# Boltzmann Learning Rule

CT-466 | Week 3 - Lecture 7

Instructor: Mehar Fatima Shaikh

# Learning Rules

- Learning rules define how a neural network updates its weights and biases during training. They are based on mathematical formulas or biologically inspired mechanisms. Each rule tries to minimize error and improve performance.

# Common Learning Rules:

- **Perceptron Learning Rule**
- **Competitive Learning Rule**
- **Boltzmann Learning Rule**
- **Hebbian Learning Rule**
- **Delta (Widrow-Hoff) Rule**

# Perceptron Learning Rule

- Perceptron learning is a type of supervised learning.
- The network compares the predicted output with the actual target.
- If there is a difference between the two, it adjusts the weights.
- This adjustment reduces the error and helps the perceptron learn the correct mapping.
- Formula:
- $\Delta w_{ij} = \eta(t_j - y_j)x_i$

# Competitive Learning Rule

- Competitive learning is an unsupervised learning rule.
- In this rule, neurons compete with each other to respond to the input.
- Only the winning neuron (the one with the strongest response) updates its weights.
- This process allows the network to perform clustering, such as in Self-Organizing Maps.

# Boltzmann Learning Rule

- Boltzmann learning is a stochastic (probability-based) learning algorithm.
- It is used in Boltzmann Machines (BMs) and Restricted Boltzmann Machines (RBMs).
- The method is inspired by statistical mechanics.
- It updates the network by assigning probabilities to neuron states based on energy values and temperature.

# Hebbian Learning Rule

- Hebbian learning is an unsupervised learning rule.
- It is often summarized as: “Cells that fire together, wire together.”
- The weight update depends on the correlation between the input and the output.
- If both input and output are active together, the connection between them becomes stronger.
- Formula:

- $\Delta w_{ij} = \eta \cdot x_i \cdot y_j$
  - where $x_i$ =input, $y_j$ =output, $\eta$= learning rate.

# Delta (Widrow-Hoff) Rule

- The Delta (Widrow-Hoff) learning rule is a supervised learning rule.
- It minimizes the squared error between the target and the predicted output.
- The adjustment is done using gradient descent.
- This rule is the foundation for the backpropagation algorithm used in training neural networks.
- Formula:
- $\Delta w_{ij} = \eta(t_j - y_j)f'(net_j)x_i$

# Comparison

| Rule | Type | Key Idea |
|---|---|---|
| Perceptron | Supervised | Correct mistakes by adjusting weights |
| Competitive | Unsupervised | Winner neuron adapts |
| Boltzmann | Probabilistic | Learn via energy minimization and probability |
| Hebbian | Unsupervised | Correlation strengthens weights |
| Delta | Supervised | Minimize squared error |

# Boltzmann Learning Rule

- Boltzmann learning is a stochastic (probability-based) learning rule. It is used in Boltzmann Machines (BMs) and Restricted Boltzmann Machines (RBMs).
- The idea comes from statistical mechanics, where systems prefer low-energy states.
- Every possible configuration of neuron states (ON/OFF) has an energy value.
- Lower-energy configurations are more stable and therefore have higher probability.

# Operating Conditions

Neurons in a Boltzmann machine can function in two modes:

- **Clamped Condition**: Neurons remain in a fixed state.
- **Free Running Condition**: Neurons operate freely and can take any possible state.

# Energy Function

- Each state of the network has an energy:
- $E(v, h) = -\sum_i b_i v_i - \sum_j c_j h_j - \sum_{i,j} v_i w_{ij} h_j$
- where $v_i$ =visible unit, $h_j$ =hidden unit, $w_{ij}$ =weight.

# Probability Distribution

- The probability of a state is determined by the Boltzmann distribution:
- $P(v, h) = \frac{e^{-E(v,h)}}{Z}$
- where $Z$ is the partition function.

# Boltzmann Factor

- The **Boltzmann Factor** expresses the relative probability of a system being in a state with a given energy $E$ at temperature $T$.
- **Formula**
- $e^{-\frac{E}{k_B T}}$
- Where:
- $E$ = energy of the state
- $k_B$ =Boltzmann constant ($1.38 \times 10^{-23}\ J/K($
- $T$ =absolute temperature in Kelvin

# Boltzmann Factor

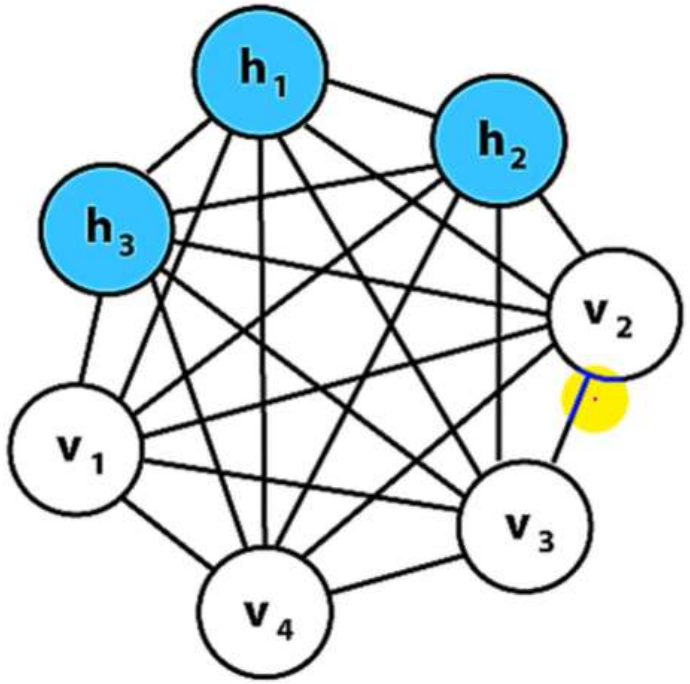
- The factor decreases **exponentially** with higher energy $E$.
- Lower-energy states are **exponentially more probable** than higher-energy states.
- Temperature $T$ controls randomness:
  - At **high** $T$, even high-energy states have significant probability.
  - At **low** $T$, the system strongly favors the lowest-energy states.

# Relation to Boltzmann Distribution

- The **Boltzmann distribution** normalizes the Boltzmann Factor across all possible states:
- $P(E_i) = \frac{e^{-E_i/k_B T}}{\sum_j e^{-E_j/k_B T}}$
- Here:
- Numerator = Boltzmann factor for state $i$
- Denominator = partition function (ensures total probability = 1)

# Boltzmann Distribution

*Boltzmann Machine or Boltzmann distribution*



# Boltzmann Distribution

- The Boltzmann distribution is a probability distribution from statistical mechanics that describes the likelihood of a system being in a particular state based on its energy and temperature.
- It is the foundation of Boltzmann Learning in neural networks.

# Boltzmann Distribution

- $P(s) = \frac{e^{-\frac{E(s)}{T}}}{Z}$
- Where:
- $P(s)$ =probability of the system (or neuron state) $s$
- $E(s)$ =energy of state $s$
- $T$= temperature (controls randomness; higher $T \rightarrow$ more randomness)
- $Z = \sum_{s} e^{-\frac{E(s)}{T}}$ =**partition function**, ensures probabilities sum to 1

# Boltzmann Distribution

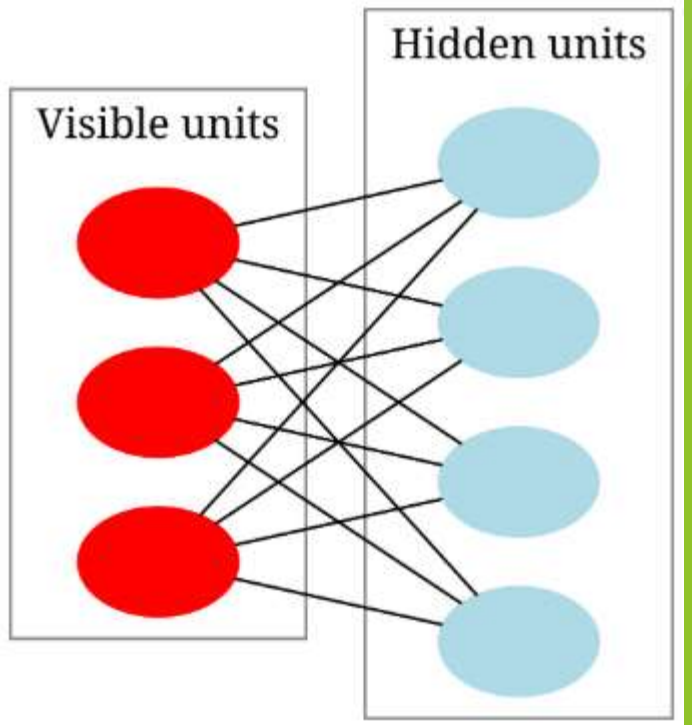
- Lower energy states are more likely
  - The smaller the $E(s)$E(s), the higher the probability.
  - This means the system tends to settle in minimum-energy configurations.
- Temperature controls randomness
  - At high $T \rightarrow$ system explores many states (more random).
  - At low $T \rightarrow$ system converges to minimum-energy states (more stable).
- In Boltzmann Machines
  - The probability of a neuron being ON or OFF depends on this distribution.
  - This is why the network is called a stochastic neural network.

# Example

- If we have two states:
- State A with energy $E = 1$
- State B with energy $E = 3$
- At low temperature $T$, **State A** will have much higher probability than State B.

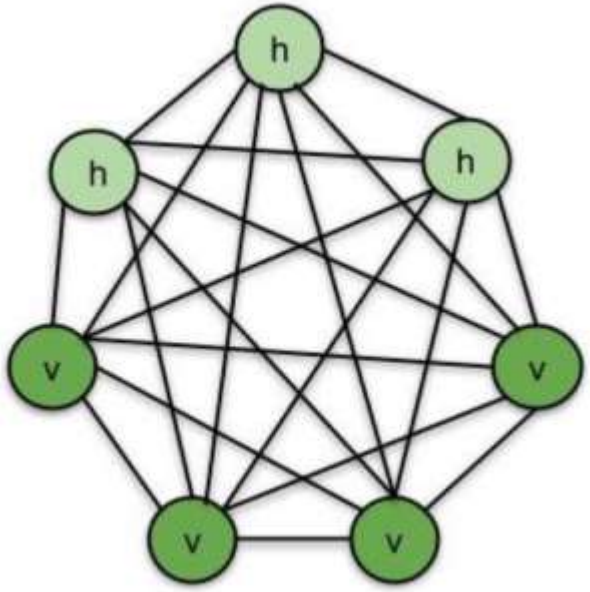
# In Neural Networks (Boltzmann Machines):

- Each neuron’s state probability is determined using the Boltzmann factor.
- This explains why the network is stochastic → neurons flip ON/OFF with probabilities derived from energy differences.



# Hopfield Network

- A **Hopfield Network** is a type of **recurrent neural network (RNN)** invented by John Hopfield in 1982.
  It is mainly used as an **associative memory system** – meaning it can store patterns and later retrieve them, even from incomplete or noisy input.



# Example

- We consider a very small Boltzmann Machine with two neurons, $s_1$ and $s_2$ .Each neuron can take the value +1 (ON) or –1 (OFF). The connection weight between them is symmetric: $w_{12} = w_{21} = 1.0$ .Both biases are zero, and the temperature is set to $T = 1$.
- The energy function is:
- $E(s_1, s_2) = -w_{12} \cdot s_1 \cdot s_2$

# Step 1: Possible states

- Since there are two neurons, there are $2^2 = 4$possible states:
- $(s_1, = +1 s_2 = +1)$
- $(s_1, = +1 s_2 = -1)$
- $(s_1, = -1 s_2 = +1)$
- $(s_1, = -1 s_2 = -1)$

# Step 2: Energy of each state

- Using $E=-w \cdot s\_1\ s\_2$:
- (+1, +1): $E=-1$
- (+1, –1): $E=+1$
- (–1, +1): $E=+1$
- (–1, –1): $E=-1$

# Step 3: Boltzmann factor

- $e^{-E/T}$
- (+1, +1): $e$^(−(−1) )=$e$^1≈2.718
- (+1, –1): $e$^(−(+1) )=$e$^(−1)≈0.367
- (–1, +1): $e$^(−(+1) )=0.367
- (–1, –1): $e$^(−(−1) )=$e$^1=2.718

# Step 4: Partition function

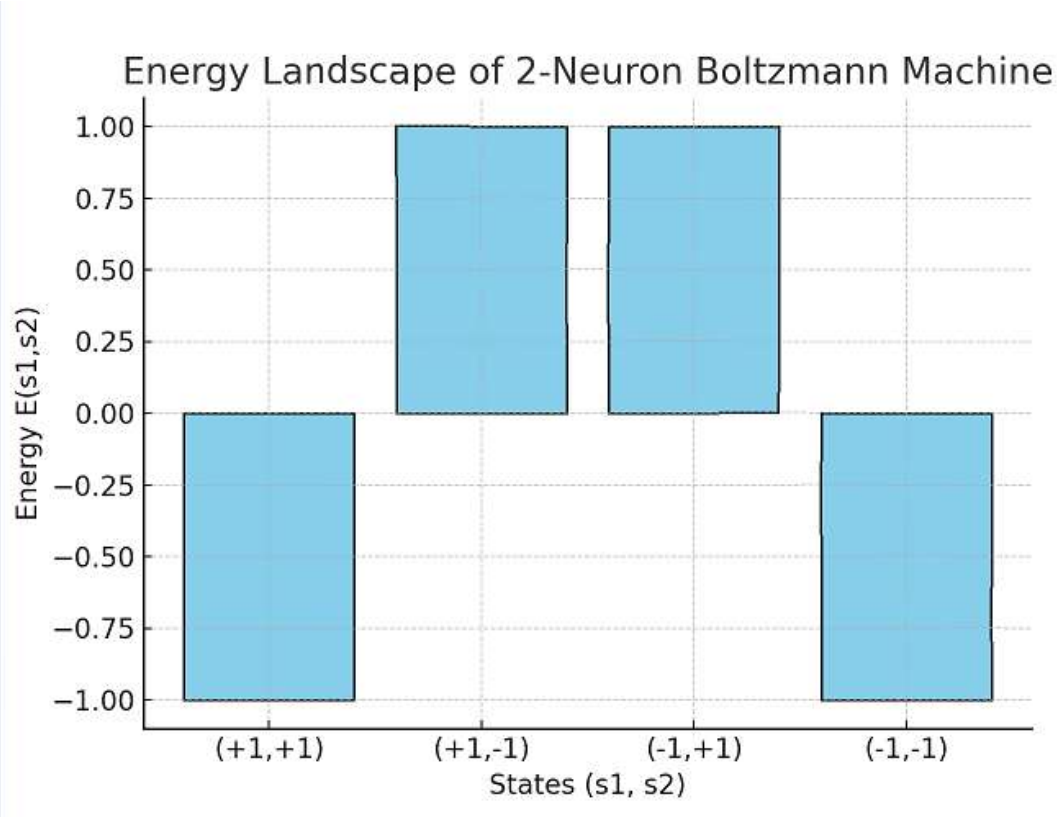
- $Z = \sum e^{-E/T} = 2.718 + 0.367 + 0.367 + 2.718 = 6.17$
- **Step 5: Probabilities**
- $P(state) = \frac{e^{-E/T}}{Z}$

- (+1, +1): 2.718/6.17≈0.44
- (+1, –1): 0.367/6.17≈0.06
- (–1, +1): 0.367/6.17≈0.06
- (–1, –1): 2.718/6.17≈0.44

# Result

- The system spends most of its time in the states (+1, +1) and (–1, –1), each with about 44% probability. The mixed states (+1, –1) and (–1, +1) are much less likely, only about 6% each. This means the network prefers both neurons to be ON together or OFF together, because the weight between them is positive.

# Diagram


Energy Landscape of 2-Neuron Boltzmann Machine
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Energy E(s1,s2)
(+1,+1)
(+1,-1)
(-1,+1)
(-1,-1)
States (s1, s2)

# Learning Rule

- Weight updates aim to reduce the difference between data distribution and model distribution:
- $\Delta w_{ij} = \eta \left( \langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model} \right)$
  - $\langle \cdot \rangle_{data}$ :expectation under training data.
  - $\langle \cdot \rangle_{model}$ :expectation under the model.

# Characteristics

- Uses Gibbs sampling to approximate expectations.
- Slow but powerful; later improved in RBMs with Contrastive Divergence (CD).

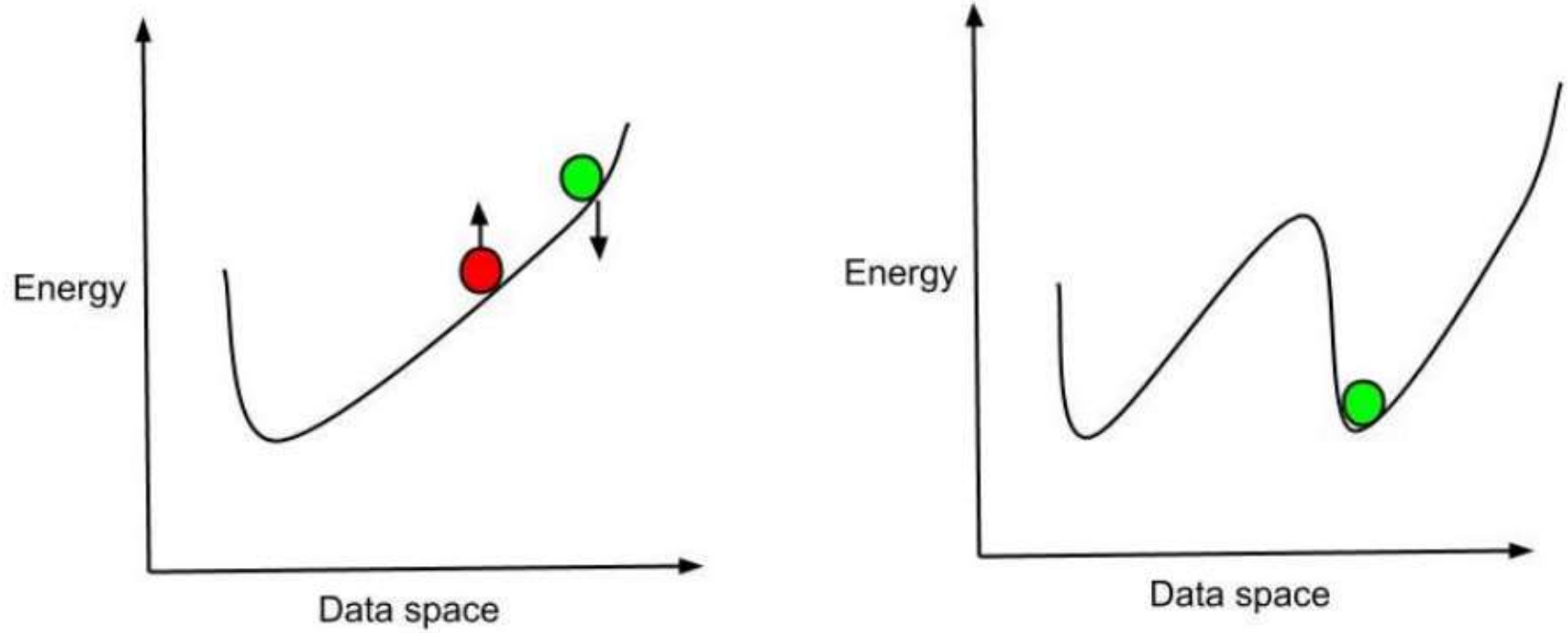
# What is a Boltzmann Machine?

- A Boltzmann Machine (BM) is a type of stochastic (probabilistic) neural network that learns to represent complex data distributions. It is based on concepts from statistical mechanics and the Boltzmann distribution.
- A Boltzmann Machine is a network of symmetrically connected, neuron-like units that make stochastic decisions about whether to be in the ON or OFF state.
- Boltzmann Machines use a simple learning algorithm that enables them to discover meaningful features in datasets composed of binary vectors.

# Working of Boltzmann machine

- A neural network is made up of input and output layers of neurons, along with multiple hidden layers.
- The neurons function in a binary way, meaning they can be in an ON state (represented by +1) or an OFF state (represented by –1).
- The network works by randomly selecting a neuron and flipping its state.

# Boltzmann Machines (BMs)



# Energy Function of a Boltzmann Machine

- A **Boltzmann machine** is defined by an **energy function**:
- $E = -\frac{1}{2}\sum_j \sum_k w_{jk} x_j x_k$
- where $j \neq k$
- $x_k$ represents the **state of neuron** $k$.

# Probability of State Change

- The **probability** that a neuron changes its state, $P$, is defined as:
- $P = \frac{1}{1+\exp\left(-\frac{\Delta E_k}{T}\right)}$
- where:
- $\Delta E_k$ =change in energy when neuron $k$flips its state
- $T$= temperature parameter (controls randomness in state changes)

# Types of Boltzmann Machines

- Restricted Boltzmann Machines (RBMs)
- Deep Belief Networks (DBNs)
- Deep Boltzmann Machines (DBMs)

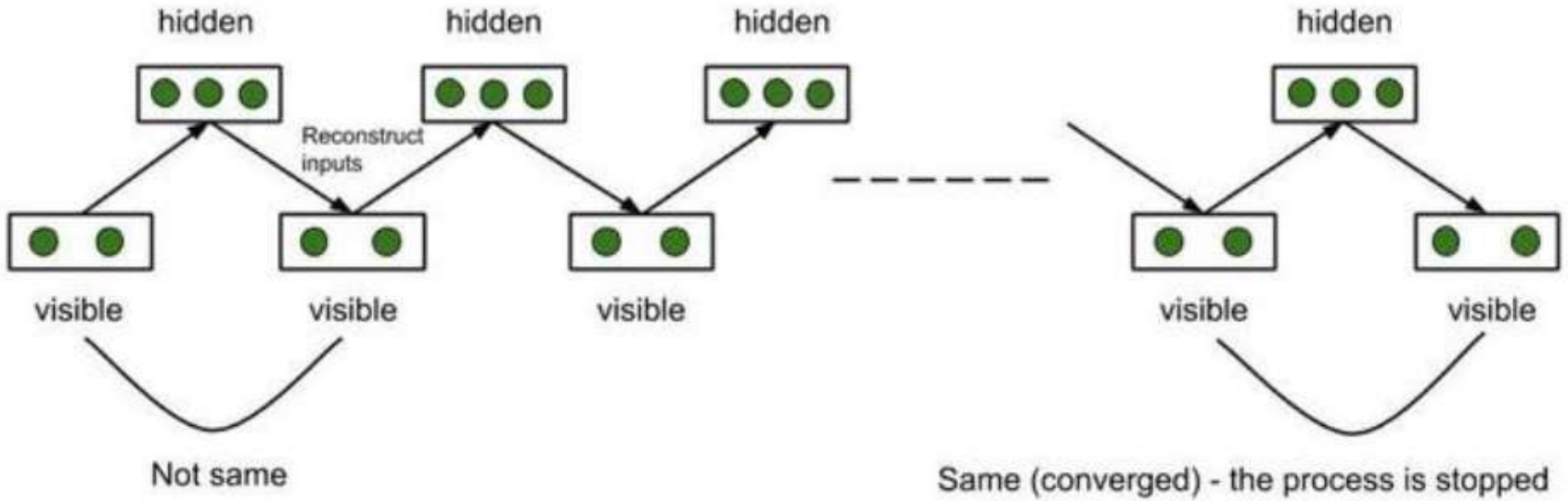
# Restricted Boltzmann Machines (RBMs):

- In a full Boltzmann machine, each node is connected to every other node and hence the connections grow **exponentially**. This is the reason we use RBMs. The restrictions in the node connections in RBMs are as follows -
- Hidden nodes cannot be connected to one another.
- Visible nodes connected to one another.

# Contrastive Divergence:

- **RBM** adjusts its weights by this method. Using some randomly assigned initial weights, RBM calculates the hidden nodes, which in turn use the same weights to reconstruct the input nodes. Each hidden node is constructed from all the visible nodes and each visible node is reconstructed from all the hidden node and hence, the input is different from the reconstructed input, though the weights are the same. The process continues until the reconstructed input matches the previous input. The process is said to be converged at this stage. This entire procedure is known as **Gibbs Sampling**.

# Contrastive Divergence:



# Gibbs Sampling in RBMs

- Gibbs sampling is a **Markov Chain Monte Carlo (MCMC)** method used to approximate probability distributions.
  In RBMs, it is used to sample neuron states since exact computation of expectations is **intractable**.
- **Process in RBMs:**
- Because RBMs have **no intra-layer connections**, conditional probabilities are simple:
  - For hidden unit:
- $P(h_j = 1 \mid v) = \sigma(b_j + \sum_i v_i w_{ij})$
  - For visible unit:
- $P(v_i = 1 \mid h) = \sigma(a_i + \sum_j h_j w_{ij})$
- where $\sigma(x)$is the sigmoid function.

# Gibbs sampling procedure:

- Start with input data $v^{(0)}$)clamped visible layer).
- Sample hidden layer: $h^{(0)} \sim P\left(h \mid v^{(0)}\right)$.
- Sample reconstructed visible: $v^{(1)} \sim P\left(v \mid h^{(0)}\right)$.
- Repeat for several steps to get samples closer to the **model distribution**.

- This process generates samples from the RBM's probability distribution.

# Contrastive Divergence (CD)

- Training RBMs requires minimizing the difference between the **data distribution** and the **model distribution**.
  The Boltzmann learning rule:
- $\Delta w_{ij} = \eta(\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model})$
- $\langle \cdot \rangle_{data}$ →expectation under the training data.
- $\langle \cdot \rangle_{model}$ →expectation under the model distribution (requires Gibbs sampling).
- Problem: Running Gibbs sampling to equilibrium is very slow.

# Solution → Contrastive Divergence (CD-k)

- Proposed by Geoffrey Hinton, CD is an **approximate method**:
- Clamp the data to the visible layer $v^{(0)}$.
- Run **k steps of Gibbs sampling** (usually $k = 1$is enough).
- Use the difference between positive (data-driven) and negative (reconstruction-driven) correlations to update weights.
- $\Delta w_{ij} = \eta(v^{(0)}h^{(0)} - v^{(k)}h^{(k)})$
- Where:
- $v^{(0)}, h^{(0)}$ =data-driven states
- $v^{(k)}, h^{(k)}$ =k-step reconstruction states

# Comparison of Gibbs Sampling and Contrastive Divergence

| Aspect | Gibbs Sampling | Contrastive Divergence |
|---|---|---|
| Purpose | Generate samples from RBM distribution | Approximate gradient for faster training |
| Process | Long Markov Chain (many iterations until equilibrium) | Short chain (1–5 steps, usually CD-1) |
| Speed | Slow, computationally expensive | Very fast |
| Use Case | Exact sampling, theoretical analysis | Practical RBM training |

# Deep Belief Networks (DBNs):

- Suppose we stack several RBMs on top of each other so that the first RBM outputs are the input to the second RBM and so on. Such networks are known as Deep Belief Networks. The connections within each layer are undirected (since each layer is an RBM). Simultaneously, those in between the layers are directed (except the top two layers - the connection between the top two layers is undirected). There are two ways to train the DBNs-
- **Greedy Layer-wise Training Algorithm -** The RBMs are trained layer by layer. Once the individual RBMs are trained (that is, the parameters - weights, biases are set), the direction is set up between the DBN layers.
- **Wake-Sleep Algorithm -** The DBN is trained all the way up (connections going up - wake) and then down the network (connections going down — sleep).

# Deep Boltzmann Machines (DBMs):

- DBMs are similar to DBNs except that apart from the connections within layers, the connections between the layers are also **undirected** (unlike DBN in which the connections between layers are directed). DBMs can extract more complex or sophisticated features and hence can be used for more complex tasks.

# Applications

- Feature learning
- Pattern recognition
- Dimensionality reduction
- Basis of **Restricted Boltzmann Machines (RBMs)**, which are widely used in:
  - Deep Belief Networks (DBNs)
  - Collaborative filtering (e.g., Netflix recommendation system)
  - Pretraining deep neural networks