UNIVERSIDAD DEL VALLE DE GUATEMALA

FACULTAD DE CIENCIAS Y HUMANIDADES

NEUROTRÓN: RED NEURAL PARA DIFERENCIACIÓN DE CLASES

GUATEMALA

2001

NEUROTRÓN: RED NEURAL

PARA DIFERENCIACIÓN DE CLASES

UNIVERSIDAD DEL VALLE DE GUATEMALA

FACULTAD DE CIENCIAS Y HUMANIDADES

NEUROTRÓN: RED NEURAL PARA DIFERENCIACIÓN DE CLASES

CARLOS ENRIQUE CASTILLO GALINDO

TRABAJO DE INVESTIGACIÓN PRESENTADO

PARA OPTAR AL GRADO ACADÉMICO DE

LICENCIADO EN INGENIERÍA EN CIENCIAS DE LA COMPUTACIÓN

GUATEMALA

2001

Vo.Bo.:

(f) _____
Ing. Luis Furlán Collver

Tribunal:

(f) _____
Ing. Luis Furlán Collver

(f) _____
Ing. Martha Ligia Naranjo

(f) _____
Ing. María Mercedes Zaghi

Fecha de aprobación: 23 de julio de 2001

PREFACIO

La aparición de las computadoras digitales y el desarrollo de las modernas teorías sobre el aprendizaje y el procesamiento neuronal, se produjeron aproximadamente al mismo tiempo, durante los últimos años de la década de los cuarenta.

A partir de ese momento, las computadoras digitales han sido utilizadas como herramientas para modelar el funcionamiento de neuronas individuales, así como agrupaciones de éstas en las así llamadas **Redes Neurales** o **Redes Neuronales**.

Es amplia la bibliografía donde se explica lo conocido sobre el funcionamiento de las neuronas reales, y también es considerable la cantidad de libros y artículos que abordan el tema desde todos los puntos de vista concebibles.

Debido a esta amplitud palpable, todos aquellos investigadores que han intentado profundizar en el funcionamiento de las redes neuronales, están en la obligación de limitar su estudio a ciertas características notables de ellas, las cuales quedan a criterio del investigador.

Uno de los modelos existentes en la actualidad es el llamado *Perceptrón*, cuya función es la de implementar una red neuronal, basada en un modelo

matemático desarrollado bajo la técnica de ajuste de pesos, y el aprendizaje mediante iteraciones finitas de entrenamiento.

El *Perceptrón* es un modelo matemático que simula una red neuronal de un nivel o capa, el cual le permite diferenciar únicamente entre dos clases de objetos al mismo tiempo y posee un margen de error generado por sus propias limitaciones de diseño.

El objetivo de este trabajo, lo constituye entonces, el desarrollo y prueba de un nuevo modelo al que se llamará ***Neurotrón***, el cual será una red neuronal especializada en la diferenciación de letra escrita, cuya principal característica será la de minimizar el error que el modelo *Perceptrón* muestra en su funcionamiento.

Como prueba de funcionamiento, al final del trabajo se muestra el resultado de la aplicación del modelo ***Neurotrón*** a un caso específico, comparando los resultados con los generados por el *Perceptrón*.

CONTENIDO

PREFACIO IV

I. INTRODUCCIÓN 7

II. REDES NEURALES 10

a) Generalidades 10

b) Memoria y Aprendizaje 12

c) Las Redes Neuronales como Modelos 15

d) Simulación de Redes Neuronales 18

III. EL NUEVO MODELO: *EL NEUROTRÓN* 22

a) Conceptualización 22

b) Representación Matemática 29

c) Aplicación desarrollada 35

IV. RESULTADOS 44

Tabla 1. Mejor función de ajuste de valores de salida $d(t) = 1$ 45

Tabla 2. Mejor función de ajuste de valores de salida $d(t) = -1$ 46

Tabla 3. Comparación de resultados **Neurotrón**, *Perceptrón* 48

V. CONCLUSIONES 50

VI. BIBLIOGRAFÍA 52

VII. APÉNDICES 53

a. Código fuente de aplicación desarrollada 53

b. Muestra de dígitos para clase del **número 3** 65

c. Muestra de dígitos para clase del **número 8** 68

I. INTRODUCCIÓN

En la actualidad se ha experimentado un crecimiento en el estudio sobre temas relacionados con el campo de la Inteligencia Artificial, siendo uno de ellos el de las Redes Neurales o Neuronales.

Como parte de uno de esos temas, en el presente trabajo de investigación se ha desarrollado un nuevo modelo de red neuronal, el cual fue llamado Neurotrón, y que persigue, desde el punto de vista científico, mostrar la capacidad de este tipo de modelos en la simulación de uno de los aspectos característicos de la mente humana: el aprendizaje y reconocimiento de clases.

Desde el punto de vista tecnológico, el trabajo intenta iniciar el camino de la investigación sobre este tipo de temas en un ambiente en el cual aún no se ha desarrollado completamente, así como proveer de un nuevo modelo cuya principal característica lo constituya un margen de certeza mayor, comparado con un modelo ya existente de nombre Perceptrón, que como ya se mencionó, es un modelo matemático que simula una red neuronal para diferenciar únicamente entre dos clases de objetos.

Los modelos de redes neuronales, una vez probados y delimitado su alcance, son objeto de constantes evaluaciones, previo a su aplicación final en la vida real (Freeman, 1993:55). Generalmente, estos modelos han sido desarrollados con la idea de simular el pensamiento humano, para probarlos con aplicaciones específicas.

Dos de los campos, dentro de los cuales los modelos de redes neuronales han incursionado, son el de la Inteligencia Artificial y el de la Genética Virtual. En estos campos, las aplicaciones resultan ser variadas, debido principalmente a que el objetivo primordial de las áreas mencionadas es el de la simulación.

Dentro del campo de la Inteligencia Artificial se puede encontrar, entre otras, la simulación del razonamiento en la toma de decisiones, relacionada con la selección de un camino a tomar desde dos puntos determinados, entre los cuales existen señales. El modelo aprende el significado de las señales mediante entrenamiento, de tal manera que al encontrar una parecida, se tomará la decisión más adecuada.

En el campo de la Genética Virtual, se han desarrollado modelos que simulan un ecosistema, para poder observar el desarrollo de especies ficticias en diferentes ambientes, y poder estimar el cambio o mutación de genes según diferentes condiciones.

En el reconocimiento de clases, el diseño del nuevo modelo, El Neurotrón, se basará en los principios establecidos por el modelo existente llamado Perceptrón. En la actualidad, existen grupos de investigación desarrollando trabajos basados en este tipo de herramientas, con la idea de simular el pensamiento humano (Kumar, 1999:50), lo que hace necesaria una investigación profunda sobre los temas de simulación y reconocimiento de patrones, entre otros.

El Neurotrón pretende ser una de esas aplicaciones, orientada a la investigación científica, que muestre el interés en nuestro medio por profundizar en el tema.

II. REDES NEURALES

a) Generalidades

Dentro del cerebro humano podemos encontrar billones de tipos de células llamadas neuronas o células nerviosas. Estas células están organizadas en redes complicadas, con la habilidad de comunicarse entre sí, por medio de impulsos eléctricos (Kumar, 1999:75).

Generalmente, una célula nerviosa está físicamente conectada a otras células similares en cantidades de miles de millones. Por medio de estas conexiones, las células nerviosas son capaces de transmitir señales eléctricas entre ellas y crear un estado de excitación en la red que forman (Winston, 1994:185).

Estas conexiones no están precisamente encendidas o apagadas, pero se puede decir que poseen cargas variables o estados de excitación, las cuales permiten que su influencia hacia las otras neuronas interconectadas sea fuerte, débil, ninguna, o cualquier otra variación entre estos estados[1].

---

[1] Se facilita la comprensión de este texto si se asume que una red neuronal no es más que un circuito eléctrico donde cada uno de los componentes se le llama *neurona*, con sus interconexiones físicas correspondientes y que al igual que en un circuito eléctrico, permite el tránsito de cargas eléctricas con un fin específico. El cerebro es un ejemplo de un circuito complicado cuyas finalidades son múltiples, pero el concepto básico es el mismo.

Muchos aspectos del funcionamiento del cerebro humano (risa, llanto, recuerdos, aprendizaje, etc.), están en relación directa con el grado de influencia entre neuronas. Es permitido entonces representar la actividad del cerebro, por medio de patrones particulares de excitación de las neuronas en la red que forman[2].

[2] El cerebro humano sigue siendo uno de los más grandes enigmas aún no comprendidos en su totalidad por el ser humano. La lógica y la ciencia realizan inmensos esfuerzos orientados a comprender su funcionamiento, y en este caso particular, desde filósofos, científicos e incluso religiosos han incursionado en un viaje cuyo puerto final es el de descifrar y emular su funcionamiento, motivados tal vez, por el natural sentido de curiosidad que acompaña al hombre en su andar por el camino del desarrollo y progreso.

b) Memoria y Aprendizaje

Una de las funciones más importantes del cerebro, es el almacenamiento y recuperación de datos en su memoria. Presenta cierta dificultad, por no decir imposible de visualizar, el hecho de imaginar al cerebro sin el manejo de recuperación de memoria a corto y largo plazo (Guillian, 1997:93).

Por ejemplo, el hecho de no contar con memoria a largo plazo significaría que no sería posible el aprender de experiencias pasadas. De hecho, nuestro pensamiento sobre nosotros mismos se basa en el recuerdo de nuestra propia historia y experiencia pasada.

Se puede decir, entonces que, de igual forma, si nuestro cerebro careciera de la habilidad para manejar memoria a corto plazo, seríamos testigos de momentos puntuales de nuestra historia, sin ninguna conexión lógica entre ellos.

Nuestras memorias (o experiencias) funcionan entonces de manera asociativa o con contenido direccionable (Guillian, 1997:97). Esto quiere decir que una memoria no está de ninguna manera aislada de las demás experiencias o memorias que, a su vez, están almacenadas en algún grupo particular de células nerviosas o neuronas.

De esto puede definirse que debido a su contenido direccionable aparente, todas las memorias son de alguna manera cadenas interconectadas de memorias.

Por ejemplo, al recordar a una persona, se realiza un proceso que ubica a la persona en particular, mediante alguna de estas formas: por el color de los ojos o del cabello, por la forma de su nariz, la altura, el sonido de la voz, o tal vez por el aroma de su perfume favorito. Esto implica que las memorias son almacenadas preservando una relación entre unas y otras.

Según estudios científicos, las unidades sensoriales encargadas de almacenar las memorias están distribuidas en todo el cerebro humano (Guillian, 1997:99). De hecho, en experimentos recientes se ha demostrado que en el proceso de recordar memorias, se ha detectado actividad sensorial en todo el cerebro (Guillian, 1997:99).

Es importante hacer notar que en el cerebro es posible encontrar una ruta que permita llegar a la memoria en su totalidad, por ejemplo, todos los aspectos de la descripción de una persona, mediante el recuerdo de únicamente una o dos de sus características (Lippmann, 1992:105).

Se puede deducir, entonces, que el cerebro consulta a la memoria por su contenido y no por el lugar físico, mediante variaciones de cargas eléctricas, donde estén almacenados dentro de la red de neuronas en el cerebro. El proceso de recuperación de memoria ha demostrado ser tan poderoso que, al

observar la fotografía borrosa de una persona, se es capaz de reconstruir con cierto grado de certeza, las características específicas del rostro de esta persona.

Por otro lado existe una gran diferencia de funcionamiento si se compara con el funcionamiento de una computadora tradicional, donde memorias específicas son almacenadas en lugares igualmente específicos dentro de la memoria de la máquina. Es de hacer notar que si sólo se cuenta con información parcial sobre este lugar específico, la memoria no puede ser recuperada en su totalidad.

c) Las Redes Neuronales como Modelos

Las redes neuronales artificiales son modelos de computadora cuya arquitectura es diseñada con el objetivo principal de la simulación del cerebro humano (Nilsson, 1986:270). Típicamente están formadas por cientos de simples unidades de proceso interconectadas entre sí, con la idea de formar una red de comunicaciones compleja entre procesadores.

Cada nodo[3] en la red es un modelo simplificado de una neurona real, cuya principal función es la de enviar señales nuevas hacia los nodos a los que está interconectado, sólo si recibe de entrada un estímulo suficientemente fuerte, proveniente de otros nodos en la red a los cuales está conectado.

La fuerza de estas conexiones[4] varía en función del propósito de la red neuronal simulada, es decir, la función de estas conexiones está diseñada para permitir que la red lleve a cabo distintas tareas, dependiendo de los diferentes patrones de excitación en la estructura.

Por otra parte, las computadoras tradicionales, con las cuales se interactúa en la actualidad, conceptualmente no han variado significativamente desde sus inicios en los años 40.

---

[3] El concepto Nodo en este contexto, debe comprenderse como el punto de conexión entre dos o más líneas de comunicación, por lo que se puede hacer la analogía entre un Nodo y una Neurona cerebral.

[4] La fuerza de conexión puede encontrar su analogía con la intensidad y la frecuencia con la que dos componentes eléctricos reciben impulsos eléctricos mientras se encuentra en funcionamiento el circuito del que forman parte.

Al mismo tiempo se han observado avances substanciales en el tamaño y velocidad de los transistores de silicona que son parte del elemento básico para construir el componente físico de un computador (hardware), no obstante, se debe hacer notar que el diseño generalizado a nivel de arquitectura no ha cambiado al mismo ritmo.

Desde este punto de vista, las computadoras aún consisten de una Unidad Central de Proceso (CPU), que secuencialmente ejecuta un grupo de instrucciones rígidas llamadas programas o software, leyendo y escribiendo datos en una unidad separada llamada memoria (Stewart, 1996:43).

Toda la inteligencia de las computadoras está, entonces, en función de este conjunto de reglas rígidas que al final son proveídas por el elemento humano: el programador. Esto nos lleva hacia la percepción de las computadoras, como un modelo que con el tiempo se acerca a una supermáquina, pero no a una mente.

Las redes neuronales son diferentes al modelo tradicional que representa una computadora. Como está indicado anteriormente, este tipo de redes está formado por unidades individuales de proceso, conectadas entre sí para formar una red.

De esta forma, su poder computacional o de cálculo, está en función de la habilidad de estos componentes para trabajar en equipo, incluyendo la capacidad de realizar tareas y procesos en forma paralela.

A diferencia de las computadoras tradicionales, no está centralizada en un CPU que ejecute una secuencia lógica de reglas o comandos, es más, dichas reglas ni siquiera están definidas en el propio modelo de la red neuronal.

Finalmente, los cálculos computacionales en el modelo de red, están por definición, relacionados con el proceso dinámico de excitación entre nodos, haciendo que esta estructura sea una aproximación más cercana a la realidad del funcionamiento del cerebro humano, permitiendo además, crear un nuevo tipo de computadora con la habilidad de ser útil en la realización de tareas complejas, semejantes a las realizadas por el cerebro.

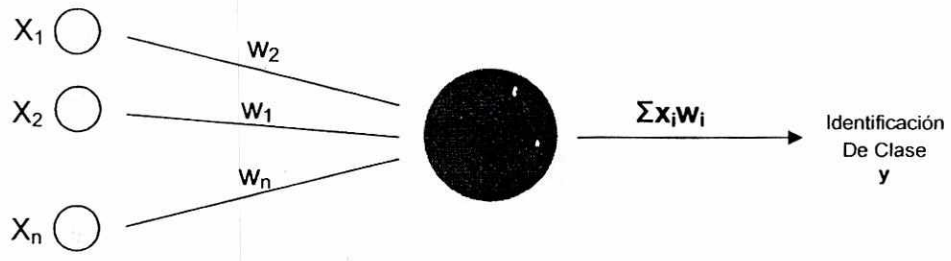
d) Simulación de Redes Neuronales

A los modelos de redes neuronales artificiales, o simplemente redes neuronales, comúnmente se los conoce con distintos nombres: Modelos Conexionistas, Modelos de Procesamiento Paralelo o Sistemas Neuromórficos (Guillian, 1997:105).

De cualquier forma como se llamen, todos estos modelos persiguen obtener un buen rendimiento, basados en conexiones densas de elementos computacionales simples.

Para citar un ejemplo de simulación, puede pensarse en una red simple como la representada en la figura 2.1, donde el nodo mas sencillo dentro de la red, almacena un valor ponderado (x_1, x_2, ... , x_n), calculado como la suma ponderada de $\mathbf{x_n}$ datos de entrada.

Estos valores ponderados deben interpretarse como el estímulo o grado de excitación del nodo, el cual será transmitido hacia otro u otros nodos dentro de la estructura a la que se está interconectado.



Funcionamiento de un nodo dentro de una Red Neuronal
Figura 2.1

En la figura, los nodos más pequeños, localizados a la izquierda de la figura 2.1, están conectados hacia el nodo más grande, localizado hacia el centro de la misma figura.

El nodo grande es estimulado por las ***n*** entradas provenientes de los nodos pequeños, con el fin de producir un resultado final, como indicador del proceso del "aprendizaje-reconocimiento"[5] de dicha estructura.

Este indicador debe ser interpretado en función del propósito para el cual fue creada la red neuronal. Por ejemplo, una red neuronal puede diseñarse para ser capaz de diferenciar entre los dígitos "0" y "8"; y según el diseño de la red, el indicador podría presentar un valor negativo cuando se reconozca el dígito "0", y un valor positivo al reconocer el dígito "8".

Los grados de excitación de los nodos de una red, como fue mencionado anteriormente, son el resultado de la ponderación de una o varias características en los casos más sencillos, pero debe aclararse que nodos más complejos, pueden incluir dentro de sus cálculos, operaciones matemáticas más complejas o incluir características más detalladas.

---

[5] El proceso de *aprendizaje-reconocimiento* se debe entender como el proceso mediante el cual el modelo representado en la figura 2.1 ajusta los valores de las *n* entradas a las que se encuentra conectado el nodo central, a lo que debe llamarse *aprendizaje*, para luego, con esos valores ajustados, probar el funcionamiento del modelo, a lo que debe llamarse *reconocimiento*. Es común, desde el punto de vista técnico, unir estos conceptos cuando se habla de redes neuronales para representar un proceso continuo que en algunos casos es paralelo. Este proceso pude encontrar su analogía en lo que comúnmente conocemos como prueba y error.

De nuevo, las características ponderadas están en función del propósito para el cual fue creada la red neuronal. Si se está reconociendo letra escrita, por ejemplo, una de las características consideradas podría ser la altura de los caracteres dibujados.

Una red neuronal como tal, está completamente especificada por las siguientes características (Freeman, 1993:105):

a. La topología de la red.
b. Las características de sus nodos.
c. El entrenamiento, y
d. Las reglas de aprendizaje

Estas reglas de aprendizaje sugieren un conjunto de valores iniciales para los pesos w_i, utilizados para ponderar las características seleccionadas, e indican el procedimiento a seguir para adaptar los mencionados pesos, durante la etapa del entrenamiento.

En general, los procedimientos a seguir y las reglas de aprendizaje son actualmente el objeto de investigaciones profundas, debido a la naturaleza específica de las aplicaciones sobre las cuales son utilizadas las redes neuronales.

Concretamente, se presta mucha atención en el conjunto de parámetros y variables a considerar, en la creación y definición de modelos matemáticos diseñados para simular una red neuronal.

III. EL NUEVO MODELO: EL NEUROTRÓN

a) Conceptualización

El Neurotrón se ha creado como una red neuronal artificial simple, diseñada para ser capaz de aprender a reconocer dos tipos diferenciables de clases, con un margen de error menor al compararlo con el modelo ya existente llamado Perceptrón.

El Neurotrón se basa en un procedimiento similar al utilizado por el cerebro humano, es decir por medio de memorias o patrones. Sin embargo, para lograr la minimización del error comparado con el margen reportado por el Perceptrón, se han tenido que definir restricciones que el mencionado modelo no posee, las cuales serán mencionadas posteriormente.

El término clase debe interpretarse como un concepto genérico que se refiere a un objeto como tal, junto con sus variantes. Un ejemplo de clase, es el conjunto formado por las diferentes formas en que se puede escribir el número "1", a la cual puede referirse como *la clase del número 1*.

El Neurotrón como modelo, cuenta con un grado de estabilidad en su funcionamiento como sistema, es decir, si algunas conexiones entre nodos (o neuronas) no responden correctamente, la memoria recuperada muestra un grado de confiabilidad aún aceptable, mediante la selección de la mejor respuesta disponible, es decir, la que cuente con el menor margen de error.

Análogamente se puede encontrar un funcionamiento similar en el cerebro humano. Durante el tiempo promedio de vida del cerebro, se registran pérdidas de células neuronales en cantidades considerables, sin embargo no ocurren pérdidas catastróficas en la memoria individual (Freeman, 1993:130).

El cerebro, por supuesto, es una estructura robusta en cuanto a este aspecto. Para el tiempo en que una persona muere, en promedio se habrá perdido alrededor del 20% de las neuronas originales (Freeman, 1993:135).

Los nodos definidos dentro de la estructura topológica del Neurotrón, son simplificaciones de neuronas reales, es decir que para cualquier nodo en la red, sólo pueden existir dos posibles estados de excitación, o que es lo mismo, dos estados de estimulación:

- ACTIVO: Cuando se está enviando estímulos a todos los nodos hacia los que se está interconectado.
- PASIVO: Cuando NO se está estimulado a ninguno de los nodos hacia los que se está interconectado.

En cualquier instante de tiempo un nodo puede cambiar su estado, dependiendo de la entrada o estimulación que reciba de las otras neuronas en la red. Con el objetivo de probar el funcionamiento del nuevo modelo, éste debe pasar primero por la etapa del entrenamiento según las reglas de aprendizaje.

El entrenamiento como tal, consistirá en ajustar los pesos w_i del modelo, en función de la información proporcionada por un conjunto de muestras representativas de las clases a diferenciar.

Es decir, si se desea entrenar el modelo para emular el aprendizaje para el reconocimiento de la clase del **número 1**, por ejemplo, se deben tomar muestras de distintos tipos de la clase en mención (ver figura 3.1), los cuales serán la entrada del modelo creado.

Estas muestras permitirán, por medio de un proceso mencionado mas adelante, el ajuste de los respectivos pesos dentro del modelo, de acuerdo a la variedad de las muestras involucradas en el entrenamiento.

Como en la vida real, mientras más muestras distintas de la misma clase se provea para el proceso de entrenamiento, mediante el cual se logra emular el aprendizaje, mejor serán los resultados obtenidos durante el proceso de reconocimiento o prueba del modelo.

Ejemplo de Muestras de la Clase del Número 1
Figura 3.1

Si se inicia el entrenamiento del modelo con cualquier muestra de una clase específica, esto provocará un valor inicial para los pesos w_i, que irán cambiando y ajustándose conforme el proceso de entrenamiento continúe con otras muestras de la misma clase.

Para ilustrar este proceso, imagínese iniciando el entrenamiento con la muestra de una clase específica. Esto provocará la excitación de sólo un grupo de neuronas dentro del enramado de la red, es decir que el peso w_i correspondiente a las neuronas excitadas, tendrán un valor distinto de cero, pero para las otras neuronas NO excitadas, el peso asociado w_i será igual a cero.

Estas neuronas estimuladas enviarán su señal a cada uno de los nodos o neuronas interconectadas a ella, mientras el proceso de entrenamiento continúe con otras muestras de la misma clase, lo que eventualmente provocará el ajuste de sus respectivos pesos w_i.

Si se continúa entrenando el modelo con diferentes muestras, con el tiempo, algunos de los nodos no estimulados (es decir $w_i=0$), iniciarán a enviar sus propias señales o estímulos hacia neuronas o nodos conectados a ellos, implicando con ello que su peso w_i tendrá un valor distinto de cero.

Esto quiere decir que dentro del proceso de aprendizaje, existe una retroalimentación de estímulos, el cual permite y origina los ajustes en los respectivos pesos w_i de los restantes nodos en la estructura.

Al conjunto de valores para los pesos w_i resultantes, después de un período de entrenamiento, se le denomina Patrón de Excitación de la Red.

Es posible pensar que los patrones de excitación registrados en la red, están directamente afectados por variaciones aleatorias en la información de entrada, llamada ruido[6].

La propiedad del Neurotrón que le permite simular el proceso de memoria, está basada en el hecho de que el patrón de excitación aleatorio o ruido, provocará la generación de un patrón estable después de un período suficientemente largo, o un número considerable de iteraciones de entrenamiento.

El tiempo y el número de iteraciones aplicadas al modelo, dependerá directamente de la clase para la cual el modelo se está entrenando. Es decir, para diferentes clases, el tiempo e iteraciones necesarias para estabilizar el modelo es distinto.

En este patrón estable, se podrán observar algunos nodos siempre activos o estimulados y otros siempre pasivos o sin estímulo, los cuales forman parte y corresponden a las memorias que deseamos almacenar.

---

[6] El concepto de ruido se utiliza en su significado más obvio. El ser humano generalmente cataloga como ruido, a toda aquella perturbación sonora que dificulta la recepción de una fuente de sonidos específica. Una frase comúnmente utilizada desde el punto de vista de la informática, y que ejemplifica este fenómeno es aquella que dice: "*la información trae basura*".

Alcanzar un patrón estable dentro en la red después de un período razonable de entrenamiento, garantiza que al probar el modelo con un patrón pobre o con ruido, éste recuerde la memoria más cercana al patrón de prueba.

Es de suponerse entonces que si el patrón de entrada es vago, la memoria recordada por el modelo puede ser la incorrecta o con un grado considerable de error, análogamente al cerebro cuando trata de recordar algo sobre lo que se tiene poca o insuficiente información.

En este punto es importante mencionar que el Neurotrón se diseño precisamente para reducir ese margen de error y por lo tanto, tienda a confundirse un número menor de veces, mediante una reducción significativa de su margen de error, al compararlo con los resultados arrojados por el modelo llamado Perceptrón en las mismas circunstancias.

Ahora bien, dada la inconveniencia de la construcción física de una computadora específica para una red neuronal artificial, es necesario crear escenarios que permitan la simulación de dicha situación por medio de una computadora convencional.

Con este propósito, se ha escrito un programa, con la técnica de programación estructurada, con el objetivo de emular exactamente el funcionamiento del modelo creado: el Neurotrón.

El simple hecho de crear un programa para simular al Neurotrón, proporciona la ventaja de la parametrización, lo que permite probar el modelo con distintas parejas de clases y poder contar con un mayor margen de referencia para la evaluación del funcionamiento del modelo como tal.

b) Representación Matemática del Neurotrón

El Neurotrón se basa en una red con **n** elementos de entrada, que serán representados como $\mathbf{x_i}$, los cuales serán ponderados por **n** pesos representados como $\mathbf{w_i}$, los cuales serán calculados y ajustados durante la etapa de entrenamiento de las dos clases a reconocer.

El modelo reportará un valor mayor que cero (0) y cercano a uno (1), cuando reconozca una de las clases, y un número menor que cero (0) y cercando a menos uno (-1), cuando reconozca un patrón perteneciente a la otra de las clases.

El Perceptrón, base sobre la cual el Neurotrón fue creado, se define matemáticamente mediante:

$$y = f_h \left(\sum_{i=0}^{n-1} w_i x_i - \eta \right) \tag{3.1}$$

donde η es una constante de corrección de error de valores de salida para y los cuales pueden ser 1 o –1.

Con el objetivo de desarrollar un modelo cuyo error de estimación disminuya, y por lo tanto lograr mayores casos de acierto, se decidió involucrar una función de ajuste que sustituyera a la constante η.

De esta manera nace la estructura del nuevo modelo creado y llamado Neurotrón, que se define matemáticamente a continuación:

$$y = f_h\left(\sum_{i=0}^{n-1} w_i x_i - f_\theta(t)\right) \tag{3.2}$$

Se determina que:

- "y=1" implicará el reconocimiento de la clase **A**.
- "y=-1" el reconocimiento de la clase **B**.

La función $f_\theta(t)$ que mejor respondió al modelo se define mediante:

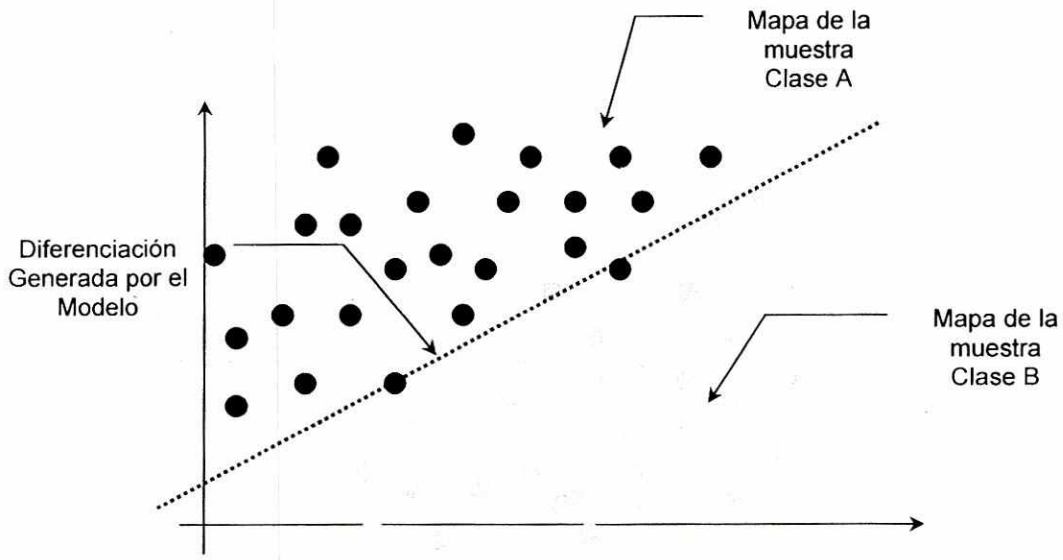
$$f_\theta(t) = \theta[d(t\text{-}1) - y(t\text{-}1)] \tag{3.3}$$

donde

- θ es el umbral para valores de salida. Debe interpretarse como una constate que ponderará la importancia entre la diferencia reportada por el modelo y la salida deseada del mismo. Se tomará como valor parametrizable cercano a cero.

- $d(t\text{-}1)$ representa a la clase a reconocer en la iteración anterior (1 o –1 según la clase.
- $y(t\text{-}1)$ representa a la clase reconocida por el modelo en la iteración anterior.

Asumiendo que la distribución de las clases en un plano está diferenciada, la figura 3.2 muestra gráficamente la forma en la que el modelo hace la diferenciación de las clases sobre las cuales fue entrenado.



Distribución y Diferenciación de Muestras
Figura 3.2

El mapa de la muestra de la Clase A y Clase B, representa la distribución de los puntos dentro de un plano de dos dimensiones.

La línea punteada que representa la Diferenciación Generada por el Modelo, representa la función generada con los pesos estables correspondientes.

El algoritmo o procedimiento de convergencia para el Neurotrón se define entonces de la siguiente manera:

Paso 1. **Inicialización de pesos y tolerancia (*threshold*) θ**

Asignar a $w_i(0)$, con $0 \leq i \leq n-1$.

Asignar a θ algún valor aleatorio pequeño cercano a cero.

Se toma $w_i(t)$ como el peso para el valor de entrada i en el instante de tiempo t y θ es el umbral en el nodo de salida.

En el instante de tiempo t, debe interpretarse como el estado de los pesos w, después de n iteraciones.

Paso 2. **Presentar nuevos datos de entrada y salida deseada (fase de entrenamiento)**

Presentar y con ello iniciar el proceso de entrenamiento, diferentes muestras de valores $x_0, x_1, \ldots, x_n$ con sus respectivos valores de salida deseados $d(t)$.

Paso 3. **Calcular el resultado de salida actual**

Calcular el resultado de la salida actual, mediante la expresión matemática que representa el modelo creado y que se presenta a continuación:

$$y(t) = f_h\left(\sum_{i=0}^{n-1} w_i(t)x_i(t) - f_\theta(t) \right) \qquad (3.4)$$

Paso 4. **Reajustar los pesos del nuevo modelo**

Reajustar los pesos del modelo mediante la siguiente expresión:

$$w_i(t+1) = w_i(t) + \eta\ [\ d(t) - y(t)\ ]\ x_i(t) \quad con\ 0 \leq i \leq n - 1 \qquad (3.5)$$

De tal forma que:

a) para todo $y(t) \approx \mathbf{1}$, se reconocerá una de las clases definidas.

b) para todo $y(t) \approx \mathbf{-1}$, se reconocerá la otra clase.

En esta ecuación η representa una fracción positiva menor que 1, cuyo valor es predeterminado y ajustable, de acuerdo a las salidas reportadas por el modelo.

Esto quiere decir que el mismo modelo, con la misma muestra de entrenamiento, puede reflejar mejores resultados con diferentes valores para η.

El valor $d(t)$ debe interpretarse como el resultado correcto deseado para los datos x_i de entrada, es decir, si los datos de

entrada son los correspondientes a una clase de objeto, $d(t)$ tendrá un valor igual a 1, y si corresponden a otra clase, el valor de $d(t)$ será igual a –1.

Nótese que en este punto, los pesos no son alterados o ajustados, si la red hace una decisión correcta.

Paso 5. **Repetir desde el Paso 2**

Este paso no será necesario realizarlo cuando ya no existan mas casos de entrenamiento.

En función de los casos de entrenamiento para una clase determinada, el algoritmo tendrá a converger hacia valores determinados y casi constantes para los pesos w_i. Por lo tanto, mientras mayor sea el número de casos utilizados para el proceso de entrenamiento, y mayor sea la diversidad de los mismos, considerando casos que incluyan ruido, mejores serán los valores w_i hacia los que el modelo convergerá.

c) Aplicación Desarrollada

Se desarrolló un modelo computacional, con el objetivo de simular y demostrar el funcionamiento del nuevo modelo desarrollado en el presente documento, al que se le denominó Neurotrón.

El programa fue desarrollado, utilizando Pascal[7] como lenguaje de programación, sobre una plataforma DOS[8] como sistema operativo, siendo los requerimientos mínimos aquellos como los exigidos por casi cualquier programa ejecutable sobre este tipo de plataforma, es decir:

- 1 MB de memoria RAM
- Procesador 80486 o mejor.
- Sistema operativo DOS o
- Cualquier versión de Microsoft Windows®

El programa utiliza un arreglo bidimensional de altura y ancho parametrizables como estructura básica de almacenamiento de información. Las dimensiones por defecto de este arreglo son siete niveles de altura, por cinco niveles de ancho.

---

[7] *Pascal* es un lenguaje de programación utilizado mayormente para desarrollo de aplicaciones científicas. Sin embargo, durante inicios de la década de los ochenta, también fue utilizado como lenguaje de programación para desarrollar aplicaciones cliente-sevidor. La versión mas reciente de Pascal es la que ahora se conoce como Visual Delphi.

[8] Que por sus siglas en idioma inglés significa Disk Operating System. Este sistema operativo fue uno de los primeros sistemas diseñados para manipular los diversos dispositivos de un computador personal. En la actualidad, solo por compatibilidad con algunas aplicaciones se conserva como parte de los sistemas operativos de Microsoft Windows®.

En este arreglo se almacenan y representan por medio de los dígitos uno y cero, aquellas posiciones activas o pasivas dentro de la estructura respectivamente, las cuales funcionarán como un mapa del símbolo o símbolos de entrada.

Como ejemplo, y asumiendo que se ha solicitado a un individuo escribir una muestra de la clase de "el número 1" sobre una cuadrícula de siete filas por cinco columnas, la figura 3.3 muestra el dígito escrito por el individuo y su representación o mapa dentro de la estructura bidimensional del programa desarrollado.

*Ejemplo de **número 1***
escrito por voluntario
Figura 3.3

Representación Matricial
*del **número 1***
escrito por voluntario

00010
01110
01010
00010
00100
00100
00100

Altura = 7 filas

Ancho = 5 columnas

Ejemplo de Representación Matricial de
dígitos escritos por voluntario
Figura 3.4

Cada posición en el arreglo o matriz bidimensional debe interpretarse como la representación de una neurona, al cual, habiendo explicado la ana'ogía, puede llamársele también como nodo. Basado en esta condición, el arreglo como un todo, es de hecho la representación de la red neuronal diseñada para simular el funcionamiento del nuevo modelo creado: el Neurotrón.

La figura 3.4 muestra un ejemplo de una representación matricial interna del símbolo perteneciente a la clase del número 1, el cual se puede utilizar como una muestra para entrenar la neurona o bien para probar su funcionamiento. Por ello, para entrenar el modelo implementado en el programa, es necesario proveer de un grupo de muestras aleatorias de las dos clases que se desean reconocer.

Como un caso de estudio práctico sobre el funcionamiento del Neurotrón, se probó el modelo con ejemplos de las imágenes correspondientes a los dígitos tres (3) y ocho (8), representados en matrices bidimensionales[9].

Los dígitos tres (3) y ocho (8) representan entonces las dos clases que el modelo aprenderá a reconocer, los cuales fueron seleccionados por las siguientes razones:

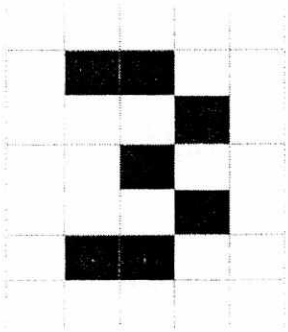
- Similitud en la representación o escritura de los números tres (3) y ocho (8).

[9] Las representaciones obtenidas para las clases del número 3 y el número 8 se encuentran trasladadas a su forma matricial en los apéndices B y C, de la sección de Apéndices, páginas 65 y 68 de este documento.

- La forma escrita de los números tres (3) y ocho (8) pueden presentar un grado razonable de ruido en sus representaciones.

La red neuronal cuenta con un nodo por cada posición en la representación matricial bidimensional de los dígitos o imágenes a reconocer. La red neuronal, por lo tanto, cuenta con 7x5 = 35 neuronas interconectadas.

Los nodos o neuronas dentro de la red neuronal del Neurotrón, pueden imaginarse como bombillas de luz que pueden estar encendidas o apagadas, activas o pasivas respectivamente, dependiendo del patrón o figura que representan, como lo muestra el ejemplo de una representación de la clase del número 3, en la figura 3.5



Ejemplo de representación ***número 3***
Figura 3.5

Para lograr un funcionamiento adecuado del modelo representado por la red neuronal, se procedió a someterlo a un proceso de entrenamiento con el objetivo de hacerle capaz de reconocer las imágenes representativas de los dígitos tres (3) y ocho (8).

Para llevar a cabo este paso, se entrenó al modelo con muestras de los dígitos tres (3) y ocho (8), como entradas al modelo, donde se tomaron conjuntos de 40 muestras aleatorias distintas de cada dígito, las cuales se aplicaron repetidamente al modelo para lograr una convergencia de pesos adecuada.

La similitud o parecido entre las representaciones de los dígitos tres (3) y ocho (8), agrega mayor dificultad al proceso de aprendizaje y ajuste de pesos del modelo.

Este factor obliga a comparar periódicamente las respuestas generadas por el modelo después de varias rondas de entrenamiento, para evaluar la posibilidad de ajustar el valor de los parámetros θ y η del modelo.

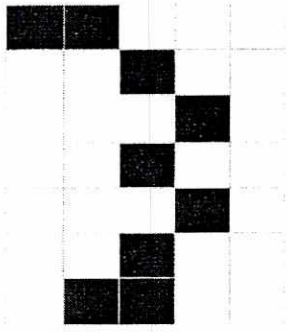
Este ajuste implica inicializar nuevamente el modelo, volver a "entrenarlo" con las muestras de las clases a reconocer, y finalmente probar de nuevo los resultados para evaluar el aprendizaje del modelo, de tal forma que poco a poco, las respuestas del Neurotrón se acerquen al patrón de entrada, aumentando el porcentaje de acierto, lo que implica una minimización del porcentaje de error o desacierto.

Al finalizar las rondas de entrenamiento, se procedió a comparar los resultados de acierto del Neurotrón, con los resultados generados por el Perceptrón.

Se encontró que el Neurotrón tiende a cometer menos errores en el reconocimiento de las clases sobre las cuales se entrenó el modelo, debido a la presencia de la función $f_0(t)$ que no está presente en el Perceptrón y que minimiza el margen de error durante el proceso de reconocimiento[10].

Generalmente, después de un período variable de entrenamiento, una red neuronal es capaz de recordar con cierto grado aceptable de certeza, los patrones o memorias correspondientes a los símbolos de entrenamiento. En este momento es aconsejable agregar una cantidad aleatoria de ruido a cualquiera de los patrones utilizados para entrenar la red neuronal, con el objetivo agregar entrenamiento a la red, para ajustar los pesos del modelo y considerar entonces, los casos donde no se tiene suficiente información para reconocer una clase determinada.

Agregar muestras con ruido, implica apagar aleatoriamente nodos encendidos en los patrones de muestra y luego utilizar esas nuevas muestras para entrenar nuevamente al modelo. Un ejemplo de una muestra con ruido adicionado, se presenta en la figura 3.6

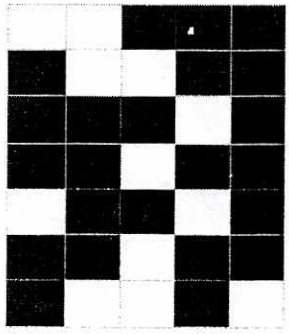


Ejemplo de representación ***número 3,*** *con "ruido" dicionado*
Figura 3.6

[10] Este cuadro comparativo se encuentra en la tabla 3 en la sección de Resultados en la página 48 de este documento.

Después de cuarenta rondas de entrenamiento, se procedió a comparar el comportamiento del Neurotrón con el del Perceptrón ante la presencia de ruido en los impulsos de entrada, y se pudo observar que el Neurotrón es capaz de recordar los patrones de entrada con mayor precisión, en los casos donde el Perceptrón fallaba.

Para cantidades considerables de ruido, un modelo tiende a confundirse y recordad el patrón equivocado. Sin embargo, es importante hacer notar que el modelo puede funcionar correctamente, si al agregar altas cantidades de ruido en el patrón de entrada, este se acerca al complemento del patrón que se desea reconocer, donde la mayoría de nodos encendidos aparecen ahora apagados y viceversa, según ejemplo en la figura 3.7.



*Ejemplo de representación **número 3,***
con "ruido" adicionado
Figura 3.7

Estas condiciones y consideraciones especiales y específicas para el caso de estudio en particular, ocasiona la aparición de cuestionamientos tales como:

- ¿Qué tan grande, en términos de nodos, debe ser la red neuronal para ser capaz de procesar los patrones de entrada y llegar a un patrón estable de excitación?

- ¿Cuál es el tiempo adecuado de entrenamiento, o cuántas son las iteraciones necesarias para generar un conjunto de pesos adecuados, capaces de producir respuestas correctas con un error mínimo?

- ¿Cuáles los las características mínimas adecuadas de las muestras, para poder obtener una convergencia adecuada en el ajuste de pesos del modelo?

Entre otras preguntas de índole similar, debemos colocarnos desde el punto de vista donde se comprende que el Neurotrón, en particular, es un modelo matemático, cuya dependencia con los datos de entrada, obliga a utilizar el proceso de prueba y error.

Es posible crear muestras depuradas que alimenten una red neuronal cuyos nodos o neuronas, manipulen características adicionales a las de

nodo encendido y nodo apagado, lo cual provocará una complejidad en el propio diseño del mismo modelo matemático.

La interrelación de las diferentes variables que definen el modelo matemático, provoca una limitante implícita en el alcance y expectativas del propio modelo, de ahí que los estudios e investigaciones deben especializarse en alguna o algunas características específicas del fenómeno que se estudia.

Por eso, el Neurotrón toma el margen de error como principal punto de estudio, debido a que una minimización de esta característica, promueve una mayor confiabilidad en este tipo de modelos, el cual sería posible utilizar para desarrollar modelos más complejos, basados en uno con un porcentaje de aciertos superior.

IV. RESULTADOS

La siguiente tabla, muestra las dos funciones $f_\theta(t)$ evaluadas, para mejorar los valores de salida del **Neurotrón**. Con un valor para θ = 0.00001 se asume que:

- La columna (1) muestra la iteración t evaluada.
- La columna (2) muestra el valor $d(t)$ esperado del modelo
- Las columnas (3) y (4) en la tabla, muestran los valores arrojados por el modelo, utilizando las funciones evaluadas de error (A) y (B) respectivamente.
- La columna (A) se define como $f_\theta(t) = \theta\ (\ d(t\text{-}1) - y(t\text{-}1)\ )$
- La columna (B) se define como $f_\theta(t) = \theta\ (\ d(t\text{-}1) / y(t\text{-}1)\ )$,

  donde $d(t\text{-}1)$ es el valor esperado en la iteración anterior y $y(t\text{-}1)$ es el valor arrojado por el modelo en la iteración anterior.

Tabla 1. Comparación de la mejor función que mejor ajusta los valores de salida del **Neurotrón** para la "*clase del número tres*", representada por $d(t) = 1$.

t	$d(t)$	$y(t)$ *corregida por función (A)*		$y(t)$ *corregida por función (B)*	
1	1	1.05658	5.65%	1.23582	23.58%
2	1	1.09325	9.32%	1.80568	80.56%
3	1	1.03508	3.50%	0.35608	64.39%
4	1	1.14062	14.1%	1.48654	48.65%
5	1	1.06456	6.45%	1.98890	98.89%
6	1	0.94456	5.54%	0.01245	98.76%
7	1	1.05465	5.46%	1.14566	14.57%
8	1	1.04256	4.25%	1.85065	85.07%
9	1	1.15460	15.4%	1.65970	65.97%
10	1	1.01954	1.95%	1.35674	35.67%
11	1	0.04568	4.5%	1.94584	94.58%
12	1	1.29521	29.5%	1.75123	75.12%
13	1	1.44056	44.1%	1.50561	50.56%
14	1	1.30508	30.5%	1.37898	37.90%
15	1	0.15088	15.1%	1.12546	12.55%
16	1	2.00124	100%	1.99968	99.97%
17	1	1.05987	5.9%	1.44566	44.57%
18	1	1.12056	12.1%	1.15524	15.52%
19	1	1.09856	9.85%	1.68568	68.57%
20	1	1.05477	5.48%	1.99899	99.90%
21	1	1.12401	12.4%	1.78954	78.95%
22	1	0.54504	45.5%	1.54864	54.85%
23	1	1.09984	9.98%	1.05888	5.89%
24	1	2.10021	110.0%	1.01068	1.10%
25	1	0.28987	71.0%	1.55978	55.98%
		Error Promedio	**23.1%**		**56.5%**

Se observa que para la función (A), el promedio de error generado por el modelo, es inferior al generado por la función (B), por lo tanto, para este caso se toma la función (A) como la que corrige mejor el error en los resultados de la red neuronal **Neurotrón**.

Tabla 2. Comparación de la mejor función que mejor ajusta los valores de salida del **Neurotrón** para la "*clase del número ocho*", representada por $d(t)$=-1.

t	$d(t)$	$y(t)$ *corregida por función (A)*		$y(t)$ *corregida por función (B)*	
26	-1	-0.02458	97.54%	-1.60589	60.59%
27	-1	-1.01257	1.26%	-1.77889	77.89%
28	-1	-1.09458	9.46%	-1.35884	35.88%
29	-1	-1.21547	24.55%	-0.45581	54.42%
30	-1	-1.00654	0.65%	-1.84548	84.55%
31	-1	-0.98578	1.42%	-1.75461	75.46%
32	-1	-1.04987	4.99%	-1.35480	35.48%
33	-1	-0.74087	25.91%	-1.68845	68.85%
34	-1	-1.00015	0.15%	-0.65840	34.16%
35	-1	-1.09458	9.46%	-1.98545	98.55%
36	-1	-0.64568	35.41%	-1.99445	99.45%
37	-1	-1.36889	36.89%	-1.44856	44.86%
38	-1	-0.35545	64.46%	-1.78260	78.26%
39	-1	-1.28655	28.66%	-1.09845	9.85%
40	-1	-0.85447	14.55%	-0.55410	44.59%
41	-1	-1.56870	56.87%	-1.35601	35.60%
42	-1	-1.00007	0.01%	-1.44465	44.47%
43	-1	-1.00547	0.55%	-1.78456	78.46%
44	-1	-0.68845	31.16%	-1.09951	9.95%
45	-1	-1.07545	7.55%	-1.80145	80.14%
46	-1	-0.60481	39.52%	-0.83225	16.78%
47	-1	-1.54504	54.50%	-1.11565	11.57%
48	-1	-1.45680	45.68%	-1.29765	29.77%
49	-1	-1.10425	10.43%	-0.88545	11.46%
50	-1	-1.29958	29.96%	-1.12658	12.66%
		Error Promedio	**25.3%**		**49.3%**

En este caso, el promedio de error generado por el modelo, utilizando la función (A), es igualmente inferior al generado por la función (B), por lo que para este caso se toma la función (A) como la que corrige mejor el error en los resultados de la red neuronal **Neurotrón**.

La comparación entre las funciones (A) y (B), se llevó a cabo observando cuál de dichas funciones arrojaba una menor desviación entre el valor *d(t)* esperado y el valor *y(t)* como resultado de la red neuronal **Neurotrón**.

Las funciones mostradas en las tablas 1 y 2, fueron creadas al azar; por lo tanto, es posible que existan una o varias funciones adicionales que trabajen mejor con el modelo, pero éste es un tópico que se deja abierto para estudios posteriores sobre la materia.

Tabla 3. Comparación de resultados entre el nuevo modelo, el Neurotrón y los resultados arrojados por el *Perceptrón*, sobre un entrenamiento de 500 iteraciones y prueba sobre 20 casos.

$\theta = 0.00001$

$\eta = 0.01$

$f_\theta(t) = \theta\ (\ d(t\text{-}1) - y(t\text{-}1)\ )$

		Neurotrón		Perceptrón	
Caso	d(t)	y(t)	e	y(t)	e
1	1	1.09023	9.023%	1.35906	35.906%
2	-1	-1.27831	27.831%	-1.68931	68.931%
3	1	0.98935	1.065%	2.89378	189.378%
4	-1	-1.19037	19.037%	0.00932	100.932%
5	1	1.39832	39.832%	-0.08593	108.593%
6	-1	-0.99994	0.001%	-0.95023	5.023%
7	1	1.00235	0.235%	1.46785	46.785%
8	-1	-1.02893	2.893%	-2.98450	198.450%
9	1	1.10854	10.854%	1.78238	78.238%
10	-1	0.00023	99.977%	-1.09323	9.323%
11	1	0.99892	0.108%	0.97723	2.277%
12	-1	-0.98298	1.702%	-0.90812	9.188%
13	1	0.98998	1.002%	1.03147	3.147%
14	-1	-1.09023	9.023%	-2.09034	109.034%
15	1	0.87823	12.177%	1.35842	35.842%
16	-1	-0.76232	23.768%	-0.89923	10.077%
17	1	1.34824	34.824%	1.00093	0.001%
18	-1	0.09023	109.023%	-0.7123	28.770%
19	1	1.00011	0.0%	1.09034	9.034%
20	-1	-1.00234	0.234%	-1.98375	98.375%
		Promedio	**20.130%**		**57.365%**

Los resultados reflejados en la tabla anterior, muestran un porcentaje de error del 20.130% para el Neurotrón, lo que implica un 79.87% de acierto, que comparado con el *Perceptrón* (con un error del 57.365%, equivalente a un 42.635% de acierto), muestra que el Neurotrón es un 46.62% mas exacto que el mencionado modelo.

Esto implica que la función de corrección incluida en el nuevo modelo creado en este trabajo de investigación, el Neurotrón, prácticamente duplica el porcentaje de acierto, comparado con el modelo existente, el *Perceptrón*.

Adicionalmente se puede observar que el Neurotrón, redujo en un 64.908% el error reportado por el *Perceptrón* al llevar este error desde un 57.365%, hasta un 20.130%.

De igual manera, se obtiene que el Neurotrón, supera en un 87.334% la capacidad de acierto del *Perceptrón*, la cual es de 42.635%.

V. CONCLUSIONES

Se ha observado que las redes neuronales artificiales se basan en simples modelos de neuronas y conexiones entre ellas. El uso de estas redes puede ser exitoso en los campos de la simulación, almacenamiento y recuperación de memorias, y para el reconocimiento de patrones que ayuden en la toma de decisiones.

Las premisas anteriores deben interpretarse como introducción hacia las conclusiones que como fruto de este trabajo de investigación se obtuvieron, y que a continuación se detallan:

Conclusión #1

El modelo creado en este trabajo de investigación, al cual se llamó **Neurotrón**, redujo en un 64.908% el error reportado por el *Perceptrón*.

Conclusión #2

El modelo creado en este trabajo de investigación, al cual se llamó **Neurotrón**, es un 46.62% más exacto que el *Perceptrón*.

Conclusión #3

El modelo creado en este trabajo de investigación, al cual se llamó **Neurotrón**, supera en un 71% la capacidad de acierto del *Perceptrón*, por lo que es recomendable la aplicación del nuevo modelo en situaciones donde el *Perceptrón* parece aplicable.

Conclusión #4

El **Neurotrón** es capaz de proveer de resultados aceptables, aun con la presencia de ruido tanto en las muestras como en las pruebas.

Conclusión #5

El **Neurotrón** requiere de períodos largos de entrenamiento para obtener una red que genere respuestas apropiadas y óptimas.

Conclusión #6

En el caso de las redes neuronales, es claro que el encargado de entrenar al modelo, incide en el comportamiento del mismo, así como en el tiempo en que el modelo se estabilice, según los parámetros que lo complementan.

VI. BIBLIOGRAFÍA

Freeman, A. James y E. Skapura. 1993. *Redes Neuronales: Algoritmos, Aplicaciones y Técnicas de Programación*. Miami. 350 págs.

Guillan, Joseph. 1997. T*ranning Neural Nets.* Miami. 150 págs.

Kumar, H. Shailesh y R. Mikkulainen. 1999. *Proceedings of Artifitial Neural Networks in Engineering.* Reino Unido. 323 pags 1999.

Lippmann, Richard. 1992. *An Introduction to Computing with Neural Nets.* IEEE-ASSP Magazine. número 30:105

Nilsson, Nils. 1996. *Principles of Artificial Intelligence.* Austin. 650 págs.

Stewart, James. 1996. *Descition Making And Learning – The Perceptron* Reino Unido. 200 págs.

Watson, Mark. 1995. *C++ Power Paradigms.* New York. 400 págs.

Winston, Patrick. 1994. *Inteligencia Artificial.* Miami. 350 págs.

VII. APÉNDICES

a. Código fuente del programa utilizado para simular el funcionamiento del Neurotrón

```
(*

      UNIVERSIDAD DEL VALLE DE GUATEMALA

      Carlos E. Castillo Galindo
      Carné 88025
      Marzo 1997

      Programa   : Modelo de Red Neuronal - Neurotrón
      Lenguaje   : Turbo Pascal ver 6.0
      Propósito  : Simular el funcionamiento del modelo matemático
                   que define la red neuronal llamada Neurotrón.

*)

program NEUROTRON;

uses crt;

const

   { número máximo de adaptaciones                                    }
   MaxTry     = 250;


   { precisión deseada en cálculos                                    }
   MaxError   = 0.000000001;


   { valor inicial de ajuste a cálculo de salidas del modelo          }
   umbral     = 0.025;


   { máximo número de nodos o neuronas en la red neuronal             }
   altura     = 7;
   ancho      = 5;
   MaxSymbols = Altura * Ancho - 1;


   { constantes de uso general                                        }
   tipo_      = '-';
   linea_     = '.';
   píxel      = '*';
   bell       = ' '; {^g^g}

type
```

```
   { estructura donde se almacenarán los objetos o símbolos          }
   objeto = record
               nivel : array[1..altura] of string[ancho];
               tipo  : integer;
            end;


   { estructura que contendrá la convergencia de pesos para cada
     nodo en la red neuronal                                         }
   pesos  = array[0..MaxSymbols] of real;



var

   { variables para almacenamiento de imagen o símbolo               }
   signo       : objeto;


   { convergencia de pesos en red neuronal, después de entrenar      }
   w           : pesos;


   { mapa de nodos o neuronas excitadas en la red                    }
   x           : pesos;


   { salida del modelo al probarlo con símbolos de entrada           }
   y_          : real;


   { umbral y precisión en cálculos y comportamiento del modelo      }
   thershold   : real;
   adaptation  : real;


   { variables para manejo de las características de la clase A      }
   nClassA     : integer;
   ClassA      : string;


   { variables para manejo de las características de la clase B      }
   nClassB     : integer;
   ClassB      : string;


   { variables de uso general                                        }
   i, j        : integer;
   archivo     : string;
   espacios    : string;
   entrada     : char;
   tecla       : char;
   continua    : bolean;



{-----------------------------------------------------------------
```

```
   procedimiento : borralineas
   parámetros    : from -> indica posición inicial
   propósito     : permite borrar una línea de pantalla desde la
                   posición <from>
 ------------------------------------------------------------------}
procedure borralineas(from : integer);
var
   i : integer;
begin
   for i := from to 24 do
   begin
      gotoxy(1,i);
      clreol;
   end;
end;



{-----------------------------------------------------------------
   procedimiento : display
   parámetros    : message    -> texto de mensaje a desplegar
                   backcolor  -> color a utilizar de fondo
                   frontcolor -> color a utilizar en letras
   propósito     : desplegar el contenido de <message> en pantalla
                   con características <backcolor>, <frontcolor>
 ------------------------------------------------------------------}
procedure display(message    : string;
                  backcolor  : integer;
                  frontcolor : integer);
begin

   { coloca atributos para desplegar mensaje                       }
   textcolor(frontcolor);
   textbackground(backcolor);

   { despliega el contenido de la variable <message>               }
   writeln(' ' + message + ' ');

   { restaura los valores default de la pantalla                   }
   textcolor(white);
   textbackground(black);
end;



{-----------------------------------------------------------------
   procedimiento : input
   parámetros    : variable   -> variable donde será almacenada
                                 la información ingresada
                   backcolor  -> color a utilizar de fondo
                   frontcolor -> color a utilizar en letras
   propósito     : permite ingresar un valor en <variable>
                   con características <backcolor>, <frontcolor>
 ------------------------------------------------------------------}
procedure input(var variable   : string;
                    backcolor  : integer;
                    frontcolor : integer);
begin


   { inicializa valor en <variable>                                }
```

```
   variable := '          ';

   { coloca atributos de pantalla                                    }
   textcolor(frontcolor);
   textbackground(backcolor);

   { realiza la lectura de datos desde el teclado hacia <variable>}
   readln(variable);

   { restaura los valores default de la pantalla                     }
   textcolor(white);
   textbackground(black);
end;



{-------------------------------------------------------------------
   función        : indentifica_fuente
   parámetros     : entrada  -> texto de mensaje a desplegar
                    caso     -> indica la clase que se identificará
   propósito      : permite indicar la fuente de información para
                    obtener muestras de entrenamiento del modelo
 -------------------------------------------------------------------}
function identifica_fuente(var entrada : char;
                               caso    : string) : boolean;
begin

   { inicializa variables generales                                  }
   entrada          := 'X';
   identifica_fuente := true;

   { inicializa ambiente para lectura de opción de entrada           }
   write(' Lectura de casos para ', caso,
         '.  (A)rchivo  (T)eclado  [  ]');
   gotoxy(57 + length(caso), wherey);
   textbackground(white);
   textcolor(white);

   { lectura de la opción seleccionada como fuente de entrada        }
   repeat
      entrada := upcase(readkey);
   until entrada in ['A', 'T'];

   { restaura los valores default de la pantalla                     }
   textcolor(white);
   textbackground(black);

   { la información es enviada al modelo, según opción escogida      }
   if (entrada = 'A') then
   begin

      { permite identificar el nombre del archivo con muestras       }
      write(' Nombre de archivo con casos para ', caso, ' : ');
      display('              ', lightgray, white);
      gotoxy(40 + length(caso), wherey - 1);
      input(archivo, lightgray, white);



      { verifica la existencia física del archivo con muestras       }
```

```
      {$I-}
      assign(casos, archivo);
      reset(casos);
      {$I+}


      { si el archivo indicado no existe, se notifica al usuario  }
      if not (ioresult = 0 and archivo <> '') then
      begin
         gotoxy(1,24);
         display('El archivo indicado no existe. Presione <enter>.',
                 lightgray,
                 red);
         gotoxy(78,24);
         repeat until keypressed;

         { la identificación de la fuente no concluyó con éxito   }
         identifica_fuente := false;
      end;
   end;
end;



{-----------------------------------------------------------------
   función        : tipocaso
   parámetros     : linea -> texto que contiene la descripción e
                    información del caso A o B (parametrizables)
   propósito      : identifica el tipo de caso (a o B), según el
                    contenido del parámetro <linea>
 -----------------------------------------------------------------}
function TipoCaso(linea : string) : integer
var

   { identificarán la muestra ingresada como de clase A o clase B }
   CA, CB : boolean;

   { índice que recorrerá la posiciones de la variable <linea>     }
   i      : integer;
begin

   { verificación para la identificación de muestra como clase A  }
   for i:= 1 to length(ClassA) do
      CA := (upcase(linea[2+i]) = upcase(ClassA[i]));

   if CA then TipoCaso :=  1;


   { verificación para la identificación de muestra como clase B  }
   for i:= 1 to length(ClassB) do
      CB := (upcase(linea[2+i]) = upcase(ClassB[i]));

   if CB then TipoCaso := -1;
end;




{-----------------------------------------------------------------
```

```
   procedimiento : lee_linea
   parámetros    : fuente -> contiene información para indicar si
                   la fuente de datos es un archivo o se hará
                   desde el teclado
   propósito     : leer una línea del símbolo con el que se estará
                   entrenando el modelo según la fuente en <fuente>
 ------------------------------------------------------------------}
procedure lee_linea(fuente : char);
var
   LineaArchivo : string;
   LeeLinea     : boolean;

begin

   { inicialización de variables y ambiente                         }
   signo.nivel[i] := espacios;
   textcolor(lightgray);
   write('       ');


   { lee una línea del símbolo, según la fuente de entrada          }
   case fuente of

      'T' : begin                  { lee símbolo desde el teclado }
               textcolor(yellow);
               readln(signo.nivel[i]);
               textcolor(white);
            end;

      'A' : begin                     { lee símbolo desde archivo }
               { inicializa variables locales para lectura        }
               LineaArchivo := '|';
               LeeLinea     := true;

               { lectura de todas las lineas que definen símbolos }
               while not (eof(casos) and LeeLinea) do
               begin
                  readln(casos, LineaArchivo);
                  if LineaArchivo = tipo_ then
                     signo.tipo := TipoCaso(LineaArchivo);

                  { almacena la línea leída del símbolo           }
                  if LineaArchivo = linea_ then
                  begin
                     delete(LineaArchivo, 1, 1);
                     for j := 1 to length(LineaArchivo) do
                        signo.nivel[i][j] := LineaArchivo[j];
                     display(signo.nivel[i], white, lightgreen);
                     LeeLinea := false;
                  end;
               end;
               continua := not eof(casos);
            end;

   { mapea el signo ingresado hacia la red neuronal excitada        }
   for j := 1 to ancho do
      if signo.nivel[i][j] = pixel
         then x[(i-1) * ancho + j - 1] :=  1
         else x[(i-1) * ancho + j - 1] := -1;
end;
{------------------------------------------------------------------
```

```
   procedimiento : LeeSigno
   parámetros    : caso -> indica si el procedimiento leerá un
                   signo para entrenar al modelo, o leerá para
                   probar el modelo
   propósito     : lectura del signo que servirá para entrenar o
                   o probar el funcionamiento del modelo
 ------------------------------------------------------------------}
procedure LeeSigno(caso : string);
begin
   BorraLineas(1);
   writeln;
   gotoxy(1,2);

   for i := 1 to altura do LeeLinea(entrada);

   writeln;

   if caso <> 'PROBAR' and entrada = 'T' then
   begin
      write(' A qué clase pertenece (',
            ClassA, '= 1  ',
            ClassB, '=-1 ? ');
      Readln(signo.tipo);
   end;

   if caso <> 'PROBAR' then
      if signo.tipo = 1
      then begin
         inc(nClassA);
         if entrada = 'A' then
            display('Caso ' + ClassA, black, lightgray);
      end
      else begin
         inc(nClassB);
         if entrada = 'A' then
            display('Caso ' + ClassB, black, lightgray);
      end;
end;



{------------------------------------------------------------------
   procedimiento : Inicializa
   propósito     : lectura del signo que servirá para entrenar o
                   o probar el funcionamiento del modelo
 ------------------------------------------------------------------}
procedure inicializa
begin

   { inicialización de variables de uso general                    }
   writeln('Inicializando ambiente...');
   nClassA    := 0;
   nClassB    := 0;
   threshold  := 0;
   adaptation := 0;
   w[0]       := 0;
   x[0]       := 0;
   espacios   := '          ');


   { inicialización de estructura de convergencia de pesos         }
```

```
   write(' Desea inicializar pesos [s/n] -> ');
   tecla      := readkey;

   for i := 1 to MaxSymbols do
   begin
      if upcase(tecla) = 'S' then w[i] := 0;
      x[i] := 0;
   end;

   if upcase(tecla) = 'S' then writeln('   Pesos inicializados.');
   writeln;


   { definición de los nombres con que se identificarán las clases}
   write(' Defina nombre de clase A : ');
   readln(classA);
   write(' Defina nombre de clase B : ');
   readln(classB);


   { definición de valores de ajuste y error                        }
   write(' Threshold (Θ = ', threshold:1:10, ') : ');
   readln(trheshold);

   write(' Adaptation (η = ', adaptation:1:10, ') : ');
   readln(adaptation);
end;



{-----------------------------------------------------------------
   función       : imagen
   propósito     : cálculo de la respuesta del modelo, según la
                   imagen almacenada en la estructura <x>, con los
                   pesos almacenados en la estructura <w>
 -----------------------------------------------------------------}
procedure inicializa
begin
   y_     := 0;
   for i  := 1 to MaxSymbols do y_ := y_ + w[i] * x[i];
   imagen := y_ - threshold;
end;



{-----------------------------------------------------------------
   función       : error
   parámetros    : x, y -> valores entre los que se calculará el
                           error
   propósito     : cálculo del nivel de error entre la respuesta
                   generada por el modelo y el resultado esperado
-----------------------------------------------------------------}
function error(x, y : real) : boolean;
begin
   errorfor := abs((x-y)/y) <= MaxError;
end;



{-----------------------------------------------------------------
```

```
   procedimiento : AdaptaPesos
   propósito     : Realiza el cálculo y ajuste de convergencia de
                   los pesos calculados en el modelo
 ------------------------------------------------------------------}
procedure AdaptaPesos
var
   wx : integer;
   wy : integer;
   mt : integer;

begin

   { inicialización de variables locales                            }
   j  := 1;
   wx := whereX;
   wy := whereY;
   mt := 0;

   { despliega información sobre signos a entrenar                  }
   gotoxy(1,7);
   writeln('   Leer signos para entrenar...  casos: ',
           nClassA:2, ',  ', nClassB:2);

   { proceso de adaptación de pesos en el modelo                    }
   textcolor(lightgray);
   repeat
      gotoxy(wx, wy);
      write(' Adaptando pesos (', j, ')...   ');
      inc(mt);

      { cálculo y despliegue de evaluación con pesos actuales       }
      y_ := imagen;
      write(y_:1:10);


      { adaptación y convergencia de los nuevos pesos para modelo }
      for i := 0 to MaxSymbols do
         w[i] := w[i] + adaptation * (signo.tipo - y_) * x[i];

      inc(j);
      if mt = MaxTry then write('Aprobado !!');
   until error(y_, signo.tipo) or (mt = MaxTry);

   if entrada = 'T' then
   begin
      textcolor(red);
      gotoxy(47, wherey);
      write('ENTER : Continuar.  ESC : Salir.');

      repeat tecla = upcase(readkey); until tecla in [#13, #27];

      contiuna := (tecla = #13);

      tecla := 'X';

      textcolor(white);
   end;
end;


{------------------------------------------------------------------
```

```
   procedimiento : PruebaSigno
   propósito     : Calcular el resultado de aplicar el modelo de
                   reconocimiento a una imagen dada.
                   El cálculo comprende la identificación de la
                   clase a la que pertenece el símbolo de entrada.
                   El modelo considera el caso en que no se cuenta
                   con suficiente evidencia para identificar un
                   símbolo.
 ------------------------------------------------------------------}
procedure PruebaSigno;
var

   { almacenará el error entre la respuesta del modelo y el
     resultado esperado, según la Clase de símbolos A              }
   eClassA : real;

   { almacenará el error entre la respuesta del modelo y el
     resultado esperado, según la Clase de símbolos B              }
   eClassB : real;

begin

   { lectura del signo según fuente de entrada                     }
   leesigno('PROBAR');


   { cálculo y despliegue de la imagen reflejada en el modelo      }
   y_ := imagen;
   writeln(' Evaluación : y = ', y_:1:10, '.');


   { identificación del resultado: Clase A, Clase B o ninguno      }
   write(' El signo ');
   if (y_ >=  umbral) then writeln('es ', ClassA, bell);
   if (y_ <= -umbral) then writeln('es ', ClassB, bell);


   { en este caso se filtran las respuestas con insuficiente
     información para identificar una clase u otra.                }
   if (eClassA = eClassB) or
      (-umbral < y_ and y_ < umbral) then
      write('no se pudo clasificar... ', bell);


   { verifica si el proceso de reconocimiento continua             }
   textcolor(red);
   gotoxy(47, wherey)
   write('ENTER : Continuar.   ESC : Salir.');
   repeat tecla = upcase(readkey) until tecla in [#13, #27];
   continua := (tecla = #13);

   if continua and entrada = 'A' then continua := not eof(casos);

   { restauración del ambiente inicial                             }
   tecla := 'X';
   textcolor(white);
end;



{-----------------------------------------------------------------
```

```
   Procedimiento principal de inicialización de ambiente,
   inicialización de estructuras, variables y mensajes.

   Programa principal controlador del flujo del proceso.
-------------------------------------------------------------}

BEGIN

   textcolor(white);
   textbackground(black);
   clrscr;

   inicializa;

   repeat
      clrscr;
      writeln;
      textcolor(red);

      display('CCG - Software            N E U R O T R O N
1997',  lightgray, red);
      display('=================================================
========',  black, white);
      display('          OPCIONES
CONVERGENCIA',  black, white);

      writeln;
      writeln('   Inicializar ambiente .......   1      Clases    : ',
              ClassA, ' & ', ClassB);
      writeln('   Leer signos para entrenar ..   2      Casos     : ',
              nClassA:2, ', ', nClassB:2);
      writeln('   Probar signos ..............   3      Parámetros: ',
              'Θ = ', threshold:1:10);
      writeln('   Salir ..................... ESC                  ',
              ' η = ', adaptation:1:10);
      writeln;

      display('=================================================
========',  black, white);
      display('                          Seleccione la opción deseada
        ',  black, white);

      gotoxy(56, wherey - 1);

      tecla := upcase(readkey);

      case tecla of
         '1' : incializa;

         '2' : if identifica_fuente(entrada, 'entrenar') then
               begin
                  continua := true;
                  while continua do
                  begin
                     leesigno('entrenar');
                     adaptapesos;
                  end;
                  if (entrada = 'A') then close(casos);
               end;

         '3' : if identifica_fuente(entrada, 'PROBAR') then
```

```
                begin
                   continua := true;
                   while continua do PruebaSigno;
                   if (entrada = 'A') then close(casos);
                end;
      end;
   until (tecla = #27);

   clrscr;
END.
```

b. Muestra de dígitos para Clase 3.

Las muestras corresponden a ejemplos de dígitos correspondientes a la clase del número 3, colocadas en el formato en que el programa desarrollado puede interpretarlas. Muestras de estos dígitos fueron tomadas como ejemplo de aplicación, principalmente por la similitud en la forma de las muestras.

```
-   3
.  ****
.     *
.   **
.  ****
.     *
.     *
.  ***
```

```
-   3
.  ****
.    **
.   **
.  ****
.     *
.    **
.  ***
```

```
-   3
.  ****
.     *
.    *
.   ***
.     *
.    *
. ***
```

```
-   3
. *****
.    *
.  ****
.     *
.    *
.  **
.
```

```
-   3
.  ****
.     *
.   **
.  ****
.     *
.     *
.  ***
```

```
-   3
.
.  ****
.     *
.   **
.     *
.  ***
.
```

```
-   3
.
.
.  ****
.    **
.   **
.     *
.   **
```

```
-   3
. *****
.     *
.     *
.  ****
.     *
.     *
. *****
```

```
-   3
.  ***
.     *
.  ***
.     *
.     *
.  ***
.
```

```
-   3
.
.  ***
.     *
.  ***
.     *
.     *
.  ***
```

```
-   3
. *****
.     *
.    **
.   ***
.     *
.    **
. ***
```

```
-   3
. ***
.    *
.    *
.   **
.     *
.    *
.  **
```

```
-   3
.  ****
.     *
.   **
.  ****
.     *
.     *
.  ***
```

c. Muestra de dígitos para Clase 8

Las muestras corresponden a ejemplos de dígitos correspondientes a la clase del número 8, colocadas en el formato en que el programa desarrollado puede interpretarlas. Muestras de estos dígitos fueron tomadas como ejemplo de aplicación, principalmente por la similitud en la forma de las muestras.

```
-   8
.  ***
. *   *
.  ***
. **   *
. *    *
. *    *
.  ***
```

```
-   8
.    **
.  *   *
. *    *
.   ***
.   * *
.   * *
.   **
```

```
-   8
.  ***
. **   *
. *    *
.  ***
. *    *
. *    *
.  ****
```

```
-   8
.   ***
.  *   *
.   * *
.   **
.  *   *
. *   *
.***
```

```
-   8
.
.  ***
.*   *
.****
.*   *
.  **
.
```

```
-   8
.
.
.  ***
.*    **
.****
.*    *
.  ***
```

```
-   8
.*****
.*    *
.*    *
.*****
.*    *
.*    *
.*****
```

```
-   8
.  ***
.*    *
.  ***
.*    *
.*    *
.  ***
.
```

```
-   8
.
.  ***
.*    *
.  ***
.*    *
.*    *
.  ***
```

```
-   8
.  ***
.**   *
.*    *
.  ****
.*    *
.*   **
.****
```

```
-   8
.    **
.**   *
.*   *
.****
.**   *
.*   *
.***
```