UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

MAESTRÍA EN CIENCIAS (FÍSICA)

MODELO DINÁMICO DE APRENDIZAJE EN REDES NEUROCOMPUTACIONALES

TESIS
QUE PARA OPTAR POR EL GRADO DE:
MAESTRO EN CIENCIAS (FÍSICA)

PRESENTA:
ÉLFEGO RUIZ GUTIÉRREZ

DR. RAFAEL ÁNGEL BARRIO PAREDES
INSTITUTO DE FÍSICA, UNAM

DR. JOSÉ LUIS MATEOS TRIGOS
INSTITUTO DE FÍSICA, UNAM

DR. DENIS PIERRE BOYER
INSTITUTO DE FÍSICA, UNAM

MÉXICO, D. F. OCTUBRE 2013

Universidad Nacional Autónoma de México
UNAM
Biblioteca Central
Dirección General de Bibliotecas de la UNAM

UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

Dedicatoria

A mi hija y amor de mi vida
Aurora Ruíz Orozco
motivo de todo mi afecto
y perseverancia...

a mi esposa,
Sirio Anel Orozco Fuentes
por su cariño y dedicación...

a mis padres,
Aurora Gutiérrez Torres
y Élfego Ruíz Schneider
por su apoyo incondicional...

a mi hermanos y amigos entrañables
Jaime Schlittler Álvarez,
Gerardo Viramontes Molina,
Emilio Castelló Deffis

y a todos mis familiares
y amigos que he encontrado
a lo largo de esta gran aventura.

¡Por un mundo en el que todos quepamos!

II

Agradecimientos

A mi director de tesis
el Dr. Rafael Ángel Barrio Paredes,
por su guía y valiosas aportaciones
a lo largo de este trabajo.

Al comité tutor y al sínode:
Dr. José Luis Mateos Trigos y
Dr. Denis Pierre Boyer
Dr. Octavio Miramontes Vidal
Dr. David P. Sanders
Dr. Alejandro Frank Höflich
por sus correcciones y
valiosas contribuciones.

A la DGAPA por el apoyo económico
otorgado a través del proyecto
PAPIIT al comienzo de la maestría.

Al CONACyT por la beca de maestría.

Al Dr. Octavio Miramontes Vidal,
por sus consejos.

A la Dra. Limei Zhang
por aclarar tantos detalles
en estos temas y por darme
un lugar en su laboratorio.

Al Dr. David P. Sanders,
por el sinnúmero de consejos
para hacer más eficientes
los cálculos y por
aclarar conceptos.

Agradezco a mis padres,
por todo el apoyo que me
han brindado.

También a mi esposa,
por su tiempo, dedicación
y cariño.

A Jaime Schlittler, Gerardo
Viramontes y Emilio Castelló
por escucharme y acompañarme
en tiempos difíciles.

A mis queridos amigos
Francisco Favela,
Henrik Jendle
y Adair Chávez
a quienes admiro mucho.

Índice general

Resumen **VII**

Abstract **IX**

Introducción **1**

1. El modelo de la neurona **5**
1.1. El modelo de Hodgkin–Huxley 6
1.2. Propiedades de las ecuaciones de Hodgkin–Huxley 8
1.3. Respuesta de una neurona ante un estímulo 10
1.3.1. Corrientes bajas, neurona integradora 11
1.3.2. Corrientes altas, neurona resonante 14
1.3.3. Corriente media, neurona biestable 18
1.4. Un modelo simplificado de neurona 19
1.4.1. Comparaciones del modelo simplificado de neurona . . . 22

2. La red de neuronas **25**
2.1. El modelo de la red 25
2.2. La actividad de la red neuronal 28
2.2.1. Ejemplo de una red conformada por una sola neurona . . 30
2.2.2. Redes de neuronas no interactuantes entre sí 34

3. Los algoritmos de aprendizaje **39**
3.1. El algoritmo tipo Monte Carlo 40
3.2. El algoritmo genético 42

4. Estudio estadístico del aprendizaje **45**
4.1. La dinámica de la probabilidad: ecuación maestra 46
4.2. Ecuaciones de campo medio y Fokker–Planck 50
4.3. La solución estacionaria 52
4.4. Muerte celular 55

5. Resultados y discusión **59**
5.1. Algoritmos de aprendizaje 59
5.1.1. Monte Carlo 59
5.1.2. Algoritmo genético 61
5.2. Estudio probabilístico del aprendizaje 64
5.2.1. La función de distribución de accesibilidad 65
5.2.2. Simulaciones en la teoría estadística del aprendizaje . . . 68
5.2.3. Robustez ante muerte celular 70

Conclusiones **73**

A. Detalles sobre el modelo de la neurona **77**

B. Detalles de los algoritmos: Los códigos **79**
B.1. Mapa conceptual de los programas 79
B.2. La red de neuronas 81
B.2.1. La neurona 81
B.2.2. La red 85
B.2.3. La red de neuronas 90
B.3. Monte Carlo 94
B.4. El algoritmo genético 97

Resumen

En esta tesis abordamos los procesos de aprendizaje desde el punto de vista estadístico, se presenta una teoría que busca explicar y, en la medida de lo posible, predecir el aprendizaje de redes de neuronas, es decir, la manera en que las redes, mediante una serie de métodos, emulan cada vez con mayor precisión una función.

Para ello hemos construido un modelo de neurona que intenta asemejarse lo más posible a la actividad del modelo de Hodgkin–Huxley, con la ventaja de ser computacionalmente económico, pero que preserva muchas de las diversas maneras en que una neurona puede responder ante diferentes estímulos.

Sin hacer la reducción al modelo de neurona binaria usualmente empleado en la literatura (perceptrón), estudiamos el comportamiento de las redes sometidas a estímulos principalmente binarios para evaluar su respuesta.

También desarrollamos dos algoritmos de aprendizaje, el primero, basado en Monte Carlo como un algoritmo de adaptación tipo prueba y error; y un algoritmo genético que usa como base el de Monte Carlo que optimiza y amplifica la eficacia del primero.

Se analiza, en un ámbito estadístico, la manera en que el aprendizaje de las redes se lleva a cabo. Dado que la mayoría de las variables son modificadas por los algoritmos de manera estocástica, mediante el uso de probabilidades, es posible entender su dinámica durante el proceso de aprendizaje. Nos apoyaremos del formalismo de los procesos estocásticos, específicamente de la ecuación maestra, para examinar con más detalle y con la intención de entender a fondo el mecanismo de aprendizaje.

Se presentan al final los resultados más significativos, con el fin de confrontar los modelos y la teoría aquí planteados.

Abstract

In this thesis we study the process of learning within a statistical approach. We present a theory aiming at explaing and even in some cases, predicting the way networks of neurons modify their behavior to emulate a target function progressively.

We built a model for the neuron that reasonably mimics the activity of the Hodgkin–Huxley model but with the virtue of being computationally light. This model emulates basically the three modalities observed in the HH model: integrator, resonator, and bistable.

We developed two algorithms to teach the neural network. The first of these is a "trial and error" method inspired in the Monte Carlo algorithm, and the second is a genetic algorithm that increases the efficiency of the first.

Then, in a statistical manner we analyze the process of learning itself. Since the algorithms mostly use random variables we rely on the formalism of the Stochastic Processes, especially with the Master Equation to study the learning process.

Finally, we present the most significant results of our theory and compare them with the simulations of the model.

Introducción

Entender los mecanismos con los que el cerebro representa y almacena secuencias temporales es evidentemente un problema fundamental que tiene gran relevancia. Tales mecanismos permiten a los individuos asociar estímulos neutros que preceden y, por lo tanto, le permiten predecir estímulos conductualmente relevantes e ignorar aquellos irrelevantes o desvinculados [1]. A la vez permite la formación de vínculos entre estímulo y respuesta que traducen patrones percibidos en comportamiento, y permiten el almacenamiento y recuperación de memorias de secuencias entre otras. [2]

Se cree que la plasticidad sináptica es el sustrato biológico de los cambios en el cerebro debidos a la experiencia [3]. Se ha observado que la eficacia de la transmisión de pulsos en la sinápsis es modulada como consecuencia de patrones de actividad en ambos extremos de la conexión [4].

Los sistemas biológicos presentan un tipo de memoria muy distinto al que se acostumbra ver en las computadoras: por ejemplo, la memoria es asociativa, es robusta ante ruido o tolerante ante error y también es paralelizable y distribuida [1]; cualidades que conviene entender ampliamente tanto para mejorar técnicas de aprendizaje como para diseñar y fabricar inteligencia artificial, que lleve a cabo tareas más eficientemente.

En el campo de las redes neurales se estudian propiedades de redes de neuronas idealizadas. En particular, se pueden distinguir tres motivaciones de investigación para este campo interdisciplinario:

- *Ingeniería*: Se busca crear máquinas que puedan "aprender", "clasificar", desempeñar "reconocimiento de patrones" o "descubrir patrones" para realizar tareas con mayor eficiencia.

- *Sistemas complejos*: Las redes neurales muestran una complejidad adaptativa cuyas propiedades son interesantes por sí mismas.

- *Biología*: Se busca entender el funcionamiento del cerebro. En este rubro se encuentra la construcción de modelos que permitan dilucidar la manera

en que la memoria y el procesamiento e interpretación del entorno se lleva a cabo. Es este campo en el cual esta tesis está principalmente enfocada.

Por lo mismo, hay una gran cantidad de modelos, cada uno con su grado de interés, que están especializados en diferentes campos. Entre ellos, una pequeña porción está destinada a ser una buena aproximación de los sistemas biológicos [1]. Destinaremos una parte del trabajo en construir uno.

Esto nos llama a estudiar particularmente los procesos de aprendizaje. Dentro de esto podemos encontrar fundamentalmente dos rubros, los algoritmos de aprendizaje supervisado y los de aprendizaje no supervisado. En el aprendizaje supervisado, se provee a la red neural un conjunto de datos y metas u objetivos y un agente, ya sea un preceptor o tutor, que guía a la red a aproximarse al objetivo. En el aprendizaje no supervisado, se le provee a la red un conjunto de ejemplos para memorizar, de tal forma que los ejemplos puedan ser recordados más tarde con la intención de reinterpretarlos y construir un conjunto más amplio, generalizado o que tome los aspectos esenciales del conjunto original [1].

Dentro de la gran colección de algoritmos de aprendizaje no supervisado sobresalen los mapas auto organizados (SOM, por sus siglas en inglés), las teorías de resonancia adaptativa (ART) y modelos tipo hebbianos que autoregulan las conexiones sinápticas bajo reglas observadas experimentalmente. [2–8].

No es claro cómo es que se lleva a cabo el aprendizaje en los organismos equipados con sistema nervioso central y en general es una interrogante abierta a la cual se ha invertido una gran cantidad de esfuerzos. Nosotros no abordaremos directamente esta cuestión; sin embargo, hemos construido una serie de herramientas en este tratado que pueden ser útiles para resolverla y que además por sí mismas encontramos interesantes.

En esta tesis se desarrollan dos modelos, un modelo de neurona que será empleado para crear redes de ellas, y un modelo de aprendizaje para la red, pues con esto se pretende que la red de neuronas pueda emular, mediante el proceso de aprendizaje, una función particular. Este modelo fue construido con la finalidad de ayudar a entender de lo que ocurre en el sistema nervioso. Por encima de esto, se formuló una teoría que respalda, predice y explica cómo sucede el aprendizaje, exponiendo las condiciones que favorecen o que limitan el aprendizaje.

Comenzando en el capítulo primero proponiendo un modelo de neurona inspirado en el de Hodgkin–Huxley (HH) . Este modelos lo empleamos como la base para la red de neuronas. Mediante el uso de programas computacionales, extraemos las propiedades más relevantes en este contexto, del modelo de HH e implementamos uno que, de la mejor manera, las emule pero que

computacionalmente no consuma muchos recursos.

En el capítulo segundo, plantearemos una manera de aproximar los mecanismos de transmisión de pulsos entre neuronas para construir un modelo de red neural. Asimismo, se expone la arquitectura y reglas que le corresponden a la red para ser evaluada por el preceptor de los algoritmos de aprendizaje.

Para el capítulo tercero, se presentan los algoritmos de aprendizaje implementados: un mecanismo de prueba y error tipo Monte Carlo y un algoritmo genético.

En el capítulo cuarto se desarrolla una teoría que intenta explicar y predecir los procesos de aprendizaje en general. Haciendo un análisis más detallado al modelo de red aquí propuesto finalizamos con el capítulo quinto, en donde se presentan los resultados más significativos y se confronta esta teoría.

Capítulo 1

El modelo de la neurona

Una neurona es una célula del sistema nervioso que se destaca por una propiedad fundamental: la excitabilidad, esto es, la neurona es capaz de formar pulsos electroquímicos como reacción a los estímulos que se le presentan. Las neuronas son células altamente especializadas en el procesamiento y transmisión de señales. Hay una variedad muy amplia de formas, tamaños y propiedades electroquímicas de estas células.

De manera esquemática, una neurona se divide en tres partes: el soma, las dendritas y el axón. El **soma** o cuerpo celular es la parte que contiene al núcleo de la célula y es donde se realiza la mayoría de las funciones fisiológicas.

Las **dendritas** son extensiones del soma en forma de ramas que recogen información en forma de señales electroquímicas para transmitirlas al cuerpo celular.

El **axón** es una prolongación que porta las señales producidas en el soma hacia otras células. La longitud de esta parte llega a medir decenas, cientos o en ocasiones miles de veces el tamaño del cuerpo celular.

Las neuronas, a lo largo de su axón, pueden hacer contacto con otras neuronas. Este contacto es llamado **sinapsis** y se distingue por ser una unión de brecha, es decir, consta de un pequeño espacio entre neurona y neurona. Esta interacción tiene el efecto de promover o inhibir la producción de pulsos, llamados potenciales de acción, en la neurona que los recibe.

Los pulsos llamados potenciales de acción son los medios principales de comunicación entre este tipo de células y son electroquímicos.

Hay neuronas que emiten espontáneamente y otras que sólo lo hacen como respuesta a estimulación adecuada que reciben [4].

1.1. El modelo de Hodgkin–Huxley

Uno de los modelos más importantes en la neurociencia computacional es el de Hodgkin–Huxley (HH), el modelo de respuesta del axón gigante de calamar. Utilizando técnicas experimentales muy novedosas en aquel entonces, Hodgkin y Huxley (1952) determinaron que el axón del calamar transporta principalmente tres corrientes: la corriente de potasio (K^+), la de sodio (Na^+) y una corriente de fuga que transporta principalmente iones de cloro (Cl^-) [9].

En el modelo se plantea que la membrana celular de la neurona se puede reducir a un circuito eléctrico. La membrana, al ser una capa muy delgada opera como un capacitor y, al mismo tiempo, al ser permeable a diferentes iones, también transporta corriente. La corriente total I que atraviesa la membrana celular es

$$\begin{aligned} I &= C\dot{V} + \sum_i I_i \\ &= C\dot{V} + \sum_i g_i(V - E_i), \end{aligned}$$

donde V es el potencial de la membrana, C es su capacitancia e I_i es la corriente del ion i el cual es proporcional a la conductancia g_i y a la diferencia entre el potencial de la membrana y el de equilibrio del ion E_i.

La conductancia g_i del ion i es una variable dinámica y responde a cambios en el potencial. Estos cambios no son inmediatos, sino que dependen de las variables de activación e inactivación dados por la estructura de los canales iónicos, estas variables son n, m y h. El conjunto completo de ecuaciones diferenciales para este modelo es:

$$C\dot{V} = I - \overline{g}_{\mathrm{K}} n^4 (V - E_{\mathrm{K}}) - \overline{g}_{\mathrm{Na}} m^3 h (V - E_{\mathrm{Na}}) - g_{\mathrm{L}}(V - E_{\mathrm{L}}), \quad (1.1)$$

$$\dot{n} = \frac{n_\infty(V) - n}{\tau_n(V)}, \quad (1.2)$$

$$\dot{m} = \frac{m_\infty(V) - m}{\tau_m(V)}, \quad (1.3)$$

$$\dot{h} = \frac{h_\infty(V) - h}{\tau_h(V)}. \quad (1.4)$$

La variable n es llamada la compuerta de activación del potasio, de igual manera m para la del sodio y h la variable de inactivación de este último ion. El exponente de estas variables en la ec. 1.1 está asociado al número de compuertas en el canal iónico, por ejemplo, para el potasio, hay cuatro compuertas, por lo tanto, ese es el grado de n.

A continuación se detallará cada elemento que conforma al sistema de ecuaciones. Los valores de las constantes que se utilizarán para este trabajo son los siguientes: los potenciales de reposo E_i según el equilibrio de Nernst,

$$E_{\mathrm{K}} = -12 \text{ mV}, \qquad E_{\mathrm{Na}} = 120 \text{ mV}, \qquad E_{\mathrm{L}} = 10.6 \text{ mV};$$

los valores de las conductancias típicamente son

$$\overline{g}_{\mathrm{K}} = 36 \, \frac{1}{\mathrm{k}\Omega \text{ cm}^2}, \qquad \overline{g}_{\mathrm{Na}} = 120 \, \frac{1}{\mathrm{k}\Omega \text{ cm}^2}, \qquad g_{\mathrm{L}} = 0.3 \, \frac{1}{\mathrm{k}\Omega \text{ cm}^2};$$

el valor para la capacitancia es $C = 1 \, \mu\mathrm{F/cm}^2$ y la corriente total es denotada por I. Las funciones de las variables de activación e inactivación están dadas por

$$i_\infty(V) = \frac{\alpha_i}{\alpha_i + \beta_i}, \qquad \tau_i(V) = \frac{1}{\alpha_i + \beta_i}, \qquad \text{para } i = n, m, h; \quad (1.5)$$

desglosándolas aún más,

$$\begin{aligned} \alpha_n &= 0.01 \frac{10 - V/\text{mV}}{\exp(1 - V/10 \text{ mV}) - 1} \text{ Hz}, & \beta_n &= 0.125 \exp(-\frac{V}{80 \text{ mV}}) \text{ Hz}; \\ \alpha_m &= 0.1 \frac{25 - V/\text{mV}}{\exp(2.5 - V/10 \text{ mV}) - 1} \text{ Hz}, & \beta_m &= 4 \exp(-\frac{V}{18 \text{ mV}}) \text{ Hz}; \\ \alpha_h &= 0.07 \exp(-\frac{V}{20 \text{ mV}}) \text{ Hz}, & \beta_h &= \frac{1}{\exp(3 - V/10 \text{ mV}) + 1} \text{ Hz}. \end{aligned} \quad (1.6)$$

Las funciones de las ecs. 1.5 (izquierda) son llamadas **funciones estacionarias** de activación e inactivación. Se puede medir el valor de estas funciones en dependencia del voltaje experimentalmente bajo un procedimiento llamado *voltage clamp* en el cual se mantiene fijo el valor del voltaje de la membrana y se mide la corriente iónica a través de los canales iónicos específicos. El valor al que llega la conductancia asintóticamente en el tiempo es proporcional a la variable de activación o inactivación. [9]

Las funciones de las ecs. 1.5 (derecha) son convencionalmente llamados **tiempos característicos** de las variables de activación o inactivación. En la figura 1.1 se muestra la dependencia de las seis funciones. Como se puede apreciar, las funciones $n_\infty(V)$ y $m_\infty(V)$ son funciones crecientes con respecto al voltaje, debido al cual se llaman funciones de activación, a diferencia de $h_\infty(V)$ que es decreciente, pues se refiere al ligando de inactivación del canal de sodio. [4]

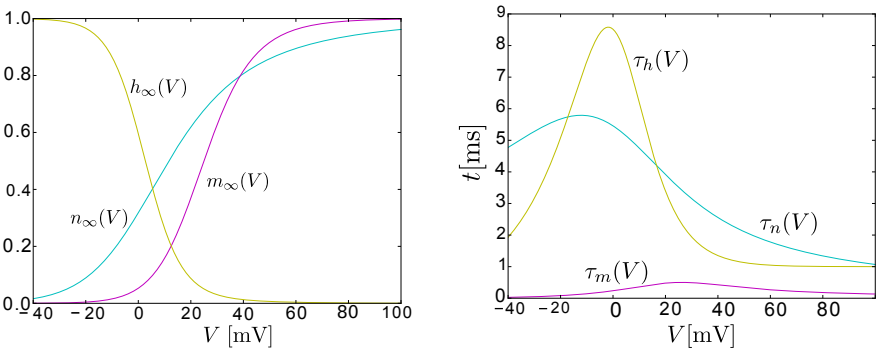

Figura 1.1: Gráficas de las funciones estacionarias de activación e inactivación (izquierda) y la dependencia en el voltaje de los tiempos característicos de las variables (derecha) como expuestas en las ecs. 1.6 respectivamente.

Dado que hay cuatro variables dinámicas V, n, m y h, el espacio fase de este sistema es de cuatro dimensiones. Comprender la dinámica de las trayectorias en el espacio fase requerirá la integración numérica de estas ecuaciones, ya que no se conoce solución analítica.

Con la intención de indagar sobre el comportamiento de la célula, haremos un experimento virtual variando controladamente la corriente total. A continuación se estudiarán algunas de las propiedades neurocomputacionales que se dilucidan a partir de este experimento.

1.2. Propiedades de las ecuaciones de Hodgkin–Huxley

En general, podemos calcular la localización en el espacio fase de los puntos fijos por medio de la suposición $\dot{V} = \dot{n} = \dot{m} = \dot{h} = 0$, es decir, la solución estacionaria. Esto lleva a la siguiente ecuación:

$$I = \overline{g}_{\mathrm{K}} n_\infty^4 (V - E_{\mathrm{K}}) + \overline{g}_{\mathrm{Na}} m_\infty^3 h_\infty (V - E_{\mathrm{Na}}) + g_{\mathrm{L}} (V - E_{\mathrm{L}}).$$

En esta ecuación, sólo V es variable; sin embargo, se trata de una ecuación trascendental y las raíces se obtienen por métodos numéricos. En la figura 1.2 (izquierda) se muestra gráficamente el valor de el punto fijo en dependencia de la corriente total.

Mediante un análisis lineal, en los eigenvalores de la matriz jacobiana derivada de las ecuaciones diferenciales se puede observar que para $I = I_2 \approx 8.41\,\mu\mathrm{A}$,

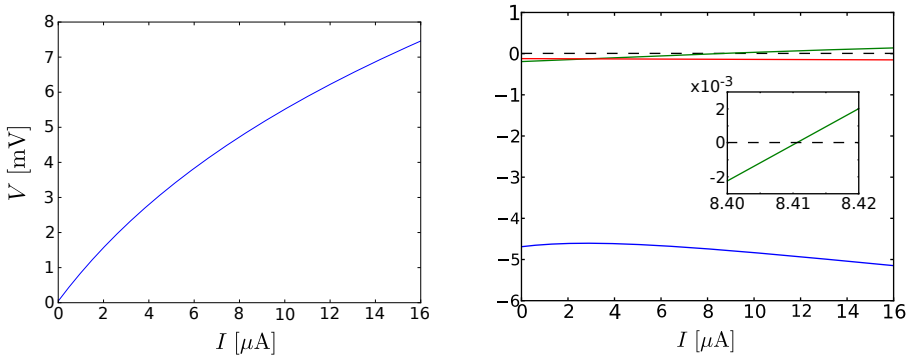

Figura 1.2: Valor del potencial de la membrana en el punto fijo (izquierda). Estabilidad de los puntos fijos según la parte real de los eigenvalores (derecha).

hay un cambio en el signo de la parte real para dos eigenvalores, esto implica que cambia la estabilidad del punto fijo. Como se puede observar en la figura 1.2 (derecha), para corrientes menores a I_2, la parte real de todos los eigenvalores es negativa, por lo cual, es un punto fijo atractor. En cambio, para $I > I_2$, la parte real de dos de ellos cambian de signo; consecuentemente, el atractor pierde estabilidad y se convierte en repulsor, lo que quiere decir que la neurona ya no permanecerá quiescente en su actividad [9].

Podemos, a partir de las variables del sistema de ecuaciones de HH, construir el espacio fase y observar la dinámica del sistema por las trayectorias que se trazan. Al integrar las ecuaciones de HH con diferentes condiciones iniciales, se puede observar que siempre se pueden producir los potenciales de acción, es decir, pulsos de cientos de milivolts de muy breve duración, o bien, trayectorias en el espacio fase que pasan rápidamente por valores altos en V. En ocasiones se observa que éstos se producen repetidamente y en otras, después del disparo, el estado de la neurona tiende al reposo. En la figura 1.3 se muestra esta dinámica.

Cuando la neurona se encuentra en una actividad de pulsos periódica, se forma un ciclo límite en el espacio fase, sin embargo, este ciclo límite no sólo aparece hasta que el punto fijo pierde su estabilidad, sino también en valores de la corriente. Dado que para estos valores de I se encuentra el punto fijo y además el ciclo límite, la neurona es **biestable**: dependiendo de las condiciones iniciales o estimulación adecuada, la célula tenderá a un comportamiento o a otro.

Para saber en qué valores de I se forma el ciclo límite, basta con comenzar con una trayectoria en el ciclo límite bajo corrientes altas donde no se presenta

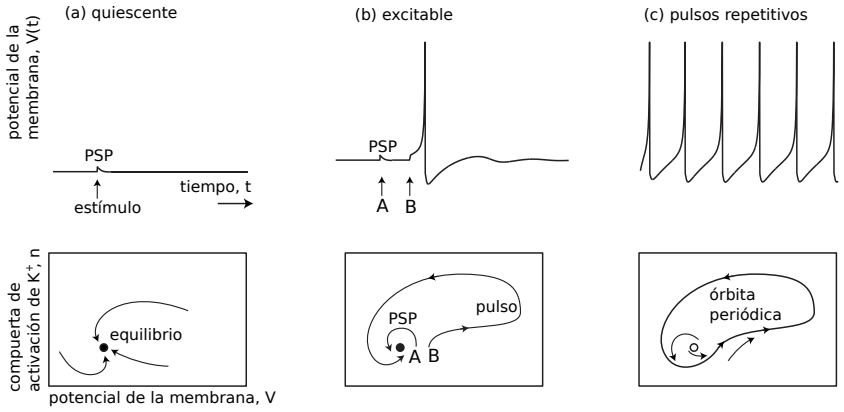

Figura 1.3: Actividad quiescente (a), excitable (b) y de pulsos repetitivos (c). En la parte superior se muestra la evolución del potencial de la membrana, en la parte inferior se muestra una proyección del espacio fase. Tomada de Izhikevich (2007) [9].

biestabilidad y posteriormente disminuir la corriente poco a poco hasta perder dicho atractor. En la figura 1.4 se muestra un diagrama de bifurcaciones en donde se puede localizar el potencial eléctrico de los atractores y los repulsores. La curva sólida que pasa cerca del origen muestra la localización del punto fijo; para valores de $I > I_2$ la línea continua intermitente, simbolizando que éste se ha tornado inestable. De la misma manera, para $I_1 \approx 5.31\mu$A se abren dos curvas por abajo y por encima de la anterior, una sólida y la otra punteada, simbolizando la proyección sobre V del ciclo límite atractor y de un repulsor que divide al espacio fase en los dominios de los atractores.

En breve, para $I < I_1$ se encuentra únicamente el punto fijo, con $I_1 < I < I_2$ hay coexistencia entre el ciclo límite y el punto fijo y para $I_2 < I$ sólo permanece el ciclo límite como único atractor.

1.3. Respuesta de una neurona ante un estímulo

Con fines de reproducir un modelo computacionalmente más modesto, es preciso estudiar la respuesta de la neurona bajo diferentes estímulos. Debido a la forma de los potenciales de acción, breves pero muy intensos, los estímulos serán, por fines prácticos, distribuciones tipo delta de Dirac que se añadirán al

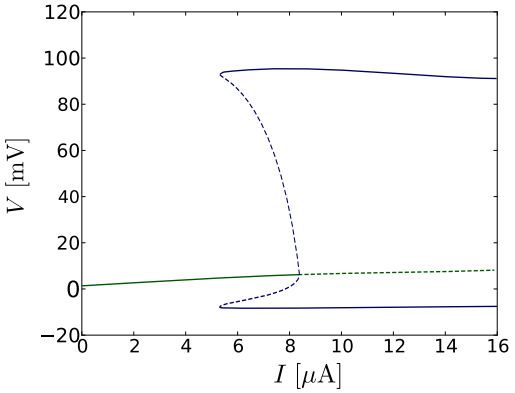

Figura 1.4: Diagrama de bifurcaciones en el comportamiento de la neurona: la linea verde representa el punto fijo, la línea azul los extremos de la proyección de los ciclos límite. En ambos casos la linea sólida representa estabilidad a diferencia de la punteada que simboliza inestabilidad.

cambio del potencial de la membrana, es decir, se añade $A\delta(t-t_s)$ a la ec. 1.1,

$$\dot{V} = f_V(V,n,m,h) \rightarrow \dot{V} = f_V(V,n,m,h) + A\delta(t-t_s),$$

donde f_V es el miembro derecho de la ec. 1.1 dividido entre C, la capacitancia de la membrana, A es la amplitud del estímulo y t_s es el instante en que lo recibe. Integrando las ecuaciones de HH se puede observar que después del estímulo, la trayectoria hará un salto de tamaño A paralelo al eje V en el espacio fase. A continuación se estudiarán las consecuencias de este tipo de estimulación, variando el estímulo tanto en amplitud, A, como en el instante, t_s, en que se aplica.

1.3.1. Corrientes bajas, neurona integradora

Comenzaremos por describir la respuesta del sistema para corrientes con valores entre 0 e I_1. Dado que en esta región el punto fijo es el único atractor, sabemos que la trayectoria eventualmente regresará a éste. Si el estímulo produce una trayectoria corta y directa de regreso al punto fijo, diremos que el estímulo no fue suficiente para producir un potencial de acción; por el contrario, si el potencial de la membrana se eleva hasta las centenas de milivolts, sabremos que se produjo una espiga de voltaje en la célula. Partiendo del reposo, la célula puede ser estimulada con diferentes valores en A con el fin de

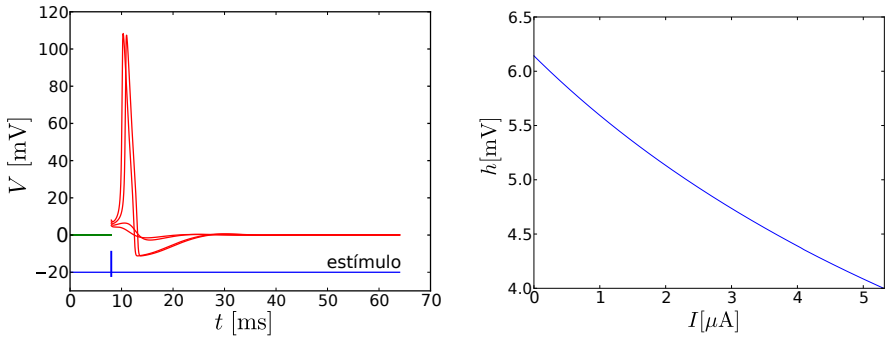

Figura 1.5: Umbral de la neurona HH. (Izquierda) Diferentes pruebas para una neurona en estado quiescente, para diferentes valores en la amplitud del estímulo, la neurona responderá con un pulso de potencial o no, el valor mínimo para el cual la neurona responderá con un pulso será el umbral. (Derecha) Diferentes valores del umbral h según la corriente, la línea sólida son los datos obtenidos al integrar las ecuaciones de HH, la linea intermitente representa el ajuste de curva.

averiguar cuál es el valor mínimo para que la célula responda con un pulso. Este valor lo llamaremos **umbral**. Es de esperarse que el umbral dependa de la corriente total y, dado que el punto fijo pierde estabilidad conforme la corriente aumenta, es de esperarse que el umbral decrecerá, véase figura 1.5.

El método para determinar el valor del umbral fue el siguiente:

1. Se toma una trayectoria lo más cercana posible al punto fijo, después de un intervalo de tiempo se produce el estímulo, es decir, se agrega A al potencial de la membrana en ese instante.

2. Se toma ese nuevo punto como inicial y se comienza a integrar numéricamente de nuevo la ecuación diferencial.

3. Si la nueva trayectoria en algún momento sobrepasa 90 mV, entonces se considera que hubo formación de un disparo y, por lo tanto, se propone un valor ligeramente menor para el estímulo A. Si por el contrario, la trayectoria no muestra la espiga de voltaje, se propondrá un valor ligeramente mayor para A.

4. Se pueden repetir estos pasos tantas veces como se desee haciendo los incrementos o decrementos en A cada vez más pequeños con la finalidad

de aumentar la precisión del valor del umbral, y así para diferentes valores de I.

Se encontró, por medio de un ajuste de curva, que el valor aproximado del umbral h en función de la corriente está dado por la siguiente expresión:

$$h(I) = 6.14 - 0.59I + 0.046I^2 - 0.0021I^3. \tag{1.7}$$

Seguido de determinar el valor del umbral, se estudió el efecto que tiene el estímulo temporalmente, es decir, cómo cambia la respuesta de la neurona en dependencia del momento del estímulo t_s. Es importante mencionar que las ecuaciones de HH son explícitamente independientes en t y, por ello, es evidente que hay invariancia respecto a corrimientos en el origen temporal, lo que significa que si la neurona está en un estado quiescente y se estimula, no importará en qué momento se haga. Sin embargo, se puede ver que si la neurona no está en el punto de reposo, o particularmente si se encuentra en el regreso de un potencial de acción, la célula responderá al estímulo dependiendo del momento en el que se le aplique. En particular, será de nuestro interés medir el intervalo de tiempo que tarda ésta para poder volver a ser estimulada. A este intervalo de tiempo lo llamaremos **período refractario**. Al igual que el umbral, es de esperarse que el período refractario cambie su valor si la corriente interna cambia.

Se determinó el período refractario al estilo en que se determinó el del umbral:

1. Partiendo de un estado quiescente se estimula a la neurona con un pulso que por poco sobrepase al umbral y se integra el conjunto de ecuaciones diferenciales.

2. Se estima un intervalo de tiempo que debe transcurrir desde el primer estímulo en que se cree que la neurona podría responder de nuevo.

3. Si la segunda perturbación causó un pulso en el potencial de la membrana, se propone un intervalo de tiempo ligeramente menor para repetir desde el paso anterior. Si, por el contrario, no hubo respuesta por parte de la célula, se propone un intervalo ligeramente mayor.

4. Se repite el paso anterior con la finalidad de aumentar la precisión. Además, se procede desde el primer paso para diferentes valores de I.

Los resultados se muestran en la figura 1.6. La frecuencia refractaria, ω, esto es el inverso del período refractario, como función de la corriente interna es aproximada por un ajuste de curva,

$$\omega(I) = 5.53\text{Hz}\sqrt{I + 0.013\mu\text{A}} + 45.4\text{Hz}. \tag{1.8}$$

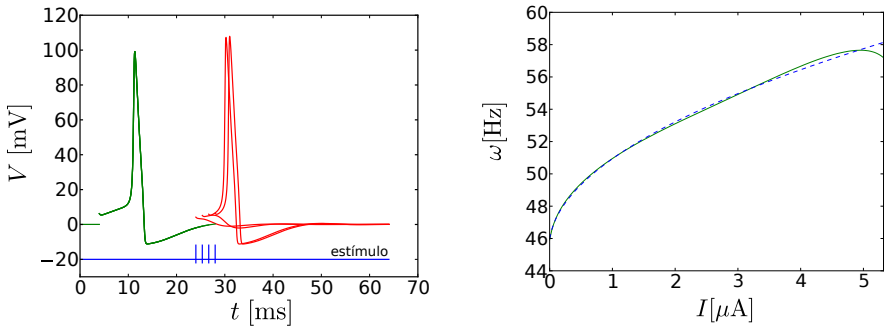

Figura 1.6: Período refractario en la neurona de Hodgkin–Huxley. (Izquierda) Gráfica representativa de la manera en que se determina el período refractario según las ecuaciones HH. El período refractario es el intervalo de tiempo que debe transcurrir después de un estímulo para que la neurona pueda volver a ser estimulada. (Derecha) Valor de la frecuencia refractaria (inverso del período refractario) en dependencia de la corriente total, la línea sólida representa el valor obtenido numéricamente. La linea intermitente muestra el ajuste de curva.

1.3.2. Corrientes altas, neurona resonante

Dado que la región de corrientes intermedias tiene características de ambos extremos, el de corriente baja y el de alta, continuaremos describiendo el caso de corrientes altas.

Si por la neurona hay flujo alto de iones, la corriente a través de la membrana es alta y, por lo tanto, el único atractor en el espacio fase es el ciclo límite. Para estudiar el efecto que producen los estímulos sobre trayectorias que tienden a un ciclo límite, es pertinente introducir el concepto de fase de oscilación. Este concepto puede ser definido para cualquier sistema que presente un ciclo límite, atractivo o repulsivo e independientemente de la dimensionalidad del espacio fase del sistema.

Mientras la célula esté emitiendo pulsos periódicamente, es posible parametrizar la trayectoria, que es una curva cerrada por medio de una fase, esto es,

$$\theta \equiv \frac{t}{T} \bmod 1, \tag{1.9}$$

donde T es el período y θ es llamada la **fase de oscilación**. Por costumbre se emplea el punto del ciclo límite donde el potencial alcanza su valor máximo para situar el cero de la fase; sin embargo, por cuestiones de precisión, hemos decidido situar el origen de la fase en el mínimo del potencial. Esto lo podemos

ver en la figura 1.7, donde el ciclo límite es mapeado al círculo (inferior derecha).

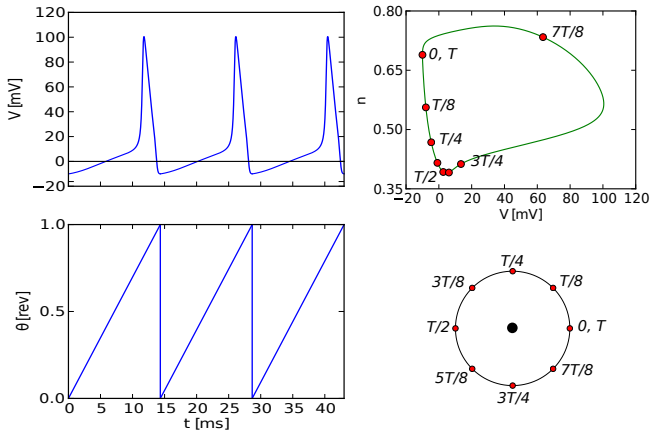

Figura 1.7: Definición de fase de oscilación, θ. El ciclo límite (*superior derecha*) es homeomorfo al circulo (*inferior derecha*).

La fase de oscilación también puede ser definida fuera del ciclo límite, siempre y cuando haya atracción a éste. Considérese, por ejemplo, a un punto y_0 en la cuenca de atracción. Con el paso del tiempo, la evolución del sistema trazará una curva $y(t)$. Estrictamente hablando, la trayectoria $y(t)$ al no estar sobre el ciclo límite no es periódica, sin embargo, tiende a serlo asintóticamente. Es decir, existe una trayectoria $x(t)$ sobre el ciclo límite que comienza en cierto punto x_0 y satisfase

$$\lim_{t\to\infty} |y(t) - x(t)| = 0. \tag{1.10}$$

Es importante señalar la importancia de escoger bien al punto x_0; de no ser así, la diferencia entre las trayectorias no tenderá a cero con el paso del tiempo. Por esta razón se le asigna a y_0 la misma fase que x_0.

Análogamente, podemos, al tiempo inicial $t = 0$, fijar un punto x_0 sobre el ciclo límite y encontrar todos los puntos y_0 que cumplen con la ec. 1.10 cuando $x(0) = x_0$. El conjunto de todos esos puntos forma una variedad que se conoce como la **isócrona** de x_0. Todos los puntos de esta variedad tienen comportamiento asintótico eventualmente indistinguible de $x(t)$, por ello, se dice que tienen la misma fase que x_0.

Las neuronas, al recibir estímulos de otras células modifican sus trayectorias en el espacio fase. Dado que el ciclo límite es el único atractor, las trayectorias

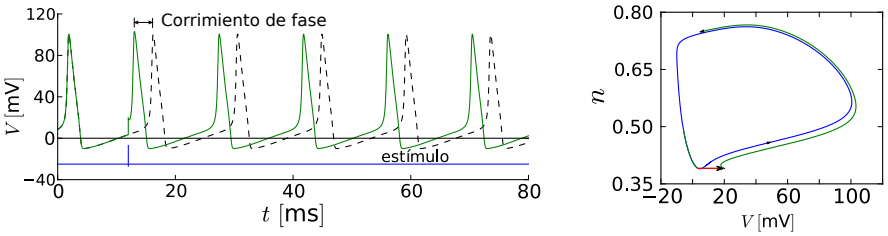

Figura 1.8: (Izquierda) Respuesta de una neurona ante un estímulo $I(t)$ (línea sólida) comparada con una neurona libre de estímulo (línea punteada). (Derecha) Proyección del espacio fase $[V, n]$ en donde se muestra la discontinuidad provocada por el estímulo al igual que el cambio de fase.

eventualmente regresarán a éste. Se puede decir entonces, que es la fase la que es modificada por el estímulo. Como se puede ver en la figura 1.8, al comparar el potencial de la membrana de la célula estimulada (línea sólida) contra la libre (línea punteada), se puede observar una diferencia de tiempo entre los picos.

Al tiempo t_s la trayectoria salta de $V(t_s^-)$ a $V(t_s^+) = V(t_s^-) + A$; en general este salto cambiará la isócrona de la trayectoria. Llamaremos **corrimiento de fase** o PRC por sus siglas en inglés (*phase reset curve*) a la diferencia entre la fase después del estímulo y la fase anterior a este,

$$\mathrm{PRC}(A, \theta) \equiv \theta_{t_s^+} - \theta.$$

Naturalmente, el corrimiento de fase dependerá de la fase θ en que se encuentre el oscilador al momento del estímulo y de la intensidad de éste, que hemos denotado por A. Conocer el valor de la PRC para cualquier fase θ y amplitud de estímulo A es de gran utilidad para dilucidar la dinámica de la neurona bajo cualquier tipo de estimulación, no obstante, esto no es una tarea fácil y por lo general, se tiene que recurrir a métodos numéricos.

Dado que el ciclo límite no cambia significativamente de forma con la corriente, supondremos que la curva de corrimiento de fase es invariante ante este parámetro. Para una neurona con corriente interna $I = 10\,\mu$A se calculó la curva de corrimiento de fase. El método para la obtención de los valores de la PRC fue el siguiente:

1. Se comienza con una trayectoria en el ciclo límite y se determina con precisión el período de éste.

2. Se produce un estímulo a la célula para después dejar evolucionar la trayectoria hasta que aproximadamente regrese al ciclo límite.

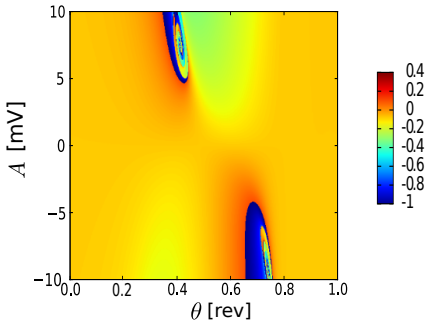

Figura 1.9: Gráfica de contornos de la PRC en el plano $A \times \theta$, los tonos indican la magnitud de la respuesta de la neurona tal como se indica en la barra de la derecha.

3. Se calcula la diferencia de tiempo que toma a una trayectoria control, libre de estímulo, alcanzar a la perturbada.

4. Se hace un barrido para todas las fases, es decir, se procede desde el primer paso para diferentes momentos en que se estimula a la célula virtual.

5. Se repiten los pasos anteriores para diferentes amplitudes de estimulación.

Los resultados numéricos que se obtuvieron se muestran en gráficas en la figura 1.9.

Para concluir el análisis, se describirá la variación de la frecuencia de oscilación en el ciclo límite en dependencia de la corriente. El método para determinar el valor de la frecuencia fue el siguiente:

1. Se integra una trayectoria que comience en el ciclo límite hasta que llega al mínimo en el potencial de la membrana.

2. Una vez ahí se continúa integrando en el tiempo hasta que la trayectoria regresa.

3. Se procede de la misma manera para varios valores de la corriente.

Los resultados se muestran en la figura 1.10. Mediante el ajuste de una curva se encuentra que

$$\omega(I) = 10.4\text{Hz}\sqrt{I - 5.1\mu\text{A}} + 46.8\text{Hz}, \tag{1.11}$$

es una buena aproximación. Esta ecuación es la continuación para corrietes $I > I_1$, pues tratamos de incluir en ω tanto la frecuencia refractaria como la frecuencia del ciclo límite.

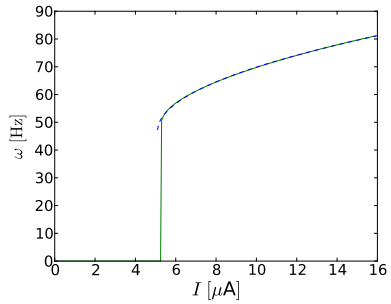

Figura 1.10: En esta figura se muestra la dependencia de la frecuencia del ciclo límite con respecto a la corriente de fondo. La línea sólida muestra los valores encontrados mediante la simulación y la línea punteada muestra el ajuste de curva.

1.3.3. Corriente media, neurona biestable

Una neurona con corrientes I entre I_1 e I_2 tiene un punto fijo como atractor así como también un ciclo límite. Por lo tanto, heredará algunas propiedades que se observan en corrientes altas y en corrientes bajas. Básicamente, dado que hay dos cuencas de atracción, habrá competencia por el comportamiento asintótico de las trayectorias.

Entre las dos cuencas de atracción existe una variedad llamada separatriz, la cual, como su nombre lo sugiere, divide en dos regiones al espacio fase. Si una neurona recibe estimulación adecuada, la trayectoria puede cruzar dicha separatriz alternando su comportamiento asintótico. Esto implica que existe una noción de umbral y éste se identifica con la separatriz.

En resumen, es de esperarse tanto un comportamiento quiescente de la neurona que, muy fácilmente puede cambiar a resonante y viceversa.

Las neuronas, en aislamiento compensan las corrientes iónicas de tal manera que la corriente total es nula. Cuando una de estas células recibe un estímulo de otra neurona, no se induce bifurcación alguna debido a lo estrechos que son los pulsos, sin embargo, cuando hay estímulación contínua y de diversas formas, la corriente total varía y las bifurcaciones son provocadas.

1.4. Un modelo simplificado de neurona

En esta sección detallaremos el modelo de neurona que se empleó para este trabajo. Este modelo fue propuesto por nosotros y pretende ser una síntesis para el modelo de neurona de HH.

Las ecuaciones diferenciales propuestas por Hodgkin y Huxley, si bien son muy precisas, también son computacionalmente muy costosas. Por ello y, debido a la cantidad de tareas que se le delegaron a la computadora, se propone un modelo de neurona que conserva muchas de las propiedades descritas en las secciones anteriores pero que facilite y reduzca considerablemente el tiempo de cálculo numérico.

El modelo de neurona que se propone será el de una cuya respuesta varía entre dos valores, cero y uno; por esta razón se le llama neurona binaria. Simularemos la corriente total de iones que cruzan la membrana celular de la neurona con una variable interna I que determina las constantes internas de la neurona.

La respuesta de la neurona, llamada **la actividad**, ψ será función del estado de la neurona y de los estímulos que reciba,

$$\psi(s) = \theta(s - h)\theta(\varphi - 1), \tag{1.12}$$

donde θ es la función escalón de Heaviside, s es el estímulo que la neurona recibe, φ es la fase y h es la cantidad de estímulo necesario para responder con un disparo, es decir, el umbral. Tanto φ como h son variables dinámicas, gobernadas por las siguientes ecuaciones.

La variable de fase está regida por

$$\dot{\varphi} = \omega_0 + \delta\omega + \mathrm{PRC}(s, \varphi)\delta(t - t_s),$$

donde ω_0 es la velocidad de recuperación, esto es, la frecuencia refractaria dada por la ecuación 1.8 si I es fijada a un valor inferior a I_1, o la frecuencia de oscilación dada por la ecuación 1.11 si $I > I_2$. La función $\mathrm{PRC}(s, \varphi)$ es la que se midió a partir del modelo HH; φ_s es la fase de oscilación al momento del disparo. $\delta\omega$ es una pequeña corrección a la frecuencia de la neurona –hemos observado en la neurona HH que si la estimulación es muy intensa, tanto positiva como negativamente, la neurona modifica ligeramente su frecuencia. Entonces, $\delta\omega$ será función de la variable interna V que se le conoce como la **variable de activación**,

$$\delta\omega = 0.0106V.$$

Para el umbral h, se formuló la siguiente ecuación:

$$h = \frac{1}{100}(h_0 - V),$$

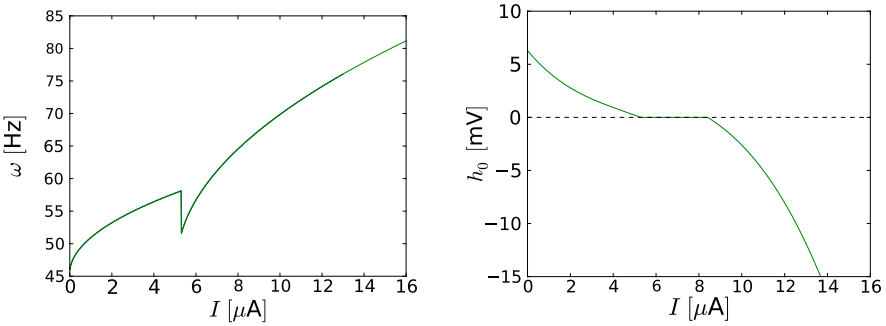

Figura 1.11: (Izquierda) Frecuencia de refractaria o de oscilación ω_0 como función de la corriente. (Derecha) Valor del umbral base h_0 como función de la corriente.

donde h_0 es el umbral de la neurona. A semejanza de los valores adquiridos con el modelo de HH anteriormente, h_0 toma la forma

$$h_0(I) = \begin{cases} 6.3 - 2.4I + 0.38I^2 - 0.029I^3 & \text{si } I < I_1 \\ 0 & \text{si } I_1 < I < I_2 \\ 6.3 - 1.45I + 0.23I^2 - 0.0173I^3 & \text{si } I_2 < I \end{cases}.$$

El umbral del modelo de esta neurona está inspirado en la ecuación 1.7. Sin embargo, como puede observarse para valores de $I > I_2$ toma valores negativos, esto con el fin de producir efectos más realistas a los que se observan el las neuronas reales, pues la célula dejará de disparar si es inhibida severamente. En la figura 1.11 se muestra con una gráfica la forma del umbral.

La variable V es una que imita el potencial de la membrana. El factor $1/100$ se incluye por normalización, debido a que la extensión en la oscilación de la neurona es de aproximadamente 100 mV y el modelo aquí propuesto oscila entre cero y uno.

V varía de acuerdo con el tiempo según la ecuación diferencial,

$$\dot{V} = -\alpha(I)V + 100s\delta(t - t_s),$$

donde α es la rapidez con que el potencial de la neurona tiende al estado quiescente. Su valor se obtiene directamente a partir de la ecuación diferencial 1.1, haciendo un desarrollo a primer orden para V de la serie de Taylor suponiendo que n, m y h toman los valores estacionarios y, por lo tanto, dependientes del voltaje,

$$\alpha(I) = 0.675 + 0.00639I.$$

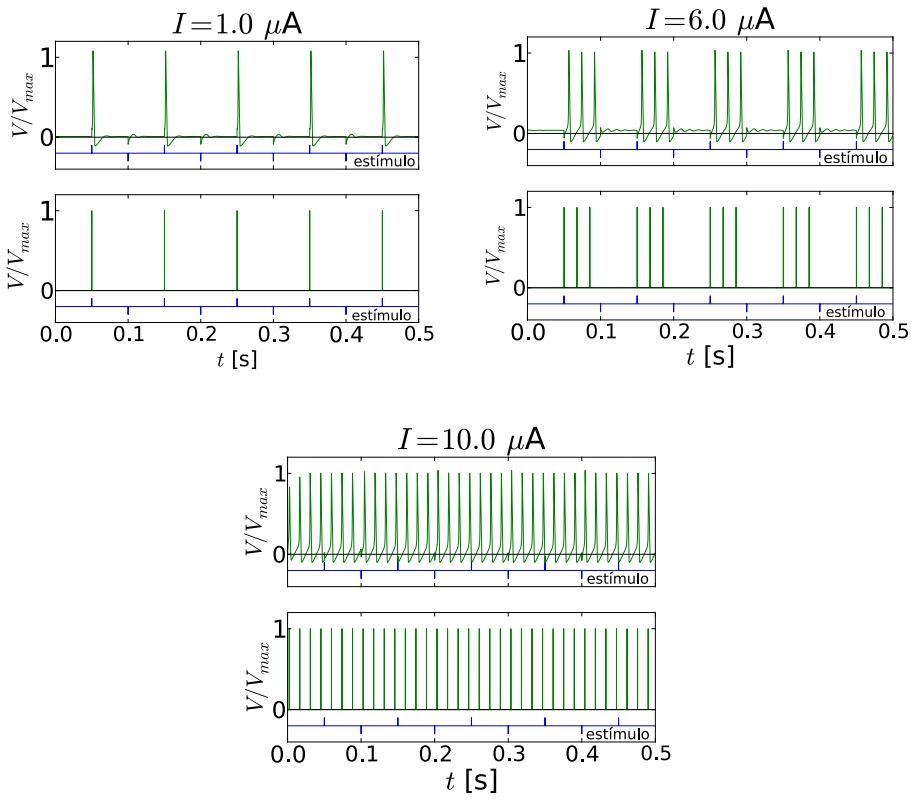

Figura 1.12: Similitudes de la respuesta ante un tren de estímulos entre el modelo de HH y el que aquí se presenta para diferentes valores de la corriente. En la parte superior se muestra el valor del potencial de la membrana celular en el modelo HH, en la parte inferior se compara con el modelo presentado.

Por último, V y φ serán restablecidos a cero después de que la célula propaga el disparo, es decir, tanto la fase como el potencial regresan a su valor base si $\psi = 1$.

Aquí concluye la presentación y descripción del modelo de neurona que hemos conformado y que se empleará como aproximación para hacer simulaciones y otros cálculos. En lo siguiente se mostrará el comportamiento y se comparará con el modelo de HH y con algunas otras aproximaciones.

1.4.1. Comparaciones del modelo simplificado de neurona

Este modelo de neurona conserva algunas de las propiedades más básicas de las neuronas que se han observado experimentalmente y, aunque aparente muchas complicaciones, permite hacer cálculos significativamente más rápidos comparado con el modelo de HH, como se puede apreciar en la fig. 1.12, en donde se muestra la respuesta a un tren de estímulos por el modelo de HH y el modelo simplificado que aquí se presenta.

Más aún, si se hace un promedio temporal de la respuesta de la neurona ante diferentes valores en la amplitud de trenes de estímulos, se puede apreciar la similitud que tiene con respecto a los modelos más simplistas de neuronas empleados para resolver problemas difusos llamados perceptrones [1, 10–13], véase fig. 1.13. En estos modelos la variable de respuesta, también llamada variable de actividad, comúnmente está dada por

$$\psi(s) = \frac{1}{1 + e^{-a(s-h)}}. \tag{1.13}$$

Sin embargo, este tipo de modelos no conservan propiedades oscilatorias nativamente, es decir, efectos como la sincronización y el amarramiento de fase no emergen naturalmente. El modelo que se propone en este capítulo pretende conservar dichas propiedades. [9, 14]

Las células nerviosas están conectadas con otras 10^4 neuronas, y reciben estímulos que se traducen en una corriente total. Dado que estamos interesados sólo en estudiar el comportamiento de un circuito en específico con un número muy limitado de neuronas, simularemos la actividad externa al asignar un valor distinto de cero a la corriente total de cada neurona. El motivo de esto, como se ha descrito a lo largo del capítulo, es que las neuronas tienen comportamientos muy variados que no deseamos perder al no contar con la llegada de tantos estímulos.

Una vez conformada y descrita la unidad fundamental, se procederá con

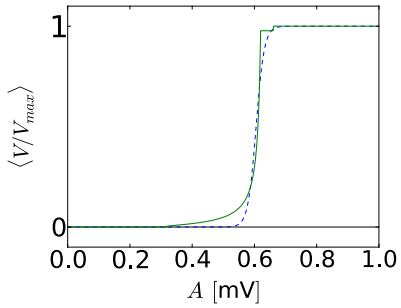

Figura 1.13: Promedio temporal de la actividad (línea sólida) y la actividad según la ecuación 1.13 (línea punteada) para $a = 762.3$ y $h = 0.0605$ según la ec. 1.13

la construcción de conjuntos organizados de ellas que estarán conectadas a manera de constituir una red. Una neurona completamente aislada no puede procesar de señales ni realizar funciones o, por lo menos, no habría manera de saber de su actividad. Al formar grupos y conectándose adecuadamente, tal como se puede apreciar en los seres dotados de sistema nervioso, emerge una gran variedad de comportamientos que dan lugar, por ejemplo, a las funciones vitales, arco-reflejos, etc. En analogía con las componentes electrónicas de una computadora, que conectándolas de manera sumamente específica es que una computadora puede realizar la infinidad de tareas que se le asignan.

Capítulo 2

La red de neuronas

En breve, las neuronas, las más de las veces, transmiten los impulsos eléctricos a lo largo del axón desde la base del axón, llamado cono axónico, al otro extremo del axón llamado botón presináptico. Los iones de calcio al final del axón, movidos por la diferencia de potencial, incitan a las vesículas presinápticas aproximarse a la membrana de la célula para derramar los neurotransmisores en la brecha sináptica. Una vez secretados, el botón postsináptico, lleno de receptores, captura estas moléculas y permite o evita el flujo de iones al interior de la célula. Más tarde, en la célula postsináptica, el paso de los iones se traduce en un cambio de voltaje transmitido por las dendritas al cuerpo celular [4]. Este proceso puede ser reducido a la conducción del voltaje de una célula a la siguiente. A continuación se detallará el modelo que se empleó para estudiar los efectos de este proceso.

2.1. El modelo de la red

Para comprender el modelo que se propone de red de neuronas, es preciso describir cómo es que una neurona se conecta al resto. Una neurona recibe estímulos que son colectados por las dendritas dando un valor a la variable de activación para más tarde, la célula envía su respuesta o manifiesta su actividad vía el axón y propagandose a las demás neuronas.

En general, para una neurona, el potencial en el cuerpo celular se modelará como una combinación lineal de los potenciales presinápticos. Los coeficientes de las combinaciones son los pesos de conexiones entre ambas neuronas. La actividad de la neurona postsináptica ψ_i será una función de la variable de activación s_i, el cual está dada por la suma pesada de todos los potenciales que

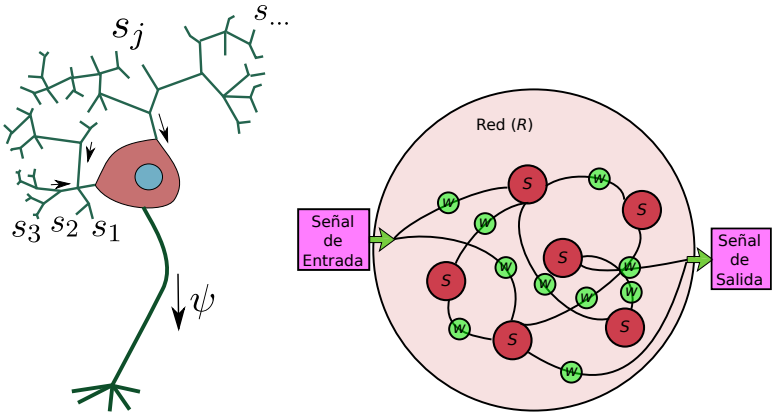

Figura 2.1: (Izquierda) Esquema de una sola neurona recibiendo un conjunto el impulsos s_i a través de las dendritas donde más tarde serán integradas al cuerpo celular de tal manera que se produzca una respuesta ψ propagada por el axón. (Derecha) Esquema de la red de neuronas, conectadas entre ellas y a las señales de entrada y de salida. La red tiene la virtud de traducir un estímulo de entrada a una señal de salida y, por lo tanto, puede ser interpretada como un mapeo. Los círculos con la letra S denotan al soma de una neurona y los círculos con una w en su interior denotan un peso sináptico.

recibe:

$$\psi_i(s_i), \qquad s_i = \sum_j w_{ij}\psi_j, \tag{2.1}$$

donde w_{ij} representa el peso de conexión o **peso sináptico** de la neurona j con la i y ψ_i es la actividad de la neurona i, es decir, la respuesta recibida de la neurona postsináptica (fig. 2.1 izquierda) que en este caso se modela con la neurona simplificada descrita en el capítulo anterior (ec. 1.12). Las constantes w_{ij} pueden interpretarse como las entradas de una matriz w, conocida como la **matriz de adyacencia**. Dado que las conexiones entre las células son unidireccionales, la matriz de adyacencia no será en general simétrica,

$$w_{ij} \neq w_{ji}.$$

En las neuronas observadas en el laboratorio se ha visto que el axón es eléctricamente muy resistivo y, por lo tanto, la manera en que se transmite un pulso *no* es meramente mediante el campo eléctrico, en su lugar, ocurre un proceso muy complejo de movilización de iones. Este proceso produce un ligero retraso temporal de la generación del potencial de acción en el cono axónico a la

llegada de éste en botón de la dendrita de la neurona presináptica [4]. Naturalmente, el retraso depende de la longitud del axón, de tal manera que en axones suficientemente cortos puede aproximarse a ser instantáneo. En nuestro caso estudiaremos redes pequeñas de relativamente pocos nodos. Aunque no se incluye una noción de distancia física para el modelo, se presupone que los axones son muy cortos y, por lo tanto, la transmisión es considerada instantánea.

Se modela al sistema como una gráfica simple[1], es decir, a lo más hay una arista que conecta a dos nodos y un nodo no se conecta consigo mismo. Se ha observado que una neurona puede estar conectada con otra en varios puntos [4]. Sin embargo, esto puede ser modelado sin pérdida de generalidad mediante una sola conexión pesada. Por otro lado, a pesar de que se ha observado que las neuronas en escasas ocasiones se conectan consigo mismas, debido a que la transmisión es instantánea, se presupone que en caso de que se presente una conexión de este estilo, no tendrá gran efecto. En conclusión, los elementos sobre la diagonal de la matriz de adyacencia se fijan a cero.

Un grupo de neuronas puede formar una red a medida que se forman conexiones entre ellas. Aunado a esto, estamos interesados en el efecto que produce la estimulación a una red, es decir, la respuesta que produce la red ante estimulación específica. Los estímulos serán enviados a través de canales de entrada, esto es, nodos que únicamente envían señales de control al resto de la red. De manera similar, la respuesta de la red se transmite por un conjunto de nodos que únicamente reciben señales. En la fig. 2.1 (derecha) se esquematiza la estructura de la red.

Podemos dar una generalización de los pesos sínapticos mencionados anteriormente en la cual se incluyan los pesos de conexión de la señal de entrada con las neuronas y también la conexión de las neuronas con la salida. Entonces, una red de N neuronas que recibe una señal de entrada de l canales y una señal de salida de m canales, la matriz de adyacencia w será de $(N+m)\times(N+l)$. Por supuesto, la variable de activación preserva forma dada en ec. 2.1, donde la suma corre de 1 a $N+l$ y la salida de la red se interpreta como un grupo más de nodos; por lo tanto, ψ_j con $j = N+1, ..., N+m$ es la salida de la red. La matriz de adyacencia se puede visualizar de la sigiente manera

$$w = \left(\begin{array}{c|c} \text{INTER} & \text{ENTRADA} \\ \hline \text{SALIDA} & 0 \end{array}\right), \tag{2.2}$$

donde INTER son las conexiones interneurales, es decir, únicamente las neuronas de la red consigo mismas, ENTRADA son las conexiones eferentes y

[1]La noción de gráfica no debe ser confundida con la representación de n-adas ordenadas en un espacio, más bien se refiere a un diagrama en el cuál se representa a un conjunto de objetos como nodos y su relación como enlaces.

SALIDA son las conexiones aferentes.

2.2. La actividad de la red neuronal

La red de neuronas, al recibir estímulos y enviar señales en respuesta, puede ser interpretada como un mapeo

$$n_w : \mathbb{R}^l \to \mathbb{R}^m,$$

donde n_w es la actividad o respuesta de la red, naturalmente dependiente de las conexiones entre las neuronas.

Por otro lado, podemos pensar en otro mapeo, de igual manera

$$\tau : \mathbb{R}^l \to \mathbb{R}^m,$$

que llamaremos **función objetivo** o meta. El nombre de ésta se debe a que da una referencia a la actividad de la red, o bien, es la función que se desea imitar por parte de la red. Es posible calcular la diferencia entre ambos mapeos E de diversas maneras, una de ellas puede ser a partir de la norma vectorial, esto es,

$$E_w(s) := |\tau(s) - n_w(s)| .$$

donde $E(s)$ será el **error** entre el objetivo y la actividad de la red neural en dependencia del estímulo recibido, $|\cdot|$ es una norma que se puede escoger arbitrariamente.

Los estímulos, en general, pueden variar con el paso del tiempo, $s = s(t)$. La respuesta de la red, debido a la naturaleza de sus componentes constitutivos, fluctuará inclusive si la estimulación es constante. Es posible que se desee calcular el error con respecto al promedio temporal de la actividad de la red, considerando promedios sobre intervalos de tiempo muy largos. Dicha cantidad se presume invariante ante la asignación del origen temporal, esto es, del estado inicial de la red.

Por otro lado, dado que el modelo de neurona empleado es binario, es muy posible contar también con estímulos binarios. Eso implica que puede haber un conjunto finito de combinaciones de estímulos para la red, $\{s_i : i = 1, 2, ..., L\}$.

Habiendo dicho esto, podemos reajustar la definición de error a una que considera promedio temporal y un conjunto finito de estímulos diferentes,

$$\begin{aligned} E(w) &:= \frac{1}{L} |\vec{\tau} - \vec{n}_w| \\ &= \frac{1}{L} \sum_{i=1}^{L} |\tau(s_i) - n_w(s_i)| . \end{aligned} \tag{2.3}$$

Al hacer un listado de todas las respuestas posibles de la red se puede generar una matriz de $m \times L$. En particular, si la respuesta de la red es de un solo canal, $m = 1$, luego entonces, se interpreta como un vector de L entradas, estas siendo $n_w(s_i)$.

El error es un parámetro escalar siempre positivo, $E(w) \geq 0$, la igualdad se mantiene cuando la actividad de la red es idéntica a la función objetivo para todos los estímulos.

En ocasiones esta noción de error, aunque intuitiva, puede ser muy restrictiva dado que exige perfecta concordancia entre ambas funciones. Por tal motivo, las más de las veces, emplearemos otra definición de error que mide la "ortogonalidad" de las señales de salida y la función objetivo. Con otrogonalidad quiere decir que si las dos funciones, la respuesta de la red y la función objetivo son paralelas, esencialmente son la misma función, burdamente dicho, se mide cualitativamente la diferencia entre las funciones.

Podremos tener, de esta manera, una medida del error a partir de la ortogonalidad entre la respuesta de la red y el objetivo. Éste se puede definir con el producto exterior[2],

$$\begin{aligned} E(w) &:= |\hat{\tau} \times \hat{n}|^2 \\ &= 1 - (\hat{\tau} \cdot \hat{n})^2 \\ &= 1 - \left[\sum_{i=1}^{L} \hat{\tau}(s_i) \hat{n}_w(s_i) \right]^2 , \end{aligned} \tag{2.4}$$

donde $\hat{\tau}$ y $\hat{n}$ representan a vectores unitarios generados por las L-adas $\tau(s_i)$ y $n_w(s_i)$.

En ocasiones, será más apropiado utilizar una definición de error que otra. A lo largo de este trabajo se emplearán principalmente las dos definiciones, ecs. 2.3 y 2.4, dependiendo de la situación. La última de estas (ec. 2.4) propone varias ventajas:

1. El error está contenido en el intervalo $[0, 1]$.

2. Toma la clase de equivalencia de los vectores αn_w, con $\alpha \in \mathbb{R}$, y los reduce a un solo vector. Por lo tanto, la medida de la diferencia entre una función y otra está enfocada a la magnitud relativa entre las señales de respuesta en lugar de la magnitud absoluta de éstas.

[2]Producto exterior, también conocido como producto vectorial o producto cruz; aunque éste debe ser extendido a las dimensiones que sean necesarias, por lo que conviene definirlo con el símbolo de Levi-Civita, i.e. $A \times B = \sum \epsilon_{ijk} A_j B_k$.

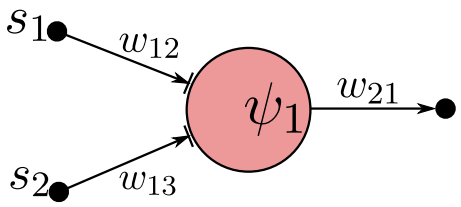

Figura 2.2: Esquema de la conexión de una sola neurona con la señal de entrada de dos canales (s_1 y s_2) y su salida. También se muestran los pesos sinápticos correspondientes que actuan sobre las señales.

3. Si $n_w(s_i) = 0,\ \forall i$, entonces el error resulta ser la unidad, lo cual es un extremo. Esto evitará que la trayectoria en el espacio fase se estanque en cuencas durante el proceso de aprendizaje, el cual será mencionado más adelante.

Una vez dada la definición de error es posible medirla para diversas configuraciones y funciones objetivo. En esa exploración se pueden encontrar funciones en las que el numero de redes con error cercano a cero es muy alto y otras ocasiones en las que la función objetivo es muy compleja y no existe configuración alguna que permita un error arbitrariamente bajo si se cuenta con un número limitado de neuronas para la red. Esto nos lleva a plantear la pregunta, ¿qué funciones son realizables por una red de un número fijo de neuronas y con qué precisión? O bien, una interrogante similar, ¿cuál es el número mínimo de neuronas que pueden realizar una función con un grado arbitrario de precisión?

En lo que respecta a esta tesis, se dará un intento o una aproximación para dar pie a una respuesta. Por lo pronto, se presentarán algunos ejemplos de redes realizando algunas de las funciones binarias más simples.

2.2.1. Ejemplo de una red conformada por una sola neurona

Para esta subsección mostraremos con tres simples ejemplos las condiciones necesarias para que una red realice una función específica. Consideremos a una red compuesta por una sola neurona. Buscamos encontrar los pesos de las conexiones de ésta con las señales de entrada y salida tal que el error sea nulo tal como se muestra en la fig. 2.2. Dado que trabajaremos con una sola neurona, para el promedio temporal podemos aproximar la actividad de la neurona con

la función escalón:

$$\psi(s) = \theta(s - h).$$

Para este sistema contaremos con una señal de entrada de dos canales y una señal de salida de un canal. La función objetivo será la compuerta lógica "OR", la tabla de valores es,

$$\begin{aligned}(0,0) &\mapsto 0,\\ (1,0) &\mapsto 1,\\ (0,1) &\mapsto 1,\\ (1,1) &\mapsto 1.\end{aligned}$$

La respuesta de la red es

$$\bar{n}_w(s) = w_{21}\theta(w_{12}s_1 + w_{13}s_2 - h),$$

donde $w_{2,1}$ es el peso de conexión con el canal de salida y $w_{1,\alpha}$, con $\alpha = 2, 3$, los pesos sinápticos de conexión con la señal de entrada.

El error que emplearemos para esta ocasión será el definido en la ec. 2.3:

$$\begin{aligned}E_w(0,0) &= |w_{21}\theta(-h)|\,,\\ E_w(1,0) &= |1 - w_{21}\theta(w_{12} - h)|\,,\\ E_w(0,1) &= |1 - w_{21}\theta(w_{13} - h)|\,,\\ E_w(1,1) &= |1 - w_{21}\theta(w_{12} + w_{13} - h)|\,.\end{aligned}$$

El error, siendo el promedio de las cuatro ecuaciones anteriores, será igual a cero si se cumple que $w_{21} = 1$ y $w_{2,\alpha} > h$. Como se puede apreciar, sólo son tres parámetros que deben ser ajustados. En la fig. 2.3 se puede observar cómo es el paisaje de error para esta función booleana.

No está de más recalcar el hecho de que si $h < 0$ no existe combinación alguna para w de tal manera que permita $E_w(0,0) = 0$. En cuyo caso estaríamos hablando de una neurona en modo marcapasos que no requiere estimulación adicional para la producción de potenciales de acción. Para las pruebas que haremos posteriormente, encontraremos ocasiones en las que las neuronas que forman la red no son aptas para realizar ciertas funciones.

Para una red de N neuronas, se pueden presentar hasta $N(N-1)$ conexiones entre ellas. Si el estímulo de entrada consta de l canales se tendrán a lo más lN conexiones y, análogamente, mN si la salida es de m canales. Esto hace un total de $N(N+l+m-1)$ pesos sinápticos en total. Como se puede apreciar, el espacio fase, o el espacio de las variables, crece cuadráticamente en su dimensionalidad con el número de neuronas. En general, para una red arbitrariamente grande,

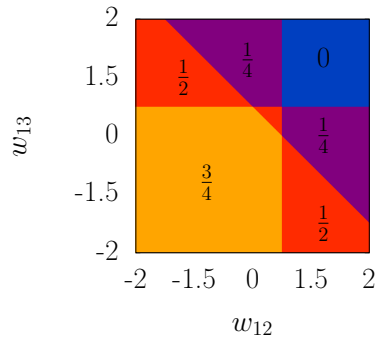

Figura 2.3: Paisaje de error para la función booleana "OR" con el error definido en ec. 2.3. El valor del error se indica dentro de las áreas resaltadas en la imagen de la derecha.

encontrar los pesos de las conexiones que satisfagan $E(w) = 0$ es una tarea muy difícil y recurriremos a varios métodos para encontrar la solución.

Con el propósito de ejemplificar un poco más lo que ocurre al proponer diferentes funciones objetivo se mostrará el procedimiento para la compuerta lógica "AND"con la misma definición de error que el ejemplo anterior. En ese caso, las cuatro combinaciones de señal de entrada producen la siguiente respuesta de la función objetivo:

$$\begin{aligned}(0,0) &\mapsto 0,\\ (0,1) &\mapsto 0,\\ (1,0) &\mapsto 0,\\ (1,1) &\mapsto 1.\end{aligned}$$

El cálculo del error muestra que

$$\begin{aligned}E_w(0,0) &= |w_{21}\theta(-h)|\,,\\ E_w(1,0) &= |w_{21}\theta(w_{12}-h)|\,,\\ E_w(0,1) &= |w_{21}\theta(w_{13}-h)|\,,\\ E_w(1,1) &= |1-w_{21}\theta(w_{12}+w_{13}-h)|\,.\end{aligned}$$

Lo cual implica que el error global será cero si se cumple que $h > 0$, $w_{21} = 1$, $w_{12} < h, w_{13} < h$ y $w_{12} + w_{13} > h$.

Comparando las condiciones donde $E(w) = 0$ de las dos funciones, podemos observar que los requerimientos del "AND" son más restrictivos que los del "OR",

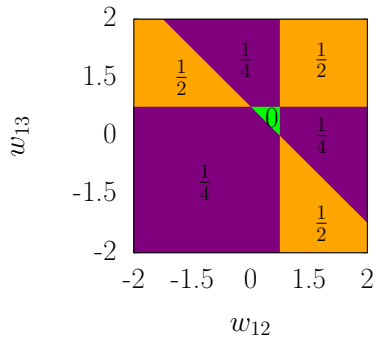

Figura 2.4: Paisaje de error para la función booleana "AND" de una red de una neurona y el error medido según ec. 2.3. Los valores del error están señalados el la imagen de la derecha.

es decir, el volumen del espacio fase donde el error es cero es menor en el "AND" que en el "OR". En la fig. 2.4 se muestra el paisaje de error para esta función objetivo.

Podríamos pensar también en otras funciones booleanas, por ejemplo el "XOR". Sin más detalles y mediante un procedimiento análogo al anterior, podemos ver que las condiciones son: $h > 0$, $w_{21} = 1$, $w_{12}, w_{13} > h$ y $w_{12}+w_{13} < h$. Es evidente que si se cumplen las primeras, la última es imposible de cumplir. Podemos concluir entonces que para una red de una neurona, el "XOR" es imposible de lograr. Para mayor claridad, en la fig. 2.5 se muestra el error para todos los valores de los pesos sinápticos de la señal de entrada.

Para funciones de dos variables binarias que devuelven un valor binario es fácil probar cuáles se pueden emular con una red de una neurona. Considérese el plano real, donde los cuatro puntos $(0,0)$, $(0,1)$, $(1,0)$ y $(1,1)$ son resaltados. Cuando la función objetivo mapea dichos puntos, dos regiones son identificadas, las que son mapeadas a cero y las mapeadas a uno. Si dichas regiones se pueden dividir por una única recta entonces es posible asegurar que una neurona podrá emular la función sin error alguno. En caso contrario, será necesaria la intervención de más de una neurona para que dicha red pueda emular a la función objetivo [1, 13].

Hemos mostrado con estos tres ejemplos que aún para funciones muy sencillas no siempre existe modo de conectar a la señal de entrada y la de salida para que la red realice idénticamente una función específica. A la vez se ha

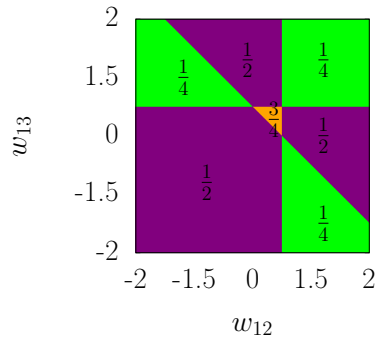

Figura 2.5: Paisaje de error para la función booleana "XOR" de una red de una neurona y el error medido según ec. 2.3. Los valores del error están señalados en las áreas correspondientes el la imagen de la derecha.

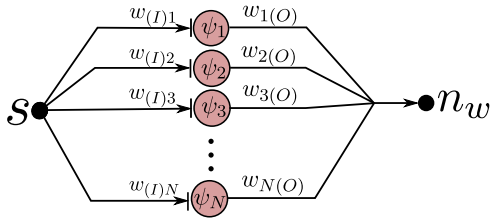

Figura 2.6: Esquema de la arquitectura de la red de neuronas no interactuantes entre sí. Por comodidad se distinguen dos tipos de pesos sinápticos etiquetados con (O) para salida y (I) para entrada.

mostrado la forma del paisaje de error, aún para estos casos sencillos puede dar intuición para casos más complejos.

Con la intención de mostrar la capacidad que tienen las redes de neuronas, se mostrará a continuación otro ejemplo.

2.2.2. Redes de neuronas no interactuantes entre sí

Podemos generalizar el caso para funciones objetivo de una variable continua en un conjunto de los números reales, extrapolando a su vez la definición del error global. Contando únicamente con redes en las cuales haya un número arbitrario de neuronas pero con la restricción de no permitir conexiones entre ellas, es decir, $w_{ij} = 0$ para $i, j = 1, 2, ..., N$ es posible probar que el error se puede disminuir arbitrariamente. Qué tan pequeño se quiera el error depende

del número de neuronas y de la correcta elección de los pesos sinápticos.

Podemos extrapolar la definición del error para variables continuas de la siguiente manera:

$$E(w) = \frac{1}{V}\int_V |\tau(s) - \bar{n}_w(s)|\, ds,$$

donde s pertenece a un conjunto finito contenido en $\mathbb{R}$ de medida V.

Para probar esta afirmación, comenzaremos por mostrar el comportamiento para un canal de salida y otro de entrada, es decir, para funciones de $n_w : A \to \mathbb{R}$, con $A \subset \mathbb{R}$ un intervalo continuo. Probaremos esta afirmación para neuronas cuya actividad es la función escalón de Heaviside. Para una red de N neuronas, podemos particionar el intervalo en N partes y construir una colección de números $\{s_i\}$, donde

$$s_i = \text{ínf}(A) + \frac{i}{N}|A|, \quad i = 0, 1, 2, ..., N;$$

y $|A|$ denota la medida de A.

Además, podemos formar otra colección de puntos,

$$\{\tau_i = \tau(s_i) : i = 0, 1, 2, ..., N\}.$$

Para una colección de N neuronas, cada una con umbral h_i, la actividad de cada neurona será,

$$\psi_i(s) = \theta(w_{(I)i}s - h_i) = \begin{cases} \theta(s - \frac{h_i}{w_{(I)i}}) & \text{si } h_i > 0 \\ \theta(\frac{h_i}{w_{(I)i}} - s) & \text{si } h_i < 0 \end{cases}.$$

Es importante separar a las neuronas en dos conjuntos: las neuronas de umbral positivo y las de umbral negativo. Para las de umbral positivo podemos ajustar w_i con una cantidad positiva, en caso de que el umbral sea negativo, $w_i < 0$. Por lo pronto, supondremos que todas las neuronas tienen umbral positivo.

La intención es que cada neurona comience a disparar ordenadamente de tal manera que se pueda cubrir todo el intervalo. Para esto, es necesario que el argumento de la función escalón, en la actividad de la neurona i, sea positivo para una $s > s_i$, es decir,

$$s_i > \frac{h_i}{w_{(I)i}}.$$

De esta manera podemos ajustar el peso de conexión del canal de entrada al valor

$$w_{(I)i} = \frac{h_i}{s_i}.$$

Para ajustar el valor del peso de conexión del canal de salida buscamos hacer que n_w en cada s_i sea igual a τ_i. Dado que las neuronas activadas ya sostienen un cierto valor de señal para el canal de salida, basta con ajustar al incremento del siguiente punto en la partición. Por lo tanto,

$$w_{i(O)} = \tau_i - \tau_{i-1},$$

con la excepción de $\tau_0 = 0$ para corregir en $i = 1$. De esta manera se puede garantizar que $n_w(s_i) = \tau_i$ para $i = 1, 2, ..., N$.

Si las neuronas tienen umbral negativo, entonces el procedimiento es análogo; sin embargo, el orden de las particiones de A se debe hacer de derecha a izquierda, es decir,

$$\{s_i = \sup(A) - \frac{i}{N}|A|\},$$

lo cual tendrá un efecto análogo al mostrado anteriormente.

Finalmente, en caso de contar con neuronas, tanto de umbral negativo como positivo, basta con ordenarlas según este parámetro de menor a mayor de tal manera que se cubra la primera parte del intervalo con las de umbral negativo y la última parte con las de umbral positivo. Inclusive se debe tener cuidado si el intervalo A contiene al cero, en cuyo caso se pueden hacer dos particiones, la que cubra a la parte negativa y la que cubra a la positiva de tal manera que

$$n_w(s) = \tau_i \text{ si } s_i < s \leq s_{i+1}.$$

El error de la red está dado por

$$\begin{aligned} E(w) &= \frac{1}{|A|}\int_A |\tau(s) - n_w(s)|ds \\ &= \frac{1}{|A|}\sum_{i=1}^{N}\int_{s_{i-1}}^{s_i} |\tau(s) - \tau_i|ds. \end{aligned}$$

Para una τ integrable, por definición, la suma tiende a cero conforme la partición del intervalo es más refinada. En otras palabras, E puede hacerse arbitrariamente pequeña si N es suficientemente grande. En la fig. 2.7 se muestra esquemáticamente la convergencia de la red de neuronas a la función objetivo conforme el número de células es mayor.

Para neuronas cuya actividad es más suave como en la aproximación 1.13, la afirmación sigue siendo válida. El teorema en el que se sostiene esta afirmación es llamado el teorema de Cybenko [17].

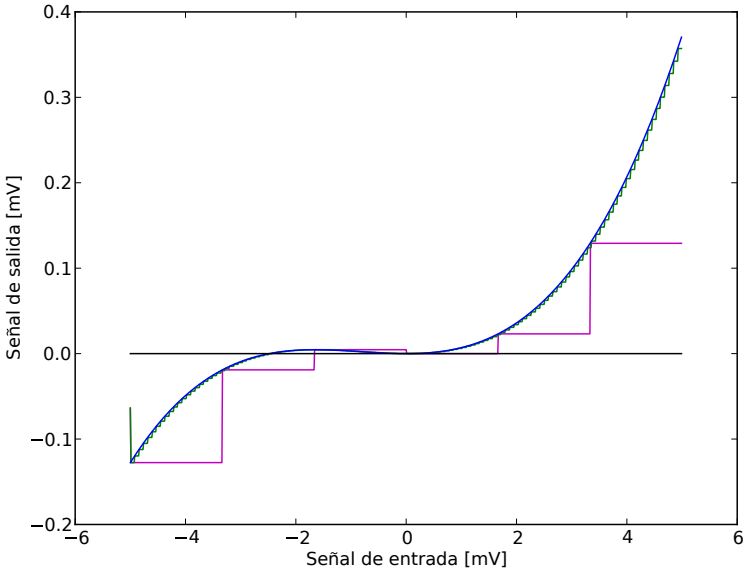

Figura 2.7: Proceso de convergencia de una función objetivo de variable continua (línea azul) por una red compuesta de 6 (línea magenta) y 128 neuronas (línea verde).

Dado que impedir la conexión entre las neuronas es muy restrictivo, es de esperarse que al remover esta restricción, una gran variedad de fenómenos van a tener lugar y posiblemente se pueda decrecer significativamente el error. La interacción entre neuronas es no-lineal y de la misma manera no podemos dar *a priori* una aproximación del comportamiento en general. La tarea de encontrar los pesos sinápticos adecuados para redes no es trivial y recurriremos a métodos computacionales que nos lo permitan.

En este capítulo hemos mostrado la manera en que se constuirán las redes de neuronas. También se ha presentado una manera de comparar funcionalmente a la red con alguna actividad deseada. De igual manera se mostró cómo es que mediante la correcta elección de los pesos sinápticos la red es capaz de realizar diversas funciones. No obstante, encontrar los pesos sinápticos adecuados no siempre será sencillo. En el capítulo siguiente indagaremos acerca de cómo encontrar los pesos sinápticos correctos y de esa manera conseguir que la red de neuronas pueda reproducir fielmente una función particular.

Capítulo 3

Los algoritmos de aprendizaje

En este capítulo se muestran dos algoritmos desarrollados para la obtención de pesos sinápticos adecuados para las redes de neuronas. Biológicamente, este proceso está asociado con el proceso mismo de aprendizaje, naturalmente este proceso es altamente complejo ya que involucra un sinnúmero de factores; no obstante, aquí presentamos un par de mecanismos relativamente simples pero que pueden dar una idea del proceso de aprendizaje en general.

La variación temporal de la conductancia del axón presináptico, de la brecha sináptica y de la dendrita postsináptica es llamada plasticidad y depende de varios factores. Por ejemplo, se ha mostrado que en la potenciación y depresión a largo plazo la concentración de los iones de calcio en la vecindad de la brecha sináptica juega un papel de suma importancia y la manera en la que cambia a partir de la manera en que disparan la célula presináptica y la postsináptica [4].

Se han construido una gran cantidad de modelos que emulan esta dinámica. Dicha variación de los pesos sinápticos tiene un gran efecto en la respuesta de la red y se cree que es la responsable del proceso de la memoria, fenómenos cognitivos, etc [2–4].

Inspirados en esa afirmación, una de las hipótesis que adoptaremos en este trabajo es que no son las neuronas por sí mismas, sino la manera en cómo están conectadas lo que le permite a un circuito operar de alguna manera en particular. Entonces, será la correcta elección, si es que existe alguna, de los pesos sinápticos entre las neuronas lo que produzca la respuesta buscada.

Si se compara con una función objetivo, el error cambiará durante dicha dinámica. El proceso de disminución de error será entonces el aprendizaje y diremos que una red ha aprendido si $E(w) = 0$.

En general, hay dos tipos de aprendizaje, supervisado y no supervisado. En el aprendizaje no supervisado, una red encuentra la manera en que debe operar mediante procesos internos. En el aprendizaje supervisado, existe una

interacción de la red con la función meta, con el error o, en general, algún agente externo que influye en la variación de los pesos sinápticos.

Tanto en el cerebro humano como en el de muchos otros seres vivos, se han identificado circuitos evolutivamente adaptados para realizar funciones específicas, por ejemplo, la red de neuronas que marca el ritmo cardiaco, la que regula el ritmo circadiano o en general circuitos que regulan procesos muy básicos, involuntarios o instintivos que no han sido aparentemente aprendidos de o gracias a agentes externos. Bajo esta observación surgen un par de incógnitas más: ¿qué mecanismos son los que gobiernan el aprendizaje de dichos circuitos? y ¿El aprendizaje *no* supervisado es mecanismo suficiente y necesario para que dichos circuitos evolucionen hasta alcanzar una función específica?

Estas preguntas siguen abiertas y en el presente trabajo no se indagará más al respecto; más bien, se ha escogido el aprendizaje supervisado como objeto de estudio.

3.1. El algoritmo tipo Monte Carlo

Probablemente, uno de los mecanismos más simples para disminuir sistemáticamente el error global de una red con respecto a una función particular es el de *prueba y error* y *sin experiencia*. El que no se forme experiencia es que la historia del sistema, en este caso la red de neuronas, no influye en la toma de decisiones futuras o discriminación alguna de los estados visitados en el espacio fase. Hemos implementado un algoritmo inspirado en el método Monte Carlo, el cual se detalla a continuación. El algoritmo consta de la siguiente estructura:

1. Una red es propuesta al azar. Las conexiones entre neuronas, así como el de los canales de entrada y salida, son asignados de manera aleatoria y las propiedades de los somas son fijados.

2. Una función objetivo es escogida, así como una medida del error.

3. El error es evaluado.

4. Se propone un ligero cambio en la red, el cual puede ser la inauguración de una conexión, su clausura o simplemente un ligero cambio fijado en un principio al valor del peso de alguna conexión escogida al azar para los índices i o j de la matriz de adyacencia.

5. El error global es nuevamente calculado y comparado con el error previo al cambio.

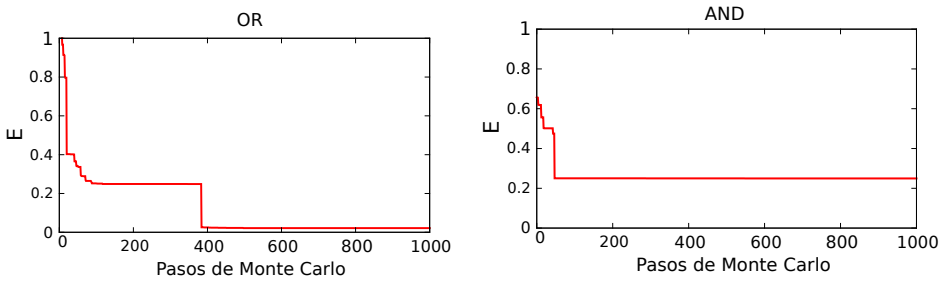

Figura 3.1: Gráficas del aprendizaje supervisado para funciones booleanas básicas. Aquí se muestra la disminución del error con el número de iteraciones del algoritmo de Monte Carlo a $\beta^{-1} = 0$.

6. Si el error resulta ser mayor al anterior, el cambio sólo se aceptará con probabilidad p, donde
$$p = \exp(-\beta \Delta E).$$
Aquí $\Delta E = E_f - E_i$ es la diferencia entre los errores posterior y previo al cambio propuesto y β es un parámetro ajustable que mide qué tan permitivo es el algoritmo ante incrementos en el error. Cuando se presentan muchos valles en el error es recomendable asignar en β valores bajos, por el contrario, si se sabe que la red se encuentra en un estado cercano al de cero error, es recomendable que β tenga valores muy altos o inclusive infinito. La forma de la probablilidad de aceptación p fue escogida arbitrariamente.

7. Se repiten desde el paso cuarto al sexto hasta alcanzar un cierto número de pasos o un error tolerablemente pequeño lo que detiene el algoritmo.

Este algoritmo ha probado ser relativamente eficazn, como se verá en el capítulo de resultados más profundamente. Dado que el error, como función de los pesos de conexión, en general será una hipersuperficie compuesta por muchos valles y crestas, basar el algoritmo de aprendizaje enteramente en el gradiente del error no tendría gran éxito, a menos que el estado se encuentre dentro de la cuenca de atracción de $E(w) = 0$. Por encima de esto, dado que es un mecanismo muy primitivo y simple, es muy probable que se puedan encontrar algunos muy similares en la naturaleza.

El algoritmo también mostró eficacia para redes con un amplio número de neuronas y para ciertas funciones lógicas binaras. En la figura 3.1 se puede apreciar el proceso de aprendizaje de la compuerta lógica “OR”. Sin embargo, pierde eficacia en algunos casos. Es también común encontrar procesos de aprendizaje

que no convergen pronto, es decir, debe pasar un número muy grande de iteraciones para que la red aprenda. Un ejemplo de ello fue el aprendizaje de la compuerta lógica "AND", dado que para ésta, sólo una de las cuatro respuestas que debe dar la red es uno y el resto es cero, si hay una respuesta distinta de cero para los demás estímulos, el error inicialmente será alto. El algoritmo rápidamente encuentra que desconectar la señal de salida disminuye significativamente el error; no obstante, esto logra estancar el aprendizaje por un gran número de pasos. Este es un ejemplo de un mínimo local muy profundo, pero no lo suficientemente profundo tal que tenga un error tolerable.

Con base en lo anterior, implementamos otro mecanismo de aprendizaje, el cual probará ser más eficiente para paisajes de error escalonados o que las trayectorias en el espacio fase encuentren obstáculos significativos antes de reducir al mínimo el error.

3.2. El algoritmo genético

Se implementó un tipo de algoritmo genético con doble intención. Primero quisimos encontrar eficazmente los pesos de conexión de las redes que emulan arbitrariamente bien una función, y segundo, un algoritmo que tenga un correlativo biológico o que modele lo mejor posible los mecanismos evolutivos.

En general, un algoritmo genético está compuesto por una población y un mecanismo evolutivo que actúa sobre la población. La población está constituida por un conjunto de individuos y los individuos son aquellos objetos que son modificados con el fin de hallar la solución a un problema; por ejemplo, en este caso, los individuos representan las matrices de adyacencia; los pesos sinápticos son modificados con la intención de que la red pueda emular con la mayor precisión posible una función en específico. Dentro de cada individuo se encuentran las variables que son modificables por el algoritmo. Estos son los **genes**, los cuales forman estructuras y compiten los unos con los otros para formar nuevos individuos; en nuestro caso cada peso sináptico estará representado por un gen.

En mayor detalle, el mecanismo evolutivo de un algoritmo genético se conforma de la siguiente manera:

1. *Inicialización.* Una población es propuesta de manera aleatoria, esto es, cada uno de los genes de los individuos de la población son inicializados con valores escogidos al azar homogéneamente en el intervalo $[-2, 2]$.

2. *Adaptación*, en ocasiones llamado mutación, es el proceso en el cual los genes son alterados o asignados nuevos valores con el fin de hacer al

individuo más apto para desempeñar una tarea o una función. Para el proceso de adaptación hemos empleado el método Monte Carlo descrito anteriormente. Con el paso del tiempo, es decir, con las iteraciones, los genes son alterados o no. A la par de cada gen, en una nueva variable, se almacena la edad del gen, esto es, el tiempo que ha transcurrido sin que se presente un cambio en éste, y en el momento que el algoritmo altera el gen, la edad de éste es restablecida a cero.

3. *Selección natural.* Después de un cierto número de pasos de adaptación, los individuos mejor adaptados son escogidos, esto es, sólo una proporción de las redes neurales con el menor error podrán continuar en el algoritmo, el resto es eliminado de la población. Por encima de esto, a cada individuo le es asignada una probabilidad p_{sel} para conformar una pareja, en nuestro caso,

$$p_{\text{sel}} = c \exp(-\alpha \bar{K}),$$

donde $\bar{K}$ es el número promedio de vecinos por nodo en la red, c es una constante de normalización y α es una constante que se fija al comienzo del algoritmo cuya función hacer un contraste entre los individuos. Esta probabilidad hace a ciertos individuos más "atractivos" que otros con el fin de promover en la población un uso modesto en el número de conexiones que a su vez se asocia al consumo de nutrientes de la red.

4. *Apareamiento y recombinación.* Los individuos supervivientes al formar parejas son apareados y sus genes son recombinados y heredados. El mecanismo de herencia consta de seleccionar genes de la pareja de individuos, los genes son dispuestos ordenadamente para competir, es decir, el gen de un padre asociado con un peso sináptico competirá con el correspondiente de la pareja. La manera en que los genes compiten es mediante un proceso de selección. Los genes dominantes tienen mayor probabilidad de ser seleccionados que los genes recesivos, la probabilidad

$$p_{\text{rec}} = c\sqrt{a+b},$$

donde a es la edad del gen, b es una constante fijada al comienzo del algoritmo y c es la constante de normalización. En caso de que alguno de los padres no tenga el gen, $a = 0$ y la competencia es para el gen en cuanto a ser heredado o no. Si ambos padres no tienen un mismo gen, la probabilidad de que se forme uno nuevo es del 1 %.

5. *Finalización.* El mecanismo se repite desde el segundo al quinto paso hasta que una mayoría de la población esté lo suficientemente adaptada, o bien, si se llega un número previamente fijado de repeticiones.

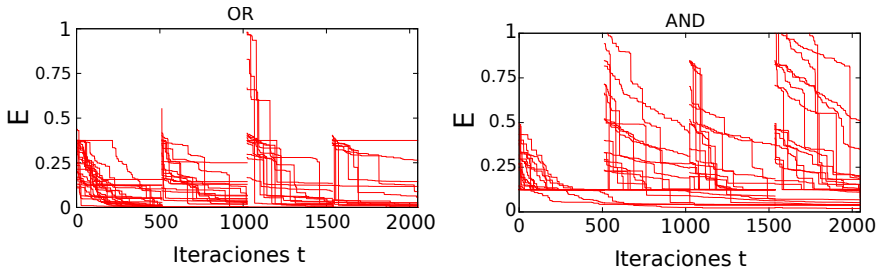

Figura 3.2: Evolución de los individuos sometidos al aprendizaje supervisado mediante el algoritmo genético para funciones booleanas básicas. En ambas figuras se puede observar cómo el error de los individuos disminuyen su error conforme se da el proceso de adaptación. Al término de éste, entra el mecanismo de recombinación que substituye a una gran cantidad de individuos, los menos adaptados. Los nuevos individuos, aunque comienzan con un error mayor al de sus semejantes, aprenden con mayor rapidez y disminuyen su error hasta superar a los miembros menos afortunados de la población.

Este algoritmo presenta dos ventajas muy importantes: además de que es eficaz para hallar redes que emulan una función con una fidelidad relativamente alta, devuelve varias versiones de redes muy diferentes entre ellas pero de una calidad similar.

La figura 3.2 es una gráfica del error de los individuos según del número de iteraciones del proceso de adaptación del algoritmo genético. Se puede observar cómo una pequeña proporción del total tiene un error bajo y persiste en el corte de generación en el cual se recombinan los individuos. Más tarde éstos, comenzando con un error alto, rápidamente se incorporan al conjunto de individuos de bajo error para continuar el aprendizaje.

El algoritmo genético complementado con el método de Monte Carlo fue empleado para estudiar algunas propiedades de las redes. Además, ambos algoritmos por sí mismos resultan interesantes, pues podrían abrir camino para la respuesta de algunas de la preguntas abiertas planteadas a lo largo de esta tesis.

Capítulo 4

Estudio estadístico del aprendizaje

En este capítulo daremos un breve análisis del algoritmo de Monte Carlo para el aprendizaje de las redes de neuronas. El análisis tiene la finalidad de dar un mejor entendimiento de la manera en que se lleva acado el proceso de aprendizaje. A la vez puede proveer de las herramientas necesarias para optimizar en ciertos casos o permite reconocer los factores que fomentan o truncan el aprendizaje.

Hace más de treinta años Rosenblatt sugirió el primer modelo de una máquina con la capacidad de aprender y la llamó perceptrón; éste fue el momento en el que el análisis matemático del aprendizaje realmente comenzó. Desde el punto de vista conceptual, la idea del perceptrón no era nueva, sin embargo, Rosenblatt la introdujo como un programa para computadora que podía ser generalizado. El perceptrón fue construido para resolver problemas de reconocimiento de patrones, en particular, para el caso más sencillo de discernir los datos en dos rubros.

En 1967, Novikoff probó que el algoritmo de Rosenblatt converge en un número infinito de pasos si el conjunto de datos son linealmente separables. Este teorema dió inicio a la teoría del aprendizaje en redes neurales, pues da una cota para el número de correcciones que una red neural conlleva durante su entrenamiento, es decir, el máximo número de pruebas que la red neural puede llegar a necesitar para funcionar adecuadamente [18].

Para el sistema que se está presentado en esta tesis, trataremos de dar un tratamiento similar e intentaremos describir el proceso de aprendizaje así como dar cotas necesarias para que una red pueda realizar la función deseada.

Consideremos un colectivo[1] de redes de neuronas. Cada réplica está conformadas por distintos valores de los pesos sinápticos, es decir, las distintas rea-

[1]Nos referimos a un colectivo a la usanza de la mecánica estadística, en ocasiones referido en francés como *ensemble*.

lizaciones cubren todos los estados accesibles de las conexiones entre ellas. De esta manera podemos dar una definición de densidad de probabilidad $p(E)dE$ de encontrar a una red neuronal con error E como el número de redes en el ensamble con error entre E y $E + dE$ respecto del número total; dicho más precisamente, el volumen ocupado en el espacio fase de redes con error entre E y $E + dE$ sobre el volumen total accesible. Es de nuestro interés conocer la evolución de las redes sujetas al aprendizaje. Por lo tanto, estamos en busca de la dinámica de la distribución del error como función del número de pasos del algoritmo de aprendizaje.

4.1. La dinámica de la probabilidad: ecuación maestra

Dado que en el algoritmo de aprendizaje no hay memoria, podemos estudiarlo como un proceso markoviano. Por lo tanto, partiremos de la **ecuación maestra** como la que gobierna la dinámica de la probabilidad.

Considérese la ecuación maestra,

$$\frac{\partial}{\partial t} f(x,t) = \int \left[f(x',t) W(x' \to x) - f(x,t) W(x \to x') \right] dx', \tag{4.1}$$

la cual describe el cambio en el tiempo de una distribución de probabilidad f; t es el tiempo, x es el evento y $W(x \to x')$ es la tasa para hacer un salto del sitio x al sitio x'. El primer término del segundo miembro cuantifica el aumento de la probabilidad en el sitio x, suponiendo que la probabilidad aumenta según el flujo de probabilidad de todos los sitios que concluyen en x. De manera análoga, la probabilidad disminuye si hay flujo del sitio x a algún otro estado del sistema, el cual es cuantificado por el segundo término.

Para analizar el ensamble de redes de neuronas, la distribución de probabilidad f será la probabilidad $p(E,t)dE$ al tiempo t mencionada anteriormente. Dada la naturaleza del algoritmo de aprendizaje, es necesario discretizar el tiempo. La distribución f será la densidad de probabilidad $p(E,t)$ después de t pasos de Monte Carlo, por lo tanto,

$$\frac{\partial f}{\partial t} \to \frac{\delta p}{\delta t} = \frac{1}{\delta t}[p(E, t + \delta t) - p(E,t)].$$

La ecuación maestra se reescribe de la siguiente manera:

$$p(E, t + \delta t) = p(E,t) + \delta t \int \left[p(E',t) W(E' \to E) - p(E,t) W(E \to E') \right] dE'. \tag{4.2}$$

Esta version se utilizó para resolver la ecuación por métodos numéricos.

Según el algoritmo de aprendizaje implementado, la probabilidad de transición de un estado con error E a uno con error E' se puede separar en el producto de dos probabilidades: una propia del sistema y la otra del algoritmo de aprendizaje,

$$W(E \to E') = g(E'|E)A(E \to E').$$

La probabilidad $g(E'|E)$, que llamaremos distribución de accesibilidad, representa la probabilidad de encontrar un estado E' dado que el sistema se encuentra en el estado E, esto es, la probabilidad de que una red cambie a un error E' si se encuentra con error E. La probabilidad $A(E \to E')$ es la razón de aceptación para un cambio de E a E', el cual es propio del algoritmo de aprendizaje.

Daremos una aproximación para la distribución de accesibilidad de la siguiente manera. Considérese al paisaje de error como función de los pesos sinápticos. La red se encontrará en una configuración específica a la cual se le asocia un error $E = E(w)$, en función de los pesos sinápticos. El algoritmo propone un ligero cambio en los pesos sinápticos w dado por δw, entonces, suponiendo que E es continua y de derivada continua,

$$\begin{aligned} E(w + \delta w) &= E(w) + \nabla E|_w \cdot \delta w + O(\delta w^2), \\ E' &= E + \delta E. \end{aligned} \tag{4.3}$$

Considérese el volumen de la variedad creada por todos los puntos con error E, el cual llamaremos $\omega(E)$. Por construcción, la densidad de estados $\rho(E) = \omega(E)/V_{\text{tot}}$, donde V_{tot} es el volumen total del espacio fase; en la fig. 4.1 se pueden apreciar ejemplos de las densidades de estados para diferentes funciones. Entonces, si hay un cambio en w, el error cambiará por ϵ. Esto crea una familia de volúmenes alrededor del isocontorno formado por E constante. El volumen generado $V(\epsilon)$ por la colección de puntos será,

$$\begin{aligned} V(\epsilon) &= \int_{E-\epsilon}^{E+\epsilon} \omega(E')dE' = \Omega(E+\epsilon) - \Omega(E-\epsilon) \\ &\simeq \Omega(E) + \omega(E)\epsilon - (\Omega(E) - \omega(E)\epsilon) \\ &= 2\omega(E)\epsilon. \end{aligned}$$

Donde $\Omega(E)$ es la primitiva de $\omega(E)$. Por lo tanto, la probabilidad de que haya un cambio de tamaño ϵ está dada por

$$2\frac{\omega(E)}{V_{tot}}\epsilon = 2\rho(E)\epsilon,$$

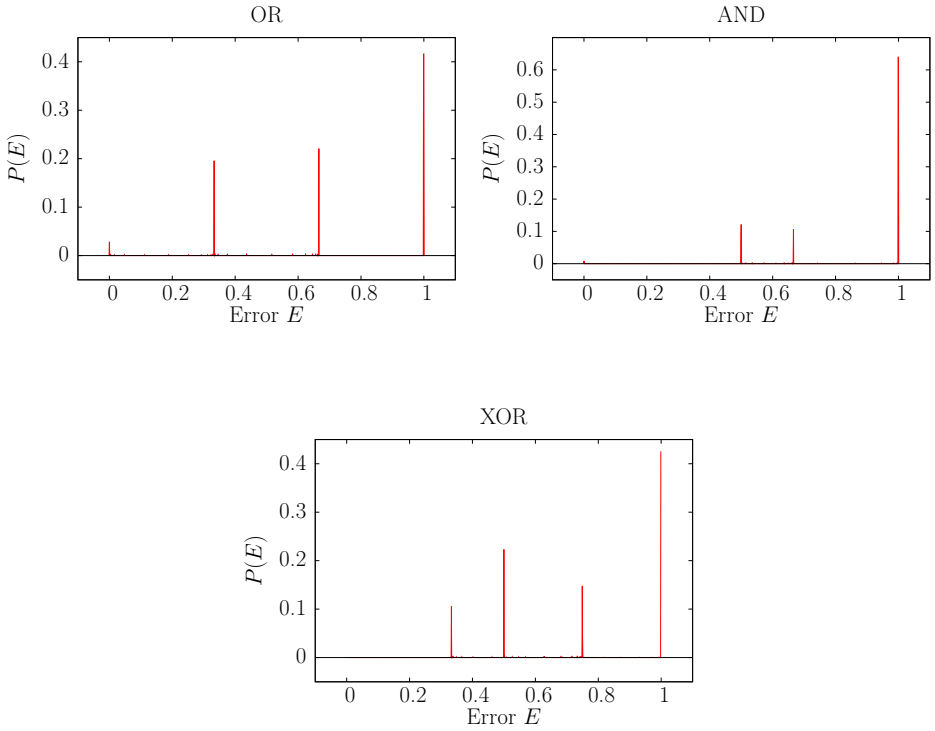

Figura 4.1: Densidades de estados para redes compuestas por una neurona emulando las funciones "OR", "AND" y "XOR" en izquierda, centro y derecha respectivamente. La medida del error está definido en la ec. 2.4. Nótese cómo la accesibilidad de estados comienza a ser reducida escalonadamente conforme el error disminuye.

para valores de ϵ dentro de la cota establecida por la magnitud del cambio en los pesos sinápticos admisible. En caso de que ϵ esté por encima de dicha cota, la probabilidad se anula, por lo tanto,

$$g(E'|E) \propto \begin{cases} \rho(E) & \text{si } |E' - E| \leq \delta E_{\text{máx}} \\ 0 & \text{si } |E' - E| > \delta E_{\text{máx}} \end{cases},$$

donde $\delta E_{\text{máx}}$ es el máximo cambio en promedio. Si bien $\delta E_{\text{máx}}$ en principio depende de la forma de E como función de w (ec. 4.3); sin embargo, por razones de simplicidad, la supondremos constante.

Si se busca una mejor aproximación, este análisis puede extenderse para diferentes distribuciones en los valores de δw. En el caso descrito anteriormente,

se presupone una distribución homogénea, sin embargo, es posible hacer la generalización a diferentes distribuciones de δw. No se indagará más en los cálculos, sin embargo, por superposición basta con multiplicar la distribución de probabilidad de los cambios en w, es decir, en general,

$$g(E'|E) \propto \rho(E) r(E' - E),$$

donde $r(E' - E)$ es la distribución de probabilidad de un cambio en el error de tamaño ϵ. Otra manera de obtener la distribución de accesibilidad es numéricamente mediante la construcción de un histograma de los posibles valores a los cuales cambia el error a E' en cada E.

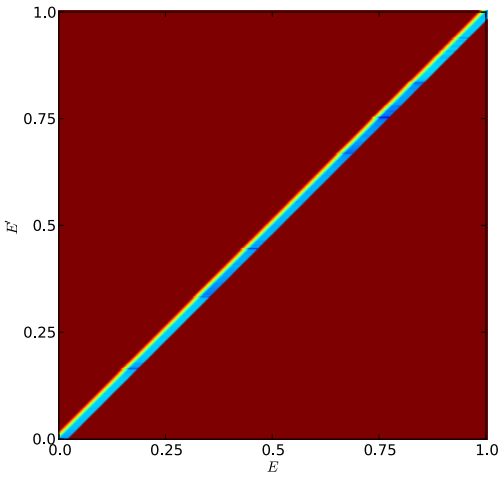

Figura 4.2: Gráfica de contorno de un ejemplo de la función $W(E \to E')$.

La razón de aceptación $A(E \to E')$ está regida por el algoritmo de aprendizaje. La forma de esta distribución, aunque trata de asemejarse a los procesos biológicos, está determinada por el mecanismo de aceptación de la condición de **balance detallado** del algoritmo de Monte Carlo,

$$A(E \to E') = \text{mín}\left\{1, e^{-\beta(E'-E)}\right\} + \gamma(E)\delta(E - E'), \tag{4.4}$$

donde el último término en el miembro derecho corresponde a la probabilidad de permanecer con la misma E propiciado por rechazo al cambio según el algoritmo. Por lo tanto, se escoge $\gamma(E)$ para que $\int A(E \to E')dE' = 1$, i.e.,

$$\gamma(E) = 1 - E + \frac{1}{\beta}[e^{-\beta(1-E)} - 1]. \tag{4.5}$$

Es prudente mencionar que este término no aparece normalmente en la literatura y puede sorprender a muchos lectores, la razón de ello es porqué no se habla de la probablilidad de aceptación como una probabilidad de transición de un estado a otro. En nuestro caso, la probablilidad de aceptación se está empleando como una parte de la probablidad de transición de la ecuación maestra y por conservación de la probabilidad es necesario hacer la corrección.

Finalmente la probabilidad de transición de probabilidad para el cambio de un estado con error E a uno con error E' se aproxima con

$$W(E \to E') = \alpha(E) g(E'|E) A(E \to E').$$

donde $\alpha(E)$ es una función que se ajusta de tal manera que se cumpla

$$\int W(E \to E') dE' = \nu \qquad \forall E, \tag{4.6}$$

donde ν es una constante, para la conservación de la probabilidad.

Si $\delta E_{\text{máx}}$ es muy pequeña, $W(E \to E')$ cada vez es más estrecha, en el límite

$$\lim_{\delta E_{\text{máx}} \to 0} W(E \to E') = \nu \delta(E - E').$$

4.2. Ecuaciones de campo medio y Fokker–Planck

A la vez de estudiar la dinámica de la probabilidad $p(E,t)$ sujeta al proceso de aprendizaje, es igualmente importante el cálculo de la dinámica más probable, esto es, el valor esperado. En general, el valor esperado está definido de la siguiente manera,

$$\langle x \rangle (t) = \int x f(x,t) dx.$$

La evolución en el tiempo está dada por,

$$\begin{aligned}
\frac{d}{dt}\langle x \rangle &= \int x \frac{\partial f}{\partial t} dx \\
&= \int x \left[\int \{ f(x',t) W(x' \to x) - f(x,t) W(x \to x') \} dx' \right] dx \\
&= \int \int (x' - x) f(x,t) W(x \to x') dx' dx \\
&= \int a_1(x) f(x,t) dx = \langle a_1(x) \rangle ,
\end{aligned}$$

donde se define

$$a_n(x) := \int (x' - x)^n W(x \to x') dx'.$$

Desarrollando en serie de $a_1(x)$ hasta segundo grado alrededor de un punto x_0,

$$a_1(x) \simeq a_1(x_0) + \partial_x a_1|_{x_0}(x - x_0) + \frac{1}{2}\partial_x^2 a_1|_{x_0}(x - x_0)^2.$$

Entonces, el valor esperado se puede reescribir

$$\langle a_1(x)\rangle \simeq a_1(x_0) + a_1'(x_0)\langle x - x_0\rangle + \frac{1}{2}a_1''(x_0)\left\langle (x - x_0)^2\right\rangle,$$

donde a_1' denota la primera derivada de a_1 y a_1'' denota la segunda derivada. Si se establece $x_0 = \langle x\rangle$, la ecuación se reduce a

$$\frac{d}{dt}\langle x\rangle = a_1(\langle x\rangle) + \frac{1}{2}a_1''(\langle x\rangle)\Delta x^2.$$

De esta ecuación, los términos de primer orden,

$$\frac{d}{dt}\langle x\rangle = a_1(\langle x\rangle), \tag{4.7}$$

constituyen la aproximación conocida como la **aproximación de campo medio**.

Si se desea hacer una mejor aproximación, es necesario conocer la dinámica de la varianza Δx^2. Para ello se considera la ecuación diferencial correspondiente:

$$\frac{d}{dt}\Delta x^2 = \frac{d}{dt}(\left\langle x^2\right\rangle - \langle x\rangle^2) = \frac{d}{dt}\left\langle x^2\right\rangle - 2\langle x\rangle\frac{d}{dt}\langle x\rangle.$$

Mediante un proceso similar, el término cuadrático puede ser expresado de la siguiente manera,

$$\frac{d}{dt}\left\langle x^2\right\rangle = \langle a_2(x)\rangle + 2\langle x a_1(x)\rangle.$$

Simplificando la expresión anterior, la derivada temporal de la varianza se puede aproximar a lo siguiente,

$$\frac{d}{dt}\Delta x^2 = a_2(\langle x\rangle) + 2a_1'(\langle x\rangle)\Delta x^2.$$

En resumen, se tiene el siguiente sistema de ecuaciones:

$$\dot{\langle x\rangle} = a_1(\langle x\rangle) + \frac{1}{2}a_1''(\langle x\rangle)\Delta x^2, \tag{4.8}$$

$$\dot{\Delta x^2} = a_2(\langle x\rangle) + 2a_1'(\langle x\rangle)\Delta x^2. \tag{4.9}$$

Siguiendo este procedimiento es posible hacer cada vez un mejor refinamiento en la dinámica del valor esperado.

El análisis anterior puede ser aplicado a una red de neuronas cuyo error inicial es E_0, en cuyo caso es de nuestro interés conocer la dinámica de ésta. Dado que hemos hecho una aproximación estadística, por medio de este análisis, es posible calcular la evolución más probable.

Si una red comienza con un error E_0, estadísticamente corresponde a un ensamble donde sólo se dispone de aquellas réplicas con error entre E_0 y E_0+dE, lo cual implica que la distribución inicial es

$$p(E, t = 0) = \delta(E - E_0).$$

Por consiguiente, podemos hacer uso de la ec. 4.7 o de las ecs. 4.8 con una mejor aproximación para las condiciones iniciales,

$$\begin{aligned} \langle E \rangle (0) &= E_0, \\ \Delta E^2(0) &= 0, \end{aligned}$$

para calcular la evolución más probable del sistema.

Este mismo análisis puede extenderse para derivar la ecuación de Fokker–Planck [19,20]. La ecuación de Fokker–Planck se expresa de la siguiente manera,

$$\frac{\partial f}{\partial t} = -\frac{\partial}{\partial x}\left[a_1(x) f(x,t)\right] + \frac{1}{2}\frac{\partial^2}{\partial x^2}\left[a_2(x) f(x,t)\right],$$

donde $a_i(x)$ son los definidos anteriormente. Genéricamente, en la fig. 4.3 se muestra el tipo de dinámica que se espera observar en la evolución de $p(E, t)$.

Dado que trabajamos mayormente con simulaciones computacionales para estudiar la dinámica probabilística de las redes, la ecuación maestra (ec. 4.1) fue empleada principalmente, dado que se puede trabajar directamente con los datos obtenidos y sin aproximaciones. La ecuación de Fokker–Planck es una aproximación que en nuestro caso poco simplifica los cálculos.

4.3. La solución estacionaria

En caso de que la dinámica de f llegue a un valor constante en el tiempo, el miembro izquierdo de ec. 4.1 se anula. Por lo tanto, en el caso estacionario se debe cumplir la ecuación,

$$\nu\phi(x) = \int \phi(x')W(x' \to x)dx'. \tag{4.10}$$

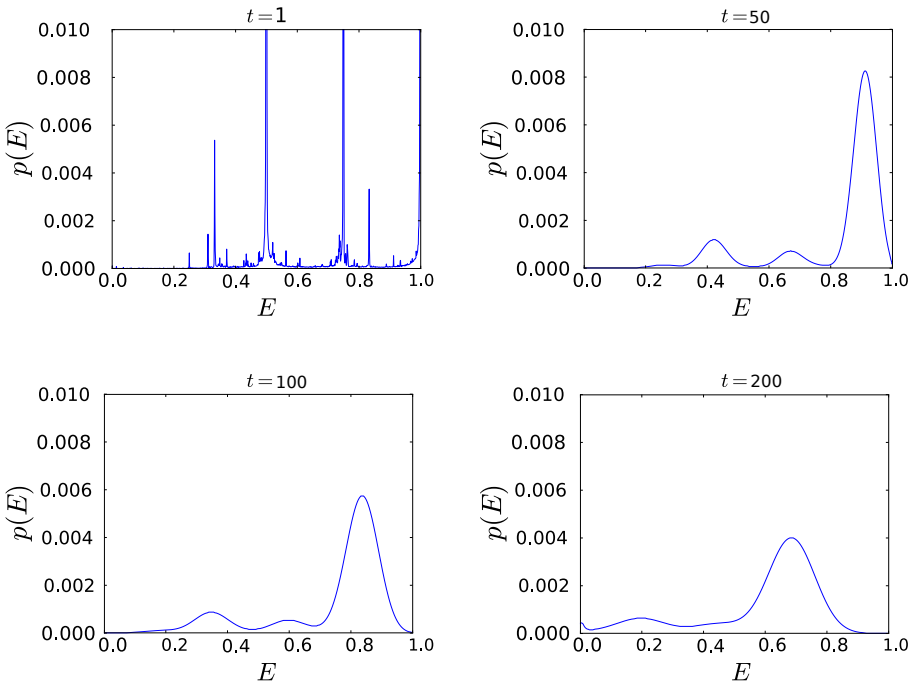

Figura 4.3: Ejemplo de la evolución de la probabilidad $p(E,t)$ para tiempos discretos gobernada por la ecuación de Fokker–Planck.

donde ϕ es la solución estacionaria y ν se presume constante dado por

$$\nu = \int W(x \to x')dx'.$$

Ésta es una ecuación integral conocida como una ecuación homogénea de Fredholm del segundo tipo. Por ser homogénea, $\phi(x) = 0$ es la solución trivial, sin embargo, evidentemente no es la que buscamos, dado que las densidades de probabilidad son soluciones de norma unitaria. Es importante subrayar que la integral debe cubrir únicamente a los puntos accesibles al sistema, es decir, los valores que toma x pueden provenir de un conjunto disconexo. Este hecho influirá en la forma de las soluciones estacionarias.

Por lo general, la solución no es única, aún eliminando la multiplicidad al normalizar las distribuciones. En caso de que el dominio sea conexo, es decir, si todos los estados son accesibles desde cualquier estado particular, podemos asegurar que la distribución final será única y dependerá fuertemente de la forma de $W(x' \to x)$.

En este caso particular, dada la forma de $A(E \to E')$, la probabilidad de aceptación, la cual proviene de la condición de balance detallado en el algoritmo de Monte Carlo, está diseñada para generar la distribución de probabilidad de Boltzmann, es decir,

$$\phi(E) \propto e^{-\beta E}.$$

En caso de haber valores inaccesibles de error para el sistema dentro del intervalo $[0, 1]$, la probabilidad se anula $p(E^*, t) = 0$ para aquellos valores inaccesibles del error E^*, pues no es posible encontrar redes con un error inadmisible. En este caso pueden suceder dos situaciones: si la brecha formada por los puntos de E^* es menor que $\delta E_{\text{máx}}$, $\phi(E)$ tendrá una forma similar a la distribución de Boltzmann, es decir, será una exponencial decreciente salvo en los puntos donde se anula. En caso de que $\delta E_{\text{máx}}$ no sea suficientemente grande, la solución estacionaria tomará la forma

$$\phi(E) = \sum_i \alpha_i \begin{cases} e^{-\beta(E-E_i^*)} & \text{si } E_i^* \leq E \\ 0 & \text{si } E_i^* > E \end{cases},$$

donde E_i^* son los supremos únicamente de los intervalos de medida mayor a $\delta E_{\text{máx}}$ y α_i son constantes que normalizan la solución las cuales dependen de la distribución inicial $p(E, t = 0)$, más precisamente, de la integral sobre cada intervalo de E accesible. En fig. 4.4 se ejemplifican los casos de soluciones estacionarias que se pueden presentar.

Por lo general, este tipo de situaciones no ocurrirán debido a que, por construcción el espacio fase (el espacio de w) es conexo y, por pequeños que sean

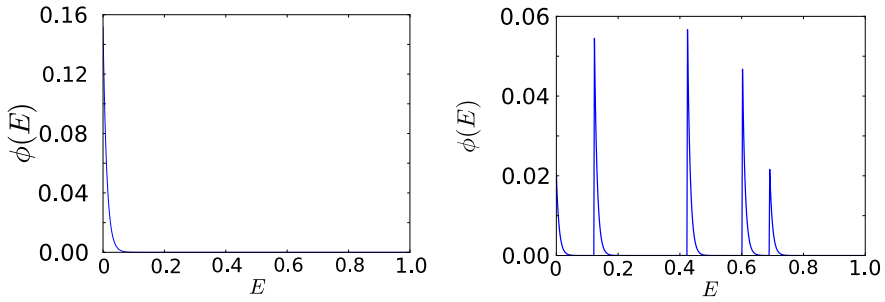

Figura 4.4: Ejemplos de los posibles estados estacionarios de la probabilidad, $p(E, t \to \infty) \to \phi(E)$. *Izquierda.* Cuando la densidad de estados no contiene intervalos nulos de suficiente medida la solución tiende a una exponencial decreciente. *Derecha.* En caso de contener intervalos nulos de suficiente extensión, la solución estacionaria es una suma de exponenciales.

los pasos de Monte Carlo, siempre hay manera de llegar de un lugar a otro. Sin embargo, es pertinente advertir que si la función objetivo es suficientemente compleja de tal suerte que la red no pueda emularla suficientemente bien, los valores inferiores del error estarán desiertos.

4.4. Muerte celular

Recordemos también que la memoria es robusta ante ruido o tolerante ante error; por ejemplo, errores en la petición de un recuerdo son corregibles, es decir, dado un grupo de datos, a pesar de contener errores, un individuo es capaz de recordar e identificar el error. Otro aspecto de la robustez es que la pérdida de células por daño o el malfuncionamiento de unas cuantas neuronas no produce necesariamente el olvido.

Se mencionó en el capítulo 3 como una hipótesis de este trabajo que las neuronas *a priori* no almacenan información, sin embargo, la pérdida de estas células dentro de un circuito naturalmente afectará su respuesta. Sin embargo, es evidente que, en general, se dará lugar a un cambio en el funcionamiento, por sutil que fuese, en caso de haber pérdida de sus componentes. Esto lo podemos asociar a la contribución de cada célula para efectuar la función que desempeña la red. Conocer el efecto que la muerte celular ocasiona en el error de la red equivale a dilucidar la cantidad de información que provee cada neurona para desempeñar su función *en el contexto de la red y del objetivo.*

En la muerte celular, todas las vías de salida de dicha neurona dejan de

transmitir pulsos a otras neuronas, entonces, la muerte de la neurona i equivale a restablecer todos los pesos sinápticos adyacentes a cero tanto de conexión a otras neuronas como al canal de salida, i.e., $w_{ij} = 0$, $j = 1, 2, 3, ..., N, N + 1, ..., N + m$. El efecto que produce esto en el espacio fase es el de proyectar el estado del sistema w a $w' = \{w_{ij} | w_{ij} = 0$ para i fija y $\forall j\}$.

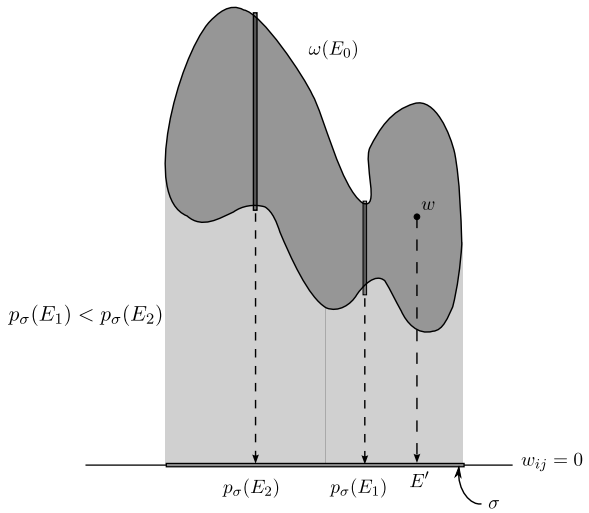

Figura 4.5: Esquema que ilustra la manera en que la variedad $\omega(E_0)$ se proyecta en σ sobre el plano $w_{ij} = 0$. Los puntos de la variedad $\omega(E_0)$ son proyectados al plano $w_{ij} = 0$, y la densidad de estados en σ, p_σ, se calcula a partir de la medida del haz sobre $\omega(E_0)$.

Comencemos por estudiar el efecto que produce el anular un peso sináptico. Supóngase que el error previo a la muerte de la neurona i era E_0. En general, habrá una región en el espacio fase $\omega(E_0)$ que contiene el estado de la red y al de muchas otras con el mismo error. En el momento de fijar a cero el peso sináptico, todos los puntos de error E_0 se colapsan a la proyección formando una nueva variedad, σ, en donde se encontrará el estado de la nueva red. El error más probable E_f con el que acabará la red después de cortar la conexión w_{ij} será entonces,

$$E_f = \frac{1}{|\sigma|} \int_\sigma E dw = \int_E E' p_\sigma(E') dE',$$

donde $p_\sigma(E')$ es la probabilidad de encontrar a una red con error E' dentro de σ y $|\cdot|$ denota medida de la variedad —en este caso es el volumen de

σ. Naturalmente, debido a que σ proviene de una proyección, la distribución de puntos dentro de esa variedad no es uniforme; más bien, en σ habrá una distribución de redes con diversos errores.

Podemos calcular $p_\sigma(E')$ de la siguiente manera. Consideremos a $\omega(E_0)$ como un conjunto de rectas perpendiculares al plano de proyección $w_{ij} = 0$ (i, j fijos). En el momento de la proyección, cada recta es colapsada a un punto. Ese punto tendrá un error asociado E' y, por lo tanto, la cantidad de redes que son reducidas a ese punto es la medida de la recta de la que provenían. Puede haber más de un punto en σ con error E', por lo que es preciso hacer la integral para todos los puntos donde se presente ese error. La probabilidad $p_\sigma(E')$ está dada entonces por

$$p_\sigma(E')dE' = \frac{1}{|\omega(E_0)|}\int_{\omega(E_0)} \delta(E' - E(w_{ij} = 0))dw. \tag{4.11}$$

Para la muerte celular, es necesario cortar todas las conexiones de la neurona muerta. Podemos extender la expresión anterior generalizando a σ como a variedad que se forma después de restablecer $w_{ij} = 0$, con i fija y para toda j; la expresión será idéntica a la ec. 4.11. En fig. 4.5 se ilustra este proceso.

Dado que las neuronas *a priori* son indistinguibles, es conveniente hablar del promedio del error por muerte celular. En promedio, el error que produce la muerte de una neurona será directamente el promedio del error después de la muerte de cada una de las neuronas en la red, dando,

$$E_f = \frac{1}{|\omega(E_0)|N}\sum_i \int_{E'} E' \int_{\omega(E_0)} \delta(E' - E(w_{ij} = 0))dwdE'. \tag{4.12}$$

La expresión anterior puede ser condensada considerando el hecho de que el argumento de la segunda integral se trata de una densidad de estados normalizada exclusiva de la variedad proyectada σ proveniente de la variedad $\omega(E_0)$. Por lo tanto,

$$E_f = \frac{1}{N}\sum_i \int_{E'} E'\rho_{\sigma_i}(E'|E_0)dE'. \tag{4.13}$$

Ahora bien, para redes completamente entrenadas (error cero) podemos evaluar el incremento en el error $\Delta E = E_f - E_0 = E_f$. Por lo tanto,

$$\Delta E = \frac{1}{N}\sum_i \int_{E'} E'\rho_{\sigma_i}(E'|0)dE'. \tag{4.14}$$

Con la expresión anterior se puede ver que, a menos que la suma varíe considerablemente con el número de neuronas de la red, es de esperarse que el

incremento en el error disminuya como el inverso de esta cantidad.

En este capítulo se ha hecho un estudio teórico del comportamiento de las redes de neuronas bajo el algoritmo de Monte Carlo. Aún queda por entender, bajo esta teoría el algoritmo genético; sin embargo, la intención de este capítulo fue la de entender mejor los procesos de aprendizaje y al mismo tiempo presentar una base más formal.

No está de más señalar la importancia de proveer de una noción de la contribución funcional promedio de cada neurona; dicho *a groso modo*, una medida de qué tan valiosos son los componentes de la red para su correcto funcionamiento. De ser posible que esta teoría de esta contribución, se daría un gran avance para dar repuesta a una de las preguntas planteadas anteriormente: "¿Cuál es el número mínimo de neuronas necesarias en la red tal que se pueda realizar alguna función?".

Capítulo 5

Resultados y discusión

En este capítulo se muestran los resultados de algunas pruebas realizadas tanto a las redes de neuronas como a los algoritmos de aprendizaje, así como comparaciones respecto a la teoría estadística del aprendizaje aquí propuesta.

En lo que sigue se empleará la definición de error de la ec. 2.4 que en ocasiones la referimos como el "error de ortogonalidad".

5.1. Algoritmos de aprendizaje

5.1.1. Monte Carlo

Comenzaremos por analizar el mecanismo de aprendizaje inspirado en el método Monte Carlo. Se realizaron algunas pruebas para observar la evolución de las redes con el número de pasos. Para esto se formaron conjuntos de mil redes, todas con valores propuestos aleatoriamente. Una vez formado un grupo, el algoritmo comenzó a adaptar las redes. Una vez que todas las redes evolucionaron un número fijo de pasos, se promedió el error para cada paso, esto es, la evolución del valor promedio del error, también conocido como curva de aprendizaje. Los resultados están graficados en la fig. 5.1.

Como se puede ver, el valor esperado del error varía según la función objetivo. Es notable cómo a las redes les es más fácil imitar el "OR", después el "AND" y por último el "XOR"; es decir, el valor esperado del error decrece más rápidamente con las iteraciones. Por otro lado, nótese que las redes de cuatro neuronas aprenden más rápido que el resto. Esto último es particularmente interesante puesto que implica que, a pesar de ser necesarios más cambios debido al incremento de variables dadas las dimensiones del espacio fase, una trayectoria que se traza en el espacio fase llega a la región de cero error.

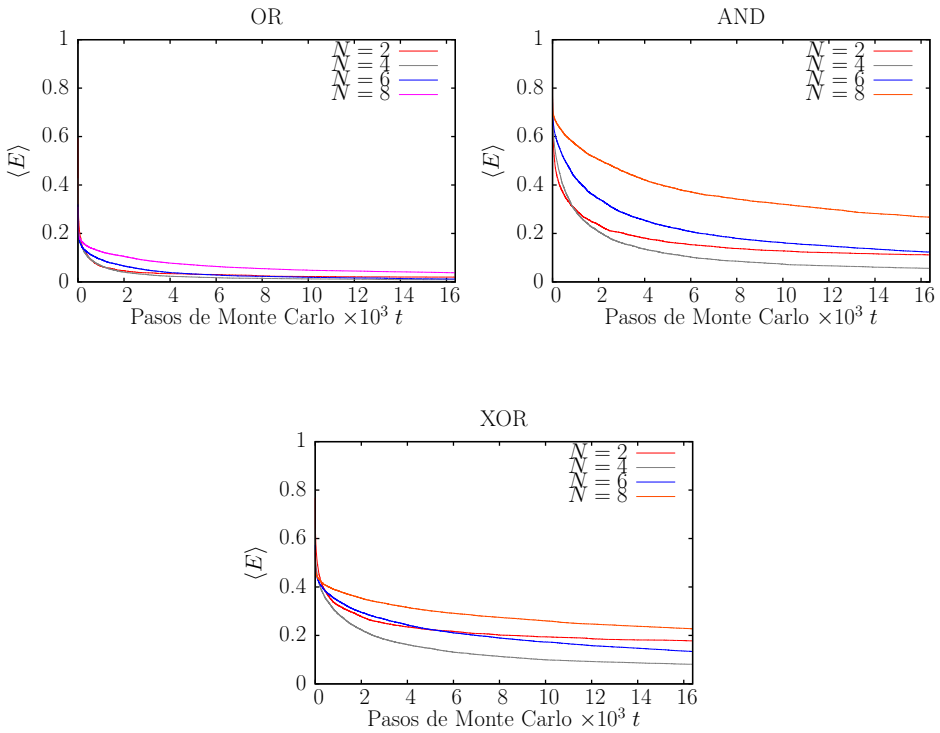

Figura 5.1: Evolución promedio del error para redes compuestas por diferente número de neuronas: 2, 4, 6 y 8. Las funciones objetivo son “OR”, “AND” y “XOR” para izquierda, derecha y abajo respectivamente.

Es importante mencionar que la evolución del promedio del error de las redes no es simplemente una curva exponencial decreciente, como posiblemente se hubiera esperado dada la naturaleza del algoritmo por su analogía con un proceso de equilibración por el contacto con un baño térmico o *reservoir* a baja temperatura. El proceso de aprendizaje es en general un tanto complejo y con la intención de explicarlo fue que se desarrollo el capítulo 4 donde se plantea una teoría estadística para abordar este fenómeno.

5.1.2. Algoritmo genético

La iniciativa de implementar el algoritmo genético fue esencialmente para optimizar el aprendizaje de las redes, en otras palabras, poder obtener la mayor cantidad de redes que emulan arbitrariamente bien una función bajo el menor costo computacional posible. En la presente sección se detallará el comportamiento del algoritmo con la intención de exponer las ventajas o desventajas con respecto a Monte Carlo.

Como se puede observar en fig. 5.2, el algoritmo genético comienza por la adaptación de los individuos haciendo uso del algoritmo de Monte Carlo. Por lo tanto, en un principio, la evolución promedio en éste es idéntico al de una corrida de Monte Carlo sobre una colección de redes independientes. Después de un número designado de pasos, cuando se presenta el corte de generación y los individuos menos adaptados, son reemplazados por combinaciones de los más adaptados, el error promedio aumenta en gran medida. No obstante, los individuos recién gestados muestran una rápida adaptación.

Queda averiguar, bajo qué condiciones, según el tipo de red y la función objetivo, el algoritmo genético supera al de Monte Carlo. Sin embargo, tener datos precisos que muestren cuándo el algoritmo genético es más eficiente que Monte Carlo tuvo un costo computacional muy alto y por ello sólo se mostrarán algunos casos. En la fig. 5.3 se muestra una comparativa para la función "XOR" variando dos parámetros: el número de pasos en el proceso adaptativo antes del corte generacional y la fracción de supervivencia de los individuos.

Como se puede observar en la gráfica de la izquierda de fig. 5.3, si el corte de generación es muy pronto, los individuos no logran adaptarse lo suficiente para engendrar individuos con genes robustos y la evolución del error promedio no decrece eficientemente (línea verde). Al ir incrementando la proporción de individuos que sobreviven el corte de generación, el algoritmo cada vez es más eficiente. Esta tendencia debe llegar a un máximo, dado que si la supervivencia es total o el corte de generación nunca llega el algoritmo es indiferenciable del de Monte Carlo. Una mejora, por sutil que fuese, fue encontrada cuando el 80 % de los individuos sobreviven al corte de generación (línea azul).

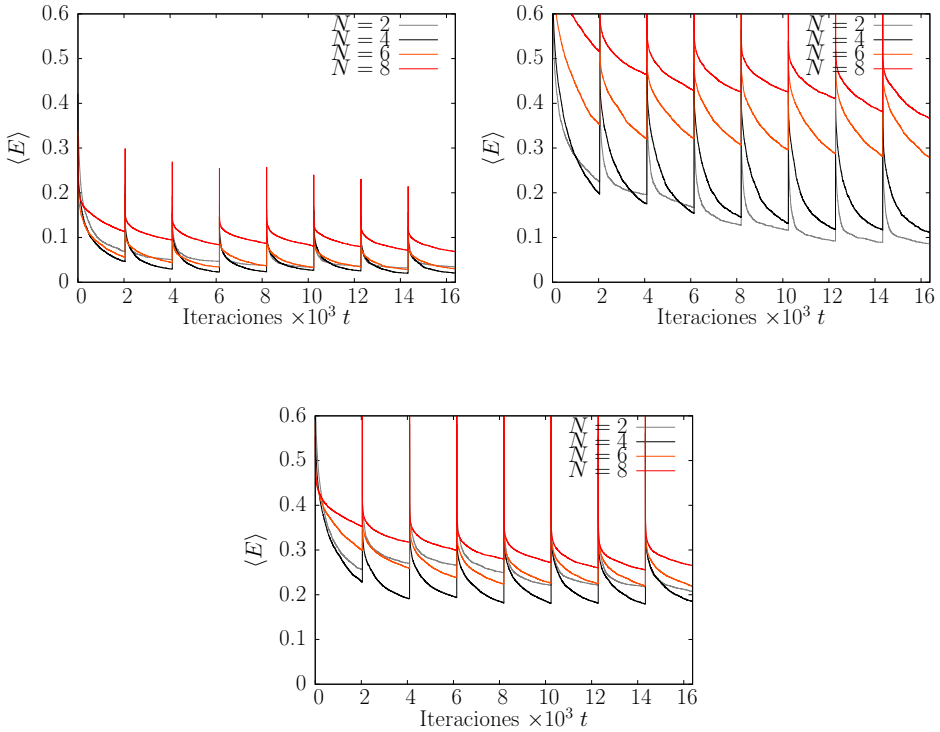

Figura 5.2: Evolución del error promedio para diversas poblaciones de redes en el algoritmo genético. En todos los casos, la población consta de 32 individuos, se realizó un corte de generación cada 2048 iteraciones de Monte Carlo. La tasa de supervivencia fue de 0.3, es decir, sólo el 30 % de la población mejor adaptada continuaba con el proceso de adaptación. Nótese cómo el error promedio de las redes incrementa significativamente después del corte de generación sin embargo, rápidamente disminuye.

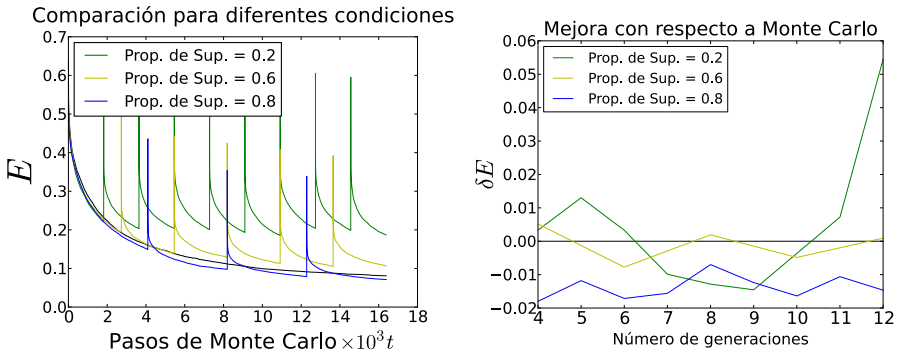

Figura 5.3: En este par de gráficas se muestra la comparación entre la evolución promedio de las redes bajo el algoritmo genético y el de Monte Carlo. En la imagen de la izquierda se hace la comparación para la evolución del error a cada paso. En la gráfica de la derecha se muestra el mínimo de la diferencia entre ambos algoritmos al variar el número total de cortes de generación y la proporción de supervivencia (prop. de sup. como en la leyenda) de los individuos de la población. Como se puede observar, los puntos negativos en el eje de las abscisas muestran ventaja del algoritmo genético sobre el de Monte Carlo siendo menor el error. De este cuadro se puede ver que el algoritmo genético es más eficiente para un número de generaciones entre cinco y ocho, pues para valores mayores o menores resultan muy prematuros los cortes de generación y para valores más bajos no se observa gran distinción de Monte Carlo puro.

En resumen, la gráfica de la derecha de fig. 5.3 muestra el comportamiento. La línea negra, referente al comportamiento del algoritmo de Monte Carlo, permite una comparación con el algoritmo genético; los puntos por debajo muestran mejora dado que el error en promedio es menor.

La disminución del error no muestra una mejora muy significativa. No obstante, el algoritmo genético produce poblaciones significativamente grandes de redes con error tolerablemente bajo. Las redes obtenidas son muy distintas entre ellas lo que permite hacer una exploración amplia de la variedad de la solución del espacio fase.

El algoritmo genético puede ser mejorado de varias maneras, tanto para adecuarlo más a los procesos biológicos como para hacerlo más eficiente.

5.2. Estudio probabilístico del aprendizaje

En esta sección se indagará respecto a la veracidad del tratado propuesto en el capítulo 4. Se intentará dar una explicación al comportamiento observado en ambos algoritmos.

Comenzaremos por mostrar la densidad de estados para redes de sólo dos neuronas. Para obtener estos datos, se procedió de la siguiente manera:

1. Una red es propuesta al azar, esto es, dentro del intervalo $[-2, 2]$ los pesos sinápticos son elegidos de manera aleatoria con una distribución homogénea.

2. El error es evaluado para dicha red.

3. La red es desechada y sólo se conserva el error.

4. Se repiten los pasos 1) al 3) tantas veces como sea posible para poder cubrir relativamente todo el espacio fase de las regiones más significativas.

5. Una vez concluido el escrutinio del espacio fase con una densidad arbitraria de ejemplares, se hace un histograma normalizado, es decir, una aproximación de la densidad de estados.

Naturalmente, para redes de dos neuronas, las tres funciones, "OR", "AND" y "XOR" son aprendibles, es decir, debido a que se presentaron redes cuyo error se puede hacer tan pequeño como se desee. Por esta razón esperamos que la densidad de estados $\rho_i(0) > 0$, para las tres funciones probadas ($i =$"OR", "AND", "XOR"). Por otro lado, cuando las dos neuronas están desconectadas del canal de salida la respuesta de la red es nula y, dado que esto representa un volumen amplio en el espacio fase, se espera observar un máximo global para $\rho_i(1)$.

En fig. 5.4 se muestra la densidad de estados. Notese que se presentan varios picos en la distribución, principalmente ubicados sobre racionales como: $0, \frac{1}{4}, \frac{1}{3}, \frac{1}{2}, \frac{2}{3}, \frac{3}{4}, 1$. El origen de dichos picos se debe a la definición de error empleada (ec. 2.4), es decir, bajo esta definición se acumulan volúmenes significativamente amplios en el espacio fase de redes cuya respuesta ante los estímulos es muy similar; y también la densidad de puntos con otros valores de error son muy escasos.

La densidad de estados es un ingrediente fundamental para poder emplear la teoría estadística del aprendizaje aquí propuesta. Mediante esta distribución, es posible inferir o aproximar otras cantidades que serán de interés, como por ejemplo la evolución del error promedio.

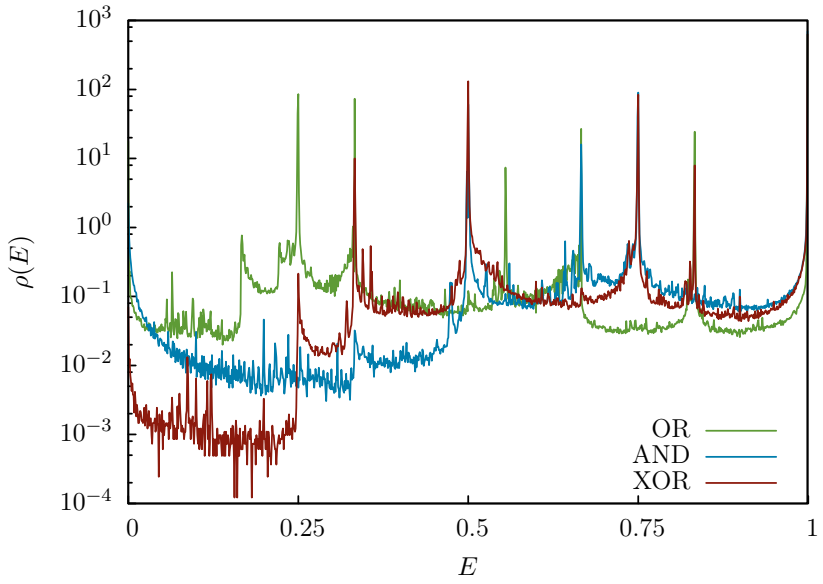

Figura 5.4: Densidad de estados para redes de $N = 2$ (dos neuronas), el eje vertical está en escala logarítmica.

5.2.1. La función de distribución de accesibilidad

La probabilidad de accesibilidad $g(E'|E)$, que es la probabilidad de que bajo un cambio infinitesimal en los pesos sinápticos, el error varíe a E' partiendo de un error E, es otro de los ingredientes imprescindibles de esta teoría. En el capítulo 4 se mostró, mediante una serie de argumentos, cómo se espera la forma de $g(E'|E)$, la cual fundamentalmente está poblada cerca de la diagonal y es esencialmente escasa en el resto del dominio. En esta sección se mostrará la forma de esta función de distribución. Estos son los pasos que se siguieron para su obtención:

1. Se prepara un ensamble de redes, en este caso de dos neuronas, y se fija, tanto la función objetivo, como la medida del error.

2. Se selecciona una red y se mide el error.

3. Se produce un cambio en un parámetro de la red empleando el algoritmo de Monte Carlo y se mide nuevamente el error. Este paso es repetido un número relativamente grande de veces para generar un histograma de los

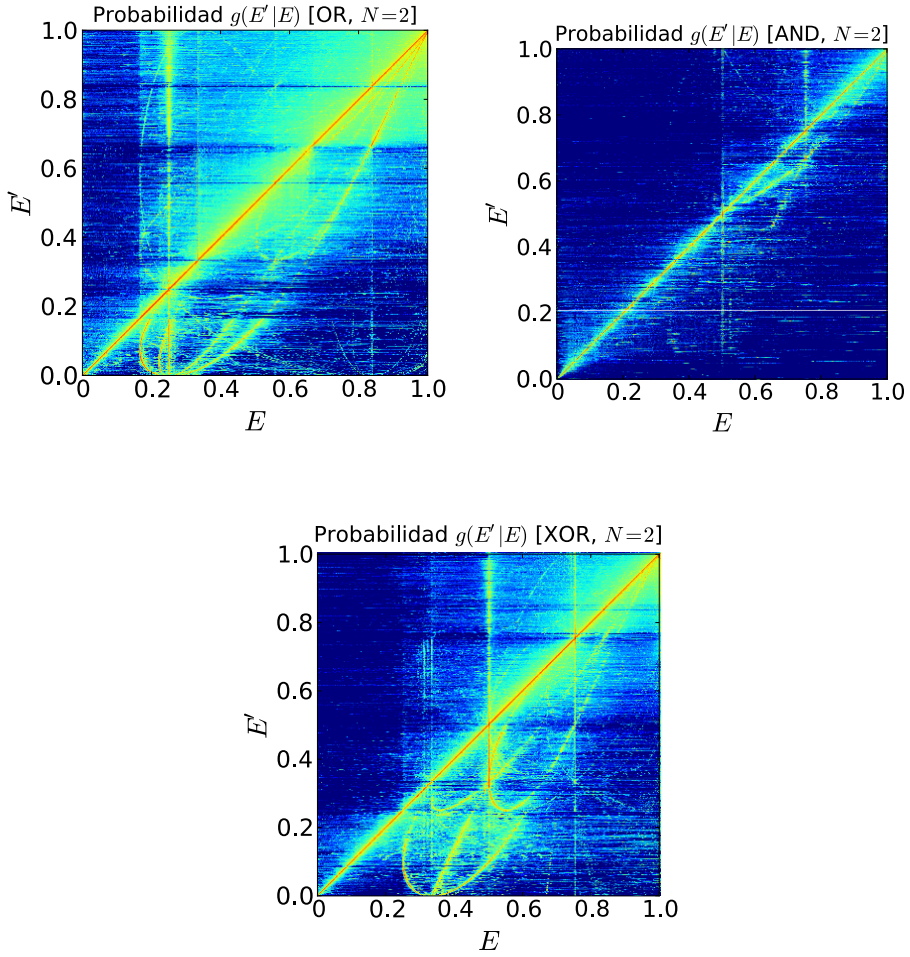

Figura 5.5: Muestra de las probabilidades $g(E'|E)$ de encontrar una red con error E' bajo un pequeño cambio partiendo de un error E. En la figura se muestran redes de dos neuronas medidas en el contexto de las funciones booleanas "OR", "AND" y "XOR" para izquierda, derecha y abajo respectivamente. Las tres imágenes son generadas normalizando g ($\int_{\forall E} g dE' = 1$).

errores visitados. Todos los cambios son rechazados con la finalidad de poblar sistemáticamente el histograma para una E.

4. La red es desechada y los pasos 1 a 3 son repetidos las veces necesarias para cubrir a todo el intervalo de error. Una vez que se haya concurrido

excesivamente un error, las nuevas redes que aparezcan y tengan dicho error son eliminadas.

Los resultados se muestran en fig. 5.5. Como se esperaba, principalmente la diagonal ($E = E'$) sobresale del resto de puntos. Sin embargo, también hay regiones fuera de la diagonal con valores significativos lo cual explica en gran medida la manera en que las redes aprenden. Si únicamente la distribución de accesibilidad tuviera valores cerca de la diagonal distintos de cero, los cambios admisibles en el error serían tan pequeños como el ancho de la diagonal. Al haber valores significativamente altos fuera de la diagonal, existe la posibilidad de que se presenten cambios finitos (macroscópicos) en el error.

Al hacer la aproximación de $g(E'|E)$ en el capítulo 4, se postuló como hipótesis que la medida del error sería una función continua y si los cambios propuestos por el algoritmo son infinitesimales, se producirían cambios infinitesimales a su vez. Considerando los datos obtenidos en la simulación, los valores de la distribución de accesibilidad lejos de la diagonal indican la presencia de las discontinuidades en el paisaje de error. Evidentemente, habrá casos en los que dichas discontinuidades aceleren el proceso del aprendizaje y en ocasiones casos en los que lo desfavorezca.

El conocimiento de dicha distribución, por lo tanto, puede ayudar a mejorar los algoritmos de aprendizaje. Para ello es necesario que el algoritmo emplee de manera inteligente la forma de esta función. Una manera podría ser correlacionando la probabilidad de cambios, en particular los que producen las discontinuidades, con el error actual de la red. Haciendo la analogía con el juego de mesa "Serpientes y Escalas"[1], en vez de que un jugador utilice un dado para avanzar en el juego, que sea él quien designe el número de posiciones que va a avanzar a su conveniencia según la estrategia que haya ideado. Esto podría explicar la manera en que las nuevas generaciones durante una corrida del algoritmo genético aprenden más rápido que la primera generación, la distribución de accesibilidad ha cambiado, los estados en los que las redes dan saltos discontinuos acortando el proceso de aprendizaje son más prominentes, es decir, de alguna manera, la función de distribución de accesibilidad es alterada.

En resumen, el conocimiento de la distribución de accesibilidad permite construir nuevas estrategias que aceleren o hagan más eficiente el proceso de aprendizaje. Esto ocurre, no sólo en el contexto de las simulaciones aquí presentadas, sino también en uno más general, inclusive biológico por ejemplo.

[1]El juego "Serpientes y Escaleras" trata de un tablero con celdas ordenadas, un comienzo y un final. El primer jugador que alcance la celda final es el ganador. Los jugadores sólo pueden emplear un dado que designa el numero de pasos que puede avanzar; sin embargo, hay atajos (escaleras) y desviaciones (serpientes) que obligan al jugador a saltar abruptamente de posición acelerando o alentando el proceso de lleagar a la meta.

5.2.2. Simulaciones en la teoría estadística del aprendizaje

Una vez juntados todos los ingredientes para integrar la ecuación maestra (ec. 4.1) se mostrará la evolución de la distribución del ensamble de redes con el número de pasos de Monte Carlo.

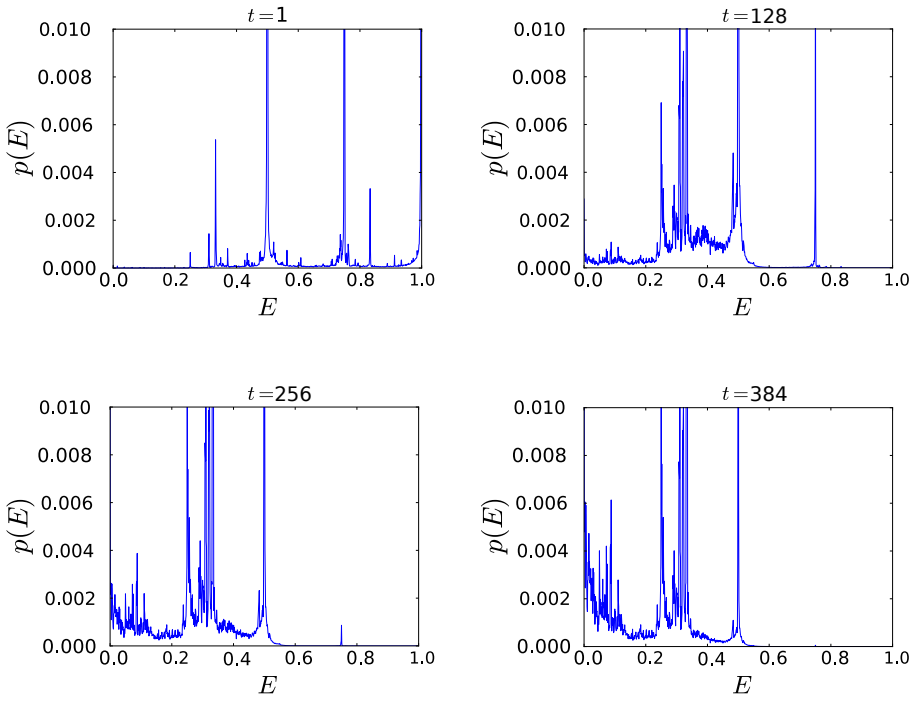

Figura 5.6: Evolución de la distribución de probabilidad del error en un ensamble de redes conforme incrementa el número de iteraciones del algoritmo de Monte Carlo. La función objetivo es la compuerta lógica "XOR" para $t = 1, 128, 256, 384$ iteraciones de izquierda a derecha y de arriba a abajo.

Como se puede observar en fig. 5.6 en un principio, la distribución de redes está muy cargada a la extrema derecha, es decir, la población de redes con error $E = 1$ es muy alta. Conforme transcurren las iteraciones, dicha distribución comienza a ensancharse y a desplazarse a la izquierda disminuyendo el valor de la probabilidad en ese punto y así para cualquier pico que se presenta en $p(E, 0)$. Nótese que la distribución es de forma escarpada indicando el contraste en la distribución de probabilidades para una pequeña vecindad en el error.

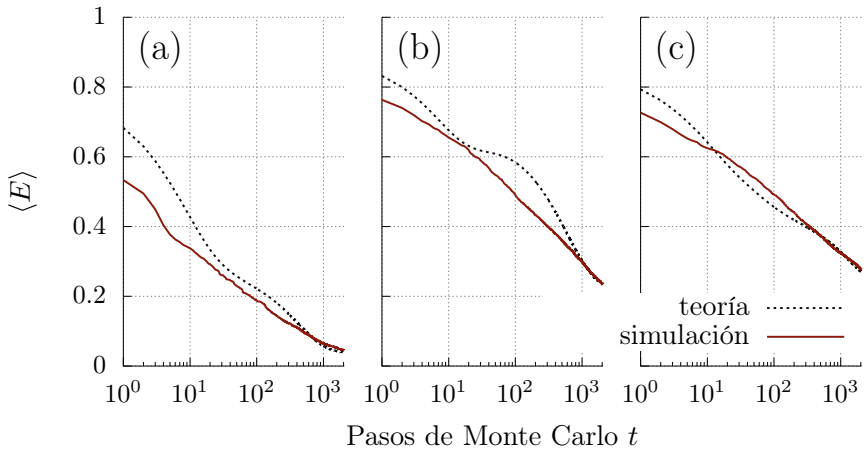

Figura 5.7: Comparación de la evolución del valor esperado del error para las tres funciones en (a), (b) y (c) para "OR", "AND" y "XOR" respectivamente. En cada gráfica se muestran dos curvas: la línea punteada representa la evolución dada por la integración numérica de la ecuación maestra, la curva sólida fue obtenida directamente de la simulación con las redes de neuronas.

Hay ciertos valores del error, como por ejemplo el pico en $E = 1/4$, donde la probabilidad toma un valor muy alto, hecho que se mantiene por un largo transcurso de iteraciones.

Una vez integrada numéricamente la ecuación maestra, es posible evaluar en cada iteración el valor esperado del error. En fig. 5.7 se muestra una serie de gráficas para el "OR", "AND" y "XOR" para izquierda, centro y derecha respectivamente. En ellas se compara la evolución del valor esperado del error obtenido por tres aproximaciones. Una de ellas es con la expresión numérica de la distribución de accesibilidad mostradas en la fig. 5.5, otra es la obtenida bajo la aproximación de campo medio empleando los argumentos que se proponen para aproximar el valor la función de distribución de accesibilidad y por último se muestra la evolución del promedio del error de las redes con el procedimiento de Monte Carlo, la misma curva que en la fig. 5.1 para $N = 2$.

Como se puede observar, la aproximación de campo medio traza una curva muy lejana a la trazada por el algoritmo de Monte Carlo, es decir, la aproximación no es muy buena. Por otro lado, la curva negra, referente a la que emplea la distribución de accesibilidad obtenida numéricamente, hace una mejor aproximación. Sin embargo, en todos los casos, hay un intervalo en los primeros quinientos pasos en donde las curvas se separan mucho.

5.2.3. Robustez ante muerte celular

La manera en la que varía el error una vez que una red pierde a uno de sus individuos en promedio da indicios de la importancia que tienen cada uno de los miembros de la red. Es de esperarse que dicha cantidad varíe según la cantidad total de nodos de procesamiento dentro de la red; más aún, es de esperarse que la pérdida sea menos significativa si la red es de muchas neuronas comparado con una red constituida de unas cuantas células. Por lo tanto, podemos estimar que la curva será una función monótona decreciente comenzando en $\Delta E = 1$ para $N = 1$, sin importar la función objetivo.

Por referencia, podríamos pensar en las redes como un sistema en el cual todas las neuronas contribuyen de igual manera y en el que no hay un comportamiento "no lineal" en la interacción entre ellas. En ese caso, con la pérdida de un componente de la red, se esperaría un decrecimiento inversamente proporcional al número de neuronas. En fig. 5.8 se muestran cuatro curvas, las respectivas a las tres funciones objetivo antes tratadas y la referencia $1/N$. Como se puede observar, para el "AND", el comportamiento es muy semejante al de la referencia; sin embargo, para el "OR" o el "XOR" no es así, dado que para esta última, la robustez es significativamente menor, es decir, la pérdida de una célula es muy significativa; en cambio, las redes emulando la función "OR" presentan, en promedio, mayor robustez que la referencia.

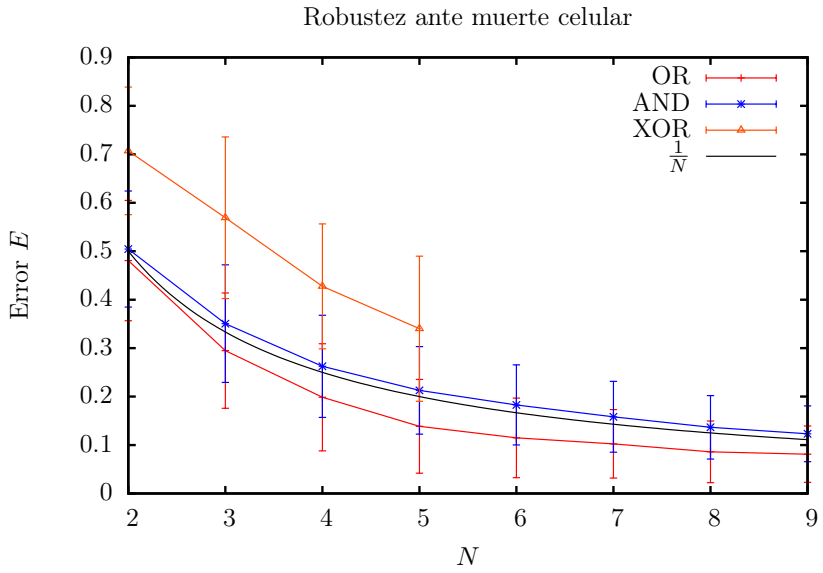

Figura 5.8: Incremento en el error después de la muerte de una neurona para redes de N nodos. En cada caso, las redes, previo a la muerte celular, comienzan con un error menor a 0.01. Después se hace un promedio de cambio en el error que ha de tener la muerte de cada una de las neuronas por separado.

Conclusiones

De acuerdo a las hipótesis planteadas, hemos visto que es posible almacenar inforamición en los pesos sinápticos y no en las neuronas *per se*. De esta manera, una red de neuronas puede realizar una amplia gama de funciones, siempre y cuando ésta cuente con suficientes neuronas, o más bien, grados libertad en el espacio de pesos sinápticos. Finalmente, mediante un análisis estadístico, es posible estimar, predecir y entender qué factores favorecen y dificultan el aprendizaje.

La enorme complejidad de las redes de neuronas difícilmente podrá ser planteada en unas cuantas ecuaciones y reglas. Para comenzar, se tiene la actividad de una neurona que modelamos con una reducción del modelo de HH, extrayendo a manera de lo posible las propiedades neurocomputacionales más relevantes en este contexto. Naturalmente, una de las limitantes que adquiere nuestro modelo viene de tomar un tipo de neurona como la base, suposición que obliga a considerar redes muy pequeñas comparadas con la extensión del sistema nervioso.

Después, al juntar muchas neuronas, modelamos la actividad de una red o un circuito pequeño. Con esta red logramos emular algunas funciones que arbitrariamente fueron escogidas con el fin de comprender el comportamiento de sus corelativos biológicos. Encontramos que no es fácil para una red emular cualquier función, hay un número mínimo de neuronas necesarias para que la red emule con precisión a la función y los pesos sinápticos que conectan a las neuronas deben escogerse bien.

Se realizaron estudios para funciones booleanas básicas: las compuertas lógicas “OR”, “AND” y “XOR”. Se hizo un análisis de cómo las redes pueden emular estas funciones y qué requerimientos deben cumplirse para ello. Por otro lado, es imperativo hacer estudios en el espacio de frecuencias, en lugar de promediar en el tiempo, y extender el estudio más allá de funciones booleanas. La mayoría de las neuronas de las cuales hemos hecho referencia para este trabajo producen pulsos de igual perfil lo que indica que más bien es en la frecuencia de los disparos en donde se codifica la información.

No está de más decir que el modelo de la red propuesto supone que los mecanismos que modifican los pesos sinápticos son ajenos a la actividad de los adyacentes. Por parte de la comunidad científica se han hecho un sinnúmero de experimentos, tratados y modelos que muestran cómo la actividad de neuronas contiguas influye en la manera en que los pulsos transmitidos son alterados para aumentar la eficacia y funcionalidad de la red. No obstante, se desconoce en gran medida la manera en que estos mecanismos producen mayor adaptabilidad en las redes y, más aún, la manera en que puedan llevar al aprendizaje.

Uno de los mecanismos más comunes de aprendizaje son los de prueba y error, cuestión que sugiere un algoritmo tipo Monte Carlo como la base para el desarrollo de las redes de neuronas. Este mecanismo presenta muchas desventajas tanto como un modelo de lo observado en seres vivos y también se muestra muy ineficiente ya que no se produce memoria o experiencia. Con la intención de mejorar ambos aspectos del mecanismo de aprendizaje, se implementó el algoritmo genético. En este algoritmo los nuevos individuos son generados de las partes más adaptadas de los veteranos, y de esta manera se ataca esta limitación. Sin embargo, las pruebas realizadas muestran escasas ventajas sobre el procedimiento exclusivamente de Monte Carlo.

La teoría estadística del aprendizaje es una teoría en vías de desarrollo. A pesar de ser una teoría muy inmadura, promete gran eficacia en la descripción del proceso del aprendizaje y puede auxiliar con vías que mejoran y adecúen los algoritmos de aprendizaje. Posiblemente, al madurar esta teoría, sea posible emplear las herramientas de teorías como la física estadística para hacer predicciones o simplificar y facilitar la obtención de resultados más precisos, dado que es posible interpretar el proceso del aprendizaje como uno que tiende al equilibrio.

Podría decirse que la medida del error se asemeja a la energía del sistema, por ejemplo,

$$H = 1 - |\hat{n}_w(s) \cdot \hat{\tau}(s)|^2. \tag{5.1}$$

De aquí podemos interpretar a la red de neuronas como un sistema en búsqueda del equilibrio, es decir, el mero proceso de aprendizaje se puede interpretar como una vía (metódica) por la cuál el error de la red disminuye y, por lo tanto, como un processo irreversible.

Hemos empleado la ecuación maestra para predecir la dinámica de un colectivo de redes. A manera de mostrar lo que ocurre con el proceso de aprendizaje propuesto se fijó la probabilidad de aceptación de manera que cumpla con las reglas del algoritmo de aprendizaje. Esto puede ser mejor aproximado al tener un mejor modelo de lo que sucede en realidad cuando las neuronas cambian los pesos de sus conexiones.

En las referencias [2, 3, 5–8] se discute la manera en que los pesos sinápticos varían a largo plazo en dependencia de la diferencia de tiempos entre el pulso postsináptico y el presináptico. Sin embargo, se han identificado varios procesos a través de los cuales las neuronas pueden alterar sus conexiones dinámicamente, cada uno con su duración y diferentes orígenes pero aún se desconoce cuantitativamente el efecto que esto tiene en el procesamiento de señales o en el proceso de aprendizaje. Lo que apunta a que hace falta investigación en esos menesteres antes de aventurarnos en estudiar procesos más complejos.

Sin duda alguna queda mucho trabajo por hacer y muchas interrogantes por resolver, comenzando por refinar los modelos que se proponen. Entre ello está el adaptar un modelo de aprendizaje que sea más congruente con la manera en que las neuronas cambian los pesos sinápticos. También hay que entender mejor los factores que favorecen o perjudican el aprendizaje y, por supuesto, respaldar con evidencia más sólida, observaciones, experimentos y demás, la veracidad de esta teoría recién formulada.

Apéndice A

Detalles sobre el modelo de la neurona

La respuesta de la neurona está dada por el siguiente conjunto de ecuaciones,

$$\begin{aligned}
\psi(s) &= \theta(s-h)\theta(\varphi-1), \\
h &= \frac{1}{100}(h_0 - V), \\
h_0 &= \begin{cases} 6.13 - 0.6I + 0.04I^2 - 0.002I^3 & \text{si } I < I_1 \\ 0 & \text{si } I_1 < I < I_2 \\ 6.3 - 1.5I + 0.23I^2 - 0.017I^3 & \text{si } I_2 < I \end{cases}, \\
\dot{V} &= -\alpha(I)V + 100s\delta(t - t_s), \\
\alpha(I) &= 0.67478 + 0.00638803I, \\
\dot{\varphi} &= \omega_0 + \delta\omega + \mathrm{PRC}(100s, \varphi)\delta(\varphi - \varphi_s), \\
\omega_0 &= \begin{cases} 5.5\sqrt{I + 0.01} + 45.4 & \text{si } I < I_1 \\ 10.4\sqrt{I - 5.1} + 46.8 & \text{si } I_1 < I \end{cases}, \\
\delta\omega &= 0.0106V.
\end{aligned}$$

Las variables V y φ son actualizadas cada vez que se cumple un ciclo o en cada recepción de un estímulo.

$$V(t^+) = (1 - \psi)V(t^-)$$

y

$$\varphi(t^+) = (1 - \psi)\varphi(t^-),$$

donde t^+y t^- son los tiempos inmediatamente posteriores y anteriores al pulso que da la neurona.

Apéndice B

Detalles de los algoritmos: Los códigos

A continuación se detallarán las partes principales de los códigos empleados para los algoritmos de aprendizaje de las redes de neuronas.

B.1. Mapa conceptual de los programas

Los programas empleados para este trabajo fueron estructurados de forma modular para facilitar su manejo, desarrollo y mantenimiento. En la figura B.1 se muestra el cianotipo y a continuación se detalla cada una de sus partes.

De manera esquemática, el módulo "Soma" es una clase genérica en donde se modela el cuerpo y cono axónico de una neurona. Dentro de esta clase, por medio de herencia se definen otras clases que especifican el funcionamiento de la célula. el modelo simplificado "MS" empleado en la mayor parte de la tesis, el modelo de Hodgkin–Huxley que se usó como la base para el modelo simplificado. De la misma manera se pueden idear e implementar otros modelos de la actividad de la neurona.

En paralelo se encuentra el módulo "Red" que es una manera genérica de representar una gráfica. En esta se encuentran básicamente dos tipos de redes con el fin de optimizar ciertas funciones. Una de ellas es la red matricial, en donde se trabaja directamente con la matriz de adyacencia. Este módulo es representado por "Mat". Mayormente, este módulo optimiza cálculos para el análisis de la red. El otro tipo de red implementado es el que se basa en listas de las conexiones para cada nodo. Este módulo optimiza las funciones de la red. El módulo está representado por "Lista". Es posible convertir de una subclase a la otra para alternar su operabilidad.

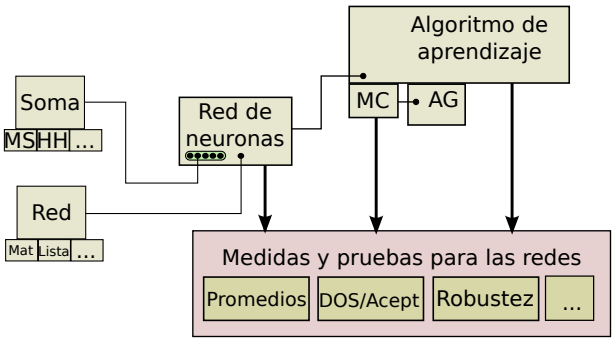

Figura B.1: Cianotipo de los programas y módulos empleados en este trabajo.

Ambos módulos mencionados anteriormente son miembros de la clase "Red de neuronas" el cual constituye al circuito de neuronas que se va a estudiar. La red de neuronas como tal es capaz de operar como una función que recibe una señal y la traduce a una respuesta y también, proponiendo una función objetivo, es capaz de calcular el error de la red. A la vez es capaz de ordenar cambios a la red según lo pida el algoritmo de aprendizaje.

El algoritmo de aprendizaje es un módulo que consta de dos elementos: el algoritmo tipo Monte Carlo ("MC") y el algoritmo genético ("AG"). Ambos módulos tienen como elemento principal a la red de neuronas. El algoritmo genético tiene como base para la adaptación de las redes el algoritmo de Monte Carlo, sin embargo, se pueden emplear otros algoritmos que optimicen este proceso.

Finalmente se encuentra el conjunto de programas diseñados para medir y analizar los resultados obtenidos de los módulos descritos anteriormente. Por ejemplo, el programa que obtiene el promedio del error conforme se itera sobre el algoritmo de Monte Carlo. Éste recibe directamente los resultados que el módulo "MC" produce a cada iteración. Otro ejemplo es el de la medida de robustez de las redes ante muerte celular, el cual es implementado en el módulo "Robustez". Un grupo de redes de neuronas, después de un largo proceso de entrenamiento, son recibidas por este programa para ser evaluadas.

Ahora se detallará con un poco más de profundidad la estructura de los módulos principales. No obstante, se añadirá como material suplementario a este trabajo el código en si puesto que es más extenso que este mismo texto y no vale la pena insertarlo aún en el apéndice.

B.2. La red de neuronas

B.2.1. La neurona

La definición de la actividad del soma, así como cada una de las funciones del cuerpo celular fueron implementadas esencialmente como están descritas en el listado de código 1. De la misma manera, se muestra en el listado de código 2 la definición del la clase "Network", la cual es la clase genérica que administra las conexiones entre nodos de una red. Seguido de esto se presenta el código del listado 3 donde se define la clase "NeuralNetwork" que emula la manera en que los objetos "Soma" serán los nodos de un objeto "Network", en términos prácticos, esta clase presenta la manera en que ambos objetos en unión forman una red neural. Entonces, todas las funciones que hace una red de neuronas se describen aquí. A manera esquemática, también se muestra a la red neural en fig. 2.1

```
1  // Neuron.h
2
3  class Soma {
4  private:
5    // here we declare the variables and internal methods
6
7    double J0; // the soma's background current
8    double h0; // the soma's base threshold
9    double w0; // the soma's base frequency
10
11   // The following function fixes the threshold depending on the
12   // background current. Via curve fitting.
13   double getThrshld(double J) {
14     double a, b, c, d;
15     if (J < 5.26467084639) {
16       a = 6.28553;
17       b = - 2.39822;
18       c = 0.37956;
19       d = -0.0286452;
20     }
21     else if (J < 8.4105295915) {
22       a = 0;
23       b = 0;
24       c = 0;
25       d = 0;
26     }
27     else {
28       a = 6.28553;
```

```
29        b =  -1.45456;
30        c = 0.230209;
31        d = -0.0173738;
32      }
33      return a + b*J + c*J*J + d*J*J*J;
34    }
35
36    // the following function fixes the frequency (and refractory period)
37    // via curve fitting
38    double getFreq(double I) {
39      double a, b, c;
40      if (I < 5.3) {
41        a = 5.52699;
42        b = -0.01258;
43        c = 45.3749;
44      }
45      else {
46        a = 10.3973;
47        b = 5.09885;
48        c = 46.8354;
49      }
50
51      return a*sqrt(J - b) + c;
52    }
53
54    // the following is the integrator of the ODEs that
55    // govern the behaviour of the soma
56    void update(double xi) {
57      input = xi;
58
59      if (noisy) {
60        J += gaussRandDist(noise)*dt;
61        J = abs(J);
62        h0 = getThrshld(J);
63        w0 = getFreq(J);
64      }
65
66      V += (0.67478 + 0.00638803*J)*(input - V)*dt;
67      w = w0 + 10.59514237167*V;
68      // w = w0 + 1119.8265371669497*V;
69      h = h0 - V;
70      if (phi >= 1.0) {
71        phi = 1.001;
72      }
73      else {
74        phi += fPRC(phi + 0.76, up_to((input - V)/10, 9.96)) + w*dt;
75      }
76    }
```

```
77
78  public:
79
80    // the following is the initializer of the class (constructor)
81    Soma (double intrCurr) {
82      J0 = intrCurr;
83      J = J0;
84      h0 = getThrshld(J);
85      h = h0;
86      w0 = getFreq(J);
87      w = w0;
88      V = 0;
89      phi = rhand();
90      alive = true;
91      noise = 0;
92      noisy = false;
93    }
94
95    // yet another constructor
96    Soma() {
97    }
98
99    // and the destructor
100   ~Soma() {
101   }
102
103   // another way of setting the somas properties and variables
104   void setSoma(double intrCurr) {
105     J0 = intrCurr;
106     J = J0;
107     h0 = getThrshld(J);
108     h = h0;
109     w0 = getFreq(J);
110     w = w0;
111     V = -0.01;
112     phi = rhand();
113     alive = true;
114     noise = 0;
115     noisy = false;
116   }
117
118   double spike(double xi) {
119     if (alive) {
120       output = Heaviside(input - h)*Heaviside(phi - 1);
121       int a = 1 - output;
122       V *= a; h *= a; phi *= a;
123       update(100*xi);
124     }
```

```
125       else {
126         output = 0;
127       }
128       return output;
129     }
130
131     void reset(void) { // in case neurons get tierd.
132       h = h0;
133       w = w0;
134       V = 0;
135       input = 0;
136       phi = rhand();
137     }
138
139     double Threshold(void) {
140       return h0;
141     }
142
143     double Frequency(void) {
144       return w0;
145     }
146
147     void kill() {
148       alive = false;
149     }
150
151     void revive() {
152       alive = true;
153     }
154
155     void setNoise(double ns) {
156       if (abs(ns) < 1e3) {
157         J = J0;
158         noise = 0.0;
159         noisy = false;
160       }
161       else {
162         noise = ns;
163         noisy = true;
164       }
165
166       return;
167     }
168
169     void regenSettings(void) {
170       h0 = getThrshld(J0);
171       h = h0;
172       w0 = getFreq(J0);
```

```
173       w = w0;
174       V = 0;
175       return;
176     }
177
178   };
```

Listado 1: Código en donde se define el objeto "Soma", el cuál emula la actividad del cono axónico y algunas otras funciones del cuerpo celular.

B.2.2. La red

```
1   // Net.h
2
3   class Network { // Connections to first neighbors w/ variable number of neighbors
4   private:
5     double KK;
6     vector< int > K;
7
8     void updateSettings(void) {
9       K.resize(NN+NO);
10
11      h.resize(NN + NO);
12      w.resize(NN + NO);
13      mp.resize(NN + NO);
14      autosomic.resize(NN + NO);
15
16      for (int i = 0; i < NN+NO; i++) {
17        h[i].resize(K[i]);
18        w[i].resize(K[i]);
19        mp[i].resize(K[i]);
20        autosomic[i].resize(K[i]);
21        for (int j = 0; j < K[i]; j++) autosomic[i][j] = 0;
22      }
23
24    }
25
26    bool checkConnections(int i, int j) {
27      // checks if the connectios are compatible
28      bool tf = true;
29
30      if ((i >= NN) && (h[i][j] >= NN)) {
31        tf = false;
32      }
```

```
33
34      if (h[i][j] == i) {
35        tf = false;
36      }
37
38      for (int l = 0; l < K[i]; l++) {
39        if ((j != l) && (h[i][j] == h[i][l])) {
40          tf = false;
41          break;
42        }
43      }
44
45      return tf;
46    }
47
48  public:
49    Network(int numInputs, int numNeurons, int numOutputs,
50         vector<int> numNeighbors, double p, double dW) {
51      prob = p; dWeightSize = dW;
52      NI = numInputs; NN = numNeurons; NO = numOutputs;
53      K = numNeighbors;
54      updateSettings();
55    }
56
57    ~Network() {
58    }
59
60
61    void changeNet(double E = 1) { // in each step autosomic gets older
62      for (int i = 0; i < NN+NO; i++)
63        for (int j = 0; j < K[i]; j++)
64          autosomic[i][j]++;
65
66      // choose new candidates
67      node = floor((NN+NO)*rhand());
68      element = 0;
69      if (K[node] > 1)
70        element = floor(K[node]*rhand());
71
72      vector<int> freeNodes;
73      int maxNei;
74
75      if (node < NN) maxNei = NN + NI - 1;
76      else maxNei = NN;
77
78      if (K[node] == 0) { // if there is no neighbor make one
79        type = '1';
80        h[node].push_back(floor((NN+NI)*rhand()));
```

```
81        w[node].push_back(0.1*dWeightSize*(2*rhand()-1));
82        autosomic[node].push_back(0);
83        while(!checkConnections(node, h[node].size()-1))
84          h[node][h[node].size()-1] = floor((NN+NI)*rhand());
85        K[node] = h[node].size();
86      }
87      else { // if there is a neighbor then...
88        if (rhand() < prob) { // change to another neighbor
89          if (K[node] >= maxNei) { // if all are connected then kill one
90            type = '2';
91            oldNeighbor = popup(h[node], element);
92            oldWeight = popup(w[node], element);
93            oldAuto = popup(autosomic[node], element);
94            K[node] = h[node].size();
95          }
96          else { // or else
97            float a = rhand();
98            if (a < 0.45) { // add one up
99              type = '3';
100             remainingNodes(freeNodes, node, h[node], maxNei, NN);
101             int newNei = freeNodes[floor(freeNodes.size()*rhand())];
102             h[node].push_back(newNei);
103             w[node].push_back(0.1*dWeightSize*(2*rhand()-1));
104             autosomic[node].push_back(0);
105             while(!checkConnections(node, h[node].size()-1))
106               h[node][h[node].size()-1] = floor((NN+NI)*rhand());
107             K[node] = h[node].size();
108           }
109           else if (a < 0.9) { // cut one connection
110             type = '4';
111             oldNeighbor = popup(h[node], element);
112             oldWeight = popup(w[node], element);
113             oldAuto = popup(autosomic[node], element);
114             K[node] = h[node].size();
115           }
116           else { // change one neighbor for another
117             type = '5';
118             oldNeighbor = h[node][element];
119             remainingNodes(freeNodes, node, h[node], maxNei, NN);
120             int newNei = freeNodes[floor(freeNodes.size()*rhand())];
121             h[node][element] = newNei;
122             oldAuto = autosomic[node][element];
123             autosomic[node][element] = 0;
124             while (!checkConnections(node, element))
125               h[node][element] = floor((NN+NI)*rhand());
126           }
127         }
128       }
```

```
129       else { // change the synaptic weight
130         type = '6';
131         oldWeight = w[node][element];
132         double dw = 0.04*dWeightSize*(2*rhand()-1)*(E+2e-2);
133         w[node][element] += dw;
134         oldAuto = autosomic[node][element];
135         autosomic[node][element] *= 0.75;
136       }
137     }
138   }
139
140   void geneAging(void) {
141     for (int i = 0; i < NN+NO; i++)
142       for (int j = 0; j < K[i]; j++)
143         autosomic[i][j]++;
144
145     return;
146   }
147
148   void restoreNet(void) { // undo anything done by 'changeNet()'
149     int l;
150     switch (type) {
151       case '1':
152         l = h[node].size();
153         popup(h[node], l-1);
154         popup(w[node], l-1);
155         popup(autosomic[node], l-1);
156         K[node] = h[node].size();
157         break;
158       case '2':
159         h[node].push_back(oldNeighbor);
160         w[node].push_back(oldWeight);
161         autosomic[node].push_back(oldAuto);
162         K[node] = h[node].size();
163         break;
164       case '3':
165         l = h[node].size();
166         popup(h[node], l-1);
167         popup(w[node], l-1);
168         popup(autosomic[node], l-1);
169         K[node] = h[node].size();
170         break;
171       case '4':
172         h[node].push_back(oldNeighbor);
173         w[node].push_back(oldWeight);
174         autosomic[node].push_back(oldAuto);
175         K[node] = h[node].size();
176         break;
```

```
177         case '5':
178           autosomic[node][element] = oldAuto;
179           h[node][element] = oldNeighbor;
180           break;
181         case '6':
182           autosomic[node][element] = oldAuto;
183           w[node][element] = oldWeight;
184           break;
185       }
186     }
187
188     void randomNet(double interval, double center) { // init a random net
189       for (int i = 0; i < NN; i++) {
190         h[i].resize(NN+NI);
191         for (int j = 0; j < NN+NI; j++)  h[i][j] = j;
192         popup(h[i], i);
193         int Q = (h[i].size()+1)*rhand();
194         for (int j = 0; j < Q; j++)
195           popup(h[i], int(h[i].size()*rhand()));
196         K[i] = h[i].size();
197       }
198
199       for (int i = NN; i < NN+NO; i++) {
200         h[i].resize(NN);
201         for (int j = 0; j < NN; j++) h[i][j] = j;
202         int Q = (h[i].size()+1)*rhand();
203         for (int j = 0; j < Q; j++) {
204           popup(h[i], int(h[i].size()*rhand()));
205         }
206         K[i] = h[i].size();
207       }
208
209       for (int i = 0; i < NN+NO; i++) {
210         w[i].resize(K[i]);
211         mp[i].resize(K[i]);
212         autosomic[i].resize(K[i]);
213         for (int j = 0; j < K[i]; j++) autosomic[i][j] = 0;
214
215         for (int j = 0; j < K[i]; j++) {
216           w[i][j] = interval*rhand() - 0.5*interval + center;
217           mp[i][j] = 40*(rhand() - 0.5) + 200;
218         }
219       }
220     }
221
222     double avNumNeighbor(void) {
223       double sum = 0;
224       for (int i = 0; i < NN+NO; i++) {
```

```
225         sum += K[i];
226       }
227       return sum/(NN+NO);
228     }
229
230     // more code not included
231     // ...
232     // ...
233
234 };
```

Listado 2: Código donde se define la clase "Network", la cual es una manera genérica de conectar los nodos de una red con pesos variables. De igualmanera se definen las funciones que alteran y deshacen cambios solicitados por el algoritmo de Monte Carlo.

B.2.3. La red de neuronas

```
1  // NeuralNetwork.h
2
3  #include "Neuron.h"
4  #include "Net.h"
5  #include "DataAnalysis.h"
6
7  class NeuralNetwork {
8
9  private:
10   int NI, NN, NO;
11   vector<int> K;
12   int someNeuron;
13   char type;
14   vector< double > input;
15   vector< double > output;
16   vector< double > currents;
17   vector< vector< double > > timeSeries;
18   vector<Soma*> *s;
19   Network *net;
20   Network *net_;
21   vector<double> V;
22
23 public:
24   NeuralNetwork() {
25     record = false;
26     SynPlas = false;
```

```
27      net_ = new Network2;
28    }
29
30    NeuralNetwork(vector<Soma*>& ss, Network* nnet) {
31      s = &ss; net = nnet;
32      NN = s->size();
33      cout << NN << endl;
34      if (NN != net->nNeurons()) cout << "Incompatibility!\n";
35      NI = net->nInputs();
36      NO = net->nOutputs();
37
38      K.resize(NN+NO);
39      V.resize(NN, 0);
40      for (int i = 0; i < NN+NO; i++) K[i] = net->nNeighbors(i);
41      input.resize(NI);
42      output.resize(NO);
43      // getCurrents();
44      record = false;
45      SynPlas = false;
46      net_ = new Network2;
47    }
48
49    ~NeuralNetwork() {
50    }
51
52    vector<double> NeuralNetworkFunction(vector<double> I) {
53      input = I;
54      vector<double> z(NN);
55      //     cout << output.size() << endl;
56
57      for (int i = 0; i < NN; i++) {
58        z[i] = 0;
59        if (net->nNeighbors(i) > 0)
60          for (int j = 0; j < net->nNeighbors(i); j++) {
61            if (net->neighbor(i,j) < NN)
62              z[i] += net->connectionWeight(i,j)*
63                V[net->neighbor(i,j)];
64            else
65              z[i] += net->connectionWeight(i,j)*
66                input[net->neighbor(i,j)-NN]*0.062;
67          }
68      }
69
70      for (int i = 0; i < NN; i++) V[i] = (*s)[i]->spike(z[i]);
71
72
73      if (record) timeSeries.push_back(V);
74
```

```
75      for (int i = NN; i < NN+NO; i++) {
76        output[i-NN] = 0;
77        if (net->nNeighbors(i) > 0) {
78          for (int j = 0; j < net->nNeighbors(i); j++) {
79
80            output[i-NN] += net->connectionWeight(i,j)*
81              V[net->neighbor(i,j)]*
82              0.021999498/dt;
83          }
84        }
85      }
86
87      return output;
88    }
89
90    vector<double> averagedFunction(vector<double> I) {
91      vector<double> avOut(NO);
92      vector<double> tmpO(NO);
93      vector<DataAnalysis> d(NO);
94
95      for (int i = 0; i < NO; i++) d[i].empty();
96
97      for (double t = 0; t < Time; t+=dt) {
98        tmpO = NeuralNetworkFunction(I);
99        for (int i = 0; i < NO; i++) d[i].fill(tmpO[i]);
100     }
101
102     for (int i = 0; i < NO; i++)
103       avOut[i] = d[i].avg();
104
105     resetSomas();
106
107     return avOut;
108   }
109
110   // more code not included
111   // ...
112   // ...
113
114 };
115
116
117 double getError(NeuralNetwork& u, char f, vector<double> input) {
118   int NO = u.getNetwork() -> nOutputs();
119
120   vector<double> y(NO, 0);
121   vector<double> z(NO, 0);
122
```

```
123   y = u.averagedFunction(input);
124   z = objective(f, input, NO);
125
126   double E = 0;
127   for (int i = 0; i < NO; i++) {
128     E += (z[i] - y[i])*(z[i] - y[i]);
129   }
130   E = sqrt(E);
131
132   return E;
133 }
134
135 double globalError(NeuralNetwork& u, char f) {
136   int NI = u.getNetwork() -> nInputs();
137
138   double E = 0;
139   vector< vector< double > > input = generateInput(NI);
140   int L = input.size();
141
142   vector<double> y(L, 0);
143   vector<double> z(L, 0);
144
145   for (int i = 0; i < L; i++) {
146     y[i] = u.averagedFunction(input[i])[0];
147     z[i] = objective(f, input[i], 1)[0];
148   }
149
150   double norm_y = sqrt(dot(y, y));
151   if (norm_y < 0.05) norm_y = 1;
152   double norm_z = sqrt(dot(z, z));
153
154   for (int i = 0; i < L; ++i)  {
155     y[i] /= norm_y;
156     z[i] /= norm_z;
157   }
158
159   double q = dot(y,z);
160   E = 1 - q*q;
161
162   return E;
163 }
```

Listado 3: En este código se detallan ciertas funciones que sirven para unir objetos de la clase "Soma" con un objeto "Network", en escencia, la clase "NeuralNetwork" administra las funciones de ambas clases y a partir de ello se definen las funciones de una red de neuronas. En este mismo código se definen también las diferente medidas del error, la definición vectorial nombrada "getError(...)", y la de ortogonalidad "globalError(..)".

B.3. Monte Carlo

En esta subsección se detalla el algoritmo de Monte Carlo, en fig. B.2 se muestra de manera esquemática el diagrama de flujo del algoritmo. Posteriormente se desglosa el código donde está definida la clase en el listado 4. Como es de esperarse, esta clase depende de la definición de error la cual está definida en el listado anterior. El algoritmo de Monte Carlo comienza por inicializar una red de manera aleatoria a partir de los parámetros definidos al comienzo del algoritmo. Una vez generada la red, el algoritmo itera *mcs* veces la función "Paso de Monte Carlo" para finalizar con una red que, por lo general, es más adecuada para realizar la función objetivo.

```
1  // MonteCarlo.h
2
3  #include "NeuralNetwork.h"
4
5  class MonteCarlo {
6  private:
7    NeuralNetwork *u;
8    int mcs;
9    char obj;
10   double finalError;
11   bool swept;
12   double beta;
13
14 public:
15   MonteCarlo() {
16   }
17
18   ~MonteCarlo() {
19   }
20
```

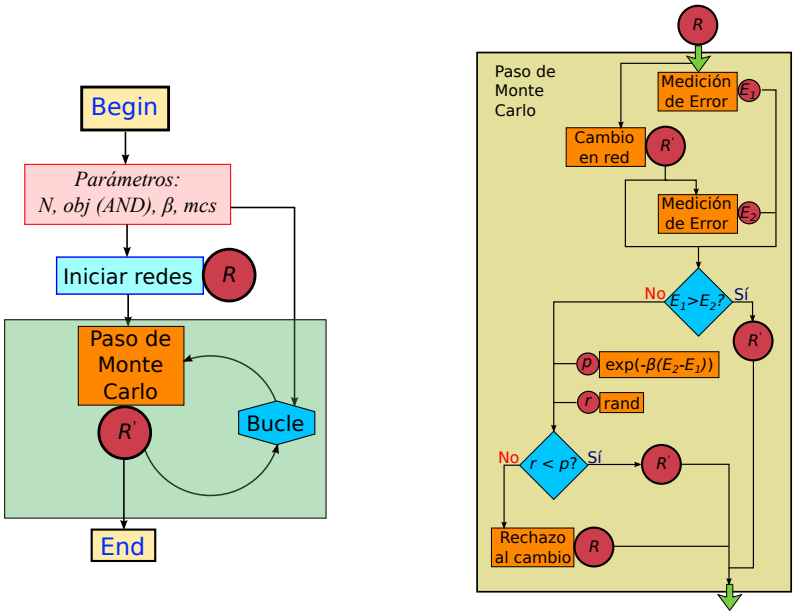

Figura B.2: A la izquierda, diagrama de flujo del algoritmo de Monte Carlo y a su derecha se desglosa la función "Paso de Monte Carlo". Para comenzar, se inicializan los parámetros, el número de neuronas de la red N, el objetivo a cumplir, p. ej., un "AND", la permisividad de aceptar errores mayores β y el número de veces que se va a iterar en el bucle *mcs*.

```
21    void setNeuralNetwork(NeuralNetwork& uu) {
22      u = &uu;
23      swept = false;
24    }
25
26    void setObjective(char ff) {
27      obj = ff;
28      swept = false;
29    }
30
31    void setSweepSize(int SZ) {
32      mcs = SZ;
33    }
34
35    void setTemperature(double Temp) {
```

```
36      beta = 1/Temp;
37    }
38
39    double giveError(void) {
40      if (!swept) {
41        finalError = globalError(*u, f);
42        swept = true;
43      }
44      return finalError;
45    }
46
47    double oneLoop(double tol = 1e-4) {
48      double Eo, Ef;
49
50      Eo = giveError();
51
52      if (Eo > tol) {
53
54        u->changeNeuralNetwork(Eo);
55        swept = false;
56
57        Ef = giveError();
58
59        if (Ef > Eo) {
60          if (rhand() > exp(-beta*(Ef-Eo))) {
61            u->restoreNeuralNetwork();
62            finalError = Eo;
63          }
64        }
65
66      }
67      else {
68        u->geneAging();
69        finalError = Eo;
70      }
71
72      return finalError;
73    }
74
75    // More code not included
76    // ...
77    // ...
78
79  };
```

Listado 4: Definición de la clase Monte Carlo.

B.4. El algoritmo genético

Por último, aquí se detalla el algoritmo genético. En fig. B.3 se muestra a manera esquemática el diagrama de flujo del algoritmo al igual que las funciones y variables principales del algoritmo. Posteriormente se muestra el código implementado, en el cual se desglosa con todo detalle específicamente cómo se llevan a cabo las funciones principales.

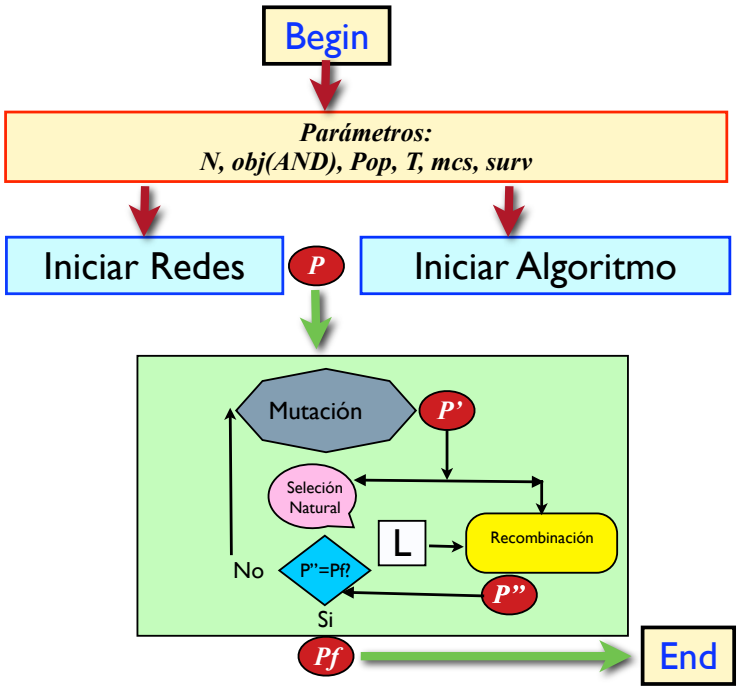

Figura B.3: Esquema y diagrama de flujo del algoritmo genético y sus partes. En el diagrama ubicado en la parte superior izquierda se muestra burdamente cómo es el flujo del algoritmo genético, en las ilustraciones adyacentes se muestra con mayor detalle cada una de las funciones involucradas el algoritmo.

```
1  // GeneticAlgorithm.h
2
3  #include "MonteCarlo.h"
4  #include "ioNeuralNetwork.h"
5
6  class GeneticAlgorithm {
```

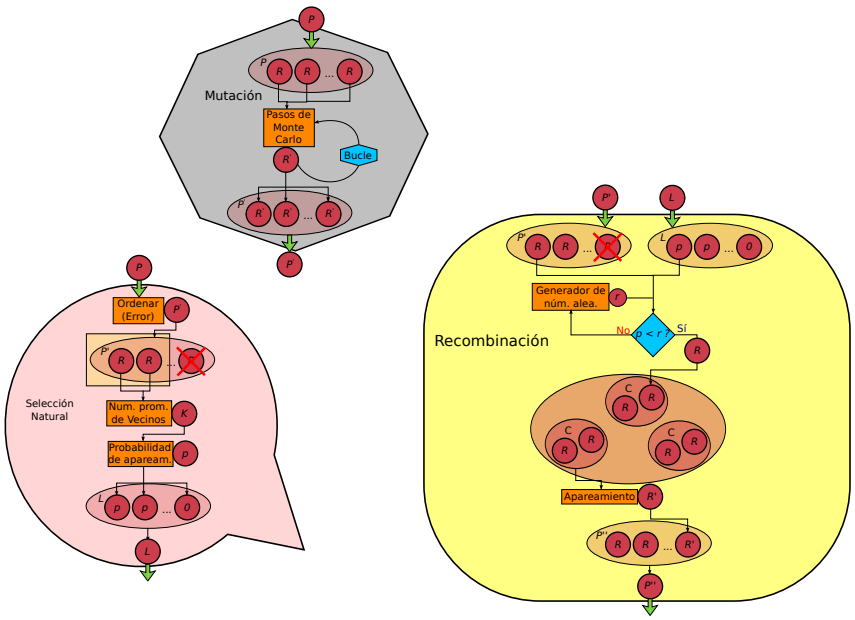

Figura B.4: Diagrama de flujo de las funciones principales del algoritmo genético mostradas en la figura B.3. Las funciones correspondientes están desglosadas y representadas aquí por los mísmos símbolos que en el diagrama principal.

```
7  private:
8    int NI;
9    int NN;
10   int NO;
11   char obj;
12   int Population;
13   int maxGen;
14   int sweepSize;
15   double prob;
16   double survivors;
17   double Temp;
18   double tolerance;
19   double startTol;
20   double alpha;
21   vector<Soma*> s;
22   vector<Network*> n;
23   NeuralNetwork u;
24   MonteCarlo mc;
```

```
25    vector<int> order;
26    vector<double> err;
27    vector< double > matingProb;
28    string graphName;
29    ostringstream errorName;
30    ofstream ofile;
31    int generation;
32    DataAnalysis groupError;
33
34  public:
35    GeneticAlgorithm() {
36      startTol = 0.25;
37      Temp = 0.1;
38      prob = 0.05;
39      tolerance = 0.01;
40      alpha = 0.5;
41    }
42
43    ~GeneticAlgorithm() {
44      for (int i = 0; i < Population; i++) delete n[i];
45      for (int i = 0; i < NN; i++) delete s[i];
46    }
47
48    void initializePopulation(bool initSomas = true) {
49      ofile.open(errorName.str().c_str());
50
51      if (initSomas) {
52        s.resize(NN);
53        for (int i = 0; i < NN; i++) {
54          s[i] = new Soma1;
55          s[i]->setSoma(expRandDist(4));
56        }
57      }
58
59      n.resize(Population);
60      for (int i = 0; i < Population; i++) {
61        n[i] = new Network2;
62        n[i]->setNetwork(NI, NN, NO, NN, prob, 5);
63      }
64
65      cout << "Starting population:\n";
66      int j = 0;
67      while (j < Population) {
68        n[j]->randomNet(1, 0.5);
69        u.setNeuralNetwork(s, n[j]);
70        mc.setNeuralNetwork(u);
71        double a = mc.giveError();
72
```

```
73        for (int k = 0; k < NN; k++) s[k]->reset();
74
75        if (a < startTol) {
76          n[j]->setNetworkError(a);
77          j++;
78          cout << "  " << j << " :  " << a << '\r';
79          cout.flush();
80        }
81      }
82      cout << endl;
83
84
85      stringstream filename;
86      filename << "results/trial-" << obj << '-' << NN << '-' << Population
87        << '-' << sweepSize << '-' << 0 << ".slz";
88
89      serializePopulation(filename.str());
90      filename.str("");
91
92      generation = 0;
93
94      mc.setSweepSize(1);
95
96      err = vector<double>(Population, 1);
97      order.resize(Population);
98    }
99
100   void update(void) {
101     generation = 0;
102
103     for (int i = 0; i < Population; i++) {
104       u.setNeuralNetwork(s, n[i]);
105       double a = globalError(u, obj);
106
107       n[i]->setNetworkError(a);
108       err[i] = a;
109     }
110
111     order = sort(err);
112
113     errorName << "results/ERR-" << obj << '-' << NN << '-' << Population
114       << '-' << sweepSize << '-' << maxGen << "r.dat";
115     ofile.open(errorName.str().c_str());
116
117     mc.setSweepSize(1);
118
119     return;
120   }
```

```
121
122   void mutate(void) {
123     for (int k = 0; k < Population; k++) {
124       cout << "  Mutating... " << k+1 << "\r";
125       cout.flush();
126       double currentError;
127       u.setNeuralNetwork(s, n[k]);
128       mc.setNeuralNetwork(u);
129
130       currentError = mc.giveError();
131
132       ofile << generation*sweepSize << " ";
133       ofile << currentError << " " << n[k]->avNumNeighbor();
134       ofile << " " << n[k]->avConnWeight() << endl;
135       for (int i = 0; i < sweepSize; i++) {
136         currentError = mc.sweep(1);
137         ofile << generation*sweepSize+i+1 << " ";
138         ofile << currentError << " " << n[k]->avNumNeighbor();
139         ofile << " " << n[k]->avConnWeight() << endl;
140
141       }
142       ofile << endl;
143
144       err[k] = mc.giveError();
145       n[k]->setNetworkError(err[k]);
146     }
147   }
148
149
150   void naturalSelection(void) {
151     cout << "\n  Generating probabilities for mating.\n";
152     vector<double> aN(Population);
153     matingProb = vector< double >(Population, 0);
154
155     for (int i = 0; i < Population; i++)
156       aN[i] = n[i] -> avNumNeighbor();
157
158     order = sort(err);
159     int l = floor( Population * survivors );
160
161     for (int i = 0; i < l; i++) {
162       int j = order[i];
163       matingProb[j] = exp( -alpha*aN[j] );
164     }
165     for (int i = l; i < Population; i++) {
166       int j = order[i];
167       matingProb[j] = 0;
168     }
```

```
169     double a = 0;
170     for (int i = 0; i < Population; i++) a += matingProb[i];
171     for (int i = 0; i < Population; i++) matingProb[i] /= a;
172
173     ofstream GnSt("geneStblty.dat");
174
175     for (int i = 0; i < NN+NO; i++) {
176       for (int j = 0; j < n[order[0]]->nNeighbors(i); j++) {
177         GnSt << n[order[0]]->geneStability(i,j) << " ";
178       }
179       GnSt << endl;
180     }
181     GnSt.close();
182   }
183
184   Network* mating(Network* parent1, Network* parent2, double tol = 0.001) {
185     int NI = parent1->nInputs();
186     int NN = parent1->nNeurons();
187     int NO = parent1->nOutputs();
188     vector< vector< int > > h(NN+NO);
189     vector< vector< double > > w(NN+NO);
190     vector< int > K(NN+NO);
191
192     for (int i = 0; i < NN+NO; i++) {
193       for (int j = 0; j < NN+NI; j++) {
194
195         if ((i >= NN) && (j >= NN)) break;
196
197         int p1gs, p2gs;
198         bool p1h = false, p2h = false;
199         int i1 = 0, i2 = 0;
200         double prob1, prob2;
201         double a = 0, b;
202
203         if (parent1->nNeighbors(i) > 0)
204         for (int i1 = 0; i1 < parent1->nNeighbors(i); i1++)
205           if (parent1->neighbor(i, i1) == j) {
206             p1h = true;
207             p1gs = parent1->geneStability(i,i1);
208             break;
209           }
210
211         if (parent2->nNeighbors(i) > 0)
212         for (int i2 = 0; i2 < parent2->nNeighbors(i); i2++)
213           if (parent2->neighbor(i, i2) == j) {
214             p2h = true;
215             p2gs = parent2->geneStability(i,i2);
216             break;
```

```
217             }
218
219           prob1 = sqrt(p1gs + 2.0);
220           prob2 = sqrt(p2gs + 2.0);
221           b = prob1 + prob2;
222           prob1 /= b;
223           prob2 /= b;
224
225           if (p1h && p2h) {
226             if (rhand() < prob1)
227               a = parent1->connectionWeight(i,i1);
228             else
229               a = parent2->connectionWeight(i,i2);
230           }
231           else if (p1h) {
232             if (rhand() < sqrt(p1gs + 2.0)/(sqrt(p1gs + 2.0) + sqrt(2.0)))
233               a = parent1->connectionWeight(i,i1);
234             else if (rhand() < 0.1)
235               a = rhand() - 0.5;
236           }
237           else if (p2h) {
238             if (rhand() < sqrt(p2gs + 2.0)/(sqrt(p2gs + 2.0) + sqrt(2.0)))
239               a = parent2->connectionWeight(i,i2);
240             else if (rhand() < 0.1)
241               a = rhand() - 0.5;
242           }
243           else if ((rhand() < 0.01) && (i != j))
244             a = rhand() - 0.5;
245
246           if (abs(a) > tol) {
247             h[i].push_back(j);
248             w[i].push_back(a);
249           }
250         }
251
252         K[i] = h[i].size();
253       }
254
255       Network* offspring;
256       offspring = new Network2(NI, NN, NO, K, 0.05, 0.5);
257
258       for (int i = 0; i < NN+NO; i++) {
259         for (int j = 0; j < K[i]; j++) {
260           offspring->setConnection(i, j, h[i][j], w[i][j]);
261         }
262       }
263
264       return offspring;
```

```
265       delete offspring;
266     }
267
268     void recombine(void) {
269       int l = floor( Population * survivors );
270       for (int k = l; k < Population; k++) {
271         int p1 = choose( rhand(), matingProb );
272         int p2 = p1;
273         while (p1 == p2) p2 = choose( rhand(), matingProb );
274
275         int son = order[k];
276
277         n[son] = mating(n[p1], n[p2]);
278       }
279       generation++;
280     }
281
282     void evolve() {
283       int l = floor( Population * survivors );
284       while (!satisfied()) {
285         cout << "Generation: " << generation+1 << endl;
286         mutate();
287         for (int i = 0; i < Population; i++) {
288           n[i]->setNetworkError(err[i]);
289           n[i]->setWeightSizeChange(5.5*err[i]+0.5);
290         }
291         cout << "Done mutating..." << endl;
292         if (Population > 3) naturalSelection();
293
294         if (maxGen > 1) {
295           stringstream filename;
296           filename << "results/trial-" << obj << '-' << NN << '-' << Population
297             << '-' << sweepSize << '-' << generation+1 << ".slz";
298
299           serializePopulation(filename.str());
300           filename.str("");
301         }
302
303         if (Population > 3)  recombine();
304         else generation++;
305       }
306     }
307
308     bool satisfied(void) {
309       bool sat = false;
310
311       if (generation >= maxGen) {
312         sat = true;
```

```
313         ofile.close();
314
315         cout << "Completed! Done evolving OK.\n\n";
316         order = sort(err);
317         cout << "Best net: " << order[0] << " ->  " << err[order[0]] << endl;
318
319         u.setNeuralNetwork(s, n[order[0]]);
320         vector< vector< double > > I(generateInput(NI));
321         for (int j = 0; j < I.size(); j++) {
322           cout << "(";
323           for (int i = 0; i < I[j].size()-1; i++) {
324             cout << I[j][i] << ", ";
325           }
326           cout << I[j][I[j].size()-1] << ") :-> ";
327
328           vector< double > O(u.averagedFunction(I[j]));
329           cout << "(";
330           for (int i = 0; i < O.size()-1; i++) {
331             cout << O[i] << ", ";
332           }
333           cout << O[O.size()-1] << "), \t(";
334           for (int i = 0; i < O.size()-1; i++)
335             cout << objective(obj, I[j], NO)[i] << ", ";
336           cout << objective(obj, I[j], NO)[O.size()-1] << ")\n";
337         }
338         cout << endl;
339
340         stringstream filename;
341         filename << "results/sorted-" << obj << '-' << NN << '-' << Population
342           << ".slz";
343
344         saveSortedPopulation(filename.str());
345
346         printWholeHimmeli();
347
348         cout << endl;
349       }
350
351       return sat;
352     }
353
354     // More code not included
355     // ...
356     // ...
357
358 };
```

Listado 5: Código en detalle de las funciones principales del algoritmo genético. Aquí se definen las funciones de mutación, seleción natural, apareamiento y recombinación de los individuos de la población.

Bibliografía

[1] David J. C. MacKay. *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, 2003.

[2] Xintian Yu, Harel Z. Shouval, and James J. Knierim. A biophysical model of synaptic plasticity and metaplasticity can account for the dynamics of the backward shift of hippocampal place fields. *AJP - JN Physiol*, 100(2):983–992, 2008.

[3] Guy Billings and Mark C. W. van Rossum. Memory retention and spike-timing-dependent plasticity. *AJP - JN Physiol*, 101(6):2775–2788, 2009.

[4] John G. Nicholls, A. Robert Martin, Bruce G. Wallace, and Paul A. Fuchs. *From Neuron to Brain*. Sinauer Associates, Inc. Publishers, 2000.

[5] H. Z. Shouval, M. F. Bear, and L. N. Cooper. A unified model of nmda receptor-dependent bidirectional synaptic plasticity. *PNAS*, 99(16):10831–10836, 2002.

[6] Luk Chong Yeung, Harel Z. Shouval, Brian S. Blais, and Leon N. Cooper. Synaptic homeostasis and input selectivity follow from a calcium-dependent plasticity model. *PNAS*, 101(41):14943–14948, 2004.

[7] Gastone C. Castellani, Elizabeth M. Quinlan, Ferdinando Bersani, Leon N. Cooper, and Harel Z. Shouval. A model of bidirectional synaptic plasticity: From signaling network to channel conductance. *Learn. Mem.*, 12:423–432, 2005.

[8] Gastone C. Castellani, Elizabeth M. Quinlan, Leon N Cooper, and Harel Z. Shouval. A biophysical model of bidirectional synaptic plasticity: Dependence on ampa and nmda receptors. *PNAS*, 98(22):12772–12777, 2001.

[9] Eugene M. Izhikevich. *Dynamical Systems in Neuroscience: The Geometry of excitability and Bursting*. MIT Press, 2007.

[10] D. Erdogmus and J.C. Principe. An error-entropy minimization algorithm for supervised training of nonlinear adaptive systems. *IEEE Transactions on Signal Processing*, 50(7):1780–1786, 2002.

[11] K. Aihara, T. Takabe, and M. Toyoda. Chaotic neural networks. *Physics Letters A*, 144(6-7):333–340, 1990.

[12] Robin J. Wilson. *Introduction to Graph Theory*. Addison Wesley Longman, 1998.

[13] Willi-Hans Steeb, Yorick Hardy, and Ruedi Stoop. *The Nonlinear Workbook: Chaos, Fractals, Cellular Automata, Neural Networks, Genetic Algorithms, Gene Expression Programming, Support Vector M*. World Scientific Publishing Company, 2005.

[14] Elfego Ruiz. *Tesis de Licenciatura: Osciladores no lineales acoplados*. Universidad Nacional Autónoma de México, 2010.

[15] Takafumi Miyamoto, Robert DeRose, Allison Suarez, Tasuku Ueno, Melinda Chen, Tai ping Sun, Michael J Wolfgang, Chandrani Mukherjee, David J Meyers, and Takanari Inoue. Rapid and orthogonal logic gating with a gibberellin-induced dimerization system. *Nature Chemical Biology*, 8:465–470, 2012.

[16] Sergi Regot, Javier Macia, Núria Conde, Kentaro Furukawa, Jimmy Kjellén, Tom Peeters, Stefan Hohmann, Eulàlia de Nadal, Francesc Posas, and Ricard Solé. Distributed biological computation with multicellular engineered networks. *Nature*, 469:207–211, 2011.

[17] G. Cybenko. Approximation by superpositions of a sigmoidal function. *MCSS*, 2(4):303–314, 1989.

[18] Vladimir Vapnik. *The Nature of Statistical Learning Theory*. Springer Verlag, 1995.

[19] R. Kubo, M. Toda, and N. Hashitsume. *Statistical Physics II: Nonequilibrium Statistical Mechanics*. Cambridge University Press, 1985.

[20] R. K. Pathria. *Statistical Mechanics*. Butterworth-Heinemann, 1996.