Undirected Graphical Models

Handout Notes - Generative and Deep Learning (GDL)

Davide Bacciu - University of Pisa

Notation. Random variables are uppercase (e.g., X, Y, H, V) and observed values are lowercase. For collections of variables we use bold notation, e.g. $\mathbf{X} = (X_1, \ldots, X_n)$. A dataset is denoted by $\mathcal{D}$ with size $N = |\mathcal{D}|$. Model parameters are denoted by θ; in energy-based models these may include weights, biases, or feature coefficients. For a clique C, the subvector of variables in that clique is written $\mathbf{X}_C$. Potential functions are denoted by $\psi_C(\mathbf{X}_C)$. The partition function is denoted by Z (or $Z(\mathbf{x})$ in conditional models). In conditional random fields, observable inputs are written $\mathbf{X}$ and structured outputs are written $\mathbf{Y}$. For Boltzmann machines, visible units are $\mathbf{V}$ and hidden units are $\mathbf{H}$, with joint state $\mathbf{S} = [\mathbf{V}, \mathbf{H}]$.

1 Why undirected graphical models?

Directed graphical models are natural when we want to describe a generative process: one variable causes or probabilistically generates another, and the graph encodes this asymmetric structure. However, not every modeling problem is naturally directional. Sometimes what we want to express is not a causal story but a set of *compatibility constraints*: certain configurations of variables should be preferred, and others should be discouraged.

This is the perspective of *undirected graphical models*, also known as *Markov random fields* (MRFs) or *Markov networks*. Instead of specifying how variables generate one another, an undirected model specifies which collections of variables interact and which local patterns are compatible. This makes undirected models especially appealing when:

- interactions are symmetric rather than causal,
- we want to encode soft constraints among neighboring variables,
- prediction is conditional and discriminative rather than fully generative,
- the model should favor globally consistent labelings or reconstructions.

Typical examples include image denoising, segmentation, sequence labeling, and energy-based models such as Boltzmann machines.

2 Markov random fields

A Markov random field is an undirected graph

$$\mathcal{G} = (\mathcal{V}, \mathcal{E}),$$

where each node $v \in \mathcal{V}$ corresponds to a random variable X_v, and each edge in $\mathcal{E}$ encodes a direct dependency between variables. The absence of an edge encodes a conditional independence statement, but unlike directed models, the semantics are local and symmetric.

The defining idea is that a variable interacts directly only with its neighbors in the graph. Thus the graph captures a notion of local dependence structure, and global probabilities are assembled from local compatibility terms.

2.1 Clique factorization

Let $\mathbf{X} = (X_1, \ldots, X_n)$ be the collection of all random variables in the graph. If the distribution is strictly positive, the Hammersley–Clifford theorem implies that it can be written as

$$p(\mathbf{x}) = \frac{1}{Z} \prod_{C \in \mathcal{C}} \psi_C(\mathbf{x}_C),$$

where:

- $\mathcal{C}$ is the set of maximal cliques of the graph,
- $\psi_C(\mathbf{x}_C) > 0$ is a *potential function* associated with clique C,
- Z is the partition function,

$$Z = \sum_{\mathbf{x}} \prod_{C \in \mathcal{C}} \psi_C(\mathbf{x}_C)$$

for discrete variables, or the analogous integral in the continuous case.

A crucial conceptual point is that ψ_C is *not* a probability distribution. It is only a positive score. What matters probabilistically is the normalized product of all clique potentials.

Worked example — Why the partition function is needed

Suppose a model on two binary variables $X_1, X_2 \in \{-1, +1\}$ is defined by

$$\psi(X_1, X_2) = \begin{cases} 3 & \text{if } X_1 = X_2, \\ 1 & \text{if } X_1 \neq X_2. \end{cases}$$

Then the unnormalized scores are

$$\psi(+1,+1) = 3, \qquad \psi(+1,-1) = 1, \qquad \psi(-1,+1) = 1, \qquad \psi(-1,-1) = 3.$$

These four values do not sum to one, so they are not yet probabilities. The partition function is

$$Z = 3 + 1 + 1 + 3 = 8.$$

Hence the normalized distribution is

$$p(x_1, x_2) = \frac{1}{8}\psi(x_1, x_2).$$

For example,

$$p(X_1 = +1, X_2 = +1) = \frac{3}{8}, \qquad p(X_1 = +1, X_2 = -1) = \frac{1}{8}.$$

The potential only expresses preference for equal signs; normalization turns those preferences into probabilities.

3 Potentials, energy, and feature functions

In practice, potentials are often parameterized rather than specified as arbitrary tables. A common choice is an exponential-family parameterization:

$$\psi_C(\mathbf{x}_C) = \exp\left(\sum_k \theta_{Ck} f_{Ck}(\mathbf{x}_C) \right),$$

where:

- f_{Ck} are *feature functions* (or sufficient statistics),
- $\theta_{Ck} \in \mathbb{R}$ are learnable parameters.

Substituting into the clique factorization yields

$$p(\mathbf{x}) = \frac{1}{Z} \exp\left(\sum_C \sum_k \theta_{Ck} f_{Ck}(\mathbf{x}_C) \right).$$

This form is often rewritten in terms of an *energy function*

$$E(\mathbf{x}) = -\sum_C \sum_k \theta_{Ck} f_{Ck}(\mathbf{x}_C),$$

so that

$$p(\mathbf{x}) = \frac{1}{Z} \exp\big(-E(\mathbf{x})\big).$$

Low-energy configurations are more probable than high-energy ones. This is the Boltzmann form and is central to energy-based modeling.

3.1 Interpretation of feature functions

A feature function encodes a local pattern or compatibility. For instance, in image restoration one may want:

- a feature that favors agreement between a latent clean pixel and the observed noisy pixel,
- a feature that favors neighboring latent pixels having the same label.

These features need not be complicated. They are often simple linear or indicator functions that express local couplings.

Worked example — Simple pairwise features for image labeling

Suppose $X_i \in \{-1, +1\}$ is an observed noisy pixel and $Y_i \in \{-1, +1\}$ its hidden clean label. Two natural features are:

$$f_i(X_i, Y_i) = X_i Y_i, \qquad f_{ij}(Y_i, Y_j) = Y_i Y_j.$$

The first feature is positive when observed and latent pixel have the same sign, and negative otherwise. If its parameter θ_i is positive, the model prefers agreement between observation and latent label.

The second feature is positive when neighboring latent labels agree, and negative otherwise. If its parameter is positive, the model prefers spatial smoothness. Thus a denoising model can be built from two intuitively meaningful pressures: fit the observed image and encourage neighboring pixels to be similar.

4 Factor graphs

Standard undirected graphs tell us which variables are directly connected, but they do not explicitly display how a clique potential itself factorizes into smaller parts. A *factor graph* makes this structure explicit.

A factor graph is bipartite:

- circular nodes represent random variables,
- square nodes represent factors or local feature-based potentials.

An edge connects a factor node to a variable node if that factor depends on that variable.

This representation is valuable for inference because message-passing algorithms such as sum-product and max-product are most naturally written on factor graphs. It also makes feature design more transparent in conditional models such as CRFs.

5 Conditional random fields

In many machine-learning tasks, part of the variables are always observed. These observed variables form the input $\mathbf{X}$, while the unknown variables of interest form the structured output $\mathbf{Y}$. When the goal is prediction of $\mathbf{Y}$ given $\mathbf{X}$, it is often wasteful to model the full joint distribution $p(\mathbf{X}, \mathbf{Y})$. Instead, one can model the conditional distribution directly.

A *conditional random field* (CRF) is an undirected model for

$$p(\mathbf{Y} \mid \mathbf{X}).$$

Its general exponential-family form is

$$p(\mathbf{y} \mid \mathbf{x}, \theta) = \frac{1}{Z(\mathbf{x})} \prod_k \exp\big(\theta_k f_k(\mathbf{x}_k, \mathbf{y}_k)\big),$$

or equivalently

$$p(\mathbf{y} \mid \mathbf{x}, \theta) = \frac{1}{Z(\mathbf{x})} \exp\left(\sum_k \theta_k f_k(\mathbf{x}_k, \mathbf{y}_k)\right),$$

with input-dependent partition function

$$Z(\mathbf{x}) = \sum_{\mathbf{y}} \exp\left(\sum_k \theta_k f_k(\mathbf{x}_k, \mathbf{y}_k)\right).$$

The key difference from a generative model is that only $\mathbf{Y}$ is normalized over. The inputs $\mathbf{X}$ are treated as given.

5.1 Why CRFs are discriminative

A CRF can use rich, overlapping, and even correlated input features without needing to define a generative model for the inputs. This is one of the main reasons CRFs became popular in sequence labeling and structured prediction. They allow us to focus modeling capacity on the conditional structure we actually care about.

6 Linear-chain CRFs

The most important special case is the *linear-chain CRF*, used for sequence labeling tasks such as part-of-speech tagging, named entity recognition, or biological sequence annotation.

Let $\mathbf{X} = (X_1, \ldots, X_T)$ be an observed input sequence and $\mathbf{Y} = (Y_1, \ldots, Y_T)$ the corresponding label sequence. A linear-chain CRF takes the form

$$p(\mathbf{y} \mid \mathbf{x}, \theta) = \frac{1}{Z(\mathbf{x})} \prod_{t=1}^{T} \exp\left(\sum_k \theta_k f_k(y_t, y_{t-1}, \mathbf{x}, t)\right).$$

Equivalently,

$$p(\mathbf{y} \mid \mathbf{x}, \theta) = \frac{1}{Z(\mathbf{x})} \exp\left(\sum_{t=1}^{T} \sum_k \theta_k f_k(y_t, y_{t-1}, \mathbf{x}, t)\right).$$

This structure mirrors a hidden Markov model in topology, but with two key differences:

- the model is conditional rather than generative,
- feature functions may depend on arbitrary properties of the input sequence.

Typical feature functions are of the form

$$f_k(y_t, y_{t-1}, \mathbf{x}, t) = \mathbb{I}[y_t = a,\ y_{t-1} = b]\ q_k(\mathbf{x}, t),$$

where q_k is an observation feature such as:

- "the current word begins with a capital letter",
- "the suffix is `-ing`",
- "the previous symbol is punctuation".

7 Inference in linear-chain CRFs

Inference in a linear-chain CRF is tractable because the graph is a chain. As in hidden Markov models, dynamic programming can be used.

Define the local clique potential

$$\psi_t(y_t, y_{t-1}, \mathbf{x}) = \exp\left(\sum_k \theta_k f_k(y_t, y_{t-1}, \mathbf{x}, t)\right).$$

Then

$$p(\mathbf{y} \mid \mathbf{x}, \theta) = \frac{1}{Z(\mathbf{x})} \prod_{t=1}^{T} \psi_t(y_t, y_{t-1}, \mathbf{x}).$$

7.1 Forward and backward messages

Define forward messages

$$\alpha_t(i) = \sum_{y_1, \ldots, y_{t-1}} \prod_{\tau=1}^{t} \psi_\tau(y_\tau, y_{\tau-1}, \mathbf{x}) \quad \text{with } y_t = i,$$

and backward messages

$$\beta_t(j) = \sum_{y_{t+1}, \ldots, y_T} \prod_{\tau=t+1}^{T} \psi_\tau(y_\tau, y_{\tau-1}, \mathbf{x}) \quad \text{with } y_t = j.$$

These satisfy the recursions

$$\alpha_t(i) = \sum_j \psi_t(i, j, \mathbf{x})\, \alpha_{t-1}(j), \qquad \beta_t(j) = \sum_i \psi_{t+1}(i, j, \mathbf{x})\, \beta_{t+1}(i).$$

The partition function is obtained by summing the last forward messages:

$$Z(\mathbf{x}) = \sum_i \alpha_T(i).$$

Worked example — Pairwise posterior in a linear-chain CRF

The pairwise marginal needed for training is

$$p(Y_t = i, Y_{t-1} = j \mid \mathbf{x}, \theta).$$

Using the chain decomposition,

$$p(Y_t = i, Y_{t-1} = j \mid \mathbf{x}, \theta) \propto \alpha_{t-1}(j)\, \psi_t(i, j, \mathbf{x})\, \beta_t(i).$$

Normalizing over all state pairs gives

$$\boxed{p(Y_t = i, Y_{t-1} = j \mid \mathbf{x}, \theta) = \frac{\alpha_{t-1}(j)\, \psi_t(i, j, \mathbf{x})\, \beta_t(i)}{\sum_{i', j'} \alpha_{t-1}(j')\, \psi_t(i', j', \mathbf{x})\, \beta_t(i')}.}$$

This is the CRF analogue of the smoothing posterior in HMMs, and it is obtained by exact sum-product inference on the chain.

8 Training linear-chain CRFs

Suppose we have a labeled dataset

$$\mathcal{D} = \{(\mathbf{x}^{(n)}, \mathbf{y}^{(n)})\}_{n=1}^{N}.$$

CRF training is usually done by conditional maximum likelihood:

$$\max_{\theta}\ \mathcal{L}(\theta), \qquad \mathcal{L}(\theta) = \sum_{n=1}^{N} \log p(\mathbf{y}^{(n)} \mid \mathbf{x}^{(n)}, \theta).$$

Substituting the linear-chain CRF form gives

$$\mathcal{L}(\theta) = \sum_{n=1}^{N} \left[\sum_{t=1}^{T_n} \sum_{k} \theta_k f_k(y_t^{(n)}, y_{t-1}^{(n)}, \mathbf{x}^{(n)}, t) - \log Z(\mathbf{x}^{(n)}) \right].$$

A common regularized objective adds an ℓ_2 penalty:

$$\mathcal{L}_{\text{reg}}(\theta) = \mathcal{L}(\theta) - \sum_{k} \frac{\theta_k^2}{2\sigma^2}.$$

8.1 Gradient of the conditional log-likelihood

Unlike naive Bayes or simple multinomial models, CRF parameters do not generally admit a closed-form solution. We optimize the objective by gradient methods.

Worked example — Gradient of the CRF objective

For one parameter θ_k, differentiate the regularized objective:

$$\frac{\partial \mathcal{L}_{\text{reg}}(\theta)}{\partial \theta_k} = \sum_{n=1}^{N} \sum_{t=1}^{T_n} f_k(y_t^{(n)}, y_{t-1}^{(n)}, \mathbf{x}^{(n)}, t) - \sum_{n=1}^{N} \frac{\partial \log Z(\mathbf{x}^{(n)})}{\partial \theta_k} - \frac{\theta_k}{\sigma^2}.$$

Now

$$Z(\mathbf{x}) = \sum_{\mathbf{y}} \exp\left(\sum_t \sum_r \theta_r f_r(y_t, y_{t-1}, \mathbf{x}, t) \right),$$

so

$$\frac{\partial \log Z(\mathbf{x})}{\partial \theta_k} = \frac{1}{Z(\mathbf{x})} \sum_{\mathbf{y}} \exp\left(\sum_t \sum_r \theta_r f_r(y_t, y_{t-1}, \mathbf{x}, t) \right) \left(\sum_t f_k(y_t, y_{t-1}, \mathbf{x}, t) \right).$$

Recognizing the conditional model distribution,

$$\frac{\partial \log Z(\mathbf{x})}{\partial \theta_k} = \sum_{\mathbf{y}} p(\mathbf{y} \mid \mathbf{x}, \theta) \sum_t f_k(y_t, y_{t-1}, \mathbf{x}, t).$$

Hence

$$\boxed{\frac{\partial \mathcal{L}_{\text{reg}}(\theta)}{\partial \theta_k} = \sum_{n,t} f_k(y_t^{(n)}, y_{t-1}^{(n)}, \mathbf{x}^{(n)}, t) - \sum_{n,t} \sum_{y,y'} f_k(y, y', \mathbf{x}^{(n)}, t)\, p(y, y' \mid \mathbf{x}^{(n)}, \theta) - \frac{\theta_k}{\sigma^2}.}$$

So the gradient is the difference between:

- the *empirical expected feature count* under the labeled data,
- the *model expected feature count* under the CRF posterior,

plus regularization.

This has exactly the familiar exponential-family structure: at optimum, empirical and model feature expectations match, up to the regularizer.

8.2 Optimization

In practice, the objective is optimized by gradient ascent, gradient descent on the negative log-likelihood, or more commonly quasi-Newton methods and stochastic-gradient variants. For SGD, one computes the single-example gradient

$$\nabla \mathcal{L}_n(\theta)$$

and updates parameters iteratively. The required pairwise marginals $p(y_t, y_{t-1} \mid \mathbf{x}^{(n)}, \theta)$ are computed by forward–backward.

9 Boltzmann machines

We now move from tractable conditional undirected models to energy-based models where inference and learning are more difficult.

A *Boltzmann machine* is a special kind of Markov random field defined on binary stochastic units. Its state vector is

$$\mathbf{S} = [\mathbf{V}, \mathbf{H}],$$

where:

- $\mathbf{V} \in \{0,1\}^{d_v}$ are visible units,
- $\mathbf{H} \in \{0,1\}^{d_h}$ are hidden units.

The model uses an energy function of the form

$$E(\mathbf{s}) = -\frac{1}{2}\mathbf{s}^\top M \mathbf{s} - \mathbf{b}^\top \mathbf{s},$$

where M is symmetric and has zero diagonal (no self-connections), and $\mathbf{b}$ is a bias vector.

The induced distribution is

$$p(\mathbf{s}) = \frac{1}{Z}\exp\big(-E(\mathbf{s})\big), \qquad Z = \sum_{\mathbf{s}} \exp\big(-E(\mathbf{s})\big).$$

9.1 Boltzmann machines as stochastic neural networks

A Boltzmann machine can also be interpreted as a stochastic recurrent neural network. Each unit updates stochastically according to a sigmoid probability determined by its local field. If

$$x_j = \sum_i M_{ij} s_i + b_j,$$

then

$$p(S_j = 1 \mid \mathbf{s}_{\setminus j}) = \sigma(x_j) = \frac{1}{1+e^{-x_j}}.$$

This local conditional is the basis for Gibbs sampling in Boltzmann machines.

10 Learning in Boltzmann machines

Suppose only visible configurations $\mathbf{v}^{(1)}, \dots, \mathbf{v}^{(N)}$ are observed in the data. Then learning proceeds by maximizing the likelihood of visible patterns:

$$\mathcal{L}(M, \mathbf{b}) = \frac{1}{N}\sum_{n=1}^{N} \log p(\mathbf{v}^{(n)} \mid M, \mathbf{b}),$$

where

$$p(\mathbf{v} \mid M, \mathbf{b}) = \sum_{\mathbf{h}} p(\mathbf{v}, \mathbf{h} \mid M, \mathbf{b}).$$

As in other latent-variable models, the hidden units are marginalized out.

Differentiating the log-likelihood gives a characteristic "data minus model" form.

Worked example — Gradient structure for Boltzmann-machine learning

Consider one weight M_{ij}. Differentiating the log-likelihood yields

$$\frac{\partial \mathcal{L}}{\partial M_{ij}} = \mathbb{E}_{p(\mathbf{h}\mid\mathbf{v})}[S_i S_j] - \mathbb{E}_{p(\mathbf{s})}[S_i S_j].$$

These two terms are often called:

- the *clamped* or *data* expectation,
- the *free* or *model* expectation.

The first term averages pairwise correlations while visible units are clamped to the data. The second term averages the same correlations under the model's equilibrium distribution. Learning therefore increases weights that are under-explained by the model and decreases weights that are over-produced by the model.

The difficulty is that the model expectation involves the full partition function and is generally intractable. This motivates approximate learning by sampling.

11 Restricted Boltzmann machines

A *restricted Boltzmann machine* (RBM) is a special Boltzmann machine with bipartite connectivity:

- visible units connect only to hidden units,
- hidden units connect only to visible units,
- there are no visible-visible or hidden-hidden edges.

This restriction is crucial because it makes the conditional distributions factorize:

$$p(\mathbf{h} \mid \mathbf{v}) = \prod_j p(h_j \mid \mathbf{v}), \qquad p(\mathbf{v} \mid \mathbf{h}) = \prod_i p(v_i \mid \mathbf{h}).$$

For binary units,

$$p(H_j = 1 \mid \mathbf{v}) = \sigma\Big(\sum_i M_{ij} v_i + c_j\Big), \qquad p(V_i = 1 \mid \mathbf{h}) = \sigma\Big(\sum_j M_{ij} h_j + b_i\Big).$$

Because all hidden units are conditionally independent given the visibles (and vice versa), block Gibbs sampling becomes easy and highly parallelizable.

12 Learning RBMs

The gradient of the RBM log-likelihood has the same general form as for Boltzmann machines:

$$\frac{\partial \mathcal{L}}{\partial M_{ij}} = \langle V_i H_j \rangle_{\text{data}} - \langle V_i H_j \rangle_{\text{model}},$$

where:

- $\langle \cdot \rangle_{\text{data}}$ is an expectation with visibles clamped to training data,
- $\langle \cdot \rangle_{\text{model}}$ is an expectation under the model equilibrium distribution.

The data term is relatively easy to estimate because $p(\mathbf{h} \mid \mathbf{v})$ factorizes. The model term is harder because it requires sampling from the model distribution after the Markov chain has mixed.

13 Contrastive divergence

Exact maximum-likelihood learning of RBMs would require running the Gibbs chain long enough to approximate the equilibrium model expectation. This can be too slow in practice. A widely used approximation is *contrastive divergence* (CD).

The most common version is CD-1:

1. clamp a data vector $\mathbf{v}^{(0)}$ on the visible layer,
2. sample hidden units $\mathbf{h}^{(0)} \sim p(\mathbf{h} \mid \mathbf{v}^{(0)})$,
3. reconstruct visibles $\mathbf{v}^{(1)} \sim p(\mathbf{v} \mid \mathbf{h}^{(0)})$,
4. sample hidden units again $\mathbf{h}^{(1)} \sim p(\mathbf{h} \mid \mathbf{v}^{(1)})$,
5. approximate the gradient by

$$\frac{\partial \mathcal{L}}{\partial M_{ij}} \approx v_i^{(0)} h_j^{(0)} - v_i^{(1)} h_j^{(1)}.$$

Worked example — Why contrastive divergence is only an approximation

The exact RBM gradient is

$$\frac{\partial \mathcal{L}}{\partial M_{ij}} = \langle V_i H_j \rangle_{\text{data}} - \langle V_i H_j \rangle_{\text{model}},$$

where the second expectation should be taken at equilibrium. In CD-1, however, the equilibrium term is replaced by a one-step reconstruction term:

$$\langle V_i H_j \rangle_{\text{model}} \approx \langle V_i H_j \rangle_{\text{recon}}.$$

This means the negative phase is computed after only one short Gibbs chain started from the data, rather than after full mixing. Hence CD is biased as an estimator of the true maximum-likelihood gradient. Nevertheless, it often works well enough in practice to learn useful features.

This practical success, despite the approximation, is one of the reasons RBMs played a historic role in early deep learning.

14 Pseudocode: training an RBM with contrastive divergence

The following procedure summarizes the standard contrastive-divergence training loop for a binary RBM. We assume visible units $\mathbf{V} \in \{0,1\}^{d_v}$, hidden units $\mathbf{H} \in \{0,1\}^{d_h}$, weight matrix $M \in \mathbb{R}^{d_v \times d_h}$, visible biases $\mathbf{b}$, hidden biases $\mathbf{c}$, learning rate η, and K Gibbs steps per update. The case $K = 1$ corresponds to the commonly used CD-1 algorithm.

Worked example — Contrastive-divergence training for a binary RBM

Input: dataset $\mathcal{D} = \{\mathbf{v}^{(n)}\}_{n=1}^{N}$, number of epochs E, learning rate η, Gibbs length K.
Output: learned parameters $(M, \mathbf{b}, \mathbf{c})$.

Initialize:
Choose small random values for M, and initialize $\mathbf{b}, \mathbf{c}$ (often to zeros).

For each epoch $e = 1, \ldots, E$**:**

1. Optionally shuffle the training examples.
2. For each training vector $\mathbf{v}^{(0)} \in \mathcal{D}$:

(a) **Positive phase (data-dependent statistics).**
For each hidden unit $j = 1, \ldots, d_h$, compute

$$p(H_j = 1 \mid \mathbf{v}^{(0)}) = \sigma\left(\sum_{i=1}^{d_v} M_{ij} v_i^{(0)} + c_j\right).$$

Sample

$$h_j^{(0)} \sim \text{Bernoulli}\big(p(H_j = 1 \mid \mathbf{v}^{(0)})\big).$$

(b) **Negative phase (short Gibbs chain).**
Set $\mathbf{v}^{(K,0)} \leftarrow \mathbf{v}^{(0)}$ and $\mathbf{h}^{(K,0)} \leftarrow \mathbf{h}^{(0)}$. For $t = 1, \ldots, K$:

i. Sample reconstructed visible units from the current hidden state:

$$p(V_i = 1 \mid \mathbf{h}^{(K,t-1)}) = \sigma\left(\sum_{j=1}^{d_h} M_{ij} h_j^{(K,t-1)} + b_i\right), \qquad i = 1, \ldots, d_v,$$

then sample

$$v_i^{(K,t)} \sim \text{Bernoulli}\big(p(V_i = 1 \mid \mathbf{h}^{(K,t-1)})\big).$$

ii. Sample hidden units again from the reconstructed visible vector:

$$p(H_j = 1 \mid \mathbf{v}^{(K,t)}) = \sigma\left(\sum_{i=1}^{d_v} M_{ij} v_i^{(K,t)} + c_j\right), \qquad j = 1, \ldots, d_h,$$

then sample

$$h_j^{(K,t)} \sim \text{Bernoulli}\big(p(H_j = 1 \mid \mathbf{v}^{(K,t)})\big).$$

Denote the final reconstructed pair by

$$\mathbf{v}^{(K)} = \mathbf{v}^{(K,K)}, \qquad \mathbf{h}^{(K)} = \mathbf{h}^{(K,K)}.$$

(c) **Parameter update.**
For each weight M_{ij}, perform the CD update

$$M_{ij} \leftarrow M_{ij} + \eta\Big(v_i^{(0)} h_j^{(0)} - v_i^{(K)} h_j^{(K)}\Big).$$

Update visible biases:

$$b_i \leftarrow b_i + \eta\big(v_i^{(0)} - v_i^{(K)}\big), \qquad i = 1, \ldots, d_v.$$

Update hidden biases:

$$c_j \leftarrow c_j + \eta\big(h_j^{(0)} - h_j^{(K)}\big), \qquad j = 1, \ldots, d_h.$$

Return $(M, \mathbf{b}, \mathbf{c})$.

Mini-batch version. In practice, updates are usually performed on mini-batches rather than one sample at a time. Then the positive and negative statistics are averaged over the mini-batch before applying the update.

Using probabilities instead of sampled hidden states. A common implementation choice is to use hidden activation probabilities in the positive phase, and sometimes also in the negative phase, instead of sampled binary states when computing gradients. This reduces update variance, although the conceptual Gibbs chain is still defined by sampling.

CD-K and persistent CD. CD-1 is the most common practical choice, but one may use larger K to reduce bias. A related alternative is *persistent contrastive divergence*, where the negative-phase chain is not restarted from the data each time, but continued across updates.