BOLTZMANN MACHINES

As modern computers become ever more powerful, engineers continue to be challenged to use machines effectively for tasks that are relatively simple for humans, but difficult for traditional problem-solving techniques. Artificial neural networks, inspired by biological systems, provide computational methods that can be utilized in many engineering disciplines. Following a brief overview of the features that characterize neural networks in general, we consider the neural networks known as Boltzmann machines.

Fixed-weight Boltzmann machines are used for constrained optimization problems, such as those arising in scheduling, management science, and graph theory. They are applied to intractible NP-complete problems to rapidly locate near-optimal solutions. Three constrained optimization problems, the traveling salesman, asset allocation, and scheduling problems, are considered below. Other problems of this type

include maximum cut, independent set, graph coloring, clique partitioning, and clique covering problems (1). A second type of Boltzmann machine is used for input-output mapping problems such as the encoder, seven-segment display, and XOR problems.

OVERVIEW OF NEURAL NETWORKS

Neural networks consist of many simple processing elements, called neurons or units, which are connected by weighted pathways. The neurons communicate with each other by sending signals over these paths. Each neuron processes the input signals that it receives to compute its activation, which becomes the signal that the neuron sends to other units. The weights on the pathways may be fixed when the network is designed or when it is trained using examples. Fixed-weight networks are used for constrained optimization problems, and adaptive weights are used for pattern classification and general mapping networks. After training, the neural network is able to recognize an input pattern that is similar to, but not exactly the same as, one of the training patterns.

Neural Network Architectures

The pattern of connections among the neurons is called the *neural network architecture.* A simple feed-forward neural network, in which the signals flow from left to right, is illustrated in Fig. 1(a). Recurrent neural networks have feedback connections, such as the connection from unit Y_2 back to unit X_3 in Fig. 1(b).

Neural Network Operation

In a typical neural network, the signal transmitted over a connection pathway is multiplied by the weight on the path. The neuron first sums the incoming signals and then processes this sum (its net-input) to determine the signal it will transmit. In many neural networks this output signal is a nonlinear function of the net input, with a range of 0 to 1 (or -1 to 1). For example, for the neural network in Fig. 1(a), the output signal from unit Y_1 could be expressed as

$$Y_1 = f(x_1 w_{11} + x_2 w_{21} + x_3 w_{31})$$

for a suitable nonlinear function f. See Refs. 2 and 3 for further discussion of neural networks.

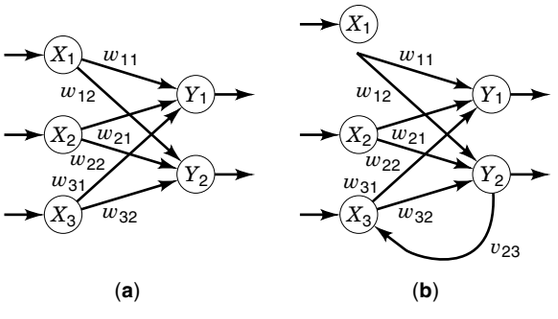

Figure 1. Simple neural networks with no hidden nodes.

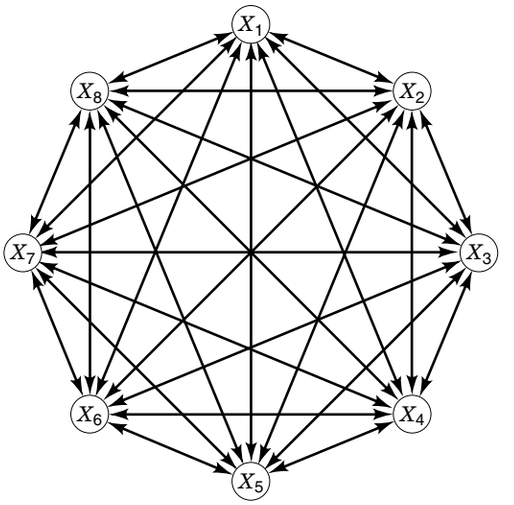

Figure 2. Fully interconnected neural network.

BOLTZMANN MACHINES

Boltzmann machines are neural networks in which the units can have only two states; the present discussion is limited to the case of binary output, that is, a unit is either off (output is 0) or on (output is 1). Furthermore, the net-input does not *determine* the output value, but only the *probability* of each output value. The massive parallelism of neural networks in general, and Boltzmann machines in particular, provides a promising approach for computers of the future.

Architecture

The architecture of a Boltzmann machine is very general. The neurons may be fully interconnected, as illustrated in Fig. 2, or only partially interconnected, as shown in Fig. 3. However, the connection pathways are always bidirectional. In other words, if neuron X_i is connected to neuron X_j, with weight w_{ij}, then X_j is also connected to X_i and the connection has the same weight (i.e., $w_{ji} = w_{ij}$).

Using a Boltzmann Machine

In recurrent neural networks, the activations of the neurons evolve in such a way that the equilibrium configuration (pat-

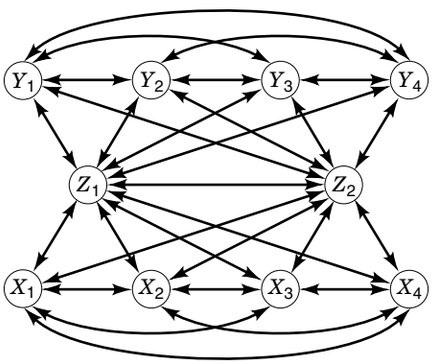

Figure 3. Partially interconnected neural network for the 4–2–4 encoder problem.

tern of activations) represents the problem solution. In a Boltzmann machine, a unit may flip its activation (from 0 to 1, or vice versa); whether this flip occurs depends on the unit's net-input and a parameter, known as temperature. The process of selecting a unit at random and allowing it to change its activation (or not, depending on the specified probability function) continues, with the temperature reduced very gradually, until the activations stabilize. The change is accepted stochastically in order to reduce the chances of the network becoming trapped in a local optimum.

The process of gradually reducing the temperature, by which the stochastic behavior of a system is gradually made less and less random, is known as *simulated annealing.* It is analogous to the physical annealing process used to produce a strong metal (with a regular crystalline structure). During annealing, a molten metal is cooled gradually in order to avoid freezing imperfections in the crystalline structure of the metal.

Boltzmann Machine Weights

Some Boltzmann machines belong to the set of neural networks for which the weights are fixed when the network is designed. These networks are used for constraint satisfaction and constrained optimization problems. Fixed-weight Boltzmann machines for constraint satisfaction and constrained optimization problems are designed so that the network converges to a minimum of an energy function or maximum of a consensus function. These two formulations are equivalent; the following discussion will use the consensus function.

Other Boltzmann machines undergo a training phase after which the network can perform the intended task using input data that are similar (but not necessarily identical) to its training input. A Boltzmann machine with learning can solve pattern completion and more general input-output mapping problems. Although training is characteristic of the majority of neural networks, fixed-weight Boltzmann machines are simpler and more widely used than adaptive-weight Boltzmann machines and will be discussed first.

Designing a Fixed-Weight Boltzmann Machine. Fixed-weight Boltzmann machines are designed by formulating a function (consensus) that describes the constraints, and objective to be optimized if there is one, of the problem. Each unit in the network represents a hypothesis; the activation of the unit corresponds to the truth or falsity of the hypothesis. The weights are fixed to represent both the constraints of the problem and the quantity to be optimized. The activity level of each unit is adjusted so that the network will converge to the desired maximum consensus; the pattern of activations then corresponds to the solution of the problem.

The connection between two units controls whether the units are encouraged to be on or off. A positive connection between two units encourages both units to be on; a negative connection encourages one or the other of the units to be off. Each unit also may have a bias (self-connection) that influences its activation regardless of the activations of the other units connected to it.

The weights for the network are determined so that the probability of accepting a change that improves the network configuration is greater than the probability of rejecting it. However, early in the solution process, when the temperature is relatively high, the probability of accepting a "bad change" or rejecting a "good change" is much closer to 0.5 than later, after the network has cooled.

Neural networks have several potential advantages over traditional techniques for certain types of optimization problems. They can find near-optimal solutions relatively quickly for large problems. They can also handle situations in which some constraints are less important than others.

Training an Adaptive-Weight Boltzmann Machine. The Boltzmann machine is also used for learning tasks. The network architecture may incorporate input, hidden, and output units. Input and output neurons are those for which the correct activations are known; any other units are hidden.

During training, a neural network is given a sequence of training patterns, each of which specifies an example of the desired activations for the input and output neurons. Boltzmann learning requires several cycles during which the network is allowed to reach equilibrium. Each cycle requires starting the network at a fairly high temperature and allowing the appropriate neurons to adjust their activations as the network cools.

For each training pattern, the network is allowed to reach equilibrium with the activations of the input and output units held fixed (clamped) to the values given for that pattern. Only the activations of the hidden units change during this phase. After this has been done several times for all training patterns, the probability of each pair of neurons being on is computed as the fraction of the time both are on, averaged over all training runs for all training patterns.

The same process is repeated with none of the activations clamped; this is called the *free-running phase* of training. Since large positive weights encourage both neurons to be on, the weight on the connection between a pair of neurons is increased if the probability of both units being on was higher in the clamped phase of training than in the free-running phase. On the other hand, if it was less likely for the units to be on simultaneously in the clamped than in the free-running phase, the weight between that pair of units is reduced.

SAMPLE APPLICATIONS OF BOLTZMANN MACHINES

Many constrained optimization problems have been solved using neural networks; if the problem can be formulated as a 0-1 programming problem, then the states of the Boltzmann machine are assigned to the variables, and the cost function and constraints are implemented as the weights of the network. The solution of the traveling salesman problem (TSP) serves as a model for other constrained optimization problems.

Boltzmann machines can be used to generate initial configurations of assets for a generic game (e.g., chess). The desired distribution of playing pieces is subject to restrictions on the number of pieces (of several different types) that are present, as well as some preferences for the relative positions of the pieces. The rules implemented in the network allow for flexibility in assigning locations for available resources while the probabilistic nature of the network introduces a degree of variability in the solutions generated (4).

The class scheduling/instructor assignment problem is an example of a problem containing both strong and weak con-

straints. For example, the strong constraints could ensure that a single instructor is not assigned two classes at once, that each class is offered exactly once, and that each instructor is assigned a fair class load. The weak constraints might specify instructors' preferences for class subjects and class time periods (5,6).

Boltzmann learning is illustrated using the encoder problem, which requires that binary patterns presented to the input units pass through a bottleneck (the hidden units) and reproduce the original pattern at the output units. Using input patterns in which only one unit is active, the network learns a more concise representation of the information at the hidden units (7).

OPERATION OF A FIXED-WEIGHT BOLTZMANN MACHINE

Recall that the neurons in a fixed-weight Boltzmann machine represent hypotheses; if the neuron is active, the hypothesis is interpreted to be true; otherwise the hypothesis is considered to be false. The weights in a Boltzmann machine for constraint satisfaction or constrained optimization represent the constraints of the problem and the quantity to be optimized. The weight w_{ij} expresses the degree of desirability that units X_i and X_j are both on. The bidirectional nature of the connection requires that $w_{ij} = w_{ji}$. A unit may also have a self-connection, w_{ii}.

A fixed-weight Boltzmann machine operates to maximize the consensus function

$$C = \sum_i \left(\sum_{j \le i} w_{ij} x_i x_j \right)$$

by letting each unit attempt to change its state. The change in consensus, if unit X_i changes its state, is given by

$$\Delta C = (1 - 2x_i) \left(w_{ii} + \sum_{j \neq i} w_{ij} x_j \right)$$

where x_i is the current state of unit X_i. However, unit X_i does not necessarily change its state even if doing so would increase the consensus. The probability of accepting the change in state is given by

$$\Pr[X_i \text{ changes state}] = \frac{1}{1 + \exp(-\Delta C/T)} \tag{1}$$

The parameter T (temperature) is gradually reduced as the network searches for a maximal consensus. Lower values of T make it more likely that the network will accept a change of state that increases consensus and less likely that it will accept a change that reduces consensus.

In general, the initial temperature should be taken large enough so that the probability of accepting the change of state is approximately 0.5, regardless of whether the change is beneficial or detrimental. The temperature is then reduced slowly so that the ratio of probabilities of two states of the network will continue to obey the Boltzmann distribution, which gives the network its name.

It is convenient to break the iterative process by which the network converges to equilibrium into a number of smaller cycles called *epochs*. Each epoch consists of a specified number of unit update attempts (usually equal to the number of units in the network). An exponential cooling schedule in which the temperature is reduced by a given factor after each epoch is common in practice:

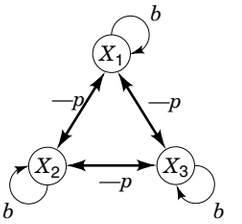

Figure 4. A simple Boltzmann machine.

$$T(k+1) = \alpha T(k)$$

Fewer epochs are required at each temperature for larger values of α (such as $\alpha = 0.98$) than for smaller α (e.g., $\alpha = 0.9$).

Simple Fixed-Weight Boltzmann Machine

The weights for a Boltzmann machine are fixed so that the network will tend to make state transitions towards a maximum of the consensus function defined previously. For example, if we wish the simple Boltzmann machine illustrated in Fig. 4 to have exactly one unit on, the weights p and b must be chosen so that improving the configuration corresponds to increasing the consensus. Each unit i is connected to every other unit j with weight $w_{ij} = -p$ ($p > 0$). These weights are penalties for violating the conditions that at most one unit is "on." In addition, each unit has a self-connection of weight $w_{ii} = b$ ($b > 0$). The self-connection weight is an incentive (bonus) to encourage a unit to become active if it can do so without causing more than one unit to be on.

The relationship between p and b can be deduced by considering the effect on consensus in the following two situations. If unit X_i is off and none of the units connected to X_i is on, allowing X_i to become active will increase the consensus of the network by the amount b. This is a desirable change; since $b > 0$, it corresponds to an increase in consensus and the network will be more likely to accept this change than to reject it.

On the other hand, if one of the units connected to X_i is already on, attempting to turn unit X_i on would result in a change of consensus of the amount $b - p$. Thus, for $b - p < 0$ (i.e., $p > b$), the effect is to decrease the consensus and the network will tend to reject this unfavorable change. Bonus and penalty connections, with $p > b > 0$, are used in the traveling salesman problem (TSP) network to represent the constraints for a valid tour and in an analogous manner for other applications of fixed-weight Boltzmann machines.

Traveling Salesman Problem

The standard TSP serves as a model for many constrained optimization problems. The requirements are that a salesman visit each of a specified group of cities once and only once, returning at the end of the trip to his initial city. It is desired that the tour be accomplished in the shortest possible total distance. Many variations on this basic problem can also be solved using essentially the same approach as described here.

Architecture. A neural network solution to the TSP is usually formulated with the units arranged in a two-dimensional array. Each row of the array represents a city to be visited; each column corresponds to a position or stage of the tour. Thus, unit $U_{i,j}$ is on if the ith city is visited at the jth step of the tour. A valid tour is given by a network configuration in which exactly one unit is on in each row and each column. An example of a valid tour, in which city B is visited first, city D second, city C third, and city A last, is illustrated in Fig. 5.

Weights. Although the connections are not shown in Fig. 5, the architecture consists of three types of connections. The units within each row (and within each column) are fully interconnected. The weights on each of these connections is $-p$; in addition, each unit has a self-connection, with weight b. If $p > b > 0$, the network will evolve toward a configuration in which exactly one unit is on in each row and each column.

To complete the formulation of a Boltzmann neural network for the TSP, weighted connections representing distances must be included. In addition to the weights described above (which represent the constraints), a typical unit $U_{i,j}$ is connected to the units $U_{k,j-1}$ and $U_{k,j+1}$ (for all $k \neq i$) by weights that represent the distances between city i and city k. Since the Boltzmann machine operates to find the maximum of the consensus function, the weights representing distances are the negative of the actual distances. Units in the last column are connected to units in the first column by connections representing the appropriate distances also. However, units in a particular column are not connected to units in columns other than those immediately adjacent.

The bonus weight b is related to the distance weights. Let d denote the maximum distance between any two cities on the tour and consider the situation in which no unit is on in column j or in row i. Since allowing Ui,j to turn on should be encouraged, the weights should be set so that the consensus will be increased if it turns on. The change in consensus will be $b - d_{k1,i} - d_{i,k2}$, where $k1$ indicates the city visited at stage $j - 1$ of the tour, and $k2$ denotes the city visited at stage $j + 1$ (and city i is visited at stage j). This change is greater than (or equal to) $b - 2d$ so ΔC will be positive if $b > 2d$.

Thus we see that if $p > b$, the consensus is larger for a feasible solution (one that satisfies the strong constraints) than for a nonfeasible solution, and if $b > 2d$ the consensus will be higher for a short feasible solution than for a longer tour.

Performance. The traveling salesman problem is a nice model for a variety of constrained optimization problems, including the asset allocation and scheduling problems discussed in the next sections. The TSP is, however, a difficult problem for the Boltzmann machine, because in order to go from one valid tour to another, several invalid tours must be accepted. The transition from valid solution to valid solution is not as difficult in many other constrained optimization problems.

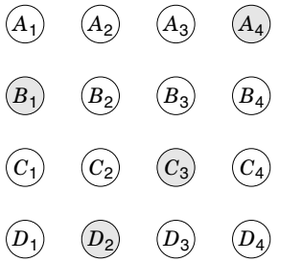

Figure 5. A valid solution of the four-city traveling salesman problem.

Asset Allocation

Consider the problem of distributing a fixed number of assets (such as chess pieces) of several different types on a two-dimensional region (i.e., the chessboard) in arrangements that satisfy certain rules regarding their relative positions with respect to assets of other types. As an example, the placement of pieces on a chessboard must follow certain strong conditions (e.g., the two pieces cannot occupy the same square on the chessboard at the same time) as well as weak conditions (e.g., black might consider it desirable to have several other chess pieces in the vicinity of black's king). There are a variety of problems of this type, including distribution of biological species and deployment of military assets.

Architecture. To illustrate the basic approach, consider first two types of chess pieces (assets) representing the black king and other black pieces. The problem to be solved by a Boltzmann machine is to generate a number of arrangements of these pieces on a chessboard, subject to specified restrictions. To accomplish this, the neural network architecture consists of two layers of units (layer X for the king and layer Y for the other pieces); each layer is an 8×8 array corresponding to the squares of a chessboard. If a unit is on in the *king layer,* it signifies that a chess piece (king) is present at the location; if a unit is on in the *other layer,* it indicates that some other chess piece is present at that location.

Weights. There are several types of weights required for this example. First, each unit has an excitatory self-connection, b, to encourage the unit to be active. Second, the units in each layer are fully interconnected among themselves, with inhibitory weights, which are determined so that the desired number of units will be active in that layer. The units corresponding to the same square on the chessboard (the same physical location) are connected by very strong inhibitory weights, to discourage having a king and another piece on the same square at a given time. Furthermore, if it is desirable to have several other pieces present in the general vicinity of the location of a king, excitatory connections between each unit in layer X and the units corresponding to nearby positions on the chessboard in layer Y are included. The connection paths between units in Fig. 6 show the inhibition between units X_{23} and Y_{23} as well as the excitation between X_{23} and the units in the Y layer that correspond to neighboring board positions.

By carefully designing the weights in the network, the network will tend to converge to a configuration that represents a desirable arrangement of the assets. However, the random nature of the unit update process causes the network to produce a variety of solutions satisfying the specified relationships among the assets.

In order to limit the number of assets of type X to the desired number, n_X, there are inhibitory connections with value

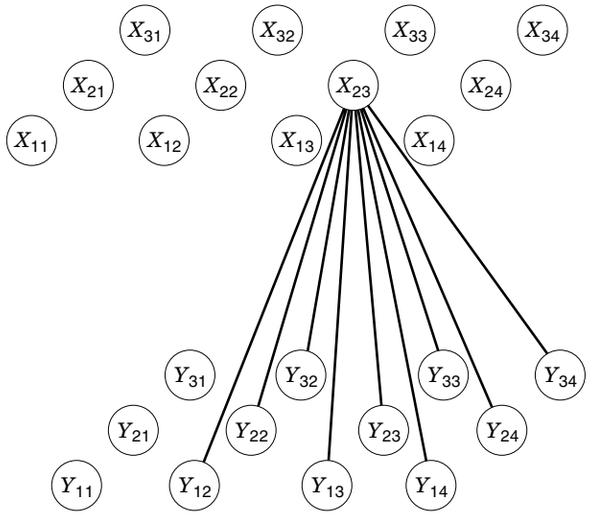

Figure 6. A portion of a Boltzmann machine for king (X) and other (Y) chess pieces.

p_X between each pair of units in layer X; similarly to limit the number of assets of type Y to the desired number, n_Y, there are inhibitory connections with value p_Y between each pair of units in layer Y. There are also excitatory connections with weight b_{XY} between the appropriate units in layer X and layer Y, to encourage a desirable arrangement of the assets.

The relations between these weights that need to be satisfied so that the network will evolve towards a configuration in which there are n_X assets of type X are deduced in a manner similar to that used for the TSP. Assume that at a particular time there are $n_X - 1$ assets of type X. If an inactive unit in layer X is selected, the total bonus signal received by this unit should exceed the total penalty signal. In the worst case, there are no units in other layers encouraging this unit to turn on, and the only bonus signal the unit will receive is b_X, its self-bonus. At the same time, it is receiving an inhibitory signal of $(n_X - 1)\, p_X$ from the other units that are on in layer X. So, to increase the probability of the unit changing states (a desirable change), we require

$$(n_X - 1)p_X \leq b_X \tag{2}$$

On the other hand, we want no more than n_X assets of type X present. If there are n_X assets of type X present and an inactive unit in layer X is selected for update, the probability that it will change state needs to be minimized. The unit receives a total penalty signal of $n_X p_X$. Under the most extreme conditions, all units in layer Y that encourage the selected unit to turn on will be on; say there are m_Y such units. This means that the unit receives a total bonus signal of $b_X + m_Y\, b_{XY}$. Since it is not desirable for the unit to turn on, we require

$$b_X + m_Y b_{XY} \leq n_X p_X \tag{3}$$

From Eq. (2) and Eq. (3) it follows that

$$(n_X - 1)p_X \leq b_X \leq b_X + m_Y b_{XY} \leq n_X p_X \tag{4}$$

These inequalities are sufficient to allow the network to evolve towards an activation pattern in which there are n_X assets; corresponding inequalities hold in layer Y.

To encourage the network to converge to a solution in which the assets have the desired relative arrangement, consider a configuration of the network in which there are n_X assets of type X, but some of these assets are located in the wrong place. If a unit that is in the wrong place is selected for update, the probability that it changes states needs to be maximized. This unit receives a total bonus signal of b_X (it does not receive any bonus from assets of type Y since it is not in the region of encouragement for any unit that is on). Furthermore, it receives a total penalty signal of $(n_X - 1)\, p_X$ from the other units that are on in layer X. It will be more likely for the unit to turn off if

$$b_X \leq (n_X - 1)p_X \tag{5}$$

Combining Eq. (2) and Eq. (5), we find that

$$b_X = (n_X - 1)p_X$$

By assigning an arbitrary value for p_X, the remaining weights can be determined.

To prevent the existence of two or more assets of different types in the same physical location a large penalty connection p_h is introduced between units in different layers but with the same subscripts. This penalty signal should override any bonus signals going into any unit.

Performance. Simulations with four types of assets corresponding to white king, white other, black king, and black other pieces illustrate that the number of assets of each type converges relatively quickly to the desired values. Fluctuations in the locations of assets of one type relative to other types continue until the temperature becomes very small. However, many valid solutions are generated quite quickly. These studies specified the number of assets of each type that should be present throughout the entire region, and that the other pieces should be near the king of the same color.

Many extensions of these ideas are possible. For example, there is no additional difficulty encountered if white and black do not have the same number of other pieces. The logic of describing more complicated board arrangements, with more different playing pieces, is a straightforward extension of this simple example. See Ref. 4 for a more detailed description of this example.

A Time-Task-Worker Scheduling Problem

The Boltzmann machine can also be used to solve the classic problem of assigning workers to cover a variety of tasks in different time periods. As a simple example, consider the problem of scheduling instructors to teach classes that may be offered at various times. This problem can be viewed as the intersection of three separate problems: scheduling classes to be given at appropriate time periods, scheduling instructors to teach at certain time periods, and assigning instructors to teach particular classes. A similar approach could be used for scheduling airline flights and pilots, or many other related problems.

The strong constraints for generating a valid schedule include: ensure that each class is taught exactly once, no in-

structor is assigned to teach more than one class during any given time period, and so on. It is also desirable that each instructor be responsible for a "fair" share of the class load. In addition, we allow for weak constraints describing instructors' preferences for certain classes and/or time periods.

The problem of producing a teaching schedule for a single instructor is closely related to the TSP, with classes corresponding to the cities to be visited and the time periods corresponding to the order in which the cities are visited. Similarly, both the assignment of classes to instructors (during each time period) and the scheduling of each class, in terms of who will teach it and at what time, are instances of the TSP.

Architecture. It is convenient to visualize the architecture of a Boltzmann machine neural network for this problem as composed of several rectangular arrays of neurons, one for each instructor, stacked on top of each other in a three-dimensional array. As a simple example problem, one might assume that there are 20 classes, to be taught by six instructors, within five possible time periods. The architecture would then be a $5 \times 6 \times 20$ array of neurons; with a 5×6 array corresponding to each class, a 6×20 array corresponding to each time period, and a 5×20 array for each instructor. An active cell, $U_{ijk} = 1$, means that at time i instructor j teaches class k.

Weights. As in the Boltzmann machines for the traveling salesman and asset allocation problems, each neuron has a self-connection to encourage the unit to be active. To allow for some variation in instructors' preferences for certain classes or time periods, or factors that make it preferable to have certain classes taught at certain times, the bias for each neuron is taken to be equal to a standard bias b plus some preference, which may be between $-m$ and m. Thus, the maximum possible bias for a unit is $b + m$ and the minimum is $b - m$.

Since each class is to be taught exactly once, only one unit should be on in the array for each class; this is accomplished by fully interconnecting the units within the plane for each class, with inhibitory connections of strength $-p$, with $p > b + m > 0$. Similarly, the units in each line corresponding to an instructor–class period combination are connected with weights of strength $-p$ to ensure that each instructor is assigned to no more than one class during any period.

Finally, the units within each class offering–time period plane must be fully interconnected to ensure that each instructor is assigned an appropriate number of classes. An inhibitory weight with strength of $-f$ is needed to ensure that all instructors teach approximately the same number of classes, r (with 20 classes and six instructors, $r = 4$). The suitable range of values for the weight f can be deduced by consideration of a single unit deciding whether to be active or inactive. Ignoring other connections for the time being, the unit should be turned on if the total number of active units in the class offering–time slot plane is less than r and turned off otherwise. To encourage a plane to have exactly r units active, we require that

$$b - m - (r - 1)f > 0 \text{ and } b + m - rf < 0$$

Thus, the base value of the bonus, b, must satisfy

$$b > (2r - 1)m$$

It is possible to have different values of r for different instructors, with either different values of b for each instructor, or the value of b for the instructor with the greatest value of r used for all instructors' class offering–class period planes. In either case p must be greater than the maximum bias applied to any one unit.

It is often the case that a group of students will require the same set of classes, which should be scheduled at different times. Within the plane for each time period, we include an inhibitory connection with strength $-c$ between units that represent classes that should not conflict, to encourage the Boltzmann machine to converge to a schedule without any such conflicts. The value of c can vary depending upon how many such conflicts there are. In general, the sum of such inhibitory connection strengths must be less than b for any given unit. Usually, however, a single class conflicts with no more than one or two other classes. The value of c can range from $-b + m + 1$ to 0 for a unit that only conflicts with one other class.

Performance. The Boltzmann machine is better suited for the class scheduling/instructor assignment problem than for the TSP, since it is easy to move from one valid schedule to another. The system must pass through only one state with a lower overall consensus to move from one valid schedule to another. Once the transition from a state corresponding to a valid schedule to one with an invalid schedule is made, only transitions resulting in positive changes in consensus are required to return to a state corresponding to a new valid schedule.

This application of Boltzmann machines to scheduling problems is based on the discussion in Ref. 6; a more extensive example, solved with a closely related neural network, is presented in Ref. 5.

Boltzmann Machine with Learning

The Boltzmann machine can also be trained for use in input–output transformations such as pattern completion problems when examples of the desired network action are available for supervised learning. The most interesting situations for which a learning algorithm is needed are the cases in which only partial information about the global states of the system is available. Thus, the network is assumed to consist of visible units (input and output units) and hidden units. Part of the training process includes letting the network converge with the activations of the visible units clamped. After training, the input units are clamped and the network is allowed to find the correct values for the output units. Hidden units are never clamped.

A simple example problem is to train the network to reproduce the input pattern on the output units after passing the signals through a hidden layer that has fewer units than the input and output layers (7). This is known as an m–p–m encoder problem, with p, the number of hidden units, less than m, the number of input units or outputs units. The architecture shown in Fig. 3 can be used for a problem with 4 input units, 2 hidden units, and 4 output units. The presence of interconnections among the hidden units, and among the output units is significant; however, there are no connections directly from the input units to the output units. A self-connection is also used for each unit, but not shown.

The agreement between the desired probabilities for the visible units and the probabilities of the visible units when the network is at equilibrium can be increased by changing the weights. Furthermore, the weight changes can be made based on local information.

Algorithm. Boltzmann learning requires information about the probability that any two units, i and j, are both on, in two different equilibrium situations:

PC_{ij} is the probability when the visible units are clamped

PF_{ij} is the probability when only the input units are clamped

The training process can be summarized in the following algorithm:

To compute the values of PC
  For each training pattern:
    Clamp visible units
    Perform several cycles of the following two steps (using a different random seed for each trial)
      Let the network converge
      For each pair of units ij, determine whether they are both on
    Average the results for this pattern to find values of PC_{ij} for each i and j
  After the cycle is completed for each training pattern, average the results to find PC values

To compute the values of PF
  For each training pattern:
    Clamp only the input units
    Perform several cycles of the following two steps
      Let the network converge
      For each pair of units, determine whether they are both on
    Average the results for this pattern to find values of PF
  After the cycle is completed for each training pattern, average the results to find PF values

Compare PC and PF (for each pair of units), and adjust the weight between them:

$$\Delta w_{ij} = \mu(PC_{ij} - PF_{ij})$$

where $\mu > 0$ is the learning rate.

The update of the weight connecting two units may be proportional to the difference between the probability that the units are both active when the network is running in the clamped mode versus the corresponding probability when the network is in the unclamped mode, as shown in the algorithm above. On the other hand, the network can also be trained using a fixed-size weight adjustment, as described in the original presentation of the network (7).

Application. As a simple example, consider the following four training vectors; only one input unit is active in each pattern, and the corresponding output pattern is the same as the input pattern.

Input	Output
(1 0 0 0)	(1 0 0 0)
(0 1 0 0)	(0 1 0 0)
(0 0 1 0)	(0 0 1 0)
(0 0 0 1)	(0 0 0 1)

During the clamped phase of training, only the 2 hidden units adjust their activations, so each epoch consists of 2 unit updates. The annealing schedule was 2 epochs at $T = 20$; 2 epochs at $T = 15$; 2 epochs at $T = 12$; and 4 epochs at $T = 10$. After the network cools, statistics are gathered for 10 epochs at $T = 10$ to determine the fraction of the time that units i and j are both on. This process is repeated for each of the four training vectors, and the results for all training vectors are averaged to give PC for each pair of units that are connected.

The process of determining PF_{ij} uses the same annealing schedule and gathers statistics for the same number of epochs at $T = 10$. However, since no units are clamped during this second phase, each epoch consists of 10 unit update attempts.

Once the values of PC_{ij} and PF_{ij} have been found, the weights are updated and the entire weight update cycle is repeated until the weights have stabilized or the differences between PC_{ij} and PF_{ij} are sufficiently small. In 250 tests of the 4-2-4 encoder problem, the network always found one of the global minima and remained at that solution. As many as 1810 weight update cycles were required, but the median number was 110 (7). After training, the network can be applied by clamping the input units and allowing the net to converge. The activations of the output units then give the response of the network.

The algorithm as originally presented uses a fixed weight-step increment if $PC_{ij} > PF_{ij}$ and the same sized decrement for the weights if $PC_{ij} < PF_{ij}$. Difficulties can occur when only a few of the 2^v possible states for the visible units are specified. Rather than trying to demand that other (nonspecified) states never occur, it is recommended to use noisy inputs with low, but nonzero probabilities. For the published simulations described previously, noise was added on each presentation of a training pattern: a component that is 1 in the true training vector was set to 0 with probability 0.15, and 0 components were set to 1 with probability 0.05.

ALTERNATIVE FORMULATIONS OF THE BASIC BOLTZMANN MACHINE

Variations

As mentioned earlier, the constraint satisfaction problems to which the Boltzmann machine is applied can be formulated as either maximization or minimization problems. Ackley, Hinton, and Sejnowski (7) define the energy of a configuration as

$$E = \sum_{i<j} w_{ij} x_i x_j + \sum_i \theta_i x_i$$

where θ_i is a threshold and biases are not used. The difference in energy if unit X_k changes from off to on is

$$\Delta E_k = -\theta_k + \sum_i w_{ik} x_i$$

The Boltzmann machine is also used with either of two slightly different acceptance conditions, namely,

1. Set the output of unit to 1 with probability given by Eq. (1) regardless of the current activity of the unit. Or
2. Accept the proposed change in activation if it improves the solution, but accept a change that moves the solution in the opposite direction, with probability given by Eq. (1). See Refs. 2 and 8 for further discussion.

Markov Chain Process

The Boltzmann machine can be described in terms of a Markov chain process. Each stage consists of the following steps:

1. Generate a potential new configuration of the network
2. Accept or reject the new configuration
3. Reduce temperature according to the annealing schedule

For the Boltzmann machine, the generating probability is given by the Gaussian distribution

$$G = T^{-0.5n} \exp\left(\frac{-D^2}{T}\right)$$

where n is the number of units in the network, and D is the number of units the activations of which change in going from current configuration to new configuration. Note that as T is reduced the generation probability G also changes.

Thus, the probability of generating a candidate configuration depends only on the temperature and the number of units that change their state. In the preceding discussion, all configurations in which exactly one unit changes its state are equally likely to be chosen as the candidate state at any time. Configurations in which more than one unit changes its state ($D > 1$) are generated with probability 0.

The probability of accepting the new configuration depends on the current temperature and the change in consensus ΔC that would result, according to Eq. (1). This form of analysis is useful for theoretical analysis of the process.

Cooling Schedules

The success of a Boltzmann machine is closely related to how slowly the temperature is decreased and how many update trials are performed at each temperature. An exponential cooling schedule, $T_k = \alpha^k T_0$, is very common in practice (8). This cools the system rather quickly at high temperatures and then very slowly at low temperatures. As long as enough trials are performed at each temperature to allow each unit to attempt to change its state several times, good results are obtained.

Geman and Geman (9) present a theoretical proof that, if $T_k \geq c/\ln(1+k)$, the system converges to an optimal state (as $k \to \infty$) where k is the number of epochs and c is a constant that does not depend on k. A very slow decrease of the temperature is necessary, but with this slow decrease, only one epoch is required at each value of k.

EXTENSIONS OF THE BOLTZMANN MACHINE

The Boltzmann machine is closely related to several more general types of evolutionary computing. The most important of these more general approaches are summarized in the following sections.

Mean-Field Annealing

One of the most popular modifications of the original Boltzmann machine replaces the probabilistic action of a binary neuron with an analog neuron, the activation of which is determined as the average (mean) value of the binary neuron at any particular temperature. The value of an arbitrary analog neuron takes the form (in the mean-field theory approximation, with an energy function E to be minimized)

$$\Delta E_i = -\theta_i + \sum_j w_{ji} v_j$$

$$v_i = \tanh\left(\frac{\Delta E_i}{T}\right)$$

In general, little change occurs to v_i for temperatures above a critical value, T_c. Thus, the annealing process can proceed more rapidly at higher temperatures and can be slowed when the temperature reaches the point at which changes to the activations of the neurons in the network occur. Alternatively, the mean-field equations can be solved iteratively. This gives a direct connection between the Boltzmann machine and the continuous Hopfield network with noise (see Ref. 3 for a discussion of the Hopfield network). For further discussion of mean-field annealing see Refs. 2, 10, and 11; it is used for applications to scheduling problems (5) and the knapsack problem (12).

Other Related Networks

High-order Boltzmann machines (HOBM) allow for terms of higher order in the consensus function than those for the standard Boltzmann machine (in which the consensus function has only first- and second-order terms). The theoretical results, such as uniqueness of the learned solution, which have been established for these HOBM do not hold for the Boltzmann machine with hidden units. See Ref. 13 for discussion and proofs.

The Helmholtz machine is a fairly general unsupervised learning architecture with feedback connections; Boltzmann machines are one simple specific variety of Helmholtz machine. For a detailed discussion see Ref. 14; this article also includes an extensive bibliography of relevant papers.

For Boltzmann machines in which the hidden and output units have a special hierarchical organization, learning can be accomplished using gradient descent (as for the popular backpropagation training algorithm of feedforward neural networks). Simulations with the N-bit parity problem and detection of hidden symmetries in square pixel arrays have demonstrated the network's ability to learn quickly and to generalize successfully. See Ref. 15 for further discussion.

SUMMARY AND CONCLUSIONS

One of the potential advantages of a neural network approach to problem solving is its inherent parallelism. Although units updating in parallel may make their decision to accept or reject a change of state based on information that is not completely up to date, several parallel schemes for the Boltzmann machine have given promising results (1). These schemes can be characterized as either synchronous or asynchronous, and either limited or unlimited. In limited parallelization, small groups of neurons that do not directly affect each other can update at the same time without any possibility of errors in the calculation of the change of consensus. This scheme, however, is not well suited to massive parallelism since the number of sets of independent units is small.

In synchronous unlimited parrallelization, all units compute their change in consensus and acceptance probabilities independently, and any potential difficulty from erroneously calculating their acceptance probability is simply ignored. In asynchronous parallelization, each unit has its own cooling schedule and state transitions are performed simultaneously and independently. Since the probability of any unit changing its state approaches 0 as the network cools, the likelihood of two connected units changing their states based on out-of-data information also decreases. Simulations using this type of parallelization for a variety of combinatorial problems give results that are comparable to other methods.

Problems from many fields can be formulated in a manner for which a layered Boltzmann machine solution is of interest. Applications to biological ecosystems and urban planning are two promising areas. The results presented here suggest that layered Boltzmann machines are an interesting neural network approach to applications for which some variation in the solutions is desirable.

BIBLIOGRAPHY

1. E. Aarts and J. Korst, *Simulated Annealing and Boltzmann Machines.* Chichester: Wiley, 1989.
2. A. Cichocki and R. Unbehauen, *Neural Networks for Optimization and Signal Processing.* Chichester: Wiley, 1993.
3. L. V. Fausett, *Fundamentals of Neural Networks: Architectures, Algorithms, and Applications.* Englewood Cliffs, NJ: Prentice Hall, 1994.
4. W. Elwasif, L. V. Fausett, and S. Harbaugh, Boltzmann machine generation of initial asset distributions. In S. K. Rogers and D. W. Ruck (eds.), *Proceedings, Applications and Science of Artificial Neural Networks,* SPIE, Vol. 2492, 1995, pp. 331–340.
5. L. Gislen, C. Peterson, and B. Soderberg, Complex scheduling with Potts neural networks, *Neural Computat.,* **4**: 805–831, 1992.
6. R. S. Schumann, Analysis of Boltzmann Machine Neural Networks with Applications to Combinatorial Optimization Problems, M.S. thesis, Florida Institute of Technology, 1992.
7. D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, A learning algorithm for Boltzmann machines, *Cognitive Sci.,* **9**: 147–169, 1985.
8. S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, Optimization by simulated annealing, *Science,* **220** (4598): 671–680, 1983.
9. S. Geman and D. Geman, Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images, *IEEE Trans. Pattern Anal. Mach. Intell.,* **PAMI-6**: 721–741, 1984.
10. C. Peterson and J. R. Anderson, A mean field learning algorithm for neural networks, *Complex Syst.,* **1**: 995–1019, 1987.
11. C. Peterson and B. Soderberg, A new method for mapping optimization problems onto neural networks, *Int. J. Neural Syst.* **1**: 3–22, 1989.
12. M. Ohlsson, C. Peterson, and B. Soderberg, Neural networks for optimization problems with inequality constraints: the knapsack problem, *Neural Computat.,* **5**: 331–339, 1993.
13. F. X. Albizuri, A. D'Anjou, M. Grana, and J. A. Lozano, Convergence properties of high-order Boltzmann machines, *Neural Netw.,* **9**: 1561–1567, 1996.
14. P. Dayan and G. E. Hinton, Varieties of Helmholtz machine, *Neural Netw.,* **9**: 1385–1403, 1996.
15. L. Saul and M. I. Jordan, Learning in Boltzmann trees, *Neural Computat.,* **6**: 1174–1184, 1994.

LAURENE V. FAUSETT
University of South
Carolina—Aiken

BOLTZMANN TRANSPORT EQUATION. See SEMICONDUCTOR BOLTZMANN TRANSPORT EQUATION.

BOOKKEEPING. See ACCOUNTING.