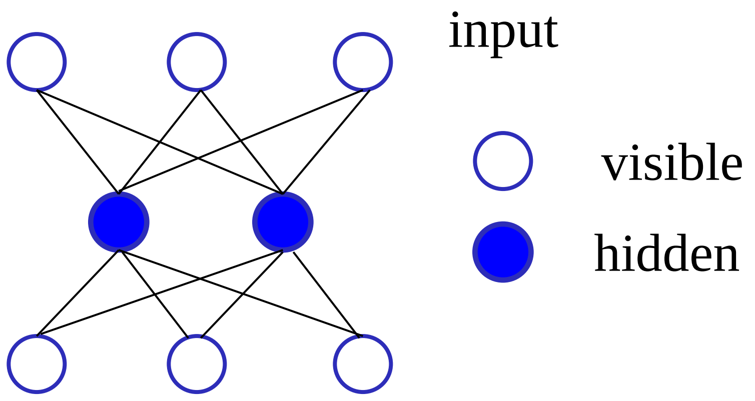
CSE 5526: Introduction to Neural Networks

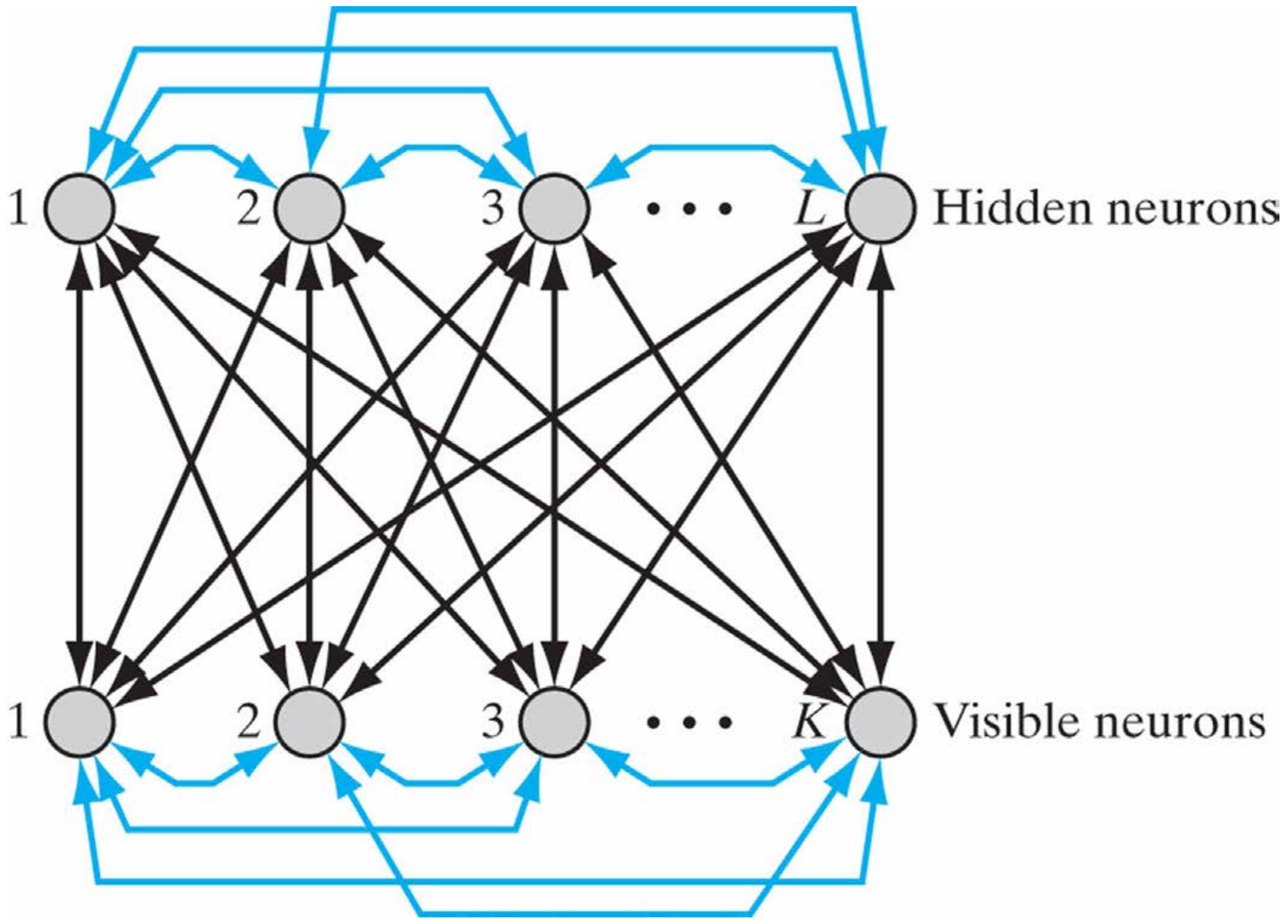
Boltzmann Machines

Boltzmann machines are a stochastic extension of Hopfield networks

- A Boltzmann machine is a stochastic learning machine that consists of visible and hidden units and symmetric connections
- The network can be layered and visible units can be either "input" or "output" (connections are not directed)



In general, the network is fully connected



Boltzmann machines have the same energy function as Hopfield networks

- Because of symmetric connections, the energy function of neuron configuration **x** is:

$$E(\mathbf{x}) = -\frac{1}{2}\sum_i \sum_j w_{ji} x_i x_j = \mathbf{x}^T W \mathbf{x}$$

- Can we start there and derive a probability distribution over configurations?

The Boltzmann-Gibbs distribution defines probabilities from energies

- Consider a physical system with many states.
  - Let p_i denote the probability of occurrence of state i
  - Let E_i denote the energy of state i
- From statistical mechanics, when the system is in thermal equilibrium, it satisfies

$$p_i = \frac{1}{Z}\exp\left(-\frac{E_i}{T}\right) \quad \text{where} \quad Z = \sum_i \exp\left(-\frac{E_i}{T}\right)$$

  - Z is called the partition function, and T is called the temperature
  - The Boltzmann-Gibbs distribution

Remarks

- Lower energy states have higher probability of occurrences
- As *T* decreases, the probability is concentrated on a small subset of low energy states

Boltzmann-Gibbs distribution example

- Consider $\boldsymbol{E} = [1, 2, 3]$
- At $T = 10$,
    - $\tilde{\boldsymbol{p}} = \exp\left(-\frac{\boldsymbol{E}}{T}\right) = [0.905, 0.819, 0.741]$
    - $Z = \sum_i \tilde{p}_i = 0.905 + 0.819 + 0.741 = 2.464$
    - $\boldsymbol{p} = \frac{1}{Z}\tilde{\boldsymbol{p}} = [0.367, 0.332, 0.301]$

Boltzmann-Gibbs distribution example

- Consider $\boldsymbol{E} = [1, 2, 3]$
- At $T = 1$,
  - $\tilde{\boldsymbol{p}} = \exp\left(-\frac{\boldsymbol{E}}{T}\right) = [0.368, 0.135, 0.050]$
  - $Z = \sum_i \tilde{p}_i = 0.368 + 0.135 + 0.050 = 0.553$
  - $\boldsymbol{p} = \frac{1}{Z}\tilde{\boldsymbol{p}} = [0.665, 0.245, 0.090]$

Boltzmann-Gibbs distribution example

- Consider $\boldsymbol{E} = [1, 2, 3]$
- At $T = 0.1$,
    - $\tilde{\boldsymbol{p}} = \exp\left(-\frac{\boldsymbol{E}}{T}\right) = [4.54 \cdot 10^{-5}, 2.06 \cdot 10^{-9}, 9.36 \cdot 10^{-14}]$
    - $Z = \sum_i \tilde{p}_i = 4.54 \cdot 10^{-5} + 2.06 \cdot 10^{-9} + 9.36 \cdot 10^{-14}$
      $= 4.54 \cdot 10^{-5}$
    - $\boldsymbol{p} = \frac{1}{Z}\tilde{\boldsymbol{p}} = [0.99995, 4.54 \cdot 10^{-5}, 2.06 \cdot 10^{-9}]$

Boltzmann-Gibbs distribution applied to Hopfield network energy function

$$p(\mathbf{x}) = \frac{1}{Z}\exp\left(-\frac{1}{T}E(\mathbf{x})\right) = \frac{1}{Z}\exp\left(\frac{1}{2T}\mathbf{x}^T W \mathbf{x}\right)$$

- Partition function For N neurons, involves 2^N terms

$$Z = \sum_{\mathbf{x}} p(\mathbf{x})$$

- Marginal over H of the neurons Involves 2^H terms

$$p(\mathbf{x}_\alpha) = \sum_{\mathbf{x}_\beta} p(\mathbf{x}_\alpha, \mathbf{x}_\beta)$$

Boltzmann machines are unsupervised probability models

- The primary goal of Boltzmann machine learning is to produce a network that models the probability distribution of observed data (at visible neurons)
  - Such a net can be used for pattern completion, as a part of an associative memory, etc.
- What do we want to do with it?
  - Compute the probability of a new observation
  - Learn parameters of the model from data
  - Estimate likely values completing partial observations

Compute the probability of a new observation

- Divide the entire net into the subset $\mathbf{x}_\alpha$ of visible units and $\mathbf{x}_\beta$ of hidden units
- Given parameters W, compute likelihood of observation $\mathbf{x}_\alpha$

$$p(\mathbf{x}_\alpha) = \sum_{\mathbf{x}_\beta} p(\mathbf{x}) = \sum_{\mathbf{x}_\beta} \frac{1}{Z} \exp\left(-\frac{1}{T} E(\mathbf{x})\right)$$

$$= \sum_{\mathbf{x}_\beta} \frac{1}{Z} \exp\left(\frac{1}{2T} \mathbf{x}^T W \mathbf{x}\right)$$

Learning can be performed by gradient descent

- The objective of Boltzmann machine learning is to maximize the likelihood of the visible units taking on training patterns by adjusting W
- Assuming that each pattern of the training sample is statistically independent, the log probability of the training sample is:

$$L(\mathbf{w}) = \log \prod_{\mathbf{x}_\alpha} P(\mathbf{x}_\alpha) = \sum_{\mathbf{x}_\alpha} \log P(\mathbf{x}_\alpha)$$

Log likelihood of data has two terms

$$L(\mathbf{w}) = \log \prod_{\mathbf{x}_\alpha} P(\mathbf{x}_\alpha) = \sum_{\mathbf{x}_\alpha} \log P(\mathbf{x}_\alpha)$$

$$= \sum_{\mathbf{x}_\alpha} \log \sum_{\mathbf{x}_\beta} \frac{1}{Z} \exp\left(-\frac{E(\mathbf{x})}{T}\right)$$

$$= \sum_{\mathbf{x}_\alpha} \left[\log \sum_{\mathbf{x}_\beta} \exp\left(-\frac{E(\mathbf{x})}{T}\right) - \log Z\right]$$

$$= \sum_{\mathbf{x}_\alpha} \left[\log \sum_{\mathbf{x}_\beta} \exp\left(-\frac{E(\mathbf{x})}{T}\right) - \log \sum_{\mathbf{x}} \exp\left(-\frac{E(\mathbf{x})}{T}\right)\right]$$

Gradient of log likelihood of data

$$\frac{\partial L(\mathbf{w})}{\partial w_{ji}}$$

$$= \sum_{\mathbf{x}_\alpha} \left[\frac{\partial}{\partial w_{ji}} \log \sum_{\mathbf{x}_\beta} \exp\left(-\frac{E(\mathbf{x})}{T}\right) - \frac{\partial}{\partial w_{ji}} \log \sum_{\mathbf{x}} \exp\left(-\frac{E(\mathbf{x})}{T}\right) \right]$$

$$= \sum_{\mathbf{x}_\alpha} \left[\frac{-\frac{1}{T} \sum_{\mathbf{x}_\beta} \exp\left(-\frac{E(\mathbf{x})}{T}\right) \frac{\partial E(\mathbf{x})}{\partial w_{ji}}}{\sum_{\mathbf{x}_\beta} \exp\left(-\frac{E(\mathbf{x})}{T}\right)} + \frac{\frac{1}{T} \sum_{\mathbf{x}} \exp\left(-\frac{E(\mathbf{x})}{T}\right) \frac{\partial E(\mathbf{x})}{\partial w_{ji}}}{\sum_{\mathbf{x}} \exp\left(-\frac{E(\mathbf{x})}{T}\right)} \right]$$

Gradient of log likelihood of data

- Then, since $\frac{\partial E(\mathbf{x})}{\partial w_{ji}} = -x_j x_i$

$$\frac{\partial L(\mathbf{w})}{\partial w_{ji}} = \frac{1}{T}\sum_{\mathbf{x}_\alpha}\left[\sum_{\mathbf{x}_\beta}\frac{\exp\left(-\frac{E(\mathbf{x})}{T}\right)x_j x_i}{\sum_{\mathbf{x}_\beta}\exp\left(-\frac{E(\mathbf{x})}{T}\right)} - \frac{\sum_{\mathbf{x}}\exp\left(-\frac{E(\mathbf{x})}{T}\right)x_j x_i}{Z}\right]$$

$$= \frac{1}{T}\left(\underbrace{\sum_{\mathbf{x}_\alpha}}_{\text{training patterns}}\sum_{\mathbf{x}_\beta}P(\mathbf{x}_\beta|\mathbf{x}_\alpha)x_j x_i - \sum_{\mathbf{x}_\alpha}\underbrace{\sum_{\mathbf{x}}P(\mathbf{x})x_j x_i}_{\text{constant w.r.t. } \sum_{\mathbf{x}_\alpha}}\right)$$

Gradient of log likelihood of data

$$\frac{\partial L(\mathbf{w})}{\partial w_{ji}} = \frac{1}{T}\left(\rho_{ji}{}^{+} - \rho_{ji}{}^{-}\right)$$

- Where $\rho_{ji}{}^{+} = \sum_{\mathbf{x}_\alpha} \sum_{\mathbf{x}_\beta} P\left(\mathbf{x}_\beta \middle| \mathbf{x}_\alpha\right) x_j x_i = \sum_{\mathbf{x}_\alpha} E_{\mathbf{x}_\beta|\mathbf{x}_\alpha}\{x_j x_i\}$
  - is the mean correlation between neurons *i* and *j* when the visible units are "clamped" to $\mathbf{x}_\alpha$
- And $\rho_{ji}{}^{-} = K \sum_{\mathbf{x}} P(\mathbf{x}) x_j x_i = E_{\mathbf{x}}\{x_j x_i\}$
  - is the mean correlation between *i* and *j* when the machine operates without "clamping"

Maximization of $L(\mathbf{w})$

- To *maximize* $L(\mathbf{w})$, we use gradient *ascent*:

$$\triangle\, w_{ji} = \varepsilon \frac{\partial L(\mathbf{w})}{\partial w_{ji}} = \frac{\varepsilon}{T}\left({\rho_{ji}}^{+} - {\rho_{ji}}^{-}\right)$$

$$= \eta\left({\rho_{ji}}^{+} - {\rho_{ji}}^{-}\right)$$

where the learning rate η incorporates the temperature T

Positive and negative phases

- Thus there are two phases to the learning process:
    1. **Positive phase**: the net operates in the "clamped" condition, where visible units take on training patterns with the desired probability distribution
    2. **Negative phase**: the net operates freely without the influence of external input

Remarks on Boltzmann machine learning

- The Boltzmann learning rule is a remarkably simple local rule, concerning only "presynaptic" and "postsynaptic" neurons
  - Such local rules are biologically plausible
- But, computing ${\rho_{ji}}^{+}$ directly requires summing over 2^{H} terms for each observed data point
- Computing ${\rho_{ji}}^{-}$ requires summing 2^{N} terms once
- Can we approximate those sums?

Sampling methods can approximate expectations (difficult integrals/sums)

- Want to compute $\rho_{ji}^{-} = \sum_{\mathbf{x}} P(\mathbf{x}) x_j x_i = E_{\mathbf{x}}\{x_j x_i\}$
- We can approximate that expectation as

$$\rho_{ji}^{-} = E_{\mathbf{x}}\{x_j x_i\} \approx \frac{1}{N} \sum_{\mathbf{x}_n} x_j x_i$$

  - Where $\mathbf{x}_n$ are samples drawn from $P(\mathbf{x}_n)$
- In general $E_{\mathbf{x}}\{f(\mathbf{x})\} \approx \frac{1}{N} \sum_{\mathbf{x}_n} f(\mathbf{x}_n)$
  - The sum converges to the true expectation as the number of samples goes to infinity
  - This is called Monte Carlo integration / summation

Gibbs sampling can draw samples from intractable distributions

- Many high-dimensional probability distributions are difficult to compute or sample from
- But Gibbs sampling can draw samples from them
  - If you can compute the probability of some variables given others
  - This is typically the case in graphical models, including Boltzmann machines
- This kind of approach is called Markov chain Monte Carlo (MCMC)

Gibbs sampling can draw samples from intractable distributions

- Consider a *K*-dimensional random vector $\mathbf{x} = (x_1, \ldots, x_K)^T$
- Suppose we know the conditional distributions
$$p(x_k | x_1, \ldots, x_{k-1}, x_{k+1}, \ldots, x_K) = p(x_k | \mathbf{x}_{\sim k})$$
- Then by sampling each x_k in turn (or randomly), the distribution of samples will eventually converge to
$$p(\mathbf{x}) = p(x_1, \ldots, x_K)$$
- Note: it can be difficult to know how long to sample

Gibbs sampling algorithm

- For iteration *n*:

  $x_1(n)$ is drawn from $p(x_1|x_2, \dots, x_K)$

  …

  $x_k(n)$ is drawn from $p(x_k|x_1, \dots, x_{k-1}, x_{k+1}, \dots, x_K)$

  …

  $x_K(n)$ is drawn from $p(x_K|x_1, \dots, x_{K-1})$
- Each iteration samples each random variable once in the natural order
- Newly sampled values are used immediately (i.e., asynchronous sampling)

Gibbs sampling in Boltzmann machines

- Need $p(x_k | x_1, \dots, x_{k-1}, x_{k+1}, \dots, x_K) = p(x_k | \mathbf{x}_{\sim k})$
- Can compute from Boltzmann-Gibbs distribution

$$p(x_k | \mathbf{x}_{\sim k}) = \frac{p(x_k = 1, \mathbf{x}_{\sim k})}{p(x_k = 1, \mathbf{x}_{\sim k}) + p(x_k = -1, \mathbf{x}_{\sim k})}$$

$$= \frac{\frac{1}{Z} \exp\left(-\frac{E^+}{T}\right)}{\frac{1}{Z} \exp\left(-\frac{E^+}{T}\right) + \frac{1}{Z} \exp\left(-\frac{E^-}{T}\right)}$$

$$= \frac{1}{1 + \exp\left(\frac{1}{T}(E^+ - E^-)\right)} = \sigma\left(\frac{\Delta E}{T}\right)$$

So Boltzmann machines use stochastic neurons with sigmoid activation

- Neuron k is connected to all other neurons:

$$v_k = \sum_j w_{kj} x_j$$

- It is then updated stochastically so that

$$x_k = \begin{cases} 1 & \text{with prob. } \varphi(v_k) \\ -1 & \text{with prob. } 1 - \varphi(v_k) \end{cases}$$

  - Where

$$\varphi(v) = \frac{1}{1 + \exp\left(-\frac{v}{T}\right)}$$

Prob. of flipping a single neuron

- Consider the prob. of flipping a single neuron *k*:

$$P(x_k \rightarrow -x_k) = \frac{1}{1 + \exp\left(\frac{\Delta E_k}{2T}\right)}$$

  where ΔE_i is the energy change due to the flip (proof is a homework problem)

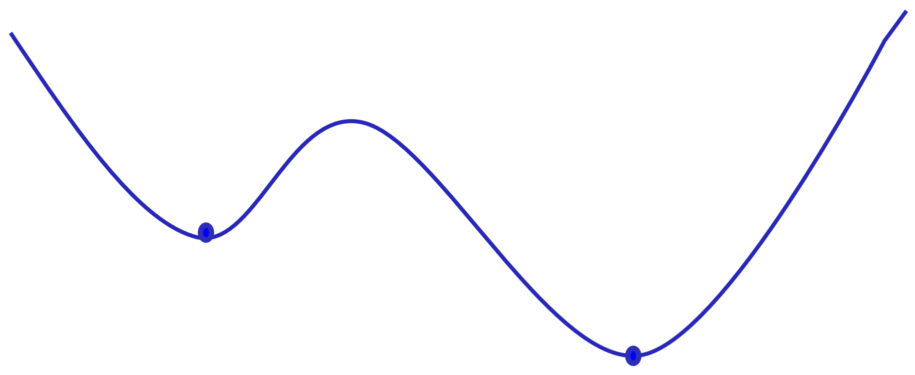
- So a change that decreases the energy is more likely than that increasing the energy

Simulated annealing

- As the temperature T decreases
  - The average energy of a stochastic system decreases
  - It reaches the global minimum as $T \to 0$
  - So for optimization problems, should favor low temps
- But, convergence is slow at low temps
  - due to trapping at local minima
- Simulated annealing is a stochastic optimization technique that gradually decreases T
  - To get the best of both worlds

Simulated annealing (cont.)

- No guarantee for the global minimum, but higher chances for lower local minima



- Boltzmann machines use simulated annealing to gradually lower *T*

Full Boltzmann machine training algorithm

- The entire algorithm consists of the following nested loops:
  1. Loop over all training data points, accumulating gradient of each weight
  2. For each data point, compute expectation $\langle x_i x_j \rangle$ with $\mathbf{x}_\alpha$ clamped and free
  3. Compute expectations using simulated annealing, gradually decreasing T
  4. For each T, sample the state of the entire net a number of times using Gibbs sampling

Remarks on Boltzmann machine training

- Boltzmann machines are extremely slow to train
  - but work well once they are trained
- Because of its computational complexity, the algorithm has only been applied to toy problems
- But: the restricted Boltzmann machine is much easier to train, stay tuned…

An example

- The encoder problem (see blackboard)
- Ackley, Hinton, Sejnowski (1985)

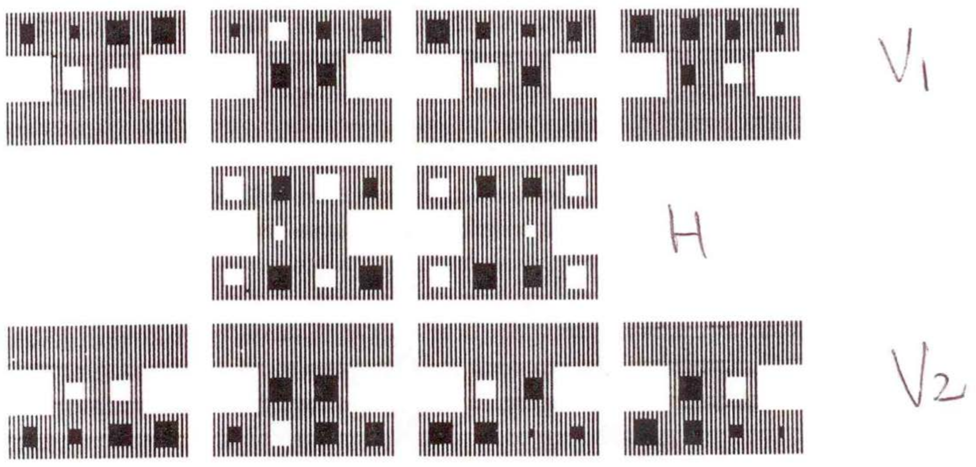


Figure 2 A solution to an encoder problem. The link weights are displayed using a recursive notation. Each unit is represented by a shaded 1-shaped box; from top to bottom the rows of boxes represent groups V_1, H, and V_2. Each shaded box is a map of the entire network, showing the strengths of that unit's connections to other units. At each position in a box, the size of the white (positive) or black (negative) rectangle indicates the magnitude of the weight. In the position that would correspond to a unit connecting to itself (the second position in the top row of the second unit in the top row, for example), the bias is displayed. All connections between units appear twice in the diagram, once in the box for each of the two units being connected. For example, the black square in the top right corner of the left-most unit of V_1 represents the same connection as the black square in the top left corner of the rightmost unit of V_1. This connection has a weight of -30.