Ch. 18: Confronting the partition function

Jerónimo Arenas-García

Universidad Carlos III de Madrid

jeronimo.arenas@uc3m.es

May 7, 2018

Contents

Monte Carlo Methods

Monte Carlo Sampling

- Evaluate expectations using the following approximation:

$$\mathbb{E}_p\{f(\mathbf{x})\} = \int p(\mathbf{x})f(\mathbf{x})d\mathbf{x} \approx \frac{1}{n}\sum_{i=1}^{n} f(\mathbf{x}^{(i)})$$

- Requires sampling from $p(\mathbf{x})$
- Unbiased estimator, variance scales with $\frac{1}{\sqrt{n}}$

Markov Chain Montecarlo

- Initialize $\mathbf{x}^{(0)}$
- For $n = 1, 2, \ldots,$ sample $\mathbf{x}^{(n)}$ from $q(\mathbf{x}|\mathbf{x}^{(n-1)})$

If $q(\mathbf{x}|\mathbf{x}^{(n-1)})$ is properly designed after a burning period the stationary distribution converges to the desired distribution $p(\mathbf{x})$.

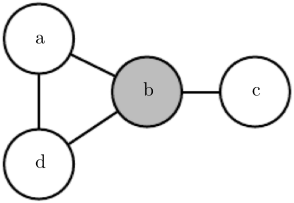
Gibbs Sampling

Can be used when sampling from $p(\mathbf{x})$ is not possible, but we can sample from $p(x_i|\mathbf{x}_{-i})$

```
- Initialize $\mathbf{x}^{(0)}$
- For $n = 1, 2, \ldots$:
    - For $i = 0, \ldots, d$:
          Sample $x_i^{(n)}$ from $p(x_i|x_0^{(n)}, x_1^{(n)}, \ldots, x_{i-1}^{(n)}, x_{i+1}^{(n-1)}, \ldots x_d^{(n-1)})$
    - Output sample $\mathbf{x}^{(n)}$
```

- Discard first samples due to burning period
- To get i.i.d. samples, use only samples *sufficiently* far away
- To accelerate calculation, run multiple chains in parallel

Undirected graph models



- Some of these variables may be observed, others may be latent variables
- Defines a probabilistic model based on the cliques in the graph:

$$\tilde{p}(\mathbf{x}, \mathbf{h}) = \Pi_j \phi(\mathcal{C}_j) = \phi(a, b, d)\phi(b, c)$$

- Partition function: $Z = \int \tilde{p}(\mathbf{x}, \mathbf{h}) d\mathbf{x} d\mathbf{h}$

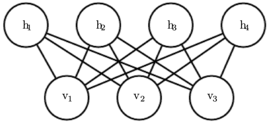
- It is common to sample from $p(\mathbf{x}, \mathbf{h})$ using Gibbs sampling, because for each variable we only need to condition on its neighbors.
- However, in order to do that, we need to deal with the partition function.

Energy-based models

Energy-based models (EBM)

$$\tilde{p}(\mathbf{x}, \mathbf{h}) = \Pi_j \phi(\mathcal{C}_j) = \Pi_j \exp\left[-E\mathcal{C}_j\right] = \exp\left[-\sum_j E(\mathcal{C}_j)\right]$$

Restricted Boltzmann Machine (RBM)



- $E(\mathbf{v}, \mathbf{h}; \boldsymbol{\theta}) = -\mathbf{b}^\top \mathbf{v} - \mathbf{c}^\top \mathbf{h} - \mathbf{v}^\top \mathbf{W} \mathbf{h}$
- $\boldsymbol{\theta} = [\mathbf{b}, \mathbf{c}, \mathbf{W}]$ are hyperparameters
- Derivatives with respect to $\boldsymbol{\theta}$ are simple

Efficient Gibbs sampling is favored by the fact that

$$p(\mathbf{h}|\mathbf{v}) = \Pi_i p(h_i|\mathbf{v})$$
$$p(\mathbf{v}|\mathbf{h}) = \Pi_i p(v_i|\mathbf{h})$$

(All h_i / v_i can be sampled in parallel)

Contents

Confronting the partition function

Some deep learning designs require the optimization of a probabilistic model characterized by an undirected graph. For instance, for the previous RBM configuration, the problem could be stated as:

$$\boldsymbol{\theta}^{o} = \arg\max_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) = \arg\max_{\boldsymbol{\theta}} \log p(\mathbf{x}, \mathbf{h}; \boldsymbol{\theta}) = \arg\max_{\boldsymbol{\theta}} \log \frac{\tilde{p}(\mathbf{x}, \mathbf{h}; \boldsymbol{\theta})}{Z(\boldsymbol{\theta})}$$

For many interesting models, computing $Z(\boldsymbol{\theta})$ is intractable:

- Consider just models with tractable $Z(\boldsymbol{\theta})$, or models that do not require its computation (Chapter 20)
- Techniques for training and evaluating models with intractable $Z(\boldsymbol{\theta})$ (this chapter)

Contents

In order to maximize the likelihood, we need to compute its gradient with respect to the model hyperparameters:

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

- Positive phase: $\nabla_{\boldsymbol{\theta}} \log \tilde{p}(\mathbf{v}, \mathbf{h}; \boldsymbol{\theta})$ Computation of these derivatives is very easy for many models, such as RBMs.
  (Chapter 19 deals with situations in which this positive phase is complicated).
- Negative phase:

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

For models with $p(\mathbf{x}) > 0, \forall x$ (such as EBM):

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

Maximizing the likelihood

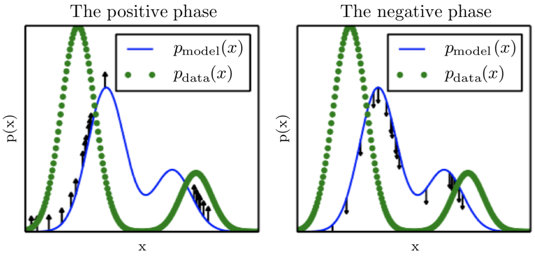
- The previous result holds also for continuous distributions
- The gradient vector then becomes:

$$\nabla_{\boldsymbol{\theta}} logp(\mathbf{x};\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \log \tilde{p}(\mathbf{v},\mathbf{h};\boldsymbol{\theta}) - \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})} \nabla_{\boldsymbol{\theta}} \log \tilde{p}(\mathbf{v},\mathbf{h};\boldsymbol{\theta})$$

- The negative phase requires an expectation for which we can recur to MCMC approximations

Maximizing the likelihood (II)

Moving in the direction of this gradient consists of two phases:



- (+) Phase: For a chunk of data, $\boldsymbol{\theta}$ is modified to increase the unnormalized log-probability for that chunk
- (-) Phase: We decrease the log-probability in the areas where the model ($\tilde{p}/Z$) currently has large probability

As a result, the model will only keep a large pdf value in areas populated with training data.

Pure Gibbs Sampling Method

Using Gibbs sampling with *m* chains running in parallel, and random initialization of the chains at each iteration results in this algorithm:

Algorithm 18.1 A naive MCMC algorithm for maximizing the log-likelihood with an intractable partition function using gradient ascent.

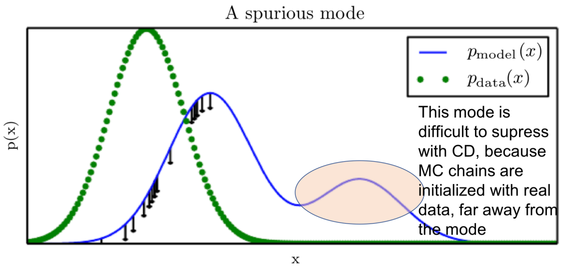
```
Set $\epsilon$, the step size, to a small positive number.
Set $k$, the number of Gibbs steps, high enough to allow burn in. Perhaps 100 to train an RBM on a small image patch.
while not converged do
  Sample a minibatch of $m$ examples $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\}$ from the training 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
      $\tilde{\mathbf{x}}^{(j)} \leftarrow$ 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
```

Pure Gibbs Sampling Implementation (II)

- The previous implementation is the direct application of chunk-based gradient algorithms for the maximization of the likelihood.
- Inefficient because of the time required to burn in the MC chains at each iteration
- Approximations to this algorithm are based on:
  - Trying to reduce the burn in interval for the MC chains
  - Focusing on reducing the probability of the model at the wrong locations during the negative phase (rather than just on regions with large probability)

Contrastive Divergence (CD)

- Initializes the Markov chains at each step with samples from the data distribution
- Alg 18.2. in the book assumes the same points used for the positive phase, but other points could be sampled for the negative phase.
- Advantage: The burn in phase can be significantly shorter
- Potential drawback: wrong regions where the model has large probability will be not visited by the MC chain, and may retain large probability after several iterations



Contrastive Divergence (II): Discussion

- CD has been experimentally shown to be biased for RBM and fully visible Boltzmann Machines, in the sense of not converging to the ML solution
- However, the bias is small, so it can be used for a first rough approximation phase, followed by fine tuning with more precise MCMC methods.
- Useful for shallow models (like RBMs) but not for deeper structures, because samples from the not visible variables are not available.

Stochastic Maximum Likelihood (SML)

- Also known as Persistent CD (PCD)
- Hypothesis: $p(\mathbf{x}; \boldsymbol{\theta})$ changes slowly between iterations
- Initializes the Markov chains at each step with their values from the previous gradient step (which are assumed to be a good approximation to samples generated by the new model after the last gradient update)
- With respect to CD:
  - It is more resistant to spurious modes
  - Can be used with deeper structures, because latent variables are available from previous iteration.
- Fast PCD: A variant that mixes together fast adaptation during the initial iterations and slow adaptation on the long term.

Contents

Pseudolikelihood

Instead of using the likelihood, we maximize to the following cost:

$$
\begin{aligned}
L(\boldsymbol{\theta}) &= \sum_{i=1}^{n} \log p(x_i|\mathbf{x}_{-i}) = \sum_{i=1}^{n} \log \frac{p(\mathbf{x})}{p(\mathbf{x}_{-i})} = \sum_{i=1}^{n} \log \frac{p(\mathbf{x})}{\sum_{x_i} p(\mathbf{x})} \\
&= \sum_{i=1}^{n} \log \frac{\tilde{p}(\mathbf{x})}{\sum_{x_i} \tilde{p}(\mathbf{x})}
\end{aligned}
$$

- Since only ratios of $p(\mathbf{x})$ are involved, $Z(\boldsymbol{\theta})$ cancels out
- *"Pseudolikelihood tends to perform poorly on tasks that require a good model of the full joint* $p(\mathbf{x})$*, [...] it can perform better than ML for tasks that require only the conditional distributions"* (e.g., inputation)
- It cannot be used with lower bounds on $\tilde{p}(\mathbf{x})$, because this function appears also in the denominator of the pseudolikelihood.

Pseudolikelihood (II)

$$L(\boldsymbol{\theta}) = \sum_{i=1}^{n} \log \frac{\tilde{p}(\mathbf{x})}{\sum_{x_i} \tilde{p}(\mathbf{x})}$$

- If x_i can take k values, the denominator implies k evaluations of $\tilde{p}(\mathbf{x})$, giving a total of $k \times n$ evaluations.
- In contrast, calculating the partition function would require k^n evaluations.
- Generalized pseudolikelihood: consider groups of variables rather than each individual variable
    - More expensive than using the pseudolikelihood
    - But a better approximation to ML, especially if the grouping resembles the structure of the data
- This method has much greater cost per gradient step than SML
- It can still perform well (and efficiently) if only one selected conditional probability is used at each step.

Score matching

Based on another design criterion that avoids the need for calculating the partition function:

$$L(\boldsymbol{x}, \boldsymbol{\theta}) = \frac{1}{2} ||\nabla_{\boldsymbol{x}} \log p_{\text{model}}(\boldsymbol{x}; \boldsymbol{\theta}) - \nabla_{\boldsymbol{x}} \log p_{\text{data}}(\boldsymbol{x})||_2^2$$

$$J(\boldsymbol{\theta}) = \frac{1}{2} \mathbb{E}_{p_{\text{data}}(\boldsymbol{x})} L(\boldsymbol{x}, \boldsymbol{\theta})$$

$$\boldsymbol{\theta}^* = \min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

This criterion can be shown to be equivalent to minimization of the average (over training data) of:

$$\tilde{L}(\boldsymbol{x}, \boldsymbol{\theta}) = \sum_{j=1}^{n} \left(\frac{\partial^2}{\partial x_j^2} \log p_{\text{model}}(\boldsymbol{x}; \boldsymbol{\theta}) + \frac{1}{2} \left(\frac{\partial}{\partial x_j} \log p_{\text{model}}(\boldsymbol{x}; \boldsymbol{\theta}) \right)^2 \right)$$

So that, we do not require to know the real data distribution.

- Since $Z(\boldsymbol{\theta})$ does not depend on **x**, derivation w.r.t. **x** removes any terms depending on such constant

Score matching (II)

- This method requires the derivatives of $p(\mathbf{x})$ and therefore can only be applied with continuous data (latent variables can be discrete though)
- The method cannot be used with lower bounds of $\tilde{p}(\mathbf{x})$, because second order derivatives of the bound do not necessarily reflect the behavior of the model pdf.
- Since deep Boltzmann machines are normally optimized with these bounds, this implies that score matching cannot be used in this context

Ratio matching

Defines a specific cost for binary data: the average over training data of

$$L^{(\mathrm{RM})}(\boldsymbol{x}, \boldsymbol{\theta}) = \sum_{j=1}^{n} \left(\frac{1}{1 + \frac{p_{\mathrm{model}}(\boldsymbol{x}; \boldsymbol{\theta})}{p_{\mathrm{model}}(f(\boldsymbol{x}), j); \boldsymbol{\theta})}} \right)^2$$

This is vector x with the sign of its j-th component flipped

- This cost try to reduce $p(\mathbf{x})$ for all data points that differ with the training examples in just one variable
- As the pseudolikelihood, it avoids the computation of $Z(\boldsymbol{\theta})$ by requiring only ratios of the probability model

Denoising score matching

Replace in score matching $p_{data}(\mathbf{x})$ by the following smoothed distribution

$$p_{\text{smoothed}}(\boldsymbol{x}) = \int p_{\text{data}}(\boldsymbol{y}) q(\boldsymbol{x} \mid \boldsymbol{y}) d\boldsymbol{y}$$

where $q(\mathbf{x}|\mathbf{y})$ is a corruption process, e.g., addition of Gaussian noise.

Noise-contrastive estimation

Considering the decomposition of the log of p_{model}

$$\log p_{\text{model}}(\mathbf{x}) = \log \tilde{p}_{\text{model}}(\mathbf{x}; \boldsymbol{\theta}) + c$$

c has been introduced as a parameter that replaces $-\log Z(\boldsymbol{\theta})$

- Objective: Learn jointly $\boldsymbol{\theta}$ and c
- Standard ML would simply make c to grow unbounded, so a different criterion must be followed

Noise-contrastive estimation (II)

- Introduce a noise distribution and a switching variable y, so that the joint model of $\mathbf{x}$ and y is:

$$p_{\text{joint}}(y = 1) = \frac{1}{2},$$

$$p_{\text{joint}}(\mathbf{x} \mid y = 1) = p_{\text{model}}(\mathbf{x}),$$

$$p_{\text{joint}}(\mathbf{x} \mid y = 0) = p_{\text{noise}}(\mathbf{x}).$$

  and similarly for the training data distribution
- Training samples for $y = 0$ can be generated if $p_{noise}(\mathbf{x})$ is selected to be easy to sample from
- Now, solve the following optimization problem:

$$\boldsymbol{\theta}, c = \arg\max_{\boldsymbol{\theta}, c} \mathbb{E}_{\mathbf{x}, \mathrm{y} \sim p_{\text{train}}} \log p_{\text{joint}}(y \mid \mathbf{x})$$

Noise-contrastive estimation (III)

$$\boldsymbol{\theta}, c = \arg\max_{\boldsymbol{\theta}, c} \mathbb{E}_{\mathbf{x}, \mathrm{y} \sim p_{\text{train}}} \log p_{\text{joint}}(y \mid \mathbf{x})$$

$$p_{joint}(1|\mathbf{x}) = \frac{p_{model}(\mathbf{x})}{p_{model}(\mathbf{x}) + p_{noise}(\mathbf{x})}; \quad p_{joint}(0|\mathbf{x}) = \frac{p_{noise}(\mathbf{x})}{p_{model}(\mathbf{x}) + p_{noise}(\mathbf{x})}$$

- NCE requires that the model pdf is easy to optimize w.r.t. $\boldsymbol{\theta}$, and that $p_{noise}(\mathbf{x})$ is:
  - Easy to sample: in order to generate noise samples for the training data
  - Easy to evaluate: in order to evaluate $p_{joint}(y|\mathbf{x})$
- Now, c cannot simply be increased, because that would decrease $p_{joint}(0|\mathbf{x})$ for the noise samples
- *NCE is based on the idea that a good generative model should be able to distinguish data from noise*

Contents

Estimating the partition function with importance sampling

- In some cases we cannot avoid computation of $Z(\boldsymbol{\theta})$
  - To evaluate a model over a set of test data
  - To monitor training performance
  - To compare two models on a set of test data (in this case, it is sufficient to know the ratio of $Z_A(\boldsymbol{\theta})$ to $Z_B(\boldsymbol{\theta})$)
- We can use importance sampling with a proposal distribution $p_0(\mathbf{x})$ which supports tractable sampling and tractable evaluation:

$$Z_1(\boldsymbol{\theta}_1) = \int \tilde{p}_1(\mathbf{x}) d\mathbf{x} = \int p_0(\mathbf{x}) \frac{\tilde{p}_1(\mathbf{x})}{p_0(\mathbf{x})} d\mathbf{x} \longrightarrow \hat{Z}_1 = \frac{1}{K} \sum_{k=1}^{K} \frac{\tilde{p}_1(\mathbf{x}^{(k)})}{p_0(\mathbf{x}^{(k)})}$$

  where samples used for estimator $\hat{Z}_1$ are taken from $p_0(\mathbf{x})$
- If $p_0(\mathbf{x})$ is not sufficiently close to $p_1(\mathbf{x})$, most elements in the sum will be very small, making $\hat{Z}_1$ a poor estimator.

Bridge sampling

- Use an intermediate distribution between $p_0(\mathbf{x})$ and $p_1(\mathbf{x})$: $p_\star(\mathbf{x})$

$$\frac{Z_1}{Z_0} \approx \sum_{k=1}^{K} \frac{\tilde{p}_*(\boldsymbol{x}_0^{(k)})}{\tilde{p}_0(\boldsymbol{x}_0^{(k)})} \Big/ \sum_{k=1}^{K} \frac{\tilde{p}_*(\boldsymbol{x}_1^{(k)})}{\tilde{p}_1(\boldsymbol{x}_1^{(k)})}$$

Samples taken from $p_0(\mathbf{x})$ — Samples taken from $p_*(\mathbf{x})$

- $p_\star(\mathbf{x})$ needs to be chosen to have a large overlap of support with both $p_0(\mathbf{x})$ and $p_1(\mathbf{x})$
- Optimal selection would be:

$$p_\star^{(opt)} = \frac{\tilde{p}_0(\mathbf{x})\tilde{p}_1(\mathbf{x})}{\frac{Z_1}{Z_0}\tilde{p}_0(\mathbf{x}) + \tilde{p}_1(\mathbf{x})}$$

- Since Z_1 is not known, we need an iterative procedure

Sometimes multiple intermediate distributions are needed: Annealed Importance Sampling