Quickly Generating Representative Samples from an RBM-Derived Process

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

Neural Computation, to appear, 2011

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 justify such approaches as ways to escape modes while approximately keeping the same asymptotic distribution of the Markov chain. In that spirit, we extend FPCD using an idea borrowed from Herding in order to obtain a pure sampling algorithm which we call the Rates-FPCD sampler. Interestingly, this sampler can *improve the model* as we collect more samples, since it optimizes a lower bound on the log-likelihood of the training data. We provide empirical evidence that this new algorithm displays substantially better and more robust mixing than Gibbs sampling.

1 Introduction

Because the exact gradient of their parameters with respect to log-likelihood cannot be computed tractably, training undirected graphical models such as Markov random fields (MRF) and variants of Boltzmann machines requires extensive use of samples taken from the model's underlying distribution. This has led to various sampling schemes for use during training. 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.

Unfortunately, sampling from an RBM's distribution is not an easy task either. While Gibbs sampling lends itself naturally to sampling in an RBM and is guaranteed to produce samples from the model at equilibrium, it is easy to observe in experiments that the time to equilibrium is considerably larger than what would be practical. And so, while the standard sampling method for RBMs is to run a Gibbs chain, it is never actually run to equilibrium but for the most trivial of cases.

Several algorithms have been successful at improving the quality of sampling, and have mostly done so by running the sampling process in parallel to learning. For instance, Stochastic Maximum Likelihood (SML) (Younes, 1998) (also known under the name Persistent Contrastive Divergence (PCD) Tieleman (2008)) keeps the current sample as an initialization to the Gibbs chain in the next training iteration, even though the parameters have changed in-between. More recent work has gone even further: the Fast Persistent Contrastive Divergence (FPCD) algorithm (Tieleman & Hinton, 2009) keeps a second set of parameters (updated with a stronger gradient but decayed to zero) which

are added to the true parameters only for sampling part of the learning algorithm. The Herding algorithm (Welling, 2009b; Welling, 2009a) outright gives up on getting a point estimate of the parameters and instead simply aims to produce samples like those of the training set through a dynamical system - it can be shown that the system will produce samples with the sought after sufficient statistics under a condition that may be checked easily, (sub-linear growth of its parameters) in spite of the fact that its parameters never converge. Interestingly, the distribution of the generated samples may not be expressed at a given time frame as a simple function of the parameters like in a Boltzmann Machine. Instead, it is defined procedurally, by the dynamics of the generating procedure. However, what matters is that the statistics of the generated samples can be made to match some target statistics (derived from the training data), even though the parameters keep changing.

It is worth noting that while much work in the literature focuses on sampling during training, sampling could be legitimately seen as an end in itself - as a creative task, for instance. In this case, what we want are samples that *mix* quickly and ideally do not require keeping the training set around. By "mixing", we mean the rate at which the distribution of a stream of samples (at some finite point t in the stream) converges to its asymptotic distribution - in other words, how quickly can we expect it to cover its major modes? Gibbs sampling, which is customarily used for this purpose, mixes very poorly, getting stuck in the same modes for thousands of iterations. Borrowing ideas from FPCD and Herding, we obtain a new sampling algorithm, Rates-FPCD, which mixes better by a considerable margin. Like herding, it changes the parameters as it is producing samples, but in such a way that they hover around an optimal value corresponding to a lower bound of the likelihood of the training data.

Although Rates-FPCD is meant to be a sampling algorithm rather than a training algorithm, and that we initialize its parameters to those of a previously trained RBM, one should not mistake the technique for an unbiased scheme in order to sample already trained RBMs. In fact, the metric we use in order to measure the quality of samples does not allow us to directly measure deviations from the original model. What we attempt to measure is how "close" the samples we produce are to the samples from a test set. Therefore, any sampling method which modifies the model in order to make it generalize better would score better, even though in the process it might deviate from the original RBM. While we believe that in practice, provided that the model was trained properly, the deviations are small, we know that they occur. Nonetheless, the bottom line is that the samples score much better than the ones obtained with other methods, which we consider to be an improvement. In section 4, we give insight as to how Rates-FPCD typically ends up improving the original model, what kind of deviation one may expect, and its convergence properties.

2 RBMs, CD, SML, 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 $\mathbf{s} = (\mathbf{x}, \mathbf{h})$:

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

where parameters are $\theta = (\mathbf{W}, \mathbf{b}, \mathbf{c})$. 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 $(\mathbf{x}^-, \mathbf{h}^-)$ (called negative samples) from the model's distribution. From one such sample an estimator $\hat{g}$ of the gradient g can be computed and an update of the model's parameters performed by stochastic gradient descent $\theta \leftarrow \theta + \epsilon \hat{g}$ as follows:

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

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

The difference between CD (for Contrastive Divergence), SML (for Stochastic Maximum Likelihood, also known as Persistent CD or PCD), FPCD (for fast PCD) and Herding (and thus their respective contributions) lies in how the negative samples are obtained.

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 $\mathbf{x}$. The CD update is a biased estimator of the log-likelihood gradient (Bengio & Delalleau, 2009). In addition, the fact that negative samples often lie in the immediate vicinity of the positive samples, can not only yield spurious modes elsewhere but can also create energy barriers (Desjardins et al., 2010) around the training examples. Gibbs sampling typically works very poorly as soon as it ventures outside of the wells corresponding to training examples. This is particularly evident when starting the Gibbs chain from a random state rather than from a real data point.

To address this problem and obtain models with better likelihood, the Stochastic Maximum Likelihood (SML) training algorithm 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. For each training example or minibatch, we run one or more iterations of a block Gibbs chain initialized at $\mathbf{h}^-_{t-1}$ to conditionally sample $\mathbf{x}^-_t$ from, and then $\mathbf{h}^-_t$ from $\mathbf{x}^-_t$. The interaction between the positive phase contributions and the negative phase contributions is only through the weights.

In Fast PCD (FCPD) (Tieleman & Hinton, 2009), improvements over SML 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 SML gradient estimator $\hat{g}$, but with a larger step size ϵ_F for θ_F, and θ_F is strongly decayed towards 0 with a kind of "weight decay",

$$\theta_F \leftarrow \alpha \theta_F + \epsilon_F \hat{g}, \tag{4}$$

with $\alpha = 0.95$ being a good standard choice, according to experiments in Tieleman and Hinton (2009). FPCD is equivalent to SML 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 SML but sampling is

replaced by deterministic updates, e.g.,

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

In addition, the update is performed with a step size of 1 (Welling (2009a) argues that changing the step size only changes the scale of the resulting dynamical system, though we have observed that it also changes mixing rate). The equivalent of convergence of the MCMC is replaced by exact maximization of probability, and is approximated by a heuristic hill-climbing procedure, alternatively updating $\mathbf{h}^-$ as per eq. 5 and then $\mathbf{x}^-$ deterministically as per

$$x_i{}^- \mid \mathbf{h}^- = \mathrm{argmax_v P(x_i = v|\mathbf{h}^-)}. \tag{6}$$

That procedure is guaranteed to converge to a fixed point and Herding thus defines a deterministic dynamical system. Pseudo-code for it is given at the beginning of `RatesHerdingSample`.

3 Temporarily Unlearning the Current State to Escape Modes

Before going any further and in order to better understand the proposed method, it is useful to lay down some principles through which the mixing of an MCMC may be improved.

Let us focus for now only on sampling procedures, and consider a generic MCMC with transition probability matrix $\mathbf{A}$, 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 $\mathbf{p}_0$, then the marginal distribution at step t is $\mathbf{p}_t = \mathbf{A}^t\mathbf{p}_0$, and the asymptotic distribution (if the chain converges)

$$\mathbf{p}_\infty = \mathbf{A}\mathbf{p}_\infty, \tag{7}$$

the eigenvector of $\mathbf{A}$ with eigenvalue 1.

Now let us say we want to speed-up mixing by escaping more rapidly from modes of $\mathbf{p}_t$, i.e., we want faster convergence and faster coverage of the different modes of the asymptotic distribution. A reasonable idea is to change $\mathbf{A}$ so that the chain 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, suppose that we remove probability mass λ from $P(s_t = i|s_{t-1} = i)$ ($\lambda \leq \min_i A_{ii}$, $\lambda < 1$) and redistribute it to other values of s_t. This gives rise to a new stochastic matrix

$$\tilde{\mathbf{A}} = \frac{\mathbf{A} - \lambda\mathbf{I}}{1 - \lambda}. \tag{8}$$

It is then interesting to observe that the new chain based on $\tilde{\mathbf{A}}$ converges to the same asymptotic distribution as $\mathbf{A}$, since

$$\tilde{\mathbf{A}}\mathbf{p}_\infty = \frac{\mathbf{A}\mathbf{p}_\infty - \lambda\mathbf{p}_\infty}{1 - \lambda} = \mathbf{p}_\infty. \tag{9}$$

Obviously, this trick is not directly applicable to an RBM (or any model of appreciable complexity), since the transition matrix $\mathbf{A}$ is never handled directly. It is however quite similar to the effect stochastic gradient updates such as those in eq. 3 would have on sampling a persistent Gibbs chain: since the effect of the negative phase contribution to the gradient is effectively to remove a factor from the parameters equal to the corresponding sufficient statistics of the negative sample, that sample is less likely to be generated in the next iteration. Eventually, through reinforcement by positive phase samples which come from the training set, negative samples that look like training examples will come back, but not before the chain was diverted away from them. In other words, a sampling procedure which "unlearns" what it yields is likely to have good mixing. Note that thanks to the intricate relationship between states represented by an RBM's parameterization, penalizing a sample

will also penalize other similar samples, forcing the chain to eventually jump from a major mode to another instead of simply cycling through closely related minor modes.

As SML, FPCD and Herding all use persistent chains, they all benefit from this effect to some extent. In SML's case, it is mitigated by the need to decay the step size in order to satisfy convergence criteria. As the step size is driven to zero, mixing properties of the persistent chain suffer from an insufficient "unlearning" effect, a problem that might even lead the model to worsen: the negative samples may cease moving, always penalizing the same region of the energy surface. FPCD addresses this concern by supplementing the sampling weights with an additional factor, whose step size may be independently adjusted. Herding, as it uses a large, constant step size and explicitly relies on the training dynamic to generate samples, is likely to reap significant benefits.

It is worth noting that FPCD seems to bring something more than Herding (and SML with a large step size, for that matter): it acts a bit more conservatively by making the effect of unlearning parameters *temporary*, with an exponential decay (with factor α in `RatesFPCDSample()`). This essentially reins in the sampling parameters so that they do not stray too far from the model's true parameters as the latter move at a slower, stable pace - or not at all. As it turns out, however, the decay is most useful during training, presumably early on, but in sampling with an already trained models as a starting point, we will show theoretically and experimentally that it can actually be a hindrance.

4 The Rates-FPCD Sampler

The "unlearning" concept put forth in the previous section leads us to ponder if a dynamical system similar to that of Herding, SML or FPCD during training, could be used to generate good samples starting from an already trained RBM – without the need to keep the full training set around. For this purpose, the strategy we adopted is the following: first, we compute sufficient statistics from the point estimate of the parameters θ found through some standard training method for an RBM (CD, SML, etc.). We call these quantities the *rates* $R = (R_w, R_b, R_c)$, which serve as the positive statistics to compute parameter updates:

$$\begin{aligned} \mathbf{x}^+ &\sim \text{data distribution } \hat{P}(\mathbf{x}) \\ \mathbf{h}^+ &\sim P(\mathbf{h}|\mathbf{x}^+;\theta) \\ R_{wij} &= E[\mathbf{h}_i^+ \mathbf{x}_j^+|\theta] \\ R_{bi} &= E[\mathbf{h}_i^+|\theta] \\ R_{cj} &= E[\mathbf{x}_j^+|\theta] \end{aligned} \tag{10}$$

These quantities summarize the $\mathbf{x}^+$ and $\mathbf{h}^+$ drawn from the joint of the empirical data distribution $\hat{P}(\mathbf{x})$ and the conditional distribution $P(\mathbf{h}|\mathbf{x};\theta)$ defined by the original model.

Then, we train a "sampling model", using the point estimate θ as the initial parameters, with a constant step size of reasonable magnitude. Each training update consists of the difference (scaled by the step size) between the rates (which are constant) and the sufficient statistics of the next sample, drawn from a persistent Gibbs chain. Each update will thus penalize the occurrence of samples similar to the last one, until the chain has had time to cycle through all of the distribution's major modes. The sampler's output is the sequence of samples from this chain. It is equivalent to training a fully observed Boltzmann machine on (hidden, visible) tuples, where hidden units are computed from the associated visible units using the mapping learned by an existing RBM. Algorithm `RatesFPCDSample` presents the Rates-FPCD sampling algorithm, where the step size is given by ϵ_F. Note that the updates are not applied directly to the original parameters - instead, we have parameters θ_F containing the difference, which adds some flexibility to the implementation.

In order to achieve the mixing properties we seek, it is necessary to train using a large enough constant step size. It can be shown that in the circumstance where a stochastic approximation algorithm

(SML/PCD) is run with a constant step size ϵ, the expected value of the distance of the current parameters to the local minimum is bounded and proportional to the square root of the step size (Borkar, 2008, 9.2.10). That is, $\limsup_{n\to\infty} E[||\theta_n - \theta^*||^2]^{\frac{1}{2}} \leq K\sqrt{\epsilon}$ (for some constant K). This gives us confidence that the parameters will hover around the local minimum encoded by the rates, as long as the step size is not excessive. This being said, taking inspiration from the FPCD algorithm, we also include an additional parameter, α, which rescales the deviation of the sampling parameters from the original parameters on every update. When $\alpha < 1$, this serves to explicitly constrain the movement of the parameters about their starting point.

`RatesFPCDSample(`θ`,` θ_F`,` α`,` ϵ_F`,` R`,` k`,` $\mathbf{x}^-$`)`
Sample from an RBM with parameters $\theta = (\mathbf{W}, \mathbf{b}, \mathbf{c})$, performing k Gibbs steps starting from an initial sample $\mathbf{x}^-$ in order to return a new sample. That new sample is typically used as the starting point for the next call to RatesFPCDSample. Differs from Gibbs sampling by the fact that there are fast parameters θ_F, which are updated using the precomputed rates $R = (R_w, R_b, R_c)$ with $R_w = E[\mathbf{x}'\mathbf{h}]$, $R_b = E[\mathbf{h}]$, and $R_c = E[\mathbf{x}]$. This is thus equivalent to Gibbs sampling when $\theta_F = \epsilon_F = 0$ (i.e. the updates are moot).

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
$\mathbf{W}_F \leftarrow \alpha\mathbf{W}_F + \epsilon_F(R_w - \mathbf{x}^{-\prime}\mathbf{h}^-)$
$\mathbf{b}_F \leftarrow \alpha\mathbf{b}_F + \epsilon_F(R_b - \mathbf{h}^-)$
$\mathbf{c}_F \leftarrow \alpha\mathbf{c}_F + \epsilon_F(R_c - \mathbf{x}^-)$
return $\mathbf{x}^-$

Since the application of the Rates-FPCD sampler on a pre-trained model will optimize it in order to reach the local maximum encoded by the rates parameters (that is, the sufficient statistics of the visible and hidden units), it is pertinent to ask how that model relates to the original. Note that if the training of the RBM had converged to a local maximum of the log-likelihood, then the parameters are already optimal with respect to the hidden variables. That is, whereas the original training aims to maximize the log-likelihood of the training set under various possible instantiations of the latent variables, the sampling process maximizes the same quantity under the instantiation defined by the original training (which is a convex optimization problem). Rates-FPCD ends up optimizing a lower bound on the log-likelihood and thus will work to *improve* the model, unless it had already converged to a local maximum. Should we want to avoid this improvement and generate samples from the original model, it is also possible to use the decay parameter α that we described previously. Setting $\alpha < 1$ guarantees that the parameters will not stray too far from their original values. However, preventing the movement of parameters hinders their optimization, meaning that while the distribution we sample from using decay might be closer to the original, it may produce samples that are further away from the training distribution.

The strategy described and justified here was originally suggested for Herding: observing that sampling requires keeping around (and using) the training set (unless there are no hidden units), (Welling, 2009a) proposes an approximation based on a set of "rates", which are averages of the positive phase sufficient statistics through training (in contrast to this, our rates are computed from applying a point estimate of the parameters on the full training data, but this is not very different). By combining the rates idea with the fast weights of FPCD, we propose a new sampling algorithm that we call the Rates-FPCD sampler (see `RatesFPCDSample`). In practice, this is FPCD with a step size of zero for the slow weights and the rates as the (constant) positive phase contribution to the update. Unlike for Herding, meaningful conditional expectations may be computed for the visible and hidden units.

For example, this is very useful to sample grayscale images. Conversely, we can imagine changing the step size for Herding (which should control the mixing rate) or using it with fast weights. We thus back-ported our additional parameters to Herding in order to make the Rates-Herding sampling algorithm (see `RatesHerdingSample`).

Finally, we consider yet another sampling algorithm where the rates are actually set to zero (which we call the Sample Penalization (SP) sampling algorithm). This draws a closer parallel to the example of section 3 (with $\tilde{\mathbf{A}}$) 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, Sample Penalization will always create a heavy distortion of the parameters (in fact, without a decay of the fast weights, the samples rapidly converge to a uniform distribution on the whole configuration space).

`RatesHerdingSample(`θ`,` θ_F`,` α`,` ϵ_F`,` R`,` $\mathbf{x}^-$`)`

Sample from an RBM with parameters $\theta = (\mathbf{W}, \mathbf{b}, \mathbf{c})$, performing deterministic inference steps starting from an initial sample $\mathbf{x}^-$ in order to return a new sample. That new sample is typically used as the starting point for the next call to RatesHerdingSample. Differs from RatesFPCDSample by the fact that the samples are computed deterministically. Differs from the original algorithm in (Welling, 2009a) by the inclusion of fast parameters with decay rate α and step size ϵ_F (the original version can be recovered if $\alpha = \epsilon_F = 1$).

while $\mathbf{x}^-$ changes **do**
  $\mathbf{h}^- \leftarrow \mathbb{1}(P(\mathbf{h}|\mathbf{x}^-; \theta + \theta_F) > \frac{1}{2})$
  $\mathbf{x}^- \leftarrow \mathbb{1}(P(\mathbf{x}|\mathbf{h}^-; \theta + \theta_F) > \frac{1}{2})$
end while
$\mathbf{W}_F \leftarrow \alpha \mathbf{W}_F + \epsilon_F(R_w - \mathbf{x}^{-\prime}\mathbf{h}^-)$
$\mathbf{b}_F \leftarrow \alpha \mathbf{b}_F + \epsilon_F(R_b - \mathbf{h}^-)$
$\mathbf{c}_F \leftarrow \alpha \mathbf{c}_F + \epsilon_F(R_c - \mathbf{x}^-)$
return $\mathbf{x}^-$

---

5 Experimental Results

We used three 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, the MNIST digit images, and a bag-of-words representation of the OpenTable dataset. The USPS dataset has 16×16 pixel digit images. 6,291 are used for training, 1,000 for hyper-parameter selection, and 2,007 for testing. The MNIST dataset has 28×28 pixel images; 50,000 for training, 10,000 for hyper-parameter selection, and 10,000 for testing (only 2000 were used to generate the likelihood curves shown here, to reduce computation time). The OpenTable dataset consists of 435,786 reviews of restaurants, from which we picked a subset of 80,000 train, 10,000 validation and 10,000 test examples. Each example is a vector of 1,000 binary values, the ith of which encoding the presence or absence of the ith most common word stem in the dataset.

To estimate the quality of the samples generated by a sampling procedure, we use the ISL (Indirect Sampling Likelihood) procedure (Breuleux et al., 2010; Desjardins et al., 2010). The main idea of this procedure is to compare the generated samples to the test set, and exploit a non-parametric density estimator to evaluate how close each test example is from any of the generated examples (i.e., what likelihood is given to the test set by the non-parametric density estimator trained on the generated samples). We plot the ISL likelihood proxy against the number of samples generated (Figure 2),

to evaluate how quickly the generated samples cover the main modes of the distribution of interest (i.e. cover the test set examples). When the ISL curve rises quickly, it means that the model mixes quickly. The asymptotic value of ISL (as the number of examples becomes large) is more indicative of the quality of the generative model in general but tells less about the quality of the mixing process (how quickly it covers the main modes).

5.1 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 step size ϵ_F, a weight decay rate α for the difference parameters θ_F and the number k of Gibbs iterations between each sample. These hyper-parameters were selected to maximize ISL, so as to compare each sampling method in its best light. To focus on the effect of the sampling method, they are all compared on the same model, which is chosen here as the best (according to ISL) among all training methods and hyper-parameters, and for the MNIST and USPS datasets, this turned out to be a model trained by FPCD. In general, the results of the comparison hold regardless of the trained model we start with.

Fig. 1 shows sequences of samples obtained from the same model on MNIST, 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 and Rates-FPCD leading and Gibbs a distant last. Note that the increase in the mixing rate is sometimes mitigated by a higher proportion of spurious samples, that is, samples that lie on the path from a major mode to another, but are not valid by themselves. This indicates that sampling through Rates-FPCD might indeed have an effect similar to an increase in temperature (where energy differences are smoothed). Despite this, the samples look faithful to the training set, unlike samples from a Gibbs chain with temperature high enough to mix well.

Fig. 2 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 as a Parzen classifier 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). Rates-Herding typically starts out with the fastest rise in ISL (faster mixing) but is caught up with by Rates-FPCD. As discussed below this may be related to a difference in optimal ϵ_F.

Note that for equal k, our implementation of Gibbs sampling is about two (on USPS) to seven (on MNIST - may be due to cache or paging issues) times faster than that of Rates-FPCD. This is mostly due to the use of slow and fast weights, the presence of parameter updates, and some additional bookkeeping, so it is an overestimate. Even taking this into account, a look at 2 shows that the latter covers the state space *at least* 20 times faster than the former on USPS. On MNIST, that would be at least 250 times faster (100 samples from Rates-FPCD being better than 200,000 Gibbs samples). A fair comparison of Rates-FPCD with Rates-Herding is more difficult to make since they give very similar advantages at a similar cost and that the latter can only produce discrete samples. Depending on the dataset we used, Rates-FPCD was either equivalent or superior to Rates-Herding. We do not believe that our experiments as they stand provide compelling reasons to prefer one over the other, when both are applicable (on binary inputs).

The optimal step size for the Rates-FPCD method, in the long run, was observed to be lower than the optimal step size for the Rates-Herding method by a factor of 10 to 100 in most cases. The step size seems to control a tradeoff between mixing rate and sampling noise, meaning that the mixing rate

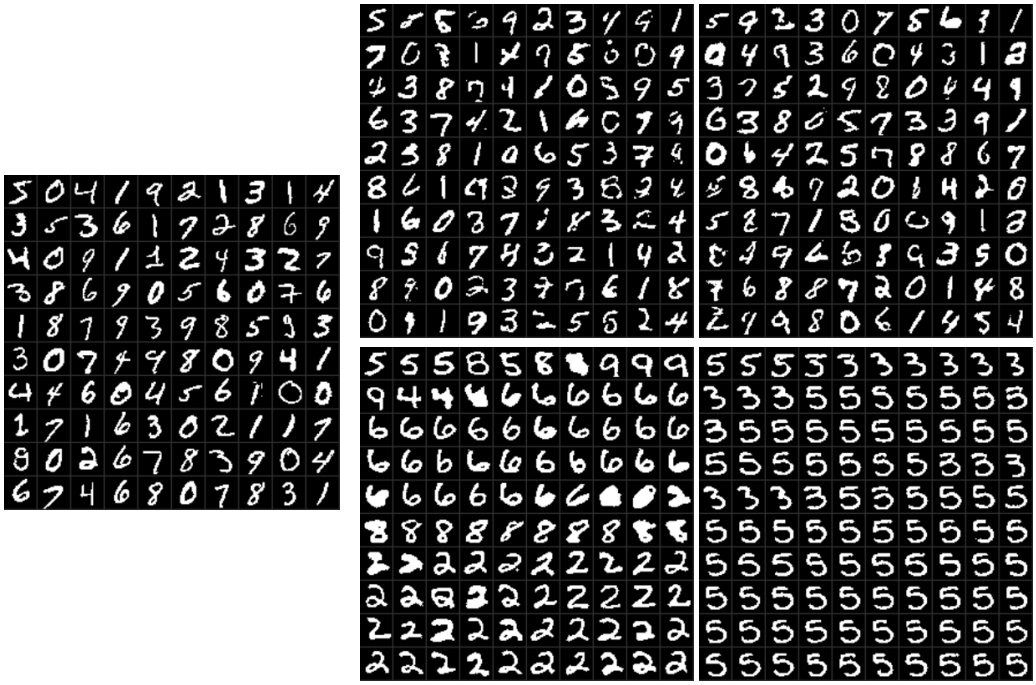

Figure 1: Left: the first 100 images from the MNIST dataset. Right: samples obtained on an FPCD-trained model using various sampling procedures, all from the same model (trained by FPCD on MNIST digits). Each sampler was run continuously for 100 seconds, which amounts to one second of CPU time for each sample shown. The samplers are Rates-FPCD (top left), Rates-Herding (top right), Sample Penalization (bottom left) or Gibbs (bottom right).

increases as the step size ϵ_F increases, but so does the frequency of spurious samples. It seems that Rates-Herding is more robust to larger step sizes, but that this 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. On other datasets, there is no clear difference, which prompts some caution in comparing both approaches.

An 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. 3, left) yielded worse results. This can be explained by the fact that the models are not fully optimized and that $\alpha = 1$ allows the sampler to optimize them further, thereby improving the quality of samples according to our metric. It is worth noting that such is not the case during training ($\alpha < 1$ works better in that case). We believe this may be because there is a need to keep the fast and slow weights evolving in sync while the relationship between the visible and hidden units is learned.

The Sample Penalization method systemically performed better than Gibbs, but still significantly worse than the other methods. It can be seen on fig. 1 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), as expected according to the earlier discussion.

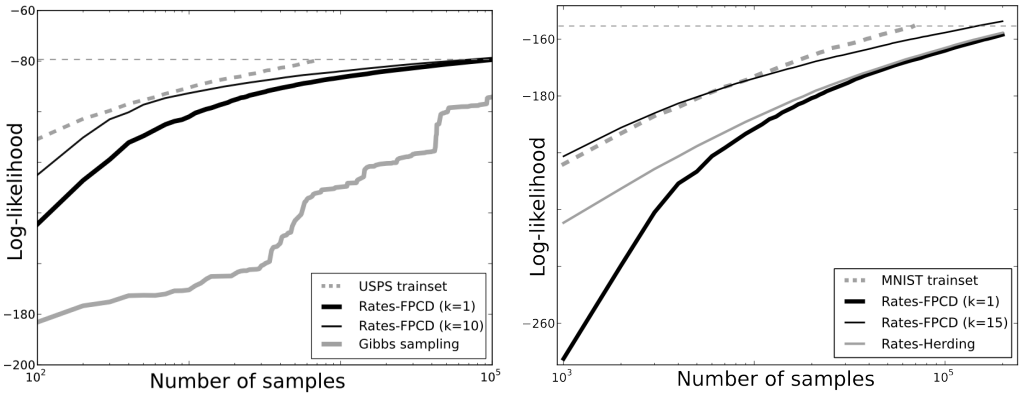

Figure 2: Left: Comparison of different sampling schemes in terms of their sample-based coverage log-likelihood (ISL) on USPS, with respect to number of samples generated. All schemes were applied to the same best model found. The sampling schemes compared are, in order of decreasing final performance: Rates-FPCD with k=10 and k=1 (thin black, thick black), the 7291 samples of the training and validation sets (dashed grey), Gibbs sampling (grey). Note that Rates-FPCD with k=10, although it does better than for k=1 with fewer samples, takes about ten times the computation time. Rates-Herding, not shown to avoid visual clutter, lies in-between the two Rates-FPCD curves. Right: Same, but on MNIST: Rates-FPCD with k=15 (thin black), 68,000 samples from the MNIST dataset (dashed grey), rates-Herding (grey), Rates-FPCD with k=1 (thick black). Gibbs sampling performed too poorly to be visible in the window shown (it peaked below -400 of log-likelihood on this model, and over all trials, it never passed -325) Note: the effective k (average number of iterations to equilibrium) for rates-Herding is, on average, 14.5.

Experiments on the OpenTable dataset showed a different, but just as interesting picture. On this dataset, the ISL performance of Gibbs sampling was on par with Rates-FPCD for some models (models trained with SML or FPCD), but Gibbs sampled failed badly on other models, in particular those trained using CD. As expected, we find Rates-FPCD to work more *consistently*. As shown in figure 3 on the right, when applied on models trained using CD, Gibbs sampling quickly and systemically gets stuck in a very narrow mode (usually, this mode is a vector of zeros). This causes the ISL measure to decrease, since an increasingly disproportionate weight is given to a single sample. Rates-FPCD, on the other hand, does not encounter any obvious mixing problems. We surmise that CD training creates very deep spurious energy wells for Gibbs samplers to get stuck in, but that Rates-FPCD optimizes them away during sampling (thereby improving the model, as explained in section 4). To a lesser extent, we also observe better consistency from Rates-FPCD on models trained with SML or FPCD, the mean and variance of the ISL performance being respectively higher and lower.

6 Conclusion

In this paper, we set out to evaluate several sampling methods over RBM models whose objective is to mix well: Rates-Herding, the essence of which had previously been proposed (Welling, 2009a) and which we extended with variable step size, Sample Penalization, which directly tries to move away from just visited configurations, and Rates-FPCD, which is inspired from Rates-Herding, FPCD, and Sample Penalization. Interestingly, the Rates-FPCD sampling procedure can generate samples that get better as we generate more of them, as it optimizes a lower bound on the log-likelihood of the training data. We showed that in cases where Gibbs sampling does not mix well, Rates-Herding

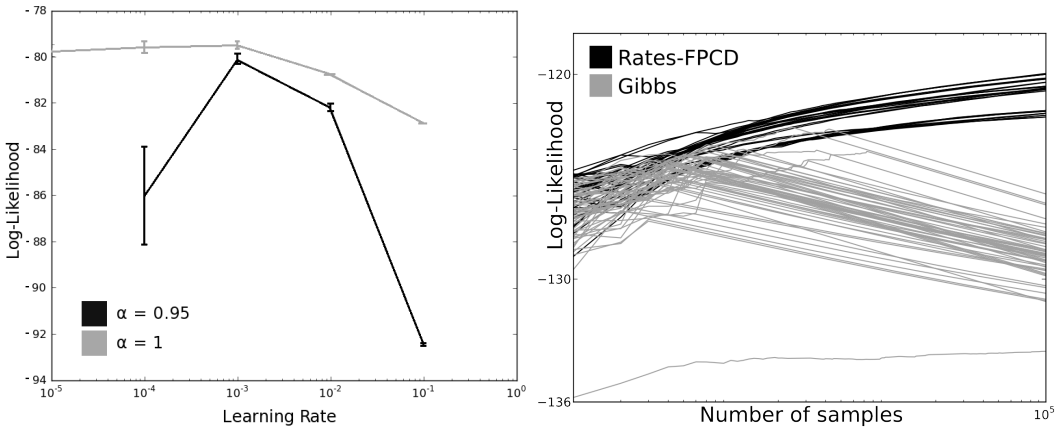

Figure 3: Left: Comparison of ISL for Rates-FPCD sampling with (bottom, black, $\alpha = 0.95$) or without (top, grey, $\alpha = 1$) weight decay on θ_F, with different step sizes. The sampling is applied on the best model found (an FPCD-trained model). Right: ISL performance with respect to the number of samples drawn (as with other plots, x axis is in log scale), for Gibbs sampling (red) and Rates-FPCD (blue), on all models trained on the OpenTable dataset with CD (all values of the training hyper-parameters that we tried). Besides one outlier which somehow manages to do even worse, the Gibbs samplers systemically get stuck in a unique mode, hence the downwards slope (here, for the most part, they end up sampling null vectors). Rates-FPCD, on the other hand, does not get stuck in such modes, and yields systemically better performance. Note that for the OpenTable dataset, Gibbs sampling performs well on models trained by SML and FPCD.

and Rates-FPCD produced samples that **much more quickly visited modes and covered the test examples**, both from visual inspection and quantitatively.

References

Bengio, Y., & Delalleau, O. (2009). Justifying and generalizing contrastive divergence. *Neural Computation*, *21*, 1601–1621.

Borkar, V. S. (2008). *Stochastic approximation: A dynamical systems viewpoint*. Cambridge University Press.

Breuleux, O., Bengio, Y., & Vincent, P. (2010). *Unlearning for better mixing* (Technical Report 1349). Université de Montréal/DIRO.

Desjardins, G., Courville, A., Bengio, Y., Vincent, P., & Delalleau, O. (2010). Tempered Markov chain monte carlo for training of restricted Boltzmann machine. *JMLR W&CP: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2010)* (pp. 145–152). Chia Laguna Resort, Sardinia, Italy.

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).