**CHAPTER** **8:** **PRUNING** **NEURAL** **NETWORKS**

* ¿Qué es la poda?
* Poda incremental
* Poda selectiva
* Ejemplos de poda

En los capítulos 6 y 7, vimos que se puede utilizar el recocido simulado y algoritmos genéticos para entrenar redes neuronales. Estas dos técnicas emplean varios algoritmos para ajustar mejor los pesos de una red neuronal al problema al que se está aplicando. Sin embargo, estas técnicas no hacen nada para ajustar la estructura de la red neuronal.

En este capítulo, examinaremos dos algoritmos que se pueden usarse para modificar realmente la estructura de una red neuronal. Esta modificación estructural generalmente no mejorará la tasa de error de la red neuronal, pero puede hacer que la red neuronal sea más eficiente. La modificación se realiza analizando cuánto contribuye cada neurona a la salida de la red neuronal. Si la conexión de una neurona en particular con otra neurona no afecta significativamente la salida de la red neuronal, la conexión será podada. A través de este proceso, se eliminan las conexiones y neuronas que tienen sólo un impacto marginal en la salida.

Este proceso se denomina poda. En este capítulo, examinaremos cómo se cumple la poda. Comenzaremos examinando el proceso de poda con más detalle y discutiremos algunos de los métodos populares. Por último, este capítulo concluirá proporcionando dos ejemplos que demuestran la poda.

# Comprensión de la poda

La poda es un proceso utilizado para hacer que las redes neuronales sean más eficientes. A diferencia de los algoritmos genéticos o recocido simulado, la poda no aumenta la eficacia de una red neuronal. El principal objetivo de la poda es disminuir la cantidad de procesamiento necesario para utilizar la red neuronal.

La poda puede ser especialmente eficaz cuando se realiza en una gran red neuronal que tarda demasiado en ejecutarse. La poda funciona mediante el análisis de las conexiones de la red neuronal. El algoritmo de poda busca conexiones individuales y neuronas que se pueden eliminar de la red neuronal para que funcione de manera más eficiente. Mediante la poda de conexiones innecesarias, la red neuronal se puede hacer para ejecutar más rápido. Esto permite que la red neuronal realice más trabajo en una cantidad determinada de tiempo. En las próximas dos secciones examinaremos cómo podar tanto las conexiones como las neuronas.

## Conexiones de poda

La poda de conexiones es fundamental para la mayoría de los algoritmos de poda. Las conexiones individuales entre las neuronas se analizan para determinar qué conexiones tienen menos impacto en la eficacia de la red neuronal. Uno de los métodos que examinaremos eliminará todas las conexiones que tengan un peso por debajo de un determinado valor de umbral. El segundo método evalúa la eficacia de la red neuronal, ya que ciertos pesos se consideran para la eliminación. Las conexiones no son lo único que se puede podar. Mediante el análisis de las conexiones que fueron podadas, también podemos podar las neuronas individuales.

## Podar neuronas

La poda se centra en las conexiones entre las neuronas individuales de la red neuronal. Sin embargo, las neuronas individuales también se pueden podar. Uno de los algoritmos de poda que examinaremos más adelante en este capítulo está diseñado para podar neuronas, así como conexiones.

Para podar las neuronas individuales, se deben examinar las conexiones entre cada neurona y las otras neuronas. Si una neurona en particular está rodeada enteramente de conexiones débiles, no hay razón para mantener esa neurona. Si aplicamos los criterios discutidos en la sección anterior, podemos terminar con neuronas que no tienen conexiones. Esto se debe a que todas las conexiones de la neurona fueron podadas. Tal neurona puede ser podada por sí misma.

## Mejorar o degradar el rendimiento

Es posible que la poda de una red neuronal pueda mejorar su rendimiento. Cualquier modificación en la matriz de peso de una red neuronal siempre tendrá algún impacto en la precisión de los reconocimientos realizados por la red neuronal. Una conexión que tiene poco o ningún impacto en la red neuronal puede estar degradando la precisión con la que la red neuronal reconoce patrones. La eliminación de una conexión tan débil puede mejorar la salida general de la red neuronal.

Desafortunadamente, también es posible disminuir la eficacia del trabajo de la red neuronal a través de la poda. Por lo tanto, siempre es importante analizar la eficacia de la red neuronal antes y después de la poda. Dado que la eficiencia es el principal beneficio de la poda, debe tener cuidado de evaluar si una mejora en el tiempo de procesamiento vale la pena una disminución en la eficacia de la red neuronal. El programa en el ejemplo que examinaremos más adelante en este capítulo evaluará la eficacia general de la red neuronal antes y después de la poda. Esto nos dará una idea de qué efecto tuvo el proceso de poda en la eficacia de la red neuronal.

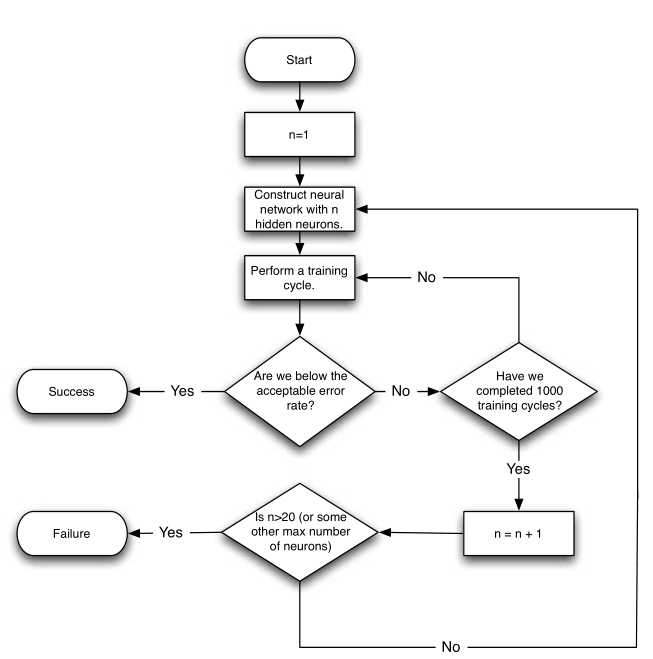
# Algoritmos de poda

Ahora revisaremos exactamente cómo se lleva a cabo la poda. En esta sección vamos a examinar dos métodos diferentes para la poda. Estos dos métodos funcionan de maneras algo opuestas. El primer método, la poda incremental, funciona aumentando gradualmente el número de neuronas ocultas hasta que se ha obtenido una tasa de error aceptable. El segundo método, la poda selectiva, funciona tomando una red neuronal existente y disminuyendo el número de neuronas ocultas siempre y cuando la tasa de error siga siendo aceptable.

## Poda incremental

La poda incremental es un enfoque de ensayo y error para encontrar un numero apropiado de las neuronas ocultas. Este método se resume en la Figura 8.1.

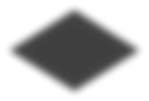
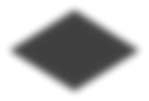
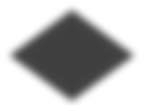
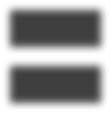
### Figura 8.1: Diagrama de flujo del algoritmo de poda incremental.





Éxito

vosotros



Empezar

No

Sí

¿Estamos por debajo de la

tasa de error ¿Aceptable?

No

¿Tenemos

1000 ciclos de ¿Formación?

Sí

S

Es n>20 (o algunos

otro número máximo de neuronas)

No

Realizar un ciclo de entrenamiento.

n = N + 1

Construcción rojo neuronal contra N neuronas ocultas.

n=1



Fracaso

El algoritmo de poda incremental comienza con una red neuronal no entrenada. A continuación, intenta entrenar la red neuronal muchas veces. Cada vez, utiliza un conjunto diferente de neuronas ocultas.

El algoritmo de entrenamiento incremental debe proporcionarse con una tasa de error aceptable. Está buscando la red neuronal con el menor número de neuronas ocultas que hará que la tasa de error caiga por debajo del nivel deseado. Una vez que se encuentra una red neuronal que se puede entrenar para caer por debajo de esta tasa, el algoritmo está completo.

Como se vio en el capítulo 5, "Feedforward Backpropagation Neural Networks", a menudo es necesario entrenar durante muchos ciclos antes de encontrar una solución. El algoritmo de poda incremental requiere que toda la sesión de entrenamiento se complete muchas veces. Cada vez que se agrega una nueva neurona a la capa oculta, la red neuronal debe ser reentrenada. Como resultado, el algoritmo de poda incremental puede tardar mucho tiempo en ejecutarse.

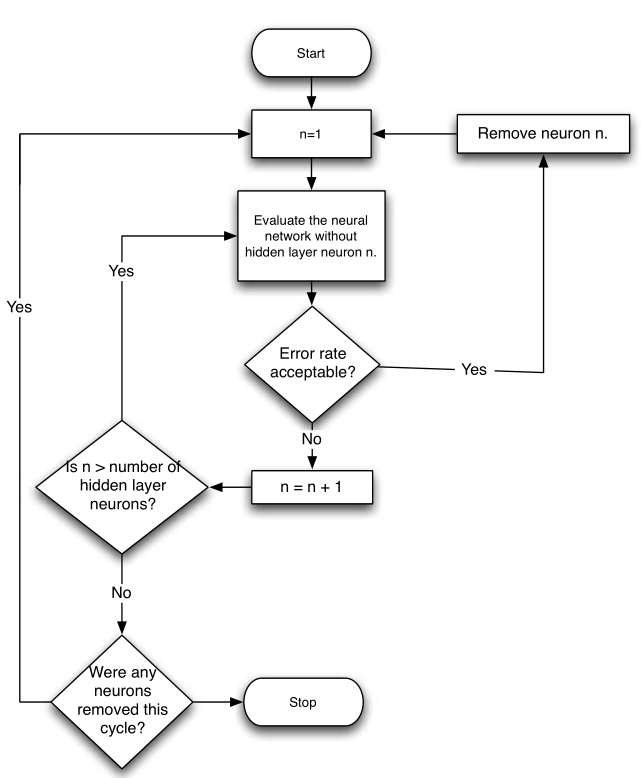
La red neuronal se entrenará para diferentes números de neuronas ocultas, comenzando inicialmente con una sola neurona. Debido a que la tasa de error no cae lo suficientemente rápido, la red neuronal de neurona oculta se abandonará rápidamente. Cualquier número de metanfetaminas se puede utilizar para determinar cuándo abandonar una red neuronal. El método que se utilizará en este capítulo es comprobar la tasa de error actual después de intervalos de 1.000 ciclos. Si el error no disminuye en un solo punto porcentual, se abandonará la orden de búsqueda l. Esto nos permite abandonar rápidamente los tamaños de capa ocultos que son demasiado pequeños para la tarea prevista.

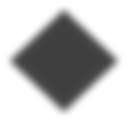
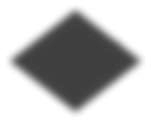
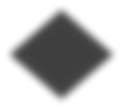
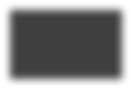
Una ventaja del algoritmo de poda incremental es que normalmente creara redes neuronales con menos neuronas ocultas que los otros métodos. La mayor desventaja es la cantidad de tiempo de procesador que se tarda en ejecutar este algoritmo. Ahora que se ha introducido en el algoritmo de poda incremental, examinaremos el algoritmo de poda selectiva.

## Poda selectiva

El algoritmo de poda selectiva difiere del algoritmo de poda incremental de varias maneras importantes. Una de las diferencias más notables es el estado inicial de la red neuronal. No se requirió entrenamiento antes de comenzar el algoritmo de poda incremental - ing. Este no es el caso el algoritmo de poda selectiva. El algoritmo de poda selectiva funciona examinando las matrices de peso de una red neuronal previamente entrenada. El algoritmo de entrenamiento selectivo intentará eliminar las neuronas sin interrumpir la salida de la red neuronal. El algoritmo utilizado para la poda selectiva se muestra en la Figura 8.2.

### Figura 8.2: Diagrama de flujo del algoritmo de poda selectiva.





Empezar

Sí

Sí

Tasa de error

¿Aceptable?

Sí

No

Es N > número de

neuronas de capa ¿Oculta?

No

¿Eran cualquier

neuronas extirpadas este ciclo?

parar

n = n + 1

Evaluar la red neuronal sin neurona de capa oculta n.

Eliminar la neurona n.

n=1

Como puede ver, el algoritmo de poda selectiva es algo así como un enfoque de prueba y error. El algoritmo de poda selectiva intenta eliminar las neuronas de la red neuronal hasta que no se puedan eliminar más neuronas sin degradar el rendimiento de la red neuronal.

Para comenzar este proceso, el algoritmo de poda selectiva recorre cada una de las neuronas ocultas. Para cada neurona oculta encontrada, el nivel de error de la red neuronal se evalúa con y sin la neurona especificada. Si la tasa de error salta más allá de un nivel predefinido, la neurona se conservará y se evaluará la siguiente neurona. Si la tasa de error no salta mucho, se eliminará la neurona.

Una vez que el programa ha evaluado todas las neuronas, el programa repite el proceso. Este ciclo continúa hasta que el programa ha hecho que uno pase a través de las neuronas ocultas sin eliminar una sola neurona. Una vez completado este proceso, se logra un nuevo trabajo neural que se realiza aceptablemente cerca del original, pero tiene menos neuronas ocultas.

La principal ventaja del algoritmo de poda selectiva es que se necesita muy poco tiempo de procesamiento para completar. El programa en el ejemplo presentado más adelante en este capitulo requiere menos de un segundo para podar una red neuronal con 10 neuronas de capa oculta. Esto es considerablemente menos tiempo que el algoritmo de poda incremental descrito en la sección anterior.

# Implementación de la poda

Ahora que se ha explicado cómo funcionan los algoritmos de poda, se le mostrará cómo implementarlos en Java. En esta sección, examinaremos una clase general diseñada para podar las redes neuronales de retropropagación de feedforward. El nombre de esta clase es simplemente "Prune". Esta clase acepta una clase "Network" y realiza poda incremental o selectiva. Comenzaremos examinando la clase de Prune class. A continuación, examinaremos los algoritmos incrementales y selectivos dentro de la clase.

## La clase Prune

La clase **Prune** contiene todos los métodos y propiedades necesarios para podar una red neuronal de retropropagación feedforward. Hay varias propiedades que son utilizadas internamente por muchos de los métodos que componen la **prune** clase. Estas propiedades se summarized en la Tabla 8.1.

### Tabla 8.1: Variables utilizadas para el proceso de ciruela pasa

|  |  |
| --- | --- |
| **variable** | **propósito** |
| backprop | El objeto backpropagation que se va a podar. |
| cycles | El número de ciclos. |
| done | Indicador para indicar si el proceso de poda incremental se realiza o no. |
| error | El error actual. |
| ideal | Los resultados ideales del conjunto de entrenamiento. |
| hiddenNeuronCount | El número de neuronas ocultas. |
| markErrorRate | Se utiliza para determinar si el entrenamiento sigue siendo efectivo. Mantiene el nivel de error determinado en los 1000 ciclos anteriores. Si no se produce ninguna caída significativa en el error durante 1000 ciclos, finaliza el entrenamiento. |
| maxError | El error máximo aceptable. |
| momentum | El impulso deseado. |
| rate | La tasa de aprendizaje deseada (para la backpropagation). |
| sinceMark | Se utiliza con markErrorRate. Este es el número de ciclos desde que se marcó el error por última vez. |
| train | El conjunto de entrenamiento. |

Ahora se le mostrará cómo se implementan los algoritmos de poda selectiva e incremental. Comenzaremos con la poda incremental.

## Poda incremental

Como recordará de principios de este capítulo, el proceso de poda incremental implica aumentar el número de neuronas en la capa oculta hasta que el trabajo de la red neuronal sea capaz de ser entrenado lo suficientemente bien. Esto debería conducirnos automáticamente a un número good de neuronas ocultas. El constructor utilizado para implementar la poda incremental es muy simple. Recopila los parámetros requeridos y los almacena en los vínculos adecuados de la clase.

Los parámetros requeridos por el constructor de poda incremental son los parámetros habituales necesarios para realizar el entrenamiento de backpropagation de un trabajo de red neuronal feedforward. La tasa de aprendizaje y el impulso son necesarios. Estas constantes de backpropagation se discuten con mayor detalle en el capítulo 5. Además, también se requiere un conjunto de entrenamiento, junto con las salidas ideales para el conjunto de entrenamiento.

El parámetro final requerido por el constructor de poda incremental es el error mínimo aceptable. Esta es la tasa de error que es suficiente para que una red neuronal sea entrenada. A medida que el algoritmo de poda incremental progresa, intentará entrenar redes neuronales con varios números de neuronas ocultas. El número final de neuronas ocultas será determinado por la red neuronal con el menor número de neuronas ocultas que es capaz de ser entrenado para alcanzar este nivel de error.

La clase **Prune** contiene dos constructores; uno es para poda incremental y el otro es para poda selectiva. El constructor de la clase **Prune** que se utilizará con la poda incremental se muestra aquí:

Prunepúblico ( doble velocidad final, doble impulso final, tren doble final[][], doble ideal final[][], doble maxError final )

Esto proporciona a la clase el entrenamiento y los conjuntos ideales que se utilizarán con backpropagation, así como un error máximo aceptable.

## Inicio del proceso de poda incremental

Para comenzar el proceso de poda incremental, debe llamar al método **startIncremental**. Esto configura la clase **Prune** para una nueva ejecución de poda. La firma para el método **startIncremental** se muestra aquí.

público void startIncremental()

En primer lugar, algunas de las propiedades se establecen en sus valores iniciales. Una de esas propiedades es el número de neuronas que existirán en la capa oculta. Inicialmente, comenzamos con una sola neurona en la capa oculta.

this.hiddenNeuronCount = 1;

this.cycles = 0;

this.done = false;

Se crea un nuevo objeto **FeedforwardNetwork** para contener la red neuronal que se preparará a medida que avance el entrenamiento.

this.currentNetwork = new FeedforwardNetwork();

A continuación, las capas se agregan a la red.

this.currentNetwork

. addLayer(nuevo FeedforwardLayer(this.train[0].length)); this.currentNetwork.addLayer(nuevo FeedforwardLayer)

this.hiddenNeuronCount)); this.currentNetwork.addLayer(nuevo

FeedforwardLayer(this.ideal[0].length)); this.currentNetwork.reset();

Se crea un objeto backpropagation para entrenar la red.

this.backprop = nueva backpropagation(this.currentNetwork, this.

tren

this.ideal, this.rate, this.momentum);

Ahora, el algoritmo de poda incremental esta lista para comenzar. En la siguiente sección, verá cómo se construye el bucle principal del algoritmo incremental.

## Bucle principal del algoritmo incremental

El algoritmo de poda incremental es intensivo en procesador y puede tardar algún tiempo en ejecutarse. Esto se debe a que el algoritmo incremental está literalmente probando diferentes números de neuronas ocultas, de forma de prueba y error, hasta que se identifica una red que tiene el menor número de neuronas mientras produce un nivel de error aceptable.

El algoritmo de poda incremental está diseñado para que un subproceso en segundo plano pueda llamar rápidamente al método **pruneIncremental** hasta que el método **getDone** indicates que el algoritmo está hecho. Por supuesto, el método **pruneIncremental** también se puede llamar desde el subproceso principal. La firma para el método **pruneIncremental** se muestra aquí:

vacío público pruneIncremental()

Si el trabajo ya se ha hecho, entonces salga.

if (this.done) { retorno;

}

El método **pruneIncremental** comienza comprobando primero si ya está

hecho. Si el algoritmo ya está hecho, el método **pruneIncremental** simplemente devuelve.

El siguiente paso es intentar un solo ciclo de entrenamiento para la red neuronal. El **incremento**de llamadas pro-gramo, que recorre todos los conjuntos de entrenamiento y calcula el error en función de las salidas ideales.

this.backprop.iteration();

Una vez presentado el conjunto de entrenamiento, se calcula el error del cuadrado medio raíz (RMS). El error rms se discutió en el capítulo 4, "Cómo una máquina aprende." Cal- culating el error RMS permite que el algoritmo de poda determine si el error ha alcanzado el nivel deseado.

this.error = this.backprop.getError();

this.cycles++;

incremento();

Cada vez que se ejecuta un ciclo de entrenamiento, la red neuronal debe comprobar si el número de neuronas ocultas debe aumentar o si el entrenamiento debe continuar con la red neuronal actual.

## Aumentar el número de neuronas

Para disuadirne si el número de neuronas ocultas debe aumentarse o no, se utiliza el método auxiliar denominado **incremento.** El método **de incremento** realiza un seguimiento del progreso del entrenamiento para la red neuronal. Si la mejora de entrenamiento para cada ciclo cae por debajo de un valor constante, se considera inútil una mayor capacitación. En este caso, incrementa el número de neuronas ocultas y continuar el entrenamiento.

El método de **incremento** comienza estableciendo una marca que indica si debe aumentar o no el número de neuronas ocultas. El valor predeterminado es **false,**loque significa que no incremente el número de neuronas. La firma para el método de incremento se muestra aquí:

incremento vacío protegido()

Comience estableciendo la variable **doit** en **false**. Esta variable se establece en **true** si se encuentra una mejor configuración de red neuronal.

booleano hacerlo = false;

El algoritmo que esta clase utiliza para determinar si el entrenamiento adicional es inútil, funciona examinando la cantidad de mejora cada 10.000 ciclos. Si la tasa de error no cambia en más de un uno por ciento dentro de 10.000 ciclos, se considera inútil un entrenamiento adicional y se incrementa el número de neuronas en la capa oculta.

Las siguientes líneas de código realizan esta evaluación. En primer lugar, cuando **markErrorRate** es cero, significa que apenas estamos empezando y aún no hemos muestreado la tasa de error. En este caso, inicializamos las variables **markErrorRate** y **sinceMark.**

if (this.markErrorRate == 0) {

this.markErrorRate = this.error;

this.sinceMark = 0;

} else {

Si **markErrorRate** no es cero, entonces estamos rastreando errores. Debemos crear el contador de ciclo **sinceMark** y determinar si se han completado más de 10.000 ciclos desde la última vez que muestreamos la tasa de error.

this.sinceMark++;

if (this.sinceMark > 10000) {

Si han pasado más de 10.000 ciclos, comprobamos si la mejora debe interpolar la **markErrorRate** y la tasa de error actual es inferior al uno por ciento. Si este es el caso, entonces fijamos la bandera para indicar que el número de en neuronas ocultas debe incrementarse.

if ((this.markErrorRate - this.error) < 0.01) { doit = true;

}

A continuación, la tasa de error se muestrea de nuevo.

this.markErrorRate = this.error; this.sinceMark = 0;

}

}

Si la tasa de error está por debajo del error aceptable, hemos encontrado el número de neuronas que recomendaremos para la red neuronal. Ahora podemos establecer la bandera **hecha** en

## verdadero.

if (this.error < this.minError) { this.done = true;

}

Si se establece el indicador, incrementamos el número de neuronas de la siguiente manera:

if (doit) { este.cycles = 0;

this.hiddenNeuronCount++;

this.currentNetwork = nuevo FeedforwardNetwork(); this.currentNetwork.addLayer(nuevo FeedforwardLayer)

this.train[0].longitud)); this.currentNetwork.addLayer(nuevo FeedforwardLayer)

this.hiddenNeuronConde)); this.currentNetwork.addLayer(nuevo FeedforwardLayer)

this.ideal[0].length)); this.currentNetwork.reset();

this.backprop = nueva Backpropagation(this.currentNetwork, this.train, this.ideal, this.rate, this.momentum);

}

Como puede ver, se construye una nueva red neuronal después del número de

neuronas se incrementa. Además, el recuento de ciclos se restablece a cero, porque comenzaremos a entrenar una nueva red neuronal.

Ahora debe estar familiarizado con cómo el algoritmo de poda incremental es imple- mented. Más adelante en este capítulo, construiremos un programa de ejemplo que hace uso de este algoritmo. Por ahora, cubriremos la implementación de la segunda poda de unlgorithm, el algoritmo de poda selectiva.

## Poda selectiva

Ahora que ha visto cómo se implementó el algoritmo de poda incremental, examinaremos la implementación del algoritmo de poda selectiva. De alguna manera, elalgoritmo de poda selectiv e funciona al revés del algoritmode poda incremental. Donde el algoritmo de poda incremental comienza pequeño y crece, el algoritmo de poda se-lective comienza con una red neuronal grande y preentrenada, y selecciona las neuronas para su eliminación.

Parautilizar la poda selectiva, puede hacer uso de una versión simplificada del constructor **Prune.** La poda selectiva no necesita saber nada sobre la tasa de aprendizaje o el impulso del proceso de backpropagation, porque no implica el algoritmo de gation backpropa. La versión simplificada del constructor se muestra a continuación.

prune público( Red de red,tren doble[][], doble ideal[][], último doble maxError)

Como puede ver, sólo está obligado a pasar un conjunto de entrenamiento de red neuronal y los resultados ideales al algoritmo de poda selectiva. El constructor es muy simple y simplemente almacena los valores que se pasó. Examinaremos ahora la aplicación de los métodos de poda selectiva. Comenzaremos examinando el bucle principal del algoritmo.

## Bucle principal del algoritmo de poda selectiva

El bucle principal del algoritmo de poda selectiva es mucho menos intensivo en procesador que el algoritmo de poda incremental. El ithm algor de poda incrementalque examinamos en la sección anterior se diseñó para ejecutarse como un subproceso en segundo plano debido al gran número de ciclos que podrían ser necesarios para encontrar una solución. Este no es el caso con el algoritmo de poda selectiva.

El algoritmo de poda selectiva está diseñado para evaluar el rendimiento de la red neuronal cuando se elimina cada neurona oculta. Si el rendimiento de la red neuronal no se degrada sustancialmente con la eliminación de una neurona, esa neurona será removed permanentemente. Este proceso continuará hasta que no se puedan eliminar neuronas adicionales sin degradar sustancialmente el rendimiento de la red neuronal. Por lo tanto, el algoritmo de poda selectiva está diseñado para realizar todo el algoritmo con una llamada de método. No hay ninguna razón para un subproceso en segundo plano, ya que este método debe ser capaz de realizar su tarea casi instantáneamente.

El método al que se debe llamar para podar selectivamente una red neuronal es el método **pruneSelective.** Ahora examinaremos cómo funciona este método. La firma para este método se muestra aquí:

público int pruneSelective()

A continuación se encuentra el cuerpo del método **pruneSelective.**

final int i = this.getHiddenCount(); mientras que (findNeuron()) {

;

}

retorno (i - this.getHiddenCount());

Como se puede ver en el código anterior, el número actual de neuronas ocultas se almacena en la variable **i** para su uso futuro. A continuación, entramos en un bucle y iterar hasta que no se puede eliminar ninguna addi- neuronas tionales. Por último, el método **pruneSelective** devuelve el número de neuronas que se eliminaron de la red neuronal. La nueva red neuronal optimizada se almacena en la propiedad **currentNetwork** de la clase **Prune** y se puede tener acceso mediante el método **getCurrentNetwork.**

El trabajo real realizado por el algoritmo de poda selectiva se realiza mediante el método **findNeuron.** Es el método **findNeuron** que realmente identifica y vuelve a mover las neuronas que no tienen un efecto adverso en la tasa de error. El **método findNeuron** eliminará una neurona, si es posible, y devolverá **true.** Si no se puede volver a mover ninguna neurona, el método **findNeuron** devuelve un valor de **false**. Como se puede ver en el código anterior en el **pruneSelective** método, el **findNeuron** método se llama siempre y cuando el valor devuelto es **true**. La firma para el método **findNeuron** se muestra aquí:

hallazgo booleano protegidoNeuron()

Ahora examinaremos el contenido del método **findNeuron.** Este método se produce calculando la tasa de error actual.

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

A continuación, se vuelve a calcular la tasa de error y evaluamos el efecto que esto tiene en la cualificación de la red neuronal. Mientras la calidad siga estando por debajo de **maxError,**seguimos podando.

prueba final de FeedforwardNetwork = this.clipHiddenNeuron(i); final doble e = determineError(prueba);

if (e < this.maxError) {

Si tenemos una red aceptable cuando se elimina la neurona, salimos del método y se devuelve true. El valor de **true** indica que la neurona fue eliminada, y debemos intentar eliminar neuronas adicionales.

this.currentNetwork = prueba; volver verdadero;

}

}

Si no se eliminaron neuronas, se devuelve false. Esto indica que otros pro-

cesing sin valor adicional.

devolver falso;

Ahora se le mostrará el proceso por el cual las neuronas se eliminan de la red neuronal.

## Eliminación de neuronas

La eliminación de una neurona de un **objeto FeedforwardLayer** es un proceso relativamente sencillo. La tarea de eliminar una neurona oculta individual es en- tapasulated dentro del **prune** método de la **FeedforwardLayer** clase en el **clipHiddenNeuron** método. Lo primero que hace el método **clipHiddenNeuron** es crear una nueva red neuronal con una neurona menos.

ciruela del vacío público ( neurona int final) {

En primer lugar, elimina la fila de la matriz de esta capa que corresponde a la neurona objetivo.

if (this.matrix != null) { setMatrix(MatrixMath.deleteRow(this.matrix, neuron));

}

A continuación, elimina la columna de la matriz de la capa anterior que corresponde a la

neurona dirigida.

final FeedforwardLayer anterior = this.getPrevious(); if (anterior != null) {

if (previous.getMatrix() != null) { previous.setMatrix(

MatrixMath.deleteCol(previous.getMatrix(), neurona));

}

Esto completa el proceso para eliminar la neurona objetivo y concluye cómo

la clase **Prune** está construida. Cubrimos la poda incremental y selectiva. Ahora veremos cómo hacer uso de esta clase en un programa Java.

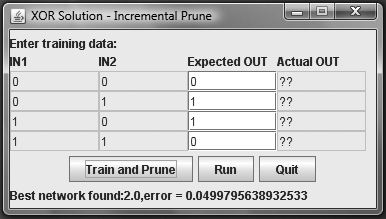
## Uso de la clase Prune

En esta sección, examinaremos dos programas que se proporcionan como ejemplos para demostrar el uso de la clase **Prune.** Estos ejemplos demuestran los dos métodos de poda que hemos discutido en este capítulo. Comenzaremos con los incrémenes- ejemplo de poda de recuento.

## El ejemplo de poda incremental

Este ejemplo se basa en el problema XOR que se presentó en el capítulo 5. Por lo tanto, el código completo para este programa no se mostrará aquí. Más bien, se muestra el código adicional que se agregó al ejemplo XOR en chapter 5 para admitir la poda. Puede ver la salida del ejemplo de poda incremental en la figura 8.3.

### Figura 8.3: El ejemplo de poda incremental.



Cuando se pulsa el botón "Prune/Train", se crea un hilo de fondo. Este subproceso en segundo plano ejecuta el método de **ejecución** del ejemplo de poda. Ahora examinaremos el contenido del método **run.**

La firma para el método de **ejecución** se muestra aquí:

corrida pública del vacío( )

El método **run** comienza leyendo los datos introducidos por el usuario. Tanto los conjuntos de trenes como los resultados ideales se leen desde la ventana.

xorDatadoble final[][] = getGrid(); doble xorideal final [][] = getIdeal(); int actualización = 0;

A continuación, se crea un objeto **Prune** denominado **prune**. Este objeto utilizará una tasa de aprendizaje de 0,7 y un impulso de 0,5. Se llama al método **startIncremental** para prepararse para la poda incremental.

prune final = nueva Ciruela(0,7, 0,5, xorData, xorIdeal, 0,05); prune.startIncremental();

Mientras no se haga el algoritmo de poda, el bucle continuará.

mientras (! prune.getDone()) {

Se realizará una iteración de la formación incremental.

prune.pruneIncremental(); update++;

La pantalla se actualiza cada diez iteraciones.

if (actualización == 10) {

this.status.setText("Ciclos:" + prune.getCycles()

+ ",Neuronas ocultas:" + prune.getHiddenNeuronCount()

+ ", Error actual=" + prune.getError()); actualización = 0;

}

}

Por último, el sistema informa de la mejor red encontrada.

this.status.setText("Mejor red encontrada:"

+ prune.getHiddenNeuronCount()

+ ",error = " + prune.getError());

A continuación, esta red se copia en la red actual.

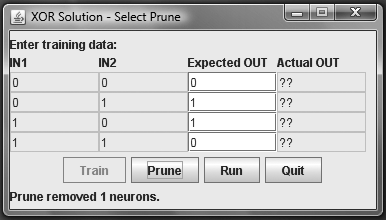
this.network = prune.getCurrentNetwork(); this.btnRun.setEnabled(true);

En esta sección, ha visto cómo se utilizó la clase **Prune** para implementar una poda incremental de una red neuronal. En la siguiente sección, verá cómo utilizar la clase **Prune** para una poda selectiva de una red neuronal.

## Ejemplode poda selectiva

Este ejemplo de poda selectiva también se basa en el problema XOR que se mostró en el capítulo 5; por lo tanto, el código completo no se muestra aquí. En su lugar, solo se presenta el código adicional que se agregó al ejemplo XOR del capítulo 5 para admitir la poda selectiva. Puede ver la salida del ejemplo de poda incremental en la figura 8.4.

### Figura 8.4: El ejemplo de poda selectiva.



Dado que el ejemplo de poda selectiva requiere que una red neuronal ya esté presente para podar, debe comenzar haciendo clic en el botón "Entrenar". Esto entrenará la red neuronal usando el método de backpropagation. La red neuronal que se muestra en este ejemplo contiene inicialmente diez neuronas en la capa oculta. Al hacer clic en la "Ciruela" pero- tonelada comenzará una poda selectiva e intentará eliminar algunas de las neuronas de la capa oculta.

El código necesario para implementar un algoritmo de poda selectiva es muy simple. Sea- porque este algoritmo se ejecuta muy rápidamente, no haynecesidad de un subproceso en segundoplano. Esto simplifica en gran medida el uso del algoritmo de poda selectiva. Cuando se hace clic en el botón "Podar", se ejecuta el método de **ciruela pasa.** La firma para la **ciruela** meth-od se muestra aquí:

ciruela pasas del vacío público()

El método **prune** comienza obteniendo los datos de la cuadrícula.

xorDatadoble final[][] = getGrid(); doble xorideal final [][] = getIdeal();

prune final = nueva Ciruela(this.network, xorData, xorIde- al,0.05);

recuento final de int = prune.pruneSelective(); this.network = prune.getCurrentNetwork();

this.status.setText("Prune eliminado " + count + " neuronas."); this.btnTrain.setEnabled(false);

A continuación, se crea una instancia de un objeto **Prune.** Se llama al método **pruneSelective** que devuelve un recuento de las neuronas que se eliminaron durante la ciruela pasa. Este valor se muestra al usuario y la nueva red se copia en la red actual. El usuario ahora puede ejecutar la red para ver, de primera mano,el rendimiento de la red neuronal. Usted encontrará that el algoritmo de poda selectiva es generalmente capaz de eliminar dos o tres neuronas. Mientras que esto deja más neuronas que el algoritmo incremental, sig- nificantly menos tiempo de procesador se requiere, y no hay necesidad de volver a entrenar la red neuronal.

# Resumen de Chapter

Como aprendiste en este capítulo, es posible podar las redes neuronales. La poda de una red neuronal elimina las conexiones y las neuronas con el fin de hacer la red neuronal más eficiente. El objetivo de la poda no es hacer que la red neuronal sea más eficaz para reconocer patrones, sino hacerlo más eficiente. Hay varios al-gorithms diferentes para podar una red neuronal. En este capítulo, examinamos dos de estos algoritmos.

El primer algoritmo que examinamos se llamó algorithm. Este algoritmo entrena nuevas redes neuronales a medida que el número de neuronas ocultas está en- arrugado. El algoritmo de poda incremental eventualmente se asienta en la red neuronal que tiene el menor número de neuronas en la capa oculta, pero todavía mantiene un nivel er- ror aceptable. Mientras que el algoritmo incremental a menudo encontrará el número ideal de neuronas ocultas, en general, se necesita una cantidad considerable de tiempo para ejecutar.

El segundo algoritmo que examinamos se llamó algoritmo de poda selectiva. El algoritmo de poda selectiva comienza con una red neuronal que ya ha sido entrenada. A continuación, el algoritmo elimina las neuronas ocultas siempre y cuando el error permanezca bajo un nivel d especificado. Aunque el algoritmo de poda selectiva a menudo no encontrará el número óptimo de neuronas que ejecutará considerablemente más rápido que el algoritmo incremental.

El objetivo principal de la poda de la red neuronal es mejorar la eficiencia. Al podar una red neural, usted es capaz de crear una red neuronal que se ejecutará más rápido y requerirá menos ciclos de procesador. Si la red neuronal ya está funcionando lo suficientemente rápido, debe evaluar si la poda está justificada. Incluso cuando la eficiencia es de gran importancia, debe sopesar las compensaciones entre la eficiencia y una reducción en la eficacia de su red neuronal.

Hasta ahora, este libro se ha centrado principalmente en el uso de redes neuronales para reconocer patrones. Las redes neuronales can también se enseñan a predecir tendencias futuras. Al proporcionar una red neuronal con una serie de valores basados en el tiempo, puede predecir valores posteriores. Los próximos dos capítulos se centrarán en las redes neuronales predictivas. Este tipo de red neuronal a veces se denomina red neuronal temporal.

# vocabulario

Importancia de la conexión Poda incremental

Poda selectiva

# Preguntas para revisión

1. Describir los pasos del método de poda incremental.
2. Describir los pasos del método de poda selectiva.
3. Usted tiene una red neuronal que ya ha sido entrenada y produce buenos resultados. Sin embargo, te preguntas si podrías ser capaz de eliminar varias neuronas ocultas. ¿Qué método de poda usaría para lograr esto? ¿Por qué?
4. Le gustaría eliminar una neurona de la capa oculta. ¿Qué se debe hacer, en todo caso, con la matriz de peso entre la capa de entrada y la capa oculta? ¿Qué se debe hacer, en todo caso, con la matriz de peso entre la capa oculta y la capa de salida?
5. Qué método de poda debe utilizar uno de los algoritmos de aprendizaje como parte de su proceso.