5

Energy-Based Models and Boltzmann Machines

Because Deep Belief Networks (DBNs) are based on Restricted Boltzmann Machines (RBMs), which are particular *energy-based models*, we introduce here the main mathematical concepts helpful to understand them, including *Contrastive Divergence* (CD).

5.1 Energy-Based Models and Products of Experts

Energy-based models associate a scalar energy to each configuration of the variables of interest [107, 106, 149]. Learning corresponds to modifying that energy function so that its shape has desirable properties. For example, we would like plausible or desirable configurations to have low energy. Energy-based probabilistic models may define a probability distribution through an energy function, as follows:

$$P(\mathbf{x}) = \frac{e^{-\text{Energy}(\mathbf{x})}}{Z}, \tag{5.1}$$

i.e., energies operate in the log-probability domain. Th above generalizes *exponential family* models [29], for which the energy function $\text{Energy}(\mathbf{x})$ has the form $\eta(\theta) \cdot \phi(x)$. We will see below that the conditional distribution of one layer given another, in the RBM, can be taken

from any of the exponential family distributions [200]. Whereas any probability distribution can be cast as an energy-based models, many more specialized distribution families, such as the exponential family, can benefit from particular inference and learning procedures. Some instead have explored rather general-purpose approaches to learning in energy-based models [84, 106, 149].

The normalizing factor Z is called the *partition function* by analogy with physical systems,

$$Z = \sum_{\mathbf{x}} e^{-\text{Energy}(\mathbf{x})} \tag{5.2}$$

with a sum running over the input space, or an appropriate integral when $\mathbf{x}$ is continuous. Some energy-based models can be defined even when the sum or integral for Z does not exist (see Section 5.1.2).

In the *product of experts* formulation [69, 70], the energy function is a sum of terms, each one associated with an "expert" f_i:

$$\text{Energy}(\mathbf{x}) = \sum_i f_i(\mathbf{x}), \tag{5.3}$$

i.e.,

$$P(\mathbf{x}) \propto \prod_i P_i(\mathbf{x}) \propto \prod_i e^{-f_i(\mathbf{x})}. \tag{5.4}$$

Each expert $P_i(\mathbf{x})$ can thus be seen as a detector of implausible configurations of $\mathbf{x}$, or equivalently, as enforcing constraints on $\mathbf{x}$. This is clearer if we consider the special case where $f_i(\mathbf{x})$ can only take two values, one (small) corresponding to the case where the constraint is satisfied, and one (large) corresponding to the case where it is not. [69] explains the advantages of a product of experts by opposition to a *mixture of experts* where the product of probabilities is replaced by a weighted sum of probabilities. To simplify, assume that each expert corresponds to a constraint that can either be satisfied or not. In a mixture model, the constraint associated with an expert is an indication of belonging to a region which excludes the other regions. One advantage of the product of experts formulation is therefore that the set of $f_i(\mathbf{x})$ forms a distributed representation: instead of trying to partition the space with one region per expert as in mixture models, they

partition the space according to all the possible configurations (where each expert can have its constraint violated or not). [69] proposed an algorithm for estimating the gradient of $\log P(\mathbf{x})$ in Equation (5.4) with respect to parameters associated with each expert, using the first instantiation [70] of the Contrastive Divergence algorithm (Section 5.4).

5.1.1 Introducing Hidden Variables

In many cases of interest, $\mathbf{x}$ has many component variables $\mathbf{x}_i$, and we do not observe of these components simultaneously, or we want to introduce some non-observed variables to increase the expressive power of the model. So we consider an observed part (still denoted $\mathbf{x}$ here) and a *hidden* part $\mathbf{h}$

$$P(\mathbf{x}, \mathbf{h}) = \frac{e^{-\text{Energy}(\mathbf{x},\mathbf{h})}}{Z} \tag{5.5}$$

and because only $\mathbf{x}$ is observed, we care about the marginal

$$P(\mathbf{x}) = \sum_{\mathbf{h}} \frac{e^{-\text{Energy}(\mathbf{x},\mathbf{h})}}{Z}. \tag{5.6}$$

In such cases, to map this formulation to one similar to Equation (5.1), we introduce the notation (inspired from physics) of *free energy*, defined as follows:

$$P(\mathbf{x}) = \frac{e^{-\text{FreeEnergy}(\mathbf{x})}}{Z}, \tag{5.7}$$

with $Z = \sum_{\mathbf{x}} e^{-\text{FreeEnergy}(\mathbf{x})}$, i.e.,

$$\text{FreeEnergy}(\mathbf{x}) = -\log \sum_{\mathbf{h}} e^{-\text{Energy}(\mathbf{x},\mathbf{h})}. \tag{5.8}$$

So the free energy is just a marginalization of energies in the log-domain. The data log-likelihood gradient then has a particularly interesting form. Let us introduce θ to represent parameters of the model. Starting from Equation (5.7), we obtain

$$\begin{aligned} \frac{\partial \log P(\mathbf{x})}{\partial \theta} &= -\frac{\partial \text{FreeEnergy}(\mathbf{x})}{\partial \theta} + \frac{1}{Z} \sum_{\tilde{\mathbf{x}}} e^{-\text{FreeEnergy}(\tilde{\mathbf{x}})} \frac{\partial \text{FreeEnergy}(\tilde{\mathbf{x}})}{\partial \theta} \\ &= -\frac{\partial \text{FreeEnergy}(\mathbf{x})}{\partial \theta} + \sum_{\tilde{\mathbf{x}}} P(\tilde{\mathbf{x}}) \ \frac{\partial \text{FreeEnergy}(\tilde{\mathbf{x}})}{\partial \theta}. \end{aligned} \tag{5.9}$$

Hence the average log-likelihood gradient over the training set is

$$E_{\hat{P}}\left[\frac{\partial \log P(\mathbf{x})}{\partial \theta}\right] = -E_{\hat{P}}\left[\frac{\partial \text{FreeEnergy}(\mathbf{x})}{\partial \theta}\right] + E_P\left[\frac{\partial \text{FreeEnergy}(\mathbf{x})}{\partial \theta}\right] \tag{5.10}$$

where expectations are over $\mathbf{x}$, with $\hat{P}$ the training set empirical distribution and E_P the expectation under the model's distribution P. Therefore, if we could sample from P and compute the free energy tractably, we would have a Monte-Carlo method to obtain a stochastic estimator of the log-likelihood gradient.

If the energy can be written as a sum of terms associated with at most one hidden unit

$$\text{Energy}(\mathbf{x}, \mathbf{h}) = -\beta(\mathbf{x}) + \sum_i \gamma_i(\mathbf{x}, \mathbf{h}_i), \tag{5.11}$$

a condition satisfied in the case of the RBM, then the free energy and numerator of the likelihood can be computed tractably (even though it involves a sum with an exponential number of terms):

$$\begin{aligned}
P(\mathbf{x}) &= \frac{1}{Z} e^{-\text{FreeEnergy}(\mathbf{x})} = \frac{1}{Z} \sum_{\mathbf{h}} e^{-\text{Energy}(\mathbf{x},\mathbf{h})} \\
&= \frac{1}{Z} \sum_{\mathbf{h}_1} \sum_{\mathbf{h}_2} \cdots \sum_{\mathbf{h}_k} e^{\beta(\mathbf{x}) - \sum_i \gamma_i(\mathbf{x},\mathbf{h}_i)} \\
&= \frac{1}{Z} \sum_{\mathbf{h}_1} \sum_{\mathbf{h}_2} \cdots \sum_{\mathbf{h}_k} e^{\beta(\mathbf{x})} \prod_i e^{-\gamma_i(\mathbf{x},\mathbf{h}_i)} \\
&= \frac{e^{\beta(\mathbf{x})}}{Z} \sum_{\mathbf{h}_1} e^{-\gamma_1(\mathbf{x},\mathbf{h}_1)} \sum_{\mathbf{h}_2} e^{-\gamma_2(\mathbf{x},\mathbf{h}_2)} \cdots \sum_{\mathbf{h}_k} e^{-\gamma_k(\mathbf{x},\mathbf{h}_k)} \\
&= \frac{e^{\beta(\mathbf{x})}}{Z} \prod_i \sum_{\mathbf{h}_i} e^{-\gamma_i(\mathbf{x},\mathbf{h}_i)}
\end{aligned} \tag{5.12}$$

In the above, $\sum_{\mathbf{h}_i}$ is a sum over all the values that $\mathbf{h}_i$ can take (e.g., two values in the usual binomial units case); note how that sum is much easier to carry out than the sum $\sum_{\mathbf{h}}$ over all values of $\mathbf{h}$. Note that all sums can be replaced by integrals if $\mathbf{h}$ is continuous, and the same principles apply. In many cases of interest, the sum or integral (over a single hidden unit's values) is easy to compute. The numerator of the

likelihood (i.e., also the free energy) can be computed exactly in the above case, where $\text{Energy}(\mathbf{x}, \mathbf{h}) = -\beta(\mathbf{x}) + \sum_i \gamma_i(\mathbf{x}, \mathbf{h}_i)$, and we have

$$\text{FreeEnergy}(\mathbf{x}) = -\log P(\mathbf{x}) - \log Z = -\beta(\mathbf{x}) - \sum_i \log \sum_{\mathbf{h}_i} e^{-\gamma_i(\mathbf{x}, \mathbf{h}_i)}. \tag{5.13}$$

5.1.2 Conditional Energy-Based Models

Whereas computing the partition function is difficult in general, if our ultimate goal is to make a decision concerning a variable y given a variable $\mathbf{x}$, instead of considering all configurations $(\mathbf{x}, y)$, it is enough to consider the configurations of y for each given $\mathbf{x}$. A common case is one where y can only take values in a small discrete set, i.e.,

$$P(y|\mathbf{x}) = \frac{e^{-\text{Energy}(\mathbf{x}, y)}}{\sum_y e^{-\text{Energy}(\mathbf{x}, y)}}. \tag{5.14}$$

In this case the gradient of the conditional log-likelihood with respect to parameters of the energy function can be computed efficiently. This formulation applies to a discriminant variant of the RBM called Discriminative RBM [97]. Such conditional energy-based models have also been exploited in a series of probabilistic language models based on neural networks [15, 16, 130, 169, 170, 171, 207]. That formulation (or generally when it is easy to sum or maximize over the set of values of the terms of the partition function) has been explored at length [37, 106, 107, 149, 153]. An important and interesting element in the latter work is that it shows that such energy-based models can be optimized not just with respect to log-likelihood but with respect to more general criteria whose gradient has the property of making the energy of "correct" responses decrease while making the energy of competing responses increase. These energy functions do not necessarily give rise to a probabilistic model (because the exponential of the negated energy function is not required to be integrable), but they may nonetheless give rise to a function that can be used to choose y given $\mathbf{x}$, which is often the ultimate goal in applications. Indeed when y takes a finite number of values, $P(y|\mathbf{x})$ can always be computed since the energy function needs to be normalized only over the possible values of y.

5.2 Boltzmann Machines

The Boltzmann machine is a particular type of energy-based model with hidden variables, and RBMs are special forms of Boltzmann machines in which $P(\mathbf{h}|\mathbf{x})$ and $P(\mathbf{x}|\mathbf{h})$ are both tractable because they factorize. In a Boltzmann machine [1, 76, 77], the energy function is a general second-order polynomial:

$$\text{Energy}(\mathbf{x}, \mathbf{h}) = -\mathbf{b}'\mathbf{x} - \mathbf{c}'\mathbf{h} - \mathbf{h}'W\mathbf{x} - \mathbf{x}'U\mathbf{x} - \mathbf{h}'V\mathbf{h}. \tag{5.15}$$

There are two types of parameters, which we collectively denote by θ: the offsets $\mathbf{b}_i$ and $\mathbf{c}_i$ (each associated with a single element of the vector $\mathbf{x}$ or of the vector $\mathbf{h}$), and the weights W_{ij}, U_{ij} and V_{ij} (each associated with a pair of units). Matrices U and V are assumed to be symmetric,[1] and in most models with zeros in the diagonal. Non-zeros in the diagonal can be used to obtain other variants, e.g., with Gaussian instead of binomial units [200].

Because of the quadratic interaction terms in $\mathbf{h}$, the trick to analytically compute the free energy (Equation (5.12)) cannot be applied here. However, an MCMC (Monte Carlo Markov Chain [4]) sampling procedure can be applied in order to obtain a stochastic estimator of the gradient. The gradient of the log-likelihood can be written as follows, starting from Equation (5.6):

$$\begin{aligned}
\frac{\partial \log P(\mathbf{x})}{\partial \theta} &= \frac{\partial \log \sum_{\mathbf{h}} e^{-\text{Energy}(\mathbf{x},\mathbf{h})}}{\partial \theta} - \frac{\partial \log \sum_{\tilde{\mathbf{x}},\mathbf{h}} e^{-\text{Energy}(\tilde{\mathbf{x}},\mathbf{h})}}{\partial \theta} \\
&= -\frac{1}{\sum_{\mathbf{h}} e^{-\text{Energy}(\mathbf{x},\mathbf{h})}} \sum_{\mathbf{h}} e^{-\text{Energy}(\mathbf{x},\mathbf{h})} \frac{\partial \text{Energy}(\mathbf{x},\mathbf{h})}{\partial \theta} \\
&\quad + \frac{1}{\sum_{\tilde{\mathbf{x}},\mathbf{h}} e^{-\text{Energy}(\tilde{\mathbf{x}},\mathbf{h})}} \sum_{\tilde{\mathbf{x}},\mathbf{h}} e^{-\text{Energy}(\tilde{\mathbf{x}},\mathbf{h})} \frac{\partial \text{Energy}(\tilde{\mathbf{x}},\mathbf{h})}{\partial \theta} \\
&= -\sum_{\mathbf{h}} P(\mathbf{h}|\mathbf{x}) \frac{\partial \text{Energy}(\mathbf{x},\mathbf{h})}{\partial \theta} + \sum_{\tilde{\mathbf{x}},\mathbf{h}} P(\tilde{\mathbf{x}},\mathbf{h}) \frac{\partial \text{Energy}(\tilde{\mathbf{x}},\mathbf{h})}{\partial \theta}.
\end{aligned} \tag{5.16}$$

[1] For example, if U was not symmetric, the extra degrees of freedom would be wasted since $\mathbf{x}_i U_{ij} \mathbf{x}_j + \mathbf{x}_j U_{ji} \mathbf{x}_i$ can be rewritten $\mathbf{x}_i(U_{ij} + U_{ji})\mathbf{x}_j = \frac{1}{2}\mathbf{x}_i(U_{ij} + U_{ji})\mathbf{x}_j + \frac{1}{2}\mathbf{x}_j(U_{ij} + U_{ji})\mathbf{x}_i$, i.e., in a symmetric-matrix form.

Note that $(\partial \text{Energy}(\mathbf{x}, \mathbf{h})/\partial\theta)$ is easy to compute. Hence if we have a procedure to sample from $P(\mathbf{h}|\mathbf{x})$ and one to sample from $P(\mathbf{x}, \mathbf{h})$, we can obtain an unbiased stochastic estimator of the log-likelihood gradient. [1, 76, 77] introduced the following terminology: in the *positive phase*, $\mathbf{x}$ is *clamped* to the observed input vector, and we sample $\mathbf{h}$ given $\mathbf{x}$; in the *negative phase* both $\mathbf{x}$ and $\mathbf{h}$ are sampled, ideally from the model itself. In general, only approximate sampling can be achieved tractably, e.g., using an iterative procedure that constructs an MCMC. The MCMC sampling approach introduced in [1, 76, 77] is based on *Gibbs sampling* [4, 57]. Gibbs sampling of the joint of N random variables $S = (S_1, \ldots, S_N)$ is done through a sequence of N sampling sub-steps of the form

$$S_i \sim P(S_i | S_{-i} = \mathbf{s}_{-i}) \tag{5.17}$$

where S_{-i} contains the $N - 1$ other random variables in S, excluding S_i. After these N samples have been obtained, a step of the chain is completed, yielding a sample of S whose distribution converges to $P(S)$ as the number of steps goes to ∞, under some conditions. A sufficient condition for convergence of a finite-state Markov Chain is that it is aperiodic[2] and irreducible.[3]

How can we perform Gibbs sampling in a Boltzmann machine? Let $\mathbf{s} = (\mathbf{x}, \mathbf{h})$ denote all the units in the Boltzmann machine, and $\mathbf{s}_{-i}$ the set of values associated with all units except the ith one. The Boltzmann machine energy function can be rewritten by putting all the parameters in a vector $\mathbf{d}$ and a symmetric matrix A,

$$\text{Energy}(\mathbf{s}) = -\mathbf{d}'\mathbf{s} - \mathbf{s}'A\mathbf{s}. \tag{5.18}$$

Let $\mathbf{d}_{-i}$ denote the vector $\mathbf{d}$ without the element $\mathbf{d}_i$, A_{-i} the matrix A without the ith row and column, and $\mathbf{a}_{-i}$ the vector that is the ith row (or column) of A, without the ith element. Using this notation, we obtain that $P(\mathbf{s}_i|\mathbf{s}_{-i})$ can be computed and sampled from easily in a Boltzmann machine. For example, if $\mathbf{s}_i \in \{0, 1\}$ and the diagonal of A

[2] Aperiodic: no state is periodic with period $k > 1$; a state has period k if one can only return to it at times $t + k$, $t + 2k$, etc.

[3] Irreducible: one can reach any state from any state in finite time with non-zero probability.

is null:

$$
\begin{aligned}
P(\mathbf{s}_i = 1|\mathbf{s}_{-i}) &= \frac{\exp(\mathbf{d}_i + \mathbf{d}'_{-i}\mathbf{s}_{-i} + 2\mathbf{a}'_{-i}\mathbf{s}_{-i} + \mathbf{s}'_{-i}A_{-i}\mathbf{s}_{-i})}{\exp(\mathbf{d}_i + \mathbf{d}'_{-i}\mathbf{s}_{-i} + 2\mathbf{a}'_{-i}\mathbf{s}_{-i} + \mathbf{s}'_{-i}A_{-i}\mathbf{s}_{-i}) \\ &\qquad + \exp(\mathbf{d}'_{-i}\mathbf{s}_{-i} + \mathbf{s}'_{-i}A_{-i}\mathbf{s}_{-i})} \\
&= \frac{\exp(\mathbf{d}_i + 2\mathbf{a}'_{-i}\mathbf{s}_{-i})}{\exp(\mathbf{d}_i + 2\mathbf{a}'_{-i}\mathbf{s}_{-i}) + 1} = \frac{1}{1 + \exp(-\mathbf{d}_i - 2\mathbf{a}'_{-i}\mathbf{s}_{-i})} \\
&= \text{sigm}(\mathbf{d}_i + 2\mathbf{a}'_{-i}\mathbf{s}_{-i}) \qquad (5.19)
\end{aligned}
$$

which is essentially the usual equation for computing a neuron's output in terms of other neurons $\mathbf{s}_{-i}$, in artificial neural networks.

Since two MCMC chains (one for the positive phase and one for the negative phase) are needed for each example $\mathbf{x}$, the computation of the gradient can be very expensive, and training time very long. This is essentially why the Boltzmann machine was replaced in the late 1980's by the back-propagation algorithm for multi-layer neural networks as the dominant learning approach. However, recent work has shown that short chains can sometimes be used successfully, and this is the principle of Contrastive Divergence, discussed in Section 5.4 to train RBMs. Note also that the negative phase chain does not have to be restarted for each new example $\mathbf{x}$ (since it does not depend on the training data), and this observation has been exploited in persistent MCMC estimators [161, 187] discussed in Section 5.4.2.

5.3 Restricted Boltzmann Machines

The *Restricted* Boltzmann Machine (RBM) is the building block of a Deep Belief Network (DBN) because it shares parametrization with individual layers of a DBN, and because efficient learning algorithms were found to train it. The undirected graphical model of an RBM is illustrated in Figure 4.5, showing that the $\mathbf{h}_i$ are independent of each other when conditioning on $\mathbf{x}$ and the $\mathbf{x}_j$ are independent of each other when conditioning on $\mathbf{h}$. In an RBM, $U = 0$ and $V = 0$ in Equation (5.15), i.e., the only interaction terms are between a hidden unit and a visible unit, but not between units of the same layer. This form of model was first introduced under the name of *Harmonium* [178], and learning algorithms (beyond the ones for Boltzmann Machines) were

discussed in [51]. Empirically demonstrated and efficient learning algorithms and variants were proposed more recently [31, 70, 200]. As a consequence of the lack of input–input and hidden-hidden interactions, the energy function is bilinear,

$$\text{Energy}(\mathbf{x}, \mathbf{h}) = -\mathbf{b}'\mathbf{x} - \mathbf{c}'\mathbf{h} - \mathbf{h}'W\mathbf{x} \tag{5.20}$$

and the factorization of the free energy of the input, introduced with Equations (5.11) and (5.13) can be applied with $\beta(\mathbf{x}) = \mathbf{b}'\mathbf{x}$ and $\gamma_i(\mathbf{x}, \mathbf{h}_i) = -\mathbf{h}_i(\mathbf{c}_i + W_i\mathbf{x})$, where W_i is the row vector corresponding to the ith row of W. Therefore the free energy of the input (i.e., its unnormalized log-probability) can be computed efficiently:

$$\text{FreeEnergy}(\mathbf{x}) = -\mathbf{b}'\mathbf{x} - \sum_i \log \sum_{\mathbf{h}_i} e^{\mathbf{h}_i(\mathbf{c}_i + W_i\mathbf{x})}. \tag{5.21}$$

Using the same factorization trick (in Equation (5.12)) due to the affine form of $\text{Energy}(\mathbf{x}, \mathbf{h})$ with respect to $\mathbf{h}$, we readily obtain a tractable expression for the conditional probability $P(\mathbf{h}|\mathbf{x})$:

$$\begin{aligned} P(\mathbf{h}|\mathbf{x}) &= \frac{\exp(\mathbf{b}'\mathbf{x} + \mathbf{c}'\mathbf{h} + \mathbf{h}'W\mathbf{x})}{\sum_{\tilde{\mathbf{h}}} \exp(\mathbf{b}'\mathbf{x} + \mathbf{c}'\tilde{\mathbf{h}} + \tilde{\mathbf{h}}'W\mathbf{x})} \\ &= \frac{\prod_i \exp(\mathbf{c}_i\mathbf{h}_i + \mathbf{h}_i W_i\mathbf{x})}{\prod_i \sum_{\tilde{\mathbf{h}}_i} \exp(\mathbf{c}_i\tilde{\mathbf{h}}_i + \tilde{\mathbf{h}}_i W_i\mathbf{x})} \\ &= \prod_i \frac{\exp(\mathbf{h}_i(\mathbf{c}_i + W_i\mathbf{x}))}{\sum_{\tilde{\mathbf{h}}_i} \exp(\tilde{\mathbf{h}}_i(\mathbf{c}_i + W_i\mathbf{x}))} \\ &= \prod_i P(\mathbf{h}_i|\mathbf{x}). \end{aligned}$$

In the commonly studied case where $\mathbf{h}_i \in \{0, 1\}$, we obtain the usual neuron equation for a neuron's output given its input:

$$P(\mathbf{h}_i = 1|\mathbf{x}) = \frac{e^{\mathbf{c}_i + W_i\mathbf{x}}}{1 + e^{\mathbf{c}_i + W_i\mathbf{x}}} = \text{sigm}(\mathbf{c}_i + W_i\mathbf{x}). \tag{5.22}$$

Since $\mathbf{x}$ and $\mathbf{h}$ play a symmetric role in the energy function, a similar derivation allows to efficiently compute and sample $P(\mathbf{x}|\mathbf{h})$:

$$P(\mathbf{x}|\mathbf{h}) = \prod_i P(\mathbf{x}_i|\mathbf{h}) \tag{5.23}$$

and in the binary case

$$P(\mathbf{x}_j = 1|\mathbf{h}) = \text{sigm}(\mathbf{b}_j + W'_{\cdot j}\mathbf{h}) \tag{5.24}$$

where $W_{\cdot j}$ is the jth column of W.

In [73], binomial input units are used to encode pixel gray levels in input images as if they were the probability of a binary event. In the case of handwritten character images this approximation works well, but in other cases it does not. Experiments showing the advantage of using Gaussian input units rather than binomial units when the inputs are continuous-valued are described in [17]. See [200] for a general formulation where $\mathbf{x}$ and $\mathbf{h}$ (given the other) can be in any of the exponential family distributions (discrete and continuous).

Although RBMs might not be able to represent efficiently some distributions that could be represented compactly with an unrestricted Boltzmann machine, RBMs can represent any discrete distribution [51, 102], if enough hidden units are used. In addition, it can be shown that unless the RBM already perfectly models the training distribution, adding a hidden unit (and properly choosing its weights and offset) can always improve the log-likelihood [102].

An RBM can also be seen as forming a multi-clustering (see Section 3.2), as illustrated in Figure 3.2. Each hidden unit creates a two-region partition of the input space (with a linear separation). When we consider the configurations of say three hidden units, there are eight corresponding possible intersections of three half-planes (by choosing each half-plane among the two half-planes associated with the linear separation performed by a hidden unit). Each of these eight intersections corresponds to a region in input space associated with the same hidden configuration (i.e., code). The binary setting of the hidden units thus identifies one region in input space. For all $\mathbf{x}$ in one of these regions, $P(\mathbf{h}|\mathbf{x})$ is maximal for the corresponding $\mathbf{h}$ configuration. Note that not all configurations of the hidden units correspond to a non-empty region in input space. As illustrated in Figure 3.2, this representation is similar to what an ensemble of two-leaf trees would create.

The sum over the exponential number of possible hidden-layer configurations of an RBM can also be seen as a particularly interesting form

of mixture, with an exponential number of components (with respect to the number of hidden units and of parameters):

$$P(\mathbf{x}) = \sum_{\mathbf{h}} P(\mathbf{x}|\mathbf{h})P(\mathbf{h}) \tag{5.25}$$

where $P(\mathbf{x}|\mathbf{h})$ is the model associated with the component indexed by configuration $\mathbf{h}$. For example, if $P(\mathbf{x}|\mathbf{h})$ is chosen to be Gaussian (see [200, 17]), this is a Gaussian mixture with 2^n components when $\mathbf{h}$ has n bits. Of course, these 2^n components cannot be tuned independently because they depend on shared parameters (the RBM parameters), and that is also the strength of the model, since it can generalize to configurations (regions of input space) for which no training example was seen. We can see that the Gaussian mean (in the Gaussian case) associated with component $\mathbf{h}$ is obtained as a linear combination $\mathbf{b} + W'\mathbf{h}$, i.e., each hidden unit bit $\mathbf{h}_i$ contributes (or not) a vector W_i in the mean.

5.3.1 Gibbs Sampling in RBMs

Sampling from an RBM is useful for several reasons. First of all it is useful in learning algorithms, to obtain an estimator of the log-likelihood gradient. Second, inspection of examples generated from the model is useful to get an idea of what the model has captured or not captured about the data distribution. Since the joint distribution of the top two layers of a DBN is an RBM, sampling from an RBM enables us to sample from a DBN, as elaborated in Section 6.1.

Gibbs sampling in fully connected Boltzmann Machines is slow because there are as many sub-steps in the Gibbs chain as there are units in the network. On the other hand, the factorization enjoyed by RBMs brings two benefits: first we do not need to sample in the positive phase because the free energy (and therefore its gradient) is computed analytically; second, the set of variables in $(\mathbf{x}, \mathbf{h})$ can be sampled in two sub-steps in each step of the Gibbs chain. First we sample $\mathbf{h}$ given $\mathbf{x}$, and then a new $\mathbf{x}$ given $\mathbf{h}$. In general product of experts models, an alternative to Gibbs sampling is hybrid Monte-Carlo [48, 136], an MCMC method involving a number of free-energy gradient computation sub-steps for each step of the Markov chain. The RBM structure

is therefore a special case of product of experts model: the ith term $\log \sum_{\mathbf{h}_i} e^{(\mathbf{c}_i + W_i \mathbf{x})\mathbf{h}_i}$ in Equation (5.21) corresponds to an expert, i.e., there is one expert per hidden neuron and one for the input offset. With that special structure, a very efficient Gibbs sampling can be performed. For k Gibbs steps, starting from a training example (i.e., sampling from $\hat{P}$):

$$\begin{aligned} \mathbf{x}_1 &\sim \hat{P}(\mathbf{x}) \\ \mathbf{h}_1 &\sim P(\mathbf{h}|\mathbf{x}_1) \\ \mathbf{x}_2 &\sim P(\mathbf{x}|\mathbf{h}_1) \\ \mathbf{h}_2 &\sim P(\mathbf{h}|\mathbf{x}_2) \\ &\vdots \\ \mathbf{x}_{k+1} &\sim P(\mathbf{x}|\mathbf{h}_k). \end{aligned} \tag{5.26}$$

It makes sense to start the chain from a training example because as the model becomes better at capturing the structure in the training data, the model distribution P and the training distribution $\hat{P}$ become more similar (having similar statistics). Note that if we started the chain from P itself, it would have converged in one step, so starting from $\hat{P}$ is a good way to ensure that only a few steps are necessary for convergence.

5.4 Contrastive Divergence

Contrastive Divergence is an approximation of the log-likelihood gradient that has been found to be a successful update rule for training RBMs [31]. A pseudo-code is shown in Algorithm 1, with the particular equations for the conditional distributions for the case of binary input and hidden units.

5.4.1 Justifying Contrastive Divergence

To obtain this algorithm, the **first approximation** we are going to make is replace the average over all possible inputs (in the second term of Equation (5.10)) by a single sample. Since we update the parameters often (e.g., with stochastic or mini-batch gradient updates after one or a few training examples), there is already some averaging going on across

Algorithm 1

`RBMupdate`$(\mathbf{x}_1, \epsilon, W, \mathbf{b}, \mathbf{c})$

This is the RBM update procedure for binomial units. It can easily adapted to other types of units.

$\mathbf{x}_1$ is a sample from the training distribution for the RBM

ϵ is a learning rate for the stochastic gradient descent in Contrastive Divergence

W is the RBM weight matrix, of dimension (number of hidden units, number of inputs)

$\mathbf{b}$ is the RBM offset vector for input units

$\mathbf{c}$ is the RBM offset vector for hidden units

Notation: $Q(\mathbf{h}_{2\cdot} = 1|\mathbf{x}_2)$ is the vector with elements $Q(\mathbf{h}_{2i} = 1|\mathbf{x}_2)$

for all hidden units i **do**
- compute $Q(\mathbf{h}_{1i}=1|\mathbf{x}_1)$ (for binomial units, $\text{sigm}(\mathbf{c}_i + \sum_j W_{ij}\mathbf{x}_{1j})$)
- sample $\mathbf{h}_{1i} \in \{0,1\}$ from $Q(\mathbf{h}_{1i}|\mathbf{x}_1)$

end for

for all visible units j **do**
- compute $P(\mathbf{x}_{2j}=1|\mathbf{h}_1)$ (for binomial units, $\text{sigm}(\mathbf{b}_j + \sum_i W_{ij}\mathbf{h}_{1i})$)
- sample $\mathbf{x}_{2j} \in \{0,1\}$ from $P(\mathbf{x}_{2j} = 1|\mathbf{h}_1)$

end for

for all hidden units i **do**
- compute $Q(\mathbf{h}_{2i}=1|\mathbf{x}_2)$ (for binomial units, $\text{sigm}(\mathbf{c}_i + \sum_j W_{ij}\mathbf{x}_{2j})$)

end for

- $W \leftarrow W + \epsilon(\mathbf{h}_1\mathbf{x}_1' - Q(\mathbf{h}_{2\cdot} = 1|\mathbf{x}_2)\mathbf{x}_2')$
- $\mathbf{b} \leftarrow \mathbf{b} + \epsilon(\mathbf{x}_1 - \mathbf{x}_2)$
- $\mathbf{c} \leftarrow \mathbf{c} + \epsilon(\mathbf{h}_1 - Q(\mathbf{h}_{2\cdot} = 1|\mathbf{x}_2))$

updates (which we know to work well [105]), and the extra variance introduced by taking one or a few MCMC samples instead of doing the complete sum might be partially canceled in the process of online gradient updates, over consecutive parameter updates. We introduce additional variance with this approximation of the gradient, but it does not hurt much if it is comparable or smaller than the variance due to online gradient descent.

Running a long MCMC chain is still very expensive. The idea of k-step Contrastive Divergence (CD-k) [69, 70] is simple, and involves a **second approximation**, which introduces some bias in the gradient: run the MCMC chain $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{k+1}$ for only k steps *starting from the observed example* $\mathbf{x}_1 = \mathbf{x}$. The CD-$k$ update (i.e., not the log-likelihood gradient) after seeing example $\mathbf{x}$ is, therefore,

$$\Delta\theta \propto \frac{\partial \text{FreeEnergy}(\mathbf{x})}{\partial\theta} - \frac{\partial \text{FreeEnergy}(\tilde{\mathbf{x}})}{\partial\theta} \tag{5.27}$$

where $\tilde{\mathbf{x}} = \mathbf{x}_{k+1}$ is the last sample from our Markov chain, obtained after k steps. We know that when $k \to \infty$, the bias goes away. We also know that when the model distribution is very close to the empirical distribution, i.e., $P \approx \hat{P}$, then when we start the chain from $\mathbf{x}$ (a sample from $\hat{P}$) the MCMC has already converged, and we need only one step to obtain an unbiased sample from P (although it would still be correlated with $\mathbf{x}$).

The surprising empirical result is that even $k = 1$ (CD-1) often gives good results. An extensive numerical comparison of training with CD-k versus exact log-likelihood gradient has been presented in [31]. In these experiments, taking k larger than 1 gives more precise results, although very good approximations of the solution can be obtained even with $k = 1$. Theoretical results [12] discussed in Section 5.4.3 help to understand why small values of k can work: CD-k corresponds to keeping the first k terms of a series that converges to the log-likelihood gradient.

One way to interpret Contrastive Divergence is that it is approximating the log-likelihood gradient *locally* around the training point $\mathbf{x}_1$. The stochastic reconstruction $\tilde{\mathbf{x}} = \mathbf{x}_{k+1}$ (for CD-k) has a distribution (given $\mathbf{x}_1$) which is in some sense centered around $\mathbf{x}_1$ and becomes more spread out around it as k increases, until it becomes the model distribution. The CD-k update will decrease the free energy of the training point $\mathbf{x}_1$ (which would increase its likelihood if all the other free energies were kept constant), and increase the free energy of $\tilde{\mathbf{x}}$, which is in the neighborhood of $\mathbf{x}_1$. Note that $\tilde{\mathbf{x}}$ is in the neighborhood of $\mathbf{x}_1$, but at the same time more likely to be in regions of high probability under the model (especially for k larger). As argued by [106],

what is mostly needed from the training algorithm for an energy-based model is that it makes the energy (free energy, here, to marginalize hidden variables) of observed inputs smaller, shoveling "energy" elsewhere, and most importantly in areas of low energy. The Contrastive Divergence algorithm is fueled by the *contrast* between the statistics collected when the input is a real training example and when the input is a chain sample. As further argued in the next section, one can think of the unsupervised learning problem as discovering a decision surface that can roughly separate the regions of high probability (where there are many observed training examples) from the rest. Therefore, we want to penalize the model when it generates examples on the wrong side of that divide, and a good way to identify where that divide should be moved is to compare training examples with samples from the model.

5.4.2 Alternatives to Contrastive Divergence

An exciting recent development in the research on learning algorithms for RBMs is use of a so-called persistent MCMC for the negative phase [161, 187], following an approach already introduced in [135]. The idea is simple: keep a background MCMC chain $\ldots \mathbf{x}_t \rightarrow \mathbf{h}_t \rightarrow \mathbf{x}_{t+1} \rightarrow \mathbf{h}_{t+1} \ldots$ to obtain the negative phase samples (which should be from the model). Instead of running a short chain as in CD-k, the approximation made is that we ignore the fact that parameters are changing as we move along the chain, i.e., we do not run a separate chain for each value of the parameters (as in the traditional Boltzmann Machine learning algorithm). Maybe because the parameters move slowly, the approximation works very well, usually giving rise to better log-likelihood than CD-k (experiments were against $k = 1$ and $k = 10$). The trade-off with CD-1 is that the variance is larger but the bias is smaller. Something interesting also happens [188]: the model systematically moves away from the samples obtained in the negative phase, and this interacts with the chain itself, preventing it from staying in the same region very long, substantially improving the mixing rate of the chain. This is a very desirable and unforeseen effect, which helps to explore more quickly the space of RBM configurations.

Another alternative to Contrastive Divergence is Score Matching [84, 85, 86], a general approach to train energy-based models in which the energy can be computed tractably, but not the normalization constant Z. The score function of a density $p(\mathbf{x}) = q(\mathbf{x})/Z$ is $\psi = (\partial \log p(\mathbf{x}))/\partial \mathbf{x}$, and we exploit the fact that the score function of our model does not depend on its normalization constant, i.e., $\psi = (\partial \log q(\mathbf{x}))/\partial \mathbf{x}$. The basic idea is to match the score function of the model with the score function of the empirical density. The average (under the empirical density) of the squared norm of the difference between the two score functions can be written in terms of squares of the model score function and second derivatives $(\partial^2 \log q(\mathbf{x}))/\partial \mathbf{x}^2$. Score matching has been shown to be locally consistent [84], i.e., converging if the model family matches the data generating process, and it has been used for unsupervised models of image and audio data [94].

5.4.3 Truncations of the Log-Likelihood Gradient in Gibbs-Chain Models

Here, we approach the Contrastive Divergence update rule from a different perspective, which gives rise to possible generalizations of it and links it to the reconstruction error often used to monitor its performance and that is used to optimize auto-encoders (Equation (4.7)). The inspiration for this derivation comes from [73]: first from the idea (explained in Section 8.1) that the Gibbs chain can be associated with an infinite directed graphical model (which here we associate with an expansion of the log-likelihood gradient), and second that the convergence of the chain justifies Contrastive Divergence (since the expected value of Equation (5.27) becomes equivalent to Equation (5.9) when the chain sample $\tilde{\mathbf{x}}$ comes from the model). In particular, we are interested in clarifying and understanding the bias in the Contrastive Divergence update rule, compared to using the true (intractable) gradient of the log-likelihood.

Consider a converging Markov chain $\mathbf{x}_t \Rightarrow \mathbf{h}_t \Rightarrow \mathbf{x}_{t+1} \Rightarrow \cdots$ defined by conditional distributions $P(\mathbf{h}_t|\mathbf{x}_t)$ and $P(\mathbf{x}_{t+1}|\mathbf{h}_t)$, with $\mathbf{x}_1$ sampled from the training data empirical distribution. The following theorem,

demonstrated by [12], shows how one can expand the log-likelihood gradient for any $t \geq 1$.

Theorem 5.1. Consider the converging Gibbs chain $\mathbf{x}_1 \Rightarrow \mathbf{h}_1 \Rightarrow \mathbf{x}_2 \Rightarrow \mathbf{h}_2 \cdots$ starting at data point $\mathbf{x}_1$. The log-likelihood gradient can be written

$$\frac{\partial \log P(\mathbf{x}_1)}{\partial \theta} = -\frac{\partial \text{FreeEnergy}(\mathbf{x}_1)}{\partial \theta} + E\left[\frac{\partial \text{FreeEnergy}(\mathbf{x}_t)}{\partial \theta}\right] + E\left[\frac{\partial \log P(\mathbf{x}_t)}{\partial \theta}\right] \tag{5.28}$$

and the final term converges to zero as t goes to infinity.

Since the final term becomes small as t increases, that justifies truncating the chain to k steps in the Markov chain, using the approximation

$$\frac{\partial \log P(\mathbf{x}_1)}{\partial \theta} \simeq -\frac{\partial \text{FreeEnergy}(\mathbf{x}_1)}{\partial \theta} + E\left[\frac{\partial \text{FreeEnergy}(\mathbf{x}_{k+1})}{\partial \theta}\right]$$

which is exactly the CD-k update (Equation (5.27)) when we replace the expectation with a single sample $\tilde{\mathbf{x}} = \mathbf{x}_{k+1}$. This tells us that the bias of CD-k is $E[(\partial \log P(\mathbf{x}_{k+1}))/\partial\theta]$. Experiments and theory support the idea that CD-k yields better and faster convergence (in terms of number of iterations) than CD-$(k-1)$, due to smaller bias (though the computational overhead might not always be worth it). However, although experiments show that the CD-k bias can indeed be large when k is small, empirically the update rule of CD-k still mostly moves the model's parameters in the same quadrant as log-likelihood gradient [12]. This is in agreement with the good results can be obtained even with $k = 1$. An intuitive picture that may help to understand the phenomenon is the following: when the input example $\mathbf{x}_1$ is used to initialize the chain, even the first Markov chain step (to $\mathbf{x}_2$) tends to be in the right direction compared to $\mathbf{x}_1$, i.e., roughly going down the energy landscape from $\mathbf{x}_1$. Since the gradient depends on the change between $\mathbf{x}_2$ and $\mathbf{x}_1$, we tend to get the direction of the gradient right.

So CD-1 corresponds to truncating the chain after two samples (one from $\mathbf{h}_1|\mathbf{x}_1$, and one from $\mathbf{x}_2|\mathbf{h}_1$). What about stopping after the first one (i.e., $\mathbf{h}_1|\mathbf{x}_1$)? It can be analyzed from the following log-likelihood gradient expansion [12]:

$$\frac{\partial \log P(\mathbf{x}_1)}{\partial \theta} = E\left[\frac{\partial \log P(\mathbf{x}_1|\mathbf{h}_1)}{\partial \theta}\right] - E\left[\frac{\partial \log P(\mathbf{h}_1)}{\partial \theta}\right]. \quad (5.29)$$

Let us consider a mean-field approximation of the first expectation, in which instead of the average over all $\mathbf{h}_1$ configurations according to $P(\mathbf{h}_1|\mathbf{x}_1)$ one replaces $\mathbf{h}_1$ by its average configuration $\widehat{\mathbf{h}}_1 = E[\mathbf{h}_1|\mathbf{x}_1]$, yielding:

$$E\left[\frac{\partial \log P(\mathbf{x}_1|\mathbf{h}_1)}{\partial \theta}\right] \simeq \frac{\partial \log P(\mathbf{x}_1|\widehat{\mathbf{h}}_1)}{\partial \theta}. \quad (5.30)$$

If, as in CD, we then ignore the second expectation in Equation (5.29) (incurring an additional bias in the estimation of the log-likelihood gradient), we then obtain the right-hand side of Equation (5.30) as an update direction, which is minus the gradient of the *reconstruction error*,

$$-\log P(\mathbf{x}_1|\widehat{\mathbf{h}}_1)$$

typically used to train auto-encoders (see Equation (4.7) with $\mathbf{c}(\mathbf{x}) = E[\mathbf{h}|\mathbf{x}]$).[4]

So we have found that the truncation of the Gibbs chain gives rise to first approximation (one sample) to roughly reconstruction error (through a biased mean-field approximation), with slightly better approximation (two samples) to CD-1 (approximating the expectation by a sample), and with more terms to CD-k (still approximating expectations by samples). Note that reconstruction error is deterministically computed and is correlated with log-likelihood, which is why it has been used to track progress when training RBMs with CD.

[4] It is debatable whether or not one would take into account the fact that $\widehat{\mathbf{h}}_1$ depends on θ when computing the gradient in the mean-field approximation of Equation (5.30), but it must be the case to draw a direct link with auto-encoders.

5.4.4 Model Samples Are Negative Examples

Here, we argue that training an energy-based model can be achieved by solving a series of classification problems in which one tries to discriminate training examples from samples generated by the model. In the Boltzmann machine learning algorithms, as well as in Contrastive Divergence, an important element is the ability to *sample from the model*, maybe approximately. An elegant way to understand the value of these samples in improving the log-likelihood was introduced in [201], using a connection with boosting. We start by explaining the idea informally and then formalize it, justifying algorithms based on training the generative model with a classification criterion *separating model samples from training examples*. The maximum likelihood criterion wants the likelihood to be high on the training examples and low elsewhere. If we already have a model and we want to increase its likelihood, the contrast between where the model puts high probability (represented by samples) and where the training examples are indicates how to change the model. If we were able to approximately separate training examples from model samples with a decision surface, we could increase likelihood by reducing the value of the energy function on one side of the decision surface (the side where there are more training examples) and increasing it on the other side (the side where there are more samples from the model). Mathematically, consider the gradient of the log-likelihood with respect to the parameters of the FreeEnergy($\mathbf{x}$) (or Energy($\mathbf{x}$) if we do not introduce explicit hidden variables), given in Equation (5.10). Now consider a highly regularized two-class probabilistic classifier that will attempt to separate training samples of $\hat{P}(\mathbf{x})$ from model samples of $P(\mathbf{x})$, and which is only able to produce an output probability $q(\mathbf{x}) = P(y = 1|\mathbf{x})$ barely different from $\frac{1}{2}$ (hopefully on the right side more often than not). Let $q(\mathbf{x}) = \text{sigm}(-a(\mathbf{x}))$, i.e., $-a(\mathbf{x})$ is the discriminant function or an unnormalized conditional log-probability, just like the free energy. Let $\tilde{P}$ denote the empirical distribution over $(\mathbf{x}, y)$ pairs, and $\tilde{P}_i$ the distribution over $\mathbf{x}$ when $y = i$. Assume that $\tilde{P}(y = 1) = \tilde{P}(y = 0) = 1/2$, so that $\forall f,\ E_{\tilde{P}}[f(\mathbf{x}, y)] = E_{\tilde{P}_1}[f(\mathbf{x}, 1)]\tilde{P}(y = 1) + E_{\tilde{P}_0}[f(\mathbf{x}, 0)]\tilde{P}(y = 0) = \frac{1}{2}(E_{\tilde{P}_1}[f(\mathbf{x}, 1)] + E_{\tilde{P}_0}[f(\mathbf{x}, 0)])$. Using this, the average conditional

log-likelihood gradient for this probabilistic classifier is written

$$\begin{aligned} E_{\tilde{P}}\left[\frac{\partial \log P(y|\mathbf{x})}{\partial \theta}\right] &= E_{\tilde{P}}\left[\frac{\partial (y \log q(\mathbf{x}) + (1-y)\log(1-q(\mathbf{x})))}{\partial \theta}\right] \\ &= \frac{1}{2}\left(E_{\tilde{P}_1}\left[(q(\mathbf{x})-1)\frac{\partial a(\mathbf{x})}{\partial \theta}\right] + E_{\tilde{P}_0}\left[q(\mathbf{x})\frac{\partial a(\mathbf{x})}{\partial \theta}\right]\right) \\ &\approx \frac{1}{4}\left(-E_{\tilde{P}_1}\left[\frac{\partial a(\mathbf{x})}{\partial \theta}\right] + E_{\tilde{P}_0}\left[\frac{\partial a(\mathbf{x})}{\partial \theta}\right]\right) \end{aligned} \tag{5.31}$$

where the last equality is when the classifier is highly regularized: when the output weights are small, $a(\mathbf{x})$ is close to 0 and $q(\mathbf{x}) \approx 1/2$, so that $(1 - q(\mathbf{x})) \approx q(\mathbf{x})$. This expression for the log-likelihood gradient corresponds exactly to the one obtained for energy-based models where the likelihood is expressed in terms of a free energy (Equation (5.10)), when we interpret training examples from $\tilde{P}_1$ as positive examples ($y = 1$) (i.e., $\tilde{P}_1 = \hat{P}$) and model samples as negative examples ($y = 0$, i.e., $\tilde{P}_0 = P$). The gradient is also similar in structure to the Contrastive Divergence gradient estimator (Equation (5.27)). One way to interpret this result is that if we could improve a classifier that separated training samples from model samples, we could improve the log-likelihood of the model, by putting more probability mass on the side of training samples. Practically, this could be achieved with a classifier whose discriminant function was defined as the free energy of a generative model (up to a multiplicative factor), and assuming one could obtain samples (possibly approximate) from the model. A particular variant of this idea has been used to justify a boosting-like incremental algorithm for adding experts in products of experts [201].