REDES NEURONALES EVOLUCIONARIAS
3. DIVIDE Y VENCERÁS

José del Carmen Rodríguez Santamaría
https://www.ejristos.com
Bogotá, Colombia

November 15, 2024

Abstract

Investigamos las posibilidades de evolución en la construcción de redes neuronales (RNs) que reconozcan imágenes. En este artículo exploramos el efecto de dividir en pequeñas porciones la cuadrícula donde aparecen los dibujos.

Contents

1 INTRODUCCIÓN 3

2 LA ARQUITECTURA VIEJA 5

3 LA NUEVA ARQUITECTURA 9

4 RNs TOLERANTES 15

5 RNs EVOLUCIONARIAS TOLERANTES 16

6 CONCLUSION 22

7 RESPUESTAS A LOS EJERCICIOS 23

8 BIBLIOGRAFÍA 33

1 INTRODUCCIÓN

Nuestro objetivo es utilizar la Evolución para diseñar redes neuronales (RN) que clasifiquen imágenes. Trabajamos con modelos de juguetes para obtener información y permitir realizar cálculos en computadoras comunes. Nuestras imágenes consisten en patrones que se dibujan en una cuadrícula de 3×3 cuyos píxeles son blancos y negros y que, por lo tanto, pueden describirse mediante cadenas binarias de 9 bits de largo. Estos se denominan mapas de bits.

Cada mapa de bits se clasifica según el criterio del vecino más cercano según la distancia BBB como una deformación de uno de dos arquetipos, una imagen de 1 y otra de 0. La distancia BBB es la distancia Bit a Bit que cuenta el número de lugares donde dos cadenas binarias difieren.

Ya hemos diseñado una RN con dos perceptrones que puede clasificar correctamente 260 mapas de bits de 512. Nuestros perceptrones son, de hecho, puertas booleanas generalizadas que son tolerantes al ruido y a la variabilidad. Probamos en nuestra *Introducción* que la Evolución puede ajustar conjuntos separados de diversos parámetros de la RN para sintonizarla. (Rodríguez, [2] 2024). También demostramos que la perfección, clasificar correctamente los 512 mapas de bits, es alcanzable pero con un coste exagerado de casi 250 perceptrones. También demostramos que la Evolución tiene un cierto poder para simplificar la RN (Rodríguez, [3] 2023).

En esta ocasión, nos inspiramos en la antigua estrategia en Ciencias de la Computación de *Divide y Conquistarás*, que divide un problema complejo en subproblemas, cuyas soluciones se combinarán para generar una solución al problema original (geeksforgeeks, [1] 2024). Por lo tanto, dividimos la cuadrícula 3×3 en 3 filas. Esto provoca que cada mapa de bits se divida en tres submapas de bits que deben

ser reconocidos por circuitos separados, los cuales se unirán para producir una clasificación final de mapas de bits completos.

Los arquetipos quedan así, el de la izquierda representa el número 1, el de la derecha, el 0.

```
010     111
010     101
010     111
```

Los mapas de bits correspondientes son 010010010 y 111101111.

Ahora tenemos la misma grilla, pero cada arquetipo, así como cada imagen, tienen tres mapas de bits asociados. Los del arquetipo 1 son 010 010 010 y los del arquetipo 0 son 111 101 111. Este es nuestro gran invento. La misma estrategia se ha utilizado para pasar de las palabras a las sílabas, que permitieron al antiguo egipcio inventar un sistema de escritura, y de las palabras a los fonemas, que permitieron a los fenicios inventar el alfabeto. El resultado neto es una gran ganancia de expresabilidad y ajuste. Por lo tanto, esperamos que nuestro esquema adoptado permita decidir los parámetros de los perceptrones de reconocimiento localmente y, por lo tanto, se mejore la estabilidad frente al ruido y la variabilidad.

Nuestro primer problema es: idear con nuestra propia mente una RN que tome como entrada los 3 submapas de bits de cada imagen y reproduzca todos los resultados de la RN con la arquitectura antigua con dos perceptrones. El segundo problema es hacer que Evolución ajuste los parámetros de la RN para que se reproduzcan los mismos resultados.

Nuestro plan es reutilizar lo que sabemos para llegar más lejos. Así que en este momento conviene una revisión.

2 LA ARQUITECTURA VIEJA

Unimos dos perceptrones en una red neuronal (NN), ver figura 1, y demostramos que puede clasificar correctamente 260 de los 512 posibles mapas de bits donde la distancia BBB definió la asignación correcta. Cada mapa de bits fue asignado al arquetipo más cercano según esta distancia. Algunos detalles siguen:

Un perceptrón debe decidir si un estímulo es lo suficientemente fuerte como para dispararlo. La fuerza del estímulo viene dada por una suma total S, y debe ser más fuerte que un cierto umbral T. Entonces, debemos decidir si $S \geq T$.

Procedamos paso a paso. Nuestra tarea es diseñar un perceptrón que se active cuando reciba un estímulo específico. El estímulo es un mapa de bits, en este caso 010010010, el mapa de bits del arquetipo 1.

El primer paso es asociar una cláusula booleana al mapa de bits del arquetipo. Booleano significa que se refiere a VERDADERO o FALSO. Se puede negar una variable, operación que invierte la salida. Una cláusula puede tener varias variables unidas por operadores adecuados, AND, OR.

La cláusula booleana asociada a un mapa de bits se construye de la siguiente manera: un 1 en el mapa de bits se reemplaza por una variable booleana y un 0 por una negación de una variable booleana. Concretamente:

El mapa de bits de 1, 010010010, induce la cláusula booleana

$$\sim n_0 \wedge n_1 \wedge \sim n_2 \wedge \sim n_3 \wedge n_4 \wedge \sim n_5 \wedge \sim n_6 \wedge n_7 \wedge \sim n_8$$

El $\wedge$ es el operador booleano AND que dice que un mapa de bits es en realidad una lista de condiciones que deben cumplirse para detectar un patrón. Los subíndices realizan un seguimiento de la posición de cada bit.

Una cláusula es una función: para cada cadena de longitud 9 de valores booleanos, genera VERDADERO o FALSO. Es la **función de especificación**. Si el resultado es VERDADERO, decimos que el mapa de bits dado ha sido reconocido como el arquetipo. Si el resultado es FALSO, nos metemos en problemas: los perceptrones que implementarán nuestras ideas se disparan o no se disparan. Cuando hay disparo, cuando la RN responde VERDADERO, el arquetipo ha sido reconocido. Pero si no se dispara, no debemos inventar una respuesta: simplemente no sabemos qué sucede. La salida es DESCONOCIDA.

El siguiente paso es asociar una expresión algebraica en variables reales a la cláusula booleana. Entonces, asociamos una expresión dicotómica afín a cada cláusula booleana. Dicotómica significa que cada variable solo puede tomar valores 1 o 0. Afín significa que se podría agregar una constante.

Tenemos dos reglas de sustitución que asocian un polinomio real de primer grado a cada carácter del mapa de bits o variable de la cláusula asociada:

- $1 \to e_i$,
- $0 \to -e_j + 1$.

donde cada variable está marcada por un subíndice que declara su posición en el mapa de bits comenzando desde el índice 0. El coeficiente de la variable es el peso, la constante que la acompaña es el sesgo, la suma de todos los sesgos es el sesgo total.

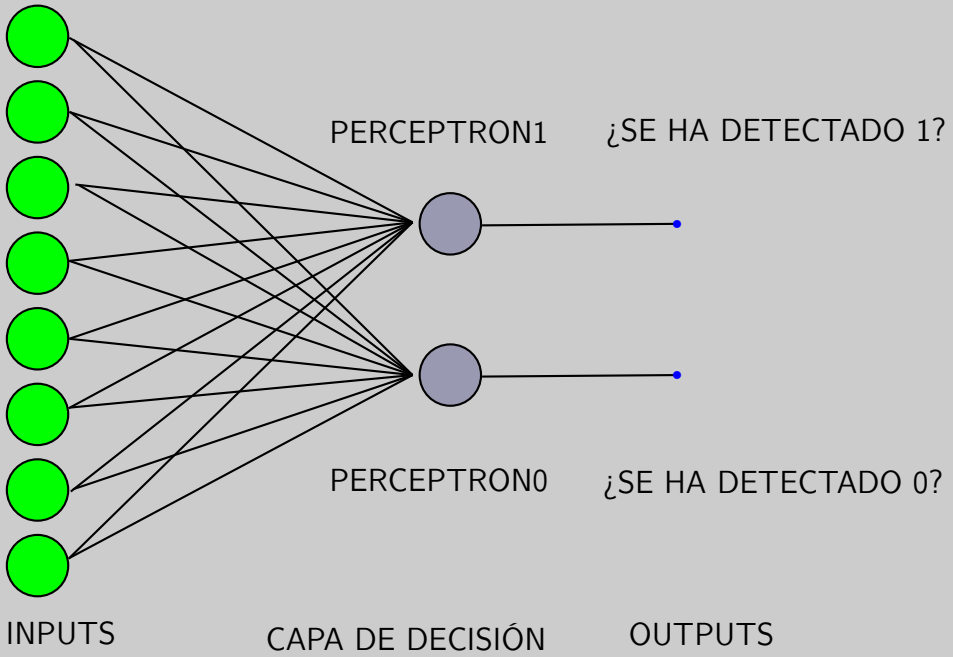


Figure 1: Esta RN fue diseñada por la inteligencia humana. Su tarea es reconocer imágenes distorsionadas de 1 o 0. ¿Cómo funciona? Tiene 9 entradas, una para cada píxel en la cuadrícula binaria 3×3. Las entradas representan un mapa de bits posiblemente distorsionado de las imágenes de 1 o 0. Todas las entradas están conectadas a cada celda o perceptrón de la capa de decisión, que contiene solo 2 celdas o perceptrones. La celda superior debe generar 1 cuando se presenta una imagen de 1 y cero en caso contrario. La celda inferior debe generar 1 cuando se presenta 0, y 0 en caso contrario. Las imágenes pueden estar un poco distorsionadas por el ruido y/o la variabilidad. La salida general es una cadena, con la salida del Perceptrón 1 seguida de la del Perceptrón 0. Advertencia: esta salida debe analizarse para ver si contiene una contradicción. Esto sucede cuando ambos perceptrones emiten 1, eso significa que el primer perceptrón ha percibido un 1 y el otro un 0. Si la salida es 10, dice que el primer perceptrón percibió un 1, mientras que el otro no encontró nada. La RN percibió un 1. Si la salida es 01, la entrada era desconocida para el primer perceptrón, mientras que el segundo percibió un 0. La RN percibió un 0. Aquí no hay contradicción. Lo mismo ocurre con la salida 00, que se interpreta como DESCONOCIDA. Por tanto, la lógica de esta RN tiene tres valores: se ha detectado 1, se ha detectado 0, DESCONOCIDO.

Para un perceptrón AND, el umbral T es el número de bits porque para responder SÍ o UNO, todas las entradas deben estar activadas.

Así, en la arquitectura antigua, la suma S asociada al perceptrón que se dispara tras el registro del arquetipo 1, con mapa de bits 010010010, es:

$$S = -e_0 + 1 + e_1 - e_2 + 1 - e_3 + 1 + e_4 - e_5 + 1 - e_6 + 1 + e_7 - e_8 + 1 = -e_0 + e_1 - e_2 - e_3 + e_4 - e_5 - e_6 + e_7 - e_8 + 6$$

donde contamos bits desde 0 hasta 8. Como hay 6 ceros, el sesgo es 6. Esta suma S debe ser mayor, más fuerte, que el umbral, T, para activar el perceptrón: $S \geq T$. El valor de T para un perceptrón AND es la longitud de cada mapa de bits, 9. Por último, la desigualdad $S \geq T$ dice:

$$S = -e_0 + 1 + e_1 - e_2 + 1 - e_3 + 1 + e_4 - e_5 + 1 - e_6 + 1 + e_7 - e_8 + 1 = -e_0 + e_1 - e_2 - e_3 + e_4 - e_5 - e_6 + e_7 - e_8 + 6 \geq 9$$

Si reemplazamos las variables por los bits correspondientes obtenemos $-0+1-0-0+1-0-0+1-0+6 = 9 \geq 9$. Como esto es cierto, el arquetipo1 ha sido reconocido por el perceptrón.

Observemos que la RN no reconoce ningún otro mapa de bits. Porque otro mapa de bits será diferente al del arquetipo1. Si 0 se cambia por 1, el lado izquierdo de la desigualdad recibe -1, y si el mapa de bits tiene 0 en lugar de 1, el lado izquierdo de la desigualdad pierde 1. Así, cualquier discrepancia con el arquetipo1 provoca una pérdida de fuerza del estímulo: el perceptrón no se disparará.

1 *Ejercicio.* *Escriba la desigualdad $S \geq T$ correspondiente al arquetipo0, 111101111. ¿Cuántos mapas de bits reconoce el perceptrón correspondiente?* Respuesta

Las dos desigualdades antes mencionadas representan la arquitectura de la figura 1 con dos perceptrones, uno que reconoce el arquetipo1, que representa 1 y el otro, arquetipo0, que representa 0.

3 LA NUEVA ARQUITECTURA

Dividimos la cuadrícula en 3 filas e ideamos un sistema de análisis para cada una de ellas, cuyas salidas debe unirse de alguna manera para producir una respuesta única.

Nuestra tarea es pasar de un sistema, cuya entrada es una descripción de un mapa de bits como una sola cadena, como 010010010, a otro en el que los mapas de bits se describen mediante 3 tríadas, como 010 010 010. Cada descripción genera su propia arquitectura. La primera es lo vieja, la segunda es la nueva que hay que diseñar.

Para completar los detalles de la nueva arquitectura, dejemos que la descripción del perceptrón nos guíe. La desigualdad $S \geq T$ de la arquitectura antigua se adapta mejor a nosotros en su forma primitiva:

$$S = \\ -e_0 + 1 + e_1 - e_2 + 1 - e_3 + 1 + e_4 - e_5 + 1 - e_6 + 1 + e_7 - e_8 + 1 \\ \geq 9$$

Recordemos: cuando el bit es 1, el peso es 1 y el sesgo es 0, si el bit es 0, el peso es -1 y el sesgo 1. Aquí nos ocupamos del arquetipo 010010010. Esta desigualdad se puede reescribir como

$$S = (-e_0 + 1 + e_1 - e_2 + 1) + (-e_3 + 1 + e_4 - e_5 + 1) + \\ (-e_6 + 1 + e_7 - e_8 + 1) \geq 3 + 3 + 3$$

donde hemos aplicado la ley asociativa de los números reales a entradas agrupadas por tríadas. Inmediatamente vemos aquí 3 perceptrones que reconocen cada uno un tercio del mapa de bits original:

$S_0 = (-e_0 + 1 + e_1 - e_2 + 1) \geq 3$, perceptron P_{10}
$S_1 = (-e_3 + 1 + e_4 - e_5 + 1) \geq 3$, perceptron P_{11}
$S_2 = (-e_6 + 1 + e_7 - e_8 + 1) \geq 3$, perceptron P_{12}

donde el 3 derecho representa el hecho de que, en la nueva arquitectura, tratamos con mapas de bits de longitud igual a 3. Cada desigualdad representa un perceptrón que reconoce el submapa de bits 010. Lo que necesitamos ahora es unirlos para reconocer el mapa de bits del arquetipo 010010010. Esa tarea la realiza el perceptrón P_1 que especificamos ahora:

¿Es el perceptrón P_1 de tipo AND, OR, NOR o NAND? Nuestra respuesta es: el perceptrón de decisión P_1 debe ser de tipo AND porque para reconocer el mapa de bits de un arquetipo, se deben reconocer los tres submapas de bits al mismo tiempo.

Hemos definido así la arquitectura presentada en la figura 2 y desde otra perspectiva en la figura 3. Las entradas están conectadas a celdas sensoriales P_{ij} que conforman la llamada capa oculta, cuya salida alimenta la capa de decisión que contiene las celdas que deben reconocer los patrones, P_1 para el arquetipo de UNO y P_0 para CERO. Las celdas que alimentan a P_1 son P_{10}, P_{11} y P_{12}. Los que alimentan a P_0 son P_{00}, P_{01} y P_{02}.

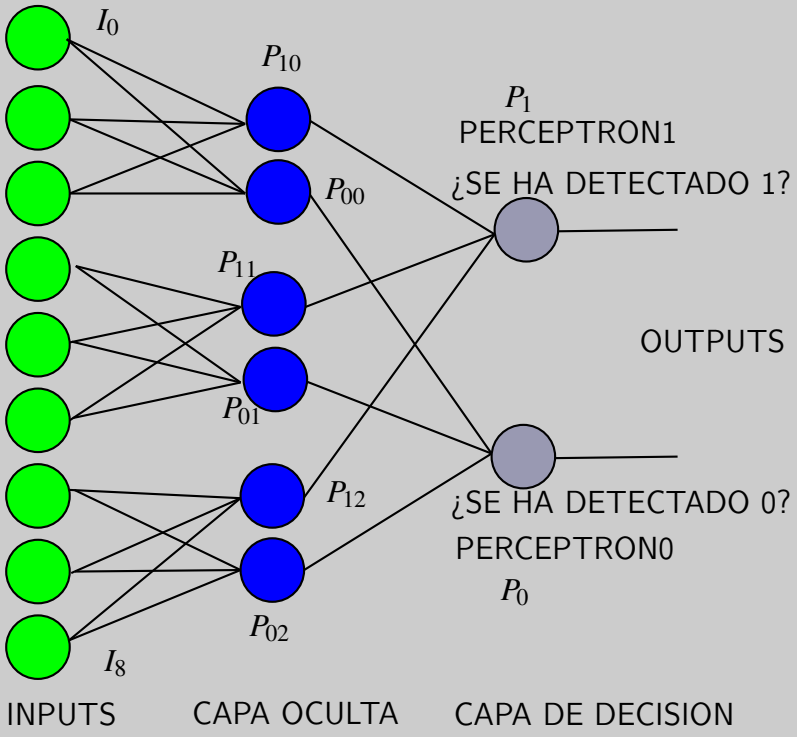


Figure 2: Esta RN es nuestra primera implementación de nuestra directiva *divide y vencerás*. Su tarea es reconocer imágenes distorsionadas de 1 o 0. ¿Cómo funciona? Tiene 9 entradas, una para cada píxel en la cuadrícula binaria 3×3. Las entradas, I_0 ... I_8, representan un mapa de bits posiblemente distorsionado de las imágenes de 1 o 0. La salida de la RN es producida por los perceptrones $P1$ y $P0$. El primero se dispara cuando percibe una imagen que representa 1, mientras que el segundo se dispara cuando percibe un 0. Las entradas se agrupan por tríadas siguiendo el orden de lectura de los píxeles de la cuadrícula que representan. Cada tríada de entradas es analizada por dos perceptrones de la capa oculta, el primero, P_{1j}, trabaja para P_1, y el segundo, P_{0j} para P_0. Nuestra tarea es definir los parámetros de cada perceptrón para reproducir exactamente el comportamiento de la arquitectura antigua que tenía solo un par de perceptrones y no agrupaba las entradas de ninguna manera.

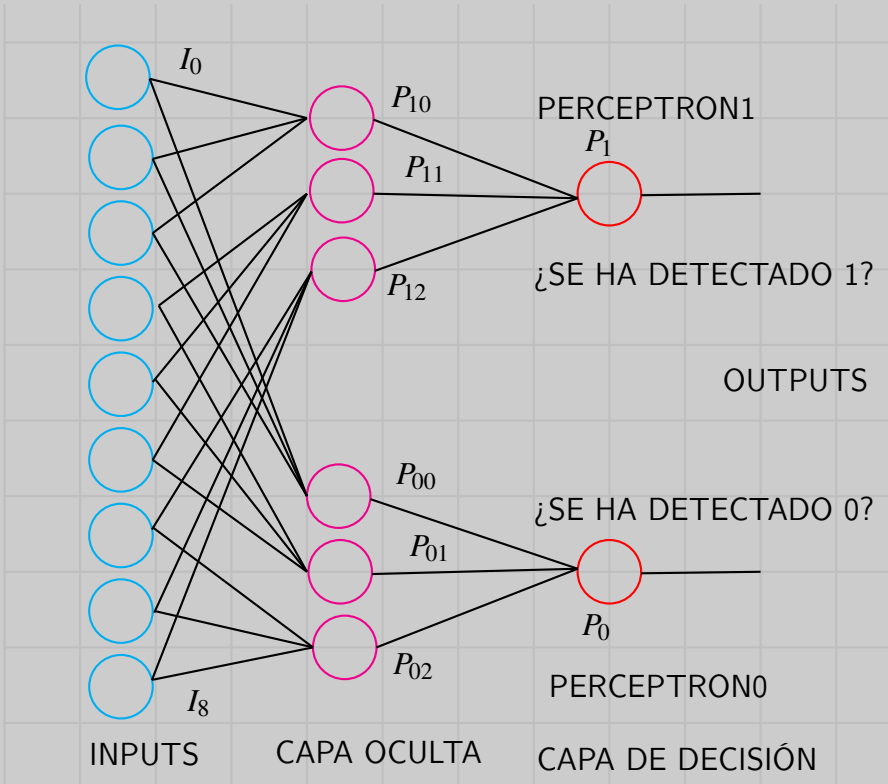


Figure 3: Se puede observar fácilmente en la figura 2 que la entrada se ha dividido en tres partes. Pero la arquitectura de la RN no es visible de inmediato. Pero en esta nueva figura eso se ha resuelto. Se ha hecho más claro, más transparente, más modificable, más evolucionable. Cuando las cosas se vuelven complejas, pequeños giros en las ideas y las presentaciones correspondientes pueden marcar una gran diferencia en la capacidad de evolución, la capacidad de complexificarse fácilmente, incluir más funciones o expandir su dominio.

Hemos implementado estas ideas en el programa `divideP1.html`. El código es el siguiente:

```
<!DOCTYPE html>
<html>

<body>
    <h1><h1>Program divideP1.html
        <br> Perceptron P1 and its slaves</h1>
    <p>The bitmap of archetype 1, 010010010, must be recognized
        <br>by a NN that divides the bitmap in three parts,
        <br>010 010 010, which are processed by three perceptrons
        <br>of the hidden layer,
        <br>whose output feed the decision perceptron P1.
    </p>

    <div id="myDoc"></div>

    <script>
        "use strict";

        const archetype = "010010010";

        //Output of generic perceptron
        function outputPerceptron(inputs, weights, bias, threshold, name) {
            let sum = 0;
            let output = 0;
            for (let i = 0; i < inputs.length; i++) {
                sum += inputs[i] * weights[i];
            }
            document.getElementById("myDoc").innerHTML +=
                "<br>" + name + " :  sum = " + sum + ", bias = " + bias;
            sum += bias;
            if (sum >= threshold) { output = 1 }
            return output;
        }

        function main() {
            //All three AND-perceptrons of the hidden layer
            //have the same weights, bias and threshold
            //because all are target to the same sub-bitmap.
            let weights = [-1, 1, -1];
            let bias = 2;
            let threshold = 3;
            //The bitmap of the archetype
            //is divided in three parts
            let bm0 = archetype.substring(0, 3), //010
                bm1 = archetype.substring(3, 6), //010
                bm2 = archetype.substring(6);  //010
            document.getElementById("myDoc").innerHTML +=
                "Archetype = 010010010.  <br> Subparts :  "
                + bm0 + "  " + bm1 + "  " + bm2;
            //HIDDEN LAYER
            //Perceptron P10
            let inputs = bm0;
            let name = "P10";
            let output0 = outputPerceptron(bm0, weights, bias, threshold, name);
            //Perceptron P11
            inputs = bm1;
            name = "P11";
            let output1 = outputPerceptron(inputs, weights, bias, threshold, name);
            //Perceptron P12
            inputs = bm2;
            name = "P12";
            let output2 = outputPerceptron(inputs, weights, bias, threshold, name);

            //DECISION LAYER
            //processes  the outputs of the hidden layer
            //with an AND-perceptron assuming that
            //each perceptron of the hidden layer
            //recognizes its sub-bitmap.
            //So, their outputs must be all 1.
            inputs = [output0, output1, output2];
            let weightsP1 = [1, 1, 1];
            bias = 0;
            threshold = 3;
            name = "P1";
            let outputP1 = outputPerceptron(inputs, weightsP1, bias, threshold, name);
            let finalOutput = "NO";
            if (outputP1) { finalOutput = "YES" };
            document.getElementById("myDoc").innerHTML +=
```

```
                "<br><br>Does this NN recognize the archetype?   <br>" + finalOutput;
        }
        main();

    </script>
</body>
</html>
```

2 *Ejercicio.* *Ejecute el programa* `divideP1.html` *y verifique que reconozca el arquetipo 1.*

3 *Ejercicio.* *¿Cuántos mapas de bits reconoce la RN del programa* `divideP1.html`*? Redacte una prueba verbal más un programa que la pruebe.* Respuesta

4 *Ejercicio.* *Diseñar una RN que reconozca el arquetipo 0 que se ha dividido en tres partes. Debe especificar la arquitectura y los parámetros de todos sus perceptrones.* Respuesta

Tenemos dos circuitos de perceptrones, el primero reconoce el arquetipo 1, el segundo el arquetipo 0. Nuestra siguiente tarea es unirlos para conseguir una RN funcional. Entonces, debemos desarrollar el código de una RN que acepte solo los arquetipos de 1 y 0 y que los clasifique correctamente. Si la entrada es el arquetipo de 1, la RN debe generar 10, si es el de 0, la salida debe ser 01. Si la entrada es cualquier otra, la salida debe ser 00. La salida 11 nunca debe aparecer porque eso equivale a una contradicción: significa que un mapa de bits dado representa ambos arquetipos, el de 1 y el de 0. Nuestra solución a esta tarea aparece en el programa `divideConquerNN.html`

5 *Ejercicio.* *Ejecute el programa* `divideConquerNN.html` *y verifique que reconozca los dos arquetipos, el 1 y el 0, y nada más. Al arquetipo1 debe responder* 10*, al arquetipo0* 01*, y a cualquier otra entrada* 00*. La salida* 11 *nunca debe aparecer.* Respuesta

6 *Desafío.* *El programa* `divideConquerNN.html` *funciona correctamente pero su código es redundante porque los dos métodos que reconocen los mapas de bits de los dos arquetipos son casi iguales. Simplifique el código.*

Tenemos ahora una RN que reconoce dos triadas de submapas de bits, la primera es del arquetipo 1 y la segunda del de 0. Todos los perceptrones de esta RN son de tipo AND. Nos enfrentamos ahora a otra tarea: implementar la tolerancia al ruido y a la variabilidad.

4 RNs TOLERANTES

Hemos diseñado una RN con dos circuitos de perceptrones que reconoce sólo dos arquetipos. Cada circuito reconoce un arquetipo específico. Eso sucede porque hemos estado tratando con una ecuación: $S = T$ que dice que el estímulo debe ser exactamente igual al umbral. Si se producen algunos desajustes, $S \leq T$, el estímulo pierde fuerza y el perceptrón no puede disparar. Por lo tanto, para hacer que nuestra RN sea tolerante a fallas, agregamos un término, $addBias$, el sesgo adicional, para compensar el déficit creado por los desajustes. De esa manera, obtenemos $S + addBias > T$. El problema es la posible aparición de contradicciones que ocurren cuando se disparan ambos circuitos, en cuyo caso ambos proclaman que el mapa de bits de entrada es una deformación reconocible de su arquetipo asignado.

Entonces, el problema es encontrar los valores de los sesgos adicionales de los 8 perceptrones para maximizar el número de disparos, de coincidencias, sin generar contradicciones.

7 *Desafío.* *Desarrolle un programa para explorar los posibles valores de los sesgos adicionales de cada uno de los 8 perceptrones de nuestra RN. Realice un pronóstico del máximo número de mapas de bits reconocidos, pero sin caer en contradicciones ni entre circuitos ni con la distancia BBB.*

Planteemos a la Evolución la tarea de optimizar los valores de los sesgos adicionales para maximizar las coincidencias pero sin contradicciones tanto entre circuitos como contra la distancia BBB.

5 RNs EVOLUCIONARIAS TOLERANTES

El Proceso Evolutivo debe especificar los valores de los sesgos adicionales de los 8 perceptrones de nuestra RN, que hasta ahora han sido 0. Esta familia de parámetros es responsable de la tolerancia al ruido y a la variabilidad, es decir, permite reconocer imágenes distorsionadas como deformaciones de uno de los dos arquetipos dados.

Dado que la única tarea que se le plantea a la Evolución es especificar los 8 valores de los sesgos adicionales, nuestros cromosomas deben codificar sus valores y nada más.

¿Cómo codificaremos los sesgos adicionales? En un programa anterior utilizamos 4 caracteres para cada sesgo, digamos la cadena 1234 codificaba el número real 12.34, el valor de un sesgo. Ahora, razonemos así: la distancia BBB entre dos mapas de bits toma valores enteros, 0, 1,..., 9. Por lo tanto, los sesgos adicionales deben tomar exactamente los mismos valores para rodear más y más mapas de bits comenzando desde cada arquetipo. Por lo tanto, nuestros cromosomas deben tener un carácter para cada sesgo y tenemos 8 perceptrones, por lo tanto, necesitamos 8 caracteres por cromosoma. Enumeremos los perceptrones: P10, P11, P12, P1, P00, P01, P02, P0. Un cromosoma quedaría así: 20628543 y la lista establece la correspondencia con los sesgos de los perceptrones.

Para empezar, modifiquemos el código del programa `divideConquerNN.html` para que sea compatible con la Evolución y con nuestro propósito: la RN debe ampliar su tolerancia y por tanto el programa debe ser capaz de leer valores para los sesgos adicionales de todos sus 8 perceptrones. En el caso de Evolución, el código debe incorporar correctamente los sesgos adicionales que deben leerse en los cromosomas. Tengamos en cuenta que estos son cadenas, mientras que los sesgos

son números. Debemos registrar el efecto de cambiar los valores de sesgos adicionales sobre el número de mapas de bits aceptados por la nueva RN pero sin contradicciones.

Nuestro primer paso para resolver la tarea está en el programa `divideConquerNN2.html`. Allí se definió una cadena de 8 caracteres que contiene los 8 valores de los sesgos adicionales. Luego, esta cadena se transformó en un vector de números enteros y sus valores se aplicaron a los 8 perceptrones en el orden P10, P11, P12, P1, P00, P01, P02, P0. Los primeros 4 eran P1 y sus esclavos, los otros 4 eran P0 y sus esclavos.

8 *Ejercicio.* *Ejecute el programa `divideConquerNN2.html` con los 8 valores de sesgos adicionales iguales a 0 y verifique que esta RN acepta solo dos mapas de bits, los dos arquetipos.* Respuesta

A continuación, creamos el programa `divideConquerNN3.html` para generar valores aleatorios de los sesgos adicionales y probarlos en mapas de bits aleatorios.

9 *Ejercicio.* *Ejecute el programa `divideConquerNN3.html` y saque su conclusión sobre la aparición de contradicciones.* Respuesta

Por último, tenemos en el programa `divideConquerEvolNN.html` nuestra implementación de la Optimización Evolutiva de los sesgos adicionales.

10 *Ejercicio.* *Satisfaga su curiosidad jugando con el programa `divideConquerEvolNN.html` para ver cómo la Evolución encuentra los valores máximos para los sesgos adicionales de los 8 perceptrones de la RN dada pero sin caer en contradicciones.* Respuesta

11 *Misterio.* *Como vimos en nuestra Introducción, debemos clasificar 512 mapas de bits como deformaciones de 1 o 0 según la distancia BBB pero sin contradicciones. Nuestra RN inicial con*

2 perceptrones que aceptaron como entrada mapas de bits completos fue capaz de reconocer 260. Aquí hemos experimentado con la idea de dividir una cuadrícula grande en subcuadrículas. Nuestra esperanza era que, de la misma manera que nuestro alfabeto fenicio permite una mayor expresabilidad y un reconocimiento más preciso de los mensajes, los subpatrones asociados a subcuadrículas también podrían mejorar los resultados en la tarea de reconocimiento visual de patrones con mapas de bits. Pero tuvimos un fracaso estrepitoso, lo cual es un misterio para nosotros. Porque esperábamos que nuestros resultados fueran mejores que los anteriores, pero en cambio el desempeño se dividió por dos.

Fracasamos y, hasta esta línea, no entendemos por qué.

Después de pensar un rato, se nos ocurrieron algunas ideas para explicar nuestro misterioso fracaso.

La primera causa posible es que la codificación de la Evolución pueda estar defectuosa. Pero creemos que la evolución funciona aquí porque si la mutación y la recombinación son moderadas o bajas, vemos que el top de la aptitud aumenta. Como se esperaba. Pero si estas son elevadas, la Evolución deambula sin descanso. Como se esperaba. Por lo tanto, no consideramos que el código pueda estar equivocado.

¿Qué otra cosa?

Esperábamos mejorar el rendimiento de la antigua RN con dos perceptrones. Pero nuestros resultados dicen que el rendimiento se dividió por dos. Al parecer no es el código. Por tanto, consideremos la posibilidad de que **hayamos estado trabajando con conceptos erróneos no expresados explícitamente**. Entonces, revisemos nuestro trabajo y nuestras ideas.

En pocas palabras, todo lo que hemos estado haciendo es comparar dos sistemas de desigualdades, el primero correspondiente a perceptrones de la arquitectura antigua, mientras que el segundo se refiere a un sistema de desigualdades en el que un mapa de bits se

divide en tres partes con maquinaria de reconocimiento separada.

El perceptrón de la arquitectura antigua que reconoce las deformaciones del arquetipo 1 se describe mediante la desigualdad:

$$S = (-e_0 + 1 + e_1 - e_2 + 1) + (-e_3 + 1 + e_4 - e_5 + 1) + (-e_6 + 1 + e_7 - e_8 + 1) + addBias1 \geq 9 = 3 + 3 + 3 = T$$

La parte correspondiente de la nueva arquitectura se representa mediante el siguiente sistema de desigualdades

$$\begin{aligned} S_0 &= (-e_0 + 1 + e_1 - e_2 + 1) + addBias10 \geq 3, \\ S_1 &= (-e_3 + 1 + e_4 - e_5 + 1) + addBias11 \geq 3, \\ S_2 &= (-e_6 + 1 + e_7 - e_8 + 1) + addBias12 \geq 3, \end{aligned}$$

donde el 3 a la derecha representa el hecho de que en la nueva arquitectura, tratamos en los tres casos con mapas de bits de longitud igual a 3. Hemos incorporado explícitamente los sesgos adicionales responsables de la tolerancia a fallas, que son valores enteros positivos menores que 9.

A primera vista, hemos dividido una gran desigualdad en tres. Como esto: de $7 = 1+2+4 > 2+2+2 = 6$, extraemos tres desigualdades, $1 > 2$, $2 \geq 2$, $4 > 3$. Obtuvimos dos desigualdades válidas y una falsa. Entonces, esta lectura, que alimentó nuestra imaginación, es engañosa. Pero **el Método Científico, de realizar predicciones con experimentos y pensamiento crítico, nos ha liberado.**

Por otro lado, podemos sumar tres desigualdades de números reales positivos para obtener una desigualdad sólida: de $3 > 2$, $2 > 1$, $3 > 1$ podemos obtener $6 = 3+2+1 > 2+1+1 = 4$. Ahora bien, dado que la desigualdad correspondiente al perceptrón de la arquitectura antigua se puede deducir de los correspondientes sistemas de

desigualdades de la nueva, sumándolos, obtenemos una predicción:

Si la nueva arquitectura dispara, la vieja también. Es decir, la vieja no puede ser peor que la nueva. Esta es una abierta contradicción con lo que nos impulsó a hacer este trabajo, porque esperábamos que la nueva arquitectura fuera mejor que la vieja. Nuestros conceptos fundamentales se basaban en una analogía con los alfabetos fonéticos que finalmente resultó errónea. Somos testigos de que un lema inspirador de la Informática -*divide y vencerás*- no funciona aquí. Por tanto, no es incondicionalmente correcto.

Sigamos con otras preguntas.

12 ***Ejercicio.*** *Nos parece que la arquitectura antigua no es peor que la nueva y que lo más posible es que la antigua sea la mejor. Pruébelo. Para lograr esto, debe verificar las dos predicciones siguientes. Primera: si la nueva arquitectura reconoce un mapa de bits, la antigua también lo reconoce. Segundo: hay mapas de bits que son reconocidos por la arquitectura antigua pero no por la nueva.* *Respuesta*

13 ***Ejercicio.*** *Sumar desigualdades de números reales positivos y con la misma dirección equivale a sumar congruentemente pesos de diversas balanzas. Por tanto, se pueden sumar desigualdades de números reales positivos para producir una desigualdad válida. De esto concluimos que la arquitectura antigua extiende o generaliza a la nueva, en sus diversas versiones degeneradas, lo que significa que cuando la nueva arquitectura se dispara, la antigua también debe dispararse. Pero el recuento directo demostró que esto es falso: hay 6 eventos en los que se disparó la nueva arquitectura pero no la antigua. Por lo tanto,* ***ninguna RN generaliza a la otra****. Pero hemos estado pensando en los últimos minutos en que la arquitectura antigua generaliza la nueva. Detectar la causa de nuestra confusión.* *Respuesta*

14 *Ejercicio.* *Redacte toda nuestra historia como una aplicación del Método Científico.* Respuesta

Nuestros resultados principales son, por tanto:

1. Hemos considerado dos RNs, la antigua con dos perceptrones, cada uno con muchas entradas, y la nueva con 8 perceptrones y menos entradas a cada perceptrón. En sentido estricto, las dos RNs son independientes y deben estudiarse por separado.

2. Nuestra implementación de la estrategia *divide y vencerás* no funcionó. Por tanto, no es incondicionalmente correcta. Quizás nuestro enfoque sea erróneo: no es la cuadrícula la que debe dividirse, sino los símbolos dibujados en la cuadrícula misma.

3. La arquitectura antigua generaliza estadísticamente a la nueva: en casi el 90% de los casos, cuando la nueva arquitectura se activa, la antigua también.

4. La arquitectura antigua tiene 2 celdas mientras que la nueva tiene 8, y si consideramos la degeneración, el número de celdas puede superar las 32. Entonces, la nueva arquitectura no es buena para el planeta. La degeneración se refiere al hecho de que la nueva arquitectura necesita al menos 4 cromosomas diferentes, especificaciones de tolerancia, sesgos adicionales, para intentar ser tan buena como la antigua RN.

15 *Aplicación.* *Nuestros hallazgos se pueden expresar de manera sucinta: las RNs cuyos perceptrones tienen muchas entradas para analizar imágenes grandes son más baratas y más efectivas que los sistemas de perceptrones con menos entradas para analizar subpatrones. Esta es una explicación del hecho de que las neuronas siempre tengan muchas dendritas o entradas cuyo número puede llegar a 1000.*

16 ***Desafío.*** *Replantear nuestra lucha contra la complejidad siguiendo un formato axiomático, es decir, cuando todo fluye correctamente a partir de principios verdaderos.*

17 ***Graduación.*** *Nuestra tesis es que las redes neuronales, tanto biológicas como artificiales, cuyas unidades tienen muchas entradas para analizar patrones complejos son mejores que sistemas con muchos subsistemas que analizan subpatrones. La justificación final es que las primeras pueden clasificar correctamente más mapas de bits y a un coste menor. Apreciado Lector: búrlese del Autor de este artículo indicando claramente su evidencia con que respalda esta tesis. Ahora es el momento adecuado para que haga Usted algo mejor.*

6 CONCLUSION

Las RNs son entidades complejas cuyo estudio puede fácilmente enredar al investigador. En nuestro caso, pensamos que podríamos obtener un gran beneficio aplicando la conocida estrategia de las Ciencias de la Computación de *Divide y vencerás*. Pero fallamos. Sin embargo, utilizamos el Método Científico para ayudarnos a comprender las múltiples y por fin simples causas de confusión. Ese método compara predicciones de creencias con resultados observados y con discusiones despiadadas. Verificamos que ese método es un gran destructor de mitos. De todos modos, consideramos que las RNs con numerosas entradas para reconocer patrones grandes son más baratas y tienen más capacidades de reconocimiento que conglomerados de RNs pequeñas que apuntan a subpatrones. Presentamos ésto como una explicación al por qué las neuronas están tan obsesionadas con grandes números de entradas o dendritas que pueden llegar a 1000.

7 RESPUESTAS A LOS EJERCICIOS

1, página 8. El mapa de bits del arquetipo0 es 111101111. Por lo tanto, la desigualdad correspondiente dice:

$$S = e_0 + e_1 + e_2 + e_3 - e_4 + 1 + e_5 + e_6 + e_7 + e_8 =$$
$$e_0 + e_1 + e_2 + e_3 - e_4 + e_5 + e_6 + e_7 + e_8 + 1 \geq 9$$

Si reemplazamos las variables por los bits correspondientes, obtenemos $1+1+1+1-0+1+1+1+1+1 = 9 \geq 9$. Como esto es cierto, decimos que este perceptrón ha reconocido el arquetipo 0. Este perceptrón no reconoce ningún otro mapa de bits porque cualquier cambio será castigado con una pérdida de fuerza en el estímulo y el umbral no será alcanzado.

3, página 14. La RN del programa `divideP1.html` reconoce sólo un mapa de bits. La razón es que cada perceptrón de la capa oculta reconoce un único mapa de bits. Además, el perceptrón de decisión P_1 es un perceptrón AND que exige que todas sus entradas sean 1 o verdadero. Por lo tanto, todo el conjunto también reconoce sólo un mapa de bits. Nuestro razonamiento es puesto a prueba por el programa `divideConquerP1b.html` donde la RN se alimenta con mapas de bits aleatorios. No se espera que ninguno sea reconocido a menos que coincida con el arquetipo que representa 1. La prueba ha sido superado.

4, página 14. La idea es exactamente la misma que en el problema de reconocer el arquetipo 1. Sin embargo, los tres submapas de bits no son iguales y, por tanto, el código tiene una variación menor. Concretamente:

El mapa de bits del arquetipo de 0 es 111101111. Los tres submapas de bits son 111 101 111. Por lo tanto, los parámetros de los perceptrones AND que los reconocen son:

```
Para 111:
inputs = [1,1,1]
pesos = [1,1,1]
sesgo = 0
umbral = 3

Para 101:
inputs = [101]
pesos = [1, -1,1]
sesgos = 1
umbral = 3
```

Los perceptrones de la capa oculta están conectados al perceptrón de decisión P0, que es un perceptrón AND. El código correspondiente está en el programa `divideP0.html`.

5, página 14. El program es correcto.

8, página 17. La prueba fue superada: cuando todos los sesgos adicionales se igualan a cero, sólo se deben reconocer los dos arquetipos porque esto equivale a establecer una igualdad $S = T$.

9, página 17. Las exploraciones muestran rápidamente que cuando la tolerancia es alta, es decir, cuando los sesgos aleatorios adicionales toman valores enteros, 0,1,...,9, abundan las contradicciones. Pero cuando la tolerancia es baja, es decir, cuando sesgos aleatorios adicionales toman valores 0, 1 y 2, es posible que no aparezcan contradicciones.

10, página 17. Comenzamos nuestro estudio del poder de la Evolución para ajustar los valores de los sesgos adicionales para maximizar las coincidencias sin contradicciones mediante el uso de valores bajos de los sesgos adicionales. Eso fue controlado por la variable *usedAlphabet*, línea 150. Para tolerancia baja, es 01, para valores

altos es 0123456789. Empezamos trabajando con una población de 100 individuos.

Para poner a prueba el código, utilizamos solo 0 como valor posible para los 8 sesgos adicionales. El resultado fue correcto: cuando no hay sesgos adicionales, sólo se reconocen los arquetipos, y todo lo demás debe permanecer como desconocido.

A continuación, nos preguntamos qué sucede cuando ampliamos el alfabeto. Para empezar, usamos el alfabeto 01: en la población inicial el número de coincidencias llegó a 128 bajo 0 desajustes, por lo que el número de incógnitas fue 384 y no hubo más Evolución durante 10 generaciones.

Con el alfabeto 012 los desajustes desaparecieron en la generación 18 con 128 coincidencias y 384 desajustes. No hubo más avances hasta la generación 30, cuando el programa se detuvo.

Alfabeto 0123: debido a problemas con la memoria, el tamaño de la población se redujo a 30. Además, los desajustes nunca desaparecieron, por lo que los castigamos con más fuerza. La función de puntuación utilizada hasta aquí era:

```
score = numberOfMatches - 7 * numberOfMismatches;
```

Ahora usaremos:

```
score = numberOfMatches - 20 * numberOfMismatches;
```

La dinámica, según el mejor resultado de cada generación, se estabilizó antes de la generación 50: 128 coincidencias, 384 incógnitas y 0 desajustes. Observado durante 500 generaciones.

Con los mismos parámetros, ejecutamos el programa con el alfabeto 01234: la población se estableció en 25 para aliviar los problemas de memoria. La evolución logró después de 700 generaciones un máximo de 64 coincidencias bajo 0 desajustes. Por esta razón, la función de puntuación se redefinió como

```
score = numberOfMatches - 13 * numberOfMismatches;
```

Se redujo el castigo por desajustes para permitir una evolución más rápida. Sin embargo, la situación se mantuvo igual. Pero como nos atormentaba el deseo de saber si 128 era el número máximo de coincidencias sin desajustes, nos vimos obligados a ejecutar el programa durante muchas generaciones más. Así, le dimos una oportunidad al programa en una computadora portátil con 16 Gigas (hemos estado trabajando con 8 Gigas con Mozilla bajo Ubuntu Linux 24.04).

Con una población de 75 habitantes, alfabeto 01234, el programa se ejecutó durante 100.000 generaciones. En la generación 100, el número de coincidencias logrado fue de 92 con 0 desajustes y permaneció invariante por el resto de la vida. Concluimos que el alfabeto 01234 es demasiado ruidoso y perturba el papel asertivo de la Evolución. Pero para excluir cualquier conclusión falsa debida al apresuramiento, ejecutamos durante 100.000 generaciones el alfabeto 0123. En este caso, en la generación 100 ya se alcanzaron 128 coincidencias sin un solo desajuste y la situación permaneció invariable durante 9900 generaciones más.

¿Cuál podría ser el resultado cuando ejecutamos el alfabeto 012 durante 100000 generaciones? Se alcanzaron 92 coincidencias con 0 desajustes en la generación 100. La situación permaneció congelada por el resto de la vida. El cromosoma ganador fue el mismo todo el tiempo. En otra serie, se encontraron 128 coincidencias con el cromosoma 11101110 en la primera generación, sin discrepancias. Esta situación se mantuvo sin cambios durante 100.000 generaciones. En una tercera corrida, un campeón inútil fue diferente en la generación 100 al de la primera generación. Sin cambios durante las 99900 generaciones siguientes.

¿Y qué pasa con el alfabeto 01? El valor máximo se alcanzó en la generación inicial: 128 coincidencias con cero desajustes con el cromosoma 11100101. La situación permaneció congelada durante 99999 generaciones más.

Nuestra principal preocupación es: ¿por qué todo se congela tan rápidamente? Nuestra primera idea fue que las tasas de mutación y

recombinación eran demasiado bajas. Entonces, en lugar de una tasa de mutación de 0,0005 y una tasa de recombinación de 0,1, la tasa de mutación se amplificó 10 veces y la de recombinación 3 veces. El alfabeto era 012. Resultado: la generación inicial ya encontró el valor máximo, 128 coincidencias con 0 discrepancias. El cromosoma correspondiente era 11100101 pero luego, el número de coincidencias se redujo a 92 en la generación 100 con el cromosoma 11101200. El valor 128 se recuperó con el cromosoma 11101110 en la generación 3800 y permaneció así hasta la generación 100000. Todo eso sucedió mientras el número de coincidencias siempre era 0.

La conclusión neta es que el número máximo de coincidencias de esta RN con 8 perceptrones es 128, que es casi la mitad del número que alcanzó la antigua RN con 2 perceptrones que se consideró en nuestra *Introducción*. Esto es un gran fracaso para nosotros y no tenemos idea de la causa, aparte de imaginar que la variabilidad alcanzada no fue representativa. Por lo tanto, se debe realizar un experimento: calentar la variabilidad aumentando las tasas de mutación y recombinación mientras se disminuye el tamaño de la población para aumentar los efectos de muestreo. Con ello se podrían visitar regiones más grandes del espacio disponible y el 128 sería desafiado. Así, la tasa de mutación se fijó en 0,05 y la tasa de recombinación en 0,4, con un tamaño de población de 25 (en lugar de 75) y los programas se ejecutaron durante 200.000 generaciones.

Resultados: Se reportaron diversos valores de coincidencias como los mejores de la generación, con y sin desajustes. El valor más alto sin discrepancias fue 128. Por lo tanto, creemos firmemente que 128 es el valor máximo que puede alcanzar la nueva RN.

cada uno de estos experimentos toman casi 80 minutos en una computadora portátil antigua dedicada con Windows 10 y Microsoft Edge con 16 GB de RAM.

12, página 20. Si bien la pregunta parece admitir una respuesta sencilla, debemos aclarar una sutileza antes de volver a ella.

La comparación de desigualdades algebraicas nos ha permitido hacer deducciones aparentemente correctas. Entonces, sigamos con ellas, pero ésta vez, críticamente. Pasar de la arquitectura antigua a la nueva, implica la división del total del sesgo adicional en la arquitectura antigua en 3 partes en la nueva. Este procedimiento de división se puede realizar de diversas formas. Por ejemplo, si el sesgo adicional total es 2, podemos dividirlo en tres partes de varias formas: como $0+1+1$, $1+0+1$ o $1+1+0$, decimos que la división es degenerada. Pero atención, el Proceso Evolutivo encuentra un cromosoma óptimo que es constante, no se puede variar a menos que ejecutemos el programa una vez más. De aquí obtenemos una predicción:

El valor máximo, 128 coincidencias con 0 discrepancias, es degenerado, es decir, se puede alcanzar con varios cromosomas que deben encontrarse en series diversas.

Para probar la predicción, ejecutamos el programa `divideConquerEvolNN.htm` con un tamaño de población de 100 durante 10 generaciones.

Obtuvimos los siguientes cromosomas que codifican 8 sesgos adicionales que produjeron el número máximo de coincidencias, 128:

```
01010101
01011110
11100101
11101110.
```

La predicción se verificó: el sesgo adicional óptimo de la arquitectura antigua no se corresponde con un cromosoma sino con un grupo de ellos en la nueva arquitectura. Por lo tanto, no excluimos la posibilidad de que el conjunto completo de cromosomas óptimos de la nueva arquitectura pueda ser tan bueno como la arquitectura anterior o incluso mejor. Esto debe ser verificado y para ello desarrollamos el programa `divideConquerEvolNN2.html` que toma un cromosoma y genera la correspondiente reclasificación de los 512 mapas de bits, 256 de los cuales fueron asignados por la distancia

BBB como una deformación del arquetipo1 y los otros 256 como del arquetipo0.

Con los cuatro cromosomas citados obtenemos cuatro reclasificaciones que se pegaron en una hoja de cálculo para compararlas. Descubrimos que los diferentes resultados se complementan entre sí. Todo el conjunto fue capaz de alcanzar 200 coincidencias sin un solo desajuste que se distribuyeron en clases según la distancia BBB a los arquetipos de la siguiente manera:

Distance	Archetype1	Archetype0
0	1	1
1	9	9
2	36	36
3	48	48
4	6	6
5	0	0
Total	100	100

En la primera columna vemos la distancia BBB a cada uno de los arquetipos. En el segundo, el número de mapas de bits que la nueva arquitectura fue capaz de clasificar correctamente y sin contradicciones como deformación del arquetipo1. La tercera columna muestra los valores correspondientes para el arquetipo0.

La tabla correspondiente producida por la arquitectura antigua es, según el programa `compareTwoClassifications3.html` de nuestra *Intro*, la siguiente

Distance	Archetype1	Archetype0
0	1	1
1	9	9
2	36	36
3	84	84
4	0	0
5	0	0
Total	130	130

El número de mapas de bits que están correctamente clasificados coincide para las distancias 0, 1 y 2. Para la distancia 3 es mejor la arquitectura antigua (clasifica correctamente las 84 instancias posibles, mientras que la nueva sólo 48), para la distancia 4 es mejor la nueva, 6 contra 0. Así, lo que establecimos en el ejercicio anterior es falso: la arquitectura antigua no siempre se dispara cuando lo hace la nueva. Sin embargo, cuando la nueva arquitectura se dispara, la vieja también lo hace en una proporción elevada, más del 90%.

13, página 20. La fuente de la confusión es que nuestro objetivo, consignado en la función de puntuación del Proceso Evolutivo, es incompatible con la supuesta propiedad de la suma de desigualdades. La tarea que se nos asignó a nosotros y a la Evolución fue diseñar una RN que clasifique los mapas de bits en una de dos clases maximizando el número de coincidencias pero sin discrepancias ni contradicciones. Obtuvimos tres subtareas representadas por tres desigualdades y recordamos que las desigualdades de números reales se pueden sumar. De esa manera, predijimos que la arquitectura antigua se activa cuando lo hace la nueva y, por lo tanto, la arquitectura antigua generaliza a la nueva. Pero el conteo directo dio que esto era falso. ¿Por qué? El problema es que el resultado de sumar las desigualdades se ocupa de las coincidencias pero no de las contradicciones o desajustes, como se indica explícitamente en la función de puntuación. Por lo tanto, no podemos afirmar que cuando la nueva arquitectura se dispara, la vieja también lo hace.

14, página 21. El Método Científico es un procedimiento para probar creencias mediante la comparación de sus predicciones con los resultados experimentales observados y con discusiones esclarecedoras. Si las predicciones se cumplen y se resisten a una discusión despiadada, la creencia gana credibilidad y vale la pena crear más tests. Si los resultados contradicen la creencia, ésta se abandona o se modifica para volver a ponerla a prueba.

Nuestra creencia inicial era la estrategia *divide y vencerás* de las

Ciencias de la Computación para resolver problemas: las tareas se pueden subdividir para obtener subtareas más manejables y de más fácil solución. En nuestra opinión, esta máxima queda claramente ilustrada por los alfabetos que dividen el problema de transcribir expresiones en transcripciones de fonemas que deben unirse. Aspiramos a hacer lo mismo con los gráficos, dividiendo una cuadrícula donde los gráficos se dibujan en subcuadrículas.

Para poner a prueba nuestra creencia, construimos dos RNs. La antigua tenía dos perceptrones y estaba destinada a reconocer patrones dibujados en una cuadrícula de 3×3. La nueva tenía como objetivo reconocer subpatrones en cada fila de la cuadrícula, 3 subproblemas en total.

Basándonos en la idea de los alfabetos, imaginamos tácitamente que la nueva RN sería más potente que la antigua, es decir, se dispararía correctamente más veces que la antigua. Esto resultó ser falso: mediante conteo directo, encontramos que mientras la antigua era capaz de reconocer, sin desajustes, 260 patrones, la nueva era capaz de reconocer sólo 128, también sin desajustes. Esto muestra que hay muchos casos en los que la RN antigua se activa pero la nueva no. Vemos que la predicción es falsa: la nueva arquitectura no es mejor que la antigua.

A continuación, al sumar las desigualdades que caracterizan a las sub-RNs, predijimos que la antigua RN debe dispararse cuando la nueva lo hace, es decir, si la nueva RN se dispara, la antigua también debe hacerlo. El conteo directo mostró que esto también es falso porque la nueva RN detectó 6 mapas de bits para la distancia 4, pero el número correspondiente era 0 para la antigua RN. Por eso, nuestro veredicto final es: **ninguna RN generaliza la otra**, pero estadísticamente, la vieja generaliza la nueva y además es más barata.

También obtuvimos una predicción que fue verificada por la observación: si el sesgo adicional total es 2, podemos dividirlo en tres partes como $0+1+1$, $1+0+1$ o $1+1+0$, decimos que la división es degenerada. Esto implica que debe haber varios cromosomas para la

nueva arquitectura que representen el valor máximo observado, 128. Encontramos 4 de ellos, que juntos fueron capaces de reconocer 200 mapas de bits, una mejora de casi el 50%. Sin embargo, ésta no es lo suficientemente interesante porque estamos muy lejos del valor máximo posible, reconocer 512 mapas de bits de 512.

8 BIBLIOGRAFÍA

References

[1] Divide and Conquer Algorithm (2024)
https://www.geeksforgeeks.org/divide-and-conquer/ 3

[2] José Rodríguez (2024)
Evolutionary Neural Networks 1. Introduction.
https://www.ejristos.com/Evol%20Neural%20Networks/EvolNNintro.pdf
https://www.ejristos.com/Evol%20Neural%20Networks/EvolNNintroSpanish.pdf 3

[3] José Rodríguez (2024)
Evolutionary Neural Networks 2. Simplification.
https://ejristos.com/Evol%20Neural%20Networks/EvolNN2.pdf
https://ejristos.com/Evol%20Neural%20Networks/EvolNN2Spanish.pdf 3

Estos enlaces estaban activos el día 15/Nov/2024