Boltzmann Machines

Deep Learning models are broadly classified into supervised and unsupervised models.

Supervised DL models:

Artificial Neural Networks (ANNs)

Recurrent Neural Networks (RNNs)

Convolutional Neural Networks (CNNs)

Unsupervised DL models:

Self Organising Maps (SOMs)

Boltzmann Machines

Autoencoders

Boltzmann Machines is an **unsupervised DL model** in which **every node is connected to every other node.**

That is, **unlike the ANNs, CNNs, RNNs and SOMs, the Boltzmann Machines are undirected (or the connections are bidirectional).**

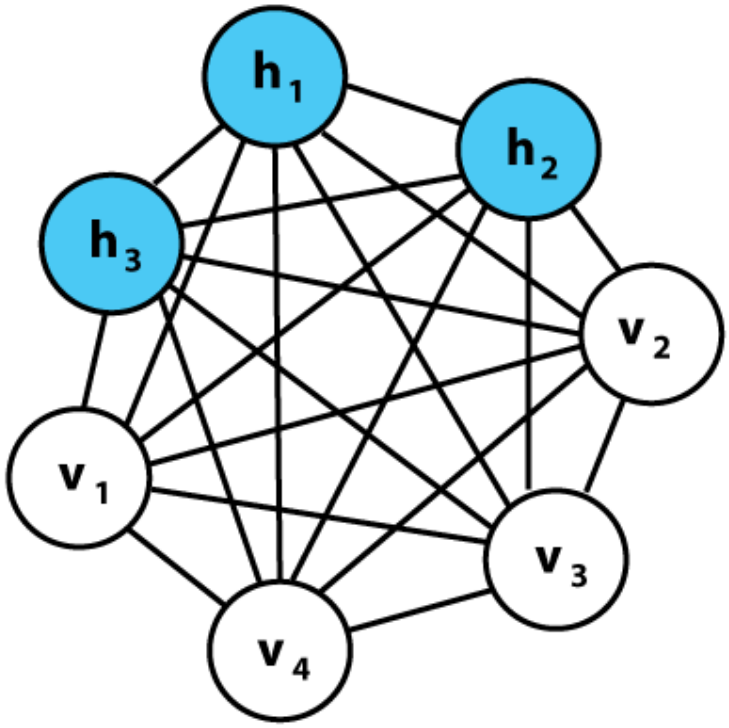
Boltzmann Machine is not a deterministic DL model but a **stochastic or generative DL model**.

It is rather a representation of a certain system.

There are **two types of nodes in the Boltzmann Machine**

— **Visible nodes** — **those nodes which we can and do measure**, and

the Hidden nodes – **those nodes which we cannot or do not measure**.



A **Boltzmann Machine is a network of symmetrically connected**, neuron like units that make **stochastic decisions about whether to be on or off**.

Boltzmann machines have a simple learning algorithm that **allows them to discover interesting features in datasets composed of binary vectors**.

The **learning algorithm is very slow in networks with many layers of feature detectors**, but it **can be made much faster by learning one layer of feature detectors** at a time.

Boltzmann machines are **used to solve two quite different computational** problems.

For a **search problem, the weights on the connections are fixed** and are **used to represent the cost function of an optimization problem**.

The **stochastic dynamics of a Boltzmann machine then allow** it to sample binary s**tate vectors that represent good solutions to the optimization problem**.

For a learning problem, the Boltzmann machine is shown a set of binary data vectors and it must **find weights on the connections so that the data vectors are good solutions to the optimization problem** defined by those weights.

To **solve a learning problem, Boltzmann machines make many small updates to their weights, and each update requires them to solve many different search** problems.

The stochastic dynamics of a Boltzmann machine

$$z_i = b_i + \sum_j s_j w_{ij}$$

When unit i is **given the opportunity to update its binary state**, it **first computes its total input, zi** , **which is the sum of its own bias, bi** , and the **weights on connections coming from other active units**:

where **wij is the weight on the connection between i and j**, and sj is 1 if unit j is on and 0 otherwise. Unit i then turns on with a probability given by the logistic function:

$$prob(s_i = 1) \quad = \quad \frac{1}{1 + e^{-z_i}}$$

If the units are updated sequentially in any order that does not depend on their total inputs, the network will eventually reach a Boltzmann distribution (also called its equilibrium or stationary distribution) in which the probability of a state vector, v, is determined solely by the "energy" of that state vector relative to the energies of all possible binary state vectors:

$$P(\mathbf{v}) = e^{-E(\mathbf{v})} / \sum_{\mathbf{u}} e^{-E(\mathbf{u})}$$

As in Hopfield nets, the energy of state vector v is defined as

$$E(\mathbf{v}) = -\sum_i s_i^{\mathbf{v}} b_i - \sum_{i<j} s_i^{\mathbf{v}} s_j^{\mathbf{v}} w_{ij}$$

where svi is the binary state assigned to unit i by state vector v.

If the weights on the connections are chosen so that the energies of state vectors represent the badness of those state vectors as solutions to an optimization problem, then the stochastic dynamics of a Boltzmann machine can be viewed as a way of escaping from poor local optima while searching for good solutions.

The total input to unit i, zi , represents the difference in energy depending on whether that unit is off or on, and the fact that unit i occasionally turns on even if zi is negative means that the energy can occasionally increase during the search, thus allowing the search to jump over energy barriers.

Learning in Boltzmann Machines

Given a training set of state vectors (the data), learning consists of finding weights and biases (the parameters) that make those state vectors good.

More specifically, the aim is to find weights and biases that define a Boltzmann distribution in which the training vectors have high probability.

$$\sum_{\mathbf{v}\in\text{data}} \frac{\partial \log P(\mathbf{v})}{\partial w_{ij}} = \langle s_i s_j \rangle_{\text{data}} - \langle s_i s_j \rangle_{\text{model}}$$

where **sisj** i data is the expected value of **sisj** in the data distribution and hsisj imodel is the expected value when the Boltzmann machine is sampling state vectors from its equilibrium distribution at a temperature of 1

To perform **gradient ascent in the log probability** that the Boltzmann machine would generate the observed data when sampling from its equilibrium distribution, wij is incremented by a small learning rate times the RHS of the above equation.

To perform gradient ascent in the log probability that the Boltzmann machine would generate the observed data when sampling from its equilibrium distribution, wij is incremented by a small learning rate times the RHS

If the observed data specifies a binary state for every unit in the Boltzmann machine, the learning problem is convex: There are no non-global optima in the parameter space. Learning becomes much more interesting if the Boltzmann machine consists of some “visible” units, whose states can be observed, and some “hidden” units whose states are not specified by the observed data.

The hidden units act as latent variables (features) that allow the Boltzmann machine to model distributions over visible state vectors that cannot be modelled by direct pairwise interactions between the visible units.

A surprising property of Boltzmann machines is that, even with hidden units, the learning rule remains unchanged. This makes it possible to learn binary features that capture higher-order structure in the data

With hidden units, the expectation hsisj i data is the average, over all data vectors, of the expected value of sisj when a data vector is clamped on the visible units and the hidden units are repeatedly

updated until they reach equilibrium with the clamped data vector.

It is surprising that the learning rule is so simple because $\partial \log P(v)/\partial w_{ij}$ depends on all the other weights in the network.

Fortunately, the difference in the two correlations in above equation tells wij everthing it needs to know about the other weights.

This makes it unnecessary to explicitly propagate error derivatives, as in the backpropagation algorithm.