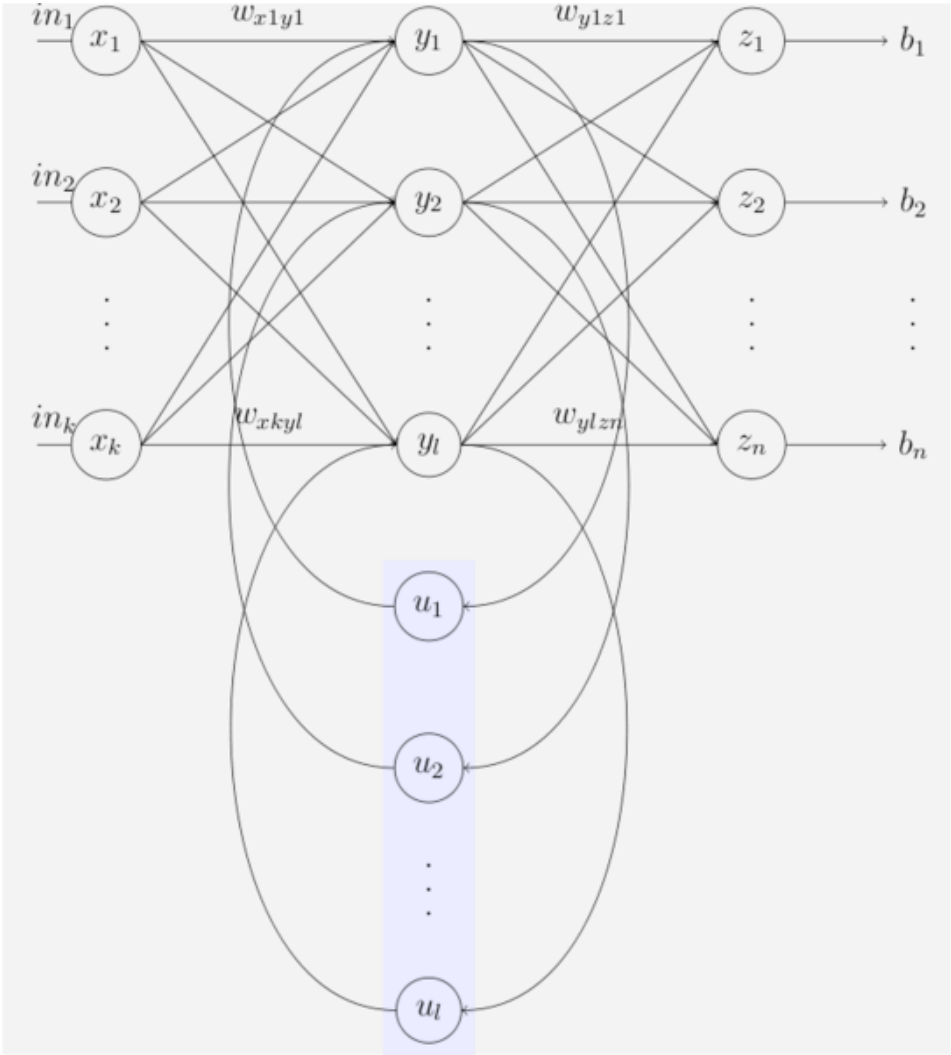
Recurrent (or Auto-Associative) Neural Networks

Elman Network

- includes a layer of "context units" connected from hidden layer with weight of 1
- at each iteration, hidden unit outputs copied into context units



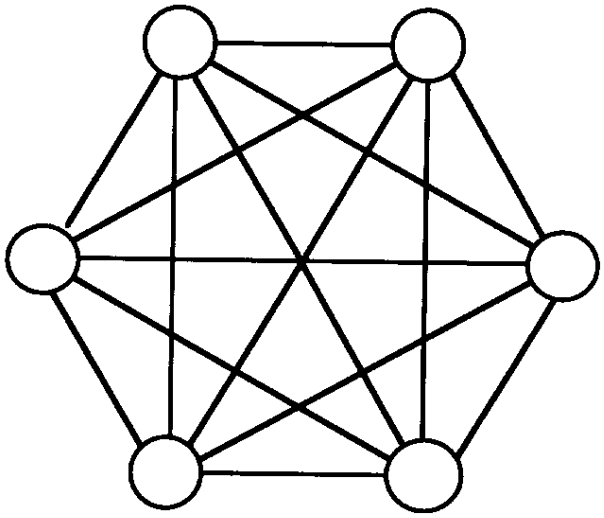
Jordan Network

- similar to Elman network but:
  - context units connected to output layer
  - values in context units (c_i) are updated, not replaced, by values of output units (o_i), i.e., $c_i = o_i + \gamma c_i$

Associative Memory

- store a set of *p* patterns
- when presented with a new pattern, network responds by producing closest stored pattern “in memory”
- uses relaxation learning

Hopfield Networks



- N interconnected neurons, potentially completely recurrent
- all neurons are both input & output; all weights are symmetric (but $w_{ii}=0$)
- activation values are binary (typically -1/+1)

- activation rule: $a_j = \begin{cases} +1 & \text{if } \Sigma_i w_{ij} a_i > \theta_j \\ -1 & \text{otherwise} \end{cases}$

  where θ_j is the threshold

Learning and Recall

- set weights to ensure learned patterns are stable
- pattern x^p stable if, when clamped (activation fixed for the given neurons), all neurons stable
- recalling a pattern is achieved by activating (possibly a subset of) neurons with the corresponding inputs and then letting the network settle

Hebbian learning

- $w_{ij} \leftarrow$ correlation of activations (+1)
- for *n* patterns x^p

  $w_{ij} = \sum_p x_i^p x_j^p$ if $i \neq j$, else 0
- limits of learning: for a network of *N* neurons, recall errors become severe above 0.15*N* "memories"
  - stored patterns become unstable
  - spurious stable states appear

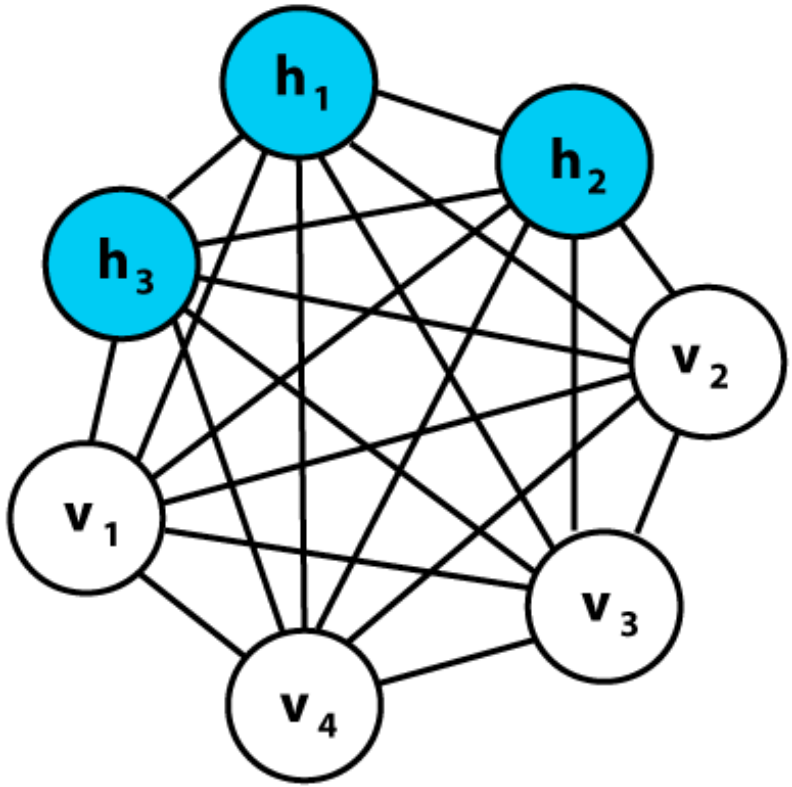
Why simple Hebbian Learning Fails

- don't know how to set weights to or from "hidden" units (or those that aren't part of a learned pattern)

  ∴ can't generalize beyond simple Hopfield Networks

Boltzmann Machines

[Ackley, Hinton, & Sejnowski 1985]



- similar to Hopfield network
  - units take on values of 0 or 1
  - weights are symmetric
  - but allows for hidden units (h) as well as visible units (v)

- weights adjusted through stochastic update rule based on simulated annealing; thus, Boltzmann machine can be viewed as a Monte Carlo version of Hopfield network

Energy

- global energy of network is: $E = -\Sigma_{i \neq j} w_{ij} a_i a_j + \Sigma_i \theta_i a_i$
  - first term suggests that the network has minimal energy when both units *i* and *j* are correlated and connected by a positive weight
- thus, the difference in global energy resulting from unit *i* being off (0) rather than on (1) is:

  $\Delta E_i = E_{i\text{-}off} - E_{i\text{-}on} = 0 - (-\Sigma_j w_{ij} a_j + \theta_i) = \Sigma_j w_{ij} a_j - \theta_i$
- Boltzmann Factor: energy of a state is proportional to the negative log probability of that state, i.e., higher energy states have lower probability

Simulated Annealing

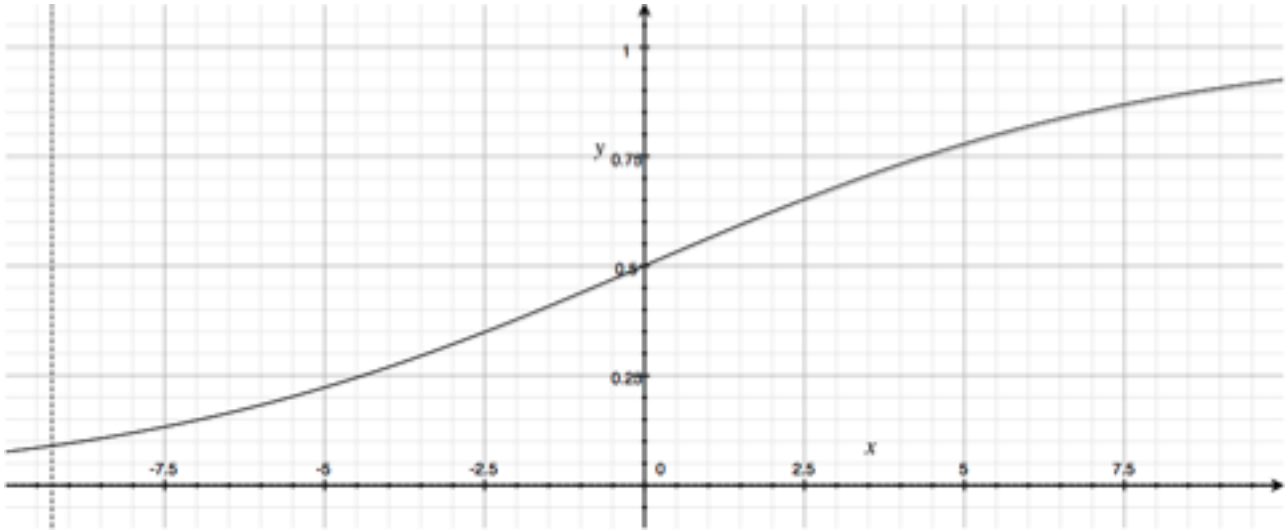
- start with high T
- network run by repeatedly choosing a unit and setting its state according to activation function
- gradually reduce T to 1 until energy level fluctuates around global minimum
- at low temperature, there is a strong bias to states with low energy

Probabilistic Activation Function

- Boltzmann machine units are stochastic; probabilistic activation function:

  $P(a_i \leftarrow 1) = 1/(1 + e^{-\Delta E_i/T})$

  where T is the temperature of the system



Boltzmann Learning

- learning works by adjusting weights so that global states with highest probabilities get lowest energies
- divide units into visible units (V) and hidden units (H)
- training data vectors applied to V
- distribution of training vectors denoted $P^+(V)$
- converged distribution of visible units of Boltzmann machine denoted $P^-(V)$
- want to minimize Kullback-Leibler distance (asymmetric divergence error term) over all possible states of V:

$$G = \Sigma_v P^+(v) \ln (P^+(v) / P^-(v))$$

Contrastive Hebbian Learning: Waking Phase

- for each training pattern, input and output units are clamped to pattern
- network permitted to settle (annealing)
- frequency of correlated activation $\langle a_i a_j \rangle^{\text{clamped}}$ between all units measured for some fixed time at thermal equilibrium
- incremental version of Hebbian learning used:

  $\Delta w_{ij} = \alpha \langle a_i a_j \rangle^{\text{clamped}}$

Contrastive Hebbian Learning: Sleeping

- network run freely (no units clamped) and annealed
- frequency of correlated activation $\langle a_i a_j \rangle^{\text{free}}$ between all units measured for some fixed time at thermal equilibrium
- reverse incremental version of Hebbian learning used:

$$\Delta w_{ij} = -\alpha \langle a_i a_j \rangle^{\text{free}}$$

Motivation

- when network response is identical between clamped and non-clamped phases, weights are stable
- waking and sleeping learning will balance each other
- when network response without clamping differs from its response with clamping, the difference is error and is compensated for by sleeping-phase learning

Restricted Boltzmann Machine (RBM)

- learning time for Boltzmann machine can be excessive
- can be made efficient by not allowing visible-visible or hidden-hidden connections:
  - starting with a data vector on the visible units, update all of the hidden units in parallel.
  - update all of the visible units in parallel to get a "reconstruction".
  - can use the trained hidden units as input data for a new, higher-level RBM