CSCI699 - USC, Fa2019

Lecture 4: Confronting the Partition Function

September 10, 2019

Lecturer: Prof. Xiang Ren *Scribe: Nathan Bartley*

1 Review

Recall: In many cases it's difficult to directly sample from $p(x)$. We can get around this in a couple of different ways:

1. Importance Sampling, where you transform the problem such that you sample directly from a surrogate distribution $q(x)$.
2. Rejection Sampling, where you also use a surrogate distribution $q(x)$ but accept each sampled point with a probability in proportion to the two densities.
3. Markov Chain Monte Carlo methods, where we define a random process (the **Markov Chain**) that converges to $p(x)$ after some time steps. After convergence, we sample from this approximate distribution.

   In order to construct such a Markov chain, we have to find a transition distribution A. **Gibbs sampling** is a useful method for defining such a transition distribution, and hence a Markov chain. For more information regarding Gibbs sampling, see the previous lecture notes and [3].

2 Partition Functions

Many undirected graphical models are defined by an unnormalized probability distribution $\tilde{p}(x;\Theta)$, i.e., $\tilde{p}$ is non-negative and has nonzero finite integral (not necessarily equal to one).

We normalize $\tilde{p}$ by dividing by the partition function $Z(\Theta)$

$$p(x;\Theta) = \frac{1}{Z(\Theta)}\tilde{p}(x;\Theta)$$

We know that by construction, the partition function $Z(\Theta) = \int \tilde{p}(x)dx$ or when using discrete variables, $\sum_x \tilde{p}(x)$. $Z(\Theta)$ depends on the parameters of the model, making it complicated when taking the gradient of the log-likelihood function.

$$\nabla_\Theta \log p(x;\Theta) = \nabla_\Theta \log \tilde{p}(x;\Theta) - \nabla_\Theta \log Z(\Theta)$$

$$\textit{"Positive Phase"} - \textit{"Negative Phase"}$$

Like with many complicated densities, computing $Z(\Theta)$ can become intractable. This means that, in many cases, the positive phase of the gradient might be straightforward to compute, but not the negative phase.

There are three different approaches that arise to address this:

1. Design a model with a tractable $Z(\Theta)$ (e.g., softmax function)
2. Do not compute $p(x)$ at all
3. Take MCMC approaches to estimate $Z(\Theta)$

To further describe the behavior of the gradient, we explore the following

$$\nabla_\Theta \log Z = \frac{\nabla_\Theta Z}{Z} = \frac{\sum_x \nabla_\Theta \tilde{p}(x)}{Z}$$

$$\text{when } p(x) < 0 \, \forall x$$

$$\begin{aligned}
\Rightarrow \tilde{p}(x) &= \exp(\log(\tilde{p}(x))) \\
&\Rightarrow \frac{\sum_x \nabla_\Theta \tilde{p}(x)}{Z} \\
&= \frac{\sum_x \nabla_\Theta \exp(\log(\tilde{p}(x)))}{Z} \\
&= \frac{\sum_x \exp(\log(\tilde{p}(x))\nabla_\Theta \log(\tilde{p}(x))}{Z} \\
&= \sum_x p(x) \nabla_\Theta \log(\tilde{p}(x)) \\
&= \mathop{\mathbb{E}}_{x \sim p(x)} [\nabla_\Theta \log(\tilde{p}(x))] \\
&= \nabla_\Theta \log(Z)
\end{aligned}$$

This equivalence to the expected value of the gradient of the model allows us to use Monte Carlo methods to compute the gradient. The gradient requires the following conditions:

1. $\tilde{p}$ must be integrable for every Θ
2. $\nabla_\Theta \tilde{p}(x)$ must exist for all Θ and almost all x
3. There must exist some $R(x)$ that bounds $\nabla_\Theta \tilde{p}(x)$, i.e. $\max_i \left| \frac{d}{d\Theta_i} \tilde{p}(x) \right| \leq R(x) \forall \Theta$ and almost all x

Most ML models meet these conditions.

Intuition. One way to think about how we split the log-likelihood gradient across a "positive" and "negative" phase is that we are trying to increase the log $\tilde{p}(x)$ for x drawn from the data, and decrease the log $\tilde{p}(x)$ drawn from the model distribution.

3 Stochastic Maximum Likelihood and Contrastive Divergence

Now that we have a set of approaches for addressing an intractable partition function. Because we can directly estimate $\nabla_\Theta \log Z$, we can imagine readily using MCMC methods every time we need to compute the gradient.

- It is infeasible to burn in a set of Markov chains every time!

```
Algorithm 1: Gibbs Update
  Data: $x^{(i)}$
  Result: updated $x^{(i)}$
1 Draw a sample $a \sim p(x_i|x_{-i}^{(t)})$ where $x_{-i}^{(t)}$ is the set of all variables in $x^{(t)}$ except for the ith
    variable.
2 $x^{(i)} \leftarrow$ a;
3 return a
```

```
Algorithm 2: Naive MCMC algorithm
   Data: $\epsilon$ : Step size for updating $\Theta$
   $K$ : No. Gibbs steps for burn-in
   Result: $\Theta$ : learned model parameters
1  while not converged do
2      Sample minibatch $\{x^{(1)}, ..., x^{(m)}\}$;
3      $g \leftarrow \frac{1}{m}\sum_{i=1}^{m} \nabla_\Theta \log \tilde{p}(x^{(i)}; \Theta)$;
4      Initialize $\tilde{x}^{(1)}, ..., \tilde{x}^{(m)}$ to a set of random values;
5      for i = 1 : K do
6          for j = 1 : m do
7              $\tilde{x}^{(j)} \leftarrow$ GibbsUpdate($\tilde{x}^{(j)}$);
8          end
9      end
10     $g \leftarrow g - \frac{1}{m}\sum_{i=1}^{m} \nabla_\Theta \log \tilde{p}(\tilde{x}^{(i)}; \Theta)$;
11     $\Theta \leftarrow \Theta + \epsilon g$;
12 end
13 return $\Theta$;
```

Intuition. Because the negative phase involves drawing samples from the model's distribution, it tries to find points it believes in strongly. A useful analogy is to think about it like the human brain during sleep: it has been proposed that the brain maintains a probabilistic model of its world and follows $\nabla_\Theta \log \tilde{p}$ during the day, as it experiences events, and follows the negative gradient to minimize log Z while dreaming (i.e., while sleeping and experiencing events sampled from the current iteration of the model).

Limitations. Because the naive MCMC algorithm intializes the chains randomly, a better approach would be to initialize the chains closer to the input distribution. This will reduce the total number of burn-in steps necessary for every gradient step. This change yields the **Contrastive Divergence** algorithm.

Algorithm 3: Contrastive Divergence algorithm

```
   Data: $\epsilon$ : Step size for updating $\Theta$
   $K$ : No. Gibbs steps to mix when initialized from $p_{data}$
   Result: $\Theta$ : learned model parameters
1  while not converged do
2      Sample minibatch $\{x^{(1)}, ..., x^{(m)}\}$;
3      $g \leftarrow \frac{1}{m}\sum_{i=1} m\nabla_\Theta \log \tilde{p}(x^{(i)}; \Theta)$;
4      for  i = 1 : m do
5          $\tilde{x}^{(i)} \leftarrow x^{(i)}$;
6      end
7      for  i = 1 : K do
8          for j = 1 : m do
9              $\tilde{x}^{(j)} \leftarrow$GibbsUpdate($\tilde{x}^{(j)}$);
10         end
11     end
12     $g \leftarrow g - \frac{1}{m}\sum_{i=1}^{m} \nabla_\Theta log\tilde{p}(\tilde{x}^{(i)}; \Theta)$;
13     $\Theta \leftarrow \Theta + \epsilon g$;
14 end
15 return $\Theta$;
```

Intuition. The algorithm can be thought of as penalizing the model for having a Markov chain that changes the input rapidly when input comes from the data. This is reminiscient of autoencoder training. Additionally, the Contrastive divergence update direction is **not** the gradient of any function (cycles could happen).

Limitations. Contrastive divergence is good for shallow models like RBMs, but they fail to suppress regions of high probability that are far from actual training examples, i.e., **spurious modes** where there is high p_{model} by low p_{data}. Contrastive divergence is not good for training deeper models directly. Training shallow models and stacking them is possible. A way to resolve some of these problems is to initialize the Markov chains at each step with the states of the previous gradient step. This yields **stochastic maximum likelihood** or **Persistent contrastive divergence**.

Algorithm 4: Stochastic Maximum Likelihood / Persistent Contrastive Divergence algorithm

```
Data: $\epsilon$ : Step size for updating $\Theta$
$K$ : No. Gibbs steps to mix when sampling from $p(x;\Theta+\epsilon g)$, initialized from $p(x;\Theta)$
Result: $\Theta$ : learned model parameters
1  Initialize $\tilde{x}^{(i)}, ..., \tilde{x}^{(m)}$ to random values;
2  while not converged do
3      Sample minibatch $x^{(1)}, ..., x^{(m)}$;
4      $g \leftarrow \frac{1}{m}\sum_{i=1}^{m} \nabla_{\Theta} log\tilde{p}(x^{(i)};\Theta)$;
5      for $i$ = 1 : $K$ do
6          for $j$ = 1 : $m$ do
7              $\tilde{x}^{(j)} \leftarrow$GibbsUpdate$(\tilde{x}^{(j)})$;
8          end
9      end
10     $g \leftarrow g - \frac{1}{m}\sum_{i=1}^{m} \nabla_{\Theta} \log \tilde{p}(\tilde{x}^{(i)};\Theta)$;
11     $\Theta \leftarrow \Theta + \epsilon g$;
12 end
13 return $\Theta$;
```

Practical considerations. PCD has the following considerations:

- It is able to train deep models efficiently
- It is vulnerable if K is too small or ϵ is too large
- After training the model, draw samples from a fresh Markov chain initialized from a random starting point. Since it's a persistent model, the negative chains may have already "seen" all the points
- Contrastive Divergence has lower variance, but Persistent Contrastive Divergence has higher variance.

NB: We can accelerate mixing during learning by Fast PCD, setting $\Theta = \Theta^{(slow)} + \Theta^{(fast)}$, two copies of the same parameters, but with different learning rates.

3.1 Summary

Now we have methods that can estimate $\nabla_{\Theta} \log Z$. We can use some method to tackle $log\tilde{p}(x)$ and then some MCMC method on the partition function gradient.

We can also use a method that gives us a lower-bound on $\tilde{p}$ for the positive phase (like variational inference). Other methods in this chapter will not be able to use bounds, as they require more information about the distribution.

4 Pseudolikelihood

We can bypass $Z(\Theta)$ entirely by computing ratios of probability:

$$\frac{p(x)}{p(y)} = \frac{\frac{1}{Z}\tilde{p}(x)}{\frac{1}{Z}\tilde{p}(y)} = \frac{\tilde{p}(x)}{\tilde{p}(y)}$$

To describe further, suppose we partition x into a, b, and c.

$$p(a|b) = \frac{p(a,b)}{p(b)} = \frac{p(a,b)}{\sum_{a,c} p(a,b,c)} = \frac{\tilde{p}(a,b)}{\sum_{a,c} \tilde{p}(a,b,c)}$$

if $|a|$ or $|c|$ large, we would need to marginalize lots of variables. We can move c into b to reduce the number of evaluations. Via the chain rule of probability we know that:

$$\log p(x) = \log p(x_1) + \log p(x_2|x_1) + ... + \log p(x_n|x_{1:n-1})$$

This then yields the **pseudo-likelihood**:

$$\sum_{i=1}^{n} \log p(x_i|x_{-i})$$

Intuition. If each random variable has k different variables, then this only requires kn evaluations rather than k^n necessary to compute the partition function.

Estimation by maximizing pseudolikelihood is asymptotically consistent, meaning that:

$$\lim_{n\to\infty} \arg\max_{\Theta} \sum_{i=1}^{n} \log p(x_i|x_{-i}) = \Theta^*_{MLE} \tag{1}$$

4.1 Generalized Pseudolikelihood Estimator

We can generalize Pseudolikelihood by considering m different sets: $S^{(i)}$, i = (1,...,m). When $m = 1$ we recover log-likelihood, and when $m = n$, we recover regular pseudolikelihood

$$\sum_{i=1}^{m} \log p(x_{S^{(i)}}|x_{-S^{(i)}}) \tag{2}$$

Practical Considerations. Pseudolikelihood tends to perform poorly on tasks that require a good model of the full joint $p(x)$, like density estimation and sampling. Pseudolikelihood tends to perform better than Maximum Likelihood Estimation on imputation and tasks requiring only conditional distributions. It is also good for data with structure, like images, and for training single-layer models (or deeper models with approximation inference methods).

Limitations. Pseudolikelihood is one of the methods that cannot be used in variational inference, or other methods that only give bounds on $\tilde{p}(x)$. It also has much greater complexity than Stochastic Maximium Likelihood due to computing all of the conditional distributions. Generalized Pseudolikelihood can perform decently when only one conditional is computed per example.

5 Score Matching and Ratio Matching

Another way of training a model without estimating Z nor $\nabla_\Theta Z$ is by minimizing the expected square difference in $\nabla x \log p_{model}(x;\Theta)$ and $\nabla_x \log p_{data}(x;\Theta)$, where the **score** is $\nabla_x \log p(x)$. We can define our task as:

$$\begin{aligned} L(x;\Theta) &= \frac{1}{2}\|\nabla_x \log p_{model}(x;\Theta) - \nabla_x \log p_{data}(x;\Theta)\|_2^2 \\ &= \sum_{j=1}^{n}(\frac{d^2}{d_{x_j}^2}\log p_{model}(x;\Theta) + \frac{1}{2}(\frac{d}{d_{x_j}}\log p_{model}(x;\Theta))^2) \\ J(\Theta) &= \frac{1}{2}\mathop{\mathbb{E}}_{p_{data}(x)} L(x;\Theta) \\ \Theta^* &= \min_\Theta J(\Theta) \end{aligned}$$

Practical considerations. A benefit of using score matching is that we do not need to know the true data generating process. However, the structure of the likelihood function makes it inapplicable to discrete data (since the gradients are with respect to x). Addtionally, score-matching can help pretrain the first hidden layer of a deeper model, but since deeper hidden layers tend to contain discrete variables, it is not helpful for deeper layers.

Like with pseudolikelihood, score matching needs to evaluate $log\tilde{p}(x)$ and the respective gradients, meaning that approximations that only give bounds are not applicable. Generalized score matching does not work in high-dimensional discrete spaces where the observed probability of many events is zero. This is where we can apply **ratio matching**.

Ratio matching applies specifically to binary data, and is defined as:

$$L^{(RM)}(x;\Theta) = \sum_{j=1}^{n}(\frac{1}{1+\frac{p_{model}(x;\Theta)}{p_{model}(f(x,j;\Theta))}}) \tag{3}$$

Where $f(x,j)$ returns x where the bit at position j is flipped. It has been found that ratio matching outperforms Stochastic Maximum Likelihood, Pseudolikelihood, and Generalized Score Matching in terms of the ability of models trained with ratio matching to denoise test set images.

Intuition. Ratio matching, like pseudolikelihood, can be thought of as pushing down on all fantasy states that have only one variable different from training data.

Practical Consideration. Ratio matching can be useful as a basis for dealing with high-dimensional sparse data, e.g., word count vectors. This kind of data is hard for MCMC-based methods because it will keep yielding dense values until it has learned to represent the sparsity in the data distribution. This is explored more in [1].

5.1 Denoising Score Matching

We can regularize score matching by introducing a **corrpution process** $q(x|y)$:

$$p_{smoothed}(x) = \int p_{data}(y)q(x|y)dy \tag{4}$$

Practical Consideration. Since we do not usually have access to the true p_{data}, any consistent estimator will eventually turn p_{model} into a set of Dirac distributions centered on the training data. This is where the regularization by q can help.

NB: Several autoencoder training algorithms are equivalent to Score Matching or Denoising Score Matching, which help overcome the same Z problem.

6 Noise-Contrastive Estimation

Up until now we have seen

- Stochastic Maximum Likelihood and Contrastive Divergence estimate the gradient of $\log Z$.
- Score Matching and Pseudolikelihood avoid Z altogether.

Noise-Contrastive Estimation (NCE) represents p_{model} as:

$$\log p_{model}(x) = \log \tilde{p}_{model}(x; \Theta) + c \tag{5}$$

where c is an approximation of $-\log Z(\Theta)$. NCE works by estimating Θ and c simultaneously. $\log p_{model}(x)$ may not correspond exactly to a valid probability distribution, but it will be more and more valid as the estimate of c improves.

NB: NCE can be good for tractable partition functions too.

Intuition. NCE works by removing the unsupervised learning problem of estimating $p(x)$ to a supervised binary classification problem where one of the categories corresponds to the data generated by the model.

It works via the following:

1. Introduce a (tractable) noise distribution $p_{noise}(x)$

2. Construct a new model over x and a binary "switch" variable y s.t.

$$p_{joint}(y=1) = 0.5$$
$$p_{joint}(x|y=1) = p_{model}(x)$$
$$p_{joint}(x|y=0) = p_{noise}(x)$$

3. Similarly we construct a switch variable for p_{train} and p_{noise}

We can now find MLE of this supervised learning problem via:

$$\Theta, c = \arg\max_{\Theta, c} \mathop{\mathbb{E}}_{x,y\sim p_{train}} \log p_{joint}(y|x)$$

The p_{joint} is now essentially a logistic model:

$$\begin{aligned} p_{joint}(y=1|x) &= \frac{p_{model}(x)}{p_{model}(x) + p_{noise}} \\ &= \frac{1}{1 + \frac{p_{noise}(x)}{p_{model}(x)}} \\ &= \frac{1}{1 + \exp(\log(\frac{p_{noise}(x)}{p_{model}(x)}))} \\ &= \sigma(-\log \frac{p_{noise}(x)}{p_{model}(x)}) \\ &= \sigma(\log p_{model}(x) - \log p_{data}(x)) \end{aligned}$$

Practical Considerations. NCE is simple to apply as long as $\log \tilde{p}_{model}$ is easy to compute (p_{noise} is easy by construction). NCE works best when there are few random variables, but does work well on random variables that take on a lot of values (e.g., conditional distribution of a word conditioned on its context). Like Score Matching and Pseudolikelihood, NCE does not work if there is only a lower bound on $\tilde{p}$.

Example. A simplified version of NCE was also presented in [4] as the Negative Sampling objective function. The Skipgram model's original objective function as defined by the authors is to maximize:

$$\frac{1}{T}\sum_{t=1}^{T} \sum_{-c\leq j\leq c, j\neq 0} \log p(w_{t+j}|w_t)$$

where

$$p(w_{t+j}|w_t) = p(w_O|w_I) = \frac{exp(v'^{T}_{w_O} v_{w_I})}{\sum_{w=1}^{W} exp(v'^{T}_{w} v_{w_I}}$$

As we know, it can be complicated to compute this gradient, as it is proportional to the size of the data. The authors use the following objective instead of a usual NCE approach:

$$\log \sigma(v_{w_O}^{'T} v_{w_I}) + \sum_{i=1}^{K} \mathop{\mathbb{E}}_{w_i \sim p_n(w)} [\log \sigma(-v_{w_i}^{'T} v_{w_I}]$$

This replaces every $\log p(w_O|w_I)$ in the Skipgram model. This allows us to use samples from the noise distribution without having to use any of the associated probability values (as in NCE).

Extensions. **Self-contrastive estimation** is NCE where the model distribution is copied to define the new noise distribution before each step. Intuitively, it is distinguishing reality from its own evolving beliefs, rather than just a fixed baseline (sounds a lot like GANs!).

7 Estimating the Partition Function

It might be the case that we wish to compute the normalized likelihood of the data (e.g., for monitoring training performance). For this we will need to estimate $Z(\Theta)$. For instance, assume we have $p_A(x;\Theta_A) = \frac{1}{Z_A}\tilde{p}_A(x;\Theta_A)$ and $p_A(x;\Theta_A) = \frac{1}{Z_B}\tilde{p}_B(x;\Theta_B)$. Suppose we have a test sample of m examples $x^{(1)}, ..., x^{(m)}$. If $\sum_i \log p_A(x^{(i)};\Theta_A) - \sum_i \log p_B(x^{(i)};\Theta_B) > 0$, then model A > B, but to evaluate the expression we need the partition function. We can rearrange things so that we only need the ratio, but we still **indirectly** need some estimate of the partition function:

$$\sum_i \log p_A(x^{(i)};\Theta_A) - \sum_i \log p_B(x^{(i)};\Theta_B) = \sum_i (\log \frac{\tilde{p}_A(x^{(i)};\Theta_A)}{\tilde{p}_B(x^{(i)};\Theta_B)}) - m \log \frac{Z(\Theta_A)}{Z(\Theta_B)}$$

We estimate the ratio using different **importance sampling** routines.

7.1 Simple importance sampling

let $p_0 = \frac{1}{Z_0}\tilde{p}_0(x)$ be a tractable sampling and evaluation proposal distribution (something we will try to get to match p_1).

$$\begin{aligned} Z_1 &= \int \tilde{p}_1(x)dx \\ &= \int \frac{p_0(x)}{p_0(x)}\tilde{p}_1(x)dx \\ &= Z_0 \int p_0(x)\frac{\tilde{p}_1(x)}{\tilde{p}_0(x)}dx \\ \implies \tilde{Z}_1 &= \frac{Z_0}{K}\sum_{k=1}^{K}\frac{\tilde{p}_1(x^{(k)})}{\tilde{p}_0(x^{(k)})} s.t. x^{(k)} \sim p_0 \end{aligned}$$

Our ratio estimate $\tilde{r} = \frac{1}{K}\sum_{k=1}^{K}\frac{\tilde{p}_1(x^{(k)})}{\tilde{p}_0(x^{(k)})} x^{(k)} \sim p_0$

Practical considerations. If p_0 is close to p_1 then the above is a good estimator. However p_1 is usually complicated and high-dimensional, so most samples from p_0 end up being low under p_1. Because of this, the estimate $\tilde{Z}_1$ also tends to have high variance. When p_0 is not close to p_1 we can use other methods such as **Annealed Importance Sampling**.

7.2 Annealed Importance Sampling

When $D_{KL}(p_0 \| p_1)$ is high, we can make use of Annealed Importance Sampling (AIS). The intuition behind this approach is that we try to bridge the gap between p_0 and p_1 with a bunch of intermediate distributions. We essentially learn the ratio $\frac{Z_1}{Z_0}$ as $\Pi_{j=0}^{n-1} \frac{Z_{\eta_{j+1}}}{Z_{\eta_j}}$. As long as p_{η_j} and $p_{\eta_{j+1}}$ we can use simple importance sampling to get each $\frac{Z_{\eta_{j+1}}}{Z_{\eta_j}}$.

Practical considerations. A common and popular choice for intermediate distributions is

$$p_{\eta_j} \alpha p_1^{\eta_j} p_0^{1-\eta_j}$$

In order to sample from such a distribution we define Markov chain transition functions $T_{\eta_j}(x'|x)$ s.t. $p_{\eta_j}(x) = \int p_{\eta_j}(x') T_{\eta_j}(x|x') dx'$ This allows us to define our AIS sampling strategy:

Algorithm 5: AIS Sampling Strategy

```
1 for k = 1:K do
2     $x_{\eta_1}^{(k)} \sim p_0(x) x_{\eta_2}^{(k)} \sim T_{\eta_1}(x_{\eta_2}^{(k)} | x_{\eta_1}^{(k)})...$
3     $x_{\eta_n}^{(k)} \sim T_{\eta_{n-1}}(x_{\eta_n}^{(k)} | x_{\eta_{n-1}}^{(k)})$
4 end
```

The importance weight for any instance x_k is $w^{(k)} = \Pi_{j=0}^{n-1} \frac{\tilde{p}_{\eta_{j+1}}(x_{\eta_{j+1}}^{(k)})}{\tilde{p}_{\eta_j}(x_{\eta_{j+1}}^{(k)})}$ (using $\log w^{(k)}$ for stability when necessary). We can approximate the ratio $\frac{Z_1}{Z_0} \approx \frac{1}{K} \sum_{k=1}^{K} w^{(k)}$.

Practical Considerations. AIS is computationally intensive to run, and as such if the divergence between p_0 and p_1 is not that great, you can use something like **Bridge sampling**.

7.3 Bridge Sampling

Instead of chaining together a bunch of intermediate distributions we can use just one p_*, called the **bridge distribution**.

$$\frac{Z_1}{Z_0} \approx \frac{\sum_{k=1}^{K} \frac{\tilde{p}_*(x_0^{(k)})}{\tilde{p}_0(x_0^{(k)})}}{\sum_{k=1}^{K} \frac{\tilde{p}_*(x_1^{(k)})}{\tilde{p}_1(x_1^{(k)})}} \tag{6}$$

We know that the optimal bridge distribution is:

$$p_*^{(opt)} \propto \frac{\tilde{p}_0(x)\tilde{p}_1(x)}{r\tilde{p}_0(x) + \tilde{p}_1(x)} \tag{7}$$

The strategy for achieving it is to estimate the ratio $\tilde{r}$ and iterate.

Practical Considerations. If there is a large overlap of support between p_* and p_0, as well as between p_* and p_1, then the KL divergence between p_0 and p_1 can be much larger than for standard importance sampling.

Extensions. Linked Importance Sampling, described by [5], connects AIS and Bridge sampling to allow for a more flexible learning method that is not as sensitive to the KL-divergence between p_0 and p_1. [2] makes use of bridge sampling estimates of short Markov Chains (from parallel tempering), with AIS estimates overtime to **directly** estimate the Z of an RBM every iteration.

References

[1] Y. Dauphin and Y. Bengio. Stochastic ratio matching of rbms for sparse high-dimensional inputs. In *Advances in Neural Information Processing Systems*, pages 1340–1348, 2013.

[2] G. Desjardins, Y. Bengio, and A. C. Courville. On tracking the partition function. In *Advances in neural information processing systems*, pages 2501–2509, 2011.

[3] S. Ermon. Gibbs sampling. `https://ermongroup.github.io/cs323-notes/probabilistic/gibbs/`.

[4] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[5] R. M. Neal. Estimating ratios of normalizing constants using linked importance sampling. *arXiv preprint math/0511216*, 2005.