neuron is that its output is a nonlinear function of the sum of its inputs. The assumption is that when memories are stored in the brain, the strengths of the connections between neurons change.

One of the uses of neural network models is pattern recognition. If we see someone more than once, the person's face provides input that helps us to recall the person's name. In the same spirit, a neural network can be given a pattern, for example, a string of ± 1s, that partially reflect a previously memorized pattern. The idea is to store memories so that a computer can recall them when the inputs are close to a particular memory.

We now consider an example of a neural network due to Hopfield. The network consists of N neurons, and the state of the network is defined by the state of each neuron S_i, which in the Hopfield model takes on the values -1 (not firing) and $+1$ (firing). The strength of the connection between neuron i and neuron j is denoted by w_{ij}, which is determined by the M stored memories:

$$w_{ij} = \sum_{s=1}^{M} S_i^\alpha S_j^\alpha, \tag{14.4}$$

where S_i^α represents the state of neuron i in stored memory α. Given the initial state of all the neurons, the dynamics of the network is simple. We choose a neuron i at random and change its state according to its input, which is $\sum_{i \neq j} w_{ij} S_j$, where S_j represents the current state of neuron j. Then we change the state of neuron i by setting

$$S_i = \begin{cases} +1 & \text{for } \sum_{i \neq j} w_{ij} S_j > 0 \\ -1 & \text{for } \sum_{i \neq j} w_{ij} S_j \leq 0. \end{cases} \tag{14.5}$$

The threshold value of the input has been set equal to zero, but other values could be used as well.

The `HopfieldApp` class in Listing 14.8 implements this model of a neural network and stores memories based on user input. The state of the network is stored in the array `S[i]` and the connections between the neurons are stored in the array `w[i][j]`. The user initially clicks on various cells to toggle their values between -1 and $+1$ and presses the Remember button to store a pattern. Then the user presses the Randomize button to initialize the S_i by setting S_i to ± 1 at random. After the memories are stored, press the Start button to update the neurons using the Hopfield algorithm to try to recall one of the stored memories.

Listing 14.8 `HopfieldApp` class.

```
package org.opensourcephysics.sip.ch14;
import org.opensourcephysics.controls.*;
import org.opensourcephysics.frames.*;

// Hopfield model of a neural network
public class HopfieldApp extends AbstractSimulation {
   LatticeFrame lattice;
   int N;        // total number of neurons
   double[][] w; // connection array (N by N elements)
   int numberOfStoredMemories;

   public HopfieldApp() {
      lattice = new LatticeFrame("Hopfield state");
      lattice.setToggleOnClick(true, -1, 1);
```

```
      lattice.setIndexedColor(-1, java.awt.Color.blue);
      lattice.setIndexedColor(0, java.awt.Color.blue);
      lattice.setIndexedColor(1, java.awt.Color.green);
      lattice.setSize(600, 120);
   }

   public void doStep() {
      int[] S = lattice.getAll();
      for(int counter = 0;counter<N;counter++) {
         // chooses random neuron index
         int i = (int) (N*Math.random());
         double sum = 0;
         for(int j = 0;j<N;j++) {
            sum += w[i][j]*S[j];
         }
         S[i] = (sum>0) ? 1 : -1;
      }
      lattice.setAll(S);
   }

   public void initialize() {
      N = control.getInt("Lattice size");
      w = new double[N][N];
      lattice.resizeLattice(N, 1);
      for(int i = 0;i<N;i++) {
         lattice.setAtIndex(i, -1);
      }
      lattice.setMessage("# memories = "+(numberOfStoredMemories = 0));
   }

   public void reset() {
      control.setValue("Lattice size", 8);
      lattice.setMessage("# memories = "+(numberOfStoredMemories = 0));
   }

   public void addMemory() {
      int[] S = lattice.getAll();
      for(int i = 0;i<N;i++) {
         for(int j = i+1;j<N;j++) {
            w[i][j] += S[i]*S[j];
            w[j][i] += S[i]*S[j];
         }
      }
      lattice.setMessage("# memories = "+(++numberOfStoredMemories));
   }

   public void randomizeState() {
      for(int i = 0;i<N;i++) {
         lattice.setAtIndex(i, Math.random()<0.5 ? -1 : 1);
      }
      lattice.repaint();
   }

   public static void main(String args[]) {
      SimulationControl control =
           SimulationControl.createApp(new HopfieldApp());
```