Approximate Inference

1. Sparce coding

1.1 Linear factor models

Many of the research frontiers in deep learning involve building a probabilistic model of the input, $p_{model}(\mathbf{x})$. Such a model can, in principle, use probabilistic inference to predict any of the variables in its environment given any of the other variables. Many of these models also have latent variables $\mathbf{h}$, with $p_{model}(\mathbf{x}) = \mathbb{E}_{\mathbf{h}} p_{model}(\mathbf{x}|\mathbf{h})$. These latent variables provide another means of representing the data.

Next, we describe the simplest probabilistic model with latent variables: linear factor model.

A **linear factor model** is defined by the use of a stochastic linear decoder function that generates $\mathbf{x}$ by adding noise to a linear transformation of $\mathbf{h}$.

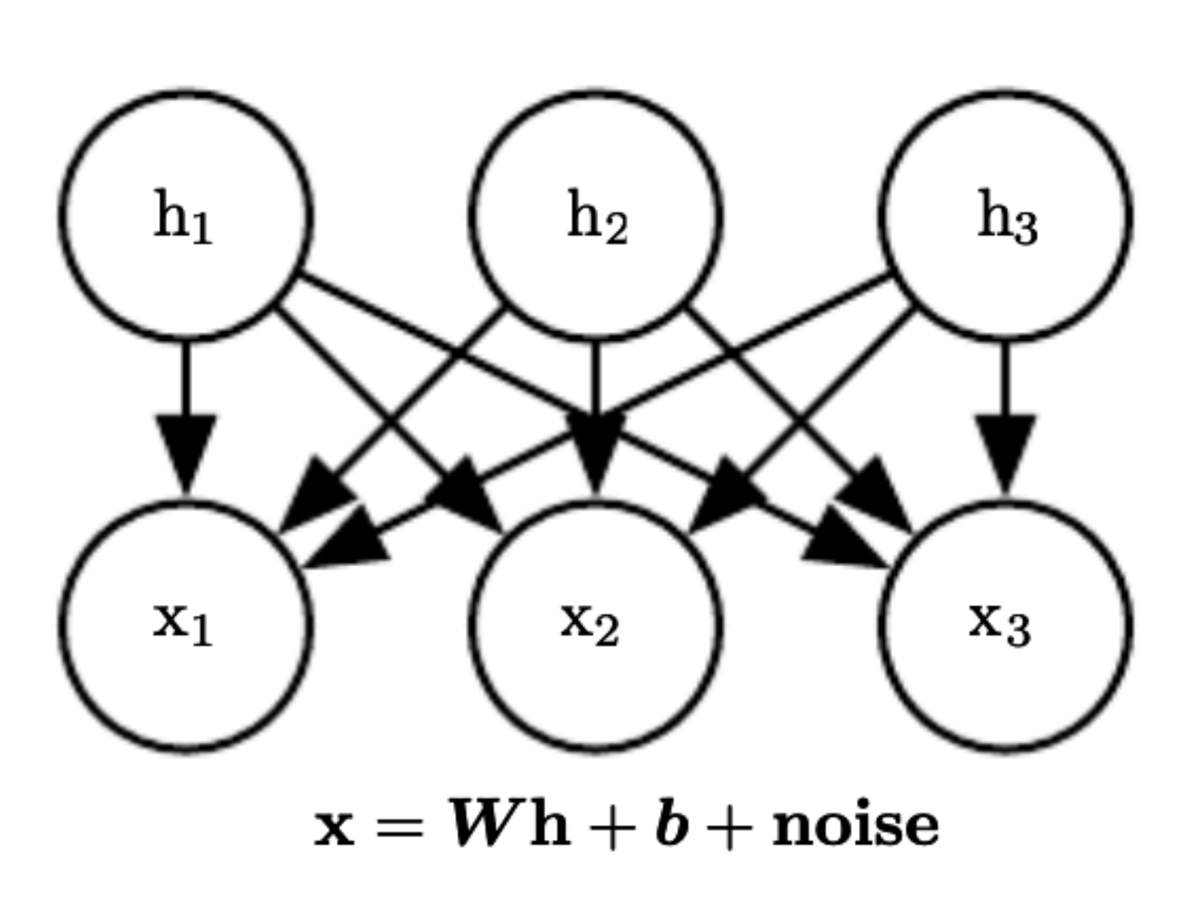
A linear factor model describes the data-generation process as follows. First, we sample the explanatory factors $\mathbf{h}$ from a distribution

$$\mathbf{h} \sim p(\mathbf{h}), \tag{1}$$

where $p(\mathbf{h})$ is a factorial distribution with $p(\mathbf{h}) = \prod_i p(h_i)$, so that it is easy to sample from. Next we sample the real-valued observable variables given the factors

$$\mathbf{x} = \mathbf{W}\mathbf{h} + \mathbf{b} + noise, \tag{2}$$

where the noise is typically Gaussian and diagonal (independent across dimensions). This is illustrated in figure below:



The directed graphical model describing the linear factor model family

1.2 Sparse coding

Sparse coding (Olshausen and Field, 1996) is a linear factor model that has been heavily studied as an unsupervised feature learning and feature extraction mechanism.

Sparse coding models typically assume that the linear factors have Gaussian noise with isotropic precision β:

$$p(\mathbf{x}|\mathbf{h}) = \mathcal{N}(\mathbf{x}; \mathbf{W}\mathbf{h} + \mathbf{b}, \frac{1}{\beta}\mathbf{I}). \tag{3}$$

The distribution $p(\mathbf{h})$ is chosen to be one with sharp peaks near 0 (Olshausen and Field, 1996). Common choices include factorized Laplace, Cauchy or factorized Student t-distributions. For example, the Laplace prior parametrized in terms of the sparsity penalty coefficient λ is given by

$$p(h_i) = Laplace(h_i; 0, \frac{2}{\lambda}) = \frac{\lambda}{4} e^{-\frac{1}{2}\lambda|h_i|}, \tag{4}$$

and the student t prior by

$$p(h_i) \propto \frac{1}{(1 + \frac{h_i^2}{\nu})^{\frac{\nu+1}{2}}}. \tag{5}$$

Training sparse coding with maximum likelihood is intractable. Instead, the training alternates between encoding the data and training the decoder to better reconstruct the data given the encoding. This approach will be justified further as a principled approximation to maximum likelihood later.

For models such as PCA, we use a parametric encoder function that predicts $\mathbf{h}$ and consists only of multiplication by a weight matrix. The encoder that we use with sparse coding is not a parametric encoder. Instead, the encoder is an optimization algorithm, which solves an optimization problem in which we seek the single most likely code value:

$$\mathbf{h}^* = f(\mathbf{x}) = \arg\max_{\mathbf{h}} p(\mathbf{h}|\mathbf{x}). \tag{6}$$

When combined with equation (3) and (4), this yields the following optimization problem

$$\arg\max_{\mathbf{h}} p(\mathbf{h}|\mathbf{x}) = \arg\max_{\mathbf{h}} \log p(\mathbf{h}|\mathbf{x}) = \arg\min_{\mathbf{h}} \lambda\|h\|_1 + \beta\|\mathbf{x} - \mathbf{W}\mathbf{h}\|_2^2. \tag{7}$$

Due to the imposition of an L^1 norm on $\mathbf{h}$, this procedure will yield a sparse $\mathbf{h}^*$.

To train the model rather than just perform inference, we alternate between minimization with respect to $\mathbf{h}$ and minimization with respect to $\mathbf{W}$. In this presentation, we treat β as a hyperparameter. Typically it is set to 1 because its role in this optimization problem is shared with λ, and there is no need for both hyperparameters. In principle, we could also treat β as a parameter of the model and learn it.

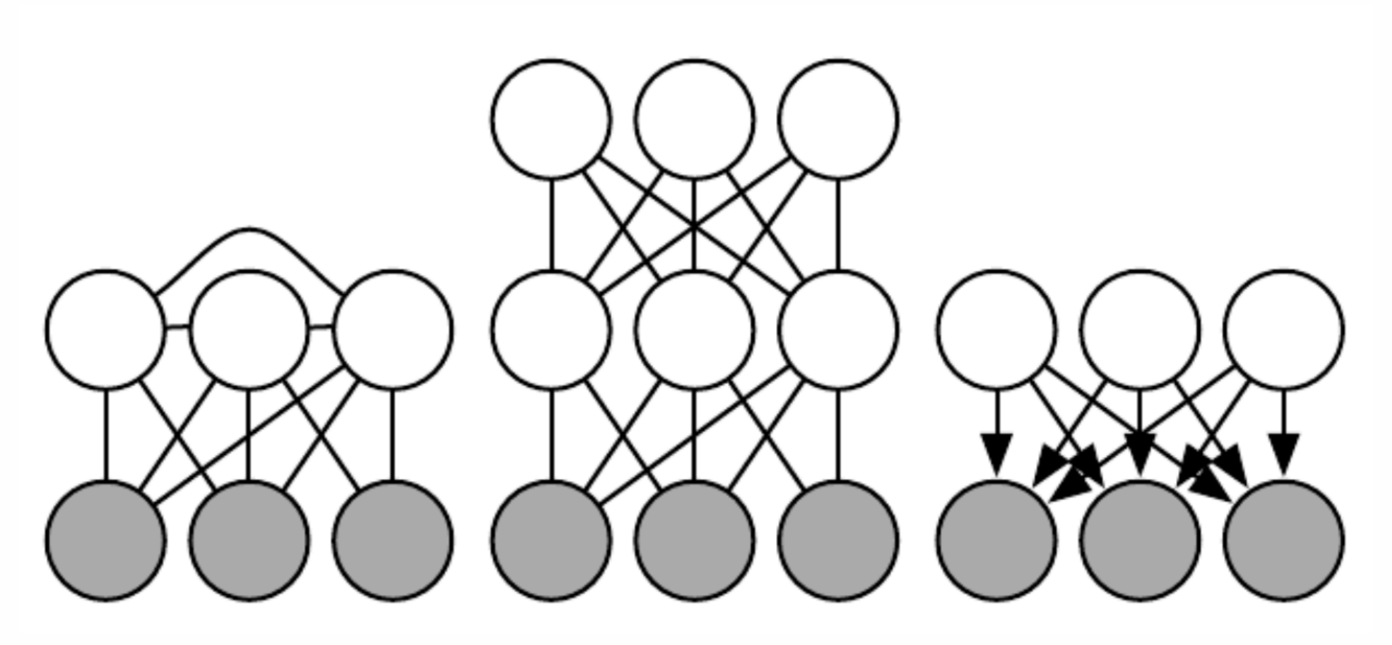
2. Approximate inference

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 $\mathbf{v}$ and a set of latent variables $\mathbf{h}$. The challenge of inference usually refers to the difficult problem of computing $p(\mathbf{h}|\mathbf{v})$ or taking expectations with respect to it. Such operations are often necessary for tasks like maximum likelihood learning.

Most graphical models with multiple layers of hidden variables 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.

Next, we will introduce several of the techniques for confronting these intractable inference problems.

Intractable inference problems in deep learning usually arise from interactions between latent variables in a structured graphical model. See figure 19.1 for some examples.



Intractable inference problems in deep learning are usually the result of interactions between latent variables in a structured graphical model. These interactions can be due to edges directly connecting one latent variable to another or longer paths that are activated when the child of a V-structure is observed

2.1 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. Approximate inference algorithms may then be derived by approximating the underlying optimization problem.

To construct the optimization problem, assume we have a probabilistic model consisting of observed variables $\mathbf{v}$ and latent variables $\mathbf{h}$. We would like to compute the log-probability of the observed data, $\log p(\mathbf{v};\boldsymbol{\theta})$. Sometimes it is too difficult to compute $\log p(\mathbf{v};\boldsymbol{\theta})$ if it is costly to marginalize out $\mathbf{h}$. Instead, we can compute a lower bound $\mathcal{L}(\mathbf{v},\boldsymbol{\theta},q)$ on $\log p(\mathbf{v};\boldsymbol{\theta})$. This bound is called the **evidence lower bound (ELBO)**. Another commonly used name for this lower bound is the **negative variational free energy**. Specifically, the evidence lower bound is defined to be

$$\mathcal{L}(\mathbf{v},\boldsymbol{\theta},q) = \log p(\mathbf{v};\boldsymbol{\theta}) - D_{\mathrm{KL}}(q(\mathbf{h}|\mathbf{v})\|p(\mathbf{h}|\mathbf{v};\boldsymbol{\theta})), \tag{8}$$

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

Because the difference between $\log p(\mathbf{v})$ and $\mathcal{L}(\mathbf{v},\boldsymbol{\theta},q)$ is given by the KL divergence, and because the KL divergence is always nonnegative, we can see that $\mathcal{L}$ always has at most the same value as the desired log-probability. The two are equal if and only if q is the same distribution as $p(\mathbf{h}|\mathbf{v})$.

$\mathcal{L}$ 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:

$$\begin{aligned}
\mathcal{L}(\boldsymbol{v}, \boldsymbol{\theta}, q) &= \log p(\boldsymbol{v}; \boldsymbol{\theta}) - D_{\mathrm{KL}}(q(\boldsymbol{h} \mid \boldsymbol{v}) \| p(\boldsymbol{h} \mid \boldsymbol{v}; \boldsymbol{\theta})) \\
&= \log p(\boldsymbol{v}; \boldsymbol{\theta}) - \mathbb{E}_{\mathbf{h} \sim q} \log \frac{q(\boldsymbol{h} \mid \boldsymbol{v})}{p(\boldsymbol{h} \mid \boldsymbol{v})} \\
&= \log p(\boldsymbol{v}; \boldsymbol{\theta}) - \mathbb{E}_{\mathbf{h} \sim q} \log \frac{q(\boldsymbol{h} \mid \boldsymbol{v})}{\frac{p(\boldsymbol{h}, \boldsymbol{v}; \boldsymbol{\theta})}{p(\boldsymbol{v}; \boldsymbol{\theta})}} \\
&= \log p(\boldsymbol{v}; \boldsymbol{\theta}) - \mathbb{E}_{\mathbf{h} \sim q} \left[\log q(\boldsymbol{h} \mid \boldsymbol{v}) - \log p(\boldsymbol{h}, \boldsymbol{v}; \boldsymbol{\theta}) + \log p(\boldsymbol{v}; \boldsymbol{\theta})\right] \\
&= \; - \mathbb{E}_{\mathbf{h} \sim q} \left[\log q(\boldsymbol{h} \mid \boldsymbol{v}) - \log p(\boldsymbol{h}, \boldsymbol{v}; \boldsymbol{\theta})\right].
\end{aligned}$$

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

$$\mathcal{L}(\mathbf{v}, \boldsymbol{\theta}, q) = \mathbb{E}_{\mathbf{h} \sim q}[\log p(\mathbf{h}, \mathbf{v})] + H(q). \tag{9}$$

We can thus think of inference as the procedure for finding the q that maximizes $\mathcal{L}$. Exact inference maximizes $\mathcal{L}$ perfectly by searching over a family of functions q that includes $q(\mathbf{h}|\mathbf{v})$.

Throughout this chapter, we show how to derive different forms of approximate inference by using approximate optimization to find q. We can make the optimization procedure less expensive but approximate by restricting the family of distributions q that the optimization is allowed to search over or by using an imperfect optimization procedure that may not completely maximize $\mathcal{L}$ but may merely increase it by a significant amount.

No matter what choice of q we use, $\mathcal{L}$ is a lower bound. 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.

2.2 MAP inference and sparse coding

We usually use the term inference to refer to computing the probability distribution over one set of variables given another. When training probabilistic models with latent variables, we are usually interested in computing $p(\mathbf{h}|\mathbf{v})$. An alternative form of inference is to compute the single most likely value of the missing variables, rather than to infer the entire distribution over their possible values. In the context of latent variable models, this means computing

$$\mathbf{h}^* = \arg\max_{\mathbf{h}} p(\mathbf{h}|\mathbf{v}). \tag{10}$$

This is known as **maximum a posteriori** inference, abbreviated as **MAP** inference.

MAP inference is usually not thought of as approximate inference—it does compute the exact most likely value of $\mathbf{h}^*$. However, if we wish to develop a learning process based on maximizing $\mathcal{L}(\mathbf{v}, \boldsymbol{\theta}, q)$, then it is helpful to think of MAP inference as a procedure that provides a value of q. In this sense, we can think of MAP inference as approximate inference, because it does not provide the optimal q.

Recall from section 2.1 that exact inference consists of maximizing $\mathcal{L}(\mathbf{v}, \boldsymbol{\theta}, q) = \mathbb{E}_{\mathbf{h}\sim q}[\log p(\mathbf{h}, \mathbf{v})] + H(q)$ with respect to q over an unrestricted family of probability distributions, using an exact optimization algorithm. We can derive MAP inference as a form of approximate inference by restricting the family of distributions q may be drawn from. Specifically, we require q to take on a Dirac distribution

$$q(\mathbf{h}|\mathbf{v}) = \delta(\mathbf{h} - \boldsymbol{\mu}). \tag{11}$$

This means that we can now control q entirely via $\boldsymbol{\mu}$. Dropping terms of $\mathcal{L}$ that do not vary with $\boldsymbol{\mu}$, we are left with the optimization problem

$$\boldsymbol{\mu}^* = \arg\max_{\boldsymbol{\mu}} \log p(\mathbf{h} = \boldsymbol{\mu}, \mathbf{v}), \tag{12}$$

which is equivalent to the MAP inference problem.

MAP inference is commonly used in deep learning as both a feature extractor and a learning mechanism. It is primarily used for sparse coding models.

Recall from section 1.2 that sparse coding is a linear factor model that imposes a sparsity-inducing prior on its hidden units. A common choice is a factorial Laplace prior, with

$$P(h_i) = \frac{\lambda}{2} e^{-\lambda|h_i|}. \tag{13}$$

The visible units are then generated by performing a linear transformation and adding noise

$$p(\mathbf{v}|\mathbf{h}) = \mathcal{N}(\mathbf{v}; \mathbf{W}\mathbf{h} + \mathbf{b}, \frac{1}{\beta}\mathbf{I}). \tag{14}$$

Computing or even representing $p(\mathbf{h}|\mathbf{v})$ is difficult. Every pair of variables h_i and h_j are both parents of $\mathbf{v}$. This means that when $\mathbf{v}$ is observed, the graphical model contains an active path connecting h_i and h_j. All the hidden units thus participate in one massive clique in $p(\mathbf{h}|\mathbf{v})$. If the model were Gaussian, then these interactions could be modeled efficiently via the covariance matrix, but the sparse prior makes these interactions non-Gaussian.

Because $p(\mathbf{h}|\mathbf{v})$ is intractable, so is the computation of the log-likelihood and its gradient. We thus cannot use exact maximum likelihood learning. Instead, <u>we use MAP inference and learn the parameters by maximizing the ELBO</u> defined by the Dirac distribution around the MAP estimate of $\mathbf{h}$.

If we concatenate all the $\mathbf{h}$ vectors in the training set into a matrix $\mathbf{H}$, and concatenate all the $\mathbf{v}$ vectors into a matrix $\mathbf{V}$, then the sparse coding learning process consists of minimizing

$$J(\mathbf{H}, \mathbf{W}) = \|H_{i,j}\|_1 + \|\mathbf{V} - \mathbf{H}\mathbf{W}^T\|_2^2. \tag{15}$$

We can minimize J by alternating between minimization with respect to $\mathbf{H}$ and minimization with respect to $\mathbf{W}$. Both subproblems are convex. In fact, the minimization with respect to $\mathbf{W}$ is just a linear regression problem. And minimization with respect to $\mathbf{H}$ requires specialized algorithms such as the feature-sign search algorithm (Lee et al., 2007).

However, minimization of J with respect to both arguments is usually not a convex problem.

2.3 Variational inference and learning

We have seen how the evidence lower bound $\mathcal{L}(\mathbf{v}, \boldsymbol{\theta}, q)$ is a lower bound on $\log p(\mathbf{v}; \boldsymbol{\theta})$, how inference can be viewed as maximizing $\mathcal{L}$ with respect to q, and how learning can be viewed as maximizing $\mathcal{L}$ with respect to $\boldsymbol{\theta}$. We have seen that learning algorithms based on MAP inference enable us to learn using a point estimate of $p(\mathbf{h}|\mathbf{v})$ rather than inferring the entire distribution. Now we develop the more general approach to variational learning.

The core idea behind variational learning is that we can maximize $\mathcal{L}$ over a restricted family of distributions q. This family should be chosen so that it is easy to compute $\mathbb{E}_q \log p(\mathbf{h}, \mathbf{v})$. A typical way to do this is to introduce assumptions about how q factorizes.

A common approach to variational learning is to impose the restriction that q is a factorial distribution

$$q(\mathbf{h}|\mathbf{v}) = \prod_i q(h_i|\mathbf{v}). \tag{16}$$

This is called the **mean field** approach. More generally, we can impose any graphical model structure we choose on q, to flexibly determine how many interactions we want our approximation to capture. This fully general graphical model approach is called **structured variational inference** (Saul and Jordan, 1996).

The beauty of the variational approach is that we do not need to specify a specific parametric form for q. We specify how it should factorize, but then the optimization problem determines the optimal probability distribution within those factorization constraints.

2.3.1 Discrete latent variables

Variational inference with discrete latent variables is relatively straightforward. We define a distribution q, typically one where each factor of q is just defined by a lookup table over discrete states. In the simplest case, $\mathbf{h}$ is binary and we make the mean field assumption that q factorizes over each individual h_i. In this case we can parametrize q with a vector $\hat{\mathbf{h}}$ whose entries are probabilities. Then

$$q(h_i = 1|\mathbf{v}) = \hat{h}_i. \tag{17}$$

After determining how to represent q, we simply optimize its parameters. With discrete latent variables, this is just a standard optimization problem. In principle the selection of q could be done with any optimization algorithm, such as gradient descent.

However, because this optimization must occur in the inner loop of a learning algorithm, it must be very fast. To achieve this speed, we typically use special optimization algorithms that are designed to solve comparatively small and simple problems in few iterations. A popular choice is to iterate fixed-point equations, in other words, to solve

$$\frac{\partial}{\partial \hat{h}_i}\mathcal{L} = 0 \tag{18}$$

for $\hat{h}_i$. We repeatedly update different elements of $\hat{\mathbf{h}}$ until we satisfy a convergence criterion.

To make this more concrete, we show how to apply variational inference to the **binary sparse coding model**.

In the binary sparse coding model, the input $\mathbf{v} \in \mathbb{R}^n$ is generated from the model 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 $\mathbf{h} \in \{0, 1\}^m$:

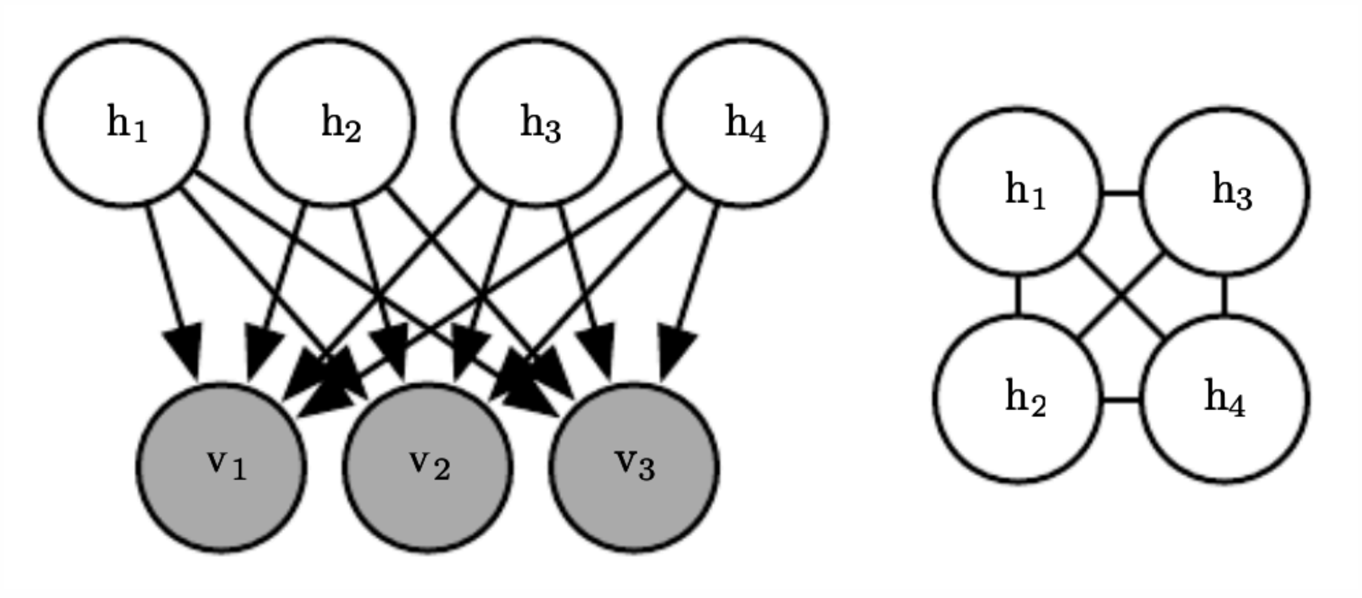
$$p(h_i = 1) = \sigma(b_i),$$
$$p(\mathbf{v}|\mathbf{h}) = \mathcal{N}(\mathbf{v}; \mathbf{W}\mathbf{h}, \boldsymbol{\beta}^{-1}), \tag{19}$$

where $\mathbf{b}$ is a learnable set of biases, $\mathbf{W}$ is a learnable weight matrix, and $\boldsymbol{\beta}$ is a learnable, 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:

$$\begin{aligned}
\frac{\partial}{\partial b_i} \log p(\mathbf{v}) &= \frac{\frac{\partial}{\partial b_i} p(\mathbf{v})}{p(\mathbf{v})} \\
&= \frac{\frac{\partial}{\partial b_i} \sum_{\mathbf{h}} p(\mathbf{h}, \mathbf{v})}{p(\mathbf{v})} \\
&= \frac{\frac{\partial}{\partial b_i} \sum_{\mathbf{h}} p(\mathbf{h}) p(\mathbf{v}|\mathbf{h})}{p(\mathbf{v})} \\
&= \frac{\sum_{\mathbf{h}} p(\mathbf{v}|\mathbf{h}) \frac{\partial}{\partial b_i} p(\mathbf{v})}{p(\mathbf{v})} \\
&= \sum_{\mathbf{h}} p(\mathbf{v}|\mathbf{h}) \frac{\frac{\partial}{\partial b_i} p(\mathbf{v})}{p(\mathbf{v})} \\
&= \mathbb{E}_{\mathbf{h}\sim p(\mathbf{h}|\mathbf{v})} \frac{\partial}{\partial b_i} \log p(\mathbf{h}).
\end{aligned} \tag{20}$$

This requires computing expectations with respect to $p(\mathbf{h}|\mathbf{v})$. Unfortunately, $p(\mathbf{h}|\mathbf{v})$ is a complicated distribution. See figure below for the graph structure of $p(\mathbf{h}, \mathbf{v})$ and $p(\mathbf{h}|\mathbf{v})$. 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.



The graph structure of a binary sparse coding model with four hidden units

We can resolve this difficulty by using variational inference and variational learning instead.

We can make a mean field approximation:

$$q(\mathbf{h}|\mathbf{v}) = \prod_i q(h_i|\mathbf{v}). \tag{21}$$

The latent variables of the binary sparse coding model are binary, so to represent a factorial q we simply need to model m Bernoulli distributions $q(h_i|\mathbf{v})$. A natural way to represent the means of the Bernoulli distributions is with a vector $\hat{\mathbf{h}}$ of probabilities, with $q(h_i = 1|\mathbf{v}) = \hat{h}_i$.

We impose a restriction that is never equal to 0 or to 1, in order to avoid errors when computing, for example, $\log \hat{h}_i$. In a software implementation, however, machine rounding error could result in 0 or 1 values. In software, we may wish to implement binary sparse coding using an unrestricted vector of variational parameters $\mathbf{z}$ and obtain $\hat{\mathbf{h}}$ via the relation $\hat{\mathbf{h}} = \sigma(\mathbf{z})$.

Now the evidence lower bound is given by

$$
\begin{aligned}
&\mathcal{L}(\boldsymbol{v}, \boldsymbol{\theta}, q) && (19.29)\\
&= \mathbb{E}_{\mathbf{h}\sim q}[\log p(\boldsymbol{h}, \boldsymbol{v})] + H(q) && (19.30)\\
&= \mathbb{E}_{\mathbf{h}\sim q}[\log p(\boldsymbol{h}) + \log p(\boldsymbol{v} \mid \boldsymbol{h}) - \log q(\boldsymbol{h} \mid \boldsymbol{v})] && (19.31)\\
&= \mathbb{E}_{\mathbf{h}\sim q}\left[\sum_{i=1}^{m} \log p(h_i) + \sum_{i=1}^{n} \log p(v_i \mid \boldsymbol{h}) - \sum_{i=1}^{m} \log q(h_i \mid \boldsymbol{v})\right] && (19.32)\\
&= \sum_{i=1}^{m} \left[\hat{h}_i(\log \sigma(b_i) - \log \hat{h}_i) + (1 - \hat{h}_i)(\log \sigma(-b_i) - \log(1 - \hat{h}_i))\right] && (19.33)\\
&\quad + \mathbb{E}_{\mathbf{h}\sim q}\left[\sum_{i=1}^{n} \log \sqrt{\frac{\beta_i}{2\pi}} \exp\left(-\frac{\beta_i}{2}(v_i - \boldsymbol{W}_{i,:}\boldsymbol{h})^2\right)\right] && (19.34)\\
&= \sum_{i=1}^{m} \left[\hat{h}_i(\log \sigma(b_i) - \log \hat{h}_i) + (1 - \hat{h}_i)(\log \sigma(-b_i) - \log(1 - \hat{h}_i))\right] && (19.35)\\
&\quad + \frac{1}{2} \sum_{i=1}^{n} \left[\log \frac{\beta_i}{2\pi} - \beta_i \left(v_i^2 - 2v_i \boldsymbol{W}_{i,:} \hat{\boldsymbol{h}} + \sum_j \left[W_{i,j}^2 \hat{h}_j + \sum_{k \neq j} W_{i,j} W_{i,k} \hat{h}_j \hat{h}_k\right]\right)\right]. && (19.36)
\end{aligned}
$$

While these equations are somewhat unappealing aesthetically, they show that $\mathcal{L}$ can be expressed in a small number of simple arithmetic operations. The evidence lower bound $\mathcal{L}$ is therefore tractable. We can use $\mathcal{L}$ as a replacement for the intractable log-likelihood.

In principle, we could simply run gradient ascent on both $\mathbf{v}$ and $\mathbf{h}$, and this would make a perfectly acceptable combined inference and training algorithm. Usually, however, we do not do this, for two reasons.

First, this would require storing $\hat{\mathbf{h}}$ for each $\mathbf{v}$. We typically prefer algorithms that do not require per example memory. It is difficult to scale learning algorithms to billions of examples if we must remember a dynamically updated vector associated with each example.

Second, we would like to be able to extract the features $\hat{\mathbf{h}}$ very quickly, in order to recognize the content of $\mathbf{v}$.

For both these reasons, we typically do not use gradient descent to compute the mean field parameters $\hat{\mathbf{h}}$. Instead, we rapidly estimate them with fixed-point equations

$$\frac{\partial}{\partial \hat{h}_i}\mathcal{L}(\mathbf{v}, \boldsymbol{\theta}, \hat{\mathbf{h}}) = 0. \tag{22}$$

We can then iteratively apply the solution to the equation for $i = 1, \cdots, m$, and repeat the cycle until we satisfy a convergence criterion.

To make this more concrete, we show how to derive the updates for the binary sparse coding model in particular.

$$\frac{\partial}{\partial \hat{h}_i}\mathcal{L}(\boldsymbol{v}, \boldsymbol{\theta}, \hat{\boldsymbol{h}}) \tag{19.38}$$

$$= \frac{\partial}{\partial \hat{h}_i}\left[\sum_{j=1}^{m}\left[\hat{h}_j\left(\log\sigma(b_j) - \log\hat{h}_j\right) + (1-\hat{h}_j)(\log\sigma(-b_j) - \log(1-\hat{h}_j))\right]\right. \tag{19.39}$$

$$\left. + \frac{1}{2}\sum_{j=1}^{n}\left[\log\frac{\beta_j}{2\pi} - \beta_j\left(v_j^2 - 2v_j\boldsymbol{W}_{j,:}\hat{\boldsymbol{h}} + \sum_k\left[W_{j,k}^2\hat{h}_k + \sum_{l\neq k}W_{j,k}W_{j,l}\hat{h}_k\hat{h}_l\right]\right)\right]\right] \tag{19.40}$$

$$= \log\sigma(b_i) - \log\hat{h}_i - 1 + \log(1-\hat{h}_i) + 1 - \log\sigma(-b_i) \tag{19.41}$$

$$+ \sum_{j=1}^{n}\left[\beta_j\left(v_jW_{j,i} - \frac{1}{2}W_{j,i}^2 - \sum_{k\neq i}\boldsymbol{W}_{j,k}\boldsymbol{W}_{j,i}\hat{h}_k\right)\right] \tag{19.42}$$

$$= b_i - \log\hat{h}_i + \log(1-\hat{h}_i) + \boldsymbol{v}^\top\boldsymbol{\beta}\boldsymbol{W}_{:,i} - \frac{1}{2}\boldsymbol{W}_{:,i}^\top\boldsymbol{\beta}\boldsymbol{W}_{:,i} - \sum_{j\neq i}\boldsymbol{W}_{:,j}^\top\boldsymbol{\beta}\boldsymbol{W}_{:,i}\hat{h}_j. \tag{19.43}$$

To apply the fixed-point update inference rule, we solve for the $\hat{h}_i$ that sets equation (22) to 0:

$$\hat{h}_i = \sigma\left(b_i + \mathbf{v}^T\boldsymbol{\beta}\mathbf{W}_{:,i} - \frac{1}{2}\mathbf{W}_{:,i}^T\boldsymbol{\beta}\mathbf{W}_{:,i} - \sum_{j\neq i}\mathbf{W}_{:,j}^T\boldsymbol{\beta}\mathbf{W}_{:,i}\hat{h}_j\right). \tag{23}$$

In this example, we have derived an update rule that updates a single unit at a time. It would be advantageous to be able to update more units simultaneously. Some graphical models, such as deep Boltzmann machines, are structured in such a way that we can solve for many entries of $\hat{\mathbf{h}}$ simultaneously. Unfortunately, binary sparse coding does not admit such block updates. Instead, we can use a heuristic technique called **damping** to perform block updates. In the damping approach, we solve for the individually optimal values of every element of $\hat{\mathbf{h}}$, then move all the values in a small step in that direction. This approach is no longer guaranteed to increase $\mathcal{L}$ at each step, but it works well in practice for many models.

2.3.2 Continuous latent variables

When our graphical model contains continuous latent variables, we can still perform variational inference and learning by maximizing $\mathcal{L}$. However, we must now use calculus of variations when maximizing $\mathcal{L}$ with respect to $q(\mathbf{h}|\mathbf{v})$.

In most cases, practitioners need not solve any calculus of variations problems themselves. Instead, there is a general equation for the mean field fixed-point updates. If we make the mean field approximation

$$q(\mathbf{h}|\mathbf{v}) = \prod_i q(h_i|\mathbf{v}), \tag{24}$$

and fix $q(h_j|\mathbf{v})$ for all $j \neq i$, then the optimal $q(h_i|\mathbf{v})$ can be obtained by normalizing the unnormalized distribution:

$$\tilde{q}(h_i|\mathbf{v}) = \exp\left(\mathbb{E}_{\mathbf{h}_{-i}\sim q(\mathbf{h}_{-i}|\mathbf{v})} \log \tilde{p}(\mathbf{v}, \mathbf{h})\right), \tag{25}$$

as long as p does not assign 0 probability to any joint configuration of variables.

Equation (25) is a fixed-point equation, designed to be iteratively applied for each value of i repeatedly until convergence. However, it also tells us more than that. It tells us the functional form that the optimal solution will take, whether we arrive there by fixed-point equations or not. This means we can take the functional form from that equation but regard some of the values that appear in it as parameters, which we can optimize with any optimization algorithm we like.

As an example, consider a simple probabilistic model, with latent variables $\mathbf{h} \in \mathbb{R}^2$ and just one visible variable, v. Suppose that $p(\mathbf{h}) = \mathcal{N}(\mathbf{h}; 0, \mathbf{I})$ and $p(v|\mathbf{h}) = \mathcal{N}(v; \mathbf{w}^T\mathbf{h}, 1)$. We could actually simplify this model by integrating out $\mathbf{h}$; the result is just a Gaussian distribution over v.

The true posterior is given, up to a normalizing constant, by

$$\begin{aligned} p(\mathbf{h}|\mathbf{v}) &\propto p(\mathbf{h}, \mathbf{v}) = p(h_1)p(h_2)p(v|\mathbf{h}) \\ &\propto \exp\left(-\frac{1}{2}[h_1^2 + h_2^2 + (v - h_1w_1 - h_2w_2)^2]\right) \\ &= \exp\left(-\frac{1}{2}[h_1^2 + h_2^2 + v^2 + h_1^2w_1^2 + h_2^2w_2^2 - 2vh_1w_1 - 2vh_2w_2 + 2h_1w_1h_2w_2]\right). \end{aligned} \tag{26}$$

Because of the presence of the terms multiplying h_1 and h_2 together, we can see that the true posterior does not factorize over h_1 and h_2.

Applying equation (25), we find that

$$\begin{aligned} &\tilde{q}(h_1 \mid \boldsymbol{v}) \\ =&\exp\left(\mathbb{E}_{\mathrm{h}_2\sim q(\mathrm{h}_2|\boldsymbol{v})} \log \tilde{p}(\boldsymbol{v}, \boldsymbol{h})\right) \\ =&\exp\Big(-\frac{1}{2}\mathbb{E}_{\mathrm{h}_2\sim q(\mathrm{h}_2|\boldsymbol{v})} \left[h_1^2 + h_2^2 + v^2 + h_1^2w_1^2 + h_2^2w_2^2 \right. \\ &\quad \left. -2vh_1w_1 - 2vh_2w_2 + 2h_1w_1h_2w_2\right]\Big). \end{aligned}$$

From this, we can see that there are effectively only two values we need to obtain from $q(h_2|\mathbf{v})$: $\mathbb{E}_{h_2 \sim q(h|\mathbf{v})}[h_2]$ and $\mathbb{E}_{h_2 \sim q(h|\mathbf{v})}[h_2^2]$. Writing these as $\langle h_2 \rangle$ and $\langle h_2^2 \rangle$, we obtain

$$\tilde{q}(h_1 \mid \boldsymbol{v}) = \exp\left(-\frac{1}{2}\left[h_1^2 + \langle h_2^2\rangle + v^2 + h_1^2 w_1^2 + \langle h_2^2\rangle w_2^2 \right.\right.$$
$$\left.\left. -2vh_1 w_1 - 2v\langle h_2\rangle w_2 + 2h_1 w_1 \langle h_2\rangle w_2\right]\right).$$

From this, we can see that $\tilde{q}$ has the functional form of a Gaussian. We can thus conclude that $q(\mathbf{h}|\mathbf{v}) = \mathcal{N}(\mathbf{h}; \boldsymbol{\mu}, \boldsymbol{\beta}^{-1})$ where $\boldsymbol{\mu}$ and diagonal $\boldsymbol{\beta}$ are variational parameters, which we can optimize using any technique we choose.

It is important to recall that we did not ever assume that q would be Gaussian; its Gaussian form was derived automatically by using calculus of variations to maximize q with respect to $\mathcal{L}$. Using the same approach on a different model could yield a different functional form of q.