Information and Communication Theory Group
Delft University of Technology
http://ict.ewi.tudelft.nl/

P.O. Box 5013
2600 GA Delft
The Netherlands

**January 26, 2010**

**EWI-ICT TR 2010–001**

# On Herding in Deep Networks

**Laurens van der Maaten**
ICT Group, Delft University of Technology

## Abstract

Maximum likelihood learning in Markov Random Fields (MRFs) with multiple layers of hidden units is typically performed using contrastive divergence or one of its variants. After learning, samples from the model are generally used to estimate expectations under the model distribution. Recently, Welling proposed a new approach to working with MRFs with a single layer of hidden units. The approach, called herding, tries to combine the two stages, learning and sampling, into a single stage. Herding runs the network as a deterministic dynamical system, similar to a Hopfield network. Expectations computed over trajectories of the dynamical system can be shown to converge to expectations computed over the data.
In this technical report, we investigate herding in MRFs with multiple layers of hidden units, so-called deep networks. We derive the herding dynamics for deep networks, and we investigate the effect of three important characteristics on the performance of classifiers that are trained on energy averages over the herding trajectories: (1) the effect of the step size employed in the herder, (2) the effect of different initializations of the network in the positive and in the negative phase, and (3) the effect of different network architectures. From the results of our experiments, we observe that, although in networks with a single layer of hidden units the performance can be proven to be equal for finite step sizes, the step size highly influences the performance of herders in deep networks. Moreover, our results suggest that herding in deep networks requires a different type of network architecture than deep MRF models that are trained using maximum likelihood learning. Presumably, these experimental observations can be explained by a decoupling phenomenon in the top hidden units: the top hidden units run decoupled from the data, as a result they only feed noise into the network. In order to successfully herd in deep networks, future work should develop and investigate approaches to prevent higher hidden layers from decoupling.

# On Herding in Deep Networks

**Laurens van der Maaten**
ICT Group, Delft University of Technology

## 1 Introduction

The recent development of new training approaches for deep networks, most of which are based on unsupervised greedy layer-by-layer training procedures [8], has led to renewed interest in these type of networks [1, 2, 5, 6, 8, 9, 16, 17, 19, 20, 22, 28]. Deep networks model the distribution over the data space using layers of hidden variables, in which the layers are formed by hidden units that are not interconnected[1]. The hidden variables may be either deterministic, probabilistic, or a combination of those two. The use of deterministic hidden variables gives rise to feed-forward neural networks such as autoencoders [9], whereas the use of probabilistic hidden variables gives rise to models such as deep Boltzmann machines [22]. A combination of deterministic and probabilistic hidden variables is used in models such as Deep Belief Networks [8]. In the remainder of this paper, we focus on models in which all data and hidden variables are random variables, in which the structure of the deep network specifies a specific type of factorized joint distribution.

A key characteristic of deep networks is that each layer of hidden variables (or features) can capture higher-order relations between the variables in the layers below it. As opposed to shallow architectures such as kernel machines that only provide a single layer of nonlinearity, deep networks perform multiple nonlinear transformations of the data to construct the data representation in the top hidden layer. Though both kernel machines and deep networks are universal approximators [25, 23], deep networks may be exponentially more efficient in the number of training samples they require to learn complex nonlinear functions [2, 3]. For a detailed overview on the differences between deep and shallow architectures, and the potential advantages of the use of deep networks, we refer to [2].

The typical way of working with deep Markov Random Field models consists of two main stages[2]. First, the parameters of the model are learned by maximizing the log-likelihood of the training data under the model. Second, the trained model is used to perform inference, for instance, to infer a posterior over classes or to infer a latent representation of the data. Recently, Welling [30, 31] proposed a radically different approach by defining a dynamical system, a so-called 'herder', that deterministically generates a sequence of parameter settings for the network. The average of the energies of data under these model instantations can in turn be used to perform classification or to construct latent data representations. Welling used herding only in fully observed MRFs [31] and in MRFs with a single layer of hidden variables [30], i.e., in a Restricted Boltzmann Machine [24]. In this paper, we extend herding to deep MRF models.

The outline of the remainder of this paper is as follows. In section 2, we give an overview of the structure and the maximum likelihood learning procedure of deep Boltzmann machines. Section 3

[1]We should note here that it is possible to work with Markov Random Fields in which all variables are interconnected, but such networks are usually very hard to train [21].

[2]One could also consider a fully Bayesian treatment of deep MRFs, but such a treatment is doubly-intractable [18].

introduces herding and derives the herding equations for deep Boltzmann machines. In section 4, we present the results of experiments with herding in deep Boltzmann machines with a variety of network architectures. The results of these experiments are discussed in section 5. Conclusions and directions for future work are presented in section 6.

## 2 Deep Boltzmann Machines

A deep Boltzmann machine [22] is a Markov Random Field (MRF) with binary variables that form $M$ layers of hidden variables, which have no lateral connections (i.e., two hidden variables in the same layer are never interconnected), and hidden layers are only connected to the two layers that are directly below and above them (except for the bottom and top layer, which are only connected to the layer above and below them, respectively). An example of a deep Boltzmann machine with $M = 3$ hidden layers is shown in Figure 1.

As for all undirected graphical models [4], the joint distribution over all nodes in the model is given by a Gibbs distribution

$$p(\mathbf{v}, \mathbf{h}^{(1)}, \ldots, \mathbf{h}^{(M)}|\theta) = \frac{1}{Z(\theta)} \exp\left(E\left(\mathbf{v}, \mathbf{h}^{(1)}, \ldots, \mathbf{h}^{(M)}; \theta\right)\right), \tag{1}$$

where we denoted the visual nodes by $\mathbf{v}$, the hidden nodes in layer $m$ by $\mathbf{h}^{(m)}$, the model parameters by $\theta$, the partition function by $Z(\theta)$, and where $E$ represents an energy function that is the sum of the potential functions over all maximal cliques in the graph. Denoting the weight matrices by $\mathbf{W}^{(m)}$, we define a linear energy function for the model

$$E\left(\mathbf{v}, \mathbf{h}^{(1)}, \ldots, \mathbf{h}^{(M)}; \theta\right) = -\mathbf{v}^T\mathbf{W}^{(1)}\mathbf{h}^{(1)} - \mathbf{v}^T\mathbf{b}^{(0)} - \sum_{m=2}^{M} \mathbf{h}^{(m-1)\,T}\mathbf{W}^{(m)}\mathbf{h}^{(m)} - \sum_{m=1}^{M} (\mathbf{h}^{(m)})^T\mathbf{b}^{(m)}, \tag{2}$$

where the model parameters are $\theta = \{\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(M)}, \mathbf{b}^{(0)}, \ldots, \mathbf{b}^{(M)}\}$. In the remainder of the paper, we will omit the biases on all random variables to avoid the notation from becoming too cluttered. The linear energy function and the binary nature of the nodes leads to the following conditional distribution over the visual nodes $\mathbf{v}$ given the states of the hidden nodes $\mathbf{h}^{(1)}$ in the first hidden layer

$$p\left(\mathbf{v}|\mathbf{h}^{(1)}\right) = \sigma\left(\mathbf{v}^T\mathbf{W}^{(1)}\mathbf{h}^{(1)}\right),$$

where $\sigma$ represents the sigmoid function $\sigma(x) = \frac{1}{1+\exp(-x)}$. Similarly, the conditional distributions over the $m$-th hidden layer $\mathbf{h}^{(m)}$ given the states of the nodes in the adjacent layers are given by

$$p\left(\mathbf{h}^{(m)}|\mathbf{h}^{(m-1)}, \mathbf{h}^{(m+1)}\right) = \sigma\left(\mathbf{h}^{(m-1)T}\mathbf{W}^{(m)}\mathbf{h}^{(m)} + \mathbf{h}^{(m)T}\mathbf{W}^{(m+1)}\mathbf{h}^{(m+1)}\right), \quad 1 < m < M,$$
$$p\left(\mathbf{h}^{(M)}|\mathbf{h}^{(M-1)}\right) = \sigma\left(\mathbf{h}^{(M-1)T}\mathbf{W}^{(M)}\mathbf{h}^{(M)}\right),$$

where we misused the notation by assuming that $\mathbf{v} = \mathbf{h}^{(0)}$.

The log-likelihood of a training point under a deep Boltzmann machine can be obtained by combining Equation 1 and 2, marginalizing out the hidden variables, and taking the logarithm of the resulting expression. The log-likelihood of a dataset with $N$ data points is thus proportional to

$$\mathcal{L}(\theta) = -\log Z(\theta) + \frac{1}{N}\sum_{n=1}^{N} \log \sum_{\mathbf{h}_n^{(1)},\ldots,\mathbf{h}_n^{(M)}} \exp\left(-\mathbf{v}_n^T\mathbf{W}^{(1)}\mathbf{h}_n^{(1)} - \sum_{m=2}^{M} \mathbf{h}_n^{(m-1)\,T}\mathbf{W}^{(m)}\mathbf{h}_n^{(m)}\right),$$

where $\mathbf{v}_n$ denotes the $n$-th of the $N$ training points, and $\mathbf{h}_n^{(m)}$ its corresponding node activations in the $m$-th layer (note that finding the activations $\mathbf{h}_n^{(1)}, \ldots, \mathbf{h}_n^{(M)}$ that maximize the log-likelihood is generally intractable in deep MRFs).

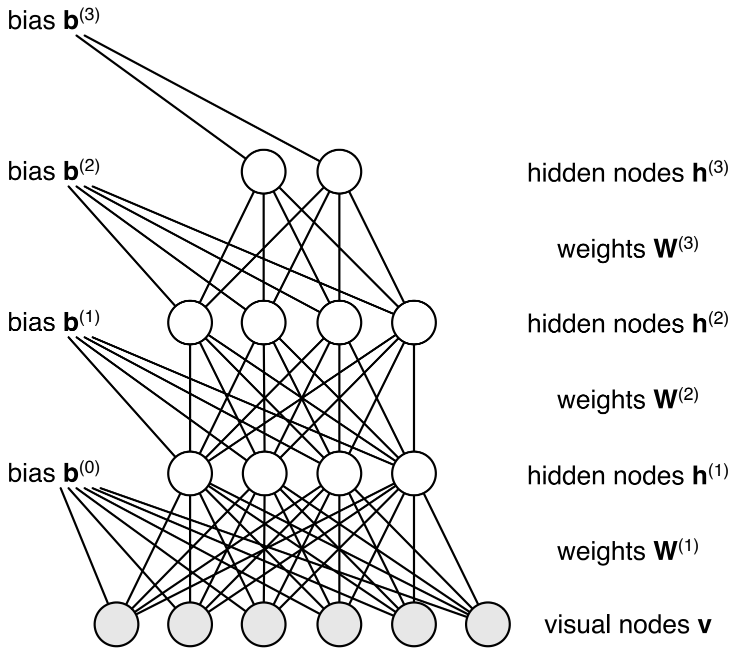

bias $\mathbf{b}^{(3)}$
bias $\mathbf{b}^{(2)}$
hidden nodes $\mathbf{h}^{(3)}$
weights $\mathbf{W}^{(3)}$
bias $\mathbf{b}^{(1)}$
hidden nodes $\mathbf{h}^{(2)}$
weights $\mathbf{W}^{(2)}$
bias $\mathbf{b}^{(0)}$
hidden nodes $\mathbf{h}^{(1)}$
weights $\mathbf{W}^{(1)}$
visual nodes $\mathbf{v}$

Figure 1: Example of a deep Boltzmann machine with $M = 3$ hidden layers.

Maximum likelihood learning in (deep) Boltzmann machines [10] is computationally very expensive, as it requires randomly initialized Markov chains to approach their equilibrium distribution for each parameter update, making it impractical for use on all but very small toy datasets. Recently, a computationally feasible training algorithm for deep Boltzmann machines was proposed that consists of two main stages [22]. First, the weights of the network are pretrained using a greedy layer-by-layer training procedure that employs Restricted Boltzmann Machines [8]. Second, the weights of the pretrained deep Boltzmann machine are finetuned using a variant of contrastive divergence [7, 26] that employs mean-field approximations in the positive phase, thereby maximizing a variational lower bound on the log-likelihood. An overview of the learning procedure is given on the following page.

```
Learning Procedure for Deep Boltzmann Machines [22]:
Given: training set of $N$ training samples $\{\mathbf{v}_n\}_{n=1}^N$, initial learning rate $\alpha^0$

  • Train a stack of Restricted Boltzmann Machines [8] to initialize $\theta^0 = \{\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(m)}\}$, and
    randomly generate $P$ fantasy particles $\left\{\tilde{\mathbf{v}}_p^0, \tilde{\mathbf{h}}_p^{(1),0}, \ldots, \tilde{\mathbf{h}}_p^{(M),0}\right\}_{p=1}^P$.
  • For $t = 1$ to $T$ iterations:
      – For each training example $\mathbf{v}_n$, $n = 1$ to $N$:
          ∗ Set $\boldsymbol{\mu}_n^{(0)} = \mathbf{v}_n$, and randomly initialize $\boldsymbol{\mu}_n^{(1)}, \ldots, \boldsymbol{\mu}_n^{(M)}$ and perform mean-field updates:
              $\boldsymbol{\mu}_n^{(m)} \leftarrow \sigma\left((\boldsymbol{\mu}_n^{(m-1)})^T \mathbf{W}^{(m)} \boldsymbol{\mu}_n^{(m)} + (\boldsymbol{\mu}_n^{(m)})^T \mathbf{W}^{(m+1)} \boldsymbol{\mu}_n^{(m+1)}\right), 1 \le m < M,$
              $\boldsymbol{\mu}_n^{(M)} \leftarrow \sigma\left((\boldsymbol{\mu}_n^{(M-1)})^T \mathbf{W}^{(M)} \boldsymbol{\mu}_n^{(M)}\right), 1 \le m < M,$
      – For each fantasy particle $\tilde{\mathbf{v}}_p$, $p = 1$ to $P$:
          ∗ Obtain a new state $\left\{\tilde{\mathbf{v}}_p^t, \tilde{\mathbf{h}}_p^{(1),t}, \ldots, \tilde{\mathbf{h}}_p^{(M),t}\right\}$ by running a Gibbs sampler for $k$ steps,
            initializing at the previous state $\left\{\tilde{\mathbf{v}}_p^{t-1}, \tilde{\mathbf{h}}_p^{(1),t-1}, \ldots, \tilde{\mathbf{h}}_p^{(M),t-1}\right\}$.
      – Perform gradient update of parameters:
          $\mathbf{W}^{(1)} \leftarrow \mathbf{W}^{(1)} + \alpha^t \left(\frac{1}{N}\sum_{n=1}^N \mathbf{v}_n (\boldsymbol{\mu}_n^{(1)})^T - \frac{1}{P}\sum_{p=1}^P \tilde{\mathbf{v}}_p (\tilde{\mathbf{h}}_p^{(1)})^T\right),$   (3)
          $\mathbf{W}^{(m)} \leftarrow \mathbf{W}^{(m)} + \alpha^t \left(\frac{1}{N}\sum_{n=1}^N \boldsymbol{\mu}_n^{(m-1)} (\boldsymbol{\mu}_n^{(m)})^T - \frac{1}{P}\sum_{p=1}^P \tilde{\mathbf{h}}_p^{(m-1)} (\tilde{\mathbf{h}}_p^{(m)})^T\right), 1 < m \le M.$   (4)
      – Decrease learning rate $\alpha^t$.
```

## 3 Herding

To derive the herding equation for deep Boltzmann machines, we first note that he log-likelihood of a deep Boltzmann machine can be rewritten in terms of a variational approximation [13] as follows

$$\mathcal{L}(\theta) = -\log Z(\theta) + \frac{1}{N}\sum_{n=1}^N \max_{q_n} \left( \mathbb{E}\left[ -\mathbf{v}_n^T \mathbf{W}^{(1)} \mathbf{h}_n^{(1)} - \sum_{m=2}^M \mathbf{h}_n^{(m-1)\,T} \mathbf{W}^{(m)} \mathbf{h}_n^{(m)} \right]_{q_n} + \mathcal{H}(q_n) \right),$$

where the distribution $q_n(\mathbf{h}_n^{(1)}, \ldots, \mathbf{h}_n^{(M)})$ represents the variational approximation to the posterior distribution $p(\mathbf{h}_n^{(1)}, \ldots, \mathbf{h}_n^{(M)}|\mathbf{v}_n, \theta)$, and $\mathcal{H}(q_n)$ is the Shannon entropy of this approximation. This variational approximation of the log-likelihood is derived in Appendix A. Similarly, the partition function can also be written in terms of a variational approximation as follows

$$\log Z(\theta) = \max_r \left( \sum_{\mathbf{v}, \mathbf{h}^{(1)}, \ldots, \mathbf{h}^{(M)}} \left[ \mathbb{E}\left[ -\mathbf{v}^T \mathbf{W}^{(1)} \mathbf{h}^{(1)} - \sum_{m=2}^M \mathbf{h}^{(m-1)\,T} \mathbf{W}^{(m)} \mathbf{h}^{(m)} \right]_r + \mathcal{H}(r) \right] \right),$$

where the distribution $r(\mathbf{v}, \mathbf{h}^{(1)}, \ldots, \mathbf{h}^{(M)})$ is the variational approximation to the posterior distribution $p(\mathbf{v}, \mathbf{h}^{(1)}, \ldots, \mathbf{h}^{(M)}|\theta)$.

As the energy function is linear in its parameters $\theta$, we can introduce a rescaling constant $\eta$ (a "temperature") for these parameters by replacing $\mathbf{W}^{(m)}$ by $\frac{\mathbf{W}^{(m)}}{\eta}$ for $\forall m$ in Equation 3. We denote the resulting function by $\mathcal{L}_\eta(\theta)$.

We can investigate the behavior of the function $\mathcal{L}_\eta(\theta)$ in the limit as $\eta$ goes to zero. In the limit $\eta \to 0$, the entropy terms simply vanish. Without the respective entropy terms, for a given set of weights $\theta$, the distributions $\{q_n\}$ and $r$ that maximize their respective terms are delta-peaks. As a result, we can simply replace the expectations in the log-likelihood function by maximizations. The zero-temperature limit of the log-likelihood function thus amounts to

$$\mathcal{L}_0(\theta) = \lim_{\eta \to 0} \mathcal{L}_\eta(\theta) = \frac{1}{N} \sum_{n=1}^{N} \max_{(\mathbf{h}_n^{(1)},\ldots,\mathbf{h}_n^{(M)})} \left[ -\mathbf{v}_n^T \mathbf{W}^{(1)} \mathbf{h}_n^{(1)} - \sum_{m=2}^{M} \mathbf{h}_n^{(m-1)\,T} \mathbf{W}^{(m)} \mathbf{h}_n^{(m)} \right] - \quad (5)$$

$$\max_{(\mathbf{v},\mathbf{h}^{(1)},\ldots,\mathbf{h}^{(M)})} \left[ -\mathbf{v}^T \mathbf{W}^{(1)} \mathbf{h}^{(1)} - \sum_{m=2}^{M} \mathbf{h}^{(m-1)\,T} \mathbf{W}^{(m)} \mathbf{h}^{(m)} \right]. \quad (6)$$

The resulting function $\mathcal{L}_0(\theta)$ is a piecewise linear function that reaches its maximum of 0 when all weights are zero. The zero-temperature limit of the log-likelihood function is, therefore, sometimes referred to as a *tipi function* [31].

In the remainder of the paper, we use the notation $(\mathbf{h}_n^{(1)*}, \ldots, \mathbf{h}_n^{(M)*})$ and $(\mathbf{v}^*, \mathbf{h}^{(1)*}, \ldots, \mathbf{h}^{(M)*})$ to represent the values that maximize their respective terms in Equation 6. In the zero-temperature limit, there is no point in trying to maximize the log-likelihood function with respect to its parameters $\theta$, as it is known which parameter setting maximizes the function[3]. Instead, the idea of herding is to run gradient ascent on the function $\mathcal{L}_0(\theta)$ with a fixed step size. Due to the use of a finite step size, the gradient ascent never converges to the maximum of $\mathcal{L}_0(\theta)$ that is at $\theta = 0$, but instead, it pseudo-randomly explores the tipi function. The gradient ascent procedure essentially runs a dynamical system that generates a parameter setting at each iteration of the gradient ascent. It can be shown that the resulting dynamical system is *ergodic*: the average of statistics of data under the model instantations converge to the expected value of these statistics under the likelihood solution of the corresponding MRF as the number of pseudo-samples goes to infinity. The only condition that is required to hold is that the weights do not go to infinity, but this condition seems relatively easy to satisfy. For the detailed ergodicity proof, we refer to Appendix B.

In the deep models we investigate (where $M > 1$), herding amounts to iteratively performing the following three steps: (1) updating the hidden states with the visible states clamped until convergence, i.e., the positive phase, (2) freely running the network until convergence, i.e., the negative phase, and (3) updating the network weights by performing a gradient ascent step. The equations corresponding to these three steps are worked out below.

In the positive phase, a datavector $\mathbf{v}_n$ is clamped, and the hidden units are iteratively updated until convergence in order to maximize the first term of $\mathcal{L}_0(\theta)$. The update equations are found by maximizing the energy function with respect to the units, to find

$$\mathbf{h}_n^{(m)} \leftarrow \operatorname{sgn}\left( (\mathbf{h}_n^{(m-1)})^T \mathbf{W}^{(m)} + \mathbf{W}^{(m+1)} \mathbf{h}_n^{(m+1)} - \frac{1}{2} \right), \text{for } 1 \le m < M,$$
$$\mathbf{h}_n^{(M)} \leftarrow \operatorname{sgn}\left( (\mathbf{h}_n^{(M-1)})^T \mathbf{W}^{(M)} - \frac{1}{2} \right),$$

where again, we misused notation by assuming $\mathbf{h}_n^{(0)} = \mathbf{v}_n$. In order to perform the maximization of the first term of $\mathcal{L}_0(\theta)$, these updates need to be run until convergence. If the network contains more than one layer of hidden units, the iterative updates converge only to a local maximum of the energy. The initialization of the hidden units is thus relevant to the results that are obtained. Three possible strategies for initialization of the hidden units in the positive phase are: (i) a single random initialization for each data point, (ii) random initializations for each of the data points, and (iii) initialization using one persistent chain per data point. In the experiments in section 4.2.2, we compare the performance of herder using these three initialization approaches for the negative phase.

In the negative phase, the network is allowed to run freely until it converges to a local maximum of the energy function in order to generate a single pseudo-sample $(\tilde{\mathbf{v}}, \tilde{\mathbf{h}}^{(1)}, \ldots, \tilde{\mathbf{h}}^{(M)})$. In contrast to

[3]Note that this is very different from the maximum-likelihood learning setting for deep Boltzmann machines, in which $\mathcal{L}_1(\theta)$ is maximized with respect to the parameters $\theta$.

the positive phase, the visible units are not clamped in the negative phase. The update equations can be obtained similar to those for the positive phase, and are given by

$$\begin{aligned}
\tilde{\mathbf{v}} &\leftarrow \operatorname{sgn}\left(\mathbf{W}^{(1)}\tilde{\mathbf{h}}^{(1)} - \frac{1}{2}\right), \\
\tilde{\mathbf{h}}^{(m)} &\leftarrow \operatorname{sgn}\left((\tilde{\mathbf{h}}^{(m-1)})^T\mathbf{W}^{(m)} + \mathbf{W}^{(m+1)}\tilde{\mathbf{h}}^{(m+1)} - \frac{1}{2}\right), \text{for } 1 \leq m < M, \\
\tilde{\mathbf{h}}^{(M)} &\leftarrow \operatorname{sgn}\left((\tilde{\mathbf{h}}^{(M-1)})^T\mathbf{W}^{(M)} - \frac{1}{2}\right),
\end{aligned}$$

where again, we misused notation by assuming $\tilde{\mathbf{h}}^{(0)} = \tilde{\mathbf{v}}$. The update equations for the negative phase are run until convergence to perform the desired energy maximization. Again, since different initializations of the network cause the updates to converge to different local energy maxima, the initialization of the network may affect the results. For the initialization of the network in the negative phase, one may use (1) random initialization or (2) initialization using a persistent chain. In our experiments in section 4.2.2, we investigate the performance of these two initialization approaches for the negative phase.

In the weight update phase, the weights are updated by performing a gradient update of the parameters using the sufficient statistics gathered in the positive and the negative phase. The weight updates are similar to the updates used in maximum likelihood learning in MRFs, and are given by

$$\begin{aligned}
\mathbf{W}^{(1)} &\leftarrow \mathbf{W}^{(1)} + \alpha\left(\frac{1}{N}\sum_{n=1}^{N}\mathbf{v}_n(\boldsymbol{h}_n^{(1)})^T - \tilde{\mathbf{v}}(\tilde{\mathbf{h}}^{(1)})^T\right), \\
\mathbf{W}^{(m)} &\leftarrow \mathbf{W}^{(m)} + \alpha\left(\frac{1}{N}\sum_{n=1}^{N}\mathbf{h}_n^{(m-1)}(\boldsymbol{h}_n^{(m)})^T - \tilde{\mathbf{h}}^{(m-1)}(\tilde{\mathbf{h}}^{(m)})^T\right), \text{for } 1 < m \leq M.
\end{aligned}$$

Note that the weight updates have three main differences compared to the updates for maximum likelihood learning in deep Boltzmann machine we discussed in section 2: (1) the step size is not adjusted over time, (2) in the positive part of the gradient, the states of the hidden units are not obtained using a mean-field approximation but using an energy maximization, and (3) the sufficient statistics in the negative part of the gradient are not obtained by averaging over phantasy particles, but by computing sufficient statistics based on a single pseudo-sample.

A potential problem of deep herding is that, although the moment constraints are satisfied in herding, there is no guarantee that herding employs the available hidden units in order to construct a good model. Herding may simply ignore the presence of the hidden units, leading to a complete decoupling of the hidden and the visible units (thus making the hidden units superfluous). When performing maximum likelihood learning in Markov Random Fields, this problem does not occur because maximum likelihood learning tries to maximize the entropy in the hidden units. Experimental results with herding in models with a single layer of hidden units did not reveal such decoupling behavior [30]. We return to the decoupling topic in our discussion in Section 5.

A second potential problem is that the update equations presented above only converge to a local energy maximum, whereas the ergodicity proof assumes convergence to the global energy maximum. This problem led us to investigate various initialization approaches in Section 4.2.2, the aim of which is to find local energy maxima that are as good as possible.

## 4 Experiments

In this section, we present the results of experiments with deep herding on the USPS handwritten digits dataset. The setup of these experiments is described in 4.1. The results of the experiments are presented in 4.2.

## 4.1 Experimental setup

We performed experiments on the USPS handwritten digits dataset. The dataset contains $1,100$ examples of each of the $10$ digits, leading to a dataset with $11,000$ data points. The digit images have a size of $16 \times 16$ pixels, and were binarized using a threshold of $0.3$. In all experiments on the USPS handwritten digits dataset, we randomly split up the data into a training set that contains $66\%$ of the data, a test set that contains $24\%$ of the data, and a validation set that contains $10\%$ of the data. For each class in the data, we run a herder on all instances in the training set that belong to that class for $2,000$ iterations. During the first $1,000$ iterations, no energies are computed, whereas during the second $1,000$ iterations, we compute the energies of the data that is used to run the herder, as well as of the test and validation data. We denote the energy of the training data with class label $c$, the test data, and the validation data at herding iteration $t$ by $E^{(t)}_{train(c)}$, $E^{(t)}_{test}$, and $E^{(t)}_{valid}$, respectively. Following Welling [30], we normalize the test and validation energies at each iteration by computing the corresponding $Z$-scores, and we perform online averaging of the test and validation energies over the $1,000$ iterations at which the energies are recorded (recall that statistics computed during herding are only meaningful is they are averaged over herding iterations). An overview of the procedure that computes an average of the normalized energies (for the validation and the test set) over herding iterations is on the next page.

**Herding Procedure for Deep Boltzmann Machine:**
**Given:** training set of $N$ training examples $\{\mathbf{v}_n\}_{n=1}^{N}$, step size $\alpha$, validation and test set.

```
• Initialize parameters $\left\{\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(m)}\right\}$ and generate a fantasy particle $(\tilde{\mathbf{v}}^0, \tilde{\mathbf{h}}^{(1),0}, \ldots, \tilde{\mathbf{h}}^{(M),0})$.
• For $c = 1$ to $C$ classes:
    – For $t = 1$ to $T$ iterations:
        ∗ For each training example $\mathbf{v}_n$ of class $c$, $n = 1$ to $N$:
            · Initialize $\mathbf{h}_n$ and run dynamical system until convergence:
              $\mathbf{h}_n^{(m)} \leftarrow \operatorname{sgn}\left(\mathbf{h}_n^{(m-1)T}\mathbf{W}^{(m)}\mathbf{h}_n^{(m)} + \mathbf{h}_n^{(m)T}\mathbf{W}^{(m+1)}\mathbf{h}_n^{(m+1)} - \frac{1}{2}\right)$, for $1 \leq m < M$,
              $\mathbf{h}^{(M)} \leftarrow \operatorname{sgn}\left((\mathbf{h}^{(M-1)})^T\mathbf{W}^{(M)} - \frac{1}{2}\right)$.
        ∗ Obtain a new state $\left\{\tilde{\mathbf{v}}^t, \tilde{\mathbf{h}}^{(1),t}, \ldots, \tilde{\mathbf{h}}^{(M),t}\right\}$ by running the dynamical system until convergence, initializing at the previous state $\left\{\tilde{\mathbf{v}}^{t-1}, \tilde{\mathbf{h}}^{(1),t-1}, \ldots, \tilde{\mathbf{h}}^{(M),t-1}\right\}$:
              $\tilde{\mathbf{v}} \leftarrow \operatorname{sgn}\left(\mathbf{W}^{(1)}\tilde{\mathbf{h}}^{(1)} - \frac{1}{2}\right)$,
              $\tilde{\mathbf{h}}^{(m)} \leftarrow \operatorname{sgn}\left((\tilde{\mathbf{h}}^{(m-1)})^T\mathbf{W}^{(m)} + \mathbf{W}^{(m+1)}\tilde{\mathbf{h}}^{(m+1)} - \frac{1}{2}\right)$, for $1 \leq m < M$,
              $\tilde{\mathbf{h}}^{(M)} \leftarrow \operatorname{sgn}\left((\tilde{\mathbf{h}}^{(M-1)})^T\mathbf{W}^{(M)} - \frac{1}{2}\right)$.
        ∗ Perform update of parameters using fixed step size:
              $\mathbf{W}^{(1)} \leftarrow \mathbf{W}^{(1)} + \alpha\left(\frac{1}{N}\sum_{n=1}^{N}\mathbf{v}_n(\mathbf{h}_n^{(1)})^T - \tilde{\mathbf{v}}^{\mathbf{t}}(\tilde{\mathbf{h}}^{(1),t})^T\right)$,
              $\mathbf{W}^{(m)} \leftarrow \mathbf{W}^{(m)} + \alpha\left(\frac{1}{N}\sum_{n=1}^{N}\mathbf{h}_n^{(m-1)}(\mathbf{h}_n^{(m)})^T - \tilde{\mathbf{h}}^{(m-1),t}(\tilde{\mathbf{h}}^{(m),t})^T\right)$, for $1 < m \leq M$.
        ∗ For each training example $\mathbf{v}_n$ of class $c$, for $n = 1$ to $N_{train}$:
            · Initialize $(\mathbf{h}_n^{(1)}, \ldots, \mathbf{h}_n^{(M)})$, and run dynamical system until convergence.
            · Compute energy of training example $\mathbf{v}_n$ under the converged solution:
              $E_{train(c)}^{(t)}(\mathbf{v}_n, \mathbf{h}_n^{(1)}, \ldots, \mathbf{h}_n^{(M)}) = \mathbf{v}_n\mathbf{W}^{(1)}\mathbf{h}_n^{(1)} + \sum_{m=2}^{M}\mathbf{h}_n^{(m-1)}\mathbf{W}^{(m)}\mathbf{h}_n^{(m)}$.
        ∗ Similarly, compute $E_{valid}^{(t)}$ for each validation example and $E_{test}^{(t)}$ for each test example.
        ∗ Normalize energies of test and validation examples $\tilde{E}_{test}^{(t)}$ and $\tilde{E}_{valid}^{(t)}$ (compute $Z$-score):
              $\hat{E}_{valid}^{(t)}(\mathbf{v}_n, \mathbf{h}_n^{(1)}, \ldots, \mathbf{h}_n^{(M)}) = \frac{E_{valid}^{(t)}(\mathbf{v}_n) - \mu_{E_{train(c)}^{(t)}(\mathbf{v}_n, \mathbf{h}_n^{(1)}, \ldots, \mathbf{h}_n^{(M)})}}{\sigma_{E_{train(c)}^{(t)}(\mathbf{v}_n, \mathbf{h}_n^{(1)}, \ldots, \mathbf{h}_n^{(M)})}}$,
              $\hat{E}_{test}^{(t)}(\mathbf{v}_n, \mathbf{h}_n^{(1)}, \ldots, \mathbf{h}_n^{(M)}) = \frac{E_{test}^{(t)}(\mathbf{v}_n) - \mu_{E_{train(c)}^{(t)}(\mathbf{v}_n, \mathbf{h}_n^{(1)}, \ldots, \mathbf{h}_n^{(M)})}}{\sigma_{E_{train(c)}^{(t)}(\mathbf{v}_n, \mathbf{h}_n^{(1)}, \ldots, \mathbf{h}_n^{(M)})}}$.
        ∗ Perform online averaging of $\tilde{E}_{test}^{(t)}$ and $\tilde{E}_{valid}^{(t)}$.
```

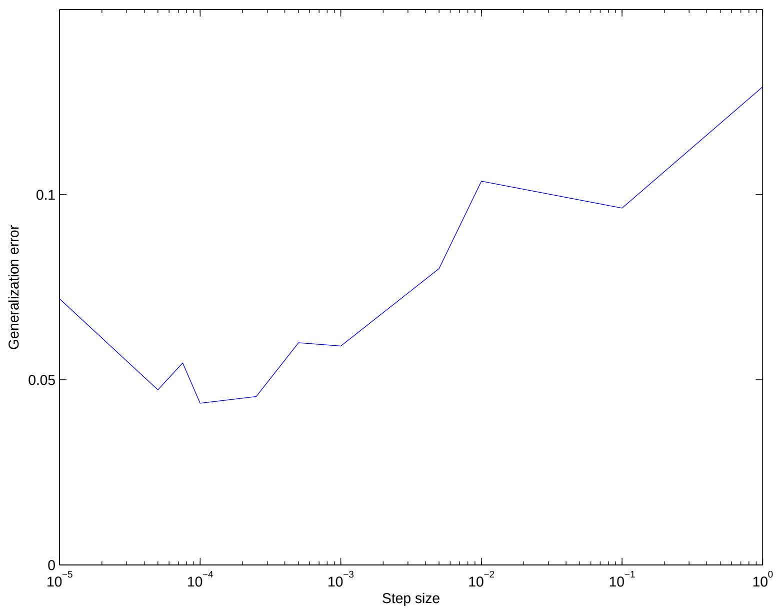


Figure 2: Generalization error of logistic classifiers trained on normalized herding energies for different step sizes (using a $256 - 100 - 50$ network).

The above procedure produces $C$ average normalized energies for each of the data points in the validation set and test set. These average normalized energies can thus be viewed as $C$-dimensional features, which can be used to perform clustering, construct low-dimensional data representation, and to train classifiers on. We opt to train various classifiers on the average normalized energies of the validation test set, and use the trained classifiers to classify the data points in the test set, in order to assess the performance of the herder. In particular, we use logistic regression classifier, linear discriminant classifiers, and 7-nearest neighbor classifiers. Also, we perform classification based on energy maxima in the average normalized energies.

Following [30], we use a $[-1, 1]$-representation for all visible and hidden units in the deep Boltzmann machine in all stages of the process (i.e., in the herding and in the energy computations). Such a representation is also used in, for instance, Ising models [11].

## 4.2 Results

This subsection presents the results of our experiments on the USPS dataset. The subsection consists of three main parts. First, we investigate the effect of varying the step size $\alpha$ of the herder. Second, we investigate the effect of using different initialization strategies in the positive phase of the herder. Third, we investigate the performance of herding in a variety of network architectures (all of which have to layers of hidden features).

### 4.2.1 Investigating step sizes

In Figure 2, we present a graph that shows generalization errors of logistic regression classifiers that were trained and test on average normalized energies obtained by performing herding in a $256-100-50$ network. The generalization errors were determined for a variety of values of the step size $\alpha$ of the herder. In these experiments, we use a single random initialization in the positive phase and a persistent chain to initialize the network in the negative phase. The reader should note that the values for $\alpha$ are on a logarithmic scale.

The results reveal that, for deep networks, the selection of the step size of the herder may have a significant influence on the quality of the extracted features. This is an interesting observation, as for networks with a single hidden layer one can prove that the selection of the value of $\alpha$ does not influence the results, because of a scale-invariance property [30]. In our experiments with $256-100-50$ networks, the lowest classification error of $3.85\%$ was obtained using a step size of $\alpha = 10^{-4}$. We also performed similar experiments with two-layer networks with different architectures. The results of these experiments are not presented here, but they showed exactly the same pattern: a step size of approximately $10^{-4}$ appears optimal for all networks with two hidden layers[4].

The results suggest that, when herding is performed in networks with more than one layer of hidden units, having too much entropy in, specifically, the top hidden layer leads to inferior results. The entropy in the top hidden layer can be regularized by decreasing the step size. However, a step size that is too small leads

### 4.2.2 Investigating initialization strategies

In Table 1 and 2, we present the results of experiments in which we measured classification errors of logistic regression classifiers, linear discriminant classifiers, and 7-nearest neighbor classifiers trained on normalized energies obtained by performing herding in a $256 - 100 - 50$ network, using three different initialization strategies in the positive phase and two different initialization strategies in the negative phase. We also present the generalization error of a classifier that classifies a test instance by determining for which herder (note we train one herder per class) the average normalized energy is maximal. In particular, we investigate three strategies to initialize the network in the positive phase: (1) using a single random initialization for all data points, (2) using a different random initialization for each data point, and (3) using a persistent chain to obtain the initialization. We investigate two strategies to initialize the network for the negative phase: (1) using random initialization and (2) using a persistent chain to obtain the initialization. In all experiments, we used a step size $\alpha$ of $10^{-4}$.

| *Positive initialization* | *Logistic regression* | *LDC* | *7-Nearest neighbor* | *Maximum energy* |
|---|---|---|---|---|
| Single random | $3.91\%$ | $7.09\%$ | $6.91\%$ | $12.45\%$ |
| Multiple random | $5.00\%$ | $8.18\%$ | $6.73\%$ | $14.18\%$ |
| Persistent | $\mathbf{3.18}\%$ | $\mathbf{6.09}\%$ | $\mathbf{5.09}\%$ | $\mathbf{12.09}\%$ |

Table 1: Generalization errors of four classifiers trained on normalized energies obtained by herding in networks using various initializations for the positive phase. In the negative phase, we used initialization using a persistent chain. The best performance is typeset in boldface.

| *Positive initialization* | *Logistic regression* | *LDC* | *7-Nearest neighbor* | *Maximum energy* |
|---|---|---|---|---|
| Single random | $11.36\%$ | $12.55\%$ | $19.36\%$ | $33.18\%$ |
| Multiple random | $\mathbf{10.00}\%$ | $\mathbf{11.36}\%$ | $16.73\%$ | $36.73\%$ |
| Persistent | $10.09\%$ | $11.45\%$ | $\mathbf{15.73}\%$ | $\mathbf{21.27}\%$ |

Table 2: Generalization errors of four classifiers trained on normalized energies obtained by herding in networks using various initializations for the positive phase. In the negative phase, we used random initialization. The best performance is typeset in boldface.

From the results in Table 1 and 2, we observe that using a persistent chain in the negative phase has significant advantages compared to using random initialization. This result is in line with observations on maximum likelihood learning of deep Boltzmann machines [22]. From the results in the tables, we also observe that it appears to be beneficial to use a persistent chain to initialize the network in the positive phase. We surmise the use of a persistent chain in the positive phase is beneficial because it finds better local energy maxima on average: as a result of the small weight updates in the positive phase, the persistent chain initializes the network relatively close to the local energy maximum of the previous herding iteration. Because in the previous herding iteration, we increased the probability of

[4]Preliminary experiments with networks with three hidden layers suggested that even lower step sizes are required for such networks.

that local energy maximum, we are likely to end up in a better local energy maximum in the present iteration than a random initialization would. For similar reasons, the use of a positive persistent chain may be helpful when performing maximum likelihood learning of deep Boltzmann machines as well.

### 4.2.3 Investigating network structures

In Table 3, we present the results of experiments in which we measured classification errors of logistic regression classifiers, linear discriminant classifiers, and 7-nearest neighbor classifiers trained on average normalized energies obtained by running the herding procedure in two-layer networks with a variety of architectures. We also present the generalization error of a classifier that classifies a test instance by determining for which herder (note we train one herder per class) the average normalized energy is maximal. In all experiments, we use a single random initialization in the positive phase and a persistent chain to initialize the negative phase. The step size $\alpha$ was fixed to $10^{-4}$.

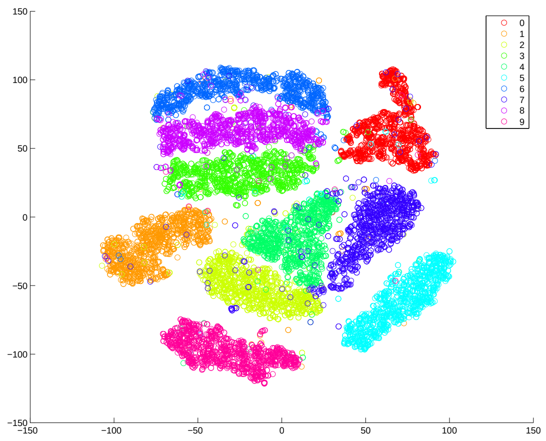
| *Architecture* | *Logistic regression* | *LDC* | *7-Nearest neighbor* | *Maximum energy* |
|---|---|---|---|---|
| $256-50-30$ | $4.55\%$ | $8.36\%$ | $6.36\%$ | $14.18\%$ |
| $256-50-50$ | $7.73\%$ | $13.00\%$ | $9.73\%$ | $17.91\%$ |
| $256-75-30$ | $4.64\%$ | $7.36\%$ | $6.82\%$ | $12.64\%$ |
| $256-75-50$ | $4.36\%$ | $7.64\%$ | $5.55\%$ | $12.45\%$ |
| $256-75-75$ | $4.73\%$ | $8.64\%$ | $6.91\%$ | $13.55\%$ |
| $256-100-30$ | $3.45\%$ | $\mathbf{5.73}\%$ | $\mathbf{4.55}\%$ | $12.82\%$ |
| $256-100-50$ | $\mathbf{3.18}\%$ | $6.09\%$ | $5.09\%$ | $12.09\%$ |
| $256-100-75$ | $5.36\%$ | $9.72\%$ | $8.82\%$ | $14.27\%$ |
| $256-100-100$ | $5.09\%$ | $10.27\%$ | $9.27\%$ | $16.18\%$ |
| $256-125-30$ | $3.82\%$ | $5.91\%$ | $5.73\%$ | $\mathbf{10.64}\%$ |
| $256-125-50$ | $4.91\%$ | $7.64\%$ | $6.45\%$ | $12.82\%$ |
| $256-125-75$ | $4.64\%$ | $7.18\%$ | $6.18\%$ | $11.91\%$ |
| $256-150-100$ | $5.55\%$ | $10.91\%$ | $9.00\%$ | $11.82\%$ |

Table 3: Generalization errors of three classifiers trained on normalized energies obtained by herding in networks with various architectures. The best performance is typeset in boldface.

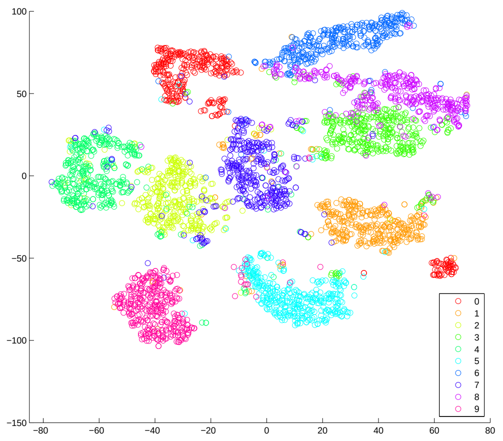
The most prominent observation we can make from the presented results is that adding more units to the hidden layer(s) does not necessarily improve the performance of the herder. These results are not in line with the results presented for networks with a single hidden layer [30], where more hidden units appeared to improve the performance of the classifiers. In particular, the presented results suggest that it is beneficial to use network architectures in which each hidden layer has significantly less hidden units than the layer below it. In particular the best-performing network architectures appear to be the $256-100-30$, $256-100-50$, and $256-125-30$ networks. The results of our experiments with various network architectures are in large contrast with results that were presented for deep architectures trained using contrastive divergence or one of its variants [8, 22, 26, 27]. In such architectures, it is generally beneficial to have one or more large layers of hidden units that construct an overcomplete representation of the data. As in Section 4.2.2, the results suggest that the top hidden layer is primarily contributing noise to the network energy, i.e., that it is running completely decoupled from the data. We return to this issue in our discussion in Section 5.

On the USPS dataset, the lowest generalization error of $3.18\%$ was achieved by performing herding in a $256-100-50$ network, and classifying the test data points using a logistic regression classifier. The performance of $3.18\%$ is almost as good as the results presented for herding in networks with a single hidden layer [30]. However, in none of our experiments, we were able to outperform herding in a single-layer network with, say, 500 or 1000 hidden units.

In Figure 3, we show a visualization of the USPS dataset constructed by performing dimensionality reduction using t-SNE [29] on the average normalized energies of the validation and test data. The normalized energies were obtained by performing herding in the $256-100-50$ network (note that the labels for the validation and test energy need not be known in order to construct the visualization; the labels are only used to color the points in the map). We also show a visualization of the same data that was produced by running t-SNE on the same data, but now, using the pixel values as high-dimensional input[5]. The results show that the proposed herding procedure can exploit the available label information to improve the separation between classes, for instance, between the classes 7 and 9. Also, the number of 'outliers' in the map of the herding energies appears to be lower than the number of outliers in the map of the original pixel data.



(a) Map of the original pixel data.



(b) Map of the normalized energies produced by herding.

Figure 3: Visualizations of the USPS validation and test data constructed by t-SNE [29]. The maps reveal that herding can exploit label information to obtain better separation between classes.

## 5 Discussion

From the results presented in the previous section, we make two main observations.

First, we observe that the results of our experiments with a range of step sizes $\alpha$ reveal that herding in deep networks is not scale-invariant. This contradicts with the scale-invariance property that (provably) holds for herding in fully observed MRFs and in MRFs with a single layer of hidden units.

Second, we observe that herding in deep networks appears to require the use of completely different network architectures than those that are typically used when training deep Boltzmann machines or other types of deep neural networks [9, 15, 22]. Whereas maximum likelihood learning in deep networks typically perform well when at least some of the hidden layers have lots of units (thereby constructing an overcomplete data representation), herding appears to work best in networks in which higher-level hidden layers have significantly less units than the layers below them.

These two observations suggest that the top units in our deep networks are effectively not contributing to the herding process: they appear to run essentially decoupled from the data. As a result, the input of the top hidden units into the middle will be primarily noise, which hampers the performance of the herding procedure. Lowering the step size or reducing the number of top hidden units essentially reduces the amount of noise, which leads to a better generalization performance[6], explaining the two experimental observations above. In order for herding in deep networks to outperform herding in single-layer networks, the development of an approach that prevents this decoupling is required. One could add connections between the visible units and the top layer hidden units, but this essentially leads to a single-layer network with more hidden units and some lateral connections between the hidden units. Another approach may be to treat the deep network as a stack of single-layer networks, as is often done when training deep networks using maximum likelihood (see Section 2), but it is unclear how such an

[5]Note that the plots are not meant as a qualitative comparison. In the construction of the right plot, label information was exploited by the herder, whereas the left plot was constructed in a fully unsupervised manner.

[6]We note that inserting a small amount of noise may actually be beneficial, as the noise may serve as a sort of regularizer. However, such a 'regularizer' most likely only works if it consists of a few top hidden units.

approach should be implemented in practice. We leave the development of approaches that prevent decoupling during herding in deep networks for future work.

An additional problem with herding in deep networks is that is not possible to identify global energy maxima in the positive and the negative phase in polynomial time. A similar problem occurs when training deep Boltzmann machines using maximum likelihood (see Section 2), which prompts the use of mean-field approximations [22]. In herding, it appears to be possible to address the problem by using sensible initialization procedures. Also, errors that are due to getting stuck in poor local energy maxima are likely to average out over herding iterations. Certainly, the local energy maxima problem is less severe than the decoupling problems discussed above.

## 6 Conclusions

In this paper, we investigated the application of herding to deep Boltzmann machines. We investigated the effect of varying step size, initialization strategy, and network architecture on the performance of deep herders. Hitherto, we were not able to establish a performance improvement of deep herders over herding in a Markov Random Field with a single layer of hidden units. Most likely, our experimental results can be explained from a decoupling phenomenon that occurs in the deeper layers of our networks: these layers run freely while completely ignoring the input data.

A large number of directions for future work into herding remain. First and foremost, approaches that may prevent decoupling should be investigated. We made some suggestions in Section 5. From a mathematical perspective, it is interesting to investigate the characteristics of the dynamical system in herding, for instance, what is the attractor set of a herder. From a learning perspective, many alternative network architectures may be investigated, e.g., fully connected architectures [21], architectures with some lateral connections [19], architectures in which some variables are conditioned upon [14], architectures in which the joint probability of class labels and features is modeled [8], and architectures that have non-binary observed variables [32].

## 7 Acknowledgements

Laurens van der Maaten acknowledges support by Rubicon grant no. 680.50.0908 from the Netherlands Organisation for Scientific Research (NWO), and by the EU-FP7 Network of Excellence on Social Signal Processing (SSPNet). The author thanks Max Welling for many helpful discussions and for his comments on early drafts of this report. The author also thanks Geoffrey Hinton for helpful discussions.

## A Derivation of the variational approximation

Recall that the log-likelihood of a data point $\mathbf{v}_n$ under a deep Boltzmann machine is given by

$$\mathcal{L}^{(n)}(\theta) = \log \sum_{\mathbf{h}_n^{(1)},\ldots,\mathbf{h}_n^{(M)}} \exp\left( -\mathbf{v}_n^T \mathbf{W}^{(1)} \mathbf{h}_n^{(1)} - \sum_{m=2}^{M} \mathbf{h}_n^{(m-1)\,T} \mathbf{W}^{(m)} \mathbf{h}_n^{(m)} \right) + \text{const.}$$

We can rewrite the log-likelihood function in terms of the free energy as follows

$$\begin{aligned}
\mathcal{L}^{(n)}(\theta) &= \log \sum_{\mathbf{h}_n^{(1)},\ldots,\mathbf{h}_n^{(M)}} p(\mathbf{v}_n, \mathbf{h}_n^{(1)}, \ldots, \mathbf{h}_n^{(M)} | \theta) \\
&= \log \sum_{\mathbf{h}_n^{(1)},\ldots,\mathbf{h}_n^{(M)}} q_n(\mathbf{h}_n^{(1)}, \ldots, \mathbf{h}_n^{(M)}) \frac{p(\mathbf{v}_n, \mathbf{h}_n^{(1)}, \ldots, \mathbf{h}_n^{(M)} | \theta)}{q_n(\mathbf{h}_n^{(1)}, \ldots, \mathbf{h}_n^{(M)})} \\
&\geq \sum_{\mathbf{h}_n^{(1)},\ldots,\mathbf{h}_n^{(M)}} q_n(\mathbf{h}_n^{(1)}, \ldots, \mathbf{h}_n^{(M)}) \log \frac{p(\mathbf{v}_n, \mathbf{h}_n^{(1)}, \ldots, \mathbf{h}_n^{(M)} | \theta)}{q_n(\mathbf{h}_n^{(1)}, \ldots, \mathbf{h}_n^{(M)})},
\end{aligned}$$

where we introduced the variational distribution $q_n$, and where the inequality is due to Jensen's inequality [12]. The right-hand side of the inequality is typically referred to as the free energy $F^{(n)}(q_n;\theta)$, and it is a lower bound on the log-likelihood. The free energy breaks up into two terms

$$\begin{aligned}
F^{(n)}(q_n;\theta) &= \sum_{\mathbf{h}_n^{(1)},\ldots,\mathbf{h}_n^{(M)}} q_n(\mathbf{h}_n^{(1)},\ldots,\mathbf{h}_n^{(M)}) \log \frac{p(\mathbf{v}_n,\mathbf{h}_n^{(1)},\ldots,\mathbf{h}_n^{(M)}|\theta)}{q_n(\mathbf{h}_n^{(1)},\ldots,\mathbf{h}_n^{(M)})} \\
&= \sum_{\mathbf{h}_n^{(1)},\ldots,\mathbf{h}_n^{(M)}} q_n(\mathbf{h}_n^{(1)},\ldots,\mathbf{h}_n^{(M)}) \log p(\mathbf{v}_n,\mathbf{h}_n^{(1)},\ldots,\mathbf{h}_n^{(M)}|\theta) \\
&\qquad - \sum_{\mathbf{h}_n^{(1)},\ldots,\mathbf{h}_n^{(M)}} q_n(\mathbf{h}_n^{(1)},\ldots,\mathbf{h}_n^{(M)}) \log q_n(\mathbf{h}_n^{(1)},\ldots,\mathbf{h}_n^{(M)}) \\
&= \mathbb{E}\left[\log p(\mathbf{v}_n,\mathbf{h}_n^{(1)},\ldots,\mathbf{h}_n^{(M)}|\theta)\right]_{q_n} - \mathcal{H}(q_n).
\end{aligned}$$

The free energy is equal to the log-likelihood iff $q_n(\mathbf{h}_n^{(1)},\ldots,\mathbf{h}_n^{(M)})$ equals $p(\mathbf{h}_n^{(1)},\ldots,\mathbf{h}_n^{(M)}|\mathbf{v}_n,\theta)$. In other words, if we maximize the free energy over $q_n(\mathbf{h}_n^{(1)},\ldots,\mathbf{h}_n^{(M)}|\theta)$, we obtain the equality

$$\mathcal{L}^{(n)}(\theta) = \max_{q_n}\left(\mathbb{E}\left[\log p(\mathbf{v}_n,\mathbf{h}_n^{(1)},\ldots,\mathbf{h}_n^{(M)}|\theta)\right]_{q_n} - \mathcal{H}(q_n)\right).$$

For deep Boltzmann machines, this expression amounts to

$$\mathcal{L}^{(n)}(\theta) = \max_{q_n}\left(\mathbb{E}\left[-\mathbf{v}_n^T\mathbf{W}^{(1)}\mathbf{h}_n^{(1)} - \sum_{m=2}^{M}\mathbf{h}_n^{(m-1)\,T}\mathbf{W}^{(m)}\mathbf{h}_n^{(m)}\right]_{q_n} - \mathcal{H}(q_n)\right),$$

which completes the result.

## B Proof of ergodicity

The proof of ergodicity shows that the average sufficient statistics of the pseudo-samples obtained from a herder converge to the expected value of these statistics under the maximum likelihood solution of the corresponding MRF (as the number of pseudo-samples goes to infinity). The only condition that is required to hold is that the weights do not go to infinity. Mathematically, ergodicity can be expressed as follows:

**Proposition:** If $\forall i,j,m \quad \lim_{T\to 0}\frac{1}{T}W_{ij}^{(m)} = 0$, then

$$\lim_{T\to\infty}\frac{1}{T}\sum_{t=1}^{T}\tilde{h}_i^{(m-1),t}W_{ij}^{(m),t}\tilde{h}_j^{(m),t} \to \lim_{T\to\infty}\frac{1}{T}\sum_{t=1}^{T}\frac{1}{N}\sum_{n=1}^{N}h_{n,i}^{(m-1),t}W_{ij}^{(m),t}h_{n,j}^{(m),t},$$

where we misused notation by assuming $v_i = h_i^0$ and $\tilde{v}_i = \tilde{h}_i^0$.

**Proof:** First, note that at each iteration $t$, the change in the weights can be written as

$$\Delta W_{ij}^{(m),t} = \left(\frac{1}{N}\sum_{n=1}^{N}h_{n,i}^{(m-1),t}W_{ij}^{(m),t}h_{n,j}^{(m),t}\right) - \left(\tilde{h}_i^{(m-1),t}W_{ij}^{(m),t}\tilde{h}_j^{(m),t}\right). \quad (7)$$

Second, not that by definition, $\Delta W_{ij}^{(m),t} = W_{ij}^{(m),t} - W_{ij}^{(m),t-1}$. We can average both definitions of $\Delta W_{ij}^{(m),t}$ over time $t$ to obtain the expression

$$\begin{aligned}
\frac{1}{T}\sum_{t=1}^{T}\Delta W_{ij}^{(m),t} &= \frac{1}{T}\left(W_{ij}^{(m),t} - W_{ij}^{(m),0}\right) \\
&= \frac{1}{T}\sum_{t=1}^{T}\left(\frac{1}{N}\sum_{n=1}^{N}h_{n,i}^{(m-1),t}W_{ij}^{(m),t}h_{n,j}^{(m),t}\right) - \frac{1}{T}\sum_{t=1}^{T}\left(\tilde{h}_i^{(m-1),t}W_{ij}^{(m),t}\tilde{h}_j^{(m),t}\right).
\end{aligned}$$

In the limit $T \to \infty$, assuming that the weights are finite in all iterations, the term $\frac{1}{T}\left(W_{ij}^{(m),t} - W_{ij}^{(m),0}\right)$ goes to 0. Hence, in the limit $T \to \infty$, the term in Equation 7 thus has to go to 0 as well, which proves the result.

## References

[1] A. Ahmed, K. Yu, W. Xu, Y. Gong, and E.P. Xing. Training hierarchical feed-forward visual recognition models using transfer learning from pseudo tasks. In *Proceedings of the* 10th *European Conference on Computer Vision*, pages 69–82, 2008.

[2] Y. Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009.

[3] Y. Bengio and Y. LeCun. Scaling learning algorithms towards AI. In L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, editors, *Large-Scale Kernel Machines*, pages 321–360. The MIT Press, 2007.

[4] P. Clifford. Markov Random Fields in statistics. In G.R. Grimmett and D.J.A. Welsh, editors, *Disorder in Physical Systems. A Volume in Honour of John M. Hammersley*, pages 19–32. Oxford Press, 1990.

[5] R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the* 25th *International Conference on Machine Learning*, pages 160–167, 2008.

[6] R. Hadsell, A. Erkan, P. Sermanet, M. Scoffier, U. Muller, and Y. LeCun. Deep belief net learning in a long-range vision system for autonomous off-road driving. In *Proceedings of Intelligent Robots and Systems*, pages 628–633, 2008.

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

[8] G.E. Hinton, S. Osindero, and Y. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

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

[10] G.E. Hinton and T. Sejnowski. Optimal perceptual inference. In *IEEE Conference on Computer Vision and Pattern Recognition*, 1983.

[11] E. Ising. Beitrag zur Theorie des Ferromagnetismus. *Zeitschrift für Physik*, 31:253–258, 1925.

[12] J.L.W.V. Jensen. Sur les fonctions convexes et les inégalités entre les valeurs moyennes. *Acta Mathematica*, 30(1):175–193, 1906.

[13] M.I. Jordan, Z. Ghahramani, T.S. Jaakkola, and L.K. Saul. An introduction to variational methods for graphical models. In M.I. Jordan, editor, *Learning in Graphical Models*, pages 105–162, 1999.

[14] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the* 18th *International Conference on Machine Learning*, pages 282–289, 2001.

[15] H. Larochelle, Y. Bengio, J. Louradour, and P. Lamblin. Exploring strategies for training deep neural networks. *Journal of Machine Learning Research*, 10(Jan):1–40, 2009.

[16] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In *Proceedings of the* 24th *International Conference on Machine Learning*, pages 473–480, 2007.

[17] I. Levner. *Data Driven Object Segmentation.* PhD thesis, Department of Computer Science, University of Alberta, 2008.

[18] I. Murray, Z. Ghahramani, and D. MacKay. MCMC for doubly-intractable distributions. In *Proceedings of the $22^{nd}$ Annual Conference on Uncertainty in Artificial Intelligence*, pages 359–366. AUAI Press, 2006.

[19] S. Osindero and G.E. Hinton. Modeling image patches with a directed hierarchy of Markov Random Fields. In *Advances In Neural Information Processing Systems*, volume 20, pages 1121–1128, 2008.

[20] M.A. Ranzato and M. Szummer. Semi-supervised learning of compact document representations with deep networks. In *Proceedings of the $25^{th}$ International Conference on Machine Learning*, pages 792–799, 2008.

[21] R. Salakhutdinov. Learning and evaluating Boltzmann Machines. Technical Report UTML TR 2008-002, Department of Computer Science, University of Toronto, 2008.

[22] R. Salakhutdinov and G.E. Hinton. Deep Boltzmann Machines. In *Proceedings of the $12^{th}$ International Conference on Artificial Intelligence and Statistics*, 2009.

[23] J. Shawe-Taylor and N. Christianini. *Kernel Methods for Pattern Analysis.* Cambridge University Press, Cambridge, UK, 2004.

[24] P. Smolensky. Information processing in dynamical systems: Foundations of harmony theory. In D.E. Rumelhart and J.L. McClelland, editors, *Parallel Distributed Processing: Volume 1: Foundations*, pages 194–281, Cambridge, MA, 1986. The MIT Press.

[25] I. Sutskever and G.E. Hinton. Deep narrow sigmoid belief networks are universal approximators. *Neural Computation*, 20(11):2629–2636, 2008.

[26] T. Tieleman. Training Restricted Boltzmann Machines using approximations to the likelihood gradient. In *Proceedings of the International Conference on Machine Learning*, volume 25, pages 1064–1071, 2008.

[27] T. Tieleman and G.E. Hinton. Using fast weights to improve persistent contrastive divergence. In *Proceedings of the $26^{th}$ International Conference on Machine Learning*, pages 1033–1040, New York, NY, 2009. ACM.

[28] L.J.P. van der Maaten. Learning a parametric embedding by preserving local structure. In *Proceedings of the $12^{th}$ International Conference on Artificial Intelligence and Statistics, JMLR W & CP*, volume 5, pages 384–391, 2009.

[29] L.J.P. van der Maaten and G.E. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(Nov):2431–2456, 2008.

[30] M. Welling. Herding dynamic weights for partially observed random field models. In *Proceedings of the $25^{th}$ Annual Conference on Uncertainty in Artificial Intelligence*, 2009.

[31] M. Welling. Herding dynamic weights to learn. In Léon Bottou and Michael Littman, editors, *Proceedings of the $26^{th}$ International Conference on Machine Learning*, pages 1121–1128. Omnipress, 2009.

[32] M. Welling, M. Rosen-Zvi, and G. Hinton. Exponential family harmoniums with an application to information retrieval. In *Advances in Neural Information Processing Systems*, volume 17, pages 1481–1488, 2004.