7. Boltzmann machines

In this section we will become acquainted with classical Boltzmann machines which can be seen obsolete being rarely applied in neurocomputing. It is interesting, after all, because is one of the fundamental algorithms of deep learning subject to the deep belief neural network.

Hinton and Sejnowski (1985) introduced Boltzmann machines, but this unsupervised neural network type remained infrequently utilised until recently. A special type of Boltzmann machines, the restricted Boltzmann machine, is important for deep belief networks.
7.1 Basis for Boltzmann machine

A *Boltzmann machine* is a fully connected two-layer neural network. Its layers are called input and output. Nonetheless, the names of layers are sometimes used differently by calling them visual and hidden layers or by dividing them into not two, but three types: input, hidden and output. These are possible, since the basic form of Boltzmann machine is the fully connected graph. The Boltzmann machine to be presented has no hidden layer between the input and output layers (Fig. 7.1). The network is fully connected, but its nodes are not connected to themselves (no self-loops). Nevertheless, a particular type, i.e., restricted Boltzmann machine is interesting (Fig. 7.2).

In the following, we start from the basic form on the basis of Fig. 7.1. Yet, the version of Fig. 7.2. is nowadays utilised in modern deep learning.
Fig. 7.1 Fully connected Boltzmann machine: In is for input nodes and Ou for output nodes.
Fig. 7.2 Restricted Boltzmann machine: all output nodes are connected to each input node, whereas there are neither connections among output nor input nodes.
Boltzmann machine’s nodes acquire only binary states, either 0 or 1. (Some research has been made for real values, but nearly all of their use centres on binary units.)

Boltzmann machines can be stacked to form layers.

Boltzmann machines have their origins in Hopfield networks presented in the early 1980s. The main problem of Hopfield network is that it settles into local minima by constant energy minimisation when we seek the globally optimal state of the network. As early as 1953, Metropolis et al. proposed an algorithm for the simulation of a solid to thermal equilibrium.
The basic idea behind *simulated annealing* is that a metal is gradually cooled, to lower and lower energy states and eventually the lowest possible energy state is reached. Kirkpatrick et al. (1983) realised that there exists an analogy between minimising the error function of a combinatorial optimisation problem and the slow cooling of a metal. Hinton and Sejnowski (1986) combined Hopfield networks and simulated annealing to result in networks known as Boltzmann machines. In this technique, local minima are avoided by adding some randomness or noise to the process so that when the process of the network moves toward a local minimum, it has a chance to escape like a moving ball that might jump over a bump in Fig. 7.3.
Fig. 7.3 A random noise or small additional energy in an appropriate direction may aid the process (ball) to move from a local minimum.
Each node in the network computes which state it should switch into to reduce the energy†, but instead of just switching (like in Hopfield network), it changes to that state depending on the value of the probability function. Sometimes the network does not switch into a lower energy state, but allows jumps to be made into higher energy states, and it is this feature that allows local minima to be escaped. The probability function is chosen so that if the node achieves a great reduction in the overall energy by changing its state, then it will probably be allowed to change, but if there is not a great deal to be gained either way, the likelihood of it changing is much more uncertain or less. It has a parameter $T$ to vary its ”temperature”†.

† In Boltzmann machines the term energy is a measure used to map the process, and temperature is a control parameter of the network.
7.2 Boltzmann machine probability

When the "temperature" parameter is lowered, the probability of assuming the correct low energy state approaches to 1 ("switched on" or "active"), and the network is said to have reached thermal equilibrium.

This can be expressed as follows. Every node $k$ in the network computes an energy gap related to states $s_i$ of its connected nodes $i$ (with weights and threshold)

$$\Delta E_k = \sum_i w_{ki}s_i - \theta_k$$ (7.1)

and switches into the state 1 that is of lower energy according to the probabilistic update rule, i.e. with probability
When Boltzmann machine queries the value of 1 of output nodes, it will randomly produce 0 or 1.

The network can settle into one of a perhaps large number of global energy states, the distribution of which is given by Boltzmann distribution. If we let \( P_k \) be the probability of the network settling into some global energy state of energy \( E_k \), then Boltzmann distribution has the form

\[
p_k = \frac{1}{1 + e^{-\Delta E_k/T}} \tag{7.2}
\]

which depends on the energy of the state and temperature \( T \) of the system.

\[
P_\alpha = e^{-E_\alpha/T} \tag{7.3}
\]
Calling $P_\beta$ to be the probability of a state with energy $E_\beta$, we can write

$$\frac{P_\alpha}{P_\beta} = \frac{e^{-E_\alpha/T}}{e^{-E_\beta/T}} = e^{-(E_\alpha - E_\beta)/T} \tag{7.4}$$

The network is allowed to settle into thermal equilibrium, when the probabilities of states no longer alter, and are dependent on their energy.
If $E_\alpha$ is a lower energy state than $E_\beta$, then we can obtain

\[
E_\alpha < E_\beta \\
e^{-\frac{(E_\alpha - E_\beta)}{T}} > 1 \\
\text{therefore } P_\alpha / P_\beta > 1 \\
\text{so } P_\alpha > P_\beta
\]

(7.5)

This means that the network approaches thermal equilibrium, lower energy states are more probable, dependent only on their relative energy.
At high temperatures, the network reaches equilibrium quickly, but good global energy states are not much more probable to occur than poor ones. Reducing the temperature while the network is running is of simulated annealing. The high temperatures enable local minima states to be avoided via higher energy states, but also allow transitions from lower minima to higher ones with almost equal probability. Ultimately the system settles down at low temperature in thermal equilibrium. The output probabilities of states become constant, not necessarily the values of the states themselves.

The effect of the temperature on the probabilistic function governing the chance of a node changing state is shown in Fig. 7.4.
Fig. 7.4 The effect of temperature on the transition probability function (7.2.) on p. 236. The probability of a transition to a higher energy state is greater at high temperatures than is at lower ones. Here threshold $\Theta$ equals 0.
The description of simulated annealing is, in fact, an oversimplification. Since the energy landscape is a high-dimensional space, the energy barrier between states is usually very degenerate. This means that there are several ways of passing from one state to another. It is even more likely that the system will move into the lower energy state.

The temperature alteration is produced by adjusting the steepness $1/T$ of the sigmoid function from Eq. (7.2), which effectively determines the probability that a node will actually go into its natural, or non-noisy, state. If the node exceeds threshold $\Theta$ by a large amount, then it will always attain value 1. Whereas if it is far enough below the threshold, it will always have value 0. For the threshold, the probability is at 0.5. (Fig. 7.4)
The function described in Fig. 7.4 follows the Boltzmann distribution used in statistical mechanics.

At high temperatures, the network moves into high energy states easily, and the overall energy of the system is high. At the other extreme, low temperatures mean that transitions to higher energy states are very rare. See Fig. 7.5. Therefore, the duration of transition between these two should be fairly long that transitions could have enough time to occur. On the other hand, to design this is not simple.
Fig. 7.5 Average energy of the network. The phase transition is shown where the average energy of the system falls quickly for a small reduction of the temperature.
Learning occurs in two phases in Boltzmann machines. An arbitrary choice is made as to which nodes are to be the input and output nodes. First, the input and output nodes are \textit{clamped to their correct values}. The network is then allowed to cycle through its states when the temperature is gradually lowered until thermal equilibrium. Second, only the input nodes are \textit{clamped to their correct values}. The network is again run as before. Weights are incremented according to the first phase and decremented according to the second one. Each weight is updated by

\[ \Delta w_{ij} = \eta (P_i^+ - P_i^-) \]  

(7.6)
where $\eta$ is a small positive constant (learning rate), $P_{ij}^+$ is the averaged probability over such runs that nodes $i$ and $j$ are both active 1 at thermal equilibrium when the input and output are both clamped, and $P_{ij}^-$ is the same probability when only input is clamped.
Boltzmann Machine algorithm

- Initialise weights with small (positive) random values. Choose input and output nodes and give their initial values, either 0 or 1.
- Calculate activation for the first phase and then for the second one as follows.
  1. Select an initial temperature.
  2. Compute the energy of state for $N$ nodes

$$\Delta E_k = \sum_{i} w_{ki} s_i - \theta_k, \; k = 0, \ldots, N - 1$$

Until thermal equilibrium, repeatedly compute the probability that node $k$ is active (adopting the state 1) by
where $\Delta E_k$ is the total input received by node $k$ and $T$ is the temperature. The activation level (state) of node $k$ is set according to this probability.

3. If the lowest temperature has been reached, then exit and the pattern of the activations upon equilibrium represents the optimised solution. Otherwise, reduce the temperature by a certain annealing schedule and repeat Step 2.

• Weight training

Each weight is trained by

$$\Delta w_{ij} = \eta \left( p^+_{ij} - p^-_{ij} \right)$$
7.4 Applying Boltzmann machines

The unrestricted form of Boltzmann machines presented above has been applied to optimisation and recognition problems. Let us browse through the travelling salesman problem (TSP), a classic one in computer science. This is interesting in algorithmics and artificial intelligence and is a basis for many practical computational tasks.

In the travelling salesman problem an algorithm is designed to determine the order of a fixed set of cities that minimises the total distance covered. TSP is a combinatorial problem.

TSP involves determining the shortest path for a travelling salesman (a metaphor, could be, e.g., a lorry, cab, bus or data in the Internet).
A travelling salesman must visit a certain number of cities, beginning one of them, then continuing one by one through all other cities never entering twice the same city. Nevertheless, whatever path is not valid, but the sum of the distances on the route from a city to another has to be the minimum of all possible paths. TSP has many variants, but the above-mentioned is its basic definition. See Fig. 7.6.

Finding the shortest path may seem like an easy run-through. However, as the number of cities increases, the number of different paths, i.e., permutations increases drastically.
Fig. 7.6 There are \( n \) cities that the travelling salesman has to visit one by one. Which order is the best, in other words, forms the shortest path?
Their number is equal to factorial

\[ n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 2 \cdot 1 = n! \]

in which \( n \) is the number of cities. This cannot be solved by a brute-force algorithm on the basis of exhaustive search, because it is a non-deterministic polynomial time (NP) hard problem. For instance, if \( n \) is 10, its factorial is already 3,628,800. For 12 cities or nodes in a graph, there exist more than 479 million different paths.
Thus, the worst case time complexity of the TSP is $O(n!)$. Algorithms based on the algorithmic design principle called dynamic programming can be employed and the complexity be bounded to $O(n^22^n)$. Dynamic programming breaks a large problem into smaller subproblems and reuses the results of already solved subproblems when there are several identical subproblems. Still, the exponential complexity is huge.

Algorithms based on the use of randomness are effective to find an operable solution for TSP. The objective is to find an approximative solution, which is close to the global minimum, the shortest path of all. In that case, only a fraction of all permutations are computed. Boltzmann machines based on the use of probabilistics are an alternative to compute TSP.