**CHAPTER** **3:** **USING** **A** **HOPFIELD** **NEURAL** **NETWORK**

* Comprensión de la red neuronal de Hopfield
* Reconocer patrones
* Uso de la auto asociación
* Construcción de una aplicación de red neuronal

Las redes neuronales han sido durante mucho tiempo el pilar de la programación inteligencia artificial (IA). Como programadores, podemos crear programas que hagan cosas bastante increíbles. Sin embargo, los programas ordinarios solo pueden automatizar tareas repetitivas, como equilibrar las chequeras o calcular el valor de una cartera de inversiones. Si bien un programa puede mantener fácilmente una gran colección de imágenes, no puede decirnos lo que se ilustra en ninguna de esas imágenes. Los programas son inherentemente poco inteligentes y creativos.

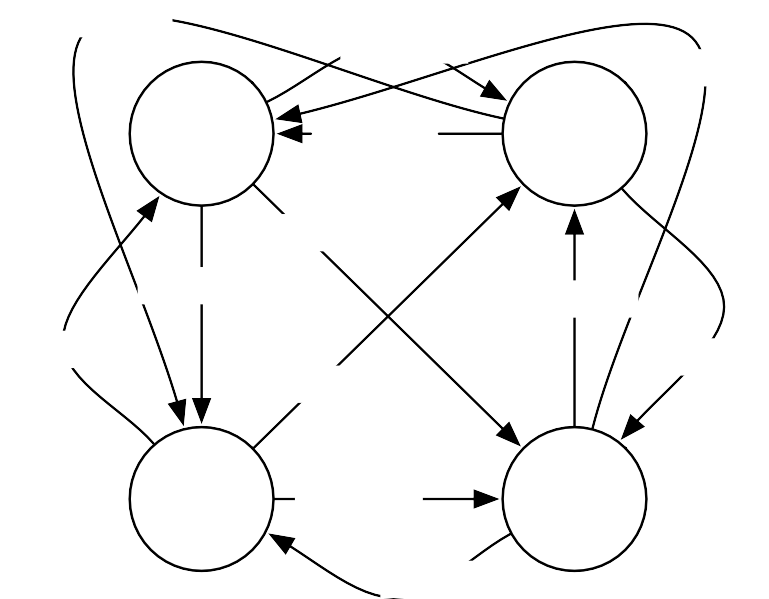
Las redes neuronales intentan dar a los programas informáticos inteligencia similar a la humana. Por lo general, están diseñados y entrenados para reconocer patrones específicos en los datos. Este capítulo le enseñará el diseño básico de una red neuronal mediante la introducción de la red neuronal hopfield.

# La red neuronal hopfield

La red neuronal hopfield es quizás el tipo más simple de red neuronal. La red neuronal hopfield es una red auto asociativa de una sola capa totalmente conectada. Esto significa que tiene una sola capa en la que cada neurona está conectada a otra neurona. Auto asociativo significa que si la red neuronal reconoce un patrón, devolverá ese patrón.

En este capítulo examinaremos una red neuronal hopfield con sólo cuatro neuronas. Es lo suficientemente pequeño como para ser fácilmente entendido, sin embargo, puede reconocer algunos patrones. Una red hopfield, con las conexiones, se muestra en la figura 3.1.

### Figura 3.1: Una red neuronal hopfield con 12 conexiones.



N2>N3

N1>N2

N4>N1

N1

N2>N1

N2

N1>N4

N1>N3

N4>N2

N3>N1

N2>N4

N3>N2

N3

N3>N4

N4

N4>N3

Construiremos un programa de ejemplo que crea la red Hopfield que se muestra en la figura 3.1. Dado que todas las neuronas de una red neuronal hopfield están conectadas a todas las demás neuronas, es posible que asumas que una red de cuatro neuronas contiene 42 o 16 conexiones. Sin embargo, 16 conexiones requerirían que cada neurona esté conectada a sí misma, así como a todas las demás neuronas. Este no es el caso en una red neuronal hopfield, por lo que el número real de conexiones es 12.

A medida que desarrollamos un programa de red neuronal de ejemplo, almacenaremos las conexiones en una matriz. Dado que cada neurona en una red neuronal hopfield está por definición conectada a todas las otras neuronas, una matriz bidimensional funciona bien. Todos los ejemplos de la red neuronal de este libro utilizarán algún tipo de matriz para almacenar sus pesos.

La Tabla 3.1 muestra el diseño de la matriz.

### Tabla 3.1: Conexiones en una red neuronal de Hopfield

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Neurona 1 (N1)** | **Neurona 2 (N2)** | **Neurona 3 (N3)** | **Neurona 4 (N4)** |
| **Neurona 1(N1)** | (n/a) | N2->N1 | N3->N1 | N4->N1 |
| **Neurona 2(N2)** | N1->N2 | (n/a) | N3->N2 | N3->N2 |
| **Neurona 3(N3)** | N1->N3 | N2->N3 | (n/a) | N4->N3 |
| **Neurona 4(N4)** | N1->N4 | N2->N4 | N3->N4 | (n/a) |

Esta matriz se denomina matriz de peso y contiene las ponderaciones asociadas a cada conexión. Esta matriz es la "memoria" de la red neuronal, y permitirá a la red neuronal recordar ciertos patrones cuando se presentan. Muchas de las redes neuronales de este libro también contendrán valores de umbral, además de las ponderaciones. Sin embargo, la red neuronal hopfield contiene solo pesos.

Por ejemplo, los valores mostrados en la Tabla 3.2 muestran los valores de peso correctos que se utilizarán para permitir que una red recuerde los patrones **0101** y **1010.** El método utilizado para crear una red con los valores contenidos en la Tabla 3.2 se cubrirá en breve. En primer lugar, se le mostrará cómo la red utiliza los valores de la Tabla 3.2 para recuperar **0101** y **1010.**

**Tabla 3.2:** **Pesos usados** **para** **recordar** **0101** **y** **1010**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Neurona 1 (N1)** | **Neurona 2 (N2)** | **Neurona 3 (N3)** | **Neurona 4 (N4)** |
| **Neurona 1 (N1)** | 0 | -1 | 1 | -1 |
| **Neurona 2 (N2)** | -1 | 0 | -1 | 1 |
| **Neurona 3 (N3)** | 1 | -1 | 0 | -1 |
| **Neurona 4 (N4)** | -1 | 1 | -1 | 0 |

# Recordando patrones

Ahora se le mostrará exactamente cómo se utiliza una red neuronal para recordar patrones. Comenzaremos presentando **el 0101** a la red Hopfield. Para ello, presentamos cada neurona de entrada, que en este caso son también las neuronas de salida, con el patrón. Cada neurona se activará en función del patrón de entrada. Por ejemplo, cuando a la neurona 1 se presenta con **0101**, su activación dará lugar a la suma de todos los pesos que tienen un 1 en el patrón de entrada. Por ejemplo, podemos ver en la tabla 3.2 que la neurona 1 tiene conexiones con las otras neuronas con los siguientes pesos:

0 -1 1 -1

Ahora debemos comparar esos pesos con el patrón de entrada de **0101:**

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 1 | 0 | 1 |
| 0 | -1 | 1 | -1 |

Sumaremos sólo los pesos correspondientes a las posiciones que contienen un 1 en el patrón de entrada. Por lo tanto, la activación de la primera neurona es –1 + –1, o –2. Los resultados de la activación de cada neurona se muestran a continuación.

N1 = -1 + -1 = -2

N2 = 0 + 1 = 1

N3 = -1 + -1 = -2

N4 = 1 + 0 = 1

Por lo tanto, las neuronas de salida, que también son las neuronas de entrada, informarán de los resultados de activación anteriores. El vector de salida final es -2, 1, –2, 1. Estos valores no tienen sentido sin una función de activación. Dijimos anteriormente que un umbral establece cuándo se disparará una neurona. Un umbral es un tipo de función de activación. Una función de activación determina el rango de valores que hará que la neurona, en este caso la neurona de salida se dispare. Un umbral es una función de activación simple que se activa cuando la entrada está por encima de un determinado valor.

La función de activación utilizada para una red Hopfield es cualquier valor mayor que cero, por lo que las siguientes neuronas se dispararán. Esto establece el umbral para la red.

El resultado de activación N1 es –2; no disparará (0)

El resultado de activación N2 es 1; se disparará (1)

El resultado de la activación N3 es –2; no se disparará (0) El resultado de activación N4 es 1; se disparará (1)

Como se puede ver, asignamos un valor binario de 1 a todas las neuronas que dispararon, y un valor binario de 0 a todas las neuronas que no dispararon. La salida binaria final de la red Hopfield será **0101**. Esto es lo mismo que el patrón de entrada. Una red neuronal auto asociativa, como una red Hopfield, se hará eco de un patrón de nuevo si se reconoce el patrón. El patrón fue reconocido con éxito. Ahora que ha visto cómo una matriz de peso de conexión puede hacer que una red neuronal recuerde ciertos patrones, se le mostrará cómo se derivó la matriz de peso de conexión.

## Derivar la matriz de peso

Probablementete estés preguntando cómo se desgranó la matriz de peso que se muestra en la Tabla 3.2. Esta sección explicará cómo crear una matriz de peso que puede recordar cualquier número de patrones. Primero debe comenzar con una matriz de peso de conexión en blanco, como se describe en la ecuación 3.1.

### Ecuación 3.1: Una matriz en blanco

[0 0]

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 | 0 |  |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |

Primero entrenaremos esta red neuronal para aceptar el valor **0101**. Para ello, primero hay que calcular una matriz sólo para el **0101**, que se llama **0101**'s matriz de contribución. A continuación, la matriz de contribución se agregará a la matriz de peso de conexión. A medida que se agregan matrices de contribución a la matriz de peso de conexión, se dice que el peso de conexión aprende cada uno de los nuevos patrones.

Comenzamos calculando la matriz de contribución de **0101**. Hay tres pasos involucrados en este proceso. En primer lugar, calculamos los valores bipolares de **0101**. La representación bipolar simplemente significa que estamos representando una cadena binaria con –1 y 1, en lugar de 0 y 1. A continuación, transponemos y multiplicamos el equivalente bipolar de **0101** por sí mismo. Por último, establecemos todos los valores de la diagonal noroeste en cero, porque las neuronas no tienen conexiones consigo mismas en una red hopfield. Tomemos los pasos uno a la vez y veamos cómo se hace esto. Empezaremos con los conversión bipolar.

Paso 1: Convierte 0101 a su equivalente bipolar.

Convertimos la entrada, porque la representación binaria tiene un defecto menor. Cero NO es el inverso de 1. Más bien –1 es el inverso matemático de 1. La ecuación 3.2 se puede utilizar para convertir la cadena de entrada de binario a bipolar.

### Ecuación 3.2: Binario a bipolar

*ḥ* ( *x* )=2x−1

Por el contrario, la ecuación 3.3 se puede utilizar para convertir de bipolar a binario.

### Ecuación 3.3: Bipolar a binario

*ḥ* ( *x* )=( *x*+1)

2

Para convertir 0101 en su equivalente bipolar, convertimos todos los ceros en -1, como se muestra:

0 = –1

1 = 1

0 = –1

1 = 1

El resultado final es la matriz –1, 1, –1, 1. Esta matriz se utilizará en el paso 2 para comenzar a crear la matriz de contribución para 0101.

Paso 2: Multiplique –1, 1, –1, 1 por su transposición.

Para este paso, consideraremos –1, 1, –1, 1 para ser una matriz:

**Ecuación 3.4: Entrada Matriz**

[ ]

−1

1

−1

1

Tomando la transposición de esta matriz tenemos:

**Ecuación 3.5:** **Matriz de** **entrada** **transpuesta**

[−1 1 −1 1 ]

Ahora debemos multiplicar estas dos matrices. La multiplicación de matrices se cubrió en el capítulo 2. Se trata de un procedimiento relativamente sencillo en el que las filas y columnas se comparan entre sí, lo que da como resultado lo siguiente:

|  |  |  |  |
| --- | --- | --- | --- |
| -1 X -1 = 1 | 1 X -1 = -1 | -1 X -1 = 1 | 1 X -1 = -1 |
| -1 X 1 = -1 | 1 X 1 = 1 | -1 X 1 = -1 | 1 X 1 = 1 |
| -1 X -1 = 1 | 1 X -1 = -1 | -1 X -1 = 1 | 1 X -1 = -1 |
| -1 X 1 = -1 | 1 X 1 = 1 | -1 X 1 = -1 | 1 X 1 = 1 |

Condensado, lo anterior da como resultado la siguiente matriz:

**Ecuación 3.6: Resultante Matriz**

[ ]

1 −1 1 −1

−1 1 −1 1

1 −1 1 −1

−1 1 −1 1

Ahora que hemos multiplicado con éxito la matriz por su inversa, estamos listos para el paso 3.

Paso 3: Ajuste la diagonal noroeste a cero.

Matemáticamente hablando, ahora vamos a restar la matriz de identidad de la matriz que derivamos en el paso 2. El resultado neto es que las células en el diago-nal noroeste se establecen en cero. La verdadera razón por la que hacemos esto es que las neuronas no tienen conexiones consigo mismas en las redes hopfield. Por lo tanto, las posiciones **[0][0]**, **[1] [1]**, **[2] [2]** y **[3] [3]** en nuestra matriz bidimensional, o matriz, se establecen en cero. Esto da como resultado la matriz de contribución final para el patrón de bits **0101**.

**Ecuación 3.7: Contribución Matriz**

[ ]

0 −1 1 −1

−1 0 −1 1

1 −1 0 −1

−1 1 −1 0

Esta matriz de contribución ahora se puede agregar a la matriz de peso de conexión. Si solo queremos que esta red reconozca **0101,** esta matriz de contribución se convierte en nuestra matriz de peso de conexión. Si también queremos reconocer **1001**, entonces calcularemos ambas matrices de contribución y sumamos los resultados para crear la matriz de peso de conexión.

Si este proceso parece un poco confuso, puede intentar ver la siguiente sección en la que realmente desarrollamos un programa que crea matrices de peso de conexión. El proceso se explica utilizando la terminología java.

Antes de terminar la discusión de determinar la matriz de peso, se debe mencionar un pequeño efecto secundario. Pasamos por varios pasos para determinar la matriz de peso correcta para **0101**. Cada vez que crea una red Hopfield que reconoce un patrón binario, la red también reconoce lo inverso de ese patrón de bits. Puedes obtener la inversa de un patrón de bits volteando los 0 a 1 y 1 a 0. La inversa de **0101** es **1010.** Como resultado, la matriz de peso de conexión que acabamos de calcular también reconoce **1010**.

# Creación de una red neuronal Java Hopfield

La red neuronal **hopfield** se implementa mediante dos clases. La primera clase, llamada **HopfieldNetwork,** es la clase principal que realiza el entrenamiento y el reconocimiento de patrones. Esta clase se basa en las clases **Matrix** y **MatrixMath,** introducidas en el capítulo 2, para trabajar con la matriz de peso de la red neural. La segunda clase, denominada **HopfieldException**, es una excepción que se genera cuando se produce un error al procesar la red Hopfield. Esto normalmente se desencadena como resultado de una mala entrada.

## La clase HopfieldNetwork

La clase **HopfieldNetwork** se muestra en el listado 3.1.

### Listado 3.1: La red neuronal hopfield (HopfieldNetwork.java)

paquete com.heatonresearch.book.introneuralnet.neural.hopfield;

importación com.heatonresearch.book.introneuralnet.neural.

Excepción. NeuralNetworkError;

importar com.heatonresearch.book.introneuralnet.neural.matrix.

BiPolarUtil;

importar com.heatonresearch.book.introneuralnet.neural.matrix.

Matriz;

importación com.heatonresearch.book.introneuralnet.neural.matrix.

MatrixMath;

/\*\*

* HopfieldNetwork: esta clase implementa una red neuronal hopfield.
* Una red neuronal hopfield está completamente conectada y consiste en un
* una sola capa. Las redes neuronales hopfield se utilizan generalmente para
* reconocimiento de patrones.

\*

* @author Jeff Heaton
* @version 2,1

\*/

clase pública HopfieldNetwork {

/\*\*

* + La matriz de peso para esta red neuronal. Un Hopfield
  + red neuronal es un
  + una sola capa, red neuronal totalmente conectada.

\*

* + Las entradas y salidas hacia/desde una neural hopfield
  + red son siempre valores booleanos.

\*/

peso matrix privadoMatrix;

hopfieldnetwork público (tamaño final int) { this.weightMatrix = nuevo Matrix(size, size);

}

/\*\*

* + Obtenga la matriz de peso para esta red neuronal.

\*

* + @return

\*/

matriz pública getMatrix() { devolver this.weightMatrix;

}

/\*\*

* + Obtenga el tamaño de esta red neuronal.

\*

* + @return

\*/

public int getSize() {

devolver this.weightMatrix.getRows();

}

/\*\*

* + Presentar un patrón a la red neuronal y recibir
  + el resultado.
  + patrón de @param
  + El patrón que se presentará a la red neuronal.
  + @return La salida de la red neuronal.
  + @throws HopfieldException
  + El patrón causó un error matemático de matriz.

\*/

booleano público[] presente(patrón booleano final[] {

salida booleana final[] = nuevo booleano[pattern.length];

convertir el patrón de entrada en una matriz con un

una sola fila.

también convertir los valores booleanos en

bipolar(-1=false, 1=true) final Matrix inputMatrix =

Matrix.createRowMatrix(BiPolarUtil

.bipolar2double(patrón));

Procesar cada valor en el patrón

para (int col = 0; col < pattern.length; col++) { Columna matrizMatrix =

this.weightMatrix.getCol(col);

columnMatrix = MatrixMath.transpose(columnMatrix);

La salida de este elemento de entrada es el producto de puntos de la

matriz de entrada y una columna de la matriz de peso.

punto doble finalProduct = MatrixMath. dotProduct(inputMatrix,

columnMatrix);

Convierta el producto punto en true o false.

if (dotProduct > 0) { output[col] = true;

} else {

output[col] = false;

}

}

retorno de salida;

}

/\*\*

* + Entrene la red neuronal para el patrón especificado.
  + La red neuronal
  + puede ser entrenado para más de un patrón. Para ello
  + simplemente llame a la
  + método de tren más de una vez.

\*

* + patrón de @param
  + El patrón para entrenar.
  + @throws HopfieldException
  + El tamaño del patrón debe coincidir con el tamaño de
  + esta red neuronal.

\*/

tren vacío público (patrón booleano final[] {

if (pattern.length != this.weightMatrix.getRows()) { throw new NeuralNetworkError(

"No se puede entrenar un patrón de tamaño"

+ pattern.length + " en una red hopfield de tamaño "

+ this.weightMatrix.getRows());

}

Cree una matriz de filas a partir de la entrada, convierta

booleano a bipolar

Matriz final m2 = Matrix.createRowMatrix(BiPolarUtil

.bipolar2double(patrón));

Transponer la matriz y multiplicar por el

matriz de entrada original

matriz final m1 = MatrixMath.transpose(m2); matriz final m3 = MatrixMath.multiply(m1, m2);

matriz 3 debe ser cuadrado por ahora, así que crear

una identidad

matriz del mismo tamaño.

identidad final de Matrix = MatrixMath.identity( m3.getRows());

restar la matriz de identidad

matriz final m4 = MatrixMath.subtract(m3, identidad);

ahora agregue la matriz calculada, para este patrón,

a la

matriz de peso existente. this.weightMatrix =

MatrixMath.add(this.weightMatrix, m4);

}

}

Para hacer uso de la red neuronal hopfield, debe crear instancias de una instancia de esta clase. El constructor toma un parámetro entero que especifica el tamaño de la red neuronal. Una vez que se ha instanciado la clase **HopfieldNetwork,** puede llamar a los métodos proporcionados para usar la red neuronal. Estos métodos se resumen en la Tabla 3.3.

### Tabla 3.3: Resumen de los métodos HopfieldNetwork

|  |  |
| --- | --- |
| **Nombre del** método | **Propósito** |
| **getMatrix** | Accede a la matriz de peso de la red neuronal. |
| **getSize** | Obtiene el tamaño de la red neuronal. |
| **Presente** | Presenta un patrón a la red neuronal. |
| **Tren** | Entrena la red neuronal en un patrón. |

Los métodos **getMatrix** y **getSize** son meth-ods de édores muy simples y no se cubrirán más. La mayor parte del trabajo se realiza con los métodos **actuales** y de **tren.** Estos métodos se tratarán en las dos secciones siguientes.

## Recuperación de patrones con la red Java Hopfield

Para recordar un patrón con la clase **HopfieldNetwork,** se debe usar el método **actual.** La firma para este método se muestra aquí:

booleano público[] presente(patrón booleano final[] lanza HopfieldException

El procedimiento para entrenar una red neuronal hopfield ya fue discutido earli- er en este capítulo. La discusión anterior explicaba cómo recordar un patrón mathemati-cally. Ahora veremos cómo implementar el recuerdo del patrón Hopfield en Java.

En primer lugar, se crea una matriz para contener la salida de la red neuronal. Esta matriz tiene la misma longitud que la matriz de entrada.

salida booleana final[] = nuevo booleano[pattern.length];

A continuación, se crea una **matriz** para contener la forma bipolar de la matriz de entrada. El método **bipolar2double** de la clase **BiPolarUtil** se utiliza para convertir la matriz **booleana.** La matriz unidimensional se convierte en una "matriz de filas". Un "matrix de fila" es una **matriz** que consta de una sola fila.

entrada final matrixMatrix = Matrix.createRowMatrix(BiPolarUtil

.bipolar2double(patrón));

Ahora debemos recorrer cada elemento de la matriz de entrada. Cada elemento de la matriz de entrada se aplicará a la matriz de peso para producir una salida.

para (int col = 0; col < pattern.length; col++) {

Cada columna de la matriz de peso representa los pesos asociados con las con-nections entre la neurona y las otras neuronas. Por lo tanto, debemos extraer la columna que corresponde a cada uno de los valores de matriz de entrada.

Columna de matrizMatrix = this.weightMatrix.getCol(col);

Ahora debemos determinar el producto de puntos de esa columna y la matriz de entrada. Nunca, antes de que eso se pueda hacer debemos transponer la columna de la matriz de peso. Esto orientará correctamente los valores de la columna para que se pueda realizar el cálculodel roducto de punto p.

columnMatrix = MatrixMath.transpose(columnMatrix);

A continuación, se calcula el producto punto.

punto doble finalProduct = MatrixMath.dotProduct(inputMatrix, columnMatrix);

Si el producto punto está por encima de cero, entonces la salida será **verdadera;**de lo contrario, el out-put será **false**.

if (dotProduct > 0) { output[col] = true;

} else {

output[col] = false;

}

}

Cero es el umbral. Un valor por encima del umbral hará que la neurona de salida se dispare; en o por debajo del umbral y la neurona de salida no se disparará. Es importante tener en cuenta que dado que la red Hopfield tiene una sola capa, las neuronas de entrada y salida son las mismas. Los umbrales se ampliarán en el siguiente capítulo cuando tratemos con la red neuronal feedforward.

Por último, se devuelve la matriz **de salida.**

retorno de salida;

La siguiente sección explicará cómo se entrena la red Java Hopfield.

## Formación de la Red Hopfield

El método **train** se utiliza para entrenar instancias de la clase **HopfieldNetwork.**

La firma para este método se muestra aquí:

public void train(final boolean[] pattern) lanza HopfieldException

La longitud del patrón debe ser la misma que el tamaño de la red neuronal. Si no es así, se produce una excepción.

if (pattern.length != this.weightMatrix.getRows()) {

lanzar nuevo HopfieldException("No se puede entrenar un patrón de tamaño "

+ pattern.length + " en una red hopfield de tamaño "

+ this.weightMatrix.getRows());

}

Si el patrón es la longitud adecuada, se crea una matriz que contiene una sola fila.

Esta fila contendrá la representación bipolar del patrón de entrada.

Matriz final m2 = Matrix.createRowMatrix(BiPolarUtil

.bipolar2double(patrón));

A continuación, la matriz de filas se transpone a una matriz de columnas.

matriz final m1 = MatrixMath.transpose(m2);

Por último, la columna multiplica la fila.

matriz final m3 = MatrixMath.multiply(m1, m2);

Multiplicar la fila por la columna da como resultado una matriz cuadrada. Esta matriz tendrá una diagonal de las que van desde su esquina noroeste hasta su esquina suroeste. Los que se deben convertir a ceros. Para ello,se crea un matr ix de identidaddel mismo tamaño.

identidad de matriz final = MatrixMath.identity(m3.getRows());

La matriz de identidad no es más que una matriz que contiene una diagonal de unos, que se pueden restar de la matriz perversa para establecer la diagonal en cero.

matriz final m4 = MatrixMath.subtract(m3, identidad);

La nueva matriz ahora se añade a la matriz de peso antigua.

this.weightMatrix = MatrixMath.add(this.weightMatrix, m4);

Esto produce una matriz de peso que probablemente reconocerá el nuevo patrón, así como los patrones antiguos.

# Ejemplo simple de Hopfield

Ahora verá cómo hacer uso de la clase **HopfieldNetwork** que se cre- ated en la última sección. El primer ejemplo implements una aplicación de consola simple que muestra el reconocimiento de patrones básico. El segundo ejemplo muestra gráficamente la matriz de peso utilizando un applet Java. Por último, el tercer ejemplo utiliza un applet Java para ilustrar cómo se puede utilizar una red neuronal hopfield para reconocer un patrón de cuadrícula.

El primer ejemplo, que es una sencilla aplicación de consola, se muestra en el listado 3.2.

### Listado 3.2: Ejemplo de consola simple (ConsoleHopfield.java)

paquete com.heatonresearch.book.introneuralnet.ch3.console;

importar com.heatonresearch.book.introneuralnet.neural.hopfield.Hop- fieldNetwork;

/\*\*

* Capítulo 3: Uso de una red neuronal hopfield

\*

* ConsoleHopfield: sencilla aplicación de consola que muestra cómo
* utilizar una red neuronal hopfield.

\*

* @author Jeff Heaton
* @version 2,1

\*/

clase pública ConsoleHopfield {

/\*\*

* + Convertir una matriz booleana en el formulario [T,T,F,F]

\*

* + @param b
  + Una matriz de boolen.
  + @return la matriz boolen en forma de cadena.

\*/

formato de cadena estática públicaBoolean(booleano final b[]) { resultado final de StringBuilder = nuevo StringBuilder(); result.append('[');

for (int i = 0; i < b.length; i++) { if (b[i]) {

result.append("T");

} else {

result.append("F");

}

si (i != b.length - 1) {

result.append(",");

}

}

resultado.append(']');

return (result.toString());

}

/\*\*

* + Un método principal simple para probar la red neuronal hopfield.

\*

* + args @param
  + No se usa.

\*/

vacío estático público main(final String args[]) {

Cree la red neuronal.

última red HopfieldNetwork = nuevo HopfieldNetwork(4);

Este patrón será entrenado

booleano final[] pattern1 = { true, true, false, false

};

Este patrón se presentará

booleano final[] pattern2 = { true, false, false, false };

booleano[] resultado;

entrenar la red neuronal con pattern1 System.out.println("Training Hopfield network with: "

+ formatBoolean(pattern1)); network.train(pattern1);

patrón actual1 y verlo resultado reconocido = network.present(pattern1); System.out.println("Patrón de presentación:"

+ formatBoolean(pattern1)

+ ", y obtuvo " + formatoBoolean(resultado));

Patrón actual2, que es similar al patrón 1.

El patrón 1 debe ser retirado. resultado = network.present(pattern2);

System.out.println("Patrón de presentación:" + formatBoolean(pattern2)

+ ", y obtuvo " + formatoBoolean(resultado));

}

}

Hay dos métodos proporcionados en el Listado 3.2. El primer método, llamado

**formatBoolean**, se utiliza para dar formato a matrices booleanas de la siguiente manera:

[T,T,F,F]

Este método permite al programa mostrar fácilmente la entrada y la salida de la red neuronal. La función **formatBoolean** es relativamente simple. Recorre en bucle cada elemento de la matriz y muestra una **T** o una **F** en función de si el elemento de matriz es **true** o **false.** Esto se puede ver en el Listado 3.2.

El segundo método, **main,** se utiliza para configurar la red Hopfield y utilizarla. En primer lugar, se crea un nuevo **HopfieldNetwork** con cuatro neuronas.

última red HopfieldNetwork = nuevo HopfieldNetwork(4);

A continuación, se crea un patrón de entrada denominado **pattern1.** Este es el patrón en el que se entrenará la red Hopfield. Dado que hay cuatro neuronas en la red, también debe haber cuatro valores en el patrón de entrenamiento.

booleano final[] pattern1 = { true, true, false, false };

A continuación, se crea un segundo patrón de entrada, denominado **pattern2.** Este patrón es ligeramente diferente al **patrón1.** Este patrón permitirá probar la red para ver si todavía reconoce el **patrón1**, aunque este patrón es ligeramente diferente.

booleano final[] pattern2 = { true, false, false, false };

Se crea una matriz booleana denominada **result** para contener los resultados de la presentación de pat- terns en la red.

booleano[] resultado;

A continuación, se informa al usuario de que estamos formando la red con **pattern1.** El

**formatBoolean** método se utiliza para mostrar **pattern1**.

System.out.println("Formación de la red Hopfield con: "

+ formatBoolean(pattern1));

La red se llama y entrena con **pattern1**.

network.train(pattern1);

Ahora, **el patrón1** se presenta a la red para ver si se reconocerá. Le decimos al usuario que estamos haciendo esto y mostramos el **resultado.**

resultado = network.present(pattern1); System.out.println("Patrón de presentación:" + formatBoolean(pattern1)

+ " y obtuvo " + formatBoolean(result));

A continuación, se presenta **pattern2,** que es similar al **patrón1.** Los mismos valores deben recuperarse para **pattern2** como se recordó para **pattern1**.

resultado = network.present(pattern2); System.out.println("Patrón de presentación:" + formatBoolean(pattern2)

+ " y obtuvo " + formatBoolean(result));

Los resultados se muestran al usuario. La salida de este programa es la siguiente:

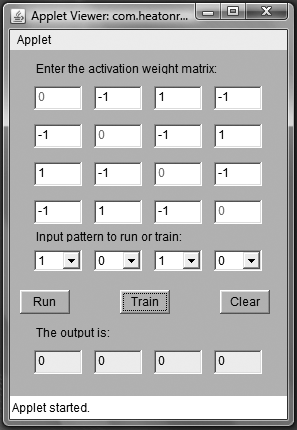
Formación de la red hopfield con: [T,T, F,F] Patrón de presentación: [T, T, F, F] y obtuvo [T,T, F,F] Patrón de presentación:[T,F, F,F] y obtuvo [T,T, F,F,F]

Este programa muestra cómo crear instancias y utilizar una red neuronal hopfield con- fuera cualquier campana o silbatos. El siguiente programa es un applet que le permitirá ver la matriz de peso a medida que la red está entrenada.

# Visualización de la matriz de peso

Este segundo ejemplo es esencialmente el mismo que el primero; sin embargo, este ejemplo utiliza un applet. Por lo tanto, tiene una GUI que permite al usuario interactuar con ella. La interfaz de usuario de este programa se puede ver en la Figura 3.2.

### Figura 3.2: Un applet hopfield



La cuadrícula 4x4 es la matriz de peso. Los cuatro campos "0/1" le permiten introducir patrones de cuatro partes en la red neuronal. Este patrón se puede presentar a la red para ser reconocido o para ser utilizado para el entrenamiento.

El applet Hopfield se muestra en el Listado 3.3.

### Listado 3.3: Visual Hopfield Weight Matrix Applet (HopfieldApplet.java)

paquete com.heatonresearch.book.introneuralnet.ch3.hopfield;

importar java.applet.Applet; importar java.awt.Color;

importar java.awt.event.FocusEvent;

importación com.heatonresearch.book.introneuralnet.neural.hopfield.

HopfieldNetwork;

/\*\*

* Capítulo 3: Uso de una red neuronal hopfield

\*

* HopfieldApplet: Applet que te permite trabajar con un Hopfield
* Red.

\*

* @author Jeff Heaton
* @version 2,1

\*/

clase pública HopfieldApplet extiende Applet implementa java.awt.event.ActionListener, java.awt.event.FocusListener {

/\*\*

* + Id. de serie para esta clase.

\*/

serie larga final estática privadaVersionUID = 624445382526901486L;

HopfieldNetwork network = nuevo HopfieldNetwork(4);

java.awt.Label label1 = nuevo java.awt.Label(); java.awt.TextField matrix[][] = nuevo java.awt.

TextField[4] [4];

entrada java.awt.Choice[] = nueva java.awt.Choice[4]; java.awt.Label label2 = nuevo java.awt.Label(); salida java.awt.TextField[] = nuevo java.awt.TextField[4]; java.awt.Label label3 = nuevo java.awt.Label(); java.awt.Button go = nuevo java.awt.Button(); java.awt.Button train = nuevo java.awt.Button(); java.awt.Button clear = nuevo java.awt.Button();

/\*\*

* + Se llama cuando el usuario hace clic en uno de los botones.

\*

* + @param evento
  + El evento.

\*/

acción pública nulaperformed(

evento java.awt.event.ActionEvent final) {

objeto object final = event.getSource(); if (object == this.go) {

runNetwork();

} si (object == this.clear) { clear();

} si (object == this.train) {

tren();

}

}

/\*\*

* + Despeje la red neuronal.

\*/

void clear() {

this.network.getMatrix().clear(); this.setMatrixValues();

}

/\*\*

* + Recoger los valores de matriz del applet y colocar
  + dentro del peso
  + matriz para la red neuronal.

\*/

private void collectMatrixValues() {

para (fila int = 0; fila < 4; fila++) {

para (int col = 0; col < 4; col++) { String str final =

this.matrix[fila][col].getText(); int valor = 0;

probar {

valor = Integer.parseInt(str);

} catch (numberformatexception final e) {

dejar que el valor predeterminado a cero,

que ya es en este punto.

}

no permita que las neuronas se autoconecte si (fila == col) {

this.network.getMatrix().set( fila, col, 0);

} else {

this.network.getMatrix().set( fila, col, valor);

}

}

}

}

público void focusGained(final FocusEvent e) {

no me importa

}

public void focusLost(final FocusEvent e) { this.collectMatrixValues(); this.setMatrixValues();

}

/\*\*

* + Configura el applet.

\*/ @Override

vacío público init() {

setLayout(null); setBackground(Color.lightGray); this.label1.setText(

"Introduzca la matriz de peso de activación:"); add(this.label1);

this.label1.setBounds(24, 12, 192, 12); this.label2.setText(

"Patrón de entrada para correr o entrenar:"); add(this.label2); this.label2.setBounds(24, 180, 192, 12);

para (fila int = 0; fila < 4; fila++) {

para (int col = 0; col < 4; col++) { this.matrix[fila][col] =

nuevo java.awt.TextField(); add(this.matrix[row][col]); this.matrix[fila][col].setBounds(24

+ (col \* 60),

36 + (fila \* 38), 48, 24);

this.matrix[row][col].setText("0");

this.matrix[fila][col]

.addFocusListener(this);

if (fila == col) {

this.matrix[fila][col]

.setEnabled(false);

}

}

}

para (int i = 0; i < 4; i++) {

this.output[i] = nuevo java.awt.TextField(); this.output[i].setEditable(false);

this.output[i].setText("0"); this.output[i].setEnabled(true); add(this.output[i]); this.output[i].setBounds(24 + (i \* 60),

300, 48, 24);

this.input[i] = nuevo java.awt.Choice(); this.input[i].add("0");

this.input[i].add("1"); this.input[i].select(0); add(this.input[i]); this.input[i].setBounds(24 + (i \* 60),

200, 48, 24);

}

this.label3.setText("La salida es:"); add(this.label3); this.label3.setBounds(24, 276, 192, 12);

this.go.setLabel("Ejecutar"); add(this.go);

this.go.setBackground(java.awt.Color.lightGray); this.go.setBounds(10, 240, 50, 24);

this.train.setLabel("Tren"); add(this.train);

this.train.setBackground(java.awt.Color.lightGray); this.train.setBounds(110, 240, 50, 24);

this.clear.setLabel("Claro"); add(this.clear);

this.clear.setBackground(java.awt.Color.lightGray); this.clear.setBounds(210, 240, 50, 24);

this.go.addActionListener(this); this.clear.addActionListener(this); this.train.addActionListener(this);

}

/\*\*

* + Cotejar la entrada, presentarla a la red neuronal,
  + a continuación, mostrar los resultados.

\*/

void runNetwork() {

patrón booleano final[] = nuevo booleano[4];

Lea la entrada en una matriz booleana. para (fila int = 0; fila < 4; fila++) {

final int i = this.input[row].getSelectedIndex(); si (i == 0) {

patrón[fila] = false;

} else {

patrón[fila] = verdadero;

}

}

Presente la entrada a la red neuronal.

resultado booleano final[] = this.network.present(pattern);

Visualice el resultado.

para (fila int = 0; fila < 4; fila++) { si (resultado[fila]) {

this.output[row].setText("1");

} else {

this.output[row].setText("0");

}

Si el resultado es diferente de la entrada, mostrar en amarillo. if (resultado[fila] == patrón[fila]) {

this.output[row].setBackground( java.awt.Color.white);

} else {

this.output[row].setBackground( java.awt.Color.yellow);

}

}

}

/\*\*

* + Establezca los valores de matriz en el applet de la matriz
  + valores almacenados en la red neuronal.

\*/

private void setMatrixValues() {

para (fila int = 0; fila < 4; fila++) {

for (int col = 0; col < 4; col++) { this.matrix[row][col].setText(""

+ (int) this.network. getMatrix().get(row, col));

}

}

}

/\*\*

* + Se llama cuando se hace clic en el botón de tren. Tren para
  + el patrón actual.

\*/

tren vacío() {

booleano final[] booleanoInput = nuevo booleano[4];

Recopile el patrón de entrada. para (int x = 0; x < 4; x++) {

booleanInput[x] = (this.input[x]. getSelectedIndex() != 0);

}

Entrene el patrón de entrada. this.network.train(booleanInput); this.setMatrixValues();

}

}

Este applet se puede ejecutar en línea desde la mayoría de los navegadores accediendo a los siguientes

Url:

[**http://www.heatonresearch.com/articles/61/page1.html**](http://www.heatonresearch.com/articles/61/page1.html)

Para utilizar el applet, siga estos pasos:

1. Observe que la matriz de peso de activación está vacía (contiene todos los ceros). Esta red neuronal no tiene conocimiento. Enseñemos a reconocer el patrón 1001. Escriba 1001 en la solicitud "Patrón de entrada para ejecutar o entrenar:". Haga clic en el botón "Entrenar". Observe que la matriz de peso se ajusta para absorber el nuevo conocimiento.
2. Ahora pruébalo. Introduzca el patrón 1001 en la solicitud "Entrada pat- tern para correr o entrenar:". (Todavía debería estar allí desde su entrenamiento.) Ahora haga clic en "Ejecutar." La salida será "1001." Esta es una red autoasociativa, por lo tanto se hace eco de la entrada si la reconoce.
3. Vamos a probarlo un poco más. Introduzca el patrón 1000 y haga clic en "Ejecutar." La salida ahora será "1001." La red neuronal no reconocía "1000;" lo más parecido que sabía era "1001". Se imaginó que hizo un error escribiendo e intentó una corrección!
4. Ahora, observe un efecto secundario. Ingrese "0110", que es el bina- ry inverso de lo que la red fue entrenada con ("1001"). Las redes hopfield SIEMPRE se entrenan para la inversa binaria de un patrón. Por lo tanto, si usted ingresa "0110", la red lo reconocerá.
5. Del mismo modo, si entras en el "0100", la red neuronal superará al "0110" pensando que eso es lo que querías decir.
6. Una prueba final: probemos "1111", que está totalmente fuera de la base y no está en absoluto cerca de nada que la red neuronal sepa. La red neuronal responde con "0000". No trató de corregirte. ¡No tiene idea de lo que quieres decir!
7. Juega con él un poco más. Se puede enseñar múltiples patrones. A medida que entrenas nuevos patrones, sebasa en la matriz que ya está en la memoria. Al pulsar "Borrar", se borra la memoria.

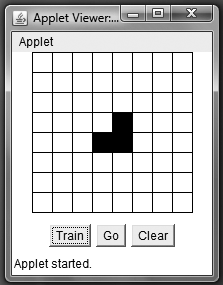
La red hopfield funciona exactamente de la misma manera que lo hizo en el primer ejemplo. Los charranes pat-terns se presentan a una red neuronal hopfield de cuatro neuronas para el entrenamiento y la regnición. Todo el código adicional que se muestra en el listado 3.3 simplemente conecta la red hopfield a una GUI usando el Java Swing and Abstract Windowing Toolkit (AWT).

# Applet de reconocimiento de patrones hopfield

Las redes hopfield pueden ser mucho más grandes que cuatro neuronas. Para el tercer ejemplo, examinaremos una red Hopfield de 64 neuronas. Esta red está conectada a una cuadrícula 8x8, que un applet le permite dibujar. A medida que dibuja patrones, puede entrenar el network con ellos o presentarlos para el reconocimiento.

La interfaz de usuario del applet se puede ver en la figura 3.3.

### Figura 3.3: Un applet Hopfield de reconocimiento de patrones.



El código fuente de este ejemplo se proporciona en el listado 3.4.

### Listado 3.4: Reconocimiento de patrones de salto (PatternApplet.java)

paquete com.heatonresearch.book.introneuralnet.ch3.pattern;

importar java.applet.Applet; importar java.awt.BorderLayout; importar java.awt.Button; importar java.awt.Graphics; importar java.awt.Panel;

impuerto java.awt.event.ActionEvent; importar java.awt.event.ActionListener; importar java.awt.event.MouseEvent; importar java.awt.event.MouseListener;

importar com.heatonresearch.book.introneuralnet.neural.hopfield.Hop- fieldNetwork;

/\*\*

* Capítulo 3: Uso de una red neuronal hopfield

\*

* PatternApplet: un applet que muestra una cuadrícula 8X8 que
* presenta patrones a un Hopfield
* red neuronal.

\*

* @author Jeff Heaton
* @version 2,1

\*/

la clase pública PatternApplet amplía los implementos de Applet

MouseListener, ActionListener {

/\*\*

\* Id. de serie para esta clase.

\*/

serie larga final estática privadaVersionUID = 1882626251007826502L;

final público int GRID\_X = 8; final público int GRID\_Y = 8; final público int CELL\_WIDTH = 20; final público int CELL\_HEIGHT = 20; hopfieldNetwork hopfield público; cuadrícula booleana privada[];

margen int privado; botón de panel privadoPanel;

botón de botón privadoTrain; botón de botón privadoGo; botón de botón privadoClear;

botón de botón privadoClearMatrix;

acción pública voidPerformed(final ActionEvent e) { if (e.getSource() == this.buttonClear) {

clear();

} si (e.getSource() == this.buttonClearMatrix) { clearMatrix();

} más si (e.getSource() == this.buttonGo) { go();

} si (e.getSource() == this.buttonTrain) { train();

}

}

/\*\*

\* Despeje la rejilla.

\*/

public void clear() { int index = 0;

para (int y = 0; Y < esto. GRID\_Y; y++) {

para (int x = 0; x < esto. GRID\_X; x++) { this.grid[index++] = false;

}

}

repintado();

}

/\*\*

\* Borre la matriz de peso.

\*/

vacío privado clearMatrix() { this.hopfield.getMatrix().clear();

}

/\*\*

\* Ejecute la red neuronal.

\*/

público void go() {

this.grid = this.hopfield.present(this.grid); repintado();

}

@Override

vacío público init() {

this.grid = nuevo booleano[this. GRID\_X \* esto. GRID\_Y]; this.addMouseListener(this);

this.buttonTrain = nuevo Button("Train"); this.buttonGo = nuevo Button("Go"); this.buttonClear = nuevo botón ("Borrar");

this.buttonClearMatrix = nuevo botón ("Borrar matriz"); this.setLayout(new BorderLayout()); this.buttonPanel = nuevo Panel(); this.buttonPanel.add(this.buttonTrain); this.buttonPanel.add(this.buttonGo); this.buttonPanel.add(this.buttonClear); this.buttonPanel.add(this.buttonClearMatrix); this.add(this.buttonPanel, BorderLayout.SOUTH);

this.buttonTrain.addActionListener(this); this.buttonGo.addActionListener(this); this.buttonClear.addActionListener(this); this.buttonClearMatrix.addActionListener(this);

this.hopfield = nuevo HopfieldNetwork(this. GRID\_X \* esto. GRID\_Y);

}

mouse void públicoClicked(evento final MouseEvent) {

no usard

}

ratón vacío públicoEntered(final MouseEvent e) {

no utilizado

}

ratón vacío públicoExited(final MouseEvent e) {

no utilizado

}

público void mousePressed(final MouseEvent e) {

no utilizado

}

ratón vacío públicoReleased(final MouseEvent e) { final int x = ((e.getX() - this.margin) / this. CELL\_WIDTH);

final int y = e.getY() / this. CELL\_HEIGHT;

if (((x >= 0) y & (x < esto. GRID\_X)) && ((y >= 0) && (y < esto. GRID\_Y))) {

índice int final = (y \* esto. GRID\_X) + x; this.grid[index] = !this.grid[index];

}

repintado();

}

@Override

pintura del vacío público (gráficos finales g) { this.margin = (this.getWidth() -

(esto. CELL\_WIDTH \* esto. GRID\_X)) / 2; int index = 0;

para (int y = 0; Y < esto. GRID\_Y; y++) {

para (int x = 0; x < esto. GRID\_X; x++) { si (this.grid[index++]) {

g.fillRect(this.margin + (x \* esto. CELL\_WIDTH), y

\* esto. CELL\_HEIGHT, esto. CELL\_WIDTH,

éste. CELL\_HEIGHT);

} else {

g.drawRect(this.margin + (x

* éste. CELL\_WIDTH), y
* éste. CELL\_HEIGHT, esto. CELL\_WIDTH,

éste. CELL\_HEIGHT);

}

}

}

}

/\*\*

\* Entrenar la red neuronal.

\*/

tren vacío público() { this.hopfield.train(this.grid);

}

}

Este applet se puede ejecutar en línea desde la mayoría de los navegadores accediendo a la siguiente URL:

[**http://www.heatonresearch.com/articles/61/page1.html**](http://www.heatonresearch.com/articles/61/page1.html)

Para hacer uso del applet, dibuje algún tipo de patrón en la cuadrícula y haga clic en "Entrenar". Ahora dibuje otro patrón y también haga clic en "Entrenar". Por último, dibuje un patrón similar a uno de los dos patrones anteriores en la misma ubicación y haga clic en "Ir". La red intentará reconocer lo que dibujó. También puede hacer clic en "Borrar" para borrar la cuadrícula. Haga clic en "Borrar matriz" borrará la matriz de entrenamiento.

En las secciones siguientes se explica cómo se construyeron los principales métodos de este programa.

**Dibujar la** **cuadrícula**

La rejilla se dibuja utilizando el método de **pintura** del applet. La firma de este meth-od se muestra aquí:

pintura del vacío público (gráficos finales g)

El tamaño de la cuadrícula se define mediante varias constantes. A menos que cambies esto, la cuadrícula en sí será 8x8 y las celdas serán de 20 píxeles cuadrados. Esto significa que los botones ocuparán más espacio que la cuadrícula; por lo tanto, se ve mejor si la cuadrícula está centrada. Se calcula un **margen** para centrar la cuadrícula.

El margen es el ancho real mínanos el ancho de la cuadrícula, dividido por dos.

this.margin = (this.getWidth() - (esto. CELL\_WIDTH \* esto. GRID\_X)) / 2;

A continuación, se crea la variable de **índice.** Esta variable mantiene la posición actual en la matriz **booleana** **de cuadrícula.** Aunque la cuadrícula aparece visualmente como una matriz 8x8, debe presentarse en la red neuronal como un patrón booleano plano de 64 partes.

int index = 0;

A continuación, se establecen dos bucles para recorrer en bucle las coordenadas **x** e **y** de la cuadrícula.

para (int y = 0; y < esto. GRID\_Y; y++) { para (int x = 0; x < esto. GRID\_X; x++) {

Si el elemento de cuadrícula de esta celda es **true,** dibuje un rectángulo y rellene.

if (this.grid[index++]) {

g.fillRect(this.margin + (x \* esto. CELL\_WIDTH), y

\* esto. CELL\_HEIGHT, esto. CELL\_WIDTH, esto. CELL\_HEIGHT);

Si el elemento de cuadrícula de esta celda es **false** y, a continuación, dibuje un rectángulo vacío.

} else {

g.drawRect(this.margin + (x \* esto. CELL\_WIDTH), y

\* esto. CELL\_HEIGHT, esto. CELL\_WIDTH, esto. CELL\_HEIGHT);

}

}

}

Java llama al método **paint** cada vez que la cuadrícula necesita ser redibujada. Además, otros métodos obligarán a llamar al método **de pintura** mediante el método **de repintado.**

## Alternar posiciones de cuadrícula

Java llama al método **mouseReleased** cada vez que se ha liberado el mouse. Por supuesto, el ratón debe haber sido presionado primero. La firma para el método **mouseReleased** se muestra aquí:

ratón vacío públicoReleased(final MouseEvent e)

En primer lugar, las coordenadas **x** e **y** se calculan en términos de la posición de cuadrícula y no de las coordenadas de píxel que se pasan al evento **mouseReleased.**

final int x = ((e.getX() - this.margin) / this. CELL\_WIDTH); final int y = e.getY() / this. CELL\_HEIGHT;

Estas coordenadas deben caer sobre la cuadrícula. Si el clic estaba fuera de la cuadrícula, se omitirá el clic.

if (((x >= 0) y & (x < esto. GRID\_X)) && ((y >= 0) && (y < esto. GRID\_Y))) {

El **índice** en la matriz de cuadrícula unidimensional se calcula, a continuación, se alterna mediante el booleano **!** operador.

índice int final = (y \* esto. GRID\_X) + x; this.grid[index] = !this.grid[index];

}

Por último, el applet vuelve a pintar la cuadrícula.

repintado();

Por lo tanto, la cuadrícula se puede dibujar mediante el seguimiento cuando se libera el ratón.

## Formación y presentación de patrones

Los botones "Entrenar" e "Ir" le permiten entrenar y reconocer patrones respectivamente. Se llama al método **de tren** cada vez que se hace clic en el botón "Entrenar". La firma del método de **tren** se muestra aquí:

tren público vacío()

El método **de tren** de la red neuronal se llama con la variable de **cuadrícula.**

probar {

this.hopfield.train(this.grid);

Es muy poco probable que se lance una **HopfieldException,** ya que el applet está diseñado para enviar cuadrículas del tamaño correcto. Sin embargo, dado que **hopfieldException** es una excepción comprobada, debe controlarse.

} catch (final HopfieldException e) { e.printStackTrace();

}

}

Se realiza un control mínimo de errores para la excepción. La pila se va a la consola.

El método **go** se llama cada vez que se hace clic en el botón "Ir". La firma de la

**go** método se muestra aquí:

vacío público go()

La cuadrícula se presenta a la red neuronal para su procesamiento. Los resultados del **método actual** se copian en la variable de **cuadrícula.** Al volver a **pintar** se muestran los resultados del reconocimiento de red.

probar {

this.grid = this.hopfield.present(this.grid); repintado();

Una vez más, es muy poco probable que se lance una **HopfieldException,** ya que el applet está diseñado para enviar cuadrículas del tamaño correcto, pero debe controlarse.

} catch (final HopfieldException e) { e.printStackTrace();

}

}

Y de nuevo, se realiza un control mínimo de errores para la excepción. La pila se va a la consola.

# Resumen del capítulo

Las redes neuronales son uno de los conceptos más utilizados en Intelli- Gence artificial. Las redes neuronales son particularmente útiles para reconocer patrones. Son capaces de reconocer algo incluso cuando está distorsionado.

Una red neuronal puede tenercapas de entrada, salida y ocultas. Las capas de entrada y out-put son las únicas capas requeridas. Las capas de entrada y salida pueden ser las mismas neuronas. Las redes neuronales normalmente se presentan patrones de entrada que producirán algún patrón de salida.

Si una red neuronal mimics el patrón de entrada que se presentó con, entonces se dice que esa red es autoassociativa. Por ejemplo, si se presenta una red neuronal con el patrón "0110", y la salida también es "0110", entonces se dice que esa red es autoassociativa.

Un ne twork neural calcula su salida en función del patrón de entrada y la matriz de peso de conexión interna de la redneuronal. Los valores de los pesos de conexión determinarán la salida de la red neuronal.

Una red neuronal hopfield es una red neuronal autoassociativa totalmente conectada. Esto significa que cada neurona está conectada a todas las demás neuronas de la red neuronal. Una red neuronal hopfield se puede entrenar para reconocer ciertos patrones. Entrenar una red neuronal hopfield implica realizar algunas manipulaciones básicas de matriz en el patrón de entrada que se va a reconocer.

En este capítulo se explica cómo construir una red neuronal hopfield simple. Esta red fue entrenada usando un algoritmo de entrenamiento simple. Losalgoritmos Training permiten ajustar los pesos para producir las salidas deseadas. Hay muchos algoritmos avanzados de tren. El capítulo 4 introducirá algoritmos de entrenamiento más complejos.

# Vocabulario

Función de activación Autoasociación Bipolar

Red neuronal hopfield Red neuronal de una sola capa

# Preguntas para revisión

* 1. Una red neuronal típica de Hopfield contiene seis neuronas. ¿Cuántas connec- ciones producirá esto?
  2. Convierte 1 binario a bipolar.
  3. Convertir -1 bipolar a binario.
  4. Considerar a cuatro neuronas Hopfield Neural Red Con el Siguientes Peso ma- Trix.

[ ]

0 1 −1 −1

1 0 −1 −1

−1 −1 0 1

−1 −1 1 0

¿Qué salida producirá una entrada de 1101?

* 1. Considere una red hopfield de cuatro neuronas. Produzca una matriz de peso que reconozca el patrón 1100.