Parametric Herding

Yutian Chen **Max Welling**
Bren School of Information and Computer Science
University of California, Irvine
Irvine, CA, USA
{yutianc, welling}@ics.uci.edu

Abstract

A parametric version of herding is formulated. The nonlinear mapping between consecutive time slices is learned by a form of self-supervised training. The resulting dynamical system generates pseudo-samples that resemble the original data. We show how this parametric herding can be successfully used to compress a dataset consisting of binary digits. It is also verified that high compression rates translate into good prediction performance on unseen test data.

1 INTRODUCTION

A deterministic nonlinear dynamical system was recently introduced in (Welling, 2009b; Welling, 2009a) as an alternative method for learning in Markov random field models (MRF). The proposed method, called herding, uses the data to drive the dynamics of both the weights as well as the (hidden and visible) random variables. Unlike learning in the traditional sense, the weights will never converge to a fixed point. Instead, their trajectories are non-periodic and generate complicated attractor sets (e.g. with fractal Hausdorff dimension). The relevant information is encoded in these trajectories, or equivalently, in the properties of the attractor. The surprising conclusion is therefore that a simple, *deterministic* nonlinear dynamical system can effectively capture the intricate dependencies present in a data stream and transfer that information to the task of making predictions on unseen data.

Herding seems a genuinely new approach to learning rooted in the theory of nonlinear dynamical systems. Maximum Likelihood learning can be understood as a dynamical system but with a single fixed point. Similarly, MCMC sampling of the posterior distribution of the parameters given data can also be thought of as a *stochastic* nonlinear dynamical system. Learning systems based on chaotic dynamics appears to be a third thus-far unexplored possibility that deserves some attention. The herding system under consideration does not require explicit random number generation because it derives its pseudo-randomness from the inherent chaos of the learning equations. Similar to bagging and Bayesian posterior sampling however, herding averages predictions over trajectories in model space, resulting in a variance reduction of the relevant estimators.

The nonlinear dynamical system in question turns out to be on the boundary between order and chaos, formally to be classified as “weakly chaotic” with polynomial sensitivity to initial conditions. One can show that the system is a special case of a larger class of weakly chaotic systems known as “piecewise isometries” (Goetz, 1996). In fact, herding is a special case of piecewise translations. Certain properties have been proven for such systems in the mathematics literature. For instance, it is known that piecewise isometries have vanishing topological entropy. This means that the number of distinct subsequences of length T grows polynomially in T (Goetz, 1996), in contrast to the exponential growth for stochastic and fully chaotic systems which have non-vanishing topological entropy. We are currently investigating the implications of these observations.

Two classes of herding algorithms have been studied so far. The algorithm of (Welling, 2009b) uses average sufficient statistics (ASS) as input to the system and returns pseudo-samples that respect these ASS. These samples can in turn be used for making predictions. In this case, the data are only represented through their ASS which severely limits the amount of correlations that can be modeled. A second class of herding systems was studied in (Welling, 2009a) where hidden units were introduced. In this case, the actual data itself (not just a few ASS) are used to drive the system. The output of this class of systems is again a sequence of pseudo-samples that respects certain (higher order) constraints similar to those enforced by Markov random field models (MRF) at their maximum likelihood solution. In

Appearing in Proceedings of the 13^{th} International Conference on Artificial Intelligence and Statistics (AISTATS) 2010, Chia Laguna Resort, Sardinia, Italy. Volume 9 of JMLR: W&CP 9. Copyright 2010 by the authors.

addition, the system outputs a sequence of hidden representations which can in turn be used for subsequent tasks such as classification or regression.

The second variety of herding requires all the data as input in order to generate pseudo-samples. Since we are required to store all data in memory in order to make predictions, we should view this method as inherently *non-parametric*. For instance, the pseudo-samples can be used to generate a non-parametric kernel estimate of the density function, or the hidden representations can be used as a basis for nearest-neighbor classification. However, it makes one wonder if one can define a *parametric* variant of herding that decouples the data and replaces them with a collection of parameters. In this paper we will show that this can indeed be achieved. Similar to vector quantization (VQ), we will replace the data with a smaller, weighted set of templates. Interestingly, these templates will be trained using regression, a supervised learning technique. We will show in experiments that this parametric herding system performs very well in compressing the training data sequence, taking the complexity of the model components into account. We also show that this directly translates into good performance on test data (as expected through the MDL principle). We believe that this contribution will bring herding one step closer to being a practical algorithm for machine learning problems.

2 HERDING AS A DYNAMICAL SYSTEM

In the following we will review herding from a slightly different perspective, namely that of a dynamical system. We first introduce visible random variables x_α where α labels a subset of the total set of random variables (denoted with $\mathbf{x}$). Similarly, we introduce hidden variables z_α and feature functions $g_\alpha(x_\alpha, z_\alpha)$. When we write $x_{\alpha n}$ we shall mean the observation n on the subset x_α. Similarly to the construction of Markov random fields, we first define an energy over joint configurations $\mathbf{s} = (\mathbf{x}, \mathbf{z})$ as follows,

$$\mathcal{E} = -\sum_\alpha w_\alpha g_\alpha(x_\alpha, z_\alpha) \tag{1}$$

where w_α is a weight associated to feature g_α. Herding will now be defined as a nonlinear dynamical system for the weights (see Figure 1),

$$\mathbf{w}_t = F_t(\mathbf{w}_{t-1}) \tag{2}$$

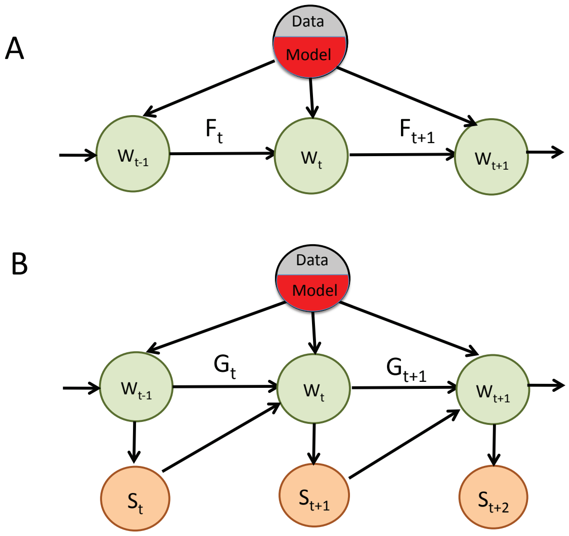

Figure 1: A: Herding as a dynamical system for the weights w_α. The original herding algorithm was driven by data, while the proposed herding system is driven by a model of the data. B: Same as A, but now depicted as a dynamical system over the joint space $\mathbf{w}, \mathbf{s}$.

where the mapping F_t is defined through,

$$w_{\alpha t} = w_{\alpha, t-1} + \rho_{\alpha t}(\mathbf{w}_{t-1}) \tag{3}$$

$$\rho_{\alpha t}(\mathbf{w}_{t-1}) \doteq \bar{g}_{\alpha t}(\mathbf{w}_{t-1}) - g_{\alpha t}(s^*_{\alpha t}(\mathbf{w}_{t-1})) \tag{4}$$

$$\bar{g}_{\alpha t}(\mathbf{w}_{t-1}) \doteq \frac{1}{N}\sum_{n=1}^{N} g_\alpha(x_{\alpha n}, z^*_{\alpha n t}(\mathbf{w}_{t-1})) \tag{5}$$

$$\mathbf{z}^*_{nt}(\mathbf{w}_{t-1}) \doteq \arg\max_{\mathbf{z}_n} \sum_\alpha w_{\alpha, t-1} g_\alpha(x_{\alpha n}, z_{\alpha n}) \tag{6}$$

$$\mathbf{s}^*_t(\mathbf{w}_{t-1}) \doteq \arg\max_{\mathbf{s}} \sum_\alpha w_{\alpha, t-1} g_\alpha(s_\alpha) \tag{7}$$

We first note that the mapping F is a local translation over a vector $\boldsymbol{\rho}_t(\mathbf{w}_{t-1})$. Moreover, the translation vector $\boldsymbol{\rho}_t$ depends nonlinearly on $\mathbf{w}$ (due to the various maximizations). However, $\boldsymbol{\rho}$ depends on $\mathbf{w}$ only through the quantities $\mathbf{z}^*$ and $\mathbf{s}^*$ which are themselves defined as maximizations. Since both $\mathbf{z}$ and $\mathbf{s}$ are discrete, their values (obtained through maximization) are stable against infinitesimal perturbations of $\mathbf{w}$, implying that the mapping F is a *piecewise constant translation*. For later reference we also note that the function $\bar{g}$ does not depend on the scale of $\mathbf{w}$, i.e. multiplying all weights by a constant factor will have no effect on the outcome of the maximizations. In (Welling, 2009a) it was shown that $\mathbf{w}$ will not diverge to infinity, but stay within a compact region around the origin. The attractor set that is traced out by the dynamical system can indeed be quite intricate and seems (numerically) to have fractal Hausdorff dimension.

In the following it will also be useful to view the system from a slightly different perspective, namely as a dynamical system on the joint space $\mathbf{w}, \mathbf{s}$ (see Figure 1),

$$(\mathbf{w}_t, \mathbf{s}_t) = G_t(\mathbf{w}_{t-1}, \mathbf{s}_{t-1}) \tag{8}$$

where the states $\mathbf{s}$ is called the symbolic sequence, or "itinerary" of the dynamical system.

Even though the dynamical system is highly nonlinear with complex dynamics as a result, certain average statistics on the symbols $\mathbf{s}$ are conserved, namely,

$$\lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} \left[\frac{1}{N} \sum_{n=1}^{N} g_\alpha(x_{\alpha n}, z^*_{\alpha n t}) \right] = \lim_{T\to\infty} \left[\frac{1}{T} \sum_{t=1}^{T} g_\alpha(s^*_{\alpha t}) \right] \tag{9}$$

These are the same constraints that would be satisfied by the model $P_{\text{MRF}}(\mathbf{x},\mathbf{z}) \propto \exp[-E(\mathbf{x},\mathbf{z})]$ at its ML solution when we replace averages over pseudo-samples with averages over the model P_{MRF}. However, samples from P_{MRF} will not necessarily be identically distributed as the sample sequence $\{\mathbf{s}_t\}$ because herding does not generate samples of maximal entropy subject to these constraints (as P_{MRF} would). The characterization of the differences between herding and maximum entropy/likelihood models is currently under investigation. It represents a different inductive bias on the degrees of freedom that remain unconstrained by the data.

3 PARAMETRIC HERDING

We now turn to the key point of this paper. We note from Eqn.5 that the mapping F only depends on the data through the functions $\bar{g}_\alpha(\mathbf{w})$. Therefore, if we can learn a parameterized regression function $r_\alpha(\mathbf{w})$ that approximates this term, then we can decouple the data entirely and use r_α instead of $\bar{g}_\alpha$. We would have turned an essentially nonparametric method into a parametric one.

Fortunately, learning the regression functions r_α is very simple in principle. The reason is that by running the herding equations using data, we generate an unlimited dataset of pairs $\{\mathbf{w}_t, \bar{g}_t \doteq \bar{g}(\mathbf{w}_t)\}$. Hence, we can take any off-the-shelves regression method to learn the relation between $\mathbf{w}$ and $\bar{g}$. Interestingly, we are using supervised learning techniques to solve an unsupervised learning problem. Not unlike (Welling et al., 2002) one can argue that the system is "self-supervising".

For restricted Boltzman machines (using the ±1 representation) the features are given by $g_{ij} = z_i x_j$, $g_i = z_i$, $g_j = x_j$. Since $\bar{g}_j = \frac{1}{N}\sum_{n=1}^{N} x_{jn}$ is independent of $\mathbf{w}$ we don't need a regression function for it. For the other features, a suitable regression function that respects the bounds $r_{ij} \in [-1,+1]$ and $r_i \in [-1,+1]$ is given by,

$$r_{ij}(\mathbf{w},\boldsymbol{\theta};\mathbf{A},\boldsymbol{\pi},b) = \sum_{m=1}^{M} \pi_m \tanh\left(b \left(\sum_k w_{ik} A_{km} + \theta_i \right)\right) A_{jm} \tag{10}$$

$$r_j(\mathbf{w},\boldsymbol{\theta};\mathbf{A},\boldsymbol{\pi},b) = \sum_{m=1}^{M} \pi_m \tanh\left(b \left(\sum_k w_{jk} A_{km} + \theta_j \right)\right) \tag{11}$$

where $\{A_{km}\}$ represent M protoypes to be learned with their corresponding weights π_m. b is a scalar and $\{\mathbf{w},\boldsymbol{\theta}\}$ are dynamical variables subject to herding. The form of this function is motivated by softening the expressions for $\bar{g}_{ij}$ and $\bar{g}_j$ and introducing prototypes to replace the data. The softening is required to be able to compute gradients for optimizing the parameters of the regression function. However, the result of this is that unlike the functions $\bar{g}$, the regression functions r do depend on the scale of $\mathbf{w}$ in a nontrivial way. The parameter b is introduced to offset this effect.

We estimate the regression parameters by minimizing the average sum of squared residuals (SSE) over the time:

$$C(\mathbf{A},\boldsymbol{\pi},b) = \frac{1}{T}\sum_{t=1}^{T} SSE(t) \tag{12}$$

$$SSE(t) = \sum_i \sum_j (r_{ij}(\mathbf{w}_t,\boldsymbol{\theta}_t;\mathbf{A},\boldsymbol{\pi},b) - \bar{g}_{ij}(\mathbf{w}_t,\boldsymbol{\theta}_t))^2 + \sum_j (r_j(\mathbf{w}_t,\boldsymbol{\theta}_t;\mathbf{A},\boldsymbol{\pi},b) - \bar{g}_j(\mathbf{w}_t,\boldsymbol{\theta}_t))^2 \tag{13}$$

where $\mathbf{w}_t, \boldsymbol{\theta}_t$ are samples from herding. The objective function represents the expected SSE if we assume that herding will sample from some (unknown) distribution $\hat{p}(\mathbf{w},\boldsymbol{\theta})$. Since herding generates one pair of $((\mathbf{w}_t,\boldsymbol{\theta}_t),\bar{g}_t)$ per iteration, it's natural to run an online gradient descent algorithm to update the parameters $\boldsymbol{\pi}, \mathbf{A}, b$

4 DATA COMPRESSION

One possible way to evaluate a learning algorithm is to test its ability to compress a dataset. It has long been known that there are close connections between compression performance and the ability to generalize to new (unseen) data (Rissanen, 1989). The cost of compression, measured in bits, is usually divided in three separate terms: 1) The (constant) cost of coding the model parameters, 2) the cost of coding the states of the hidden variables of the model (linear in N) and 3) the cost of coding the residual errors (linear in N). Optimal model complexity is achieved by trading off the first term against the latter two. It was shown in (Hinton & Zemel, 1994) that by choosing the states of the hidden variables stochastically according to its posterior $p(\mathbf{z}|\mathcal{D},\boldsymbol{\theta})$ (where $\mathcal{D}$ denotes the dataset and $\boldsymbol{\theta}$ denotes the parameters) one will get back a refund equal to the entropy of this posterior distribution[1]. This bits-back trick will sometimes be used in the following although for some models, such as the RBM, it will turn out to be intractable.

This encoding scheme corresponds to MAP estimation where in addition to the usual log-likelihood terms certain regularizer terms (corresponding to log-priors for the parameters) are present. A full Bayesian treatment would require choosing parameters stochastically from its posterior distribution $p(\boldsymbol{\theta}|\mathcal{D})$ and receiving another refund equal to the entropy of this posterior distribution. However, to claim your refund, you need to be able to compute the posterior $p(\boldsymbol{\theta}|\mathcal{D})$ after all the data have been transmitted. In the case of MRF models such as the RBM this is intractable and therefore unrealistic. For this reason we will omit this bits-back term. Instead, we will encode the model parameters up to a certain precision (or quantization level), Δ, assuming a Gaussian distribution for the encoding. The cost of this encoding is therefore equal to $-\log\Delta - \log\mathcal{N}(\theta_i)$ for every parameter. The value of Δ is chosen such that if we add independent uniform quantization noise in the range $[-\Delta/2, \Delta/2]$ to all the parameters, then the contribution of these perturbations constitute less than 1% of the total compression rate.

4.1 COMPRESSION WITH VECTOR QUANTIZATION

The idea of VQ is to divide the data vector space into K partitions, and represent all the points in a partition by a common vector a.k.a. codeword $\mathbf{c}_k$. If a data point $\mathbf{x}_n$ is inside the kth partition, we compress it by storing only the index k and the error $\mathbf{x} - \mathbf{c}_k$. Given a probability distribution of a signal s, the minimal coding length we can achieve is the negative log-likelihood. Moreover, we can get arbitrary close to that value with entropy encoding on a large enough dataset. Hence, we will treat $-\log(p(s))$ as the true coding length.

For the binary image compression task in this paper, we assume that the prior for data to be in partition k is π_k and we model the error at each pixel as an independent Bernoulli random variable with probability $p_{0k} = P(x_{jn} \neq c_{jk}), \forall j$. Also, we compress the parameters $\{\mathbf{c}_k\}$ as Bernoulli distributed random variables and $\{p_{0k}, \pi_k\}$ as Normal distributed random variables with a quantization level Δ. The total coding length is thus

$$L = L_{param} + L_{code} \qquad \text{with} \tag{14}$$

$$L_{code} = \sum_{n=1}^{N} [-\log \pi_{z_n} - \log \mathcal{B}(\mathbb{I}(\mathbf{x}_n \neq \mathbf{c}_k); p_{0k_n})] \tag{15}$$

ans where L_{param} is the total coding length for all the parameters.

According to the bits-back argument (Hinton & Zemel, 1994), we can potentially encode cheaper by stochastically picking the index z_n from some distribution $q(z|\mathbf{x}_n)$ and claiming a number of bits back after transmission equal to the entropy of this distribution $\mathcal{H}(q)$. Note, however, that by sub-optimally picking the index we also incur a higher error. The optimal encoding is achieved if we use the posterior for q, i.e.

$$q(z|\mathbf{x}_n) \propto e^{-L_{code}(\mathbf{c}_k, \mathbf{x}_n)} \tag{16}$$

With the bits-back scheme, we can show that the coding length becomes

$$L_{code} = \sum_{n=1}^{N} \left[-\log \sum_k \pi_k \mathcal{B}(\mathbb{I}(\mathbf{x}_n \neq \mathbf{c}_k); p_{0k}) \right] \tag{17}$$

which is equivalent to the negative log-likelihood of a mixture of Bernoulli distributions (MoB) with the variables $\{z_k\}$ marginalized out.

4.2 COMPRESSION WITH HERDING

Recall that herding, instead of defining a model explicitly, defines a model only *implicitly* by generating a sequence of pseudo-samples. As such, it is not immediately evident how to use it to compress a data collection. Our approach will be to run herding for a very long time (e.g. $T = 10^7$ iterations) starting at some randomly initialized values for the weights and biases, $\mathbf{w}, \boldsymbol{\theta}$, at time $t = 0$ and using the learned regression functions $\{r_{ij}, r_i\}$ (see section 3). In this way we will generate a new codebook vector $\mathbf{c}_t$ at every iteration given as the visible part, $\mathbf{x}_t^*$ of the pseudo-sample $\mathbf{s}_t^* = [\mathbf{x}_t^*, \mathbf{z}_t^*]$ at iteration t. Note, that the receiver can also generate these codebooks, so they do not have to be transmitted.

The initial values, $\mathbf{w}_0, \boldsymbol{\theta}_0$, are sampled from a Normal distribution and communicated by sending a random seed and a scale factor (the standard deviation) so that they can be replicated at the receivers end who uses the same random number generator. The time indices of the pseudo-samples now serve as our hidden variables and they will be encoded under a uniform prior (a.k.a. all time indices are equally likely). We pick a time index for each data-case ($\tau_n, \ \forall n$) according to its posterior distribution which allows us to receive a modest refund equal to the entropy of this posterior (which the receiver needs to compute after all data has been transmitted). Incorporating this bits-back argument, we can compute the coding length L_{code} as:

$$L_{code} = \sum_{n=1}^{N} \left[-\log \frac{1}{T} \sum_{t=1}^{T} \mathcal{B}(\mathbb{I}(\mathbf{x}_n \neq \mathbf{c}_t); p_0) \right] \tag{18}$$

[1] Suboptimal "variational" distributions can also be used as a substitute for the posterior.

where T is the total number of iterations. Note that this equals a Bernoulli mixture model with a number of components equal to the length of herding sequence.

We encode the model parameters $A, \boldsymbol{\pi}, b$ using Normal distributions. However, since the prototypes very closely follow a mixture of two Gaussians model we use that for its encoding (see Figure 2). The residual prediction errors are encoded using a Bernoulli distribution with a single probability p_0 of making an error.

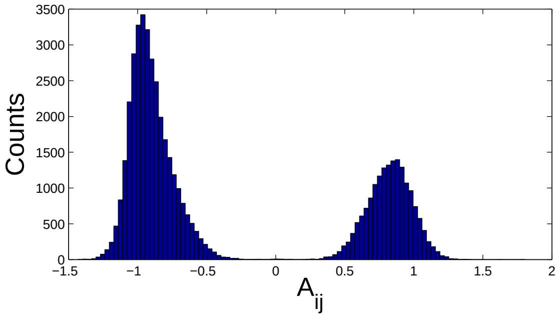

Figure 2: Histogram of the elements of prototypes $\mathbf{A}$, trained on the USPS digit image dataset

4.3 COMPRESSION WITH RBM

A natural question is how herding compares in terms of compression performance with its associated energy based model (a restricted Boltzmann machine or RBM). We have tested RBMs on the described compression task in two distinct ways. The first method (RBM-S) is similar to the strategy employed for herding and uses Gibbs sampling to generate a large codebook given by these samples. Although sampling is random and thus unrepeatable in principle, this can be circumvented on a digital computer using a pseudo-random number generator where again the sender and receiver have to agree on the seed value (and a scale factor to sample the initial values from a Normal distribution).

The second way to compress with RBMs (RBM-H) is to map the data-vector to the latent state space of the RBM and transmit that latent state ($\mathbf{z}$) along with the reconstruction error. During decoding, the signal is recovered as $\mathbf{x} = \arg\max_{\mathbf{x}} p(\mathbf{x}|\mathbf{z}) + error$. The optimal latent state is searched locally through hill climbing for a minimal reconstruction error starting from the state $\arg\max_{\mathbf{z}} p(\mathbf{z}|\mathbf{x})$. Both the latent states and errors are encoded as Bernoulli random variables with probabilities $p_i = p(z_i = 1)$, $q_j = p(error_j = 1)$[2].

5 EXPERIMENTS

We report compression results on the USPS Handwritten Digits dataset[3] which consists of 1100 examples of each digit 0 through 9 (totaling 11,000 examples). Each image has 256 pixels and each pixel has a value between [1..256] which we turned into a binary representation through the mapping $X'_i = 2\Theta(X_i - 50) - 1$ with $\Theta(x > 0) = 1$ and 0 otherwise. Each digit class was randomly split into 700 train and 400 test examples.

5.1 REGRESSION FUNCTION TRAINING

We train the regression functions on the USPS digit image dataset as follows. A MoB model is first trained by the regular EM algorithm to provide sensible initializations for the prototypes. The training is then divided into two phases. We first herd the parameters of an RBM using the data and feed the pairs $\{(\mathbf{w}, \boldsymbol{\theta}), \bar{g}(\mathbf{w}, \boldsymbol{\theta})\}$ to an online gradient descent algorithm to train $\mathbf{r}(\mathbf{w}, \boldsymbol{\theta})$. The resulting regression function has a small SSE in the area of high probability under the true distribution $\hat{p}(\mathbf{w}, \boldsymbol{\theta})$. However, when we replace the data by the prototypes, the distribution of the pseudo-samples, $\hat{p}^*(\mathbf{w}, \boldsymbol{\theta})$, is different from (but similar to), $\hat{p}(\mathbf{w}, \boldsymbol{\theta})$. We have found it beneficial to further fine-tune the regression functions by collecting pairs $\{(\mathbf{w}, \boldsymbol{\theta}), \bar{g}(\mathbf{w}, \boldsymbol{\theta})\}$ from herding using the latest estimates of the regression functions (instead of the data). In this second phase, we are actually attempting to minimize $\langle SSE \rangle_{\hat{p}^*(\mathbf{w}, \boldsymbol{\theta})}$. Figure 3 shows the average SSE over 10^4 iterations when we run herding on images of digit "8" with $\mathbf{r}(\mathbf{w}, \boldsymbol{\theta})$. The plot with fine tuned regression functions has lower and more stable SSE than those trained only by phase 1. Figure 4 shows some examples of learnt prototypes.

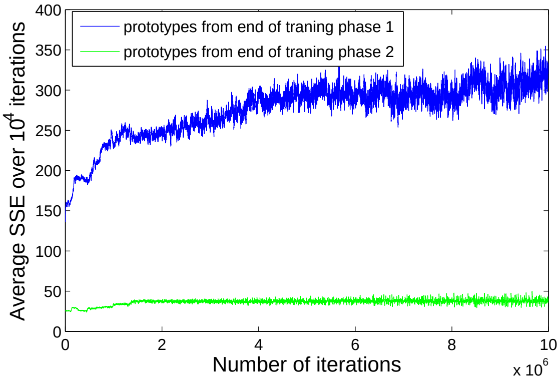

Figure 3: Average SSE of the regression functions over 10^4 iterations with 20 prototypes for an RBM with 100 hidden units, trained on 700 images. $\mathbf{r}(\mathbf{w})$ are trained through only phase 1 (top) or both phases (bottom)

Next, we test the estimated models in terms of L_{code} (see Eqn. 18) by comparing the compression performance of

[2] It should be noted that for RBMs it is intractable to apply the bits-back argument. The reason is that one needs the marginal distribution $p(\mathbf{z})$ to encode the hidden units stochastically and if one uses a variational distribution instead, it is intractable to compute the variational posterior distribution necessary to reclaim the redundant bits.

[3] Downloaded from *http://www.cs.toronto.edu/∼roweis/data.html*

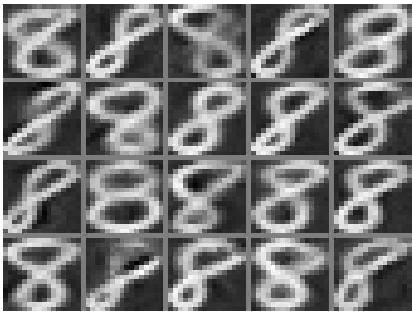

Figure 4: Examples of tuned prototypes on images of digit “8”

herding with various number of prototypes against that of herding with data. The coding lengths are plotted in Figure 5 as a function of iteration. We observe that longer herding will improve the compression performance because it becomes increasingly likely that a pseudo-sample is generated that resembles the data-case we wish to encode. However, there is a price to be paid, namely the fact that we need to encode the time indices of the pseudo-sample. It is interesting that even after 1M iterations the curve is still descending.

We further note that we have omitted the cost of encoding the parameters in this plot which explains why herding with data has the lowest coding cost[4]. However, one should keep in mind that the latter method is not a realistic method to compress the data (since it requires the data itself to run).

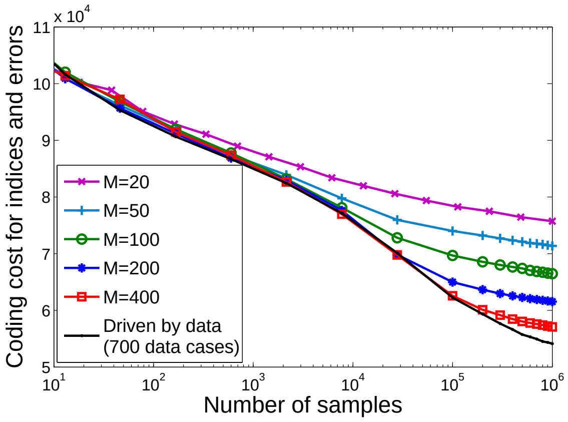

Figure 5: L_{code} for images of digit “8” by herding parameters of an RBM with 100 hidden units with various numbers of prototypes.

In (Welling, 2009a) (section 9) an alternative method to parametrize herding was proposed. There, the functions $\bar{g}_\alpha(\mathbf{w}_t, \boldsymbol{\theta}_t)$ were simply approximated by constants r_α (independent of $\mathbf{w}, \boldsymbol{\theta}$). Their values were estimated by running herding with data and averaging the values of $\bar{g}_{\alpha t}$ over time: $r_\alpha = \frac{1}{T}\sum_{t=1}^{T} \bar{g}_\alpha(\mathbf{w}_t, \boldsymbol{\theta}_t)$. In effect, we approximate a RBM with hidden variables with a fully observed one where the functions r_α play the role of observed sufficient statistics (ASS). In Figure 6, we compare the coding cost of herding driven by a regression function (100 hidden units and various numbers of prototypes), data and ASS. With 100 hidden units in the RBM, we need regression functions with about 200 prototypes to achieve similar performance to the ASS method. However, to decrease the model complexity (with fewer bits for encoding parameters) we can reduce the number of prototypes without changing the structure of RBM. This is not true for herding with ASS because it has to reduce its number of hidden units to save on parameters which severely affects the performance. Around the optimal model complexity ($M/K = 20$) for the subset of digits “8” the coding cost of ASS is much higher than that of our proposed regression function.

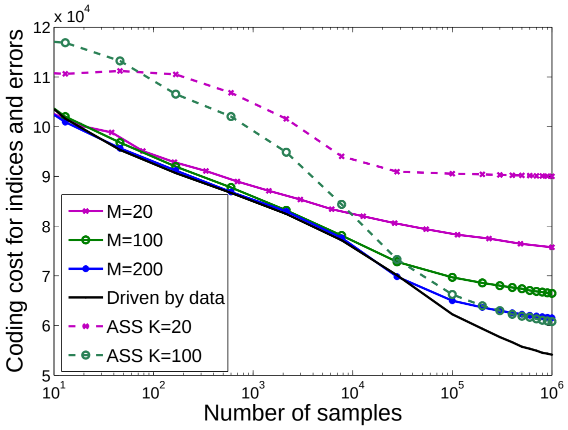

Figure 6: L_{code} of herding for images of digit “8” driven by regression function (with M prototypes and 100 hidden units), data itself, and ASS (with K hidden units).

5.2 COMPRESSION PERFORMANCE

The various algorithms are compared in terms of their total coding length on the 16x16 USPS digit image dataset. We use a small set of a images from the digit “8” and a larger set including all the digits. The bench mark is to treat every pixel as an independent Bernoulli random variable with a probability p. The average coding length of each pixel is then $\mathcal{H}(p)$, where the minimum is achieved when p is the percentage of pixels with value 1.

RBMs are trained using contrastive divergence with 10 steps (CD-10) (Hinton, 2002) or by using persistent CD (PCD) (Tieleman, 2008). Parameters such as the number of codewords for VQ, the number of hidden units for RBM and herding and the number of prototypes for herding are found by a linear or grid search for a minimal coding

[4] This was done to easily assess how accurately parametric herding approximates the “nonparametric” herding version that requires the data itself to run.

length. The samples from RBM-S are subsampled at a rate of $1 : 100$ from their Gibbs sampling sequence to reduce the autocorrelation (there is no extra coding cost involved for this but encoding and decoding will be more time consuming). Parameter quantization is set by the method described in section 4.

Figure 7 and 8 show the decrease of the total coding length with the number of samples for herding and RBM-S on the dataset of digit “8” and on the full set respectively. As the number of samples increases, the number of bits for encoding the errors decreases while that for indices increases. Given enough iterations, these plots will most likely start to increase. However, the minima are beyond the scope of our experiments. We find that herding always achieves smaller coding length than RBMs with either training methods.

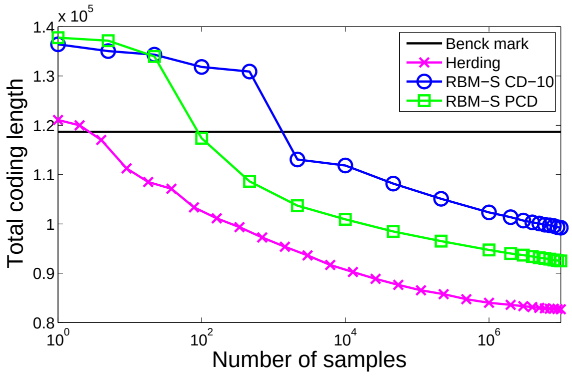

Figure 7: Total coding length against the number of samples for digit “8” for herding (x-mark) and RBM-S CD-10(circle)/PCD(square). The black line is the bench mark.

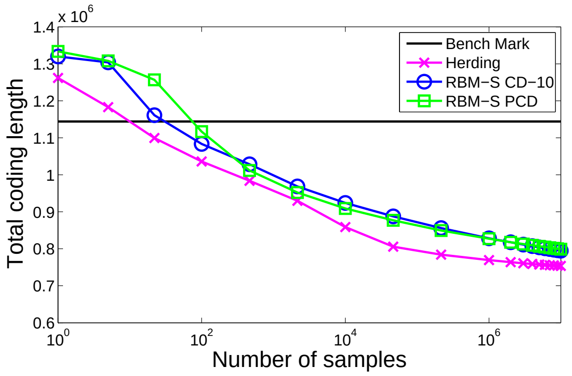

Figure 8: Total coding length against the number of samples for all the digits for herding (x-mark) and RBM-S CD-10(circle)/PCD(square). The black line is the bench mark.

Figures 9 and 10 show the coding length of various compression methods in three parts: model complexity, bits for indices and the errors. All the methods achieve about $60 \sim 70\%$ compression ratio on digit “8” and $65 \sim 70\%$ on the whole dataset compared to the benchmark. Although the differences are small, herding does seem to have the shortest coding length.

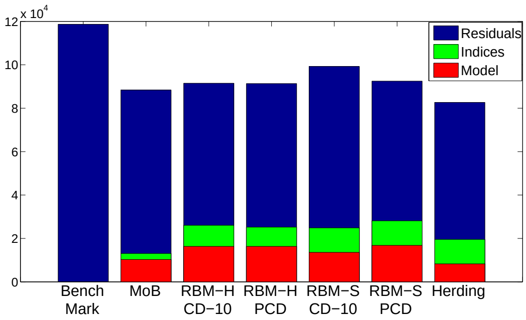

Figure 9: Total coding length for the digit “8”.

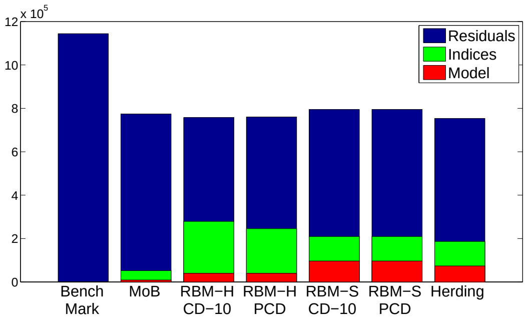

Figure 10: Total coding length for all the digits.

The gap in coding length between herding and RBM methods, even at the first iteration in figures 7 and 8 can be traced back to the fact that we can more cheaply encode the prototypes shown in figure 4 using a mixture of two Gaussians (see figure 2) than that we can encode the weights of the RBM. In fact, the RBM weights were much more sensitive to their quantization level Δ. To render the contribution of the quantization level negligible (i.e. less than 1% of the total coding cost), we had to encode the RBM parameters to a higher precision than the parameters for herding (i.e. the prototypes). If we discard the cost for encoding parameters and only compare models with an equal number of parameters then the remaining coding lengths L_{code} are comparable.

Finally, we looked at compression performance on a test set. Suppose that we want to compress a data stream of *i.i.d.* signals. A model can be trained with the first a few samples, and then applied to the remaining data. According to the MDL principle, the model with the minimal total coding length on the training data should also be optimal for unseen test data. Hence, we compare these models with their optimal choice of parameters in terms of MDL. We tested on 300 (unseen) images for each digit. The numbers of bits for encoding time indices and errors, L_{code}, are shown in figure 11 and 12 respectively for images of a single digit “8” and the full set. Note that L_{code} is equal to the log-likelihood of the test data (which the usual criterion for prediction performance) and that it doesn’t make much sense to include the cost of encoding the parameters in this case. Again, herding is performing well on both datasets. Its test coding length is shorter than MoB and RBM-S while marginally longer than RBM-H on digit “8”, and it’s slightly better than all the other algorithms on the full set.

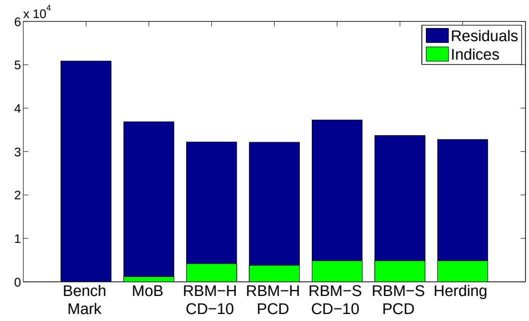

Figure 11: Coding length for test images of digit “8”.

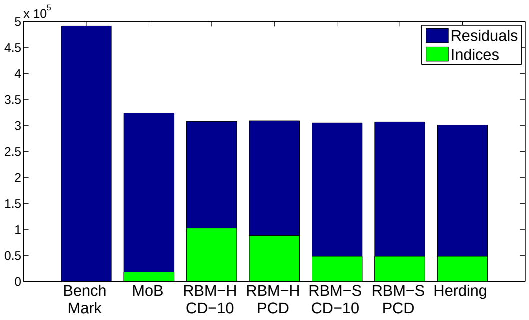

Figure 12: Coding length for test images of all the digits.

6 CONCLUSIONS

The results of this paper may be interpreted as a new method to train a (weakly chaotic) nonlinear dynamical system $(\mathbf{w}_t, \mathbf{s}_t) = G_t(\mathbf{w}_{t-1}, \mathbf{s}_{t-1})$ in such a way that the sequence of discrete states $\{\mathbf{s}_t\}$ (or at least the visible subset $\mathbf{x}_t$ of it) approximates the distribution that generated the data from which it was trained. This definition of a “model” is perhaps a little different than what we are used to. The approach is perhaps best compared with dependency networks (Heckerman et al., 2000). In this method the conditional probability distributions are learned from data which can subsequently be used as kernels for Gibbs sampling. However, the conditional probability distributions by themselves do not provide access to any analytic expression for the joint probability distribution. The latter is only accessible indirectly through the samples generated by the Markov chain. In a similar fashion, parametric herding provides a way to generate pseudo-samples which in turn indirectly represent the joint probability distribution. The important difference is that herding is a *deterministic*, but weakly chaotic dynamical system while Gibbs sampling is a stochastic system. An important advantage of herding is that it, unlike many MCMC methods, is designed to avoid getting stuck in local modes of the distribution.

Herding has also important computational advantages. Unlike MRF learning, it integrates learning and sampling into one algorithm. What we have shown in this paper is that this somewhat unconventional way of generating pseudo-samples can be successfully employed to the task of data compression. The reported compression rates are similar to the results obtained by RBMs. This basically confirms our intuition that (at least for the binary dataset that we have studied) herding samples from a distribution that is closely related to the distribution from which the data are sampled but at a lower computational cost (due to the absence of a separate learning phase and the much smaller, often negative auto-correlation between subsequent samples). We believe that this validates the approach as a viable alternative to more standard learning approaches.

Acknowledgements

This work is supported in part by NSF grants IIS- 0447903 and IIS-0535278 as well as ONR/MURI grant 00014-06-1-073.

References

Goetz, A. (1996). Dynamics of piecewise isometries. *Illinois Journal of Mathematics*, *44*, 465–478.

Heckerman, D., Chickering, D. M., Meek, C., Rounthwaite, R., & Kadie, C. (2000). Dependency networks for inference, collaborative filtering, and data visualization. *Journal of Machine Learning Research*, *1*, 49–75.

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

Hinton, G. E., & Zemel, R. S. (1994). Autoencoders, minimum description length and helmholtz free energy. *Advances in Neural Information Processing Systems 6* (pp. 3–10). Morgan Kaufmann.

Rissanen, J. (1989). *Stochastic complexity in statistical inquiry theory*. River Edge, NJ, USA: World Scientific Publishing Co., Inc.

Tieleman, T. (2008). Training restricted boltzmann machines using approximations to the likelihood gradient. *Proceedings of the International Conference on Machine Learning* (pp. 1064–1071).

Welling, M. (2009a). Herding dynamic weights for partially observed random field models. *Proc. of the Conf. on Uncertainty in Artificial Intelligence*. Montreal, Quebec, CAN.

Welling, M. (2009b). Herding dynamical weights to learn. *Proceedings of the 21st International Conference on Machine Learning*. Montreal, Quebec, CAN.

Welling, M., Zemel, R., & Hinton, G. (2002). Self-supervised boosting. *Neural Information Processing Systems*. Vancouver, Canada.