Training opposing directed models using geometric mean matching

Jörg Bornschein*, Samira Shabanian, Asja Fischer, Yoshua Bengio*
Department of Computer Science and Operations Research
University of Montreal
Montreal, Quebec, Canada

Abstract
Unsupervised training of deep generative models containing latent variables and performing inference remains a challenging problem for complex, high dimensional distributions. One basic approach to this problem is the so called Helmholtz machine and it involves training an auxiliary model that helps to perform approximate inference jointly with the generative model which is to be fitted to the training data. The top-down generative model is typically realized as a directed model that starts from some prior at the top, down to the empirical distribution at the bottom. The approximate inference model runs in the opposite direction and is typically trained to efficiently infer high probability latent states given some observed data. Here we propose a new method, referred to as geometric mean matching (GMM), that is based on the idea that the generative model should be close to the class of distributions that can be modeled by our approximate inference distribution. We achieve this by interpreting both the top-down and the bottom-up directed models as approximate inference distributions and by defining the target distribution we fit to the training data to be the geometric mean of these two. We present an upper-bound for the log-likelihood of this model and we show that optimizing this bound will pressure the model to stay close to the approximate inference distributions. In the experimental section we demonstrate that we can use this approach to fit deep generative models with many layers of hidden binary stochastic variables to complex and high dimensional training distributions.

1 Introduction and background

Training good generative models and fitting them to complex and high dimensional training data with probability mass in multiple disjunct locations remains a major challenge. This is especially true for models with multiple layers of deterministic or stochastic variables, which is unfortunate because it has been argued previously [1, 2] that deeper generative models have the potential to capture higher-level abstractions and thus generalize better. With the Helmholtz machine [3, 4], a concept was introduced that proposed to not only fit a powerful but intractable generative model \( p(x, h) \) to the training data, but also to jointly train an approximate inference model \( q(h|x) \). The \( q \) model would be used to efficiently perform approximate inference over the latent variables \( h \) of the generative model given an observed example \( x \). This basic idea has been applied and enhanced many times; initially with the wake-sleep algorithm (WS, [3, 5]) and more recently with the variational autoencoder (VAE, [6]), stochastic backpropagation and approximate inference in deep generative models [7], neural variational inference and learning (NVIL, [8]) and reweighted wake-sleep (RWS, [9]). Most of these approaches rely on the variational bound to perform appropriate inference and to obtain an objective function that contains the parameters of both the generative model \( p \) and the approximate inference model \( q \) in one joint expression (e.g. WS, VAE and NVIL). Not directly

*Jörg Bornschein is a CIFAR Global Scholar; Yoshua Bengio is a CIFAR Senior Fellow
related to the Helmholtz machine, but similar in spirit is the approach described in [10]. It also relies on a bottom-up distribution \( q(h|x) \) to perform approximate inference for the latent variables in a very deep generative model \( p(x,h) \). But the inference distribution \( q \) is fixed and only the parameters of \( p \) are learned.

In contrast to these previous approaches we here propose to interpret both \( p \) and \( q \) as approximate inference models for our actual generative model \( p^*(x,h) \). We define the target distribution \( p^* \) to be the geometric mean over the top-down and bottom-up approximate inference models, i.e. \( p^*(x,h) = \frac{1}{Z} \sqrt{p(x,h)q(x,h)} \), which gives rise to the name geometric mean matching (GMM). The motivation behind this definition is to ensure that the intractable generative model \( p^* \) stays close to the approximate inference models we have at our disposal.

In Section 2 we will introduce the GMM model in detail and discuss important theoretical properties and in Section 3 we will explain how to perform importance sampling based training and inference. The ability to model complex distributions is demonstrated empirically in Section 4.

2 Model definition and properties

We introduce the concept by defining a joint probability distribution over three variables, an observed vector \( x \) and two latent variable vectors \( h_1 \) and \( h_2 \). Analogous to a Deep Boltzmann Machine [11] we think of these as layers in a neural network with links between \( x \) and \( h_0 \) on the one side, and \( h_1 \) and \( h_2 \) on the other side. We will present the approach in the specific case of an architecture with two hidden layers, but it can be applied to arbitrary graphs of variables without loops. It can especially be used to train architectures with more than two stacked layers of latent variables.

Let \( p^*(x,h_1,h_2) \) be a joint probability distribution constructed in a specific way from two constituent distributions \( p(x,h_1,h_2) \) and \( q(x,h_1,h_2) \),

\[
p^*(x,h_1,h_2) = \frac{1}{Z} \sqrt{p(x,h_1,h_2)q(x,h_1,h_2)},
\]

where \( Z \) is a normalization constant and \( p \) and \( q \) are directed graphical models from \( h_2 \) to \( x \) and vice versa. In other words \( p \) and \( q \) can be factorized as

\[
p(x,h_1,h_2) = p(h_2)p(h_1|h_2)p(x|h_1) \quad \text{and} \quad q(x,h_1,h_2) = q(x)q(h_1|x)q(h_2|h_1).
\]

We assume that the prior distribution \( p(h_2) \) and all conditional distributions belong to some parametrized families of distributions which can be evaluated and sampled from efficiently. For \( q(x) \) we do not assume an explicit form, but define it to be the marginal

\[
q(x) = p^*(x) = \sum_{h_1,h_2} p^*(x,h_1,h_2)
\]

(3)

\[
= \frac{\sqrt{q(x)}}{Z} \sum_{h_1,h_2} \sqrt{p(x,h_1,h_2)q(h_1|x)q(h_2|h_1)}
\]

(4)

\[
= \left( \frac{1}{Z} \sum_{h_1,h_2} \sqrt{p(x,h_1,h_2)q(h_1|x)q(h_2|h_1)} \right)^2.
\]

(5)

The normalization constant \( Z \) guarantees that \( \int_{x,h_1,h_2} p^*(x,h_1,h_2) = 1 \). Using the Cauchy-Schwarz inequality \( \int f(y)g(y)dy \leq \int |f(y)|^2dy \int |g(y)|^2dy \) and identifying \( \sqrt{p(y)} \) with \( f(y) \) and \( \sqrt{q(y)} \) with \( g(y) \), it becomes clear that \( Z = \int_x \sqrt{p(y)q(y)}dy \leq 1 \) for arbitrary \( p \) and \( q \). Furthermore, we see that \( Z = 1 \) only when \( p(x,h_1,h_2) = q(x,h_1,h_2) \). We can therefore obtain a lower bound on the marginal probability \( p^*(x) \):

\[
\hat{p}^*(x) = \left( \sum_{h_1,h_2} \sqrt{p(x,h_1,h_2)q(h_1|x)q(h_2|h_1)} \right)^2 = Z^2 p^*(x) \leq p^*(x).
\]

(6)
This suggests that the model distribution \( p^*(x) \) can be fitted to some training data by maximizing the bound of the log-likelihood \( \tilde{p}^*(x) \) instead of \( p^*(x) \), as we elaborate in the following section.

Since \( \log \tilde{p}^*(x) \) can reach the maximum only when \( Z \rightarrow 1 \), the model is implicitly pressured to find a maximum likelihood solution that yields \( p(x, h_1, h_2) \approx q(x, h_1, h_2) \approx p^*(x, h_1, h_2) \).

2.1 Alternative view using the Bhattacharyya distance

Let us consider the case where we would actually maximize \( \log p^*(x) \) instead of the lower bound \( \log \tilde{p}^*(x) \). We can then decompose the objective using the Bhattacharyya distance \( D_B(p, q) = -\log \sum_y \sqrt{p(y)q(y)} \) into two terms:

\[
\log p^*(x) = 2 \log \sum_{h_1, h_2} \sqrt{p(x, h_1, h_2)q(h_1|x)q(h_2|h_1)} - 2 \log \sum_{x', h'_1, h'_2} \sqrt{p(x', h'_1, h'_2)q(x'|h'_1)q(h'_2|h'_1)}
= \log \tilde{p}^*(x) - 2 \log Z = \log \tilde{p}^*(x) + 2 D_B(p, q)
\]

with \( D_B(p, q) \geq 0 \) for arbitrary \( p, q \) and \( D_B(p, q) = 0 \) only when \( p = q \).

We can compare this to the variational approach, where the marginal probability \( \log p(y) \) of some model containing latent variables \( z \) is rewritten in terms of the KL-divergence \( D_{KL}(q \| p) = \sum_z q(z|y) \log \frac{q(z|y)}{p(z|y)} \geq 0 \) to obtain a lower bound:

\[
\log p(y) = \mathbb{E}_{z \sim q(z|y)} [\log p(y, z) - \log q(z|y)] + D_{KL}(q(z|y) \| p(z|y)) \geq \mathbb{E}_{z \sim q(z|y)} [\log p(y, z) - \log q(z|y)].
\]

(8)

Analogous to variational methods that maximize the lower bound \( \mathbf{6} \), we can thus maximize \( \log \tilde{p}^*(x) \) and it will tighten the bound as \( D_B(p, q) \) approaches zero. While this seems very similar to the variational lower bound, we should highlight that there are some important conceptual differences: 1) The KL-divergence in variational methods typically measures the distance between distributions given some training data. The Bhattacharyya distance here in contrast quantifies a property of the model \( p^*(x, h_1, h_2) \) independently of any training data. In fact, we saw that \( D_B(p, q) = -\log Z \). 2) The variational lower bound is typically used to construct approximate inference algorithms. We here use our bound \( \tilde{p}^*(x) \) just to remove the normalization constant \( Z \) from our target distribution \( p^*(x, h_1, h_2) \). Even after applying the lower-bound we still have to tackle the inference problem which manifests itself in form of the full combinatorial sum over \( h_1 \) and \( h_2 \) in equation \( \mathbf{6} \). Although it seems intuitively reasonable to use a variational approximation on top of the bound \( \tilde{p}^*(x) \), we will here not follow this direction but rather use importance sampling to perform approximate inference and learning (see section \( \mathbf{3} \)). Combining a variational method with the bound \( \tilde{p}^*(x) \) is therefore subject to future work.

We can also argue that optimizing \( \log \tilde{p}^*(x) \) instead of \( \log p^*(x) \) is beneficial in the light of the original goal we formulated in section \( \mathbf{1} \). To learn a generative model \( p^*(x) \) which is regularized to be close to the model \( q \) which we use to perform approximate inference for \( p^* \). Let us assume we have two equally well trained models \( p_{\theta_1}^* \) and \( p_{\theta_2}^* \): \( \mathbb{E}_{x \sim D} [\log p_{\theta_1}^*(x)] = \mathbb{E} [\log p_{\theta_2}^*(x)] \), but the expected bound \( \tilde{p}^*(x) \) for the first model is closer to the log-likelihood than the expected bound for the second model: \( \mathbb{E} [\log \tilde{p}_{\theta_1}^*(x)] > \mathbb{E} [\log \tilde{p}_{\theta_2}^*(x)] \). Using equation (7) we see that \( D_B(p_{\theta_1}, q_{\theta_1}) < D_B(p_{\theta_2}, q_{\theta_2}) \) which indicates that \( q_{\theta_1} \) is closer to \( p_{\theta_1}^* \) than \( q_{\theta_2} \) to \( p_{\theta_2}^* \) (when we measure their distance using the Bhattacharyya distance). According to our original goal we thus prefer solution \( p_{\theta_1}^* \), where the bound \( \tilde{p}^*(x) \) is maximized and the distance \( D_B(p, q) \) minimized.

Note that the decomposition (7) also emphasizes why our recursive definition \( q(x) = \sum_{h_1, h_2} p^*(x, h_1, h_2) \) is a consistent and reasonable one: Minimizing \( D_B(p, q) \) during learning means that the joint distributions \( p(x, h_1, h_2) \) and \( q(x, h_1, h_2) \) approach each other. This implies that the marginals \( p(h_1) \) and \( q(h_1) \) for all layers \( l \) become more similar. This also implies
Algorithm 1 Training $p^*(x)$ using important sampling with $q$ as proposal

for number of training iterations do
  • Sample example(s) $x \sim D$ from the training distribution
  for $k = 1$ to $K$ do
    • Layerwise sample latent variables $h^{(k)}$ from $q(h|x)$
    • Compute $q(h^{(k)} | x)$ and $p(x, h^{(k)})$
  end for
  • Compute unnormalized weights $\omega_k = \sqrt{\frac{p(x, h^{(k)})}{q(h^{(k)} | x)}}$
  • Normalize the weights $\tilde{\omega}_k = \frac{\omega_k}{\sum_{k'} \omega_{k'}}$
  • Update parameters in $p$ and $q$: Use gradient estimator
    $2 \sum_k \tilde{\omega}_k \frac{\partial \log p(x, h^{(k)})}{\partial \theta}$
end for

$p(x) \approx q(x)$ in the limit of $D_B(p, q) \to 0$; a requirement that most simple parametrized distributions $q(x)$ could never fulfill. It is thus a piece of good fortune that the optimal, but intuitively clumsy definition $q(x) = \sum h_1, h_2 p^*(x, h_1, h_2)$ results in the likelihood expression (6), an expression that has no higher computational complexity and that has mostly the same basic structure as the likelihood expression for any other model containing latent variables.

3 Inference and training with importance sampling

Based on the construction of $p^*(x)$ outlined in the previous section we can define a wide range of possible models. We furthermore have a wide range of potential training and appropriate inference methods we could employ to maximize $\log \hat{p}^*(x)$. In this text we concentrate on binary latent and observed variables $x$ and $h_1 \in \{0, 1\}$, and we will use simple sigmoid belief network layers for all our conditional distributions: $P(x | y) = \prod_i B(x_i | \sigma(W_i y + b_i))$ where $B(a | b)$ refers to the Bernoulli distribution for value $a$ with probability parameter $b$, and $\sigma(\cdot)$ is the logistic sigmoid function. For our top-level prior $p(h_2)$ we use a factorized Bernoulli distribution: $p(h_2) = \prod_i B(h_{2,i} | \sigma(b_{2,i}))$.

We form an estimate of $\hat{p}^*(x)$ by using important sampling instead of the exhaustive sum over $h_1$ and $h_2$ in equation (6). We use $q(h_1 | x)q(h_2 | h_1)$ as the proposal distribution which is by construction easy to evaluate and to sample from:

$$\hat{p}^*(x) = \left( \sum_{h_1, h_2} \sqrt{p(x, h_1, h_2) q(h_1 | x) q(h_2 | h_1)} \right)^2 = \left( \mathbb{E}_{h_2 \sim \hat{q}(h_2 | h_1)} \left[ \sqrt{\frac{p(x, h_1, h_2)}{q(h_1 | x) q(h_2 | h_1)}} \right] \right)^2 \approx \left( \frac{1}{K} \sum_{k=1}^K \sqrt{\frac{p(x, h_1^{(k)}, h_2^{(k)})}{q(h_1^{(k)} | x) q(h_2^{(k)} | h_1^{(k)})}} \right)^2 = \hat{p}^*(x) \text{ with } h_2^{(k)} \sim \hat{q}(h_2 | h_1^{(k)}) \text{ and } h_1^{(k)} \sim q(h_1 | x).$$

Using the same approach we can also derive the well known estimator for the marginal probability of a datapoint under the top-down generative model $p$:

$$\hat{p}(x) = \mathbb{E}_{h_2 \sim \hat{q}(h_2 | h_1)} \left[ \frac{p(x, h_1, h_2)}{q(h_1 | x) q(h_2 | h_1)} \right]$$

Analogous to the parameter updates in reweighted wake-sleep (RWS, [9]) we can derive an important sampling based estimate for the parameter gradients and use them to optimize towards a
These are unfortunately intractable as we have to evaluate the marginal \( q \) as the gradient computation. The updates do not require any form of backpropagation through more than one layer because, as far observed. The final gradient is determined by computing the normalized weighted average over the individual gradients (equation (11)). These properties are basically inherited from the RWS training algorithm \[9\]. But in contrast to RWS, and in contrast to most other algorithms which employ a generative model \( p \) and an approximate inference model \( q \), we here automatically obtain parameter updates for both \( p \) and \( q \) because we optimize \( p^* \) which contains both. The resulting training method is summarized in algorithm \[1\].

### 3.1 Sampling from the model \( p^*(x) \)

One way to approximately sample from \( p^*(x) \) is to sample proposal candidates from \( p(x, h_1, h_2) \) and then use importance resampling to obtain samples that are effectively distributed according to \( p^*(x) \). The resampling weights can be written as

\[
\omega = \frac{p^*(x, h_1, h_2)}{p(x, h_1, h_2)} = \sqrt{\frac{q(x, h_1, h_2)}{p(x, h_1, h_2)}}.
\]

(12)

These are unfortunately intractable as we have to evaluate the marginal \( q(x) = p^*(x) \). As an approximation we can use the sampling based estimate \( \hat{p}^*(x) \) (equation (9)). Note that we don’t have to evaluate the normalization constant \( Z \) as the resampling procedure only evaluates the relative magnitude of the importance weights. We thus end up with the procedure described in algorithm \[2\].

### 3.2 Conditional sampling and inpainting

We can use a similar importance resampling approach to approximately draw samples from conditional distributions, e.g. from \( p^*(h_1 | x, h_2) \). We here choose to draw the proposal samples from the mixture distribution \( p_{\text{proposal}}(h_1 | x, h_2) = \frac{1}{2} p(h_1 | h_2) + \frac{1}{2} q(h_1 | x) \) which should ensure that we...
have a symmetric chance of covering the high probability configurations of \( p^* (\mathbf{x}) \) presented to the algorithm and the right column in each block shows the reconstruction sampled digit randomly sampled from the MNIST test set. The center column shows the masked version.

**Algorithm 2** Sampling from \( p^* (\mathbf{x}) \) using important resampling

for \( k = 1 \) to \( K \) do
- Layerwise draw primary proposal sample \( \mathbf{x}^{(k)}, \mathbf{h}_1^{(k)}, \mathbf{h}_2^{(k)} \sim p(\mathbf{x}, \mathbf{h}_1, \mathbf{h}_2) \)
for \( l = 1 \) to \( K \) do
- Layerwise sample latent variables \( \tilde{h}_1^{(l)}, \tilde{h}_2^{(l)} \sim q(\mathbf{h}_1, \mathbf{h}_2 | \mathbf{x}^{(k)}) \)
end for
- Compute the estimated marginal bound \( \hat{p}^* (\mathbf{x}^{(k)}) \) using the samples \( \tilde{h}_1^{(l)}, \tilde{h}_2^{(l)} \) (see equation 9)
- Compute \( \hat{\omega}^{(k)} = \sqrt{\hat{p}^* (\mathbf{x}^{(k)})/q(\mathbf{h}_1^{(k)} | \mathbf{x})q(\mathbf{h}_2^{(k)} | \mathbf{x})/p(\mathbf{x}^{(k)}, \mathbf{h}_1^{(k)}, \mathbf{h}_2^{(k)})} \)
end for
- Subsample a final sample \( \mathbf{x} \) from the primary proposals \( \mathbf{x}^{(k)} \) proportional to their weights \( \hat{\omega}^{(k)} \)

Figure 2: Inpainting of binarized MNIST digits. The left column in each block shows the original digit randomly sampled from the MNIST test set. The center column shows the masked version presented to the algorithm and the right column in each block shows the reconstruction sampled using the algorithm described in section 3.2. All images were selected randomly.

In this section we present experimental results obtained when applying the algorithm to various small and medium scale datasets. Our main goal is to ensure that the theoretical properties discussed in section 2 translate into a robust algorithm that gives competitive results even when used with simple sigmoid belief network layers as conditional distributions. All models were trained using RMSProp (see e.g. [12] for a description) with a mini-batch size of 100. We initialize all weights according to [13] and set all biases to -1. Our implementation is available at \( \text{http://anonymized} \).
4.1 UCI binary datasets

To ascertain that GMM with the importance sampling as training method works in general we applied it to the 8 binary datasets from the UCI dataset repository that where evaluated e.g. in [14]. The results are summarized in table 4.1. We report two upper bounds on the model negative log-likelihood: $\log \hat{p}(x)$ and $\log \hat{p}(\mathbf{x})$ (see equations 9 and 10). Both are conservative estimators of the model NLL because they overestimate by an unknown offset: the former because it is a conservative unbiased estimator for the likelihood of the approximate generative model $p$ and not of the model $p^*$; the latter because it evaluates $\log p^*$ under the assumption $\mathcal{Z} = 1$.

<table>
<thead>
<tr>
<th>Model</th>
<th>ADULT</th>
<th>CONNECT4</th>
<th>DNA</th>
<th>MUSHROOMS</th>
<th>NIPS-0-12</th>
<th>OCR-LETTERS</th>
<th>RCV1</th>
<th>WEB</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto regressive</td>
<td>13.19</td>
<td>11.99</td>
<td>84.81</td>
<td>9.81</td>
<td>273.08</td>
<td>27.22</td>
<td>46.66</td>
<td>28.39</td>
</tr>
<tr>
<td>NoNAD[14]</td>
<td>13.19</td>
<td>12.58</td>
<td>82.31</td>
<td>9.68</td>
<td>272.38</td>
<td>27.31</td>
<td>46.12</td>
<td>27.87</td>
</tr>
<tr>
<td>non AR</td>
<td>16.26</td>
<td>22.66</td>
<td>96.74</td>
<td>15.15</td>
<td>277.37</td>
<td>43.05</td>
<td>48.88</td>
<td>29.38</td>
</tr>
<tr>
<td>GMM</td>
<td>13.58</td>
<td>12.09</td>
<td>90.58</td>
<td>9.40</td>
<td>270.44</td>
<td>29.21</td>
<td>46.12</td>
<td>28.02</td>
</tr>
<tr>
<td>log $\hat{p}(\mathbf{x})$</td>
<td>13.79</td>
<td>12.28</td>
<td>90.90</td>
<td>9.50</td>
<td>272.95</td>
<td>29.82</td>
<td>46.98</td>
<td>28.78</td>
</tr>
</tbody>
</table>

Table 1: Results on various binary datasets from the UCI repository. The top rows quote various baseline results from (shallow) models with autoregressive weights between their units within one layer. The second block shows results from various non-autoregressive models. In the third block we see results obtained by running GMM on these datasets. $\log \hat{p}(\mathbf{x})$ shows the expected NLL when evaluating just the top-down model $p(\mathbf{x})$ against the testset; $\log \hat{p}^*(\mathbf{x})$ shows the lower bound of the model NLL which assumes $1/Z = 1$. Both underestimate the true model NLL by an unknown amount.

4.2 Binarized MNIST

We use the MNIST dataset that was binarized according to [17] and downloaded in binarized from [18]. We train a model with 6 hidden layers containing 800,800,600,450,350,150 latent variables. We set the learning rate to $3 \times 10^{-5}$ and use otherwise the same hyperparameters as in the previous experiments. The training of the model converges after $\approx 200$ epochs. The importance sampling based estimators for $\log p$ and $\log \hat{p}^*$ report a negative log-likelihood of 92.7 and 95.1 respectively (on the MNIST testset). Again, note that both overestimate the true model NLL by an unknown amount (see previous section).

To demonstrate that the model learned a reasonable concept of binarized digits we use algorithm 2 to draw samples. The results are visualized in Figure 1. The samples obtained from $p^*(\mathbf{x})$ show almost always well defined digits with sharp (non-blurry) edges. But unfortunately they are biased: It draws more ones than any other digit. Note that the proposal samples drawn from $p$ do not show any obvious bias. We nevertheless suspect that the failure of our model $p^*$ to correctly learn the prior distribution over different model classes is mostly due to $p$ failing to learn an approximate variability of digit shapes: we observe that the estimator $\hat{p}^*(\mathbf{x})$ generally assigns a much higher probability to individual datapoints that resemble ones, than to datapoints that resemble other digit classes. This seems intuitively reasonable, because the variability within the class of ones is probably lower than the variability within other classes. One source of variability in binarized images is for example the blurry fringe of the strokes in the gray scale version of the original images. Digits with a shorter overall stroke length and with fewer gray pixels will result in binarized digits with lower variability. To compensate for the presumed lower variability within the class of ones (and their higher probability $q(x)$), $p$ would have to propose more samples resembling other digits than one. With a more detailed analysis of this issue still outstanding we want to stress that the failure of our model to learn the correct digit-class distribution (10 classes) results in a relatively small loss in terms of log-likelihood compared to its ability to model a sharp and non-blurry distribution over the 784 pixels of the images.

To highlight the models ability to generate sharp digits we use the inpainting algorithm described in section 5.2 to reconstruct partially occluded images. To obtain the results in Figure 2 we ran 20 up-and downward sweeps across all layers.
Figure 3: Results after training on TFD: A) Random selection of 12 samples drawn from $p^*(x)$ using algorithm 2 (see supplement for more). B) The left column in each block shows the input; the right column shows a random output sample generated by the inpainting algorithm (see section 3.2).

In Figure 4 we show the histograms of the importance weights when we use $p(x,h)$ as a proposal distribution for $p^*(x,h)$ (top); and when we use $q(h|x)$ as a proposal for $p^*(h|x)$ (bottom). According to our goal formulated in section 1 both $p$ and $q$ should stay close to $p^*$. The weights of the former occur whenever we perform approximate inference (e.g. during learning); the weights of the latter occur when we sample from the model. Unsurprisingly we observe that the quality of both proposal distributions is roughly symmetric relative to the target distribution $p^*$. As a control we also plotted a histogram of the weights when $p(x,h_1|h_2)$ is used as a proposal for $p^*(x,h_1|h_2)$ – a sampling mechanism we never used for any of our algorithms, but that is totally symmetric to the histogram for the $q(h|x)$ proposals. The resulting weight histogram was visually indistinguishable from the one we show in Figure 4 (bottom).

4.3 Toronto Face Database

We also trained models on the 98058 examples from the unlabeled section of the Toronto face database (TFD, [19]). Each training example is of size $48 \times 48$ pixels and we interpret the gray-level as Bernoulli probability for the bottommost layer. We observe that training proceeds rapidly during the first few epochs but mostly only learns the mean-face. During the next few hundred epochs training proceeds much slower but the log-likelihood bound $\log \hat{p}^*(x)$ increases steadily. Figure 5 A shows random samples from a model with 1000, 700, 700, 300 latent variables in 4 hidden layers. It was trained with a learning rate of $3 \times 10^{-5}$; all other hyperparameters were set to the same values as before. Figure 5 B shows the results from inpainting experiments with the very same model (50 up- and downward sweeps).

5 Conclusion and future work

We introduced a new scheme to construct probabilistic generative models which are automatically regularized to be close to approximate inference distributions we have at our disposal. Using the Bhattacharyya distance we derived a lower-bound on the log-likelihood and we demonstrated that the bound can be used to fit deep generative models with multiple layers of latent variables to complex training distributions. Note that our definition for $p^*$ forced us to choose a prior distribution $q(x)$ which will be part of our generative model $p^*(x,h)$. This is different from the typical variational approaches to train Helmholtz machines where we would think of $q(h|x)$ solely as an approximate inference method given a training example $x$, and where $q(x)$ would be the (empirical) training distribution – something we cannot assume because $q(x)$ is part of our model $p^*$.

But many question remain open and are subject to future work: During our experiments we have not made any serious attempts to calculate the normalization constant $Z$. Knowing $Z$ would enable us to report proper log-likelihood estimates (or tighter bounds) given some test data. For now we can just suspect that $\log 1/Z$ is significantly larger than 0 because our estimators for $\log \hat{p}(x)$ and $\log \hat{p}^*(x)$ report numbers that differ by multiple nats (for MNIST and TFD experiments). A first attempt to es-
timate $Z$ could certainly be made by applying AIS [20] or RAISE [21]. Another direction for future research could involve semisupervised learning: The symmetric nature of the generative model $p^\ast$ (it is always close to the bottom-up and top-down directed models $q$ and $p$) might make it suitable for learning tasks that require inference given changing sets of observed and hidden variables. And last but not least we have a wide range of potential choices for our parametrized conditional distributions. Assuming continuous latent variables for example and eventually choosing an alternative inference method might make $p^\ast$ a better suited model for some training distributions.

Acknowledgments

We thank the developers of Theano [22] and Blocks [23] for their awesome work.

References


Gradient for \( \log \hat{p}^*(x) \)

\[
\frac{\partial}{\partial \Theta} \log \hat{p}^*(x) = \frac{\partial}{\partial \Theta} \log \left( \sum_h \sqrt{p(x, h)q(h|x)} \right)^2
\]

\[
= \sum_h \frac{2}{\sqrt{p(x, h)q(h|x)}} \sum_h \sqrt{p(x, h')q(h'|x)} \frac{\partial}{\partial \Theta} \log \sqrt{p(x, h')q(h'|x)}
\]

\[
\approx 2 \sum_{k=1}^{K} \tilde{\omega}_k \frac{\partial}{\partial \Theta} \log \sqrt{p(x, h^{(k)})q(h^{(k)}|x)}
\]

with \( h^{(k)}_1 \sim q(h_1 | x) \), \( h^{(k)}_2 \sim q(h_2 | h_1) \) and the

importance weights
\[
\omega_k = \sqrt{\frac{p(x, h^{(k)})}{q(h^{(k)} | x)}}, \quad \tilde{\omega}_k = \frac{\omega_k}{\sum_k \omega_k}
\]

Details for the UCI experiments

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Hidden units</th>
<th>Learning rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADULT</td>
<td>100, 70, 50, 25</td>
<td>(10^{-2})</td>
</tr>
<tr>
<td>CONNECT4</td>
<td>300, 110, 50, 21</td>
<td>(10^{-2})</td>
</tr>
<tr>
<td>DNA</td>
<td>400, 350, 25</td>
<td>(10^{-3})</td>
</tr>
<tr>
<td>MUSHROOMS</td>
<td>100,75, 35, 11</td>
<td>(10^{-2})</td>
</tr>
<tr>
<td>NIPS-0-12</td>
<td>200, 100, 50, 25</td>
<td>(10^{-3})</td>
</tr>
<tr>
<td>OCR-LETTERS</td>
<td>600, 500, 100, 55, 30</td>
<td>(10^{-3})</td>
</tr>
<tr>
<td>RCV1</td>
<td>500, 120, 70, 35</td>
<td>(10^{-3})</td>
</tr>
<tr>
<td>WEB</td>
<td>650, 580, 70, 35, 11</td>
<td>(10^{-3})</td>
</tr>
</tbody>
</table>

Table 2: Hyperparameters for the UCI experiments.

Training convergence for binarized MNIST

Convergence of expected \( \log \hat{p}^*(x) \) and \( \log \hat{p}(x) \) (on the validation set) over SGD updates.
Samples the TFD model