Chapter 19

Approximate inference

TODO: somewhere in this chapter, point out that variational inference implicitly defines a recurrent net, stochastic approximate inference implicitly defines a stochastic recurrent net

Misplaced TODO: discussion of the different directions of the KL divergence, and the effects on ignoring / preserving modes

Many probabilistic models are difficult to train because it is difficult to perform inference in them. In the context of deep learning, we usually have a set of visible variables $\boldsymbol{v}$ and a set of latent variables $\boldsymbol{h}$. The challenge of inference usually refers to the difficult problem of computing $p(\boldsymbol{h} \mid \boldsymbol{v})$ or taking expectations with respect to it. Such operations are often necessary for tasks like maximum likelihood learning.

Many simple graphical models with only one hidden layer, such as restricted Boltzmann machines and probabilistic PCA are defined in a way that makes inference operations like computing $p(\boldsymbol{h} \mid \boldsymbol{v})$ or taking expectations with respect to it simple. Unfortunately, most graphical models with multiple layers of hidden variables, such as deep belief networks and deep Boltzmann machines have intractable posterior distributions. Exact inference requires an exponential amount of time in these models. Even some models with only a single layer, such as sparse coding, have this problem.

Intractable inference problems usually arise from interactions between latent variables in a structured graphical model. See Fig. 19.1 for some examples. These interactions may be due to direct interactions in undirected models or "explaining away" interactions between mutual ancestors of the same visible unit in directed models.

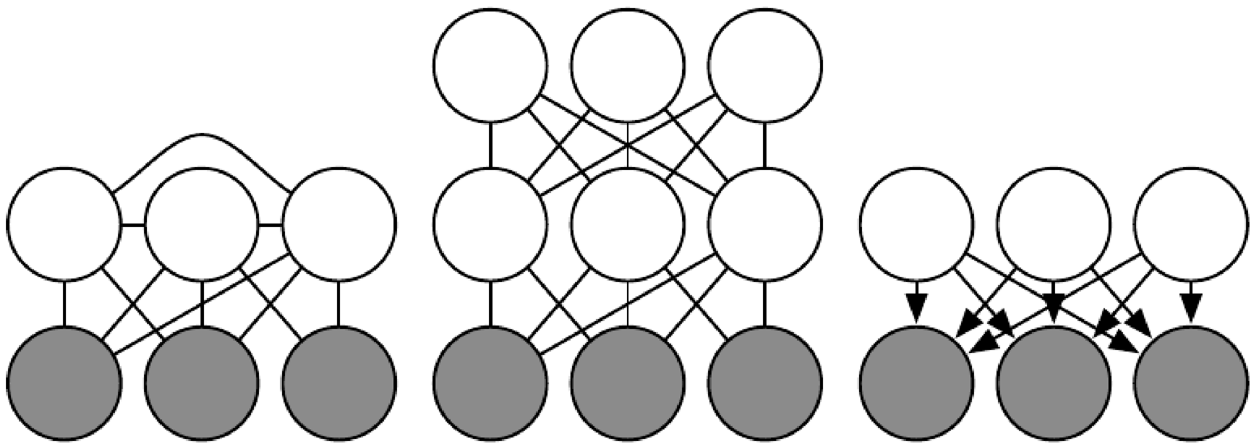

Figure 19.1: Intractable inference problems are usually the result of interactions between latent variables in a structured graphical model. These can be due to direct edges, or due to paths that are activated when the child of a V-structure is observed. Left) A semi-restricted Boltzmann machine with connections between hidden units. These direct connections between latent variables make the posterior distribution complicated. Center) A deep Boltzmann machine, organized into layers of variables without intra-layer connections, still has an intractable posterior distribution due to the connections between layers. Right) This directed model has interactions between latent variables when the visible variables are observed, because ever two latent variables are co-parents. Note that it is still possible to have these graph structures yet have tractable inference. For example, probabilistic PCA has the graph structure shown in the right, yet simple inference due to special properties of the specific conditional distributions it uses (linear-Gaussian conditionals with mutually orthogonal basis vectors). MISPLACED TODO– make sure probabilistic PCA is at least defined somewhere in the book

19.1 Inference as Optimization

Many approaches to confronting the problem of difficult inference make use of the observation that exact inference can be described as an optimization problem.

Specifically, assume we have a probabilistic model consisting of observed variables $\boldsymbol{v}$ and latent variables $\boldsymbol{h}$. We would like to compute the log probability of the observed data, $\log p(\boldsymbol{v};\boldsymbol{\theta})$. Sometimes it is too difficult to compute $\log p(\boldsymbol{v};\boldsymbol{\theta})$ if it is costly to marginalize out $\boldsymbol{h}$. Instead, we can compute a lower bound on it. This bound is called the *evidence lower bound* (ELBO). Other names for this lower bound include the negative *variational free energy* and the negative *Helmholtz free energy*. Specifically, this lower bound is defined as TODO– figure out why I was making q be bm in some but not all places

$$\mathcal{L}(\boldsymbol{v},\boldsymbol{\theta},\boldsymbol{q}) = \log p(\boldsymbol{v};\boldsymbol{\theta}) - D_{\mathrm{KL}}(q(\boldsymbol{h})\|p(\boldsymbol{h}\mid\boldsymbol{v};\boldsymbol{\theta}))$$

where q is an arbitrary probability distribution over $\boldsymbol{h}$.

TODO: the below equations framebust It is straightforward to see that this is a lower bound on $\log p(\boldsymbol{v})$:

$$\begin{aligned}
\ln p(\boldsymbol{v}) &= \ln p(\boldsymbol{v}) + \sum_{\boldsymbol{h}} q(\boldsymbol{h}\mid\boldsymbol{v}) \ln\left(\frac{p(\boldsymbol{v},\boldsymbol{h})}{q(\boldsymbol{h}\mid\boldsymbol{v})}\right) - \sum_{\boldsymbol{h}} q(\boldsymbol{h}\mid\boldsymbol{v}) \ln\left(\frac{p(\boldsymbol{v},\boldsymbol{h})}{q(\boldsymbol{h}\mid\boldsymbol{v})}\right)\\
&= \sum_{\boldsymbol{h}} q(\boldsymbol{h}\mid\boldsymbol{v}) \ln\left(\frac{p(\boldsymbol{v},\boldsymbol{h})}{q(\boldsymbol{h}\mid\boldsymbol{v})}\right) - \sum_{\boldsymbol{h}} q(\boldsymbol{h}\mid\boldsymbol{v}) \ln\left(\frac{p(\boldsymbol{v},\boldsymbol{h})}{q(\boldsymbol{h}\mid\boldsymbol{v})}\right) + \sum_{\boldsymbol{h}} q(\boldsymbol{h}\mid\boldsymbol{v}) \log p(\boldsymbol{v})\\
&= \sum_{\boldsymbol{h}} q(\boldsymbol{h}\mid\boldsymbol{v}) \ln\left(\frac{p(\boldsymbol{v},\boldsymbol{h})}{q(\boldsymbol{h}\mid\boldsymbol{v})}\right) - \sum_{\boldsymbol{h}} q(\boldsymbol{h}\mid\boldsymbol{v}) \left[\ln\left(\frac{p(\boldsymbol{v},\boldsymbol{h})}{q(\boldsymbol{h}\mid\boldsymbol{v})}\right) - \ln p(\boldsymbol{v})\right]\\
&= \sum_{\boldsymbol{h}} q(\boldsymbol{h}\mid\boldsymbol{v}) \ln\left(\frac{p(\boldsymbol{v},\boldsymbol{h})}{q(\boldsymbol{h}\mid\boldsymbol{v})}\right) - \sum_{\boldsymbol{h}} q(\boldsymbol{h}\mid\boldsymbol{v}) \ln\left(\frac{p(\boldsymbol{v},\boldsymbol{h})}{p(\boldsymbol{v})q(\boldsymbol{h}\mid\boldsymbol{v})}\right)\\
&= \sum_{\boldsymbol{h}} q(\boldsymbol{h}\mid\boldsymbol{v}) \ln\left(\frac{p(\boldsymbol{v},\boldsymbol{h})}{q(\boldsymbol{h}\mid\boldsymbol{v})}\right) - \sum_{\boldsymbol{h}} q(\boldsymbol{h}\mid\boldsymbol{v}) \ln\left(\frac{p(\boldsymbol{h}\mid\boldsymbol{v})}{q(\boldsymbol{h}\mid\boldsymbol{v})}\right)\\
&= \mathcal{L}(q) + \mathrm{KL}(q\|p)
\end{aligned}$$

Because the difference $\log p(\boldsymbol{v})$ and $\mathcal{L}(\boldsymbol{v},\boldsymbol{\theta},\boldsymbol{q})$ is given by the KL-divergence and because the KL-divergence is always non-negative, we can see that $\mathcal{L}$ always has at most the same value as the desired log probability, and is equal to it if and only if q is the same distribution as $p(\boldsymbol{h}\mid\boldsymbol{v})$.

Surprisingly, $\mathcal{L}$ can be considerably easier to compute for some distributions q. Simple algebra shows that we can rearrange $\mathcal{L}$ into a much more convenient form:

$$\mathcal{L}(\boldsymbol{v},\boldsymbol{\theta},\boldsymbol{q}) = \log p(\boldsymbol{v};\boldsymbol{\theta}) - D^{\mathrm{KL}}\quad(q(\boldsymbol{h})\|p(\boldsymbol{h}\mid\boldsymbol{v};\boldsymbol{\theta}))$$

$$= \log p(\boldsymbol{v};\boldsymbol{\theta}) - \mathbb{E}_{\boldsymbol{h}\sim q} \frac{\log q(\boldsymbol{h})}{\log p(\boldsymbol{h} \mid \boldsymbol{v})}$$

$$= \log p(\boldsymbol{v};\boldsymbol{\theta}) - \mathbb{E}_{\boldsymbol{h}\sim q} \frac{\log q(\boldsymbol{h})}{\log \frac{p(\boldsymbol{h},\boldsymbol{v})}{p(\boldsymbol{v})}}$$

$$= \log p(\boldsymbol{v};\boldsymbol{\theta}) - \mathbb{E}_{\boldsymbol{h}\sim q} \left[\log q(\boldsymbol{h}) - \log p(\boldsymbol{h},\boldsymbol{v}) + \log p(\boldsymbol{v})\right]$$

$$= \ -\mathbb{E}_{\boldsymbol{h}\sim q} \left[\log q(\boldsymbol{h}) - \log p(\boldsymbol{h},\boldsymbol{v})\right].$$

This yields the more canonical definition of the evidence lower bound,

$$\mathcal{L}(\boldsymbol{v},\boldsymbol{\theta},\boldsymbol{q}) = \mathbb{E}_{\boldsymbol{h}\sim q}\left[\log p(\boldsymbol{h},\boldsymbol{v})\right] + H(q). \tag{19.1}$$

The first term of $\mathcal{L}$ is known as the *energy term*. The second term is known as the *entropy term*. For an appropriate choice of q, both terms can be easy to compute. The only question is how close to $p(\boldsymbol{h} \mid \boldsymbol{v})$ the distribution q will be. This determines how good of an approximation $\mathcal{L}$ will be for $\log p(\boldsymbol{v})$.

We can thus think of inference as the procedure for finding the q that maximizes $\mathcal{L}$. Exact inference maximizes $\mathcal{L}$ perfectly. Throughout this chapter, we will show how many forms of approximate inference are possible. No matter what choice of q we use, $\mathcal{L}$ will give us a lower bound on the likelihood. We can get tighter or looser bounds that are cheaper or more expensive to compute depending on how we choose to approach this optimization problem. We can obtain a poorly matched q but reduce the computational cost by using an imperfect optimization procedure, or by using a perfect optimization procedure over a restricted family of q distributions.

19.2 Expectation Maximization

Expectation maximization (EM) is a popular training algorithm for models with latent variables. It consists of alternating between two steps until convergence:

- The *E-step* (Expectation step): Set $q(\boldsymbol{h}^{(i)}) = p(\boldsymbol{h}^{(i)} \mid \boldsymbol{v}^(i);\boldsymbol{\theta})$ for all indices i of the training examples $\boldsymbol{v}^{(i)}$ we want to train on (both batch and minibatch variants are valid). By this we mean q is defined in terms of the *current* value of $\boldsymbol{\theta}$; if we vary θ then $p(\boldsymbol{h} \mid \boldsymbol{v};\boldsymbol{\theta})$ will change but $q(\boldsymbol{h})$ will not.

- The M-step (Maximization step): Completely or partially maximize $\sum_i \mathcal{L}(\boldsymbol{v}^(i),\boldsymbol{\theta},q)$ with respect to $\boldsymbol{\theta}$ using your optimization algorithm of choice.

This can be viewed as a coordinate ascent algorithm to maximize $\mathcal{L}$. On one step, we maximize $\mathcal{L}$ with respect to q, and on the other, we maximize $\mathcal{L}$ with respect to $\boldsymbol{\theta}$.

Stochastic gradient ascent on latent variable models can be seen as a special case of the EM algorithm where the M step consists of taking a single gradient step. Other variants of the EM algorithm can make much larger steps. For some model families, the M step can even be performed analytically, jumping all the way to the optimal solution given the current q.

Even though the E-step involves exact inference, we can think of the EM algorithm as using approximate inference in some sense. Specifically, the M-step assumes that the same value of q can be used for all values of $\boldsymbol{\theta}$. This will introduce a gap between $\mathcal{L}$ and the true $\log p(v)$ as the M-step moves further and further. Fortunately, the E-step reduces the gap to zero again as we enter the loop for the next time.

The EM algorithm is a workhorse of classical machine learning, and it can be considered to be used in deep learning in the sense that stochastic gradient ascent can be seen as EM with a very simple and small M step. However, because $\mathcal{L}$ can not be analytically maximized for many interesting deep models, the more general EM framework as a whole is typically not explored in the deep learning research community.

TODO–cite the emview paper

19.3 MAP Inference: Sparse Coding as a Probabilistic Model

TODO synch up with other sections on sparse coding

Many versions of sparse coding can be cast as probabilistic models. For example, suppose we encode visible data $\boldsymbol{v} \in \mathbb{R}^n$ with latent variables $\boldsymbol{h} \in \mathbb{R}^m$. We can use a prior to encourage our latent code variables to be sparse:

$$p(h) = TODO.$$

We can define the visible units to be Gaussian with an affine transformation from the code to the mean of the Gaussian:

$$\boldsymbol{v} \sim \mathcal{N}(\boldsymbol{v} \mid \boldsymbol{\mu} + \boldsymbol{W}\boldsymbol{h}, \boldsymbol{\beta}^{-1})$$

where $\boldsymbol{\beta}$ is a diagonal precision matrix to maintain tractability.

Computing $p(\boldsymbol{h} \mid \boldsymbol{v})$ is difficult. TODO explain why

One operation that we can do is perform *maximum a posteriori* (MAP) inference, which means solving the following optimization problem:

$$\boldsymbol{h}^* = \arg\max p(\boldsymbol{h} \mid \boldsymbol{v}).$$

This yields the familiar optimization problem

TODO synch with other sparse coding sections, make sure the other sections talk about using gradient descent, feature sign, ISTA, etc.

This shows that the popular feature extraction strategy for sparse coding can be justified as having a probabilistic interpretation–it may be MAP inference in this probabilistic model (there are other probabilistic models that yield the same optimization problem, so we cannot positively identify this specific model from the feature extraction process).

Excitingly, MAP inference of $\boldsymbol{h}$ given $\boldsymbol{v}$ also has an interpretation in terms of maximizing the evidence lower bound. Specifically, MAP inference maximizes $\mathcal{L}$ with respect to q under the constraint that q take the form form of a Dirac distribution. During learning of sparse coding, we alternate between using convex optimization to extract the codes, and using convex optimization to update $\boldsymbol{W}$ to achieve the optimal reconstruction given the codes. This turns out to be equivalent to maximizing $\mathcal{L}$ with respect to $\boldsymbol{\theta}$ for the q that was obtained from MAP inference. The learning algorithm can be thought of as EM restricted to using a Dirac posterior. In other words, rather than performing learning exactly using standard inference, we learn to maximize a bound on the true likelihood, using exact MAP inference.

19.4 Variational Inference and Learning

One common difficulty in probabilistic modeling is that the posterior distribution $p(\boldsymbol{h} \mid \boldsymbol{v})$ is infeasible to compute for many models with hidden variables $\boldsymbol{h}$ and visible variables $\boldsymbol{v}$. Expectations with respect to this distribution may also be intractable.

Consider as an example the *binary sparse coding* model. In this model, the input $\boldsymbol{v} \in \mathbb{R}^n$ is formed by adding Gaussian noise to the sum of m different components which can each be present or absent. Each component is switched on or off by the corresponding hidden unit in $\boldsymbol{h} \in \{0,1\}^m$:

$$p(h_i = 1) = \sigma(b_i)$$

$$p(\boldsymbol{v} \mid \boldsymbol{h}) = \mathcal{N}(\boldsymbol{v} \mid \boldsymbol{W}\boldsymbol{h}, \beta^{-1})$$

where $\boldsymbol{b}$ is a learn-able set of biases, $\boldsymbol{W}$ is a learn-able weight matrix, and $\boldsymbol{\beta}$ is a learn-able, diagonal precision matrix.

Training this model with maximum likelihood requires taking the derivative with respect to the parameters. Consider the derivative with respect to one of the biases:

$$\frac{\partial}{\partial b_i} \log p(\boldsymbol{v})$$

$$= \frac{\frac{\partial}{\partial b_i} p(\boldsymbol{v})}{p(\boldsymbol{v})}$$

$$= \frac{\frac{\partial}{\partial b_i} \sum_{\boldsymbol{h}} p(\boldsymbol{h}, \boldsymbol{v})}{p(\boldsymbol{v})}$$

$$= \frac{\frac{\partial}{\partial b_i} \sum_{\boldsymbol{h}} p(\boldsymbol{h}) p(\boldsymbol{v} \mid \boldsymbol{h})}{p(\boldsymbol{v})}$$

$$= \frac{\sum_{\boldsymbol{h}} p(\boldsymbol{v} \mid \boldsymbol{h}) \frac{\partial}{\partial b_i} p(\boldsymbol{h})}{p(\boldsymbol{v})}$$

$$= \sum_{\boldsymbol{h}} p(\boldsymbol{h} \mid \boldsymbol{v}) \frac{\frac{\partial}{\partial b_i} p(\boldsymbol{h})}{p(\boldsymbol{h})}$$

$$= \sum_{\boldsymbol{h}} p(\boldsymbol{h} \mid \boldsymbol{v}) \frac{\frac{\partial}{\partial b_i} p(\boldsymbol{h})}{p(\boldsymbol{h})}$$

$$= \mathbb{E}_{\boldsymbol{h} \ p(\boldsymbol{h} \mid \boldsymbol{v})} \frac{\partial}{\partial b_i} \log p(\boldsymbol{h}).$$

This requires computing expectations with respect to $p(\boldsymbol{h} \mid \boldsymbol{v})$. Unfortunately, $p(\boldsymbol{h} \mid \boldsymbol{v})$ is a complicated distribution. See Fig. 19.2 for the graph structure of $p(\boldsymbol{h}, \boldsymbol{v})$ and $p(\boldsymbol{h} \mid \boldsymbol{v})$. The posterior distribution corresponds to the complete graph over the hidden units, so variable elimination algorithms do not help us to compute the required expectations any faster than brute force.

One solution to this problem is to use *variational methods*. Variational methods involve using a simple distribution $q(\boldsymbol{h})$ to approximate the true, complicated posterior $p(\boldsymbol{h} \mid \boldsymbol{v})$. The name "variational" derives from their frequent use of a branch of mathematics called *calculus of variations*. However, not all variational methods use calculus of variations.

TODO variational inference involves maximization of a BOUND TODO variational inference also usually involves a restriction on the function family

TODO

19.4.1 Discrete Latent Variables

TODO– BSC example

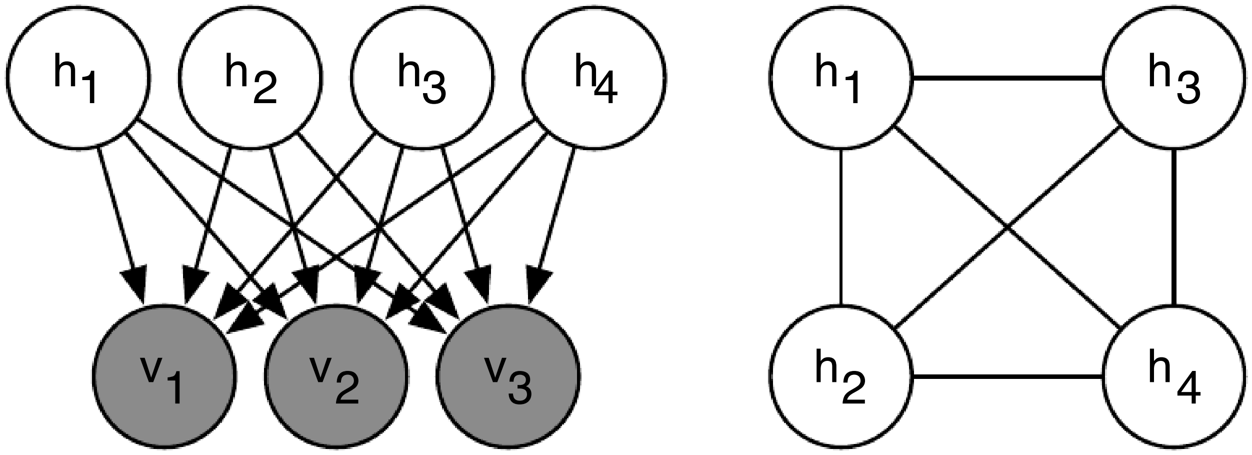

Figure 19.2: The graph structure of a binary sparse coding model with four hidden units. Left) The graph structure of $p(\boldsymbol{h}, \boldsymbol{v})$. Note that the edges are directed, and that every two hidden units co-parents of every visible unit. Right) The graph structure of $p(\boldsymbol{h} \mid \boldsymbol{v})$. In order to account for the active paths between co-parents, the posterior distribution needs an edge between all of the hidden units.

19.4.2 Calculus of Variations

Many machine learning techniques are based on minimizing a function $J(\boldsymbol{\theta})$ by finding the input vector $\boldsymbol{\theta} \in \mathbb{R}^n$ for which it takes on its minimal value. This can be accomplished with multivariate calculus and linear algebra, by solving for the critical points where $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = 0$. In some cases, we actually want to solve for a function $f(\boldsymbol{x})$, such as when we want to find the probability density function over some random variable. This is what *calculus of variations* enables us to do.

A function of a function f is known as a *functional* $J[f]$. Much as we can take partial derivatives of a function with respect to elements of its vector-valued argument, we can take *functional derivatives*, also known as *variational derivatives* of a functional $J[f]$ with respect to respect to individual values of the function $f(\boldsymbol{x})$. The functional derivative of the functional J with respect to the value of the function f at point $\boldsymbol{x}$ is denoted $\frac{\delta}{\delta f(x)} J$.

A complete formal development of functional derivatives is beyond the scope of this book. For our purposes, it is sufficient to state that for differentiable functions $f(\boldsymbol{x})$ and differentiable functions $g(y, \boldsymbol{x})$ with continuous derivatives, that

$$\frac{\delta}{\delta f(\boldsymbol{x})} \quad g\left(f(\boldsymbol{x}), \boldsymbol{x}\right) d\boldsymbol{x} = \frac{\partial}{\partial y} g(f(\boldsymbol{x}), \boldsymbol{x}). \tag{19.2}$$

To gain some intuition for this identity, one can think of $f(\boldsymbol{x})$ as being a vector with uncountably many elements, indexed by a real vector $\boldsymbol{x}$. In this (somewhat incomplete view), the identity providing the functional derivatives is the same as we would obtain for a vector $\boldsymbol{\theta} \in \mathbb{R}^n$ indexed by positive integers:

$$\frac{\partial}{\partial \theta_i} \sum_j g(\theta_j, j) = \frac{\partial}{\partial \theta_i} g(\theta_i, i).$$

Many results in other machine learning publications are presented using the more general *Euler-Lagrange equation* which allows g to depend on the derivatives of f as well as the value of f, but we do not need this fully general form for the results presented in this book.

To optimize a function with respect to a vector, we take the gradient of the function with respect to the vector and solve for the point where every element of the gradient is equal to zero. Likewise, we can optimize a functional by solving for the function where the functional derivative at every point is equal to zero.

As an example of how this process works, consider the problem of finding the probability distribution function over $x \in \mathbb{R}$ that has maximal Shannon entropy. Recall that the entropy of a probability distribution $p(x)$ is defined as

$$H[p] = -\mathbb{E}_x \log p(x).$$

For continuous values, the expectation is an integral:

$$H[p] = -\int p(x) \log p(x) dx.$$

We cannot simply maximize $H(x)$ with respect to the function $p(x)$, because the result might not be a probability distribution. Instead, we need to use Lagrange multipliers, to add a constraint that $p(x)$ integrate to 1. Also, the entropy increases without bound as the variance increases, so we can only search for the distribution with maximal entropy for fixed variance σ^2. Finally, the problem is underdetermined because the distribution can be shifted arbitrarily without changing the entropy. To impose a unique solution, we add a constraint that the mean of the distribution be μ. The Lagrangian functional for this optimization problem is

$$\mathcal{L}[p] = \lambda_1 \left(\int p(x) dx - 1 \right) + \lambda_2 \left(\mathbb{E}[x] - \mu \right) + \lambda_3 \left(\mathbb{E}[(x - \mu)^2] - \sigma^2 \right) + H[p]$$

$$= \int \left(\lambda_1 p(x) + \lambda_2 p(x) x + \lambda_3 p(x)(x - \mu)^2 - p(x) \log p(x) \right) dx - \lambda_1 - \mu \lambda_2 - \sigma^2 \lambda_3.$$

To minimize the Lagrangian with respect to p, we set the functional derivatives equal to 0:

$$\forall x, \frac{\delta}{\delta p(x)} \mathcal{L} = \lambda_1 + \lambda_2 x + \lambda_3 (x - \mu)^2 - 1 - \log p(x) = 0.$$

This condition now tells us the functional form of $p(x)$. By algebraically re-arranging the equation, we obtain

$$p(x) = \exp\left(-\lambda_1 - \lambda_2 x + \lambda_3 (x-\mu)^2 + 1\right).$$

We never assumed directly that $p(x)$ would take this functional form; we obtained the expression itself by analytically minimizing a functional. To finish the minimization problem, we must choose the λ values to ensure that all of our constraints are satisfied. We are free to choose any λ values, because the gradient of the Lagrangian with respect to the λ variables is zero so long as the constraints are satisfied. To satisfy all of the constraints, we may set $\lambda_1 = \log \sigma\sqrt{2\pi}$, $\lambda_2 = 0$, and $\lambda_3 = -\frac{1}{2\sigma^2}$ to obtain

$$p(x) = \mathcal{N}(x \mid \mu, \sigma^2).$$

This is one reason for using the normal distribution when we do not know the true distribution. Because the normal distribution has the maximum entropy, we impose the least possible amount of structure by making this assumption.

What about the probability distribution function that *minimizes* the entropy? It turns out that there is no specific function that achieves minimal entropy. As functions place more mass on $x = \mu \pm \sigma$ and less on all other values of x, they lose entropy. However, any function placing exactly zero mass on all but two points does not integrate to one, and is not a valid probability distribution. There thus is no single minimal entropy probability distribution function, much as there is no single minimal positive real number.

19.4.3 Continuous Latent Variables

TODO: Gaussian example from IG's thesis? TODO: S3C example

19.5 Stochastic Inference

TODO: Charlie Tang's SFNNs? Is there anything else where sampling-based inference actually gets used?

19.6 Learned Approximate Inference

TODO: wake-sleep algorithm

In chapter 18.2 we saw that one possible explanation for the role of dream sleep in human beings and animals is that dreams could provide the negative phase samples that Monte Carlo training algorithms use to approximate the negative gradient of the log partition function of undirected models. Another possible explanation for biological dreaming is that it is providing samples from $p(\boldsymbol{h}, \boldsymbol{v})$ which can be used to train an inference network to predict $\boldsymbol{h}$ given $\boldsymbol{v}$. In some senses, this explanation is more satisfying than the partition function explanation. Monte Carlo algorithms generally do not perform well if they are run using only the positive phase of the gradient for several steps then with only the negative phase of the gradient for several steps. Human beings and animals are usually awake for several consecutive hours then asleep for several consecutive hours, and it is not readily apparent how this schedule could support Monte Carlo training of an undirected model. Learning algorithms based on maximizing $\mathcal{L}$ can be run with prolonged periods of improving q and prolonged periods of improving $\boldsymbol{\theta}$, however. If the role of biological dreaming is to train networks for predicting q, then this explains how animals are able to remain awake for several hours (the longer they are awake, the greater the gap between $\mathcal{L}$ and $\log p(v)$, but $\mathcal{L}$ will remain a lower bound) and to remain asleep for several hours (the generative model itself is not modified during sleep) without damaging their internal models. Of course, these ideas are purely speculative, and there is no hard evidence to suggest that dreaming accomplishes either of these goals. Dreaming may also serve reinforcement learning rather than probabilistic modeling, by sampling synthetic experiences from the animal's transition model, on which to train the animal's policy. Or sleep may serve some other purpose not yet anticipated by the machine learning community.

TODO: DARN and NVIL? TODO: fast DBM inference