Critique of Boltzmann Machines – D. Ackley & G. Hinton

Massively parallel networks can be utilized to communicate instances of a problem with a fraction of the knowledge of the system. This computational capacity is determined by the communication bandwidth between the elements of the hardware. For such systems to exist, two conditions must be satisfied:

1. A search technique that is suitable for parallel network &
2. A way of choosing internal representations for using the hardware connections efficiently to encode the constraints in the domain.

This paper focusses on satisfying large number of weak constraints (like image representation/encoding) that can be broken at some cost (Energy). Ultimately, these networks are generative models that can regenerate from just a partial input.

The Hopfield Networks had provided some substantial result in encoding these internal representations, but a huge setback was that they get stuck at local minima while determining the desired representations. They characterize a network of binary units interconnected by the weights which ultimately reflect the input representations of a given problem.

To overcome the problem of the network getting stuck in local minima, Gef. Hinton proposed stochastic binary units, which by the addition of some noise skip the local minima. The structure, learning and computations are discussed below:

STRUCTURE

- It is an interconnected network of visible and hidden units.
- The units can either be 0(off) or 1(on).
- The units are stochastically binary (explained later).
- Visible units(v) can be clamped i.e., can be given input.
- Hidden units(h) are never clamped.
- Hidden units are utilized to interpret complex constraints between these input units.

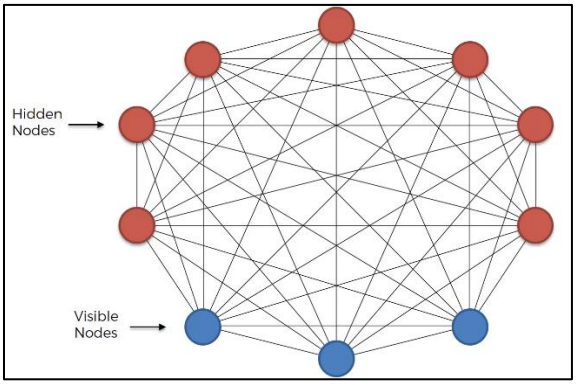

Figure 1: Boltzmann Machine

- The weights (in *Fig. 2*) between the units reflect their correlation, i.e., a positive value implies the units support each and a negative implies vice versa.
- The weights are symmetric between two units.
- The cost which happens as a result of violating the constraints is given by the Global State Energy as:

$$\boldsymbol{E = -\sum_{i<j} w_{ij} s_i s_j} \qquad \boldsymbol{-(1)}$$

  - Here, the s_i, s_j are two units (hypothesis) and w_{ij} is the weight between them.
  - Hence, the total Energy is the sum of the product of the weights, given that the pair of respective states are 1.
  - It must be noted that this also includes a threshold unit θ_i that is absorbed into the equation by adding an imaginary unit 1.
- The difference between the Energies of a unit when they are on and off is called the energy gap given by:

$$\boldsymbol{\Delta E_k = \sum_{i} w_{ki} s_i} \qquad \boldsymbol{-(2)}$$

  - Here it is the sum of all the weights combined with the counterpart weights input to unit k.
- Below is an illustration of the same.

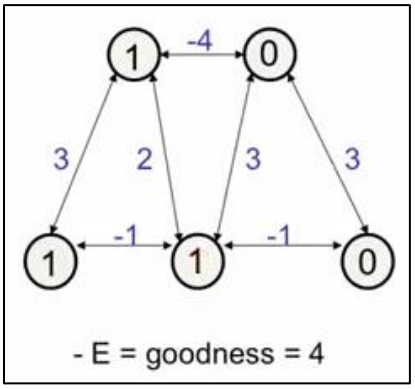

Figure 2: Boltzmann Machine with Constraint Satisfaction

There are two major tasks involved in successfully regenerating a desired problem (set of states) in Boltzmann Machines (BM):

1. Training the weights so that they represent the constraints placed by a problem and thereby accommodate all the input states/configurations placed to the network which results in a unique probability distribution that can generate the desired states.
2. Achieving, global minima of Energy by stochastically manipulating the units so that the resulting network reduces to the desired configuration (Either while clamped or during free run).

We shall start with the explanation and algorithm of the 2nd task first.

Algorithm (Hopfield Net – Asynchronous Update):

1. ***Repeat:***

2. *Randomly select a unit i.*
3. *Compute ΔE_k. If $+ve$, set $s_k = 1$, else 0.*
4. ***Until** minimum(local) energy E is reached.*

The problem with this algorithm is that it gets stuck at local minima. Hence, Gef. Cleverly averted it by adding noise (see *Fig. 3*). This is created by a version of the Metropolis algorithm which generates Boltzmann distribution.

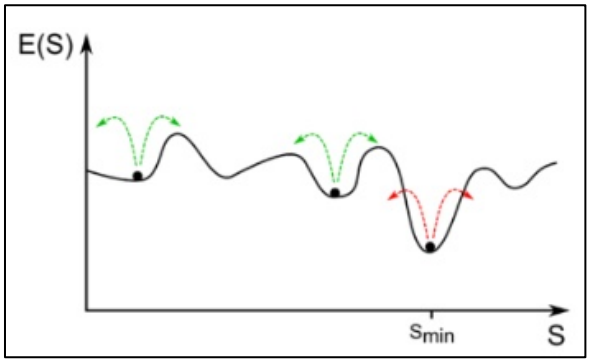

Figure 3: Escaping local minima by adding noise that is governed by the Boltzmann Distribution

According to this, instead of deterministically setting the units 1 or 0 based on the energy gap, set them to 1 probabilistically based on the energy gap as:

$$\boldsymbol{p_k(on) = \frac{1}{1 + e^{-\frac{\Delta E_k}{T}}}} \qquad \boldsymbol{-(3)}$$

- This is a sigmoid function which is largely inspired by the Boltzmann Distribution with temperature T, which is a controlling factor for "Thermal Equilibrium".

Few concepts to be absorbed here:

1. The Boltzmann distribution is the pdf that is achieved by the particles at "Thermal Equilibrium".
2. So, at "Thermal Equilibrium", all the states/particles adhere to the Boltzmann Factor (relative probability of two states) as:

$$\boldsymbol{\frac{P_\alpha}{P_\beta} = e^{-\frac{E_\alpha - E_\beta}{T}}} \qquad \boldsymbol{-(4)}$$

3. So, it must be understood that by constantly applying $eq.(3)$ on the units for setting on or off, the global states achieve "Thermal Equilibrium" which satisfy $eq.(4)$ for a given $\boldsymbol{T}$.
4. The temperature $\boldsymbol{P \propto T \propto \frac{1}{E}}$, i.e., at lower temperatures, for probability to be high, energy must be low (sensitive to small energy gaps) and at higher temperature it is less sensitive to the energy gap.
5. Hence, we start from high temperatures (can have many local minima so high noise) and move towards lower temperature (lower noise) for achieving global minima of Energy. This procedure is called Simulated Annealing.

Therefore, the updated algorithm is as below:

Algorithm for Thermal Equilibrium (Boltzmann Machine):

1. *Set high temperature T.*
2. ***Repeat:***
3. *Randomly select a unit i.*
4. *Compute ΔE_k.*
5. *Set $s_k = 1$ with the probability of $eq.(3)$*
6. ***Until** Thermal Equilibrium is reached*
7. ***Repeat:** Reduce T (A few iterations – Simulated Annealing).*
8. *Repeat Steps 2 to 6.*

Now, we have established an algorithm for finding the complete configuration of the Boltzmann machine given an input to the visible states.

All through this process we had assumed that we know the weights W. But now we shall train the network to obtain W (can be considered as memory or an encode of constraints).

The Learning Algorithm:

The relation between the log probabilities and the weights can be stated as below, as Energy is a linear function of the weights:

$$\frac{\partial \ln \mathbf{P_\alpha}}{\partial \mathbf{w_{ij}}} = \frac{\mathbf{1}}{\mathbf{T}}\left[\mathbf{s_i^\alpha s_j^\alpha} - \mathbf{p'_{ij}}\right] \qquad -(\mathbf{5})$$

s_i^α, s_j^α are the states of the i^{th} and j^{th} units.

p'_{ij} the probability of the states s_i^α & s_j^αbe 1.

The above equation can be applied to each state $\boldsymbol{\alpha}$ and converging to a set of weights of the BM, if each of the probabilities $\mathbf{P_\alpha}$ is given. However, it is impractical to provide each of the probabilities. Instead, we do as below:

1. ***Training Algorithm:***
   a. *Initialize the weights to 0.*
   b. *Repeat:*
   c. *Clamp the visible node with each input vector α and run until achieving thermal equilibrium to obtain a probability distribution $P(V_\alpha)$.*
   d. *Now, allow the network to run freely (without any clamping) to attain thermal equilibrium and hence get distribution $P'(V_\alpha)$.*
   e. *Compute the divergence (a measure of the distance between two probability distributions),*

$$\boldsymbol{G} = \sum_{\boldsymbol{\alpha}} \boldsymbol{P(V_\alpha)}\ \boldsymbol{ln}\left(\frac{\boldsymbol{P(V_\alpha)}}{\boldsymbol{P'(V_\alpha)}}\right) \qquad -(\mathbf{6})$$

   f. *Since G is dependent on probability P and P on energy E which is in turn dependent on weights w, and also since their relationships are linear, we can directly compute the gradient descent for each w as:*

$$\frac{\partial G}{\partial w_{ij}} = -\frac{1}{T}(p_{ij} - p'_{ij})$$

$$\Rightarrow \Delta w_{ij} = \epsilon(p_{ij} - p'_{ij}) \qquad -(7)$$

- p_{ij} *is the probability of unit* i *being on when clamped, &*
- p'_{ij} *is the probability of unit* i *being on when run freely.*
- ϵ *is the learning rate.*

g. Update w.

2. *Hence, when G=0(ideally) or very low, we attain a model which represents the given problem set at "Thermal Equilibrium".*
3. *"Noisy clamping" is done to avoid unwanted patterns over visible units with limited success.*

ENCODER PROBLEM:

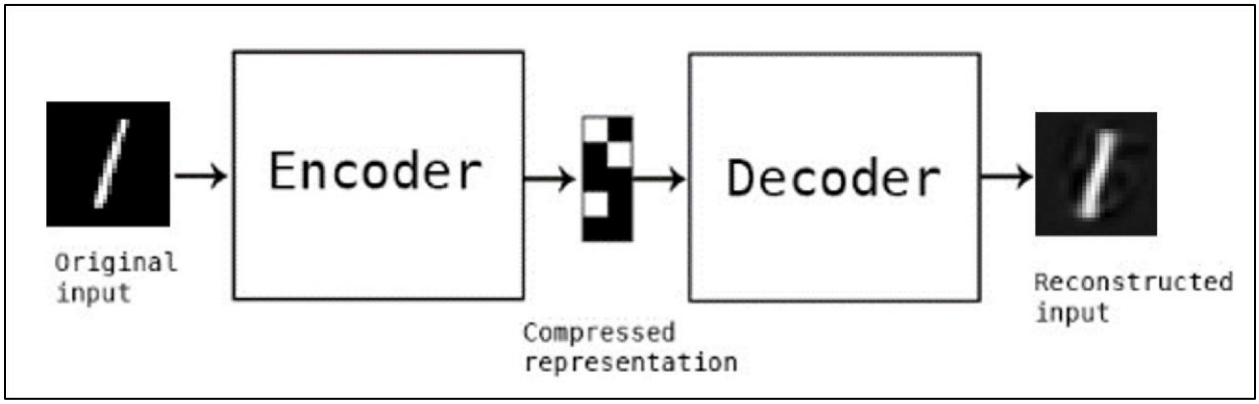

Figure 4: Image is encoded into small number of bits

According to information theory of encoding, $h < v$. For lossless communication between the visible groups, $h \geq \log v$. The minimal case must be $h = \log v$.

Consider two groups of visible units, V1 and V2 connected through hidden units H.

Let's assume a 4-2-4 encoder respectively. The training is very similar to the procedure described above. It interestingly contains 3 phases:

1. Weights are zero and update to negative values. Using **winner take all approach**, where the units express lateral inhibitions, only one node is made to be active for one input.
2. The weights of all the hidden units must become positive and also develop symmetry in weights between corresponding units in V1 and V2.
3. Sorts out all the conflicts where a code is exploited multiple times while some are not used.

The median time required to discover four different codes in 4-2-4 was 110 learning cycles.

Similar experiments were conducted with 4-3-4 and 8-3-8 encoders with extremely good results.

Although the 40-10-40 encoder uses twice the theoretical minimum of hidden units, the bandwidth is still limited.

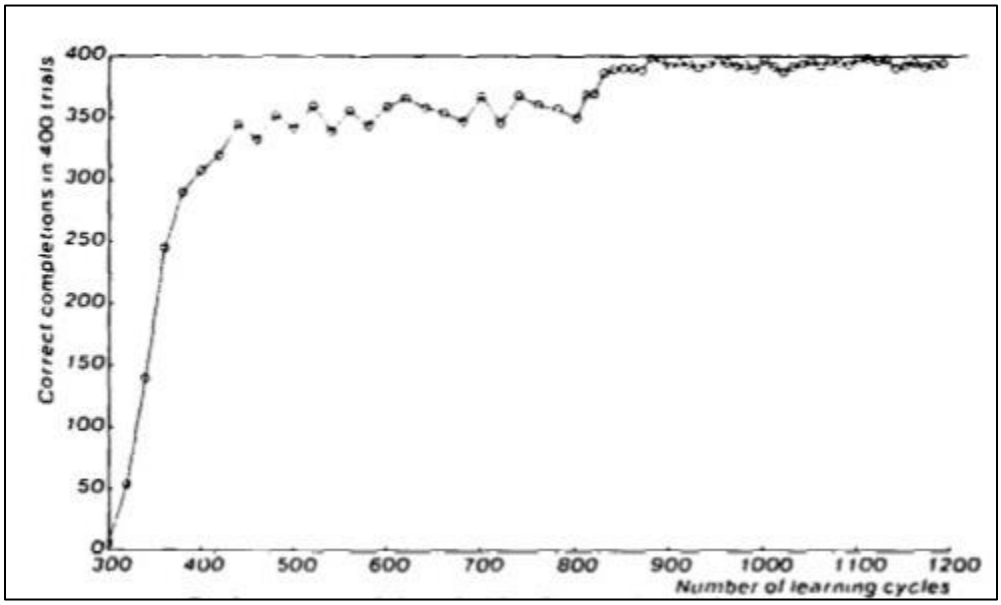

Figure 5: Performance of the 40-10-40 encoder with final asymptotes at 98.6% correctness

Further, the author has briefly discussed the application of the encoder in communication systems.

The clever exploitation of statistical mechanics to constraint satisfaction searches in massive parallel networks, has yielded credible results. Similar systems can also be built using Gaussian Distribution.

References:

1. Hopfield nets and Boltzmann machines – Neural Networks for Machine Learning Geoffrey Hinton
2. Boltzmann Machines
3. Slides - Brief Introduction to Boltzmann Machine
4. Energy Based Models (EBM)
5. Maxwell-Boltzmann Distribution generated by Metropolis Monte Carlo Simulation