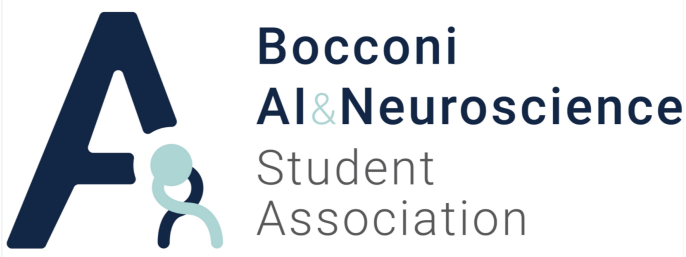
A learning algorithm for Boltzmann machines

David H. Ackley, Geoffrey E. Hinton, Terrence J. Sejnowski

Bassi Giuseppe Berlinghieri Giovanni Ayan Beisenbek

Bocconi AI & Neuroscience Student Association

April 3, 2025



Setting: Overview

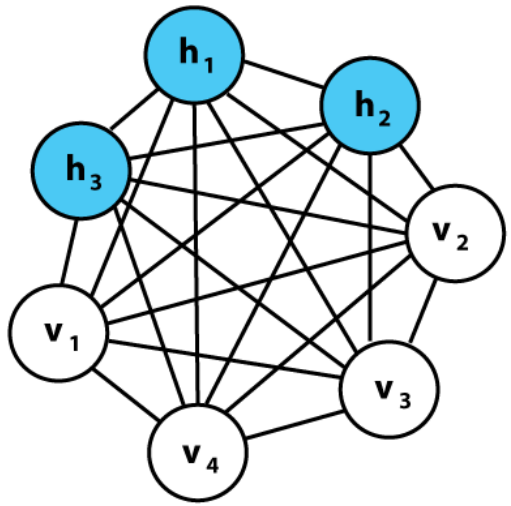
Boltzmann machines (BMs) are stochastic, symmetric, binary – recurrent neural networks (RNNs) that employ an unsupervised learning algorithm.

Recurrent Neural Networks:

N neurons, each neuron i has a state $x_i(t)$. Neurons are interconnected by weights $w_{ij} \in \mathbb{R}$, and they may also receive external inputs $I_i(t)$.

- In our setting, we assume that $I_i(t)$ is time independent, denote it by b_i, and refer to it as the *bias*. The total input, or *local field*, on neuron i is:

$$h_i(t) = \sum_{j=1}^{N} w_{ij} x_j(t) + b_i$$



Binary: A neural network is binary if each neuron satisfies:

$$x_i(t) \in \{-1, 1\}$$

Setting: Stochastic Dynamics

Deterministic Setting: In a deterministic setting, updates are given by

$$x_i(t+1) = \text{sign}(h_i(t)) = \text{sign}\left(\sum_{j=1}^{N} w_{ij}x_j(t) + b_i\right).$$

Stochastic Setting: In BMs we introduce a *stochasticity* (or noise) *parameter* $T \in [0, +\infty]$, called temperature, and a corresponding parameter $\beta := 1/T$. For convenience, let us also define the sigmoid function

$$\sigma : x \mapsto \frac{1}{1+e^{-x}}.$$

The dynamics is now the following:

$$P(x_i(t+1) = \pm 1) = \frac{e^{\pm\beta h_i(t)}}{e^{\beta h_i(t)} + e^{-\beta h_i(t)}} = \frac{1}{1+e^{\mp 2\beta h_i(t)}} = \sigma(\pm 2\beta h_i(t)).$$

Setting: Stochastic Dynamics (2)

$$P(x_i(t+1) = \pm 1) = \frac{e^{\pm \beta h_i(t)}}{e^{\beta h_i(t)} + e^{-\beta h_i(t)}} = \frac{1}{1 + e^{\mp 2\beta h_i(t)}} = \sigma(\pm 2\beta h_i(t)).$$

- If $T = +\infty$, then $\beta = 0$ and $\sigma(\pm 2\beta h_i(t)) = \sigma(0) = 1/2$, meaning perfect stochasticity.
- If $T = 0$, then $\beta = +\infty$, and we recover the purely determinism (Hopfield Model). Here, the system always goes downhill in energy, until a local minimum is reached.
- BMs work with $0 < T < \infty$.
  - Low temperatures $\implies$ strong bias in favor of states with low energy.
  - High temperatures $\implies$ faster convergence to an equilibrium, but the system does not effectively discriminate between small energy differences.
  - simulated annealing: gradually lowering the temperature $\implies$ As the temperature is reduced, the network becomes increasingly sensitive to finer energy variations, allowing it to settle into a better minimum.

Convergence of Glauber's Dynamics

Symmetric: A technical assumption ($w_{ij} = w_{ji}$ for every i and j) that permits the following theorem (makes the system conservative rather than dissipative, so that we can define an energy function):

Theorem: Let $\mathcal{X} = \{-1, 1\}^N$ be the set of possible states of N binary spins (neurons). Let $\beta = 1/T$ be the inverse of the temperature $T \in [0, +\infty]$. Consider a (Glauber) Markov Dynamics $\{\mathbf{X}(t)\}_{t \in \mathbb{N}}$ on $\mathcal{X}$ with local transitions defined with asynchronous updates which follow the probabilistic rule

$$P(x_i(t+1) = x_i(t)) = \sigma(2\beta x_i(t) h_i(t)),$$

with

$$h_i(t) = \sum_{j=1}^{N} w_{ij} x_j(t) + b_i.$$

Convergence of Glauber's Dynamics

Consider a (Glauber) Markov Dynamics $\{\mathbf{X}(t)\}_{t\in\mathbb{N}}$ on $\mathcal{X}$ with local transitions defined with asynchronous updates which follow the probabilistic rule

$$P(x_i(t+1) = x_i(t)) = \sigma(2\beta x_i(t) h_i(t)),$$

with

$$h_i(t) = \sum_j w_{ij} x_j(t) + b_i.$$

Then the system always converges in distribution to a unique stationary distribution

$$\pi(\mathbf{x}) = \frac{1}{Z} \exp\left(-\beta E(\mathbf{x})\right), \quad \text{where} \quad Z = \sum_{\mathbf{x}\in\mathcal{X}} \exp(-\beta E(\mathbf{x}))$$

and

$$E(\mathbf{x}) = -\sum_{i<j} w_{ij} x_i x_j - \sum_i b_i x_i,$$

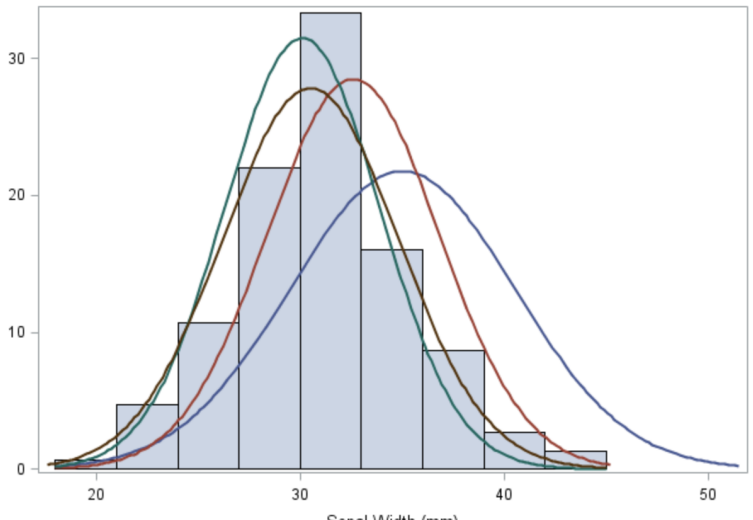
with $E : \mathcal{X} \to \mathbb{R}$ known as the energy function of the system.

Problem Statement

Given a sample $\mathbf{x}^\mu \in \{0,1\}^N$ for all $\mu \in \{1,\dots,M\}$ with **true distribution** $P_{\text{data}}(\mathbf{x})$, our goal is to find a distribution $P_{\text{model}}(\mathbf{x}) = P_\theta(\mathbf{x})$ that approximates it. Statisticians have identified at least two classical approaches to tackle this problem:

- **Maximum Likelihood Estimator:** find the most likely parameter θ by maximizing the log-likelihood function

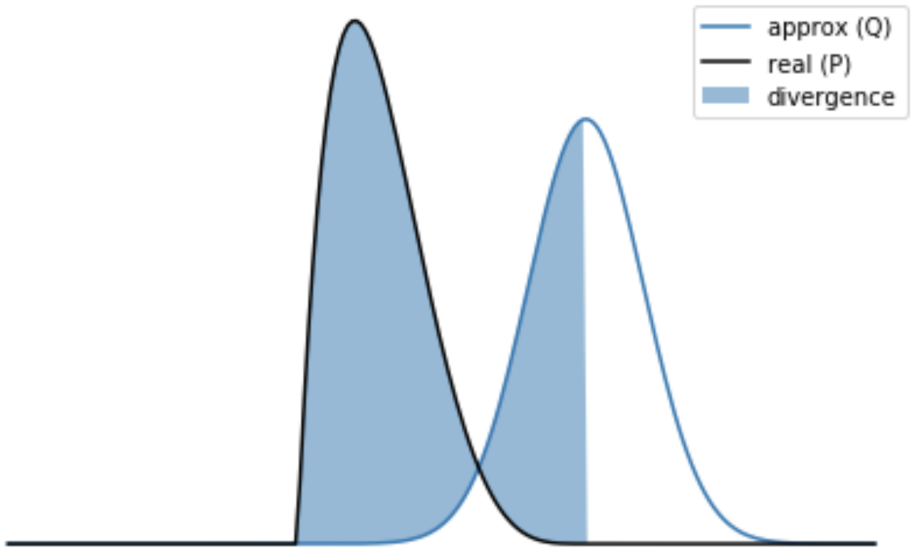
$$\mathscr{L}(\theta;\mathbf{x}) = \log\left(\prod_{\mu=1}^{M} P_\theta(\mathbf{x})\right) = \sum_{\mu=1}^{M} \log P_\theta(\mathbf{x}).$$



Problem Statement (2)

- **Kullback-Leibler Divergence (KL):** minimize this non symmetric "distance" between $P_{\text{data}}(\mathbf{x})$ and $P_{\text{model}}(\mathbf{x})$:

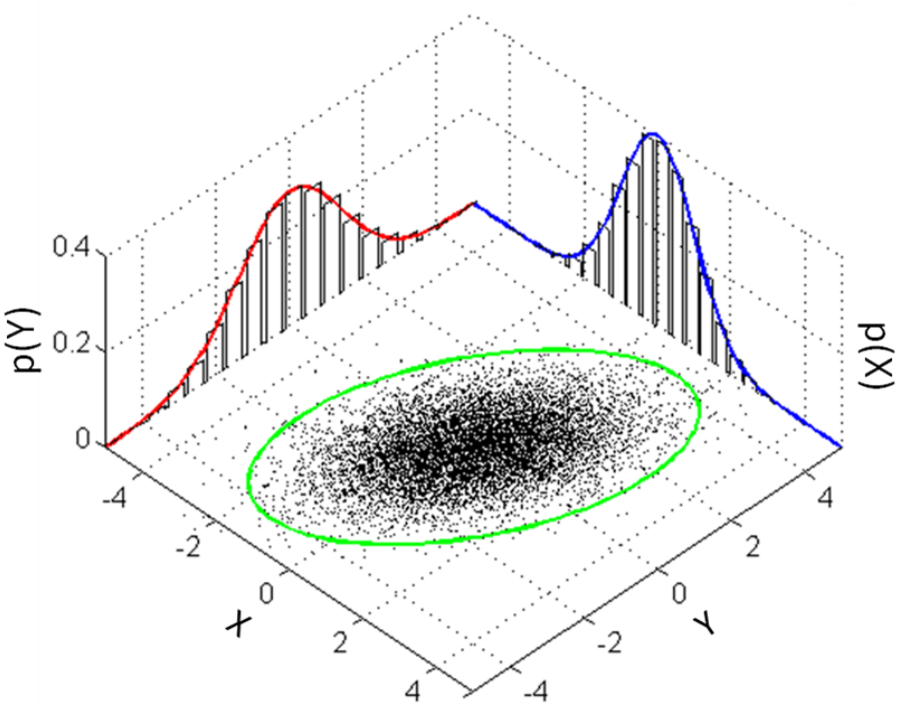
$$K(P_{\text{data}} \| P_{\text{model}}) = \mathbb{E}_{P_{\text{data}}} \left(\log \frac{P_{\text{data}}(\mathbf{X})}{P_\theta(\mathbf{X})} \right) = \sum_{\mathbf{x} \in \{0,1\}^N} P_{\text{data}}(\mathbf{x}) \log \left(\frac{P_{\text{data}}(\mathbf{x})}{P_\theta(\mathbf{x})} \right).$$



Parametric Assumption for $P_\theta(\mathbf{x})$

Underlying these approaches is the need for an assumption about P_θ, that is, a parametric expression (depending on θ) to describe it. Recall that a multivariate Gaussian distribution in N dimensions is given by

$$P(\mathbf{x}) = \frac{1}{Z} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})\right) = \frac{1}{Z} \exp\left(\mathbf{x}^T \mathbf{W}\mathbf{x} + \mathbf{b}^T \mathbf{x}\right).$$



Discrete Exponential Family

The discrete analogue of the latter distribution is the following:

$$P_\theta(\mathbf{x}) = \frac{1}{Z} \exp\left(\sum_{i<j} w_{ij} x_i x_j + \sum_i b_i x_i\right),$$

where $w_{ij} \in \mathbb{R}$ and $b_i \in \mathbb{R}$ form the parameter vector θ.

- For good reason, it is referred to as the **discrete exponential family**, the **binary Markov random field**, or the **Ising model distribution**.
- It's exactly the probability distribution describing the state to which a Boltzmann machine converges!
- BMs act as generative models: once the neural network converges, it can generate new data following the distribution P_θ.

Boltzmann machines - computational tasks

- BMs act as generative models: once the neural network converges, it can generate new data following the distribution P_θ.
  - BMs are useful generative models, with applications in image generation, text synthesis, drug discovery, music composition, data augmentation, and more.
- We then compare these generated data with the original ones and update the parameters so that P_{model} approximates P_{true}.

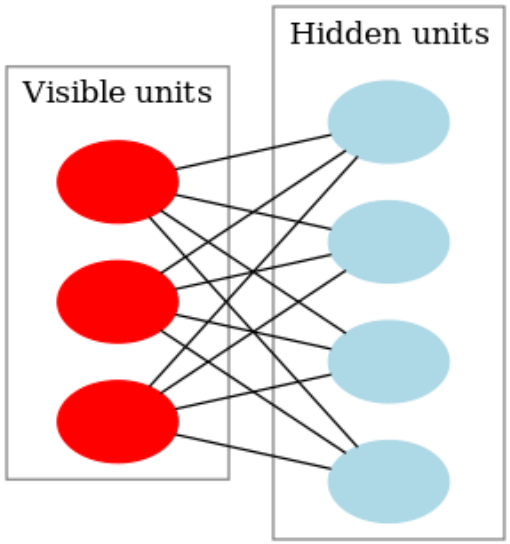
- **Learning Phase:** The parameters are updated to better approximate P_{true} with P_θ, thereby modifying $E_\theta(\mathbf{x})$.
- **Sampling/Generation Phase:** The Boltzmann machine is allowed to evolve until convergence. The resulting state is stochastic, with its probability P_θ determined solely by the parameters.

Restricted Boltzmann Machines (RBMs)

Neural network organized as a bipartite graph with two layers:

- **Visible Layer:** Consisting of visible neurons $\mathbf{x} = (x_1, \ldots, x_N) \in \{0,1\}^N$. Update:

$$P(h_j = 1 \mid \mathbf{x}) = \sigma\left(\sum_{k=1}^{N} w_{jk} x_k + b_j\right)$$



- **Hidden Layer:** Comprising hidden neurons $\mathbf{h} = (h_1, \ldots, h_M) \in \{0,1\}^M$. Update:

$$P(x_k = 1 \mid \mathbf{h}) = \sigma\left(\sum_{j=1}^{M} w_{jk} h_j + c_k\right)$$

No connections within the same layer. The updates occur in a alternating, bipartite fashion.

Restricted Boltzmann Machines — Energy Function

- Energy:

$$E(\mathbf{x}, \mathbf{h}) = -\sum_{j,k} w_{jk} h_j x_k - \sum_k c_k x_k - \sum_j b_j h_j.$$

- Boltzmann distribution of states

$$P_\theta(\mathbf{x}, \mathbf{h}) = \frac{1}{Z} e^{-E(\mathbf{x},\mathbf{h})}$$

- Partition function

$$Z = \sum_{\mathbf{x},\mathbf{h}} e^{-E(\mathbf{x},\mathbf{h})}$$

The key difference is that, with no recurrent connections and a clear separation between the two parts of the network, the units within each layer become conditionally independent given the other layer. This greatly improves sampling speed and optimization stability.

Learning Algorithm for RBMs

Remember that our goal is to learn the underlying distribution of the data. Suppose we have a dataset of examples

$$\mathbf{x}^{\mu} \in \mathbb{R}^{N} \quad \text{(visible vectors)}$$

for $\mu = 1, 2, \ldots, M$ and we want to fit the model so that it assigns a high probability to these examples.

For a single data point $\mathbf{x}^{\mu}$, the log-likelihood under the model is:

$$\log P(\mathbf{x}^{\mu}) = \log \sum_{\mathbf{h}} P(\mathbf{x}^{\mu}, \mathbf{h}).$$

Thus, the training objective becomes:

$$\mathscr{L}(\theta; \mathbf{x}) = -\frac{1}{M} \sum_{\mu} \log P(\mathbf{x}^{\mu}) = -\frac{1}{M} \sum_{\mu} \log \left(\sum_{\mathbf{h}} P(\mathbf{x}^{\mu}, \mathbf{h}) \right).$$

Gradient of the Loss Function

Our problem reduces on minimizing the training objective

$$\mathscr{L}(\theta; \mathbf{x}) = -\frac{1}{M} \sum_{\mu} \log P(\mathbf{x}^{\mu})$$

via gradient descent. After some derivations, the gradient of $\mathscr{L}$ with respect to the parameters θ (where $\theta = \{w_{ij}, b_i, c_j\}$) is:

$$\nabla_{\theta} \mathscr{L} = \frac{1}{M} \sum_{\mu} \left\langle \frac{\partial E(\mathbf{x}^{\mu}, \mathbf{h})}{\partial \theta} \right\rangle_{\mathbf{h}} - \left\langle \frac{\partial E(\mathbf{x}, \mathbf{h})}{\partial \theta} \right\rangle_{\mathbf{x},\mathbf{h}},$$

where $\langle \cdot \rangle_x$ and $\langle \cdot \rangle_{x,h}$ are just different notations for $\mathbb{E}[\cdot]$ with respect to, respectively, x and x, h.

Gradient of the Loss Function (2)

The gradient of $\mathscr{L}$ with respect to the parameters θ is:

$$\nabla_\theta \mathscr{L} = \frac{1}{M} \sum_\mu \left\langle \frac{\partial E(\mathbf{x}^\mu, \mathbf{h})}{\partial \theta} \right\rangle_{\mathbf{h}} - \left\langle \frac{\partial E(\mathbf{x}, \mathbf{h})}{\partial \theta} \right\rangle_{\mathbf{x},\mathbf{h}}.$$

This can be written more compactly as:

$$\nabla_\theta \mathscr{L} = \mathbb{E}_{\text{data}} \left[\frac{\partial E(\mathbf{x}, \mathbf{h})}{\partial \theta} \right] - \mathbb{E}_{\text{model}} \left[\frac{\partial E(\mathbf{x}, \mathbf{h})}{\partial \theta} \right],$$

where we have:

- $\mathbb{E}_{\text{data}}[\cdot]$: expectation over $P(\mathbf{h}|\mathbf{x})$ for $\mathbf{x}$ sampled from the dataset - i.e., the positive phase.
- $\mathbb{E}_{\text{model}}[\cdot]$: expectation over $P(\mathbf{x}, \mathbf{h})$, the model's joint distribution - i.e., the negative phase.

Parameter Updates

Exploiting the energy function of the network, we obtain for a fixed data point:

$$\frac{\partial E}{\partial w_{jk}} = -h_j\, x_k, \qquad \frac{\partial E}{\partial b_j} = -h_j, \qquad \frac{\partial E}{\partial c_k} = -x_k.$$

The parameter updates for gradient descent are (with learning rate η):

- **Weights:**

$$\Delta w_{jk} = \eta \left(\frac{1}{M} \sum_{\mu} \langle h_j\, x_k^{\mu} \rangle - \langle h_j\, x_k \rangle \right)$$

- **Hidden Biases:**

$$\Delta b_j = \eta \left(\frac{1}{M} \sum_{\mu} \langle h_j \rangle^{\mu} - \langle h_j \rangle \right)$$

- **Visible Biases:**

$$\Delta c_k = \eta \left(\frac{1}{M} \sum_{\mu} x_k^{\mu} - \langle x_k \rangle \right)$$

Motivation for Contrastive Divergence

- **Intractability:** Directly computing the partition function in RBMs requires summing over all possible configurations of **x** and **h**, which grows exponentially with the number of units.
- **Negative Phase Complexity:** The negative phase, which requires computing an expectation over the Boltzmann distribution, is particularly expensive.
- **Solution:** We use Monte Carlo sampling to approximate the negative phase, thereby sidestepping the need to compute the full partition function.

Contrastive Divergence (CD) approximates the negative phase via short Gibbs chains, drastically reducing the computational overhead and making RBM training feasible.

Basic Idea of Contrastive Divergence

The algorithm is based on two main phases:

Positive Phase:
- Given a training sample $\mathbf{x}^{(0)}$, sample the hidden activations $\mathbf{h}^{(0)}$ from $P(\mathbf{h} \mid \mathbf{x}^{(0)})$.
- Record the **positive statistic** $\mathbf{x}^{(0)}\mathbf{h}^{(0)}$.

Negative Phase:
- Perform k steps of Gibbs sampling to obtain the model samples $\mathbf{x}^{(k)}$ and $\mathbf{h}^{(k)}$.
- Record the **negative statistic** $\mathbf{x}^{(k)}\mathbf{h}^{(k)}$.

The difference between these two statistics is then used to update the model parameters.

Parameter Updates

The parameter updates are computed as follows:

$$\Delta w_{ij} = \eta\left(x_i^{(0)}\, h_j^{(0)} - x_i^{(k)}\, h_j^{(k)}\right)$$

$$\Delta c_i = \eta\left(x_i^{(0)} - x_i^{(k)}\right)$$

$$\Delta b_j = \eta\left(h_j^{(0)} - h_j^{(k)}\right)$$

where η is the learning rate. These updates are then repeated for each data-point in the training set.

Contrastive Divergence: Online Pseudocode

Algorithm 1 Contrastive Divergence (CD-k) for an RBM – Online Version

Require: Training set $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\}$; number of epochs $\mathrm{maxEpochs}$; learning rate η; number of Gibbs steps k.

Initialize weights $\mathbf{W}$ and biases $\mathbf{b}, \mathbf{c}$ (small random values).
for $\mathrm{epoch} = 1$ to $\mathrm{maxEpochs}$ **do**
  Shuffle the training set.
  for $n = 1$ to N **do**
    $\mathbf{x}_{\text{data}} \leftarrow \mathbf{x}^{(n)}$ ▷ Positive Phase: Compute hidden activations
    $\mathbf{h}^{(0)} \sim \sigma\left(\mathbf{W}^\top \mathbf{x}_{\text{data}} + \mathbf{c}\right)$ ▷ Negative Phase: Run k steps of Gibbs sampling
    Set $\mathbf{h} \leftarrow \mathbf{h}^{(0)}$
    for $\ell = 1$ to k **do**
      Sample $\mathbf{x}^{(\ell)} \sim \sigma\left(\mathbf{W}\,\mathbf{h} + \mathbf{b}\right)$
      Sample $\mathbf{h} \sim \sigma\left(\mathbf{W}^\top \mathbf{x}^{(\ell)} + \mathbf{c}\right)$
    end for
    Set $\mathbf{x}^{(k)} \leftarrow \mathbf{x}^{(\ell)}$ ▷ // Final visible sample ▷ Parameter Updates:
    $\mathbf{W} \mathrel{+}= \eta \left[\mathbf{x}_{\text{data}} (\mathbf{h}^{(0)})^\top - \mathbf{x}^{(k)}\,\mathbf{h}^\top\right]$
    $\mathbf{b} \mathrel{+}= \eta \left[\mathbf{x}_{\text{data}} - \mathbf{x}^{(k)}\right]$
    $\mathbf{c} \mathrel{+}= \eta \left[\mathbf{h}^{(0)} - \mathbf{h}\right]$
  end for
end for

Contrastive Divergence: Mini-Batch Pseudocode

Algorithm 2 Contrastive Divergence for an RBM – Mini-Batch Version

Require: Training set $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\}$; number of epochs $\mathrm{maxEpochs}$; mini-batch size B; learning rate η; number of Gibbs steps k.

Initialize $\mathbf{W}$, $\mathbf{b}$ (visible bias), $\mathbf{c}$ (hidden bias) with small random values.
for $\mathrm{epoch} = 1$ to $\mathrm{maxEpochs}$ **do**
  Shuffle the training set and split into mini-batches of size B.
  for each mini-batch $\mathcal{B}$ **do**
    Set $\Delta\mathbf{W} \leftarrow \mathbf{0}$, $\Delta\mathbf{b} \leftarrow \mathbf{0}$, $\Delta\mathbf{c} \leftarrow \mathbf{0}$.
    for each sample $\mathbf{x}_{\text{data}} \in \mathcal{B}$ **do** ▷ Positive Phase:
      $\mathbf{h}^{(0)} \sim \sigma\Big(\mathbf{W}^\top \mathbf{x}_{\text{data}} + \mathbf{c}\Big)$ ▷ Negative Phase: Run k steps of Gibbs sampling
      Set $\mathbf{h} \leftarrow \mathbf{h}^{(0)}$
      for $\ell = 1$ to k **do**
        Sample $\mathbf{x}^{(\ell)} \sim \sigma\Big(\mathbf{W}\,\mathbf{h} + \mathbf{b}\Big)$
        Sample $\mathbf{h} \sim \sigma\Big(\mathbf{W}^\top \mathbf{x}^{(\ell)} + \mathbf{c}\Big)$
      end for
      Set $\mathbf{x}^{(k)} \leftarrow \mathbf{x}^{(\ell)}$. ▷ Accumulate Updates:
      $\Delta\mathbf{W} \mathrel{+}= \mathbf{x}_{\text{data}}\,(\mathbf{h}^{(0)})^\top - \mathbf{x}^{(k)}\,\mathbf{h}^\top$
      $\Delta\mathbf{b} \mathrel{+}= \mathbf{x}_{\text{data}} - \mathbf{x}^{(k)}$
      $\Delta\mathbf{c} \mathrel{+}= \mathbf{h}^{(0)} - \mathbf{h}$
    end for ▷ Update Parameters for the Mini-Batch:
    $\mathbf{W} \mathrel{+}= \frac{\eta}{B}\,\Delta\mathbf{W}$
    $\mathbf{b} \mathrel{+}= \frac{\eta}{B}\,\Delta\mathbf{b}$
    $\mathbf{c} \mathrel{+}= \frac{\eta}{B}\,\Delta\mathbf{c}$
  end for
end for

Remarks on Training Methods

- **Online Training:** Updates are computed on a per-sample basis, which can result in a noisy gradient estimate.
- **Mini-Batch Training:** Gradients are averaged over a batch of samples, reducing variance and leading to more stable updates.
- In practice, CD-1 (i.e., $k = 1$) often performs surprisingly well, making RBM training efficient.

Encoder Problem

- **Overview:** Boltzmann Machine's learning algorithm's communication between two visible layers (V_1 and V_2) through a hidden layer (H).
- **Structure:**
  - Two visible layers V_1 and V_2 (each with v units).
  - Hidden layer H with $h < v$.
  - Each visible layer is completely connected internally and each is completely connected to H, but H is not connected with each other
- **Task:** Transmit one-hot activations (one unit active per group) via H.
- **Learning Goal:** Minimize divergence G between clamped (input) and free-running distributions.

Learning Phases

- **Phase 1: Negative phase / Winner-Take-All Inhibition**
  - All weights set to zero, developing negative weights within V_1 and V_2

$$E = -\sum_{i<j} w_{ij} s_i s_j$$

  - Only one unit in each visible group is active at a time.
  - Reduces 2^8 patterns to 4×4 and then to 4 in 4-2-4 encoder
- **Phase 2: Positive phase**
  - Positive weight between some visible units and hidden units.
  - Every units have a strong connection with each other.

$$w_{V_1 \to H} \approx w_{H \to V_2}$$

  - Most of the codes have been used, but there may be some similarities
- **Phase 3: Conflict Resolution**
  - Weight between units with the same hidden code now becomes more negative
  - One unit gets the new code to avoid collision by adjusting the weights to hidden units

Experiments and Results

- **4-2-4 Encoder ($v = 4$, $h = 2$):**
  - Perfect binary codes (00, 01, 10, 11).
  - Median convergence: 110 cycles.
- **4-3-4 Encoder ($v = 4$, $h = 3$):**
  - Optimal Hamming spacing (e.g., 000, 011, 101, 110).
  - Median convergence: 270 cycles.
- **8-3-8 Encoder ($v = 8$, $h = 3$):**
  - Harder; often finds 7/8 codes (median: 210 cycles).
  - Full solution rare (median: 1570 cycles).
- **40-10-40 Encoder:**
  - Scalability test: 98.6% accuracy.
  - Codes spaced with Hamming distance ≥ 2.

Representation in Parallel Networks

- **Approaches:**
  - Local representations (one unit = one concept).
  - Distributed representations (pattern across units = concept).
- **Local Representations:**
  - Pros: Modular, easy to modify (e.g., add/remove connections).
  - Cons: Biologically implausible for large-scale systems.
- **Distributed Representations:**
  - Pros: Fault-tolerant, efficient use of resources.
  - Cons: Harder to design/modify (knowledge is diffuse).

Energy Landscapes and Learning

- Distributed representations = energy minima in the network.
- **Learning Algorithm:**
  - Shapes the energy landscape to match environmental constraints.
  - Example: Encoder problem creates efficient codes as low-energy states.
- Mathematical simplicity of Boltzmann distribution enables coherent weight updates.

Communication Between Modules

- **Challenge:** Transmitting concepts (e.g., "wormy apple") between systems.
- **Provided Solutions:**
  - Message-passing (like a computer bus): Sequential, requires shared code.
  - Dedicated hardware pathways: Biologically implausible for new concepts.
- **Boltzmann Machine Solution:**
  - Shared communication lines with emergent codes.
  - Combines efficiency of message-passing with biological plausibility.