFO

Article history:
Received 28 June 2017
Available online 2 December 2017

Keywords:
Restricted Boltzmann machine
Markov chain Monte Carlo
Contrastive divergence
Population Monte Carlo

ABSTRACT

Estimating the log-likelihood gradient with respect to the parameters of a Restricted Boltzmann Machine (RBM) typically requires sampling using Markov Chain Monte Carlo (MCMC) techniques. To save computation time, the Markov chains are only run for a small number of steps, which leads to a biased estimate. This bias can cause RBM training algorithms such as Contrastive Divergence (CD) learning to deteriorate. We adopt the idea behind Population Monte Carlo (PMC) methods to devise a new RBM training algorithm termed Population-Contrastive-Divergence (pop-CD). Compared to CD, it leads to a consistent estimate and may have a significantly lower bias. Its computational overhead is negligible compared to CD, but the variance of the gradient estimate increases. We experimentally show that pop-CD can significantly outperform CD. In many cases, we observed a smaller bias and achieved higher log-likelihood values. However, when the RBM distribution has many hidden neurons, the consistent estimate of pop-CD may still have a considerable bias and the variance of the gradient estimate requires a smaller learning rate. Thus, despite its superior theoretical properties, it is not advisable to use pop-CD in its current form on large problems.

1. Introduction

Estimating the log-likelihood gradient with respect to the parameters of an undirected graphical model, such as a Restricted Boltzmann Machine (RBM, [6,8,15]), is a challenging task. As the analytic calculation of the gradient is infeasible, it is often approximated using Markov Chain Monte Carlo (MCMC) techniques. Getting unbiased samples from the model distribution requires to run the Markov chain until convergence, but in practice the chain is only iterated for a fixed amount of steps and the quality of the samples is often unknown.

Arguably, k-step Contrastive Divergence (CD-k, [8]) is the algorithm most often used for RBM training. In each training iteration of CD-k a new Markov chain is initialized with a sample from the dataset, and k steps of block-Gibbs-sampling are performed. In practice, k is often set to one, which obviously results in a considerable bias of the gradient approximation. Even though the bias is bounded [5], it was reported to cause divergence in the long run of the optimization [4], which is difficult to detect by heuristics [14]. Moreover, it has been shown that the gradient field of the CD approximation does not belong to any objective function [16].

After the introduction of CD, other sampling schemes were proposed for the gradient approximation, the most prominent ones being Persistent Contrastive Divergence (PCD, [17]) and Parallel Tempering (PT, [3,7,13]). The former uses a persistent Monte Carlo Chain during training. While initially started from a training example as in CD, the chain is not discarded after each gradient update but successively reused in following learning iterations. This is done with the hope that the chain stays close to the model distribution, while performing PCD-based gradient ascent. However, this requires a very small learning rate to guarantee that the model distribution changes slowly enough to compensate the small mixing rate of Gibbs-sampling. To solve this problem, PT was suggested for RBM training [3,13]. This sampling method runs multiple tempered replica chains. A Metropolis-based swapping operator allows samples to swap between the chains in order to achieve faster mixing. This is paid for by a higher computational cost. While CD is inherently a parallel algorithm that can be applied simultaneously to the full data set, PT performs a step of all parallel Markov chains for every sample. This makes PT a serial algorithm that is difficult to transfer to GPUs, whereas implementing CD on a GPU is straightforward.

This paper introduces a different direction for RBM training that is based on the Population Monte Carlo (PMC) method [2]. In PMC sampling, a set of samples is re-weighted using importance sam-

* Corresponding author.
E-mail address: oswin.krause@di.ku.dk (O. Krause).

https://doi.org/10.1016/j.patrec.2017.11.022
0167-8655/

pling after each application of the transition operator so that the weighted samples are unbiased. Furthermore, the weights can be used to re-sample the points. We will present a new algorithm combining importance sampling as in PMC and CD learning. It can be implemented as efficiently as CD without suffering from a high bias in RBMs with a small number of hidden neurons.

2. Background

RBMs and Contrastive Divergence. An RBM is an undirected graphical model with a bipartite structure. The standard binary RBM consists of m visible variables $\boldsymbol{V} = (V_1, \dots V_m)$ taking states $\boldsymbol{v} \in \{0,1\}^m$ and n hidden variables $\boldsymbol{H} = (H_1, \dots H_n)$ taking states $\boldsymbol{h} \in \{0,1\}^n$. The joint distribution is a Gibbs distribution $p(\boldsymbol{v}, \boldsymbol{h}) = \frac{1}{Z}\tilde{p}(\boldsymbol{v}, \boldsymbol{h}) = \frac{1}{Z}e^{-\mathcal{E}(\boldsymbol{v},\boldsymbol{h})}$ with energy $\mathcal{E}(\boldsymbol{v}, \boldsymbol{h}) = -\boldsymbol{v}^T\boldsymbol{W}\boldsymbol{h} - \boldsymbol{v}^T\boldsymbol{b} - \boldsymbol{c}^T\boldsymbol{h}$, where $\boldsymbol{W}, \boldsymbol{b}, \boldsymbol{c}$ are the weight matrix and the visible and hidden bias vectors, respectively. The normalization constant $Z = \sum_{\boldsymbol{v}}\sum_{\boldsymbol{h}} e^{-\mathcal{E}(\boldsymbol{v},\boldsymbol{h})}$ (also referred to as partition function) is typically unknown, because it is calculated by first marginalizing over either the visible or the hidden units and then summing over the states of the remaining units, which is exponential in $\min\{n, m\}$.

In this paper, we focus on the problem of maximum log-likelihood fitting of the RBM distribution to a data set $S = \{\boldsymbol{v}_1, \dots \boldsymbol{v}_\ell\}, \boldsymbol{v}_i \in \{0,1\}^m$. The gradient of $\log p(\boldsymbol{x}_i)$ w.r.t. the model parameters $\boldsymbol{\theta} = (\boldsymbol{W}, \boldsymbol{b}, \boldsymbol{c})$ is given by

$$\frac{\partial \log p(\boldsymbol{v}_i)}{\partial \boldsymbol{\theta}} = -E_{p(\boldsymbol{h}|\boldsymbol{v}_i)}\left[\frac{\partial \mathcal{E}(\boldsymbol{v}_i, \boldsymbol{h})}{\partial \boldsymbol{\theta}}\right] + E_{p(\boldsymbol{v},\boldsymbol{h}')}\left[\frac{\partial \mathcal{E}(\boldsymbol{v}, \boldsymbol{h}')}{\partial \boldsymbol{\theta}}\right] . \quad (1)$$

While the first term of the derivative can be computed analytically, the calculation of the second term requires to sum over all $\boldsymbol{v} \in \{0,1\}^m$, which is intractable. Instead, samples are used to obtain an estimate of the expectation under the RBM distribution. The RBM training algorithm most often used in literature is CD, which approximates the expectation over $p(\boldsymbol{v}, \boldsymbol{h}')$ by a sample gained after k steps of Gibbs-sampling starting from sample $\boldsymbol{v}_i$.

Population Monte Carlo. Being a MCMC method, PMC [2] aims at estimating expectations of the form

$$E_{p(\boldsymbol{x})}[f(\boldsymbol{x})] = \int p(\boldsymbol{x}) f(\boldsymbol{x}) \mathrm{d}\boldsymbol{x} . \quad (2)$$

When samples $\boldsymbol{x}_1, \dots \boldsymbol{x}_N$ from p are available, this expectation can be approximated by

$$E_{p(\boldsymbol{x})}[f(\boldsymbol{x})] \approx \frac{1}{N}\sum_{i=1}^{N} f(\boldsymbol{x}_i) . \quad (3)$$

Often it is not possible to generate samples from p directly. In this case, MCMC methods provide a way to sample from p by running k steps of a Markov chain that has p as its stationary distribution.

In practice, it is computationally too demanding to choose k large enough to guarantee convergence. Consequently, the samples $\boldsymbol{x}_i$ generated by the Markov chain are not drawn from p but from another distribution which we will refer to as q. Thus, using the samples directly in (3) leads to a biased estimate for (2).

In this case, importance sampling, where q is regarded as proposal distribution, could allow for getting an unbiased estimate of (2) based on samples from q:

$$E_{p(\boldsymbol{x})}[f(\boldsymbol{x})] = E_{q(\boldsymbol{x})}\left[\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})} f(\boldsymbol{x})\right] \quad (4)$$

However, the distribution q that results from running a Markov chain for k-steps is usually unknown and therefore a weighting scheme as in Eq. (4) can not be applied. Instead one can use any known conditional distribution $\kappa(\boldsymbol{x}'|\boldsymbol{x}) > 0$ as proposal distribution for importance sampling since

$$E_{p(\boldsymbol{x})}[f(\boldsymbol{x})] = E_{q(\boldsymbol{x})}\left[E_{\kappa(\boldsymbol{x}'|\boldsymbol{x})}\left[\frac{p(\boldsymbol{x}')}{\kappa(\boldsymbol{x}'|\boldsymbol{x})} f(\boldsymbol{x}')\right]\right] . \quad (5)$$

This re-weighting scheme does not require the knowledge of the probabilities $q(\boldsymbol{x})$ but just the ability to sample from q.

Eq. (5) requires that we know $p(\boldsymbol{x})$, but in probabilistic modelling often only the unnormalized probability $\tilde{p}(\boldsymbol{x}) = Zp(\boldsymbol{x})$ is available. Thus, Z needs to be estimated as well. Since $E_{\kappa(\boldsymbol{x}'|\boldsymbol{x})}\left[\frac{\tilde{p}(\boldsymbol{x}')}{\kappa(\boldsymbol{x}'|\boldsymbol{x})}\right] = Z$ for all $\boldsymbol{x}$, we get a biased importance sampling estimate by

$$E_{p(\boldsymbol{x})}[f(\boldsymbol{x})] \approx \sum_{i=1}^{N} \frac{\omega_i f(\boldsymbol{x}_i)}{\sum_{j=1}^{N}\omega_j} , \quad (6)$$

where $\boldsymbol{x}_i' \sim q(\cdot)$, $\boldsymbol{x}_i \sim \kappa(\cdot|\boldsymbol{x}_i')$, and $\omega_i = \tilde{p}(\boldsymbol{x}_i)/\kappa(\boldsymbol{x}_i|\boldsymbol{x}_i')$. The estimate becomes unbiased in the limit of $N \to \infty$ and thus the estimator is consistent.

A PMC method makes use of Eqs. (5) and (6) to create a Markov chain, where in step t states $\boldsymbol{y}_i^{(t)} \sim \kappa(\cdot|\boldsymbol{x}_i^{(t-1)})$, $i = 1, \dots, N$ are sampled. For the N states importance weights ω_i are estimated and the states are then re-sampled accordingly. The re-sampling procedure results in the samples $\boldsymbol{x}_1^{(t)}, \dots, \boldsymbol{x}_N^{(t)}$ which corresponds to the t-th state of the chain. The chain thus only depends on the choice of $\kappa(\cdot|\boldsymbol{x})$ which should be chosen to have heavy tails as to reduce the variance of the estimate.

3. Population-Contrastive-Divergence

In this section, we introduce the main contribution of this paper, the Population-Contrastive-Divergence (pop-CD) algorithm for RBM training. This algorithm exploits the features of CD and Population-MCMC by combining the efficient implementation of CD-k with the low bias re-weighting scheme introduced by PMC.

Let us rewrite the log-likelihood gradient in Eq. (1) by setting $f(\boldsymbol{v}) = E_{p(\boldsymbol{h}''|\boldsymbol{v})}\left[\frac{\partial \mathcal{E}(\boldsymbol{v},\boldsymbol{h}'')}{\partial \boldsymbol{\theta}}\right]$ and estimating $E_{p(\boldsymbol{v})}[f(\boldsymbol{v})] = E_{p(\boldsymbol{v},\boldsymbol{h}'')}\left[\frac{\partial \mathcal{E}(\boldsymbol{v},\boldsymbol{h}'')}{\partial \boldsymbol{\theta}}\right]$ based on the re-weighting scheme of Eq. (5). For this, we choose q to be the distribution of the hidden variables generated by $k-1$-steps of block-wise Gibbs-sampling when starting from the i-th training example $\boldsymbol{v}_i$, which we will call $q_i^{(k-1)}(\boldsymbol{h}')$ in the following. The proposal distribution κ is set to the conditional distribution of the visible variables given the state of the hidden, i.e., to $p(\boldsymbol{v}|\boldsymbol{h}')$. This yields

$$\frac{\partial \log p(\boldsymbol{v}_i)}{\partial \boldsymbol{\theta}} = -E_{p(\boldsymbol{h}|\boldsymbol{v}_i)}\left[\frac{\partial \mathcal{E}(\boldsymbol{v}_i, \boldsymbol{h})}{\partial \boldsymbol{\theta}}\right] + E_{q_i^{(k-1)}(\boldsymbol{h}')}\left[E_{p(\boldsymbol{v}|\boldsymbol{h}')}\left[\frac{p(\boldsymbol{v})}{p(\boldsymbol{v}|\boldsymbol{h}')} f(\boldsymbol{v})\right]\right] . \quad (7)$$

In the case of $k = 1$ we can write $q_i^{(0)}(\boldsymbol{h}') = p(\boldsymbol{h}'|\boldsymbol{v}_i)$ and for $k \geq 2$ the distribution $q_i^{(k-1)}$ can be written as a recursive relation over the distribution of $q_i^{(k-2)}$:

$$q_i^{(k-1)}(\boldsymbol{h}') = \sum_{\boldsymbol{h}'' \in H}\sum_{\boldsymbol{v} \in V} p(\boldsymbol{h}'|\boldsymbol{v}) p(\boldsymbol{v}|\boldsymbol{h}'') q_i^{(k-2)}(\boldsymbol{h}'')$$

Now, based on the considerations in the previous section, it follows from (7) that we can estimate the second term of the gradient given the training set $S = \{\boldsymbol{v}_1, \dots \boldsymbol{v}_\ell\}$ by (6), where $\boldsymbol{h}_i' \sim q_i^{(k-1)}(\cdot)$, $\boldsymbol{v}_i' \sim p(\cdot|\boldsymbol{h}_i')$ and $\omega_i = \tilde{p}(\boldsymbol{v}_i')/p(\boldsymbol{v}_i'|\boldsymbol{h}_i')$. This results in the following estimate of the log-likelihood gradient

$$-\frac{1}{\ell}\sum_{i=1}^{\ell} E_{p(\boldsymbol{h}|\boldsymbol{v}_i)}\left[\frac{\partial \mathcal{E}(\boldsymbol{v}_i, \boldsymbol{h})}{\partial \boldsymbol{\theta}}\right] + \sum_{i=1}^{\ell} \frac{\omega_i E_{p(\boldsymbol{h}|\boldsymbol{v}_i')}\left[\frac{\partial \mathcal{E}(\boldsymbol{v}_i', \boldsymbol{h})}{\partial \boldsymbol{\theta}}\right]}{\sum_{j=1}^{\ell}\omega_j} , \quad (8)$$

which we will refer to as pop-CD-k. Setting $\omega_i = 1$ or $\ell = 1$ leads to CD-k.

In an actual implementation, it is advisable to work with $\log\omega$ for increased numerical stability. Algorithm 1 describes pop-CD in pseudo code.

Algorithm 1: k-step pop-CD.

Input: RBM $(V_1, \dots, V_m, H_1, \dots, H_n)$, training batch S of size ℓ
Output: gradient approximation Δw_{ij}, Δb_j and Δc_i for $i = 1, \dots, n,\ j = 1, \dots, m$

1 init $\Delta w_{ij}^+ = \Delta w_{ij}^- = \Delta b_j^+ = \Delta b_j^- = \Delta c_i^+ = \Delta c_i^- = 0$ for $i = 1, \dots, n,\ j = 1, \dots, m,\ \omega_S = 0$
2 **forall the** $\vec{v} \in S$ **do**
3 $\vec{v}^{(0)} \leftarrow \vec{v}$
4 **for** $t = 0, \dots, k-1$ **do**
5 sample $\vec{h}^{(t)} \sim p(\vec{h} \mid \vec{v}^{(t)})$
6 sample $\vec{v}^{(t+1)} \sim p(\vec{v} \mid \vec{h}^{(t)})$
7 $\omega \leftarrow \frac{p(\vec{v}^{(k)})}{p(\vec{v}^{(k)} \mid \vec{h}^{(k-1)})}$
8 $\omega_S \leftarrow \omega_S + \omega$
9 **for** $i = 1, \dots, n,\ j = 1, \dots, m$ **do**
10 $\Delta w_{ij}^+ \leftarrow \Delta w_{ij}^+ + p(H_i = 1 \mid \vec{v}^{(0)}) \cdot v_j^{(0)}$
11 $\Delta w_{ij}^- \leftarrow \Delta w_{ij}^- + \omega p(H_i = 1 \mid \vec{v}^{(k)}) \cdot v_j^{(k)}$
12 **for** $j = 1, \dots, m$ **do**
13 $\Delta b_j^+ \leftarrow \Delta b_j^+ + v_j^{(0)}$
14 $\Delta b_j^- \leftarrow \Delta b_j^- + \omega v_j^{(k)}$
15 **for** $i = 1, \dots, n$ **do**
16 $\Delta c_i^+ \leftarrow \Delta c_i^+ + p(H_i = 1 \mid \vec{v}^{(0)})$
17 $\Delta c_i^- \leftarrow \Delta c_i^- + \omega p(H_i = 1 \mid \vec{v}^{(k)})$
18 **for** $i = 1, \dots, n,\ j = 1, \dots, m$ **do**
19 $\Delta w_{ij} \leftarrow \frac{1}{\ell}\Delta w_{ij}^+ - \frac{1}{\omega_S}\Delta w_{ij}^-$
20 $\Delta b_j \leftarrow \frac{1}{\ell}\Delta b_j^+ - \frac{1}{\omega_S}\Delta b_j^-$
21 $\Delta c_i \leftarrow \frac{1}{\ell}\Delta c_i^+ - \frac{1}{\omega_S}\Delta c_i^-$

Runtime. Going from CD to pop-CD does not change the asymptotic runtime. Eq. (8) does not produce a noteworthy computational overhead compared to CD, because the only entities that are additionally computed are the weights ω_i. The weights in turn are inexpensive to determine as the distribution $p(\boldsymbol{v}|\boldsymbol{h}^{(k-1)})$ is already computed (see Algorithm 1, line 6) and calculating $p(\boldsymbol{v}^{(k)})$ can reuse $\boldsymbol{W}\boldsymbol{v}^{(k)}$, which has already been computed for determining $p(H_i = 1 \mid \boldsymbol{v}^{(k)})$ for the gradient update. Thus, the cost to compute ω_i is negligible compared to the sampling cost. The memory requirements increase compared to standard CD, however, only by the number of model parameters and, thus, typically negligible compared to the data set size.

Bias. It is known that Eq. (6) is biased for two reasons. The first reason is that the same samples are used to estimate the gradient as for the estimation of Z. The second one is that, even assuming an unbiased estimator for Z, this does not lead to an unbiased estimate of $1/Z$. It is easy to remove the bias from using the same samples by rewriting (6) as

$$E_{p(\boldsymbol{x})}[f(\boldsymbol{x})] \approx \frac{N-1}{N} \sum_{i=1}^{N} \frac{\omega_i f(\boldsymbol{x}_i)}{\sum_{j=1, j\neq i}^{N} \omega_j} \quad , \tag{9}$$

that is, by simply removing the ith sample from the estimator of Z when estimating $p(\boldsymbol{x}_i)$. This yields an estimator which converges in expectation over all possible sets of N samples to

$$E\left[\frac{N-1}{N} \sum_{i=1}^{N} \frac{\omega_i f(\boldsymbol{x}_i)}{\sum_{j=1, j\neq i}^{N} \omega_j}\right] = ZE\left[\frac{N-1}{\sum_{j=1}^{N-1} \omega_j}\right] E_{p(\boldsymbol{x})}[f(\boldsymbol{x})] \quad ,$$

which is still biased in the regime of finite sample size since $E\left[\frac{N-1}{\sum_{j=1}^{N-1} \omega_j}\right] \neq \frac{1}{Z}$. Applying this result to the problem of estimating the log-likelihood gradient (1), the pop-CD gradient converges to

$$E_{p(\boldsymbol{h}|x_i)}\left[\frac{\partial \mathcal{E}(\boldsymbol{x}_i, \boldsymbol{h})}{\partial \boldsymbol{\theta}}\right] - ZE\left[\frac{N-1}{\sum_{j=1}^{N-1} \omega_j}\right] E_{p(\boldsymbol{v},\boldsymbol{h})}\left[\frac{\partial \mathcal{E}(\boldsymbol{v}, \boldsymbol{h})}{\partial \boldsymbol{\theta}}\right] \quad .$$

Thus, using (9), only the length of the right term of the gradient will be biased and not its direction.

In practice, it is not advisable to employ (9) instead of (6) as usually only a small number of samples is used. In this case, the estimate of Z will have a high variance and the length of the gradient obtained by (9) can vary by several orders of magnitude. This variance leads to a small effective sample size in (6).

Alternative formulations. In our approach, the weights $\omega_i = \tilde{p}(\boldsymbol{v}_i')/p(\boldsymbol{v}'|\boldsymbol{h}_i')$ solely depend on the current samples $\boldsymbol{h}_i'$. Therefore, only little information about the distribution $q(\boldsymbol{h}')$ enters the learning process. Now, one could ask the question whether it is beneficial to use the full sample $(\boldsymbol{v}', \boldsymbol{h}')$ from the sampling distribution q instead and defined in Eq. (7) the proposal distribution to be $\kappa(\boldsymbol{v}, \boldsymbol{h}|\boldsymbol{v}', \boldsymbol{h}')$. In this case, the weights become $\omega = \tilde{p}(\boldsymbol{v}, \boldsymbol{h})/\kappa(\boldsymbol{v}, \boldsymbol{h}|\boldsymbol{v}', \boldsymbol{h}')$. As the typical choice for q is the block-wise Gibbs-sampler, we get $\kappa(\boldsymbol{v}, \boldsymbol{h}|\boldsymbol{v}', \boldsymbol{h}') = p(\boldsymbol{v}|\boldsymbol{h})p(\boldsymbol{h}|\boldsymbol{v}')$ and $\omega = \tilde{p}(\boldsymbol{h})/p(\boldsymbol{h}|\boldsymbol{v}')$, which again depends on a single $\boldsymbol{v}'$.

A different approach is to tie several samples of $q(\boldsymbol{h}')$ together in a mixture proposal distribution $\kappa(\boldsymbol{v}|\boldsymbol{h}_1', \dots, \boldsymbol{h}_\tau') = \frac{1}{\tau}\sum_i p(\boldsymbol{v}|\boldsymbol{h}_i')$. The advantage of this approach is that the proposal distribution takes distribution information of $\boldsymbol{h}'$ into account. In fact, $q(\boldsymbol{v}|\boldsymbol{h}_1', \dots, \boldsymbol{h}_\tau') \overset{\tau\to\infty}{\longrightarrow} \sum_{\boldsymbol{h}\in H} q(\boldsymbol{h})p(\boldsymbol{v}|\boldsymbol{h})$. The latter is a good proposal distribution assuming $q(\boldsymbol{h}) \approx p(\boldsymbol{h})$, however, the computational cost of computing the importance weights rises linearly with the number of samples in the mixture.

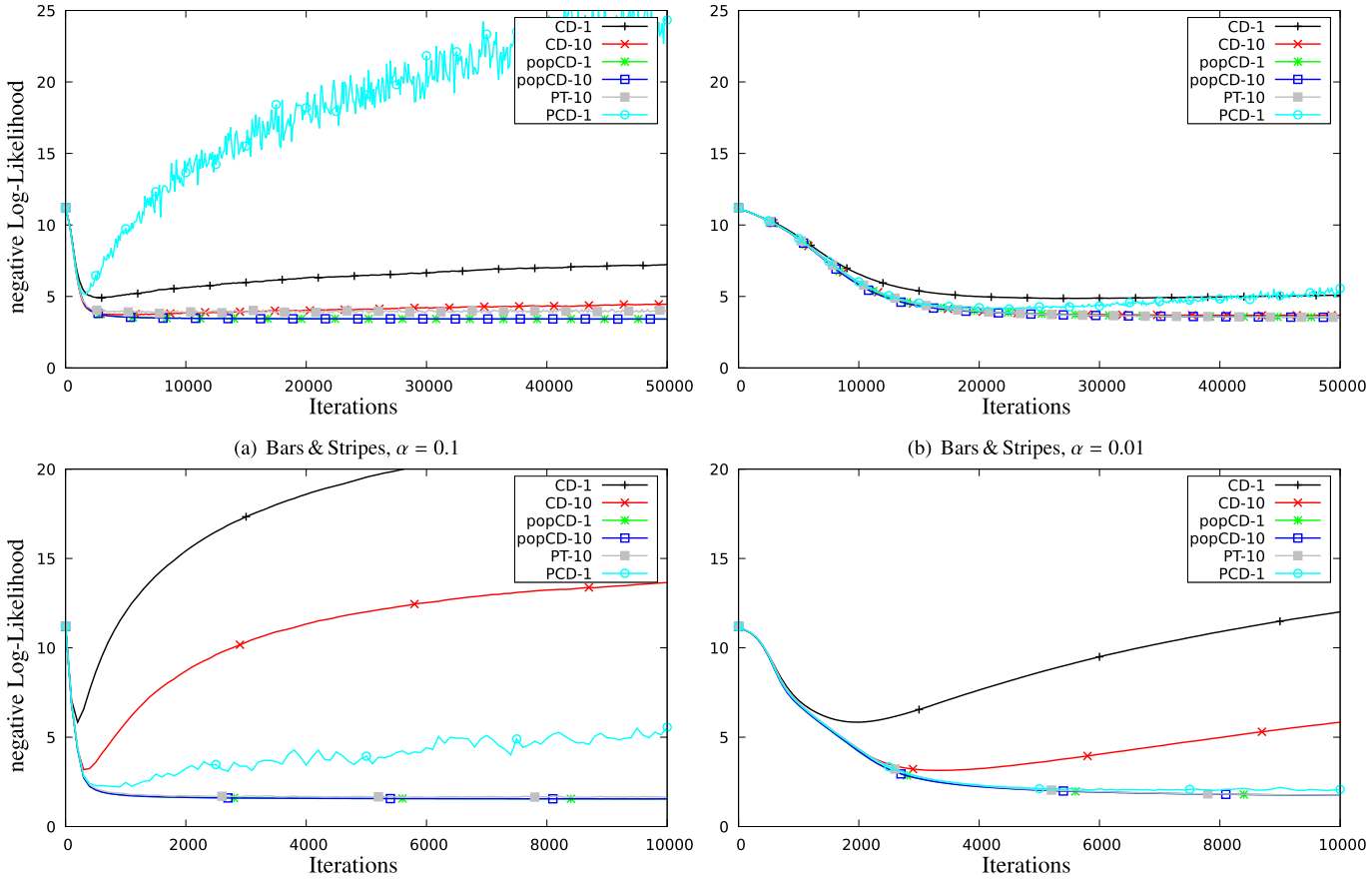
4. Experiments

Experimental setup

We compared the new pop-CD-k algorithm experimentally to previously proposed methods. We implemented the algorithms in the machine learning library SHARK [9]. We chose two small artificial datasets, Bars & Stripes [11] and the data set used by [3] and [1] which we will refer to as Artificial Modes, as well as two larger real world datasets, MNIST [10] and Letters[1]. Artificial Modes was created to generate distributions for which Gibbs-sampling has problems to mix by setting its control parameter to $p = .001$. The datasets Bars & Stripes and Artificial Modes are known to lead to a bias causing the log-likelihood to diverge for CD and PCD [3,4]. This makes them good benchmarks for analysing the bias of different gradient estimation strategies.

In all experiments, except the ones on Bars & Stripes, we performed mini-batch learning. We *a priori* set the size of the batches, and thus also the number of samples generated for the estimation of the model mean, to be a divisor of the full dataset size. The batch sizes were 32 for Bars & Stripes, 500 for Artificial Modes, 246 for Letters, and 500 for MNIST. We did not use a momentum term in all our experiments.

[1] http://ai.stanford.edu/~btaskar/ocr/.



(a) Bars & Stripes, $\alpha = 0.1$

(b) Bars & Stripes, $\alpha = 0.01$

(c) Artificial Modes, $\alpha = 0.1$

(d) Artificial Modes, $\alpha = 0.01$

Fig. 1. Training curves for Bars & Stripes and Artificial Modes with 16 hidden neurons and different learning rates. The two pop-CD variants cannot be distinguished. Furthermore, the PT curve cannot be distinguished from pop-CD on Artificial Modes.

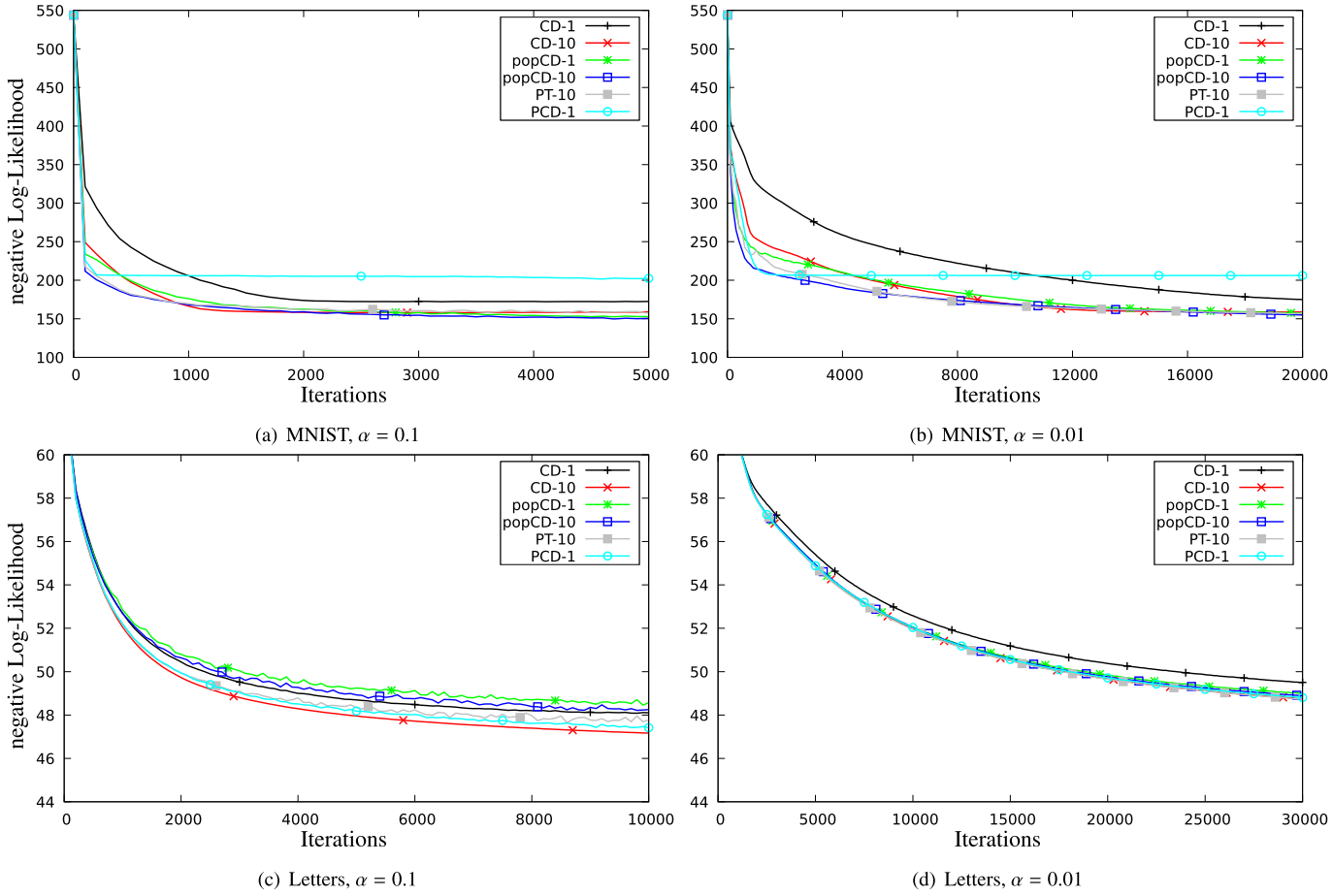
The first set of experiments considered the behaviour of the algorithm in settings in which the normalisation constant of the RBM can be computed analytically. We selected the number of hidden neurons to be 16 and we compared the algorithms CD-1, CD-10, PCD-1, pop-CD-1, pop-CD-10, and PT with 10 parallel chains (and inverse temperatures uniformly distributed in [0, 1]). We choose the learning rates to be $\alpha = 0.1$ and $\alpha = 0.01$ for all datasets. We let the algorithm run for 50,000 and 10,000 iterations (where one iteration corresponds to one step of gradient ascent) on Bars & Stripes and Artificial Modes, respectively. On MNIST we trained for 5000 iterations with $\alpha = 0.1$ and for 20,000 with $\alpha = 0.01$, and on Letters we trained for 10,000 iterations with $\alpha = 0.1$ and 30,000 iterations with $\alpha = 0.01$. We evaluated the log-likelihood every 100 iterations. All experiments were repeated 25 times and the reported log-likelihood values are the means over all trials.

In our second set of experiments, we trained RBMs with a large number of hidden neurons. Here the normalisation constant was estimated using Annealed Importance Sampling (AIS, [12]) with 512 samples and 50,000 intermediate distributions. The learning parameters were $\alpha = 0.1$ with $5 \cdot 10^5$ iterations and $\alpha = 0.01$ with $5 \cdot 10^6$ iterations for Letters and for MNIST $\alpha = 0.01$ with $4 \cdot 10^4$ iterations and $\alpha = 0.001$ with $4 \cdot 10^5$ iterations. Due to the long running times we do not consider PCD and PT and only show one trial for MNIST. For Letters, the results are the average over 10 trials.

In a third set of experiments, we investigated the bias and the variance of the methods. We first trained an RBM using CD-1 with $\alpha = 0.1$ and 16 hidden neurons on all four problems. The number of training iterations was 10,000 for Bars & Stripes, Artificial Modes and Letters while 5000 on MNIST. Furthermore, we repeated the large experiments by training an RBM with $\alpha = 0.01$ and 500 hidden units for 15,000 iterations on MNIST and with $\alpha = 0.1$, 100 hidden units and 10^5 iterations on Letters. After that, we calculated a "ground truth " estimate of the true gradient. This was done by PT with 100 intermediate chains. The chain was given 50,000 iterations burn-in time and afterwards 200,000 samples were taken to compute the estimate. Then, we generated 50,000 gradient estimates of CD-k and pop-CD-k for $k \in \{1, 10\}$. Finally, we estimated the bias of the methods as the squared distance between the empirical average of the methods and the estimated ground truth and the variance as the expectation of the squared distance between the single samples and their mean.

Results

The results of the first set of experiments can be seen in Fig. 1 for the artificial datasets and in Fig. 2 for MNIST and Letters. Fig. 1 shows that pop-CD-1 as well as pop-CD-10 outperformed CD-1 and CD-10. The proposed algorithm reached significantly higher log-likelihood values and, while CD and PCD diverged, pop-CD did not show any sign of divergence. Using $k = 1$ and $k = 10$ sampling steps lead to almost the same performance of pop-CD. The efficient pop-CD-1 performed on a par or better than the computationally more expensive PT-10. On the two real datasets, as illustrated in Fig 2, CD did not cause any divergence. However, while all algorithms performed almost the same on Letters, pop-CD-1 still reaches higher likelihood values than CD-1 on MNIST, which stopped showing improvement after 3000 iterations



(a) MNIST, $\alpha = 0.1$ (b) MNIST, $\alpha = 0.01$

(c) Letters, $\alpha = 0.1$ (d) Letters, $\alpha = 0.01$

Fig. 2. Training curves for MNIST and Letters with 16 hidden neurons.

in Fig. 2(a). Furthermore, one can observe that pop-CD performed slightly better with a smaller learning rate of $\alpha = 0.01$ than with a learning rate of $\alpha = 0.1$.

The results for the second set of experiments are given in Fig. 3. The plots for the Letters data set (Fig. 3(a) and (b)) look qualitatively the same, aside from the scaling of the abscissa. In both cases, CD-1 and CD-10 performed better than pop-CD-1 and pop-CD-10. On MNIST (Fig. 3(c) and (d)), both CD-1 and CD-10 diverged after the initial learning phase. For $\alpha = 0.01$ pop-CD showed a very slow learning progress, while for $\alpha = 0.001$ all four algorithms exhibit similar behaviour. While all algorithms diverge in the long run, divergence was less pronounced for pop-CD-k than CD-k.

Table 1 shows the measured bias and variance for the third experimental setup. For the experiments with 16 hidden neurons, CD-k always had a smaller variance than pop-CD-k, often by an order of magnitude or more. The bias of pop-CD-k was in contrast much smaller, which explains the experimental results we observed. In the experiments with 500 hidden units, pop-CD-k had a larger bias as well.

In all our experiments pop-CD-1 took approximately 10% more time to finish compared to CD-1, while for pop-CD-10 there was only a negligible difference to CD-10.

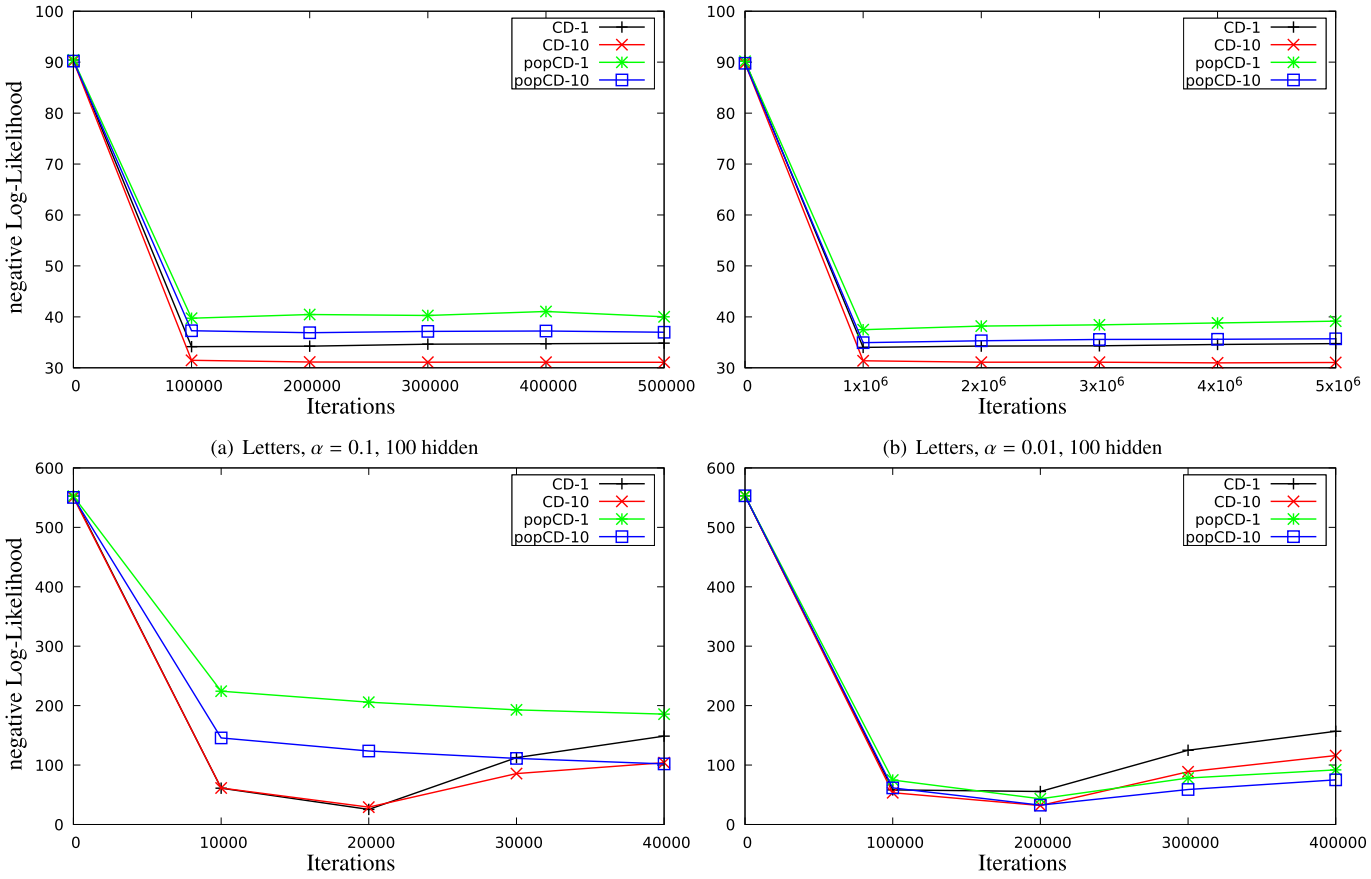
Table 1
Estimated bias and variance of CD-k and pop-CD-k divided by the number of parameters.

Problem	k	hidden	CD-k		Pop-CD-k	
			Variance	Bias	Variance	Bias
Bars & Stripes	1	16	$5 \cdot 10^{-5}$	$2 \cdot 10^{-2}$	$5 \cdot 10^{-4}$	$2 \cdot 10^{-4}$
	10	16	$2 \cdot 10^{-4}$	$2 \cdot 10^{-2}$	10^{-3}	10^{-4}
Artificial Modes	1	16	$2 \cdot 10^{-6}$	$2 \cdot 10^{-1}$	$4 \cdot 10^{-4}$	$8 \cdot 10^{9}$
	10	16	$5 \cdot 10^{-6}$	$2 \cdot 10^{-1}$	$4 \cdot 10^{-4}$	$6 \cdot 10^{-9}$
Letters	1	16	$3 \cdot 10^{-4}$	$2 \cdot 10^{-3}$	$2 \cdot 10^{-2}$	$7 \cdot 10^{-4}$
	10	16	$4 \cdot 10^{-4}$	$2 \cdot 10^{-3}$	10^{-2}	$3 \cdot 10^{-4}$
Letters	1	100	$7 \cdot 10^{-5}$	$3 \cdot 10^{-3}$	10^{-2}	10^{-2}
	10	100	$2 \cdot 10^{-4}$	$3 \cdot 10^{-3}$	$2 \cdot 10^{-2}$	$9 \cdot 10^{-3}$
MNIST	1	16	10^{-4}	10^{-2}	$3 \cdot 10^{-3}$	$7 \cdot 10^{-4}$
	10	16	$2 \cdot 10^{-4}$	10^{-2}	$2 \cdot 10^{-3}$	$7 \cdot 10^{-4}$
MNIST	1	500	$3 \cdot 10^{-5}$	$8 \cdot 10^{-3}$	$6 \cdot 10^{-3}$	10^{-2}
	10	500	$6 \cdot 10^{-5}$	$7 \cdot 10^{-3}$	$6 \cdot 10^{-3}$	$3 \cdot 10^{-3}$

Discussion

In most experiments, the new method performed on a par or even better compared to computational much more expensive sampling techniques such as PT. However, for problems where the bias of CD does not notably impair the learning, pop-CD can be slower than CD, because it may require smaller learning rates for stochastic gradient descent.

The fact that pop-CD fares better with smaller learning rates can be explained by a larger variance, as measured in all our experiments, see Table 1. This is due to the bias-variance-trade-off, which is a well known issue for importance sampling based estimators. While having potentially low bias, the variance of importance sampling based estimators depends on how well the proposal function approximates the target distribution. Thus, our results indicate that the used proposal function can lead to a high variance and may thus not be the optimal choice for complex tar-



(a) Letters, $\alpha = 0.1$, 100 hidden

(b) Letters, $\alpha = 0.01$, 100 hidden

(c) MNIST, $\alpha = 0.01$, 500 hidden

(d) MNIST, $\alpha = 0.001$, 500 hidden

Fig. 3. Training curves for MNIST and Letters using larger RBMs.

get distributions. When working with a small sample size compared to the difficulty of the estimation problem or with a bad proposal distribution, the estimate of the normalisation constant can have a large variance, which leads to a biased estimate of the inverse normalisation constant. This is supported by our results in the experiments with a large number of hidden units as well as by the bias estimates, see Table 1.

We observed that more steps of Gibbs-sampling help when using pop-CD for larger models. The reason is that more sampling steps increase the sample quality and thus reduce the variance of importance sampling.

5. Conclusions

Improving the sample quality of the Markov chain (by increasing the number of sampling steps or employing advanced sampling techniques like PT) is not the only direction leading to low bias estimates—importance sampling can also reduce the bias efficiently. We have proposed a new variant of the Contrastive Divergence (CD) algorithm inspired by the Population Monte Carlo method. The new learning algorithm, termed Population-CD (pop-CD), incorporates importance weights into the sampling scheme of CD. This makes the bias of the log-likelihood gradient estimate independent of the Markov chain and sampling operators used, with negligible computational overhead compared to CD. However, this comes at the cost of an increased variance of the estimate. Further the bias of the method depends on how well the empirical mean of the importance weights estimates the normalisation constant of the distribution.

In contrast to CD, pop-CD is consistent. For problems with a small number of hidden neurons it clearly outperforms CD with the same computational cost, leading to higher likelihood values. However, for RBMs with many hidden neurons, pop-CD may require a large number of samples to achieve a smaller bias than CD—therefore, CD is still recommended in that case. The reason for this is that our current estimator relies on samples from a set of simple proposal distributions, $p(\boldsymbol{v}|\boldsymbol{h}')$, which do not use information about the distribution $q(\boldsymbol{h}')$ of the hidden samples. An importance sampling based estimator using a simple proposal distribution can get worse than an estimator which is directly based on samples from the Markov chain. Hence, future work must investigate whether there exists a better choice for the proposal distribution bridging the gap between $q(\boldsymbol{h}')$ and the true distribution $p(\boldsymbol{h})$.

Acknowledgments

Christian and Oswin acknowledge support from the Innovation Fund Denmark through the *Danish Center for Big Data Analytics Driven Innovation* (DABAI).

References

[1] K. Brügge, A. Fischer, C. Igel, The flip-the-state transition operator for restricted Boltzmann machines, Mach. Learn. 13 (2013) 53–69.
[2] O. Cappé, A. Guillin, J.-M. Marin, C.P. Robert, Population monte carlo, J. Comput. Graph. Stat. 13 (4) (2004).
[3] G. Desjardins, A.C. Courville, Y. Bengio, P. Vincent, O. Delalleau, Tempered Markov chain Monte Carlo for training of restricted Boltzmann machines, in: Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS 2010), in: JMLR: W&C, 9, 2010, pp. 145–152.

[4] A. Fischer, C. Igel, Empirical analysis of the divergence of Gibbs sampling based learning algorithms for restricted Boltzmann machines, in: K. Diamantaras, W. Duch, L.S. Iliadis (Eds.), International Conference on Artificial Neural Networks (ICANN 2010), LNCS, 6354, Springer-Verlag, 2010, pp. 208–217.
[5] A. Fischer, C. Igel, Bounding the bias of contrastive divergence learning, Neural Comput. 23 (3) (2011) 664–673.
[6] A. Fischer, C. Igel, Training restricted boltzmann machines: an introduction, Pattern Recognit. 47 (2014) 25–39.
[7] A. Fischer, C. Igel, A bound for the convergence rate of parallel tempering for sampling restricted Boltzmann machines, Theor. Comput. Sci. 598 (2015) 102–117.
[8] G.E. Hinton, Training products of experts by minimizing contrastive divergence, Neural Comput. 14 (8) (2002) 1771–1800.
[9] C. Igel, T. Glasmachers, V. Heidrich-Meisner, Shark, J. Mach. Learn. Res. 9 (2008) 993–996.
[10] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE 86 (11) (1998) 2278–2324.
[11] D.J.C. MacKay, Information Theory, Inference, and Learning Algorithms, 7, Cambridge University Press, 2002.
[12] R.M. Neal, Annealed importance sampling, Stat. Comput. 11 (2) (2001) 125–139.
[13] R. Salakhutdinov, Learning in Markov random fields using tempered transitions, in: Y. Bengio, D. Schuurmans, J. Lafferty, C.K.I. Williams, A. Culotta (Eds.), Advances in Neural Information Processing Systems 22 (NIPS), 2009, pp. 1598–1606.
[14] H. Schulz, A. Müller, S. Behnke, Investigating convergence of restricted Boltzmann machine learning, in: H. Lee, M. Ranzato, Y. Bengio, G.E. Hinton, Y. LeCun, A.Y. Ng (Eds.), NIPS 2010 Workshop on Deep Learning and Unsupervised Feature Learning, 2010.
[15] P. Smolensky, Information processing in dynamical systems: Foundations of harmony theory, in: D.E. Rumelhart, J.L. McClelland (Eds.), Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol. 1: Foundations, MIT Press, 1986, pp. 194–281.
[16] I. Sutskever, T. Tieleman, On the convergence properties of contrastive divergence, in: Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS 2010), in: JMLR: W&C, 9, 2010, pp. 789–795.
[17] T. Tieleman, Training restricted Boltzmann machines using approximations to the likelihood gradient, in: Proceedings of the 25th International Conference on Machine Learning (ICML 2008), ACM, 2008, pp. 1064–1071.