Hopfield Networks

John Hopfield
Princeton, Caltech, Princeton

Approaches to Hopfield Nets

- Extend linear associative memory ideas by adding cyclic connections, or
- Recurrent neural nets without sequential input, or
- Special case of Kosko's BAM (Bi-Directional Associative Memory), or
- Derive from Cohen-Grossberg theorem

Hopfield Nets

- Generally considered to be **fixed-weight** models; **they don't learn**.
- However, one way to get the weights is through the Hebbian outer-product summation used in the Linear Associative Model.
- Extensions do learn: e.g.Boltzmann network.

Applications

- Model of memory as a dynamical system.
- A technique for finding solutions to certain optimization problems.
- The *practical* applications do not seem so plentiful.

Commentary from James Anderson's book (1995)

- "Hopfield [1982] made the portentous comment: 'This case is isomorphic with an Ising model,' thereby allowing a deluge of physical theory (and physicists) to enter neural network modeling. This flood of new participants transformed the field. In 1974 Little and Shaw made a similar identification of neural network dynamics with the Ising model, but for whatever reason, their idea was not widely picked up at the time".

Commentary from James Anderson's book (1995)

- "Theoretical physicists are an unusual lot, acting like gunslingers in the old West, anxious to prove themselves against a really good problem. And there aren't that many really good problems that might be solvable. As soon as Hopfield pointed out the connection between a new and important problem (network models of brain function) and an old and well-studied problem (the Ising model), many physicists rode into town, so to speak, with the intention of shooting the problem full of holes and then, the brain understood, riding off into the sunset looking for a newer, tougher problem. (Who was that masked physicist?)".

Commentary from James Anderson's book (1995)

- "Unfortunately, the problem of brain function turned out to be more difficult than expected, and it is still unsolved, although a number of interesting results about Hopfield nets were proved. At present, many of the traveling theoreticians have traveled on".

Hopfield Memory

- As with the Linear Associative Memory, the "stored patterns" are represented by the weights.
- To be effective, the patterns should be reasonably orthogonal.
- The number of neurons needed to store p patterns:
  $n \geq 7p$
  which is not very efficient use of neurons.

Model Variants

- Basic: Discrete state, discrete time, asynchronous
- Same as basic, but synchronous
- Continuous state, discrete time
- Continuous state, continuous time

Basic Model

- N neurons, fully connected in a cyclic fashion:
  - Values are +1, -1.
  - Each neuron has a weighted input from all **other** neurons.
  - Weights are **symmetric**: $w_{ij} = w_{ji}$
    and self-weights = $w_{ii} = 0$
  - Activation function on each neuron i is

$$f(\text{net}) = \text{sgn}(\text{net}) = \begin{cases} 1 \text{ if net} > 0 \\ -1 \text{ if net} < 0 \end{cases} \quad (\text{net}_i = \Sigma w_{ij} x_j)$$

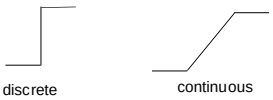
  - If net = 0, then the output is the same as before, **by convention**.

Thresholds

- There are no thresholds or biases. However, these could be represented by units that have all weights = 0 and thus never change their output.

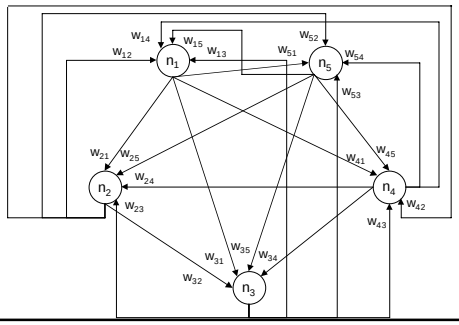
Continuous-State Variant

- On the previous slide, sgn is the same as hardlims (symmetric hard-limiter).
- Could allow continuous neuron outputs and replace it with satlins (symmetric saturating limiter)



- One advantage of the continuous version is that it makes it easier to visualize certain phenomena such as attractors.

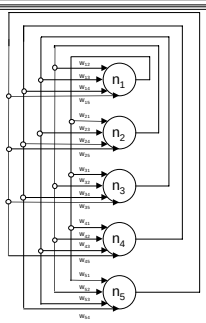
Hopfield Net



Hopfield Net

$$\begin{pmatrix} 0 & w_{12} & w_{13} & w_{14} & w_{15} \\ w_{21} & 0 & w_{23} & w_{24} & w_{25} \\ w_{31} & w_{32} & 0 & w_{34} & w_{35} \\ w_{41} & w_{42} & w_{43} & 0 & w_{45} \\ w_{51} & w_{52} & w_{53} & w_{54} & 0 \end{pmatrix}$$

$w_{ij} = w_{ji}$



Operation (Basic Version)

- Each neuron's output is initially forced to a specified value; this is the "input" state.
- Repeat forever:
  A neuron that has f(net) ≠ current output is "fired", changes its output to 1 or -1 according to the definition of f.
- The firable neuron is chosen arbitrarily.
- When and if the network stabilizes, the current state is the "output".

Operation: Synchronous Variation

- All **firable** neurons are first identified, then all change their state **simultaneously**.
- While this may be viewed as an expedient, it may create behavioral anomalies.

Operational Principle

- Energy Minimization:
  - For an appropriate definition of "energy", **each single firing** can be show to **decrease** the energy.
  - Energy cannot be decreased forever; there is a definite minimum.
  - Therefore operation must eventually terminate.

Final State

- For **asynchronous** (basic) behavior, a **unique** final state is **not** guaranteed: it could be a **local minimum**.
- For **synchronous** behavior, **if** there is a final state, it still is a **local minimum** (it is also reachable by asynchronous firing). However, the network could instead **oscillate** forever.

Working an Example

- Two patterns: (1, -1, 1) and (-1, 1, -1)
- Compute the outer products, sum, normalize, and set diagonals to 0:
  - $(1, -1, 1)^T * (1, -1, 1) +$

$(-1, 1, -1)^T * (-1, 1, -1) =$

Force Diagonal to 0

$$\begin{pmatrix} 2 & -2 & 2 \\ -2 & 2 & -2 \\ 2 & -2 & 2 \end{pmatrix} \Rightarrow \frac{1}{3}\begin{pmatrix} 0 & -2 & 2 \\ -2 & 0 & -2 \\ 2 & -2 & 0 \end{pmatrix}$$

$\frac{1}{3}$ = 1/(number of neurons)

Working an Example

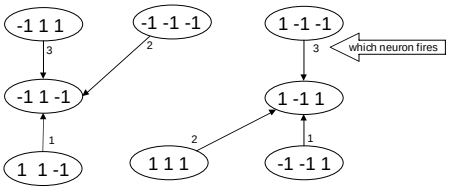
- Eight states total: (-1, -1, -1) … (1, 1, 1)
- For each state, compute the possible next states using the firing rule and the weight matrix:

$$\frac{1}{3}\begin{pmatrix} 0 & -2 & 2 \\ -2 & 0 & -2 \\ 2 & -2 & 0 \end{pmatrix}$$

- Then plot the transitions, noting where the patterns occur.

Working an Example (asynchronous)

- $\frac{1}{3}\begin{pmatrix} 0 & -2 & 2 \\ -2 & 0 & -2 \\ 2 & -2 & 0 \end{pmatrix}$ $\begin{pmatrix} -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \\ -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 \\ -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 \end{pmatrix}$

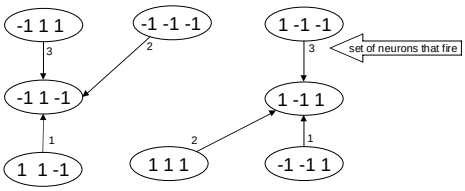


Working an Example (synchronous)

- $\frac{1}{3}\begin{pmatrix} 0 & -2 & 2 \\ -2 & 0 & -2 \\ 2 & -2 & 0 \end{pmatrix}$ $\begin{pmatrix} -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \\ -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 \\ -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 \end{pmatrix}$
- $= \frac{1}{3}\begin{pmatrix} 0 & 4 & 0 & 0 & 0 & 4 & -4 & 0 \\ 4 & 0 & 4 & 0 & 0 & -4 & 0 & -4 \\ 0 & 0 & -4 & -4 & 4 & 4 & 0 & 0 \end{pmatrix}$
- next states = $\begin{matrix} -1 & 1 & -1 & -1 & 1 & 1 & -1 & 1 \\ 1 & -1 & 1 & 1 & -1 & -1 & 1 & -1 \\ -1 & 1 & -1 & -1 & 1 & 1 & -1 & 1 \end{matrix}$

Working an Example (synchronous)

- next states = $\begin{matrix} -1 & -1 & -1 & -1 & 1 & 1 & -1 & 1 \\ 1 & -1 & 1 & 1 & -1 & -1 & 1 & -1 \\ -1 & 1 & -1 & -1 & -1 & 1 & -1 & 1 \end{matrix}$

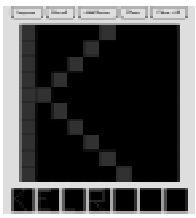


Comparison

- In this example, the asynchronous and synchronous behaviors worked out to be the same.
- This won't always be the case.
- Firing a neuron in the asynchronous *could* **disable** one of the neurons that would have fired simultaneously in the synchronous case.
- Conceivably, the synchronous case could therefore have **cycles** in its behavior.
- See if you can find an example.

Possible Demos

- http://suhep.phy.syr.edu/courses/modules/MM/SIM/Hopfield/



- matlab: demohop1, 2, 3 (uses continuous activation with satlins)

Proving that an Asynchronous Hopfield Net Terminates

- Define an energy function:

$$E(y_1, y_2, \ldots, y_n) = -\Sigma \Sigma w_{ij} y_i y_j$$

where $(y_1, y_2, \ldots, y_n)$ is the vector of neuron outputs, w_{ij} is the weight from neuron j to neuron i, and the double sum is over i and j.

- Remember that w is symmetric ($w_{ij} = w_{ji}$) and diagonal terms are 0.

Proving that an Asynchronous Hopfield Net Terminates

- Observation: The energy function is bounded from below.
- **Claim**: Firing any transition *decreases* the value of the energy function.

$$E(y_1, y_2, \ldots, y_n) = -\Sigma \Sigma w_{ij} y_i y_j$$

- Therefore the net cannot fire forever.

Proof of Claim

- When a neuron i fires, the *increase* (new-old) in energy is entirely due to the contribution to $\Sigma w_{ij} y_i y_j$ of y_i. Since w is symmetric, the amount of this increase is $-\Sigma w_{ij} y_i' y_j - -\Sigma w_{ij} y_i y_j$ where y_i' represents the new value of y_i and the sum is over $i \neq j$ only.
- Since neuron i changes, $y_i' = -y_i$, so the energy increase is

$$2\Sigma w_{ij} y_i y_j = 2 y_i \Sigma w_{ij} y_j$$

where the right-hand summation is over j, where $j \neq i$ only.

Proof of Claim

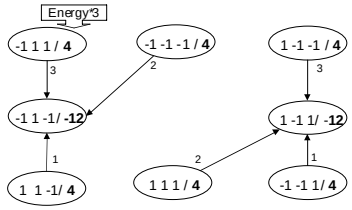
- The energy *increase* is

$$2 y_i \Sigma w_{ij} y_j$$

- If $y_i = 1$ ($y_i' = -1$), then we must have $\Sigma w_{ij} y_j < 0$ in order to activate the neuron, so the increase is 2*1*(negative) which is *negative*.
- If $y_i = -1$ ($y_i' = 1$), then we must have $\Sigma w_{ij} y_j > 0$ in order to activate the neuron, so the increase is 2*(-1)*(postive), which is negative.
- So there is a net energy decrease either way.

Checking Energy

- $\frac{1}{3}\begin{bmatrix} 0 & -2 & 2 \\ -2 & 0 & -2 \\ 2 & -2 & 0 \end{bmatrix}$



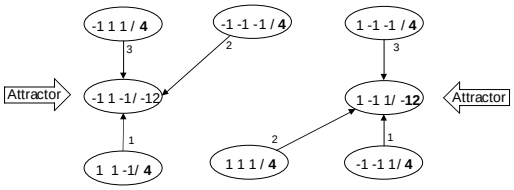
Comment in Hertz, Krogh, and Palmer, 1991

- "Gerard Toulouse has called Hopfield's use of symmetric connections a 'clever step backwards from biological realism'. The cleverness arises from the existence of an energy function".
- Toulouse, et al. Spin glass model of learning by selection. Proc. of the National Academy of Sciences, 83, 1695-1698, 1986.

Attractors

- Minimal energy states are known as "attractors" in the theory of dynamical systems.
- There can also be "repellors" and "saddles" (aka "meta-stable states").

Attractors



Stored Patterns Correspond to Attractors

- When the Hebb rule is used with *orthogonal* patterns, stored patterns correspond to attractors (stable, or minimum-energy, states).
- The reasoning is analogous to the case with the linear associative memory.

Stored Patterns Correspond to Attractors

- The supervised Hebb weight matrix is given by
  $W = \Sigma p^T p$
  where the summation is over all patterns p as row vectors ($p^T p$ is the outer product).
- Let q be a pattern. Assuming **linear** activation functions for the moment, we have *stability* (i.e. minimum energy) if Wq = q.
- Also, stored patterns are eigenvectors of W.

Spurious Attractors

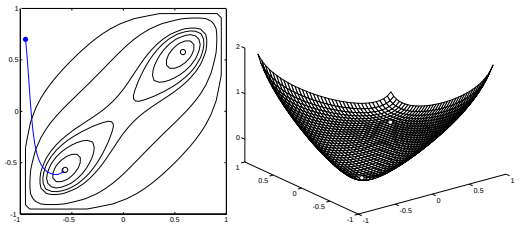
- The converse may be false, i.e. not every attractor is necessarily a pattern.
- For example, if **p** is an attractor, then so is -**p** (i.e. the *negative* of an image).
- Also, certain linear combinations of attractors may be attractors themselves.

Spurious Attractors

- These aspects limit the applicability of Hopfield nets as pattern retrieval devices.
- The following paper is said to present a technique for minimizing the number of spurious attractors:

Li, Michel, and Porod, Analysis and synthesis of a class of neural networks: linear systems operating on a closed hypercube, IEEE Trans. on Circuits and Systems, **36**, 11, 1405-1422, Nov. 1989.

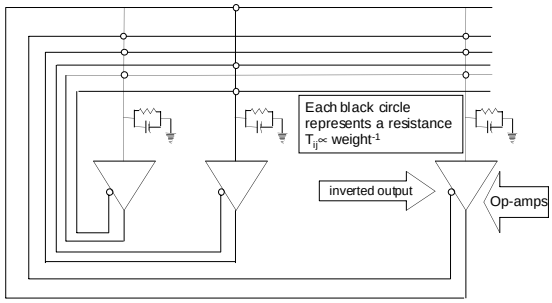
Attractors in a Continuous Analog of the Example



Lyapunov Functions

- For the continuous case, the energy function is called a Lyapunov function.
- The Hopfield network minimizes the value of the Lyapunov function.

Physical Realization of a Continuous Hopfield Net



Equations of Operation

$$C\frac{dn_i(t)}{dt} = \sum_{j=1}^{S} T_{i,j} a_j(t) - \frac{n_i(t)}{R_i} + I_i$$

n_i - input voltage to the ith amplifier
a_i - output voltage of the ith amplifier
C - amplifier input capacitance
I_i - fixed input current to the ith amplifier

$$|T_{i,j}| = \frac{1}{R_{i,j}} \qquad \frac{1}{R_i} = \frac{1}{\rho} + \sum_{j=1}^{S} \frac{1}{R_{i,j}} \qquad n_i = f^{-1}(a_i) \qquad a_i = f(n_i)$$

Commercial Success

- At least one company, Attrasoft http://attrasoft.com/new.htm claims to have products based on Hopfield nets and Boltzmann machines.

Boltzmann Machines

Learning by Correlation

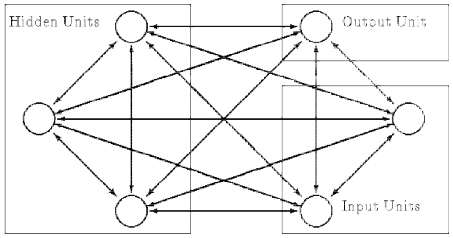
Boltzmann Machine

- Proposed by Ackley, Hinton, and Sejnowski, 1985
- Extends Hopfield model with learning
- Based on probabilitstic operation during training, correlations.
- Deterministic operation once weights determined.

Boltzmann Machine Structure

- The Boltzmann Machine is a Hopfield network, in which
- the neurons are divided into two subsets:
  - **visible**, which are further divided into:
    - input
    - output
  - **hidden**

Boltzmann Machine Structure



Boltzmann Machine Operation

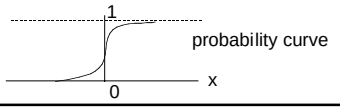
- There are two modes of operation:
  - clamped mode
  - free mode
- In **clamped** mode, the input and output of visible neurons are held fixed, while the hidden neurons are allowed to vary.
- In **free** mode, **only the inputs are held fixed** and other neurons are allowed to vary.

Stochastic Operation (used in Training)

- All neurons have output in {+1, -1}.
- The activation function determines not the exact next input, but rather the **probability** of the neuron's output being set to 1:
  - f(net) = probability that output is set to 1
- where $f(x) = 1/(1 + \exp(-2\beta x))$
  - where β is a parameter to be determined.

Stochastic Operation

- $f(x) = 1/(1 + \exp(-2\beta x))$
- Obviously this is a sigmoid:
  - With $\beta = 0$, the probability of setting to 1 is 0.5, i.e. total randomness.
  - As β increases, the probability of setting to 1 is > 0.5 if $x > 0$, and < 0.5 if $x < 0$.



Controlling β

- In order to stabilize the network, β is gradually increased from 0 over time.
- For a given input, initially changes in neuron state will be random, but eventually it will be determined *only* by the activation value (weighted sum of inputs) without randomness.
- At this point, the network will then stabilize.

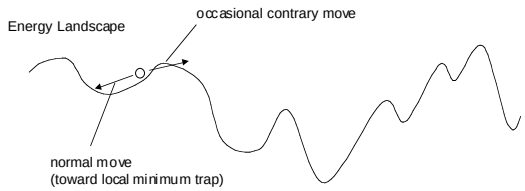
Controlling β

- We can think of the increase in β as "cooling" the network.
- Thus, we can govern beta as $\beta = 1/T$ where T is the "temperature", which decreases with time (according to a *schedule*).
- The overall process is known as "simulated annealing".

Role of Annealing in Stabilization

- The idea is to avoid local minima.
- Annealing provides an approach.
- Generally, move in direction of decreasing energy.
- **Occasionally, accept a move that increases energy.**
- This will be done with high probability at first, but lower as annealing progresses.

Role of Annealing in Stabilization



The probability of making a contrary move is inversely proportional to the energy increase and to the temperature (higher probability earlier in the annealing schedule).

Energy-Based Simulation

- As we know from Hopfield, theory, making a single transition according to the activation function will decrease the energy.
- So we can simply decide to "flip" a neuron based on whether the flip lowers the energy (defined as $-\Sigma \Sigma w_{ij} y_i y_j$).

Energy-Based Simulation

- To include the annealing temperature:
  - If a flip lowers the energy, do it.
  - If a flip raises the energy by ΔE, do it with probability $\exp(-\Delta E/T)$.
    - In other words, generate a random number $\rho \in [0, 1]$ and if $\rho < \exp(-\Delta E/T)$, flip the neuron, otherwise leave it as is.

Energy-Based Simulation

- This technique was originally used in the famous Metropolis, Rosenbluth, Teller equation of state calculations in statistical mechanics.

Energy-Based Simulation

- In order to compute ΔE, it is not necessary to fully compute the energy before and after. Instead could just use $\Delta E = \Sigma\, w_{ij}\, y_j$ where i is the neuron being flipped.

Boltzmann Distribution

- The name of the machine derives from the fact that, at steady state, if s and t are two states with energies E_s and E_t respectively, then the probabilities of being in those states P[s] vs. P[t] satisfy

  $$P[s]/P[t] = \exp(E_t - E_s)/T$$

  where T is the temperature. This is known as the "Boltzmann distribution" or "Boltzmann-Gibbs distribution".

Multiple Simulations Per Sample

- Suppose we set the input and output neurons according to a specific **sample**.
- We then anneal the network.
  The final state reached is not necessarily unique, due to the probabilistic moves made along the way.
- We can observe, over **several** such simulations, which neurons' outputs are **correlated** at the end, represented as a **correlation** $\rho_{ij} = E[y_i\, y_j]$, the average value of the product of the outputs of neurons i and j.

Learning Rule

- Let ρ_{ij}^{+} be the correlation value when the network is run in **clamped** mode, and ρ_{ij}^{-} be the correlation value when the network is run in **free** mode.
- The Boltzmann learning rule is

  $$\Delta w_{ij} = \eta(\rho_{ij}^{+} - \rho_{ij}^{-})$$

  where η is the learning rate.
- In other words, whether weights are changed depends on the difference between the correlations in clamped vs. free mode.

Summary of Boltzmann Learning

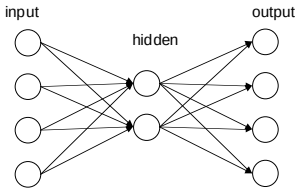
- Repeat until weights stabilize
  - For each sample:
    - Run the sample multiple times in both clamped and free mode.
      - In each run, annealing takes place.
      - Accumulate statistics from each run.
    - Compute the weight update for the sample based on the statistics and the learning rule.

Conclusion of Learning

- At the conclusion of a number of weight-adjustment cycles, we have a Hopfield net that can be operated deterministically.

Boltzman Example

- Ackley, Hinton, and Sejnowski, 1985 present the following example:
- 4 line one-hot encoder-decoder



Boltzman Example

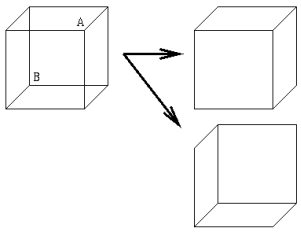
- To prevent weights from growing too large, used a "noisy" **clamping** technique: each *on* bit of a clamped vector is set to *off* with prob. 0.15 and each *off* bit set to *on* with prob. 0.05.
- Network was **unclamped** and allowed to reach equilibrium. Statistics were gathered for the same number of annealings as in the clamped case.
- Annealing schedule: (time units @ temperature) 2@20, 2@15, 2@12, 4@10.

Additional Examples

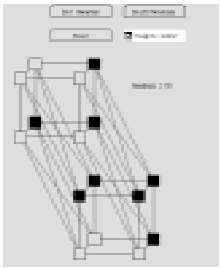
- 4-3-4 encoder/decoder converged quickly
- 8-3-8: more difficult
- 40-10-40: converged in 98.6% of runs.

Necker Cube Demo

(http://www2.psy.uq.edu.au/~mav/java/Necker.html)



Necker Cube Demo



Neurons representing whether vertex is shown in front or back

Blue weights are positive, red are negative.

Constraints Represented

- Each vertex can have only one interpretation. Therefore there are negative weights connecting units representing different interpretations of the same vertex.
- The same interpretation cannot be given to more than one vertex - so units representing the same interpretation are connected with negative weights.
- Units from the same interpretation should be on together, so locally consistent units are connected with positive weights.

Speedup Possibility

- Training of a Boltzmann machine is extremely slow.
- A possible speedup is to use the "mean-field" approximation to get the correlation values.
- This approach is due to Peterson and Anderson, 1987.

Mean-Field Theory

- If f is a function of two variables, then the expectation
  E[f(x, y)]
  can be *approximated* by
  f(E[x], E[y])
- This idea can be applied to the weight change rule of the Boltzmann machine, which entails computing

$$\rho_{ij} = E[y_i y_j] \approx E[y_i] E[y_j]$$

Mean-Field Theory

- For the Boltzmann distribution, the probability that node i takes value 1 at temperature T can be shown to be:

$$p_1 = 1/(1+\exp(-\Sigma w_{ij} E[y_j]/T))$$

- So the *expected* output of node i is

$$E[y_i] = 1*p_1 + (-1)*(1- p_1)$$

$$= \tanh(\Sigma w_{ij} E[y_j] / 2T)$$

Mean-Field Theory

- We now have n non-linear equations in n unknowns $E[y_j]$ which can be solved ***deterministically*** by using successive approximations (without simulation!, but we still have to anneal).
- We can then use these approximations to update the weights:

$$\Delta w_{ij} = \eta(E^+[y_i] E^+[y_j] - E^-[y_i] E^-[y_j])$$

where + and - designate clamped vs. free as before.