Unlearning for Better Mixing

Olivier Breuleux, Yoshua Bengio, and Pascal Vincent
Dept. IRO, Université de Montréal
breuleux@gmail.com, yoshua.bengio@umontreal.ca,
vincentp@iro.umontreal.ca

Technical Report 1349, November 5, 2009, Dept. IRO, U. Montreal

Abstract

Two learning algorithms were recently proposed – **Herding** and **Fast Persistent Contrastive Divergence** (FPCD) – which share the following interesting characteristic: they exploit changes in the model parameters while sampling in order to escape modes and mix better, during the sampling process that is part of the learning algorithm. We first justify such approaches as ways to escape modes while approximately keeping the same asymptotic distribution of the Markov chain. We then extend FPCD using an idea borrowed from Herding in order to obtain a pure sampling algorithm and show empirically that this FPCD-sampler yields substantially better samples than Gibbs sampling. Because these algorithms entangle the model and the sampling algorithm and we want to evaluate both (but particularly how well the sampling schemes mix), it is not always easy to evaluate them, so we propose a "black-box" approach based on how well and how quickly the samples generated by a model "cover" the test set examples. We empirically study these algorithms and variations with this perspective and these new evaluation tools in order to better understand their strengths and limitations.

1 Introduction

Undirected graphical models such as Markov random fields (MRF) and variants of Boltzmann machines are equipped with stochastic sampling procedures that are normally considered independently of the model itself, characterized by its parameters. Here, we study the models structured like the Restricted Boltzmann Machines (RBM) (Hinton et al., 2006), i.e., with a set of visible (observed) variables, a set of hidden variables, and an energy function that only contains terms for (hidden,visible) pairs. Recent work on training algorithms for the RBM - the Fast Persistent Contrastive Divergence (FPCD) algorithm (Tieleman & Hinton, 2009) - is creating new interactions between learning and sampling by allowing two sets of weights (parameters): slow weights represent the model, while fast 'dynamic' weights (added on top of slow weights with a fast weight decay towards 0 and a large learning rate) are used *during learning* to help the sampling procedure better explore the modes of the distribution while learning. Herding is another very recently proposed algorithm (Welling, 2009b; Welling,

2009a) in which learning and sampling seem entangled. Herding can be obtained by considering the zero-temperature limit of a Markov random field, i.e. where sampling is replaced by minimization (of the energy). A Herding algorithm defines a dynamical system in which samples and weights evolve deterministically, producing a stream of samples whose statistics match those of the training samples under some hypotheses which can be guaranteed at a computational cost. These algorithms underscore several core principles which may be generalized to pure generative procedures for MRFs (and RBM models in particular) in order to improve mixing, which we define as the rate at which a stream of samples covers its underlying equilibrium distribution. We find that mixing is very poor for Gibbs sampling, the method that is typically used to draw samples during or after training, and therefore there is much room for improvement using the proposed new methods. Another important issue which we will address is how to evaluate sample quality and mixing quantitatively.

2 RBMs, CD, PCD, FPCD, and Herding

Although many of the ideas presented here should apply to more general MRFs, we restrict our attention here to RBMs with binary units, applied to binary or binarized data. RBMs and Herding can be defined from the following energy function, where $\mathbf{x}$ represents a vector of input or visible units and $\mathbf{h}$ represents a vector of latent or hidden units, with overall state $\mathbf{s} = (\mathbf{x}, \mathbf{h})$:

$$\text{Energy}(\mathbf{s}) = -\left(\sum_i b_j h_j + \sum_i c_i x_i + \sum_{ij} W_{ij} x_i h_j \right) \tag{1}$$

where parameters are $\theta = (b, c, W)$. The observed data likelihood is $P(\mathbf{x}) \propto \sum_{\mathbf{h}} e^{-\text{Energy}(\mathbf{x},\mathbf{h})}$. Following the Boltzmann Machine (Hinton et al., 1984) terminology, the gradient of the negative log-likelihood, which we aim to minimize, can be decomposed into a "positive phase" part (where $\mathbf{x}$ is clamped to the observed data vector) and a "negative phase" part (where an $\mathbf{x}^-$ is sampled from the model itself):

$$\begin{aligned} \frac{\partial - \log P(\mathbf{x})}{\partial \theta} &= + \sum_{\mathbf{h}} P(\mathbf{h}|\mathbf{x}) \frac{\partial \text{Energy}(\mathbf{x}, \mathbf{h})}{\partial \theta} \\ &\quad - \sum_{\mathbf{x}^-, \mathbf{h}^-} P(\mathbf{x}^-, \mathbf{h}^-) \frac{\partial \text{Energy}(\mathbf{x}^-, \mathbf{h}^-)}{\partial \theta} \end{aligned} \tag{2}$$

The structure of RBMs is such that the positive phase contribution of the gradient can be computed exactly and efficiently. The negative phase contribution is trickier and requires estimating the relevant statistics using samples $(\mathbf{x}^-, \mathbf{h}^-)$ (we will call these negative samples) from the model's distribution. Supposing that one such sample is obtained, an estimator $\hat{g}$ of the gradient g can be computed and an update of the model's

parameters is performed by stochastic gradient descent $\theta \leftarrow \theta + \epsilon \hat{g}$ as follows:

$$\begin{aligned} W &\leftarrow W + \epsilon(\mathbf{x}' E[\mathbf{h}|\mathbf{x}] - \mathbf{x}^{-\prime} E[\mathbf{h}^-|\mathbf{x}^-]) \\ b &\leftarrow b + \epsilon(E[\mathbf{h}|\mathbf{x}] - E[\mathbf{h}^-|\mathbf{x}^-]) \\ c &\leftarrow c + \epsilon(\mathbf{x} - \mathbf{x}^-) \end{aligned} \tag{3}$$

where ϵ is a learning rate and $E[\mathbf{h}|\mathbf{x}]$ (resp. $E[\mathbf{h}^-|\mathbf{x}^-]$) may be computed directly and efficiently from the observed sample(s) $\mathbf{x}$ (resp. negative phase sample x^-).

The major difference between CD (for Contrastive Divergence), PCD (for Persistent CD), FPCD (for fast PCD) and Herding (and thus their respective contributions) is how the negative samples are obtained - besides this, they all use the same update mechanism.

The Contrastive Divergence (CD) training algorithm (Hinton, 1999; Hinton, 2002; Hinton et al., 2006) obtains these samples by iterating one or more steps of a block Gibbs chain initialized on an observed example $\mathbf{x}$. A disadvantage of the CD approach is that the negative samples will often lie in the immediate vicinity of the positive samples, which can not only allow many spurious modes elsewhere but also create energy barriers around the training examples that makes Gibbs sampling work very poorly as soon as it ventures outside of the wells corresponding to training examples (and this is particularly evident when starting the Gibbs chain from a random state rather than from a real data point).

To solve this problem, the Persistent Contrastive Divergence (PCD) training algorithm (Tieleman, 2008) (see also earlier theoretical analysis by Younes (1998)) estimates the negative phase contribution of the gradient by sampling continuously from a unique Gibbs chain initialized at the very beginning, but whose transition probabilities slowly change as a result of parameter updates. Effectively, this means that at each time step we run one iteration of a block Gibbs chain initialized at $\mathbf{h}^-_{t-1}$ to conditionally sample $\mathbf{x}^-_t$ from, and then $\mathbf{h}^-_t$ from $\mathbf{x}^-_t$.

In Fast PCD (FCPD) (Tieleman & Hinton, 2009), improvements over PCD are obtained by decoupling the parameters used in the positive and negative phases. One maintains two sets of parameters, the ("slow") model parameters θ and "fast" parameters θ_F, and we use their sum $\theta^- = \theta + \theta_F$ for the negative phase Markov chain stochastic updates. These fast parameters are updated from the same PCD gradient estimator $\hat{g}$, but with a larger learning rate ϵ_F for θ_F, and θ_F is strongly decayed towards 0 with a kind of "weight decay",

$$\theta_F \leftarrow \alpha \theta_F + \epsilon_F \hat{g},$$

with $\alpha = 0.95$ being a good standard choice, according to Tieleman and Hinton (2009). FPCD is equivalent to PCD when $\epsilon_F = 0$.

In Herding (Welling, 2009b; Welling, 2009a), the MRF is taken to its 0-temperature limit (i.e. $P(\mathbf{x}, \mathbf{h}) \propto \exp(-\text{Energy}(\mathbf{x}, \mathbf{h})/T)$ with $T \to 0$) so that we proceed like in PCD but sampling is replaced by deterministic updates, e.g.,

$$h_j{}^- \mid \mathbf{x}^- = \text{argmax}_\text{v} \text{P}(\text{h}_\text{j} = \text{v}|\mathbf{x}^-). \tag{4}$$

In addition, the update is performed with a learning rate of 1 (Welling (2009a) argues that changing the learning rate only changes the scale of the resulting dynamical system). The equivalent of convergence of the MCMC is replaced by exact minimization of the energy, and is approximated by a heuristic hill-climbing procedure, alternatively updating $\mathbf{h}^-$ as per eq. 4 and then $\mathbf{x}^-$ deterministically as per

$$x_i{}^- \mid \mathbf{h}^- = \mathrm{argmax_v} \mathrm{P}(\mathrm{x_i} = \mathrm{v} | \mathbf{h}^-). \tag{5}$$

That procedure is guaranteed to converge to a fixed point and Herding thus defines a deterministic dynamical system.

3 The Rates FPCD Sampler

Both FPCD and Herding introduce new ideas for sampling, albeit they do so differently. FPCD can be understood by observing that the mixing of a chain which is periodically interrupted by updates, as would be the case during learning, is better than that of a model whose parameters are unchanging. Fast parameters allow us to make sure that the chain that we sample from always changes in a way that promotes mixing, while the true parameters may change more slowly in order to satisfy training convergence criteria. Although the distribution for the negative phase is not the same as the one defined by the RBM as a model, the strategy works. Herding, on the other hand, cannot work at all with a point estimate of parameters because all chains converge deterministically to a fixed point. The model may only be defined through a dynamical system involving a trajectory over parameters as well as over states. Yet, that system was shown to define a distribution whose sufficient statistics[1] match those of the training set. Together, the ideas put forward by FPCD and Herding lead us to ponder if a dynamical system could be defined whose distribution would closely match that of a point estimate of an RBM's parameters.

The "samples" obtained from the original Herding algorithm are those $\mathbf{x}^-$ visited by the negative phase Markov chain along the trajectory of the Herding dynamical system while it "learns". Note how sampling using that system requires keeping around (and using) the training set. Welling (2009a) proposes an approximation based on a set of "rates", which are averages of the positive phase sufficient statistics. The Rates-Herding samples are then obtained by replacing the positive phase contributions (which depend on the training examples and on the current parameters) by these rates statistics.

We may observe that there is nothing in the rates idea which is specific to Herding. By combining the rates idea as well as the fast weights of FPCD, we propose a new sampling algorithm that we call the Rates-FPCD sampler. Here, a variant of the FPCD algorithm is used but purely for sampling, and the positive phase contributions are changed to relevant aggregate statistics on the training set, instead of requiring training set examples themselves. Furthermore, since this is a sampling algorithm, it may be used to sample from any RBM trained with any method: the slow parameters θ are thus initialized to the parameters of an already trained model and are kept unchanged, while the fast parameters θ_F are updated in the sampling procedure. Algorithm `RatesFPCDSample` gives the pseudo-code.

[1] the sufficient statistics of the MRF, i.e., averages of $\mathbf{x}$, $\mathbf{h}$, and $\mathbf{x}'\mathbf{h}$.

`RatesFPCDSample`$(\theta, \theta_F, k, \epsilon_F, \alpha, \tilde{\theta}, \mathbf{x}^-)$
Sample from an RBM with parameters $\theta = (W, b, c)$ (trained beforehand using CD, PCD, FPCD or Herding), performing k Gibbs steps between each FPCD update with negative phase learning rate ϵ_F and weight decay α. Uses the rates $\tilde{\theta} = (\tilde{W}, \tilde{b}, \tilde{c})$ with $\tilde{W} = E[\mathbf{x}'\mathbf{h}]$, $\tilde{b} = E[\mathbf{h}]$, and $\tilde{c} = E[\mathbf{x}]$ estimated with $(\mathbf{x}, \mathbf{h})$ pairs with $\mathbf{x}$ from the training set and $\mathbf{h} \sim P(\mathbf{h}|\mathbf{x}; \theta)$. The caller must also remember the previous state in the chain, i.e. the previous sample $\mathbf{x}^-$ (in our experiments, $\mathbf{x}_0^-$ is initialized to a training example) and the previous fast parameters $\theta_F = (W_F, b_F, c_F)$, both of which are updated here.

for i=1 to k **do**
  $\mathbf{h}^- \sim P(\mathbf{h}|\mathbf{x}^-; \theta + \theta_F)$
  $\mathbf{x}^- \sim P(\mathbf{x}|\mathbf{h}^-; \theta + \theta_F)$
end for
$W_F \leftarrow \alpha W_F + \epsilon_F(\tilde{W} - \mathbf{x}^{-\prime}\mathbf{h}^-)$
$b_F \leftarrow \alpha b_F + \epsilon_F(\tilde{b} - \mathbf{h}^-)$
$c_F \leftarrow \alpha c_F + \epsilon_F(\tilde{c} - \mathbf{x}^-)$
return $\mathbf{x}^-$

Note that the rates-based sampling algorithm for Herding of Welling (2009a) (called here Rates-Herding) is a special case of this algorithm where $\alpha = 1$ (but we could also try it with $\alpha < 1$) and instead of a fixed number of MCMC steps k, we usually allow the deterministic updates to converge to a fixed point (which takes less than 15 steps on average in all of our experiments, but this depends on the dataset).

4 Temporarily Unlearning the Current State to Escape Modes

Let us focus for now only on the sampling procedure, and consider a generic MCMC with transition probability matrix A, i.e., the state s_t at step t in the chain is obtained from the state s_{t-1} at step $t-1$ by sampling from the multinomial $P(s_t|s_{t-1} = j)$ with $P(s_t = i|s_{t-1} = j) = A_{ij}$. If we sample s_0 from distribution p_0, then the marginal distribution at step t is $p_t = A^t p_0$, and the asymptotic distribution (if the chain converges)

$$p_\infty = A p_\infty, \tag{6}$$

the eigenvector of A with eigenvalue 1.

Now let us say we want to speed-up mixing by escaping more rapidly from modes of p_t, i.e., if we want faster convergence and faster coverage of the different modes of the asymptotic distribution. A reasonable idea is to change A so that it does not return to states it visited recently (and in particular, the last one visited). To reduce the probability of staying in the same state, imagine that we remove probability mass λ from $P(s_t = i|s_{t-1} = i)$ and redistribute it to other values of s_t. This gives rise to a

new stochastic matrix

$$\tilde{A} = \frac{A - \lambda I}{1 - \lambda} \tag{7}$$

Note that we must have $\lambda \leq \min_i A_{ii}$. It is then interesting to observe that the new chain based on $\tilde{A}$ converges to the same asymptotic distribution as A, since

$$\tilde{A} p_\infty = \frac{A p_\infty - \lambda p_\infty}{1 - \lambda} = p_\infty. \tag{8}$$

As first noted in Tieleman and Hinton (2009), FPCD probably works well because the negative phase samples temporarily change the energy landscape so as to decrease the probability of those very configurations that were sampled in the chain, increasing the chances of quickly escaping a mode as soon as it is visited. We surmise that the fast mixing observed with Herding (see section 6.2 for empirical evidence) also occurs for the same reason: changing the parameters while mixing so that the configurations just visited in the chain get a smaller probability. We call that effect an **unlearning** effect because we want to (at least temporarily) unlearn to generate the configurations just visited in the chain. Note however that in both cases of FPCD and Herding, and unlike in the above theoretical $\tilde{A}$ chain, the changes do not only affect the sampled configuration (nor just the probability of staying in it given one is in it), but other configurations and other conditional probabilities as well, through the intricate parametrization of the stochastic matrix A induced by the RBM structure combined with block Gibbs sampling. We do not want to eliminate this parametrization because it is also what gives rise to generalization in the model. So FPCD and Herding are at least approximately trying to maintain the same asymptotic distribution while achieving faster mixing (as we will see experimentally). This approximation may come at a price, which is observed by visualizing samples: although mixing is faster, "spurious" samples that typically are intermediate configurations between the modes visited may occur at higher rates (although rather rarely in practice in our experiments).

With this perspective, it can be seen that FPCD brings something more than Herding (and PCD with a large learning rate, for that matter): it acts a bit more conservatively by making the effect of unlearning parameters *temporary*, with an exponentially decaying factor (α in `RatesFPCDSample()`). Basically, the (Rates-)FPCD update does not affect the model, only the chain, and the effect of each negative sample $\mathbf{x}^-$ disappears exponentially fast. It is not clear, however, that this effect is necessary in order for fidelity to the distribution to be preserved, because for a well-trained model, the expected value of the negative phase contributions should be equal or very close to the rates (this is in fact our learning criterion) and thus the expected value of the fast parameters would be zero.

It is also interesting to consider yet another sampling algorithm where the rates are effectively set to zero (which we will call the Sample Penalization sampling algorithm). This draws a closer parallel to the example with $\tilde{A}$ where we simply penalize the transition of a state to itself. Assuming that a sufficient weight decay is applied to the fast parameters θ_F, the effective parameters θ^- for the negative phase and thus the distribution that we effectively sample from might stay sufficiently close to the original while improving mixing to prove useful. However, note that unlike Rates-FPCD, Sam-

ple Penalization creates a distortion of the parameters that is not zero-mean, favoring smaller weights, or equivalently higher temperature or more noise in the chain.

5 Indirect Sampling Likelihood for Estimating the Quality of Samples from a Purely Generative Model

While an RBM clearly defines a probability distribution over all state vectors, the cost of evaluating objectively how good that distribution is, typically by evaluating the log-likelihood of a set of test examples under that distribution, is usually prohibitive if not impossible. In addition to this, the Herding algorithm confounds the sampling procedure with the "model", and in fact the only way we know to define the "model" is through the distribution of samples obtained. Finally, we would like ways to assess how well different *sampling* methods work, for the same underlying trained model. We can visualize a sequence of samples to get a qualitative picture of the quality of the samples (i.e. the model) and of the mixing (i.e. how fast we cover the main modes of the distribution), but this is subjective and probably unfit for model selection. Indeed, visualization of the samples can be biased towards generative models that overfit, since simply spewing out the training examples would be visually pleasing. To evaluate quantitatively the quality of a generative model's distribution as well as the quality of the sampling procedure used, we propose a new method, called *Indirect Sampling Likelihood* (ISL), that basically aims to compare the generated samples to the examples in a test set. Assume that our generative model has been trained with a training set T. The basic idea is the following:

1. Generate samples S from our trained generative model.
2. Use either S (generated samples only) or $S \cup T$ (training plus generated samples) to train a density model $\mathcal{P}$ for which the likelihood can be computed tractably.
3. Compute and return the likelihood of the test set under $\mathcal{P}$.

In practice, we chose to use a non-parametric kernel density estimator for $\mathcal{P}$, which can be seen as a mixture model with one component per training example $\mathbf{x}_i$, and a hyper-parameter that controls how much probability mass is transferred from $\mathbf{x}_i$ to some set of neighbors. Because our experiments are with d-dimensional binary vectors, our kernel probability (i.e. each component of the mixture) is a factorized Bernoulli:

$$\mathcal{P}(\mathbf{y}) = \frac{1}{N} \sum_{i=1}^{N} \prod_{j=1}^{d} \beta^{1_{\mathbf{y}_j = \mathbf{x}_{ij}}} (1-\beta)^{1_{\mathbf{y}_j \neq \mathbf{x}_{ij}}} \tag{9}$$

where N is the number of generated samples. When the hyper-parameter β is 1, $\mathcal{P}$ is the empirical distribution associated with the samples $\{\mathbf{x}_i\}$. Decreasing β smooths the distribution, and when $\beta = \frac{1}{2}$, the distribution is so smoothed out as to obtain the uniform distribution over binary vectors. β is chosen (from a grid) to optimize likelihood on the generated samples. Since $\mathcal{P}(\mathbf{y})$ is a summation, it is easy to accumulate

terms for each $\mathbf{y}$ as samples are generated and thus we can compute a precise plot of the average of $\log \mathcal{P}(\mathbf{y})$ versus the number of samples $|S|$ at little extra cost.

An intuitive way to understand ISL when $\mathcal{P}$ is a local non-parametric model as above is that it provides a measure of **coverage** of the test set. For each test example that is "far" from the generated samples, we will pay a fixed high price (roughly proportional to $-d \log(1-\beta)$ here). Thus ISL is a coherent way to "count" the number of test examples that are "near" (or "far") from the generated samples. Note that as long as N is smaller than the number of training examples for the generative model, we would not expect the ISL with generated samples only to be better than the likelihood obtained by training $\mathcal{P}$ with T only (which is what a grossly overfitting generative model would achieve), since T is already coming from the right distribution, whereas S is not. Hence when using generated samples only, and S is small, overfitting models would be favored. However, with training plus generated samples, they would not be. On the other hand, with generated samples only, we get a clearer picture of the quality of the **mixing** process when looking at the curve of ISL in terms of the number of generated samples. Indeed, when $|S|$ is small, if the generative model does not mix well, then the ISL (with generated samples only) will raise much slower due to poor mixing, because many test examples will end up far from any of the generated samples. In fact, we find that the slope of ISL in terms of $|S|$ is a good indicator of the quality of mixing (otherwise evaluated by visual inspection of the samples). When $|S|$ is large, the curve's asymptote would indicate the overall quality of the generative model.

6 Experimental Results

We used two datasets for which it made sense to binarize the data and for which visualizing samples helped us understand qualitatively the properties and quality of the learned model and the sampling procedure: the USPS digits images and the MNIST digit images. The USPS dataset has 16×16 pixel digit images. 6291 are used for training, 1000 for hyper-parameter selection, and 2007 for testing. The MNIST dataset has 28×28 pixel images; 50000 for training, 10000 for hyper-parameter selection, and 10000 for testing (only 2000 were used to generate the ISL curves to reduce computation time).

6.1 Evaluating ISL

In order to evaluate ISL, we compared the log-likelihood assigned to the test set by density estimation using the samples from several models to its log-likelihood under the model's true underlying distribution as calculated analytically. That analytic calculation was made possible by using a very small number of hidden units (16). We found ISL to reflect the analytic log-likelihood rather accurately in the sense that although it did not yield the same values, both systems tended to order models similarly. Fig. 1 shows both analytic and sampling-based negative log-likelihood (NLL) obtained for several models. Models trained by Herding are not shown since there is no way to compute an analytic and meaningful log-likelihood for them (a likelihood would correspond to ISL as $|S| \to \infty$). The sampling was performed using Rates-Herding with the

same learning rate of 10^{-2} (this may not be optimal). The analytic NLL is in general smaller, but it is difficult to assess the relevance of this fact because the true probability distribution of the models may not be directly comparable with the Parzen estimator we use for the number of samples we use. What is more relevant is that the general aspect and order of the curves seem to be preserved and this indicates that ISL might be a fair way to compare models (at least for a small number of units). There is one obvious exception, which are CD-trained models: their analytic NLL is vastly worse than that of other models, but they generate samples that are comparatively similar to those of other models. Note that ISL depends on both the model and the sampling process. So the discrepancy may be explained by the fact that CD training generates negative samples that are always close to training examples and thus modes that are very far from the training samples may incidentally arise and never be penalized. These modes might be extremely difficult to attain from a starting point that is within the training set and thus the distribution, seen through iterative sampling, might look better than it actually is. Worth noting is that this phenomenon does not seem to occur when training with -1/1 units.

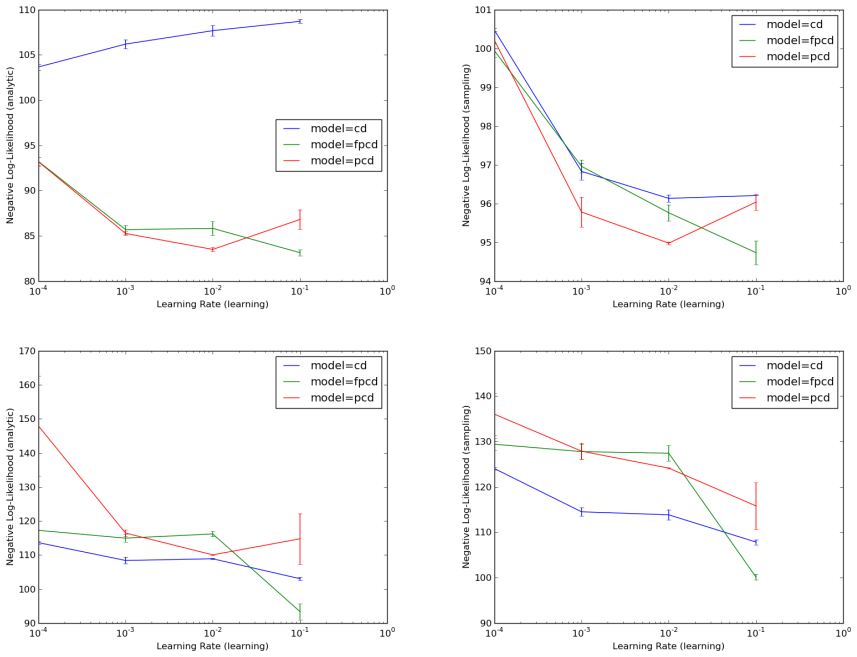

Figure 1: Top left: analytic NLL of USPS's test set for various models trained with CD, PCD and FPCD with 0/1 units, with the learning rate on the x-axis. Top right: the NLL obtained with ISL for the same models. Bottom left: analytic NLL for equivalent models, using -1/1 units. Bottom right: NLL obtained with ISL for the same models. Smaller is better.

Fig. 2 shows sequences of samples obtained from a single model, initialized to the same training sample, but using different sampling methods. Visually speaking, the log-likelihood of the test set under a density defined by these samples seems to be a fair assessment of the mixing capabilities of the models shown, with Rates-Herding a distant first and Gibbs a distant last.

6.2 Comparing the Mixing Achieved with Different Samplers

Four classes of sampling methods were evaluated: Rates-FPCD, Rates-Herding, Sample Penalization and Gibbs sampling, the latter of which is the standard sampling procedure for RBMs. When applicable, the methods are parametrized by a learning rate ϵ_F, a fast weight decay rate α and the number k of Gibbs iterations between each sample. These hyper-parameters were selected to maximize ISL, to compare each sampling method in its best light. To focus on the effect of the sampling method, they are all tested on the same best model obtained (which was obtained by FPCD training). Fig. 3 plots the ISL with generated samples only of the USPS test set for these sampling methods, in function of the number of generated samples. It is readily apparent that Gibbs sampling performs very poorly (note that it often performs *much* worse). By correlating the curve's "bumps" with visual inspection of samples, we find that they indicate the times at which the chain suddenly switches from a mode to the next. In contrast, the curves for Rates-FPCD and Rates-Herding are quite smooth and well-mixing. Furthermore, the performance of the training set itself is eventually surpassed by a slight margin (we contend that the margin would become wider if more samples were drawn): -78.5 (Rates-FPCD) versus -80.0 (train set). Note how Rates-Herding starts out with the fastest rise in ISL (faster mixing) but is quickly surpassed by Rates-FPCD. As discussed below this may be related to a difference in optimal ϵ_F.

The optimal learning rate for the Rates-FPCD method, in the long run, was observed to be lower than the optimal learning rate for the Rates-Herding method by a factor of 10 to 100 in most cases (note: both in figures 2 and 3, the learning rate is lower for Rates-FPCD, which partly explains its apparently poorer mixing). In any case, as one could expect, the mixing rate increases as the learning rate increases, but so does the frequency of spurious samples. It seems that Rates-Herding is more robust to larger learning rates, but that advantage subsides as more samples are drawn and rapid mixing is less of a concern. A tentative explanation for this robustness is that Herding provides a halting criterion for the number of iterations k between each produced sample: the chain is run until a fixed point is reached. Presumably, this allows Herding to loop exactly as many times as needed in order to guarantee that only low-energy samples will be produced. In the long run, however, Rates-FPCD tends to perform better. This is most observable on MNIST where Rates-FPCD, for a comparable k, scores significantly higher, which suggests that there may be some inherent limitations to Rates-Herding.

Another interesting observation is that the optimal value for the decay rate α for sampling purposes is 1, meaning that no decay is applied at all and the fast parameters may roam freely. To apply a decay of 0.95 (see Fig. 4, left) yielded systemically and significantly worse results, although it is worth noting that such is not the case during training ($\alpha < 1$ works better in that case).

The Sample Penalization method systemically performed better than Gibbs, but still significantly worse than the other methods. It can be seen on fig. 2 that its good mixing is mitigated by a skewing in the distribution: the samples produced by this method, while recognizable, appear different from the true distribution (they seem larger and bolder in general and not only on the figure shown). This is unsurprising since the expectation of the fast parameters under this method is not zero, so on average the effective parameters used for sampling will deviate of a certain quantity to which our experiments suggest the model's distribution is not invariant. It is nonetheless interesting to see that the method does work to a limited extent.

Interestingly, some sampling runs on models trained on the MNIST dataset produce slightly superior density estimators to that produced by the training set itself, for up to 35,000 samples (which is a significant amount). While this result may be surprising, it is not impossible. If only one sample may be used to make a Parzen density estimator for a given distribution, that distribution's mean would most probably yield better results than any individual sample, even if there is no probability mass on that mean. If our sampling method produces "prototypical" samples, it may therefore perform better even for a large number of samples. However, we would expect its advantage to diminish as the number of samples increases - and indeed, we observe this, all of our sampling models performing worse than the full training set for an equal number of samples. Nonetheless, as can also be seen in fig. 2, these "prototypical" samples seem to cover an impressive amount of variability in the dataset. As more samples are drawn, the performance of the training set is once again surpassed: with twice as many samples as the training set, we observe an improvement of up to 2.2% in estimated log-likelihood.

The sampling for the best models shown is rather slow (20 hours for 100,000 samples on MNIST), which is to be expected due to their size (1000 hidden units) and the high k used while sampling. Using a small k, while it degrades the sampling quality, does not degrade it significantly, can be much faster and does not seem to hinder model selection when used for that purpose in conjunction with ISL. Furthermore, we may consider parallel sampling by running several chains with different initializations, so we believe that a model's quality may be evaluated very efficiently using our method.

6.3 Training with Herding

Using ISL, we may evaluate quantitatively the quality of the samples produced by models trained by Herding. This allows us to assess Herding's sensitivity to various hyper-parameters and compare it to other learning methods. As proved in Welling (2009a), Herding is invariant to a scaling of λ of its parameters and learning rate. It is however not invariant to the relative scaling of these two parameters. Fig. 4 (right) shows Herding's performance as a function of learning rate, when the weights are initialized between -1 and 1. Interestingly, the optimal learning rate is not 1, as previously suggested. 0/1 units are particularly sensitive and perform very badly unless the learning rate is small. -1/1 units, on the other hand, are much less sensitive and it is worth noting that they perform systemically better, although we observe that the gap becomes narrower as the number of hidden units is increased. These findings are consistent with Welling (2009a) on the same dataset (USPS), who had no luck making

the model work on 0/1 units with $\epsilon = 1$. The same type of learning rate sensitivity is observed on MNIST, with different minima. Both types of units on MNIST also behave more similarly than they do on USPS and work poorly with large learning rates.

7 Conclusion and Future Work

In this paper, we set out to evaluate several sampling methods over RBM models whose objective is to mix well: Rates-Herding and Rates-FPCD, both inspired from Welling (2009a), and Sample Penalization, a novel method. In order to do this, we defined the ISL framework based on Parzen density estimation. We showed that within this framework, Rates-Herding and Rates-FPCD produced stable density estimators for several models which correlated with their true log-likelihood. We also showed these methods to be significantly better than Gibbs sampling, both qualitatively and quantitatively. Furthermore, in certain cases we managed to produce sequences of samples which performed better than the whole training set. We used ISL to evaluate models trained using Herding (a task for which no other quantitative methods exist to our knowledge) and found it sensitive to the learning rate hyper-parameter.

Future work on this topic would involve generalizing our methods to continuous units (which is non-trivial in the case of Herding), comparing them to established density estimators such as AIS, evaluating how well our measurements on a model correlate with that model's quality as initialization to a deep supervised network, and investigating prospective uses of samples as ancillary data sources for classification tasks.

References

Hinton, G. E. (1999). Products of experts. *Proceedings of the Ninth International Conference on Artificial Neural Networks (ICANN)* (pp. 1–6). Edinburgh, Scotland: IEE.

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

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

Hinton, G. E., Sejnowski, T. J., & Ackley, D. H. (1984). *Boltzmann machines: Constraint satisfaction networks that learn* (Technical Report TR-CMU-CS-84-119). Carnegie-Mellon University, Dept. of Computer Science.

Tieleman, T. (2008). Training restricted boltzmann machines using approximations to the likelihood gradient. *Proc. ICML 2008* (pp. 1064–1071). Helsinki, Finland.

Tieleman, T., & Hinton, G. (2009). Using fast weights to improve persistent contrastive divergence. *Proc. ICML 2009* (pp. 1033–1040). Montreal, Quebec, Canada.

Welling, M. (2009a). Herding dynamic weights for partially observed random field models. *Proceedings of the 25th Conference in Uncertainty in Artificial Intelligence (UAI'09)*. Morgan Kaufmann.

Welling, M. (2009b). Herding dynamic weights to learn. *Proc. ICML 2009*.

Younes, L. (1998). On the convergence of markovian stochastic algorithms with rapidly decreasing ergodicity rates. *Stochastics and Stochastics Models* (pp. 177–228).

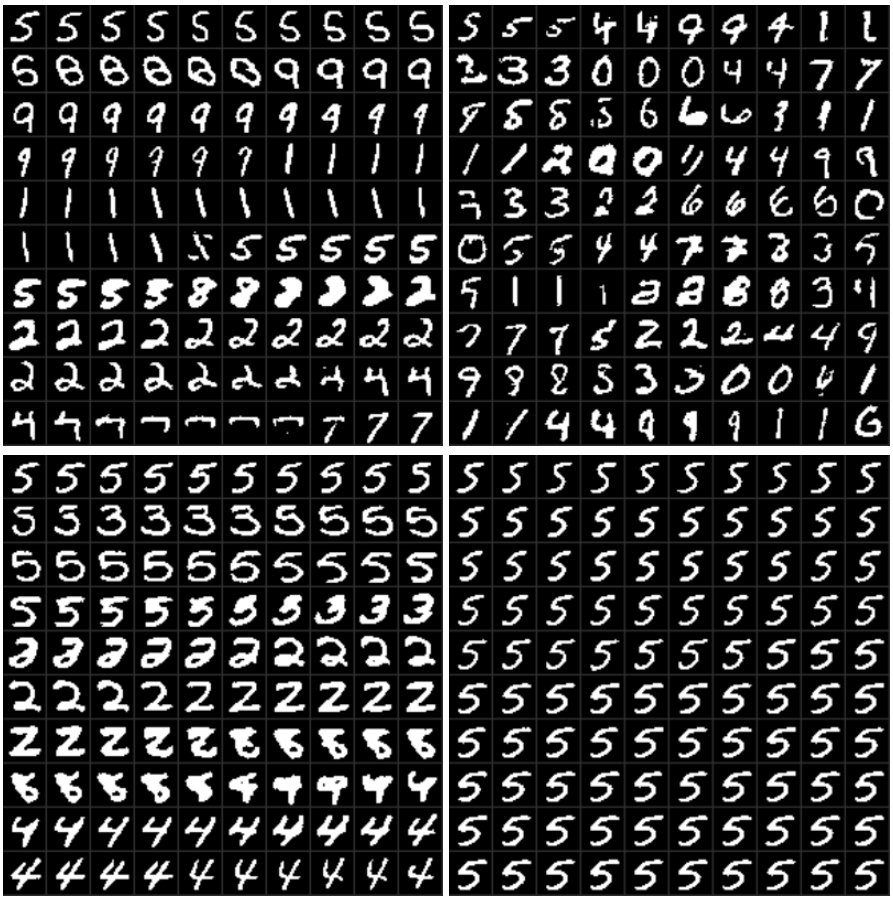

Figure 2: First 100 samples obtained on an FPCD-trained model using various sampling procedures, all initialized using the same training sample: Rates-FPCD (top left), Rates-Herding (top right), Sample Penalization (bottom left) or Gibbs (bottom right). Training was on MNIST digits. The ISL (samples only, using these 100 samples) values for the test set are coherent with this visual assessment: -290.09 for Rates-FPCD, -257.56 for Rates-Herding, -379.53 for Sample Penalization, -396.78 for Gibbs and -255.90 for the first 100 samples of the training set (not shown). For all models, $k = 15$ while sampling.

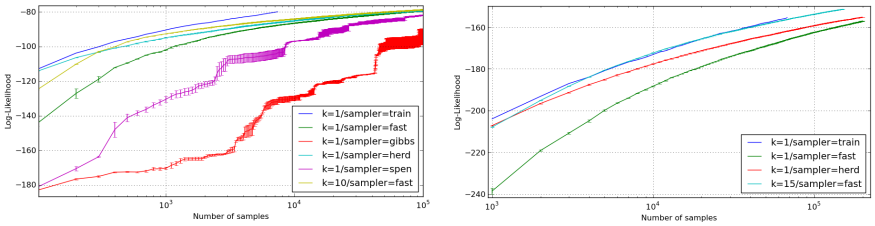

Figure 3: Top: Comparison of different sampling schemes in terms of their sample-based coverage log-likelihood (ISL) on USPS, with respect to number of samples generated. All schemes were applied to the same best model found. The sampling schemes compared are, in order of decreasing final performance: rates-FPCD with k=10 (yellow), rates-Herding (cyan), rates-FPCD with k=1 (green), the 7291 samples of the training and validation sets (blue), rates-FPCD with only negative terms (magenta), Gibbs (red). Bottom: Same, but on MNIST: rates-FPCD with k=15 (cyan), rates-Herding (red), rates-FPCD with k=1 (green) and the 68,000 samples of the training and validation sets (blue). Gibbs and Sample Penalization performed too poorly to be visible in the window shown. Note: the effective k for rates-Herding is, on average, 8.5 (top) and 14.5 (bottom).

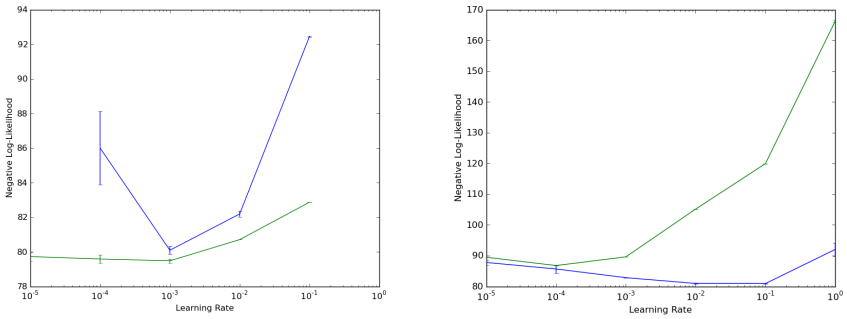

Figure 4: Left: Comparison of ISL for Rates-FPCD sampling with (top, blue, $\alpha = 0.95$) or without (bottom, green, $\alpha = 1$) θ_F weight decay, with different learning rates. The sampling is applied on the best model found (an FPCD-trained model). Right: Effect of learning rate on ISL for USPS dataset, for Herding models with 0/1 units (top, green) and -1/1 units (bottom, blue). Weight parameters were initialized uniformly between -1 and 1 in all cases.