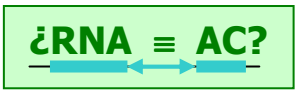
Relación entre Redes Neuronales Artificiales y Autómatas Celulares



Juan C. Vázquez [1], Marcelo M. Marciszack [1], Osvaldo F. Martínez [1]
Julio Castillo [2], Leticia Constable [2], Marcelo Archidiácono [2], Diego Serrano [2]

[1] Universidad Tecnológica Nacional, Facultad Regional Córdoba, Departamento de Ingeniería en Sistemas de Información, Maestro López y Cruz Roja Argentina, 5016, Ciudad Universitaria, Córdoba, Argentina
jcvazquez@acm.org, marcisza@bbs.frc.utn.edu.ar, martinezof@arnet.com.ar
http://www.frc.utn.edu.ar
[2] Universidad Tecnológica Nacional, Facultad Regional Córdoba, Departamento de Ingeniería en Sistemas de Información, Maestro López y Cruz Roja Argentina, 5016, Ciudad Universitaria, Córdoba, Argentina
Alumnos Asistentes del Proyecto E25/078 RNA-AC

Resumen. Los autómatas celulares son modelos matemáticos utilizados para estudiar fenómenos de autoorganización y para simular sistemas físicos, químicos y biológicos de alta complejidad, quedando enmarcados en la teoría como modelos de sistemas evolutivos. Por su lado, las redes neuronales artificiales son modelos de la Inteligencia Artificial que personifican el enfoque subsimbólico y suelen ser utilizados para reconocimiento de patrones, aproximación de funciones complejas y otras muchas aplicaciones. Se intentan determinar posibles relaciones entre estos dos modelos mirando las redes neuronales tipo perceptron multicapa como un sistema evolutivo durante su fase de aprendizaje.
Se proponen distintas codificaciones de la historia evolutiva de los pesos sinápticos como cadenas binarias y un algoritmo de descubrimiento que determine si esta historia podría ser generada como evolución espacio-temporal de algún autómata celular. Esta relación permitiría comprender aspectos aún poco claros en ambos modelos.

1. Introducción

El proyecto RNA-AC, estudia posibles relaciones entre los momentos de aprendizaje y reconocimiento de las redes neuronales artificiales (RNA) y la evolución espacio-temporal de los autómatas celulares unidimensionales (AC), modelos computacionales distintos; esta búsqueda está basada en la experimentación computacional siguiendo el enfoque fenotípico que utilizó *Stephen Wolfram* [Wolfram-1994] en sus trabajos de la década de 1980.

Llamamos *experimentación computacional*, al desarrollo de programas de computación que implementen los distintos modelos bajo estudio, junto con su posterior ejecución reiteradas veces variando los parámetros involucrados para registrar los resultados obtenidos y conocer así su comportamiento. Con *enfoque fenotípico* nos referimos a que se buscan relaciones *a posteriori*, observando el resultados de los experimentos e intentando extraer conclusiones, en contraposición a un enfoque genotípico que trataría de *deducir a priori* alguna relación para luego contrastarla contra experimentos diseñados a tal fin.

Existen numerosos paquetes de software que implementan los modelos de AC y RNA; con el objeto de contrastar los resultados de nuestros algoritmos, se han accedido a muchos de ellos y se han probado algunos [López-2004], pero en general éstos no tienen detalles de implementación, su funcionalidad es limitada, no están bien documentados o son poco claros; además, no permiten guardar historia de la evolución producida por los pesos sinápticos de las redes neuronales durante su aprendizaje, por lo se han debido desarrollar la mayoría de nuestras herramientas en nuestro laboratorio. El problema principal es que todos las usan y pocos se interesan por qué ocurre "dentro" de la red durante su aprendizaje y funcionamiento, que es lo que a nuestros propósitos interesa.

2. Modelos Formales Bajo Estudio

2.1. Autómata celular unidimensional

En el caso más simple, un autómata celular consiste de un arreglo unidimensional infinito de celdas contiguas que pueden contener cada una un valor entero entre **0** (cero) y ***k-1*** para $\mathbf{k} \in \mathbf{Z}^+$, $\mathbf{k} \geq \mathbf{2}$. En el caso $\mathbf{k = 2}$ llamamos al autómata celular *elemental*.

Los valores de las celdas evolucionan sincrónicamente en etapas de tiempo discreto, de acuerdo a una regla sencilla idéntica para todas las celdas del autómata (usualmente una función booleana en autómatas celulares elementales) que involucra solamente el valor contenido en la celda misma y en sus ***r*** vecinas inmediatas a ambos lados, en el instante anterior; ***r*** recibe el nombre de *rango de la regla* y hablamos en este caso de una r-vecindad.

t	•••	0	0	1	0	0	•••
	$a_i^{(t+1)} = a_{i-1}^{(t)} \vee a_{i+1}^{(t)}$						
t + 1	•••	**0**	**1**	**0**	**1**	**0**	•••

Fig. 1. Un instante en la evolución de un autómata celular unidimensional elemental, su regla de evolución y su estado en el instante subsiguiente *t + 1*. Los puntos al inicio y al final del arreglo indican que en el modelo teórico, éste es infinito. En cada localidad o celda del autómata sólo se puede encontrar un valor entero que en cada instante de tiempo es actualizado mediante la regla de evolución

En la Fig. 1, el autómata celular es elemental (***k = 2***, ***r = 1***), su regla de evolución es la suma lógica de los valores de la celda inmediata anterior e inmediata posterior a la celda en proceso y el rango de la regla es **1**. El subíndice de la notación corresponde a la posición de la celda dentro del arreglo y el superíndice entre paréntesis indica la etapa del valor accedido. Nótese que para celdas binarias (sólo valores cero o uno) y una 1-vecindad (tres celdas), pueden establecerse 256 reglas distintas.

El arreglo se supone infinito, por lo que a los fines prácticos debe elegirse un número entero positivo ***N*** de celdas e imponerse una condición de contorno a las celdas extremas del arreglo finito, la cual puede consistir en mantenerlas en un estado fijo (activo o no-activo, cero o uno) o en tomar el arreglo como circular.

Desde el punto de vista de la Teoría de la Computación, pero teniendo en cuenta que debe ser programada en computadores reales, podemos definir el AC como una máquina formal. Así, definimos el caso general de un autómata celular unidimensional como la quíntupla:

$$(\Sigma, N, C, r, \varphi) \qquad (1)$$

donde:

- Σ es un conjunto finito y no vacío de ***k*** símbolos, denominado *alfabeto* del autómata. Usualmente está compuesto por los enteros de **0** a **k-1**
- **N** es la cantidad de celdas del autómata celular real
- **C** especifica la condición de contorno impuesta a las celdas extremas. Usualmente se mantiene su valor fijo igual al neutro de la función φ o se establece circularidad asumiendo que a la última celda le sigue la primera y viceversa
- **r** es el tamaño de la vecindad que incide sobre el próximo valor de las celdas
- $\varphi:\Sigma^{2r+1}\rightarrow\Sigma$ es la regla de actualización de los valores de las celdas que tomando **r** celdas a cada lado de la celda objetivo y ella misma, calcula su nuevo valor

Nótese que no se ha incluido un conjunto de estados, común en las máquinas abstractas; en realidad, el autómata sí puede decirse que tiene un estado en cada instante de tiempo constituido por la cadena $\alpha_t \in \Sigma^N$ de símbolos del alfabeto que se encuentra en el instante **t** en las celdas; normalmente esto es referido en la bibliografía como la *configuración* del autómata y no como su estado. La aplicación de φ a cada celda del autómata en el tiempo **t** traduce la cadena α_{t-1} en la cadena α_t por lo que podríamos hablar inclusive de una función $\Phi:\Sigma^N \rightarrow \Sigma^N$ de transición de estado a estado, pero creemos que la definición dada es más clara en este punto de nuestro estudio.

Dada una configuración inicial α_0 arbitraria, a medida que transcurre el tiempo (discreto), se genera una sucesión de configuraciones $\alpha_0, \alpha_1, \ldots \alpha_t, \ldots$ que denominamos *evolución espacio-temporal* del autómata celular unidimensional.

Una de las características de estas máquinas que presentan interés, es el tipo de evolución espacio-temporal que desarrollan; a partir de una configuración inicial aleatoria se generan estructuras que conforman ciertos patrones, contradiciendo la segunda ley de la termodinámica, ya que disminuye su entropía con el tiempo; si bien esto no es un sistema físico, ésta aparente contradicción se debe a que la evolución del AC es un fenómeno irreversible. Wolfram catalogó estas estructuras en *cuatro categorías* y especuló sobre su relación con los cuatro tipos de lenguajes formales

Fig. 2. Un autómata celular elemental tipo II de Wolfram. Nótese cierta irregularidad en las primeras etapas de evolución, para luego seguir con una configuración fija de valores

categorizados por Chomsky (regulares, independientes del contexto, sensibles al contexto, con estructura de frase) [Hopcroft-1979] y con los cuatro tipos de máquinas de estado finito que reconocen esos mismos lenguajes (autómatas finitos, autómatas con pila, autómatas linealmente acotados y máquinas de Turing).

Para nuestro trabajo, es de particular interés el *TIPO II DE WOLFRAM* que es una clase de autómatas celulares que evolucionan con patrones irregulares o no definidos al principio, pero que a partir de un instante y en adelante llegan a una configuración fija y estable, como el que se muestra en la fig. 2. Se verá más adelante porqué esto es importante para nosotros.

2.2. Redes Neuronales Artificiales

En inteligencia artificial, se proponen básicamente dos enfoques para lograr de un programa comportamiento inteligente: el *enfoque simbólico* sostiene que el objetivo se logrará emulando la forma en que los humanos piensan, esto es, haciendo manipulación de símbolos mediante reglas lógicas y procesando con esta metodología grandes bases de conocimiento; por otro lado, el *enfoque subsimbólico* propone emular por hardware o software la forma en la que la naturaleza resuelve los problemas, en particular, el llamado modelo conexionista que emula el "hardware" del sistema nervioso, con la esperanza de que el comportamiento inteligente surgirá como una propiedad *emergente* de la complejidad de millones de elementos simples cooperando interconectados.

Una red neuronal artificial es un modelo computacional inspirado en el sistema nervioso de los animales superiores (en realidad en una simplificación extrema del mismo), que responde al enfoque subsimbólico de la IA. Consiste de una cantidad importante de elementos de modesto poder de cálculo, *las neuronas*, fuertemente interconectadas por canales que permiten el transporte de información entre ellas (axones y dendritas).

Se han diseñado infinidad de topologías distintas de conexionado de redes neuronales y aún muchos tipos distintos de neuronas. En [Hilera-1995], [Bríο-1997] y otros autores que figuran en las referencias puede encontrarse una extensa y variada clasificación de redes neuronales.

Esquemáticamente, una red neuronal artificial se desempeña como una función que recibe de su entorno entradas (m-tuplas de números reales) y entrega al mismo salidas (n-tuplas de números reales), como resultado del procesamiento de las entradas.

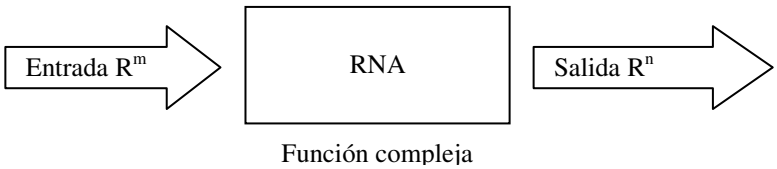


Fig. 3. La red neuronal artificial puede verse como una función compleja que traduce m-tuplas en n-tublas de números reales

Si la relación existente entre las entradas y las salidas es conocida y expresable mediante algoritmos (o fórmulas matemáticas) de complejidad a lo sumo polinómica, no se precisa de una RNA; por el contrario, si la función es desconocida (*aunque con fuertes indicios de existencia*) o la complejidad del algoritmo que la expresa es factorial o exponencial, entonces suele ser una opción recomendable, ya que se han diseñado algoritmos para que se le pueda *enseñar la tarea a efectuar mediante ejemplos*, logrando un *aprendizaje* automático de la relación existente entre entradas y salidas.

También podemos pensar en la red como un grafo dirigido (**N, C**) conexo y pesado, donde **N** es el conjunto de nodos (las neuronas de la red) y **C** el conjunto de arcos que conectan los nodos entre sí (definiendo quién hace *sinapsis* con quién).

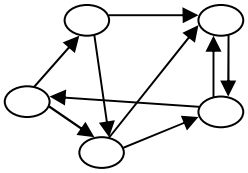


Fig. 4. La red neuronal artificial puede verse como una grafo conexo

Este grafo debe ser dotado de la capacidad de *almacenar conocimiento* y de *aprender*, esto es, lograr generalizar la relación existente entre entradas y salidas a partir de un conjunto finito de ejemplos de entrenamiento.

Para *determinar* una red neuronal artificial deben realizarse ciertas tareas a saber:

1) Definir un conjunto finito y no vacío **N** de elementos simples de cálculo de algún tipo (*neuronas*), no necesariamente todas iguales.
2) Definir una arquitectura de conexiones entre las neuronas, esto es, definir un conjunto **C** de pares de elementos de **N** que representen las relaciones entre las neuronas. En general, el conjunto **C** se elige de tal forma que no queden neuronas aisladas y su conformación impacta fuertemente en el modelo de red que se está definiendo y en el método de aprendizaje de la misma.
3) Definir una dinámica de actualización: sincrónica o asincrónica.
4) Elegir una tarea a realizar como un conjunto de pares de entradas y salidas $(\vec{e}(t) \in R^m, \vec{s}(t) \in R^n)$.

Por su lado, de cada elemento de la red es tal que, en un instante discreto ***t*** de tiempo:

a) Recibe una entrada $\vec{e}(t) \in R^d$ para algún $d \in Z^+$.

b) La procesa usando de una función de entrada f_e que pondera la entrada mediante pesos $\vec{p}(t) \in R^d$ (para el mismo $d \in Z^+$ que las entradas) que miden la eficacia de las conexiones sinápticas; al valor obtenido $f_e(\vec{e}(t), \vec{p}(t))$ se lo denomina *potencial pos-sináptico*.

c) Con el resultado $f_e(\vec{e}(t), \vec{p}(t))$, una función de activación f_a, que toma en cuenta el *estado actual* $q(t)$ y un *umbral de activación* propio de la neurona (como un peso más y con entrada constante igual a -1), calcula el nuevo *estado* $q(t+1)$ de la neurona.

d) Con el resultado $f_a(f_e(\vec{e}(t), \vec{p}(t)), \theta(t), q(t))$, una función de salida f_s calcula la salida de la neurona como un número real.

En definitiva, el proceso que hace la neurona general está determinado por:

$$q(t+1) = f_a(f_e(\vec{e}(t), \vec{p}(t)), \theta(t), q(t)) \tag{2}$$

$$s(t) = f_s(f_a(f_e(\vec{e}(t), \vec{p}(t)), \theta(t), q(t))) \tag{3}$$

aunque podría ser cualquier otro tipo de neurona, como por ejemplo, un autómata que basado en sus entradas, los pesos, el umbral y el estado actual, tomados como una cadena, cambie de estado y diga *acepto* (1) o *rechazo* (0) en su salida.

En nuestro trabajo utilizamos lo que podemos denominar *la neurona típica*; en ella, las entradas son ponderadas por los pesos sinápticos y sumadas, la función de activación resta el umbral a la entrada ponderada y la función de salida es de una sigmoidea aplicada al nivel de activación actual; no se toma en cuenta el estado anterior de la neurona.

$$s(t) = \frac{1}{1 + e^{-\alpha \cdot \left(\sum_{k=1}^{d} (e_k(t) \times p_k(t)) - \theta(t) \right)}} \tag{4}$$

En (4), el factor α determina la inclinación de la función sigmoidea en vecindades del origen. Con este tipo de neuronas se construyeron redes neuronales del tipo *perceptron multicapa* cuya arquitectura se muestra en fig. 5.

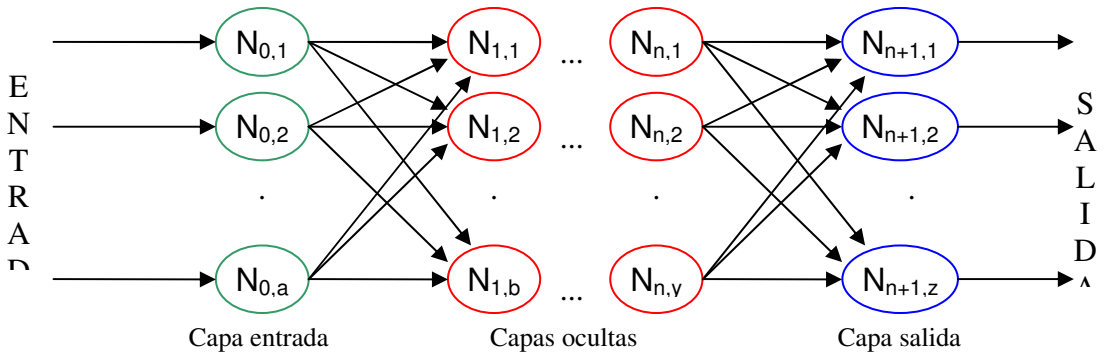


Fig. 5. Perceptron Multicapa. En esta arquitectura las neuronas se disponen en *capas* conectándose todas las neuronas de cada capa con las de la siguiente, siempre hacia delante (*feedforward*)

Los perceptrones multicapa son lo suficientemente conocidos como para no extendernos aquí sobre detalles de los mismos.

El proceso de aprendizaje en una red de este tipo se realiza mediante un proceso iterativo denominado *Retropropagación de Errores* (más conocido por su nombre en inglés *backpropagation* [BP]). Matemáticamente, este

algoritmo recurre a la técnica de *descenso por el gradiente*, que intenta disminuir en cada iteración el error producido por la red, medido como la diferencia entre la salida esperada y la salida calculada por la red; en realidad la función de error total se toma como el error medio cuadrático que produce la red y es una función sólo de los pesos sinápticos, supuesta la topología de conexionado y la cantidad y tipo de neuronas como parámetros fijos. El algoritmo se implementa según los siguientes pasos:

1. Asignar un valor aleatorio pequeño en el (-1,1) a todos los pesos sinápticos de la red y a todos los umbrales de las neuronas.
2. Elegir en forma aleatoria uno de los ***p*** pares de entrenamiento: $(\overrightarrow{e^{(r)}}, \overrightarrow{s^{(r)}})$ y calcular hacia delante la salida de cada neurona i-ésima, de cada capa j-ésima de la red, obteniendo:

$$s_{j,i}^{(r)}(t) = \begin{cases} e_i^{(r)} & \text{para j = 0 (capa entrada)} \\ \dfrac{1}{1+e^{-\alpha.(\sum_{k=1}^{h} s_{j-1,k}^{(r)}(t) \times p_{j,i,k}(t) - \theta_{j,i}(t))}} & \text{para j > 0 (resto de las capas).} \end{cases}$$

Aquí, h es la cantidad de neuronas de la capa (j - 1) - ésima y e sin índices, la base de los logaritmos neperianos.

3. Calcular los deltas de cada neurona i-ésima de la capa de salida, utilizando la salida real de la red y la salida esperada de la red:

$$\delta_{n+1,i}^{(r)}(t) = (s_{n+1,i}^{(r)}(t) - s_i^{(r)}).\frac{\partial s_{n+1,i}^{(r)}}{\partial f_a}, \text{ para nuestra función } \frac{\partial s_{n+1,i}^{(r)}}{\partial f_a} = s_{n+1,i}^{(r)}(t).(1 - s_{n+1,i}^{(r)}(t))$$

$$\delta_{n+1,i}^{(r)}(t) = (s_{n+1,i}^{(r)}(t) - s_i^{(r)}).(s_{n+1,i}^{(r)}(t).(1 - s_{n+1,i}^{(r)}(t)))$$

4. Calcular los deltas de cada neurona i-ésima de las capas j-ésimas precedentes (j=n,n-1,...,2,1 excluida la capa 0 de entrada), utilizando retropropagación de errores:

$$\delta_{j,i}^{(r)}(t) = \frac{\partial s_{j,i}^{(r)}}{\partial f_a}.(\sum_{k=1}^{h} \delta_{j+1,k}^{(r)}(t).p_{j+1,k,i}(t)), \text{ para nuestra función } \frac{\partial s_{j,i}^{(r)}}{\partial f_a} = s_{j,i}^{(r)}(t).(1 - s_{j,i}^{(r)}(t))$$

y *h* es la cantidad de neuronas de la capa $(j+1)$ - ésima.

$$\delta_{j,i}^{(r)}(t) = (s_{j,i}^{(r)}(t).(1 - s_{j,i}^{(r)}(t))).(\sum_{k=1}^{h} \delta_{j+1,k}^{(r)}(t).p_{j+1,k,i}(t))$$

5. Actualizar (sincrónicamente) los pesos usando:

$p_{j,i,k}(t+1) = p_{j,i,k}(t) + \eta.\delta_{j,k}^{(r)}(t).s_{j-1,i}^{(r)}(t)$ donde η es un factor de aprendizaje $0 < \eta < 1$.

6. Volver al paso 2 y repetir para cada patrón hasta que el error total tenga un nivel aceptablemente pequeño, predeterminado.

Este algoritmo realiza su trabajo cambiando en cada iteración el valor de los pesos sinápticos para que el error total de la red disminuya paulatinamente, logrando que las salidas reales calculadas desde las entradas ejemplo, se acerquen tanto como se quiera a las salidas conocidas para las mismas. *Nuestra suposición es que, si la red aprende su tarea, en algún momento estos pesos deben dejar de variar y mantenerse fijos o a lo sumo variando en forma oscilante (con una pequeña oscilación) alrededor de un valor fijo o asintóticamente hacia él*; si esto ocurre debería poder mostrarse esto como la evolución de un autómata celular del Tipo II de Wolfram y si esto es posible, tenemos la relación que estamos buscando: *puede verse el proceso de aprendizaje de la RNA como la evolución de un AC.*

Para estudiar esta relación, hemos desarrollado distintas codificaciones de la evolución de los pesos sinápticos de la red durante el proceso de aprendizaje, para poder graficarlos *al modo autómata celular.*

3. Diseño de la codificación de datos de la evolución de las RNA

Dada una arquitectura de conexionado, el conocimiento logrado por las RNA está representado por el conjunto de valores que adoptan los pesos sinápticos. En el modelo MLP que nos ocupó, los pesos son ajustados mediante aprendizaje fuera de línea por el algoritmo de retropropagación de errores.

Estos pesos son números en algún formato de punto flotante (de simple o de doble precisión), que van cambiando en MLP-BP a medida que la retropropagación de errores hace su trabajo de descenso por el gradiente de la función de error. En cada iteración se obtiene un conjunto de pesos sinápticos de la red y al juego completo de estos conjuntos es lo que llamamos *evolución de la RNA*, en este caso, durante su fase de aprendizaje.

Estos números de punto flotante, no se pueden graficar como están para que la evolución de la RNA sea comparable a la evolución especio-temporal de los AC; por ello se debe traducir estos números de punto flotante a binario (en primera instancia) de tal forma que los ceros y unos puedan mostrarse en un gráfico como puntos blancos y negros, respectivamente.

Hay aquí dos problemas que solucionar, que creemos son centrales para nuestro objetivo:

i) Conversión de cada valor de punto flotante a binario.

ii) Cómo ubicar las cadenas binarias obtenidas en el punto anterior en pantalla para mostrar su evolución.

3.1. Conversión de números de punto flotante a binario

Los lenguajes de programación y los equipos de computación actuales utilizan en general el estándar aritmético IEEE P754 de 1985 para representar en memoria los números de punto flotante. Este estándar establece la cantidad de dígitos binarios de una palabra de 32 bits (simple precisión) o de 64 bits (doble precisión y doble precisión extendida) utilizados para representar en notación exponencial el signo del número, el exponente del mismo y la mantisa, ***en ese orden***; además prescribe la forma en que el exponente y la mantisa se almacenan ya que no conforman la representación binaria pura de los mismos, sino complementos respecto de otros valores.

Punto Flotante de Simple Precisión (7 dígitos significativos)		
Signo de mantisa	Exponente	Mantisa
1 bit	8 bits	23 bits

Punto Flotante de Doble Precisión (15 dígitos significativos)		
Signo de mantisa	Exponente	Mantisa
1 bit	11 bits	52 bits

Esto hace que la representación binaria de un número de punto flotante "como está" en memoria, sea poco representativo para nuestros fines; el tema de complemento altera la información de crecimiento o decrecimiento de números para compararlos a nivel binario. Usaremos esta representación binaria como primera prueba de conversión pero con pocas esperanzas de que sea fructífera:

i-1) Cadena binaria del número en punto flotante, como está según el estándar IEEE P754 en memoria

En programación esto se logra redefiniendo el número de punto flotante como una cadena de caracteres del mismo largo y concatenando la secuencia binaria ASCII de cada carácter en una gran cadena binaria.

Hemos determinado que los valores de los pesos sinápticos tienden asintóticamente en algún momento del aprendizaje de las redes MLP-BP hacia valores fijos; por ello los números de punto flotante fijan sus dígitos más significativos y sólo varían sus dígitos menos significativos, por lo que se nos ocurre que las siguientes formas de efectuar la conversión a binario de los mismos puede ser significativa de la evolución de la RNA:

i-2) Cada dígito del número en punto flotante (todos los de la parte entera y algunos de la parte decimal) se convertirá a binario utilizando cuatro bits según el código BCD 8-4-2-1 tradicional.

i-3) El número en punto flotante es multiplicado por la unidad seguida de ceros de tal forma que los dígitos del número pasen a conformar un número entero largo y se tomará la cadena de bits que lo represente.

La conversión (i-2) es sencilla de programar ya que se formatea el número de punto flotante como una cadena de caracteres ASCII y luego (salteando la coma y cambiando los espacios que pudieran existir por ceros) se traduce dígito a dígito de la cadena a otra cadena de ceros y unos de largo fijo; el signo se podría indicar con un dígito uno al principio de esta cadena si el peso es negativo y cero si es positivo; sólo hay que tener cuidad al diseñar la rutina en la alineación para respetar lugares de unidades, decenas, etc.

Con respecto a (i-3), la cosa también parece sencilla. Sin embargo, cuando ensayamos prototipos de conversión en C, nos dimos cuenta (deberíamos habernos acordado de este tema) que los números negativos se almacenan como el complemento a dos del número entero en cuestión, por lo que la evolución quedaba distorsionada. Por ello pensamos en mezclar (i-2) con (i-3):

i-4) El número en punto flotante es multiplicado por la unidad seguida de ceros de tal forma que los dígitos del número pasen a conformar un número entero largo. Cada dígito de este número es entonces convertido a binario utilizando cuatro bits según el código BCD 8-4-2-1.

Nuevamente el tema de signo se codifica en este formato como un bit adicional en la posición de más a la izquierda: 1 = negativo, 0 = positivo.

Una quinta posibilidad, se agregó luego de que vimos que los pesos sinápticos de las MLP-BP que entrenamos con nuestros prototipos se mantienen acotados en general en un rango [-100,+100], si bien este rango podría cambiar para alguna red:

i-5) El número en punto flotante es multiplicado por la unidad seguida de ceros de tal forma que los dígitos del número pasen a conformar un número entero largo. Luego se barre toda la sucesión de valores obtenidos para todos los pesos y se obtiene el menor de ellos (el más negativo). Se vuelve a barrer toda la sucesión y a cada valor se le suma el mínimo calculado cambiado de signo, de tal forma que todos los valores pasen a ser positivos o nulos. Luego cada dígito de este número es entonces convertido a binario utilizando cuatro bits según el código BCD 8-4-2-1.

Así, ya no es necesario preocuparse por cómo codificar números negativos, por lo que inclusive se podrían aplicar en secuencia (i-5) y luego (i-3) tomando la representación binaria directa almacenada en memoria para efectuar la conversión:

i-6) El número en punto flotante es multiplicado por la unidad seguida de ceros de tal forma que los dígitos del número pasen a conformar un número entero largo. Luego se barre toda la sucesión de valores obtenidos para todos los pesos y se obtiene el menor de ellos (el más negativo). Se vuelve a barrer toda la sucesión y a cada valor se le suma el mínimo calculado cambiado de signo, de tal forma que todos los valores pasen a ser positivos o nulos. Se toma entonces como conversión, la cadena de bits de la memoria que almacena el entero largo.

3.2. Cómo ubicar las cadenas binarias obtenidas en el punto anterior en pantalla para mostrar su evolución

El segundo problema no tiene una solución razonable. Lo primero que seguramente veremos es cómo evoluciona gráficamente cada peso de la red usando cada una de las seis codificaciones propuestas; pero el tema de juntarlos a todos para conformar un gran gráfico al modo AC es otra cuestión.

A priori, no hay prioridades explícitas en la teoría para definir el orden en que los gráficos de los distintos pesos sinápticos puedan ser concatenados para obtener un único gráfico; podrían mostrarse juntos todos los pesos de las neuronas de una misma capa de la MLP-BP, pero aún dentro de una capa, la denominación de neuronas como *primera, segunda, …* es meramente visual; las funciones en las que se involucran estos pesos suelen ser sumatorias que verifican la propiedad conmutativa, por lo que la posición no es relevante.

Por ello decidimos ver los pesos sinápticos por separado y experimentar visualmente con distintas concatenaciones que pueden efectuarse, sin pretender ser exhaustivos y cubrirlas todas ya que las mismas responden a una ley factorial (permutaciones).

4. Generación de prototipos de AC y MLP-BP

En [Wolfram-1994] se muestra un catálogo de los distintos autómatas celulares de diverso tipo, y sus evoluciones espacio-temporales, pero queríamos verlos mostrados por un programa nuestro.

Recién iniciado el proyecto, construimos programas en VFP y C# para ver la evolución de autómatas celulares unidimensionales elementales, esto es, gobernados por los parámetros $r = 1$ (1-vecindad) y $k = 2$ (celdas binarias). Se incluyeron las opciones de cambio de condiciones de frontera para las celdas extremas (valore fijo o circularidad) como la de establecimiento de una configuración inicial de valores aleatorios colocados a las celdas o un único valor central. El gráfico de la figura 6, es una muestra de uno de estos prototipos en funcionamiento.

Puede verse aquí un ejemplo de evolución del AC bajo la regla 18 según la nomenclatura sugerida por [Wolfram-1994]; se generó evolución del autómata para 400 etapas utilizando la condición de frontera circular y partiendo de una configuración inicial de una sola celda central con el valor uno y todas las demás en cero.

Claramente se generan patrones en la figura; éstos responden a la propagación en el tiempo de la información original de la configuración inicial; cada celda influye en el valor de **2r+1** celdas en cada iteración por lo que en el tiempo **t**, el valor de las celdas de la configuración inicial debe haber afectado al menos a **2rt+t** celdas.

Un pequeño cambio en la configuración inicial del autómata puede acarrear tremendos cambios en los patrones generados.

Hemos pasado largo tiempo viendo la evolución que estos programas muestran de los AC, con la idea de acostumbrarnos al tipo de estructuras autoorganizadas; esto nos sirvió para los primeros cotejos visuales de la evolución gráfica de los pesos sinápticos codificados según alguno de los procedimientos descriptos anteriormente.

También se construyeron programas en VFP y C# para entrenamiento de redes neuronales tipo MLP; estos programas almacenan la historia de evolución de los pesos sinápticos en archivos para posterior análisis; otro programa luego los toma y genera conversiones a cadenas binarias y muestra gráficos de evolución según las codificaciones anteriormente propuestas.

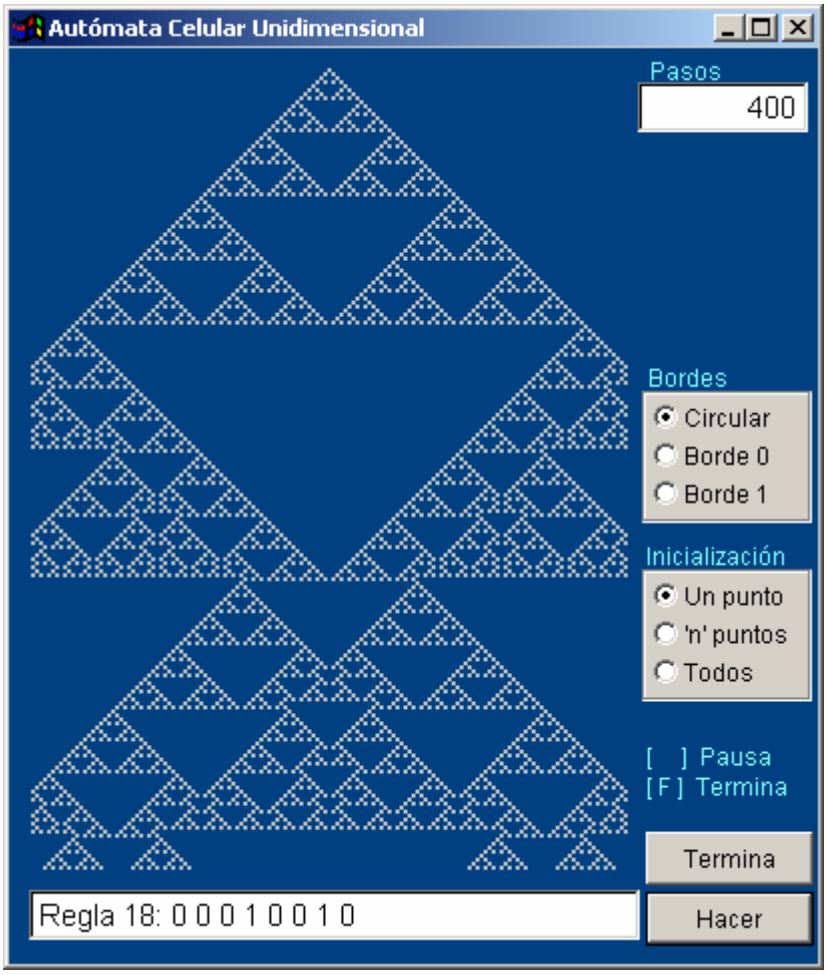


Fig. 6. Autómata Celular Elemental, mostrando la evolución de 400 iteraciones bajo la regla 18.

En una de las pruebas tempranas se obtuvo el gráfico de la fig. 7, utilizando el archivo de pesos generado por un programa aproximador de funciones mediante redes MLP. Realmente el gráfico tiene toda lo que estábamos buscando !!!. En el detalle ampliado, puede verse claramente que el formato es similar a un autómata tipo II de Wolfram.

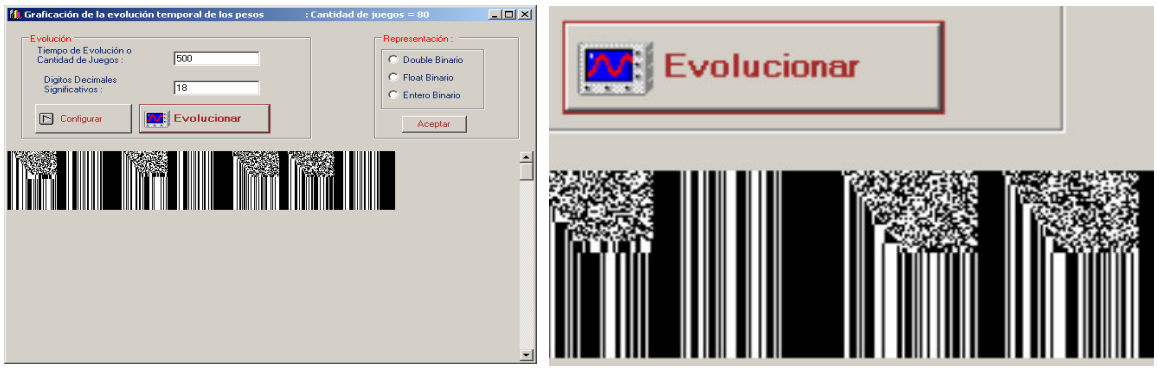


Fig. 7. Gráfico generado por pesos de la RNA “aproximador de funciones”, producido por un error.

Hubo algarabía en el laboratorio cuando vimos este gráfico, pero era demasiado fácil. Una revisión del programa que generó los pesos sinápticos reveló que tenía un error algorítmico, ya que cambiando los ejemplos de entrenamiento, el programa volvía a mostrar como resultado la misma función.

La buena noticia fue que ya podíamos ver archivos de pesos codificados según distintas opciones y en forma gráfica y habíamos “visto” a dónde queríamos llegar, aunque aún no habíamos llegado.

5. Diseño de algoritmo y construcción de programa de cotejo

Como indicamos, el primer abordaje en la búsqueda de semejanzas entre los gráficos de evolución de una RNA y de un AC, es el cotejo visual. Estando concientes que esto es lento y cargado de subjetividad, es que pensamos en la necesidad de buscar un método algorítmico de cotejo.

La evolución de los autómatas celulares es un proceso irreversible, esto es, desde una configuración alcanzada en el tiempo ***t*** es en general imposible obtener la configuración en el tiempo ***t - 1*** porque en el proceso directo se destruyó parte de la información; además, cada función booleana tiene infinitas funciones equivalentes (aunque esto no es un problema en sí, precisamente por la equivalencia de las mismas).

Sumado a esta restricción teórica del modelo, los autómatas celulares (aún los elementales con k = 2), pueden evolucionar de acuerdo a vecindades de cualquier tamaño, por lo que un algoritmo de descubrimiento de la regla que se implementa debería probar con r = 0, 1, 2, …, N, guardar todas las posibles funciones booleanas que traducen la configuración (t-1) a la configuración (t) y compararlas para ver alguna regularidad; esto es muy difícil de efectuar.

Todo esto nos llevó a pensar en la imposibilidad de construir un algoritmo general que analizando una evolución espacio-temporal generada por nuestras codificaciones en binario de los valores de los pesos sinápticos de redes MLP-BP durante su fase de aprendizaje, ***descubra*** la regla que podría implementar un AC equivalente.

Sin embargo si creemos que podemos cotejar algorítmicamente la ***plausibilidad*** de que una evolución de RNA corresponda efectivamente a la de algún AC elemental, llegando incluso a determinar por este medio la regla de evolución. La idea es la siguiente:

a) Partimos de una evolución almacenada en archivo plano, esto es una serie de registros de longitud fija con sólo ceros y unos obtenidos como conversión de los valores de los pesos sinápticos de una MLP-BP durante el proceso de aprendizaje, con alguna de las codificaciones propuestas. Cada registro corresponde a una etapa ***t*** en el proceso de aprendizaje.

b) Si los datos corresponden a algún AC elemental, quede decir que el registro t-ésimo se genera a partir del registro (t-1)-ésimo aplicando alguna regla que involucra una r-vecindad, de la cual no conocemos su rango. Entonces pueden probarse vecindades de rango 1, 2, …, N (donde N es el largo del registro o un entero prefijado mayor a uno) para todo el registro y ver si en algún caso resultan compatibles con los datos.
Por ejemplo, si en el registro t-ésimo tenemos para la i-ésima celda:

$$a_i^{(t)} = 0$$

siendo la 1-vecindad de la posición i-ésima del registro anterior:

$$a_{i-1}^{(t-1)} = 1 \qquad a_i^{(t-1)} = 0 \qquad a_{i+1}^{(t-1)} = 1$$

debería resultar para cualquier otro **0** del registro ***t*** una vecindad de **101** en el registro anterior; si esto no sucede en, digamos, más del 90% de los casos (para permitir algún ruido) quiere decir que la 1-vecindad no es compatible con la evolución de ningún autómata celular elemental.

c) Siguiendo el esquema anterior, si para **t = 1, 2, 3, …, último registro** puede conseguirse una r-vecindad común para algún $\mathbf{r \in [0,N]}$ compatible, entonces puede decirse que ***existe el AC elemental***, aunque no sabemos cual. Diremos en este caso que la relación buscada es plausible y deberemos buscar la regla por algún otro medio.

Este algoritmo ya está codificado y en funcionamiento; el mismo *descubre* si existe un AC que genera las cadenas binarias provenientes de la historia evolutiva de los pesos sinápticos de redes neuronales MLP durante su entrenamiento con BP. Estamos ahora desarrollando un esquema de prueba automático, para poder procesar gran cantidad de modelos de redes y problemas ejemplo, con el fin de obtener una estadística relevante.

6. Conclusiones

El proceso de aprendizaje en redes neuronales artificiales del tipo *Perceptron Multicapa*, puede ser visto como un proceso evolutivo donde los pesos sinápticos se adaptan en el tiempo. Esta historia puede ser codificada de diversas formas para ser contrastada contra el modelo de *Autómatas Celulares Elementales*. Este cotejo ya puede realizarse con nuestro programa de *descubrimiento* en forma automática para gran cantidad de situaciones posibles.

Aún no hemos determinado si existe una relación clara entre la evolución de los AC elementales y las RNA pensadas como sistemas evolutivos, pero tenemos todas las herramientas necesarias para confirmarlo o refutarlo. La experimentación que se desarrollará en las próximas etapas del proyecto RNA-AC nos brindará las respuestas finales.

Referencias

[Avnet-2000] Avnet J. (2000),
Computation, Dynamics and the Phase-Transition, http://www.theory.org/complexity/cdpt,
Fecha de consulta : 17/07/2003.

[Bose-1996] Bose N. / Liang P. (1996), *Neural Network Fundamentals with Graphs, Algorithms and Applications,* MacGraw-Hill, U.S.A..

[Brío-1997] del Brío M. / Molina Sanz A. (1997), *Redes Neuronales y Sistemas Borrosos,* RA-MA, Madrid, España.

[Castro-2002] Castro J., Mantas C., Benítez J. (2002),
Interpretation of Artificial Neural Networks by Means of Fuzzy Rules,
IEEE Transactions on Neural Network, Vol. 13, #1, pp. 101-116.

[Darwin1859] Darwin, C. (1859),
On the origins of species by means of natural selection, or the preservation of favoured race in the struggle for life, Traducción: *El origen de las especies,* (1992) Editorial Planeta, Barcelona, España.

[Das-1995] Das R., Crutchfield J., Mitchel M., Hanson J. (1995),
Proceedings of the Sixth International Conference on Genetic Algorithms, Morgan Kaufmann, pp. 336-343.

[Denker-1987] Denker J., Shwartz D., Wittner B., Solta S., Howard R., Jackel K., Hopfield J. (1987), *Large Automatic Learning, Rule Extraction and Generalization*, Complex Systems, Vol. 1, pp. 877-922.

[Deolikar-2002] Deolalikar V. (2002),
A Two-Layer Paradigm Capable of Forming Arbitrary Decision Regions in Input Space,
IEEE Transactions on Neural Network, Vol. 13, #1, pp. 15-21.

[Hilera-1995] Hilera J. / Martínez V. (1995), *Redes Neuronales Artificiales*, Addison-Wesley Iberoamericana, Wilmington, Delaware, USA.

[Hopcroft-1979] Hopcroft J. / Ullman J. (1979), *Introduction to Automata Theory, Language and Computation*, Addison-Wesley, New York, USA.

[López-2004] López Izquierdo J. M. (2004), *Entrenando y simulando una red neuronal en C#,*
www.microsoft.com/spanish/msdn/comunidad/mtj.net/voices/art173.asp, Microsoft MSDN Latinoamérica,
Fecha de consulta: octubre/2004.

[Newman-1994] Newman J. (1994),
El Mundo de las Matemáticas, Vol. 6, Grijalbo, Barcelona, España.

[Patterson-1996] Patterson D. W. (1996), *Artificial Neural Networks: Theory and Applications*, Prentice-Hall, Korea.

[Rumelhart-1986] Rumelhart D. y McClelland J. (1986),
Parallel Distributed Processing, Vol. 1: *Foundations*, MIT Press.

[Sipper-1997] Sipper M. (1997), *Evolution of Parallel Cellular Machines: The Cellular Programming Approach*, Springler, Berlín, Germany.

[Wolfram-1984] Wolfram S. (1984),
Universality and Complexity in Celullar Automata, Physica D, vol. 10, pp. 1-35.

[Wolfram-1985] Wolfram S. (1985),
Twenty Problems in Theory of Cellular Automata, Physica Scripta, vol. T9, pp. 170-183.

[Wolfram-1994] Wolfram S. (1994), *Cellular Automata and Complexity (collected papers)*, Addison-Wesley, Massachusetts, USA.

//