**CHAPTER** **7:** **UNDERSTANDING** **SIMULATED** **ANNEALING**

* ¿Qué es el recocido simulado?
* Para ¿Qué se utiliza el recocido simulado?
* Implementación de recocido simulado en Java
* Aplicación de recocido simulado al problema del vendedor ambulante

En el capítulo 6, "Comprensión de algoritmos genéticos", se introdujo en algoritmos genéticos y cómo se pueden utilizar para entrenar una red neuronal. En este capítulo aprenderás sobre otro algoritmo popular que puedes usar, recocido simulado. Como verás, también se puede aplicar a otras situaciones.

El programa de muestra que se presentará en este capítulo resuelve el problema de los vendedores ambulantes, al igual que el algoritmo genético en el capítulo 6. Sin embargo, en este programa, el recocido simulado se utilizará en lugar del algoritmo genético. Esto le permitirá ver algunas de las ventajas que el recocido simulado ofrece sobre un algoritmo genético.

Comenzaremos con un fondo general del proceso de recocido simulado. A continuación, construiremos una clase que es capaz de utilizar recocido simulado para resolver el problema del vendedor ambulante. Por último, exploraremos cómo se puede utilizar el recocido simulado para entrenar una red neuronal.

# Fondo de recocido simulado

El recocido simulado fue desarrollado a mediados de la década de 1970 por Scott Kirkpatrick y varios otros investigadores. Fue desarrollado originalmente para optimizar mejor el diseño de chips de circuito integrado (IC) simulando el proceso real de recocido.

El recocido es el proceso metalúrgico de calentar un sólido y luego enfriarlo lentamente hasta que cristaliza. Los átomos de estos materiales tienen valores de alta energía a temperaturas muy altas. Esto da a los átomos una gran cantidad de libertad en su capacidad para restructure así mismos. A medida que se reduce la temperatura, los niveles de energía de los átomos disminuyen. Si el proceso de enfriamiento se lleva a cabo demasiado rápido, se verán muchas irregularidades y defectos en la estructura del cristal. El proceso de enfriamiento demasiado rápido es como un *rapid quenching*. Idealmente, la temperatura debe reducirse lentamente para permitir que se forme una estructura de cristal más consistente y estable, lo que aumentará la durabilidad del metal.

El recocido simulado busca emular este proceso. Comienza en una muy alta temperatura, en la que se permite que los valores de entrada asuman una amplia gama de valores aleatorios. A medida que avanza el entrenamiento, se permite que la temperatura baje, restringiendo así el grado en que se permite que las entradas varíen. Esto a menudo conduce el algoritmo de recocido simulado a una mejor solución, del mismo modo que un metal logra una mejor estructura de cristal a través del proceso de recocido real.

## Aplicaciones de recocido simulado

Dado un número especificado de entradas para una ecuación arbitraria, el recocido simulado se puede utilizar para determinar aquellas entradas que producirán el resultado mínimo para la ecuación. En el caso del vendedor viajero, esta ecuación es el cálculo de la distancia total que debe recorrer el vendedor. Como aprenderemos más adelante en este capítulo, la ecuación es la función de error de una red neuronal.

Cuando se introdujo por primera vez el recocido simulado, el algoritmo era muy popular para el diseño de chip de circuito integrado (IC). La mayoría de los chips IC se componen de muchas puertas lógicas internas. Estas puertas permiten que el chip realice las tareas que fue diseñado para realizar. Así como las ecuaciones algebraicas a menudo se pueden simplificar, también lo pueden hacer los tendidos de chip IC. El recocido simulado se utiliza a menudo para encontrar un diseño de chip IC que tiene menos puertas lógicas que el original. El resultado es un chip que genera menos calor y funciona más rápido.

La matriz de peso de una red neuronal proporciona un excelente conjunto de entradas para el algoritmo de recocido simulado para minimizar. Diferentes conjuntos de pesos se utilizan para la red neuronal hasta que se encuentra uno que produce un retorno suficientemente bajo de la función error.

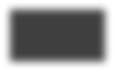
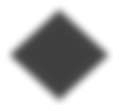
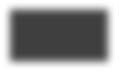
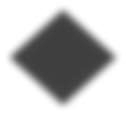
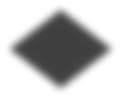
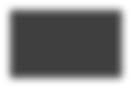
# Comprensión del recocido simulado

Las secciones anteriores discutieron el fondo del recocido simulado algo-rithm y presentaron varias aplicaciones para las que se utiliza. En esta sección, se le mostrará cómo implementar el algoritmo de recocido simulado. Primero examinaremos el algoritmo y luego desarrollaremos una clase de algoritmo de recocido simulada que se puede utilizar para resolver el problema del vendedor ambulante, que se introdujo en el capítulo 6.

## La estructura de un algoritmo de recocido simulado

Hay varios pasos distintos por los que debe pasar el proceso de recocido simulado a medida que se reduce la temperatura y se aplica randomness a los valores de entrada. La Figura 7.1 presenta un diagrama de flujo de este proceso.

### Figura 7.1: Visión general del proceso de recocido simulado.



Empezar

Mejor que

solución actual?

Sí

No

No

Alcanzado máximo

intentos por esta temperatura?

Sí

No

bajar

temperatura atada a alcanzar?

Sí

parar

Disminuya la temperatura en la velocidad de specified.

Reemplace la solución actual por una nueva solución.

Aleatorizar según la temperatura actual.

Como se puede ver en la figura 7.1, hay dos procesos principales que tienen lugar en el algoritmo de recocido simulado. En primer lugar, para cada temperatura, el algoritmo de recocido simulado se ejecuta a través de una serie de ciclos. El número de ciclos está predeterminado por el programador. A medida que se ejecuta un ciclo, las entradas se aleatorizan. En el caso del problema de los vendedores ambulantes, estos insumos son el orden de las ciudades que visitará el vendedor ambulante. Solo se conservarán las aleatorizaciones que produzcan un conjunto de entradas más adecuado.

Una vez completado el número especificado de ciclos de entrenamiento, se puede reducir la temperatura. Una vez bajada la temperatura, se determina si la temperatura ha alcanzado o no la temperatura más baja permitida. Si la temperatura no es inferior a la temperatura más baja permitida, entonces la temperatura se baja y se llevará a cabo otro ciclo de aleatorizaciones. Si la temperatura es inferior a la temperatura más baja permitida, finaliza el algoritmo recocido simulado.

En el núcleo del algoritmo de recocido simulado está la aleatorización de los valores de entrada. Esta aleatorización es en última instancia lo que hace que el recocido simulado altere los valores de entrada que el algoritmo está tratando de minimizar. El proceso de aleatorización a menudo debe personalizarse para diferentes problemas. En este capítulo discutiremos los métodos de aleatorización que se pueden utilizar tanto para el problema del vendedor viajero como para la capacitación de la red neuronal. En la siguiente sección, examinaremos cómo se produce la aleatorización.

## ¿Cómo se aleatorizan las entradas?

Una parte importante del proceso de recocido simulado es cómo se ejecutaron las entradas. El proceso de aleatorización toma los valores de entrada anteriores y la temperatura actual como entradas. A continuación, los valores de entrada se aleatorizan según la templanza. Una temperatura más alta resultará en más aleatorización; una temperatura más baja resultará en menos aleatorización.

No hay ningún método específico definido por el algoritmo de recocido simulado para cómo aleatorizar las entradas. La naturaleza exacta por la que esto se hace a menudo depende de la naturaleza del problema que se está resolviendo. Al comparar los métodos utilizados en los ejemplos de recocido simulado para la optimización weight de la red neuronal y el problema del vendedor viajero, podemos ver algunas de las diferencias.

## Recocido simulado y redes neuronales

El método utilizado para aleatorizar los pesos de una red neuronal es algo más sim-pler que el algoritmo de recocido simulado del vendedor viajero, que vamos a des-cuss siguiente. La matriz de peso de una red neuronal se puede considerar como una matriz lineal de números de punto flotante. Cada peso es independiente de los demás. No importa si dos pesos contienen el mismo valor. La única restricción importante es que hay rangos en los que todos los pesos deben estar dentro.

Por lo tanto, el proceso generalmente utilizado para aleatorizar la matriz de peso de un trabajo de red neuronal es relativamente simple. Usando la temperatura, se aplica una relación aleatoria a todos los pesos de la matriz. Esta relación se calcula utilizando la temperatura y un número aleatorio. Cuanto mayor sea la temperatura, más likely es que la relación causará un cambio mayor en la matriz de peso. Lo más probable es que una temperatura más baja produzca un cambio menor. Este es el método que se utiliza para el algoritmo de recocido simulado que se presentará más adelante en este capítulo.

# Recocido simulado y el problema del vendedor ambulante

El método utilizado para aleatorizar la ruta del vendedor viajero es algo más complejo que el método utilizado para aleatorizar los pesos de una red neuronal. Esto se debe a que existen restricciones en la ruta del problema del vendedor ambulante que no existen al optimizar la matriz de peso de la red neuronal. La restricción más significativa es que la aleatorización del camino debe ser lo suficientemente consensúada como para evitar que el vendedor ambulante visite la misma ciudad másde una vez, y al mismo tiempo asegurarse de que visita cada ciudad una vez; ninguna ciudad puede ser omitida.

Puede pensar en la aleatorización de vendedores ambulantes como la reordenación de ele- ments en una lista de tamaño fijo. Esta lista de tamaño fijo es la ruta de acceso que debe seguir el vendedor viajero. Dado que el vendedor ambulante no puede saltar ni volver a visitar las ciudades, su camino siempre tendrá el mismo número de "paradas" que hay ciudades.

Como resultado de las restricciones impuestas por el problema del vendedor ambulante, la mayoría de los métodos de aleatorización utilizados para este problema cambian el orden de la ruta anterior a través de las ciudades. Al simply reorganizar los datos, y no modificar los valores originales, podemos estar seguros de que el resultado final de esta reorganización no saltará, ni revisará las ciudades.

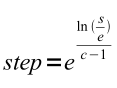
Este es el método que se usa para aleatorizar la ruta del vendedor viajero en el ejemplo de este capítulo. Utilizando una combinación de la temperatura y la distancia entre dos ciudades, el algoritmo de recocido simulado determina si se deben cambiar las posiciones de las dos ciudades. Verá la implementación real de Java de esta metanfetaminamás adelante en este capítulo.

## Reducción de la temperatura

Hay varios métodos diferentes que se pueden utilizar para la reducción de la temperatura; examinaremos dos. El más común es simplemente reducir la temperatura en una cantidad fija a través de cada ciclo. Este es el método que se utiliza en este capítulo para el problema del vendedor viajero.

Otro método es especificar un principio y final temperature. Para utilizar este método, debemos calcular una relación en cada paso del proceso de recocido simulado. Esto se hace mediante el uso de una ecuación que garantiza que la cantidad de paso hará que la temperatura caiga a la temperatura final en el número de ciclos especificados. La ecuación 7.1 describe cómo disminuir logarítmicamente la temperatura entre una temperatura inicial y la temperatura final. Calcula la relación y asegura que el tem- perature disminuya naturalmente para cada ciclo.

### Ecuación 7.1: Escala de la temperatura



ln ( *s* )

*E*

*paso* =*e* *c*−1

Las variables son **s** para iniciar la temperatura, **e** para terminar la temperatura, y **c** para el recuento de ciclos. La ecuación se puede implementar en Java de la siguiente manera:

relación doble = Math.exp(Math.log(stopTemperature/ startTemperature)/(cycles-1));

La línea anterior calcula una relación que debe multiplicarse contra la temperatura actual. Esto producirá un cambio que hará que la temperatura alcance la temperatura final en el número especificado de ciclos. Este método se utiliza más adelante en este capítulo cuando el recocido simulado se aplica al entrenamiento de la red neuronal.

# Implementación de recocido simulado

El código fuente que acompaña a este libro proporciona una clase de recocido simulado genérica. Esta clase abstracta se denomina **SimulatedAnnealing** y se puede usar para implementar una solución de recocido simulada para una variedad de problemas. Usaremos esta clase de recocido simulada tanto para el ejemplo de red neuronal como para el problema del vendedor ambulante.

En esta sección se describirá cómo funciona la clase **genérica SimulatedAnnealing.** La aplicación del recocido simulado a las redes neuronales y el problema de viajar salesman se cubrirán más adelante en este capítulo.

## Entradas al algoritmo de recocido simulado

Hay varias variables que se deben establecer en la clase **SimulatedAnnealing** para que funcione correctamente. Estas variables suelen ser establecidas por el constructor de una de las clases que subclase la clase **SimulatedAnnealing.**

### Tabla 7.1: Entradas de recocido simuladas

|  |  |
| --- | --- |
| **variable** | **propósito** |
| startTemperature | La temperatura a la que hay que empezar. |
| stopTemperature | La temperatura a la que parar. |
| Ciclos | El número de ciclos a utilizar. |

El algoritmo de entrenamiento de recocido simulado funciona muy parecido a cualquier otro algoritmo de entrenamiento en este libro. Una vez configurado, progresa a través de una serie de iteraciones.

## Procesamiento de iteraciones

La clase **SimulatedAnnealing** contiene un método denominado **iteración**

que está llamado a procesar cada iteración del proceso de formación.

iteración del vacío público() lanza NeuralNetworkError {

En primer lugar, se crea una matriz para contener la mejor solución.

UNIT\_TYPE mejorArray[];

A continuación, se determina el error inicial.

setError(determineError()); bestArray = this.getArrayCopy();

A continuación, el proceso de entrenamiento se recorre en un ciclo a través de un número especificado de veces. Para cada pase de entrenamiento, se llama al método **randomize.** Este método es abstracto y debe implementarse para cualquier problema que se va a resolver mediante recocido simulado.

para (int i = 0; i < this.cycles; i++) { doble curError;

randomize();

El error se determina después de llamar a **randomize.**

curError = determineError();

Si se trataba de una mejora, se guarda la matriz recién creada.

if (curError < getError()) { bestArray = this.getArrayCopy(); setError(curError);

}

}

Una vez completado el ciclo, se almacena la mejor matriz.

this.putArray(bestArray);

Se calcula una relación que disminuirá la temperatura al nivel deseado. Esta es la implementación java de la ecuación 7.1, que se mostró anteriormente.

relación doble final = Math.exp(Math.log(getStopTemperature()

/ getStartTemperature())

/ (getCycles() - 1));

La temperatura se escala en esta cantidad.

relación this.temperature \*=;

Esta clase de recocido simulada es, por supuesto, abstracta; por lo tanto, sólo implementa el algoritmo de recocido simulado a un nivel primitivo. Los ejemplos de este capítulo que realmente ponen la clase **SimulatedAnnealing** para usar deben implementar la función **randomize** para sus situaciones únicas.

# Recocido simulado para el problema del vendedor ambulante

El recocido simulado puede proporcionar soluciones potenciales al problema del vendedor ambulante. El problema de los vendedores ambulantes se introdujo en el capítulo 6, "Entender- ing algoritmos genéticos." Aparte del hecho de que el problema de vendedor ambulante de este capítulo utiliza recocido simulado, es el mismo que el programa presentado en el capítulo 6.

El problema del vendedor ambulante de recocido simulado implementa una versión spe- cial de la clase **SimulatedAnnealing.** El método más importante es el método **randomize.** La firma para el método **randomize** se muestra aquí:

aleatorización del vacío público()

En primer lugar, se determina la **longitud** de la ruta.

longitud final int = this.path.length;

A continuación, iteramos a través del bucle varias veces igual a la temperatura. Cuanto mayor sea la temperatura, más iteraciones. Cuantos más iteraciones, más "excitado" se vuelve el camino subyacente y más cambios se realizan.

para (int i = 0; i < this.temperature; i++) {

Se eligen dos ubicaciones de índice aleatorio dentro de la ruta de acceso.

int index1 = (int) Math.floor(length \* Math.random()); int index2 = (int) Math.floor(length \* Math.random());

Un número de distancia básico se calcula en función de los dos puntos aleatorios.

final doble d = distancia (index1, index1 + 1)

+ distancia (index2, index2 + 1)

* distancia (index1, index2)
* distancia (index1 + 1, index2 + 1);

Si el cálculo de distancia es mayor que cero, los elementos de matriz de la ruta de acceso se excitan.

if (d > 0) {

Las ubicaciones del índice, **index1** e **index2**, se ordenan si es necesario.

if (index2 < index1) {

final int temp = index1; index1 = index2;

index2 = temp;

}

Todas las ciudades entre los dos puntos de índice aleatorios están ordenadas.

para (; index2 > index1; index2--) {

final int temp = this.path[index1 + 1]; this.path[index1 + 1] = this.path[index2]; this.path[index2] = temp;

index1++;

}

Cuando la nueva ruta de acceso se devuelve a la función de **iteración,** se evaluará.

Si se trata de una mejora en la última ruta, se conservará.

# Recocido simulado para redes neuronales

El recocido simulado también se puede aplicar a las redes neuronales. Este libro proporciona una clase denominada **NeuralSimulatedAnnealing**. Al hacer uso de la clase genérica **SimulatedAnnealing** introducida anteriormente en este capítulo, esta clase incluye una solución de entrenamiento para redes neuronales. La lista 7.1 muestra un programa sencillo que utiliza recocido simulado para entrenar una red neuronal para el operador XOR.

### Listado 7.1: Annealing simulado y el operador XOR (AnnealXOR.java)

paquete com.heatonresearch.book.introneuralnet.ch7.xor;

importar com.heatonresearch.book.introneuralnet.neural.feedforward.

FeedforwardLayer;

importar com.heatonresearch.book.introneuralnet.neural.feedforward.

FeedforwardNetwork;

importar com.heatonresearch.book.introneuralnet.neural.feedforward.

train.anneal.NeuralSimulatedAnnealing;

/\*\*

* Capítulo 7: Entrenamiento con recocido simulado

\*

* XOR: Aprende el patrón XOR con un neural de avance
* red que utiliza recocido simulado.

\*

* @author Jeff Heaton
* @version 2,1

\*/

clase pública AnnealXOR {

doble XOR\_INPUT estática pública[][] =

{ { 0.0, 0.0 }, { 1.0, 0.0 },

{ 0.0, 1.0 }, { 1.0, 1.0 } };

doble XOR\_IDEAL estática pública[][] =

{ { 0.0 }, { 1.0 }, { 1.0 }, { 0.0 } };

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

nuevo FeedforwardNetwork(); network.addLayer(nuevo FeedforwardLayer(2)); network.addLayer(nuevo FeedforwardLayer(3)); network.addLayer(nuevo FeedforwardLayer(1)); network.reset();

entrenar la red neuronal

último tren NeuralSimulatedAnnealing = nuevo NeuralSimulatedAnnealing(

red, XOR\_INPUT, XOR\_IDEAL, 10, 2, 100);

int epoch = 1;

hacer {

train.iteration();

System.out

.println("Epoch #" + epoch

+ " Error:" + train.getError()); epoch++;

} mientras que ((epoch < 100) && (train.getError() > 0.001));

red = train.getNetwork();

probar la red neuronal System.out.println("Resultados de la red neuronal:"); para (int i = 0; i < XOR\_IDEAL.length; i++) {

final doble real[] = network.computeOutputs(XOR\_INPUT[i]); System.out.println(XOR\_INPUT[i][0] + ","

+ XOR\_INPUT[i][1]

+ ", real=" + real[0]

+ ",ideal=" + XOR\_IDEAL[i][0]);

}

}

}

La clase **NeuralSimulatedAnnealing** implementa la interfaz **Train** y, por lo tanto, se puede utilizar al igual que la retropropagación y los algoritmos genéticos discutidos en capítulos anteriores. El **NeuralSimulatedAnnealing** se crea una instancia de la siguiente manera:

último tren NeuralSimulatedAnnealing = nuevo NeuralSimulatedAnnealing(

red, XOR\_INPUT, XOR\_IDEAL, 10, 2, 100);

La clase **NeuralSimulatedAnnealing** implementa un método **de aleatorización** especial. Este método excita el estado de la red neuronal de una manera que es muy similar a cómo funciona la implementación de salesman itinerante. La firma para el método **randomize** se muestra aquí:

aleatorización del vacío público()

En primer lugar, **MatrixCODEC** se utiliza para serializar la red neuronal en una matriz de

**Variables** dobles.

matriz doble final[] = MatrixCODEC.networkToArray(this.network);

A continuación, recorremos la matriz.

para (int i = 0; i < array.length; i++) {

Cada elemento de matriz se excita aleatoriamente en función de la temperatura.

doble adición = 0.5 - (Math.random()); añadir /= getStartTemperature();

añadir \*= this.temperature; array[i] = array[i] + add;

}

Por último, **MatrixCODEC** se utiliza para convertir la matriz de nuevo en una red neuronal.

MatrixCODEC.arrayToNetwork(array, this.network);

Cuando esta nueva red neuronal se devuelve a la función **de iteración,** se evaluará. Si es una mejora, se conservará.

# Resumen del capítulo

En este capítulo, aprendió sobre el algoritmo de recocido simulado. El recocido simu-lated algorithm se basa en el proceso real de recocido. El punto clave detrás del proceso de recocido es que un metal que se permite enfriar lentamente formará estructuras cristalinas más consistentes y, por lo tanto, más fuertes. La razón es que las temperaturas más altas resultan en niveles de energía más altos para los átomos que componen el metal. En los niveles de energía más altos, los átomos tienen mayor libertad de movimiento. A medida que el metal se enfría, esta libertad de movimiento se reduce. Esto permite que los átomos se asienten en patrones de cristal consistentes.

El proceso de recocido simulado es muy similar al proceso de recocido real. Se presentan una serie de valores de entrada en el algoritmo de recocido simulado. El algoritmo de recocido de lado simu quiere optimizar estos valores de entrada para que se pueda minimizar una ecuación arbitraria. Algunos ejemplos de ecuaciones a minimizar son el error

para una red neuronal o la distancia que viaja un vendedor viajero. Los valores de entrada, que impulsan el algoritmo de recocido simulado, pueden ser la matriz de peso de una red neuronal o la ruta actual entre ciudades por la queviaja un vendedor de traveling.

Para presentar un ejemplo relativamente simple de cómo utilizar el recocido simulado, este capítulo volvió una vez más al problema del vendedor viajero. El problema de los vendedores ambulantes también se utilizó en el capítulo 6 junto con algoritmos genéticos. Reutilizar el problema de los vendedores ambulantes nos permite comparar fácilmente el rendimiento de las algasgenéticas con el recocido simulado.

# vocabulario

Los ciclos de recocido de recocido excitan

Temperatura de recocido simulada

# Preguntas para revisión

1. Describir el proceso de recocido metalúrgico.
2. ¿Cómo se aleatorizan los elementos de matriz al encontrar soluciones para el problema de los vendedores de viajes?
3. ¿Cómo se aleatorizan los elementos de matriz al entrenar una red neuronal?
4. ¿Cuál es la diferencia entre un ciclo y una iteración en el algoritmo an-nealing simulado presentado en este capítulo?
5. ¿Cómo nos aseguramos de que la temperatura cambie naturalmente entre ciclos?