Chapter 7
Other Important NN Models

- **Continuous Hopfield mode** (in detail)
  - For combinatorial optimization
- **Simulated annealing** (in detail)
  - Escape from local minimum
- Baltzmann machine (brief)
- Other models (brief)
  - Reinforcement learning (between supervised and unsupervised learning)
  - Probabilistic neural networks
  - Recurrent BP networks
  - Networks of radial basis functions

Continuous Hopfield Model

- Architecture:
  - Fully connected with symmetric weights $w_{ij} = w_{ji}, w_{ii} = 0$
  - Activation u_i : with $\frac{du_i}{dt} = \sum_{j=1}^{n} w_{ij} v_j + q_i = in_i$
  - Output (state) $v_i = g(u_i)$ where g is a sigmoid function to ensure binary/bipolar output
  - Compute $u_i(t)$ by first-order Taylor expansion

$$u_i(t+d) = \int in_i(t)dt \approx u_i(t) + \frac{du_i(t)}{dt} \cdot d + \ldots = u_i(t) + in_i \cdot d$$

- Computation: all units change their output (states) at the same time, based on states of all others.

- Convergence:
  - define an energy function,
  - show that if the state update rule is followed, the system's energy always decreasing.

$$E = -\frac{1}{2}\sum_{ij} v_i w_{ij} v_j - \sum_i q_i v_i$$

$$\frac{dE}{dt} = \sum_i \frac{\partial E}{\partial v_i} \cdot \frac{dv_i}{du_i} \cdot \frac{du_i}{dt} \quad \text{(chain rule)}$$

$$\frac{\partial E}{\partial v_i} = -\sum_{j \neq i} w_{ij} v_j - q_i = -in_i$$

$$\frac{dv_i}{du_i} = g'(u_i) > 0,$$

$$\frac{du_i}{dt} = in_i = -\frac{\partial E}{\partial v_i}$$

$$\frac{dE}{dt} = -\sum_i g'(u_i)(in_i)^2 \leq 0$$

- $\dot{E}$ asymptotically approaches zero when $g'(u_i)$ approaches 1 or 0 for all i.
- The system reaches a local minimum energy state
- Gradient descent: $\frac{du_i}{dt} = -\frac{\partial E}{\partial v_i}$
- Instead of jumping from corner to corner in a hypercube as the discrete HM does, the system of continuous HM moves in the interior of the hypercube along the gradient descent trajectory of the energy function to a local minimum energy state.

H model for optimization (TSP)

- Constraint satisfaction combinational optimization.

  A solution must satisfy a set of given constraints (strong) and be optimal w.r.t. a cost or utility function (weak)
- TSP: (geometric vs geographic)
  - n points(cities) on a plane form a fully connected graph (can go from any city to any other),
  - direct travel distance between 2 cities = length of the edge between them = Euclidean distance between them.
  - Objective: find a optional tour which, starting from a city, visits every city exactly once, then returns to the start city.
  - Those (legal) tours are Hamiltonian circuits of the graph.
  - The optimality is measured by the length of the circuit = sum of lengths of all legs on the circuit.

- Constraints:
  1. Each city can be visited no more than once
  2. Every city must be visited
  3. TS can only visit cities one at a time (implicit)
  4. Tour (circuit) should be the shortest
- NP-hard problem (harder than its NP-complete version)
  - A legal tour (satisfying constraints 1 – 3) can be represented by a permutation of all n cities (because the graph is fully connected). Total # of permutations: n!
  - Each circuit corresponds to n different permutations (with n different starting city), total # of undirected circuits: n!/n
  - Each circuit has two directions, total # of distinct circuits: n!/2n

    need to find the shortest among the n!/2n circuits.
  - Grows faster than exponential.

$$\frac{(n+1)!}{2(n+1)} \Big/ \frac{n!}{2n} = \frac{(n+1)!}{n!} \cdot \frac{2n}{2(n+1)} = n \text{ but } \frac{2^{n+1}}{2^n} = 2$$

- Solving TSP by continuous Hopfield model:
  1. Design the network structure
  2. Define an energy function
     - punish violation of (strong) constraint with large amount of energy
     - lower energy associates to shorter circuits (weak constraint)
  3. Find weight matrix from the energy function such that energy always decreases whenever the network moves (according to H model and $\boldsymbol{W}$)
- Design the network structure:

  Different possible ways to represent TSP by NN:

  node - city: hard to represent the order of cities in forming a circuit (SOM solution)

  node - edge: n out of n(n-1)/2 nodes must become activated and they must form a circuit.

- **Hopfield's solution**:
  - n by n network, each node is connected to every other node.
    - Node output approach 0 or 1
    - row: city
    - column: position in the tour:

      Tour: B-A-E-C-D-B

	1	2	3	4	5
A	0	1	0	0	0
B	1	0	0	0	0
C	0	0	0	1	0
D	0	0	0	0	1
E	0	0	1	0	0

  - Output (state) of each node is denoted:

    v_{xi} where x : city index

    i : position index

    $v_{xi} = 1$: city x is visited as the i^{th} city in a tour

    $v_{xi} = 0$: city x is not visited as the i^{th} city in a tour

    d_{xy} = distance between cities x and y

Energy function

$$E = \frac{A}{2}\sum_{x=1}^{n}\sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n} v_{xi}v_{xj}$$ (penalty for the row constraint: no city shall be visited more than once)

$$+\frac{B}{2}\sum_{i=1}^{n}\sum_{x=1}^{n}\sum_{y=1,y\neq x}^{n} v_{xi}v_{yi}$$ (penalty for the column constraint: cities can be visited one at a time)

$$+\frac{C}{2}\left(\sum_{x=1}^{n}\sum_{i=1}^{n} v_{xi} - n\right)^2$$ (penalty for the tour legs: it must have exactly n cities)

$$+\frac{D}{2}\sum_{x=1}^{n}\sum_{y=1,y\neq x}^{n}\sum_{i=1}^{n} d_{xy}v_{xi}(v_{y,i+1}+v_{y,i-1})$$ (penalty for the tour length)

- A, B, C, D are constants, to be determined by trial-and-error.
- If all v_{xi} approach either 0 or 1, and if those with $v_{xi} \approx 1$ represent a Hamiltonian circuit, then

$$d_{xy}v_{xi}(v_{y,i+1}+v_{y,i-1}) = \begin{cases} d_{xy} & \text{if city } y \text{ is either before or after city } x \text{ in the circuit} \\ 0 & \text{otherwise} \end{cases}$$

the fourth term represents the circuit length here ***i+1***, ***i-1*** are modulo n. With the 5 city example B-A-E-C-D-B.

$$\overbrace{d_{AE}+d_{AB}}^{v_{A2}}+\overbrace{d_{BA}+d_{BD}}^{v_{B1}}+\overbrace{d_{CD}+d_{CE}}^{v_{C4}}+\overbrace{d_{DB}+d_{DC}}^{v_{D5}}+\overbrace{d_{EC}+d_{EA}}^{v_{E3}}$$
$$=2(d_{AE}+d_{AB}+d_{BD}+d_{CD}+d_{CE})$$

- Obtaining weight matrix:

  Method 1: Rewrite ***E*** in the form of

$$E=-\frac{1}{2}\sum_{xi}^{n}\sum_{yj\neq xi}^{n}w_{xi,yj}\cdot v_{xi}\cdot v_{yj}-\sum_{i}q_{xi}\cdot v_{xi}$$

  where $w_{xi,yj}, q_{xi}$ are in terms of parameters ***A, B, C, D***.

  no systematic procedure for such conversion

Method 2: Determine local function of motion $\dot{\boldsymbol{u}}_{\boldsymbol{xi}}$ from $\boldsymbol{E}$

- it should cause $\boldsymbol{E}$ to always decrease
- since by continuous HM, $\dot{\boldsymbol{u}}_{xi} = \sum_{yj \neq xi} \boldsymbol{w}_{x_i, y_j} \cdot \boldsymbol{v}_{y_j} + q_{x_i}$

  it might be easier to find $\boldsymbol{w}_{\boldsymbol{xi},\boldsymbol{yj}}, q_{\boldsymbol{xi}}$ from $\dot{\boldsymbol{u}}_{\boldsymbol{xi}}$

(1) determining $\dot{u}_{xi}(t)$ from E so that with $\dot{u}_{xi}(t), \dot{E}(t) < 0$

(gradient descent approach again)

$$\frac{dE}{dt} = \sum_{xi} \frac{\partial E}{\partial v_{xi}} \cdot \frac{dv_{xi}}{du_{xi}} \cdot \frac{du_{xi}}{dt}$$

if $\dot{u}_{xi} = -\frac{\partial E}{\partial v_{xi}}$, then $\frac{dE}{dt} = -\sum \frac{dv_{xi}}{du_{xi}} \cdot (\frac{\partial E}{\partial v_{xi}})^2 \le 0$

$$\dot{u}_{xi} = -\frac{\partial E}{\partial v_{xi}}$$

$$= -A\sum_{j \neq i} v_{xj}$$ (row inhibition: x = y, i != j)

$$- B\sum_{y \neq x} v_{yi}$$ (column inhibition: x != y, i = j)

$$- C(\sum_{y}\sum_{j} v_{xj} - n)$$ (global inhibition: x != y, i = j)

$$- D\sum_{y \neq x} d_{xy}(v_{y,i+1} + v_{y,i-1})$$ (tour length)

(2) Since $\dot{u}_{x1} = in_{ki} = \sum_{yj \neq xi} w_{xi,yj} \cdot v_{yj} + q_{ki}$, weights thus should include the following

- A: between nodes in the same row
- B: between nodes in the same column
- C: between any two nodes
- D: dxy between nodes in different row but adjacent column

$$w_{xi,yj} = -A\,d_{xy}(1 - d_{ij}) - B\,d_{ij}(1 - d_{xy}) - C - Dd_{xy}(d_{j,i+1} + d_{j,i-1})$$

where $d_{xy} = \begin{cases} = 1 & \text{if } x = y \\ = 0 & \text{otherwise} \end{cases}$

- each node also has a positive bias $q_{xi} = Cn$

Notes

1. Since $w_{xi,yj} = w_{yj,xi}$, (assuming $w_{xi,xi} = 0$) , W can also be used for discrete model.
2. Initialization: randomly assign $\dot{u}_{xi}(0)$ between 0 and 1 such that $\sum_{x,i} v_{xi}(0) = n$
3. No need to store explicit weight matrix.

4. Hopfield's own experiments

   A = B = D = 500, C = 200, n = 15

   20 trials (with different distance matrices): all trials converged:

   16 to legal tours, 8 to shortest tours, 2 to second shortest tours.

5. General methodology of using Hopfield model for combinatorial optimization.
   - represent the problem: solution space to state space
   - sum-up problem constraints and cost functions and other considerations into an energy function E:
     - E is defined over the state space
     - lower energy states correspond to better solutions
     - penalty for constraint violation
   - determine the local function of motion (gradient descent)

     $$\dot{\boldsymbol{u}}_i = -\partial \boldsymbol{E} / \partial \boldsymbol{v}_i$$

   - work out weight matrix from $\dot{\boldsymbol{u}}_i$
   - The hard part is to establish that
     - Any solution (to the given problem) corresponds to a (local) minimum energy state of the system
     - Optimal solution corresponds to a globally minimum energy state of the system.

6. Problems of continuous HM for optimization

- Only guarantees **local** minimum state (***E*** always decreasing)
- No general guiding principles for determining parameters (e.g., A, B, C, D in TSP)
- Energy functions are hard to come up and they may result in different solution qualities

$$E = \frac{A}{2}\sum_x \sum_i \sum_{j \neq i} v_{xi} v_{xj} + \frac{B}{2}\sum_i \sum_x \sum_{y \neq x} v_{xi} v_{xi} +$$
$$\frac{C}{2}\left\{\sum_x (1 - \sum_i v_{xi})^2 + \sum_i (1 - \sum_x v_{xi})^2\right\}$$
$$\frac{D}{2}\sum_x \sum_i \sum_{y \neq x} d_{xy} v_{xi} (v_{y,i+1} + v_{y,i-1})$$

another energy function for TSP

Simulated Annealing

- A general purpose global optimization technique
- Motivation

  BP/HM:

  – Gradient descent to minimal error/energy function E.

  – Iterative improvement: each step improves the solution.

  – As optimization: stops when no improvement is possible without making it worse first.

  – Problem: trapped to local minimal .

  key: $\Delta \boldsymbol{E} = \boldsymbol{E}(\boldsymbol{t}+1) - \boldsymbol{E}(\boldsymbol{t}) \leq 0 \quad \forall \boldsymbol{t}$

  – possible solution to escaping from local minimal:

  allow $\boldsymbol{E}$ to increase occasionally (by adding random noise).

Annealing process in metallurgy

- To improve quality of metal works.
- Energy of a state (a config. of atoms in a metal piece)
  - depends on the relative locations between atoms.
  - minimum energy state: crystal lattice, durable, less fragile/crisp
  - many atoms are dislocated from crystal lattice, causing higher (internal) energy.
- Each atom is able to randomly move
  - If and how far an atom moves depends on the temperature (T)
- Dislocation and other disruptions can be eliminated by the atom's random moves: **thermal agitation**.
  - takes too long if done at room temperature
- Annealing: (to shorten the agitation time)
  - **starting at a very high T, gradually reduce T**
- SA: apply the idea of annealing to NN optimization

Statistical Mechanics

- System of multi-particles,
  - Each particle can change its state
  - Hard to know the system's exact state/config. , and its energy.
  - Statistical approach: probability of the system is at a given state.

    assume all possible states obey Boltzmann-Gibbs distribution:

    E_a : the energy when the system is at state a

    P_a : the probability the system is at state a

$P_a = \frac{1}{z} e^{-b E_a}$, where $z = \sum_a e^{-b E_a}$ is the normalization factor so $\sum_r P_a = 1$

$b = (K_B T)^{-1}$, where K_B : Boltzmann constant, T : absolute temperature

Ingnoring K_B and using T for artificial temperature

$P_a = \frac{1}{z} e^{-E_a/T}$, and $\frac{P_a}{P_b} = \frac{e^{-E_a/T}}{e^{-E_b/T}} = e^{-(E_a - E_b)/T} = e^{-\Delta E/T}$

Let $E_a < E_b \quad \Delta E < 0$

(1) $P_a / P_b > 1 \rightarrow P_a > P_b$

(2) P_a and P_b differ little with high T, more opportunity to change state in the beginning of annealing.

P_a and P_b differ a lot with low T, help to keep the system at low E state at the end of annealing.

when T→0, $P_a / P_b \rightarrow \infty$ (system is infinitely more likely to be in the global minimum energy state than in any other state).

(3) Based on B-G distribution

$$P(S_a)P(S_a \rightarrow S_b) = P(S_b)P(S_b \rightarrow S_a)$$

$$\frac{P(S_a \rightarrow S_b)}{P(S_b \rightarrow S_a)} = \frac{P(S_b)}{P(S_a)} = e^{-(E_b - E_a)/T}$$

- Metropolis algorithm for optimization(1953)

  $\boldsymbol{S}_a$: current state

  a new state $\boldsymbol{S}_b$ differs from $\boldsymbol{S}_a$ by a small random displacement, then accept $\boldsymbol{S}_b$ **will be accepted** with the probability

$$\boldsymbol{P}(\boldsymbol{s}_a \rightarrow \boldsymbol{s}_b) = \begin{cases} 1 & \text{if } (\boldsymbol{E}_b - \boldsymbol{E}_a) < 0 \\ \boldsymbol{e}^{-(\boldsymbol{E}_b - \boldsymbol{E}_a)/\boldsymbol{T}} & \text{otherwise} \end{cases}$$

Random noise introduced here

1. the system is allowed to move to state b if it reduces $\boldsymbol{E}$
2. the system is allowed to occasionally move to state b (with probability $\boldsymbol{e}^{-\Delta \boldsymbol{E}/\boldsymbol{T}}$) even it increases $\boldsymbol{E}$,

Simulated Annealing in NN

- Algorithm (very close to Metropolis algorithm)
  1. set the network to an initial state S

     set the initial temperature T>>1
  2. do the following steps many times until thermal equilibrium is reached at the current T

     2.1. randomly select a state displacement $\Delta \boldsymbol{s}$

     2.2. compute $\Delta \boldsymbol{E} = \boldsymbol{E}(\boldsymbol{s} + \Delta \boldsymbol{s}) - \boldsymbol{E}(\boldsymbol{s})$

     2.3. if $\Delta \boldsymbol{E} < 0$ then $\boldsymbol{s} := \boldsymbol{s} + \Delta \boldsymbol{s}$

     else generate a random number $\boldsymbol{p}$ between 0 and 1

     if $\boldsymbol{p} \le \boldsymbol{e}^{-\Delta \boldsymbol{E}/\boldsymbol{T}}$ then $\boldsymbol{s} := \boldsymbol{s} + \Delta \boldsymbol{s}$
  3. reduce T according to the coding schedule
  4. if T > T-lower-bound, then go to 2 else stop

- Comments
  - thermal equilibrium (step 2) is hard to test, usually with a pre-set iteration number/time
  - displacement $\Delta \boldsymbol{S}$ may be randomly generated
    - choose one component of $\boldsymbol{S}$ to change or
    - changes to all components of the entire state vector
    - $\Delta \boldsymbol{S}$ should be small
  - cooling schedule
    - Initial T: 1/T ~ 0 (so any state change can be accepted)
    - Simple example: $\boldsymbol{T} := \boldsymbol{T} \cdot b$ where $0 < b << 1$
    - Another example:

$$\boldsymbol{T}(\boldsymbol{k}) = \frac{\boldsymbol{t}(0)}{\log(1+\boldsymbol{k})} \quad \text{non-linear, slower at the end}$$

SA for discrete Hopfield Model

- In step 2, each time only one node say xi is selected for possible update, all other nodes are fixed.

E_a : energy with $x_i = 1$; E_b : energy with $x_i = 0$

P_i : the probability to set $x_i = 1$;

$$P_i = \frac{e^{-E_a/T}}{e^{-E_a/T} + e^{-E_b/T}} = \frac{1}{1 + e^{-(E_b - E_a)/T}}$$

This acceptance criterion is difference from Metropolis alg.

if $\Delta E_i = E_b - E_a > 0$ ($x_i = 1$ is better) $P_i > 0.5$

if $\Delta E_i < 0$ ($x_i = 0$ is better) $P_i < 0.5$

when T is small $P_i \approx \begin{cases} 1 & \text{if } \Delta E_i > 0 \\ e^{\Delta E_i / T} & \text{if } \Delta E_i < 0 \end{cases}$

- Localize the computation

$$E = -\frac{1}{2}\sum_{j=1}^{n}\sum_{j\neq i} w_{ij} x_i x_j - q_{\ k} x_k$$

$$\Delta E_k = (E_{x_k=0} - E_{x_k=1}) = \sum w_{kj} x_j + q_{\ k} = \ln_k$$

$$P_k = \frac{1}{1+e^{-\ln k/T}}$$ no need to compute E_a and E_b

- It can be shown that both acceptance criterion guarantees the B-G distribution if a thermal equilibrium is reached.
- When applying to TSP, using the energy function designed for continuous HM

Variations of SA

- Gauss machine: a more general framework

$$\ln_k = \sum \boldsymbol{w}_{kj} \boldsymbol{x}_j + q_k + e$$

where e is the random noise

if $e = 0$ we have ***HM***

if e obeys Gaussian distritution with ***mean*** $= 0$ and ***SD*** $= \boldsymbol{T}\sqrt{8/p}$

very close to SA with $(1+\boldsymbol{e}^{-\Delta \boldsymbol{E}/\boldsymbol{T}})^{-1}$ as acceptance criterian

if e obeys logistic distritution with ***mean*** $= 0$ and ***SD*** $= \boldsymbol{T}\sqrt{8/p}$

exact SA with $(1+\boldsymbol{e}^{-\Delta \boldsymbol{E}/\boldsymbol{T}})^{-1}$

- Cauchy machine

  e obeys Cauchy distribution

  acceptance criteria: $\frac{1}{2}+\frac{1}{p}\arctan(\frac{-\Delta E}{T})$

  faster cooling : $T(k)=\frac{T_0}{K}$

- Need special random number generator for a particular distribution

  Gauss: $P(x)=\frac{1}{\sqrt{2p}}e^{-\frac{x^2}{2}}$ density function

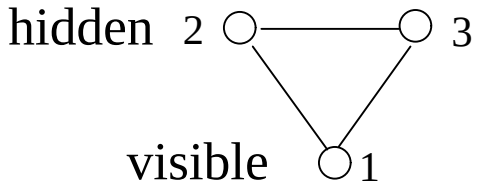
Boltzmann Machine (BM)

- **Hopfield model + hidden units + simulated annealing**
- **BM Architecture**
  - a set of visible units: nodes can be accessed from outside
  - a set of hidden units: (may be empty)
    - adding hidden nodes to increase the computing power
  - connection between nodes
    - Fully connected between any two nodes (not layered)
    - Symmetric connection: $w_{ij} = w_{ji} \quad w_{ii} = 0$
  - nodes are the same as in discrete HM $net_i = \sum_{j \neq i} w_{ij} x_j + q_i$
  - energy function: $E = -\frac{1}{2} \sum_i \sum_{j \neq i} w_{ij} x_i x_j - \sum_i q_i x_i$

- **BM computing** (SA), with a given set of weights

  1. Apply an input pattern to the visible units.
     – some components may be missing---pattern completion;
     – some components may be permanently clamped to the input values (as recall key or problem input parameters).
  2. Assign randomly 0/1 to all unknown units
     (including all hidden units and visible units with missing input values).
  3. Perform SA process according to a given cooling schedule. Specifically, at any given temperature T. an random picked non-clamped unit i is assigned value of 1 with probability $(1+e^{-net_i/T})^{-1}$, and 0 with probability $1-(1+e^{-net_i/T})^{-1}$

- **BM learning** (obtaining weights from examplers)
  - what is to be learned?
    - **probability distribution** of **visible** vectors in the environment.
    - examplers: randomly drawn from the entire population of possible visible vectors.
    - construct a model of the environment that has the same prob. distri. of visible nodes as the one in the exampler set.
  - There may be many models satisfying this condition
    - because the model involves hidden units.



$P(x_1 = 1) = 0.4$ (from exaplers)

need a model with

$P(1,0,0)$
$+ P(1,0,1)$
$+ P(1,1,0)$
$+ P(1,1,1)$
$= 0.4$

Infinite ways to assign prob. to individual states

- let the model have equal probability of theses states (max. entropy);
- let these states obey B-G distribution (prob. proportional to energy).

– **BM Learning rule**:

$\{V_a\}$: the set of examlers (visible vectors)

$\{H_b\}$: the set of vectors appearing on the hidden units

two phases:

- **clamping phase**: each exampler V_a is clamped to visible units.
- **free-run phase**: none of the visible unit is clamped

$P^+(V_a)$: probability that exampler V_a is applied in clamping phase (determined by the training set)

$P^-(V_a)$: probability that the system is stabilized with V_a at visible units in free-run (determined by the model)

- learning is to construct the weight matrix such that $P^-(V_a)$ is as close to $P^+(V_a)$ as possible.
- A measure of the closeness of two probability distributions (called maximum livelihood, asymmetric divergence or **information gain**):

$$G = \sum_a P^+(V_a) \ln \frac{P^+(V_a)}{P^-(V_a)}$$

- It can be shown $G \geq 0$, and $G = 0$ only if $P^+(V_a) = P^-(V_a)$
- BM learning takes the gradient descent approach to minimal G

$$\frac{\partial \boldsymbol{G}}{\partial \boldsymbol{w}_{ij}} = \sum_{\boldsymbol{a}} \boldsymbol{P}^{+}(\boldsymbol{V}_{\boldsymbol{a}}) \frac{\partial}{\partial \boldsymbol{w}_{ij}} \ln \frac{\boldsymbol{P}^{+}(\boldsymbol{V}_{\boldsymbol{a}})}{\boldsymbol{P}^{-}(\boldsymbol{V}_{\boldsymbol{a}})}$$

$$= \sum_{\boldsymbol{a}} \boldsymbol{P}^{+}(\boldsymbol{V}_{\boldsymbol{a}}) \frac{\boldsymbol{P}^{-}(\boldsymbol{V}_{\boldsymbol{a}})}{\boldsymbol{P}^{+}(\boldsymbol{V}_{\boldsymbol{a}})} \frac{\partial}{\partial \boldsymbol{w}_{ij}} \frac{\boldsymbol{P}^{+}(\boldsymbol{V}_{\boldsymbol{a}})}{\boldsymbol{P}^{-}(\boldsymbol{V}_{\boldsymbol{a}})}$$

$$= \sum_{\boldsymbol{a}} \boldsymbol{P}^{-}(\boldsymbol{V}_{\boldsymbol{a}}) \frac{-\boldsymbol{P}^{+}(\boldsymbol{V}_{\boldsymbol{a}})}{\left(\boldsymbol{P}^{-}(\boldsymbol{V}_{\boldsymbol{a}})\right)^{2}} \frac{\partial}{\partial \boldsymbol{w}_{ij}} \boldsymbol{P}^{-}(\boldsymbol{V}_{\boldsymbol{a}})$$

$$= -\sum_{\boldsymbol{a}} \frac{\boldsymbol{P}^{+}(\boldsymbol{V}_{\boldsymbol{a}})}{\boldsymbol{P}^{-}(\boldsymbol{V}_{\boldsymbol{a}})} \frac{\partial}{\partial \boldsymbol{w}_{ij}} \boldsymbol{P}^{-}(\boldsymbol{V}_{\boldsymbol{a}})$$

$V_a \wedge H_b$: visible nodes are collectively in state a
hidden nodes are collectively in state b

x_i^{ab} : x_i is either a visible or a hidden node

$$\frac{\partial G}{\partial w_{ij}} = \frac{1}{T}(P^-_{ij} - P^+_{ij}) \text{ where } P^-_{ij} = \sum_{ab} P^-(V_a \wedge H_b) x_i^{ab} x_j^{ab}$$

$$P^+_{ij} = \sum_{ab} P^+(V_a \wedge H_b) x_i^{ab} x_j^{ab}$$

$$\text{then} \quad \Delta w_{ij} = -m \frac{\partial G}{\partial w_{ij}} = m(P^+_{ij} - P^-_{ij})$$

P^-_{ij} : how likely both x_i^{ab} and x_j^{ab} are "on" over all possible visible states a and hidden state b in free runs

P^+_{ij} : how likely both x_i^{ab} and x_j^{ab} are "on" over all possible visible states a and hidden state b in clamped runs

BM Learning algorithm

1. compute P^{+}_{ij}

   1.1. clamp one training vector to the visible units of the network

   1.2. anneal the network according to the annealing schedule until equilibrium is reached at the desired minimum temperature.

   1.3. continue to run the network for several more cycles. After each cycle, determine which pairs of connected unit are “on” simultaneously.

   1.4. average the co-occurrence results from 1.3

   1.5. repeat steps 1.1 to 1.4 for all training vectors and average the co-occurrence results to estimate P^{+}_{ij} for each pair of connected units.

2. Compute P^{-}_{ij}

   the same steps as 1.1 to 1.5 except no visible unit is clamped.

3. Calculate and apply weight change $\Delta w_{ij} = m(p_{ij}^{+} - p_{ij}^{-})$

4. Repeat steps 1 to 3 until $p_{ij}^{+} - p_{ij}^{-}$ is sufficiently small.

- **Comments on BM learning**
  1. BM is a stochastic machine not a deterministic one.
  2. It has higher representative/computation power than HM+SA (due to the existence of hidden nodes).
  3. Since learning takes gradient descent approach, only local optimal result is guaranteed (computation still guarantees global optimal if temperature decreases infinitely slow during SA).
  4. Learning can be extremely slow, due to repeated SA involved
  5. Speed up:
     - Hardware implementation
     - Mean field theory: turning BM to deterministic.

$$\text{approximating } P_{ij} \text{ by } \langle x_i \rangle \langle x_j \rangle$$

$$\langle x_i \rangle = \tanh\left[\frac{1}{T}\left(\sum w_{ij} \langle x_j \rangle\right)\right]$$