Redes neuronales modulares: topología, características y aplicación.

UNED. 2010-2012
Máster en física de sistemas complejos.
Trabajo fin de Máster.

Autor: Héctor Corte León.
Tutora: Elka Korutcheva.

Resumen – *En este trabajo se muestra un modelo de red neuronal estocástica basada la versión original de Hopfield. El modelo además de poseer neuronas estocásticas, que cambian de estado de acuerdo a una regla probabilistica, posee aprendizaje mediante entrenamientos múltiples, adapatandose al uso que tenga la red, además posee olvido pasivo de forma que la red nunca llega a saturarse. Nuestro desarrollo se centra en la búsqueda de los valores adecuados de cada parámetro del modelo, en la elaboración de un algoritmo y de un código de Matlab para trabajar con el modelo y en la realización de varias simulaciones para comprobar que todo está funcionando de forma correcta y que se verifican varias observaciones descritas para modelos similares en artículos relativamente recientes. Es de destacar que nuestro algoritmo de simulación nos permite llegar a ver el olvido pasivo en funcionamiento al medir de forma numérica la capacidad de la red y observar que utilizando olvido pasivo la capacidad se mantiene constante pese a seguir entrenando la red.*

Indice

1 Introducción.

En este trabajo presentamos un estudio de un modelo de red neuronal basado en el modelo clásico de red neuronal tipo Hopfield.

Las redes neuronales tipo Hopfield se han utilizado ampliamente para modelar la memoria asociativa a corto plazo [1][2][39]. Resultan sencillas de simular y además su relación con los vidrios de spin hace que existan muchas herramientas para analizar su dinámica [1]. Sin embargo existen muchas variaciones sobre el modelo original y en muchos de los casos su dinámica no está bien comprendida aun.

Aquí nos centraremos en la que hasta ahora es la versión más compleja del modelo inicial de Hopfield. Utilizaremos una versión con neuronas tipo McCulloch-Pitts estocásticas. La actualización de las neuronas tiene lugar de forma sincrona. Para el aprendizaje utilizamos el denominado entrenamiento múltiple (multipleshootlearning), que consiste en una versión de la ley de Hebb donde el aprendizaje de los patrones tiene lugar de forma gradual [3]. Además exploramos la utilización del olvido pasivo como técnica para evitar la saturación de la red [4][5].

Hemos seleccionado este modelo porque mantiene la simplicidad del modelo original y a la vez incorpora características que se han probado sirven para acercar el modelo a su análogo biológico. Entre ellas hay que destacar el olvido pasivo. Además en esta versión, en vez de utilizar la ley de Hebb para entrenar la red, se utiliza un aprendizaje mediante entrenamientos múltiples que permite que la red pueda aplicarse a la resolución de un rango muy amplio de problemas. Además el entrenamiento múltiple se parece más al refuerzo a largo plazo, proceso descrito para sistemas biológicos reales [6][7][8].

Nosotros nos centraremos durante el trabajo en desarrollar el modelo, intentar explicar sus límites, los conceptos básicos conocidos por la comunidad de investigadores, y sobretodo en establecer un algoritmo adecuado para trabajar con él. La importancia del algoritmo radica en que este tipo de modelos, al ser tan rico en parámetros resulta muy difícil de ajustar para lograr un correcto funcionamiento. Además, este modelo concreto añade una autorregulación mediante olvido pasivo que debe ajustarse con mucho cuidado para que todo funcione. Unida a la dificultad con el ajuste de los parámetros, hay que señalar que este tipo de sistemas dinámicos puede simularse de muchas formas distintas, por eso un algoritmo específico para realizar las simulaciones es necesario.

Parte de los parámetros se estiman de forma analítica utilizando mecánica estadística, mientras el resto se analizan mediante simulaciones numéricas del modelo y medida sobre los resultados.

Para poder simular el modelo se ha escrito un código en Matlab que aparece en el Apéndice. Dicho código permite realizar diversos tipos de simulaciones con la red y explorar algunas de sus aplicaciones.

2 Modelo de red neuronal tipo Hopfield con neuronas tipo McCulloch Pitts estocásticas.

2.1 Descripción del modelo.

Aquí describimos el modelo que vamos a estudiar y del cual queremos aprender las características principales. En redes neuronales la elección de un modelo va condicionada por el tipo de simulaciones que se deseen realizar con él. En nuestro caso deseamos estudiar las capacidades computacionales, por lo que nos alejamos de los modelos biologicamente equivalentes y nos acercamos más a los llamados modelos funcionales. El primero de dichos modelos funcionales fue la red tipo Hopfield que funcionaba como memoria asociativa almacenando patrones que luego podían ser recuperados a partir de información parcial respecto al patrón original. Aquí estudiaremos la versión más moderna de dicho modelo funcional.

Las redes neuronales tipo Hopfield consisten en N neuronas que están todas conectadas entre sí. Decimos que la red es completa si existe autoffedback de una neurona en si misma, en nuestro caso no tienen autofeedback [6] porque se ha comprobado que la supresión del mismo hace que la red sea más resistente a entrar en regímenes caóticos [9]. En nuestro caso, además del autofeedback eliminamos la condición de simetría en los pesos entre neuronas, esto quiere decir que cuando la señal va de una neurona a otra vuelve por un camino distinto y por tanto con una ampliación o atenuación distintas. Violando la condición de simetría hacemos que la red sea capaz de acceder a un mayor número de estados.

Además del tipo de conexiones, debemos especificar como tiene lugar la actualización de estados de las neuronas, como nosotros estamos interesados en las propiedades computacionales de la red, trabajaremos con una red sincrona, es decir, el tiempo es discreto y todas las neuronas se actualizan a la vez. En el modelo original de Hopfield la actualización de las neuronas tenía lugar de forma asíncrona, aquí elegimos la actualización sincrona por ser más fácil para las simulaciones, y recuperamos la parte aleatoria introduciendo una neurona estocástica.

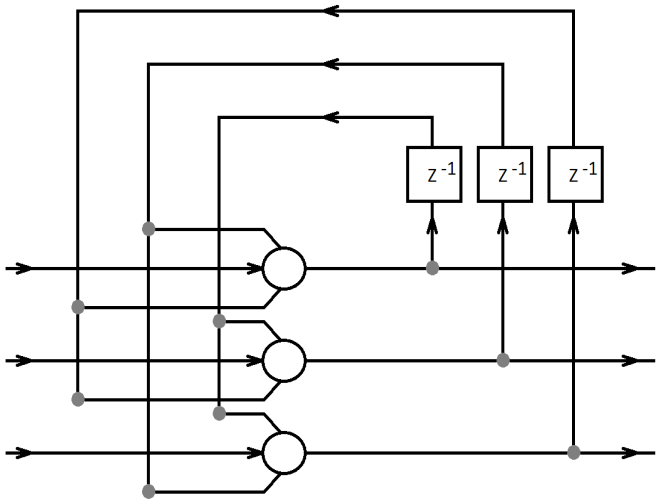


Figura 1. *Red Hopfield con 3 neuronas sin autofeedback y con entradas que se suman a las señales de las neuronas.*

En la figura 1 podemos ver un esquema de como quedaría la red con 3 neuronas. Cada neurona además de las señales de las otras neuronas recibe una señal externa que se suma al resto de señales. Los bloques indican un retraso en la señal que se vuelve a alimentar a la red.

En la misma figura vemos que parte de la señal proviene de la izquierda y parte sale por la derecha. La señal que entra por la izquierda es la señal externa que reciben las neuronas. La parte que sale por la derecha es la lectura que hacemos del estado de la red. En principio estas dos conexiones son para interaccionar con la red, pero pueden ser utilizadas para conectar la red con otras redes.

Para la evolución de este tipo de red neuronal alternamos dos tipos de evolución, una evolución:

Evolución dinámica. Se mide usando t. Es durante la cual evolucionan las neuronas y cambia el estado de la red. No es comparable con el tiempo real que tardan las neuronas biológicas en evolucionar, pero si que se supone que la evolución dinámica es mucho más rápida que el siguiente tipo de evolución.

Evolución de aprendizaje. Se mide usando T. Es durante la cual tiene lugar la actualización de los pesos. Se supone que cada cierto número de pasos de evolución dinámica tiene lugar una evolución de aprendizaje para modificar los pesos en función de la actividad de las neuronas. Esto hace que el aprendizaje pueda considerarse como no supervisado y se acerca más al modelo biológico [3].

Combinando los dos tipos de evolución podemos escribir el estado de la neurona i de la red en el instante t+T como $s_i^T(t) \in \{-1,1\}$.

Los periodos dinámicos tienen una duración de τ intervalos, mientras que la actualización de los pesos dura un sólo periodo. Esto además de ser biologicamente más correcto nos va a permitir utilizar distintas estrategias para los algoritmos de entrenamiento de la red porque el entrenamiento forma parte del proceso de simulación.

Este tipo de redes suelen llamarse memorias asociativas porque se les puede hacer memorizar patrones y luego dichos patrones pueden ser recuperados a partir de señales con cierto nivel de ruido [10]. Un uso muy común de este tipo de redes suele ser el de reconocimiento de caracteres o de formas. Si se acondiciona adecuadamente la señal de entrada, también pueden servir para segmentar imágenes o para actuar como filtro en señales con ruido. Nosotros vamos a centrarnos en el funcionamiento como memoria asociativa.

Pasamos ahora a describir el modelo de neurona utilizado y las ecuaciones que rigen el funcionamiento del modelo.

Para el modelo de neurona utilizamos la versión estocástica [4,11] [6] del modelo de McCulloch-Pitts que podemos ver en la figura 2. Dicho modelo combina el modelo determinista de McCulloch-Pitts y la actualización aleatoria de neuronas del modelo original de red Hopfield [12]. por lo que en conjunto la red tendrá un funcionamiento similar al modelo original de Hopfield.

Las ecuaciones que rigen la evolución del modelo son las siguientes [13]. Para la actualización de los estados de las neuronas:

$$h_i^T(t+1)=\sum_{j\neq i}^{N} w_{ij}^T S_j^T(t)+\varepsilon\zeta_i(t) \tag{1}$$

$$s_i^T(t+1)=\begin{cases} +1\ with\ probability\ 1/2\left(1+\tanh\left(\beta h_i^T(t+1)\right)\right) \\ -1\ with\ probability\ 1-1/2\left(1+\tanh\left(\beta h_i^T(t+1)\right)\right) \end{cases} \tag{2}$$

El factor $\beta=1/temperatura$ sirve para simular ruido térmico en la red y garantizar que pueden existir transiciones entre estados. Suele definirse por analogía con la temperatura en los vidrios de spin. El término $\varepsilon\zeta_i$ representa el patrón externo aplicado sobre la neurona i en el instante t. El factor ε indica la intensidad del patrón externo.

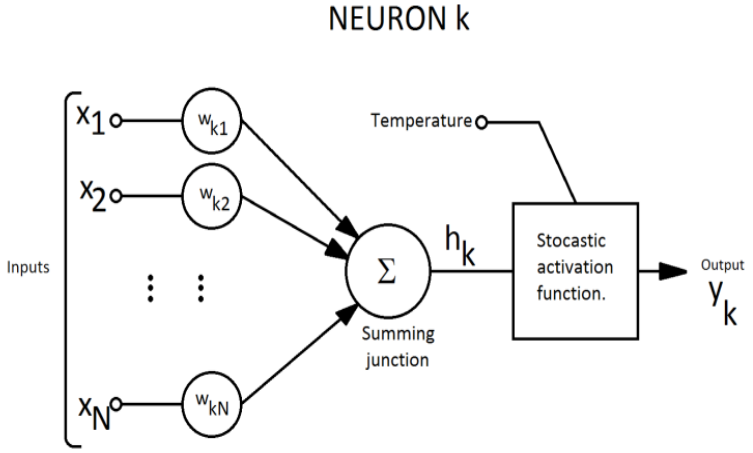


Figura 2. *Modelo de neurona estocástico.*

Para la actualización de la matriz de pesos seguiremos la siguiente ecuación basada en [14] [15] [16] [18] [13] [15] [17] [10] [6] [19][39]:

$$w^{T+1}=\lambda w^T+\frac{\gamma}{N}(\hat{s^T})(\hat{s^T}) \tag{3}$$

Donde hemos hecho:

$$\hat{s^T}=\frac{1}{\tau}\sum_{k=0}^{\tau} s^{T-1}(k) \tag{4}$$

El término γ recibe el nombre de tasa de aprendizaje (learning rate). En nuestro caso lo tomaremos con valor 1 para simplificar los resultados y el análisis. El término λ es el olvido pasivo (forgetting rate) y sirve para que la red no se sature a medida que aprende nuevos patrones [15]. Para $\lambda<1$ tenemos que la red olvida los patrones de forma pasiva con cada aprendizaje. Para $\lambda=1$ no se olvidan patrones, pero como discuten en [15], esta situación lleva a que tras aprender muchos patrones la red adquiera un comportamiento caótico. Para valores mayores que 1 la red refuerza los patrones aprendidos anteriormente y acaba cayendo en régimen caótico.

Por último, para preservar la condición de no autofeedback, tras cada entrenamiento se hace:

$$w_{ii}=0 \tag{5}$$

Este tipo de modelo que hemos descrito recibe el nombre genérico de Stochastic Hopfield Neural Network, SHNN, y se está utilizando sobretodo para analizar los efectos de ciertas enfermedades neurológicas sobre la capacidad para recordar.

2.2 Tipos de entrenamientos.

Antes de seguir adelante con el estudio del modelo conviene repasar ciertos aspectos relacionados con el entrenamiento.

Como en el resto de modelos de red, no existe una única manera de entrenar las redes Hopfield, sin embargo en el caso de estas el tipo de entrenamiento define la aplicación para la que va a servir la red.

Basándonos en las referencias encontradas podemos decir que existen principalmente los siguientes tipos o categorías de entrenamientos [1,16] [7][39]:

Entrenamiento único (Oneshootlearning): El entrenamiento tiene lugar una sola vez durante la simulación generalmente al principio. Este tipo de entrenamiento suele utilizarse cuando no se desea un estudio dinámico de la red sino que se quieren aprovechar sus propiedades para realizar algún tipo de cálculo o bien una tarea específica de la que ya se tienen ejemplos.

- *Ley de Hebb:* Es la primera forma encontrada para enseñar patrones a una red neuronal. Está basado en la ley enunciada por Hebb en su libro sobre neurología. Consiste en fijar la matriz de pesos utilizando para ello los patrones que deseamos sean recordados.
- *Pesos a partir de función de energía:* La analogía entre redes tipo Hopfield y vidrios de spin permite definir funciones de energía que sirven para determinar la matriz de pesos de la red. [20]. Este tipo de entrenamiento puede ser útil para resolver problemas de optimización como el Traveling Salesman Problem. En ocasiones este tipo de entrenamiento también está asociado al llamado problema inverso, conocido el comportamiento del vidrio de spin, recuperar la función de energía que da el tipo de interacciones entre spines.

Múltiples entrenamientos (Multipleshootlearning): El entrenamiento tiene lugar de forma periódica y la matriz de pesos cambia de acuerdo a la actividad previa de la red. Este tipo de entrenamiento es el adecuado para imitar el funcionamiento del cerebro o cuando se desea que la red vaya adaptándose al uso que se dé de ella.

- *LTP: Refuerzo a largo plazo (Long Term Potentiation)* es el tipo de entrenamiento en el que la matriz de pesos va modificándose a largo plazo en función de la actividad previa y se refuerzan las conexiones que tienen lugar de forma simultanea [21]. Según [11] cuando los pesos toman valores positivos se corresponde con conexión excitatoria, cuando toman valores negativos recibe el nombre de conexión inhibitoria. [22]. Suele llamarse LTP,

pero también incluye la LTD ó Long Term Depresion que consiste en que los pesos de neuronas que no suelen activarse a la vez van reduciéndose poco a poco [6]. Este tipo de entrenamiento es adecuado para simular redes en las que los patrones a recordar no son fijos o en las que se deseé realizar una adaptación al uso [10].

- *Olvido pasivo (Passive forgetting):* Más que un entrenamiento propiamente dicho es una modificación del LTP en el que en cada entrenamiento la matriz de pesos del paso anterior pasa a la siguiente iteración multiplicada por un factor que hace que se vayan olvidando los patrones más viejos [6]. Este factor se ha estudiado y se sabe que sirve para que la matriz de pesos no se sature y que la red siga funcionando de forma correcta pese a que haya sido entrenada varias veces [15].

3 Distancia Hamming.

En esta sección hablamos de una herramienta muy utilizada a la hora de analizar redes Hopfield, la distancia Hamming y el solapamiento [39].

3.1 Distancia Hamming y Solapamiento.

La SHNN funciona a partir de patrones que son memorizados y a partir de señales que le son introducidas. Para evaluar la eficiencia de la red en la recuperación de patrones se introduce una medida de la distancia entre patrones. Dicha medida es la llamada distancia Hamming [1].

Esta medida de distancia se define en principio para cadenas de letras de la siguiente manera. La distancia Hamming es el número de letras distintas entre dos cadenas de letras de la misma longitud. Por ejemplo entre la cadena “martes” y “mortes” la distancia es uno porque ha cambiado una de las letras. Si hubiésemos cambiado dos letras la distancia sería 2 y así sucesivamente.

Para matrices la distancia es el número de valores distintos entre las dos matrices. La distancia Hamming no mide por cuanto son distintos los valores, sólo si son distintos o no los valores. Como nosotros queremos poder trabajar con matrices (será donde almacenaremos nuestros patrones), necesitamos introducir una distancia entre matrices que sea independiente del tamaño de las mismas. Es por eso que introducimos la distancia Hamming relativa que es simplemente la distancia Hamming dividida entre el número de caracteres que pueden cambiar. En el caso de las cadenas de letras una distancia relativa de 0 indica que las palabras son idénticas, y una distancia de 1 indica que todas las letras son distintas.

En el caso de matrices donde cada elemento de la matriz toma o valor 1 o -1, como en nuestros patrones, una distancia relativa de 0 indica que las matrices son iguales, una distancia de 0,5 indica que las dos matrices son distintas y los elementos han cambiado al azar, y una distancia de 1 indica que la matrices no comparten ningún valor, sin embargo, como sólo son posibles dos valores, esto indica que la matriz es justamente la opuesta. Por eso en matrices que toman valores -1 o 1 la mayor distancia posible entre dos matrices es de 0.5.

Hablamos de distancia porque entre otras propiedades se cumple que si sumamos la distancia de A a B y la distancia de A a C el resultado es mayor o igual que la distancia de B a C. A veces en vez de decir que dos patrones son distintos simplemente decimos que tienen cierto nivel de ruido uno respecto del otro.

La definición matemática de la distancia Hamming relativa, o simplemente distancia Hamming, es la siguiente. Supongamos que tenemos dos vectores con N componentes, A y B, si $A, B \in \{-1,1\}$ entonces podemos escribir la distancia Hamming entre A y B como:

$$H(A,B) = \frac{1}{2}\left(1 - \frac{1}{N}\sum_i A_i B_i\right) \tag{6}$$

Esto nos permite escribir una expresión muy útil: $\sum_i A_i B_i = N(1 - 2\mathrm{H}(A,B))$ (7)

Algunos textos en vez de utilizar la distancia Hamming utilizan el solapamiento [23][24], que se define como:

$$m(A,B)=\frac{1}{N}\sum_i A_i B_i=(1-2\mathrm{H}(A,B)) \tag{8}$$

El solapamiento da una idea de cómo de idénticos son dos vectores Si dos vectores son totalmente idénticos el solapamiento toma valor 1, si son opuestos toma valor -1. Para vectores sin ninguna correlación entre si toma valor cero.

La distancia Hamming sirve en general para evaluar la eficiencia de la red a la hora de recuperar un patrón. Para ello definimos la eficiencia al recuperar el patrón A habiendo obtenido el B cómo

$$Eficience(A,B)=1-H(A,B) \tag{9}$$

Según lo que hemos visto la eficiencia varía entre 0 y 1, correspondiendole el valor 1 a la máxima eficiencia y el valor 0 a la mínima eficiencia (que en nuestro caso se corresponde con obtener el patrón opuesto). Según la definición de distancia que hemos dado, si en vez de obtener el patrón que buscábamos obtenemos un patrón al azar la eficiencia será de 0,5.

3.2 Patrones equiespaciados.

A partir de la distancia Hamming podemos obtener una relación muy útil. Supongamos que tenemos vectores de dimensión N y queremos generar una serie de patrones de tal forma que la distancia mínima entre patrones sea exactamente cierto valor. ¿Cuántos patrones podemos generar de esta forma?

Supongamos que el primer vector toma la forma:

$$A=(1,1,1,1,1,...,1) \tag{10}$$

para que el segundo vector, B diste del primero una cantidad ρ basta con ver que:

$$\rho=H(A,B)=\frac{1}{2}\left(1-\frac{N-R}{N}\right) \quad ; \quad R:=numero\ de\ términos\ distintos \tag{11}$$

y entonces con hacer que los R primeros términos de A sean distintos obtenemos un vector B que dista ρ de A:

$$B=(-1,-1,-1,..,-1,1,...,1) \tag{12}$$

Si ahora queremos obtener un vector C que diste al menos ρ tanto de A como de B necesitamos hacer que sus R/2 primero términos sean distintos de A pero iguales a B, que sus R/2 siguientes términos sean iguales a los de A y distintos de los de B, y que sus siguientes R/2 términos sean distintos tanto de los de B como de los de A. Siguiendo este esquema obtenemos un patrón como el de la siguiente figura 3.

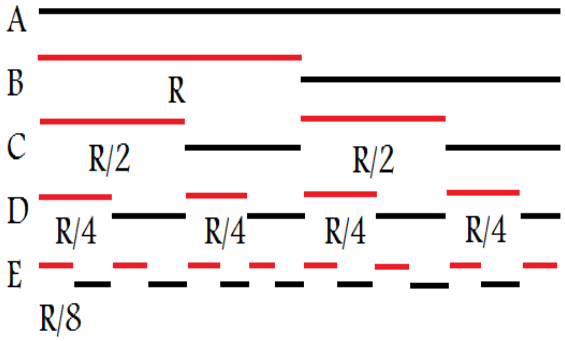


Figura 3. *Esquema de cómo ir obteniendo patrones equiespaciados.*

Esta forma de construir patrones puede continuar mientras el cociente entre R y las potencias de de 2 sigan dando números enteros. Puede verse que el número de patrones M se obtiene mediante la formula:

$$M \text{ es el máximo entero tal que } \frac{R}{2^{(M-2)}} \in \mathbb{N} \tag{13}$$

Sustituyendo R nos queda:

$$M \text{ es el máximo entero tal que } \frac{N\rho}{2^{(M-3)}} \in \mathbb{N} \tag{14}$$

Si tenemos en cuenta que ρ tiene que variar de forma discreta nos queda que la mínima variación de ρ es:

$$min\ \Delta\rho = \frac{1}{2N} \tag{15}$$

Así el número de patrones para cada valor posible de ρ lo podemos escribir como:

$$M \text{ es el máximo entero tal que } \frac{N\rho}{2^{(M-3)}} \in \mathbb{N} \qquad \rho = 0, \frac{1}{2N}, \frac{2}{2N}, \frac{3}{2N}, \ldots, \frac{N}{2N} \tag{16}$$

En la siguiente gráfica podemos ver cómo varia el número de patrones en función de la distancia que queremos tener, y cuales son los valores permitidos de distancia en función del tamaño N de los vectores para N=100. Puede verse que al combinar entre sí todos las condiciones se obtiene una figura muy compleja. La principal condición es que exigimos que los patrones estén equiespaciados, por eso la gráfica es tan compleja. Si se permite cierto margen para que los patrones no sean exactamente equiespaciados entonces la gráfica se simplifica.

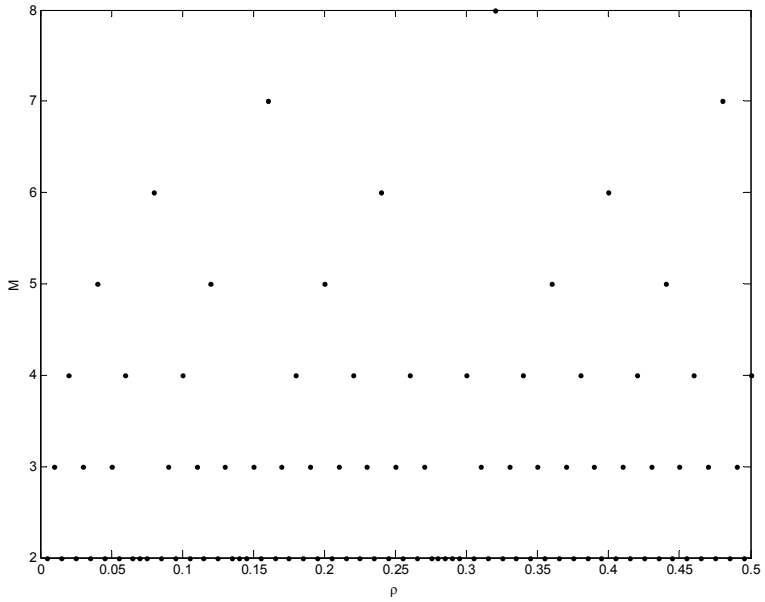


Figura 4. *Número de patrones equiespaciados, M, para vectores de 100 componentes en función de la distancia entre patrones* ρ .

Los patrones equiespacidos van a permitirnos realizar ciertos cálculos analíticos, sin embargo el número de patrones disponibles es tan pequeño que en general trabajaremos con patrones que disten una cierta distancia mínima, sin exigir que sean equiespaciados.

4 Análisis del valor optimo de β en función del resto de parámetros.

En la sección 2 describimos el modelo que vamos a utilizar, y en la sección 3 introdujimos una herramienta para evaluar su funcionamiento. Ahora vamos a empezar a analizar detalles del modelo para entender cómo funciona y cuales son los valores óptimos del mismo.

Según el modelo, tenemos varios parámetros que pueden ser fijados:

N:Número de neuronas. En general vendrá fijado por la aplicación.

τ: Número de iteraciones entre entrenamientos. Tendremos que decidir qué valor utilizar en función del resto de parámetros.

λ: Olvido pasivo. Deberemos buscar el valor optimo para no caer en el caos.

β=1/Temperatura: La temperatura se introduce para hacer que la red escape de estados estacionarios. Habrá que determinar qué valores permiten estabilidad y cuales no.

ε: Es la intensidad con que se introduce la señal externa. No será necesario fijarlo, pero nos dará una referencia para fijar la temperatura.

γ: Es el learning rate. Por simplicidad lo tomaremos con valor 1.

De momento vamos a suponer que fijamos los parámetros N, τ, λ, ε, γ; y que sólo nos interesa estudiar β.

4.1 Condiciones de estabilidad y convergencia.

Por regla general hay dos tipos de procedimientos para estudiar la estabilidad o la evolución de los estados en las redes tipo Hopfield [25] [39].

El primero de ellos es la llamada **aproximación del campo medio** (**mean field aproximation)** [16][19]. Consiste en suponer que cada neurona en vez de recibir una señal específica de las otras neuronas ve un campo medio producido por el estado del resto de neuronas de la red y mediado por la matriz de pesos. Aquí no la vamos a utilizar porque la aproximación del campo medio funciona bien cuando N toma valores muy grandes y cuando el aprendizaje es del tipo oneshootlearning. La aproximación del campo medio falla con aprendizajes tipo multipleshoot porque en parte elimina los detalles de la matriz de pesos que introduce el multipleshoot.

La otra técnica clásica consiste en el llamado **truco de la réplica** (**replica trick)** [23]. Consiste en hallar la función de partición suponiendo que se puede determinar la función a partir de muchas réplicas del sistema. Cada réplica comienza con un estado distinto y así se explora todo el espacio de estados del sistema. De nuevo este método falla para aprendizajes tipo multipleshoot. El problema está aquí en que habría que tener en cuenta además de todos los inicios posibles, todos los aprendizajes posibles, ya que el aprendizaje ya no ocurre al principio, sino que tiene lugar durante la simulación. Esto hace que analizar el problema mediante la técnica de las réplicas sea inabordable.

Como nuestro objetivo es llegar a simular la red neuronal y podemos extraer de su funcionamiento los mejores valores para los parámetros, optamos por aplicar métodos menos potentes al análisis de los parámetros pero que aun así puedan darnos una idea de qué valores son los adecuados [26].

Vamos a suponer que los entrenamientos ya han tenido lugar y que ahora estamos en una fase en la cual sólo nos interesa recuperar los patrones almacenados. Empezamos planteando cual es la forma que toma la matriz de pesos tras dichos entrenamientos. Desarrollando la expresión (3) nos queda:

$$w_{ij}^{T+1}=\frac{\gamma}{N\tau^2}\sum_{k=1}^{\tau}X_i^T(k)X_j^T(k)+\lambda\frac{\gamma}{N\tau^2}\sum_{k=1}^{\tau}Y_i^{T-1}(k)Y_j^{T-1}(k)$$
$$+\lambda^2\frac{\gamma}{N\tau^2}\sum_{k=1}^{\tau}Z_i^{T-2}(k)Z_j^{T-2}(k)+....+\lambda^T\frac{\gamma}{N\tau^2}\sum_{k=1}^{\tau}V_i^0(k)V_j^0(k) \tag{17}$$

Para simplificar esta notación, vamos a suponer que hemos enseñado a la red M patrones y que los nombramos como $\overset{i}{\xi}$ indicando la i que es el patrón número i y que se ha enseñado a la red hace i aprendizajes. (Luego el ultimo patrón que le hemos enseñado a la red es el $\overset{0}{\xi}$). Si además suponemos que el aprendizaje ha sido tal que durante casi todas las iteraciones hemos mantenido el mismo patrón en la red podemos aproximar y escribir:

$$w_{ij}^{T+1}\approx\frac{\gamma}{N}\sum_{k=0}^{M}\lambda^k\overset{k}{\xi}_i\overset{k}{\xi}_j \tag{18}$$

Vemos que cuando tomamos $\lambda=1$ $\gamma=1$ esta regla para el aprendizaje resulta ser de nuevo la ley de Hebb originalmente utilizada por Hopfield [12,27] [7].

4.1.1 Primera condición de estabilidad.

Supongamos que introducimos en la red una señal $\overset{1}{S}'$ y que dicha señal genera tras un cierto tiempo un estado $\overset{1}{S}$ en las neuronas de la red. Veamos cuales son las condiciones que se deben cumplir para que dicho patrón sea estable y se mantenga. Para ello, vamos a suponer que estamos en el paso de tiempo $T+t$ y que queremos obtener el siguiente estado de la red:

$$h_i^T(t+1)=\sum_{j\neq i}^{N}w_{ij}^T\overset{1}{S}_j+\alpha\overset{1}{S}_i' \tag{19}$$

A partir de este valor hallamos el siguiente estado de la red:

$$s_i^T(t+1)=\begin{cases}+1\ with\ probability\ 1/2\left(1+\tanh\left(\beta h_i^T(t+1)\right)\right)\\-1\ with\ probability\ 1-1/2\left(1+\tanh\left(\beta h_i^T(t+1)\right)\right)\end{cases} \tag{2}$$

para estar seguros de que obtendremos $S_i^T(t+1)=\overset{1}{S}_i$ deben cumplirse las siguientes condiciones:

$$\overset{1}{S}_i=1\rightarrow\tanh\left(\beta h_i^T(t+1)\right)\geq 1\rightarrow h_i^T(t+1)\geq\delta^+(\beta) \tag{20}$$

$$\overset{1}{S}_i=-1\rightarrow\tanh\left(\beta h_i^T(t+1)\right)\leq -1\rightarrow h_i^T(t+1)\leq\delta^-(\beta,1) \tag{21}$$

Donde $\delta^+(\beta,1)=\frac{arctanh(1)}{\beta}$; $\delta^-(\beta,1)=\frac{arctanh(-1)}{\beta}$ (22)

Estas dos condiciones pueden comprimirse en una sola expresión, para que el estado $\overset{1}{S}$ sea estable debe cumplirse que:

$$|h_i^T(t+1)| \geq |\delta(\beta,1)| \tag{23}$$

donde $\delta(\beta,a)=\frac{arctanh(a)}{\beta}$ (24)

Utilizamos ahora la expresión (18). Recordar que para utilizarla estamos suponiendo que el entrenamiento fue tal que cada patrón se mantuvo tanto tiempo en la red que podemos suponer que en promedio la red se mantuvo siempre en dicho patrón durante el entrenamiento del patrón. Nos queda:

$$h_i^T(t+1)=\sum_{j\neq i}^{N}\frac{\gamma}{N}\sum_{k=0}^{M}\lambda^k\overset{k}{\xi}_i\overset{k}{\xi}_j\overset{1}{S}_j+\alpha\overset{1}{S}_i' \tag{25}$$

Utilizando la expresión (7):

$$\sum_j\overset{k}{\xi}_j\overset{1}{S}_j=N(1-2\mathrm{H}(\overset{k}{\xi},\overset{1}{S})) \tag{26}$$

Podemos escribir:

$$h_i^T(t+1)=\gamma\sum_{k=0}^{M}\lambda^k\overset{k}{\xi}_i(1-2\mathrm{H}(\overset{k}{\xi},\overset{1}{S}))+\alpha\overset{1}{S}_i' \tag{27}$$

Y así la condición de estabilidad puede expresarse:

$$|\gamma\sum_{k=0}^{M}\lambda^k\overset{k}{\xi}_i(1-2\mathrm{H}(\overset{k}{\xi},\overset{1}{S}))+\alpha\overset{1}{S}_i'|\geq|\delta(\beta,1)| \tag{28}$$

Como $\overset{1}{S}_i'\overset{1}{S}_i'=1$ podemos sacar factor común y sacar un término fuera del valor absoluto al tener módulo 1.

$$|\gamma\sum_{k=0}^{M}\overset{1}{S}_i'\lambda^k\overset{k}{\xi}_i(1-2\mathrm{H}(\overset{k}{\xi},\overset{1}{S}))+\alpha|\geq|\delta(\beta,1)| \tag{29}$$

Utilizando desigualdades podemos poner:

$$|\alpha|>|\delta(\beta,1)|-|\gamma\sum_{k=0}^{M}\lambda^k\overset{1}{S}_i'\overset{k}{\xi}_i(1-2\mathrm{H}(\overset{k}{\xi},\overset{1}{S}))| \tag{30}$$

Esto nos da una primera condición débil. Si se cumple que

$$|\alpha|>|\delta(\beta,1)| \tag{31}$$

entonces el patrón $\overset{1}{S}_i$ será estable, sea o no parecido a los patrones almacenados. Como se tiene que $\lim_{\beta\to 0}|\delta(\beta,1)|\to\infty$ lo que podemos hacer es rebajar la condición y en vez de exigir valor 1 pedir por ejemplo valor 0,99, así tenemos una idea de cual debe ser el valor de α.

Otra forma de plantear la ecuación es la siguiente:

$$|\sum_{k=0}^{M} \lambda^k S_i^{1\prime} \xi_i^k (1-2\mathrm{H}(\overset{k}{\xi}, \overset{1}{S}))| > \frac{|\delta(\beta,1)|-|\alpha|}{\gamma} \tag{32}$$

La constante γ que representa el reinforment learning siempre es positiva.

Si ahora nos fijamos en el patrón número m y suponemos que es el más cercano a la señal $S^{1\prime}$ podemos escribir:

$$|\lambda^m S_i^{1\prime} \xi_i^m (1-2\rho) + \sum_{\substack{k=0 \\ k\neq m}}^{M} \lambda^k S_i^{1\prime} \xi_i^k (1-2\mathrm{H}(\overset{k}{\xi}, \overset{1}{S}))| > \frac{|\delta(\beta,1)|-|\alpha|}{\gamma} \tag{33}$$

donde ρ es la distancia al patrón m.

Sacamos factor común a $S_i^{1\prime} \xi_i^m$ y nos queda:

$$|\lambda^m (1-2\rho) + \sum_{\substack{k=0 \\ k\neq m}}^{M} \lambda^k \xi_i^m \xi_i^k (1-2\mathrm{H}(\overset{k}{\xi}, \overset{1}{S}))| > \frac{|\delta(\beta,1)|-|\alpha|}{\gamma} \tag{34}$$

Y volviendo a utilizar desigualdades nos quedamos con una condición de estabilidad mucho más fuerte que es la siguiente:

$$|\lambda^m (1-2\rho)| > \frac{|\delta(\beta,1)|-|\alpha|}{\gamma} - |\sum_{\substack{k=0 \\ k\neq m}}^{M} \lambda^k \xi_i^m \xi_i^k (1-2\mathrm{H}(\overset{k}{\xi}, \overset{1}{S}))| \tag{35}$$

Esta expresión nos indica el valor máximo que puede tomar m en función de los parámetros que hallamos escogido. Puede verse que depende mucho de los productos $\xi_i^m \xi_i^k$ por lo que no será una situación analítica, sino que dependerá en gran medida de los patrones que hallamos utilizado y del orden en que hallan sido introducidos. Esta expresión en principio serviría para decirnos cual es la capacidad de la red (número máximo de patrones que pueden almacenarse), pero parece que no depende del tamaño N de la misma, cuando es algo bien establecido que el número de patrones que pueden almacenarse depende del tamaño de la red. En realidad esta expresión si que depende de N, pero de forma indirecta, depende de N a través de los $\xi_i^m \xi_i^k$, porque cuanto mayor sea N mayor número de patrones distintos podremos tener y por tanto menor podrá ser la suma, permitiendo mayor valor de m.

4.1.2 Segunda condición de estabilidad.

Pasamos ahora a otro tipo de análisis para terminar de acotar los valores de β. Aquí aplicaremos una técnica similar al truco de las réplicas para la función de partición [24] [28].

Según nuestro modelo (2), dado un valor de $h_i^T(t+1)$ el siguiente estado de la red viene dado por:

$$s_i^T(t+1)=\begin{cases} +1 \text{ with probability } 1/2\left(1+\tanh\left(\beta\, h_i^T(t+1)\right)\right) \\ -1 \text{ with probability } 1-1/2\left(1+\tanh\left(\beta\, h_i^T(t+1)\right)\right) \end{cases} \tag{2}$$

donde

$$h_i^T(t+1)=\sum_{j\neq i}^{N}\frac{\gamma}{N}\sum_{k=0}^{M}\lambda^k\overset{k}{\xi}_i\overset{k}{\xi}_j\overset{1}{S}_j+\varepsilon\overset{1}{S}_i' \tag{36}$$

Supongamos ahora que realizamos varias simulaciones para ver cual es la evolución promedio que sigue la red. Para ello vamos a tomar $\overset{1}{S}_i'$ como si fuese una variable aleatoria tal que $\langle\overset{1}{S}_i'\rangle=\overset{m}{\xi}_i'$ y con desviación típica ρ tal que $H(\overset{1}{S}_i',\overset{m}{\xi}_i')<\rho$. Esto nos permite escribir:

$$\langle h_i\rangle=\sum_{j\neq i}^{N}\frac{\gamma}{N}\sum_{k=0}^{M}\lambda^k\overset{k}{\xi}_i\overset{k}{\xi}_j\langle\overset{1}{S}_j\rangle+\alpha\langle\overset{1}{S}_i'\rangle=\sum_{j\neq i}^{N}\frac{\gamma}{N}\sum_{k=0}^{M}\lambda^k\overset{k}{\xi}_i\overset{k}{\xi}_j\langle\overset{1}{S}_j\rangle+\varepsilon\overset{m}{\xi}_i \tag{37}$$

Por otro lado, utilizando (2) podemos escribir:

$$prob\left(s_i^T(t+1)=1\right)=\frac{1}{2}\left(1+\tanh\left(\beta\langle h_i^T(t+1)\rangle\right)\right) \tag{38}$$

lo que nos permite escribir:

$$\begin{aligned}\langle s_i\rangle&=(+1)\,prob\left(s_i=1\right)+(-1)\,prob\left(s_i=-1\right)\\&=\frac{1}{2}\left(1+\tanh\left(\beta\langle h_i\rangle\right)\right)-\frac{1}{2}\left(1-\tanh\left(\beta\langle h_i\rangle\right)\right)\\&=\tanh\left(\beta\langle h_i\rangle\right)\end{aligned} \tag{39}$$

Y si sustituimos nos queda

$$\langle s_i\rangle=\tanh\left(\beta\sum_{j\neq i}^{N}\frac{\gamma}{N}\sum_{k=0}^{M}\lambda^k\overset{k}{\xi}_i\overset{k}{\xi}_j\langle\overset{1}{S}_j\rangle+\varepsilon\,\beta\overset{m}{\xi}_i\right) \tag{40}$$

Utilizando ahora la expresión para el solapamiento (8) obtenemos:

$$m\left(\langle s\rangle,\overset{p}{\xi}\right)=\frac{1}{N}\sum_i\overset{p}{\xi}_i\tanh\left(\beta\sum_{j\neq i}^{N}\frac{\gamma}{N}\sum_{k=0}^{M}\lambda^k\overset{k}{\xi}_i\overset{k}{\xi}_j\langle\overset{1}{S}_j\rangle+\varepsilon\,\beta\overset{m}{\xi}_i\right) \tag{41}$$

y utilizando de nuevo el solapamiento

$$m\left(\langle s\rangle,\overset{p}{\xi}\right)=\frac{1}{N}\sum_i\overset{p}{\xi}_i\tanh\left(\beta\frac{\gamma}{N}\sum_{k=0}^{M}\lambda^k\overset{k}{\xi}_i m\left(\langle s\rangle,\overset{k}{\xi}\right)+\varepsilon\,\beta\overset{m}{\xi}_i\right) \tag{42}$$

Sacamos fuera del sumatorio el patrón m

$$m\left(\langle s\rangle,\overset{p}{\xi}\right)=\frac{1}{N}\sum_i\overset{p}{\xi}_i\tanh\left(\beta\lambda^m\gamma\overset{m}{\xi}_i m\left(\langle s\rangle,\overset{m}{\xi}\right)+\beta\frac{\gamma}{N}\sum_{\substack{k=0\\k\neq m}}^{M}\lambda^k\overset{k}{\xi}_i m\left(\langle s\rangle,\overset{k}{\xi}\right)+\varepsilon\,\beta\overset{m}{\xi}_i\right) \tag{43}$$

Como los patrones toman valores +1 o -1 podemos sacar el patrón m fuera de la tangente:

$$m\left(\langle s\rangle,\overset{p}{\xi}\right)=\frac{1}{N}\sum_i\overset{p}{\xi}_i\overset{m}{\xi}_i\tanh\left(\beta\lambda^m\frac{\gamma}{N}m\left(\langle s\rangle,\overset{m}{\xi}\right)+\beta\frac{\gamma}{N}\sum_{\substack{k=0\\k\neq m}}^{M}\lambda^k\overset{k}{\xi}_i\overset{m}{\xi}_i m\left(\langle s\rangle,\overset{k}{\xi}\right)+\varepsilon\,\beta\right) \tag{44}$$

Supongamos ahora que trabajamos en el régimen de valores de β que hacen posible desarrollar la tangente hiperbólica en serie de potencias a primer orden:

$$m(\langle s\rangle,\overset{p}{\xi})=\frac{1}{N}\sum_i \overset{p}{\xi}_i\overset{m}{\xi}_i(\beta\lambda^m\frac{\gamma}{N}m(\langle s\rangle,\overset{m}{\xi})+\beta\frac{\gamma}{N}\sum_{\substack{k=0\\k\neq m}}^{M}\lambda^k\overset{k}{\xi}_i\overset{m}{\xi}_i m(\langle s\rangle,\overset{k}{\xi})+\varepsilon\beta)+\mathrm{O}(beta^2) \tag{45}$$

$$m(\langle s\rangle,\overset{p}{\xi})\approx\beta\lambda^m\frac{\gamma}{N}m(\langle s\rangle,\overset{m}{\xi})m(\overset{m}{\xi},\overset{p}{\xi})+\beta\frac{\gamma}{N}\sum_{\substack{k=0\\k\neq m}}^{M}\lambda^k m(\overset{p}{\xi},\overset{k}{\xi})m(\langle s\rangle,\overset{k}{\xi})+\varepsilon\beta\, m(\overset{m}{\xi},\overset{p}{\xi}) \tag{46}$$

comparamos ahora el solapamiento de $\langle s\rangle$ con m respecto del solapamiento con otro patrón cualesquiera p:

$$\begin{aligned} m(\langle s\rangle,\overset{m}{\xi})-m(\langle s\rangle,\overset{p}{\xi})&\geqslant\beta\lambda^m\frac{\gamma}{N}m(\langle s\rangle,\overset{m}{\xi})+\beta\frac{\gamma}{N}\sum_{\substack{k=0\\k\neq m}}^{M}\lambda^k m(\overset{p}{\xi},\overset{k}{\xi})m(\langle s\rangle,\overset{k}{\xi})+\varepsilon\beta\\ &-(\beta\lambda^m\frac{\gamma}{N}m(\langle s\rangle,\overset{m}{\xi})m(\overset{m}{\xi},\overset{p}{\xi})+\beta\frac{\gamma}{N}\sum_{\substack{k=0\\k\neq m}}^{M}\lambda^k m(\overset{p}{\xi},\overset{k}{\xi})m(\langle s\rangle,\overset{k}{\xi})+\varepsilon\beta\, m(\overset{m}{\xi},\overset{p}{\xi}))\\ &=\beta\lambda^m\frac{\gamma}{N}m(\langle s\rangle,\overset{m}{\xi})(1-m(\overset{m}{\xi},\overset{p}{\xi}))+\varepsilon\beta(1-m(\overset{m}{\xi},\overset{p}{\xi})) \end{aligned} \tag{47}$$

Despejando obtenemos la siguiente expresión válida para todo p:

$$m(\langle s\rangle,\overset{m}{\xi})\ \geqslant\frac{\varepsilon\beta(1-m(\overset{m}{\xi},\overset{p}{\xi}))+m(\langle s\rangle,\overset{p}{\xi})}{1-\beta\lambda^m\frac{\gamma}{N}(1-m(\overset{m}{\xi},\overset{p}{\xi}))} \tag{48}$$

Para patrones equiespaciados una distancia ρ podemos escribir

$$m(\langle s\rangle,\overset{m}{\xi})\ \geqslant\frac{\varepsilon\beta 2\rho+m(\langle s\rangle,\overset{p}{\xi})}{1-\beta\lambda^m\frac{\gamma}{N}2\rho} \tag{49}$$

Debemos establecer ahora un criterio para definir cuando la red ha convergido a un patrón dado. Decimos que la red ha convergido al patrón m cuando el solapamiento de la red con el patrón m es igual a 1. Por lo que para que la red converja al patrón m debe cumplirse (50):

$$1\ \geqslant\frac{\varepsilon\beta 2\rho+(1-2\rho)}{1-\beta\lambda^m\frac{\gamma}{N}2\rho} \tag{50}$$

Volvemos ahora con la condición de linealidad de la tangente hiperbólica. Si fijamos un cierto valor μ como el máximo valor para el cual sirve la aproximación lineal, entonces podemos escribir la siguiente condición para que sea válida la linealidad de la tangente hiperbólica a partir de la expresión (44):

$$\mu\geqslant\beta\lambda^m\frac{\gamma}{N}m(\langle s\rangle,\overset{m}{\xi})+\beta\frac{\gamma}{N}\sum_{\substack{k=0\\k\neq m}}^{M}\lambda^k\overset{k}{\xi}_i\overset{m}{\xi}_i m(\langle s\rangle,\overset{k}{\xi})+\varepsilon\beta \tag{51}$$

Si ahora además de exigir linealidad exigimos que la red converja al patrón m nos queda:

$$\mu\geqslant\beta\lambda^m\frac{\gamma}{N}+\beta\frac{\gamma}{N}\sum_{\substack{k=0\\k\neq m}}^{M}\lambda^k\overset{k}{\xi}_i\overset{m}{\xi}_i(1-2\rho)+\varepsilon\beta \tag{52}$$

Despejando:

$$\beta \leqslant \frac{\mu N}{\lambda^m \gamma + \gamma \sum_{\substack{k=0 \\ k \neq m}}^{M} \lambda^k \zeta_i^k \zeta_i^m (1-2\rho) + \alpha N} \tag{53}$$

Esta formula no puede resolverse salvo que conozcamos los patrones, pero puede evaluarse numéricamente tomando $\zeta_i^k \zeta_i^m$ como una variable aleatoria uniforme que toma valores 1 ó -1.

4.2 Gráfica de estabilidad para β.

Combinando las condiciones que hallamos en la sección anterior, (31)(35)(50) y (53). Obtenemos la gráfica de la figura 5. Para elaborar dicha gráfica hemos tomado una red de N=100 neuronas con M=7 patrones equiespaciados que según (16) se consiguen cuando la distancia entre patrones es de 0,16 ó de 0,48. Nosotros tomamos $\rho = 0{,}48$ para garantizar la máxima distancia posible entre patrones. Se ha fijado $\alpha = 0{,}8$. Y para la condición de linealidad fijamos $\mu = 0{,}8$ ya que $\tanh(0{,}8) - 0{,}8 = -0{,}1360$ y por tanto es un buen valor.

Las partes que componen la gráfica son las siguientes. La línea negra indica el valor máximo de β por encima del cual se cumple (31) con $\alpha = 0{,}8$ y exigiendo 0,8 en vez de 1. Esto quiere decir que por debajo de este valor los estados de la red son libres de evolucionar a partir de la señal externa y no se quedan fijos en ella. Osea, que para β menor que dicho valor en principio la red puede funcionar como memoria asociativa.

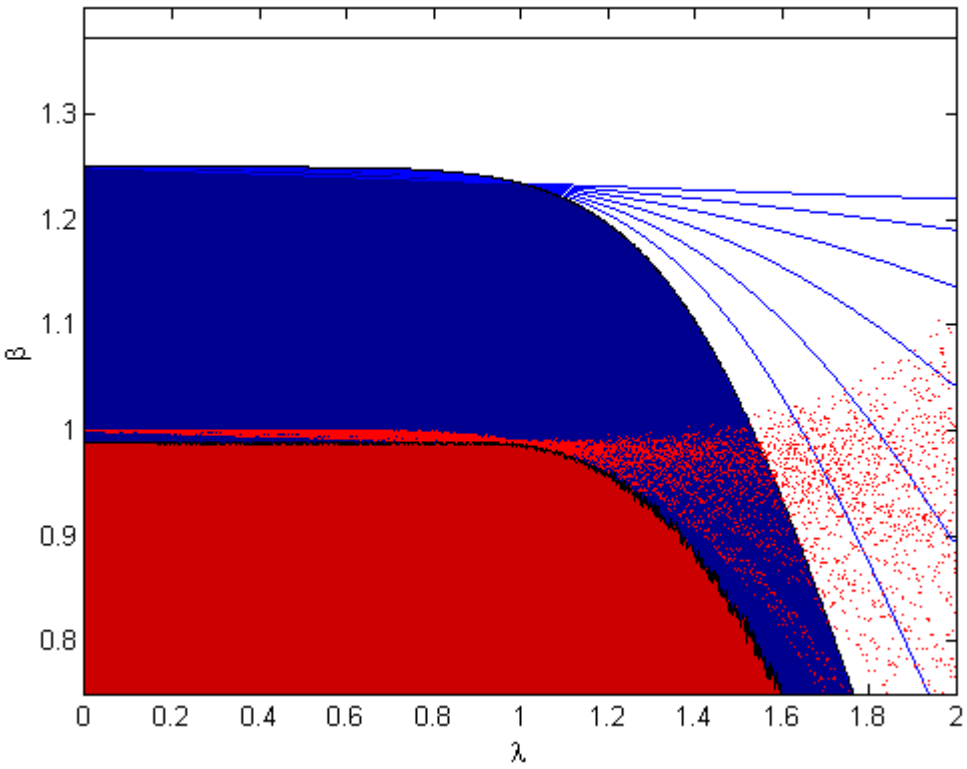


Figura 5. *Mapa de estabilidad y convergencia para β en función de λ. Para una red con N=100 entrenando 7 patrones equiespaciados una distancia 0,46.*

Las líneas azules vienen dadas por (50) e indican el valor máximo de β para poder recuperar cada uno de los 7 patrones. El área azul encerrada entre los ejes y la ultima de las líneas azules indica los valores de beta para los cuales según la aproximación lineal la red converge hacía el patrón almacenado más próximo.
Por ultimo, los puntos rojos vienen dados por (53) utilizando variables aleatorias, y el área roja indica el borde inferior de (53). El área roja indica los valores de β para los cuales se cumple la condición de linealidad con $\mu = 0{,}8$.

A partir de dicha gráfica sabemos que para valores de beta superiores a la línea negra la red evoluciona hacía la señal de la entrada, mientras que para valores de beta dentro de la zona roja la red evoluciona hacía el patrón más próximo a la señal de la entrada. Siempre y cuando el entrenamiento de la red sea como el que hemos supuesto en el análisis [26].

Esta gráfica y la información que contiene nos servirán en la sección siguiente para determinar la forma en que se deben realizar las simulaciones.

5 Análisis numérico mediante simulaciones de la relación entre λ y M.

Con los resultados de la sección anterior podemos en principio ajustar el valor de b en función del resto de parámetros para obtener estabilidad y convergencia a un patrón almacenado. Aunque no lo hemos demostrado aquí, hay estudios donde se analiza la evolución de redes similares y se demuestra que estas siempre evolucionan a un estado estacionario que coincide con uno de sus mínimos de energía [29,40]. La clave está entonces en conseguir que los patrones almacenados sean mínimos de energía para la red.

A continuación realizaremos una serie de simulaciones basándonos en los desarrollos del apartado 4 para establecer un algoritmo de simulación y para escoger los mejores valores para los parámetros. En concreto, estudiaremos el comportamiento de la red en función del parámetro λ, y los resultados que obtenemos en este apartado son los siguientes:

- Un algoritmo para realizar simulaciones con la red.
- Comprobación experimental de que los valores de β hallados en el apartado anterior son correctos.
- Determinación del nivel de ruido aceptado por la red.
- Estimación de los valores de λ para un correcto funcionamiento de la red.
- Determinación de la capacidad de la red cuando se tiene en cuenta el olvido pasivo.
- Resistencia de la red ante daño sinaptico.

5.1 Algoritmo de simulación y entrenamiento.

Para realizar las simulaciones distinguimos entre dos tipos de procesos de la red, almacenamiento y recuperación [30]. El valor de γ ya habíamos dicho que tomaba el valor 1. Para el parámetro α tomamos también el valor 1. El resto de parámetros, y el número de iteraciones se fijarán más adelante [29]. Partiendo de esto proponemos que los procesos de almacenamiento y recuperación sean de la siguiente forma [19][39].

Almacenamiento: Al tratarse de entrenamientos múltiples, durante cierto número de iteraciones expondremos el patrón que queremos enseñar a la red y dejaremos que la red evolucione con dicho patrón a la entrada [31] (experimentalmente hemos visto que con 30 iteraciones es suficiente para que una red de 500 neuronas converja, aunque el número de iteraciones puede variar, por lo que es preferible esperar a que la red se estabilice antes de realizar el entrenamiento). Durante las primeras 30 iteraciones empezaremos con una valor de $\beta=5$ e iremos bajando hasta $\beta=1/5$ de forma lineal. De esta forma, de acuerdo a (31) y a la figura 5, conseguimos sacar a la red del estado en el que esté y la dejamos evolucionar hacía el estado que menos energía tenga en función de la entrada. Una vez que la red alcanza el estado estacionario [29] se la deja evolucionar unas cuantas iteraciones más y se realiza el entrenamiento quedando almacenado dicho patrón en la red.

En principio el almacenamiento ocurrirá al principio de la simulación y luego no se volverá a entrenar la red, salvo en simulaciones donde se desee estudiar el olvido pasivo.

Como vimos en la sección 3, si exigimos patrones equiespaciados entonces para cada tamaño de red hay muy pocos patrones disponibles. Es por eso que en vez de patrones equiespaciados, trabajaremos con patrones que disten al menos cierta distancia ρ entre sí. Esto va a determinar la forma en que tiene lugar la recuperación.

Recuperación: Durante un cierto número de iteraciones expondremos una entrada a la red utilizando una señal que diste una cierta distancia de la señal de uno de los patrones almacenados. La señal se mantiene hasta que la red alcanza un estado estacionario (Experimentalmente se comprueba que con 30 iteraciones una red de 500 neuronas llega al equilibrio). Durante las primeras 30 iteraciones empezaremos con una valor de $\beta=5$ e iremos bajando hasta $\beta=1/5$ de forma lineal. Una vez alcanzado el equilibrio quitaremos la entrada a la red y dejaremos que esta siga evolucionando hacía su mínimo de energía más próximo. En principio, si la entrada no dista mucho del patrón almacenado y si este se encuentra todavía bien almacenado la red evolucionará hacía él.

Combinando estos dos procesos de forma adecuada podemos realizar todas las simulaciones que necesitamos. Los detalles concretos se explican en cada simulación en las secciones siguientes.

Para realizar dichas simulaciones se ha escrito un código en Matlab con ciertos puntos de semejanza con lo expuesto en[31]. Las diversas partes del código aparecen en el apéndice C y posteriores. En este trabajo no podemos extendernos demasiado con el código porque queremos centrarnos en los resultados de las simulaciones, pero hay que destacar que el tiempo y el trabajo invertidos en dicho código superan las 100 horas/hombre. Realizar la programación, depuración y correcto funcionamiento de las simulaciones es un trabajo que se viene realizando desde hace aproximadamente un año. Se ha tenido especial cuidado en que el código final sea lo más simple posible para el modelo y a la vez mantener la eficiencia computacional. Al tratarse de un modelo que permite diversos entrenamientos mezclados con la simulación han sido necesarias algunas medidas especiales a la hora de diseñar el código, pero el resultado final permite realizar cualquier tipo de simulación que permitiese el modelo, por lo que estamos orgullosos del mismo. Además se ha diseñado apostando por el futuro y se ha tenido especial cuidado en hacer que sea posible simular varias redes en paralelo pudiendo así realizar simulaciones donde distintas redes neuronales evolucionan por separado pero pudiendo interaccionar entre ellas de forma esporádica.

5.2 Primera simulación: verificación de correcto funcionamiento y nivel de ruido aceptado por el modelo.

En esta primera simulación queremos determinar si el modelo de red está funcionando como esperamos que funcione y su resistencia al ruido en la señal de entrada. Para ello tomaremos tres valores $\lambda = 0{,}7\,,\,1\,,\,1{,}2$, y tres tamaños de red, N=100, 400, 900 y tres distancias distintas entre patrones $\rho = 0{,}4\,,\;0{,}45\,,\;0{,}5$, y realizaremos la siguiente simulación:

- Generaremos 5 patrones separados entre sí al menos ρ y entrenaremos con ellos la red según el protocolo que hemos establecido.
- La simulación consistirá en intentar recuperar cada patrón a partir de señales con una cantidad cada vez mayor de ruido (mayor distancia al patrón que se quiere recuperar). Para cada distancia al patrón que se desea recuperar se harán 5 intentos con señales distintas para poder evaluar le eficiencia promedio en la respuesta. La eficiencia de la red se calcula según la fórmula (9) de la sección 3: restandole a 1 la distancia entre la salida de la red y la salida esperada, de forma que si coinciden la eficiencia es 1, mientras que si no coinciden para nada la eficiencia es 0,5.

A continuación mostramos los resultados más importantes, los datos completos de la simulación aparecen en el apéndice A. Y el código de Matlab en el apéndice D.

El primer resultado que obtenemos es una confirmación de que el modelo de red está funcionando correctamente. Para $\lambda = 1$ es decir, sin olvido pasivo. Para N=400 y una distancia mínima entre patrones de $\rho = 0{,}4$ obtenemos lo que se puede ver la figura 6.

En la figura vemos que cuando la señal está próxima al patrón que se desea recuperar la eficiencia en la recuperación es total, y se mantiene hasta alcanzar al menos 0,3 unidades de distancia. Esto quiere decir que si la distancia de la señal al patrón es menor que 0,3 entonces la red consigue recuperar el patrón que se almacenó.

Para valores de ruido superiores a 0,4 se observa una caída abrupta de la señal. Esto es razonable, ya que cuando la distancia de la señal al patrón es equivalente a la distancia entre patrones es difícil identificar a qué patrón hace referencia la señal.

A partir de esta simulación podemos concluir que la red está funcionando como se espera que funcione y los valores escogidos para β son adecuados ya que conseguimos pasar de un patrón a otro y recuperarlos siempre que el ruido no sea superior a cierto valor [39].

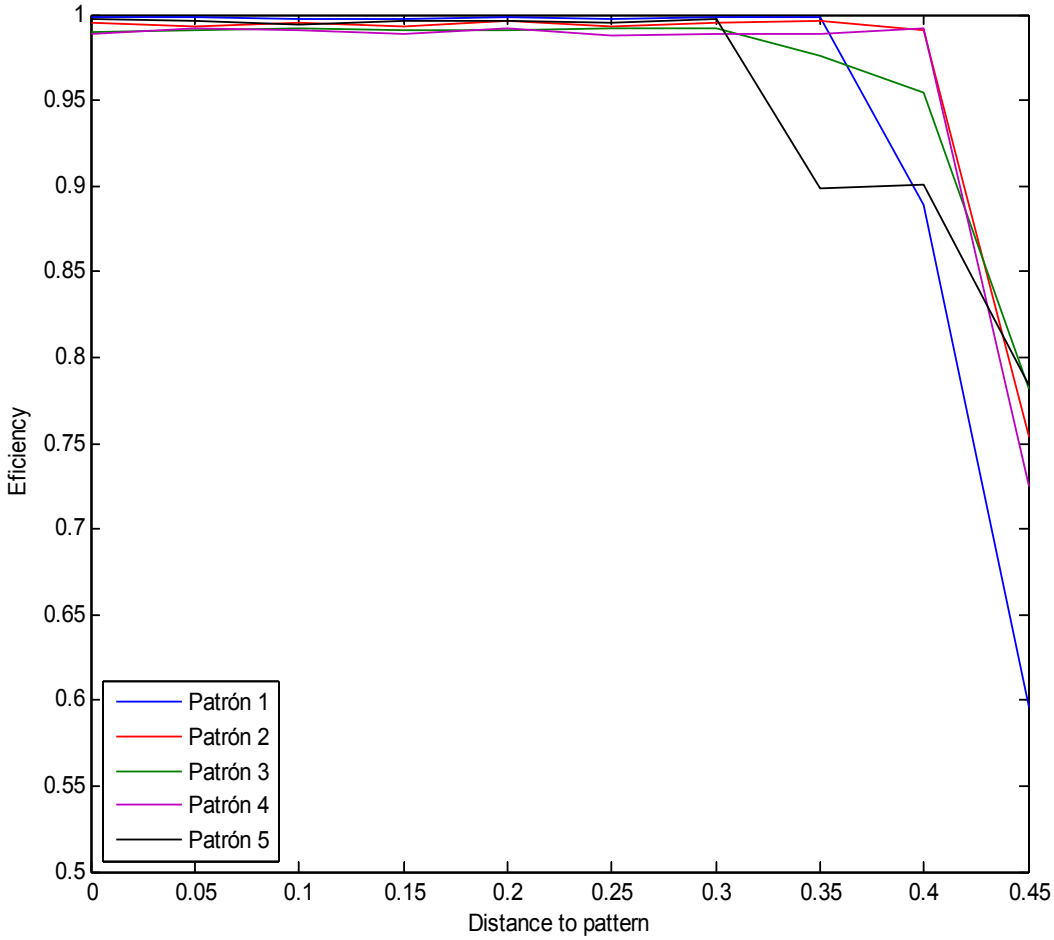


Figura 6. *Para una red con N=400 neuronas, con 5 patrones entrenados con distancia mínima* $\rho=0,4$ *, y con* $\lambda=1$*. Eficiencia de la red en función de la distancia entre la señal de entrada y el patrón que se desea recuperar. Promediado de la eficiencia para 5 intentos distintos para cada patrón.*

La siguiente simulación que mostramos es la correspondiente a $\lambda=1,2$ para los mismos valores de N y de ρ. Esto se correspondería a un mecanismo de compensación ante daño neuronal [32]. Por ejemplo cuando tienen lugar enfermedades como el Alzheimer se ha detectado que para compensar la muerte celular algunas conexiones son reforzadas aumentando la eficiencia de ciertas sinapsis. Por lo que esperamos ver que los primeros patrones almacenados se recuperan con mayor eficiencia que el resto.

En la figura 7 podemos ver los resultados de la simulación. El patrón 1, en azul, es el primero que enseñamos a la red, mientras que el patrón 5, en negro, es el último. Podemos ver que el primer patrón es recordado siempre sin importar la distancia de la señal al patrón. Esto es porque los pesos asociados a dicho patrón han sido reforzados más veces que el resto, por lo que cuando la red duda entre varios patrones se decanta siempre por este. Podemos comprobar esto al ver que el resto de patrones decae su eficiencia muy rápido con la distancia al patrón que se desea recuperar de forma ordenada. Des pues del 1 el 2, después del 2 el 3... y por ultimo el 5º patrón que es el que menos veces se ha reforzado es el que peor eficiencia tiene [10].

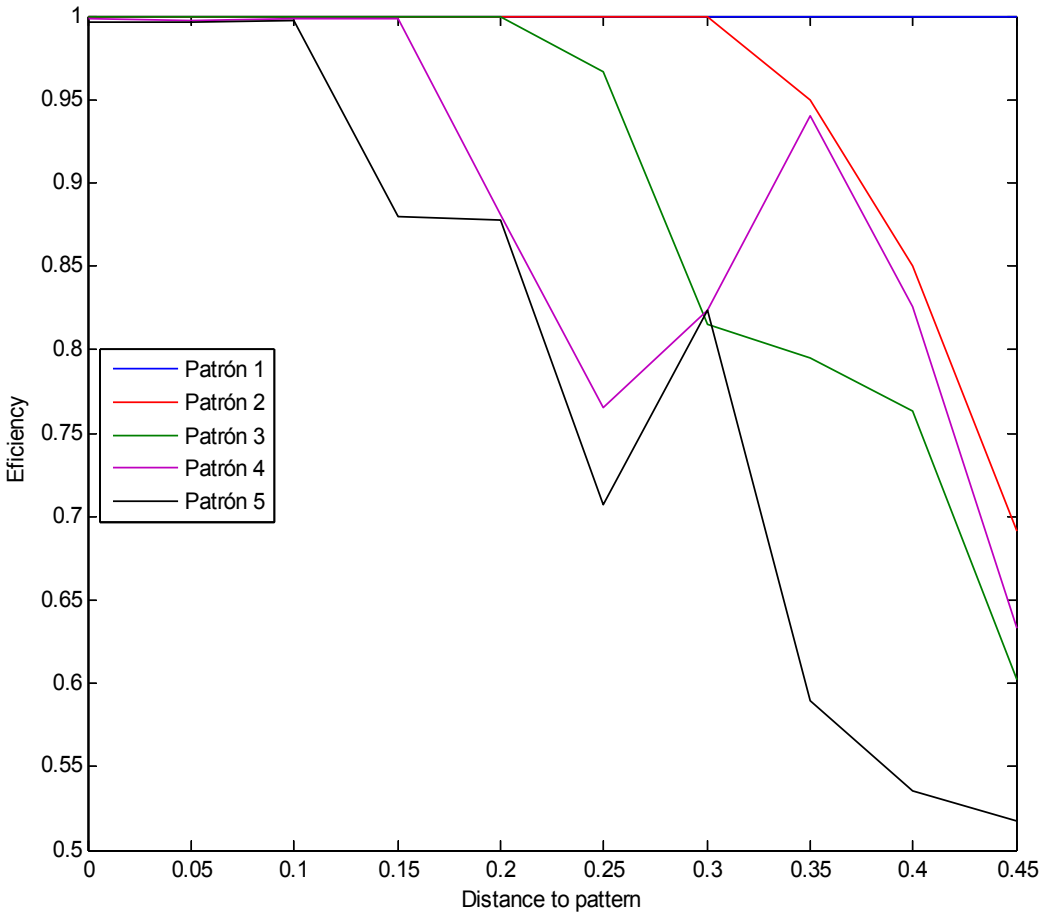


Figura 7. *Para una red con N=400 neuronas, con 5 patrones entrenados con distancia mínima* $\rho=0,4$ *, y con* $\lambda=1,2$*. Eficiencia de la red en función de la distancia entre la señal de entrada y el patrón que se desea recuperar. Promediado de la eficiencia para 5 intentos distintos para cada patrón.*

Estos resultados concuerdan con lo esperado [39], sin embargo parece que el valor de λ es demasiado grande, porque sólo con 5 patrones vemos que ya el 5º patrón sufre una degradación importante. Cómo la degradación es muy importante, a partir de ahora cuando utilicemos valores de λ mayores que 1 emplearemos valores menores que 1,2.

Por último simulamos una red con $\lambda=0,7$. Esto es lo que correspondería a un olvido pasivo de los patrones [6]. En un principio esto sirve para que la red pueda seguir almacenando nuevos patrones sin llegar a saturarse, ya que poco a poco va olvidando los patrones viejos.

En la figura 8 podemos ver también para N=400 y $\rho=0,4$ lo que ocurre para los 5 patrones. Podemos comprobar que ocurre justo lo contrario que con $\lambda>1$ los patrones más viejos son recordados con peor eficiencia mientras que los más nuevos se recuerdan sin problemas.

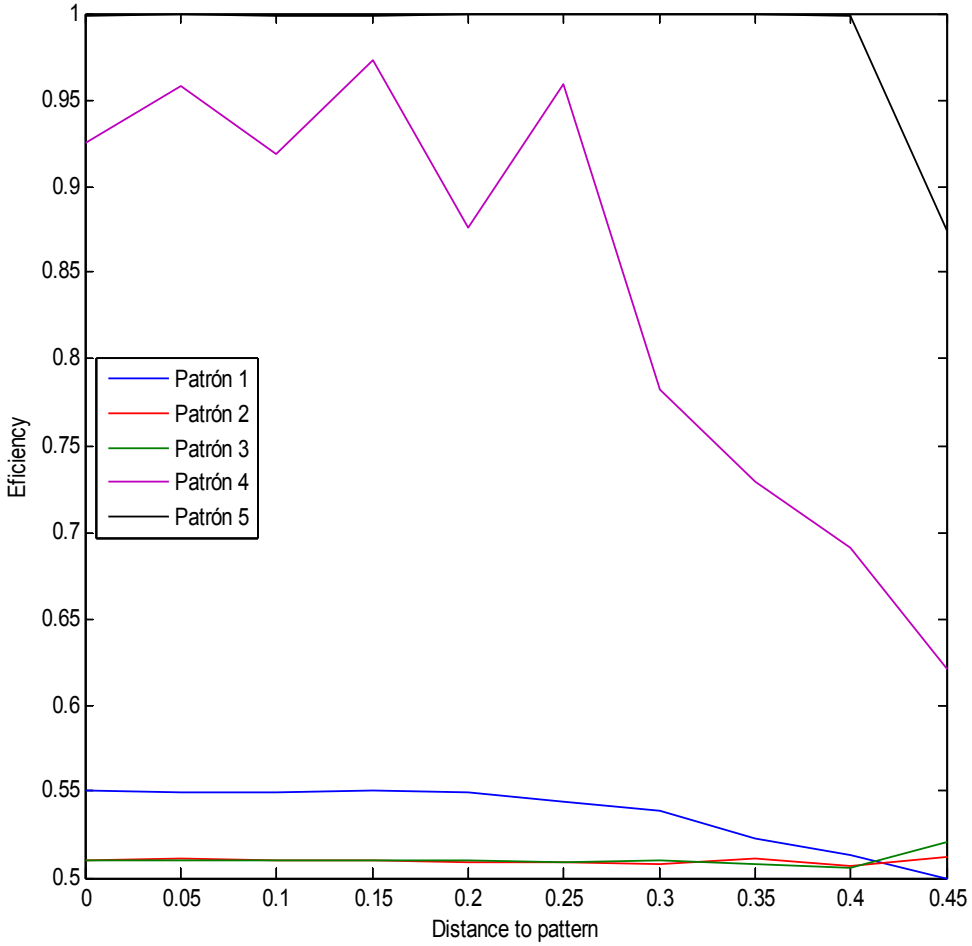


Figura 8. *Para una red con N=400 neuronas, con 5 patrones entrenados con distancia mínima* $\rho=0,4$ *, y con* $\lambda=0,7$*. Eficiencia de la red en función de la distancia entre la señal de entrada y el patrón que se desea recuperar. Promediado de la eficiencia para 5 intentos distintos para cada patrón.*

Más concretamente podemos ver que para este valor de λ el olvido pasivo es tan grande que se recuerda bien el ultimo patrón aprendido, en negro, el penúltimo, en lila, ya tiene problemas, y el resto de patrones los ha olvidado por completo la red [6]. Podemos ver que la eficiencia de los otros patrones es la que correspondería a que la señal convergiese a un patrón distanciado como mínimo $\rho=0,4$ del patrón que queremos recordar, esto se corresponde con que la red converja a un patrón distinto cada vez.

Como en el caso anterior la desviación de l respecto al valor unidad es demasiado grande, porque la red sólo recuerda dos de los patrones, el resto se han olvidado. A partir de estas simulaciones y las contenidas en el apéndice A podemos concluir que el modelo está funcionando de forma correcta y que el algoritmo para entrenar y recuperar patrones sirve para la tarea.

Además comprobamos que el ruido que acepta la red depende de la distancia mínima entre patrones. Esto es importante porque con una elección adecuada de patrones podemos hacer que la red sea más o menos resistente al ruido.

5.3 Segunda simulación: capacidad de la red en función de λ.

En el apartado anterior hemos visto que los valores de λ utilizados varían demasiado respecto al valor unidad y ello implica o bien un olvido muy rápido o un refuerzo demasiado grande, por eso en esta simulación vamos a utilizar valores más próximos entre si. Para tres valores $\lambda = 0{,}99\,,\,1\,,\,1{,}01$, para tres tamaños de red, N=100, 400, 900 y para una distancia entre patrones $\rho = 0{,}4$, realizaremos la siguiente simulación:

- Generaremos 50 patrones separados entre sí al menos ρ e iremos entrenando la red con ellos. Cada vez que enseñemos un nuevo patrón a la red evaluaremos su eficiencia al recuperar los patrones ya almacenados en la misma forma que hicimos en la primera simulación.

Los resultados completos de la simulación aparecen en el apéndice B. Y el código de Matlab en el apéndice E. Aquí mostramos los resultados más interesantes:

Nos centramos de nuevo en la red con N=400. Para $\lambda = 1$ obtenemos los resultados de la figura 9.

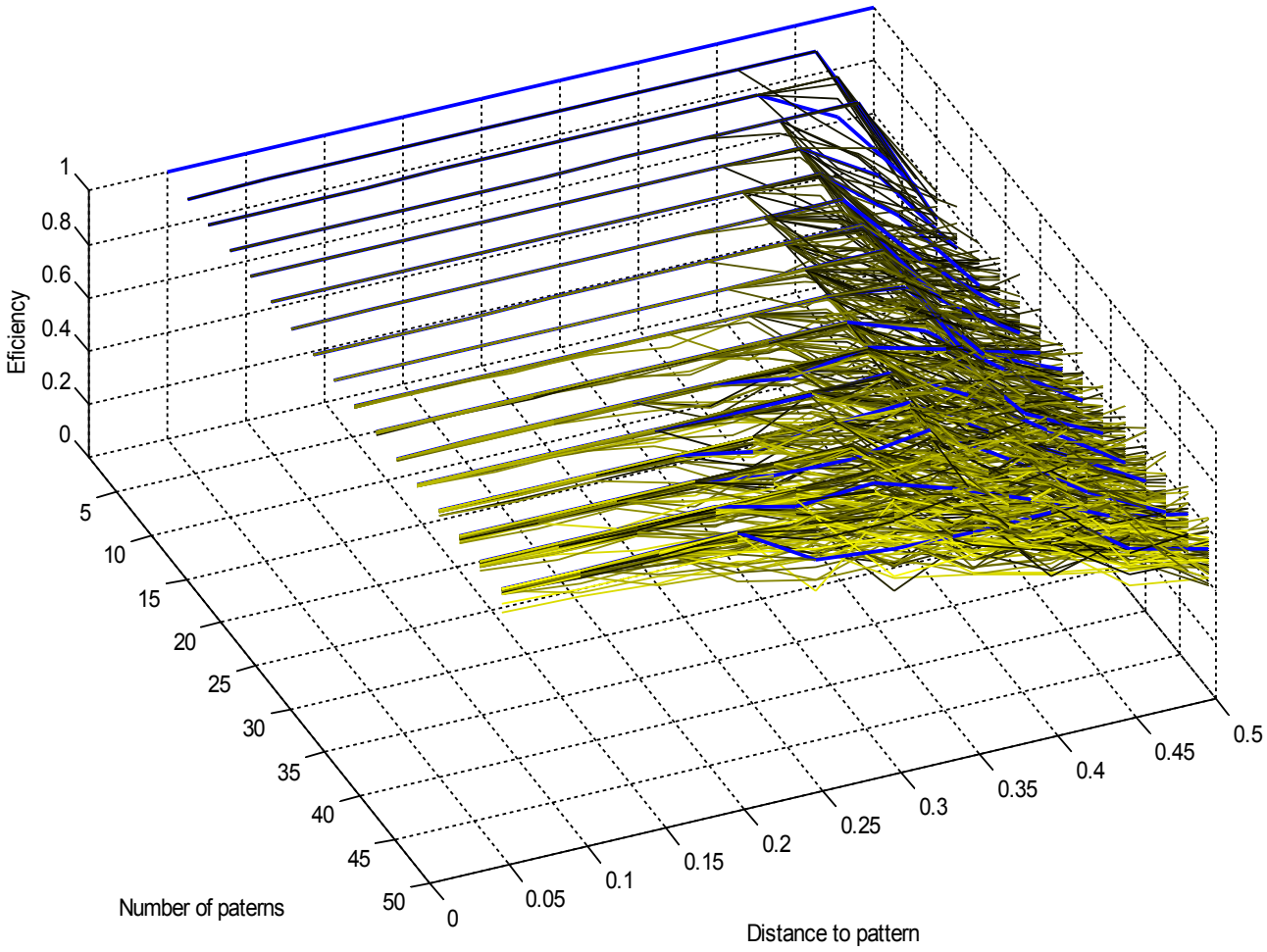


Figura 9. *Red con N=400 neuronas, vamos entrenando patrones con distancia mínima* $\rho = 0{,}4$ *, y con λ=1. Eficiencia de la red en función de la distancia entre la señal de entrada y el patrón que se desea recuperar. La red muestra que a medida que se añaden patrones su eficiencia empeora y la resistencia al ruido es menor.*

En ella se marca en azul el primer patrón entrenado. Podemos ver que a medida que se enseñan más patrones a la red el ruido que soporta la entrada para poder identificar el patrón se reduce. Se observa que la perdida de eficiencia es la misma para cada patrón. Vemos entonces que sin olvido pasivo a medida que enseñamos patrones a la red su eficiencia va reduciéndose hasta que es incapaz de recuperar ningún patrón.

Para $\lambda = 1{,}01$ obtenemos los resultados de la figura 10. Aquí podemos ver lo que ocurre cuando la matriz de pesos no está acotada. Al multiplicar los pesos por $\lambda > 1$ en cada entrenamiento, enseguida llegamos a la saturación. Esto significa que a partir de cierto número de entrenamientos la red es incapaz de recordar ningún patrón y su comportamiento se vuelve caótico [39], por eso la eficiencia de todos los patrones sufre una caída repentina. La red está devolviendo patrones aleatorios.

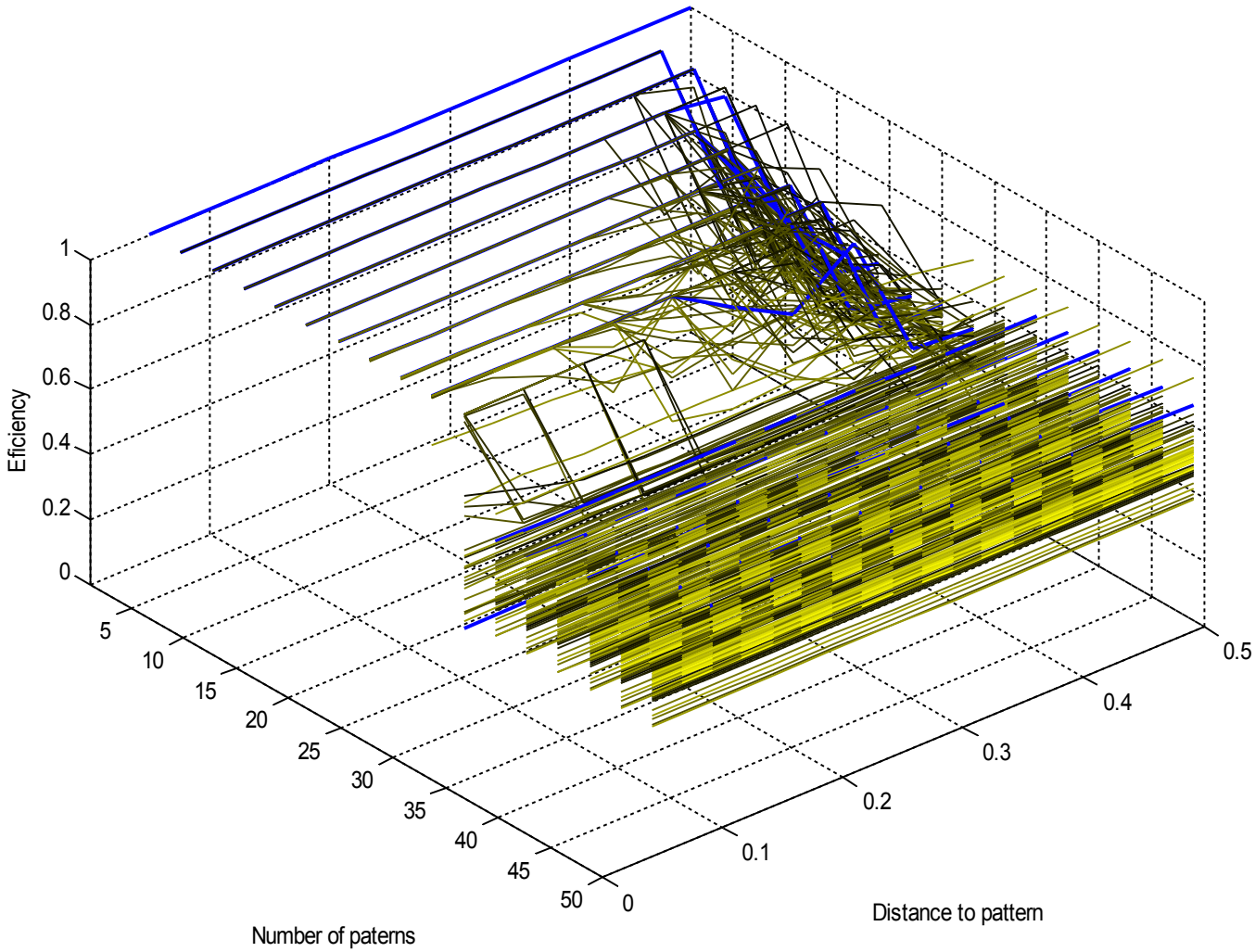


Figura 10. *Para una red con N=400 neuronas, vamos entrenando patrones con distancia mínima* $\rho = 0{,}4$ *, y con* $\lambda = 1{,}01$*. Eficiencia de la red en función de la distancia entre la señal de entrada y el patrón que se desea recuperar. Promediado de la eficiencia para 5 intentos distintos para cada patrón. La red muestra que a medida que se añaden patrones su eficiencia empeora y llega a saturarse y caer en comportamiento caótico.*

A partir de esta simulación podemos concluir que el valor de λ cuando λ>1 sigue siendo demasiado grande. Esto nos lleva a plantear que quizás más que una sola constante para todos los pesos, sería aconsejable quizás una constante que dependa del valor del peso para reforzar los valores pequeños y dejar los grandes inalterados [32]. Una alternativa consiste en introducir un valor de corte por encima del cual los pesos no aumenten [15][5], aunque esto también lleva a comportamiento caótico. Lo mejor es ajustar λ de forma que la matriz de pesos siempre permanezca acotada dentro de cierto rango.

En la siguiente simulación volvemos a tomar los mismos valores de N y de ρ, y para un valor de $\lambda = 0{,}99$ obtenemos los resultados de la figura 11. Vemos que a primera vista el resultado es es muy parecida a la figura 9 para $\lambda = 1$.

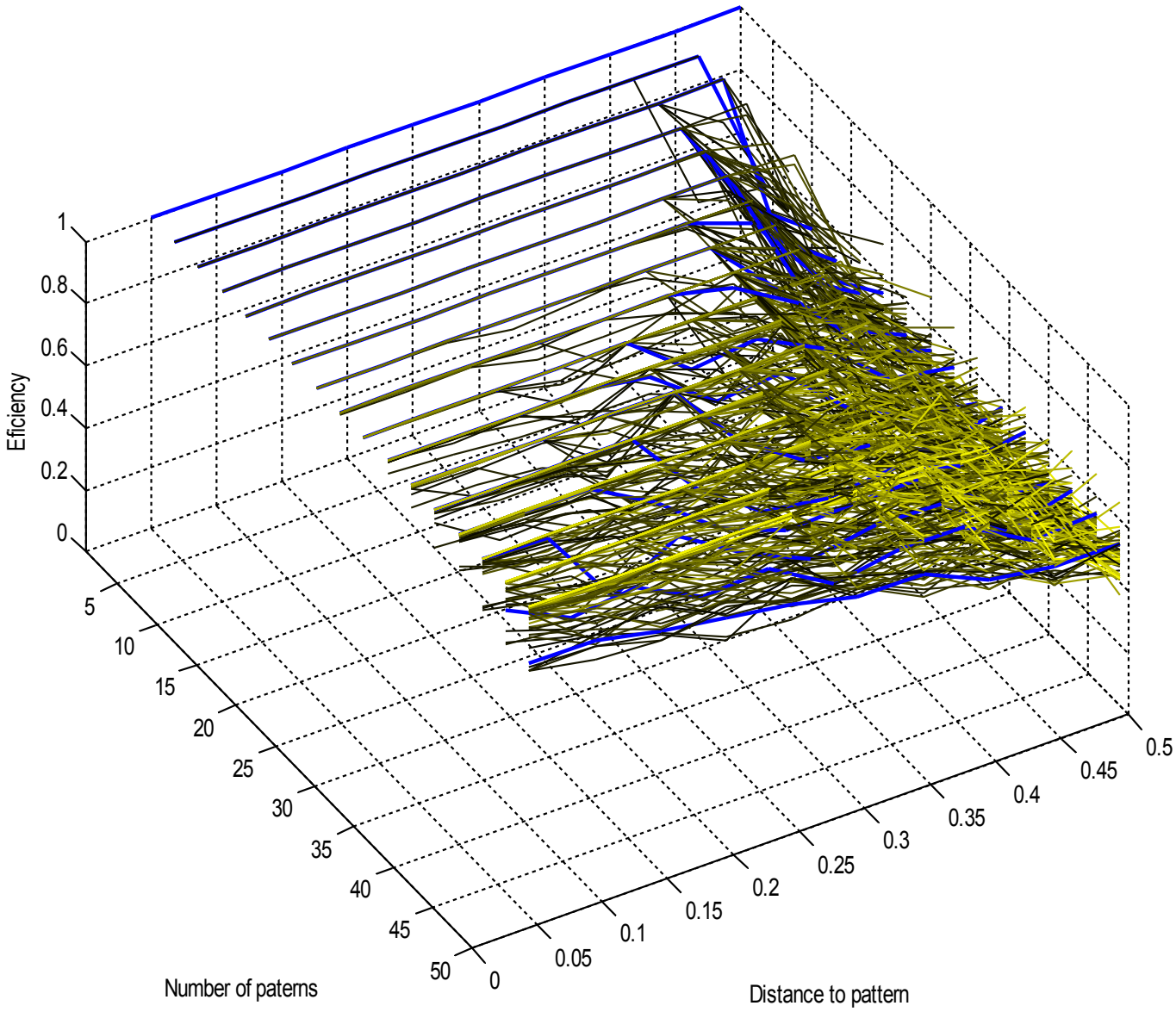


Figura 11. *Para una red con N=400 neuronas, vamos entrenando patrones con distancia mínima* $\rho = 0{,}4$ *, y con* $\lambda = 0{,}99$*. Eficiencia de la red en función de la distancia entre la señal de entrada y el patrón que se desea recuperar. Promediado de la eficiencia para 5 intentos distintos para cada patrón. Aparentemente la eficiencia se reduce a medida que se enseñan patrones.*

Aunque la gráfica parece la misma que con λ=1, hay que recordar que el efecto esperado del olvido pasivo es el de ir olvidando los viejos patrones en función de los nuevos [6]. Por eso volvemos a representar dicha gráfica 11 pero sólo mostrando los 3 últimos patrones enseñados en cada paso de la simulación. Podemos ver el resultado en la figura 12, y se aprecia algo muy distinto. Podemos ver que para dichos patrones el ruido aceptado en la entrada de la red parece no disminuir tan rápido con el número de patrones enseñados. De aquí podemos extraer que la red está olvidando los patrones viejos y dejando "espacio" para los nuevos [15].

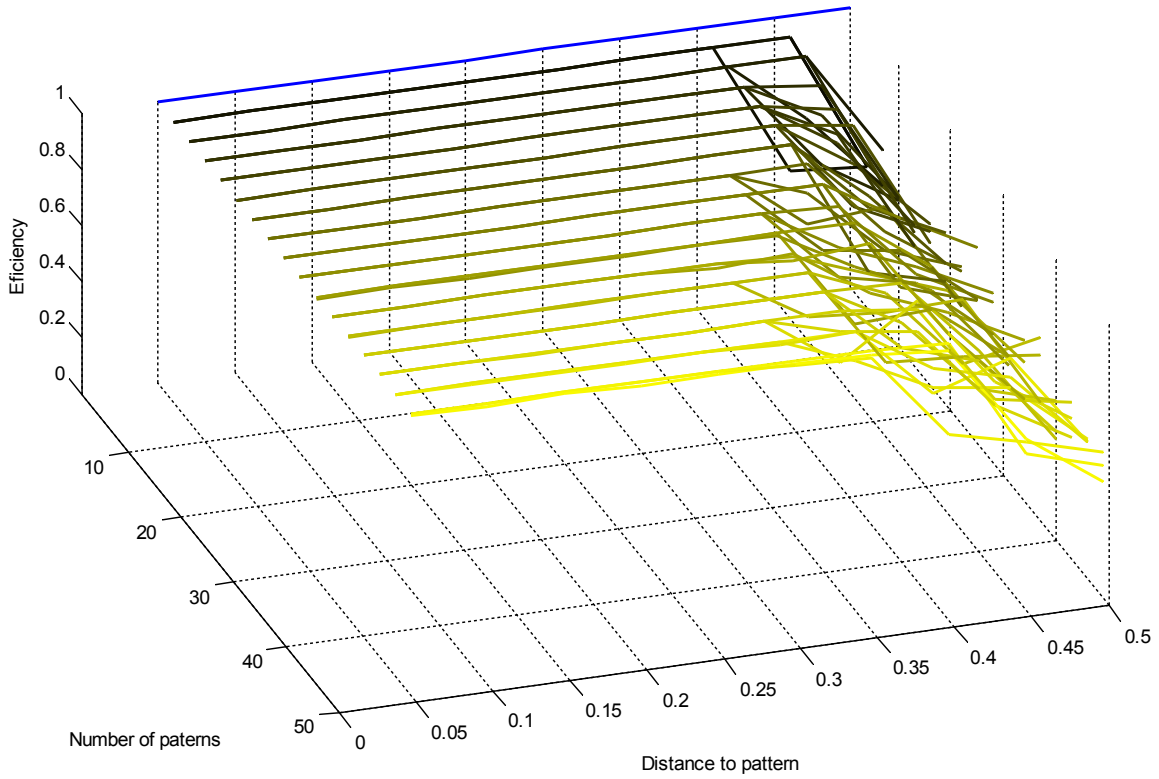


Figura 12. *Para una red con N=400 neuronas, vamos entrenando patrones con distancia mínima* $\rho=0{,}4$ *, y con λ=0,99. Eficiencia de la red en función de la distancia entre la señal de entrada y el patrón que se desea recuperar. Promediado de la eficiencia para 5 intentos distintos para cada patrón. Se han dejado sólo los últimos 3 patrones enseñados en cada iteración para comprobar que la red es capaz de recordarlos de forma adecuada.*

Utilizando estas simulaciones y las contenidas en el apéndice vamos a calcular la capacidad de la red para este modelo y esta forma de entrenar patrones [25].

La capacidad de una red se mide como el número de patrones que es capaz de recordar la red dividido entre el tamaño de la red.

$$\alpha=\frac{M}{N} \tag{54}$$

El valor máximo de la capacidad se conoce como capacidad crítica de la red α_C .

Utilizando el mismo convenio que en [15], se dice que un patrón es recordado por la red cuando se recupera con una eficiencia de 1 a partir de una señal que diste menos de 0,02 del patrón que se desea recordar.

Aplicando este criterio obtenemos la siguiente gráfica en la figura 13. Donde hemos añadido además los resultados para una red con N=1600 neuronas. El código de Matlab está en el apéndice F.

En la figura 13 vemos los resultados obtenidos por las 4 redes. Para cada red tenemos tres casos, λ=0,99 λ=1 y λ=1,01.

Observando los casos con λ=1 vemos que la capacidad de la red aumenta hasta cierto valor, próximo a λ=0,14 y luego la red es incapaz de recuperar ningún patrón. En el caso de la red con N=1600 neuronas no llegamos a verlo porque esa caída se presentaría con más de 225 patrones entrenados, y el proceso para hallar la gráfica conlleva tiempo exponencial de ejecución. Estos resultados son consistentes con [25][33] donde se ha reportado una capacidad crítica de α=0,14 para un modelo de red similar y con patrones que distan ρ=0.5 entre sí.

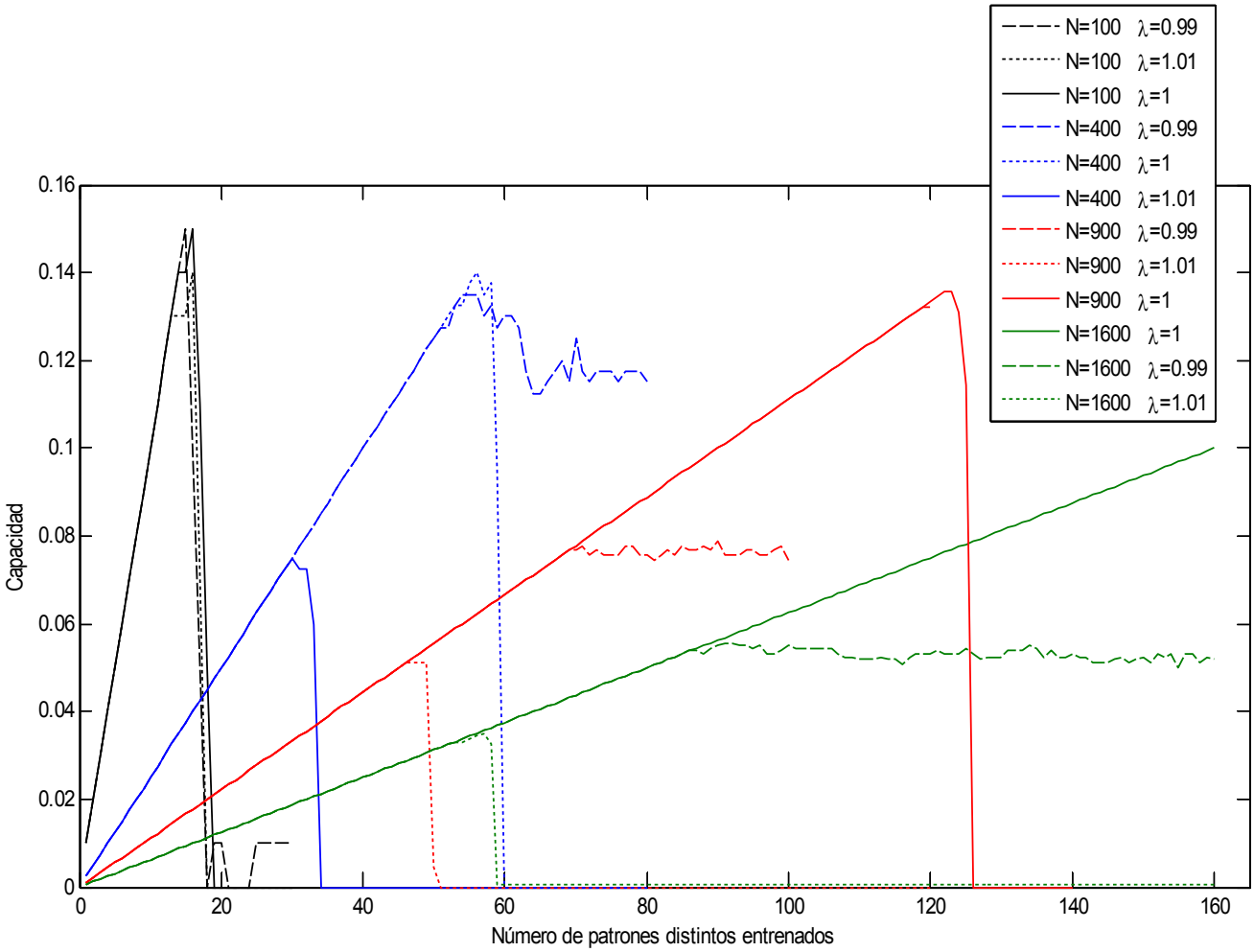


Figura 13. *Capacidad de 4 redes en función del número de patrones entrenados utilizando 3 valores distintos para el olvido pasivo.*

Si nos fijamos en los casos con λ=1.01 vemos que rápidamente se alcanza un estado a partir del cual la red deja de responder. En el caso con N=100 ese estado está muy próximo a la capacidad crítica, pero en el resto de casos se observa que ocurre cada vez a una capacidad menor de la red. Esto es debido a que al aumentar el tamaño de la red también aumenta el número de señales que recibe una neurona dada, ya que está conectada a todas las otras neuronas. Una buena estrategia para hacer la red más resistente podría consistir en limitar el número de conexiones que tiene cada neurona [34], de forma que al aumentar el tamaño de la red no aumente el número de conexiones y por tanto la red siga siendo igual de resistente [35].

Algo similar podemos ver en el caso con λ=0,99. En el caso con N=100 la red se satura antes de que podamos ver algún efecto de este valor distinto de λ, pero con la red con N=400 vemos que antes de que la red se sature se alcanza un estado a partir del cual aunque entrenemos más patrones la capacidad no aumenta, esto significa que a partir de ese punto cada nuevo patrón sustituye a uno viejo ya almacenado. Lo mismo vemos con N=900 y con N=1600, sólo que para estas redes la estabilización ocurre a menor capacidad. De nuevo esto lo podemos relacionar con el hecho de que cada neurona recibe más señales a medida que aumentamos el tamaño de la red [36].

A partir de estos resultados comprobamos que el olvido pasivo está funcionando correctamente como se esperaba que funcionase y que realmente previene a la red de llegar a saturarse. Aunque todavía queda por solucionar el hecho de que a medida que aumenta el tamaño de la red parece que para un mismo valor de λ la capacidad a la que la red se estabiliza parece disminuir.

5.4 Tercera simulación: resistencia de la red ante mecanismos de degradación.

Esta es la última simulación que vamos a realizar. Consiste en simular un proceso de degradación similar a las enfermedades que dañan al cerebro. Concretamente una de las causas del Alzheimer, la muerte de sinapsis entre neuronas.

El estudio de enfermedades neuronales es un área que está en pleno desarrollo actualmente, y el uso de simulaciones mediante redes neuronales está ayudando a comprender aspectos de las enfermedades que hasta ahora eran de difícil explicación [37]. Las simulaciones que se están llevando a cabo se pueden catalogar en dos ramas, las que se denominan biologicamente equivalentes y las que se denominan funcionalmente equivalentes. Las biologicamente equivalentes son simulaciones que intentan simular el funcionamiento de partes del cerebro utilizando para ello modelos que intentan simular todos los detalles. Las funcionalmente equivalentes son simulaciones donde se intenta simular el funcionamiento de partes del cerebro sin simular cada detalle del mismo.

Las simulaciones que nosotros hemos estado haciendo encajarían dentro de las funcionalmente equivalentes. Concretamente, servirían para modelar la memoria a corto y largo plazo [17]. Nosotros vamos a centrarnos en simular la degradación que sufre la memoria cuando el cerebro sufre la enfermedad de Alzheimer [10].

A nivel funcional la enfermedad de Alzheimer consiste en una degradación de las sinapsis neuronales que hace que deje de pasarse información entre neuronas. Dicha degradación a veces va a acompañada de mecanismos de compensación, bien propios del cuerpo humano bien inducidos por medicamentos. Nosotros hemos visto ya el efecto de un posible mecanismo de compensación cuando usábamos valores de λ mayores que 1. Ahora vamos a ver el efecto que tiene la muerte sináptica sobre la eficiencia de la red.

La simulación que realizamos consiste en lo siguiente.

- Para N=400 neuronas tomamos $\lambda=1$ y generaremos 5 patrones con distancia mínima entre ellos de $\rho=0{,}45$. Bajo estas condiciones, estudiaremos la eficiencia de la red en función del número de conexiones que han sido borradas (pesos puestos a cero). Para cada porcentaje de muerte sináptica probaremos con 5 señales con un ruido respecto al patrón que se desea recuperar de $\rho=0{,}1$ (Para este valor tan pequeño vimos en la figura 6 que la eficiencia de la red debería ser próxima a 1). La eficiencia de cada señal se promedia 10 veces.

El resultado de la simulación puede verse en la figura 14 (el código de Matlab está en el apéndice G). Comprobamos que la eficiencia de la red para todos los patrones es prácticamente 1 cuando no existe muerte sináptica (luego dichos patrones están almacenados y se recuperan correctamente cuando la red está bien). Luego a medida que vamos degradando la red la eficiencia va reduciéndose, para caer de forma abrupta cuando la degradación de la red está entre el 60% y el 70%. A partir de ahí la red está devolviendo patrones aleatorios y no es capaz de recordar ninguno de los patrones.

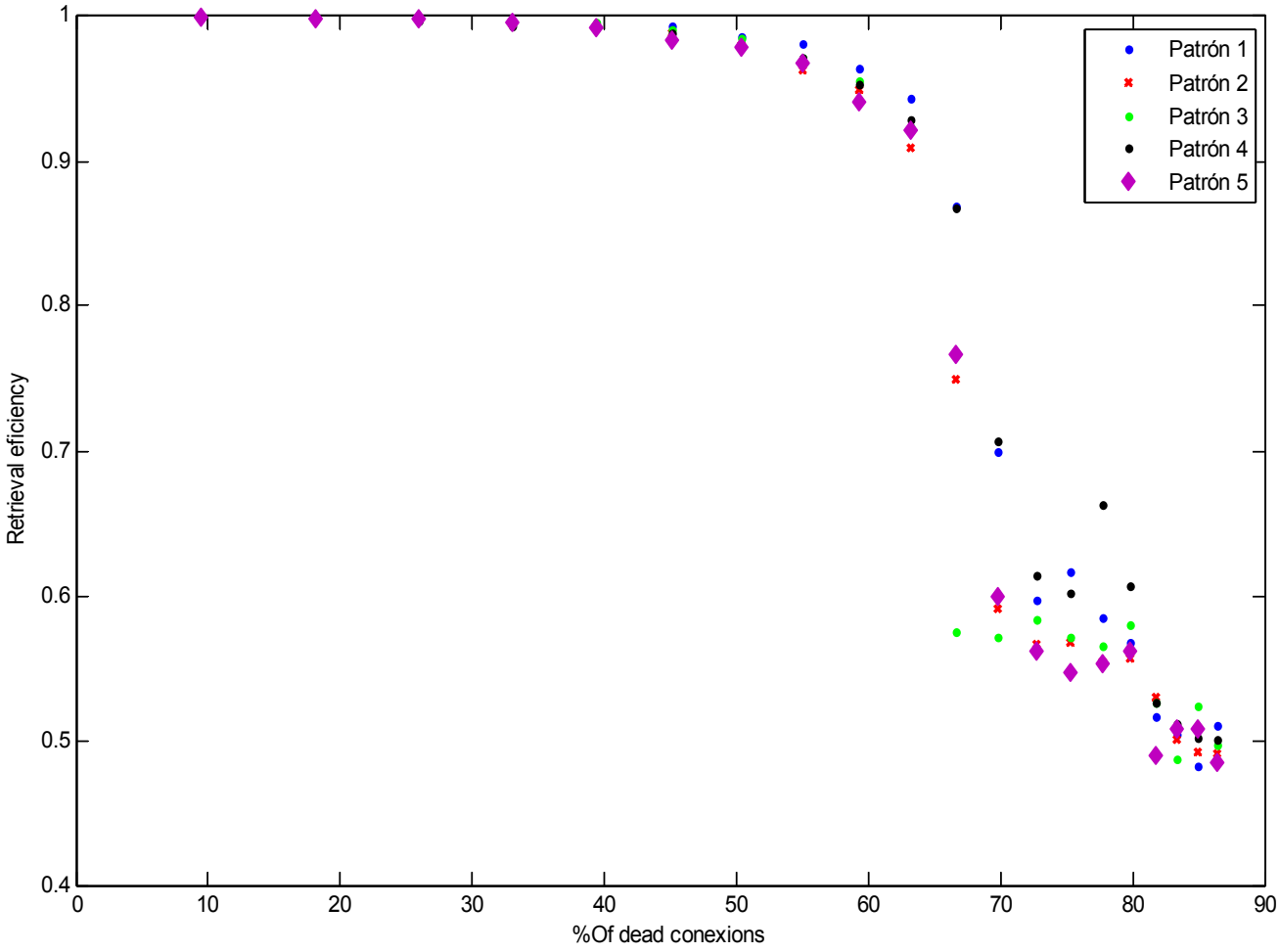


Figura 14. *Degradación de la eficiencia de una red con N=400, λ=1, distancia entre patrones ρ=0,4. Efecto de la muerte sináptica en la eficiencia en la recuperación de patrones.*

Vemos que la red sufre un efecto similar al reportado en los pacientes que sufren Alzheimer, donde tras cierto deterioro neuronal el paciente comienza a olvidar recuerdos.

Este resultado es particularmente importante porque es uno de los resultados obtenidos por un modelo de red similar [17][38]. Es de vital importancia que la red mantiene su eficiencia prácticamente hasta casi el final, cuando deja de recordar un patrón su eficiencia cae hasta devolver patrones aleatorios. Ciertos estudios [17] [10] han analizado mecanismos de compensación que mitigan la caída de rendimiento al combinar este mecanismo de degradación con una multiplicación de los pesos como las que hemos visto nosotros en apartados anteriores. Dichos estudios no consiguen aumentar la resistencia de la red, pero hacen que la perdida de eficiencia no sea tan abrupta.

En otros estudios se ha investigado el número mínimo de conexiones necesarias para que la red siga funcionando, y se ha encontrado que existe un mínimo para que la red siga funcionando, y por debajo de él la red comienza a devolver patrones aleatorios [35].

6 Conclusiones.

Este trabajo fin de máster tenía como objetivo el estudio de un modelo de red tipo Hopfield estocástica. Se pretendía estudiar el funcionamiento del modelo, las principales características del mismo y la elaboración de un algoritmo y un código para poder simular dicho modelo.

Este tipo de redes resulta particularmente interesante porque tienen la propiedad de ser modulares, es decir, su funcionamiento no viene condicionado por el tamaño de la red como puede ocurrir por ejemplo con los perceptrones, y podemos ampliar o reducir su tamaño sin problemas para el resultado final.

Nosotros hemos planteado un modelo de red que se está imponiendo como versión más avanzada del modelo de Hopfield y que está resultando ser bastante útil para el estudio de enfermedades neurológicas así como para tareas de procesado de la información.

El modelo de red se describe en la sección 2 como una red Hopfield con neuronas estocásticas que sufren un entrenamiento progresivo y están sujetas a un olvido pasivo. Como característica de este modelo tenemos que al ser estocástico resulta sencillo que la red cambie de estado, el entrenamiento progresivo permite imitar el funcionamiento del cerebro y da lugar a estrategias de entrenamiento y simulación mucho más variadas, y por ultimo la incorporación de un olvido pasivo da como lugar la posibilidad de poder entrenar la red de forma continua sin llegar a saturar su capacidad.

Además del modelo de red, en la sección 3 hemos descrito y ampliado con detalles un par de herramientas básicas al tratar con redes tipo Hopfield.

Utilizando las herramientas de la sección 3 y el modelo descrito en la sección 2 dedicamos la sección 4 a analizar ciertas condiciones de convergencia y estabilidad. Utilizamos allí unos principios similares a los del campo medio y los del truco de la réplica pero más sencillos dado que sólo nos interesa obtener unos margenes para la temperatura que debemos utilizar en el modelo.

Basándonos en la literatura disponible y con ciertos conocimientos heredados de trabajos previos con redes tipo Hopfield escribimos un algoritmo que se muestra en la sección 5 y que sirve para realizar entrenamientos y simulaciones con el modelo. El algoritmo viene acompañado de varios scripts en Matlab para poder realizar las simulaciones. El código de Matlab no se comenta aquí, pero viene incluido en el apéndice ya que se pretende seguir trabajando con él en futuras investigaciones.

Utilizando el algoritmo y los rangos de valores encontrados en la sección 4, dedicamos la sección 5 a realizar distintos tipos de simulaciones. La primera de las simulaciones nos mostró que el modelo de red funcionaba correctamente y que pese a tratarse de una red estocástica obteníamos una buena relación entre la eficiencia de la red y el ruido en la señal de entrada. También obtuvimos los primeros resultados que sugerían un correcto funcionamiento del olvido pasivo. Cuando el olvido pasivo era superior a uno vimos que se podía entender como si fuese un mecanismo de compensación, pero que rápidamente llevaba a un deterioro de los patrones almacenados.

En la segunda simulación analizamos en detalle el funcionamiento del olvido pasivo sometiendo a distintas redes a procesos de almacenamiento masivo hasta saturarlas.

Realizando un análisis de la capacidad obtuvimos una capacidad crítica para este tipo de red próxima a valores reportados para otras redes similares, confirmando así el correcto funcionamiento del modelo. Además, pudimos observar como el olvido pasivo funcionaba limitando la capacidad máxima que alcanzaba la red y evitando así la saturación de la misma.

En la ultima de las simulaciones probamos algo distinto. Intentamos simular un proceso degenerativo asociado al Alzheimer que consiste en la muerte de conexiones sinapticas. Aquí volvimos a obtener resultados similares a los conseguidos por distintos grupos y que muestran una gran resistencia al deterioro en estos tipos de redes, soportando hasta un 70% de muerte sináptica antes de perder eficiencia.

Además del presente trabajo, esta investigación ha dado como resultado un código de Matlab para seguir experimentando con el modelo de red neuronal. Dicho código está pensado para poder trabajar con varias redes conectadas entre si y poder simular así sistemas más complejos que un sólo módulo de memoria. La realización de dicho código ha llevado la mayor parte de la carga del trabajo ya que requiere la correcta programación de variables para poder realizar simulaciones con redes muy grandes y con simulaciones con muchas iteraciones. Su depuración desde la idea original hasta los resultados mostrados aquí ha llevado casi un año de continuos ensayos y mejoras. Por eso aunque la longitud del trabajo no permite destacar las diversas partes del código y las soluciones adoptadas, tenemos que decir que es uno de los mayores logros del presente trabajo, ya que además nos servirá de punto de partida para posteriores investigaciones.

Uno de los objetivos para el futuro y que estaba en marcha durante este trabajo aunque no se pudo incluir aquí por estar todavía en su fase inicial consiste en utilizar dicho código en colaboración con otro estudiante de este máster para realizar simulaciones sobre mecanismos de degradación y compensación en el cerebro humano. Concretamente se ha estado trabajando con el propósito de poder simular el efecto de la depresión en un modelo simplificado de cerebro, centrándose la investigación en mecanismos de muerte neuronal y de compensación.

7 Agradecimientos.

Quisiera por ultimo agradecer a mi tutora, Elka Korutcheva, por la dedicación que ha mostrado durante la elaboración de este trabajo y por el apoyo que ha resultado esencial. Así mismo quisiera agradecer a un compañero del máster, Ángel Alija, porque trabajando juntos hemos podido ver más detalles que por separado. Y finalmente a la UNED por aportar los medios para llevar a cabo la investigación y por ser la anfitriona del máster.

8 Referencias.

[1] Simon Haykin; “Neural Networks: A comprehensive Foundation”; Prentice Hall (1999)

[2] Steven J. Cooper; “Donald O. Hebb's synapse and learning rule: a history and commentary”; Neuroscience and Biobehavioral Reviews 28, 951-874, (2005).

[3] Luz Y, Shamir M (2012) Balancing Feed-Forward Excitation and Inhibition via Hebbian Inhibitory Synaptic Plasticity. PLoS Comput Biol 8(1): e1002334.

[4] J. M. Cortes, J.J. Torres, J. Marro, P. L. Garrido, H. J. Kappen; “Effects of Fast Presynaptic Noise in Attractor Neural Networks”; Neural Computation, Vol **18**, 3, 614-633, (2006).

[5] Jian K. Liu; “Learning Rule of Homeostatic Synaptic Scaling: Presynaptic Dependent or Not”; Neural Comput. **12**, 3145-3161, (2011).

[6] Benoît Siri, Hugues Berry, Bruno Cessac, Bruno Delord, Mathias Quoy; “A mathematical analysis of the effects of Hebbian learning rules on the dynamics and structure of discrete-time random recurrent neural networks”; Neural Computation **20**, 2937-2966, (2008).

[7] Bi Guo-qiang, Poo Mu-ming; “Synaptic Modification By Correlated Activity: Hebb's Postulate Revisited”; Annu. Rev. Neurosci. **24**, 139-166, (2001).

[8] German Barrionuevo, Thomas H. Brown; “Associative long-term potentiation in hippocampal slices” Proc. Natt. Acad. Sci. USA, Vol. 80, 7347-7351, (1983)

[9] He. Guoguang, Chen Luonan, Aihara Kazuyuki; “Associative memory with a controlled chaotic neural network”; Neurocomputing **71**, 2794-2805, (2008).

[10] Eytan Ruppin, David Horn, Nir Levy, James A. Reggia; “Computational Studies of Synaptic Alterations in Alzheimer's Disease”; Neural Modeling of Brain and cognitive Disorders, World Scientific, (1996).

[11] Stephen Mrsland; “Machine Learning: An Algorithmic Perspective”; Chapman & Hall /CRC (2009)

[12] J.J. Hopfield; “Neural networks and physical systems with emergent collective computational abilities”; Proc. Natl. Acad. Sci. USA **79**, 2554-2558 (1982).

[13] E. Rupin, M. Usher; “An Attractor Neural network Model of Semantic Fact Retrieval”; IJCNN International Joint Conference on Neural Networks, Vol 1-3, c683-c688, (1990).

[14] Hoppensteadt, F., and E. Izhikevich; “Weakly Connected Neural Networks” Springer Verlag, (1997).

[15] Giorgio Parisi; “A memory wich forgets”; J. Phys. A: Math. Gen. 19 L617 (1986).

[16] J. J. Torres, L. Pantic, H. J. Kappen; “On the storage capacity of attractor neural networks with depressing synapses”; Physical Review E, (2002)

[17] Eytan Ruppin, James A. Reggia; “A Neural Model of memory Impairment in Diffuse Cerebral Atrophy”; BJPsych 166, 19-28, (1995).

[18] M. Herrmann, E. Ruppin, M. Usher; “A Neural model of the dynamic activation of memory”; Biological Cybernetics, Vol 68, 5, 455-463, (1993)

[19] Lovorka Pantic, Joaquín J. Torres, Hilbert J. Kappen, Stan C. A. M. Gielen; “Associative memory with dynamic synapses”; Neural Comput. **12**, 2903-2923, (2002).

[20] Jiahai Wang, Zhanghui Kuang, Yalan Zhou, Rong-Long Wang; “Improved Stochastic Competitive Hopfield Network for Polygonal Approximation”; Expert Systems with Applications, 38, 4109-4125, (2011).

[21] E. Domany, J. L. van Hemmen, K. Schulten; “Models of Neural Networks II”; Springer (1994)

[22] S. Canals, M. Beyerlein, H. Merkle, N. K. Logothetis, “Functional MRI Evidence for LTP-Induced Neural Network Reorganization”, Current Biology 19, 398-403, (2009)

[23] Tomoyuki Kimoto, Tatsuya Uezsu, Masato Okada, “Multiple Stability of a Sparsely Encoded Attractor neural Network model for the Inferior Temporal Cortex”;

[24] Sawako Tanimoto, masato Okada, Tomoyuki Kimoto, Tatsuya Uezu; “Distinction of Coexistent Attractors in an Attractor Neural Network Model using a Relaxation Process of Fluctuations in Firing Rates -Analysis with Statistical mechanics-”; Journal of the Physical Society of Japan, Vol. **75**, No. 10, (2006).

[25] M. V. Tsodyks, M.V. Feigel'Man; “The Enhanced Storage Capacity in Neural Networks with Low Activity Level”; Europhys. Lett. **6**, 101, (1988).

[26] Yukihiro Tsuboshita, Masato Okada; “Statistical-Mechanical Analysis of Attractor Dynamics in a Hysteric Neuron Network”;
journal of the Physical Society of Japan, Vol **79**, No. 2, (2010).

[27] D. O. Hebb; “The organization of Behavior”; John Wiley & Sons, (1949).

[28] Jun-ichi Inoue; “Retrieval Phase Diagrams of Non-monotonic Hopfield Networks”; arXiv:cond-mat/9604065v2 [cond-mat.dis-nn] 25 Aug 1997.

[29] Tatsuya Uezu, Aya Hirano, Masato Okada; “Retrieval Properties of Hopfield and Correlated Attractors in an Associative memory Model”; Journal of the Physical Society of Japan, Vol. 73, 2004.

[30] Yang Li, Ping Zhu, Xiaoping Xie, Guoguang He, Kazuyuki Aihara; “learning-induced pattern classification in chaotic neural network”; Physics letters A 376, 412-417, (2012).

[31] E. Domany, J. L. van Hemmen, K. Schulten; “Models of Neural Networks III”; Springer (1996)

[32] D. Horn, E. Ruppin; “Compensatory mechanishms in an attractor neural network model of Schizophrenia”; Neural Comput. 182-205, (1995).

[33] Giorgio Parisi; “Asymmetric neural networks and the process of learning”; J. Phys. A: Math. Gen. 19 (1986).

[34] Gal Chechik, Issac Meilijson, Eytan Ruppin; “Effective neuronal learning with ineffective Hebbian Learning Rules”; neural Comput. 817-840, (2001).

[35] Anafi RC, Bates JHT (2010) Balancing Robustness against the Dangers of Multiple Attractors in a Hopfield-Type Model of Biological Attractors. PloS ONE 5(12): e14413.

[36] Robert J. McEliece, Edward C. Posner, Eugene R. Rodemich, Santosh S. Venkatesh; “The Capacity of the Hopfield Associative Memory” IEEE Transactions on Information Theory, Vol. IT-33, No. 4, (1987).

[37] D. Horn, E. Ruppin, M. Usher, M. Herrmann; “Neural network modeling of memory deterioration in Alzheimer's disease”; Neural computation, Vol. 5, 736-749, (1993).

[38] J. R. Carrie; “Evaluation of a Neural Network Model of Amnesia in Diffuse Cerebral Atrophy”; BJPsych 163, 217-222, (1993).

[39] B. Müller, J. Reinhardt, M. T. Strickland; "Neural Networks: An Introduction"; 2ºed; (Springer); 1995.

[40] S.Coombes, J. G. Taylor; “The Storage and Stabilization of Patterns in a Hopfield Net”; Neural Network World, Vol 5, **2**, 133-150, (1995).

9 Apéndice.

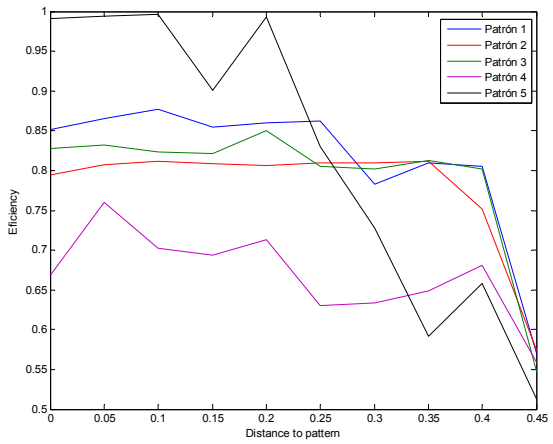
Apéndice A.

Generamos 5 patrones con una distancia entre ellos de por lo menos ρ . Evaluamos cual es la eficiencia de la red para recuperar un patrón en función de la distancia entre el input y el patrón que se desea recuperar. Se realizan 5 intentos y se halla la eficiencia promedio en la recuperación. La eficiencia viene dada por 1 menos la distancia entre el patrón que se desea recuperar y el que se recupera.

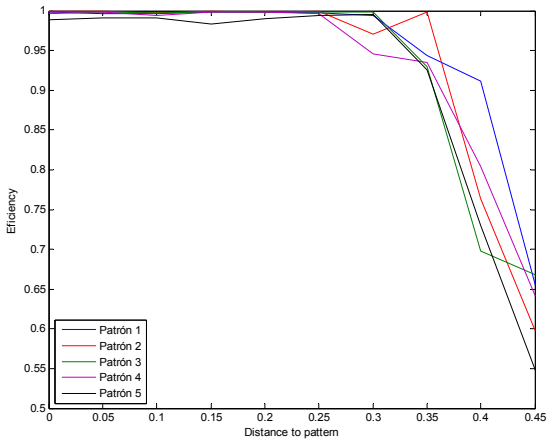
$\lambda = 1$

N=100.

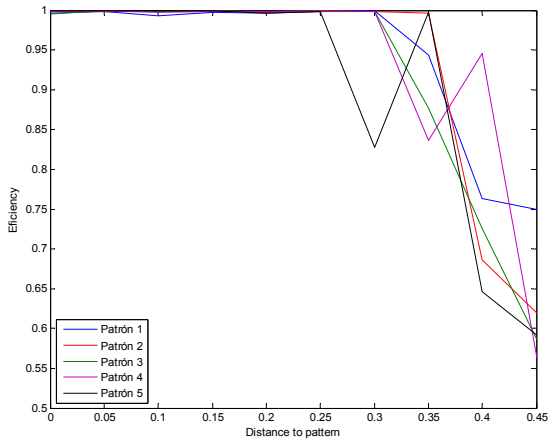
$\rho = 0{,}4$



$\rho = 0{,}45$

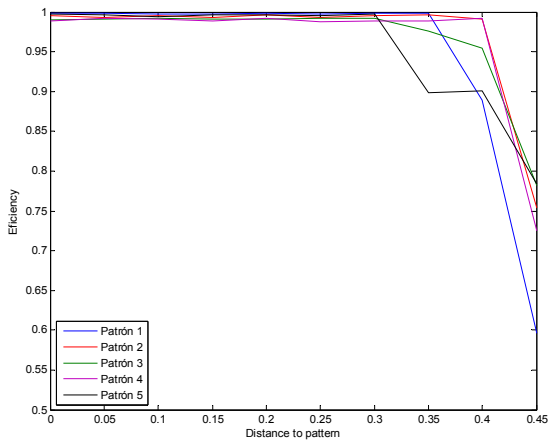


$\rho = 0{,}5$

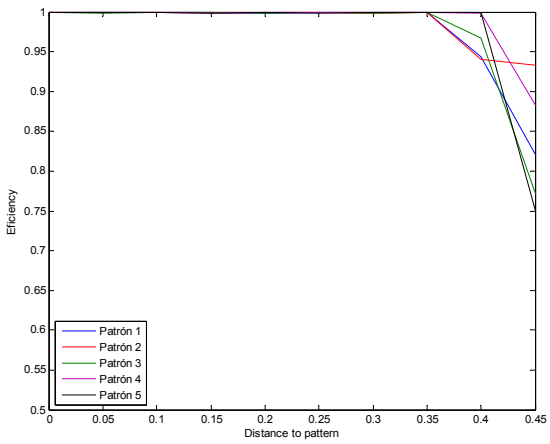


N=400.

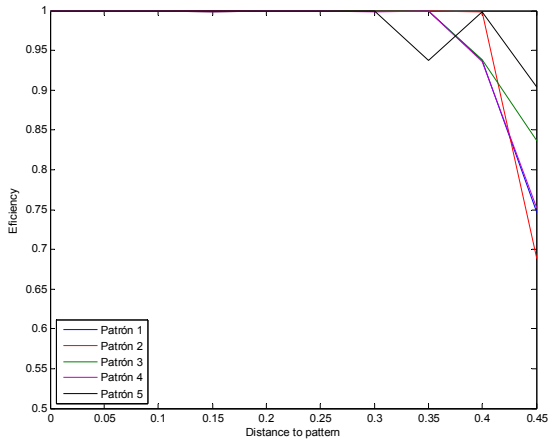
ρ=0,4



ρ=0,45

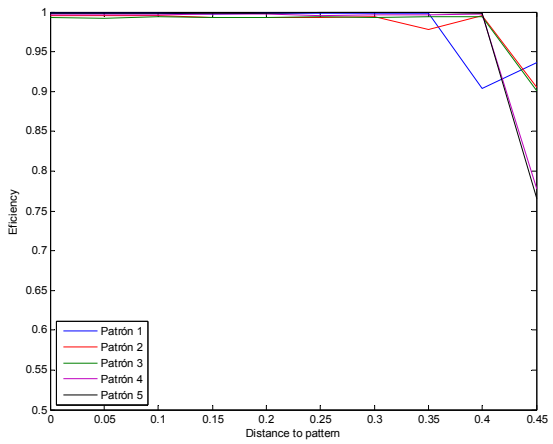


ρ=0,5

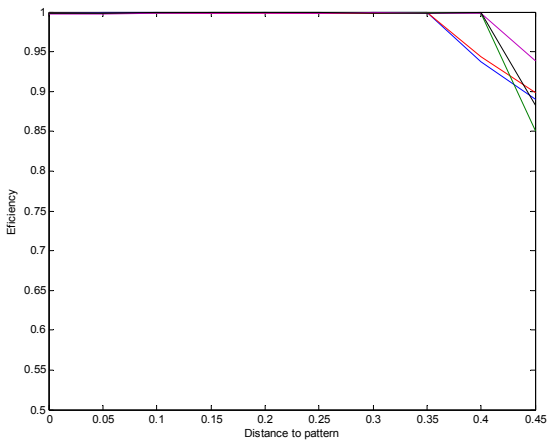


N=900.

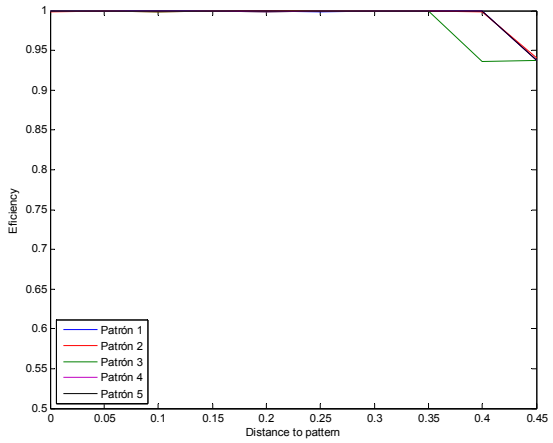
$\rho = 0{,}4$



$\rho = 0{,}45$



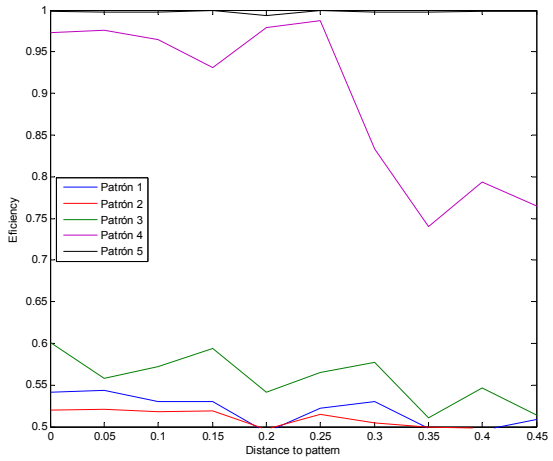
$\rho = 0{,}5$



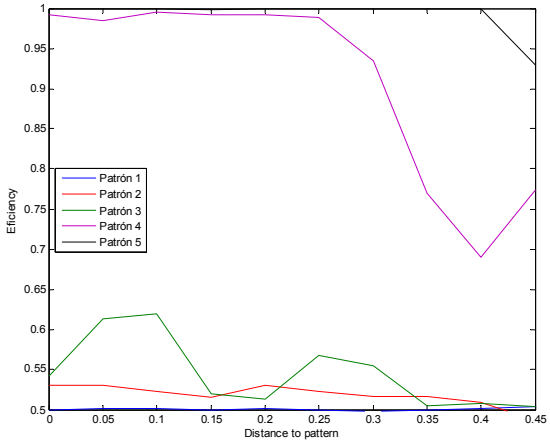
$\lambda = 0{,}7$

N=100.

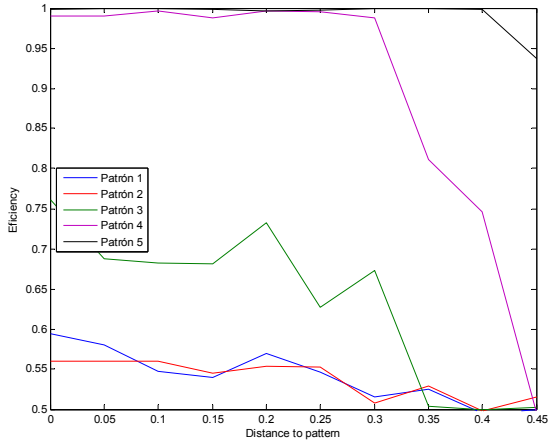
$\rho = 0{,}4$



$\rho = 0{,}45$

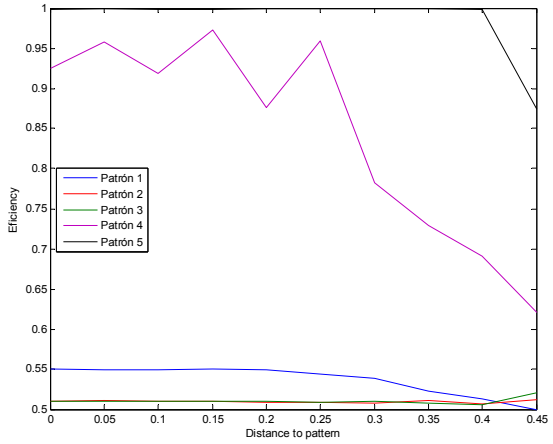


$\rho = 0{,}5$

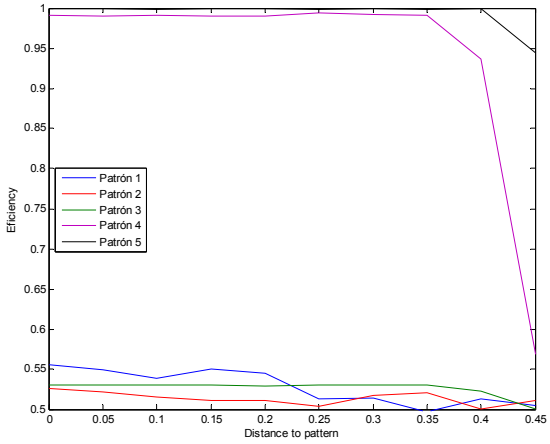


N=400.

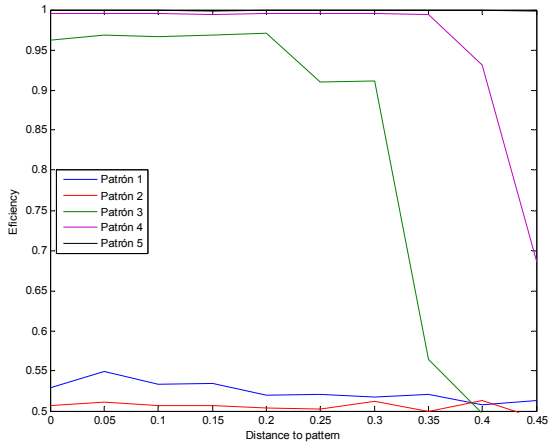
$\rho = 0{,}4$



$\rho = 0{,}45$

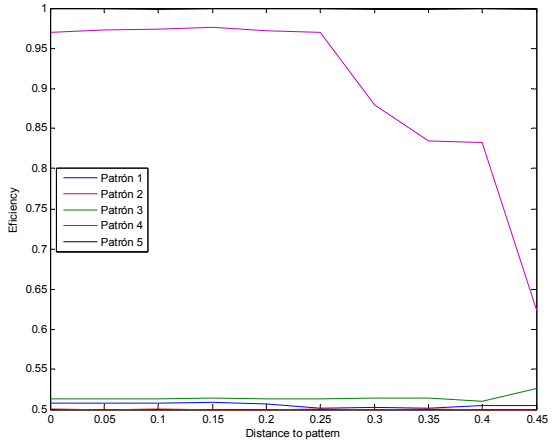


$\rho = 0{,}5$

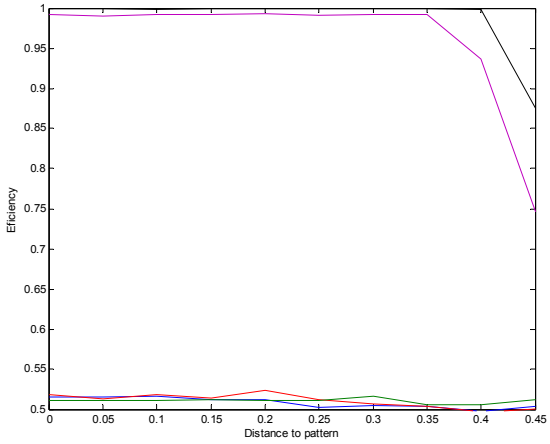


N=900.

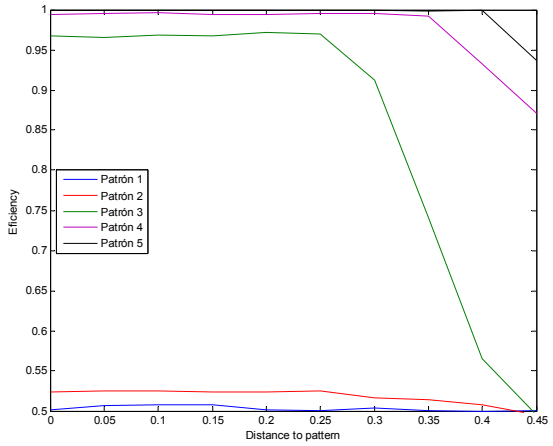
ρ=0,4



ρ=0,45



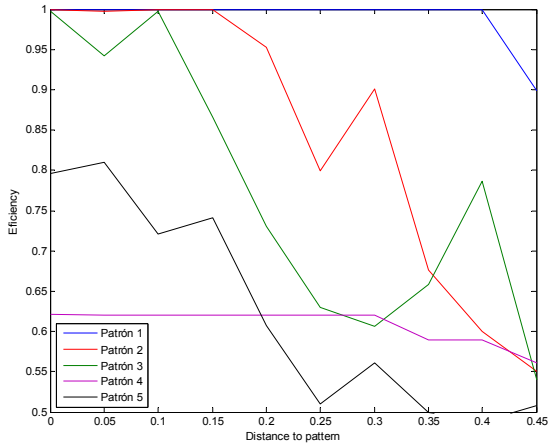
ρ=0,5



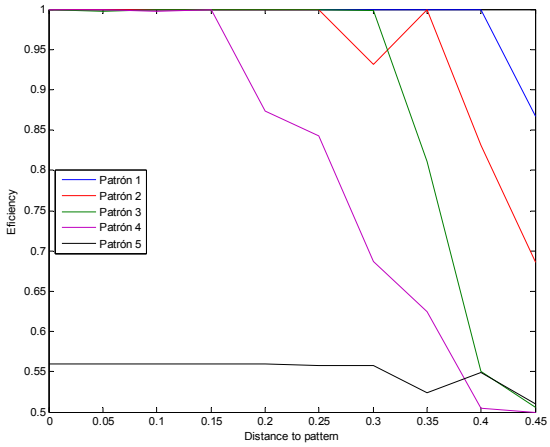
$\lambda = 1{,}2$

N=100.

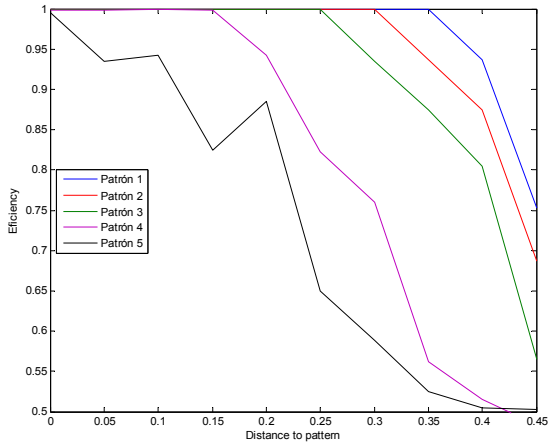
$\rho = 0{,}4$



$\rho = 0{,}45$

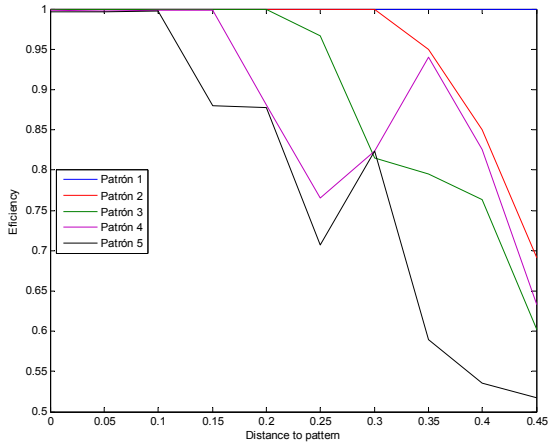


$\rho = 0{,}5$

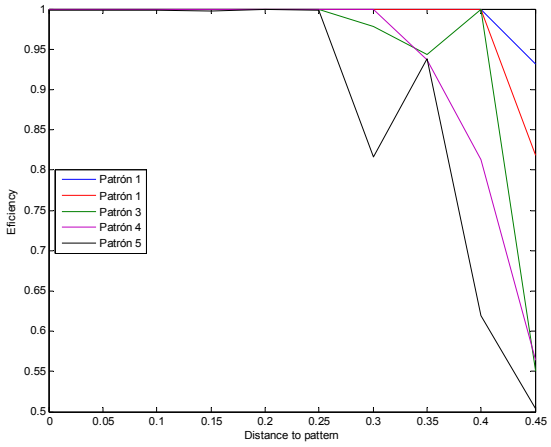


N=400.

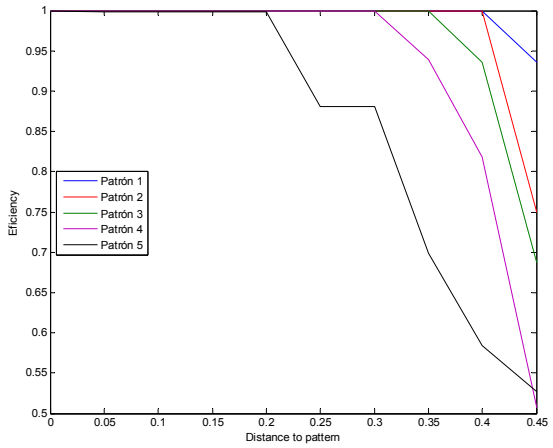
$\rho = 0{,}4$



$\rho = 0{,}45$

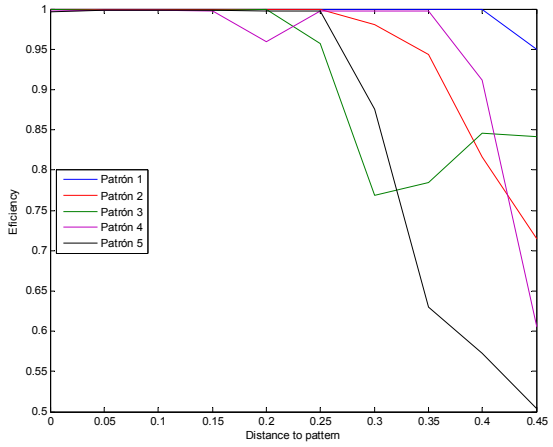


$\rho = 0{,}5$

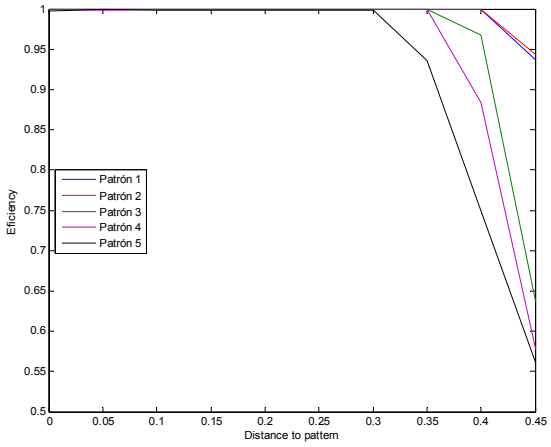


N=900.

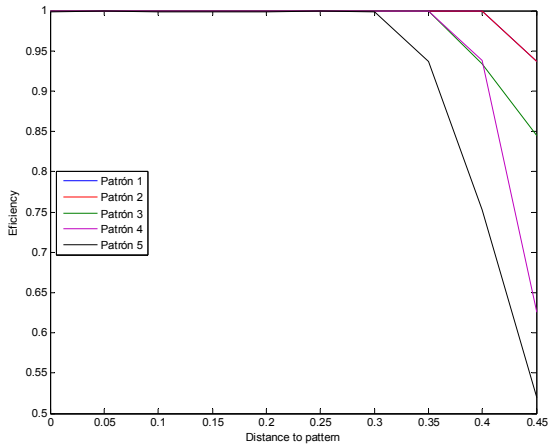
$\rho = 0{,}4$



$\rho = 0{,}45$



$\rho = 0{,}5$



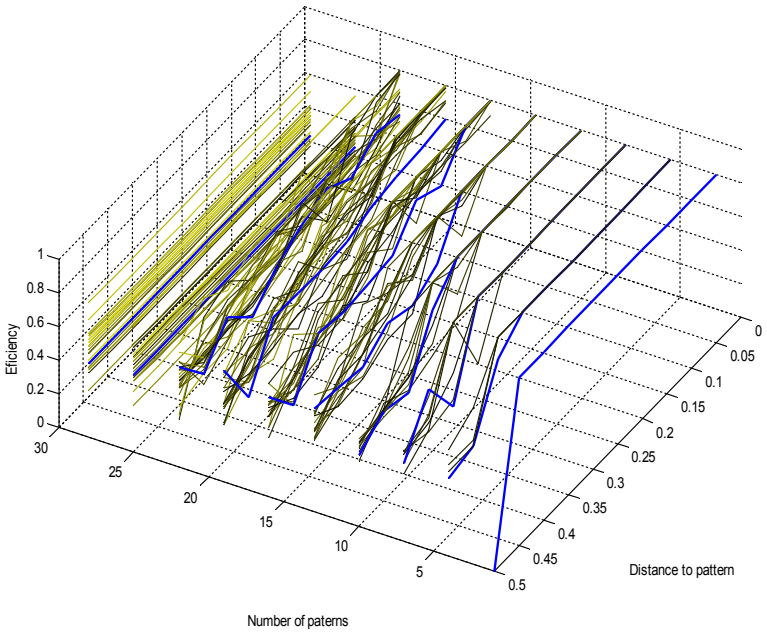
Apéndice B.

Generamos patrones con una distancia entre ellos de por lo menos ρ . Cada vez que enseñamos un nuevo patrón evaluamos cual es la eficiencia de la red para recuperar todos los patrones almacenados en función de la distancia entre el input y el patrón que se desea recuperar. Se realizan 5 intentos y se halla la eficiencia promedio en la recuperación. La eficiencia viene dada por 1 menos la distancia entre el patrón que se desea recuperar y el que se recupera. En las gráficas no aparecen todas las curvas, están sólo cada vez que se enseñan 3 patrones.

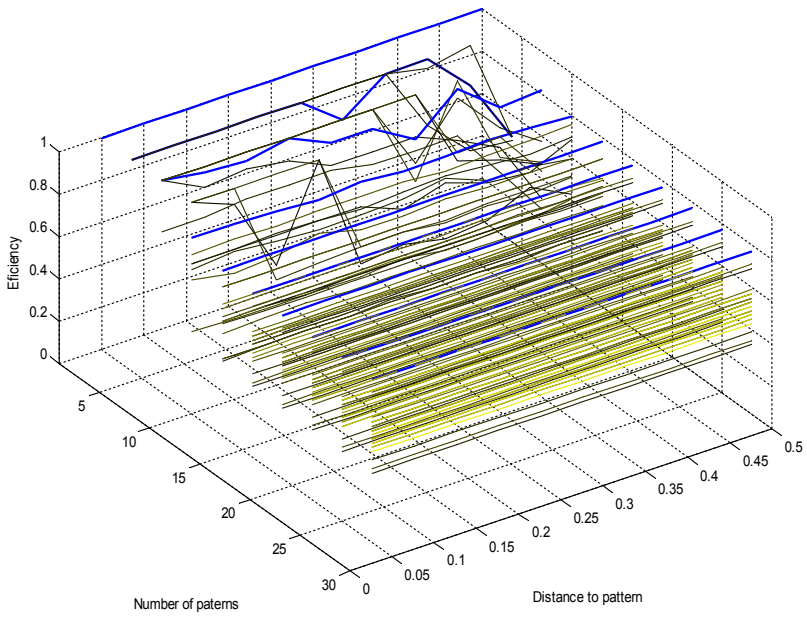
$\rho = 0{,}4$

N=100.

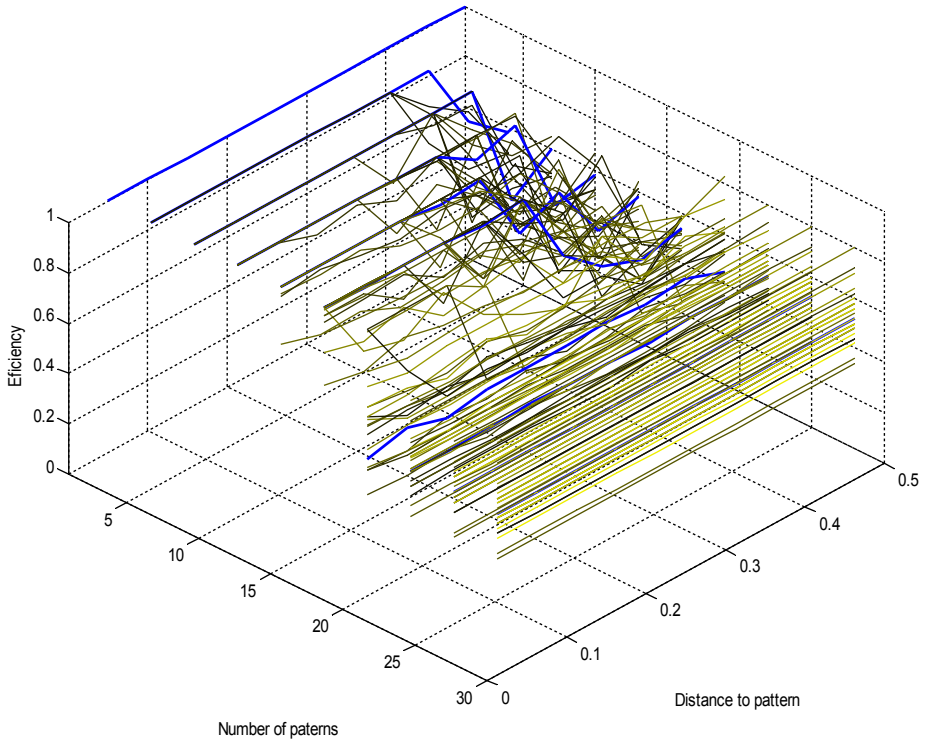
$\lambda = 0{,}99$



$\lambda = 1$

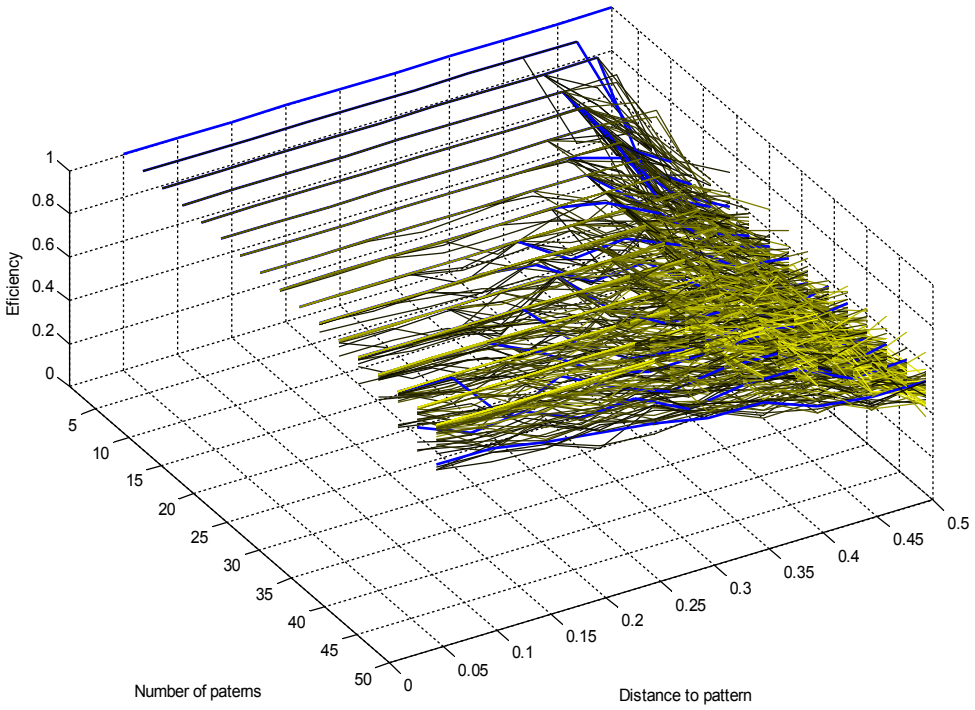


$\lambda = 1{,}01$

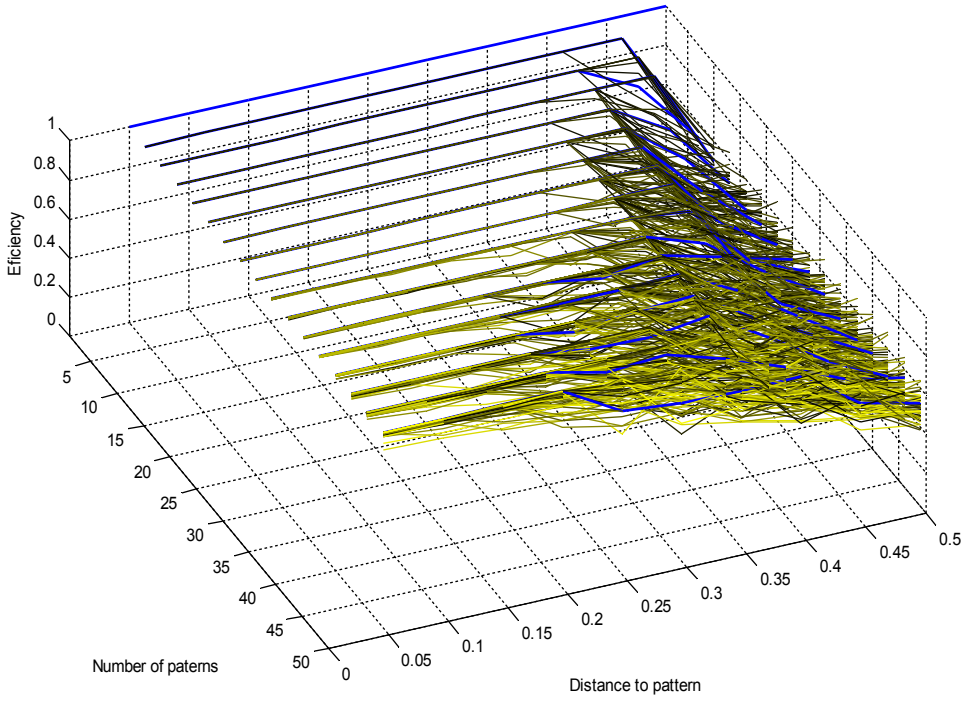


N=400.

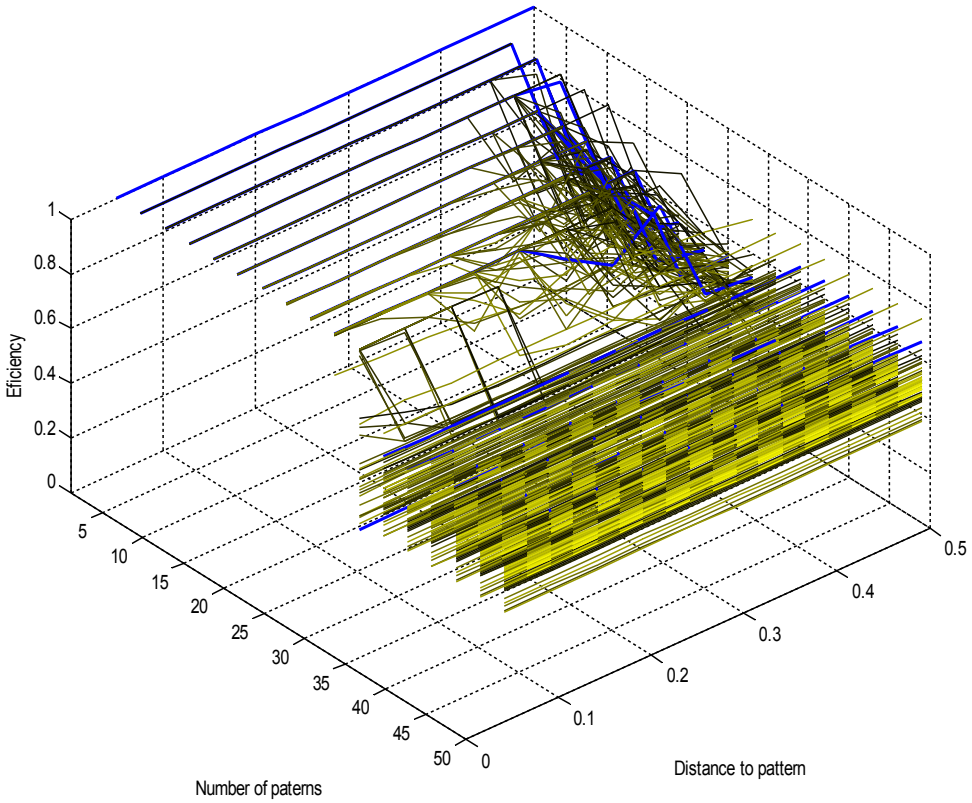
$\lambda = 0{,}99$



$\lambda = 1$

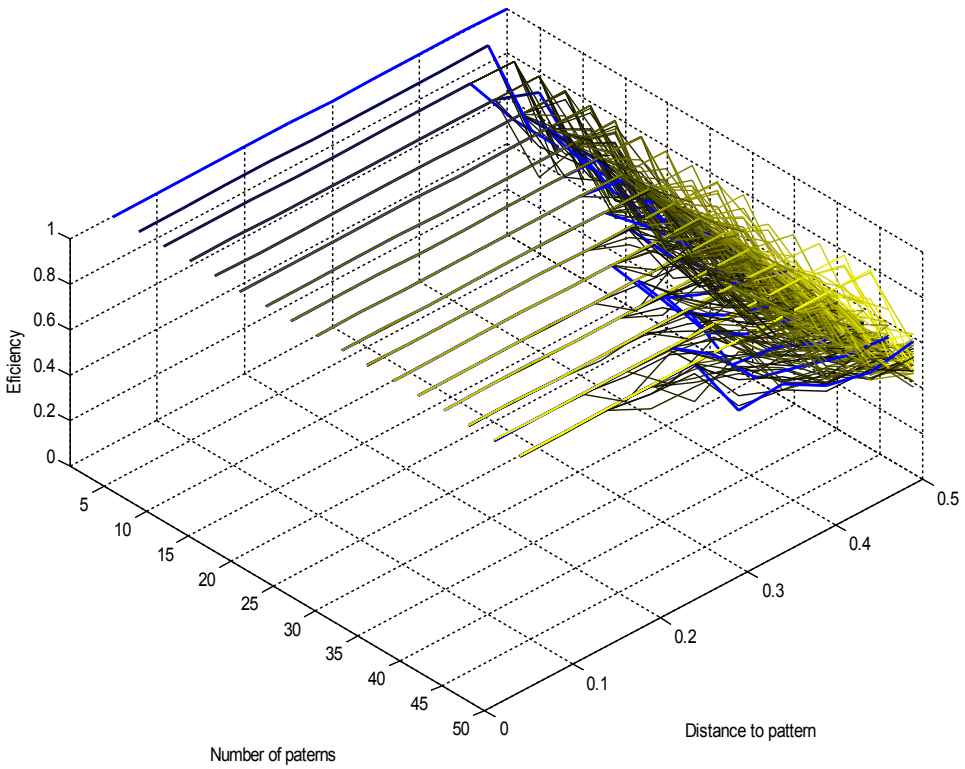


$\lambda = 1{,}01$

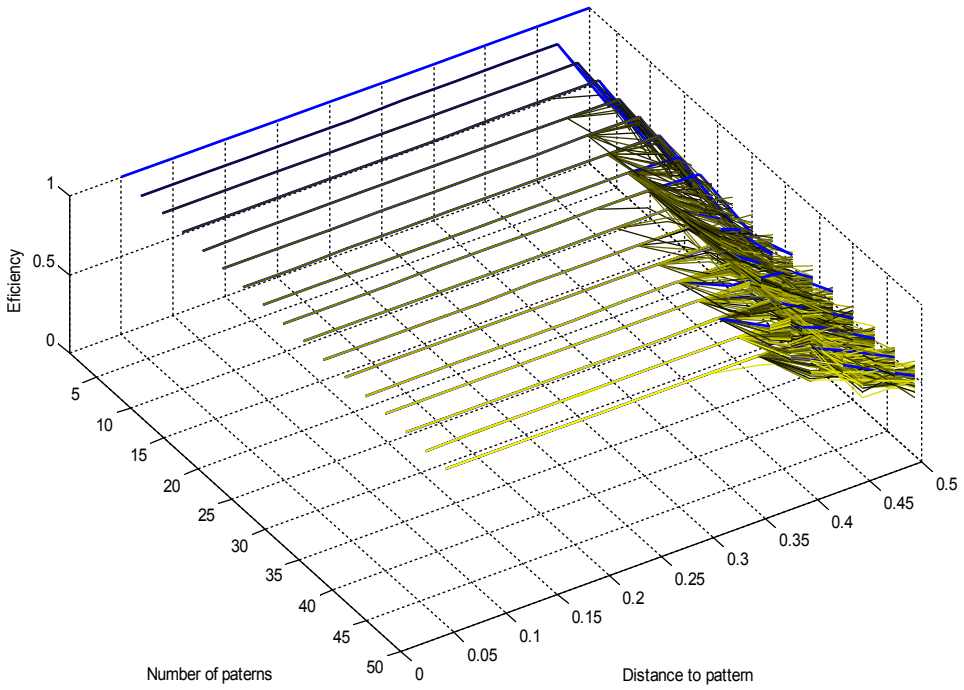


N=900.

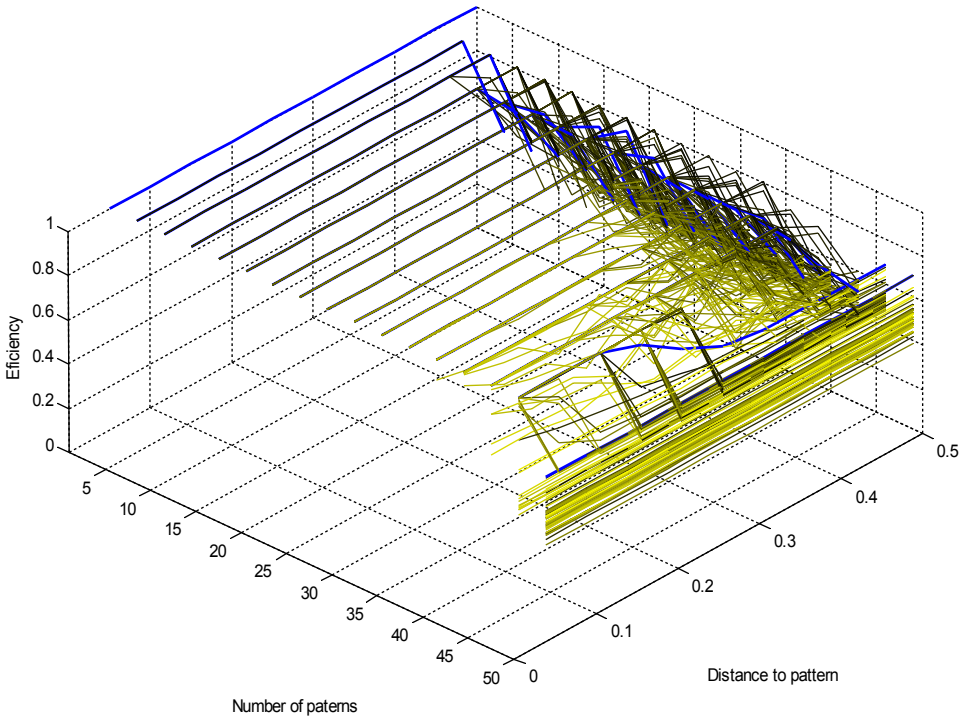
$\lambda = 0{,}99$



$\lambda = 1$



$\lambda = 1{,}01$


Eficiency
1
0.8
0.6
0.4
0.2
0
5
10
15
20
25
30
35
40
45
50
Number of paterns
0
0.1
0.2
0.3
0.4
0.5
Distance to pattern

Apéndice C.

El modelo de red que utilizamos en este trabajo se define como una clase dentro de Matlab con sus correspondientes métodos y propiedades. Este es el código para la clase. Los métodos son solamente dos, uno para crear la red y otro para realizar simulaciones. Según se ajusten los parámetros de las simulaciones la red realizará recuperaciones o entrenamientos.

```
classdef hopfieldnet
%Red Neural Hopfield con neuronas estocasticas. Definición de clase para
%Matlab.
%
%
%
%
% This function was written by :
%               Héctor Corte
%               B.Sc. in physics 2010
%               Battery Research Laboratory
%               University of Oviedo
%               Department of Electrical, Computer, and Systems Engineering
%               Campus de Viesques, Edificio Departamental 3 - Office 3.2.05
%               PC: 33204, Gijón (Spain)
%               Phone: (+34) 985 182 559
%               Fax: (+34) 985 182 138
%               Email: leo_corte@yahoo.es
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   properties
        %La red neural se supone distribuida sobre una matriz de m x n
        %neuronas. De esta forma es más fácil introducir patrones, pero hay
        %libertad para que cada neurona se conecte con cualquier otra.
        %Aquí se definen los parámetros que usa la red.

        %Número de neuronas.
        m
        n
        %Matriz de pesos.
        W
        %La suma de las señales que recibe cada neurona.
        H
        %Los estados de las neuronas
        S
        %Numero de veces que se ejecuta la simulación sin actualizar pesos
        tau
        %Número de veces que se ejecuta el entrenamiento entre iteraciones
        %sin entrenamiento.
        T
        %Número de pasos que tiene la simulación.
        times
        %Beta, sirve para fijar el nivel de ruido.
        B
        %En la siguiente propiedad iremos almacenando la actividad media que
        %tuvieron las neuronas desde la ultima vez que se entrenó.
        media
        %Este contador servirá para contar cuantas iteraciones han tenido
        %lugar entre entrenamientos.
        contador
        %Esta matriz/vector va a indicar qué neuronas están activas y cuales
        %muertas.
```

```
        dead
        %Por ultimo la tasa de olvido pasivo.
        lambda
    end

    methods
          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
          %Aquí definimos los métodos que se van a poder usar con la red.
          %El primero es el de creación, simplemente inicializa las
          %propiedades de la red.
          function obj = hopfieldnet(varargin)
              obj.dead{1,1}=ones(obj.m*obj.n,obj.m*obj.n);
              obj.contador=0;
              obj.B=1;
              obj.T=1;
              obj.tau=1;
              obj.m=varargin{1};
              obj.n=varargin{2};
              obj.W=cell(1,1);
              obj.S=cell(1,1);
              obj.H=cell(1,1);
              obj.W{1,1}=zeros(obj.m*obj.n,obj.m*obj.n);
              obj.S{1,1}=rand(obj.m*obj.n,1);
              obj.H{1,1}=rand(obj.m*obj.n,1);
              obj.media{1,1}=zeros(obj.m*obj.n,1);
              obj.dead{1,1}=ones(obj.m*obj.n,obj.m*obj.n);
              obj.lambda=1;
          end
          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
          %El siguiente método es el principal, sirve para simular el
          %funcionamiento de la red. Variando los parámetros simulamos las
          %iteraciones normales o los entrenamientos.
          function [outputs,obj]=sim(obj,quimic,tau,T,times,varargin)
              %Está programado para que las entradas típicas sean:


              %
[outputs,obj2]=sim(obj,tau,T,times,epsilon1,nepsilon1,epsilon2,nepsilon2,epsilo
n3,nepsilon3,...)

              %Donde:
              %quimic es una variable que no se utiliza pero que se había
pensado en un principio para modificar la actividad de la red en función de la
concentración de ciertos neurotransmisores.
              % tau es el número de iteraciones entre entrenamientos.
              % T es el número de iteraciones durante el entrenamiento
              %times es el número total de iteraciones que se harán.
              %epsilon1 es la señal que se sumará primero.
              %nepsilon1 es el número de iteraciones que se sumará ese
              %patrón. --> Si la señal es de ceros, entonces es como si no
              %hubiese nada.
              %Podemos poner tau menor que times y entonces sólo simulamos,
              %nunca entrenamos.
              %epsilon2 es el segundo patron......
              %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
              alf=0.*obj.H{1,1};
              [a,~]=size(obj.W{1,1});
              obj.tau=tau;
              obj.T=T;
              obj.times=times;
              temp=0;%Va a controlar que hagamos los pasos que se indican en
times.
                 for kk=0:(nargin-5)/2-1
```

```
                    %Simulamos para cada señal de las que hemos
                    %introducido.

                    for l=1:a %Con esto eliminamos la autorecursividad
                       obj.W{1,1}(l,l)=0;
                    end


                    %Vamos cargando cada señal.
                    inputsmatrix=varargin{2*kk+1};
                    inputs=zeros(obj.n*obj.m,1);

                    %%%%%%%%%%%%%Simulación%%%%%%%%%%%%%%%%%%%
                              for i=1:(obj.n*obj.m)
                                  inputs(i)=inputsmatrix(i);
                              end
                              K=varargin{2*kk+2};%Va a controlar que cada patrón
se enseñe sólo las veces que viene indicado.
                              k=0;
                              for t=1:obj.tau
                                  if k>=K || temp>=obj.times
                                      break
                                  end
                                  k=k+1;
                                  temp=temp+1;
                                  obj.contador=obj.contador+1;
                                  obj.H{1,1}=obj.W{1,1}*obj.S{1,1}+1*inputs;
                                  %obj.S{1,1}=2.*(obj.H{1,1}>0)-1;

alf=1/2*(1+tanh(obj.B.*(obj.H{1,1})))>(rand(obj.m*obj.n,1));
                                  %
%alf=abs(tanh(B.*(obj.H{1,1})))>(rand(obj.m*obj.n,1));
                                  %Modificar estas dos líneas es porque
                                  %antes no estaba bien codificado
                                  %%obj.S{1,1}=(alf).*(sign(obj.H{1,1}))+(1-
alf).*(-sign(obj.H{1,1}));
                                  obj.S{1,1}=(alf).*(ones(obj.m*obj.n,1))+(1-
alf).*(-ones(obj.m*obj.n,1));

                                  obj.media{1,1}=obj.media{1,1}+1*obj.S{1,1};

                              end
                     %%%%%%%%%%%%%Entrenamiento%%%%%%%%%%%%%%%%%%%
                              for t=1:obj.T

                                  if k>=K || temp>=obj.times
                                      break
                                  end
                                  k=k+1;
                                  temp=temp+1;
                                  %Realizamos una simulación.
                                  obj.H{1,1}=obj.W{1,1}*obj.S{1,1}+1*inputs;

alf=abs(tanh(obj.B.*(obj.H{1,1})))>(rand(obj.m*obj.n,1));
                                  obj.S{1,1}=(alf).*(sign(obj.H{1,1}))+(1-
alf).*(-sign(obj.H{1,1}));
                                  %Hallamos la acftividad media de las
                                  %neuronas.
                                  obj.contador=obj.contador+1;
                                  obj.media{1,1}=obj.media{1,1}+obj.S{1,1};
                                  obj.media{1,1}=obj.media{1,1}/obj.contador;
                                  %Actualizamos los pesos utilizando la
                                  %versión con olvido pasivo y LTP/LTD
```

```
                                    obj.W{1,1}=obj.lambda*obj.W{1,1}+1/
(obj.n*obj.m)*(obj.media{1,1})*(obj.media{1,1})';
                                    %Por seguridad volvemos a imponer la
                                    %condición de no autofeedback.
                                    for l=1:a
                                        obj.W{1,1}(l,l)=0;
                                    end
                                    %Ahora mantenemos muertas las neuronas que
                                    %estén muertas
                                    obj.W{1,1}=obj.dead{1,1}.*obj.W{1,1};
                                     obj.contador=0;
                                    obj.media{1,1}=0.*obj.media{1,1};

                                end
                                if temp>=obj.times
                                    break
                                end
                            end
                            %Cuando termina la simulación damos a la salida el estado
                            %actual de la red.
                            outputs=0.*inputsmatrix;
                            for i=1:(obj.n*obj.m)
                                outputs(i)=obj.S{1,1}(i);
                            end
                    end
               end
          end
```

Como complemento a la definición de clase se define una función para la distancia Hamming.

```
function H=Hammingdist(A,B)
%Esta función sirve para calcular la distancia Hammming.
%Aprobechando que las matrices tienen entradas -1 o´ 1 esta función da el
%solapamiento entre las matrices A y B.
H=sum(sum(A~=B,1));
end
```

Apéndice D.

Este código sirve para simular una red con n x n neuronas y evaluar su resistencia al ruido. Para ello se le enseñan 5 patrones y se analiza la eficiencia de la red en función del ruido de la señal de entrada.

```
%%Primera simulación
function eval02(n,mu)
% n x n Tamaño de la red.
% mu distancia mínima entre patrones.

numpaterns=5; %Número de patrones distintos
numof=10;%Número de puntos que tomamos a lo largo de la degradacion

A=cell(numpaterns,numof);%Será donde guardaremos los patrones y también los
%patrones con ruido.
B=cell(numpaterns);%Será donde guardaremos la eficiencia a la hora de
%recuperar cada patrón en función del ruido de la señal.

%Comenzamos generando los patrones que vamos a necesitar.
for i=1:numpaterns
    %Generamos los patrones exigiendo una distancia mínima entre ellos.
    if i>1
   A{i,1}= ruido(A{i-1,1},mu);%Así aseguramos la máxima distancia posible
   %entre patrones. En principio, como es un proceso aleatorio y el tamaño
   %de la red es suficientemente grande, este proceso basta para garantizar
   %una distancia mínima entre patrones.
    else
      A{i,1}= patern(n,n);
    end
    %Para cada patrón generamos una
   for j=2:numof
   A{i,j}=ruido(A{i,1},0+j*0.5/numof);
    end
end
MyNet=hopfieldnet(n,n);
MyNet.B=1/5;
[MyNetoutputs1,MyNet]=sim(MyNet,[],100,0,1,A{i,1},100,0.*A{i,1},0); %Esto es
para iniciar las variables.

%Entrenamiento%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=1:numpaterns
MyNet.media{1,1}=0.*MyNet.media{1,1};%Tenemos que reiniciar el contador
%porque no queremos que la red aprenda parte del estado de reposo.
MyNet.contador=0;
MyNet.B=1/5;%También reiniciamos el factor B.
for k=1:30
    %Entrenamos la red manteniendo la señal a la entrada durante 30
    %iteraciones y variando el factor beta.
    [MyNetoutputs1,MyNet]=sim(MyNet,[],30,1,1,A{i,1},1);
    MyNet.B=5+k/30*(-5+1/5);
end
%En la iteración 31 entrenamos la red.
[MyNetoutputs1,MyNet]=sim(MyNet,[],0,1,1,A{i,1},1);
MyNet.media{1,1}=0.*MyNet.media{1,1};%Volvemos a reiniciar el contador.
MyNet.contador=0;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Recuperación%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Para cada patrón generamos una señal con un nivel de ruido cada vez mayor
```

```
%y evaluamos la respuesta de la red al recuperarla. Para cada nivel de
%ruido y para cada patrón se realizan 5 intentos para promediar la
%respuesta.
media=0;
for i=1:numpaterns
    for j=1:numof
        media=0;
        for lo=1:5
            MyNet.contador=0;
            MyNet.B=1/5;
            MyNet.B=10;
            for mn=1:30
                %Para cada señal se hacen 30 iteraciones y se evalua la
                %salida de la red para ver a donde ha convergido.
                MyNet.B=1/5+mn/30*(-1/5+5);
                [MyNetoutputs1,MyNet]=sim(MyNet,
[],30,0,1,A{i,j},25,0.*A{i,j},250);
            end
            [MyNetoutputs1,MyNet]=sim(MyNet,
[],30,0,30,0.*A{i,j},30,0.*A{i,j},250);
            A{i,j}=ruido(A{i,1},0+(j-1)*0.5/numof);%Para la misma distancia
            %al patrón original, generamos una señal distinta y así poder
promediar.
            media=media+ 1-Hammingdist(A{i,1},MyNetoutputs1)/(n*n);
        end
        B{i}(1,j)=0+(j-1)*0.5/numof;
        B{i}(2,j)=media/5;%Determinamos la eficiencia promedio.
    end
end
%Por ultimo representamos los resultados.
figure
for i=1:numpaterns
    plot(B{i}(1,1:numof),B{i}(2,1:numof),'-');
    hold on
end
xlabel('Distance to pattern');
ylabel('Eficiency');
end

function [B]=patern(n,m)
%Con esta función generamos patrones aleatorios.
B=2*round(rand(n,m))-1;
end

function [B]=ruido(A,h)
%Utilizando un patrón A como base, y generamos otro patrón B que diste al
%menos h de A.
[n,m]=size(A);
seed0=ones(n,m);
seed=ones(n,m);
k=randi(n*m,1,100*n*m);
i=1;
while (Hammingdist(seed0,seed)/(n*m)<h)==1
    seed(k(i))=-seed(k(i));
    i=i+1;
end
B=seed.*A;
end
```

Apéndice E.

Este código sirve para simular una red con n x n neuronas y evaluar su funcionamiento a medida que le enseñamos más y más patrones. El código va enseñando patrones a la red y luego evalúa la eficiencia recuperando todos los patrones almacenados en función del ruido de la señal de entrada.

```
%Segunda simulación. Primera parte.
function eval01(n,lambda)
% n x n Tamaño de la red.
% lambda es el valor de la tasa de olvido pasivo.

numpaterns=20;%Número máximo de patrones a evaluar.
numof=10;%Número de puntos que tomamos a lo largo de la degradacion.
mu=0.4;%Distancia mínima entre patrones.

A=cell(numpaterns,numof);%Será donde guardaremos los patrones y también los
%patrones con ruido.
B=cell(numpaterns,numpaterns);%Será donde guardaremos la eficiencia a la hora
de
%recuperar cada patrón en función del ruido de la señal.

%Comenzamos generando los patrones que vamos a necesitar.
for i=1:numpaterns
    %Generamos los patrones exigiendo una distancia mínima entre ellos.
    if i>1
        A{i,1}= ruido(A{i-1,1},mu);%Así aseguramos la máxima distancia posible
        %entre patrones. En principio, como es un proceso aleatorio y el tamaño
        %de la red es suficientemente grande, este proceso basta para
garantizar
        %una distancia mínima entre patrones.
    else
        A{i,1}= patern(n,n);
    end
    %Para cada patrón generamos una
    for j=2:numof
        A{i,j}=ruido(A{i,1},0+j*0.5/numof);
    end
end
%Generamos la red y precargamos las variables que va a utilizar.
MyNet=hopfieldnet(n,n);
MyNet.lambda=lambda;
MyNet.B=5;
[MyNetoutputs1,MyNet]=sim(MyNet,[],100,0,1,A{i,1},100,0.*A{i,1},0);

%%%%En el siguiente algoritmo iremos enseñando patrones a la red y
%%%%evaluando si es capaz de recordar los ya aprendidos.
for i=1:numpaterns
    %Primero entrenamos la red usando el procedimiento habitual
    MyNet.media{1,1}=0.*MyNet.media{1,1};%Tenemos que reiniciar el contador
porque no queremos que la red aprenda parte del estado de reposo.
    MyNet.contador=0;
    for k=1:30
        %Entrenamos la red manteniendo la señal a la entrada durante 30
        %iteraciones y variando el factor beta.
        [MyNetoutputs1,MyNet]=sim(MyNet,[],30,1,1,A{i,1},1);
        %Variamos el valor de beta a medida que entrenamos.
        MyNet.B=5+k/30*(-5+1/5);
    end
    %En la iteración 31 entrenamos la red.
    [MyNetoutputs1,MyNet]=sim(MyNet,[],0,1,1,A{i,1},1);
```

```
    MyNet.media{1,1}=0.*MyNet.media{1,1};%Volvemos a reiniciar el contador.
    MyNet.contador=0;

    %A continuación hacemos la recuperación.
    %Aquí intentamos recuperar los patrones y evaluamos la respuesta
    %promediando 10 señales distintas para cada patrón y para cada nivel de
    %ruido.
    for k=1:i
        for j=1:numof
            %El nivel de ruido se toma para tener tantos puntos como los
            %que especifiquemos en numof
            B{i,k}(2,j)=0;
            for r=1:10
            for mn=1:30
                %Aquí variamos el valor de beta de acuerdo a lo expuesto en
                %la descripción de la recuperación.
                MyNet.B=1/5+mn/30*(5-1/5);
                [MyNetoutputs1,MyNet]=sim(MyNet,[],30,0,1,A{k,j},25);
            end
            [MyNetoutputs1,MyNet]=sim(MyNet,[],30,0,30,0.*A{k,j},35);
            B{i,k}(1,j)=0+j*0.5/numof;
            1-Hammingdist(A{k,1},MyNetoutputs1)/(n*n);
            B{i,k}(2,j)=B{i,k}(2,j)+1-Hammingdist(A{k,1},MyNetoutputs1)/(n*n);
            end
            B{i,k}(2,j)=B{i,k}(2,j)/10;
        end
    end
end

%Al final, para simplificar la visualización, mostramos sólo los resultados
%cada vez que se aprenden 3 nuevos patrones.
contador=3;
figure;
for i=1:numpaterns
    if contador==3
        contador=0;
        for k=1:i
            if k==1
                plot3(i*ones(1,numof),B{i,k}(1,1:numof),B{i,k}(2,1:numof),'-
b','linewidth',2);
            else
                plot3(i*ones(1,numof),B{i,k}(1,1:numof),B{i,k}
(2,1:numof),'-','color',[k/numpaterns,k/numpaterns,0]);
            end
            hold on
        end
    end
    contador=contador+1;
end
xlabel('Number of paterns');
ylabel('Distance to pattern');
zlabel('Eficiency');
axis([1,numpaterns,0,0.5,0,1]);
end



function [B]=patern(n,m)
%Con esta función generamos patrones aleatorios.
B=2*round(rand(n,m))-1;
end

function [B]=ruido(A,h)
```

```
%Utilizando un patrón A como base, y generamos otro patrón B que diste al
%menos h de A.
[n,m]=size(A);
seed0=ones(n,m);
seed=ones(n,m);
k=randi(n*m,1,100*n*m);
i=1;
while (Hammingdist(seed0,seed)/(n*m)<h)==1
    seed(k(i))=-seed(k(i));
    i=i+1;
end
B=seed.*A;
end
```

Apéndice F.

Este código es una modificación del código del apéndice E. Aquí simplemente calculamos la respuesta para señales que distan 0,02 del patrón que se quiere recuperar. De esta forma contamos el número de patrones que recuerda la red.

```
%Segunda simulación. Segunda parte.
function eval01b(n,lambda,numpaterns)

% n x n Tamaño de la red.
% lambda es el valor de la tasa de olvido pasivo.
%numpaterns Número máximo de patrones a evaluar.

numof=1;%Número de puntos que tomamos a lo largo de la degradacion.
mu=0.45;%Distancia mínima entre patrones.

A=cell(numpaterns,numof);%Será donde guardaremos los patrones y también los
%patrones con ruido.
B=cell(numpaterns,numpaterns);%Será donde guardaremos la eficiencia a la hora
de
%recuperar cada patrón en función del ruido de la señal.

%Comenzamos generando los patrones que vamos a necesitar.
for i=1:numpaterns
    %Generamos los patrones exigiendo una distancia mínima entre ellos.
    if i>1
        A{i,1}= ruido(A{i-1,1},mu);%Así aseguramos la máxima distancia posible
        %entre patrones. En principio, como es un proceso aleatorio y el tamaño
        %de la red es suficientemente grande, este proceso basta para
garantizar
        %una distancia mínima entre patrones.
    else
        A{i,1}= patern(n,n);
    end
    %Para cada patrón generamos una
    for j=2:numof
        A{i,j}=ruido(A{i,1},0.02);
    end
end
%Generamos la red y precargamos las variables que va a utilizar.
MyNet=hopfieldnet(n,n);
MyNet.lambda=lambda;
MyNet.B=5;
[MyNetoutputs1,MyNet]=sim(MyNet,[],100,0,1,A{i,1},100,0.*A{i,1},0);

%%%%En el siguiente algoritmo iremos enseñando patrones a la red y
%%%%evaluando cuantos es capaz de recordar de los ya aprendidos.

for i=1:numpaterns
    %Primero entrenamos la red usando el procedimiento habitual
    MyNet.media{1,1}=0.*MyNet.media{1,1};%Tenemos que reiniciar el contador
porque no queremos que la red aprenda parte del estado de reposo.
    MyNet.contador=0;
    for k=1:30
        %Entrenamos la red manteniendo la señal a la entrada durante 30
        %iteraciones y variando el factor beta.
        [MyNetoutputs1,MyNet]=sim(MyNet,[],30,1,1,A{i,1},1);
        %Variamos el valor de beta a medida que entrenamos.
        MyNet.B=5+k/30*(-5+1/5);
    end
    %En la iteración 31 entrenamos la red.
```

```
    [MyNetoutputs1,MyNet]=sim(MyNet,[],0,1,1,A{i,1},1);
    MyNet.media{1,1}=0.*MyNet.media{1,1};%Volvemos a reiniciar el contador.
    MyNet.contador=0;


    %A continuación hacemos la recuperación.
    %Aquí intentamos recuperar los patrones y evaluamos la respuesta
    %promediando 10 señales distintas para cada patrón y para cada nivel de
    %ruido.
    for k=1:i
        for j=1:numof
            %El nivel de ruido se toma para tener tantos puntos como los
            %que especifiquemos en numof
            B{i,k}(2,j)=0;
            for r=1:10
                for mn=1:30
                    %Aquí variamos el valor de beta de acuerdo a lo expuesto en
                    %la descripción de la recuperación.
                    MyNet.B=1/5+mn/30*(5-1/5);
                    [MyNetoutputs1,MyNet]=sim(MyNet,[],30,0,1,A{k,j},25);
                end
                [MyNetoutputs1,MyNet]=sim(MyNet,[],30,0,30,0.*A{k,j},35);
                B{i,k}(1,j)=0.02;
                B{i,k}(2,j)=B{i,k}(2,j)+1-Hammingdist(A{k,1},MyNetoutputs1)/
(n*n);
            end
            B{i,k}(2,j)=B{i,k}(2,j)/10;
        end
    end


end
%Por ultimo representamos la capacidad en función del número de patrones
%enseñados.
contador=0;
y=zeros(1,numpaterns);
figure;
for i=1:numpaterns
    contador=0;
    for k=1:i
        if B{i,k}(2,1)>0.8
            contador=contador+1;
        end
    end
    y(i)=contador/(n*n);
end
plot(1:numpaterns,y,'k');
end


function [B]=patern(n,m)
%Con esta función generamos patrones aleatorios.
B=2*round(rand(n,m))-1;
end

function [B]=ruido(A,h)
%Utilizando un patrón A como base, y generamos otro patrón B que diste al
%menos h de A.
[n,m]=size(A);
seed0=ones(n,m);
seed=ones(n,m);
k=randi(n*m,1,100*n*m);
i=1;
```

```
while (Hammingdist(seed0,seed)/(n*m)<h)==1
    seed(k(i))=-seed(k(i));
    i=i+1;
end
B=seed.*A;
end
```

Apéndice G.

Este código sirve para evaluar la eficiencia de la red a medida que vamos matando sinapsis.

```
%Tercera simulación
function eval03(n,lambda)
%Vamos a ir haciendo pruebas para ver si podemos analizar como va perdiendo
%efectividad la red. Para ello entrenaremos 5 patrones y luego evaluaremos
%la efectividad de la red a medida que vamos eliminando sinapsis.


% n El número de neuronas de la red es n x n.
% lambda Es la tasa de olvido pasivo.
r=0.45;%Distancia mínima entre patrones.
d=0.1;%Distancia de las señales al patrón.

numpaterns=5;
A{1,1}=patern(n,n);
for k=2:numpaterns
    A{k,1}= ruido(A{k-1,1},r);
end

%Generamos la red y precargamos variables.
hop=hopfieldnet(n,n);
hop.lambda=lambda;
hop.tempera=1/5;
hopoutputs1=cell(1,numpaterns);
[hopoutputs1{1},hop]=sim(hop,[],30,0,1,A{1,1},1);

%Entrenamos la red.
for i=1:numpaterns
    hop.media{1,1}=0.*hop.media{1,1};%Tenemos que reiniciar el contador porque
no queremos que la red aprenda parte del estado de reposo.
    hop.contador=0;
    hop.tempera=1/5;
    for k=1:30
        [hopoutputs1{i},hop]=sim(hop,[],30,0,1,A{i,1},1);
        hop.tempera=1/5+k/30*(-1/5+5);
        hop.media{1,1}=0.*hop.media{1,1};%Tenemos que reiniciar el contador
porque no queremos que la red aprenda parte del estado de reposo.
        hop.contador=0;
    end
    [hopoutputs1{i},hop]=sim(hop,[],300,0,300,A{i,1},300);
    [hopoutputs1{i},hop]=sim(hop,[],0,1,1,A{i,1},1);
    hop.media{1,1}=0.*hop.media{1,1};%Tenemos que reiniciar el contador porque
no queremos que la red aprenda parte del estado de reposo.
    hop.contador=0;
    [hopoutputs1{i},hop]=sim(hop,[],30,0,30,0.*A{i,1},30);
    hop.media{1,1}=0.*hop.media{1,1};%Tenemos que reiniciar el contador porque
no queremos que la red aprenda parte del estado de reposo.
    hop.contador=0;
end

%%%%%%%%Ahora tenemos la red entrenada. Vamos a ir matando conexiones y
%%%%%%%%viendo cual es la capacidad para recuperar patrones.
%Precargamos variables.
[a,b]= size(hop.dead{1,1});
m=1;
t=1;
promedio1=0;
```

```
promedio2=0;
promedio3=0;
promedio4=0;
promedio5=0;
npromedios=5;

H1=cell(1,numpaterns);
DF=ruido(A{1,1},d);

randx=randi([1,a],1,2*a*b);
randy=randi([1,b],1,2*a*b);

Deadconexions=0;
DeadconexionsF=zeros(1,2*a*b);
H1F=zeros(1,2*a*b);
H2F=zeros(1,2*a*b);
H3F=zeros(1,2*a*b);
H4F=zeros(1,2*a*b);
H5F=zeros(1,2*a*b);
%Matamos algunas neuronas
for k=1:2*a*b
    hop.dead{1,1}(randx(k),randy(k))=0;
    m=m+1;
    Deadconexions=100-sum(sum(hop.dead{1,1}))/(a*b)*100;
    %Vamos matando conexiones al azar, y sólo analizamos la eficiencia cada
    %cierto porcentaje de muertes.
    if m==round(2*a*b/20)
        hop.W{1,1}=hop.dead{1,1}.*hop.W{1,1};
        %Intentamos recuperar los patrones con la red dañada
        %Se hacen una serie de promedios para calcular la eficiencia.
        for j=1:npromedios
            for i=1:numpaterns
                hop.media{1,1}=0.*hop.media{1,1};%Tenemos que reiniciar el
contador porque no queremos que la red aprenda parte del estado de reposo.
                hop.contador=0;
                hop.tempera=1/5;
                DF=ruido(A{i,1},d);
                for op=1:30
                    hop.tempera=1/5+k/30*(-1/5+5);
                    [hopoutputs1{i},hop]=sim(hop,[],30,0,1,DF,1);
                end
                [hopoutputs1{i},hop]=sim(hop,[],300,0,300,DF,300);
                [hopoutputs1{i},hop]=sim(hop,[],30,0,30,0.*DF,30);
                hop.media{1,1}=0.*hop.media{1,1};%Tenemos que reiniciar el
contador porque no queremos que la red aprenda parte del estado de reposo.
                hop.contador=0;
                H1{i}=Hammingdist(A{i,1},hopoutputs1{i});
            end
            promedio1=promedio1+H1{1};
            promedio2=promedio2+H1{2};
            promedio3=promedio3+H1{3};
            promedio4=promedio4+H1{4};
            promedio5=promedio5+H1{5};
        end
        DeadconexionsF(t)=Deadconexions;
        H1F(t)=promedio1/npromedios;
        H2F(t)=promedio2/npromedios;
        H3F(t)=promedio3/npromedios;
        H4F(t)=promedio4/npromedios;
        H5F(t)=promedio5/npromedios;
        t=t+1;
        promedio1=0;
        promedio2=0;
```

```
        promedio3=0;
        promedio4=0;
        promedio5=0;
        m=1;
    end
end

%Una vez calculadas todas las simulaciones limpiamos las variables y
%representamos la eficiencia con cada patrón a medida que van borrandose
%conexiones.
DeadconexionsF(t+1:end)=[];
H1F(t+1:end)=[];
H2F(t+1:end)=[];
H3F(t+1:end)=[];
H4F(t+1:end)=[];
H5F(t+1:end)=[];

figure
plot(DeadconexionsF,1-H1F/(n*n),'.b')
hold on
plot(DeadconexionsF,1-H2F/(n*n),'.r')
plot(DeadconexionsF,1-H3F/(n*n),'.g')
plot(DeadconexionsF,1-H4F/(n*n),'.k')
plot(DeadconexionsF,1-H5F/(n*n),'.y')
xlabel('%Of dead conexions')
ylabel('Retrieval eficiency')

end

function [B]=patern(n,m)
%Con esta función generamos patrones aleatorios.
B=2*round(rand(n,m))-1;
end

function [B]=ruido(A,h)
%Utilizando un patrón A como base, y generamos otro patrón B que diste al
%menos h de A.
[n,m]=size(A);
seed0=ones(n,m);
seed=ones(n,m);
k=randi(n*m,1,100*n*m);
i=1;
while (Hammingdist(seed0,seed)/(n*m)<h)==1
    seed(k(i))=-seed(k(i));
    i=i+1;
end
B=seed.*A;
end
```