Nombre:	Alfredo Weitzenfeld
Afiliación:	Instituto Tecnológico Autónomo de México Departamento Académico de Computación Río Hondo #1, San Angel, México D.F.
Puesto:	Profesor de Tiempo Completo
Máximo grado:	Doctorado en Ciencias de la Computación University of Southern California, Los Angeles, CA, USA
Teléfono:	56284000 ext 3614
Fax:	56162211
Email:	alfredo@lamport.rhon.itam.mx
Web Site:	http://cannes.rhon.itam.mx
Áreas de Interés:	Redes Neuronales, Agentes Autónomos, Sistemas de Simulación

NSL
Lenguaje de Simulación de Redes Neuronales
Un Sistema para el Modelado Biológico y Artificial[1]

Alfredo Weitzenfeld

NSL (*Neural Simulation Language*) es un sistema de simulación para el desarrollo de redes neuronales biológicas y artificiales. NSL permite la simulación de modelos con diferentes niveles de detalle neuronal, desde los más sencillos que constan de pocas neuronas hasta los más complejos compuestos de millones de neuronas. NSL incorpora un rico ambiente de modelado y simulación que integra un lenguaje compilado y otro interpretado por razones de eficiencia e interacción, además de bibliotecas numéricas, interfaces gráficas y herramientas de visualización. El sistema es orientado a objetos, ofreciendo un ambiente de simulación para usuarios con poca experiencia de programación, al igual que para aquellos más experimentados. NSL fue desarrollado originalmente hace 10 años habiendo sido utilizado durante ésta última década para la investigación y docencia, dando lugar al desarrollo de numerosos modelos neuronales en diversos laboratorios en el mundo. Actualmente se cuenta con NSL3.0, la tercera generación del sistema bajo diferentes plataformas de procesamiento y en base a programación en C++ y Java.

1 Introducción

Los sistemas biológicos son altamente complejos y su comprensión requiere de sofisticadas herramientas de desarrollo que apoyen múltiples niveles de modelado y simulación. Para lograr esto, es necesario contar con lenguajes y ambientes de software que por un lado sean lo suficientemente eficientes dada la intensidad de cómputo de la gran mayoría de los modelos neuronales existentes, y por otro lado sean lo suficientemente sofisticados para apoyar el desarrollo de sistemas altamente complejos. En particular, las redes neuronales pueden modelarse y simularse en base a tres niveles de abstracción: (1) módulos neuronales, (2) redes neuronales, (3) y neuronas:

1. En el nivel más alto, las redes neuronales son organizadas en base a estructura y comportamiento en módulos inspirados en el concepto de *esquemas* [4]. En el caso de redes biológicas, estos esquemas son generados según estudios neuroetológicos de animales vivos. En el caso de redes artificiales, la organización en módulos sigue patrones de modularización y distribución similar al desarrollo de aplicaciones en la ingeniería de software.
2. En el nivel de redes neuronales, se modelan arquitecturas de decenas hasta millones de neuronas que implementen el comportamiento correspondiente al nivel de módulos. En

[1] Este trabajo está apoyado por el NSF en EEUU (proyecto #IRI-9505864) y CONACyT en México (proyecto #546500-5-C018-A), además del proyecto REDII de CONACyT (#DAJ J002-222-98) y la Asociación Mexicana de Cultura, A.C.

este nivel, el modelo de neuronas utilizado es extremadamente sencillo como el modelo de neurona *McCulloch-Pitts* [12] y el modelo de *integrador con fugas* [3] donde la tasa de disparo de una neurona es una función no-lineal correspondiente al potencial de membrana (cada neurona se modela como un sólo *compartimiento*). En el caso de redes biológicas, se analizan los datos neuroanatómicos y neurofisiológicos para explicar los mecanismos neuronales básicos correspondiente a los modelos esquemáticos desarrollados en el nivel de comportamiento. En el caso de redes artificiales, los modelos puede haber sido inspirados por modelos biológicos o haber sido desarrollados en base a modelos matemáticos no lineales totalmente ajenos a la biología.

3. En el nivel de neuronas detalladas, los modelos de neuronas incluyen múltiples compartimentos [15] que incorporan membranas activas y una variedad de canales, como el modelo *Hodgkin-Huxley* [9]. Este nivel es principalmente utilizado para el modelado biológico con el objetivo primordial de comprender los diferentes mecanismos electroquímicos de bajo nivel, tales como la inhibición presináptica responsables por ejemplo del aprendizaje y la memoria en los animales y el ser humano.

Aunque existen muchas herramientas para facilitar la tarea del modelado y simulación de sistemas biológicos y artificiales, estos sistemas varían en su costo, complejidad y tipo de modelado apoyado [6] [13]. El principal desafío de estos sistemas es por un lado ofrecer ambientes de simulación generales y amigables, mientras que por otro lado ser eficientes y suficientemente expresivos para modelar los sistemas neuronales más complejos. Algunos de estos sistemas apoyan arquitecturas y algoritmos neuronales predefinidos mientras que otros permitan el desarrollo de nuevas arquitecturas y algoritmos, tal como NSL [24].

2 NSL

El *Lenguaje de Simulación Neuronal* (*NSL* por sus siglas en inglés) provee un ambiente de modelado y simulación para redes neuronales de gran tamaño, basado en un modelo neuronal sencillo. NSL tuvo sus orígenes en 1989 en la Universidad del Sur de California apoyando modelado neuronal de un sólo nivel y bajo un número limitado de plataformas. La versión actual NSL3.0 [26] es un esfuerzo conjunto entre la Universidad del Sur de California (USC) y el Instituto Tecnológico Autónomo de México (ITAM). El sistema sigue un diseño orientado a objetos implementado bajo dos ambientes de desarrollo, C++ y Java, ejecutando en una gran variedad de plataformas. Además, ofrece un ambiente de desarrollo que apoya usuarios principiantes y sofisticados a la vez. Una de las grandes ventajas de NSL sobre es otros simuladores es la posibilidad de desarrollar nuevas arquitecturas de redes neuronales con diferentes niveles de abstracción. Como aspecto esencial y particular de NSL es el apoyo para el desarrollo de módulos de redes neuronales que pueden ser interconectadas de manera jerárquica, donde las neuronas se modelan de manera sencilla y las interconexiones están sujetas a variadas reglas de aprendizaje. Para modelar redes neuronales es necesario describir: (1) los módulos neuronales que definen el modelo completo (2) las neuronas que componen la red neuronal para cada módulo, (3) las interconexiones de la red neuronal, (4) y la dinámica de la red neuronal. Algunos de los modelos que han sido desarrollados en NSL incluyen redes artificiales clásicas como *Hopfield* [10] y *Backpropagation* [16][27], redes biológicas como la *Retina* [17] y el

modelo *Córtico-Subcóritco* para la generación de movimientos sacádicos [8], y aplicaciones como reconocimiento de caras [28] y percepción de profundidad [11]. En este artículo revisaremos un modelo extremadamente sencillo conocido como el *Selector Máximo* o "Winner Take All" [2].

2.1 Módulos Neuronales

Los módulos son las estructuras básicas del modelo computacional de NSL. Estas estructuras son jerárquicas y pueden implementarse de manera general mediante redes neuronales u otros procesos, como se muestra en la Figura 1.

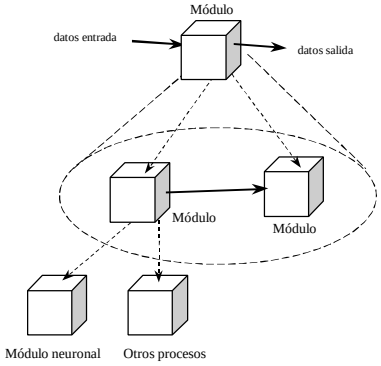

Figura 1. Modelo computacional jerárquico de NSL basado en módulos a ser implementados por redes neuronales u otros procesos.

Los módulos neuronales, de especial interés en NSL, encapsulan a las redes neuronales en base a la neurona como elemento básico de composición, como se muestra en la Figura 2.

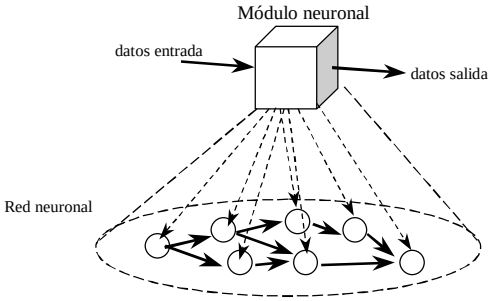

Figura 2. Módulo neuronal en NSL implementado mediante redes neuronales compuesto por múltiples neuronas.

La descripción de un módulo en NSL es similar a la descripción de una *clase* en los lenguajes orientados a objetos en el sentido de que se describe un *patrón* del cual los propios objetos son luego instanciados. Más allá de esto, los módulos en NSL pueden ser procesados de manera concurrente y utilizan comunicación en base a puertos de entrada y salida e interconexiones entre ellos.

2.2 Neuronas

El modelo básico de neurona en NSL es de un sólo compartimento, teniendo una salida y varias entradas, como se muestra en la Figura 3. El estado interno de la neurona es descrito

por un único valor, su potencial de membrana *mp*, el cual depende de las entradas de la neurona y de su historia previa. La salida es descrita por otro valor, su *tasa de disparo mf*, y puede servir como entrada a varias otras neuronas, incluyendo a sí misma. Al variar los valores de las entradas a una neurona, varía también su potencial de membrana y su tasa de disparo.

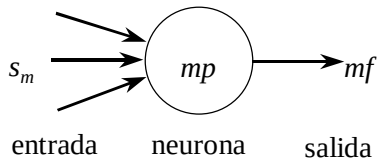

Figura 3. Modelo neuronal de un solo compartimento es representado por un valor *mp* correspondiente al potencial de membrana, y un valor *mf* correspondiente a su tasa de disparo, la única salida de la neurona. s_m representa el conjunto de entradas a la neurona.

El potencial de membrana *mp* es descrito por la ecuación diferencial

$$t_m \frac{dmp}{dt} = f(s_m, mp)$$

la cual depende de las entradas s_m, el valor previo de *mp* y la constante de tiempo t_m. La elección de *f* depende del modelo neuronal particular. Por ejemplo, el *integrador con fugas* [3] es descrito por

$$f(s_m, mp) = -mp + s_m$$

La *tasa de disparo mf* (salida de la neurona) se obtiene aplicando una *función umbral* al potencial de la membrana de la neurona,

$$mf = s\,(mp)$$

donde *s* es usualmente una función no lineal. Algunas de las funciones de *umbral* más comunes son *rampa*, *escalón*, *saturación* y *sigmoidal*.

En NSL dos estructuras de datos son requeridas para representar una neurona en el modelo de integrador con fugas. Una estructura de datos representa el potencial de membrana *mp* y la otra la tasa de disparo *mf*. Para estas estructuras se puede utilizar diferentes tipos *primitivos*, como **Float**, **Double** o **Int**. Por ejemplo, utilizando tipos **Double**, las estructuras serían:

```
NslDouble mp;
NslDouble mf;
```

La dinámica del potencial de membrana *mp* se representa mediante la ecuación diferencial

```
nslDiff(mp,τm,f(sm,mp));
```

donde **nslDiff** corresponde a una ecuación diferencial de primer orden con constante de integración t_m y $f(s_m,m)$ corresponde a la función de entrada igual a "$-m+ s_m$" en el modelo de integrador con fugas

```
nslDiff(mp,τm,-mp + sm);
```

La tasa de disparo *mf* se describe como

```
mf = σ(mp);
```

donde *s* representa la función de disparo o salida correspondiente a una función umbral.

2.3 Interconexiones

Cuando se construyen redes neuronales, la salida de una neurona sirve como entrada a otras neuronas. Las conexiones entre neuronas llevan un peso de conexión que describe cómo las neuronas son afectadas entre sí. Las conexiones son llamadas excitatorias o inhibitorias dependiendo de si su peso es positivo o negativo, respectivamente. La fórmula más común para la entrada a una neurona es

$$s_v = \sum_{i=0}^{n-1} w_i uf_i$$

donde uf_i es la tasa de disparo de la neurona up_i cuya salida está conectada a la entrada *i* de la neurona *vp*, y w_i es el peso de la conexión, como se muestra en la Figura 5 (*up* y *vp* son análogas a *mp*, mientras que *uf* y *vf* son análogas a *mf*).

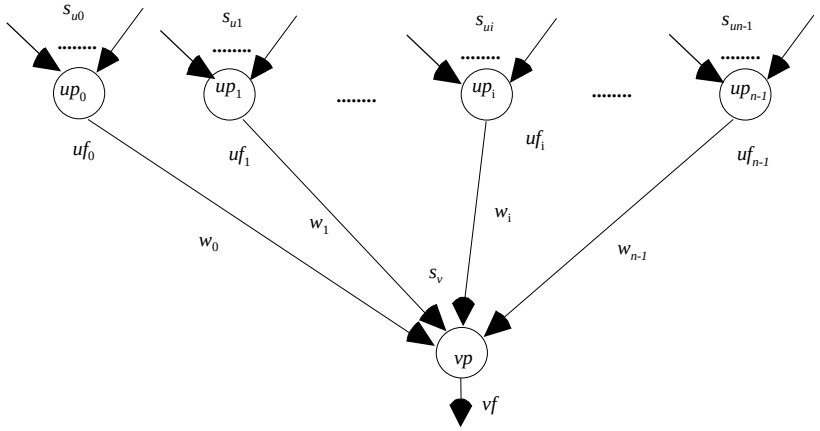

Figure 5. La neurona *vp* recibe entradas de las neuronas $up_1,...,up_n$, correspondientes a sus tasas de disparo, $uf_1,...,uf_n$, multiplicadas por los pesos $w_1,...,w_n$, respectivamente.

La descripción de las conexiones de entrada a la neurona *vp* en NSL es dada por una ecuación de la siguiente forma

```
sv = w0uf0 + w1uf1 + ...
```

2.4 Capas y Máscaras

Cuando se modela un gran número de neuronas se vuelve extremadamente tedioso nombrar a cada una de ellas por separado. Además, en el cerebro usualmente encontramos redes neuronales estructuradas en capas de neuronas homogéneas de dos dimensiones, con patrones de interconexión regulares entre las capas. Por tales motivos, NSL extiende la abstracción de la neurona básica a capas de neuronas y las conexiones sencillas a máscaras de conexión, respectivamente.

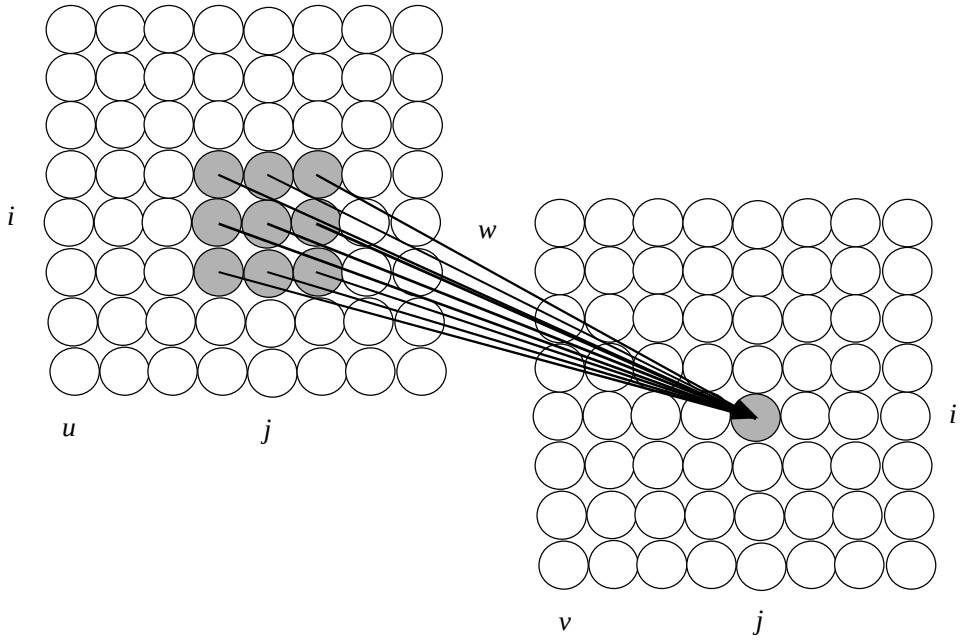

Figura 6. *w* representa la conexión o máscara de *convolución* entre las capas *u* y *v* correspondientes a la ecuación *sv=w@uf*. En este ejemplo *w* es una máscara de 3x3 que se superpone sobre una ventana en *u* del mismo tamaño, para obtener un valor de un elemento en *v* (en este ejemplo las capas *u* y *v* consisten de 8x8 neuronas cada una).

Una interconexión entre neuronas puede ser procesada computando una transformación al estilo de una *convolución espacial*[2] entre una máscara y una capa. Por ejemplo, como se muestra en la Figura 6, si *uf* representa un arreglo de salidas de una capa de neuronas *u*, y *sv* representa el arreglo de entradas a un elemento en la capa *v*, entonces, para una máscara $w(k,l)$ (para $-d \le k,l \le d$) representando el peso entre los elementos de $uf(i+k,j+l)$ (para $-d \le k,l \le d$) y los elementos $sv(i,j)$ para cada *i* y *j*, tenemos

$$sv(i,j) = \sum_{k=-d}^{d} \sum_{l=-d}^{d} w(k,l) uf(i+k, j+l)$$

En lugar de describir esta ecuación mediante múltiples expresiones individuales, NSL utiliza el operador "@" para lograr una única expresión de manera muy concisa,

```
sv = w@uf
```

El tamaño o dimensión de *w* y *uf* son totalmente arbitrarios sin afectar la expresión, esto es posible gracias a la regularidad de la conexión.

Para representar capas neuronales en NSL, **NslDouble** es extendida por tipos con dimensiones entre 0 y 4, por ejemplo, **NslDouble2** corresponde a un arreglo de **2** dimensiones de elementos tipo **Double**. Para simplificar las operaciones, las máscaras de interconexión son también definidas en NSL con estructuras similares cuya actividad describe pesos de conexión en lugar de potenciales de membrana o tasas de disparo. Por

[2] El término "convolución espacial" se utiliza exclusivamente por razones matemáticas.

ejemplo, las capas de las neuronas mostradas en la Figura 5 serían descritas por cuatro estructuras,

```
NslDouble1 up(n);
NslDouble1 uf(n);
NslDouble0 vp;
NslDouble0 vf;
```

Las primeras dos estructuras definen capas neuronales de una dimensión, donde "n" define el tamaño de la capa.

En este ejemplo se utiliza el operador "@" igual como antes,

```
   sv = w@uf
```

aunque en este ejemplo la máscara de pesos *w* es de un sólo elemento.

2.4 Simulación

Una vez descrito el modelo se procede a su simulación. La simulación consiste principalmente de: (1) asignación de valores a parámetros definidos en el modelo; (2) asignación de entradas a la red neuronal; (3) especificación de control de la simulación, como pasos de integración y tiempos a simular; y (4) especificación de aspectos de visualización para el modelo [23]. Estos aspectos serán mostrados directamente mediante un ejemplo en la siguiente sección.

3 Ejemplo

Por ejemplo, la arquitectura de la red neuronal correspondiente al modelo del *Selector Máximo* se basa en el modelo original de *Didday* [7], como se muestra en la Figura 6.

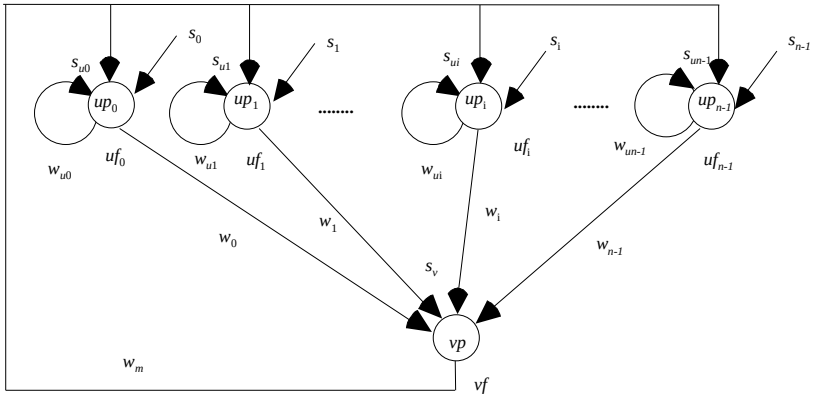

Figura 6. La red neuronal en la figura corresponde a la arquitectura del modelo *Selector Máximo* [7], donde s_i representa las entradas a la red, up_i y *vp* representan potenciales de membrana mientras que uf_i y *vf* representan tasas de disparo. w_m, w_{ui}, y w_i corresponden a los pesos de las conexiones.

Este es un modelo extremadamente sencillo, aunque muy ilustrativo, el cual recibe al menos dos entradas de diferente magnitud, y genera como salida un sólo valor correspondiente al máximo de las distintas entradas.

La descripción completa del modelo *Máximo Selector* está dado por las siguientes ecuaciones

$$t_u \frac{dup_i}{dt} = -up_i + w_u uf_i - w_m vf - h_u + s_i \qquad 1 \le i \le n$$

$$t_v \frac{dvp}{dt} = -vp + w_n \sum_{i=1}^{n} uf_i - h_v$$

En el modelo, w_m, w_{ui} y w_i representan los pesos de las conexiones teniendo que $w_{ui} = w_{u0} = ... = w_{un} = w_u$ y $w_i = w_0 = ... = w_n$. Se tiene adicionalmente los parámetros *hu* y *hv* con la restricción de que $0 \le hu$, y $0 \le hv < 1$.

Las funciones umbral son descritas por un *escalón* para uf_i

$$uf_i = \begin{cases} 1 & up_i > 0 \\ 0 & up_i \le 0 \end{cases}$$

y una *rampa* para *vf*

$$vf = \begin{cases} vp & vp > 0 \\ 0 & vp \le 0 \end{cases}$$

3.1 Modelado del Selector Máximo

La descripción del modelo completo incluye un módulo **MaxSelector** un módulo principal **MaxSelectorModel**. En **MaxSelector** se describe la dinámica de la red neuronal.

```
nslModule MaxSelector (int n) extends NslModule ()
{
   private: NslDouble1 s(n);
   private: NslDouble1 up(n);
   private: NslDouble1 uf(n);
   private: NslDouble0 vp();
   private: NslDouble0 vf();
   private NslDouble0 tu();
   private NslDouble0 tv();
   private NslDouble0 wu();
   private NslDouble0 wm();
   private NslDouble0 wn();
   private NslDouble0 hu();
   private NslDouble0 hv();

   public void simRun(){
         nslDiff(up,tu, -up + wu@uf - wm@vf - hu + s);
         uf = nslStep(up);
         nslDiff(vp,tv, -vp + nslSum(wn@uf) - hv);
         vf = nslRamp(vp);
   }
}
```

Nótese la estructura de un módulo en una patrón similar a la definición de una clase en los lenguajes orientados a objetos. La especificación de la estructura del módulo incluye la declaración de todas las variables en el módulo. El método **simRun** contiene las expresiones que son procesadas repetidas veces por el simulador según los parámetros de control de la simulación que asigne el usuario. Estas expresiones incluyen en este ejemplo operaciones

sobre vectores y escalares. Por ejemplo, **nslSum**(w_n@*uf*) multiplica el peso de la conexión w_n por la tasa de disparo *uf*, devolviendo finalmente la sumatoria del resultado de la convolución. Esto es necesario ya que la capa *vp* consiste de una sola neurona. **nslStep** y **nslRamp** son las funciones umbral para las dos capas de neuronas en la red, respectivamente.

El módulo es inicializado mediante la definición de un *modelo* de la siguiente forma,

```
nslModel MaxSelectorModel
{
   MaxSelector ms(10);
}
```

Parte de la inicialización incluye asignar el número de neuronas que van a ser instanciadas en la capa *up*, en nuestro ejemplo 10.

3.2 Simulación del Selector Máximo

La simulación de una red neuronal se controla interactivamente o mediante archivos. Esto incluye la (1) asignación de parámetros, (2) asignación de entradas, (3) especificación de control de la simulación y (4) visualización.

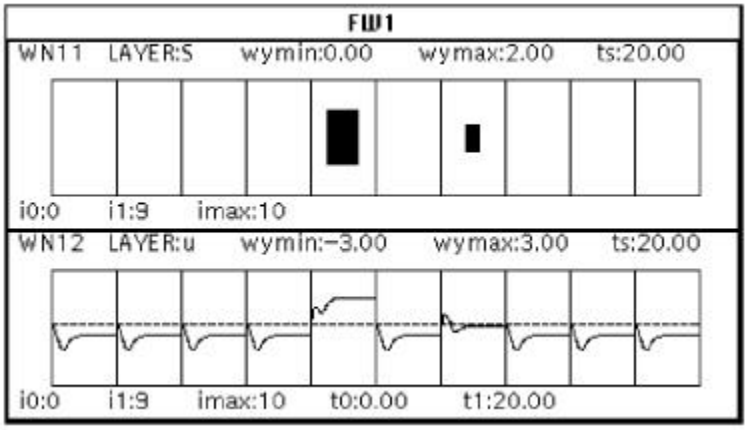

Figura 7. La salida del modelo *Selector Máximo*, muestra la capa de entrada *s*, en un gráfico espacial, y el potencial de membrana de la capa *up*, en un gráfico temporal.

Se asignan los parámetros cuyos valores no fueron especificados originalmente en el modelo. Esto se hace para permitir al usuario modificar los valores interactivamente. Aunque omitiremos esto, cabe resaltar que NSL utiliza un lenguaje de "scripts" basado en Tcl/Tk [14]. Se asignan las entradas externas al módulo de manera similar a la asignación de parámetros en el caso de entradas constantes. En este ejemplo, se asignan valores al vector *s*, con valores 0 excepto 1.0 que es asignada a la entrada con mayor actividad y 0.5 a otro de los elementos. El control de la simulación incluye la especificación del tiempo de ejecución del modelo y un intervalo de tiempo de simulación. La salida de la red al finalizar la simulación se muestra en la Figura 7. La entrada de mayor magnitud es la única que se mantiene con valor positivo al finalizar la ejecución del modelo.

4 Líneas Futuras

NSL es parte integral en las diversas áreas de investigación que se llevan a cabo en el laboratorio *CANNES* (*Comportamiento Adaptivo Neuronal, Neurociencias y Simulaciones*) en el ITAM y en sus colaboraciones con otras universidades tales como la USC. Actualmente CANNES tiene como objetivos[3]:

1. El desarrollo de modelos computacionales que expliquen los mecanismos neuronales básicos responsables para el comportamiento de los animales a diferentes niveles de abstracción (auspiciado por el proyecto de colaboración NSF-CONACyT: *Metodología de Simulación de Nivel Múltiple: Un Enfoque Computacional y Experimental para Sistemas Neuronales* [25]).
2. El diseño de lenguajes y arquitecturas de software distribuidas para apoyar el modelado y simulación de estos sistemas con acceso directo desde el Web (auspiciado por el proyecto REDII de CONACyT: *Modelado y Simulación Neuronal en el Web* [1]).
3. La aplicación de estos modelos dentro del marco de la teoría del cerebro y psicología cognitiva para el diseño de agentes robóticos autónomos y robots reales móviles (auspiciado por el proyecto de colaboración NSF-CONACyT: *Robots Ecológicos: Un Enfoque Teórico Esquemático* [5][22]).

Actualmente se está integrando NSL con el *Lenguaje de Esquemas Abstracto* (*ASL* por sus siglas en inglés) [21]. ASL incorpora aspectos de la programación orientada a objetos concurrentes [18], utilizando un modelo jerárquico apoyando procesamiento concurrente y distribuido [19][20].

Agradecimientos

Se agradece en especial a todos los tesistas del laboratorio CANNES del ITAM que han apoyado y siguen apoyando el desarrollo de estos sistemas, en particular Salvador Mármol, Claudia Calderas, Roberto Olivares, Oscar Peguero, Sebastián Gutiérrez y Francisco Peniche y Francisco Otero. Asimismo se agradece al grupo dirigido por Michael Arbib y Amanda Alexander en la Universidad del Sur de California con quienes se mantiene una estrecha colaboración en el desarrollo de NSL.

Referencias

[1] Alexander, A., Arbib, M.A., Weitzenfeld, A., 1999, Web Simulation of Brain Models, *1999 International Conference on Web-Based Modeling and Simulation*, 31(3):29-33, Enero 17-20, San Francisco, CA.

[2] Amari, S., Arbib, M.A., 1977, *Competition and Cooperation in Neural Nets*, Systems Neuroscience (J. Metzler, ed.), pp 119-165, Academic Press.

[3] Arbib, M.A., 1989, *The Metaphorical Brain 2: Neural Networks and Beyond*, Wiley.

[3] Para mayor información sobre estos proyectos favor de accesar el servidor Web de CANNES: http://cannes.rhon.itam.mx.

[4] Arbib, M.A., 1992, Schema Theory, in the *Encyclopedia of Artificial Intelligence*, 2da Edition, Editor Stuart Shapiro, 2:1427-1443, Wiley.

[5] Arkin, R.C., Cervantes-Perez, F., and Weitzenfeld, A., 1997, Ecological Robotics: A Schema-Theoretic Approach, en *Intelligent Robots: Sensing, Modelling and Planning*, Eds. R.C.Bolles, H.Bunke, y H.Noltemeier, pp 377-393, World Scientific.

[6] De Schutter, E., 1992, A Consumer Guide to Neuronal Modeling Software, en *Trends in Neuroscience*, 15(11):462-464.

[7] Didday, R.L. , 1976, A Model of Visuomotor Mechanisms in the Frog Optic Tectum, *Math. Biosci.* 30:169-180.

[8] Dominey, P., Arbib, M.A., 1992, A Cortico-Subcortical Model for Generation of Spatially Accurate Sequential Saccades, Cerebral Cortex, Marzo/Abril 2:153-175.

[9] Hodgkin, A.L., Huxley, A.F., 1952, A Quantitative Description of Membrane Current and its Application to Conduction and Excitation in Nerve, *J. Physiol*. London, 117:500-544.

[10] Hopfield, J., "Neural Networks and Physical Systems with Emergent Collective Computational Abilities", Proc. Of the National Academy of Sciences 79:2554-2558, April 1982.

[11] House, D., 1985, Depth Perception in Frogs and Toads: A study in Neural Computing, Lecture Notes in Biomathematics 80, Springer-Verlag.

[12] McCulloch, W.S., Pitts, W.H., 1943, *A Logical Calculus of the Ideas Immanent in Nervous Activities*, Bull. Math. Biophys., 5:15-133.

[13] Murre, J., 1995, Neurosimulators, e *Handbook of Brain Theory and Neural Networks*, Ed. Michael Arbib, MIT Press.

[14] Ousterhout, J., 1994, *Tcl and the Tk Toolkit*, Addison-Wesley.

[15] Rall, W., 1959, Branching Dendritic Trees and Motoneuron Membrane Resistivity, *Exp. Neurol*., 2: 503-532.

[16] Rumelhart, D.E., Hinton, G.E., and Williams, R.J., 1986, Learning internal representations by error propagation, in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* (D.E. Rumelhart, J.L. McClelland, and PDP Research Group, Eds.), vol. 1, Foundations, Cambridge, MA: MIT Press, pp. 318-362.

[17] Teeters, J. L., Arbib, M. A., Corbacho, F. & Lee, H. B. (1993). Quantitative modeling of responses of anuran retina: Stimulus shape and size dependency. *Vision Research*, 33, 2361-2379.

[18] Wegner, P., 1990, Concepts and Paradigms of Object-Oriented Programming, *OOPS Messenger*, 1(1):7-87.

[19] Weitzenfeld, A., 1992, A Unified Computational Model for Schemas and Neural Networks in Concurrent Object-Oriented Programming, *Tesis Doctoral*, Computer Science Dept., University of Southern California, Los Angeles.

[20] Weitzenfeld, A., Arbib, M., 1991, A Concurrent Object-Oriented Framework for the Simulation of Neural Networks, *ECOOP/OOPSLA '90 Workshop on Object-Based Concurrent Programming, OOPS Messenger*, 2(2):120-124.

[21] Weitzenfeld, A., 1993, ASL: Hierarchy, Composition, Heterogeneity, and Multi-Granularity in Concurrent Object-Oriented Programming, *Workshop on Neural Architectures and Distributed AI: From Schema Assemblages to Neural Networks*, Oct 19-20, Center for Neural Engineering, USC, Los Angeles, CA.

[22] Weitzenfeld, A., Arkin, R.C., Cervantes-Perez, F., Olivares, R., Corbacho, F., 1998, A Neural Schema Ssytem Architecture for Autonomous Robots, *Proc. of 1998 International Symposium on Robotics and Automation*, Diciembre 12-14, Saltillo, Coahuila, México.

[23] Weitzenfeld, A., Arkin, R.C., Cervantes-Perez, F., Peniche, J.F., 1998, Visualization of Multi-level Neural-based Robotic Systems, *2nd Conference on Visual Computing*, Abril 20-24, Mexico, DF, Mexico.

[24] Weitzenfeld, A., Arbib, M.A., 1994, NSL Neural Simulation Language, en *Neural Network Simulation Environments*, Ed. J. Skrzypek, Kluwer.

[25] Weitzenfeld, A., Arbib, M., Cervantes, F., Rudomin, P., Alexander, A., 1998, Multi-level Simulation Methodology: A Computational and Experimental Approach to Neural Systems, *NSF Design and Manufacturing Grantees Conference*, Monterrey, Mexico, 651A-652A.

[26] Weitzenfeld, A., Alexander, A., Arbib, M.A., 1999, *NSL - Neural Simulation Language: System and Applications*, MIT Press (en preparación).

[27] Werbos, P., 1974, Beyond Regression, New Tools for Prediction and Analysis in the Behavioral Sciences, *Tésis Doctoral*, Harvard University, Boston, MA.

[28] Wiskott, L., Fellous, J.-M., Krüger, N., von der Malsburg, C. 1997, Face recognition by elastic bunch graph matching, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):775-779.