**Boltzmann Machine**

A **Boltzmann machine** is the name given to a type of [stochastic recurrent neural network](http://www.answers.com/topic/stochastic-neural-network) by[Geoffrey Hinton](http://www.answers.com/topic/geoffrey-hinton) and [Terry Sejnowski](http://www.answers.com/topic/terry-sejnowski). Boltzmann machines can be seen as the [stochastic](http://www.answers.com/topic/stochastic-process-2),[generative](http://www.answers.com/topic/generative-model) counterpart of [Hopfield nets](http://www.answers.com/topic/hopfield-net). They were one of the first examples of a neural network capable of learning internal representations, and are able to represent and (given sufficient time) solve difficult combinatoric problems. However, due to a number of issues discussed below, Boltzmann machines with unconstrained connectivity have not proven useful for practical problems in machine learning or inference. They are still theoretically intriguing, however, due to the locality and [Hebbian](http://www.answers.com/topic/hebbian-theory) nature of their training algorithm, as well as their parallelism and the resemblance of their dynamics to simple physical processes. If the connectivity is constrained, the learning can be made efficient enough to be useful for practical problems.

***Structure***

A Boltzmann machine, like a [Hopfield network](http://www.answers.com/topic/hopfield-net), is a network of units with an "energy" defined for the network. It also has [binary](http://en.wiktionary.org/wiki/binary) units, but unlike Hopfield nets, Boltzmann machine units are[stochastic](http://www.answers.com/topic/stochastic). The global energy, *E*, in a Boltzmann machine is identical in form to that of a Hopfield network:

E = -\sum_{i<j} w_{ij} \, s_i \, s_j + \sum_i \theta_i \, s_i

Where:

* *wij* is the connection strength between unit *j* and unit *i*.
* *si* is the state, s_i \in \{0,1\}, of unit *i*.
* θ*i* is the [threshold](http://www.answers.com/topic/threshold) of unit *i*.

The connections in a Boltzmann machine have two restrictions:

* w_{ii}=0 \qquad \forall i. (No unit has a connection with itself.)
* w_{ij}=w_{ji}\qquad \forall i,j. (All connections are [symmetric](http://www.answers.com/topic/symmetry).)

Thus, the difference in the global energy that results from a single unit *i* being 0 versus 1, writtenΔ*Ei*, is given by:

\Delta E_i = \sum_j w_{ij} \, s_j - \theta_i

A Boltzmann machine is made up of stochastic units. The probability, *pi* of the *i*-th unit being on is given by:

p_i = \frac{1}{1+\exp (-\frac{1}{T} \Delta E_i)}

where the [scalar](http://www.answers.com/topic/scalar-physics) *T* is referred to as the [temperature](http://www.answers.com/topic/temperature) of the system.

The network is run by repeatedly choosing a unit and setting its state according to the above formula. After running for long enough at a certain temperature, the probability of a global state of the network will depend only upon that global state's energy, according to a [Boltzmann distribution](http://www.answers.com/topic/boltzmann-distribution). This means that log-probabilities of global states become linear in their energies. This relationship is true when the machine is "at [thermal equilibrium](http://www.answers.com/topic/thermodynamic-equilibrium)", meaning that the probability distribution of global states has converged. If we start running the network from a high temperature, and gradually decrease it until we reach a [thermal equilibrium](http://www.answers.com/topic/thermodynamic-equilibrium) at a low temperature, we are guaranteed to converge to a distribution where the energy level fluctuates around the global minimum. This process is called [simulated annealing](http://www.answers.com/topic/simulated-annealing).

If we want to train the network so that the chance it will converge to a global state is according to an external distribution that we have over these states, we need to set the weights so that the global states with the highest probabilities will get the lowest energies. This is done by the following training procedure.

***Training***

The units in the Boltzmann Machine are divided into "visible" units, V, and "hidden" units, H. The visible units are those which receive information from the "environment", i.e. our training set is a set of binary vectors over the set V. The distribution over the training set is denoted *P* + (*V*).

On the Boltzmann Machine side, as recalled, the distribution over the global states is converging as we reach a [thermal equilibrium](http://www.answers.com/topic/thermodynamic-equilibrium). We denote the converged distribution, after we marginalize it over the visible units *V*, as *P* − (*V*).

Our goal is to approximate the "real" distribution *P* + (*V*) using the *P* − (*V*) which will be produced (eventually) by the machine. To measure how similar the two distributions are we use the[Kullback-Leibler divergence](http://www.answers.com/topic/kullback-leibler-divergence-3), *G*:

G = \sum_{v}{P^{+}(v)\ln{\frac{P^{+}(v)}{P^{-}(v)}}}

Where the sum is over all the possible states of *V*. *G* is a function of the weights, since they determine the energy of a state, and the energy determines *P* − (*v*), as promised by the[Boltzmann distribution](http://www.answers.com/topic/boltzmann-distribution). Hence, we can use a [gradient descent](http://www.answers.com/topic/gradient-descent) algorithm over *G*, so a given weight, *wij* is changed by subtracting the [partial derivative](http://www.answers.com/topic/partial-derivative) of *G* with respect to the weight.

There are two phases to Boltzmann machine training, and we switch iteratively between them. One is the "positive" phase where the visible units' states are clamped to a particular binary state vector sampled from the training set (according to *P* + ). The other is the "negative" phase where the network is allowed to run freely, i.e. no units have their state determined by external data. Surprisingly enough, the gradient with respect to a given weight, *wij*, is given by the very simple equation (proved in Ackley et al.):

\frac{\partial{G}}{\partial{w_{ij}}} = -\frac{1}{T}[p_{ij}^{+}-p_{ij}^{-}]

Where:

* p_{ij}^{+} is the probability of units *i* and *j* both being on when the machine is at equilibrium on the positive phase.
* p_{ij}^{-} is the probability of units *i* and *j* both being on when the machine is at equilibrium on the negative phase.

This result follows from the fact that at the [thermal equilibrium](http://www.answers.com/topic/thermodynamic-equilibrium) the probability *P* − (*s*) of any global state *s* when the network is free-running is given by the [Boltzmann distribution](http://www.answers.com/topic/boltzmann-distribution) (hence the name "Boltzmann machine").

Remarkably, this learning rule is fairly biologically plausible because the only information needed to change the weights is provided by "local" information. That is, the connection (or [synapse](http://www.answers.com/topic/chemical-synapse)biologically speaking) does not need information about anything other than the two neurons it connects. This is far more biologically realistic than the information needed by a connection in many other neural network training algorithms, such as [backpropagation](http://www.answers.com/topic/backpropagation).

The training of a Boltzmann machine does not use the [EM algorithm](http://www.answers.com/topic/expectation-maximization-algorithm), which is heavily used in[machine learning](http://www.answers.com/topic/machine-learning). By minimizing the KL-divergence, it is equivalent as maximizing the log-likelihood of the data. Therefore, the training procedure performs gradient ascent on the log-likelihood of the observed data. This is in contrast to the EM algorithm, where the posterior distribution of the hidden nodes must be calculated before the maximization of the expected value of the complete data likelihood during the M-step.

Training the biases is similar, but uses only single node activity:

\frac{\partial{G}}{\partial{\theta_{i}}} = -\frac{1}{T}[p_{i}^{+}-p_{i}^{-}]

***Problems***

The Boltzmann machine would be a rather general computational medium. For instance, if trained on photographs, the machine would model the distribution of photographs, and could use that model to, for example, complete a partial photograph.

Unfortunately, there is a serious practical problem with the Boltzmann machine, namely that the learning seems to stop working correctly when the machine is scaled up to anything larger than a trivial machine. This is due to a number of effects, the most important of which are:

* the time the machine must be run in order to collect equilibrium statistics grows exponentially with the machine's size, and with the magnitude of the connection strengths
* connection strengths are more plastic when the units being connected have activation probabilities intermediate between zero and one, leading to a so-called variance trap. The net effect is that noise causes the connection strengths to random walk until the activities saturate.