Biblioteca de componentes de modelado conexionista para Scicos

José Manuel Cuadra Troncoso
José Mira Mira
Dpto. de Inteligencia Artificial - ETSI Informática - UNED
jmcuadra@dia.uned.es

Índice

1. Introducción

Scicos [Sci07a] es un simulador de sistemas dinámicos que se encuentra dentro de la distribución del entorno de cálculo numérico Scilab [Sci07b]. En la red se encuentran manuales y tutoriales [Sci], algunos en castellano, de libre descarga. Aunque de Scilab sólo se necesitan unas nociones elementales, p. ej el capítulo segundo del manual en castellano [Esc05] y la primera sección del tercero, Scicos debe ser conocido con cierta profundidad, para lo cual se ha de descargar el tutorial de B. A. Delicado [Del98] y comprender al menos el contenido de su páginas 3 a 32. Toda esta documentación puede estar algo obsoleta para la última versión del programa, la 4.1.2 al escribir este papel, a la documentación actualizada se accede desde el menú de Scilab[1].

La biblioteca de módulos conexionistas para Scicos se compone de una paleta de bloques y circuitos de Scicos con los que elaborar circuitos neuronales de inspiración biológica. La paleta de bloques se encuentra en el fichero `bioconexionismo.cos`, se han incluido algunos bloques habituales de Scicos para hacer más cómodo el diseño, los bloques propiamente conexionistas tienen el fondo coloreado y una identificación al pie, ver figura 1. No se han creado GUIs específicas para la edición de los bloques, salvo algunos elementales, así el estudiante deberá internarse dentro de la estructura de los bloques para comprender su diseño y el alcance de sus modificaciones. Sin embargo para investigaciones de cierta complejidad es recomendable el uso de contextos de diagramas y la creación de GUIs para editar parámetros. En el resto de secciones iremos explicando los bloques y los circuitos así como simulaciones de ejemplo realizadas con ellos. Es muy conveniente tener la documentación y el tutorial de Scicos a mano, abrir en Scicos la paleta de bloques y ejecutar las simulaciones descritas mientras se siguen las explicaciones, de esta manera la lectura se convierte en un tutorial.

En los circuitos de Scicos se pueden considerar dos clases de bloques: los básicos, los propios de Scicos o los definidos por el usuario mediante funciones computacionales y, quizás, de interfaz, y los superbloques que son circuitos compuestos por bloques básicos y/o superbloques. El uso de superbloques permite observar el circuito a diferentes niveles de granularidad, podemos disponer de una vista general de funciones y conexiones principales u observar los distintos superbloques al nivel de detalle que deseemos hasta llegar a sus funciones básicas. Esta biblioteca está diseñada siguiendo este principio de encapsulación de funcionalidad, de manera que, a partir de bloques básicos y superbloques sencillos: umbrales, operaciones matemáticas, sinapsis, reglas de aprendizaje, etc., se construyen bloques más complejos: transductores, neuronas, con los que a su vez se pueden contruir otros como neuronas sensoras, osciladores de neuronas, etc.

El desarrollo de esta biblioteca se plantea como incremental pudiendo nutrirse tanto de aportaciones de los profesores como de los estudiantes, así como de la comunidad universitaria en general. La aportaciones se incluirán oficialmente en la biblioteca cuando sean aprobadas por sus administradores, papel que desempeñan los autores de esta guía.

Convenciones tipográficas Los nombres de ficheros, directorios y las órdenes con `letra de máquina de escribir`.

Los nombres de los bloques, si no están en un título, con *letra cursiva*.

Los caracteres < y > conteniendo una cadena de caracteres no se teclean, indican sustitución.

[1] Los menús de Scicos han cambiado de posición con la versión 4.1.2, consultar la ayuda de Scilab: entrar en la sección *Scicos* luego en *whatis_scicos* y en *Menu_entries*.

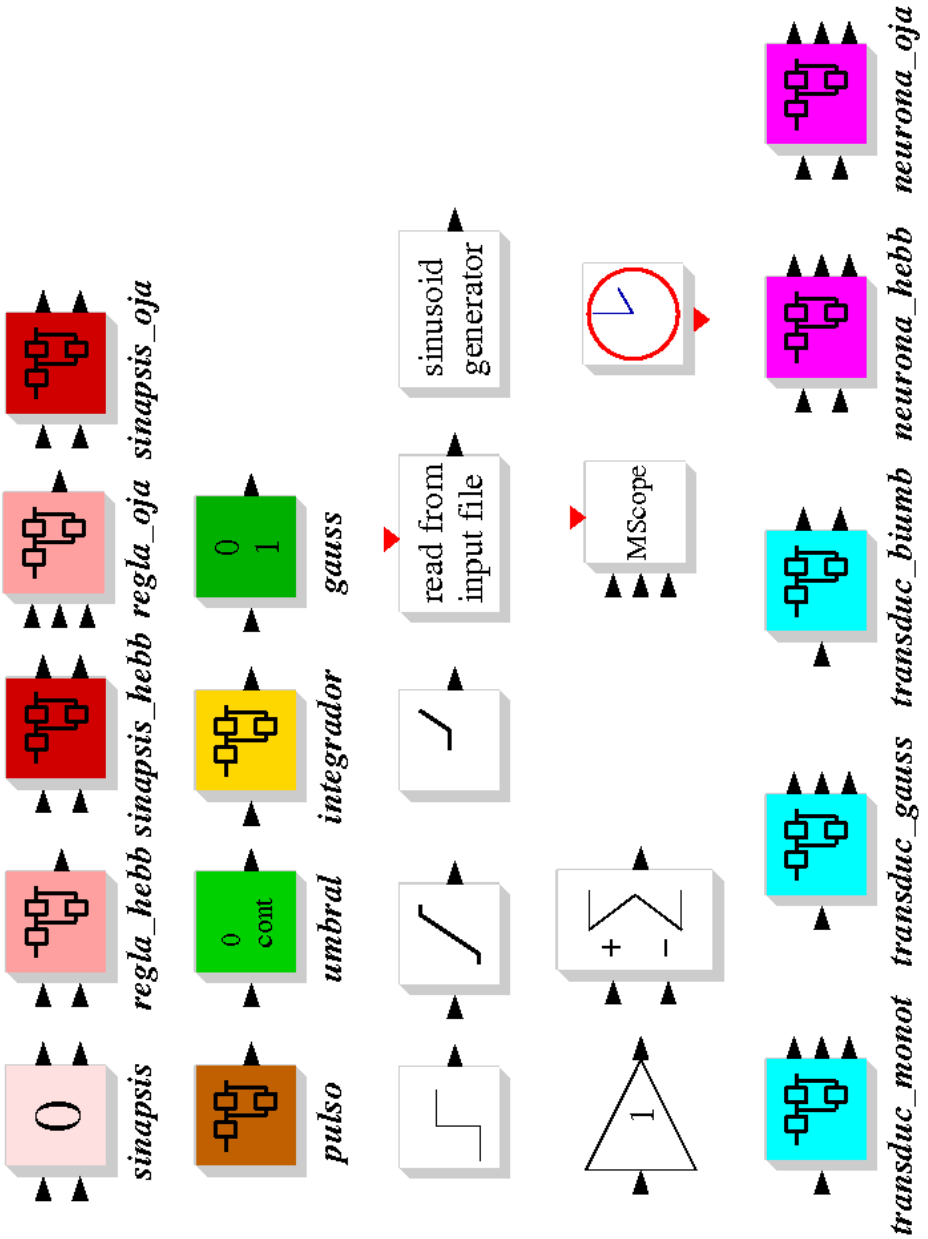

Figura 1: Paleta de bloques conexionistas.

2. Instalación y uso

2.1. Requisitos

El requisito principal es tener instalado Scilab/Scicos-4.1.2 [Sci07b, Sci07a], las instrucciones de instalación, muy simples, se describen en la página web anterior. Preferiblemente se descargará la versión binaria, no la fuente.

2.1.1. Requisitos para Linux

En Scicos se pueden escribir las funciones computacionales de los bloques en C, Fortran o en lenguaje interpretado de Scilab. Dado que en esta biblioteca se usan funciones computacionales escritas en C es necesario disponer del compilador de C, GCC [GNUa], y de la utilidad make `[GNUb]`.

Generalmente las distribuciones de Linux incluyen los paquetes `gcc` y `make`, además en muchas distribuciones suelen instalarse por defecto o eligiendo una instalación para desarrollador al instalarlas.

2.1.2. Requisitos para MSWindows

En Scicos se pueden escribir las funciones computacionales de los bloques en C, Fortran o en lenguaje interpretado de Scilab. Para compilar funciones escritas en C en este sistema operativo pueden usarse varias versiones del compilador Visual C++ [Micb], hay una versión libre [Micc] que necesita la instalación de Microsoft Platform SDK[2] [Mica], o el compilador de uso gratuito lcc-win32 [Nav][3]. Las instrucciones para el uso de este compilador en Scilab se describen en el fichero `Readme_LCC.txt` del directorio `lcc` de Scilab.

No es necesario este requisito si no se van a escribir funciones computacionales en lenguaje C.

2.2. Instalación

La biblioteca se descarga desde la carpeta "`Documentos públicos de Métodos neuronales bio-inspirados`" del foro general de la asignatura Métodos neuronales bio-inspirados en la web de posgrados UNED [UNE07]. La biblioteca se encuentra en los ficheros comprimidos `bioconexionismo.zip` (para Windows) o `bioconexionismo.tar.gz` (para Linux).

Para instalar esta biblioteca hay que descomprimir el archivo `bioconexionismo` (`.zip` o `.tar.gz`) , lo cual crea la carpeta `bioconexionismo` .

Para trabajar con la biblioteca se abre Scilab y cambiamos al directorio `bioconexionismo`, para lo cual se introduce en la consola de Scilab la orden

```
-->chdir('<ruta_a_bioconexionismo>');
```

o se configura el programa para que use `<ruta_a_bioconexionismo>` como directorio de trabajo.

La primera vez que se trabaje hay que compilar la funciones computacionales en C, para lo cual y se teclea en una sola línea (este paso no es necesario ejecutarlo en Windows).

```
-->ilib_for_link(['sinapsisblk','umbral','gauss'],'cblocks.o',[],'c',
'Makelib','loader.sce','cblocks')
```

Una vez compilados los bloques hay cargarlos en Scilab y cargar sus funciones de interfaz, tecleamos:

[2]Descarga conjunta superior a 1GB.
[3]Descarga de unos 20 MB

```
-->exec initbiocon.sce
```

La última orden hay que ejecutarla cada vez se abra Scilab y se vaya a utilizar la biblioteca[4].

Ya podemos abrir Scicos

```
-->scicos();
```

y a continuación cargar la paleta (Pallete▷Load as Palette) `bioconexionismo.cos`, para crear y editar simulaciones. Si lo que se desea es abrir los bloques para inspeccionarlos hay que copiar (arrastrar) los bloques a la ventana principal de Scicos o cargar la paleta con File▷Open. Podemos añadir la paleta a la lista de paletas de Scicos en el menú con Pallete▷Pal Editor y posteriormente cargarla con Pallete▷Palettes.

Nota Al ejecutar algunas simulaciones aparece un mensaje de advertencia sobre "No continuos-time state", no se debe tomar ninguna medida al respecto sino pulsar el botón Ok.

3. La paleta de bloques

En la paleta los bloques los podemos agrupar en varias clases que se describen a continuación.

3.1. Bloques generadores de señales

Entre los bloques generadores de señales se encuentran algunos propios de Scicos, que se hallan en la segunda y tercera filas de la figura 1, como el *escalón* (*step*), la *rampa* (*ramp*), un *lector de datos de fichero* (*read from input file*) y un *generador sinusoidal* (*sinusoid generator*). Además tenemos el bloque *pulso* que produce un pulso cuadrado de amplitud, duración y frecuencia asignables. El manejo de los bloques propios de Scicos se puede encontrar en la bibliografía y se pueden usar cualesquiera bloques de la paleta fuentes (sources) de Scicos para generar señales.

3.1.1. El bloque pulso

El diagrama se ve en la figura 2. Los valores máximo y mínimo se hallan en los bloques *constante* (*constant*), por defecto 1 y 0 respectivamente. El inicio y el fin del pulso se establecen con los tiempos iniciales de los relojes, izquierdo y derecho respectivamente, el periodo de ambos relojes debe ser el mismo y corresponde al periodo del pulso.

3.1.2. Simulación del movimiento de fuentes de señales

Para simular el movimiento de las fuentes de señales hay que multiplicar dicha señal por otra que nos va dando la distancia recorrida por la fuente (ecuación del movimiento), en este manual se ha tomado el criterio acotarla entre 0 (distancia "infinita" al receptor) y 1(distancia 0 al receptor). De esta manera para simular el acercamiento de la fuente a velocidad constante multiplicamos (modulamos) la señal del estímulo por la salida de un bloque rampa con pendiente (slope) positiva, si fuera negativa tendríamos un alejamiento, ver figura 3, el fichero `transductor-mov.cos` ofrece un ejemplo.

[4]Para más comodidad esta orden podría incluirse en el fichero `scilab.star` en el directorio de Scilab.

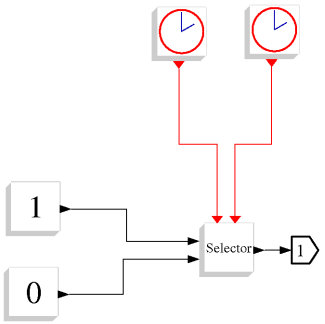

Figura 2: El bloque pulso

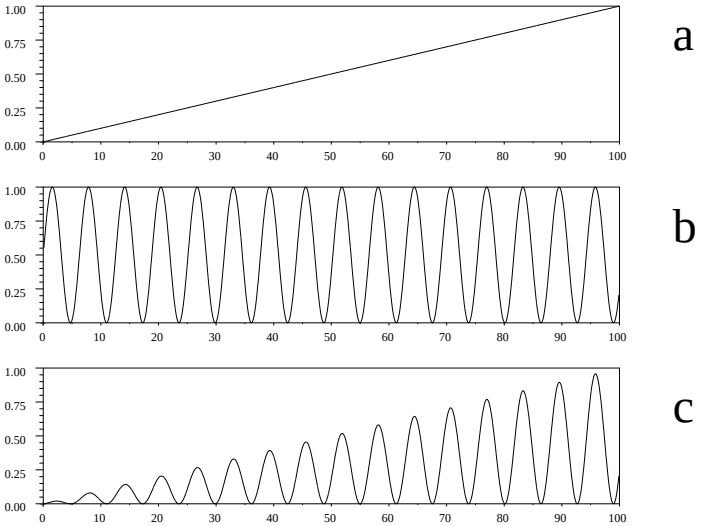

Figura 3: Simulación del movimiento de una fuente.
a) distancia recorrida, hacia el receptor, en función del tiempo, b) intensidad de la señal emitida por la fuente y c) intensidad de la señal medida en el receptor. Esta simulación se encuentra en el fichero `transductor-mov.cos`.

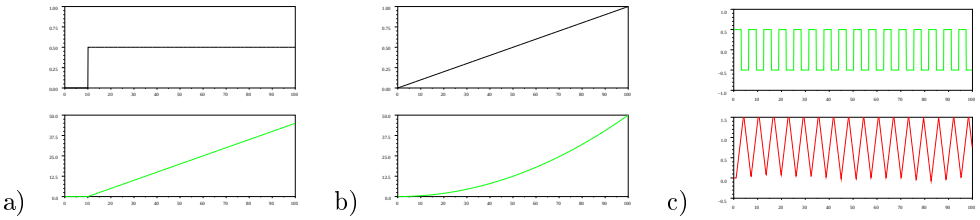

Figura 4: Bloque integrador
a) la integral de un escalón es una rampa, b) la de una rampa es una parábola y c) la de una onda cuadrada, centrada en 0, es una onda de dientes de sierra.

3.2. Bloque de umbral

Este bloque se denomina abreviadamente *umbral,* no está creado a partir de otros bloques sino mediante una función computacional en C y de interfaz, ver sección 2. El umbral puede ser continuo o discreto. La ecuación del umbral continuo es

$$umb(t) \quad = \quad \begin{cases} min & si\ t < \theta \\ t & si\ t \geq \theta \end{cases} \tag{1}$$

y la del umbral discreto es

$$umb(t) \quad = \quad \begin{cases} min & si\ t < \theta \\ max & si\ t \geq \theta \end{cases} \tag{2}$$

siendo θ el valor umbral, min el valor constante que se devuelve si no se sobrepasa el umbral (generalmente $min = 0$) y max el valor constante que se devuelve si se sobrepasa el umbral en el caso discreto (generalmente $max = 1$).

Para editar el bloque umbral se dispone de una GUI donde además de las constantes comentadas se elige el tipo, continuo o discreto, del bloque.

3.3. Bloques de transformaciones matemáticas

Bajo este epígrafe hemos asociado bloques que realizan diversas transformaciones matemáticas. Hay bloques propios de Scicos: *saturación*, *ganancia* y *sumatorio* y los bloques *integrador* y *gaussiana.*

3.3.1. El bloque integrador

El bloque *integrador* (*integrator*) es el mismo de Scicos al que, por razones técnicas de las simulaciones (cierre de lazo algebraico[5]), se le ha añadido un *retardo* (*continuous fix delay*). Su respuesta es la integral de la entrada, por defecto

$$R(t) \ = \ \int_{t_0}^{t} E(\tau) d\tau \ + C \tag{3}$$

donde C es la constante de integración, $C = 0$ y t_0 el instante de tiempo en el que comienza a actuar el integrador. Así, con $t_0 = 0$, si $E(t) = 0{,}01t$ se tendrá $R(t) = 0{,}01t^2/2$, como se muestra en la figura 4b.

[5]Este mensaje de error aparece cuando unimos directa o indirectamente la salida de un bloque con su entrada en cierto tipo de bloques. Si aparece habrá que introducir un retardo tras la salida, aunque esto puede alterar, incluso bastante, los resultados de la simulación.

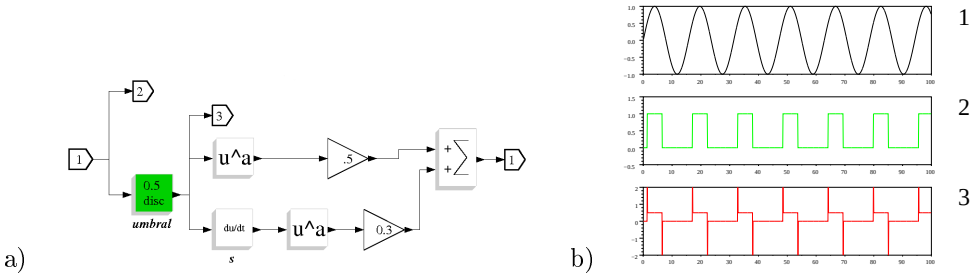

Figura 5: Transductor monótono discreto
a) circuito, b1) entrada, b2) entrada umbralizada con $\theta = 0{,}5$ y b3) salida con $k_1 = 0{,}5$, $n = 2$, $k_2 = 0{,}3$ y $m = 1$, esta simulación se encuentra en el fichero `transductor01.cos`. La salida 1 del circuito corresponde a la gráfica 3, la 2 a la 1 y la 3 a la 2.

3.3.2. El bloque gaussiana

El bloque *gaussiana* realiza la siguiente función (función de densidad de la distribución normal $N(\mu, \sigma)$, ver figura 6b):

$$N(t) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(t-\mu)^2}{2\sigma^2}} \tag{4}$$

La media μ (por defecto 0) y la desviación típica σ (por defecto 1) se editan en una GUI. Este bloque no está creado a partir de otros bloques sino mediante una función computacional en C y otra de interfaz, ver sección 2.

3.4. Bloques transductores

Los bloques transductores (sensores) han de recibir señales de un generador y suelen constar de un elemento de umbral, discreto o continuo, seguido de cierta transformación. Los transductores pueden disponer de varias salidas, para una exhaustiva monitorización. En el caso de tres salidas estas corresponden de arriba a abajo: la salida del transductor, la entrada y la entrada umbralizada. Se pueden dejar sin conectar las salidas que se deseen, salvo la primera.

3.4.1. Transductores de respuesta monótona

Una fórmula para la función de respuesta de transductores monótonos de gran generalidad es:

$$R(t) \quad = \quad k_1 (E^*(t))^n + k_2 \left(\frac{dE^*(t)}{dt} \right)^m \tag{5}$$

donde $E^*(t) = umb(E(t))$ es la entrada al transductor umbralizada.

El bloque *transduc_monot* puede funcionar en modo discreto o continuo dependiendo del tipo de umbral seleccionado. Describiremos el caso discreto, ver figura 5 y ecuación 5.

La edición de k_1 se realiza en el bloque *ganancia* superior y la k_2 en el inferior, la de n en el bloque *potencia* superior y la de m en el inferior. Los valores por defecto son tales que $R(t) = E^*(t)$.

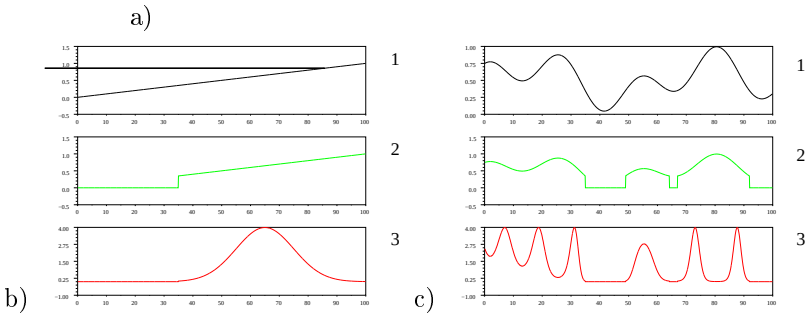

Figura 6: Transductor no-monótono gaussiano continuo
a) circuito, b1) y c1) entradas, b2) y c2) entradas umbralizadas con $\theta = 0{,}35$ y b3) y c3) salidas con $\mu = 0{,}65$ y $\sigma = 0{,}1$, estas simulaciones se encuentran en los ficheros `transductor03.cos` y `transductor04.cos`. La salida 1 del circuito corresponde a la gráfica 3, la 2 a la 1 y la 3 a la 2.

3.4.2. Transductores de respuesta no-monótona

En este caso no es posibles dar una fórmula para la función de respuesta de cierta generalidad.

Transductores no-monótonos gaussianos El caso más frecuente en la Naturaleza es el de respuestas con un máximo central, como p. ej. una gaussiana. En la figura 6 se representa el bloque *transduc_gauss*, que puede operar en modo continuo o discreto, dependiendo del tipo de umbral.

Transductores no-monótonos biumbral Las respuestas no-monótonas también se pueden obtener por inhibición lateral de neuronas con distinto valor umbral, ver [Bra86] y [MG07], de manera que la neurona de mayor umbral inhibe a la de menor umbral cuando la señal es fuerte. Si sólo consideramos la salida de la neurona de menor umbral, el circuito sólo se activará con valores medios de la señal. En la figura 7 se representa el bloque *transduc_biumb*, de umbral continuo, el lector puede crear la versión discreta o mezclarlas.

3.5. Sinapsis

Las sinapsis se crean a partir del bloque *sinapsis*, que no está creado a partir de otros bloques sino mediante una función computacional en C y otra de interfaz, ver sección 2. A este bloque se le conecta un determinado bloque de regla de aprendizaje para formar los distintos tipos de sinapsis. Para las sinapsis de peso fijo es mejor utilizar el bloque *ganancia*.

3.5.1. El bloque sinapsis

Este bloque es una ganancia (peso) variable, dispone de dos entradas y dos salidas, ver figura 8. La primera entrada porta la señal a la que hay que aplicar

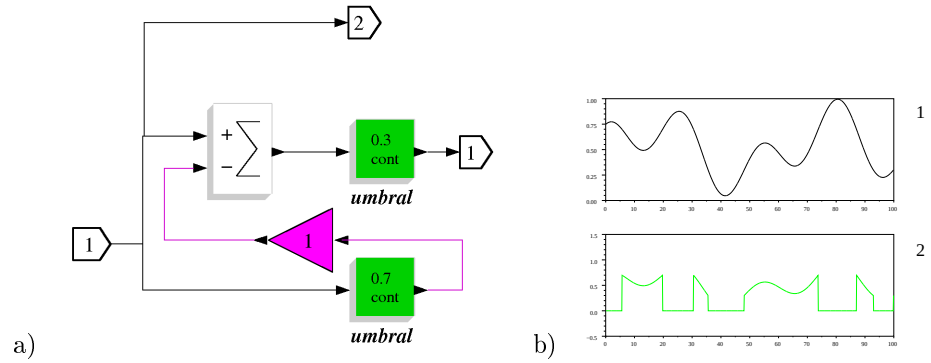

Figura 7: Transductor no-monótono bi-umbral
a) circuito, b1) entrada y b2) salida con $\theta_1 = 0{,}3$, $\theta_2 = 0{,}7$ y peso de la inhibición $= 1$, esta simulación se encuentra en el fichero `transductor05.cos`. La salida 1 del circuito corresponde a la gráfica 2 y la 2 a la 1.

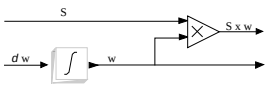

Figura 8: Esquema del bloque *sinapsis*
En este esquema se usan algún bloque ficticio, no bloques de esta biblioteca o de Scicos, el bloque realmente está construido de otra manera.

el peso y la segunda el diferencial del peso. La primera salida conduce la señal multiplicada por el peso y la segunda el peso tras la integración, la única salida que es imprescindible conectar es la primera. El peso incrementado queda guardado como nuevo valor de la ganancia variable. El valor inicial del peso por defecto es 0, aunque en muchos casos hay que darle un valor pequeño distinto de cero para posibilitar el inicio del aprendizaje (bootstrapping). El peso se edita en una GUI.

Este bloque ha sido diseñado de manera que impide la creación de lazos algebraicos en muchas situaciones. En la figura 8 puede verse que una de las entradas es dw que tras ser integrada produce w, o sea, con este bloque se pueden implementar sinapsis con reglas de aprendizaje continuas.

3.5.2. El bloque regla hebbiana

Este bloque se denomina abreviadamente *regla_hebb*, [Heb49]. Consta de dos entradas que se multiplican entre sí y luego por una ganancia (la constante de aprendizaje μ[6]), ver figura 9a. El producto final, el diferencial del peso, es dirigido hacia la única salida del bloque, por lo tanto dicho diferencial será

$$dw(t) = \mu x(t) y(t) \ , \tag{6}$$

siendo $x(t)$ la entrada a la sinapsis e $y(t)$ la salida de la neurona de llegada. El valor por defecto de la constante de aprendizaje es $\mu = 0{,}001$, que se edita en el bloque ganancia, el rango de valores debe ser μ es $0 < \mu < 1$ aunque normalmente es $\mu \ll 1$.

[6]Queda claro por el contexto donde aparezca cuándo μ se refiere a la constante de aprendizaje o a la media de la gaussiana.

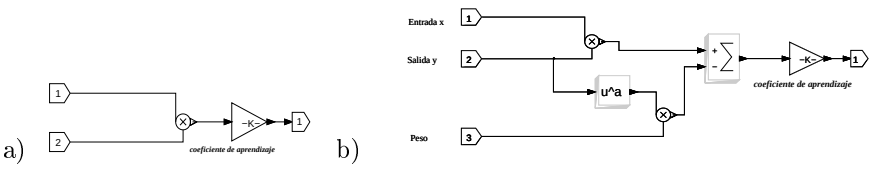

Figura 9: Circuito para la regla de aprendizaje hebbiana a nivel de sinapsis. a) regla original de Hebb y b) corrección de Oja.

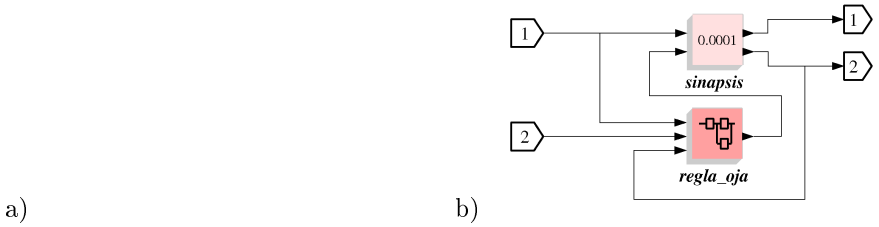

Figura 10: Circuito para la sinapsis con aprendizaje hebbiano. a) regla original de Hebb y b) corrección de Oja.

3.5.3. El bloque regla de Oja

Este bloque se denomina abreviadamente *regla_ oja*, [Oja82]. Su funcionamiento es similar al del bloque *regla_ hebb* aunque dispone de tres entradas, ver figura 9b. La primera entrada del bloque porta la propia entrada a la sinapsis $x(t)$, la segunda la salida de la neurona de llegada $y(t)$ y la tercera el peso $w(t)$. Ahora el diferencial del peso es

$$dw(t) = \mu \left(x(t)y(t) - y^2(t)w(t) \right) \quad . \tag{7}$$

3.5.4. El bloque sinapsis con regla hebbiana

Este bloque se denomina abreviadamente *sinapsis_ hebb*. Se construye conectando la salida de un bloque *regla_ hebb*, *dw*, a la segunda entrada de un bloque *sinapsis* básico. Consta de dos entradas, la primera corresponde a la primera entrada del bloque *sinapsis* y la segunda a la segunda del bloque *regla_ hebb*. Tiene las mismas dos salidas que el bloque *sinapsis*, ver figura 10a.

3.5.5. El bloque sinapsis con regla de Oja

Este bloque se denomina abreviadamente *sinapsis_ oja*. Se construye conectando la salida de un bloque *regla_ oja* a la segunda entrada de un bloque *sinapsis* básico y la segunda salida del bloque *sinapsis* a la tercera entrada del bloque *regla_ oja*. Consta de dos entradas, la primera corresponde a la primera entrada del bloque *sinapsis* y la segunda a la segunda del bloque *regla_ oja*. Tiene las mismas dos salidas que el bloque *sinapsis*, ver figura 10b.

3.6. Neuronas

Se han implementado dos clases de neuronas con aprendizaje hebbiano una con la regla original y la otra con la de Oja. La neuronas disponen de N (en los ejemplos $N = 2$) entradas conectadas a N sinapsis y $N + 1$ salidas, la primera

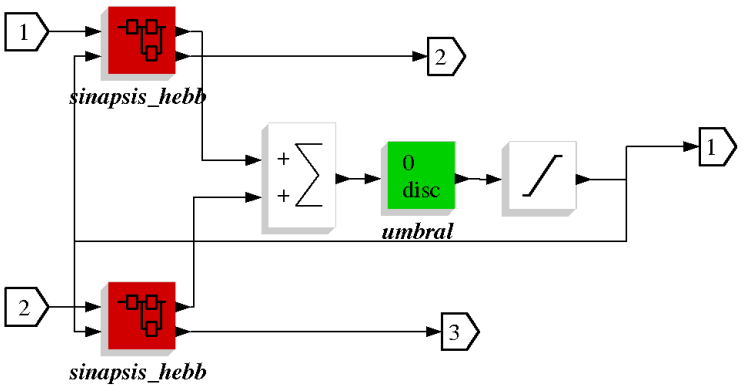

Figura 11: Neurona hebbiana de umbral discreto con dos sinapsis.
Si sustituimos los bloque *sinapsis_hebb* por bloques *sinapsis_oja* tendremos una neurona de Oja.

salida corresponde a la salida de la neurona $y(t)$ y el resto a los pesos $w_i(t)$ de las N sinapsis, la única salida que es imprescindible conectar es la primera. La neuronas se construyen conectando bloques *sinapsis* a un bloque *sumador*, la suma pasa por un bloque de umbral (discreto o continuo) a continuación por un bloque de *saturación*[7] que nos da la salida de la neurona. Para los aprendizajes de tipo hebbiano hay que realimentar con función de dicha salida las reglas de aprendizaje de las sinapsis.

3.6.1. El bloque neurona hebbiana

El bloque se denomina abreviadamente *neurona_hebb* y puede operar en modo continuo o discreto, según disponga de umbral continuo o discreto. En la implementación disponible consta de dos sinapsis hebbianas de entrada, ver figura 11 y comienzo de esta sección 3.6.

3.6.2. El bloque neurona de Oja

El bloquese denomina abreviadamente *neurona_oja* y puede operar en modo continuo o discreto, según disponga de umbral continuo o discreto. Es idéntico al bloque *neurona_hebb* salvo por el tipo de sinapsis de entrada. En la implementación disponible consta de dos sinapsis de Oja de entrada, ver figura 11 y comienzo de esta sección 3.6.

4. Ejemplos

En los pies de algunas de las figuras del texto se han referenciado archivos de simulaciones muy simples que el lector puede estudiar. A continuación vamos a describir dos simulaciones realizadas con bloques descritos anteriormente.

4.1. Primer experimento

4.1.1. Descripción

El objetivo de la primer experimento será comparar el comportamiento de los pesos al usar la regla original de Hebb con el que produce la corrección de Oja, ya

[7] Se pueden emplear cualquier tipo de función limitadora como la sigmoide.

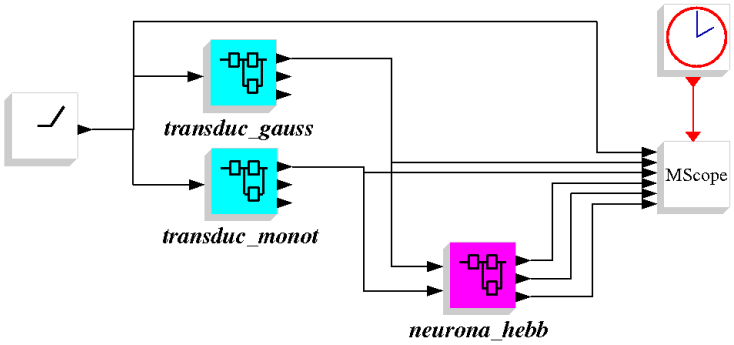

Figura 12: Neurona sensorial hebbiana

La neurona representada en la figura 11, ahora con umbral continuo, es conectada a dos receptores formando una neurona sensorial. Este diagrama se encuentra en el fichero `hebbiano2.cos`.

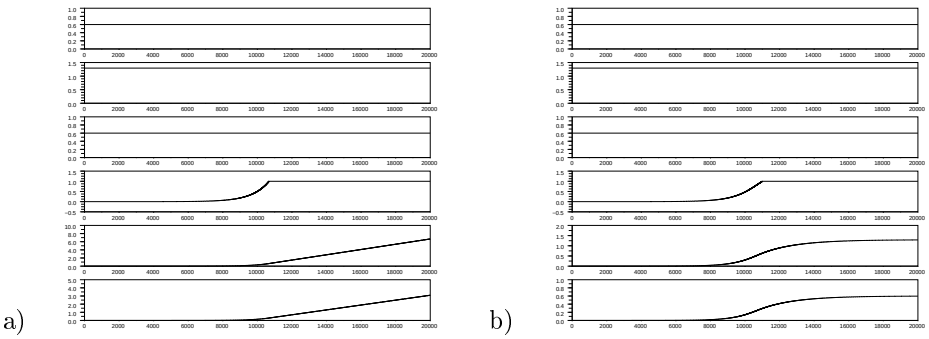

Figura 13: Aprendizaje hebbiano .

Se muestran dos sesiones de entrenamiento de la neurona de la figura 12: a) con regla original de Hebb y b) con corrección de Oja. Seguir en el texto 4.1.2 el comentario.

que los pesos en la regla original divergen y la solución dada por Oja hace que estos converjan. Para ello se va a construir un circuito de una neurona sensorial, ver figura 12, con dos receptores distintos (bloques *transduc_gauss* y *transduc_monot* en modo continuo ambos), que será alimentada con una señal constante de intensidad superior a los umbrales de los transductores. El umbral de la neurona es despreciable para evitar problemas de inicio del aprendizaje (bootstrapping). Cada receptor está conectado a una sinapsis de Hebb o de Oja, según el caso. Por último la salida de la neurona y los pesos de las sinapsis se monitorizarán en un bloque *osciloscopio* (*Mscope*). El experimento se compone de dos simulaciones cuyos ficheros son `hebbiano2.cos` y `hebbiano2-oja.cos`, respectivamente. En ambas simulaciones el periodo de reloj fue de 0.1 uts (unidades tiempo simulado) y el tiempo final de integración (tiempo total de la simulación) 20000 uts, o sea, el bucle de cada simulación se repitió 200000 veces. Para coeficiente de aprendizaje de la neurona (ver 3.5.2) se eligió un valor pequeño $\mu = 0{,}00005$ y un peso inicial $w_0 = 0{,}00001$.

4.1.2. Resultados

La figura 13 nos muestra los resultados del experimento. Las gráficas corresponden de arriba a abajo a:

1. Intensidad de la señal emitida por la fuente, un bloque rampa con pendiente

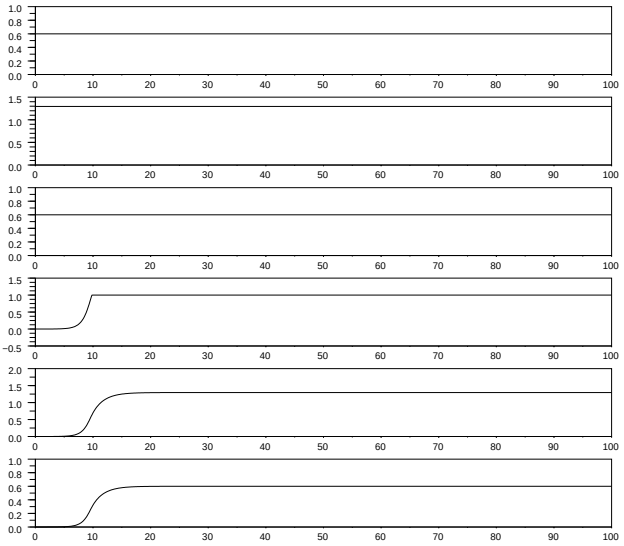

Figura 14: Aumento del coeficiente de aprendizaje
Aprendizaje hebbiano de la neurona de la figura 12, con corrección de Oja, con coeficiente de aprendizaje aumentado. Comparar con figura 13b y seguir en el texto 4.2.2 el comentario.

nula y ordenada (initial output) igual a 0.6.

2. Respuesta del transductor gaussiano continuo con $\theta = 0{,}35$, $\mu = 0{,}45$ y $\sigma = 0{,}1$.
3. Respuesta al transductor monótono continuo con $\theta = 0{,}35$, $k_1 = 0{,}5$, $n = 2$, $k_2 = 0{,}3$ y $m = 1$.
4. Respuesta de la neurona.
5. Peso de la primera sinapsis.
6. Peso de la segunda sinapsis.

La subfiguras a) y b) de la figura 13 tienen las tres primeras gráficas iguales, o casi, y su comentario sigue a continuación. La intensidad constante de la señal hace que las respuestas de los transductores sean también constantes, lo que simplifica la interpretación de los resultados. La respuesta de la neurona es prácticamente nula hasta poco antes de que el aumento de los pesos se torne apreciable y va aumentando con estos hasta llegar al nivel de saturación.

Sin embrago las gráficas correspondientes a los pesos difieren. En a) (regla hebbiana original) los pesos muestran un crecimiento sin aparente punto de inflexión, lo que sugiere su divergencia hacia infinito. En b) el crecimiento de los pesos se estabiliza tomando estos valores aproximados de 1.3 y 0.6, al cabo de aproximadamente 15000 uts.

Queda pues comprobado que la corrección de Oja estabiliza el crecimiento de los pesos, lo que no es conseguido por la formulación original de Hebb.

4.2. Segundo experimento

4.2.1. Descripción

En la misma línea del primer experimento vamos ahora a comprobar los efectos que sobre el aprendizaje, empleando la regla de Oja, tendrá un aumento significa-

tivo de la constante de aprendizaje μ, tomaremos $\mu = 0{,}005$ un valor 100 veces mayor que en el primer experimento y $w_0 = 0{,}0001$. Un efecto previsible de tal aumento será un incremento de la velocidad del aprendizaje, de hecho para obtener resultados similares a los de experimento anterior solamente han sido necesarias 100 uts de tiempo de simulación. Para realizar esta simulación cambiar en el fichero `hebbiano2-oja.cos` los dos valores comentados y también el periodo de refresco del osciloscopio para hacerlo igual al tiempo de simulación. También se puede editar el contexto del diagrama.

4.2.2. Resultados

La variables representadas en las gráficas de la figura 14 son las mismas que las descritas en 4.1.2. Los pesos alcanzan aproximadamente los mismos valores que en experimento anterior, pero ahora lo hacen en aproximadamente 20 uts, un tiempo considerablemente menor que en dicho experimento.

El incremento, dentro del rango experimentado, de la constante de aprendizaje no modifica el valor límite de los pesos pero acorta el tiempo de aprendizaje.

Índice de figuras

Referencias

[Bra86] Valentino Braitenberg. *Vehicles: experiments in synthetic psychology.* MIT, 1986.

[Del98] Bernardo A. Delicado. Introducción al modelado y simulación de sistemas físicos con la toolbox Scicos de Scilab, 1998. `http://pauillac.inria.fr/cdrom/ftp/scilab/contrib/delicado/scicos.zip`.

[Esc05] Héctor Manuel Mora Escobar. Introducción a Scilab, 2005. `http://www.matematicas.unal.edu.co/~hmora/sci.pdf`.

[GNUa] GNU. GNU C, C++, Java Linux compiler. `http://gcc.gnu.org`.

[GNUb] GNU. GNU Make. `http://www.gnu.org/software/make/`.

[Heb49] Donald O. Hebb. *The Organization of Behavior: A Neuropsychological Theory.* Wiley, 1949.

[MG07] José Mira Mira and Ana E. Delgado García. Redes neuronales de inhibición lateral: Formulaciones analítica, simbólica e inferencial. 2007. `http://posgrados.informatica.uned.es/`.

[Mica] Microsoft. Microsoft platform SDK. `http://msdn2.microsoft.com/en-us/express/aa700755.aspx`.

[Micb] Microsoft. Visual C++. `http://msdn2.microsoft.com/es-es/visualc/default.aspx`.

[Micc] Microsoft. Visual Studio C++ 2005 Express. `http://msdn2.microsoft.com/en-us/express/aa975050.aspx`.

[Nav] Jacob Navia. LCC compiler. `http://www.cs.virginia.edu/~lcc-win32/`.

[Oja82] E. Oja. A simplified model as a principal component analyzer. *J. Math. Biol.*, 15(3):267–273, 1982.

[Sci] Scilab. Manuales y tutoriales de Scilab/Scicos. `http://www.scilab.org/contrib/index_contrib.php?page=download\&category=MANUALS`.

[Sci07a] Scicos team. Scicos, 2007. `http://www.scicos.org`.

[Sci07b] Scilab group. Scilab, 2007. `http://www.scilab.org`.

[UNE07] UNED. Uned posgrados, 2007. `http://posgrados.informatica.uned.es/`.