The Partition Function

Sargur N. Srihari
srihari@cedar.buffalo.edu

Topics

- Definition of Partition Function

1. The log-likelihood gradient
2. Stochastic maximum likelihood and contrastive divergence
3. Pseudolikelihood
4. Score matching and Ratio matching
5. Denoising score matching
6. Noise-contrastive estimation
7. Estimating the partition function

Definition of Partition Function

- Undirected PGMs are defined by an unnormalized probability distribution $\tilde{p}(\boldsymbol{x}, \boldsymbol{\theta})$
  - which we must normalize by dividing by a *partition function* $Z(\boldsymbol{\theta})$ to obtain a valid probability distribution

$$p(\boldsymbol{x}, \boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \tilde{p}(\boldsymbol{x}, \boldsymbol{\theta})$$

  - Where the partition function is either an integral or sum over unnormalized probabilities

$$Z(\boldsymbol{\theta}) = \int \tilde{p}(\boldsymbol{x}, \boldsymbol{\theta}) d\boldsymbol{x} \qquad Z(\boldsymbol{\theta}) = \sum_{x} \tilde{p}(\boldsymbol{x}, \boldsymbol{\theta})$$

  - This operation is intractable for many models

Dealing with Partition Functions

- Three approaches:
  1. Use a model with a tractable partition function
  2. Model is used in ways where $p(\boldsymbol{x})$ is not computed at all
  3. Directly confront the challenge of intractable partition functions
- We consider techniques for training and evaluating models with intractable partition functions

1. Log-likelihood gradient

- Learning undirected models is difficult because:
  - Partition function depends on parameters
- Gradient of the log-likelihood wrt parameters has a term corresponding to gradient of partition function

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

- Corresponds to positive and negative phase of learning

Tractability: Positive, Negative phases

- For most undirected models: negative phase is difficult
- Models with no latent variables or few interactions between latent variables have a tractable positive phase
- RBM: straight-forward positive phase, difficult negative phase
- Here we look at difficulties with the negative phase

Gradient of $\log Z$

$$
\begin{aligned}
&\nabla_{\boldsymbol{\theta}} \log Z \\
&= \frac{\nabla_{\boldsymbol{\theta}} Z}{Z} \\
&= \frac{\nabla_{\boldsymbol{\theta}} \sum_{\mathbf{x}} \tilde{p}(\mathbf{x})}{Z} \\
&= \frac{\sum_{\mathbf{x}} \nabla_{\boldsymbol{\theta}} \tilde{p}(\mathbf{x})}{Z}.
\end{aligned}
$$

- For models that guarantee $p(\boldsymbol{x})>0$ for all x we can substitute $\exp(p\sim(\boldsymbol{x}))$ for $p\sim(\boldsymbol{x})$

$$
\begin{aligned}
&\frac{\sum_{\mathbf{x}} \nabla_{\boldsymbol{\theta}} \exp\left(\log \tilde{p}(\mathbf{x})\right)}{Z} \\
&= \frac{\sum_{\mathbf{x}} \exp\left(\log \tilde{p}(\mathbf{x})\right) \nabla_{\boldsymbol{\theta}} \log \tilde{p}(\mathbf{x})}{Z} \\
&= \frac{\sum_{\mathbf{x}} \tilde{p}(\mathbf{x}) \nabla_{\boldsymbol{\theta}} \log \tilde{p}(\mathbf{x})}{Z} \\
&= \sum_{\mathbf{x}} p(\mathbf{x}) \nabla_{\boldsymbol{\theta}} \log \tilde{p}(\mathbf{x}) \qquad = \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})} \nabla_{\boldsymbol{\theta}} \log \tilde{p}(\mathbf{x}).
\end{aligned}
$$

2. Stochastic Maximum Likelihood

- Naiive way of implementing

$$= \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})} \nabla_{\boldsymbol{\theta}} \log \tilde{p}(\mathbf{x}).$$

- Is to compute it by burning a set of Markov chains from a random initialization every time the gradient is needed
- When learning is performed by SGD this means chains must be burned in once per gradient step
- This approach leads to following training procedure

Naiive MCMC algorithm

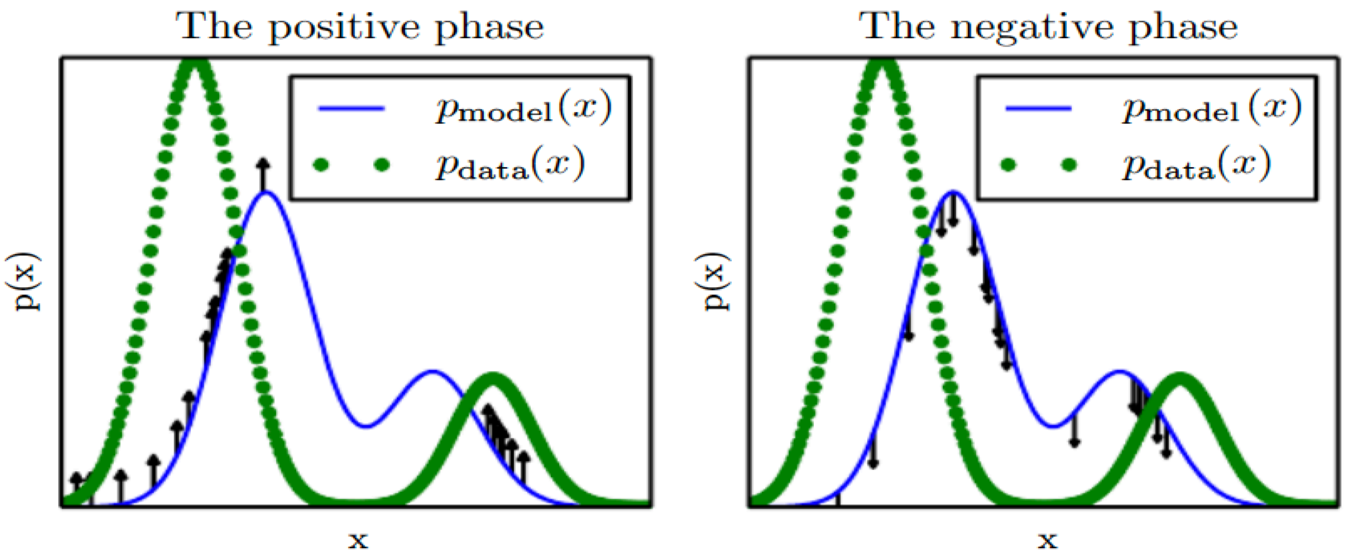
- Maximizing log-likelihood with an intractable partition function using gradient ascent

Set ϵ, the step size, to a small positive number.
Set k, the number of Gibbs steps, high enough to allow burn in. Perhaps 100 to train an RBM on a small image patch.
while not converged **do**
  Sample a minibatch of m examples $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\}$ from the training set.
  $\mathbf{g} \leftarrow \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} \log \tilde{p}(\mathbf{x}^{(i)}; \boldsymbol{\theta})$.
  Initialize a set of m samples $\{\tilde{\mathbf{x}}^{(1)}, \ldots, \tilde{\mathbf{x}}^{(m)}\}$ to random values (e.g., from a uniform or normal distribution, or possibly a distribution with marginals matched to the model's marginals).
  for $i = 1$ to k **do**
    for $j = 1$ to m **do**
      $\tilde{\mathbf{x}}^{(j)} \leftarrow \text{gibbs\_update}(\tilde{\mathbf{x}}^{(j)})$.
    end for
  end for
  $\mathbf{g} \leftarrow \mathbf{g} - \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} \log \tilde{p}(\tilde{\mathbf{x}}^{(i)}; \boldsymbol{\theta})$.
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \epsilon \mathbf{g}$.
end while

MCMC: balance between forces

- Can view MCMC approach as trying to balance between two forces
  - One pushing up on the model distribution where the data occurs
  - Another pushing down on the model distribution where the model samples occur
- Next figure illustrates this process
- The two forces correspond to
  - Maximizing $\log \tilde{p}$
  - Minimizing $\log Z$

Phases of MCMC Algorithm



Positive Phase:
We sample points from the data distribution and push up on their unnormalized probability

Negative Phase:
We sample points from the model distribution and push down on their unnormalized probability.
This counteracts positive phase's tendency to just add a constant to the unnormalized probability everywhere

When data distribution and model distribution are equal, the positive phase has the Same chance to push up at a point as the negative phase has to push down, When this happens, there is no longer any gradient (in expectation) and training must terminate.

Interpreting the negative phase

- Negative phase involves drawing samples from the model's distribution
  – Can think of it as finding points that the model believes in strongly
- Because the negative phase acts to reduce the probability of those points, they are generally considered to represent the model's incorrect beliefs about the world
- Referred to as *hallucinations* or *fantasy particles*

Neuroscientific analogy

- Negative phase proposed as explanation of dreaming in humans and other animals
  - Brain maintains a probabilistic model of the world
    - Follows the gradient of $\log \tilde{p}$ while experiencing real events while awake
    - Follows the negative gradient of $\log \tilde{p}$ to minimize $\log Z$ while sleeping and experiencing events sampled from the present model
- Not proven with neuroscientific experiments
  - In ML it is necessary to use positive and negative phases simultaneously
    - Rather than separate periods of wakefulness and REM sleep

A design less expensive than MCMC

- Given positive/negative phases of learning
  - We can design a less expensive alternative to naiive MCMC
- Main cost of naiive MCMC:
  - Cost of burning-in the Markov chains from a random initialization at each step
- Natural solution:
  - Initialize Markov chains from a distribution that is very close to the model distribution
    - So that burn-in operation does not take many steps

Contrastive Divergence algorithm

- Initializes the Markov chain at each step with samples from the data distribution
  - This is presented in algorithm given next
    - Obtaining samples from data distribution is free
    - Because they are already in the data set
- Initially data distribution is not close to model distribution, so the negative phase is inaccurate
  - Positive phase can increase model's data probability
    - After positive phase has time to act, the model distribution is closer to the data distribution
    - And the negative phase starts to become accurate

Contrastive Divergence Algorithm

- Using gradient ascent for optimization

Set ϵ, the step size, to a small positive number.
Set k, the number of Gibbs steps, high enough to allow a Markov chain sampling from $p(\mathbf{x};\boldsymbol{\theta})$ to mix when initialized from p_{data}. Perhaps 1-20 to train an RBM on a small image patch.
while not converged **do**
  Sample a minibatch of m examples $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\}$ from the training set.
  $\mathbf{g} \leftarrow \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} \log \tilde{p}(\mathbf{x}^{(i)};\boldsymbol{\theta})$.
  for $i = 1$ to m **do**
    $\tilde{\mathbf{x}}^{(i)} \leftarrow \mathbf{x}^{(i)}$.
  end for
  for $i = 1$ to k **do**
    for $j = 1$ to m **do**
      $\tilde{\mathbf{x}}^{(j)} \leftarrow \text{gibbs\_update}(\tilde{\mathbf{x}}^{(j)})$.
    end for
  end for
  $\mathbf{g} \leftarrow \mathbf{g} - \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} \log \tilde{p}(\tilde{\mathbf{x}}^{(i)};\boldsymbol{\theta})$.
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \epsilon\mathbf{g}$.
end while