**Universidad de las Ciencias Informáticas**
**Facultad 5**


# TITULO: APLICACIONES DE LAS REDES NEURONALES EN ENTORNOS VIRTUALES.

**Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas**

**Autores:**
Arianna Herrera Bacallao
Yunerkis Prevot Urgellés

**Tutor:**
Ms. C. Yuniesky Coca Bergolla

Ciudad de La Habana
Junio, 2008

## Declaración de autoría

Declaro que somos los únicos autores de este trabajo y autorizamos a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

_____ _____

Arianna Herrera Bacallao Yunerkis Prevot Urgellés

_____

Mr. C. Yuniesky Coca Bergolla

# Datos de contacto

Mr. C.  Yuniesky Coca Bergolla
E-mail: ycoca@uci.cu
Telf. 8372474

Graduado de Ciencias de la Computación en la Universidad Central de Las Villas en el año 2003. Profesor de la Universidad de las Ciencias Informáticas desde ese mismo año. Es Jefe de Dpto. de las asignaturas de la especialidad de la Facultad 5 “Entornos Virtuales”, es miembro del Grupo de Desarrollo de Elementos Virtuales Inteligentes. Obtuvo el Sello Forjadores del Futuro en el 2007. Defendió la maestría en el año 2007.

## Agradecimientos

*Agradezco a todas las personas que confiaron en mí,*
*que me apoyaron y me guiaron.*
*A la Revolución Cubana.*
*A mis padres, mi familia, a Abdiel, a mis amigos y mis compañeros de siempre.*
*A mi tutor, y a todos los profesores que hicieron posible este trabajo.*
*Y a Prevot, por ser un amigo incondicional.*
*Arianna*

*Agradezco a todas las personas que de una manera u otra*
*hicieron posible que yo llegara hasta aquí.*
*A la Revolución.*
*A toda mi familia en general, amigos, y compañeros de todos los tiempos.*
*A mi tutor, y a todos los profesores que contribuyeron con mi formación.*
*Y a Arianna, por su confianza en mí.*
*Prevot*

## Dedicatoria

*A mis papás Matilde y Urbano, por haber confiado en mi.*
*A mis hermanos Yoslay y Alejandro.*
*A Alida por su ayuda incondicional.*
*A Abdiel por estar siempre conmigo cuando lo he necesitado.*
*A mis abuelos por su cariño.*
*A mi tío Pedro por su apoyo.*
*A toda mi familia.*
*A mis amigos y compañeros de grupo.*
*Y a Prevot por su paciencia.*
*Arianna*

*A mi hermana Yuimé por haber visto siempre por mí.*
*A mi mamá Claudia, por apoyarme por sobre todas las cosas.*
*A mi hermano Lisay, por estar siempre ahí cuando lo he necesitado.*
*A mis hermanos Odelvis y Yorli.*
*A Idarmis, Carlitos y Jorge.*
*A mis tíos Ana, Rafelito, Iche y Jorge.*
*A toda mi familia.*
*A todos mis amigos.*
*A todos mis compañeros de estudio.*
*Y a Arianna, por su preocupación.*
*Prevot*

# Resumen

Uno de los problemas más comunes ante el cual se enfrentan los desarrolladores de aplicaciones para realidad virtual, es cómo incorporar inteligencia artificial en sus productos. Desean aplicaciones interactivas para los usuarios finales, las cuales sean capaces de sorprender con los atractivos de un medio mucho más inteligente y lo menos predecible posible.
En esta tesis le brindamos la posibilidad de descubrir las redes neuronales como posibles técnicas de inteligencia artificial a tener en cuenta para lograr los resultados deseados. Cómo utilizarlas en la solución de problemas reales concretos, y a su vez, cómo podríamos reutilizar el diseño de una red dada para una solución en otra y algunos de los resultados obtenidos en su aplicación en entornos virtuales.

## Palabras claves
Entornos virtuales, realidad virtual, inteligencia artificial, redes neuronales artificiales.

# ÍNDICE DE CONTENIDO

## íNDICE DE FIGURAS

# ÍNDICE DE TABLAS

# Introducción

La Informática es una de las ciencias que mayor impacto ha tenido en el desarrollo económico y social a nivel mundial desde finales del siglo pasado hasta el presente. Su evolución ha sido vertiginosa desde su nacimiento, permitiendo el surgimiento de diferentes disciplinas dentro de la misma, que son el centro de atención de los científicos, especialistas y del mercado.
Una de estas áreas es la Realidad Virtual, donde las industrias de juegos y simulación han ocupado los planos vanguardistas de desarrollo e investigación.
Las razones de su popularidad están dadas por el nivel de realidad en la representación de los escenarios y por el poder de interacción con los usuarios. Este último desempeña un papel preponderante en el nivel de aceptación de un producto de Realidad Virtual, ya que los usuarios finales no solo necesitan de un sistema gráfico con un alto nivel de detalles y similitud al mundo real, sino además que sea capaz de retar su inteligencia.
Estas razones conllevaron a que la Inteligencia Artificial tuviera un área sólida de investigación y aplicación en la Realidad Virtual. Muchas son las técnicas que tratan de imprimirles un comportamiento inteligente a los jugadores que interactúan con el usuario (agentes autónomos) dentro de un mundo virtual. Una de las técnicas más populares por su capacidad de proporcionar un alto poder de autonomía son las redes neuronales artificiales (RNA).
Nuestro país ha dado pasos de avances en el mundo de la Informática. A pesar de las limitaciones del bloqueo económico por parte de los Estados Unidos se han desarrollado soluciones tecnológicas de impacto social y económico, dentro de las cuales no faltan aplicaciones de Realidad Virtual.
La Universidad de las Ciencias Informáticas se ha convertido en uno de los centros de vanguardia nacionales en la aplicación, investigación y desarrollo de las nuevas tecnologías. En la misma existen 10 facultades que se dedican a la investigación y el desarrollo en diferentes perfiles del software.
Las redes neuronales artificiales han sido aplicadas en la clasificación de patrones, aproximación de funciones, predicción, control, análisis de datos, compresión de datos, memorización asociativa y categorización de datos, por lo que son objeto de investigación en las ciencias médicas, en la industria militar, en las ciencias económicas, agrícolas, meteorológicas y en la industria de los juegos. En esta última, se han incorporado en algunos de los videos-juegos más populares para la creación de agentes autónomos. En los entornos donde son colocados estos agentes son capaces de brindar un comportamiento inteligente e impredecible para los usuarios.
La facultad 5 se especializa en el perfil de entornos virtuales y automatización. En ella tienen lugar los proyectos de Realidad Virtual, específicamente para juegos y simuladores. Los productos que están actualmente en desarrollo están incorporando diversas técnicas de Inteligencia Artificial, pero no han podido emplear las Redes Neuronales, debido al desconocimiento de cómo podrían aplicarse estas en programas de Realidad Virtual, que tratan de ejecutarse en tiempo real. Esto imposibilita que se puedan obtener productos con agentes autónomos basados en redes neuronales que sean capaces de imprimirle un alto nivel de dinamismo.
Ante este problema, nos hacemos la siguiente pregunta: ¿Cómo aplicar las redes neuronales en los proyectos de Realidad Virtual de la facultad 5?
Como objeto de estudio nos planteamos las Redes Neuronales Artificiales y como objetivos generales**,** modelar propuestas de usos de las redes neuronales en los proyectos de Realidad Virtual que se desarrollan en la facultad 5.
Los objetivos específicos que debemos cumplir son los siguientes:

- Analizar los tipos de redes neuronales existentes.
- Analizar las aplicaciones existentes de redes neuronales en entornos virtuales.
- Seleccionar los tipos de redes neuronales que se pueden emplear dentro de los entornos virtuales.
- Describir los posibles escenarios donde se puedan utilizar las redes neuronales en entornos virtuales.

- Proponer topologías eficientes de redes neuronales para escenarios específicos en entornos virtuales.

Nuestro campo de acción son las redes neuronales aplicables a entornos virtuales.
Los posibles resultados son:

- Tipos de redes neuronales aplicables a entornos virtuales.
- Posibles escenarios donde se puedan utilizar las redes neuronales dentro de entornos virtuales.
- Topologías de redes neuronales adecuadas para ser utilizadas en escenarios específicos de entornos virtuales.
- Llegar a una propuesta de pasos a seguir para diseñar una red neuronal acorde al escenario dado en el entorno virtual.

Los métodos que pretendemos utilizar para realizar nuestra investigación son los siguientes:

**Métodos Teóricos**

- Analítico-Sintético
- Inductivo-Deductivo
- Histórico-Lógico

**Métodos Empíricos**

- Observación
- Experimentación

# Capítulo I Fundamentación Teórica

## 1.1 Introducción

Una Red Neuronal Artificial (RNA), es un sistema de procesadores paralelos conectados entre sí en forma de grafo dirigido, donde cada elemento de procesamiento (neuronas) de la red se representa como un nodo. Estas conexiones establecen una estructura jerárquica que busca nuevos modelos de procesamiento para solucionar problemas concretos del mundo real. Lo importante en el desarrollo de las redes neuronales es su útil comportamiento al aprender, reconocer y aplicar relaciones entre objetos y tramas de objetos propios del mundo real, por lo cual las Redes Neuronales se utilizan como una herramienta para resolver problemas difíciles (Bourg y Glenn, 2004).

## 1.2 Ventajas que ofrecen las redes neuronales.

Debido a su constitución y fundamentos, las redes neuronales artificiales presentan un gran número de características semejantes a las del cerebro. Por ejemplo, son capaces de aprender de la experiencia, de generalizar de casos anteriores a nuevos casos, de abstraer características esenciales a partir de entradas que representan información irrelevante, etc. Esto hace que ofrezcan numerosas ventajas y que este tipo de tecnología se esté aplicando en múltiples áreas (Matich, 2001). Entre las principales ventajas se incluyen:

- **Adaptación:** Poseen la capacidad de aprender nuevos patrones de entrada para los cuales no fueron entrenadas.
- **Auto-organización:** Una red neuronal puede crear su propia organización o representación de la información que recibe mediante una etapa de aprendizaje.
- **Inmune al ruido:** Las redes neuronales pueden reconocer correctamente parámetros de entrada que hayan sido dañados o con algunas diferencias en relación a los valores aprendidos.
- **Generalización:** Son capaces de inferir resultados para información que no fue suministrada durante el proceso de aprendizaje.
- **Procesamiento no lineal:** Los cómputos neuronales pueden ser realizados en paralelo, lo cual garantiza el procesamiento no lineal de la información recibida.

## 1.3 Tipos de funciones de activación.

Las funciones de activación están presentes en cada una de las unidades de procesamiento (neuronas) de una red. Una función toma las entradas a la neurona y las transforma en una salida (Bourg y Glenn, 2004). Existen varias funciones de activación (Hilera, J. R., Martínez, 2000), la tabla 1.1 muestra algunas de las más usadas.

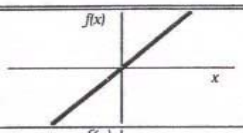
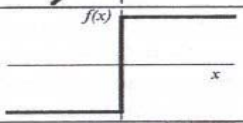
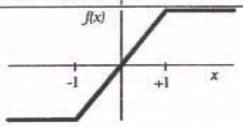
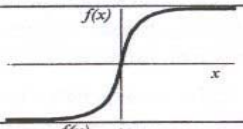
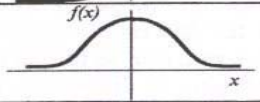
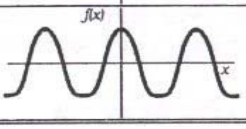
| | Función | Rango | Gráfica |
|---|---|---|---|
| **Identidad** | $y = x$ | $[-\infty, +\infty]$ |  |
| **Escalón** | $y = sign(x)$<br>$y = H(x)$ | $\{-1, +1\}$<br>$\{0, +1\}$ |  |
| **Lineal a tramos** | $y = \begin{cases} -1, & si\ x < -l \\ x, & si\ +l \le x \le -l \\ +1, & si\ x > +l \end{cases}$ | $[-1, +1]$ |  |
| **Sigmoidea** | $y = \frac{1}{1+e^{-x}}$<br>$y = tgh(x)$ | $[0, +1]$<br>$[-1, +1]$ |  |
| **Gaussiana** | $y = Ae^{-Bx^2}$ | $[0,+1]$ |  |
| **Sinusoidal** | $y = A\,\mathrm{sen}(\omega x + \varphi)$ | $[-1,+1]$ |  |

**Tabla 1.1 Funciones de activación más populares.**

## 1.4 Topologías de Redes Neuronales.

Consiste en la organización de las neuronas en la red formando capas o agrupaciones de neuronas más o menos alejadas de la entrada y salida de la red. Los parámetros fundamentales de la red son: el número de capas, el número de neuronas por capa, el grado de conectividad y el tipo de conexiones entre neuronas.
En términos topológicos podemos clasificar las redes en: redes de una sola capa y redes con múltiples capas.

### 1.4.1 Redes monocapa.

Las redes monocapa son redes con una sola capa. Para unirse, las neuronas crean conexiones laterales para conectar con otras neuronas de su capa. Las redes más representativas son la red de Hopfield, la red BRAIN-STATE-IN-A-BOX o memoria asociativa y las máquinas estocásticas de Boltzmann y Cauchy (Matich, 2001).
Entre las redes neuronales monocapa, existen algunas que permiten que las neuronas tengan conexiones a sí mismas y se denominan autorecurrentes.
Las redes monocapa han sido ampliamente utilizadas en circuitos eléctricos ya que debido a su topología, son adecuadas para ser implementadas mediante hardware, usando matrices de diodos que representan las conexiones de las neuronas, se utilizan además típicamente en tareas relacionadas con lo que se conoce como autoasociación, por ejemplo para regenerar informaciones de entrada que se presentan distorsionadas o incompletas.

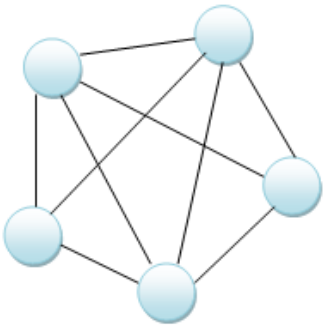


**Figura 1.1 Red Neuronal de una sola capa.**

### 1.4.2 Redes Multicapa.

Las redes multicapa son aquellas que disponen las neuronas agrupadas en varias capas. Una de las formas para distinguir la capa a la que pertenece una neurona, consistiría en fijarse en el origen de las señales que recibe a la entrada y el destino de la señal de salida. Normalmente, todas las neuronas de una capa reciben señales de entrada de otra capa anterior, más cercana a la entrada de la red, y envían su señal de salida a una capa posterior, más cercana a la salida de la red. A estas conexiones se les denominan conexiones hacia adelante o feedforward.
Sin embargo en un gran número de estas redes también existe la posibilidad de conectar las salidas de las neuronas de capas posteriores a las entradas de capas anteriores, a estas conexiones se les denomina conexiones hacia atrás o feedback.
Estas dos posibilidades permiten distinguir entre dos tipos de redes: las redes con conexiones hacia adelante (redes feedforward), y las redes que disponen de conexiones tanto hacia adelante como hacia atrás (redes feedforward/feedback).

### 1.4.3 Redes con conexiones hacia adelante (feedforward).

Las señales de estas se propagan hacia adelante a través de las capas de la red. No existen conexiones hacia atrás, y normalmente tampoco autorecurrentes, ni laterales, excepto los modelos de red propuestos por Kohonen.
Las redes feedforward más conocidas son: PERCEPTRÓN, ADALINE, MADALINE, LINEAR ADAPTATIVE MEMORY, DRIVE-REINFORCEMENT, BACKPROPAGATION (Es una variante del PERCEPTRÓN MULTICAPA (PMC) que utiliza Backpropagation como algoritmo de entrenamiento). Todas ellas son útiles en aplicaciones de reconocimiento o clasificación de patrones.

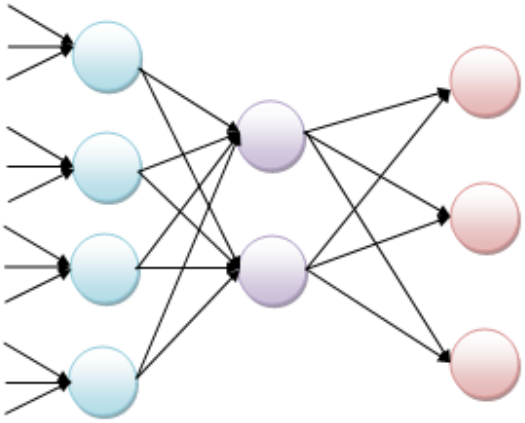


**Figura 1.2 Ejemplo de Red Multicapa con conexiones hacia adelante (4 neuronas en la capa de entrada, 2 en la capa oculta y 3 en la capa de salida).**

### 1.4.4 Redes con conexiones hacia adelante y hacia atrás (feedforward/feedback)

En este tipo de redes circula información tanto hacia delante como hacia atrás durante el funcionamiento de la red. Para que eso sea posible existen conexiones feedforward y feedback entre las neuronas.
En general, suelen ser bicapas, existiendo por lo tanto dos conjuntos de pesos: los correspondientes a las conexiones feedforward de la primera capa (capa de entrada) hacia la segunda (capa de salida) y los de las conexiones feedback de la segunda a la primera. Los valores de los pesos de estos tipos de conexiones no tienen por qué coincidir, siendo diferentes en la mayor parte de los casos.
Este tipo de estructura (bicapa) es particularmente adecuada para realizar una asociación de una información o patrón de entrada (en la primera capa) con otra información o patrón de salida en la segunda capa (lo cual se conoce como heteroasociación), aunque también pueden ser utilizadas para la clasificación de patrones.
Algunas redes tienen un funcionamiento basado en lo que se denomina resonancia, de tal forma que las informaciones en la primera y segunda capa interactúen entre sí hasta que alcanzan un estado estable. Esto permite un mejor acceso a las informaciones almacenadas en la red.
Los dos modelos de red de dos capas más conocidos son la red ART (Adaptative Resonante Theory) y la red BAM (Bidirectional Associative Memory).
También en este grupo de redes existen algunas conexiones laterales entre neuronas de la misma capa. Estas conexiones se diseñan como conexiones excitadoras (con peso positivo) o inhibidoras (con peso negativo), estableciéndose una competición entre las neuronas correspondientes.

## 1.5 Clasificación de las redes neuronales respecto al aprendizaje.

El aprendizaje es el proceso por el cual una red neuronal modifica sus pesos en respuesta a una información de entrada, este proceso se va realizando mientras el error no sea lo suficientemente pequeño (que tienda a cero).Cuando esto se logra, se dice que el proceso de aprendizaje ha terminado (la red ha aprendido).
Un aspecto importante respecto al aprendizaje es conocer cómo se modifican los valores de los pesos; cuáles son los criterios para cambiar el valor asignado a las conexiones cuando se pretende que la red aprenda una nueva información.

Estos criterios determinan la regla de aprendizaje. Se suelen considerar dos tipos de reglas, las que responden a lo que se conoce como aprendizaje supervisado y aprendizaje no supervisado. Una de las clasificaciones de redes neuronales obedece al tipo de aprendizaje utilizado. Así se pueden distinguir:

- Neuronas con aprendizaje supervisado
- Neuronas con aprendizaje no supervisado

La diferencia fundamental entre ambos tipos es la existencia o no de un agente externo (supervisor) que controle el proceso de aprendizaje.
Otro criterio para diferenciar las reglas de aprendizaje se basan en considerar si la red puede aprender durante su funcionamiento habitual (aprendizaje ONLINE), o si el aprendizaje supone una desconexión de la red; es decir su inhabilitación hasta que el proceso termine (aprendizaje OFFLINE).
Cuando el aprendizaje es OFFLINE se distingue entre una fase de aprendizaje o entrenamiento y una fase de operación o funcionamiento, existiendo un conjunto de datos de entrenamiento y un conjunto de datos de prueba que serán utilizados en la correspondiente fase. En las redes con aprendizaje OFFLINE, los pesos de las conexiones permanecen fijos después que termina el entrenamiento. Debido a su carácter estático, estos sistemas no presentan problemas de estabilidad en su funcionamiento (Simon, 1994).
En las redes con aprendizaje ONLINE no se distingue entre fase de entrenamiento y operación. Los pesos varían siempre que se presenta una nueva información al sistema. Debido a su carácter dinámico, el estudio de la estabilidad es un aspecto fundamental de este tipo de aprendizaje.

### 1.5.1 Aprendizaje supervisado.

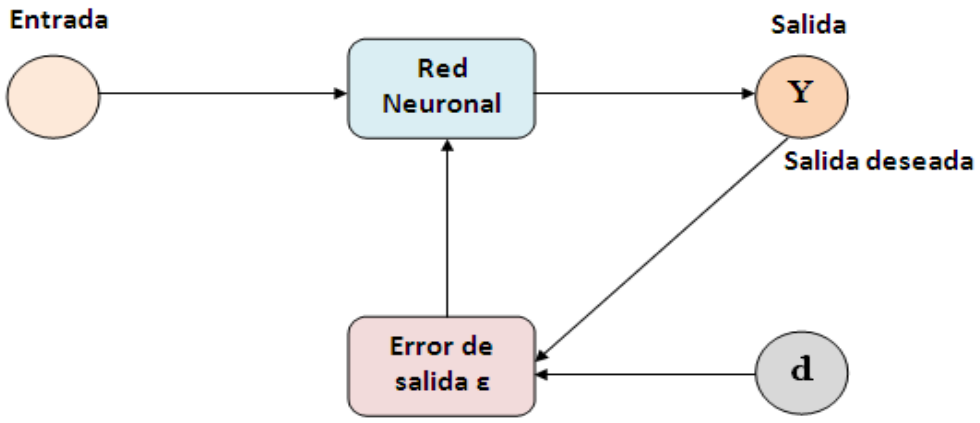



**Figura 1.3 Representación gráfica del Aprendizaje Supervisado.**

El proceso de aprendizaje se realiza mediante un entrenamiento controlado por un agente externo (supervisor o maestro) que determina la respuesta que debería generar la red a partir de una entrada determinada. El supervisor comprueba la salida de la red y en caso que esta no coincida con la deseada, se procederá a modificar los pesos de las conexiones, con el fin de que la salida obtenida se aproxime a la deseada.

Se suelen considerar tres formas de llevar a cabo el aprendizaje:

- Aprendizaje por corrección de error.
- Aprendizaje por refuerzo.
- Aprendizaje estocástico.

### 1.5.2 Aprendizaje no supervisado

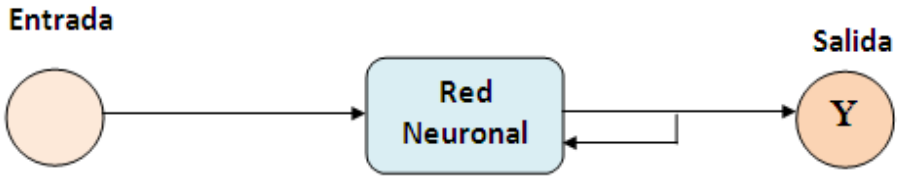



**Figura 1.4 Representación gráfica del Aprendizaje No Supervisado.**

El aprendizaje no supervisado es un método de aprendizaje automático donde un modelo es ajustado a las observaciones. Se distingue del aprendizaje supervisado por el hecho de que no hay un conocimiento a priori. En el aprendizaje no supervisado, un conjunto de datos de objetos de entrada es tratado. Así, el aprendizaje no supervisado típicamente trata los objetos de entrada como un conjunto de variables aleatorias, siendo construido un modelo de densidad para el conjunto de datos.
Como la red no recibe ninguna información por parte del entorno que le indique si la salida generada en respuesta a una determinada entrada es o no correcta, suele decirse que estas redes son capaces de autoorganizarse.
El funcionamiento se basa en la búsqueda de:

- Características
- Regularidades
- Correlaciones
- Categorías

Todo ello del conjunto de datos de entrada.

- Interpretación de las salidas: Existen diferentes interpretaciones que se le puede dar a la salida generada por una red utilizando este tipo de aprendizaje y que dependen de la estructura y el algoritmo.
- Grado de Familiaridad o Similitud: existente entre la información actual y la información pasada.
- Clusterización: es el establecimiento de categorías o clases. La red se encarga de encontrar las características o propiedades propias de cada clase.
- Codificación: versión codificada de la entrada.
- Mapeo de Características (Feature Mapping): Las entradas parecidas de la capa de salida se disponen geométricamente, representando un mapa topográfico de las características de los datos de entrada.

### 1.5.3 Aprendizaje híbrido.

El aprendizaje híbrido se caracteriza por combinar los dos tipos de aprendizajes vistos: aprendizaje supervisado y no supervisado. Las redes neuronales pueden utilizar el aprendizaje híbrido para disminuir el tiempo computacional que se puede alcanzar con uno solo de los existente (Simon,1994).

Un ejemplo de red neuronal que utiliza este tipo de aprendizaje es la Función de Base Radial (RBF), la cual, utiliza aprendizaje no supervisado para establecer los pesos óptimos entre las conexiones de las capas de entrada y oculta, y aprendizaje supervisado para establecer los pesos óptimos para las conexiones entre las capas oculta y de salida.

### 1.5.4 Aprendizaje por corrección de error.

Consiste en ajustar los pesos de las conexiones de la red en función de la diferencia entre los valores deseados y los obtenidos en la salida de la red; es decir, en función del error cometido en la salida.

Una regla o algoritmo simple podría ser el siguiente:

$\Delta w_{ji} = \alpha y_i (d_j - y_j)$.

Siendo:

$\Delta w_{ji}$: Variación del peso de la conexión ente las neuronas i y j ($\Delta w_{ji} = w_{ji}^{actual} - w_{ji}^{anterior}$).

$y_i$: Valor de la salida de la neurona i.

$d_j$: Valor de salida deseado para la neurona j.

$y_j$: Valor de salida obtenido para la neurona j.

$\alpha$: Factor de aprendizaje (0 < $\alpha \leq$ 1) que regula la velocidad del aprendizaje.

Un algoritmo muy conocido que permite un aprendizaje rápido es denominado regla delta o regla del mínimo error cuadrado (LMS Error: Least-Mean-Squared Error), que se aplicó en las redes conocidas como ADALINE y MADALINE).
En él se definió una función que permitiría cuantificar el error global cometido en cualquier momento durante el proceso de entrenamiento de la red, lo cual es importante, ya que cuanta más información se tenga sobre el error cometido, más rápido se puede aprender.

Este error medio se expresa de la siguiente forma:

$$Error_{global} = \frac{1}{2P} \sum_{k=1}^{P} \sum_{j=1}^{N} (y_j^{(k)} d_j^{(k)})^2$$

Siendo:

N: Número de neuronas de salida (en el caso de ADALINE N=1).

P: Número de informaciones que debe aprender la red.

$\frac{1}{2} \sum_{j=1}^{N} (y_j^k - d_j^k)^2$:

Error cometido en el aprendizaje de la información k-ésima.
Se trata de encontrar pesos para las conexiones que minimicen esta función de error. Para ello, el ajuste de los pesos de las conexiones de la red se puede hacer de forma proporcional a la variación relativa del error que se obtiene al variar el peso correspondiente:

$$\Delta w_{ji} = k \frac{\partial Error_{global}}{\partial w_{ji}}$$

Mediante este procedimiento, se llegan a obtener un conjunto de pesos con los que se consigue minimizar el error medio.
Otro algoritmo de aprendizaje por corrección de error lo constituye la regla delta generalizada o algoritmo de retropropagación del error (error Backpropagation). Se trata de una generalización de la regla delta para poder aplicarla a redes de conexiones hacia adelante (feedforward) con capas o niveles internos ocultos de neuronas que no tienen relación con el exterior.
Estas redes multicapa pueden utilizarse en muchas aplicaciones, pero su proceso de aprendizaje es mucho más lento, debido a que durante el mismo se debe explorar el espacio de posibles formas de utilización de neuronas de las capas ocultas; es decir, establecer cuál es su papel en la red.

### 1.5.5 Aprendizaje por refuerzo.

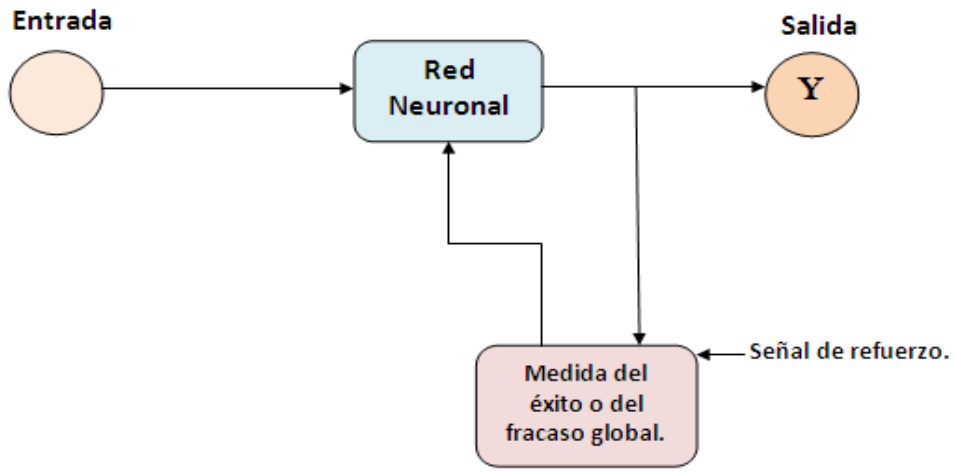



**Figura 1.5 Representación gráfica del Aprendizaje Reforzado.**

Es un aprendizaje más lento que el anterior, que se basa en la idea de no disponer de un ejemplo completo del comportamiento deseado; es decir, de no indicar durante el entrenamiento exactamente la salida que se desea que proporcione la red ante una determinada entrada.
En el aprendizaje por refuerzo la función del supervisor se reduce a indicar mediante una señal de refuerzo si la salida obtenida en la red se ajusta a la deseada (éxito = +1 o fracaso = -1), y en función de ello se ajustan los pesos basándose en un mecanismo de probabilidades.

Un ejemplo de algoritmo es el Linear Reward-Penality o $L_{R-P}$ (algoritmo lineal con recompensa y penalización) presentado por Narendra y Thathacher en 1974. Este algoritmo ha sido ampliado por Barto y Anandan, quienes en 1985 desarrollaron el denominado Associative Reward-Penality o $A_{R-P}$ (algoritmo asociativo con recompensa y penalización), que se aplica en redes con conexiones hacia adelante de dos capas cuyas neuronas de salida presentan una función de activación estocástica.
Otro algoritmo conocido es el Adaptive Heuristic Critic, introducido por Barto, Sutton y Anderson en 1983, que se utiliza en redes feedforward de tres capas especialmente diseñadas para que una parte de la red sea capaz de generar un valor interno de refuerzo que es aplicado a las neuronas de salida de la red.

**Aprendizaje hebbiano.**

Este tipo de aprendizaje se basa en el postulado formulado por Donald O. Hebb en 1949: "Cuando un axón de una celda A está suficientemente cerca como para conseguir excitar a una celda B y repetida o persistemente toma parte en su activación, algún proceso de crecimiento o cambio metabólico tiene lugares en una o ambas celdas, de tal forma que la eficiencia de A, cuando la celda a activar es B, aumenta. Por celdas, Hebb entiende un conjunto de neuronas fuertemente conectadas a través de una estructura compleja. La eficiencia podría identificarse por la intensidad o magnitud de la conexión, es decir, con el peso.

Se puede decir que el aprendizaje consiste en el ajuste de los pesos de las conexiones de acuerdo con la correlación (multiplicación en el caso de valores binarios +1 y –1) de los valores de activación (salidas) de las neuronas conectadas.

$$\Delta w_{ij} = y_i * y_j$$

Esta expresión responde a la idea de Hebb, puesto que si las dos unidades son activas (positivas), se refuerza la conexión; por el contrario, cuando una es activa y la otra pasiva, se debilita la conexión.
Existen muchas variaciones de dicho aprendizaje, por ejemplo, Sejnowski en 1977 utilizó la correlación de covarianza de los valores de activación de las neuronas. Sutton y Barto en 1981 utilizaron la correlación del valor medio de una neurona con la varianza de la otra. Klopf en 1986 propuso una correlación entre las variaciones de los valores de activación en dos instantes de tiempo sucesivos, aprendizaje que denominó drive-reinforcement y que utilizó en redes del mismo nombre con topología feedforward de dos capas.
Otra versión en este aprendizaje es el hebbiano diferencial, que utiliza la correlación de las derivadas en el tiempo de las funciones de activación de las neuronas.

### 1.5.6 Aprendizaje estocástico.

Este tipo de aprendizaje consiste básicamente en realizar cambios aleatorios en los valores de los pesos de las conexiones de la red y evaluar su efecto a partir del objetivo deseado y de distribuciones de probabilidad.
En el aprendizaje estocástico se suele hacer una analogía en términos termodinámicos, asociando la red neuronal con un sólido físico que tiene cierto estado energético. En el caso de la red, la energía de la misma representaría el grado de estabilidad de la red, de tal forma que el estado de mínima energía correspondería a una situación en la que los pesos de las conexiones consiguen que su funcionamiento sea el que más se ajusta al objetivo deseado.
Según lo anterior, el aprendizaje consistiría en realizar un cambio aleatorio de los valores de los pesos y determinar la energía de la red. Si la energía es menor después del cambio; es decir, si el comportamiento de la red se acerca al deseado, se acepta el cambio; de lo contrario, se aceptaría el cambio en función de una determinada y preestablecida distribución de probabilidades.

**Aprendizaje de Boltzmann.**

Las máquinas de Boltzmann son redes recurrentes simétricas que consisten en unidades binarias (+1 para “on” y -1 para “off”). Por simetría los pesos en la conexión entre la unidad i y la unidad j son iguales al peso en la conexión entre la unidad j y la unidad i (wij = wji).
Un subconjunto de neuronas visibles interactúa con el entorno, el resto, las ocultas, no lo hacen. Cada neurona es una unidad estocástica que genera un resultado (o estado) de acuerdo con la distribución de Boltzmann de la mecánica estadística.
Una maquina de Boltzmann opera de dos modos:

- **Restringido**: en donde las neuronas visibles están restringidas a estados específicos determinados por el entorno.
- **Libres**: Todas las neuronas (visibles y ocultas) operan libremente.

El objetivo del aprendizaje de Boltzmann es ajustar los pesos de las conexiones de tal forma que las unidades visibles satisfagan una distribución de probabilidad particular deseada.
De acuerdo con esto, el cambio en los pesos es dado por:

$$\Delta w'_{ji} = \eta (\overline{\rho}_{ij} - \rho_{ij})$$

Donde n es la ratio de aprendizaje , $\overline{\rho}_{ij}$ y $\rho_{ij}$ son las correlaciones entre los estados de la unidad i y j. Cuando la red opera en modo restringido o en modo libre respectivamente. Los valores de p se estiman habitualmente mediante el método de Monte Carlos, que es muy lento.

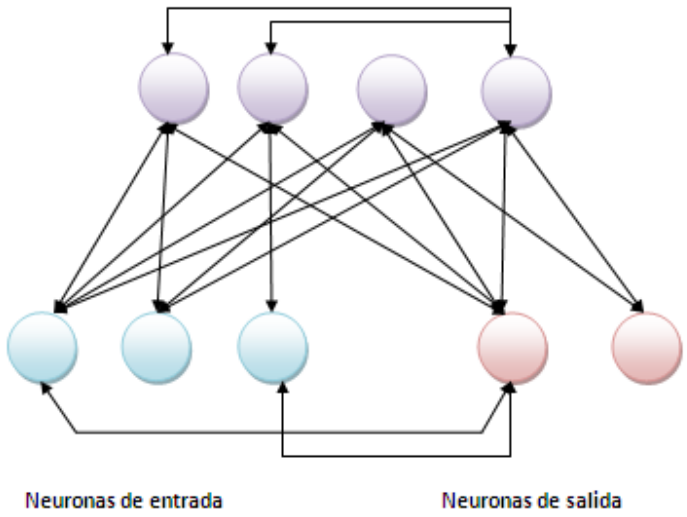



**Figura 1.6 Representación gráfica del Aprendizaje de Boltzmann.**

**Arquitectura de la máquina de Boltzmann.**

La máquina de Boltzmann constituye una estructura recurrente con salidas binarias, se caracteriza por una función de energía E, cuyo valor está definido por los estados de cada neurona.
En la máquina de Boltzmann ninguna de las neuronas tiene auto feedback, esta escoge aleatoriamanete una neurona para aplicar el aprendizaje y conmutar de estado a una temperatura T, la cual no es una temperatura física, sino que corresponde a un símil del temple.
Aplicando sucesivamente esta regla, la máquina de Boltzmann alcanzará el equilibrio térmico.
El Aprendizaje de Boltzmann es una regla de aprendizaje estocástico obtenido a partir de principios teóricos de información y de termodinámica. El objetivo del aprendizaje de Boltzmann es ajustar los pesos de las conexiones de tal forma que el estado de las unidades visibles satisfaga una distribución de probabilidades deseada en particular.

### 1.5.7 Aprendizaje competitivo y cooperativo.

En dicho aprendizaje suele decirse que las neuronas compiten (y cooperan) unas con otras con el fin de llevar a cabo una tarea dada.
La competición entre neuronas se realiza en todas las capas de la red, existiendo en estas neuronas conexiones recurrentes de autoexcitación y conexiones de inhibición por parte de neuronas vecinas. Si el aprendizaje es cooperativo, estas conexiones con las vecinas serán de excitación.
El objetivo de este aprendizaje es clusterizar los datos que se introducen en la red. De esta forma, las informaciones similares son clasificadas formando parte de la misma categoría, y por tanto deben activar la misma neurona de salida. Las categorías deben ser creadas por la misma red, puesto que se trata de aprendizaje no supervisado, a través de las correlaciones entre los datos.
En este tipo de redes, cada neurona tiene asignado un peso total, suma de todos los pesos de las conexiones que tiene a su entrada. El aprendizaje afecta sólo a las neuronas ganadoras (activas), redistribuyendo este peso total entre todas las conexiones que llegan a la neurona vencedora y repartiendo esta cantidad por igual entre todas las conexiones procedentes de unidades activas. Por tanto, la variación del peso de una conexión entre una unidad i y otra j será nula si la neurona j no recibe excitación por parte de la neurona i y otra j será nula si la neurona j no recibe excitación por parte de la neurona i, y se modificará si es excitada por dicha neurona i.
Existe otro caso particular de aprendizaje competitivo, denominado teoría de la resonancia adaptativa, desarrollado por Carpenter y Grossberg en 1986 y utilizado en la red feedforward /feedback de dos capas conocida como ART. Esta red realiza un prototipado de las informaciones que recibe a la entrada, generando como salida un ejemplar o prototipo que representa a todas las informaciones que podrían considerarse pertenecientes a la misma categoría.

En la tabla 1.2 se muestran los tres tipos de aprendizajes, con las diferentes reglas de aprendizaje que se pueden utilizar para simular cada uno, las arquitecturas de redes neuronales que los implementan, los algoritmos de aprendizaje para cada arquitectura, y algunas de las tareas que pueden realizar estos tipos de redes neuronales.

| Paradigma | Regla de Aprendizaje | Arquitectura | Algoritmo de Aprendizaje | Tareas |
|---|---|---|---|---|
| **Supervisado** | Corrección del error. | Perceptrón o perceptrón multicapa. | Algoritmos de aprendizaje del perceptrón, retropropagación del error, ADALINE, MADALINE. | Clasificación de patrones, aproximación de funciones, predicción, control. |
| | Boltzmann. | Recurrente. | Algoritmo de aprendizaje Boltzmann. | Clasificación de patrones. |
| | Hebbian. | Feedforward Multicapa. | Análisis de discriminante lineal. | Análisis de datos, clasificación de patrones. |
| | Competitivo. | Competitivo. | Aprendizaje por Vector de | Compresión de datos, |

| | | | | |
|---|---|---|---|---|
| | | | cuantización (LVQ). | Clasificación de clases interiores. |
| | | Red ART | ART Map | Clasificación de patrones, Clasificación de clases interiores. |
| **No supervisado.** | Corrección del error. | Feedforward Multicapa. | Proyección de Sampson. | Análisis de Datos. |
| | Hebbian. | Feedforward o Competitivo. | Análisis principal de componentes. | Análisis de Datos, Compresión de datos. |
| | | Red de Hopfield. | Aprendizaje de memoria asociativa. | Memoria asociativa. |
| | Competitivo. | Competitivo | Vector de cuantización | Categorización, compresión de datos. |
| | | Mapas Autoorganizados de Kohonen | Mapas Autoorganizados de Kohonen | Categorización y análisis de datos. |
| | | Redes ART | ART1, ART2 | Categorización. |
| **Híbrido** | Corrección del error y competitivo. | Redes RBF. | Algoritmo de aprendizaje RBF. | Clasificación de patrones, aproximación de funciones, predicción y control. |

**Tabla 1.2 Tipos de aprendizaje.**

### 1.6 Tipologías de Redes Neuronales

Una Red Neuronal Artificial (RNA) puede considerarse como un grafo dirigido ponderado en el que neuronas artificiales son nodos y hojas dirigidas y ponderadas con conexiones entre salidas y entradas de neuronas.
Dependiendo del patrón de conexión pueden dividirse en:

- **Redes de propagación hacia delante (feedforward)**: son aquellas en las que los grafos no tienen bucles.
- **Recurrentes o de retroalimentación (feedback)** en las cuales los bucles ocurren debido a conexiones de retroalimentación.

La figura 1.7 muestra ejemplos de redes neuronales para cada uno de los patrones de conexiones vistos.

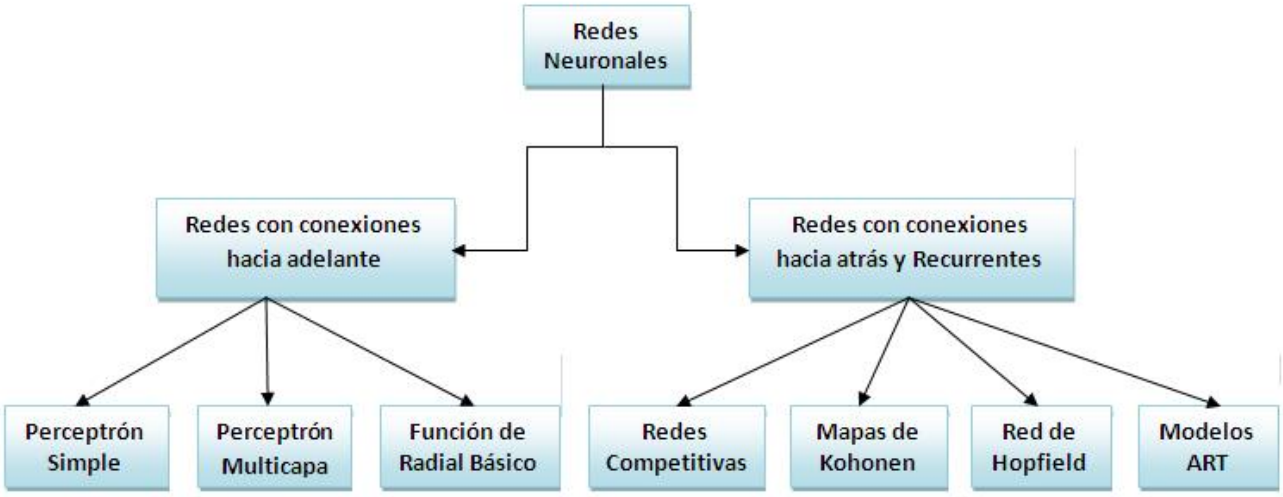



**Figura 1.7 Taxonomía de las Redes con conexiones hacia adelante, con conexiones hacia atrás y Recurrentes.**

**Redes de propagación hacia adelante.**

- **Estáticas:** producen sólo un conjunto de valores como resultado, para una secuencia de valores de una entrada.
- **Sin memoria:** en el sentido en que su respuesta es una entrada independiente del estado previo de la red.
-

**Perceptrón:**

Un perceptrón es una unidad básica de inferencia en forma de discriminador lineal, que suele formar parte de una red neuronal artificial.

Un perceptrón puede clasificar datos que sean linealmente separables. En el caso de un perceptrón con dos entradas deberá poder trazarse una única línea que separe las dos clases que permite identificar el perceptrón (Bollella, 2003).

Las funciones AND y OR son linealmente separables y por lo tanto pueden ser aprendidas por un perceptrón. Ver Figura 1.8

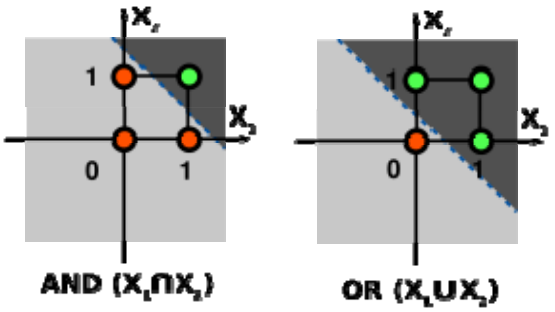



**Figura 1.8 Funciones AND y OR.**

La función XOR no puede ser aprendida por un único perceptrón puesto que requiere al menos de dos líneas para separar las clases (0 y 1). Debe utilizarse al menos una capa adicional de perceptrones para permitir su aprendizaje. Ver figura 1.9. Sólo permite entradas binarias o bipolares, y sus salidas tendrán dos estados (0 ó 1).

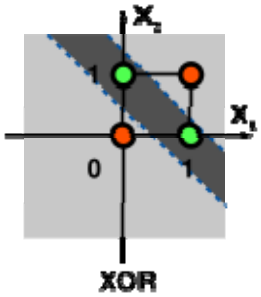



**Figura 1.9 Función XOR.**

### 1.6.1 Perceptrón Monocapa.

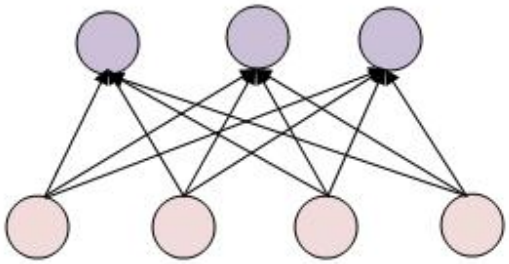


**Figura 1.10 Ejemplo de perceptrón monocapa. Los círculos en rosado representan las entradas que se van a propagar sobre las neuronas del perceptrón (círculos en violeta).**

El perceptrón monocapa contiene una sola capa de neuronas, donde los vectores de entrada propagan cada una de sus componentes a través de todas las neuronas de la única capa del perceptrón. Las salidas de la red estarán en correspondencia con el tipo de función de transferencia (o activación) utilizada (Ver epígrafe 1.2). Si la función de transferencia es lineal, entonces la salida del perceptrón monocapa será igual al vector de entrada que encuestó al perceptrón monocapa.
Un perceptrón monocapa, cuyos nodos sean elementos lineales y con pesos escogidos pudiera ejecutar exactamente las funciones de un Perceptrón Multicapa (Ver epígrafe 1.6.2) con características similares de linealidad.

### 1.6.2 El Perceptrón Multicapa.

El Perceptrón Multicapa (PMC) es una red de alimentación hacia adelante con una o más capas de nodos entre las capas de entrada y salida. Ver figura 1.11.

Las capas pueden clasificarse en tres tipos:

- **Capa de entrada**: Constituida por aquellas neuronas que introducen los patrones de entrada en la red. En estas neuronas no se produce procesamiento.
- **Capas ocultas**: Formada por aquellas neuronas cuyas entradas provienen de capas anteriores y las salidas pasan a neuronas de capas posteriores.
- **Capa de salida**: Neuronas cuyos valores de salida se corresponden con las salidas de toda la red.

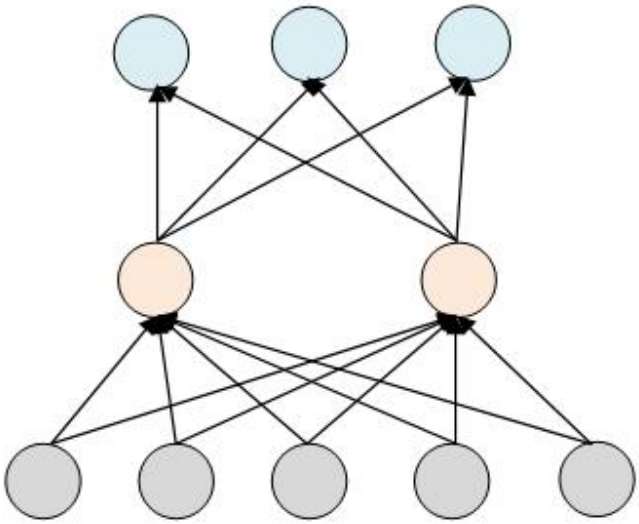


**Figura 1.11 Modelo de PMC de tipo feed-forward (5 neuronas en la capa de entrada, una capa oculta con 2 neuronas y en la capa de salida 3 neuronas).**

El PMC puede ser local o totalmente conectado. En el primer caso cada salida de una neurona de la capa "i" es entrada de todas las neuronas de la capa "i+1", mientras que en el segundo, cada neurona de la capa "i" es entrada de una serie de neuronas (región) de la capa "i+1".
No más de tres capas son requeridas en un PMC, ya que una red de tres capas puede generar arbitrariamente regiones de decisiones de cualquier complejidad. El número de nodos en la capa oculta debe ser mayor que uno cuando las regiones de decisión están desconectadas o enredadas y no pueden formar un área convexa (Bollella, 2003).

El **número de nodos de la capa oculta requiere**, en el peor caso, ser igual al número de las regiones desconectadas en las entradas distribuidas.
El **número de nodos en la capa de entrada** debe típicamente ser suficiente para proveer a la red de los valores que va a leer desde el entorno.
El **número de neuronas de la capa de salida** estará en correspondencia con la cantidad de respuestas que podría brindar la red para las entradas especificadas.
El hecho de que el PMC sea una RNA formada por múltiples capas, le permite resolver problemas que no son linealmente separables, lo cual es la principal limitación del perceptrón monocapa.
Generalmente, el método de aprendizaje utilizado por los PMC es el supervisado, dónde se pueden utilizar algunos de los procedimientos vistos en el epígrafe 1.5. Un PMC con solamente una capa oculta y que emplee el método de Backpropagation para realizar el aprendizaje supervisado, se les conoce como redes del tipo Backpropagation (Ver epígrafe 1.6.4).

### 1.6.3 Red de Hopfield.

Las redes de Hopfield son redes de adaptación probabilística, recurrentes, funcionalmente entrarían en la categoría de las memorias autoasociativas, es decir, que aprenden a reconstruir los patrones de entrada que memorizaron durante el entrenamiento. Son arquitecturas de una capa con interconexión total, funciones de activación booleana de umbral (cada unidad puede tomar dos estados, 0 ó 1, dependiendo de si la estimulación total recibida supera determinado umbral), adaptación probabilística de la activación de las unidades, conexiones recurrentes y simétricas, y regla de aprendizaje no supervisado.

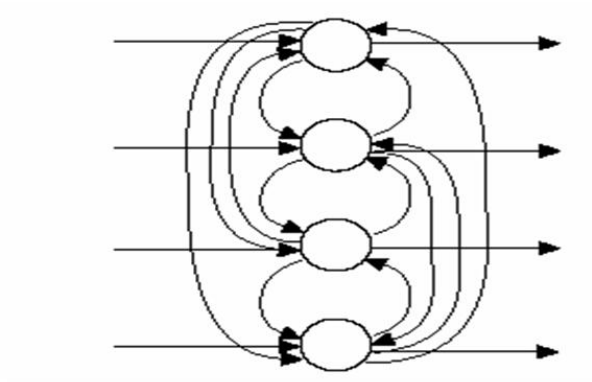


**Figura 1.12 Red de Hopfield. Modelo compuesto por una sola capa donde cada neurona se conecta con las restantes.**

La red de Hopfield (Ver Figura 1.12) consiste en un conjunto de N elementos de procesado interconectadas que actualizan sus valores de activación de forma asíncrona e independiente del resto de los elementos de procesado. Todos los elementos son a la vez de entrada y salida. Los valores de activación son binarios.
El estado del sistema está dado por los valores de activación Yk. La entrada de la neurona k en el ciclo temporal t+1 viene dada por

$$s_k(t+1) = \sum_{j \neq k} y_j(t) w_{jk} + \theta_k$$

Para obtener el nuevo valor de activación se aplica una función umbral (Bollella, 2003).
Cuando un elemento de procesado mantiene su valor de activación se dice que es estable. Se llama estado estable a aquel en el cual todos los elementos de procesado son estables.
Con la restricción extra de simetría en los pesos de conexión, Wjk=Wkj, el sistema puede ser descrito mediante una función energía de la forma:

$$E = -\frac{1}{2}\sum_{j \neq k}\sum_{k} y_j y_k w_{jk} - \sum_{k} \theta_k y_k$$

### 1.6.4 La Red Backpropagation.

En 1986, Rumelhart, Hinton y Williams, formalizaron un método para que una red neuronal aprendiera la asociación que existe entre los patrones de entrada y las clases correspondientes, utilizando varios niveles de neuronas.
El método Backpropagation (propagación del error hacia atrás), basado en la generalización de la regla delta, a pesar de sus limitaciones, ha ampliado de forma considerable el rango de aplicaciones de las redes neuronales.
El funcionamiento de la red Backpropagation (BPN) consiste en el aprendizaje de un conjunto predefinido de pares de entradas-salidas dados como ejemplo: primero se aplica un patrón de entrada como estímulo para la primera capa de las neuronas de la red, se va propagando a través de todas las capas superiores hasta generar una salida, se compara el resultado en las neuronas de salida con la salida que se desea obtener y se calcula un valor de error para cada neurona de salida. A continuación, estos errores se transmiten hacia atrás, partiendo de la capa de salida hacia todas las neuronas de la capa intermedia que contribuyan directamente a la salida, recibiendo el error aproximado a la neurona intermedia a la salida original. Este proceso se repite, capa por capa, hasta que todas las neuronas de la red hayan recibido un error que describa su aporte relativo al error total. Basándose en el valor del error recibido, se reajustan los pesos de conexión de cada neurona, de manera que en la siguiente vez que se presente el mismo patrón, la salida obtenida esté cercana a la deseada (Bollella, 2003).
La importancia de la red Backpropagation consiste en su capacidad de autoadaptar los pesos de las neuronas de las capas intermedias para aprender la relación que existe entre un conjunto de patrones de entrada y sus salidas correspondientes. Es importante la capacidad de generalización, facilidad de dar salidas satisfactorias a entradas que el sistema no ha visto nunca en su fase de entrenamiento. La red debe encontrar una representación interna que le permita generar las salidas deseadas cuando se le dan entradas de entrenamiento, y que pueda aplicar, además, a entradas no presentadas durante la etapa de aprendizaje para clasificarlas.

### 1.6.5 Modelos de la Teoría de la Resonancia Adaptativa (Modelos ART).

Sobre el dilema estabilidad-plasticidad: ¿Cómo podemos aprender nuevas cosas (plasticidad) asegurándonos la estabilidad de conservar el conocimiento existente?
Los modelos ART (Carpenter y Grossberg ART1, ART2, ARTMap) intentan solucionar este problema.
La red tiene un suplemento suficiente de unidades de salida, pero no se usarán hasta que resulte necesario. Una unidad se dice que está comprendida (no-comprometida) si está (no está) siendo usada.
El algoritmo de aprendizaje modifica los prototipos almacenados de una categoría sólo si el vector de entrada es lo suficientemente similar a ellos (si son resonantes).
La extensión de similaridad es controlada por un parámetro de vigilancia, p, con $0<p<1$, que también determina el número de categorías.
Cuando el vector de entrada no es lo suficientemente similar a algún prototipo existente, se crea una nueva categoría y se asigna a una unidad no-comprometida como prototipo inicial. Si no hay una unidad tal que la entrada nueva no produce respuesta en la red.

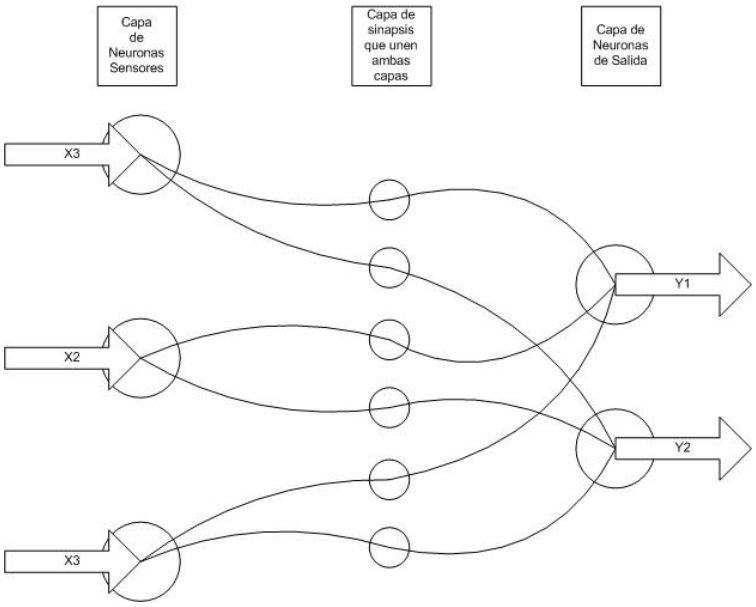



**Figura 1.13 Modelo ART.**

Los modelos ART (Ver Figura 1.13) se caracterizan por presentar aprendizaje competitivo, entrenamiento no supervisado, y se aplican a problemas de clasificación. Los modelos son redes formadas por tres capas. En la capa de entrada no se realiza ningún preprocesamiento de los datos de entrada, poseen una capa oculta y otra de salida, la cual está compuesta por neuronas competitivas.
La capa de entrada y la capa oculta siempre tienen el mismo número de neuronas. Cada neurona de la capa de entrada es entrada de una única neurona de la capa oculta, por esta razón también se habla de las redes ART como redes de dos capas, y no de tres, aunque ambos puntos de vista son correctos.

### 1.6.6 Redes Neuronales Autoorganizados: Mapas de Kohonen.

Los Mapas Auto-organizados (SOM) preservan la topología. En proyecciones que preservan la topología de patrones de entrada cercanos se activarán unidades de salida cercanas en el mapa.

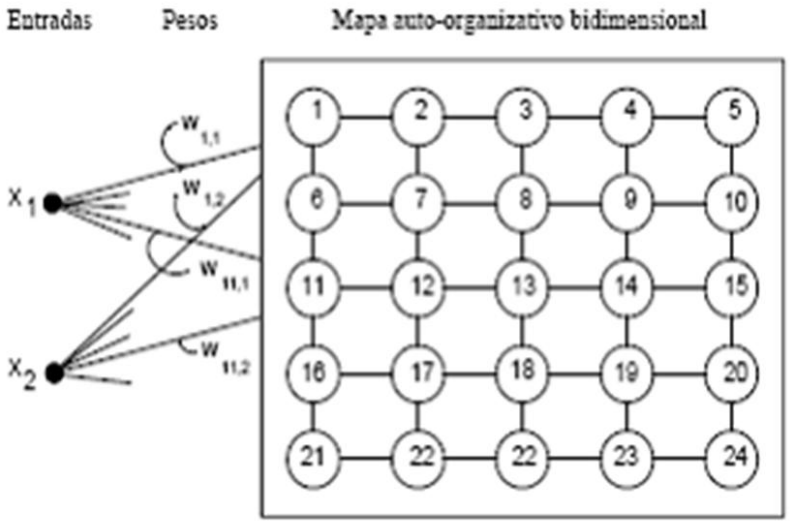



**Figura 1.14 Mapa de Kohonen.**

Un SOM de Kohonen (Ver Figura 1.14) consiste fundamentalmente en un arreglo bidimensional de unidades, cada una conectada a todos los n-nodos de entrada.
Si Wij es el vector n-dimensional asociado con la unidad localizada en (i, j) del arreglo 2D, cada neurona computa la distancia euclidiana entre el vector de entrada X y el peso almacenado en el vector Wij.
Este SOM es un tipo especial de red de aprendizaje competitivo, que define una vecindad espacial para cada unidad de salida. La forma de la vecindad local puede ser cuadrada, rectangular o circular. El tamaño de la red inicial suele fijarse a ½ o a 2/3 del tamaño total y reducirla posteriormente de acuerdo con un plan (por ejemplo, una función exponencial decreciente). Utilizan aprendizaje no supervisado para el entrenamiento.

### 1.6.7 Función de Base Radial.

Las Funciones de Base Radial (RBF) son una clase especial de red multicapa hacia adelante que tiene tres capas (Ver Figura 1.15). La capa de entrada realiza un procesamiento lineal. Cada unidad de la capa oculta emplea una función de base radial, tal como un kernel gaussiana, como función de activación. La función de Base Radial o función núcleo se centra en el punto especificado por el vector de peso asociado con la unidad.

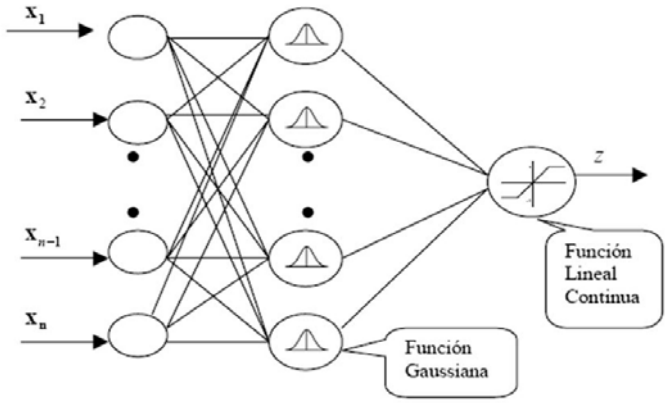



**Figura 1.15 Función de Base Radial.**

Tanto las posiciones como las anchuras de estos núcleos deben aprenderse por patrones de entrenamiento. Cada unidad resultante implementa una combinación lineal de estas funciones de base radial (Brío y Molina, 2003).
Las RBF utilizan aprendizaje híbrido en su entrenamiento. El aprendizaje es no supervisado para las neuronas de la capa oculta, y supervisado para las neuronas de la capa de salida.

### 1.6.8 Redes Recurrentes.

Son redes formadas por muchas neuronas fuertemente interconectadas y que interaccionan muchas veces cuando la red es activada.
Redes recurrentes son aquellas que poseen conexiones de realimentación, como es visto en la figura 1.16, las cuales proporcionan un comportamiento dinámico. El modelo de Hopfield es un ejemplo de red neuronal recurrente.

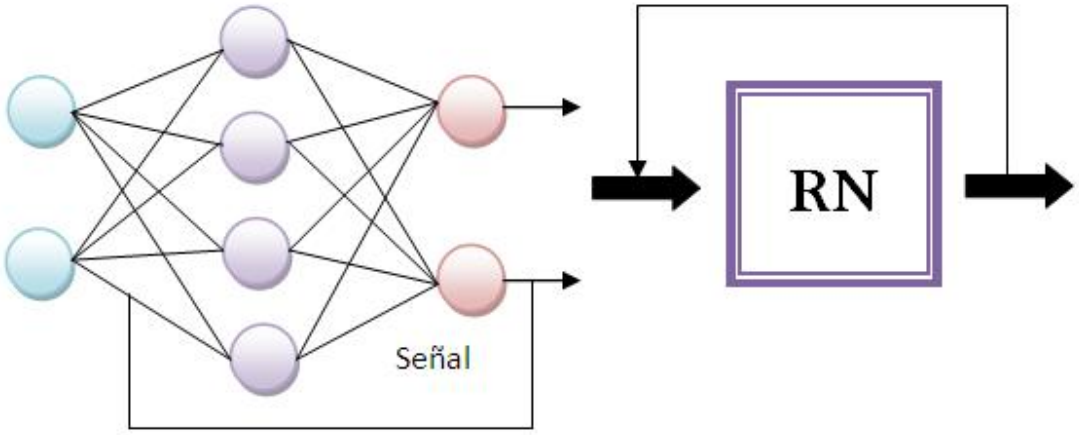



**Figura 1.16 Modelo de red recurrente: la señal de salida retroalimenta nuevamente a la red.**

En general los siguientes parámetros son importantes para definir la arquitectura de una red neuronal: número de capas, número de neuronas en cada capa y tipo de conexión entre dos neuronas, que definen la red de feedforward o Recurrentes.

### 1.6.9 Otros tipos de Redes Neuronales.

| | Modelos de Redes Neuronales | Siglas |
|---|---|---|
| **1.** | Adaline(Adaptative Linear Neural Element) | ADA |
| **2.** | Adaptative Resonante Theory Networks | ARTN |
| **3.** | Bidirectional Associative Memory | BAM |
| **4.** | Boltzmann Machina | BOLTMA |
| **5.** | Brain-State-in a Box Networks | BSBN |
| **6.** | Cauchy Machine | CAUMA |
| **7.** | Cerebellar Model Articulation Controller | CMAC |
| **8.** | Counter-Propagation Networks | CPN |
| **9.** | Delta Bar Delta Networks | DBDN |
| **10.** | Finite Impulse Response Multilayer Perceptron | FIR-MP |
| **11.** | Functional-link Networks (Plynomial Neural Networks) | FLN |

| | | |
|---|---|---|
| **12.** | Fuzzy ARTMAP Classification Networks | FARTMAP |
| **13.** | General Regression Neural Networks | GRNN |
| **14.** | Group Method of Data Handling(Polynomial Neural Networks) | GMDH |
| **15.** | Hamming Networks | HAMN |
| **16.** | Hierarchical Networks-Neocognitron | HNN |
| **17.** | Hopfield | HOPF |
| **18.** | Jordan's sequential networks | JORDAN |
| **19.** | Learning Vector Quantization | LVQ |
| **20.** | Logicon Projection Network | LPN |
| **21.** | Madaline(Multiple adalines) | MAD |
| **22.** | Modular Neural Network | MNN |
| **23.** | Multilayer Feedforward | MLFF |
| **24.** | Nonlinear Autoregressive Moving Average Network | NARMA |
| **25.** | Pipelined Recurrent Neural Networks | NARMA |
| **26.** | Probabilistic Neural Networks | PNN |
| **27.** | Radial Basis Function Networks | RBFN |
| **28.** | Real-Time Recurrent Networks | RTRN |
| **29.** | Recirculation Networks | RCN |
| **30.** | Self-Organizing feature Map | SOFM |
| **31.** | Sequential Cascaded Recurrent Networks | SCRN |
| **32.** | Sigma-Pi Network (Polynomial Neural Networks) | SPN |
| **33.** | Simple recurrent networks (Elman) | ELMAN |
| **34.** | Spation-Temporal Pattern Recognition | STPR |
| **35.** | Support Vector Machine | SVM |
| **36.** | Time-Delay Neural Networks | TDNN |
| **37.** | Tree Neural Networks | TNN |
| **38.** | Wavelet Neural Networks | WNN |

**Tabla 1.3 Tipos de redes neuronales.**

# Capítulo II Comparación de los modelos más representativos de las Redes Neuronales con el Perceptrón Multicapa.

## 2.1 Perceptrón Multicapa y la Realidad Virtual.

### 2.1.1 ¿Por qué las redes neuronales?

Los mundos virtuales tienen entre sus objetivos principales la recreación de ambientes similares a los presentes en la vida real. El usuario necesita interactuar con componentes capaces de ofrecerle un comportamiento inteligente dentro del entorno simulado. Estos componentes son llamados agentes autónomos.

Un agente autónomo es un sistema situado en un entorno, lo siente y actúa sobre él. Estos se diferencian de los programas por su autonomía. Son proactivos, es decir, que no solo actúan, sino que actúan siguiendo sus propios objetivos. Además son persistentes, o sea, que no se pueden apoyar, incluso, cuando el escenario no esté interactuando con ellos, los agentes siguen funcionando, recolectando información, aprendiendo y comunicándose con otros agentes.

¿Qué técnica de inteligencia artificial permite alcanzar tales resultados?

Con la que mejor se pueden alcanzar tales resultados son las Redes Neuronales Artificiales.

Las características de las redes neuronales que posibilitaron su selección fueron las siguientes: poseen auto-organización, por lo que ofrecen posibilidades de procesado robusto y además poseen adaptatividad, ya que utilizan aprendizaje adaptativo. El procesado no lineal aumenta la capacidad de la red de aproximar-clasificar y su inmunidad frente al ruido. En el procesado paralelo normalmente se usa un gran número de células de procesado con un alto nivel de ínter-conectividad.

Otros argumentos a tener en cuenta son la habilidad para generalizar información durante el aprendizaje, poseen información contextual en su interior e infiere soluciones directamente desde los datos.

Las principales ventajas de las redes neuronales sobre otras técnicas de inteligencia artificial son que simplifican el tamaño de los programas y poseen la capacidad de adaptarse y aprender dentro del entorno en que son utilizadas.

Las principales desventajas están referidas sobre la base de que la reacción y los resultados no son predecibles. Admiten y soportan el aprendizaje y la adaptación. Requieren de algoritmos más complejos y generalmente de recursos computacionales para trabajar. Y son mucho más difíciles de depurar.

Curiosamente, dos de sus desventajas constituyen a su vez fortalezas importantes de las redes neuronales. El hecho de que sus resultados sean impredecibles, a pesar de las molestias que pueda ocasionar en ocasiones por situaciones inesperadas, nos garantiza de antemano un comportamiento verdaderamente inteligente, ya que el usuario no siempre podrá adivinar la respuesta que debería dar la red.

Un agente inteligente que sea capaz de aprender y adaptarse a las nuevas condiciones creadas, permitirá que el usuario, a medida que aumente su nivel de desarrollo dentro del entorno con el que interactúa, encuentre una resistencia mucho más elevada, de acuerdo con las nuevas habilidades adquiridas, sino la interacción sería verdaderamente aburrida.

Muchos son los argumentos que se pueden esgrimir para tomar estas potencialidades de las redes neuronales a favor o en contra, pero la verdad es que, primero debe analizarse el sistema en el cual se va a desarrollar la red para saber hasta donde se puede explotar estas características como ventajas o desventajas.

## Capítulo II Comparación de los modelos más representativos de las Redes Neuronales con el Perceptrón Multicapa

### 2.1.2 El Perceptrón Multicapa (PMC) como aproximador universal de funciones.

El empleo de las redes neuronales en juegos, se puede considerar exitoso. Se han utilizado en distintas variantes, con diferentes tipos de aprendizaje y topologías, a pesar de sus inconvenientes. Dentro de los modelos el más utilizado en la solución de problemas ha sido el Perceptrón Multicapa (PMC).

El desarrollo del PMC durante los últimos treinta años ha resultado curioso. Partiendo de un perceptrón monocapa y observando sus limitaciones computacionales, se llegó a la arquitectura PMC, y aplicándolo a numerosos problemas, se comprobó experimentalmente que este era capaz de representar complejos mappings y de abordar problemas de clasificación de gran envergadura, de una manera eficaz y relativamente simple. Estos problemas de clasificación, se fueron escenificando en los juegos de realidad virtual por los desarrolladores, los resultados alcanzados en los estudios de los perceptrones, fueron abriéndole un campo de aplicación en la industria del entretenimiento y de la simulación.

Estudios realizados por grupos de expertos llegaron a la conclusión que un PMC de una única capa oculta puede aproximar hasta el nivel deseado cualquier función continua en un intervalo, por lo tanto, las redes neuronales multicapa unidireccionales son aproximadores universales de funciones. Esta propiedad, ha sido una de las causantes del éxito de los PMC. La mayoría de los escenarios de los juegos son representados por funciones que brindan salidas ante ciertas entradas que constituyen estados iniciales. La representación de los estados de un jugador como variables, facilita la representación del problema, aunque la solución es mucho mejor brindada por las redes cuyo aporte está en funcionar como una caja negra de soluciones difíciles de tracear.

Aunque estas funciones (PMC) para dar las respuestas lógicas sobre un dominio del problema necesitan de los algoritmos de entrenamiento que son los encargados de darle el conocimiento a la red.

### 2.1.3 Backpropagation (BP) como algoritmo de entrenamiento más conocido del PMC.

EL BP, constituye un método de aprendizaje supervisado de gran generalidad, lo que presenta ventajas e inconvenientes. Su ventaja principal es que se puede aplicar a multitud de problemas diferentes, proporcionando con frecuencia, buenas soluciones con no demasiado tiempo de desarrollo. Estos factores, lo han convertido en el método de aprendizaje supervisado más utilizado en los PMC y dentro de todos los tipos de aprendizaje.

Las empresas desarrolladoras de videojuegos (el sector más encumbrado dentro del mercado de la informática) poseen limitantes en cuanto al tiempo de elaboración de los productos. Lograr aplicaciones de alto vuelo con inteligencia artificial implícita y con protagonismo de los agentes autónomos presentes puede ser un poco más lento que con el empleo de las redes neuronales (cuyo código es mucho menor en tamaño que el de una máquina finita de estado). Al utilizar redes neuronales, el mayor tiempo se le dedica al diseño de la red para un problema dado, cuando este está resuelto, las líneas de códigos toman menor tamaño que otras técnicas de Inteligencia Artificial (IA) con la ventaja adicional, de su persistencia en el escenario de utilización.

No obstante, si se requiere una solución realmente excelente, habrá que dedicar más tiempo al desarrollo del sistema neuronal; teniendo en cuenta diferentes cuestiones adicionales que todavía no se han abordado.

Como desventaja se encuentra, entre otras, su lentitud de convergencia, uno de los precios que hay que pagar por disponer de un método general de ajuste funcional que no requiere (en principio) información a priori (Bello, 1993). Sin embargo, se debe tener en cuenta que el BP no requiere tanto esfuerzo computacional como el que sería necesario si se tratasen de obtener los pesos de la red mediante la evaluación directa de las derivadas (Bello, 1993).

Otro problema del BP es que puede incurrir en el denominado sobreaprendizaje (o sobreajuste), fenómeno directamente relacionado con la capacidad de generalización de la red. La memorización o generalización, es un inconveniente de la red, que limita una de sus potencialidades: inmunidad frente al

ruido, obligando a que la red se parezca a una máquina finita de estado. Para un juego desfavorece porque limita el funcionamiento de la red y el comportamiento del agente autónomo se puede hacer demasiado predecible si se comporta similar a una máquina finita de estado.

### 2.1.4 Ejemplos de aplicación de PMC.

La utilización de las redes neuronales y sobre todo de los PMC, ha llamado la atención de desarrolladores de juegos e investigadores, lo que ha conllevado a que existan trabajos que las respalden y promuevan su utilización. A continuación se hacen referencias breves a ejemplos presentes en dos de los libros más populares de inteligencia artificial para juegos.

**Ejemplo 1**

En el libro "AI Game Programming Wisdom 2" (Champandard, 2004) se explica como se utilizó una red neuronal (se empleó un PMC) como alternativa a los mapas de influencia. Los mapas de influencia, a través de una suma pesada, brindan una perspectiva estratégica del juego, permiten realizar una valoración del mismo y tomar decisiones basadas en el estado actual del juego.

La mayor desventaja que se encontró fue la complejidad computacional que involucra a las redes neuronales. Se trató de minimizar este problema, entrenando la red en modo offline, permitiendo a la red trabajar como máquina finita de estado, donde toma una entrada simple y determina el rendimiento para la misma.

Los resultados fueron favorables, se aparentó un mundo más realista y humano. La red permitió automatizar muchos de los tediosos procesos que están involucrados en la utilización de una suma ponderada, tales como la determinación de coeficientes y la selección de las variables pertinentes.

**Ejemplo 2**

En el libro AI for Game Developers (Bourg y Glenn, 2004) se realiza una explicación de un modelo (Ver figura 2.1) de PMC que puede ser implementado en cualquier escenario virtual de estrategias.

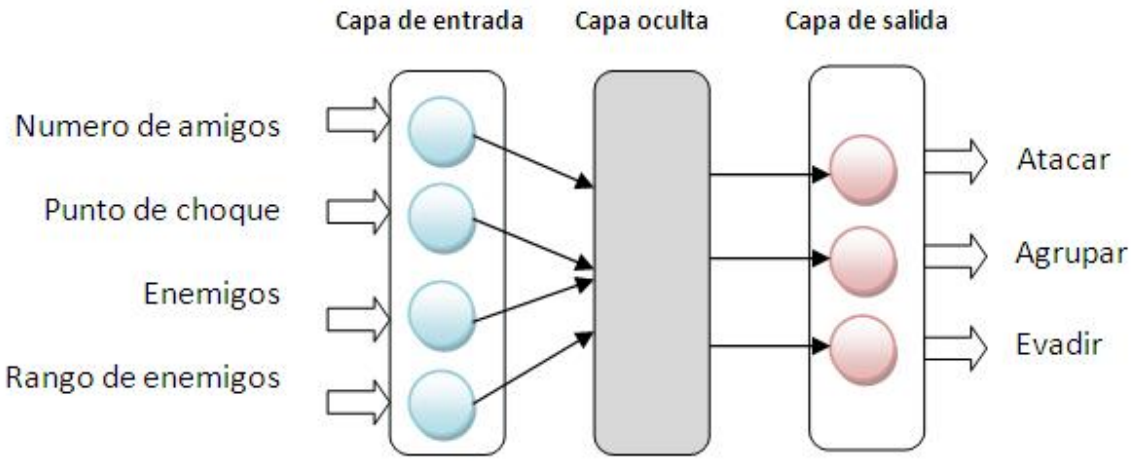



**Figura. 2.1 Modelo de 3 capas de un PMC.**

Este modelo se construyó para controlar los comportamientos de una criatura en un juego de rol persistente. Se empleó una red con la idea de mejorar el proceso de toma de decisiones de las criaturas.
El perceptrón posee 3 capas, con 4 neuronas en la capa de entrada y 3 neuronas en la capa de salida.

## Capítulo II Comparación de los modelos más representativos de las Redes Neuronales con el Perceptrón Multicapa

Las neuronas de la capa de entrada reciben la información del entorno en el que actúan respecto a número de compañeros dispuestos a combatir (Número de amigos), Puntos de golpe o vida (Punto de choque), Enemigos comprometidos a combatir (Enemigos) y Rango del enemigo (Rango de enemigos).
En la capa de salida Atacar (Atacar), Reunirse con sus compañeros (Agrupar) y Evadir (Evadir), constituyen acciones a realizar por el PMC sobre su radio de acción.
Este modelo pudo haber recogido más información del juego, como averiguar si los enemigos eran magos o no, o cualquier otro valor que pueda representar alguna de las categorías que pueden estar presentes dentro de un juego.
Una variante de este modelo puede verse en los demos (Red Neuronal 4-3-3), donde encontrará un video que ejemplifica su funcionamiento, puede observarse el proceso de entrenamiento, funcionamiento de la red, y cómo la red se adapta a los valores reales.
En la carpeta pruebas, puede encontrar la red para realizar las pruebas necesarias, para los valores deseados y otra variante donde se presenta el costo en tiempo de ejecución de los modelos.

### 2.1.5 Redes Neuronales y Videojuegos.

Los videos juegos se han llevado el protagonismo en el empleo de las redes neuronales, y dentro de estos, los juegos de combate y estrategias. Algunos ejemplos de juegos que utilizan redes neuronales son: Battle-Cruiser: 3000AD, Black & White, Creatures, Dirt Track Racing, y Heavy Gear.
En Battle-Cruiser: 3000AD (BC3K) se utilizan redes neuronales para controlar los NPCs así como para guiar negociaciones, comercio, y combate. BC3K las usa para toma de decisiones orientadas a objetivos básicos y pathfinding, con una combinación de aprendizaje supervisado y no supervisado.
En Black & White, el jugador tiene una criatura que aprende del jugador y de otras criaturas. La mente de la criatura incluye una combinación de representaciones simbólicas y subalterno-simbólicas, con deseos representados, como redes neuronales. Las Criaturas de los juegos usan técnicas de vida artificial, incluyendo redes heterogéneas en las que las neuronas son divididas en lóbulos que tienen juegos individuales de parámetros. En la combinación con los algoritmos genéticos, las criaturas usan las redes para aprender conductas y preferencias, con el tiempo.
El juego Dirt Track Racing usa una red neuronal para manejar alrededor de la pista. Finalmente, Heavy Gear las usa para la parte del control de mecanismos Mech, cada Mech usa redes especializadas para aspectos particulares.

### 2.1.6 PMC y Realidad Virtual.

Para los juegos, las redes neuronales ofrecen algunas ventajas importantes sobre las técnicas tradicionales de inteligencia artificial. Primero, usando redes neuronales, los desarrolladores simplifican el código de máquinas finitas de estados o de sistemas basados en reglas, relegando las decisiones a uno o más procesos de entrenamiento. Segundo, las redes neuronales ofrecen el potencial para que los juegos de inteligencia artificial se adapten a la manera en que son jugados los juegos.
A pesar de estas ventajas, las redes neuronales no han ganado un amplio uso en los videojuegos. Los desarrolladores de juegos, han usado las redes neuronales en algunos juegos populares, pero a la larga, su uso es limitado en comparación a la cantidad de juegos existentes.

Esto se debe a varios factores. Primero, las redes neuronales manejan grandes problemas no lineales, los cuales no pueden ser tratados por las técnicas tradicionales. Esto en ocasiones, hace que sea difícil que se comprenda que hace la red en cada momento y cómo llega a las conclusiones que brinda como resultado, lo que convierte en desconcertante la fase de pruebas.
Segundo, es difícil predecir en cada momento qué valor dará la red como salida, especialmente si es programada para aprender o adaptarse dentro del juego. Estos dos factores convierten las pruebas y

debuggueo de las redes neuronales relativamente difíciles comparadas con el de las máquinas finitas de estado u otras.
Estas desventajas no han imposibilitado, que su uso se haya ido expandiendo y sean cada vez más los interesados en su aplicación. Su presencia exitosa en los juegos (Ver epígrafe 2.1.5), muestra a los PMC como soluciones viables para la implementación de agentes autónomos. Las posibilidades de aplicación en juegos, está en los límites de la imaginación de los desarrolladores. Aunque su costo computacional es considerable, el desarrollo de modelos pequeños (Ver epígrafe 2.1.4) puede apalear esta limitante.

## 2.2 Red de Hopfield y Perceptrón Multicapa.

### 2.2.1 Funcionamiento de Hopfield.

Las redes de Hopfield, son redes muy interesantes por su propiedad de reconocer y recuperar patrones que están llenos de ruidos, se consideran redes muy rápidas en su funcionamiento, y resultan atractivas para ser utilizadas en los entornos virtuales.
En el siguiente epígrafe se realiza una comparación con el Perceptrón Multicapa, que ha sido probado con éxito en los juegos y simuladores, que a su vez servirá de base para esgrimir criterios ante la posibilidad de utilizar una u otra en entornos virtuales.

### 2.2.2 Comparación con el Perceptrón Multicapa.

En el artículo (Vicentín, Quintana, Insfrán, 2006) se realiza una comparación interesante entre el perceptrón multicapa y Hopfield. Esta puede resultar útil para una futura selección entre una y otra para ser empleada en los entornos virtuales.
Los puntos tenidos en cuenta fueron: el Proceso de entrenamiento y los Recursos computacionales utilizados.
El proceso de entrenamiento se refiere al proceso de selección de la topología de la red a emplear y los Recursos computacionales al costo que tiene sobre la PC el proceso de entrenamiento.

| Aspectos | Hopfield | Perceptrón multicapa |
|---|---|---|
| Diseño de la red | Simple. | Compleja. |
| Recursos computacionales | Mínimos. | Altos. |

**Tabla 2.1 Comparación entre Hopfield y Perceptrón Multicapa.**

### 2.2.3 Diseño de la red.

Hopfield se caracteriza por poseer un proceso de diseño de red simple, ya que una vez hecha la estructura de datos correspondiente solo resta por encontrar una buena configuración de patrones de entrenamiento.
Por su parte, la complejidad del perceptrón multicapa se debe a que se requiere un tiempo considerable, ya que su diseño depende completamente de la metodología prueba y error para la determinación de la cantidad de capas ocultas, números de neuronas en las mismas y valor de los factores de aprendizaje y momento.

## Capítulo II Comparación de los modelos más representativos de las Redes Neuronales con el Perceptrón Multicapa

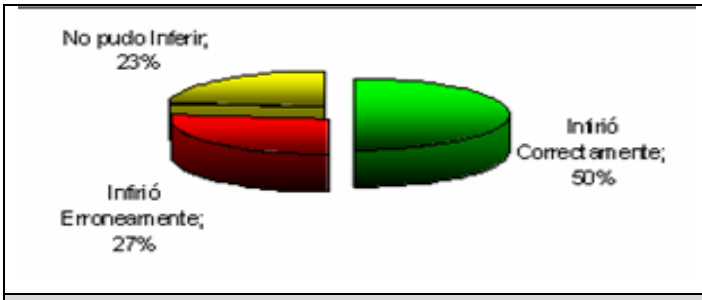
Las redes cuyo diseño sea simple permite que un mayor número de desarrolladores se inclinen por estas, (Vicentín, Quintana, Insfrán, 2006), por lo que favorecería a las redes de Hopfield sobre los perceptrones multicapa en el momento de seleccionar una para aplicaciones que se deseen ejecutar en tiempo real.

### 2.2.4 Recursos computacionales.

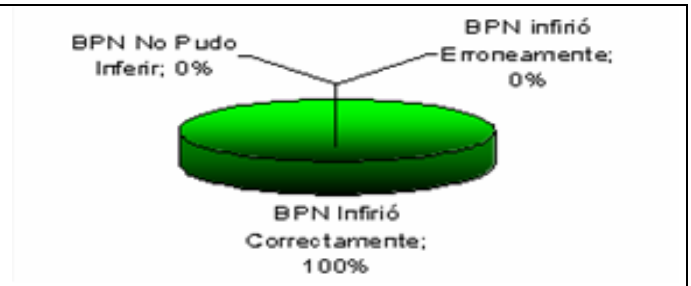
Los recursos computacionales empleados por Hopfield en la fase de entrenamiento son mínimos debido a que todo el procesamiento se resume a cálculos matriciales. Por su parte, los recursos para el perceptrón multicapa son altos, ya que requiere *n* iteraciones hasta que la red converja. Una vez entrenada la red, el funcionamiento (reconocimiento) puede ejecutarse con recursos computacionales mínimos.
Si recordamos que los perceptrones multicapas son muy empleados con entrenamiento online en algunos juegos, la simplicidad de Hopfield a todas luces garantizaría este tipo de entrenamiento en entornos virtuales, sumándole a esto la facilidad en su diseño, entonces habría que preguntarse por qué no han ganado popularidad en los productos de realidad virtual de hoy en día. En el siguiente epígrafe se analiza más al respecto.

### 2.2.5 Rendimiento de la Red.

El rendimiento es la capacidad de una red de procesar correctamente un conjunto de patrones de entrada. Este se considera bueno si la red es capaz de coincidir con las salidas deseadas para la mayoría de los patrones presentados en la fase de prueba. La Tabla 2.2 muestra una comparación de los resultados obtenidos para Hopfield y el Perceptrón Multicapa.




**Resultados de la red de Hopfield.**




**Resultados del Perceptrón Multicapa.**

**Figura 2.2 Resultados del rendimiento para Hopfield y Perceptrón Multicapa.**

Las características tenidas en cuenta en (Vicentín, Quintana, Insfrán, 2006), están referidas a los detalles del modelo de un cliente determinado: Edad, Nacionalidad, Estado Civil,…. Estos valores pueden sustituirse por otros para ser empleados en agentes autónomos u otras funcionalidades en entornos virtuales: Dimensión del escenario, Jerarquía, Si está solo o acompañado,… o cualquier otro de acuerdo al tipo de descripción del entorno que se necesite suministrar para dotar de inteligencia y acción un juego o aplicación de simulación. Esto se puede realizar debido a que las redes neuronales son funciones matemáticas cuyos valores deben tener un significado semántico para los que las necesiten emplear.
Según las pruebas, la Red de Hopfield solo pudo inferir correctamente la mitad de los datos de entradas, erróneamente un 27% y no se pudo inferir un 23%. Sin embargo, si se analizan los resultados del Perceptrón Multicapa se puede notar una marcada diferencia, ya que se infirieron correctamente todos los datos suministrados.

## Capítulo II Comparación de los modelos más representativos de las Redes Neuronales con el Perceptrón Multicapa

En los resultados expuestos en el artículo (Vicentín, Quintana, Insfrán, 2006) se concluyó que "la implementación de una red de Hopfield no es adecuada para este dominio de problema, esto es debido a que es muy difícil cumplir con la ortogonalidad necesaria en los patrones, de manera que la red aprenda los suficientes para que en su funcionamiento infiera correctamente las salidas ante cada entrada. Además, el número de neuronas correspondientes al modelo de clientes presentado y las características intrínsecas de este algoritmo limita el número de patrones que la red puede aprender, para una posterior recuperación perfecta de la red en la etapa de funcionamiento según lo esperado por el usuario final en este tipo de escenario."
Existen varios problemas asociados a la Red de Hopfield (Vicentín, Quintana, Insfrán, 2006). Los dos más importantes se refieren a la cantidad limitada de datos que se pueden almacenar y la necesidad de que estos datos sean ortogonales entre sí, o sea que dados dos patrones de entrada deben diferir en al menos la mitad de sus componentes. Otro de los problemas achacados a las redes de Hopfield es su tendencia a caer en mínimos locales, como en las redes de retropropagación (Brío y Molina, 2003) y que requieren mucho tiempo de procesamiento hasta converger a una solución estable, lo que limita su aplicabilidad.

### 2.2.6 Aprovechando las ventajas de Hopfield en la Realidad Virtual.

Los resultados expuestos podrían inclinar la balanza para desechar de antemano la posibilidad de utilizar las redes de Hopfield en entornos virtuales por su inestabilidad en el rendimiento. El mismo escenario de comparación pudo haber sido cualquier ejemplo de un entorno de realidad virtual o juego, debido a la "flexibilidad" de las funciones matemáticas de aplicarse en "dominios" de problemas diferentes.
En Hopfield, su simplicidad en el diseño y sencillez en el proceso de entrenamiento, se convierten en agua si el interés es utilizarlos en un entorno virtual donde los agentes que las porten necesitan evolucionar o adaptarse con el menor grado de error posible a las nuevas condiciones del entorno.
Hopfield no garantiza este proceso de adaptación, ni brinda la confiabilidad que logran los Perceptrones Multicapas. No infiere una cuarta parte de los juegos de datos suministrados, y en otra, la respuesta dada es errónea, lo que imprime un grado muy alto de incertidumbre y no por su capacidad de "deducir" sino por limitaciones técnicas de estas redes.
La ventaja que posee Hopfield de ser un algoritmo muy simple de implementar, puede ser adecuada y conveniente en situaciones donde el dominio del problema sea identificable, y los patrones de entrada sean claramente ortogonales. Pero encontrar patrones ortogonales es realmente una tarea muy engorrosa para los desarrolladores, los que en ocasiones prefieren apostar, a pesar del costo computacional, a otras técnicas más seguras. Cuando los patrones sean claramente ortogonales, y el dominio de datos a aprender sea relativamente pequeño, Hopfield debe ser tenida en cuenta por los pocos recursos que consume. Permitiría aumentar el número de agentes autónomos que funcionan y sobre todo, se entrenan en tiempo real, lo cual en número, es más complicado de lograr para los perceptrones. El reconocimiento limitado de patrones con ayuda de Hopfield, puede ser recompensado, si se trabaja muy bien el dominio del problema del juego, con el aumento del número de agentes que pueden entrenarse y funcionar en tiempo real.
En contraposición con Hopfield, los perceptrones multicapa, a pesar de las desventajas en cuanto al uso de recursos computacionales requeridos y complejidad en la implementación, se encuadra perfectamente para cualquier tipo de escenarios debido a la robustez de dicha red, y además, se pueden alcanzar topologías capaces de funcionar en varios agentes en tiempo real. Y han sido estas razones, las de mayor peso, en que sean los perceptrones multicapas los más populares en mundos virtuales. No obstante, Hopfield puede ser una variante eficaz si se desea dotar a las aplicaciones de un mayor número de agentes no tan perfectos individualmente, pero que a la vista del usuario, pueden resultar realmente inteligentes por la cantidad numerosa que pueden ser simuladas en tiempo real.

## Capítulo II Comparación de los modelos más representativos de las Redes Neuronales con el Perceptrón Multicapa

### 2.3 Función de Base Radial y el Perceptrón Multicapa.

Las Funciones de Base Radial (RBF) son redes neuronales muy similares en cuanto a arquitectura al Perceptrón Multicapa (PMC), y su mayor diferencia está en el proceso de aprendizaje.
Una de las características más importantes de ambas, es su capacidad de aproximación universal, ya que son capaces de establecer cualquier relación no lineal entre la salida y la entrada con el grado de precisión deseado.

#### 2.3.1 Proceso de Entrenamiento.

En general, las RBF pueden entrenarse con procedimientos más rápidos que los perceptrones multicapa. En una primera etapa del entrenamiento se determinan los parámetros que gobiernan las funciones de base que corresponden a cada neurona, usando métodos relativamente rápidos como los algoritmos de aprendizaje no supervisado.
La segunda etapa implica determinar los pesos de la capa final, que requiere la solución de un problema lineal, y por tanto, también es rápida, aunque la mayoría de los métodos para entrenar PMC pueden también aplicarse a las redes RBF.
Ambas redes son de tipo feedforward. El único principio que la diferencia es la manera en que las unidades ocultas combinan valores que vienen de las capas precedentes. En la red PMC se usan los productos internos, mientras RBF utiliza distancias Euclidianas.
La función de base radial normalmente tiene una sola capa oculta por lo que la función de combinación se basa en la distancia de Euclidean entre el vector de la entrada y el vector de peso. Las redes de RBF no tienen nada exactamente igual que el término bias en un PMC. Pero algunos tipos de RBFs tienen una "anchura" asociada con cada unidad oculta o con la capa oculta entera; en lugar de agregarlo en la función de combinación como una bias, se divide la distancia de Euclidean por la anchura.

#### 2.3.2 El entrenamiento híbrido y el curso de la dimensionalidad.

Una comparación de varias arquitecturas debe separar los problemas de entrenamiento de los problemas arquitectónicos para evitar fuentes comunes de confusión. Las redes RBF son a menudo entrenadas por métodos "híbridos" en que los pesos ocultos (los centros) se obtienen primero por un aprendizaje no supervisado, y luego los pesos de la salida se obtienen por el aprendizaje supervisado.
No se aplica a menudo el entrenamiento híbrido a PMC porque no es conocido ningún método eficaz para aprendizaje no supervisado que entrene las unidades ocultas, excepto sólo cuando es una entrada.
El número de unidades ocultas requerido por los métodos híbridos se vuelve un problema serio cuando se aumenta el número de entradas. De hecho, el número requerido de unidades ocultas tiende a aumentar exponencialmente con el número de entradas.
El entrenamiento supervisado para las redes de RBF puede hacerse por "Backpropagation" u otros métodos de optimización, o por la regresión del subconjunto.

#### 2.3.3 Función de Base Radial (RBF) y Realidad Virtual.

La arquitectura de PMC es más popular en las aplicaciones prácticas que las RBF, esto las ha convertido en la vanguardia de todas las redes neuronales. Sin embargo, según lo visto en los epígrafes anteriores, RBF es un tipo de red neuronal que ofrece mejoras aceptables sobre los PMC.
Si bien PMC se puede entrenar en tiempo real para arquitecturas pequeñas, RBF, por poseer un proceso de entrenamiento mucho más rápido, sobresale como otro tipo de red candidata a ser empleada en los

entornos virtuales, sobre todo por poderse explotar la posibilidad de un entrenamiento online, que siempre es un objetivo atractivo dentro de un juego.
Si Hopfield (Brío y Molina, 2003) tiene limitantes en el aprendizaje del número de patrones, RBF, al parecerse a PMC, logra convertirse en un excelente clasificador universal, debido al carácter local de las funciones de base radial, que posibilita el reconocimiento de patrones de cualquier complejidad en su distribución, por tanto, pueden representar cualquier problema que se desee dentro de un mundo virtual.
Las limitantes de la RBF están dadas por el número de neuronas de su capa oculta, el cual tiende a aumentar exponencialmente a medida que aumenta el número de entradas. Este incremento se convierte en un factor en contra, ya que mientras mayor sea el número de neuronas de la capa oculta, mayor será el costo computacional para el entrenamiento y la consulta a la red.
A pesar de este inconveniente, se debe recordar que generalmente, los agentes autónomos dentro de los entornos virtuales, se deben diseñar con el menor número de entradas posibles, lo cuál en la práctica se logra satisfactoriamente (Ver figuras 3.1, 3.2, 3.3 y 3.4). Si se utilizan pocas neuronas en la capa de entrada de una RBF, entonces el crecimiento de las neuronas de la capa oculta es mínimo, por lo que esta limitante de las redes RBF, no tiene un efecto considerable para agentes sencillos, lo cual garantiza que RBF, es candidata a mejorar la eficiencia de los PMC dentro de los mundos virtuales.

## 2.4 Modelos de la Teoría de la Resonancia Adaptativa y Realidad Virtual.

### 2.4.1 Introducción.

Las redes neuronales, cuando se comparan con Máquinas Finitas de Estados (MFE), simplifican el tamaño del código de estas (Fonge,2004), por lo que tienen en común la capacidad de conservar el conocimiento en forma de estados. En el caso de las MFE es suministrado por reglas programadas, y en el de las redes por los ciclos de entrenamiento.
Este "conocimiento" de los estados está guardado en los pesos existentes entre las conexiones de las neuronas. Cuando a una red se le encuesta con los mismos valores que a una MFE, la respuesta de las dos técnicas, en la gran mayoría de los casos, coinciden.
Sin embargo, las capacidades de las redes de adaptarse, permite que ocurran variaciones en los pesos, lo cuál favorece el aprendizaje de nuevos patrones, el aumento de la generalización de la red, y en este mismo proceso de generalización, surgen las primeras tendencias al olvido. Entonces, para algunos valores específicos, la RN no coincidirá con la MFE.
Una red que no se adapte, solo tendrá a su favor la propiedad de generalización, a diferencia de las MFE.
La adaptación, según las teorías evolutivas, garantiza que sobrevivan los individuos más preparados en una población. Los individuos (o NPCs) en un juego, que mejor se adapten a las nuevas circunstancias, serán los que con mayor probabilidad pasen las distintas pruebas a que son sometidos por la interactividad del juego.
La huella para determinar la adaptación (o evolución) biológica son los genes, y los genes de las redes neuronales son sus pesos. Los pesos, al ser modificados en conjunto, tienden a olvidar los primeros patrones aprendidos cuando las nuevas circunstancias distan de las que se presentaron en la fase de entrenamiento. Si la desviación estándar de los patrones a aprender no es muy grande en comparación a los aprendidos, el factor de olvido es insignificante, de lo contrario, ocurre una desmemorización para los patrones del entrenamiento inicial.
El proceso de conservar la información aprendida en las RNA se conoce como estabilidad y la capacidad de la red de adaptarse a los nuevos patrones es la plasticidad.
La gran mayoría de los tipos de redes existentes pueden ejecutar correctamente uno solo de estos procesos. La red que se caracteriza por su plasticidad, no logra estabilizar o recordar con efectividad el conocimiento adquirido en sus fases iniciales de diseño, mientras que las caracterizadas por su estabilidad en el conocimiento, es debido a que no aprende nuevos patrones o la desviación estándar de los patrones a aprender es muy pequeña, lo cual evita un olvido reconocible.

## Capítulo II Comparación de los modelos más representativos de las Redes Neuronales con el Perceptrón Multicapa

Los Modelos de la Teoría de la Resonancia Adaptativa (ART), desarrollada por Grossberg y Carpenter, intentan solucionar estos dilemas.

### 2.4.2 Dilema de la Estabilidad y Plasticidad del Aprendizaje.

El conseguir un modelo de RNA capaz de resolver uno de estos problemas es sencillo, conseguir un modelo que sea capaz de dar respuesta a ambos no lo es. Las redes más conocidas como el Perceptrón multicapa o el Adaline (Valluru, 1995) son capaces de aprender como han de responder ante unos patrones de entrada pero, una vez entrenados, el intentar que aprendan nuevos patrones puede suponer el "olvido" de lo aprendido previamente.
El modelo ART soluciona el dilema de la estabilidad y plasticidad del aprendizaje mediante un mecanismo de realimentación entre las neuronas competitivas de la capa de salida.
Cuando a la red se le presenta un patrón de entrada este se hace resonar con los prototipos de las categorías conocidas por la red, si el patrón entra en resonancia con alguna clase entonces es asociado a esta y el centro de cluster es desplazado ligeramente para adaptarse mejor al nuevo patrón que le ha sido asignado. En caso contrario, si el patrón no entra en resonancia con ninguna clase, pueden suceder dos cosas: si la red posee una capa de salida estática entrará en saturación pues no puede crear una nueva clase para el patrón presentado pero tampoco puede asignarlo a una clase existente, si la red posee una capa de salida dinámica se creará una nueva clase para dicho patrón, esto no afectará a las clases ya existentes.
En la fase de reconocimiento se efectúa una operación con los datos de entrada y los pesos asociados a cada neurona de la capa de salida, el resultado de esta operación debe indicar qué clase tiene mayor prioridad para ver si los datos de entrada entran en resonancia con ella. Por ejemplo, se podría calcular la distancia euclidiana entre los datos de entrada y los pesos, la clase ganadora sería aquella cuyo peso estuviese más cerca de los datos de entrada y por lo tanto sería la primera a la que se le intentaría asociar dicho patrón.

### 2.4.3 ART en mundos virtuales.

Los modelos ART son muy prácticos para NPC´s que necesiten aprender, pero nunca olvidar. La implementación de los mismos debe realizarse haciendo una exquisita selección de los datos a aprender, y con un control preciso de cómo se pueden generar nuevos datos durante el juego.
Se debe definir un máximo de categorías estáticas para los modelos ART, con capacidad para aprender nuevos valores, sino se define, el agente que utilice este modelo aprenderá sin límites y como no borra la información anterior, obliga que en cada consulta a la red sea mayor el número de categorías en las cuáles buscar, incurriendo así en el aumento del tiempo de ejecución y el rendimiento de la red se hace más lento, lo que influye en que el algoritmo no se ejecute en tiempo real para agentes con muchas categorías.
No obstante, no es muy práctica su aplicación en juegos, ya que no es interés siempre en los NPC´s recordar los valores iniciales de su primer aprendizaje. La misión de los agentes es adaptarse al nivel de juego de los usuarios. Los agentes deben, cuando se inicia la aplicación, poseer un nivel de juego promedio, el cual debe ir aumentando a medida que el jugador pueda ir adquiriendo las habilidades necesarias para ir venciendo los obstáculos del juego. A medida que el usuario va aprendiendo, se hace necesario que los NPC´s lo vayan haciendo al unísono para poder ofrecer una rivalidad que garantice la presencia de los jugadores (evitar la pérdida de interés de los usuarios por la carencia de retos).
En los modelos ART, el aprendizaje se realiza a través de la modificación resonante de las categorías aprendidas, las cuáles son tomadas por un factor de vigilancia. Este factor de vigilancia indica la cercanía del nuevo vector y las categorías aprendidas. En ocasiones, el nuevo vector es una mejoría radical de algunas de las categorías aprendidas, pero como no entran dentro del valor de vigilancia, entonces se les crea una nueva categoría, generándose cierta contradicción por no encontrarse en rango del factor de vigilancia.

## Capítulo II Comparación de los modelos más representativos de las Redes Neuronales con el Perceptrón Multicapa

Si en el entrenamiento inicial de la red, las primeras respuestas eran un poco ingenuas, a medida que aprende se desea que el modelo, de respuestas más inteligentes para una misma encuesta, pero como esa categoría que responde a esta encuesta nunca fue olvidada, siempre se mantendrá dando el mismo valor de salida, lo cual a la larga es una gran desventaja. La red se adaptará a lo que no conoce, pero no evolucionará, si era tonta en sus inicios, seguirá siéndolo en el tiempo, lo que con nuevas memorias.
Este fallo es perceptible para los usuarios de los juegos, las ingenuidades son permitidas a los agentes autónomos en momentos iniciales para usuarios inexpertos, pero a medida que estos van incrementando su habilidad de juego, las redes deben ir mejorando su nivel de respuesta.
Este fenómeno puede moldearse un poco si los agentes que tengan implementados modelos ART, poseen un nivel de conocimiento elevado desde su entrenamiento inicial, y a medida que el usuario eleve su nivel del juego, ellos aprenden valores para nuevas decisiones. Así se garantiza que su limitante esté en su incultura y no en “patrones mediocres” aprendidos.

### 2.5 Redes Neuronales Autoorganizadas: Mapas de Kohonen y Perceptrón Multicapa.

#### 2.5.1 Introducción.

Una de las redes más populares en aplicaciones y software son los Mapas de Kohonen. Estas poseen la capacidad de ir modelándose en función de los patrones de entradas, lo que las convierte en una propiedad de interés, que puede ser útil en diversos escenarios.
Los mapas de Kohonen son utilizados para el reconocimiento de patrones (entre otros usos), lo que puede resultar útil de acuerdo con la creatividad de los diseñadores y programadores de juegos y simuladores.
¿Imaginaría usted poseer una agente que sea capaz de clasificar un NPC o jugador en función de su comportamiento? Analice el siguiente ejemplo para más detalles:
Se ha creado un NPC para un juego de fútbol, que registra determinadas variables con ciertos valores de comportamiento, las cuales pueden ser: velocidad de desplazamiento, orientación en el terreno, efectividad en el golpeo del balón, velocidad en la conducción del balón, capacidad de posicionamiento para anotar gol, capacidad defensiva, etc. Luego de creado este NPC, el sistema es capaz de ubicarlo como delantero, centro o defensa de acuerdo a sus características. Esto es posible de manera sencilla con los mapas de Kohonen.
Los mapas de Kohonen agrupan por áreas los elementos que poseen rasgos comunes entre sí. Es decir, que luego de ser entrenados, ellos pueden servir como clasificadores para los nuevos valores suministrados, siempre que se hayan definido previamente los elementos distintivos para cada área (por ejemplo, a la izquierda del mapa estarán los defensas, al centro los delanteros y al final los centrales).
Ya sabemos que estas “maravillas” las pueden realizar los mapas, pero ¿serán capaces de lograrlo en tiempo real?
Para responder a esta pregunta, se buscó la complejidad temporal de estas estructuras. Como redes al fin, ellas cuentan de una etapa de entrenamiento y otra de consulta o funcionamiento.
El entrenamiento de una red puede ser online (durante el juego) u offline (antes de empezar o después de haberse concluido). No fue interés analizar cuál variante de entrenamiento es más conveniente utilizar, sino si realmente se puede hacer el entrenamiento y la consulta a la red en tiempo real.
Según los códigos analizados (Ver las aplicaciones utilizadas) la complejidad temporal para el entrenamiento del Mapa de Kohonen es O (n^4), que es relativamente grande si la intención es utilizarlos en aplicaciones de realidad virtual.
Sin embargo, la complejidad de la consulta a la red es de O (n^3), la cual es sólo un grado menor al del entrenamiento y aun así sigue siendo elevado. Un análisis superficial, haría rechazar automáticamente ambas complejidades debido a su tamaño, no obstante, la complejidad temporal para el entrenamiento del Mapa de Kohonen es igual a la del Perceptrón Multicapa visto en sesiones anteriores, aunque la de

consulta de este último es cuadrática, y hay que recordar que los perceptrones son las redes más difundidas en aplicaciones de realidad virtual.
La complejidad de los algoritmos, brinda una idea de su rapidez, que es al final, el factor de mayor importancia en una aplicación a la vista del usuario. Pero, para un mismo algoritmo se pueden tener diferentes tiempos de ejecución, y estos están determinados por la distribución de los valores de sus entradas. Para los Mapas de Kohonen el tiempo de ejecución depende de la cantidad de valores de entrenamiento, del diseño de la estructura de la red, y la cantidad de ciclos predefinidos para el proceso de aprendizaje.
Una valoración inicial de la complejidad temporal de Kohonen no sería un argumento suficiente para rechazarlos ni aceptarlos en aplicaciones que corren en tiempo real, pero unida al tiempo de ejecución en memoria serían concluyentes.
Por las razones anteriores, se hace necesario un análisis del comportamiento de los Mapas de Kohonen para sus dos fases más importantes: el entrenamiento y la consulta (o respuesta).

### 2.5.2 Tiempos de entrenamiento.

El entrenamiento de una red es el proceso mediante el cual la red "aprende el conocimiento necesario" para su correcto funcionamiento, este varía de acuerdo al tipo de red y el método implementado. Generalmente se pueden identificar tres tipos de entrenamiento: entrenamiento supervisado, no supervisado e híbrido.
Los mapas de Kohonen utilizan entrenamiento no supervisado, mientras que el perceptrón multicapa utiliza aprendizaje supervisado.

### 2.5.3 Análisis del tiempo de ejecución del entrenamiento de los Mapas de Kohonen.

En las pruebas realizadas se utilizó un Mapa de Kohonen implementado en C++ y como entorno de desarrollo integrado C++ Builder 6.
Las variables predefinidas para la aplicación fueron: número de neuronas para las columnas (campo Columna en la tabla) y las filas (Fila), cantidad de pesos asociados a cada neurona (Pesos) y total de iteraciones para el entrenamiento (Número de Iteraciones). Estos valores fueron variados indistintamente en las pruebas realizadas.
El resto de los campos de la tabla 1 son el área (Área), que es el resultado de la multiplicación del número de neuronas de las columnas por el número de neuronas de las filas; y el parámetro tiempo (Tiempo), es el tiempo máximo que demoró en ejecutarse el entrenamiento del Mapa para una misma tupla de prueba (Ej. Columna: 20 Fila: 20 Pesos: 3 Número de Iteraciones: 200).
La generación aleatoria de los pesos del mapa, provoca que se obtengan diferentes valores de tiempo para una misma tupla de prueba, por lo que se seleccionó el mayor tiempo de ejecución entre todas las pruebas para una misma tupla.
Imagínese que posee una tupla1 (Área: 200 Pesos: 3 Número de Iteraciones: 200) y otra tupla2 (Área: 100 Pesos: 3 Número de Iteraciones: 200). Si el tiempo máximo de ejecución para la tupla1 fue de 0.099 y el tiempo máximo para la tupla2 fue de 0.120, sería ilógico creer que se ha obtenido el tiempo máximo para la tupla1. Un mayor número de neuronas implica un mayor número de procesamiento, y a su vez, de tiempo de ejecución. Lo cual obligaría a realizar más pruebas para obtener el verdadero tiempo máximo correspondiente para la tupla1, que debe ser al menos, igual al tiempo máximo de la tupla2.
Para evitar un mayor número de pruebas innecesarias, en casos como estos, se inició el estudio con áreas grandes y luego se fueron disminuyendo poco a poco. En los casos en que los tiempos máximos de áreas menores fueron mayores que los de áreas superiores, se procedió a sustituir los viejos por los nuevos valores obtenidos.

| Columna | Fila | Área | Pesos | Número de Iteraciones | Tiempo |
|---|---|---|---|---|---|
| 20 | 20 | 400 | 3 | 200 | 0.156 |
| 18 | 18 | 324 | 3 | 200 | 0.156 |
| 16 | 16 | 256 | 3 | 200 | 0.156 |
| 14 | 14 | 196 | 3 | 200 | 0.156 |
| 12 | 12 | 144 | 3 | 200 | 0.156 |
| 10 | 10 | 100 | 3 | 200 | 0.156 |
| 8 | 8 | 64 | 3 | 200 | 0.156 |
| 6 | 6 | 36 | 3 | 200 | 0.156 |
| 4 | 4 | 16 | 3 | 200 | 0.156 |
| 2 | 2 | 4 | 3 | 200 | 0.156 |

**Tabla 2.2 Comportamiento del tiempo de entrenamiento para iteraciones (200) y pesos (3) constantes y variaciones en el área.**

La tabla 2.2 muestra el comportamiento del tiempo para un área variable entre 400 y 4 neuronas, para pesos y número de iteraciones constantes, con valores de 3 y 200 respectivamente. Para cada una de estas configuraciones el tiempo fue de 0.156 segundos, lo que permitió llegar a la conclusión de que para valores pequeños de los pesos, y número de iteraciones igual a 200, la variación del área de los Mapas de Kohonen no afecta el tiempo de ejecución máximo en CPU (Consultar tabla para Kohonen en los Anexos).

¿Qué ventajas reportaría esto?

Que los desarrolladores podrían ser todo lo creativos que deseen a la hora de diseñar mapas de área menor a 400 neuronas, con 200 ciclos de entrenamientos y pesos menores a 3.

Las desventajas estarían dadas por poseerse una cota superior en cuanto al área, pesos y ciclos de entrenamiento, y la otra si el tiempo no es lo suficientemente pequeño para ser utilizado en un entorno virtual.

Se necesita comprobar si los resultados obtenidos para un número de iteraciones constantes iguales a 200 se pueden alcanzar para números de iteraciones superiores. Véase la tabla 2.3.

| Columna | Fila | Área | Pesos | Número de Iteraciones | Tiempo |
|---|---|---|---|---|---|
| 20 | 20 | 400 | 3 | 250 | 0.266 |
| 18 | 18 | 324 | 3 | 250 | 0.234 |
| 16 | 16 | 256 | 3 | 250 | 0.234 |
| 14 | 14 | 196 | 3 | 250 | 0.25 |
| 12 | 12 | 144 | 3 | 250 | 0.25 |
| 10 | 10 | 100 | 3 | 250 | 0.25 |
| 8 | 8 | 64 | 3 | 250 | 0.25 |
| 6 | 6 | 36 | 3 | 250 | 0.218 |
| 4 | 4 | 16 | 3 | 250 | 0.218 |

**Tabla 2.3 Comportamiento del tiempo de entrenamiento para iteraciones (250) y pesos (3) constantes y variaciones en el área.**

En este caso, el número de iteraciones es de 250, y los demás parámetros de entrada de la red se mantienen similares a los de la tabla 2.2. Sin embargo, aquí el tiempo no es constante, sino que varía ligeramente. Si para 200 iteraciones el tiempo era de 0.156 segundos, para 250 oscila desde 0.218 hasta 0.266. Veamos como se comporta para 300 iteraciones.

| Columna | Fila | Área | Pesos | Número de Iteraciones | Tiempo |
|---|---|---|---|---|---|
| 20 | 20 | 400 | 3 | 300 | 0.312 |
| 18 | 18 | 324 | 3 | 300 | 0.312 |
| 16 | 16 | 256 | 3 | 300 | 0.312 |
| 14 | 14 | 196 | 3 | 300 | 0.312 |
| 12 | 12 | 144 | 3 | 300 | 0.312 |
| 10 | 10 | 100 | 3 | 300 | 0.312 |
| 8 | 8 | 64 | 3 | 300 | 0.235 |
| 6 | 6 | 36 | 3 | 300 | 0.235 |
| 4 | 4 | 16 | 3 | 300 | 0.219 |

**Tabla 2.4 Comportamiento del tiempo de entrenamiento para iteraciones (300) y pesos (3) constantes y variaciones en el área.**

La tabla 2.4 refleja el incremento gradual del tiempo de ejecución del Mapa de Kohonen en comparación con los resultados anteriores. Aquí se puede observar como para números grandes de iteraciones (300) y áreas superiores a 10 el tiempo de ejecución del mapa alcanza los 0.312 segundos, sin embargo, para valores pequeños del área (menos de 100 neuronas) los tiempos son similares a los resultados vistos en la tabla 2.3.

Por tanto se puede concluir que el aumento del número de iteraciones provoca un aumento considerable del tiempo, que se hace mucho mayor con el aumento del área de la red. No obstante, los tiempos que se van obteniendo en cada incremento son relativamente altos, y puede que no sean muy prácticos para las aplicaciones de realidad virtual. Aunque el número de ciclos de entrenamiento de un mapa debe calcularse en un valor medio aceptable, de acuerdo a las características propias de una red neuronal, el efecto que posee su incremento en un mapa de Kohonen, no es lo que pudiera decirse “aceptable” para ser ejecutado en tiempo real en un entorno virtual.

Al tenerse en cuenta esta hipótesis se procedió a valorar el comportamiento del tiempo de ejecución de un mapa de kohonen para números pequeños de iteraciones. Ver la tabla 2.5.

| Columna | Fila | Área | Pesos | Número de Iteraciones | Tiempo |
|---|---|---|---|---|---|
| 20 | 20 | 400 | 3 | 150 | 0.141 |
| 18 | 18 | 324 | 3 | 150 | 0.141 |
| 16 | 16 | 256 | 3 | 150 | 0.125 |
| 14 | 14 | 196 | 3 | 150 | 0.125 |
| 12 | 12 | 144 | 3 | 150 | 0.125 |
| 10 | 10 | 100 | 3 | 150 | 0.125 |
| 8 | 8 | 64 | 3 | 150 | 0.125 |
| 6 | 6 | 36 | 3 | 150 | 0.125 |
| 4 | 4 | 16 | 3 | 150 | 0.125 |

**Tabla 2.5 Comportamiento del tiempo de entrenamiento para iteraciones (150) y pesos (3) constantes y variaciones en el área.**

Para este conjunto de muestra (Tabla 2.5) con un número de iteraciones igual a 150, el tiempo para las dos áreas máximas (400 y 324) es 0.141, relativamente inferior al 0.156 de la Tabla 2.2, y para áreas menores se mantiene constante a 0.125. Esto refleja una ligera disminución (En comparación con los resultados de la tabla 1 para 200 iteraciones) en el tiempo de ejecución para menores números de entrenamiento.

| Columna | Fila | Área | Pesos | Número de Iteraciones | Tiempo |
|---|---|---|---|---|---|
| 20 | 20 | 400 | 3 | 100 | 0.094 |
| 18 | 18 | 324 | 3 | 100 | 0.093 |
| 16 | 16 | 256 | 3 | 100 | 0.093 |
| 14 | 14 | 196 | 3 | 100 | 0.093 |
| 12 | 12 | 144 | 3 | 100 | 0.093 |
| 10 | 10 | 100 | 3 | 100 | 0.093 |
| 8 | 8 | 64 | 3 | 100 | 0.078 |
| 6 | 6 | 36 | 3 | 100 | 0.078 |
| 4 | 4 | 16 | 3 | 100 | 0.078 |

**Tabla 2.6 Comportamiento del tiempo de entrenamiento para iteraciones (100) y pesos (3) constantes y variaciones en el área.**

Para 100 iteraciones los tiempos son inferiores a los 0.1 segundos, lo que convierte en atractivo el total de ciclos empleados para aplicaciones que se deseen utilizar en mundos virtuales. El tiempo disminuye hasta 0.078 segundos para áreas inferiores a 100 neuronas, mientras que las superiores poseen cota máxima de 0.093 y 0.094 segundos.
Las pruebas de la Tabla 2.7, fueron realizadas para un número de iteraciones igual a 50 para el período de entrenamiento.

| Columna | Fila | Área | Pesos | Número de Iteraciones | Tiempo |
|---|---|---|---|---|---|
| 20 | 20 | 400 | 3 | 50 | 0.061 |
| 18 | 18 | 324 | 3 | 50 | 0.047 |
| 16 | 16 | 256 | 3 | 50 | 0.047 |
| 14 | 14 | 196 | 3 | 50 | 0.047 |
| 12 | 12 | 144 | 3 | 50 | 0.047 |
| 10 | 10 | 100 | 3 | 50 | 0.047 |
| 8 | 8 | 64 | 3 | 50 | 0.047 |
| 6 | 6 | 36 | 3 | 50 | 0.047 |
| 4 | 4 | 16 | 3 | 50 | 0.047 |

**Tabla 2.7 Comportamiento del tiempo de entrenamiento para iteraciones (50) y pesos (3) constantes y variaciones en el área.**

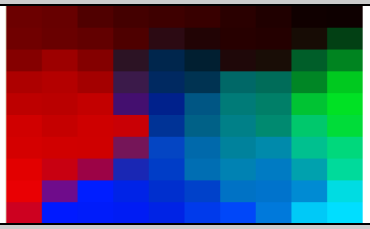
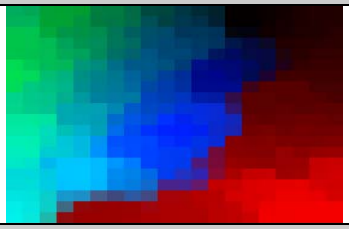
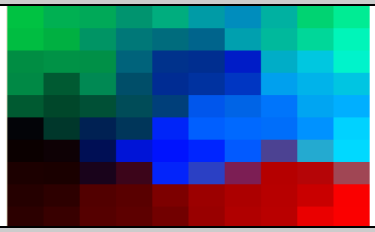
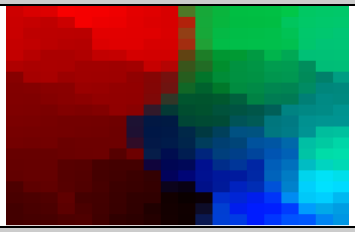
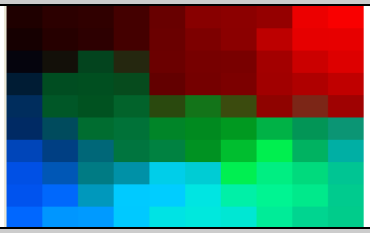
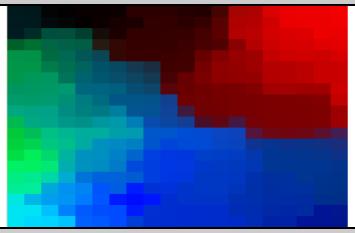
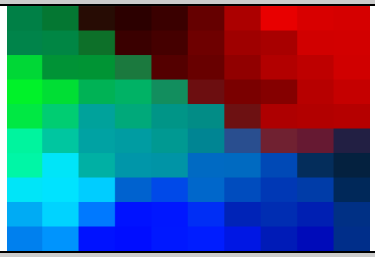
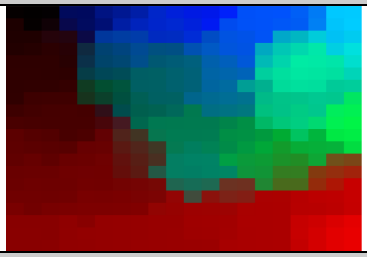
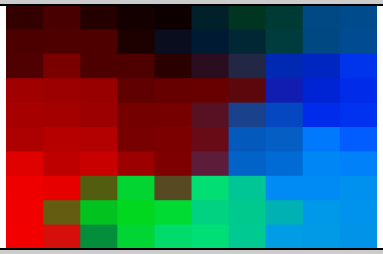
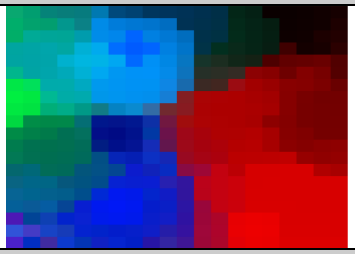
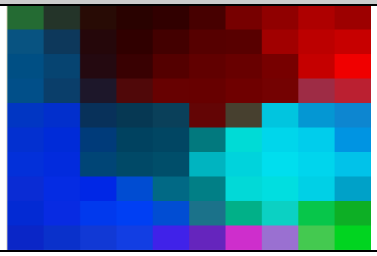
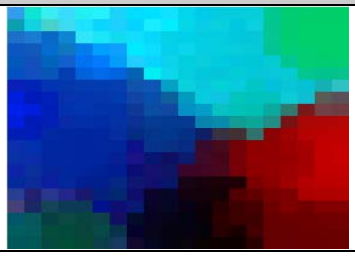
El tiempo de ejecución del entrenamiento de la red para un número de iteraciones igual a 50 es relativamente pequeño, lo que lo acerca a un tiempo real, y convierte a estas topologías pequeñas con pocos ciclos de entrenamiento en fuertes candidatas para ser utilizadas en aplicaciones de realidad virtual. Este valor de 50 iteraciones va en detrimento de la red durante su aprendizaje. Es necesario hacer énfasis en lo peligroso que puede resultar utilizar valores de entrenamiento muy bajos, porque se corre el riesgo que las redes no aprendan correctamente los patrones de entrenamiento, impidiendo su correcto funcionamiento.

## Capítulo II Comparación de los modelos más representativos de las Redes Neuronales con el Perceptrón Multicapa

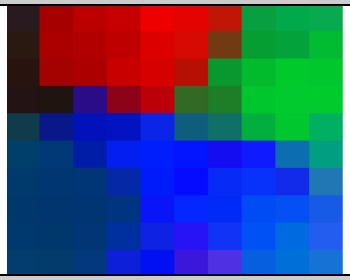
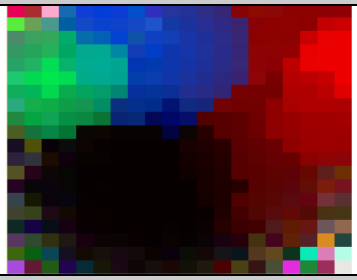
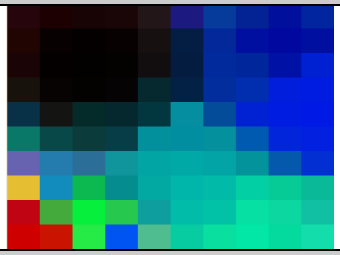
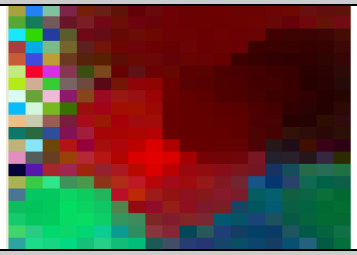
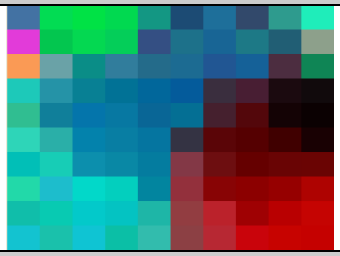
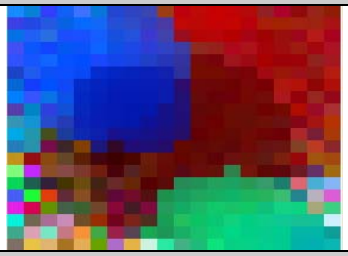
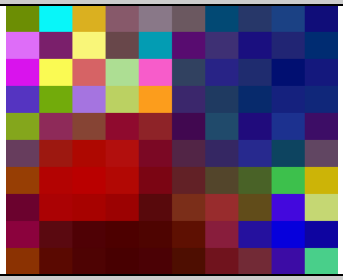
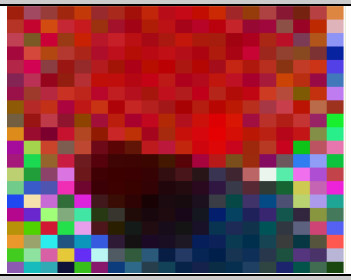
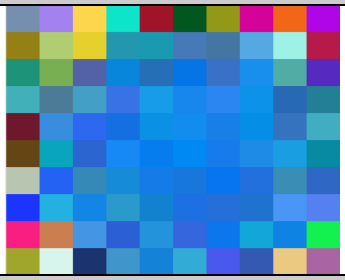
### 2.5.4 Estudio de los resultados de los mapas de Kohonen para diferentes ciclos de entrenamiento.

En la Figura 2.3 se puede observar los estados del Mapa de Kohonen para diferentes ciclos de entrenamientos con topologías 10-10-3 y 20-20-3. Estos resultados se obtuvieron luego de pruebas con valores aleatorios de entrenamiento.
La primera columna muestra el agrupamiento en colores para una red con 10 neuronas en las columnas, 10 en las filas para un área total de 100 neuronas. En la segunda columna tenemos una red con 20 neuronas en las filas y 20 en las columnas. En ambos casos el peso es 3 y los ciclos varían desde 300 hasta 25.
La propiedad fundamental de los mapas de Kohonen durante el proceso de entrenamiento es que tratan de parecerse a los valores suministrados en el mismo. Esto se puede imaginar de manera sencilla: si le pasamos los valores correspondientes a una imagen, en el proceso de entrenamiento el mapa tratará de "representar esa imagen". Si los valores de entrenamiento representan otras características, entonces el mapa en el entrenamiento, agrupará los patrones semejantes entre sí y permitirá al especialista observar patrones de agrupamiento que quizás no haya podido captar de manera visual.
¿Cómo se puede saber si los estados de entrenamiento presentados son correctos? Estas pruebas fueron realizadas teniendo en cuenta los colores RGB. Cada neurona representará a un color después de concluido el entrenamiento. En los resultados de la primera columna se puede observar a simple vista que el área de colores está cuadriculada. Esto es porque el área es de 100 neuronas. En la segunda, aunque se observa la cuadriculación, el poseer 400 neuronas hace menos nítida las diferencias entre las zonas de colores distintos.
Un mapa correcto será aquel dónde no existan áreas distintas con un mismo color. Esto quiere decir, que los colores iguales estarán siempre en una misma zona.
Al estudiar las imágenes, es fácil percatarse que hasta 100 ciclos de entrenamiento los resultados son deseables y sin mucho ruido para ambas topologías. Sin embargo, para el mapa de 400 neuronas y 50 ciclos, se observa como existen dos áreas que poseen un azul intenso (una superior de 16 neuronas en forma de cruz, rodeadas de otras de azul cielo y un área inferior mucho más extensa de azul intenso), lo que indica la presencia de ruido, como muestra de un bajo entrenamiento.
La suavidad en las fronteras de colores va dando una idea de la "calidad" del entrenamiento, y aunque a simple vista parezca no existir mucha diferencia, esto se debe al no empleo de una muestra de valores estáticos de entrenamiento y los mismos valores iniciales de los pesos para todas las pruebas.
¿Se puede entrenar un mapa de Kohonen con 50 iteraciones? No es aconsejable en sentido general, aunque algunos problemas específicos quizás puedan ser resueltos con un número pequeño de ciclos.
Para 25 iteraciones se logra observar una distribución de colores aceptables, pero se nota que apenas pudo disminuirse el tamaño de los radios para las neuronas ganadoras durante el entrenamiento.
Para 20, 15 y 10 iteraciones se notan a simple vista neuronas con diferentes colores que todavía no han recibido actualizaciones en sus pesos iniciales, lo que convierte a estos valores en inaceptables para el proceso de entrenamiento.
Los valores 5, 1 y 0 son una muestra del estado inicial del mapa en sus primeros pasos de entrenamiento.

## Capítulo II Comparación de los modelos más representativos de las Redes Neuronales con el Perceptrón Multicapa

| 10-10-3 | 20-20-3 |
|---|---|
| 300 | 300 |
|  |  |
| 200 | 200 |
|  |  |
| 150 | 150 |
|  |  |
| 100 | 100 |
|  |  |
| 50 | 50 |
|  |  |
| 25 | 25 |
|  |  |

**Capítulo II Comparación de los modelos más representativos de las Redes Neuronales con el Perceptrón Multicapa**

| 20 | 20 |
| --- | --- |
|  |  |
| 15 | 15 |
|  |  |
| 10 | 10 |
|  |  |
| 5 | 5 |
|  |  |
| 1 | 1 |
|  |  |
| 0 | 0 |

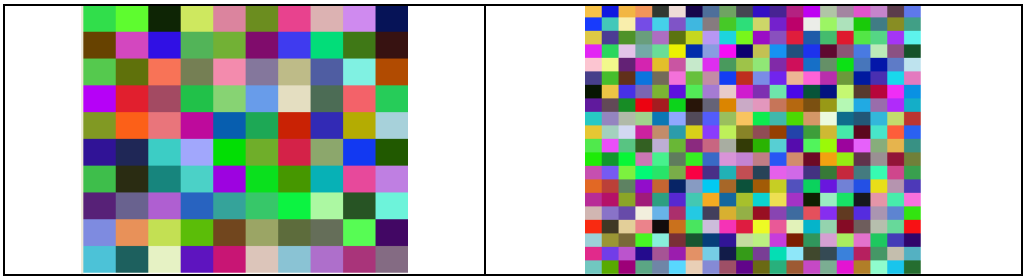


**Figura 2.3 Estados del Mapa de Kohonen para diferentes ciclos de entrenamientos con topologías 10-10-3 y 20-20-3.**

### 2.5.5 Análisis de la influencia de los pesos en el Mapa de Kohonen.

La tabla 2.8 muestra el comportamiento del tiempo, en un área de 400 neuronas. Inicialmente las pruebas se realizaron para 200 ciclos de entrenamientos y valor 1 de peso, con los cuales se obtuvo un tiempo de 0.141 segundos. Mientras que con pesos igual a 3 y 250 iteraciones el tiempo aumentó hasta 0.266 segundos. Luego, con un número de iteraciones constantes (400) y variando el número de los pesos desde 5 hasta 20, el tiempo se disparó hasta 0.422 segundos, para llegar a cerrar con 0.625 segundos para 20 pesos por neuronas.

| Columna | Fila | Área | Pesos | Número de Iteraciones | Tiempo |
|---|---|---|---|---|---|
| 20 | 20 | 400 | 1 | 200 | 0.141 |
| 20 | 20 | 400 | 3 | 250 | 0.266 |
| 20 | 20 | 400 | 5 | 400 | 0.422 |
| 20 | 20 | 400 | 6 | 400 | 0.422 |
| 20 | 20 | 400 | 7 | 400 | 0.422 |
| 20 | 20 | 400 | 8 | 400 | 0.422 |
| 20 | 20 | 400 | 9 | 400 | 0.516 |
| 20 | 20 | 400 | 10 | 400 | 0.516 |
| 20 | 20 | 400 | 15 | 300 | 0.516 |
| 20 | 20 | 400 | 18 | 400 | 0.625 |
| 20 | 20 | 400 | 20 | 400 | 0.625 |

**Tabla 2.8 Comportamiento del tiempo de entrenamiento para variaciones en los pesos de las neuronas.**

La influencia del peso (Ver Tablas 1-6) para números de ciclos y áreas pequeñas, tiene una repercusión mínima en el tiempo de entrenamiento. A medida que la topología de la red se va haciendo más grande (Ver Tablas 8), si se incrementan considerablemente los pesos de cada neurona, para un número de iteraciones constantes, la influencia del peso es mucho más determinante en el tiempo de entrenamiento de la red, aumentándolo a medida que se incrementen los pesos.

### 2.5.6 ¿Por qué no se realizaron pruebas para áreas mayores a 400 neuronas?

Hasta este momento, se ha visto que la mayoría de las pruebas realizadas fueron para áreas iguales o inferiores a 400 neuronas. Los estudios de prueba tuvieron en cuenta en momentos iniciales, valores mayores que 400, pero los tiempos elevados para estas topologías nos llevó a desecharlas desde temprano. Ver Tabla 2.9.

| Columna | Fila | Área | Pesos | Número de Iteraciones | Tiempo(s) |
|---|---|---|---|---|---|
| 20 | 20 | 400 | 3 | 50 | 0.061 |
| 20 | 20 | 400 | 3 | 100 | 0.094 |
| 20 | 20 | 400 | 3 | 150 | 0.141 |
| 20 | 20 | 400 | 3 | 200 | 0.156 |
| 20 | 20 | 400 | 3 | 250 | 0.266 |
| 20 | 20 | 400 | 3 | 300 | 0.312 |
| 20 | 20 | 400 | 5 | 400 | 0.422 |
| 25 | 25 | 625 | 3 | 200 | 0.203 |
| 30 | 30 | 900 | 10 | 200 | 0.282 |
| 35 | 35 | 1225 | 10 | 200 | 0.406 |
| 50 | 50 | 2500 | 20 | 200 | 1.14 |

**Tabla 2.9 Comportamiento del tiempo para áreas iguales y superiores a 400 neuronas.**

La tabla 2.9 muestra el comportamiento del tiempo para áreas iguales a 400 en sus primeras 7 filas, en las cuáles se varía el número de iteraciones desde 50 hasta 300 para 3 como valor de peso. Para 50 ciclos, el tiempo es de 0.061 segundos, lo que se considera bueno; sin embargo, para 150 ciclos (valor cercano al ideal) el tiempo es de 0.141, lo que se puede considerar elevado para redes que pretendan formar parte de un entorno virtual, dentro de los cuáles se les dedica muy poco espacio a las técnicas de IA por los altos recursos que demandan sus gráficos.
Las áreas superiores a 400, poseen tiempos elevados, que las convierten en inconsistentes para estos ambientes virtuales.

### 2.5.7 Estudio de las variaciones de los ciclos de entrenamiento.

Se había visto cómo influía la variación de las áreas y los pesos en el tiempo de entrenamiento para diferentes ciclos de entrenamiento, pero no se vio de manera general, que aporte tiene la variación del número de iteraciones sobre el tiempo de la red.
Según la tabla 2.10, el tiempo de entrenamiento es inferior a los 0.1 segundos para ciclos de entrenamiento igual o inferior a 100, lo que convierte a las topologías, con las cuales se alcanzan tales resultados, en candidatas para ser usadas en entornos virtuales.

| Columna | Fila | Área | Pesos | Número de Iteraciones | Tiempo |
|---|---|---|---|---|---|
| 20 | 20 | 400 | 6 | 25 | 0.032 |
| 14 | 14 | 196 | 7 | 25 | 0.031 |
| 20 | 20 | 400 | 8 | 50 | 0.063 |
| 20 | 20 | 400 | 9 | 50 | 0.063 |
| 4 | 4 | 16 | 3 | 75 | 0.078 |
| 20 | 20 | 400 | 3 | 75 | 0.093 |
| 4 | 4 | 16 | 3 | 100 | 0.093 |
| 20 | 20 | 400 | 3 | 100 | 0.094 |

| | | | | | |
|---|---|---|---|---|---|
| 20 | 20 | 400 | 3 | 125 | 0.11 |
| 16 | 16 | 256 | 3 | 125 | 0.11 |
| 20 | 20 | 400 | 3 | 150 | 0.141 |
| 18 | 18 | 324 | 3 | 150 | 0.141 |
| 20 | 20 | 400 | 3 | 175 | 0.156 |
| 18 | 18 | 324 | 3 | 175 | 0.156 |
| 20 | 20 | 400 | 20 | 200 | 0.359 |
| 20 | 20 | 400 | 3 | 250 | 0.266 |
| 18 | 18 | 324 | 3 | 250 | 0.234 |
| 20 | 20 | 400 | 18 | 300 | 0.453 |

**Tabla 2.10 Comportamiento del tiempo para variaciones en los ciclos de entrenamiento.**

En este caso se tienen áreas de hasta 400 neuronas, con pesos iguales a 3 para 100 ciclos de entrenamiento. Cuando las neuronas tienen como máximo pesos iguales a 9 y área de 400, se necesitan 50 iteraciones para que puedan entrenarse en tiempo menor a 0.1 segundos.

### 2.5.8 Análisis de los tiempos de entrenamiento para el Perceptrón Multicapa.

Hasta el momento se han analizado los mapas de Kohonen, redes que no son muy difundidas en los entornos virtuales. En esta sesión se realizará un análisis de los tiempos de entrenamiento para el Perceptrón Multicapa.
Las variables que se definieron para esta prueba fueron: cantidad de neuronas en la capa de entrada (Entrada), cantidad de neuronas en la capa oculta (Oculta), cantidad de neuronas en la capa de salida (Salida), ciclos de entrenamiento o número de iteraciones (Ciclos) y tiempo de entrenamiento (Tiempo) en segundos.
Los ciclos de entrenamiento definidos por defecto fueron 50000, un número grande, que ejerce una influencia considerable en el tiempo de ejecución de cualquier algoritmo. Este número puede ser mucho más pequeño (Ver epígrafe 3.3), lo que está en dependencia de cada problema en particular.
Los perceptrones multicapas pueden resolver problemas de cualquier complejidad, es por ello que se utilizó esta topología estándar para las pruebas, lo cual no entró en contradicción a la hora de realizar los entrenamientos con números aleatorios.

| Entrada | Oculta | Salida | Ciclos | Tiempo |
|---|---|---|---|---|
| 4 | 3 | 3 | 50000 | 0.032 |
| 6 | 3 | 3 | 50000 | 0.047 |
| 10 | 3 | 3 | 50000 | 0.047 |
| 18 | 3 | 3 | 50000 | 0.047 |
| 20 | 3 | 3 | 50000 | 0.047 |
| 9 | 9 | 9 | 50000 | 0.047 |
| 10 | 10 | 10 | 50000 | 0.047 |
| 20 | 20 | 20 | 50000 | 0.047 |
| 30 | 30 | 30 | 50000 | 0.062 |
| 35 | 35 | 35 | 50000 | 0.063 |
| 40 | 40 | 40 | 50000 | 0.079 |
| 50 | 50 | 50 | 50000 | 0.094 |
| 60 | 60 | 60 | 50000 | 0.125 |
| 70 | 70 | 70 | 50000 | 0.125 |

**Tabla 2.12 Tiempos de entrenamiento para un Perceptrón Multicapa**

La Tabla 2.12 muestra los valores de tiempo obtenidos para redes con tres capas y con variaciones del número de neuronas en cada capa. Para redes que tienen hasta 20 neuronas en las capas de entrada, oculta y de salida se obtiene como tiempo máximo en el entrenamiento de hasta 0.047 segundos.
Para topologías con 30 neuronas en cada una de sus capas, el tiempo máximo es de 0.062 segundos y de hasta 50 neuronas por cada capa, alcanza 0.094s. Se obtienen resultados por encima de 0.1s para redes superiores a las 50 neuronas por capa.
Las topologías que poseen valores de entrenamiento (recordar que este proceso se realizó con valores aleatorios) por debajo de las 50 neuronas por cada capa, parecen ser las candidatas fundamentales para ser entrenadas en tiempo real.

### 2.5.9 Tiempos de consulta

**Análisis de los tiempos de ejecución de las consultas para los Mapas de Kohonen.**

La Tabla 2.13 muestra un conjunto de mapas de Kohonen a partir de 400 neuronas, el tiempo de entrenamiento para estas áreas (el resto no se consideró por ser demasiado elevado) y el tiempo de consulta al mapa.

| Columna | Fila | Área | Peso | Numero de Iteraciones | Tiempo de Entrenamiento | Tiempo de Consulta |
|---|---|---|---|---|---|---|
| 20 | 20 | 400 | 9 | 50 | 0.063 | 0 |
| 20 | 20 | 400 | 3 | 75 | 0.093 | 0 |
| 20 | 20 | 400 | 3 | 100 | 0.094 | 0 |
| 20 | 20 | 400 | 3 | 125 | 0.11 | 0 |
| 20 | 20 | 400 | 20 | 200 | 0.35 | 0 |
| 20 | 20 | 400 | 18 | 300 | 0.453 | 0 |
| 50 | 50 | 2500 | 3 | 200 | - | 0 |
| 100 | 100 | 10000 | 3 | 200 | - | 0 |
| 200 | 200 | 40000 | 3 | 200 | - | 0 |
| 300 | 300 | 90000 | 3 | 200 | - | 0.016 |
| 400 | 400 | 160000 | 3 | 200 | - | 0.016 |
| 500 | 500 | 250000 | 3 | 200 | - | 0.046 |
| 600 | 600 | 360000 | 3 | 200 | - | 0.062 |
| 700 | 700 | 490000 | 3 | 200 | - | 0.094 |
| 800 | 800 | 640000 | 3 | 200 | - | 0.11 |

**Tabla 2.13 Mapas de Kohonen.**

Resulta interesante a la vista las diferencias de tiempo (entrenamiento y consulta) existentes para ambas topologías (Está dado por la complejidad temporal de ambos métodos y por el tamaño del código de ambas implementaciones).
Si bien se consideró innecesario calcular el tiempo de entrenamiento para áreas superiores a 400 por considerarse demasiado grande para ser utilizados en aplicaciones de realidad virtual, el tiempo es 0 para áreas de 40 000 neuronas, lo cual es atractivo para ser utilizado en tiempo real. Otro argumento de valor

es que el tiempo de consulta fue inferior a los 0.1 segundos para áreas iguales o inferiores a 490 000 neuronas, 700 en las columnas y en las filas.
El resultado más significativo es que aunque no se puedan entrenar topologías grandes de Mapas de Kohonen en tiempo real, las consultas en tiempo real si se pueden realizar a mapas considerablemente grandes. Esto le brinda al desarrollador que desee emplear un mapa, la posibilidad de ser todo lo creativo que desee.

### 2.5.10 Seleccionar los tipos de redes neuronales que se pueden emplear dentro de los entornos virtuales.

Al realizarse los análisis de tiempo para los algoritmos de entrenamiento y consulta de los Mapas de Kohonen y el Perceptrón Multicapa quedó pendiente la selección de cuáles topologías serían las aconsejables para ser utilizadas en aplicaciones en tiempo real.
La comparación en estos casos con el Perceptrón Multicapa se debió a que este es el tipo de red más popular en el mundo de las redes neuronales, y a su vez, la más exitosa en juegos y simuladores que han apostado por estas técnicas (Ver epígrafe 2.1.5)
Para la selección de las topologías que podrían ejecutarse en tiempo real, se tomaron instancias de estas redes y se pusieron a funcionar en la herramienta Scene ToolKit (STK) (Consultar anexo STK) . El mundo virtual que se usó para las pruebas, no estaba optimizado, lo cuál es un factor a favor, porque las ganancias de espacio en memoria que se obtienen al ser optimizado pueden ser empleadas en representar otros elementos de realidad virtual (incluidas otras técnicas de inteligencia artificial).
Las características de la PC dónde se realizaron las pruebas fueron las siguientes:

- Procesador: 3GHz
- Tarjeta: NVIDIA
- Video: 128
- RAM: 1Gb
- SO: GNU/Linux

Cuando una instancia de un Mapa de Kohonen fue colocada en el mundo virtual, se trató de observar que el entorno no sintiera el efecto del entrenamiento a que fue sometido el mapa.
En la tabla 2.14 se muestran algunas de las topologías que fueron puestas a pruebas junto con el tiempo de ejecución calculado durante el proceso de entrenamiento.

| Columna | Fila | Área | Pesos | Numero de Iteraciones | Tiempo |
|---|---|---|---|---|---|
| 20 | 20 | 400 | 6 | 25 | 0.032 |
| 14 | 14 | 196 | 7 | 25 | 0.031 |
| **20** | **20** | 400 | **8** | **50** | **0.063** |
| **20** | **20** | 400 | **9** | **50** | **0.063** |
| 20 | 20 | 400 | 3 | 75 | 0.093 |
| 20 | 20 | 400 | 3 | 100 | 0.094 |
| 20 | 20 | 400 | 3 | 125 | 0.11 |
| 16 | 16 | 256 | 3 | 125 | 0.11 |

**Tabla 2.14 Análisis de los Entrenamientos en Tiempo Real de Kohonen.**

Los mejores resultados fueron obtenidos para topologías con área similar o inferior a 400 neuronas, sin embargo, para esta área no siempre se obtuvieron los resultados esperados para diferentes ciclos de entrenamiento. Cómo las pruebas fueron realizadas para conocer cuál de estas instancias podía

entrenarse en tiempo real, se varió el número de iteraciones para el entrenamiento, dónde se obtuvieron diversos resultados para estos. Para ciclos mayores o iguales a 75 se observó como el mundo virtual se paraba al ordenársele a la red que se entrenara, lo cual demostraba que no estaban listas para ser entrenadas en tiempo real. Los mejores resultados se alcanzaron para un número de iteraciones igual a 50. Para 50 ciclos de entrenamiento y 9 como números de pesos de las neuronas, no hubo problemas en el proceso de entrenamiento en tiempo real. La aplicación seguía su curso automáticamente.

Otra conclusión importante que se obtuvo es que el tiempo por debajo del que tienen que ejecutarse las redes neuronales para poder ser utilizadas en aplicaciones de realidad virtual (con las características de la PC descritas) es de 0.063. Cualquier tiempo superior a este por muy cercano que sea, estaría pendiente de la disponibilidad de recursos de la PC y de la cantidad de elementos del mundo y su nivel de optimización.
Según el análisis del tiempo de entrenamiento, límite (Valores en rojo) para ejecutar las aplicaciones en tiempo real en el mundo seleccionado, para los Perceptrones Multicapa, la máxima topología a utilizar es 50 neuronas en la capa de entrada, 50 en la oculta y 50 en la de salida, para 50000 iteraciones de entrenamiento.

| Entrada | Oculta | Salida | Ciclos | Tiempo de Entrenamiento |
|---|---|---|---|---|
| 4 | 3 | 3 | 50000 | 0.032 |
| 6 | 3 | 3 | 50000 | 0.047 |
| 10 | 3 | 3 | 50000 | 0.047 |
| 18 | 3 | 3 | 50000 | 0.047 |
| 20 | 3 | 3 | 50000 | 0.047 |
| 9 | 9 | 9 | 50000 | 0.047 |
| 10 | 10 | 10 | 50000 | 0.047 |
| 20 | 20 | 20 | 50000 | 0.047 |
| 30 | 30 | 30 | 50000 | 0.047 |
| **50** | **50** | **50** | **50000** | **0.063** |
| 80 | 80 | 80 | 50000 | 0.094 |
| 90 | 90 | 90 | 50000 | 0.11 |

**Tabla 2.15 Análisis de los Entrenamientos en Tiempo del Perceptrón Multicapa.**

Los resultados alcanzados dan una idea de cuál es la máxima topología con sus respectivos ciclos de entrenamiento que se puede poner a aprender para una sola instancia de los Mapas de Kohonen o los Perceptrones multicapa en tiempo real. No obstante, en ambos casos existen resultados de interés.
Los Mapas de Kohonen deben usar una topología (aunque pequeña) aceptable para un agente virtual, incluso se puede considerar hasta grande si pensamos en un agente autónomo con solo algunas características de un mapa. La mayor limitante que enfrentan estos es el número de ciclos de entrenamiento (Ver epígrafe 2.5.7.) que utilizan, para 400 neuronas como área. Esto puede ser apaleado, si se disminuye el número de neuronas a utilizar, la cantidad de pesos de cada neurona, y se aumentan el número de iteraciones para el entrenamiento a un valor aceptable para el aprendizaje.
En el Perceptrón Multicapa ocurre lo contrario a los mapas de Kohonen. La topología de 50 neuronas en cada una de las tres capas, es demasiado grande para un agente virtual. Si la idea es poseer un mundo con varios agentes, entonces lo práctico es utilizar números pequeños de neuronas para cada capa (Bourg y Glenn, 2004). Recordar que mientras menor sea el número de neuronas, más rápido será el entrenamiento, y más rápida la consulta a la red, además de que evitaría tener que buscar numerosos

juegos de datos para el proceso de aprendizaje de la red neuronal, que para este caso es supervisado (Champandard,2004).

### 2.5.11 Análisis de los tiempos de consulta (Kohonen) y entrenamiento (Perceptrón Multicapa)

La tabla 16 muestra una "intersección" del tiempo de ejecución para la consulta a un mapa de Kohonen (mejor que su tiempo de entrenamiento) y el tiempo de entrenamiento del Perceptrón Multicapa (peor que su tiempo de consulta).

| Área | T. Consulta | Topología | Tiempo de Entrenamiento |
|---|---|---|---|
| 400 | 0 | - | - |
| 400 | 0 | - | - |
| 400 | 0 | - | - |
| 400 | 0 | 4-3-3 | 0.032 |
| 400 | 0 | 6-3-3 | 0.047 |
| 400 | 0 | 10-3-3 | 0.032 |
| 2500 | 0 | 18-3-3 | 0.016 |
| 10000 | 0 | 20-3-3 | 0.016 |
| 40000 | 0 | 9-9-9 | 0.016 |
| 90000 | 0.016 | 10-10-10 | 0.047 |
| 160000 | 0.016 | 20-20-20 | 0.047 |
| 250000 | 0.046 | 30-30-30 | 0.047 |
| **360000** | **0.062** | **50-50-50** | **0.063** |
| 490000 | 0.094 | 80-80-80 | 0.094 |
| 640000 | 0.11 | 90-90-90 | 0.11 |

**Tabla 2.16 Análisis de posibilidades de uso en tiempos reales para Kohonen.**

Como se había explicado, el tiempo real para estas redes subsistir en una aplicación de realidad virtual es de 0.063 segundos. Para Kohonen el tiempo de consulta "real" se alcanza para un área de 360000 neuronas, y para los perceptrones para topologías 3 capas con 50 neuronas en cada una de ellas.
En la sección anterior se realizó un análisis (Epígrafe 2.5.10) de lo poco usual de utilizar una topología de 50 neuronas por cada una de las 3 capas para los perceptrones. Para los mapas de Kohonen, el análisis hizo énfasis en el número de iteraciones de entrenamiento para 400 neuronas.
A pesar de esto, los mapas de Kohonen pueden resultar más interesantes ser usados en topologías grandes. Ellos son "muy buenos" reconociendo patrones geográficos, habilidad que no siempre es necesaria en todos los agentes autónomos de un juego, y que si es usada por solo un agente, entonces puede ser de interés que este único personaje se encargue de funcionar como un "sabelotodo", aunque si se necesitaran varios con sus características, entonces habría que disminuir el mapa a aprender.
Para estas grandes áreas se perfilan dos problemas fundamentales. El primero está en la codificación del área que va a aprender el agente o los agentes, lo cual puede resultar verdaderamente tedioso.
El segundo y más serio, es el tiempo de aprendizaje, que sería demasiado costoso, no solo para aplicaciones de realidad virtual, sino que el entrenamiento de redes tan grandes consumen muchos recursos de hardware.

Para esta última desventaja, se puede realizar un entrenamiento previo de la red cuando se haya concluido su diseño junto con las pruebas necesarias, guardar los pesos óptimos en ficheros que pueden ser cargados por la aplicación antes de iniciar la aplicación, lo que implicaría un costo mucho menor que en anteriores soluciones.

### 2.6 Resolviendo un problema desde cero con redes neuronales y Joone.

#### 2.6.1 Introducción

Una de las dificultades frecuentes es cómo aplicar las redes neuronales como solución a una situación determinada. Imagine un problema, con origen real, pero acotado para brindar una solución asequible.
Un proyecto de la facultad 5 necesita de un sistema inteligente que sea capaz de detectar la causa o las causas que pueden originar una avalancha de alarmas.
La ocurrencia de un fallo provoca que la alarma correspondiente se dispare. Un fallo también puede disparar a la vez más de una alarma. Para simplificar el problema, se asume que las causas de una avalancha de alarmas son diferentes en cada caso.
Entonces, visto el problema hasta aquí ¿cuál fue la propuesta inicial de solución con redes neuronales?
En la propuesta de solución inicial para dicho problema, se consideró que se recibiría un grafo como entrada, con el número de alarmas activadas. Se parte diciendo que una alarma se activa si es 1 o no se activa si es 0.
Es decir, que en este momento se puede percatar de que se puede tener ***p*** alarmas simultáneas (con ***p***>=1, ***p***<=N, N, número total de alarmas del sistema) disparadas y a las ***p*** alarmas se le asocia una única causa desencadenadora.
Ahora solo queda enfrentarse al reto de diseñar la red neuronal que responda al problema planteado.
En este momento se asume que nuestra red tendrá entradas binarias, que el número de neuronas de la capa de entrada estará asociado con el número de alarmas que posea el sistema. Teniendo estas variables, faltaría por conocer cuántas neuronas debería tener la capa de salida, si debe o no tener capas ocultas y de tenerlas, cuántas neuronas tendrían.

#### 2.6.2 El número de neuronas de la capa de salida.

Conociendo que se tienen N neuronas en la capa de entrada, y siendo ***p*** (con ***p***>=1, ***p***<=N) el número de alarmas simultáneas para cada entrada a la red y para cada ***p*** se tiene una sola causa, entonces se necesitará una salida diferente para cada ***p*** de entrada a la red.
Ahora, si ***p***=N, esto quiere decir que existirá una sola combinación de valores simultáneos ya que todas las alarmas estarían activadas. Sin embargo, si ***p*** <N, con ***p***>0, se pueden escoger ***p*** alarmas simultáneas entre las N entradas; pero, ¿cuántas combinaciones diferentes (**z**) se pueden obtener al seleccionar ***p*** elementos distintos de un conjunto N? La respuesta es:

$$z=\binom{N}{p}$$

A partir de esto se pueden seleccionar ***z*** combinaciones diferentes entre las N entradas para ***p*** alarmas simultáneas y por tanto se obtendría también z causas posibles para cada combinación ***p*** posible a seleccionar. El siguiente análisis puede representarse sencillamente mediante la fórmula:

#Neuronas de la capa de salida=$2^N-1$, con N número de neuronas de entrada.

Es decir que si se tienen como entradas a nuestro sistema 4 alarmas, el número de neuronas de la capa de salida debe ser 15, o sea, se reconocerían 15 causas posibles para cada una de las posibles combinaciones de entrada (no olvidar que esto es una limitación impuesta a la solución)

Si se precisa en la esencia de estos datos, se puede ver que solamente se está teniendo en cuenta entradas binarias y salidas binarias, por tanto como entradas recibirá todas las combinaciones posibles de números binarios para las N alarmas que pueda tener el sistema en la capa de entrada y solo se activará (tomará valor 1) una neurona de la capa de salida (que se corresponde con la causa que originó la avalancha) mientras las demás permanecerán con valor cero.

### 2.6.3 La capa oculta.

Después de conocido el número de neuronas para las capas de entrada y de salida se debería averiguar si se va o no a usar capa oculta. Teóricamente está demostrado que 3 capas son suficientes para clasificar cualquier patrón.
El número de capas de una red está determinado por la claridad con que se necesitan sus respuestas. Una red neuronal de una sola capa permite clasificar datos que se encuentren separados en dos regiones por una línea, donde los datos que se encuentran a ambos lados de la línea pertenecen al mismo conjunto o dominio, si por casualidad estos conjuntos o dominios no se pueden separar por esta única línea, entonces se hace necesaria la creación de una nueva capa.
Por lo antes mencionado, es fácil percatarse que los datos binarios que se usan para N=2 (que es el menor número de entradas reales con sentido) no se pueden separar mediante una línea por lo que se necesitaría introducir una capa oculta.
¿Cuántas neuronas se necesitan en la capa oculta? No existe una fórmula para la pregunta, la respuesta está en función del problema y la experiencia del especialista.
Para ello se hace el siguiente análisis: si la red posee 2 neuronas en la capa de entrada, según lo visto en el acápite anterior, le corresponde como salida 3 neuronas, por lo que las combinaciones para la capa de entrada al empezar a fragmentarse pueden lograr salidas robustas para entradas relativamente pequeñas ya que solamente se tienen 3 neuronas de salida. Sin embargo, el escenario sería diferente si se tienen 3 neuronas de entrada porque se necesitarían 7 neuronas de salida, y este resultado se vería más debilitado si se continúa aumentando el número de alarmas (neuronas) para la entrada ya que el crecimiento está vinculado a la función exponencial $2^N$. ¿Qué consecuencias trae consigo esta situación? Pues bien, que el número de capas ocultas de la red estará determinado por el valor de N.
De regreso a la propuesta inicial de red, se seleccionan 3 neuronas en la capa de entrada y por consecuencia 7 en la capa de salida. Pero como no se tenia interés en debilitar los pesos de la red y debido a que se iba a utilizar como entrada solamente valores binarios, lo que implica una disminución en el número de patrones de entrenamiento, se apuesta por tener en cuenta una capa oculta y esta a su vez con 5 neuronas para que sirvieran como puente a una ampliación de los valores de las 3 neuronas de entrada y luego se ampliara nuevamente pero sin un efecto considerable para evitar la pérdida de la semántica de los valores de entrada al tratar de ampliarse hasta las 7 neuronas de salida directamente, por lo que entrarían a una capa de 3 neuronas, pasaría por una proyección hasta una capa de 5 neuronas que suavizaría la nueva proyección hasta la última capa de salida de 7 neuronas.

### 2.6.4 El método de entrenamiento a utilizar.

Decidir que tipo de entrenamiento va a utilizar nuestra red neuronal está en función del problema que debe resolver. Para el caso de análisis se necesita del conocimiento de un experto para determinar la posible causa de una avalancha específica de alarmas, por lo que el método de entrenamiento es supervisado.
El experto debe suministrar cuáles son las causas para cada combinación de entrada. Este procedimiento puede resultar bastante tedioso mientras N tiende a un número grande, ya que cada vez será más difícil para el experto determinar la causa real por la cual se activaron las alarmas y es aquí dónde las propiedad de las redes neuronales se pondrían de manifiesto, porque sería capaz de llegar a una conclusión cuando

el patrón entrado no ha sido pasado en el periodo de entrenamiento. No obstante, según la particularidad de la situación dada, cuando N tiende a un número muy grande, el análisis sería otro e inclusive se podría utilizar aprendizaje no supervisado o establecer un híbrido entre ambos, pero esto quedaría para próximas soluciones.

### 2.6.5 Función de activación

La selección de activación es otro de los elementos a considerar para la presentación de un diseño de red neuronal. A la hora de su selección se debe tener en cuenta el tipo de función y el orden de su derivada, el cuál tiene un efecto directo sobre el costo de la red debido a que cada neurona dentro de la red hace una llamada a la función de activación para poder ofrecer su salida.
Cómo el ejemplo utiliza valores binarios como entrada para su entrenamiento, sin pensarlo mucho seguro les viene a la mente que la función más atractiva para estas situaciones es la sigmoidea, que puede describir su trayectoria para las $0 =< y =< 1$, lo cual resulta conveniente para el caso.

### 2.6.6 Tipo de Sinapsis

Cuando se considera la inclusión o no de una capa oculta y las posibles limitaciones que impondría a la hora de procesar las señales, se hace el análisis con el presupuesto de que una neurona emitía sus salidas a todas las neuronas de la capa siguiente, y ella recibía las señales de todas las salidas de la capa anterior.
La selección del tipo de sinapsis está en correspondencia con el tipo de entrenamiento a utilizar por la red. Para el ejemplo se usa sinapsis completa entre una neurona y todas las de la capa siguiente y ella como receptora de la información de todas las neuronas de la capa precedente.

### 2.6.7 ¿Cómo le fue a la red neuronal?

Después de tener en cuenta algunas de las características principales para diseñar una red neuronal, se ofrecen los resultados obtenidos. Como framework de desarrollo se empleó Joone (Consultar anexo Joone).
Valores de entrenamiento:

| E1 | E2 | E3 | S1 | S2 | S3 | S4 | S5 | S6 | S7 |
|---|---|---|---|---|---|---|---|---|---|
| 1.0; | 0.0; | 0.0; | 1.0; | 0.0; | 0.0; | 0.0; | 0.0; | 0.0; | 0.0; |
| 0.0; | 1.0; | 0.0; | 0.0; | 1.0; | 0.0; | 0.0; | 0.0; | 0.0; | 0.0; |
| 0.0; | 0.0; | 1.0; | 0.0; | 0.0; | 1.0; | 0.0; | 0.0; | 0.0; | 0.0; |
| 1.0; | 1.0; | 0.0; | 0.0; | 0.0; | 0.0; | 1.0; | 0.0; | 0.0; | 0.0; |
| 1.0; | 0.0; | 1.0; | 0.0; | 0.0; | 0.0; | 0.0; | 1.0; | 0.0; | 0.0; |
| 0.0; | 1.0; | 1.0; | 0.0; | 0.0; | 0.0; | 0.0; | 0.0; | 1.0; | 0.0; |
| 1.0; | 1.0; | 1.0; | 0.0; | 0.0; | 0.0; | 0.0; | 0.0; | 0.0; | 1.0; |

Las primeras tres columnas corresponden a las posibles entradas y las restantes a las salidas deseadas. Como se puede ver, esta red se va a caracterizar por memorizar y no por generalizar, pero si se excluyen algunos datos de entrenamiento podría generalizar soluciones no incluidas durante el entrenamiento. La generalización garantiza la inmunidad de las redes neuronales ante el ruido.
Es aconsejable mantener en los datos de entrada valores de ruido de manera aleatoria para evitar la distribución del aprendizaje, pero en la práctica, es recomendable evitar el ruido con valores booleanos.

La proporción de aprendizaje fue definida para 0.7 y el momentum en 0.6 para lograr un aprendizaje rápido, y se definieron 10.000 iteraciones para tratar de garantizar la convergencia, número que puede ser decrementado o incrementado de acuerdo a como alcance la red su convergencia.
Luego de concluirse el proceso de aprendizaje se observó que el error global de aprendizaje fue de 0.00145, lo que se puede considerar perfectamente bueno. Al estudiar la red para los valores de entrada entrenados se observó que siempre respondió a la perfección para las salidas deseadas, por lo que cumplió las expectativas planteadas.

### 2.6.8 ¿Cómo puede servir este ejemplo dentro de un entorno virtual?

Para utilizar una red neuronal, se tiene en cuenta el escenario al que se va a aplicar. Las redes neuronales guardan su conocimiento en vectores numéricos, lo cuál permite usar un mismo modelo de red en diferentes escenarios siempre que las relaciones que se establezcan entre las variables independientes y dependientes en un dominio, posean paralelamente la misma concordancia en el otro sin entrar en ambigüedades.
Por ejemplo, si se tiene una red neuronal encargada de funcionar como un AND y el nuevo problema tiene definido en su dominio valores que no entran en discrepancias con los que posee la red ni viceversa, entonces se puede reutilizar la misma en el nuevo problema.
Entonces para poder emplear esta red en un entorno virtual, solo habría que buscar un problema donde las relaciones entre las variables independientes y dependientes tengan la misma semántica que la red diseñada, pero que a su vez, no existan valores nuevos que entren en discrepancias o que no puedan ser generalizados por la misma.
Para los entornos virtuales, además se deben tener en cuenta algunos problemas de implementación.
Esto es un hecho bien conocido, ya que a la lógica de los juegos les es asignado un tiempo computacional muy pequeño. Esto requiere que se salve el mayor tiempo de simulación posible.
Las redes recurrentes deberían también ser evitadas, debido a sus capacidades. La simulación de redes recurrentes requiere múltiples iteraciones y tiempo que podría ser mejor gastado simulando múltiples redes feed-forward. En adición, su dependencia de los algoritmos genéticos puede ser una razón extra para desechar esta solución.
Finalmente, para las funciones de activación, usar ecuaciones no lineales permitirá la clasificación de patrones complejos. Usando funciones derivables permitirá que el algoritmo de Backpropagation ejecute el descenso del gradiente para los vectores de entrada. Las funciones sigmoideas dan esto.

# Capítulo III Resultados Generales.

### 3.1 Cuestiones importantes de las redes neuronales para entornos virtuales.

#### 3.1.1 Recomendaciones generales para redes neuronales.

**El procesamiento de datos.**

El objetivo del proceso de aprendizaje es permitir a la red neuronal ejecutarse bien para todos los casos en los que es invocada. Parte de esto se logra mediante la generalización intrínseca, y otra parte es responsabilidad de los diseñadores. Una de las mayores tareas cuando se aplican redes neuronales involucra el uso del conocimiento de problemas específicos para el aprendizaje. Esto se hace para procesar los datos durante el entrenamiento y la simulación.

Características simples pueden ser combinadas con características complejas que sean más relevantes. Generalmente estas técnicas incluyen ecuaciones matemáticas (para comparar valores usando su ratio) y algoritmos complejos (para extraer el promedio del color de una imagen).

El levantamiento simétrico permite reducir el número de situaciones que la red neuronal debe manejar. Esto es realizado por la búsqueda de mapeo de espacios de partes indeseadas hacia otras secciones preferidas. Pruebas sencillas antes de la simulación pueden determinar si las condiciones fueron conocidas. Si no, los valores apropiados pueden ser negados, arrojados o cambiados. Por ejemplo, al entrenarse un carro para girar a la izquierda solamente: si el obstáculo izquierdo está más allá que el derecho, los valores son cambiados. De igual manera, la dirección del comportamiento en el vector de salida es negada.

La normalización asegura que los valores tengan un formato significativo mediante el escalado y desconfiguración de los mismos. Por ejemplo, usando la información relativa a la posición actual del agente, o convirtiendo todos los valores al rango [0,1], lo cual es muy bien manejado por la red neuronal.

El propósito del aprendizaje es reemplazar los valores iniciales aleatorios de los pesos con un juego óptimo de datos.

#### 3.1.2 Aprendizaje supervisado.

El aprendizaje supervisado entrena a la red neuronal dando una salida deseada para cada vector de entrada. Este es el caso del método Backpropagation. La belleza de esta aproximación es la de ser capaz de entrenar la red tanto de manera online como offline. No obstante, adquirir los datos a usar en el entrenamiento es un proceso importante, y para juegos, existen varias opciones.

El entrenamiento interactivo permite al jugador humano supervisar el proceso de aprendizaje. Una interfaz configurable puede permitirle corregir la salida de la red, y producir directamente el ejemplo entrenado. Alternativamente, él puede confirmar la certeza de los resultados, en cada caso, la salida deseada es determinada por el incremento de los valores de soluciones alternativos. Opcionalmente, un hilo automatizado puede ejecutar la supervisión independiente del entrenamiento.

Clonar comportamientos permite a la red neuronal imitar a los humanos. En este caso, el jugador actúa sin preocupación directa sobre el entrenamiento. Otro proceso encargado de esto, es el monitoreo de las acciones humanas, mientras lo asociemos con un vector de entrada.

Este es el proceso más delicado, con características seleccionadas y tiempo de reacción, siendo dos de las mayores trampas: ¿qué factores fueron tomados en cuenta y cuando ellos influyeron en la decisión? La clave es introducir la noción del estado, haciéndolo tan persistente como sea posible.

Los componentes codificados duramente pueden ser reproducidos en orden para proveer entrenamiento inicial de una red adaptativa o para combinar múltiples funciones. En este caso, se necesitan reglas formales para dividir el espacio de entrada entre varios componentes. En los datos de entrenamiento, el espacio completo de entrada necesita ser representado, no solo en situaciones comunes de juegos. Una selección aleatoria de los ejemplos de entrenamiento provee una distribución satisfactoria.

**Aprendizaje no supervisado.**
Con el aprendizaje no supervisado, no se necesita un comportamiento feedback para que la red aprenda. En algunos casos, esto se logra usando redes neuronales como parte de un esquema de aprendizaje nivelado mucho más alto.

**Aprendizaje reforzado.**
El aprendizaje reforzado intenta aprender el mapeo óptimo desde los estados a la acción. Una red neuronal puede aprender estos mapas, dando una representación de los estados como un vector y la probabilidad de cada acción como un vector de salida. La aproximación online es preferida, desde que se requieren múltiples iteraciones.
Los algoritmos genéticos y otras terceras partes de algoritmos optimizados pueden además permitir que la red neuronal aprenda los valores óptimos de los pesos, y además potencialmente la estructura.
Preferiblemente, estos podrían ser aprendidos antes de iniciar el juego, con una considerable variación en los resultados que puede ser notada.

**Offline vs. Online.**
El aprendizaje offline tiene lugar antes de la simulación. Si el entrenamiento es usado, esto implica que todos los ejemplos de entrenamiento están disponibles. La calidad alcanzada por los algoritmos de entrenamiento en este caso es frecuentemente superior. Desde la perspectiva del juego, al ejecutarlo, este no sufre, debido a que el entrenamiento dinámico se realiza fuera de este.
El aprendizaje online permite a la red aprender dentro del juego. Tecnológicamente hablando, esto puede ser problemático para una red neuronal. Se necesita aprender nuevos vectores de entrada mientras todavía se asegura que no son olvidadas instancias previas. Los algoritmos de entrenamiento incremental tratan con este problema. Ellos pueden además ser una pérdida de calidad en los resultados, ya que hasta los datos inapropiados pueden ser aprendidos.
Los esquemas híbridos parecen idealmente acomodados para juegos, combinando lo mejor de los dos tipos de aprendizajes anteriores. La red, inicialmente es entrenada de acuerdo con parámetros razonables por defecto, mientras el aprendizaje online es utilizado para prevenir confusión en los datos.

**Entrenamiento supervisado utilizando Backpropagation.**
Este método recuerda los estándares para el aprendizaje de redes neuronales. El campo entero ha evolucionado junto con los algoritmos de entrenamiento. Con cuatro décadas de investigaciones y prácticas, los procedimientos están bien documentados. En esta sección no pretendemos duplicar estos esfuerzos sino concentrarnos en los problemas prácticos de optimización.

**Incremental versus Batch.**
Excluyendo el análisis de soluciones numéricas fuertes, existen dos tipos de algoritmos.
El algoritmo de entrenamiento incremental, que actualiza los pesos para cada uno de los ejemplos de entrenamiento, lo que implica que no converge para un juego específico de datos. No obstante, el aprendizaje online es posible. La rutina original del Backpropagation cicla repetidamente hasta que se obtiene un resultado satisfactorio. Para prevenir que el algoritmo caiga en un estado sub-óptimo, frecuentemente se adiciona el momento a los deltas de los pesos. El mayor problema con esta aproximación es la selección de la proporción de aprendizaje, la cual puede ser tediosa. Una solución a esto es conocida como aproximación estocástica o annealing (approximation or annealing) la proporción de aprendizaje es lentamente decrementada para animar al entrenamiento a converger hacia un máximo global.
En el proceso de aprendizaje Batch los datos de entrenamiento completo se cambian antes de actualizar los pesos. Entendiblemente, esto requiere un entrenamiento de tipo offline. Esto tiene muchas ventajas, incluidas la ausencia de proporciones de aprendizaje, lo cual permite al algoritmo converger

automáticamente a una solución. Lo mejor, y además uno de los mas simples algoritmos para esto es RProp. Esta combinación podría ser seleccionada siempre que sea posible.

**Ruido.**
Incluso si el ambiente es teóricamente perfecto, pueden levantarse datos contradictorios cuando se simplifica su representación. Especialmente, al mismo vector de entrada pueden corresponderle diferentes vectores de salida. Para los datos de entrenamiento, esto puede ser un problema, ya que esto revela incertidumbre acerca del resultado requerido: esto podría dañar seriamente la generalización. Estos casos pueden resolverse con la adición de entradas sin ambigüedad. Sorprendentemente, adicionando temblores artificiales para algunos ejemplos en pequeños juegos de datos pueden prevenir el sobreentrenamiento y reforzar la generalización. Se mantiene el ruido aleatorio para evitar la distribución del aprendizaje. En la práctica, se recomienda evitar el ruido con valores boléanos y además considerar el ruido en los vectores de entrada y de salida.

### 3.1.3 Recomendaciones para el entrenamiento de un Perceptrón Multicapa.

**En cuanto a la Arquitectura de la red.**
Como ya se mencionó en secciones anteriores (Ver epígrafe 2.6.2), un comienzo de construcción puede ser el de asumir que el número de neuronas en la capa oculta sean menos de la mitad de la suma del número de entradas más el de las salidas. Si no es suficiente porque el entrenamiento es pobre, se incrementa con pocas unidades y se procede a entrenar nuevamente. Este proceso se puede continuar hasta tanto no se observe mejoras en las pruebas de entrenamiento.
Sin embargo, hay que mantener clara la siguiente premisa: “La mejor red neuronal entrenada no es aquella cuyos errores están bajando continuamente”. Esto puede conducir a una memorización.
Existe una relación donde se puede determinar la cantidad de ejemplos de entrenamiento necesarios para un error deseado y un tamaño de red establecido. Esto es, $N > W/E$, donde N es la cantidad de ejemplos de entrenamiento, W la cantidad de nodos ocultos y E el error deseado.

**El entrenamiento.**
El primer paso del entrenamiento es la inicialización de la red. El inicio requiere que los pesos tengan un valor inicial. Existen varios mecanismos para la determinación de estos pesos. Pueden ser seleccionados de manera aleatoria o por aproximación mediante regresión múltiple. Sin embargo, si el proceso de entrenamiento fue iniciado y debe ser suspendido, no se requiere que la red sea entrenada desde sus comienzos. Los últimos pesos serían los pesos iniciales cuando se desee continuar el entrenamiento con nuevos grupos de datos.
La constante de aprendizaje juega un importante papel durante la definición de los parámetros para el entrenamiento. Por ejemplo, en Matlab este valor puede flotar en el sentido que la medida requiera cambios. El algoritmo automáticamente lo altera con fines de darle una mayor eficiencia al entrenamiento medido a través de la minimización del error cuadrático. Sin embargo, se repite la premisa: “La mejor red neuronal entrenada no es aquella cuyos errores están bajando continuamente”.

**Validación de los datos.**
Es conveniente añadir ruido a los pesos a través de ejemplos que permitan continuar el proceso cuando se conoce que el entrenamiento ha caído en un mínimo local.
Se debe repetir este proceso hasta conseguir que los parámetros sean satisfechos: error deseado siguiendo un patrón de caída aceptable, tiempos de entrenamiento no muy extensos. El tiempo crece exponencialmente de acuerdo al número de entradas.
Siempre es aconsejable guardar al menos el 10% de los datos (entradas y salidas) para hacer pruebas de la eficiencia de la red.

**Generalización del modelo.**

El mejor desempeño que pueda tener el modelo recién entrenado es que cuando sean presentados ejemplos nunca vistos por la red, pueda clasificarlos o aproximarlos con el mínimo error de ajuste.
Por tanto, se debe estar seguro de que no existan contradicciones en los datos y para ello se debe vigilar la preparación de los mismos como parte de los recursos para un buen desempeño de una red.

### 3.2 Consideraciones sobre el aprendizaje en los mundos virtuales.

En los mundos de los videos juegos, los más populares son aquellos que son capaces de ofrecerles a los usuarios un reto significativo a su inteligencia sin importar el nivel de destreza de los mismos. Este reto, va variando a medida que el jugador adquiere las habilidades necesarias para desempeñarse dentro del juego.
En este proceso de "adquisición de habilidades" dentro del juego, la inteligencia humana es capaz de ir reconociendo patrones de comportamiento de los NPC´s presentes en el mundo y sobre esos patrones traza el jugador las ideas para alcanzar la victoria.
Cuando el jugador la logra, se dice que venció el conjunto de habilidades requeridas por el video juego. El jugador que alcanza este punto, aumenta los deseos de seguir probando sus habilidades, por lo que aspira a encontrar una resistencia superior por parte de los NPC´s.
Para situaciones como estas, los desarrolladores de juegos crean diferentes estrategias. Colocan en el primer nivel del juego, los NPC´s con menos habilidades, y a medida que aumentan los niveles, la complejidad de estos agentes es superior al de la anterior.
Este trabajo puede ser realizado con una programación previa de estas nuevas habilidades definidas en los comportamientos de los steering behavior, pero que requieren de una definición ardua de las posibles decisiones que deben adoptar los NPC´s ante las iniciativas de los jugadores.
Las soluciones en el mundo de hoy, van dirigidas hacia el empleo de técnicas de aprendizaje que faciliten la adaptación de los NPC´s a las nuevas circunstancias, y además, brindarle al usuario un escenario menos predecible que con la utilización de técnicas estáticas de decisiones en los steering behavior.
Existen muchas algoritmos de aprendizaje, algunas de los más populares son los algoritmos genéticos y las redes neuronales, las cuales han sido empleadas con éxito en juegos famosos (Ver epígrafe 2.1.5).
Las técnicas de aprendizaje garantizan la adaptación a las nuevas circunstancias del agente que las emplee, y tal como ocurre en el proceso de evolución humana, permite que los individuos (agentes) estén mejor preparados para enfrentar la oposición de un jugador mucho más habilidoso. Este proceso encuentra diversas maneras de implementarse, y se realiza en correspondencia con las características específicas de cada juego.
Las redes neuronales, poseen algoritmos muy potentes de aprendizaje, el cual es conocido también como entrenamiento. El aprendizaje en redes neuronales consiste en ir variando los pesos de la red hasta alcanzar un valor que permite reconocer todos los patrones suministrados durante el entrenamiento (Ver epígrafe 1.5).La red en esta fase, siempre tratará de aprender los juegos de datos utilizados.
El proceso de aprendizaje en redes neuronales debe ser definido por el programador, este debe especificar cuando considere que la red debe aprender. Si el aprendizaje es online (cuando la aplicación está ejecutándose en tiempo real), entonces el programador definirá en qué momento se iniciará el proceso de adaptación a las nuevas circunstancias. Para el caso del aprendizaje offline se realizará antes o después de concluido el juego. Estas nuevas circunstancias de reentrenamiento son detectadas cuando la red posee un bajo rendimiento en su respuesta.
La baja respuesta en las redes neuronales puede ser causada por la consulta de valores distantes de los de su dominio de entrenamiento. Al estar la red expuesta a un rendimiento no deseado por el programador, entonces se procederá al aprendizaje o adaptación de la red.

Cuando se le dice a la red que aprenda, hay que tener claridad que los nuevos datos a reconocer, no sean ejemplos malos, es decir, que vayan en detrimento de su comportamiento lógico. Esto puede producir resultados indeseables, crean situaciones excepcionales dentro del juego, las cuales no serían bien tomadas por el jugador, que al final es el cliente para el que se trabaja.
Con el aprendizaje de los ejemplos buenos se debe tener cuidado en qué momento del juego invocarse. Si la red aprende del jugador todo el tiempo, es cierto que obligaría a este a superarse a sí mismo, pero llegaría un momento (en concordancia con las habilidades intelectuales del jugador) que podría imponerle una barrera muy alta, que haría abandonar a los más pesimistas.
A simple vista saltan dos cosas importantes a la hora de realizar el entrenamiento de NPC´s para juegos, evitar que el aprendizaje tenga efectos negativos sobre la red, y evitar que se alcance una red que solo aprenda ejemplos muy buenos (elitistas) que impidan al jugador obtener la victoria cuando su desempeño es muy bueno dentro del escenario de entretenimiento.
En el caso de los simuladores, la variante más potente, debe ser la segunda. El empleo de agentes autónomos que sean capaces de aprender valores correctos, garantiza un nivel mucho más elevado dentro de un simulador (el objetivo de los simuladores es evaluar a los usuarios). No obstante, el simulador necesitaría de ciertas imperfecciones dentro de los juegos ante las cuales el usuario debería tomar una decisión.
En las redes neuronales, como se había dicho arriba, el efecto del aprendizaje recae sobre el peso de las conexiones entre las neuronas. Cada vez que se invoca al aprendizaje de una red neuronal, estos pesos son modificados a favor de los nuevos valores, y en detrimento los datos aprendidos, lo que provocaría un bajo rendimiento de la red para los juegos de datos del entrenamiento inicial.
Esta es una de las desventajas del proceso de aprendizaje en redes neuronales, cada proceso implica el olvido de patrones adquiridos en etapas previas.
Este fenómeno es conocido como la incapacidad de las redes de lograr la Estabilidad-Plasticidad de su conocimiento.
La estabilidad consiste en que las redes no olviden el conocimiento aprendido y la plasticidad en que sean capaces de aprender nuevos juegos de datos. Existe una red, llamada Modelo de la Resonancia Adaptativa (ART) (Ver epígrafe 1.6.5), que trata de resolver este problema, pero no se ha tenido conocimiento de su aplicación en simuladores de realidad virtual ni juegos.
Por esta razón se debe tener mucho cuidado a la hora de especificar cuándo y qué aprender para evitar el olvido de patrones, el aprendizaje incorrecto y elitista.

## 3.3 Posibles escenarios para aplicar las redes neuronales en entornos virtuales y algunos modelos.

Las redes neuronales son técnicas que se pueden aplicar en disímiles escenarios dentro de una amplia variedad de aplicaciones. Los usos más comunes incluyen memorización, reconocimiento de patrones, aprendizaje, predicción y toma de decisiones; aunque su utilidad sólo se limita a lo que puede imaginarse. Tratar de describir cuales podrían ser estos escenarios resultaría una tarea infinita debido a que varían de aplicación en aplicación, y cada cual puede realizar una interpretación diferente del problema. Sería más factible realizar una ejemplificación de dónde y cómo han sido generalmente empleadas para poder quedarse con la idea de posibles situaciones, aunque no son estas las únicas que existen.

### 3.3.1 Selección de entradas y salidas.

En un juego, las entradas constituyen variables del mundo, similares a las usadas por una máquina de estado, que representa a atributos, eventos o caracteres del juego. La salida de una red neuronal puede ser una decisión, una clasificación, o una predicción.
Para seleccionar las entradas de una red, se toman variables propias del mundo (Bourg y Glenn, 2004) las que pueden ser referentes al estado del jugador, cantidad de municiones, fuerza, vitalidad y capacidad de ataque.

La salida podría ser un conjunto de posibles decisiones que el agente inteligente puede tomar, como hablar, correr lejos, atacar, o evadir, entre otras. También se pueden tomar como una clasificación de cómo el agente siente ciertos aspectos del juego, si aborrece, si detesta, si decide ser neutral, o prefiere amar.

### 3.3.2 Generalización de posibles escenarios para juegos de batallas.

**Conocimiento, "entendimiento" y explotación del terreno.**

Si se quiere emular un comportamiento inteligente se debe tener en cuenta que los jugadores encuentran lugares para esconderse o refugiarse. Rutas alternas a destinos importantes y hasta lugares excelentes para emboscadas. En la actualidad, todo esto es realizado solo mediante la codificación explícita previa de las reglas correspondientes.
La aplicación de los agentes autónomos y las redes neuronales logrará crear NPCs que interactúen dinámicamente con el terreno en el que se encuentren. Esto se logra, mediante el aprendizaje y la percepción del medio.

**Uso eficiente del trabajo en equipo.**

Desde el inicio, los videojuegos han consistido en su mayoría en un personaje que debe destruir a "los malos" para salvar algo (un planeta, una princesa, una chica, etcétera). Igualmente, durante todo este tiempo los NPC's son creados y colocados en los diferentes escenarios para que el jugador se encuentre con ellos. El comportamiento de estos es totalmente autónomo con respecto al de sus compañeros. Aplicando redes neuronales se podrán crear NPC's que interactúen entre sí para lograr un mismo fin ya que este es uno de los aspectos básicos de los agentes.

**Habilidad para cazar.**

En la mayoría de los juegos de video, el jugador va avanzando por los escenarios buscando rutas secretas, ítems que le puedan ayudar en su viaje y otras cosas; es el jugador quien va encarando a los enemigos y es él quien debe emboscarlos y cazarlos. Se podría esperar, de un comportamiento inteligente que los NPCs tuvieran la capacidad de cazar y perseguir al jugador. Esto, en combinación con el trabajo en equipo le daría a los juegos una nueva dimensión de jugabilidad.

**Instinto de supervivencia.**

Si se quiere emular un comportamiento verdaderamente inteligente, es necesario introducir el instinto de supervivencia, cosa que en los juegos no se había podido ver hasta épocas recientes, ya que en general cuando los NPC's enemigos veían a un jugador, corrían hacia él con la única determinación de "matar o morir". Esto hace que el comportamiento de estos no parezca inteligente, ya que hasta el menos inteligente de los seres vivos tiene un instinto de supervivencia muy arraigado.

### 3.3.3 Algunos ejemplos prácticos.

A continuación se muestran ejemplos donde se puede ver de manera clara cómo serviría la aplicación de estas metodologías en los videojuegos. La utilización de redes neuronales en juegos puede extrapolarse hasta los simuladores, o cualquier otra aplicación interactiva dentro de los entornos virtuales.

**Ejemplo 1**

En una persecución de un automóvil a otro, el jugador es el automóvil que será perseguido y el computador es el automóvil seguidor. El computador al principio no sabrá cómo es el escenario ni el perfil del jugador. Conforme avanza el juego, el computador se va creando un perfil del comportamiento del jugador, además va aprendiendo la forma del escenario, con lo cual la dificultad irá aumentando. Esto ocasionará que el jugador siempre encuentre un reto aceptable en el juego, aumentando la vida útil del sistema.

**Ejemplo 2**

En un juego de Rol, que los monstruos y personajes actúen de manera inteligente, de tal forma que nunca ataquen del mismo modo y que simulen el comportamiento inteligente, y que al ser agentes, puedan comunicarse y realizar simulaciones como por ejemplo, si un monstruo de una raza específica se encuentra a otro de otra raza, se atacarían, y eso llevaría a que disminuyeran. O si existe un enemigo maligno en el juego que también actúe sin necesidad de que el jugador llegue a un punto. El agente que es el malo del juego, puede seguir actuando dentro del juego aunque el jugador no lo vea.
Se puede destacar que de alguna manera, la implementación de esta tecnología implicaría el uso de mundos virtuales en los cuales deberán actuar los agentes autónomos. Esto no es un problema, ya que en la actualidad existen juegos que crean mundos virtuales y en los cuales se puede programar jugadores desde el exterior.

**Ejemplo 3.**

Se desea proponer un caso de aplicación de las técnicas descritas anteriormente como muestra comprobatoria de que estas funcionan.
Se crea entonces un juego de Rol (RPG), en el cual se desarrolla una historia fantástica en la que existen varios personajes, uno de ellos controlado por el jugador; los demás son NPC´s controlados por el computador. En este tipo de juegos la finalidad del jugador es completar diversas misiones y acertijos dentro del mundo fantástico para lograr un fin específico en la historia del juego, por ejemplo liberar un pueblo, salvar una princesa. El jugador puede moverse libremente por el mundo e ir descubriendo lugares y cosas interesantes.
Lo que hace diferente a este tipo de juegos de los demás es que el caracter que controla el jugador comienza con un estado inicial (status) que se compone de varios parámetros (fuerza, velocidad, energía mágica, astucia, etcétera); estos parámetros se contabilizan mediante puntos, los cuales irán aumentando a medida que el jugador valla avanzando en el juego. Además, el jugador, al ir adentrándose en el juego irá aumentando sus puntos de experiencia, los cuales le ayudarán a aumentar su estado.
El jugador puede recorrer el mundo eligiendo las rutas que desee, para así ir a cierta ciudad y si en ella se encuentra con una misión que realizar puede elegir entre realizarla o ir a otro pueblo para realizar alguna otra misión.
Uno de los factores más importantes en estos juegos es el de las batallas contra los enemigos, que casi siempre son monstruos fantásticos que habitan en los diferentes hábitats del mundo. Cuando el jugador está viajando a través del mundo, puede enfrentarse contra alguno (o algunos) de estos monstruos y si logra derrotarlos aumentarán sus puntos de experiencia.
En la actualidad, en este tipo de peleas, la inteligencia de los NPC's se reproduce mediante scripts o máquinas de estado finito. Utilizando una técnica de redes neuronales, tal como una Red de Hopfield (Vicentín, Quintana, Insfrán, 2006), podemos emular una inteligencia más completa de ciertos aspectos de los NPC´s.
Como resultado del análisis anterior, se obtiene un juego más divertido, ya que los NPC's nunca atacarán de la misma manera y podrán ir aprendiendo las diferentes técnicas que el jugador utilice para su ataque.

### 3.3.4 Modelos de redes neuronales empleadas en juegos.

Este modelo de red (Ver figura 3.1) puede ser bien empleado en una red neuronal que ocupe algún tipo de jerarquía dentro de un juego, lo cual le permitiría realizar una evaluación del escenario en que se encuentra. Dicho resultado debe ser codificado para que la respuesta tenga un efecto sobre el juego. Si la evaluación es baja, la decisión a tomar podría ser una retirada. Si la decisión es regular, quizás prefiera mantenerse en la posición actual. De ser buena, quizás se puedan desplegar la ofensiva final.

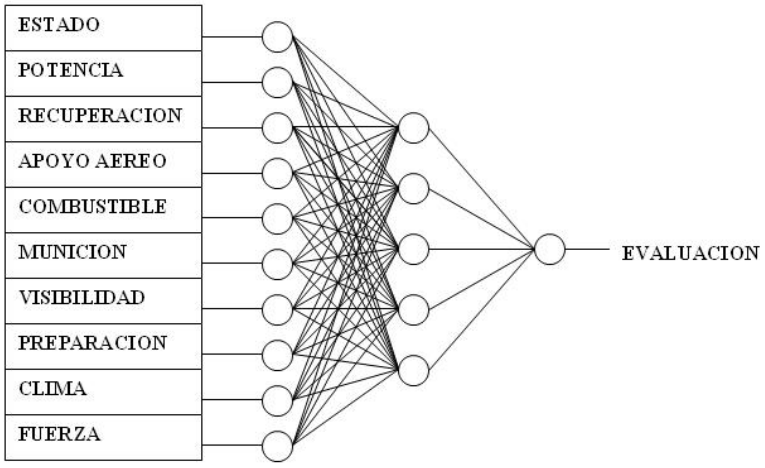



**Figura 3.1 Modelo de red neuronal con algún tipo de jerarquía dentro de un juego.**

Este perceptrón multicapa (Ver Figura 3.2) puede ser utilizado para el movimiento de objetos en un escenario dado. Recibe como parámetros de entrada la dirección de otro o de él mismo, y en función de esos valores, debe ir a la izquierda o ir a la derecha, lo cual puede ser apropiado para un juego de carreras.

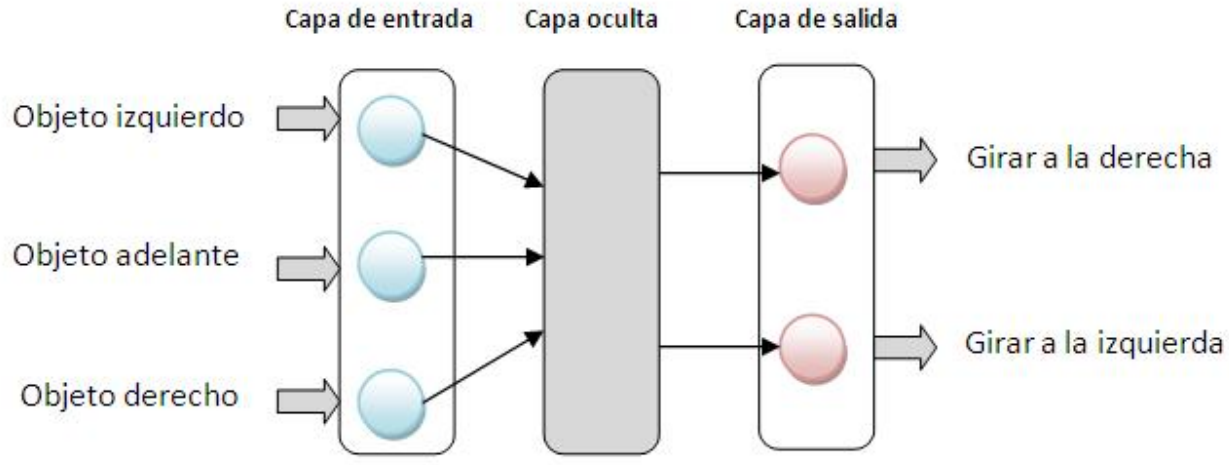



**Figura 3.2 Perceptrón Multicapa utilizado para el movimiento de objetos en un escenario dado.**

En la Figura 3.3 se muestra una red neuronal, que lleva a cabo determinadas decisiones de estrategias militares. Ella encuesta el número de unidades de combate en movimiento, tanto aéreas como terrestres y su respuesta puede ser realizar un ataque aéreo, terrestre, ambos o sencillamente no hacerlo.

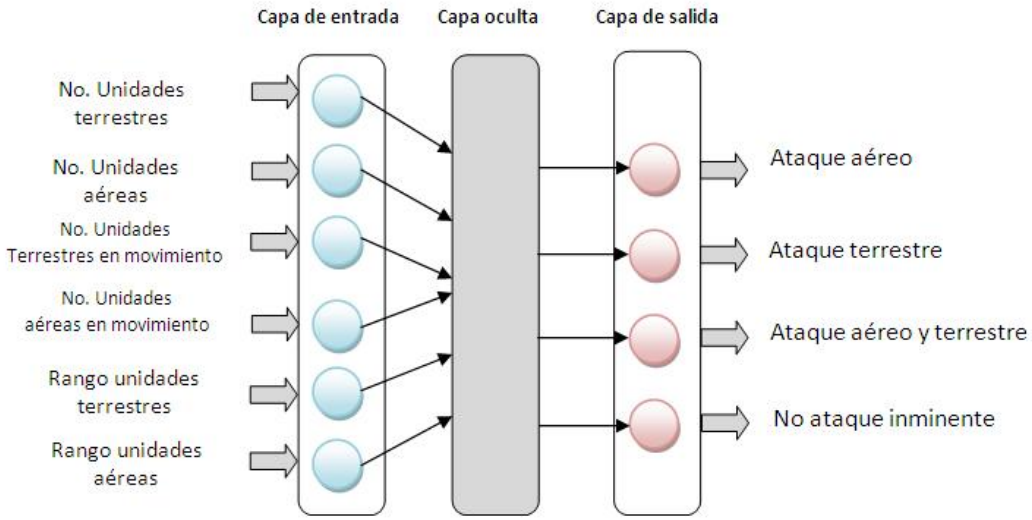



**Figura 3.3 Modelo de red neuronal usada en la toma de decisiones para estrategias militares.**

El Perceptrón Multicapa siguiente (Figura 3.4), se realizó sobre la línea de estrategias militares, la principal diferencia en comparación a los anteriores está en que se especifican la cantidad de neuronas de las capas ocultas, además de ciertas variaciones en los valores de entrada y salida, aunque pueden ser fácilmente comprendidos.

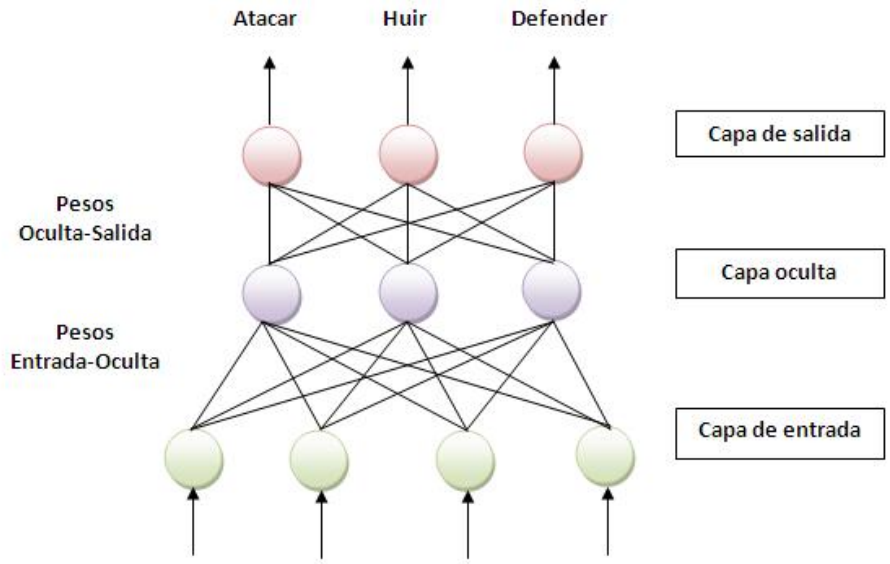



**Figura 3.4 Perceptrón Multicapa realizado sobre la línea de estrategias militares.**

### 3.4 Propuesta de pasos a seguir en el diseño de redes neuronales para entornos virtuales.

Diseñar una red neuronal es una de las tareas más engorrosas a la hora de enfrentarse a ellas. El diseño comienza con la identificación del problema a resolver y concluye cuando la red neuronal está lista para ser utilizada.
Existen muchas maneras de hacerlo aunque no se conozca una fórmula en este sentido. Varios autores han planteado sus experiencias previas en cuanto al tema. Esta puede considerarse una manera particular que puede ser seguida por los que no tienen mucha experiencia al respecto o decidan llevarla a cabo.

Pasos para el diseño de una red neuronal:

1. Identificar el problema a resolver.
2. Analizar el escenario del problema a resolver.
3. Enumerar los factores que van a influir sobre la red.
4. Enumerar el modo en que la red va a responder al medio ambiente.
5. Seleccionar el tipo de red neuronal a emplear.
6. Discriminar o compactar los factores de entrada y salida.
7. Seleccionar los valores de prueba.
8. Seleccionar el número de neuronas para las capas ocultas.

Estos pasos que se definen para diseñar una red neuronal para entornos virtuales, pueden ser empleados para crear una red con cualquier propósito general. La diferencia fundamental está dada en que las redes para entornos virtuales deben minimizar lo más que se pueda el número de neuronas sin entrar en la decadencia del problema, debido al alto costo que estas requieren al ejecutarse.

#### 1. Identificar el problema a resolver.

La identificación del problema a resolver consiste en realizar un estudio minucioso del lugar donde queremos emplear la red neuronal. Se debe analizar si no existe ninguna otra técnica de IA que pueda resolver el mismo problema, con los mismos resultados y con un costo computacional mucho menor.
Para utilizar redes neuronales, es aconsejable que los patrones de su dominio sean fácilmente identificables, lo cual tiene un efecto positivo en pasos posteriores.

#### 2. Analizar el escenario del problema a resolver.

Cuando se va a resolver un problema de cualquier índole, debe analizarse, hasta el mínimo detalle, en qué lugar se va a aplicar su técnica. Este es un elemento no siempre tenido en cuenta por los desarrolladores, que evitaría muy a menudo, tiempo de desarrollo y esfuerzo malgastado.
Es importante tener en cuenta cuál es el dominio de nuestro problema, qué factores influyen sobre él, el costo computacional del que se dispone, y cuáles serían los posibles efectos al inferir las redes o al entrenarse tanto online como offline sobre la aplicación.
En ocasiones se diseña una red perfecta, que tiene efectos muy graves sobre el entorno, debido al costo computacional de su consulta o a cualquiera de sus variantes de entrenamiento, o porque no se trataron correctamente las respuestas excepcionales, lo cual puede dejar un sabor desagradable en un juego o simulador.

**3. Enumerar los factores que van a influir sobre la red.**

Una red neuronal puede verse como una función matemática que posee entradas. Estas entradas, son características específicas que suministra el entorno sobre la red (constituyen su estímulo) y por tanto es el mismo entorno el que se va a encargar de ofrecer las diferentes combinaciones que van a afectar la técnica de IA.
Si las variables que influyen sobre la red no varían en el tiempo, quizás no tenga mucho sentido emplear una red neuronal, o no tenga sentido seleccionar estos valores como posibles entradas.
Es de suma importancia verificar que los factores que se consideren, deben interactuar directamente sobre la red neuronal, los cuales deben ser suministrados por el entorno, ya sea por otros agentes, por una "lectura" que sea capaz de hacer la red, o simplemente porque son estados propios de la aplicación que pueden ser accedidos por los agentes.
Estos factores pueden verse como las posibles variables de entrada para la red neuronal.

**4. Enumerar el modo en que la red va a responder al medio ambiente.**

Si es preciso seleccionar las entradas de la red correctamente, las salidas constituyen el efecto que van a tener estas sobre su entorno, y por tanto tiene vital importancia. La respuesta de la red, debe brindar un valor significativo para el usuario o la aplicación. Un resultado que no "aporte" al entorno, resultaría innecesario para ser considerado como posible salida.
Al igual que los valores de entrada, las salidas deben ser bien estudiadas, debe realizarse un estudio minucioso de qué es lo que se quiere de la red neuronal, y cómo debe influir sobre su medio, que es realmente el que va a procesar esa información. Si el entorno no es capaz de hacer nada con la respuesta de la red (aunque sea lógica para el diseñador) entonces las salidas no deben ser consideradas o debería cambiarse la aplicación, lo cual en ocasiones puede resultar peligroso por tratarse de cambios en los requisitos de la aplicación que se está desarrollando.

**5. Seleccionar el tipo de red neuronal a emplear.**

La selección del tipo de red neuronal va en correspondencia con el tipo de problema que se desea resolver.
Si la red neuronal se le va a aplicar a un agente que necesita un clasificador de patrones o reconocimiento de topologías, y estas topologías pueden ser representadas por imágenes, entonces lo más aconsejable es utilizar un Mapa de Kohonen.
Sin embargo, si en su problema es mucho más complicado identificar los patrones que se pueden establecer en el dominio, quizás lo más aconsejable sea utilizar un Perceptrón Multicapa. Hay que recordar que con un Perceptrón Multicapa con 3 unidades (capas) de procesamiento se puede resolver cualquier problema sin importar la complejidad.
Si un problema se puede resolver con dos o más redes satisfactoriamente, entonces se sugiere la selección del menor costo computacional, lo cual podría permitir agregar nuevas funcionalidades al sistema o sino disminuir el costo de ejecución de las funcionalidades actuales.

**6. Discriminar o compactar los factores de entrada y salida.**

Una de las limitaciones de las redes neuronales es el alto costo computacional. En este tiene mucha influencia el número de neuronas de la red, las cuales son las unidades básicas de procesamiento. Por esta razón se hace imprescindible evitar el número de neuronas innecesarias o que refieran a valores que no aportan nada a la red.
Existen valores que pueden ser compactados en uno o dos. Por ejemplo, si se necesita procesar la velocidad, el tiempo y la aceleración de un agente o proceso, lo primero que se hace es seleccionar una neurona para representar cada valor. Pero como su objetivo es disminuir el número de neuronas, entonces

debería aprovecharse la relación existente entre estas tres variables (aceleración=distancia/tiempo), y colocar dos de las tres, donde la tercera sería calculada mediante un despeje (o evaluación) de la relación planteada. Si las características del dominio lo permiten, se puede llegar a utilizar una sola neurona, o variable (podría ser la aceleración), en la cual irían de alguna manera las otras dos restantes.

**7. Seleccionar los valores de prueba.**

Si son importantes los primeros pasos, este a pesar de estar en la posición 7 no lo es menos. La selección de los valores de prueba de la red es de vital importancia. Primero, porque son ellos los que van a justificar la necesidad que existe de usar una red neuronal. Segundo, un conjunto de estos valores de prueba van a permitir darle a los pesos de la red neuronal los valores que necesitan para inferir correctamente la respuesta deseada. Tercero y último, un conjunto de estos valores va a permitir valorar el rendimiento de la red luego de concluido el proceso de entrenamiento.

Es aconsejable que dentro de estos valores de pruebas existan algunos valores con ruido, lo cual durante el entrenamiento ayudaría a que la red caiga en mínimos o máximos locales, favoreciendo la generalización sobre la memorización.

**8. Seleccionar el número de neuronas para las capas ocultas.**

La selección del número de neuronas de las capas ocultas sólo se realiza para las redes neuronales que las llevan, como el Perceptrón Multicapa y la Función de Base Radial, entre otras.

Para el caso de la primera, si esta tiene más de una capa oculta, habría que seleccionar el número de neuronas para cada una de estas capas. En la segunda, la selección se haría en una sola capa.

Este proceso de selección es uno de los más tediosos en el campo de las redes neuronales, y se realiza mediante el método de prueba y error durante el proceso de entrenamiento.

En él influyen el número de neuronas de las capas de entrada y de salidas. No existe una fórmula exacta para aplicar el método. Algunos autores sugieren empezar con un número de neuronas pequeño e ir aumentando si no se obtienen los resultados deseados en cada prueba.

Otros aconsejan que el número de neuronas inicial para las pruebas debe estar basado en que el número de neuronas en la capa oculta sea menos de la mitad de la suma del número de entradas más el de las salidas.

Otra manera de hacerlo y que está en estudio hoy en día, es mediante el empleo de algoritmos genéticos. Con estas técnicas se pueden generar no solo los pesos óptimos para cada neurona, sino el número de neuronas para la(s) capa(s) oculta(s).

# Conclusiones

Al finalizar nuestra tesis se le dió cumplimiento a los objetivos propuestos. Se realizó un análisis de las redes neuronales más populares, dónde se expusieron los elementos a favor y en contra para su futuro empleo en los entornos virtuales.
Se ejemplificó mediante un problema como podían ser utilizadas las redes neuronales para resolverlo, y se brindaron acotaciones sobre algunas de las cuestiones más importantes de las redes neuronales, como el proceso de entrenamiento y aprendizaje en mundos virtuales.
Para finalizar, se realizó una propuesta de pasos a tener en cuenta para resolver problemas de inteligencia artificial con redes neuronales.

# Recomendaciones

A este trabajo se le recomiendan los siguientes aspectos:

- Estudiar las redes neuronales no tenidas en cuenta en esta tesis.
- Proponer algoritmos de entrenamientos eficientes para los Mapas de Kohonen.
- Implementar una biblioteca con los tipos de redes neuronales que se pueden utilizar en los entornos virtuales.

# Bibliografía

**B**

(Bello, 1993) Dr. Rafael Bello Pérez. *"Curso introductorio a las redes neuronales artificiales"*. Universidad de Villa Clara.1993.

(Bollella, 2003) Ana Bollilla. "Redes Neuronales". Artículo. 2003.

(Bourg y Glenn, 2004). David M. Bourg y Glenn Seeman. *"AI for Game Developers"*.Pág.400. Julio 2004.

(Brío y Molina, 2003). Bonifacio Martín del Brío y Alfredo Sanz Molina. *"Redes Neuronales y Sistemas difusos"*. II Edición. 2003.

(Buckland, 2002). Mat Buckland. *"AI Techniques for Game Programming"*. Premier Press, Inc.2002

**C**

(Champandard,2004) Champandard, A. J., "The Dark Art of Neural Networks". *AI Game Programming Wisdom 2,* Charles River Media, 2004.

**E**

(Evans, 2001) Evans, R., "A1 in Games: A Personal View". Disponible en: www.gameai. com/blackandwhite. html, 2001.

**F**

(Fonge,2004). Jhon David Fonge. *"Artificial Intelligence for Computer Games"*. A K Peters, Ltd. 2004.

(Freeman, J. A., Skapura, 1993) Freeman, J. A., Skapura, D. M. Redes Neuronales. Algoritmos, aplicaciones y técnicas de propagación. Addison-Wesley/ Diaz de Santos. 1993.

**G**

(González, Baqueiro, Meza,2004) Alfredo González González, Omar Baqueiro Espinosa, Emmanuel Meza Cota. Metodologías de la I.A. (Agentes autónomos y Redes neuronales Supervisadas) aplicadas a NPCs (Non player characters).2004.

**H**

(Hilera, J. R., Martíne, 2000) Hilera, J. R., Martínez, V.J. Redes Neuronales Artificiales. Fundamentos, modelos y aplicaciones. Alfaomega. 2000.

**J**

(Jain y Mao, 1996) Anil K. Jain.Jianchang Mao.*"Artificial Neural Networks: A Tutorial".* 1996.

(Jones, 2003) M. Tim Jones. *"AI Application Programming"*. Charles River Media, Inc.2003.

**L**

(LaMothe, 2000) LaMothe, A., "A Neural-Net Primer," Game Programming Gems,
Charles River Media, 2000.

**M**

(Manslow,2002) Manslow, J., "Imitating Random Variations in Behavior Using a Neural Network," *AI Game Programming Wisdom,* Charles River Media, 2002.

(Manslow, 2002) Manslow, J., "Learning and Adaptation in Games," *AI Game Programming Wisdom,* Charles River Media, 2002.

(Matich, 2001) Damián Jorge Matich."*Redes Neuronales: Conceptos Básicos y Aplicaciones*". 2001

**P**

(Pérez, 2002). Juan Antonio Pérez Ortiz. *"Modelos Predictivos basados en redes neuronales recurrentes en tiempo discreto"*. Tesis Doctoral.2002.

**R**

(Rabin,2002). Steve Rabin. *"AI Game Programming Wisdom"*. Primera Edición. Charles River Media, Inc. 2002

(Rabin,2004). Steve Rabin. *"AI Game Programming Wisdom"*. Segunda Edición. Charles River Media, Inc. 2004.

**S**

(Simon,1994) Simon, S., "Neural Networks: A Comprehensive Foundation," Maxwell Macmillan International, 1994.

(Sweetser,2004) Sweetser, E, "Strategic Decision-Making with Neural Networks and Influence Maps," *AI Game Programming Wisdom 2,* Charles River Media, 2004.

(Sweetser,2002) AI Game Programming Wisdom, Charles River Media, 2002.

(Syed Merhoz Alam, 2004) Syed Merhoz Alam. Disponible en: www.codeproject.com/csharp/matrix.as. 2004.

**V**

(Valluru, 1995). Valluru B. Rao. *"C++ Neural Networks and Fuzzy Logic"*. M&T Books, IDG Books Worldwide, Inc. Fecha de Publicación: 06/01/1995.

(Vicentín, Quintana, Insfrán, 2006) Carlos Manuel Vicentín, Darío Abelardo Quintana, y Juan Carlos Insfrán. Comparación de Modelos Neuronales Utilizadas en la Toma de Decisiones. 2006.

**W**

(Watson, 1996). Mark Watson. *"AI Agents in Virtual Reality Worlds"*. John Wile/ Sons, Inc. 1996.

(Woodcock,2003) Woodcock, S., "Games Making Interesting Use of Artificial IntelligenceTechniques". Disponible en: http://gameai.com/ , 2003.

## Sitios visitados

http://ai-depot.com/
http://generation5.org/
http://www.aiwisdom.com/
http://www.gamedev.net
http://www.ai-junkie.com

# Anexos

### Anexo 1. Herramientas empleadas para la investigación.

**Joone**
El framework Joone (Java Object Oriented Neural) está disponible en (http://www.**joone**.org). Es un framework libre para crear, entrenar y probar redes neuronales.

**STK** (SceeneToolKit)
El STK es un engine gráfico desarrollado en la Universidad de las Ciencias Informáticas, por los miembros de los proyectos del polo de Realidad Virtual de la facultad 5.

**Visual y Consola SOM Test**
Es un Mapa de Kohonen implementado en C++ utilizado para las pruebas de tiempo (tanto de ejecución como de entrenamiento) y para el análisis del desarrollo del entrenamiento para números variables de iteraciones.

**Visual-Red Neuronal 4-3-3 (Con tiempo)**
Es un Perceptrón Multicapa empleado para las pruebas de tiempo del entrenamiento y consulta para diferentes topologías de las redes neuronales.

### Anexo 2. Complejidad algorítmica del Perceptrón Multicapa.

**Clase: NeuralNetworkLayer**

NeuralNetworkLayer() =>O(1)
Initialize(int NumNodes,NeuralNetworkLayer* parent,
NeuralNetworkLayer* child) =>O(n^2)
CleanUp(void) =>O(n)
RandomizeWeights(void) =>O(n^2)
CalculateErrors(void) =>O(n^2)
AdjustWeights(void) =>O(n^2)
CalculateNeuronValues(void) =>O(n^2)

**Clase: NeuralNetwork**

Initialize(int nNodesInput, int nNodesHidden,int nNodesOutput) =>O(n^2)
CleanUp() =>O(n)
SetInput(int i, double value) =>O(1)
GetOutput(int i) =>O(1)
SetDesiredOutput(int i, double value) =>O(1)
FeedForward(void) =>O(n^2)

BackPropagate(void) =>O(n^2)
GetMaxOutputID(void) =>O(n)
CalculateError(void) =>O(n)
SetLearningRate(double rate) =>O(1)
SetLinearOutput(bool useLinear) =>O(1)
SetMomentum(bool useMomentum, double factor) =>O(1)

**Clase: Control**

Initialize() =>O(n^4)
TrainTheBrain() =>O(n^4)
ReTrainTheBrain(double * vinput, double *vdesired) =>O(n^4)
Activar(double * input) =>O(n^2)
Salida() =>O(n)
Activacion() =>O(n)

**Anexo 3. Complejidad algorítmica del Mapa de Coñeen**

**Clase: som_Node**

som_Node(int,int,int)-> O(n^2)
float distance (const vector <double> &) -> O(n)
void update(const vector <double> &, const double, const double) -> O(n)

**Clase: SOM**

SOM(int,int,int,int) -> O(n^4)
int Epoch(vector <vector <double> > ) -> O(n^3)
som_Node* SOM::FindWinner(const vector <double> &)->O(n^3)
int complete()->O(1)
double getErrorValue(int,int) -> O(n^3)

**Clase: Control**

void InicializarValoresAleatorios()-> O(n^2)
void ConfigurarSOM_Entrenar(int x, int y, int w, int ciclos) -> O(n^4)

**Anexo 4. Tiempos para un Perceptrón Multicapa.**

| Entrada | Oculta | Salida | Ciclos | Tiempo | Error para el tiempo i. |
|---|---|---|---|---|---|
| 1 | 3 | 3 | 50000 | 1.531-4.079-3.797 | 0.0499-0.058-0.059 |
| 2 | 3 | 3 | 50000 | 0.625-0.109-0.531 | 0.0499-0.0499-0.0499 |
| 3 | 3 | 3 | 50000 | 0.125-0-0.234 | 0.0499-0.0499-0.0499 |

| | | | | | |
|---|---|---|---|---|---|
| 4 | 3 | 3 | 50000 | 0.032-0.015-0.031 | 0.0499-0.0499-0.0499 |
| 5 | 3 | 3 | 50000 | 0.031-0.016-0.046 | 0.0498-0.0499-0.0499 |
| 6 | 3 | 3 | 50000 | 0.047-0.016-0.016 | 0.0499-0.0498-0.0499 |
| 7 | 3 | 3 | 50000 | 0-0-0.016 | 0.048-0.0498-0.0499 |
| 8 | 3 | 3 | 50000 | 0.11-0.015-0.031 | 0.0499-0.0499-0.0499 |
| 9 | 3 | 3 | 50000 | 0.016-0.016-0.015 | 0.0499-0.0499-0.0499 |
| 10 | 3 | 3 | 50000 | 0.032-0-0 | 0.0499-0.0498-0.0498 |
| 11 | 3 | 3 | 50000 | 0-0.015-0 | 0.0498-0.0499-0.0498 |
| 12 | 3 | 3 | 50000 | 0.015-0.14-0 | 0.0499-0.0498-0.0499 |
| 13 | 3 | 3 | 50000 | 0-0.016-0.016 | 0.0499-0.0498-0.0496 |
| 14 | 3 | 3 | 50000 | 0.016-0.032-0.032 | 0.0498-0.0499-0.0498 |
| 15 | 3 | 3 | 50000 | 0.016-0-0 | 0.0498-0.0497-0.0499 |
| 16 | 3 | 3 | 50000 | 0-0.0160.047 | 0.0499-0.0495-0.0499 |
| 17 | 3 | 3 | 50000 | 0.016-0.015-0.015 | 0.0498-0.0499-0.0499 |
| 18 | 3 | 3 | 50000 | 0.016-0-0 | 0.0498-0.0498-0.0499 |
| 19 | 3 | 3 | 50000 | 0.016-0.016-0.015 | 0.0495-0.0495-0.0499 |
| 20 | 3 | 3 | 50000 | 0.016-0-0.016 | 0.0499-0.0497-0.0497 |
| 1 | 1 | 1 | 40000 | 0.922-0.984-0.984 | 0.0589-0.0765-0.0610 |
| 2 | 1 | 1 | 40000 | 0.922-0.047-1 | 0.0713-0.0499-0.0582 |
| 3 | 1 | 1 | 40000 | 0-0.219-0.016 | 0.0497-0.0499-0.0499 |
| 4 | 1 | 1 | 40000 | 0.094-0-1.563 | 0.0499-0.0499-0.0499 |
| 5 | 1 | 1 | 40000 | 0-0.016-0.031 | 0.0499-0.0499-0.0499 |
| 6 | 1 | 1 | 40000 | 0-0-0.032 | 0.0499-0.0497-0.0499 |
| 7 | 1 | 1 | 40000 | 0.047-0.016-0.031 | 0.0482-0.0499-0.0499 |
| 8 | 1 | 1 | 40000 | 0.016-0.016-0.015 | 0.0497-0.0499-0.0498 |
| 9 | 1 | 1 | 40000 | 0-0.016-0.016 | 0.0446-0.0498-0.0499 |
| 10 | 1 | 1 | 40000 | 0-0.016-0.016 | 0.0497-0.0499-0.0497 |
| 11 | 1 | 1 | 40000 | 0-0.016-0.015 | 0.0496-0.0495-0.0496 |
| 12 | 1 | 1 | 40000 | 0.031-0.015-0.016 | 0.0498-0.0497-0.0495 |
| 13 | 1 | 1 | 40000 | 0.031-0-0.015 | 0.0499-0.0494-0.0496 |
| 14 | 1 | 1 | 40000 | 0.031-0.015-0.015 | 0.0497-0.0495-0.0496 |
| 15 | 1 | 1 | 40000 | 0.016-0.015-0.015 | 0.0499-0.0497-0.0498 |
| 16 | 1 | 1 | 40000 | 0-0.015-0 | 0.0499-0.0498-0.0477 |
| 17 | 1 | 1 | 40000 | 0-0.016-0 | 0.0497-0.0496-0.0497 |
| 18 | 1 | 1 | 40000 | 0-0-0 | 0.0499-0.0495-0.0499 |
| 19 | 1 | 1 | 40000 | 0.016-0.016-0.031 | 0.0499-0.0499-0.0498 |
| 20 | 1 | 1 | 40000 | 0.016-0-0 | 0.0499-0.0499-0.0497 |

**Tabla 1. Tiempos obtenidos para un Perceptrón Multicapa**

**Anexo 5. Tiempos para los Mapas de Kohonen.**

| Columna | Fila | Área | Pesos | Numero de Iteraciones | Tiempo |
|---|---|---|---|---|---|
| 20 | 20 | 400 | 3 | 200 | 0.141[4]-0.125[4]-0.14[3] |
| 18 | 18 | 324 | 3 | 200 | 0.14[2]-0.156-0.125[2]-0.141[3] |
| 16 | 16 | 256 | 3 | 200 | 0.125-0.14[3]-0.141-0.156[5]-0.172 |
| 14 | 14 | 196 | 3 | 200 | 0.141[4]-0.14[2] |
| 12 | 12 | 144 | 3 | 200 | 0.141[4]-0.125[2]-0.14[2] |
| 10 | 10 | 100 | 3 | 200 | 0.14-0.141[4]-0.125 |
| 8 | 8 | 64 | 3 | 200 | 0.125[2]-0.141[4]-0.14[2] |
| 6 | 6 | 36 | 3 | 200 | 0.125-0.156-0.14[4]-0.141[3] |
| 4 | 4 | 16 | 3 | 200 | 0.14[3]-0.141[3] |
| 2 | 2 | 4 | 3 | 200 | 0.125[2]-0.14[5]-0.141 |
| | | | | | |
| 20 | 20 | 400 | 2 | 200 | 0.14[4]-0.141-0.125 |
| 18 | 18 | 324 | 2 | 200 | 0.125[4]-0.141[2]-0.14 |
| 16 | 16 | 256 | 2 | 200 | 0.125[3]-0.141-0.14 |
| 14 | 14 | 196 | 2 | 200 | 0.125[2]-0.14-0.141[3] |
| 12 | 12 | 144 | 2 | 200 | 0.141[2]-0.14[3]-0.125[2]-0.157 |
| 10 | 10 | 100 | 2 | 200 | 0.14[3]-0.141[2]-0.156 |
| 8 | 8 | 64 | 2 | 200 | 0.14[5]-0.141 |
| 6 | 6 | 36 | 2 | 200 | 0.141[4]-0.14[2] |
| 4 | 4 | 16 | 2 | 200 | 0.141[5]-0.157-0.14 |
| 2 | 2 | 4 | 2 | 200 | 0.141[3]-0.125[2]-0.14 |
| | | | | | |
| 20 | 20 | 400 | 1 | 200 | 0.141[5]-0.14 |
| 18 | 18 | 324 | 1 | 200 | 0.14[5]-0.141 |
| 16 | 16 | 256 | 1 | 200 | 0.125[2]-0.14[2]-0.141[2] |
| 14 | 14 | 196 | 1 | 200 | 0.141-0.14[3]-0.125 |
| 12 | 12 | 144 | 1 | 200 | 0.141[3]-0.14[3]-0.125[2] |
| 10 | 10 | 100 | 1 | 200 | 0.141[4]-0.14 |
| 8 | 8 | 64 | 1 | 200 | 0.14[3]-0.125[2]-0.141[3] |
| 6 | 6 | 36 | 1 | 200 | 0.14[5]-0.125-0.141 |
| 4 | 4 | 16 | 1 | 200 | 0.125[2]-0.141[4] |
| 2 | 2 | 4 | 1 | 200 | 0.141[4]-0.14[2]-0.125 |
| | | | | | |
| 20 | 20 | 400 | 3 | 250 | 0.266-0.218-0.234-0.235-0.219 |
| 18 | 18 | 324 | 3 | 250 | 0.218[2]-0.219[3]-0.234-0.25 |
| 16 | 16 | 256 | 3 | 250 | 0.219[4]-0.234-0.218 |
| 14 | 14 | 196 | 3 | 250 | 0.203[4]-0.218[2] |
| 12 | 12 | 144 | 3 | 250 | 0.203[5]-0.204 |
| 10 | 10 | 100 | 3 | 250 | 0.203[2]-0.218-0.219-0.204-0.187[2] |
| 8 | 8 | 64 | 3 | 250 | 0.204-0.188-0.203[2]-0.25 |

| | | | | | |
|---|---|---|---|---|---|
| 6 | 6 | 36 | 3 | 250 | 0.204-0.218-0.203[5] |
| 4 | 4 | 16 | 3 | 250 | 0.203[7]-0.204-0.218 |
| 2 | 2 | 4 | 3 | 250 | Error division x 0 |
| | | | | | |
| 20 | 20 | 400 | 3 | 300 | 0.282-0.265-0.297[2]-0.281 |
| 18 | 18 | 324 | 3 | 300 | 0.266-0.281[2]-0.297-0.265 |
| 16 | 16 | 256 | 3 | 300 | 0.266[3]-0.25[3]-0.265[2] |
| 14 | 14 | 196 | 3 | 300 | 0.25[3]-0.266[2]-0.281[2]-0.265 |
| 12 | 12 | 144 | 3 | 300 | 0.203-0.219[3]-0.234[3]-0.235[2] |
| 10 | 10 | 100 | 3 | 300 | 0.219[3]-0.235[2]-0.234[3]-0.312 |
| 8 | 8 | 64 | 3 | 300 | 0.234[3]-0.218[2]-0.235-0.219[2] |
| 6 | 6 | 36 | 3 | 300 | 0.188-0.235-0.218-0.203[4] |
| 4 | 4 | 16 | 3 | 300 | 0.203[3]-0.219[3]-0.204[2]-0.218 |
| 2 | 2 | 4 | 3 | 300 | Error division por cero |
| | | | | | |
| 20 | 20 | 400 | 3 | 350 | 0.297-0.265-0.266-0.297-0.344-0.281[2]-0.282 |
| 18 | 18 | 324 | 3 | 350 | 0.266[4]-0.281[3]-0.25-0.265-0.297 |
| 16 | 16 | 256 | 3 | 350 | 0.266[5]-0.282-0.265 |
| 14 | 14 | 196 | 3 | 350 | 0.25[4]-0.266-0.265[3] |
| 12 | 12 | 144 | 3 | 350 | 0.25[5]-0.265-0.266-0.234 |
| 10 | 10 | 100 | 3 | 350 | 0.265-0.25[6] |
| 8 | 8 | 64 | 3 | 350 | 0.266-0.25[6] |
| 6 | 6 | 36 | 3 | 350 | 0.234-0.25[6]-0.266 |
| 4 | 4 | 16 | 3 | 350 | 0.235[2]-0.25[5]-0.234[2] |
| 2 | 2 | 4 | 3 | 350 | Error division por cero |
| | | | | | |
| 20 | 20 | 400 | 3 | 400 | 0.312[3]-0.313[3]-0.297 |
| 18 | 18 | 324 | 3 | 400 | 0.297[4]-0.312[2]-0.313[3] |
| 16 | 16 | 256 | 3 | 400 | 0.297[6] |
| 14 | 14 | 196 | 3 | 400 | 0.328-0.312[2]-0.297[3]-0.281-0.282[2] |
| 12 | 12 | 144 | 3 | 400 | 0.281[7]-0.297[2] |
| 10 | 10 | 100 | 3 | 400 | 0.265-0.233-0.281[5]-0.297[2] |
| 8 | 8 | 64 | 3 | 400 | 0.266-0.281[4]-0.282-0.36 |
| 6 | 6 | 36 | 3 | 400 | 0.282[4]-0.281[2]-0.261-0.297[3] |
| 4 | 4 | 16 | 3 | 400 | 0.281[6]-0.265-0.297[2]-0.282 |
| 2 | 2 | 4 | 3 | 400 | Error division por cero |
| | | | | | |
| 20 | 20 | 400 | 3 | 175 | 0.156[2]-0.141[5]-0.14 |
| 18 | 18 | 324 | 3 | 175 | 0.141[4]-0.156-0.125[2] |
| 16 | 16 | 256 | 3 | 175 | 0.14[3]-0.141[2]-0.125-0.156 |
| 14 | 14 | 196 | 3 | 175 | 0.141[4]-0.125[4]-0.14 |

| | | | | | |
|---|---|---|---|---|---|
| 12 | 12 | 144 | 3 | 175 | 0.125[5]-0.141[2]-0.14 |
| 10 | 10 | 100 | 3 | 175 | 0.125[5]-0.141 |
| 8 | 8 | 64 | 3 | 175 | 0.125[8]-0.14 |
| 6 | 6 | 36 | 3 | 175 | 0.125[5]-0.14 |
| 4 | 4 | 16 | 3 | 175 | 0.125[5]-0.14-0.141 |
| 2 | 2 | 4 | 3 | 175 | Error division por cero |
| | | | | | |
| 20 | 20 | 400 | 3 | 150 | 0.141-01.25[6] |
| 18 | 18 | 324 | 3 | 150 | 0.141-0.125[5]-0.109 |
| 16 | 16 | 256 | 3 | 150 | 0.109[3]-0.125[5]-0.11[2] |
| 14 | 14 | 196 | 3 | 150 | 0.109[6]-0.125[2]-0.11[3] |
| 12 | 12 | 144 | 3 | 150 | 0.11[3]-0.125-0.109[3] |
| 10 | 10 | 100 | 3 | 150 | 0.109[5]-0.125 |
| 8 | 8 | 64 | 3 | 150 | 0.109[3]-0.11[3] |
| 6 | 6 | 36 | 3 | 150 | 0.125[2]-0.109[4]-0.11[3] |
| 4 | 4 | 16 | 3 | 150 | 0.11[2]-0.109[2]-0.094[2]-0.125[2] |
| 2 | 2 | 4 | 3 | 150 | Error division por cero |
| | | | | | |
| 20 | 20 | 400 | 3 | 125 | 0.11[2]-0.109[4]-0.094[3] |
| 18 | 18 | 324 | 3 | 125 | 0.094[2]-0.11-0.109[3] |
| 16 | 16 | 256 | 3 | 125 | 0.109[2]-0.093[3]-0.094[2] |
| 14 | 14 | 196 | 3 | 125 | 0.093[3]-0.094[3]-0.11 |
| 12 | 12 | 144 | 3 | 125 | 0.078-0.093-0.109-0.094[4] |
| 10 | 10 | 100 | 3 | 125 | 0.094[7]-0.079 |
| 8 | 8 | 64 | 3 | 125 | 0.093-0.094[3]-0.078[2] |
| 6 | 6 | 36 | 3 | 125 | 0.094[6]-0.078 |
| 4 | 4 | 16 | 3 | 125 | 0.094[5]-0.078[2]-0.093 |
| 2 | 2 | 4 | 3 | 125 | Error division por cero |
| | | | | | |
| 20 | 20 | 400 | 3 | 100 | 0.094[2]-0.093-0.078[2]-0.079 |
| 18 | 18 | 324 | 3 | 100 | 0.079-0.078[6] |
| 16 | 16 | 256 | 3 | 100 | 0.079-0.078[5] |
| 14 | 14 | 196 | 3 | 100 | 0.062[2]-0.078[4]-0.063-0.079 |
| 12 | 12 | 144 | 3 | 100 | 0.063[3]-0.078[2]-0.062[2] |
| 10 | 10 | 100 | 3 | 100 | 0.063[4]-0.062-0.093-0.078 |
| 8 | 8 | 64 | 3 | 100 | 0.063[4]-0.078[2]-0.062[2] |
| 6 | 6 | 36 | 3 | 100 | 0.0062[3]-0.078[3]-0.063[2] |
| 4 | 4 | 16 | 3 | 100 | 0.078-0.063[2]-0.062[3] |
| 2 | 2 | 4 | 3 | 100 | Error division por cero |
| | | | | | |
| 20 | 20 | 400 | 3 | 75 | 0.062[4]-0.063[2]-0.078 |
| 18 | 18 | 324 | 3 | 75 | 0.078-0.062[2]-0.047-0.063-0.093 |

| | | | | | |
|---|---|---|---|---|---|
| 16 | 16 | 256 | 3 | 75 | 0.062[3]-0.063[3]-0.047 |
| 14 | 14 | 196 | 3 | 75 | 0.047[6]-0.042 |
| 12 | 12 | 144 | 3 | 75 | 0.046-0.047[6] |
| 10 | 10 | 100 | 3 | 75 | 0.063-0.046-0.047[2]-0.062[3] |
| 8 | 8 | 64 | 3 | 75 | 0.047[5]-0.062[2]-0.046 |
| 6 | 6 | 36 | 3 | 75 | 0.046[3]-0.047[5] |
| 4 | 4 | 16 | 3 | 75 | 0.047[2]-0.046-0.062[2]-0.063[2]-0.078 |
| | | | | | |
| 2 | 2 | 4 | 3 | 75 | Error division por cero |
| 20 | 20 | 400 | 3 | 50 | 0.047[2]-0.031[3]-0.032-0.061 |
| 18 | 18 | 324 | 3 | 50 | 0.047[6]-0.031 |
| 16 | 16 | 256 | 3 | 50 | 0.032-0.047[4]-0.031[3] |
| 14 | 14 | 196 | 3 | 50 | 0.031[3]-0.047[3]-0.032-0.046 |
| 12 | 12 | 144 | 3 | 50 | 0.032-0.031[4]-0.047 |
| 10 | 10 | 100 | 3 | 50 | 0.032[2]-0.031[5]-0.047 |
| 8 | 8 | 64 | 3 | 50 | 0.047[5]-0.031[2]-0.032 |
| 6 | 6 | 36 | 3 | 50 | 0.047[2]-0.031[5]-0.032 |
| 4 | 4 | 16 | 3 | 50 | 0.031[6]-0.032-0.047 |
| 2 | 2 | 4 | 3 | 50 | Error division por cero |
| | | | | | |
| 20 | 20 | 400 | 3 | 25 | 0.032-0.031[3]-0.016[4]-0.015 |
| 18 | 18 | 324 | 3 | 25 | 0.031[4]-0.025-0.032 |
| 16 | 16 | 256 | 3 | 25 | 0.016[5]-0.015-0.031 |
| 14 | 14 | 196 | 3 | 25 | 0.016[4]-0.032[2] |
| 12 | 12 | 144 | 3 | 25 | 0.032-0.016[4]-0.015-0.031 |
| 10 | 10 | 100 | 3 | 25 | 0.016[3]-0.015[3]-0.031-0.032 |
| 8 | 8 | 64 | 3 | 25 | 0.016[3]-0.015[3]-0.032-0.031 |
| 6 | 6 | 36 | 3 | 25 | 0.016[3]-0.015[4]-0.031 |
| 4 | 4 | 16 | 3 | 25 | 0.015[4]-0.016[3]-0.031 |
| 2 | 2 | 4 | 3 | 25 | Error division por cero |
| | | | | | |
| 20 | 20 | 400 | 20 | 400 | 0.547-0.594-0.609 |
| 18 | 18 | 324 | 20 | 400 | 0.516-0.531 |
| | | | | | |
| 20 | 20 | 400 | 20 | 300 | 0.391-0.469-0.422 |
| 20 | 20 | 400 | 20 | 200 | 0.344-0.359 |
| 20 | 20 | 400 | 20 | 200 | 0.313-0.281 |
| 20 | 20 | 400 | 20 | 100 | 0.125[4]-0.14-0.141 |
| 20 | 20 | 400 | 20 | 25 | 0.031[2]-0.032[2]-0.046-0.047 |
| | | | | | |
| 18 | 18 | 324 | 20 | 25 | 0.031[3]-0.047[2] |

| | | | | | |
|---|---|---|---|---|---|
| 16 | 16 | 256 | 20 | 25 | 0.032-0.047 |
| | | | | | |
| 20 | 20 | 400 | 18 | 400 | 0.531-0.625 |
| 20 | 20 | 400 | 15 | 400 | 0.5[2]-0.532-0.515 |
| 18 | 18 | 324 | 15 | 400 | 0.516-0.578 |
| | | | | | |
| 20 | 20 | 400 | 18 | 300 | 0.391-0.453 |
| 20 | 20 | 400 | 18 | 200 | 0.281-0.297 |
| 20 | 20 | 400 | 18 | 100 | 0.14-0.125 |
| 20 | 20 | 400 | 18 | 25 | 0.031[3]-0.032 |
| 18 | 18 | 324 | 18 | 25 | 0.032[2]-0.031[2] |
| 16 | 16 | 256 | 18 | 25 | 0.031-0.032 |
| | | | | | |
| 20 | 20 | 400 | 15 | 300 | 0.406-0.456 |
| 20 | 20 | 400 | 15 | 200 | 0.265-0.281 |
| 20 | 20 | 400 | 15 | 100 | 0.109-0.125[3] |
| 20 | 20 | 400 | 15 | 25 | 0.031-0.047 |
| | | | | | |
| 20 | 20 | 400 | 10 | 400 | 0453-0.437 |
| 20 | 20 | 400 | 10 | 300 | 0.297-0.281-0.312-0.344 |
| 20 | 20 | 400 | 10 | 200 | 0.218-0.203[2] |
| 20 | 20 | 400 | 10 | 100 | 0.109-0.11 |
| 20 | 20 | 400 | 10 | 25 | 0.031[2]-0.016 |
| | | | | | |
| 20 | 20 | 400 | 5 | 400 | 0.359-0.375[2]-0.422 |
| 20 | 20 | 400 | 5 | 300 | 0.265[2]-0.266[2] |
| 20 | 20 | 400 | 5 | 200 | 0.188-0.172-0.203-0.187 |
| 20 | 20 | 400 | 5 | 100 | 0.078[3]-0.094[3] |
| 20 | 20 | 400 | 5 | 25 | 0.016-0.031[2]-0.032 |
| | | | | | |
| 20 | 20 | 400 | 6 | 400 | 0.391-0.375[2]-0.39-0.406 |
| 20 | 20 | 400 | 7 | 400 | 0.36-0.345 |
| 18 | 18 | 324 | 7 | 400 | 0.36-0.359-0.361 |
| 16 | 16 | 256 | 7 | 400 | 0.343-0.36-0.375 |
| | | | | | |
| 20 | 20 | 400 | 8 | 400 | 0.391-0.375-0.406-0.422 |
| 18 | 18 | 324 | 8 | 400 | 0.344[2]-0.375-0359 |
| 16 | 16 | 256 | 8 | 400 | 0.328-0.359-0.343 |
| 14 | 14 | 196 | 8 | 400 | 0.296-0.313[2]-0.313[2] |
| 12 | 12 | 144 | 8 | 400 | 0.328-0.312[3]-0.359-0.391 |
| | | | | | |
| 20 | 20 | 400 | 9 | 400 | 0.484-0.516 |

| | | | | | |
|---|---|---|---|---|---|
| | | | | | |
| 20 | 20 | 400 | 6 | 300 | 0.25-0.297-0.281[2] |
| 18 | 18 | 324 | 6 | 300 | 0.25-0.234-0.266-0.296 |
| 20 | 20 | 400 | 7 | 300 | 0.265-0.281-0.297 |
| 20 | 20 | 400 | 8 | 300 | 0.282-0.281-0.296 |
| 18 | 18 | 324 | 8 | 300 | 0.234-0.25[3] |
| 16 | 16 | 256 | 8 | 300 | 0.265-0.25 |
| 14 | 14 | 196 | 8 | 300 | 0.25-0.234 |
| 20 | 20 | 400 | 9 | 300 | 0.281-0.297[2]0.313 |
| 18 | 18 | 324 | 9 | 300 | 0.266-0.281 |
| | | | | | |
| 20 | 20 | 400 | 6 | 200 | 0.172[2]-0.219 |
| 20 | 20 | 400 | 7 | 200 | 0.156[3]-0.172[2] |
| 18 | 18 | 324 | 7 | 200 | 0.188-0.203 |
| 20 | 20 | 400 | 8 | 200 | 0.188[2]-0.187-0.171 |
| 18 | 18 | 324 | 8 | 200 | 0.188 |
| 20 | 20 | 400 | 9 | 200 | 0.219-0.203 |
| 18 | 18 | 324 | 9 | 200 | 0.172[4]-0.156 |
| 16 | 16 | 256 | 9 | 200 | 0.188-0.171-0.219-0.235 |
| | | | | | |
| 20 | 20 | 400 | 6 | 100 | 0.094-0.093-0.071 |
| 18 | 18 | 324 | 6 | 100 | 0.094 |
| 20 | 20 | 400 | 7 | 100 | 0.078[4] |
| 18 | 18 | 324 | 7 | 100 | 0.078-0.079-0.094 |
| 16 | 16 | 256 | 7 | 100 | 0.078[4]-0.094 |
| 14 | 14 | 196 | 7 | 100 | 0.078[4]-0.063 |
| 12 | 12 | 144 | 7 | 100 | 0.063[3]-0.062-0.079 |
| 10 | 10 | 100 | 7 | 100 | 0.078 |
| 20 | 20 | 400 | 8 | 100 | 0.094-0.109 |
| 20 | 20 | 400 | 9 | 100 | 0.094-0.109-0.11 |
| | | | | | |
| 20 | 20 | 400 | 6 | 25 | 0.031[2]-0.016-0.032 |
| 18 | 18 | 324 | 6 | 25 | 0.031[3]-0.016-0.015 |
| 16 | 16 | 256 | 6 | 25 | 0.031-0.015 |
| 20 | 20 | 400 | 7 | 25 | 0.016-0.015-0.031 |
| 18 | 18 | 324 | 7 | 25 | 0.016[3]-0.031 |
| 16 | 16 | 256 | 7 | 25 | 0.031[2]-0.032-0.015 |
| 14 | 14 | 196 | 7 | 25 | 0.016[3]-0.032 |
| 12 | 12 | 144 | 7 | 25 | 0.015[2] |
| 20 | 20 | 400 | 8 | 25 | 0.031[3]-0.032 |
| 18 | 18 | 324 | 8 | 25 | 0.031-0.016 |
| 20 | 20 | 400 | 9 | 25 | 0.031 |

| | | | | | |
|---|---|---|---|---|---|
| 18 | 18 | 324 | 9 | 25 | 0.031-0.032-0.016 |
| 16 | 16 | 256 | 9 | 25 | 0.031-0.031-0.015 |
| 14 | 14 | 196 | 9 | 25 | 0.031[4]-0.015 |
| 12 | 12 | 144 | 9 | 25 | 0.016-0.031 |
| 10 | 10 | 100 | 9 | 25 | 0.016-0.032 |
| 8 | 8 | 64 | 9 | 25 | 0.016-0.015-0.031-0.032 |
| 6 | 6 | 36 | 9 | 25 | 0.032-0.015-0.031[3] |
| 4 | 4 | 16 | 9 | 25 | 0.015-0.031-0.016 |
| 2 | 2 | 4 | 9 | 25 | Error division por cero |
| | | | | | |
| 20 | 20 | 400 | 6 | 75 | 0.078 |
| 18 | 18 | 324 | 6 | 75 | 0.063-0.078 |
| 16 | 16 | 256 | 6 | 75 | 0.078 |
| 20 | 20 | 400 | 7 | 75 | 0.078-0.079 |
| 18 | 18 | 324 | 7 | 75 | 0.078 |
| 16 | 16 | 256 | 7 | 75 | 0.063-0.078 |
| 14 | 14 | 196 | 7 | 75 | 0.062-0.063 |
| 12 | 12 | 144 | 7 | 75 | 0.078-0.047 |
| | | | | | |
| 20 | 20 | 400 | 8 | 50 | 0.062-0.063 |
| 18 | 18 | 324 | 8 | 50 | 0.047 |
| 20 | 20 | 400 | 9 | 50 | 0.047-0.062 |
| 18 | 18 | 324 | 9 | 50 | 0.047-0.063 |
| 16 | 16 | 256 | 9 | 50 | 0.047-0.063 |
| 14 | 14 | 196 | 9 | 50 | 0.047 |
| 12 | 12 | 144 | 9 | 50 | 0.031-0.047 |
| 10 | 10 | 100 | 9 | 50 | 0.047-0.031 |
| 8 | 8 | 64 | 9 | 50 | 0.047 |
| 6 | 6 | 36 | 9 | 50 | 0.047-0.031 |
| 4 | 4 | 16 | 9 | 50 | 0.031-0.046-0.047 |
| 2 | 2 | 4 | 9 | 50 | Error division por cero |
| | | | | | |
| 25 | 25 | 625 | 3 | 200 | 0.187[2]-0.203[2] |
| | | | | | |
| 30 | 30 | 900 | 10 | 200 | 0.265-0.266-0.282 |
| 35 | 35 | 1225 | 10 | 200 | 0.406-0.391 |
| | | | | | |
| 50 | 50 | 2500 | 20 | 200 | 1.14 [3] |
| 50 | 50 | 2500 | 10 | 200 | 0.672[3] |
| 50 | 50 | 2500 | 15 | 200 | 0.828-0.84 |
| 50 | 50 | 2500 | 18 | 200 | 0.969-1.06 |
| 50 | 50 | 2500 | 23 | 200 | 1.078-1.016-1.06-1012 |

| | | | | | |
|---|---|---|---|---|---|
| 50 | 50 | 2500 | 25 | 200 | 1.14-1.17 |
| | | | | | |

**Tabla 2. Tiempos obtenidos para los Mapas de Kohonen.**

**Nota:** Los valores presentes en el campo corresponden a las diferentes pruebas realizadas. Los valores de tiempo que poseen un número entero entre corchetes significan que ese número entero fueron las veces que se obtuvieron repetidamente esos resultados. (Ver epígrafe 2.5 para una mejor comprensión de estos resultados).

# Glosario de Términos

**A**

**Adaline:** Tipo de red neuronal. Está compuesto por una sola neurona.

**Adaptación:** Capacidad de las redes neuronales de adaptarse a nuevas circunstancias.

**Agente:** Un agente no es más que un agente autónomo

**Agentes autónomos:** Un agente autónomo es un individuo dentro de la aplicación que se caracteriza por su proactividad y persistencia.

**Aprendizaje:** Proceso similar al entrenamiento, pero que implica el reconocimiento de nuevos patrones.

**Arquitectura:** Se refiere al número de capas que va a poseer un modelo de red neuronal y el número de neuronas en cada capa.

**ART**: Siglas en inglés del modelo de red neuronal de la Teoría de la Resonancia Adaptativa.

**Autorecurrente:** Se dice que una red es autorecurrente si posee al menos una neurona que se conecta con ella misma.

**B**

**Backpropagation:** Se conoce así a un método de aprendizaje supervisado por corrección del error. También a los perceptrones multicapa que utilizan este método de entrenamiento.

**E**

**Entrenamiento:** Proceso mediante el cual las redes neuronales llevan sus pesos hasta valores capaces de responder para los juegos de datos del entrenamiento, las respuestas deseadas para un error mínimo.

**F**

**Feedback:** Se refiere con este término a las redes que tienen conexiones hacia atrás.

**Feedforward:** Se refiere con este término a las redes que tienen todas sus conexiones hacia delante.

**Función de activación:** Función matemática utilizada por las neuronas artificiales para transformar las entradas de la red en salidas.

**H**

**Hopfield:** Modelo de red neuronal recurrente de una sola capa de neuronas.

**M**

**Madaline:** Conjunto de redes Adaline.

**Mapas de Kohonen:** Modelo de Red Neuronal Artificial, conocido como Mapas Auto-organizados, desarrollado por Kohonen.

**N**

**Neurona:** Neurona básica de una Red Neuronal Artificial, que simula algunas de las funcionalidades de las neuronas biológicas.

**NPC:** Non Player Character, en español: personaje no jugador.

**O**

**Offline:** Se refiere al entrenamiento que se realiza cuando la red no está ejecutándose.

**Online:** Se refiere al entrenamiento que se realiza cuando la red está ejecutándose.

**P**

**Persistencia:** Capacidad de los agentes autónomos de conservar la información aunque no se muestre en pantalla.

**Proactividad:** Función que permite que los agentes autónomos actúen en función de sus propios intereses.

**PMC:** Perceptrón Multicapa.

**R**

**RBF:** Modelo de red neuronal conocido como Función de Radial Básico.

**Realidad Virtual:** Se refiere a la técnica que trata de modelar ambientes reales en las computadoras.

**Recurrente:** Se dice que una red es recurrente si esta se retroalimenta de sus salidas.

**Red Neuronal:** Intenta simular a un conjunto de neuronas biológicas del cerebro. Esta compuesta por varias neuronas artificiales agrupadas por capas.
**Resonancia:** Es la función que hace coincidir un patrón de entrada a un modelo ART con las categorías que tiene este almacenado.

**RNA:** Red Neuronal Artificial.

**T**

**Topología:** Se refiere al número de capas que va a poseer un modelo de red neuronal y el número de neuronas en cada capa.

1