UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

PROGRAMA DE MAESTRÍA Y DOCTORADO EN INGENIERÍA

FACULTAD DE INGENIERÍA

INVESTIGACIÓN DE ESTRUCTURAS DE ENSAMBLE EN REDES NEURONALES

T E S I S

QUE PARA OPTAR POR EL GRADO DE:

MAESTRO EN INGENIERÍA
PROCESAMIENTO DIGITAL DE SEÑALES

P R E S E N T A :

DANIEL CALDERÓN REYES

TUTORA: **TETYANA BAYDYK**

CIUDAD UNIVERSITARIA, MÉXICO, 2011

JURADO ASIGNADO:

Presidente:	Dra. Medina Gomez Lucia
Secretario:	Dr. Escalante Ramírez Boris
Vocal:	Dra. Baydyk Mykolaivna Tetyana
1er Suplente:	Dr. Kussul Ernst Mikhailovich
2º Suplente:	Dr. Tang Xu Yu

Lugar donde se realizo la tesis:

Departamento de Tecnologías de la Información (TI), Grupo Académico de Computación Neuronal del Centro de Ciencias Aplicadas y Desarrollo Tecnológico (CCADET) de la Universidad Nacional Autónoma de México (UNAM).

TUTORA DE TESIS:

_____

Dra. Tetyana Baydyk

Índice general

Hechos: Las células grises que forman tu cerebro son sobre todo neuronas; ¡y tienes unas 100 mil millones en la cabeza! Una neurona típica está conectada a miles de otras neuronas, formando una red de procesamiento de datos con una densidad increíble. Con más de 1000 billones de conexiones -o sinapsis- en el cerebro, tenemos más rutas de transmisión en el cerebro que átomos hay en el universo. Pero antes de que se te suba a la cabeza, piensa en que el número máximo de sinapsis se da en los primeros momentos de la infancia, ¡así que un niño normal de tres años tiene diez veces más que un adulto normal!

Claro que no se ha podido demostrar científicamente que haya una relación directa entre la densidad sináptica y la habilidad cognoscitiva. (Si no entiendes esta frase, pregúntale a un niño pequeño). Pero medir la potencia mental no tiene por qué ser tan difícil: las investigaciones demuestran que tenemos suficiente potencia eléctrica centelleando en nuestras neuronas como para encender una bombilla de linterna. O como para inventar una, depende de cómo lo pienses.

Lo único que puede hacer una neurona es mandar una señal a sus vecinas, y eso sólo cuando hay suficientes sinapsis de entrada activadas; pero todas juntas son el ángel que nos empuja a la acción y el dios que nos otorga el entendimiento.

¿O son nuestros pensamientos los que las animan? Cogito ergo sum.

Este trabajo se desarrolló en el Centro de Ciencias Aplicadas y Desarrollo Tecnológico (CCADET) de la Universidad Nacional Autónoma de México (UNAM) en el grupo de trabajo de Computación Neuronal bajo la tutoría de la Dra. Tetyana Baydyk y del Dr. Ernst Kussul. Se contó con el apoyo de una beca para estudios de maestría del CONACYT. Además se contó con el apoyo parcial de los siguientes proyectos: PAPIIT IN110510-3 y ICyTDF 330/2009.

Agradecimientos

El presente trabajo no es únicamente el logro de una persona, sino de un gran número a quienes me gustaría agradecer y mencionar en este apartado.

Mi más sincero agradecimiento a mi tutora, la doctora Tetyana Baydyk, quien con trabajo duro, dedicación, paciencia y una gran sonrisa me ha guiado a lo largo de este gran aprendizaje que conlleva la tarea de la investigación científica. Es un placer poder agradecerle por su gran calidez humana y por sus innumerables enseñanzas tan valiosas.

Agradezco profundamente al doctor Ernst Kussul por todo el conocimiento que me ha transmitido sin reservas y por su gran entusiasmo. Sin duda alguna agradezco cada comentario, experiencia y conocimiento que compartió conmigo a lo largo de este trabajo.

La experiencia de aprender de investigadores como ustedes ha enriquecido mi vida en más de un sentido, como parte de mi formación científica y humana, gracias por todo.

Gracias a mi familia, que en todo momento están conmigo de una u otra forma, nada estaría completo sin todos los momentos de alegría a su lado. Gracias a mi papi J. David Calderón Calderón por tu apoyo incondicional, por tus consejos y por mostrarme que puedo alcanzar mis metas y mejorar. Gracias a mi mami Aurora Reyes Villarreal porque sin ti nada de esto sería posible, aunque no te encuentres físicamente con nosotros, eres una parte fundamental de mí cada día. Gracias por formarme con cariño y sabiduría.

Agradezco a mi hermano Carlos David Calderón Reyes por toda tu ayuda y apoyo, por comprenderme y tomarme siempre en cuenta. Agradezco a mi hermana Karina Aurora Calderón Reyes por la gran confianza que siempre me has brindado, por ser cómplice de innumerables grandes ideas y por llevarlas a la práctica.

Quiero agradecer específicamente a mis compañeros procesólogos y amigos por sus aportes para hacer de esta maestría una experiencia única. Gracias a Luis Ángel Contreras Toledo por ser un excelente compañero de equipo, siempre aportando tu conocimiento, tu personalidad y buen humor. Gracias a Emilio Augusto Jiménez Madrigal, a Jorge Armado Rodríguez Vera y Armando Salomón Hernández, sin duda alguna una generación excepcional siempre dispuestos a aprender y compartir.

Gracias a todo el posgrado de la Facultad de Ingeniería, a todos los profesores que me han permitido aprender de ellos dentro y fuera de las aulas. Específicamente agradezco a la coordinadora del programa de maestría en la opción de procesamiento digital de señales, la doctora Lucía Medina Gómez, por todos sus conocimientos que compartió así como por ser un gran apoyo y guía para mis estudios de maestría.

Agradezco a mis grandes amigos que siempre están para respaldarme, quienes han dejado una huella en mi. Agradezco a quienes compartieron conmigo las experiencias del posgrado: Andrés

E., Roberto E., César B. V., Luis Enrique V. M., Ángel V. G. Y a quienes de forma indirecta me concedieron un poco de su tiempo y compañía, aunque no los mencione individualmente, muchas gracias.

Agradezco a la Facultad de Ingeniería donde me formé como ingeniero y a su posgrado que junto con el CCADET me han formado en esta maestría. Gracias a la UNAM, mi "Alma Mater", es un honor para mí el haber estudiado y aprovechado su extensa cultura y diversidad para formarme, procuraré siempre mantener en alto el nombre de mi universidad y de México.

Gracias al CONACyT que brindó el apoyo económico a lo largo de mis estudios de maestría.

Este trabajo se desarrolló en el Centro de Ciencias Aplicadas y Desarrollo Tecnológico (CCADET) de la Universidad Nacional Autónoma de México (UNAM) en el grupo de trabajo de Computación Neuronal bajo la tutoría de la Dra. Tetyana Baydyk y del Dr. Ernst Kussul. Se contó con el apoyo de una beca para estudios de maestría del CONACYT. Además se contó con el apoyo parcial de los siguientes proyectos: PAPIIT IN110510-3 y ICyTDF 330/2009.

Introducción

Hemos presenciado el nacimiento de la era de la tecnología de la información que ha cambiado nuestros hábitos de diversión, la forma de aprendizaje y hasta la forma en que se realizan los negocios. Todos estos sorprendentes avances que se atribuyen a las computadoras han sido desarrollados en unas cuantas décadas. Sin duda alguna, las computadoras son un elemento central para el desarrollo de las nuevas tecnologías, en su mayoría las que están basadas la arquitectura secuencial de Von Neumann, cuya aplicación se ha extendido exitosamente a casi cualquier campo, hasta que los nuevos retos han ocasionado el surgimiento de la tecnología de la inteligencia artificial.

El proyecto de tesis presente es un trabajo desarrollado como parte de las actividades de la maestría en ingeniería eléctrica en la disciplina de procesamiento digital de señales y conjuga diversos conocimientos en torno a la inteligencia artificial, específicamente en redes neuronales.

Objetivo de la tesis

El objetivo de la tesis es realizar una investigación de las estructuras de ensambles de neuronas dentro de las redes neuronales como unidades representativas de información. La investigación se divide en distintas vertientes como son: investigación de la capacidad informativa de la red neuronal para reconocer y restaurar ensambles, codificar información del mundo real en ensambles de neuronas, decodificación de información de los ensambles neuronales, asociación de información por medio de ensambles neuronales, aplicación de la red neuronal con ensambles neuronales como aplicación real en la solución de un problema de ingeniería.

Esta investigación comprende análisis matemático y estadístico del sistema desarrollado. La implementación total para las diversas etapas de investigación se desarrolla en un lenguaje de alto nivel con interfaces gráficas en C++.

Inicialmente se programan, analizan y se ponen a prueba la capacidad informativa de la red neuronal con ensambles. Por medio de este software se obtienen resultados numéricos que pueden ser comparados con los desarrollos matemáticos.

El programa de software comprende en segundo término el desarrollo de un simulador gráfico para la aplicación específica que se presenta, se trata del problema de evasión de obstáculos de un robot móvil. A través de este simulador se generará una base de datos que servirá para la codificación de información del mundo real en ensambles neuronales.

Finalmente se realiza un sistema que englobe las diferentes etapas para implementar la solución total en el simulador. Se codificará la base de datos en ensambles neuronales para entrenar la red neuronal. Este sistema se prueba en la solución de nuevas situaciones empleando codificación y

decodificación de información cualitativa y cuantitativa en los ensambles neuronales.

Se esperan resultados concretos, para entender la formación y asociación de ensambles neuronales. Se pretende comparar la capacidad informativa de la red neuronal para ensambles estadísticamente independientes y dependientes (información del mundo real). Se pretende establecer un método de codificación/decodificación de información cualitativa y cuantitativa práctico y óptimo. Se espera que el sistema pueda resolver en primera instancia la aplicación elegida con cierto grado de sencillez haciendo uso de las propiedades de las estructuras de ensambles neuronales.

Marco histórico de las redes neuronales

Las redes neuronales tienen una historia con muchos altibajos y comprende una gran diversidad de trabajos multidisciplinarios, desde ingeniería, psicología, hasta medicina, fisiología, matemáticas y física teórica. Sin embargo, se pueden destacar dos grandes objetivos: primero, entender el funcionamiento fisiológico y psicológico del sistema neurológico humano y segundo, producir sistemas computacionales de procesamiento [1].

Algunos de los primeros trabajos que abrieron el camino sobre esta rama de investigación que destacan es el trabajo realizado por Karl Lasheley. En su trabajo de 1950 se resume su investigación de 30 años donde Lasheley estableció que el proceso de aprendizaje es un proceso distribuido y no local a una determinada área del cerebro [2]. Un estudiante de Lasheley, D. Hebb recogió el trabajo de su maestro y enunció la regla de aprendizaje más usada en la teoría del conectivismo conocida como regla de aprendizaje Hebbiano.

Las contribuciones de Hebb aparecen publicadas en su libro ”*The Organization of Behavior´´* [3]. En su capítulo 4 se dio por primera vez, una regla para la modificación de las sinapsis, como una regla de aprendizaje fisiológica. Además propuso que la conectividad del cerebro cambia continuamente conforme un organismo aprende cosas nuevas, creándose asociaciones neuronales con estos cambios. En su postulado de aprendizaje, Hebb retomó lo sugerido por Ramón y Cajal al afirmar que la efectividad de una sinapsis variable entre dos neuronas se incrementa por una repetida activación de una neurona a través de esa sinapsis.

Una de las mayores contribuciones fue el trabajo de McCulloch y Pitts. Este trabajo define un modelo de neuronas que colecta características fisiológicas y se conoce como neurona de McCulloch - Pitts. El modelo de McCulloch-Pitts es una aproximación a lo que se conocida en 1943 acerca de la actividad sináptica neuronal. La propuesta de McCulloch y Pitts tenía la capacidad de modelar funciones lógicas por lo que se desató una euforia por estos elementos individuales.

En 1956, Rochester, Holland, Haibt y Duda presentaron un trabajo en el que, por primera vez, se verifica mediante simulaciones una teoría neuronal basada en el postulado de Hebb. Para realizar este trabajo eminentemente práctico, se tuvieron que hacer varias suposiciones que inicialmente, no estaban en el trabajo de Hebb. Por ejemplo se acotó el valor de las sinapsis que, en principio, podía crecer ilimitadamente.

John Von Neumann, en una recopilación de sus trabajos (posterior a su muerte), sugiere como posible camino para mejorar la arquitectura de las computadoras, el estudio del sistema nervioso cerebral [4].

En 1956 durante el Congreso de Dartmouth se indicó el nacimiento de la inteligencia artificial.

En 1958, Selfridge y Rosenblatt planten implementaciones físicas de sistemas conectivistas. Selfridge planteó un sistema conocido como Pandemonium [5]. Este sistema consta de una serie de capas compuestas por lo que se conoce como 'demonios'. Cada una de las diferentes capas de este sistema se reparte las diferentes tareas a realizar. Por su parte, Rosenblatt, quince años después del estudio de McCulloch y Pitts, presentó una nueva aproximación al problema del reconocimiento de patrones mediante la introducción del perceptrón. Rosenblatt, planteó una máquina 'capaz de aprender', lo que la hacía irresistiblemente atractiva para los ingenieros [6].

En 1960 Widrow y Hoff presentaron su ADALINE. Este sistema estaba regido por un algoritmo de aprendizaje muy sencillo denominado LMS (*Least Mean Square*). Propusieron un sistema adaptable que aprende de forma más precisa y rápida que los perceptores ya existentes [7]. Este trabajo posibilitó el desarrollo del área de procesamiento adaptable [8] en el campo del procesamiento digital de señales.

En 1969 publican Minsky y Papert su trabajo titulado 'Perceptrons' que paralizó durante más de diez años el avance en este campo de la inteligencia artificial. Este trabajo puso de manifiesto las limitaciones de los perceptrones. Estas limitaciones hacían referencia a la clase de problemas que se podían resolver usando estos elementos. Minsky y Papert demostraron que un perceptrón sólo podía resolver problemas linealmente separables [9]. A pesar de la gran crítica algunos investigadores continuaron trabajando con las redes neuronales y es hasta la década de los 80 que se presenta un resurgimiento.

Kohonen y Anderson propusieron el mismo modelo de memoria asociativa de forma simultánea, no obstante las diferentes formaciones de estos autores, Kohonen como ingeniero y Anderson como neurofisiólogo. Esta propuesta es un asociador lineal [10], [11], en este modelo artificial, la neurona es un sistema lineal que usa como regla de aprendizaje la regla de Hebb modificada.

En 1980, Stephen Grossberg estableció un nuevo principio de auto-reorganización desarrollando las redes neuronales conocidas como ART (*Adaptive Resonance Theory*) [12]. Grossberg ha plantado diferentes modelos neuronales con una gran utilidad práctica (principalmente en el campo del reconocimiento de patrones).

En 1982 J. Hopfield publicó un trabajo clave para el resurgimiento de las redes neuronales. Gran parte del impacto de este trabajo se debió a la fama de Hopfield como distinguido físico teórico. En él, desarrolla la idea del uso de una función de energía para comprender la dinámica de una red neuronal recurrente [13]. El principal uso de estas redes ha sido como memorias y como instrumento para resolver problemas de optimización. En el mismo año, Kohonen publicó un importante artículo sobre mapas auto organizados que se ordenan de acuerdo a una simple regla que no necesita un 'maestro', esto es, se trata de un aprendizaje no supervisado [14].

En 1983 aparecen dos trabajos de gran importancia en el desarrollo de las redes neuronales. Fukushima, Miyake e Ito presentaron una red neuronal: el 'neurocognitron', combinando ideas del campo de la fisiología, ingeniería y de la teoría neuronal. Crearon un dispositivo que es capaz de ser aplicado con éxito en problemas de reconocimiento de patrones y fue probado con la tarea de identificar números escritos a mano [15].

En 1986 apareció un trabajo, que, junto al de Hopfield, sirvió al resurgimiento del interés por las redes neuronales. En este trabajo Rumelhart, Hinton y Williams, desarrollaron el algoritmo de aprendizaje de retropropagación (backpropagation) para redes neuronales multicapa dando una serie de ejemplos en los que se muestra la potencia del método desarrollado [16]. A partir de este

año, el número de trabajos sobre redes neuronales ha aumentado exponencialmente apareciendo un gran número de aportaciones tanto a los métodos de aprendizaje como a las arquitecturas y aplicaciones de las redes neuronales.

Se podría destacar de entre las aportaciones el trabajo de Broomhead y Lowe y el de Poggio y Girosi sobre el diseño de redes neuronales en capas usando RBF (Radial Basis Functions) [17] [18], el trabajo intensivo desarrollado sobre las máquinas de vectores soporte [19], el desarrollo de la unión entre elementos neuronales y difusos y, por último, los trabajos sobre neuronas de pulsos (*spike neurons*).

Finalmente hay que hacer mención a unos de los motores del desarrollo de las redes neuronales en los últimos años: la predicción en series temporales, ampliamente investigados y usados en la generalización de las redes TDNN (orientadas especialmente a su uso sobre series temporales) realizada Eric Wan [20]. En su trabajo los pesos sinápticos eran filtros digitales de tipo FIR.

Contenido de la tesis

La tesis se divide en cinco capítulos, que desarrollan la investigación del proyecto de tesis. Se presentan los fundamentos teóricos considerados para el desarrollo del sistema y se presentan los resultados numéricos y matemáticos obtenidos para hacer congruentes la teoría y la práctica.

El primer capítulo describe los fundamentos teóricos de las redes neuronales y de las estructuras de ensambles de neuronas, sobre estos fundamentos se construye el sistema que explota sus características.

El segundo capítulo detalla y analiza el paradigma de red neuronal que se emplea y que hace posible el manejo de estructuras de ensambles. Se describe la formación de ensambles dentro de la red neuronal, la dinámica de reconocimiento y restauración de información para realizar estimaciones de la capacidad informativa para datos estadísticamente independientes.

El tercer capítulo está dedicado al simulador gráfico del robot móvil evasor de obstáculos, por medio del cual se genera una base de datos con información proveniente del mundo real. Además este simulador gráfico permite la interacción de la red neuronal para la simulación del funcionamiento bajo situaciones particulares.

En el cuarto capítulo se desarrolla la codificación y decodificación por medio de permutaciones aleatorias para generar ensambles a partir de la base de datos del simulador gráfico. Explica la metodología para poder reconocer información cualitativa y cuantitativa que es fundamental para nuestra aplicación. Se presentan resultados que demuestran las propiedades de la red neuronal con ensambles en el caso de datos estadísticamente dependientes.

En el quinto capítulo se presentan resultados y experimentos que muestran el desempeño del sistema para diferentes parámetros, a partir de los cuales se pueden construir conclusiones sólidas. Se presentan simulaciones del sistema completo funcionando y se describen sus ventajas y desventajas, lo que nos permite concluir esta etapa de investigación y especificar las futuras líneas en las que hay que trabajar para mejorar el desempeño del sistema.

Finalmente se presentan las conclusiones sustentadas en los resultados y análisis presentados y desarrollados a lo largo de los diferentes capítulos de la tesis, se realiza una discusión del potencial del sistema así como del trabajo a futuro que se debe realizar.

Capítulo 1

Fundamentos de redes neuronales

En este capítulo se resumen las bases teóricas sobre las que se basan las redes neuronales, partiendo de los conceptos primarios de biología, se presentan los primeros modelos de neuronas artificiales que dieron origen a paradigmas de redes neuronales ampliamente analizados y aplicados en la ingeniería y finalmente se introducen como memorias asociativas. Así se facilitarán los modelos más representativos sobre los que se basa este proyecto de tesis.

El cerebro es el órgano que realiza la mayor parte del procesamiento de la información, conformado por células llamadas neuronas, las cuales son consideradas las unidades de procesamiento básicas, nos han permitido la resolución de tareas muy sencillas hasta las más complicada imaginables.

Las redes neuronales artificiales están inspiradas en estos cerebros humanos, por esta razón, se presentan a continuación las principales características biológicas de las neuronas.

1.1. Biología de la neurona

La red neuronal biológica está conformada por células nerviosas llamadas neuronas, que son células especializadas cuya principal característica es la excitabilidad eléctrica, capaces de recibir estímulos y conducir impulsos nerviosos por medio de conexiones con otras neuronas.

Un cerebro humano cuenta aproximadamente con $10^{10} - 10^{11}$ neuronas y aproximadamente con $10^{14} - 10^{15}$ conexiones entre ellas [21].

Existen diferentes tipos de neuronas, que varían de acuerdo a sus tareas específicas (sensoriales, motoras, etc.), sin embargo, las partes fundamentales de cualquier neurona son: cuerpo celular, dendritas y axón. Estas se pueden apreciar en la Fig. 1.1.

Un extremo de la célula, el de entrada, tiene ramificaciones finas y cortas llamadas dendritas. La variabilidad en forma, número y tamaño repercute en el procesamiento de información que realiza la neurona. El cuerpo celular se conoce como soma.

Las neuronas tienen una ramificación larga llamada axón, esta se aparta del cuerpo celular y puede medir hasta metros. El axón es la línea de transmisión de salida de la neurona y puede dar lugar a ramificaciones colaterales.

Como se mencionó anteriormente, las neuronas están interconectadas entre sí, a estas conexiones se les conoce como sinapsis originadas por la comunicación entre los elementos de entrada

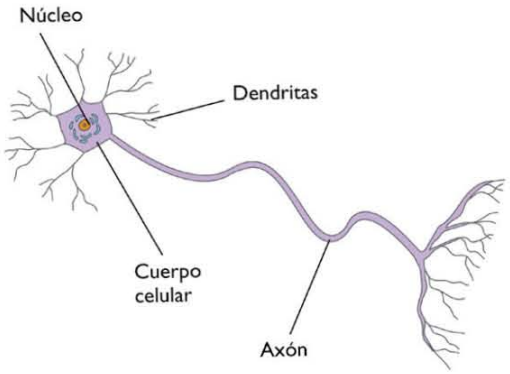


Figura 1.1: Partes de una neurona

(dendritas) de una neurona y los elementos de salida (axón) de otra neurona. Un ejemplo de sinapsis se ilustra en la Fig. 1.2. De esta forma cada neurona recibe y combina información proveniente de otras neuronas para producir una respuesta.

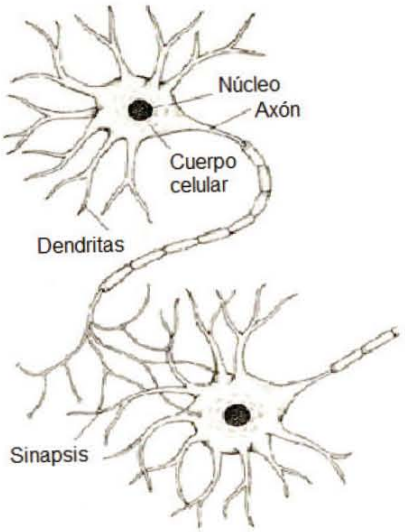


Figura 1.2: Ejemplo de sinapsis entre dos neuronas

Las sinapsis permiten que una célula influya en la actividad de otras. El dogma recibido en la teoría de redes neuronales establece que las sinapsis varían en fuerza, y las interacciones entre muchas neuronas son la clave de la naturaleza de los cómputos que realizan las redes neuronales.

Los biólogos han especificado que las neuronas transmiten la información por medio de señales eléctricas. Ya que se tratan de estructuras biológicas, estos procesos involucran concentraciones de sustancias químicas (iones de sodio y potasio generalmente) y características celulares como membranas semipermeables.

La permeabilidad de la membrana del axón influye en la concentración de los iones en el interior y exterior de las células, por lo tanto, en el potencial eléctrico de la célula con respecto al

fluido extracelular. El potencial típico de una célula es de -70mV y es un potencial de equilibrio en un estado estable de la célula [21].

Al analizar el comportamiento eléctrico el axón (salida) con respecto a estímulos eléctricos introducidos en las dendritas se pueden clasificar en dos tipos:

Estímulo de hiperpolarización: Si se aplican variaciones de tensión que llevan el potencial de la membrana a valores negativos (zona de hiperpolarización) repitiendo el patrón de entrada sin grandes variaciones, entonces se dice que es un estímulo hiperpolarizante.

Estímulo de depolarización: Si el estímulo que aplicamos rebasa cierto valor limite (potencial de acción) y obliga a la membrana a un cambio de potencial positivo (zona de depolarización) por alrededor de medio milisegundo y luego decae nuevamente.

La Fig. 1.3 muestra estos dos tipos de estímulos y sus respectivas respuestas de la neurona [21].

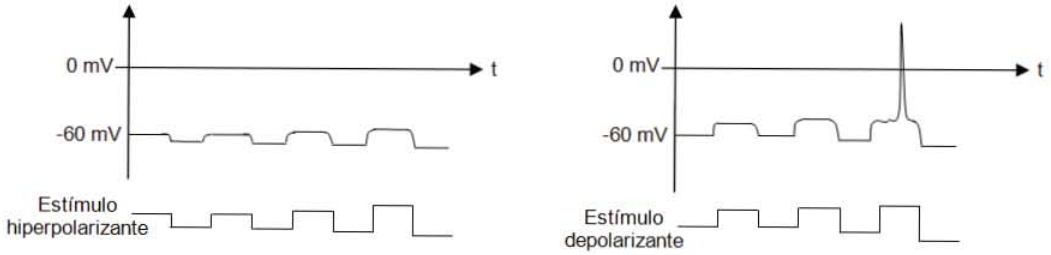


Figura 1.3: Tipos de estímulos

Este aspecto de "todo o nada" del potencial de acción tuvo una gran influencia histórica en los modelos de neuronas, ligeramente diferente del concepto de "verdadero o falso" tomado por McCulloch y Pitts [22].

De esta forma se comprende el mecanismo electroquímico del potencial de acción, para el cual se requiere un valor de $umbral$ de corriente estimulante. Por debajo de este umbral, no hay potencial de acción, por arriba del umbral, hay un potencial de acción. El proceso de activación de la neurona es repetitivo, esto significa que la célula puede disparar una serie regular de potenciales de acción, cuya frecuencia está controlada por la intensidad del estímulo. Lo anterior se puede interpretar como que la neurona se comporta como un convertidor de voltaje a frecuencia, o bien, como una codificación por frecuencia modulada.

La Fig. 1.4 muestra la respuesta de una neurona para diversos estímulos, generando potenciales de acción a diversas frecuencias de acuerdo a la intensidad del estímulo mostrado. Por ejemplo, a mayor amplitud del estímulo, mayor frecuencia del potencial de acción.

Hasta aquí se ha mostrado de forma sencilla el comportamiento observado de las neuronas, sin embargo, se sabe que hay muchas variables involucradas en el comportamiento real, por ejemplo, se sabe que una vez que se ha liberado un potencial de acción, la neurona no puede liberar un nuevo potencial por cierto tiempo. Además hay más factores involucrados, como si el impulso es continuo, sin embargo, para intervalos cortos de tiempo, se puede aceptar el funcionamiento descrito en esta sección. Con base en estas observaciones se presenta a continuación el modelado e implementación de las neuronas artificiales que pretenden imitar el comportamiento de las neuronas biológicas.

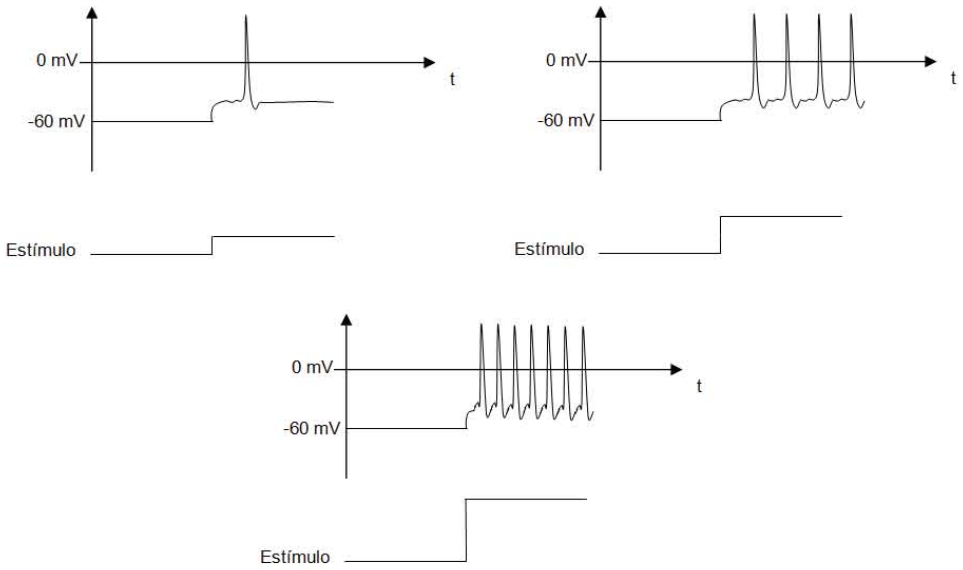


Figura 1.4: Potencial de acción para diversos estímulos

1.2. Modelos de neuronas artificiales

A partir de las neuronas biológicas, se pretende reproducir sus funciones de forma artificial, considerando que la neurona es la unidad de procesamiento fundamental y que cuenta con las funciones básicas de almacenamiento, transmisión y procesamiento [23].

Los datos pueden ser almacenados, comunicados a las unidades de información del modelo y transformados. Esto es congruente con la teoría de funciones recursivas generales, donde cualquier función puede ser reducida a la composición de algunas funciones pequeñas y primitivas.

La codificación juega un papel muy importante en el procesamiento de información, como Claude Shannon mostró en 1948, cuando hay ruido presente, la información puede ser transmitida sin pérdidas, si el código correcto con la cantidad correcta de redundancia es elegido [24].

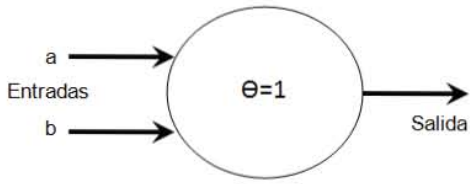
Diversos modelos de neuronas han sido desarrollados en la historia, a continuación se presentan algunos destacados y que cuentan con aplicaciones en la actualidad.

1.2.1. Neurona de dos estados

En 1943, Warren McCulloch y Walter Pitts escribieron un trabajo acerca de los cómputos que podían ser realizados por neuronas de dos estados [22]. Realizaron uno de los primeros trabajos para comprender el funcionamiento del sistema nervioso basados en elementos de cómputo de neuronas que eran abstracciones de las propiedades fisiológicas de las neuronas y sus conexiones.

El modelo que plantearon es un dispositivo binario. Cada neurona tiene un umbral fijo y suma linealmente entradas de sinapsis excitadoras con pesos idénticos, que reflejan el estado de las células presinápticas. La neurona suma sus entradas presinápticas y verifica si la suma es mayor o igual que su umbral, en este caso la neurona se activará, de lo contrario, permanecerá inactiva.

La Fig. 1.5 muestra un esquema de una neurona de dos entradas bajo el modelo propuesto por McCulloch y Pitts, este modelo realiza una disyunción lógica entre las entradas a y b con un umbral $\theta = 1$. Al realizar la suma de sus entradas y compararlas con θ nos proporcionará la salida mostrada en su tabla de verdad.



Entrada (tiempo t)		Salida (tiempo t+1)
a	b	
0	0	0
0	1	1
1	0	1
1	1	1

Figura 1.5: Esquema de neurona de McCulloch y Pitts

Si se le diera un valor de umbral de dos, entonces podría realizar la conjunción lógica.

1.2.2. Los integradores

Bruce Knight analizó un sencillo modelo integrador de una neurona en 1972 [25] basado en la neurofisiología del cangrejo herradura $limulus$. El modelo se utilizó para describir el comportamiento del axón gigante del calamar.

El modelo considera $u(t)$ como el potencial de membrana y $s(t)$ como un estímulo bajo la relación:

$$\frac{du}{dt} = s(t) \text{ cuando } (s(t) > 0). \tag{1.1}$$

Cuando u alcanza el criterio de umbral θ se dispara un impulso nervioso, enseguida, el sistema reajusta u a su valor inicial tras realizar el pico de disparo. Suponiendo que un pico fue disparado en el instante t_1, se puede representar:

$$u = \int_{t_1}^{t} s(t)dt, \tag{1.2}$$

y si el siguiente disparo se realiza en el instante t_2, se define la relación entre el umbral θ y u:

$$\theta = \int_{t_1}^{t_2} s(t)dt. \tag{1.3}$$

Suponiendo que $s(t) = s$, donde s es una constante, entonces la solución de la ecuación 1.3 es simple considerando $\Delta t = t_2 - t_1$ como una función de s y de θ como sigue:

$$\theta = s(t_2 - t_1) = s\Delta t. \tag{1.4}$$

Como la frecuencia de disparo instantáneo está definida por el inverso del tiempo entre disparos, se obtiene:

$$f = \frac{1}{\Delta t}, \tag{1.5}$$

por lo que finalmente llegamos a la relación entre magnitud del estímulo s y el umbral θ como:

$$f = \frac{s}{\theta}. \tag{1.6}$$

Esta relación lineal sencilla es en general un buen modelo que parte del comportamiento del ojo del $limulus$.

Una versión más compleja del modelo integrados considera una caída de potencial de membrana (γu) con la ecuación diferencial:

$$\frac{du}{dt} = -\gamma u + s(t). \tag{1.7}$$

Esto complica un poco la relación final entre frecuencia instantánea de disparo y la magnitud del estímulo s, que para una entrada constante se tiene:

$$f = \frac{-\gamma}{ln(1 - \frac{\gamma\theta}{s})}. \tag{1.8}$$

1.2.3. Neurona genérica

La operación de la neurona genérica conectivista es un proceso de dos etapas, en la primera, se realiza la combinación lineal independiente de las entradas con sus contribuciones sinápticas individuales (pesos). En la segunda etapa, se utiliza el resultado de la combinación lineal para como entrada de una función no lineal que resulta en la actividad final de la neurona.

La neurona genérica conectivista no dispara potenciales de acción pero tiene una salida que es un nivel de actividad continuamente graduado. La Fig. 1.6 muestra la arquitectura básica de este modelo.

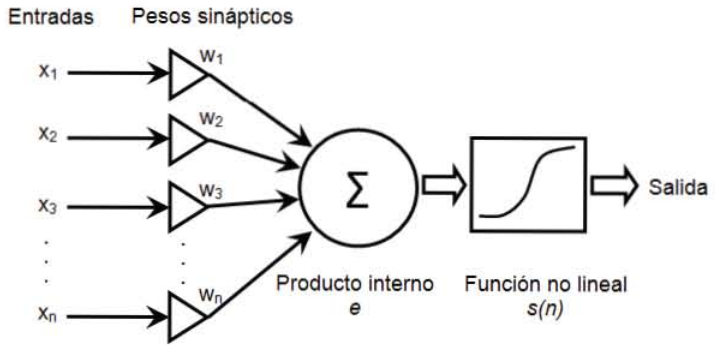


Figura 1.6: Modelo de neurona genérica conectivista

Suponiendo que se tiene una neurona con n entradas, cada una con una actividad x_n. Cada entrada cuenta con un peso de conexión sináptico w_i, que mide el grado de acoplamiento entre la neurona i y la neurona que se analiza. Se asume que los pesos de conexión son valores únicos. De esta forma la primera etapa de nuestra neurona, la combinación lineal e, se calcula por medio del producto interno de las actividades presinápticas y de los pesos de conexiones sinápticos.

$$e = (w \cdot x) = \sum_{i=1}^{n} w_i x_i = w_1 x_1 + w_2 x_2 + ... + w_n x_n, \tag{1.9}$$

donde los vectores w y x tienen por elementos los valores de los pesos de conexión w_i y de la actividad x_i en cada entrada, respectivamente.

La función no lineal $s()$ de la segunda etapa es aplicada a e para obtener la respuesta final de la neurona:

$$y = s(e) = s(w \cdot x). \tag{1.10}$$

El valor de actividad de la neurona y es un número. Esta función no lineal acota el intervalo dinámico de la neurona ya que las neuronas reales están limitadas desde una frecuencia de disparo nula hasta una frecuencia máxima f_{max} que suele ser de algunos cientos de potenciales de acción por segundo.

La función $s()$ difiere de modelo a modelo, sin embargo están diseñadas para restringir la salida dentro de un intervalo limitado, en la Fig. 1.7 se muestran tres tipos de funciones comúnmente empleadas para esta etapa, además se ilustra la interpretación del concepto de umbral de la neurona θ.

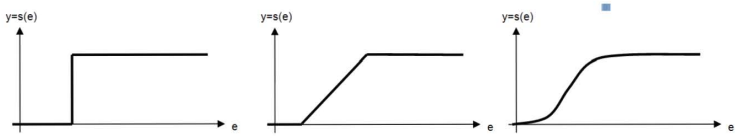


Figura 1.7: Ejemplos de funciones no lineales de activación

1.3. Redes neuronales artificiales

La computación neuronal es básicamente un proceso paralelo distribuido que tiene la capacidad de aprender de forma supervisada o no supervisada para adaptarse a la información del entorno.

El sistema de pensamiento humano es en paralelo, lo que significa que opera con un gran número de nuestras neuronas conectadas juntas. El proceso de pensamiento humano se caracteriza principalmente por ser un proceso impreciso, difuso, pero adaptable. Aprende a partir de ejemplos, experiencia y exhibe una fuerte adaptación a los cambios externos. Las redes neuronales están diseñadas de forma tal que reproduzcan la mayoría de estas propiedades, aunque aun se encuentren muy lejos de alcanzar tal desempeño.

La arquitectura de una red neuronal está basada en la forma de cómo se conectan el sistema nervioso humano. Las neuronas de nuestros cerebros están conectadas paralelamente a un gran

número de neuronas formando una máquina tipo computadora masiva paralela. La red neuronal está diseñada de tal forma que se busca imitar el estilo de procesamiento del cerebro humano. Como resultado, una red neuronal es lo suficientemente poderosa para resolver una gama de problemas que se saben son difíciles para métodos computacionales digitales convencionales. Algunas tareas típicas cognitivas incluyen el reconocimiento de un rostro, obtención de información contextualizada apropiadamente de una memoria y desempeñar tareas de clasificación demandantes.

Las características que presentan las redes neuronales artificiales, por las que son atractivas para las aplicaciones de ingeniería son:

1. Aprendizaje. Una red neuronal puede modificar su comportamiento en respuesta al entorno. Cuando se dan un conjunto de entradas con o sin las salidas deseadas, las redes neuronales se pueden autoajustar para producir respuestas consistentes.

2. Generalización. Cuando la red es entrenada, su respuesta puede ser en cierto grado insensible a variaciones menores, lo que puede ser causado por interferencia de ruido o distorsiones de corrimiento en un entorno del mundo real y sus entradas.

3. Paralelismo masivo: La información es procesada de una manera paralela masiva.

4. Tolerancia a las fallas. Una vez que se hacen las conexiones de la red, esta es capaz de entregar un comportamiento robusto. La respuesta de estas redes como un todo puede ser solamente degradado si algunos de sus elementos de proceso son dañados o alterados.

5. Establecen relaciones no lineales entre datos. Las redes neuronales son sistemas capaces de relacionar dos conjuntos de datos. Comparando con los métodos estadísticos clásicos que realizan la misma misión tienen como principal ventaja que los datos no tienen por qué cumplir con las condiciones de linealidad, formas Gaussianas y estacionalidad [26].

A continuación se describe el perceptrón, que es quizá la red neuronal más conocida. Se presenta su arquitectura y más tarde se profundizará en las reglas de aprendizaje de las redes neuronales. Finalmente se presentarán a las redes neuronales tanto como sistemas asociativos como clasificadores de patrones.

1.4. El perceptrón

El perceptron fue la primera red neuronal que causó un impacto significativo en la comunidad científica. El perceptrón fue propuesto por Frank Rosenblatt en 1958 [6]. Inicialmente se planteaba que la capacidad de tales redes neuronales era enorme, al grado de resolver casi cualquier problema, sin embargo, conforme avanzo el tiempo resultó más difícil construir dispositivos útiles basados en ellos. Los perceptrones tienen limitaciones teóricas que fueron señaladas por Minsky y Papert [9].

Las severas críticas realizadas a los perceptrones condujeron a una pérdida de interés en las redes neuronales durante la década de los setentas, sin embargo, las investigaciones no se detuvieron

por completo. Sobre todo en el campo de la psicológica se continúo investigando tales sistemas. Con el tiempo se retomó el interés en las redes neuronales hasta su resurgimiento.

El modelo del perceptrón no es único, numerosas variantes han sido propuestas y analizadas: sistemas de capas múltiples, diversas reglas de aprendizaje con sus grados de complejidad y hasta sistemas con arquitecturas complejas de realimentación.

La arquitectura propuesta por Rosenblatt involucra varias neuronas genéricas conectivistas, cuya respuesta puede ser únicamente 1 ó 0 al finalizar la función no lineal, de acuerdo al umbral θ. Rosenblatt asumía varias capas de estas neuronas con complejos conjuntos de interacciones entre ellas.

El perceptrón está comprendido por una capa sensorial (frecuentemente denominada retina) y se conecta con una segunda capa llamada capa de asociación (capa A). Las conexiones entre la capa sensorial y la asociativa podían ser aleatorias, o bien, tener una conectividad localizada.

Cada neurona de la capa asociativa se conecta únicamente un subconjunto de las neuronas de la capa sensorial, es decir, está parcialmente conectada, por lo tanto, cada neurona de la capa asociativa está calculando una función diferente de la imagen de la retina. La gran mayoría de redes neuronales asumen una conectividad total, es decir, todas las neuronas están conectadas entre sí.

La capa asociativa está asimismo conectada a una tercera capa, la capa de respuesta (capa R). Las neuronas de la capa A se conectan a las de la capa R y viceversa. Las conexiones modificables son las de la capa A a la capa R. La activación de una sola neurona de la capa R para un patrón de entrada era el objetivo del perceptrón, es decir, se asociaba una clasificación de un tipo de patrón a una sola neurona de la capa R.

Se pensaba que era indeseable tener más de una neurona encendida en la capa R a la vez, por lo que se utilizaron un conjunto de conexiones inhibidoras (conexiones que impiden la activación de otras neuronas) entre ellas. Este tipo de conexiones están basadas en la biología de la inhibición lateral [27].

La Fig. 1.8 muestra un modelo del perceptrón y las diversas capas que lo conforman.

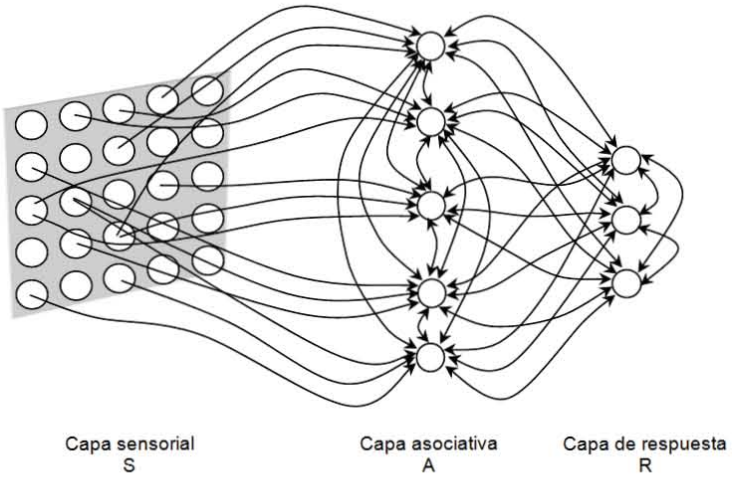


Figura 1.8: Modelo de perceptrón

Una simplificación común del perceptron, excluye las conexiones de inhibición y las de realimentación de cada neurona, como se observa en la Fig. 1.9.

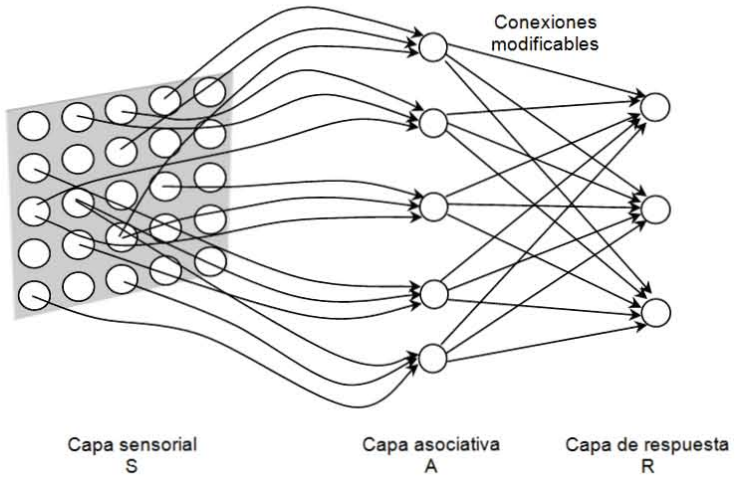


Figura 1.9: Modelo simplificado del perceptrón

El entrenamiento del perceptrón se lleva a cabo por la modificación de los valores de los pesos de conexiones entre las capas A y R. Frecuentemente se emplea una regla de entrenamiento Hebbiana, que se detalla en la siguiente sección.

1.5. Algoritmo de entrenamiento

Si bien ya se han presentado los modelos de neuronas y arquitectura de redes neuronales, ahora nos concierne describir la forma de encontrar los parámetros adecuados para alguna tarea. Si dos conjuntos de patrones deben estar linealmente separados por un perceptron, los pesos de conexiones entre las capas A y R deben de ser encontrados. Por lo que es necesario describir el algoritmo de aprendizaje del perceptron.

Un algoritmo de entrenamiento es un método adaptable por el cual una red neuronal puede ser arreglada para mostrar un comportamiento deseado. Esto se realiza presentando algunos ejemplos de entradas y salidas que mapea la red (en forma similar al aprendizaje humano). Un paso de corrección es ejecutado iterativamente hasta que la red aprenda a producir la respuesta deseada.

El algoritmo de aprendizaje es un lazo cerrado de presentación de ejemplos y correcciones de los parámetros de la red, como se muestra en la Fig. 1.10.

Un algoritmo de entrenamiento debe de adaptar los parámetros de la red de acuerdo a la experiencia previa hasta encontrar una solución, si es que existe.

Los algoritmos de entrenamiento pueden dividirse en $supervisado$ y $no\ supervisado$. El aprendizaje supervisado denota un método por el cual algunas entradas son recolectadas y presentadas a la red neuronal. La salida calculada por la red es observada y la desviación entre la salida deseada y la obtenida se mide. Los pesos son corregidos de acuerdo a la magnitud del error de acuerdo al método de aprendizaje. Este tipo de aprendizaje es conocido como aprendizaje con

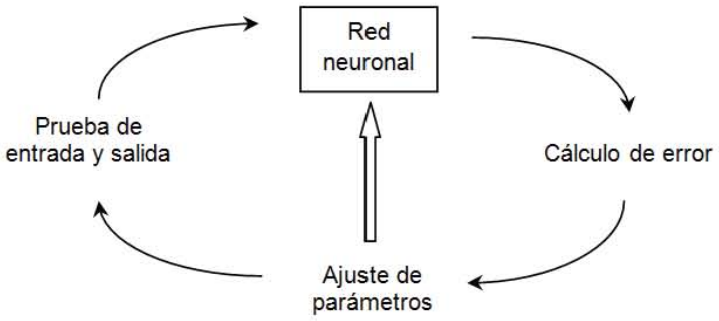


Figura 1.10: Ciclo de aprendizaje

un profesor, ya que el proceso de control sabe la respuesta correcta para el conjunto de entradas seleccionadas.

El aprendizaje no supervisado se usa cuando, para una entrada, la salida numérica exacta que debe producir la red es desconocida. Ya que no hay un profesor disponible, la red debe organizarse a sí misma para ser capaz de asociar la información con las unidades.

1.5.1. Regla de Hebb

La regla de Hebb [3] es una regla de aprendizaje supervisado que se usa frecuentemente en el entrenamiento del perceptron.

Como se ha mencionado, únicamente las conexiones entre las capas A y R son modificables.

La regla de entrenamiento de Hebb [3] es una regla basada en la observación biológica de las conexiones de las neuronas. Esta regla se usa comúnmente para el perceptron como se enuncia a continuación.

Si el sistema produce la respuesta correcta, no se modifica ningún parámetro. Pero si el sistema entrega una respuesta incorrecta, se suma un valor a a los pesos de conexiones que deberían haberse activado y se le resta un valor a a los pesos que se activaron para producir la respuesta incorrecta.

De forma matemática, esto se expresa para el peso w_k (correspondiente a la k-ésima prueba).

Si la respuesta del perceptron es correcta:

$$w_{k+1} = w_k. \tag{1.11}$$

Pero si la clasificación fue incorrecta, para toda neurona correspondiente a la clase que debió reconocerse:

$$w_{k+1} = w_k + a, \tag{1.12}$$

y para toda neurona que se activó:

$$w_{k+1} = w_k - a, \tag{1.13}$$

donde a es un valor constante establecido previamente, muchas veces se le asigna un valor unitario.

En esta regla de aprendizaje Hebbiano, el aprendizaje ocurre cuando se comete un error.

1.6. Red neuronal de Hopfield

En 1982 el físico estadounidense John Hopfield propuso un modelo de red neuronal asíncrono que tuvo un impacto inmediato en la comunidad de la inteligencia artificial, interconectando neuronas de tipo genéricas conectivistas descritas anteriormente [13].

En el modelo de Hopfield se asume que las unidades individuales preservan sus estados hasta que se actualizan por uno seleccionado. La selección es hecha aleatoriamente. Una red de Hopfield consiste de n neuronas totalmente conectadas, esto es, cada neurona está conectada a todas las demás excepto ella misma. La red es simétrica porque los pesos w_{ij} para la conexión entre la unidad i y j son iguales a los pesos w_{ji} de la conexión de la unidad j a la i. Esto se puede interpretar como una sola conexión bidireccional simple entre ambas unidades. La ausencia de una conexión de cada unidad a ella misma impide la realimentación a sus propio valor de estado.

La red neuronal de Hopfield comprende una clase de aprendizaje asociativo, ya que interrelaciona patrones de entrada y salida. Los pesos de la red neuronal de Hopfield se puede representar por medio de una matriz W simétrica cuyos elementos de la diagonal son ceros.

$$W = \begin{bmatrix} 0 & w_{12} & w_{13} & \cdots & w_{1N} \\ w_{21} & 0 & w_{23} & \cdots & w_{2N} \\ w_{31} & w_{32} & 0 & \cdots & w_{3N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{N1} & w_{N2} & w_{N3} & \cdots & 0 \end{bmatrix}. \tag{1.14}$$

Se ha demostrado que si los pesos de la matriz no contienen una diagonal de ceros, la dinámica de la red no necesariamente conduce a un estado estable. La simetría de la matriz de pesos y la diagonal cero son entonces condiciones necesarias para la convergencia a un estado estable de una red totalmente conectada asíncrona [23].

En la red neuronal de Hopfield la entrada es un vector x cuyos elementos son únicamente $\{0,1\}$ al igual que los elementos del vector de salida e. Un esquema de una red neuronal de Hopfield se presenta en la Fig. 1.11.

Como requisito de redes asociativas se tiene que todos los elementos de cómputo evalúen sus entradas y computen sus salidas simultáneamente, bajo esta consideración, la operación de la memoria asociativa puede describirse con métodos algebraicos lineales simples. La excitación de la salida de las unidades de salida es calculada usando multiplicación del vector x por la matriz W y evaluando la función de la señal $s()$ en cada nodo.

$$s(e_i) = \begin{cases} 1, & \text{si} \quad (w \cdot x) > \theta, \\ 0, & \text{si} \quad (w \cdot x) \leq \theta. \end{cases} \tag{1.15}$$

El modelo propuesto por Hopfield es un caso especial de la memoria asociativa bidireccional BAM, pero cronológicamente se propuso primero. A continuación se describe la BAM.

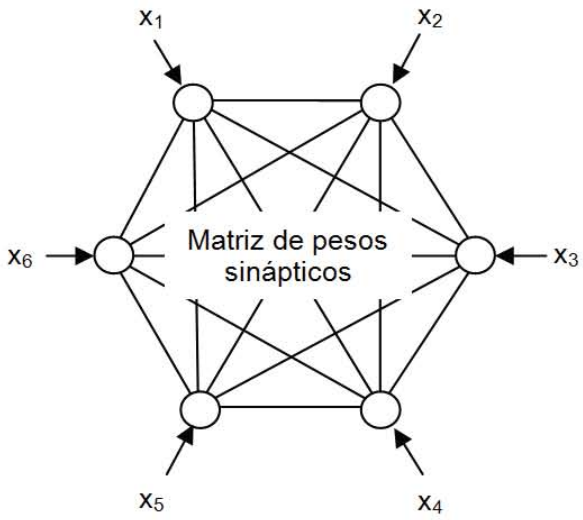


Figura 1.11: Modelo de red neuronal de Hopfield de seis neuronas

1.7. Memoria Asociativa Bidireccional BAM

Se entiende por memoria asociativa el almacenamiento y recuperación de información por asociación con otras informaciones y además permite recuperar información a partir de conocimiento parcial de su contenido, sin saber su localización de almacenamiento exacta [21].

El concepto de memoria asociativa es bastante intuitivo, se trata simplemente de asociar dos patrones. Existen dos tipos de memorias asociativas:

1. Las memorias heteroasociativas establecen correspondencias entre pares de vectores x e y, de forma que si un vector x' arbitrario está más próximo a un x, la correspondencia será $f(x') = y$. Se hace corresponder cada entrada con el vector y asociado al vector x más próximo a la entrada.

2. En una memoria autoasociativa, se iguala cualquier vector x a sí mismo x en la misma forma que se establece para la memoria heteroasociativa, de esta forma, para cualquier vector x' que esté más próximo a x, la correspondencia será $f(x') = x$.

Una BAM es un modelo definido a base de dos capas que envían información recursivamente entre ellas. La capa de entrada contiene unidades que reciben la entrada a la red y envían el resultado de su cálculo a la capa de salida. La salida de la primera capa es transportada por conexiones bidireccionales a la segunda capa, las cuales regresan el resultado de su cálculo de regreso a la primera capa usando las mismas conexiones. Como en el caso de los modelos de memoria asociativa, se puede preguntar si la información realimentada alcanza un estado donde no cambia después de algunas iteraciones, tal red es conocida como red de resonancia o memoria asociativa bidireccional (BAM). Este esquema se ilustra en la Fig. 1.12

La red mapea un vector renglón X_0 de dimensión n a un vector renglón Y_0 de dimensión k. Se denota la matriz de pesos nxk por W de tal forma que el cálculo del mapeo en el primer paso se

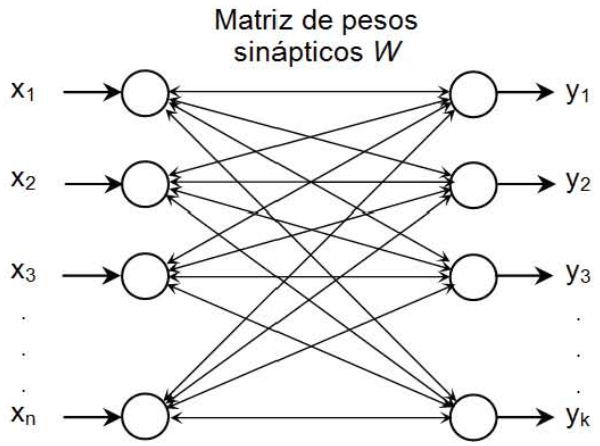


Figura 1.12: Esquema de memoria asociativa bidireccional BAM

puede escribir como

$$Y_0 = X_0 W \tag{1.16}$$

En el paso de realimentación Y_0 es tratado como la entrada y el nuevo cálculo es

$$X_1^T = W Y_0^T \tag{1.17}$$

La pregunta es si después de m iteraciones se encontrara un punto (X, Y) fijo, que es el caso de

$$Y = XWX^T = WY^T \tag{1.18}$$

La BAM es entonces una generalización de una memoria asociativa unidireccional. Un vector de entrada puede ser presentado a la red por la izquierda o derecha y después de ciertas iteraciones, la BAM encontrará su vector complementario correspondiente.

Se puede deducir que para condicionar a la BAM a converger al par de vectores (X, Y) como un punto fijo, la regla de entrenamiento de Hebb se puede usar para calcular una matriz W adecuada.

Si queremos almacenar muchos pares de vectores $(X_1, Y_1) \ldots (X_r, Y_r)$ en una BAM, entonces la regla de aprendizaje de Hebb trabaja mejor si los vectores $X_1, \ldots, X_r$ y $Y_1, \ldots, Y_r$ son pares ortogonales dentro de sus respectivos grupos.

La BAM puede ser usada para construir redes autoasociativas porque las matrices producidas por la regla de Hebb o por cómputo de pseudoinversa son simétricas. Esto es, para definir una matriz W para el vector X de forma autoasociativa $X = XW$ y $X^T = WX^T$.

Capítulo 2

Redes neuronales con estructuras de ensambles

Este capítulo presenta la definición de los ensambles de neuronas y sus principales características que se retoman en el paradigma de redes neuronales artificiales.

Se describe la metodología empleada para generar estructuras de ensambles de neuronas en nuestro paradigma de red neuronal que aportan cualidades favorables al sistema desarrollado.

Se detallan las etapas de generación de ensambles aleatorios (no correlacionados), entrenamiento y reconocimiento para estimar la capacidad informativa, es decir, el número máximo de ensambles, que puede almacenar nuestra red permitiendo recuperarlos.

2.1. Ensambles de neuronas

Los ensambles de neuronas fueron sugeridos por primera vez por Donald O. Hebb en 1949 en su trabajo "La organización del comportamiento" [3]. Hebb describe la naturaleza distribuida y la representación de la información en el sistema nervioso.

En palabras de Hebb: "Se propone... que una estimulación repetida de receptores específicos, conducirá lentamente a la formación de un 'ensamble' de células del área de asociación que puede actuar brevemente como un sistema cerrado, luego de que la estimulación ha cesado; esto prolonga el tiempo durante el cual, los cambios estructurales del aprendizaje pueden ocurrir y constituye el ejemplo más sencillo de un proceso representativo (imagen o idea)"(p. 60).

Así entonces se define a un ensamble de neuronas como un subconjunto de neuronas interconectadas que presentan conexiones (pesos sinápticos) más fuertes y representan información, conceptos o ideas en el sistema nervioso. Una sola neurona puede pertenecer a más de un solo ensamble dependiendo del contexto. Varios ensambles pueden estar activos al mismo tiempo, correspondiendo a percepciones o pensamientos elaborados.

La idea general detrás de esta teoría es que cualesquiera dos células o sistemas de células que son activadas repetidamente al mismo tiempo tenderán a ser asociadas, de modo que la actividad en una facilita la activación de la otra. Visto a nivel de ensambles, cada acción en un ensamble puede ser causada por un ensamble precedente, por un evento sensorial o normalmente por ambos.

Bajo los planteamientos de Hebb se describe también a la memoria a corto plazo como la actividad recurrente entre ensambles por estímulos y/o otros ensambles, pero la memoria a largo plazo corresponde a la formación de nuevos ensambles dentro de la red neuronal.

Varios trabajos dedicados a neuroanatomía y fisiología plantean que las neuronas que forman parte del ensamble están incrustadas en medio de otras neuronas que no son parte del conjunto, como se puede apreciar en el esquema de la Fig. 2.1. El disparo de unas cuantas neuronas del ensamble podrían llevar a la activación del ensamble completo y de forma similar, la activación de un ensamble podría ocasionar la activación de más ensambles. Por esta razón es necesario limitar el nivel de actividad en las redes neuronales con ensambles, para evitar la activación de un número excesivo de neuronas y ensambles.

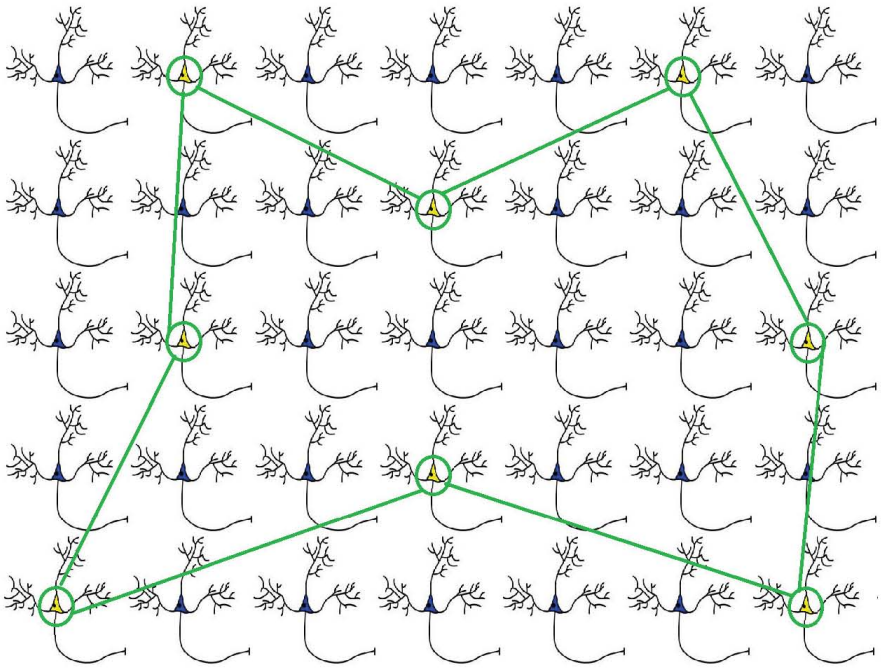


Figura 2.1: Representación de ensamble neuronal

Además, en la corteza cerebral de los mamíferos, las células de una pequeña área, a menudo responden a un solo estímulo. Cuando se hicieron por primera vez los registros eléctricos de la región visual de la corteza cerebral, se descubrió que si un electrodo penetraba perpendicularmente a la superficie de la corteza sensorial, todas las células próximas respondían a la misma orientación [28].

Esta teoría nos permite pensar que los ensambles de neuronas son las unidades elementales de cómputo, en lugar de las neuronas individuales. Aunque los ensambles de neuronas están compuestos por neuronas, su coherencia funcional nos conduce a establecer más apropiadamente interacciones entre ensambles y no entre las neuronas que lo conforman. Algunos autores [21] incluso plantean que dichos mecanismos podrían proporcionar un puente muy necesario entre las estructuras encontradas en la cognición de alto nivel y el sistema nervioso.

En resumen, se pretende aprovechar las ventajas de la capacidad asociativa de estos ensambles

neuronales. De esta forma, al contar únicamente con algunas neuronas que conforman un ensamble (información parcial) conseguiremos la activación del ensamble completo. Además, la asociación entre distintos ensambles nos permitirá asociar conceptos de distinta naturaleza.

2.2. Descripción de la metodología

Retomando estos conceptos al proyecto de esta tesis, se propone hacer una codificación distribuida en estructuras de ensambles de neuronas. Cada ensamble representará una propiedad deseada de nuestro sistema y será asociada apropiadamente para permitir una recuperación de la información a partir de datos parciales, aprovechando el uso de las propiedades de los ensambles neuronales anteriormente descritos.

Tomando la aplicación de un robot móvil que evade obstáculos, la complejidad de la maniobra del robot para evitar estrellarse depende del número de obstáculos, de la complejidad de la distribución de los mismos y de otras características como su tamaño [29].

Si tomamos el caso más sencillo de un sólo obstáculo, cuya posición con respecto al robot puede ser: {a la izquierda, al frente, a la derecha}. Y la maniobra que debe ejecutar el robot puede ser: {girar a la izquierda, seguir de frente, girar a la derecha}. Podemos generar un ensamble de neuronas que represente cada propiedad de la posición del obstáculo y de la maniobra del robot.

Para cada situación existe por lo menos una maniobra, por lo que hay asociación entre las propiedades de la situación y de la maniobra, y más aún entre los ensambles neuronales que los representan. Por ejemplo, si el obstáculo está localizado “a la izquierda” del robot, la maniobra que debe realizarse es “girar a la derecha” para evadir al obstáculo, como se ilustra en la Fig. 2.2

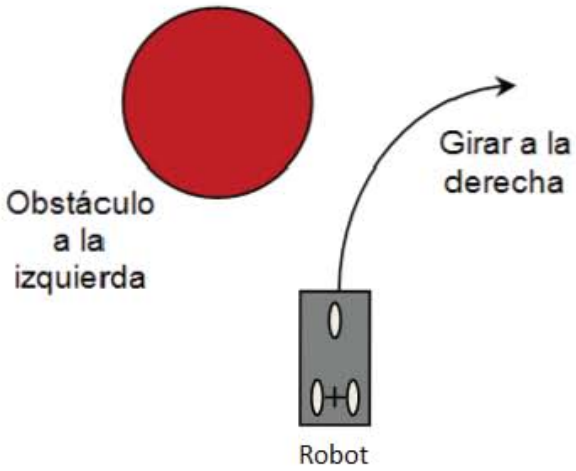


Figura 2.2: Ejemplo de maniobra “girar a la derecha”

Si el obstáculo se ubica directamente “al frente” del robot, la maniobra sólo puede ser “girar a la izquierda” o bien “girar a la derecha” pero nunca “seguir de frente” (Fig. 2.3), es decir, no hay asociación entre las propiedades “obstáculo al frente” y “seguir de frente”. Otra opción lógica para

la maniobra sería “retroceder”, sin embargo, esto implica considerar un ensamble adicional que complicaría el ejemplo que pretende ser sencillo.

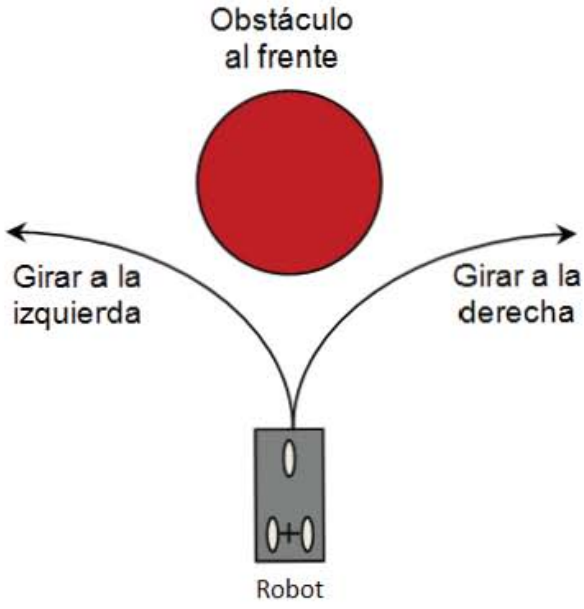


Figura 2.3: Ejemplo de maniobra para obstáculo “al frente”

Esto significa que los ensambles que asocian estas propiedades deben estar asociados en gran medida y la actividad de uno debe propagarse hasta activar el otro, haciendo uso de las neuronas que comparten o bien, por medio de conexiones sinápticas fuertes.

En la Fig. 2.4 se representan a los ensambles por medio de círculos, el área de traslape entre ellos representa el nivel de asociación entre ellos, como se aprecia, no hay traslape entre “seguir de frente” y “obstáculo al frente” porque no hay asociación entre ambos ensambles.

En la implementación, definiremos cada ensamble como un conjunto de M neuronas (tamaño de ensamble) que forman parte de una red neuronal. Para este proyecto, se emplea una red neuronal similar al modelo propuesto por Palm [34].

Nuestro paradigma de red neuronal, referida como red neuronal con ensambles [33], comprende una red de N neuronas con conectividad total, es decir, cada neurona y_i está conectada a todas las demás y_j ($j = 1, 2, ..., N$) y a sí misma por medio de pesos sinápticos w_{ij}. Cada neurona es del tipo genérico conectivista, es decir, cuenta con una sección de producto interno, un umbral y una función no lineal. Por ejemplo, la Fig. 2.5 muestra un esquema para el caso de una red neuronal con tres neuronas.

De forma similar que las redes de Hopfield, se puede hacer una representación de los pesos sinápticos por medio de una matriz W simétrica de NxN:

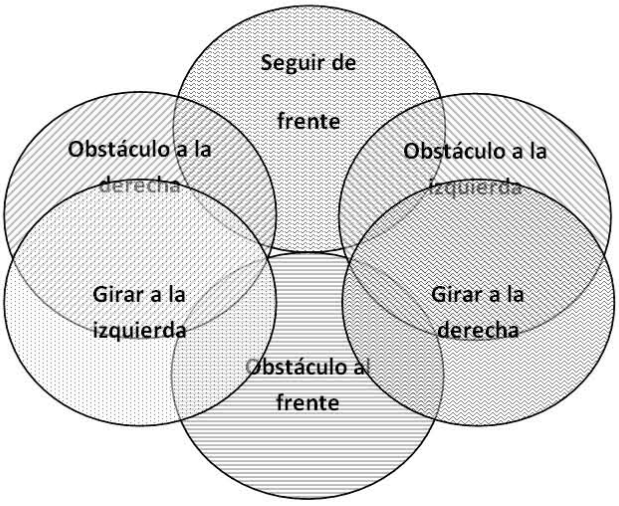


Figura 2.4: Ejemplo de asociación de ensambles neuronales

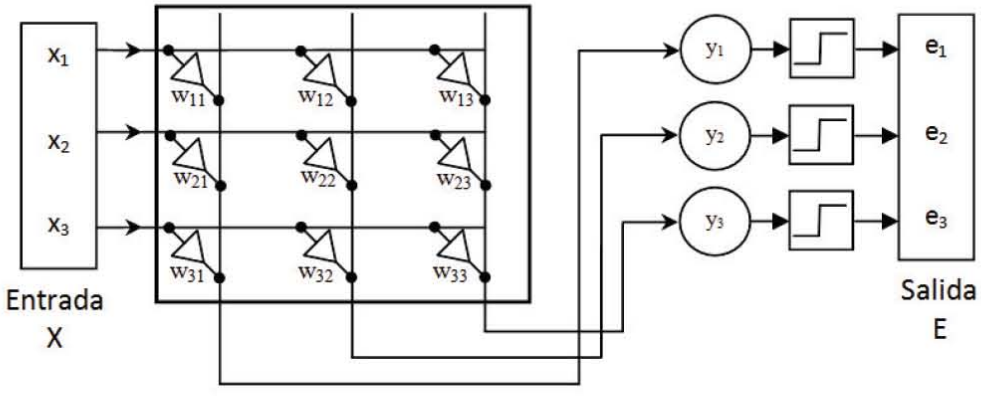


Figura 2.5: Ejemplo de red neuronal con ensambles de tres neuronas

$$W = \begin{bmatrix} w_{11} & w_{12} & w_{13} & \cdots & w_{1N} \\ w_{21} & w_{22} & w_{23} & \cdots & w_{2N} \\ w_{31} & w_{32} & w_{33} & \cdots & w_{3N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{N1} & w_{N2} & w_{N3} & \cdots & w_{NN} \end{bmatrix}. \tag{2.1}$$

La función que relaciona la entrada X con la salida E de la red neuronal queda establecida por:

$$E = S\{X^T W\}, \tag{2.2}$$

donde X^T es el vector transpuesto X, cuyos elementos $x_i = [0, 1]$ para $i = 1, 2, 3, ..., N$, E es el vector de salida, cuyos elementos $e_i = [0, 1]$ para $i = 1, 2, 3, ..., N$, y la función $S\{\}$ es la función de umbral de las neuronas:

$$S\{y_i\} = \begin{cases} 1, & \text{si} \quad y_i \geq \theta, \\ 0, & \text{si} \quad y_i < \theta, \end{cases} \; i = 1, 2, 3, ..., N, \tag{2.3}$$

θ es el valor de umbral para las neuronas (este es el sistema que limita el nivel de actividad en la red neuronal). El valor de este umbral es elegido de tal forma, que el número de neuronas activas en E es igual o menor que el tamaño del ensamble M. Expresándolo por medio del producto interno:

$$(E \cdot E) \leq M. \tag{2.4}$$

2.2.1. Creación de ensambles

El sistema codificará información en ensambles de neuronas. Estos ensambles neuronales proporcionan significativas ventajas deseables en sistemas de ingeniería para la solución de problemas por medio de técnicas de inteligencia artificial.

Para designar un ensamble neuronal A_k, se definen primero el tamaño de la red neuronal N, el número de neuronas que formarán parte de un ensamble neuronal M (tamaño de ensamble) y finalmente se define el número de ensambles K que se generarán.

Una vez que se definieron estos parámetros, se procede a designar las neuronas que serán incluidas en el conjunto. De las posible N neuronas, se eligen M neuronas de forma aleatoria para formar cada uno de los ensambles A_k para $k = 1, 2, ..., K$. El producto interno de cada ensamble cumplirá con $(A_k \cdot A_k) = M$.

Los ensambles serán representados por medio de vectores de tamaño N, cuyos elementos son: 0 ó 1, dependiendo si la neurona en cuestión forma parte del ensamble 1 (está activa) o no 0 (está inactiva).

Tomando el ejemplo de una red neuronal de tamaño $N = 10$ y ensambles de tamaño $M = 4$, un conjunto de cuatro ensambles generados $K = 4$ se muestra a continuación:

$$A_1 = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \; A_2 = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \; A_3 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{bmatrix} \; A_4 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{2.5}$$

Los ensambles se generan y se almacenan en memoria de la computadora. De esta forma se puede tener acceso en cualquier momento a cualquier ensamble A_k generado.

2.2.2. Entrenamiento

Para guardar los ensambles en la red neuronal se realiza el procedimiento de entrenamiento. Al realizar el entrenamiento, se fortalecen las conexiones de las neuronas que forman parte de un ensamble. De esta forma cuando se activen las neuronas del ensamble, se incrementará el peso de conexión entre estas neuronas.

La red neuronal con ensambles se entrena para que desempeñe el papel de una memoria autoasociativa, es decir cada vector de entrada X se mapeara en el vector de salida $E = X$. Para este fin se utiliza una regla de entrenamiento basada en la regla de Hebb [3].

El procedimiento de entrenamiento modifica los pesos de conexiones sinápticas de la matriz W, es decir, se incrementan los pesos de las conexiones entre las neuronas de cada ensamble. Esto se puede definir de forma matemática a partir del producto externo de los ensambles generados A_k:

$$W = \sum_{k=1}^{K} (A_k {A_k}^T), \tag{2.6}$$

Para simular el peso de conexión elegimos variables de 4 bits. Por esta razón la única limitante a los pesos sinápticos es $w_{ij} \leq 16$ para evitar desbordamientos de variables de 4 bits en la simulación por computadora.

Por ejemplo para el vector $A_1 = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$ el producto externo se calcula:

$$A_1{A_1}^T = \begin{bmatrix}0\\1\\1\\1\\0\\0\\0\\1\\0\\0\end{bmatrix} \times \begin{bmatrix}0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0\end{bmatrix} = \begin{bmatrix}0&0&0&0&0&0&0&0&0&0\\0&1&1&1&0&0&0&1&0&0\\0&1&1&1&0&0&0&1&0&0\\0&1&1&1&0&0&0&1&0&0\\0&0&0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0&0&0\\0&1&1&1&0&0&0&1&0&0\\0&0&0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0&0&0\end{bmatrix}. \quad (2.7)$$

De forma similar se calculan ${A_2}{A_2}^T$, ${A_3}{A_3}^T$ y ${A_4}{A_4}^T$. Finalmente al sumar estas cuatro matrices obtenemos la matriz de pesos sinápticos W entrenada para los $K = 4$ ensambles.

$$W = \begin{bmatrix}2&0&1&0&1&1&0&1&1&1\\0&1&1&1&0&0&0&1&0&0\\1&1&3&1&2&2&0&1&0&1\\0&1&1&1&0&0&0&1&0&0\\1&0&2&0&2&2&0&0&0&1\\1&0&2&0&2&2&0&0&0&1\\0&0&0&0&0&0&0&0&0&0\\1&1&1&1&0&0&0&2&1&1\\1&0&0&0&0&0&0&1&1&1\\1&0&1&0&1&1&0&1&1&2\end{bmatrix}. \quad (2.8)$$

Se puede observar que los valores de la diagonal principal tienden a ser mayores que en el resto de la matriz, esto se debe a que los valores de la diagonal principal son los valores de realimentación de cada neurona, es decir, el peso de conexión de cada neurona con sigo misma.

Los valores de la diagonal serán generalmente valores grandes, mientras que los demás valores serán menores, pero ambos dependerán de los parámetros de nuestro sistema: tamaño de la red neuronal N, tamaño de ensamble M y número de ensambles K.

Para realizar un análisis probabilístico de las características de la matriz de pesos sinápticos W, se define la variable aleatoria V como la probabilidad de que el peso de conexión sináptico w_{ij} tenga un valor $v = 0, 1, 2, ..., 16$.

Podemos dividir el estudio en:

1. Estadística de los valores de la diagonal $w_{ij_{i=j}}$ y

2. Estadística de los valores de los triángulos superior e inferior $w_{ij_{i\neq j}}$.

A continuación se presentan los análisis de las distribuciones para ambos casos.

Estadística de los valores de la diagonal

Como se mencionó anteriormente, los valores de la diagonal son valores mayores que el resto de valores de la matriz. Estos valores $w_{ij_{ij}}$, los denominaremos w_i por simplicidad.

Retomemos el ejemplo para $N = 10$, $M = 4$ y $K = 4$. Los ensambles A_k generados son los presentados en 2.5, y la matriz de pesos sinápticos W entrenada resultante es la presentada en 2.8.

Observamos que la diagonal es la suma de los ensambles A_k, por lo tanto, cada elemento de la diagonal

$$w_i = \sum_{k=1}^{K} a_{ik}, \tag{2.9}$$

donde a_{ik} representa el elemento i del ensamble A_k. Es decir, el valor w_1 es la suma del primer elemento a_1 de todos los ensambles A_k. Esto se puede observar en las matrices **??** y **??**.

De esta forma podemos analizar los valores de la variable aleatoria V_d para los valores de la diagonal principal. Tomemos el ejemplo para $f_{V_d}(0) = P\{V_d = 0\}$, esto es la probabilidad de que los pesos de la diagonal principal w_i sean 0. Este caso se logra únicamente cuando todos los valores de los K ensambles en una sola posición son cero, como lo indica la ecuación 2.9.

La probabilidad de que un elemento de un ensamble a_{ik} sea 0, la calculamos de acuerdo a la teoría de básica de probabilidad, es decir, número de ceros entre número total de neuronas. Así tenemos que:

$$P\{a_i = 0\} = \frac{N - M}{N} \tag{2.10}$$

Mientras que la probabilidad de que el elemento sea uno (la neurona pertenece al ensamble), se calcula de forma similar como número de neuronas activas entre número total de neuronas. Esto se expresa como:

$$P\{a_i = 1\} = \frac{M}{N} \tag{2.11}$$

Entonces la probabilidad de que los K vectores no contengan a la neurona i se calcula por medio de la conjunción de probabilidades, es decir el producto:

$$f_{V_d}(0) = P\{V_d = 0\} = \left(\frac{N - M}{N}\right)^K \tag{2.12}$$

De forma similar, podemos calcular la probabilidad de que el peso w_i sea 1. Esto es, sólo un ensamble contiene a la neurona i (ecuación 2.13), mientras que todos los demás $K - 1$ ensambles no la contienen (ecuación 2.10). nuevamente consideramos la multiplicación de probabilidades y además las posibles combinaciones de los K ensambles al tomar sólo un ensamble a la vez. De forma matemática:

$$f_{V_d}(1) = P\{V_d = 1\} = \left(\frac{N - M}{N}\right)^{K-1} \left(\frac{N}{M}\right) K \tag{2.13}$$

En forma general, bajo el análisis mostrado y considerando que los pesos no pueden ser mayores que 16, se determina:

$$f_{V_d}(i) = P\{V_d = i\} = \begin{cases} \left(\frac{N-M}{N}\right)^{K-i} \left(\frac{N}{M}\right)^{i} \left({}_KC_i\right), & \text{para} \quad i = 0, 1, 2, ..., 15, \\ 1 - P\{V_d \leq 15\}, & \text{para} \quad i = 16, \end{cases} \tag{2.14}$$

donde N es el tamaño de la red neuronal, M es el tamaño del ensamble, K es el número de ensambles y ${}_KC_i$ es la combinatoria de K en i.

A partir de esta distribución podemos obtener sus parámetros más importantes como son: media y varianza usando el operador esperanza. Para la media se tiene:

$$we_d = E\{w_i\} = \sum_{i=0}^{16} i f_{V_d}(i) \tag{2.15}$$

Para calcular la varianza de la distribución, podemos utilizar la ecuación 2.14 ya que:

$$\sigma_d^2 = \sum_{i=0}^{16} f_{V_d}(i)(i - we_d)^2 \tag{2.16}$$

Se desarrolló un programa en MatLab (Apéndice A) que implementa la metodología descrita y nos permite analizar la estadística de redes neuronales relativamente pequeñas (menor a 4000 neuronas) debido al consumo de memoria de este paquete.

Por medio de este programa podemos calcular de forma experimental la distribución de la variable aleatoria V_d y comparar con las expresiones desarrolladas para diversos valores de los parámetros de la red neuronal. Las siguientes gráficas muestran las funciones de probabilidad de la variable aleatoria V_d calculadas con la fórmula aritmética presentada, ecuación 2.14, (mostradas con asteríscos *) y calculadas para un caso específico (mostradas con círculos $\circ$). Así en cada gráfica se puede apreciar que el análisis desarrollado describe efectivamente la distribución de los valores de los pesos sinápticos.

Por ejemplo en la Fig. 2.6 se muestran las distribuciones de probabilidad de los pesos sinápticos para una red neuronal de tamaño $N = 400$, con tamaño de ensambles de $M = 30$ y número de ensambles $K = 50$. En este ejemplo, la mayoría de los pesos sinápticos son pequeños y es difícil apreciar la forma de la distribución.

Otro ejemplo se muestra en la Fig. 2.7 para una red neuronal de tamaño $N = 100$, número de ensambles $K = 30$ y tamaño de ensamble $M = 30$. En estás gráficas se observa que la distribución es aproximadamente normal, lo cual es razonable al considerar que provienen de la sumatoria de muchas exponenciales. La distribución queda siempre acotada entre $0 \leq i \leq 16$ por las limitaciones impuestas.

Finalmente es interesante observar la Fig. 2.8, el caso de una red neuronal de tamaño $N = 200$ con un número de ensambles $K = 60$ y tamaño de ensamble $M = 40$ ya que los pesos sinápticos de la diagonal principal son ya bastante grandes y un gran número comienza a tener el valor límite de 16 impuesto en la programación, sin embargo, la expresión establecida para la distribución sigue describiendo adecuadamente el comportamiento de los pesos sinápticos de la diagonal principal.

En cuanto al los estadísticos de la distribución, la Tabla 2.1 muestra la comparativa de los valores calculados a partir de las ecuaciones 2.15 y 2.16 y los datos de una simulación realizada

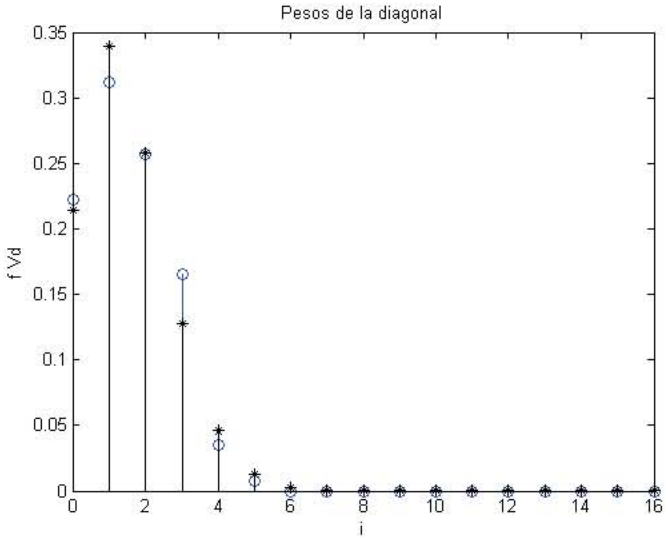


Figura 2.6: Distribuciones de probabilidad para $N = 500$, $M = 30$ y $K = 50$

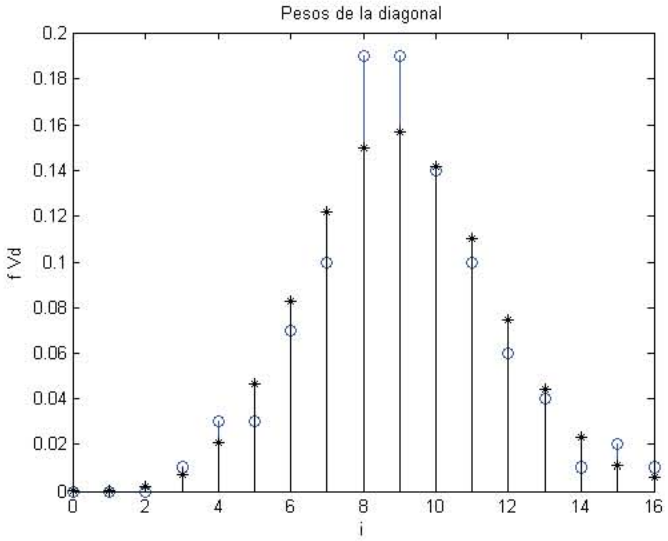


Figura 2.7: Distribuciones de probabilidad para $N = 100$, $M = 30$ y $K = 30$

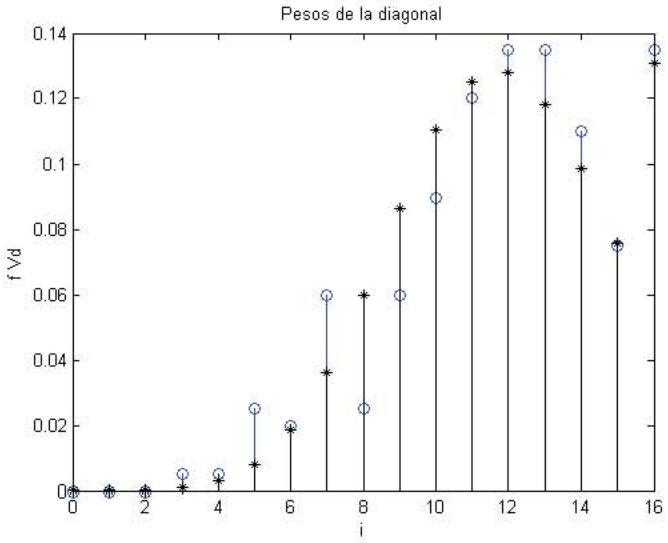


Figura 2.8: Distribuciones de probabilidad para $N = 200$, $M = 40$ y $K = 60$

con el programa en MatLab. Se observa que los valores son muy aproximados y que efectivamente se describe la distribución probabilística de los pesos sinápticos.

Tabla 2.1: Estadísticos de pesos sinápticos de la diagonal principal con MatLab y ecuaciones

Tamaño de red N	Tamaño de ensamble M	Número de ensambles K	Media we_d		Varianza σ_d^2	
			Simulación	Ecuaciones	Simulación	Ecuaciones
100	20	30	6	6	4.8081	4.7997
		50	9.94	9.9754	8.6428	7.6472
	40	30	9	8.997	4.6263	6.2534
		50	13.94	14.1462	5.4105	4.7789
400	20	30	1.5	1.5	1.3283	1.425
		50	2.5	2.5	2.3258	2.375
	40	30	2.25	2.25	2.2231	2.0812
		50	3.75	3.75	3.9023	3.4687
200	40	60	11.86	11.8392	9.0255	7.8136
1000	100	40	4	4	3.8258	3.6

Este análisis estadístico de las distribuciones de los pesos sinápticos servirán de base para un futuro análisis matemático de la capacidad de la red neuronal, sin embargo, por lo pronto, usaremos estos resultados únicamente para describir e ilustrar la restauración de ensambles en la sección de reconocimiento.

A continuación se desarrolla el análisis estadístico para los pesos sinápticos fuera de la diagonal principal.

Estadística de los valores fuera de la diagonal principal

En cuanto a los valores de los pesos sinápticos que se localizan fuera de la diagonal principal $w_{ij_{i\neq j}}$, retomaremos el ejemplo del valor externo de un ensamble A_1 para ilustrar el análisis realizado. El producto externo

$$A_1^T W = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{2.17}$$

Aquí se puede observar que para M neuronas activas en el ensamble A_1, el número de pesos $w_{ij_{i\neq j}}$ que se incrementan en una unidad son M^2, pero sin considerar la diagonal principal, sólo $(M^2 - M)$ pesos sinápticos son los que se incrementan.

Así la probabilidad de que para un ensamble A_k se incremente un peso sináptico en una unidad es el número de 1's fuera de la diagonal principal $(M^2 - M)$ entre el número total de neuronas $(N^2 - N)$, nuevamente sin considerar la diagonal principal. Esto es:

$$P\{w_{ij_{i\neq j}} = 1\} = \frac{M^2 - M}{N^2 - N} \tag{2.18}$$

Y para el caso de ceros:

$$P\{w_{ij_{i\neq j}} = 0\} = \frac{N^2 - N - M^2 + M}{N^2 - N} \tag{2.19}$$

Un peso sináptico fuera de la diagonal principal se incrementa en un valor i cuando i ensambles cuentan con la misma neurona activa en una posición y todos los demás ensambles no.

Realizando un análisis similar al que se realizó para la diagonal principal, ahora para la variable aleatoria V_{nd}, considerando i ensambles, donde las neuronas están activas y considerando las ecuaciones 2.18 y 2.19. Nuevamente el peso no puede ser mayor a 16 por definición, entonces se puede deducir la fórmula:

$$f_{V_{nd}}(i) = P\{V_{nd} = i\} = \begin{cases} \left(\frac{M^2-M}{N^2-N}\right)^i \left(\frac{N^2-N-M^2+M}{N^2-N}\right)^{k-i} ({}_K C_i), & \text{para} \quad i = 0, 1, 2, ..., 15, \\ 1 - P\{V_{nd} \leq 15\}, & \text{para} \quad i = 16. \end{cases} \tag{2.20}$$

Así a partir de esta función de distribución de probabilidad podemos calcular el valor esperado de los pesos sinápticos fuera de la diagonal principal y su varianza:

$$we_{nd} = E\{w_{ij\,i\neq j}\} = \sum_{i=0}^{16} i f_{V_{nd}}(i) \tag{2.21}$$

$$\sigma^2_{nd} = \sum_{i=0}^{16} f_{V_{nd}}(i)(i - we_d)^2 \tag{2.22}$$

De forma similar que para los pesos de la diagonal, presentamos gráficas que muestran las distribuciones para una simulación en MatLab (mostrados con $\circ$) y los calculador por medio de las expresión 2.20 (mostrados con *).

Por ejemplo en la Fig. 2.9 se muestran las distribuciones de probabilidad de los pesos sinápticos fuera de la diagonal para una red neuronal de tamaño $N = 100$, con tamaño de ensambles de $M = 20$ y número de ensambles $K = 30$. En esta gráfica la mayoría de los pesos sinápticos son pequeños y nuevamente es difícil apreciar la forma de la distribución.

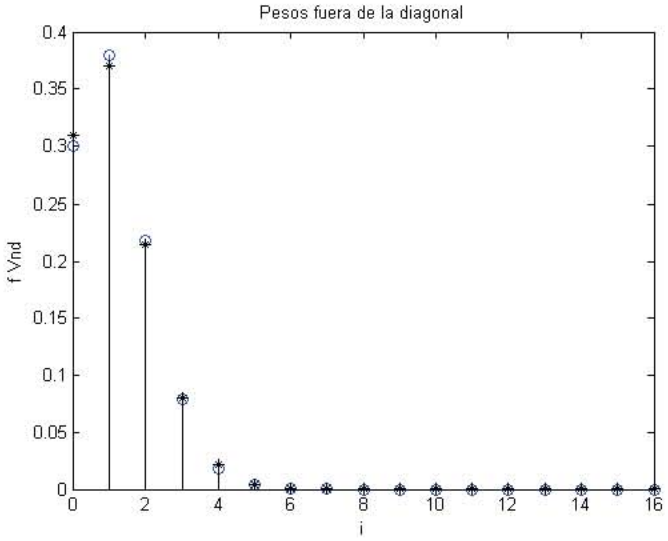


Figura 2.9: Distribuciones de probabilidad para $N = 500$, $M = 30$ y $K = 50$

Otro ejemplo se muestra en la Fig. 2.10 para una red neuronal de tamaño $N = 100$, número de ensambles $K = 30$ y tamaño de ensamble $M = 30$. En estás gráficas se observa que la distribución es nuevamente aproximadamente normal (resultado de la sumatoria de muchas exponenciales). La distribución queda siempre acotada entre $0 \leq i \leq 16$ por las limitaciones impuestas.

En cuanto al los estadísticos de la distribución $f_{V_{nd}}$, la Tabla 2.2 muestra la comparativa de los valores calculados a partir de las ecuaciones 2.21 y 2.22 y los datos de una simulación realizada con el programa en MatLab. Se observa que los valores son muy aproximados y que efectivamente se describe la distribución probabilística de los pesos sinápticos una vez más.

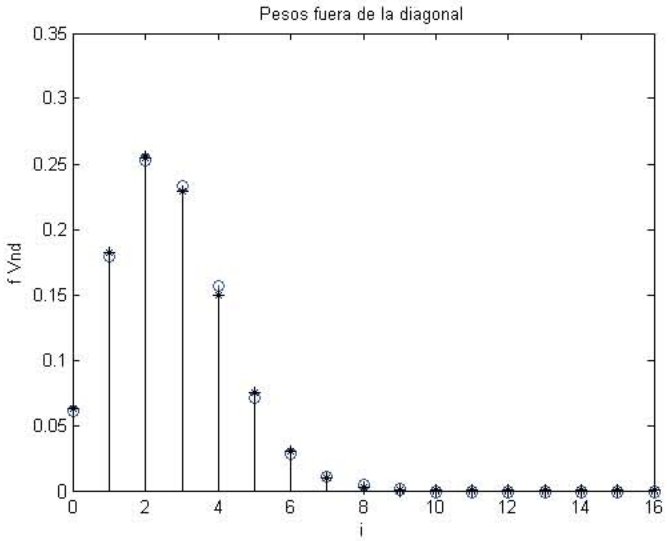


Figura 2.10: Distribuciones de probabilidad para $N = 100$, $M = 30$ y $K = 30$

Tabla 2.2: Estadísticos de pesos sinápticos fuera de la diagonal principal con MatLab y ecuaciones

Tamaño de red N	Tamaño de ensamble M	Número de ensambles K	Media we_{nd}		Varianza σ^2_{nd}	
			Simulación	Ecuaciones	Simulación	Ecuaciones
100	20	30	1.1533	1.1515	1.1304	1.1073
		50	1.9154	1.9192	2.001	1.8455
	40	30	2.633	2.6364	2.1268	2.4047
		50	4.4145	4.3939	4.5931	4.0078
400	20	30	0.071	0.0714	0.0704	0.0713
		50	0.1193	0.119	0.1189	0.1188
	40	30	0.1641	0.1635	0.1641	0.1626
		50	0.2735	0.2726	0.2766	0.2711
200	40	60	2.3624	2.3518	2.3538	2.2596
1000	100	40	0.3963	0.3964	0.3962	0.3925

En el análisis de los estadísticos de la distribución de los pesos sinápticos fuera de la diagonal principal podemos destacar la interpretación de la media we_{nd}, ya que este valor nos índica el grado de conexión de cada neurona, es decir, si $we_{nd} < 1$ se entiende que cada neurona no está conectada con el resto de las neuronas. Para ejemplificar esto, tomemos el ejemplo en que $we_{nd} = 0.2$, este valor lo podemos interpretar que de cada 10 pesos sinápticos fuera de la diagonal principal, dos tendrían un valor de 1, es decir, cada neurona está conectada con dos neuronas más y con pesos distintos de cero.

Aclaramos una vez más que este análisis estadístico de las distribuciones de los pesos sinápticos servirá de base para un futuro análisis matemático de la capacidad de la red neuronal, sin embargo, por lo pronto, se usan estos resultados únicamente para describir e ilustrar la restauración de ensambles en la siguiente sección de reconocimiento.

2.2.3. Reconocimiento

Como se ha explicado, la red neuronal con ensambles funciona como una memoria autoasociativa. Al presentar el vector de entrada X se espera que la respuesta sea igual que la entrada $E = A_X$, siendo A_X el ensamble más correlacionad (similar) al vector X. Por lo que se dice que se reconocen los ensambles.

Si probamos específicamente un vector de entrada $X = A_k$, la respuesta se calcula como $E = A_k^T W$, es decir, se presenta a la entrada exactamente uno de los ensambles de entrenamiento (ausencia de ruido).

Para ejemplificar lo que sucede, retomaremos el ensamble A_1 y la matriz W ya entrenada de las ecuaciones **??** y **??**. Esto es para una red neuronal de tamaño $N = 10$, considerando $K = 4$ ensambles de tamaño $M = 4$. Al realizar el producto $Y_1 = A_1^T W$:

$$\mathrm{Y}_1 = {A_1}^T W = \begin{bmatrix}0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0\end{bmatrix} \times \begin{bmatrix} 2 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ ⓪ & ① & ① & ① & ⓪ & ⓪ & ⓪ & ① & ⓪ & ⓪ \\ ① & ① & ③ & ① & ① & ② & ⓪ & ① & ⓪ & ① \\ ⓪ & ① & ① & ① & ⓪ & ⓪ & ⓪ & ① & ⓪ & ⓪ \\ 1 & 0 & 2 & 0 & 2 & 2 & 0 & 0 & 0 & 1 \\ 1 & 0 & 2 & 0 & 2 & 2 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ ① & ① & ① & ① & ⓪ & ⓪ & ⓪ & ② & ① & ① \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 2 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \\ 6 \\ 4 \\ 1 \\ 2 \\ 0 \\ 5 \\ 1 \\ 2 \end{bmatrix} \tag{2.23}$$

Se encerraron en círculos los valores de los pesos que contribuyen al vector resultante Y_1. Se nota que para las posiciones y_n donde las neuronas del ensamble a_n se suma un valor de la diagonal principal y el resto de los valores que se suman son de al menos 1. Por ejemplo para formar el valor $y_3 = 1 + 3 + 1 + 1$, el valor de 3 corresponde al valor de la diagonal principal y los demás 1's corresponden a la suma de los $M - 1$ neuronas.

En cambio para las posiciones que no corresponden a las neuronas activas dentro del ensamble, en este caso sólo se suman las M neuronas por los pesos sinápticos que son en general menores que 1. Por ejemplo, para $y_1 = 0 + 1 + 0 + 1$. De esta forma, el vector Y_1 tendrá valores mayores en los elementos que corresponden a las neuronas activas del ensamble A_1 que en el resto de posiciones.

La designación del umbral θ se realiza para que únicamente permanezcan M o menos neuronas activas en el vector $E_1 = S(Y_1)$, recordando que $S()$ es la función de activación de las neuronas.

$$S\{y_i\} = \begin{cases} 1, & \text{si} \quad y_i \geq \theta, \\ 0, & \text{si} \quad y_i < \theta, \end{cases} \; i = 1, 2, 3, ..., N, \tag{2.24}$$

Se puede realizar un histograma de valores del vector Y_1 para determinar el valor más apropiado de θ que cumpla la condición establecida. Para el ejemplo, el histograma se presenta en la Fig. 4.6.

Una vez que contamos con este histograma $hist(n)$, calculamos el valor del umbral θ como el máximo valor m del histograma, cuya sumatoria $\sum_{j=6}^{m} hist(j) \leq M$, donde M es el tamaño del ensamble neuronal. Para el ejemplo planteado, el valor de $\theta = 4$, ya que al establecer este umbral obtendremos el vector

$$E_1 = S(Y_1) = \begin{bmatrix}0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0\end{bmatrix},$$

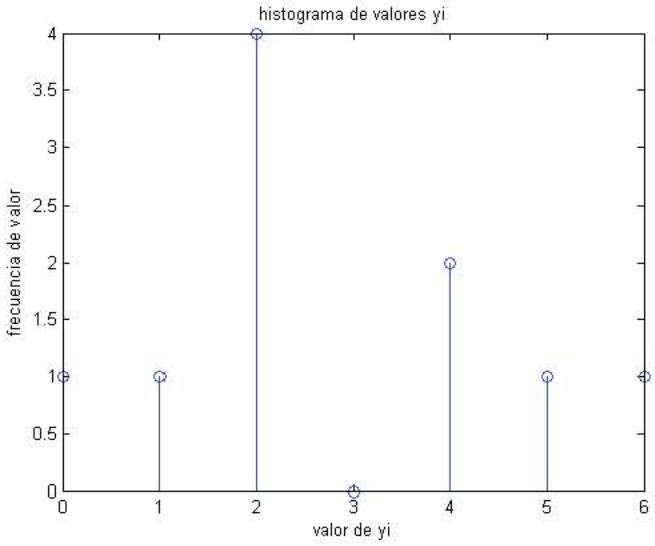


Figura 2.11: Ejemplo de histograma de valores de vector Y

que contiene únicamente $M = 4$ neuronas activas y es precisamente el ensamble original A_1 esperado, porque es una memoria autoasociativa.

Esto se complica más cuando agregamos ruido en el vector de entrada de la red neuronal. Es decir, de las M neuronas activas, se desactivaran R neuronas y R neuronas inicialmente inactivas se activaran formando el vector $X = A_k^R$. Usando nuevamente el ejemplo para ilustrar lo anterior, supongamos que al ensamble A_1 se modifica el valor de una neurona para obtener el vector A_1^1 como se observa en la Ecuación 2.27.

$$A_1 = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad A_1^1 = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \tag{2.25}$$

Como se aprecia el elemento a_8^1 cambio su valor de 1 a cero en comparación con a_8, asimismo, el elemento a_9^1 cambio de 0 a 1 en comparación con el elemento a_9. Como se cambió el valor de una neurona del total de $M = 4$ neuronas que integran el ensamble, se dice que se introdujo un nivel de ruido de $r = 100/4 = 25\,\%$ en el vector A_1^1. Llamaremos al valor r el nivel de ruido que se introduce en los ensambles.

Ahora introduciremos este vector a la red neuronal para obtener $Y_1^1 = A_1^{1^T} W$ como se muestra a continuación:

$$\mathbf{Y}_1^1 = A_1^{1^T} W = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 2 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 3 & 1 & 1 & 2 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 2 & 0 & 2 & 2 & 0 & 0 & 0 & 1 \\ 1 & 0 & 2 & 0 & 2 & 2 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 2 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 2 \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \\ 5 \\ 3 \\ 1 \\ 2 \\ 0 \\ 4 \\ 1 \\ 2 \end{bmatrix} \tag{2.26}$$

Analizaremos este caso retomando el análisis estadístico realizado anteriormente de los pesos sinápticos de la red neuronal, consideraremos los valores esperados we_d (peso sináptico esperado de la diagonal principal) y we_{nd} (pesos sináptico esperado fuera de la diagonal principal.

Para obtener el valor y_i de las neuronas que no se alteraron del ensamble original ($i = 2, 3, 4,$) se suma un valor de la diagonal principal we_d mas las $(1 - r)(M - 1)$ neuronas que no fueron alteradas con un valor de peso sináptico mayor o igual a 1 ($we_nd + 1$) y las neuronas $r(M - 1)$ neuronas alteradas por un peso sináptico probablemente pequeño we_{nd}. Llamaremos a este tipo de valores (y_2, y_3, y_4 en nuestro ejemplo) como el nivel de información correcta I_c dentro del vector Y_k^R y lo calcularemos como:

$$I_c = we_d + (1 - r)(M - 1)(1 + we_{nd}) + r(M - 1)we_{nd} \tag{2.27}$$

Para obtener el valor y_8 que es el valor que se quiere restaurar, se suman las $(1 - r)(M - 1)$ neuronas que no se alteraron multiplicados por pesos sinápticos de valor mayor o igual a 1 $(1 + we_{nd})$ más un $r(M - 1)$ multiplicado por un peso sináptico muy probablemente menor que los anteriores (we_{nd}). Llamaremos a este tipo de valores como el nivel de información por restaurar I_r dentro del vector Y_k^R y lo calcularemos como:

$$I_r = (1 - r)(M - 1)(1 + we_{nd}) + r(M - 1)we_{nd} \tag{2.28}$$

Para el valor y_9 que corresponde a información que se pretende atenuar, ya que en este caso se tendrá nuevamente la suma de un peso sináptico de la diagonal principal (we_d) más las $(M - 1)$ neuronas restantes por los pesos sinápticos correspondientes (we_{nd}), que en general son poco probables. Denominaremos este tipo de valores como el nivel de información por atenuar I_a dentro del vector Y_k^R y lo aproximaremos como:

$$I_a = we_d + (M - 1)we_{nd} \tag{2.29}$$

Para los demás valores y_i que no contienen neuronas activas $i = 1, 5, 6, 7, 10$ se tendrá únicamente ruido, que será la multiplicación de las M neuronas activas por pesos sinápticos probablemente pequeños (we_{nd}). Llamaremos a estos valores como información de ruido I_n y se calculará simplemente como:

$$I_n = M(1 + we_{nd}) \tag{2.30}$$

Se pretende entonces que el nivel de la información por restaurar I_r sea mayor que el nivel de la información por atenuar y además del nivel de la información de ruido I_n de tal forma que al comparar con el umbral θ, se logrará la activación de las neuronas para los niveles de la información correcta I_c y la información por restaurar I_r. De esta forma se restaurará el ensamble A_k original.

Para el ejemplo que se ha desarrollado, calculamos el valor del umbral como $\theta = 3$ de forma que la salida de la red neuronal es:

$$E_1^1 = S(Y_1^1) = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Este vector se restauró completamente en el ensamble original A_1 a pesar del ruido introducido.

Como se aprecia en el ejemplo, la capacidad de restauración de la red neuronal dependerá nuevamente de sus parámetros: tamaño de red neuronal N, tamaño de ensambles M y número de ensambles K, además del nivel de ruido r que se agregue a la entrada de la red neuronal.

Para probar la confianza de las ecuaciones 2.27 - 2.30, se emplea nuevamente el programa en MatLab, para comparar los valores esperados de los distintos niveles de información contra los valores promedios calculados en simulación directa, considerando un nivel de ruido de $r = 0.5$. Estos resultados se muestran en la Tabla 2.3.

Tabla 2.3: Estadísticos para la etapa de reconocimiento con MatLab y ecuaciones

Tamaño de red N	Tamaño de ensamble M	Número de ensambles K	Información correcta I_c		Información por restaurar I_r		Información por atenuar I_a		Información de ruido I_n	
			Simulación	Ecuaciones	Simulación	Ecuaciones	Simulación	Ecuaciones	Simulación	Ecuaciones
100	20	30	30.9	37.3788	29.1	31.3788	27	27.8788	22.7143	23.0303
		50	55.5	55.94	48.8	45.9646	49.1	46.44	38.6571	38.3838
	40	30	104.5333	99.9516	96.4667	90.9545	74.8667	85.4516	74.7636	79.0909
		50	155	156.0704	149.4667	141.9242	169.7333	141.5704	127.6	131.8182
400	20	30	10.5	12.3571	10.7	10.8571	3.25	2.8571	1.7806	1.4286
		50	16.2	14.2619	11.9	11.7619	4.125	4.7619	2.9395	2.381
	40	30	20.4667	21.4925	18.6	19.2425	6.4667	6.9925	5.4508	4.906
		50	24.8667	26.1541	21.6667	22.4041	8.6429	11.6541	7.3017	8.1767
200	40	60	112.75	123.0578	110.05	111.2186	112.95	103.5578	93.4286	94.0704

Los valores calculados y los obtenidos en las simulaciones aún mantienen un margen de error considerable. Es posible que las proposiciones hechas para los niveles de información se analicen con más detalle, considerando la desviación estándar de las distribuciones y así establecer un rango de valores para los niveles con mayor confiabilidad y precisión.

A pesar de esto, el procedimiento indica que es probable que se restaure un porcentaje de neuronas del ensamble original al usar la red neuronal, lo que equivale a disminuir el nivel de ruido r. Por lo que se implementa el sistema de forma recursiva, introduciendo la respuesta de la red neuronal nuevamente a la entrada. Bajo el análisis hecho, el nivel de ruido r tenderá a disminuir y finalmente es posible restaurar en gran medida el ensamble original.

De esta forma se implementa la recursividad como se ilustra en la Fig. 2.12, al finalizar P ciclos de recursividad, se obtiene el vector final E_k que es el que se compara con el ensamble original A_k por medio del producto interno. Si el producto interno indica que al menos el 90 % de las neuronas

del ensamble A_k están activas en la salida E_k, esto es $(A_k \cdot E_k) \geq 0.9M$, entonces el ensamble ha sido restaurado satisfactoriamente y se ha reconocido, de lo contrario, se ha cometido un error.

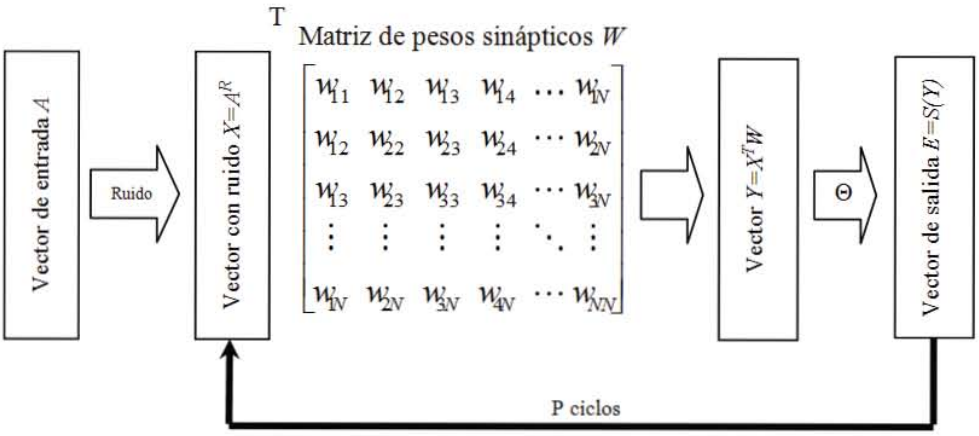


Figura 2.12: Diagrama de red neuronal recursiva

El análisis matemático para demostrar lo que se expresa aquí está aún pendiente y es una línea de investigación a futuro que ayudará a describir formalmente el comportamiento del paradigma de red neuronal que se emplea. Consideramos que el análisis presentado es una primera aproximación que nos permite entender el funcionamiento interno de la red neuronal como memoria asociativa y su capacidad de restaurar ensambles neuronales, pero puede ser desarrollado con mayor profundidad en el futuro.

A continuación se presentan los experimentos que muestran la capacidad informativa de la red neuronal.

2.3. Capacidad informativa

La capacidad informativa C de la red neuronal se define como el número máximo de ensambles A_k que se pueden guardar en la red neuronal permitiendo su restauración y recuperación en al menos un 90 % de las neuronas para nuestro paradigma. Trabajos anteriores sobre esta línea de investigación han probado capacidades varias, como la red neuronal de Hopfield [13] que presenta una capacidad aproximada de $C \leq Nln(2) \approx 0.7N$ [30], donde N es el tamaño de la red neuronal, o el modelo de Willshaw que cuenta con una capacidad informativa de $C \leq N/(ln(2)e) \approx 0.53N$[32], donde N es el tamaño de la red neuronal [31].

En nuestra investigación se considera el valor de $P = 5$ como el valor óptimo de ciclos recursivos de la red neuronal para restaurar la información de los ensambles. El nivel de ruido que se emplea en los experimentos es de $r = 0.5$, es decir, la mitad de las neuronas activas en los ensambles son alteradas y remplazadas por otras neuronas. Se prueba la capacidad informativa de la red neuronal con un programa escrito en C++ (Apéndice B) para diferentes valores de tamaños de red N y tamaños de ensambles M.

Los primeros experimentos para investigar la capacidad informativa han mostrado resultados prometedores. Los resultados se presentan en la Tabla 2.4 para ensambles independientes.

Tabla 2.4: Resultados (tamaño de red neuronal $N = 12000$)

Tamaño de ensamble M	Número de ensambles K	Número de errores	Ensambles reconocidos (%)
14	6000	59	99.017
32	7500	65	99.133
64	5000	42	99.16

Palm and Sommer [34] determinaron que el tamaño óptimo de ensambles es:

$$M = \frac{ln(N)}{ln(2)}, \tag{2.31}$$

donde N es el tamaño de la red neuronal.

Como se muestra en la Tabla 2.4, el tamaño óptimo de los ensambles es $M = 32$, en este caso la red neuronal almacenó hasta $K = 7500$ ensambles con un porcentaje de error menor a 1 %. En contraste con el tamaño de ensamble $M = 14$ que sería el óptimo de acuerdo a la ecuación 2.31.

La Tabla 2.5 muestra algunos experimentos donde el número de ensambles almacenados es mayor que el tamaño de la red neuronal.

Tabla 2.5: Resultados (tamaño de red neuronal $N = 24000$)

Tamaño de ensamble M	Número de ensambles K	Número de errores	Ensambles reconocidos (%)
15	11800	104	99.1186
32	65080	624	99.0412
64	35400	288	99.1864

En la Tabla 2.5 los resultados muestran que para una red neuronal de tamaño $N = 24000$ se almacenaron hasta $K = 65080$ ensambles satisfactoriamente de tamaño $M = 32$ y pasa el tamaño de ensamble $M = 64$ fue posible guardar $K = 35400$ ensambles. Estos experimentos nos dan una evaluación del potencial de reconocimiento de patrones de estas redes neuronales, donde es posible guardar y distinguir un gran número de ensambles neuronales.

Este potencial resulta atractivo para emplear la red neuronal con ensambles en aplicaciones prácticas, en las que los datos deben ser codificados en ensambles. Es importante mencionar que la información del mundo real contiene un gran nivel de correlación entre sí, por lo que la capacidad informativa disminuye, a diferencia de cuando se usa con información estadísticamente independiente.

En los próximos capítulos se describe la aplicación que servirá de prueba para la red neuronal con ensambles que se ha descrito.

Capítulo 3

Simulador de robot móvil

En este capítulo describimos el simulador del robot móvil que se desarrollo para probar la red neuronal con ensambles en una tarea real y estimar la capacidad informativa para datos correlacionados recolectados con este simulador. El simulador fue desarrollado en C++ y el código se puede observar en al Anexo C.

Como propósito de la red neuronal con ensambles se propone ser aplicada al problema específico de un robot móvil evasor de obstáculos. Sistemas basados en redes neuronales han sido desarrollados para el control de robots [35] y [36] o bien de brazos robóticos [37]. En este caso se propone usar la metodología descrita en el Capítulo 2 para resolver este tipo de problemas debido a la gran capacidad de almacenamiento que se tiene para datos no correlacionados, además, esta aplicación nos permitirá realizar pruebas de la capacidad informativa para datos estadísticamente correlacionados.

Es así como describiremos en este capítulo el desarrollo del simulador que nos proveerá la base de datos y permitirá probar la aplicación de la red neuronal en el problema práctico del robot móvil evasor de obstáculos.

3.1. Funciones

Las funciones que debe realizar el simulador de robot móvil son las siguientes:

1. Generación de situaciones aleatorias con obstáculos. Esta función nos permite generar escenarios posibles de obstáculos que debe enfrentar el robot móvil.

2. Ejecutar maniobras del robot móvil dentro del escenario con obstáculos a partir de datos proporcionados por el usuario. Esta función nos permite generar una base de datos de maniobras que corresponden a situaciones específicas, proporcionadas por un usuario que resuelva cada situación e introduzca los datos correspondientes.

3. Almacenamiento de situaciones y maniobras correspondientes en una base de datos. Esta función permite almacenar la base de datos en un archivo que posteriormente se codificará y se usará para el entrenamiento de la red neuronal con ensambles.

4. Ejecución de maniobras a partir de la salida de la red neuronal. Esta función nos permite probar el desempeño de la red neuronal ante situaciones verdaderas, sin importar si fueron parte del entrenamiento o no.

A continuación se describe con más detalle las características del simulador gráfico para poder cumplir con estas funciones.

3.2. Características

El simulador es un ambiente gráfico en el que se presenta un área de 400x500 pixeles de forma rectangular, esta es el área válida para el desplazamiento del robot, es decir no se pueden rebasar estos límites.

Los 15 obstáculos son de forma circular de radios en el rango de R_i =[10,40] pixeles y su centro se coloca de forma aleatoria dentro del área de desplazamiento en coordenadas $[x_i, y_i]$. El robot es representado por un rectángulo de 10x20 pixeles. La Fig. 3.1 muestra una captura de pantalla del ambiente del simulador.

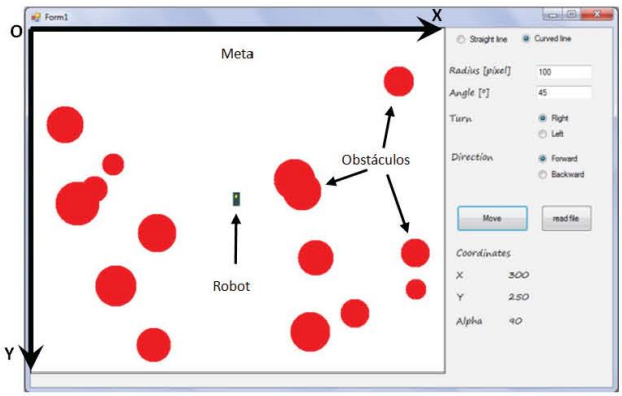


Figura 3.1: Ejemplo de situación

Inicialmente se coloca el robot en la parte central del área de desplazamiento y la meta que debe alcanzar se encuentra en la parte superior del área de desplazamiento. Se utiliza un sistema coordenado de ejes X y Y para poder describir numéricamente posiciones dentro de él.

Para describir el obstáculo O_i será suficiente contar con las coordenadas de su centro (x_i, y_i) y el valor del radio R_i.

3.3. Navegación

El robot móvil elegido está conformado por tres ruedas, una rueda móvil al frente para guiar el movimiento y dos ruedas fijas atrás para impulsar al robot Fig. 3.2.

Para describir la posición del robot móvil es necesario conocer las coordenadas de su centro (x_R, y_R) y además el valor del ángulo α, que es el ángulo que se forma entre el eje X y la línea

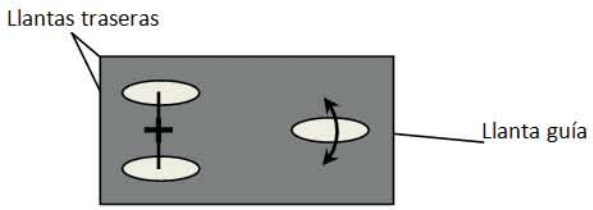


Figura 3.2: Diagrama del robot móvil

que indica el frente del robot y cruza su centro en sentido contrario a las manecillas del reloj. Estas coordenadas se muestran en la Fig. 3.3.

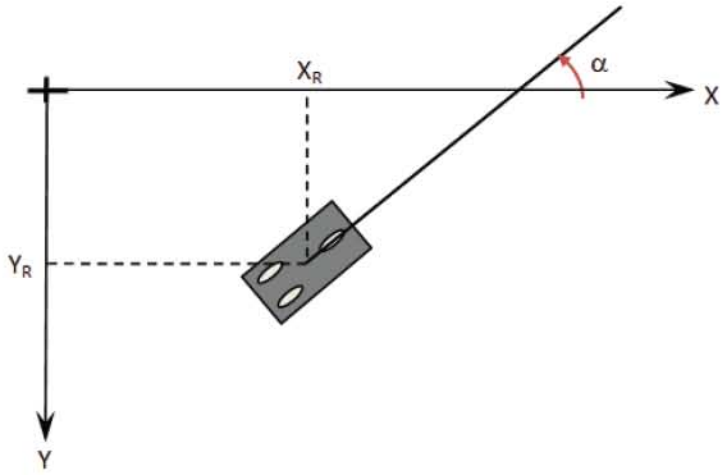


Figura 3.3: Coordenadas de posición del robot móvil

Por las características físicas del robot, este se puede desplazar únicamente de dos formas posibles y los parámetros iniciales con los que describiremos el desplazamiento son $\{k_1, k_2, k_3, RR, \phi\}$. A continuación se detallan los dos tipos de maniobras posibles y los valores de los parámetros.

1. Sobre un arco ($k_1 = -1$), ya sea hacia adelante ($k_2 = 1$) o hacia atrás ($k_2 = -1$). Además este movimiento puede realizarse hace la derecha ($k_3 = -1$) o a la izquierda ($k_3 = 1$). El ángulo del arco ϕ y el radio de la circunferencia RR completan la definición de la maniobra. Un ejemplo de este tipo de movimiento se ilustra en la Fig. 3.4, donde los valores de las constantes son $k_1 = -1, k_2 = 1$ y $k_3 = -1$.

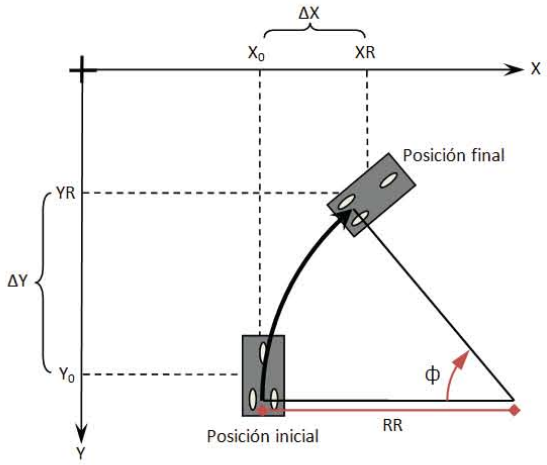


Figura 3.4: Ejemplo de desplazamiento sobre un arco

2. En línea recta ($k_1 = 1$), ya sea hacia adelante ($k_2 = 1$) o hacia atrás ($k_2 = -1$). Por último el valor de la distancia RR que recorrerá el robot móvil. En este tipo de movimiento no se necesitan más parámetros, por lo que k_3 y ϕ no son tomados en cuenta. La Fig. 3.5 muestra un ejemplo de este tipo de movimiento, donde los valores de las constantes son $k_1 = 1$, $k_2 = 1$ y k_3 no importa porque es un movimiento rectilíneo.

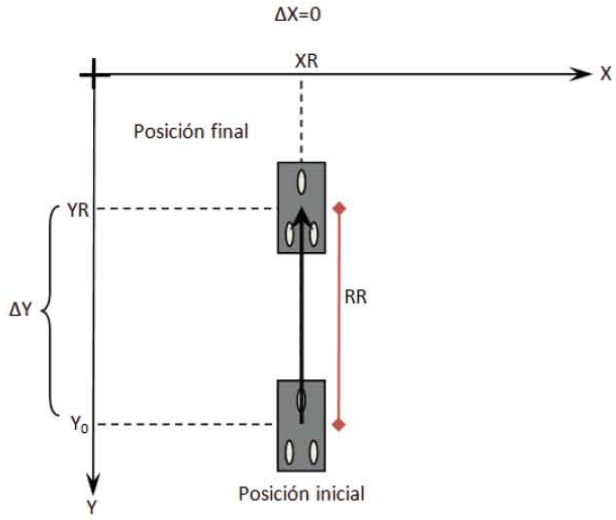


Figura 3.5: Ejemplo de desplazamiento en línea recta

El objetivo de las simulaciones es que el robot móvil alcance la parte superior del área de desplazamiento sin colisionar con ningún obstáculo que se presente en su camino. Para completar el objetivo, se hará uso únicamente de una maniobra, lo cual corresponde a la división de una tarea total en subtareas más simples.

Debido a que conocemos la posición inicial del robot (centro del área), es suficiente conocer la posición final del robot móvil (RX, RY) y el valor de RR para describir por completo la maniobra. Como se ilustró en las Fig. 3.4 y 3.5, los valores de ΔX y ΔY nos ayudan a conocer los valores de los parámetros k_1, k_2 y k_3. En cuanto al valor de ϕ, se puede determinar a partir de los incrementos ΔX y ΔY.

Las expresiones matemáticas que relacionan estos parámetros son:

$$\phi = asin(\Delta Y/RR), \tag{3.1}$$

$$k_1 = \begin{cases} 1, & \text{si} \quad \Delta X = 0, \\ -1, & \text{si} \quad \Delta X \neq 0, \end{cases} \tag{3.2}$$

$$k_2 = \begin{cases} 1, & \text{si} \quad \Delta Y > 0, \\ -1, & \text{si} \quad \Delta Y \leq 0, \end{cases} \tag{3.3}$$

$$k_3 = \begin{cases} 1, & \text{si} \quad \Delta X < 0, \\ -1, & \text{si} \quad \Delta X > 0. \end{cases} \tag{3.4}$$

Así entonces podemos describir una maniobra en su totalidad por medio de los tres parámetros XR, YR y RR.

Para la obtención de la base de datos, un usuario que conozca las características del robot móvil para controlar su desplazamiento debe proporcionar los datos necesarios para esquivar a los obstáculos y aproximar el robot lo más posible al objetivo. Para cada situación aleatoria que se genere, el usuario deberá proporcionar los datos adecuados para lograr la mejor aproximación.

La Fig. 3.6 muestra un ejemplo donde el robot móvil alcanza la meta por medio de un movimiento rectilíneo. Para este caso $k_1 = 1$, $k_2 = 1$, $RR = 200$, (k_3 y ϕ no son considerados porque $k_1 = 1$).

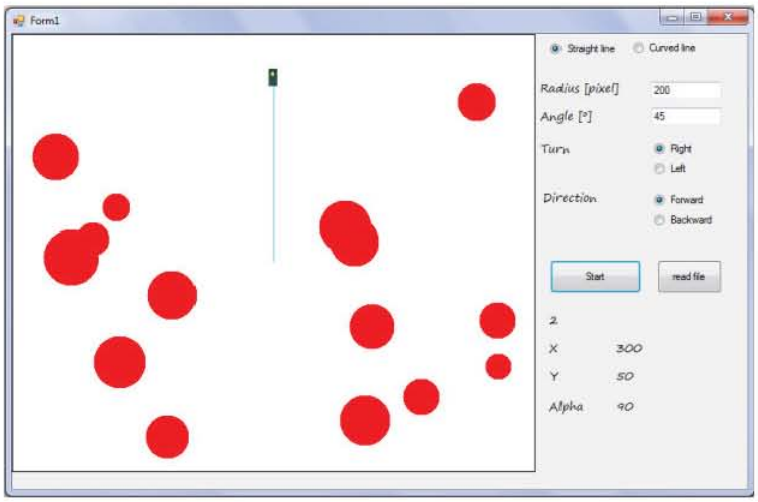


Figura 3.6: Ejemplo de maniobra rectilínea

Otro ejemplo se muestra en la Fig. 3.7, donde los obstáculos obstruyen la ruta del robot, entonces es necesaria una maniobra sobre un arco para evitar la colisión. En este caso, los valores de son $k_1 = -1$, $k_2 = 1$, $RR = 200$, $\phi = 77$.

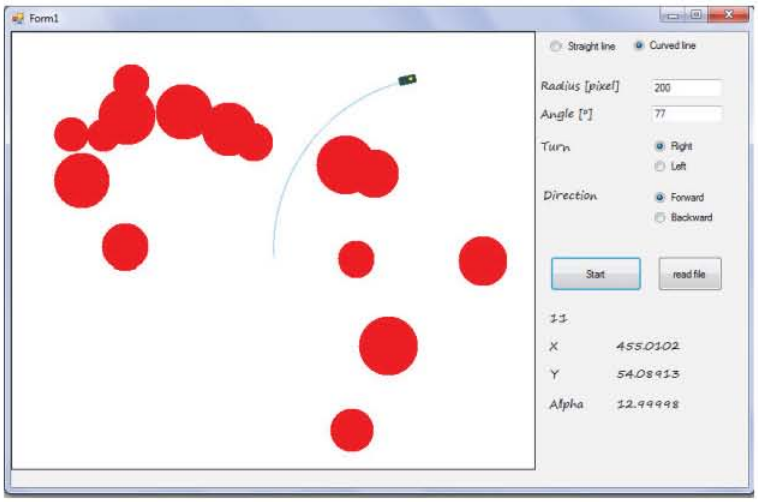


Figura 3.7: Ejemplo de maniobra sobre un arco

Para describir una situación, reduciremos el ambiente a la descripción de los tres obstáculos que se localicen más próximos al robot. Las características de estos obstáculos con respecto al robot móvil son las más importantes para decidir la maniobra más adecuada.

Entonces definiremos la situación por medio de tres parámetros relacionados a los obstáculos (con los valores de los tres obstáculos por parámetro), estos son:

1. Los radios de los obstáculos $\{R_{O1}, R_{O2}, R_{O3}\}$,
2. Las distancias entre el centro del robot móvil y el centro del obstáculo $\{D_{RO1}, D_{RO2}, D_{RO3}\}$ y finalmente
3. El ángulo entre las distancias de los centros del robot y los obstáculos y la recta que indica el frente del robot $\{\alpha_{RO1}, \alpha_{RO2}, \alpha_{RO3}\}$.

Estos parámetros se ilustran en la Fig. 3.8.

Entonces para definir una situación S_i se necesitará el conjunto de tres parámetros con nueve valores en total (tres valores por parámetro): $\{R_{O1}, R_{O2}, R_{O3}, D_{RO1}, D_{RO2}, D_{RO3}, \alpha_{RO1}, \alpha_{RO2}, \alpha_{RO3}\}$.

Para cada situación S_i existe una maniobra M_i correspondiente descrita por tres parámetros y un valor por parámetro: $\{XR, YR, RR\}$.

Por medio de estos conjuntos de valores de parámetros se conforma una base de datos a partir de situaciones aleatorias, para las cuales un usuario experto eligió la maniobra más adecuada para alcanzar la meta del robot móvil por medio de una sola maniobra.

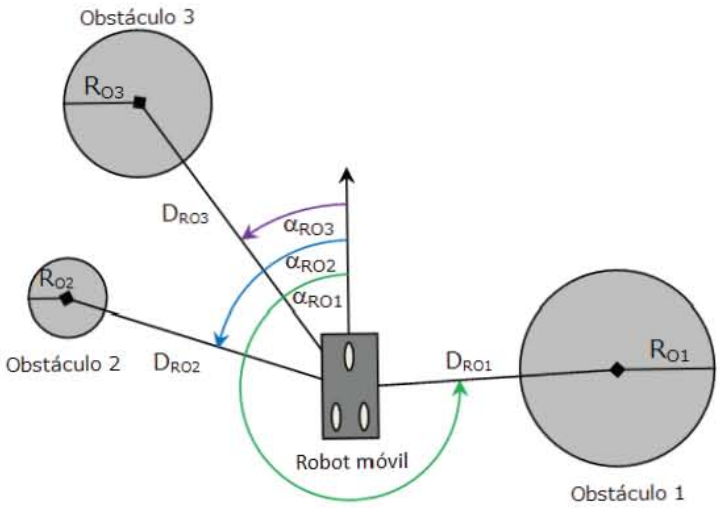


Figura 3.8: Parámetros que definen la situación del robot móvil

Capítulo 4

Codificación de maniobras del robot

En este capítulo se describe la codificación de la información de la base de datos obtenida por el simulador del robot móvil en ensambles neuronales. Se muestra como se codifican las situaciones y sus respectivas maniobras para formar un sólo ensamble. La codificación permite conservar una similitud para valores cercanos y elimina la correlación para valores lejanos.

Esta etapa es fundamental para lograr llevar a cabo la tarea que se pretende y analizar la capacidad informativa de la red neuronal para datos correlacionados. Se espera que esta disminuya en comparación con los datos no correlacionados.

La codificación consta de varias etapas que se describen a continuación para lograr codificar y decodificar la información de forma cualitativa y cuantitativa de los parámetros de los movimientos del robot móvil especificados anteriormente.

4.1. Máscaras de parámetros

Para asociar propiedades cualitativas a los ensambles creamos unas máscaras iniciales de parámetros M_k. Estas máscaras son vectores del tamaño de la red neuronal N con M neuronas activas elegidas aleatoriamente. Estas máscaras de parámetros corresponden a cada uno de los tipos de parámetros: $\{M_{OD}, M_{OR}, M_{OA}, M_{XR}, M_{YR}, M_{RR}\}$.

Estos vectores son almacenados en la memoria de la computadora y se pueden leer en cualquier momento. Estas máscaras nos permiten asociar a cada tipo de parámetro un ensamble "base" a partir de los cuales se generarán los ensambles finales con los que se entrenará la red neuronal.

Un ejemplo de máscara de parámetro se muestra en la Figura 4.1 para una red neuronal de tamaño $N = 10$ y $M = 4$. Se observa que únicamente cuatro neuronas activas conforman la máscara de parámetro.

Este ejemplo es únicamente ilustrativo, ya que en la práctica el tamaño de la red neuronal será mucho mayor así como el tamaño de los ensambles. Estas máscaras son una forma de codificación distribuida que codifican propiedades por medio de varias neuronas activas, esto implica que aunque algunas neuronas no estén activas, es probable reconocer el tipo de propiedad.

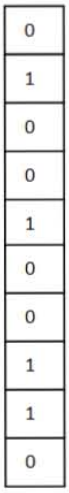


Figura 4.1: Ejemplo de máscara de parámetro

4.2. Esquemas de permutaciones aleatorias

La codificación de los valores de los parámetros se realiza por medio de permutaciones aleatorias, así podemos realizar una codificación cuantitativa. Este tipo de codificación fue realizada con permutaciones aleatorias, como las empleadas en el neuroclasificador PCNC [38].

Este tipo de codificación permite que se guarde cierta similitud entre valores cercanos y la elimina para valores lejanos. Es necesario mantener similitud (correlación) entre la codificación de valores cercanos para tolerar errores y asociar valores cercanos, sin embargo, como no elimina del todo la correlación de la información tiene impacto en la capacidad informativa de la red.

Este método es también un sistema de codificación distribuida. Así se optimiza la capacidad informativa de la red y mantiene en cierto rango la posibilidad de asociar valores próximos.

Para este propósito, se define un esquema de permutaciones aleatorias P_k para cada máscara de parámetro M_k que reacomoda los elementos del vector. Cada permutación se puede aplicar la cada máscara de parámetro correspondiente: $P_k\{M_k\}$.

Se generan los esquemas de permutaciones aleatorias $\{P_{OD}, P_{OR}, P_{OA}, P_{XR}, P_{YR}, P_{RR},\}$.

Por ejemplo, para una red neuronal de tamaño $N = 10$, un esquema de permutaciones aleatorias se presenta en la Figura 4.2. Como se ilustra, cada elemento del vector es permutado en una nueva posición, este esquema se genera de forma aleatoria una única vez y se almacena en memoria de la computadora.

Estas permutaciones se aplican a sus correspondientes máscaras de parámetros únicamente. Las permutaciones se aplican y se obtienen los vectores para diferentes valores de parámetros de la siguiente forma.

Para codificar un valor V de un parámetro k, tomamos la máscara M_k y el esquema de permutación aleatorio P_k correspondiente al tipo de parámetro. La máscara inicial M_k será permutada bajo la especificación P_k un número PE veces de forma completa y al final se permutarán únicamente los primeros PP elementos iniciales del vector resultante de las permutaciones anteriores.

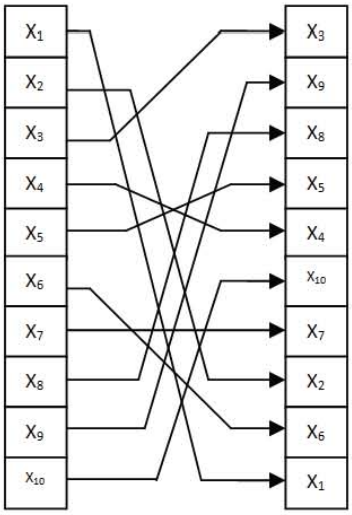


Figura 4.2: Ejemplo de permutaciones aleatorias

El número de ciclos de permutaciones completas que se aplica a la máscara de parámetro es

$$PE = int\left(\frac{V}{G}\right), \tag{4.1}$$

donde int() denota la parte entera, V es el valor del parámetro a codificar y G es el valor de separación de correlación. Este valor es el rango dentro del cual los valores codificados presentarán un nivel de correlación y fuera de este valor los vectores pierden correlación alguna.

El número de permutaciones parciales se calculan como:

$$PP = N\left(\frac{V - PE}{G}\right), \tag{4.2}$$

donde N es el tamaño de la red neuronal, V es el valor del parámetro a codificar, PE es el número de permutaciones completas a realizar (ecuación 4.1) y G es el valor de separación de correlación.

Para codificar los valores, el esquema de permutación se aplica a la máscara de parámetro PE veces y después, se aplican las permutaciones una vez más pero a los primeros PP elementos del vector, sin alterar el resto de elementos.

Consideremos el caso donde el parámetro RR tiene el valor $V = 21$, si la máscara de parámetro mostrada en la Figura 4.1 es M_{RR} y las permutaciones aleatorias mostradas en la Figura 4.2 es P_{RR}, considerando el valor de separación de correlación $G = 10$, por medio de las ecuaciones 4.1 y 4.2 obtenemos $PE = 2$ y $PP = 1$. Esto significa que el esquema de permutación debe ser aplicado a la máscara de parámetro dos veces de forma completa y entonces, el primer elemento será permutado una vez más hasta obtener $M^T_{RR=21} = \begin{bmatrix}1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1\end{bmatrix}$. Este ejemplo se ilustra en la Figura 4.3.

Si para los mismos parámetros de la red neuronal $N = 10$, $M = 4$ y $G = 10$ codificamos nuevamente M_{RR} para un valor de $V = 25$ se obtienen los valores $PE = 2$ y $PP = 5$ como se ilustra en la Figura 4.4. El vector resultante $M^T_{RR=25} = \begin{bmatrix}1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1\end{bmatrix}$ es muy

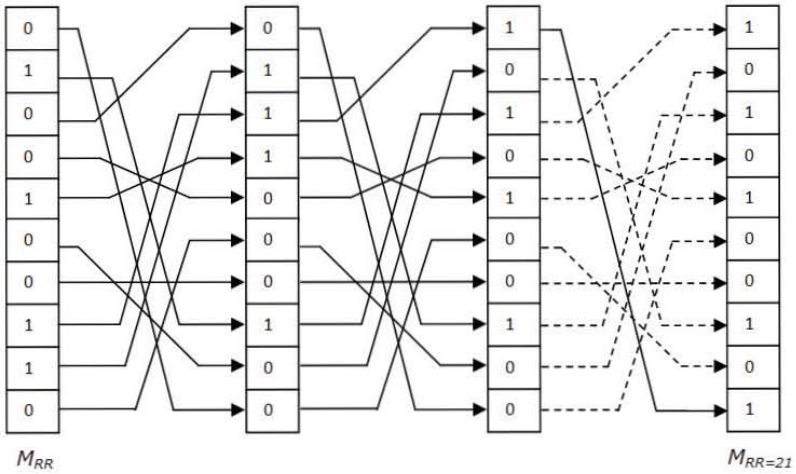


Figura 4.3: Codificación de valor por permutaciones aleatorias para $RR = 21$

similar al vector calculado en el ejemplo anterior $M_{RR=21}$ ya que son valores cercanos, de forma matemática se puede obtener el grado de similitud por medio del producto interno $(M_{RR=21} \cdot M_{RR=25}) = 3$.

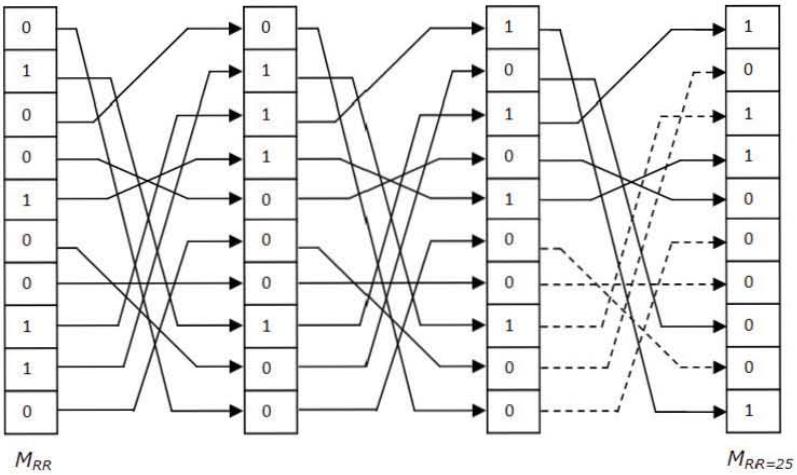


Figura 4.4: Codificación de valor por permutaciones aleatorias para $RR = 25$

Y finalmente para codificar un valor de $RR = 10$ por medio de permutaciones aleatorias con las mismas especificaciones de la red neuronal que en los casos anteriores, se obtienen los valores $PE = 1$ y $PP = 0$. Por lo que se aplican las permutaciones una sola vez a la máscara inicial para obtener $M^T_{RR=10} = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$ como se ilustra en la Figura 4.5. Este valor $V = 10$ se encuentra más lejano de los valores anteriores por la separación de correlación G, por lo que este vector no debe presentar una gran similitud con los vectores codificados anteriormente. Por medio del producto interno $(M_{RR=10} \cdot M_{RR=21}) = (M_{RR=10} \cdot M_{RR=25}) = 2$ que es menor en comparación con $(M_{RR=21} \cdot M_{RR=25})$.

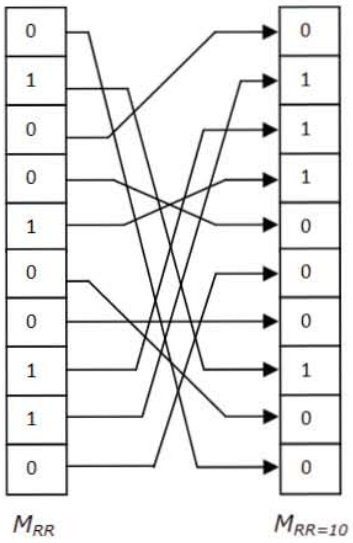


Figura 4.5: Codificación de valor por permutaciones aleatorias para $RR = 10$

En estos ejemplos, el efecto del valor de separación de correlación G no es muy grande debido al tamaño comparativo entre el tamaño de los ensambles $M = 4$ y el tamaño de la red $N = 10$. Estos ejemplos son únicamente ilustrativos y muestran en pequeña escala el proceso de codificación, sin embargo, para nuestro paradigma de red neuronal, el tamaño de los ensambles $M << N$ por lo que la gran mayoría de las neuronas de los ensambles están desactivadas y al momento de aplicar los esquemas de permutaciones es poco probable que las posiciones activas coincidan entre los vectores antes y después de las permutaciones.

4.3. Normalización

Es necesario recolectar todos los parámetros y sus valores para generar un ensamble de entrada a la red neuronal y es deseable que el tamaño de los ensambles sea constante M.

El tamaño de los ensambles se incrementará después del procedimiento de codificación cuando reunimos la información de varios parámetros y sus valores, por esta razón definimos el siguiente procedimiento de normalización [39].

Para el vector binario X, cuyos elementos contienen un número de 1's mayor que M, es decir $(X \cdot X) > M$, el procedimiento de normalización X =Norm(X) se define por los siguientes pasos:

1. Se calcula el vector $U = lcs(\bar{X})$, donde $lcs()$ es un corrimiento circular a la izquierda y $\bar{X}$ denota la negación lógica del vector X.

2. Se actualiza el vector X como $X = U \wedge X$, donde $\wedge$ indica la conjunción lógica elemento a elemento.

3. Los pasos 1 y 2 se repetirán mientras $(X \cdot X) > M$.

A continuación se presenta un ejemplo, supongamos que el tamaño de la red neuronal es $N = 15$ y el tamaño de los ensambles es $M = 4$. Contamos con el vector X tal que $(X \cdot X) = 8 > M$.

$$X^T = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \end{bmatrix}. \tag{4.3}$$

Calculamos el vector $U = lcs(\bar{X})$:

$$U^T = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}. \tag{4.4}$$

Y finalmente actualizamos el vector por medio de la conjunción lógica $X = U \wedge X$:

$$X^T = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}. \tag{4.5}$$

Se observa en (4.5) el vector que contiene únicamente $M = 4$ neuronas activas, es decir, cumple con el tamaño del ensamble. Estas neuronas que se mantienen activas son un subconjunto de las neuronas iniciales activas en (4.3), por lo que no se alteraron las posiciones de las neuronas, el método de normalización elige únicamente un número menor o igual a M neuronas que debe mantener activas, por lo que este procedimiento no altera la información codificada de forma significativa sino que es un método de control de actividad de la red neuronal.

Este método es muy práctico ya que elige prácticamente de forma aleatoria la neuronas que permanecen activas y a partir de un vector inicial X el vector final será siempre el mismo bajo las especificaciones del tamaño de la red neuronal N y del tamaño de ensamble M [39]. Esta característica es particularmente útil para la etapa de simulación en un programa de computadora.

4.4. Creación de ensambles

La descripción de una situación y su maniobra correspondiente es codificada como un solo ensamble. Para este propósito realizamos el siguiente procedimiento que integra toda la información.

1. Para cada parámetro y su valor V, el programa lee la máscara de parámetro correspondiente M_k y el esquema de permutación aleatoria P_k.

2. Se calculan el número de permutación completas PE y permutaciones parciales PP para codificar el valor V y las permutaciones son aplicadas a la máscara de parámetro respectiva $P_k(M_k)$. Como resultado de estos pasos obtenemos el conjunto de vectores:

$$\{X_{OD1}, X_{OD2}, X_{OD3}, X_{OR1}, X_{OR2}, X_{OR3},$$

$$X_{OA1}, X_{OA2}, X_{OA3}, X_{XR}, X_{YR}, X_{RR}\}.$$

3. Combinamos y normalizamos la información de estos vectores:

$$X_{OD} = Norm(X_{OD1} \vee X_{OD2} \vee X_{OD3});$$

$$X_{OR} = Norm(X_{OR1} \vee X_{OR2} \vee X_{OR3});$$

$$X_{OA} = Norm(X_{OA1} \vee X_{OA2} \vee X_{OA3});$$
$$X^S = Norm(X_{OD} \vee X_{OR} \vee X_{OA});$$
$$X^R = Norm(X_{XR} \vee X_{YR} \vee X_{RR});$$
$$X = Norm(X^S \vee X^R),$$

donde $\vee$ denota la disyunción lógica.

El final de esta secuencia, el ensamble X contiene la información codificada de la situación X^S y de la maniobra X^R.

De esta forma generamos K ensambles X_k para las diferentes simulaciones y sus maniobras correspondientes. Debido a que el número de simulaciones no es muy grande, cada ensamble será usado en el proceso de entrenamiento de la red neuronal diez veces.

El tamaño del ensamble X es M, debido al método de normalización descrito, podemos asumir que $M/2$ neuronas provienen de la información codificada de la situación X^S (tres parámetros con tres valores cada uno) y las otras $M/2$ neuronas provienen de la información de la maniobra X^R, que contiene tres parámetros con un valor cada uno.

Entonces podemos considerar, una vez más por la normalización empleada, que $M/6$ neuronas contienen la información de cada uno de los tres parámetros de la maniobra (XR, YR, RR).

4.5. Decodificación

La red neuronal es entrenada con los ensambles que representan diferentes situaciones y las maniobras correspondientes del robot. Entonces, cada vector codificado ${X^S}_k$ (únicamente con la información de la situación) es introducido a la red neuronal para obtener el vector de salida E_k, de estos vectores de salida debemos decodificar los parámetros relacionados a la maniobra XR, YR y RR.

Este procedimiento es un equivalente al usado para ensambles no correlacionados, ya que consideramos que $M/2$ neuronas del ensamble X corresponden a información de la situación X^S y las otras $M/2$ neuronas provienen de la información de la maniobra X^R, es decir, al introducir únicamente X^S y esperar que la red restaure el ensamble X, es análogo a la alteración del 50 % de las neuronas del ensamble empleado con los datos aleatorios.

Una vez más usamos el producto interno para comparar el nivel de traslape (correlación) entre el vector de salida E_k y el vector obtenido después de aplicar las permutaciones a la máscara M_z. Para el valor donde el traslape sea máximo, los valores de permutaciones PE_{max} y PP_{max} son usados para calcular el valor decodificado V_z como indican las siguientes ecuaciones:

$$V_z = PE_{max}G + \frac{PP_{max}G}{N}. \tag{4.6}$$

Si el error absoluto del valor decodificado y el valor real es menor que el valor de separación de correlación G, entonces se considera que el valor del parámetro ha sido decodificado exitosamente, pero si es mayor, entonces se dice que se cometió un error. El valor evalúa la exactitud en el valor decodificado. Además, por cada ensamble hay tres valores por decodificar.

La intención de introducir el vector ${X^S}_k$ y decodificar los valores de los parámetros de las maniobras a partir de los vectores de salida a través de la red neuronal es poder reconocer la situación y recuperar la maniobra almacenada con sus valores y mover el robot apropiadamente.

Como un índice adicional de desempeño, calculamos el error cuadrático medio, el cual nos indica que tan lejos están los valores decodificados de los originales, es decir, una medida de dispersión.

Debido a la naturaleza aleatoria de las máscaras de los parámetros y de las permutaciones aleatorias, con las propiedades de la red neuronal como una memoria asociativa, intentamos introducir el vector ${X^S}_k$ y obtener el vector de salida respectivo E_k cuyo contenido mantiene la información codificada de los valores de los parámetros de la maniobra. Por estas razones, la mayoría de los ensambles no sufren alteraciones y los valores decodificados presentan el máximo traslape para el valor original cuando los límites de la red no han sido excedidos.

Podemos anticipar el nivel de traslape máximo OL_{max} para la decodificación de un sólo parámetro de la maniobra, ya que del ensamble final X únicamente $M/6$ neuronas codifican un parámetro y su valor, por lo tanto podemos decir que el valor esperado del nivel de correlación sea de:

$$OL_{max} \leq M/6, \tag{4.7}$$

es decir, que coincidan todas las neuronas que codifican el parámetro y su valor.

En cuanto al nivel de traslape para el resto de posibles valores a decodificar OL_{noise} se puede calcular de la siguiente forma.

La probabilidad de que una neurona esté activa es N/M por el tamaño de ensamble y el tamaño de la red. Debido a la operación del producto interno, esta probabilidad se multiplica por sí misma y al realizar la suma de todos estos productos llegamos a la expresión:

$$OL_{noise} = (M^2)/N. \tag{4.8}$$

Estas medidas de tendencia central nos hablan únicamente de los valores esperados del traslape máximo (al reconocer el valor de un parámetro) y el traslape en los demás casos. Las medidas de dispersión deben ser analizadas para poder llegar a una expresión que especifique su comportamiento.

Los niveles de traslape entre el ruido y el nivel máximo deben ser diferenciables, por lo que el nivel del máximo debe ser mucho mayor que el valor del ruido, lo cual limita el tamaño de los ensambles por la relación cuadrada de la ecuación 4.8.

La Figura 4.6 muestra las gráficas para los niveles de correlación de los posibles valores decodificados en una red neuronal de tamaño $N = 16000$ usando $K = 30$ ensambles de tamaño $M = 120$ para los tres valores decodificados XR, YR y RR con $G = 10$. El máximo valor de correlación nos indica el valor decodificado y como se observa, es lo suficientemente grande para reconocer el valor correcto. Como se indico, el valor esperado del máximo es $OL_{max} \leq M/6 = 120/6 = 20$ y el nivel de ruido para los demás valores es de $OL_{noise} = M^2/N = 120^2/16000 = 0.8$ que se aprecian en las gráficas.

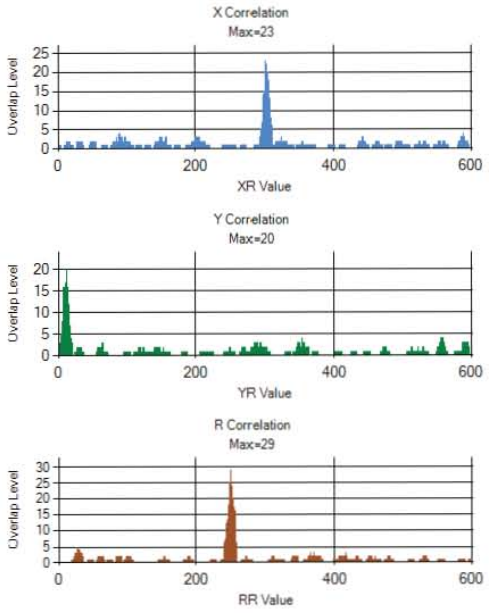


Figura 4.6: Gráficas de niveles de correlación para decodificación de valores

Para una red neuronal de tamaño $N = 8000$ con tamaño de ensambles $M = 250$ considerando un número de ensambles $K = 6$ con un valor de separación $G = 5$, los valores de traslape son $OL_{max} \leq 41.67$ y para el ruido $OL_{noise} = 7.8$. La Figura 4.7 muestra las gráficas de traslape para este ejemplo.

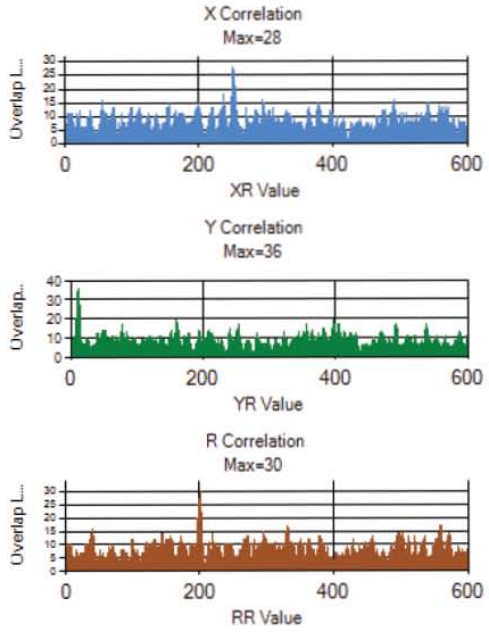


Figura 4.7: Gráficas de niveles de correlación para decodificación de valores

Los niveles de traslape obtenidos por las ecuaciones (4.7) y (4.8) deben ser probados exhaustivamente para comprobar su validez, sin embargo, se presentan como primera aproximación para mostrar el comportamiento de los niveles de traslape en la etapa de decodificación.

4.6. Capacidad informativa para datos correlacionados

A continuación se presentan los resultados para la base de datos codificada variando el número de situaciones codificados K en ensambles en una red neuronal de tamaño N y ensambles de tamaño M usando dos valores de separación de correlación G.

La primera serie de experimentos fue realizada para una red neuronal de tamaño $N = 8000$. En la Tabla 4.1 se presentan los resultados para un tamaño de ensamble $M = 60$. El mejor resultado en esta serie de experimentos es para un valor de separación de correlación $G = 10$, para el cual se pudieron almacenar $K = 20$ ensambles, es decir, hasta 20 situaciones diferentes con las maniobras respectivas del robot.

Tabla 4.1: Resultados (tamaño de red neuronal $N = 8000$ y tamaño de ensambles $M = 60$)

Valor de sep. de correlación	Número de ensambles K	Número de parámetros	Errores de valor	Errores de Valores (%)	Error cuadrático medio
10	10	30	0	0	0.658
	20	60	0	0	0.885
	30	90	3	3.3	16.607
20	10	30	0	0	2.066
	20	60	4	6.7	32.753
	30	90	22	24.4	65.358

En la Tabla 4.2 se muestran los resultados obtenidos para un tamaño de ensmable $M = 120$. Nuevamente se observa que los mejores resultados se obtienen para un valor de separación de correlación $G = 10$. Se pudieron almacenar hasta $K = 10$ ensambles que codifican situaciones y maniobras. También se puede concluir que para tamaños de ensambles mayores, la capacidad informativa de la red neuronal disminuye.

Tabla 4.2: Resultados (tamaño de red neuronal $N = 8000$ y tamaño de ensambles $M = 120$)

Valor de sep. de correlación	Número de ensambles K	Número de parámetros	Errores de valor	Errores de Valores (%)	Error cuadrático medio
10	10	30	0	0	0.447
	20	60	1	1.7	10.983
	30	90	1	1.1	9.039
20	10	30	1	3.3	17.178
	20	60	8	13.3	34.809
	30	90	26	28.9	74.162

La segunda serie de experimentos fue realizada para una red neuronal de tamaño $N = 16000$. La Tabla 4.3 presenta los resultados para un tamaño de ensamble $M = 60$, para estos parámetros, se lograron almacenar hasta $K = 20$ ensambles diferentes que codifican situaciones y maniobras, tanto para $G = 20$ como para $G = 10$, siendo para éste último valor el error cuadrático medio, es decir, cuanto menor sea el valor de separación de correlación G, la dispersión de los valores decodificados será menor.

Tabla 4.3: Resultados (tamaño de red neuronal $N = 16000$ y tamaño de ensambles $M = 60$)

Valor de sep. de correlación	Número de ensambles K	Número de parámetros	Errores de valor	Errores de Valores (%)	Error cuadrático medio
10	10	30	0	0	1.304
	20	60	0	0	1.285
	30	90	5	5.6	47.697
20	10	30	0	0	3.12
	20	60	0	0	2.63
	30	90	8	8	48.75

En la Tabla 4.4 se muestran los resultados para un tamaño de ensamble $M = 120$. En comparación con la Tabla 4.2, se observa que la capacidad informativa de la red neuronal aumenta, almacenando y recuperando hasta $K = 20$ ensambles sin errores en los valores decodificados para un valor de separación de correlación $G = 10$.

Tabla 4.4: Resultados (tamaño de red neuronal $N = 16000$ y tamaño de ensambles $M = 120$)

Valor de sep. de correlación	Número de ensambles K	Número de parámetros	Errores de valor	Errores de Valores (%)	Error cuadrático medio
10	10	30	0	0	0.447
	20	60	0	0	0.532
	30	90	1	1.1	17.402
20	10	30	0	0	0.966
	20	60	1	1.7	7.925
	30	90	13	14.4	39.413

En todos los resultados se puede observar que la decodificación fue exitosa para recuperar los valores. Comparando los experimentos se puede observar que el número de errores se incrementa cuando el tamaño de la red neuronal es más pequeño.

El valor de separación de correlación G es importante para la exactitud, para el traslape entre ensambles y la decodificación correcta de los valores, ya que especifica el grado de correlación entre vectores de acuerdo al valor numérico codificado.

Comparando las Tablas 4.1 y 4.3 por un lado, y las Tablas 4.2 y 4.4 por otro lado, podemos ver que la relación entre el tamaño de los ensambles M y el error cuadrático medio es la siguiente: entre mayor es el tamaño de los ensambles, menor es el error cuadrático medio de los valores decodificados.

Capítulo 5

Simulaciones y resultados

En este capítulo se presentan los resultados finales de los análisis de la capacidad informativa de la red neuronal tanto para datos no correlacionados como para datos correlacionados bajo la dinámica especificada en los capítulos anteriores.

Se presentan gráficas y tablas que muestran el comportamiento de la capacidad informativa de acuerdo a los parámetros de la red neuronal que son el tamaño de la red N, el número de ensambles k, el tamaño de ensambles M y finalmente el valor de separación de correlación G.

Finalmente se muestran simulaciones completas trabajando con situaciones nuevas y la red neuronal entrenada con la base de datos para mostrar las fortalezas y debilidades del sistema implementado.

5.1. Base de datos

Utilizando el simulador gráfico del robot móvil se generó una base de datos de 100 situaciones con las respectivas maniobras. Esta base de datos presentaba sin embargo la repetición de maniobras para situaciones distintas, entre ellas destaca el movimiento rectilíneo que realiza el robot para alcanzar la meta en un sólo movimiento. Esta maniobra se presentó en 49 situaciones de las 100 recolectadas.

Es deseable que las situaciones y maniobras que se usan para el entrenamiento de la red tengan un mismo número de frecuencia de ocurrencia. Esto es parecido a presentar un número similar de patrones de diversas clases en el proceso de entrenamiento de un neuroclasificador [38], [40], [41].

De las 49 maniobras idénticas, se tomaron únicamente tres de estas para usarlas en el entrenamiento de la red neuronal e incrementar la diversidad de maniobras ante diversas situaciones. A continuación se presentan los histogramas de los valores de las maniobras usados en el entrenamiento. Como se observa, estos histogramas tienden a estar distribuidos en el universo de posibles valores.

Las Figuras 5.1, 5.2 y 5.3 muestran los histogramas de los valores de las distancias entre los obstáculos y el robot, los radios de los obstáculos y el ángulo entre el frente del robot y los obstáculos recolectados en la base de datos para el entrenamiento de la red neuronal.

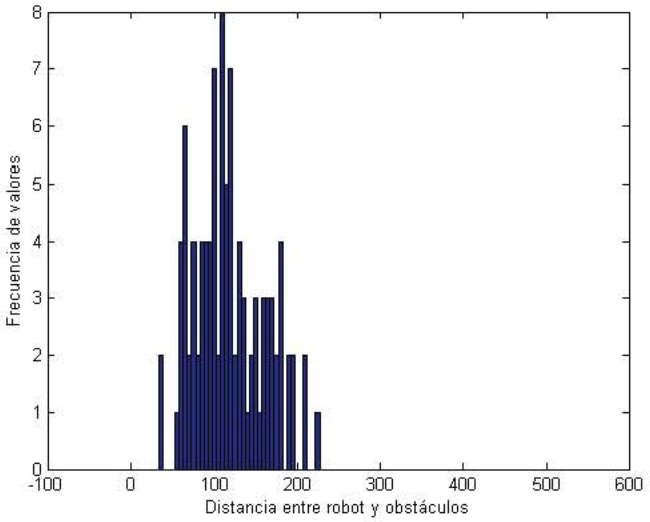


Figura 5.1: Histogramas de las distancias entre obstáculos y el robot

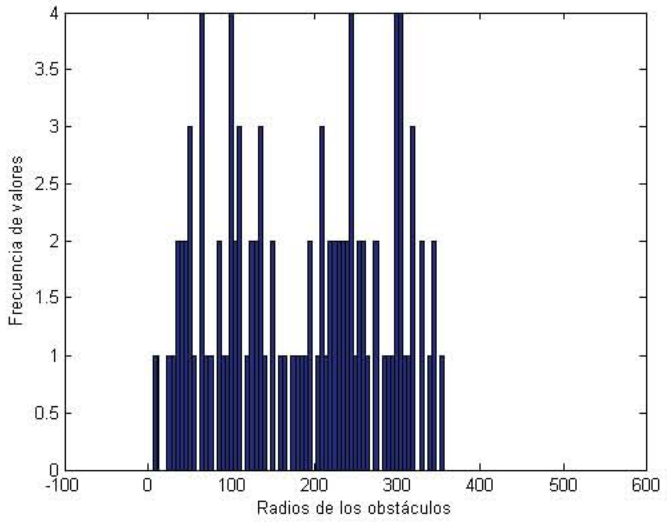


Figura 5.2: Histogramas de los radios de los obstáculos

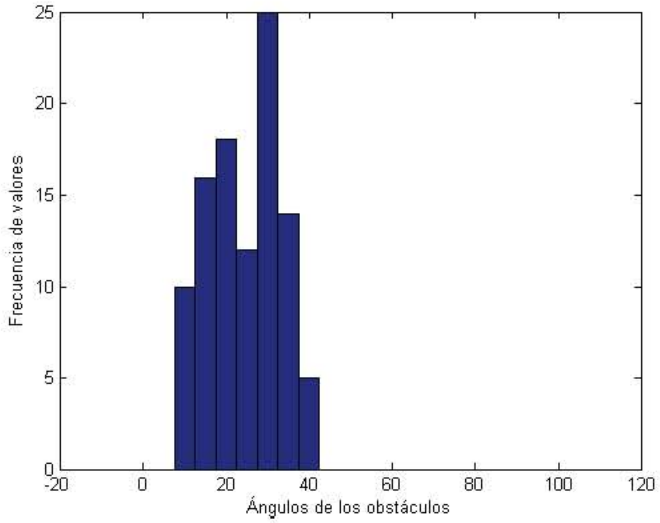


Figura 5.3: Histogramas de los ángulos entre el robot y los obstáculos

En cuanto a los valores que corresponden a la descripción de las maniobras (XR, YR y RR) se presentan los histogramas en las Figuras 5.4, 5.5 y 5.6 respectivamente. Estos histogramas se pueden apreciar nuevamente de forma distribuida sobre el rango posible de valores.

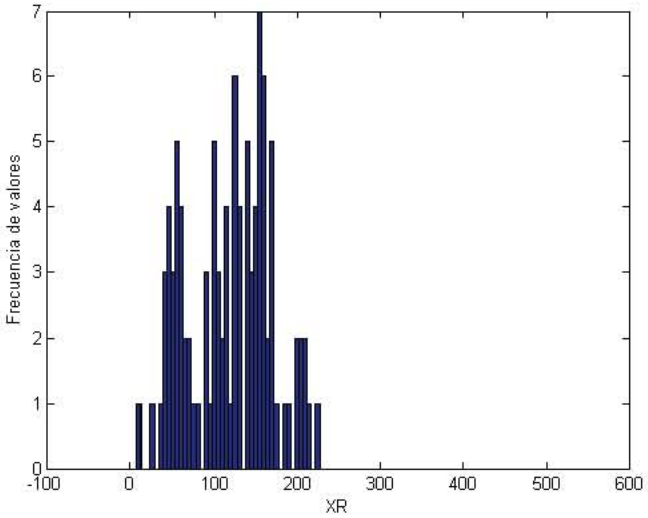


Figura 5.4: Histogramas de XR

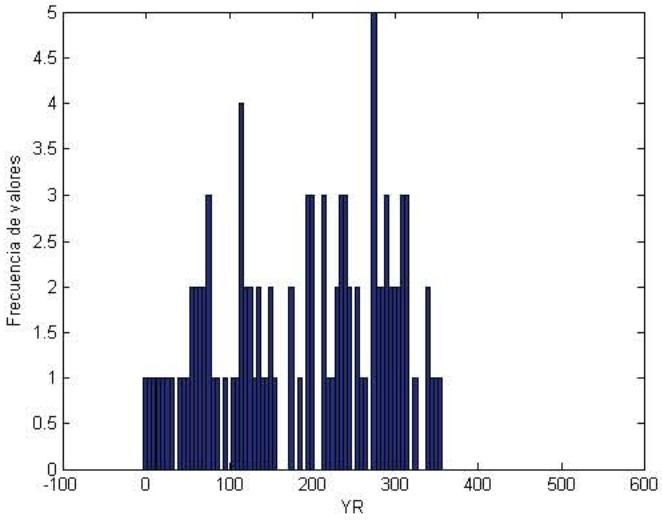


Figura 5.5: Histogramas de YR

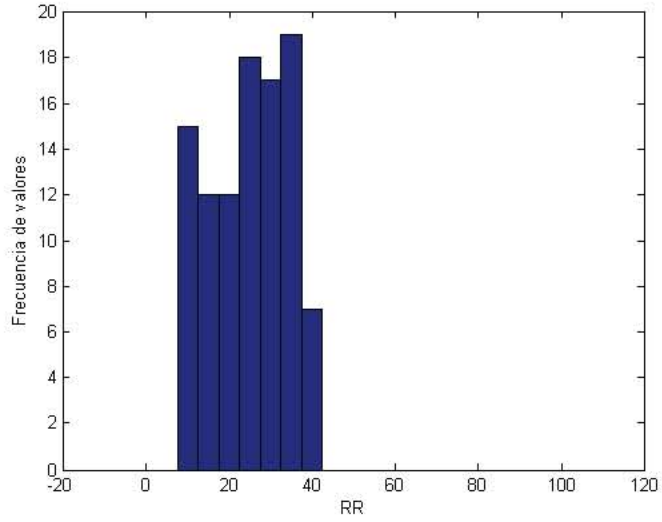


Figura 5.6: Histogramas de RR

5.2. Capacidad informativa

Se investigó la capacidad informativa de la red neuronal por medio de programas escritos en C++ (Anexos B y C) para ensambles aleatorios y ensambles correlacionados en la tarea descrita del robot móvil para la evasión de obstáculos. A continuación se presentan los resultados.

5.2.1. Ensambles aleatorios

Se investigó el mayor número de ensambles aleatorios K que puede almacenar la red neuronal con posibilidad de ser restaurados. Estos experimentos se hicieron para diversos valores del tamaño de la red neuronal N y del tamaño de los ensambles M con $R = 5$ ciclos de repetición. Se presentan los números de ensambles para los cuales la red reconoció al menos el 99 % de ellos y a su vez, cada uno fue restaurado en al menos 90 % del ensamble original ante 50 % de ruido (se modifican el 50 % de las neuronas de cada ensamble). Estos resultados se muestran en la Tabla 5.1.

Tabla 5.1: Capacidad informativa para ensambles aleatorios

N \ M	10	20	30	40	50	60	70	80	90	100	110	120
8000	200	3300	3700	8900	11700	10400	8900	7300	5700	4800	4200	3300
10000	1200	4100	4600	19100	19100	17200	14800	12500	9600	8600	8500	7400
12000	1300	4900	5500	27700	28400	26000	22900	19700	16400	13300	12200	10300
14000	1200	5600	6400	38300	39900	36900	32900	27500	23500	20600	17400	15200

Como se esperaba, existe variación de la capacidad informativa con relación al tamaño de la red neuronal N, entre mayor es el tamaño de la red N, se pueden almacenar y restaurar un número mayor de ensambles K. Esta relación se observa con mayor claridad en la Figura 5.7.

En cuanto a la influencia del tamaño de los ensambles M en el número de ensambles que se pueden almacenar K, se presentan valores óptimos para tamaños de ensambles en el rango de $M = [40, 60]$. En la Figura 5.8 se presenta la relación de M en K y se aprecia que existe un máximo en el rango especificado.

En este rango óptimo del tamaño de ensamble se tiene que el número de ensambles aleatorios memorizados K es mayor que el tamaño mismo de la red neuronal N, lo cual indica que la capacidad informativa es muy buena en comparación de otras redes neuronales [13], [34], [42].

Finalmente se presenta en la Figura 5.9 la relación del tamaño de la red neuronal N y el tamaño de los ensambles M en el número máximo de ensambles almacenados K bajo las condiciones especificadas.

5.2.2. Ensambles correlacionados

Para ensambles estadísticamente dependientes, es decir, para ensambles que corresponden a la codificación de la información anteriormente descrita para la base de datos. Se modificaron los

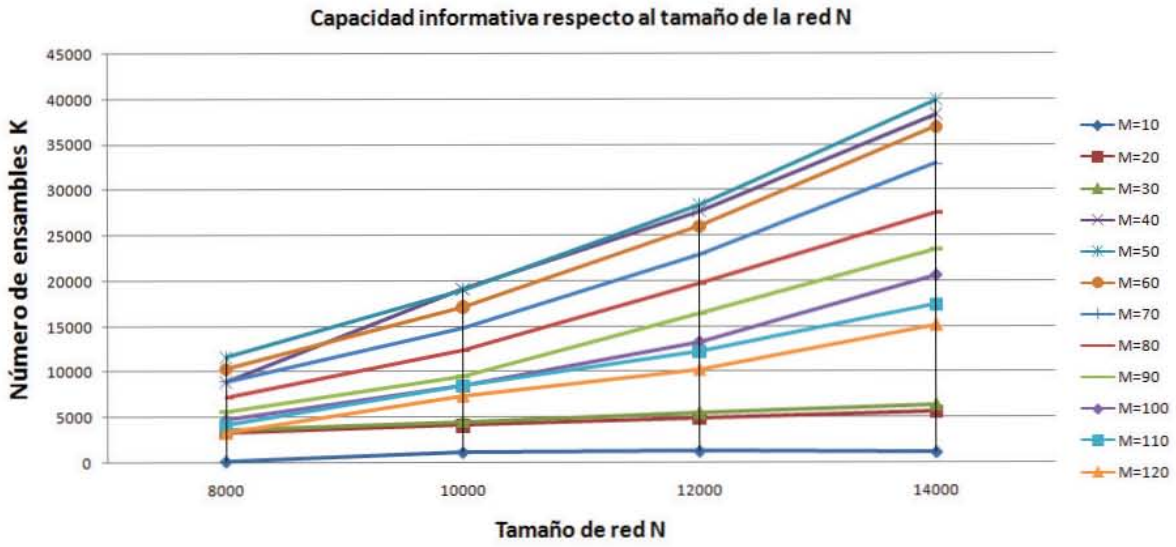


Figura 5.7: Capacidad informativa para ensambles aleatorios con respecto al tamaño de la red neuronal

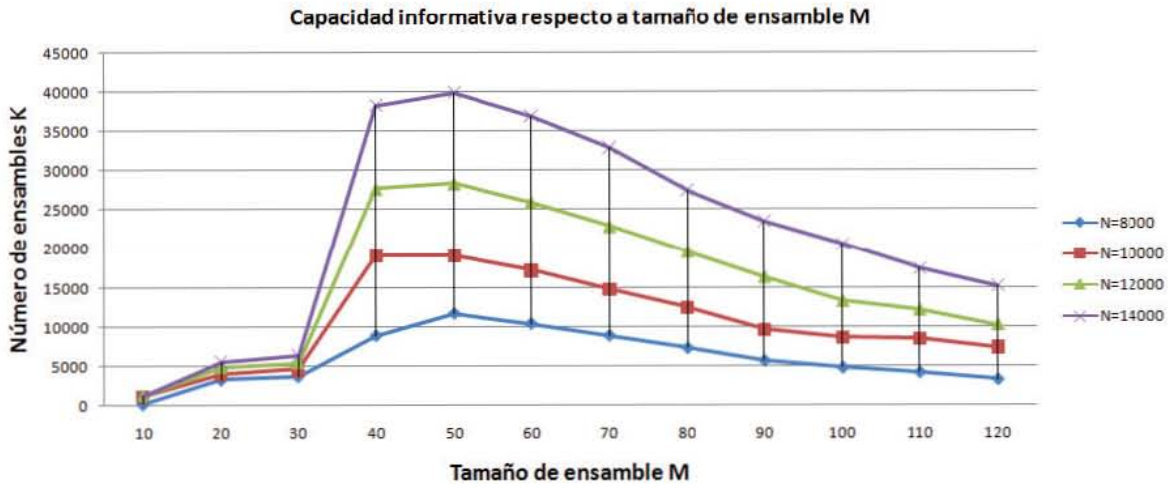


Figura 5.8: Capacidad informativa para ensambles aleatorios con respecto al tamaño de los ensambles

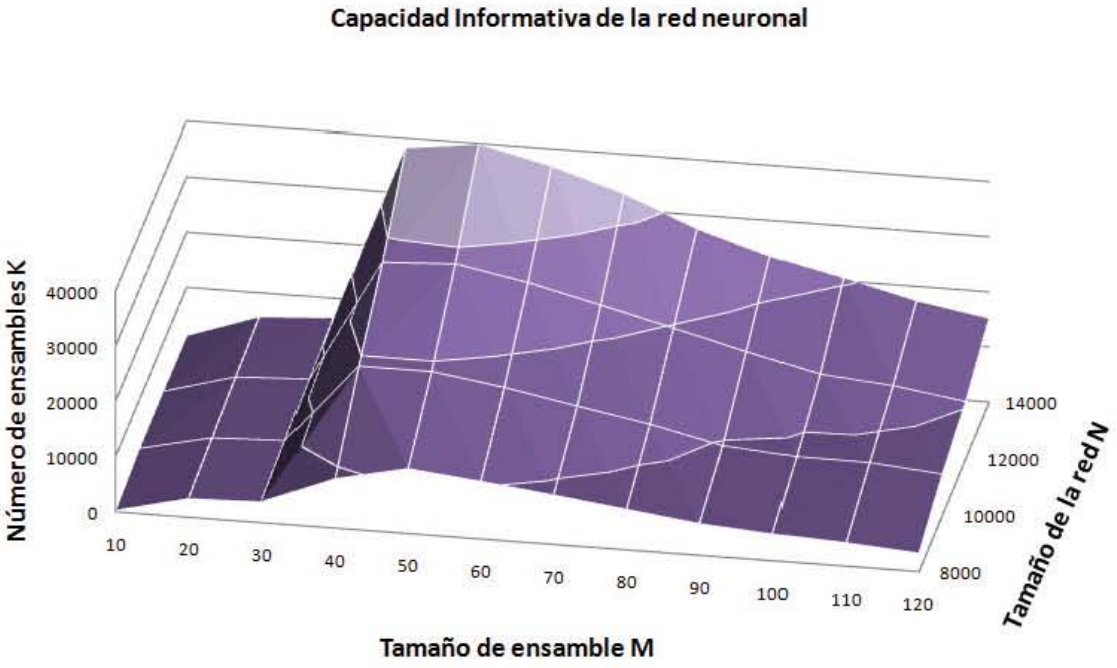


Figura 5.9: Capacidad informativa para ensambles aleatorios

parámetros del tamaño de la red neuronal N, el tamaño de los ensambles M y se buscó el mayor número de ensambles K a partir de los cuales se pueden reconocer y decodificar los parámetros de las maniobras y sus valores sin cometer errores. El valor de separación de correlación es $G = 10$. Estos resultados se muestran en la Tabla 5.2

Tabla 5.2: Capacidad informativa para ensambles estadísticamente dependientes

N \ M	10	20	30	40	50	60	70	80	90	100	110	120
8000	0	0	0	0	25	20	10	0	20	30	25	25
10000	0	0	15	0	10	20	20	25	25	30	25	25
12000	0	0	0	0	10	0	25	30	35	25	25	25
20000	0	0	0	0	30	30	30	30	35	30	25	30

Se aprecia que la capacidad informativa de la red neuronal disminuye en gran medida y el mayor número de ensambles que puede almacenar bajo estos parámetros de red neuronal es 35. Es decir puede reconocer y decodificar hasta 105 valores para los parámetros de las maniobras partiendo únicamente de la información de la situación.

Se observa nuevamente que existe relación entre la capacidad informativa para el tamaño de la red neuronal N. Al igual que para los ensambles aleatorios, para redes neuronales de mayor tamaño se pueden almacenar un mayor número de ensambles K, Figura 5.10.

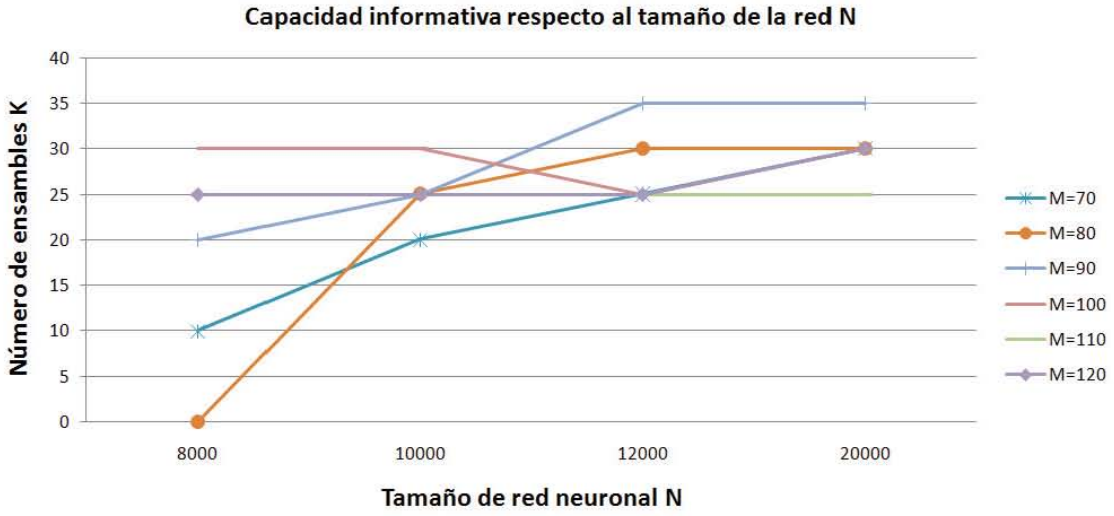


Figura 5.10: Capacidad informativa con respecto al tamaño de la red neuronal

En cuanto a la relación de la capacidad informativa con respecto al tamaño de los ensambles M, la Figura 5.11 muestra la relación existente. Se entiende que la relación es similar que para ensambles aleatorios, sin embargo, se observa que el valor óptimo del tamaño de los ensambles es diferente que para ensambles aleatorios.

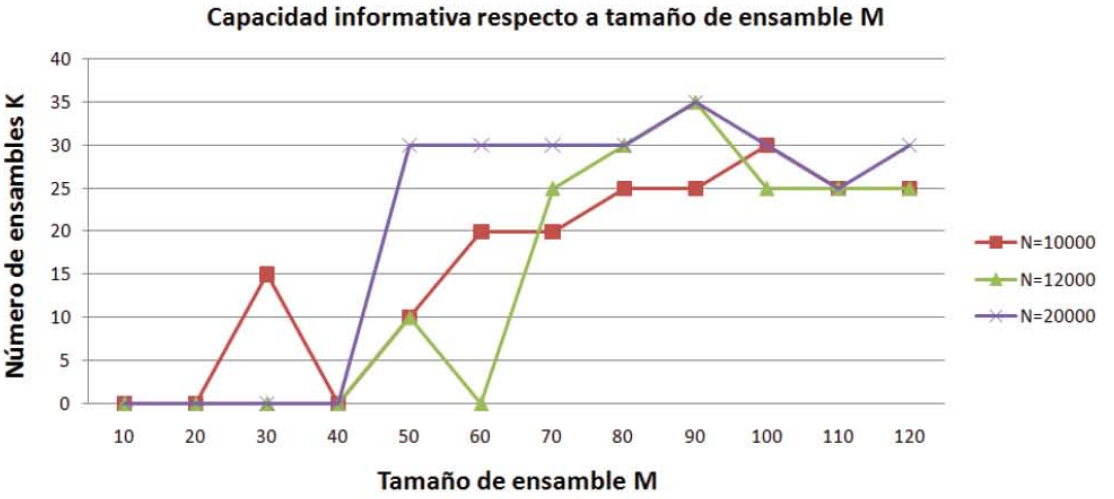


Figura 5.11: Capacidad informativa con respecto al tamaño de los ensambles

Realizar más pruebas requiere más tiempo de simulaciones, además, los códigos escritos en C++ presentan limitaciones con respecto al tamaño de los arreglos que debe manejar la memoria de la computadora, por lo que es necesario realizar modificaciones para poder realizar simulaciones tanto para tamaños de redes más grandes como para tamaños de ensambles.

La capacidad informativa se ve claramente disminuida con respecto a los ensambles no cor-

relacionados, sin embargo, esto no es del todo malo, ya que al definir el valor de separación de correlación, estamos permitiendo que los ensambles guarden aún más similitud para valores cercanos y eliminandola para valores lejanos. Este valor G es fundamental para enfrentar situaciones desconocidas.

Para recuperar la gran capacidad informativa podemos disminuir el valor, po ejemplo, para $G = 2$ con una red neuronal de tamaño $N = 20000$ y tamaño de ensamble $M = 200$ se logró memorizar y restaurar un total de $K = 55$ ensambles que comprenden 175 valores de parámetros, es decir, se puede memorizar la totalidad de la base de datos, aunque, con poca preservación de similitud en vectores que codifican situaciones similares.

5.3. Pruebas finales

Ahora que hemos estimado la capacidad informativa de la red neuronal para ensambles estadísticamente dependientes, debemos comprobar si es lo suficientemente grande como para poder resolver situaciones nuevas que se presenten al sistema. Por eso entrenamos la red neuronal de especificaciones: tamaño de red neuronal $N = 20000$, tamaño de ensambles $M = 90$ y número de ensambles $K = 35$, valor de separación de correlación $G = 10$.

Enseguida se presentan ejemplos de las simulaciones para situaciones nuevas generadas por el simulador gráfico y resueltas por la red neuronal para obtener la maniobra más adecuada para evadir los obstáculos.

Las Figuras 5.12, 5.14 y 5.15 muestran situaciones que no se presentaron en el entrenamiento, estas situaciones son codificadas y presentadas a la red neuronal para obtener las mejores maniobras.

Tras decodificar la respuesta de la red neuronal, el programa asigna los valores de los parámetros de las maniobras automáticamente y realiza el desplazamiento indicado, esto se muestra en la Figura 5.13.

Se observa que el sistema es capaz de resolver situaciones nuevas que se presenten, ejecutando diversas maniobras, por ejemplo la Figura 5.14 muestra una maniobra más ante otra situación desconocida.

Un ejemplo más de la solución de situaciones desconocidas se presenta en la Figura 5.15, en donde se aprecia que si bien la maniobra elegida es la correcta, aún es necesario realizar modificaciones en la distancia para obtener el resultado más óptimo.

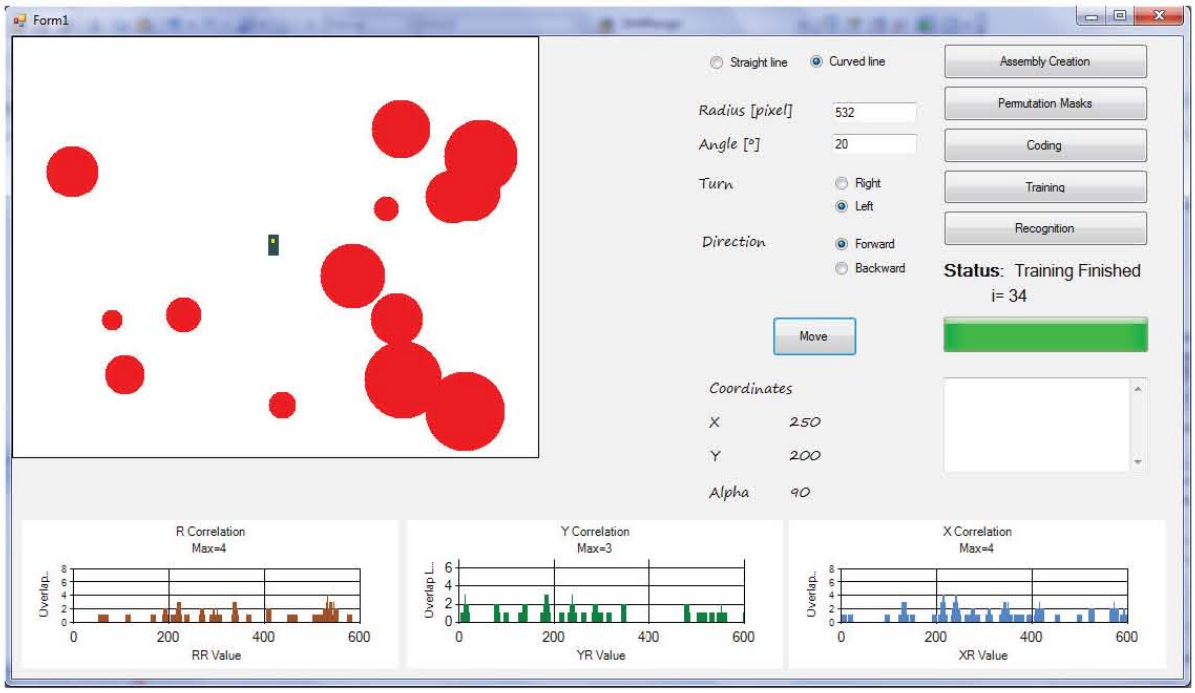


Figura 5.12: Situación nueva generada

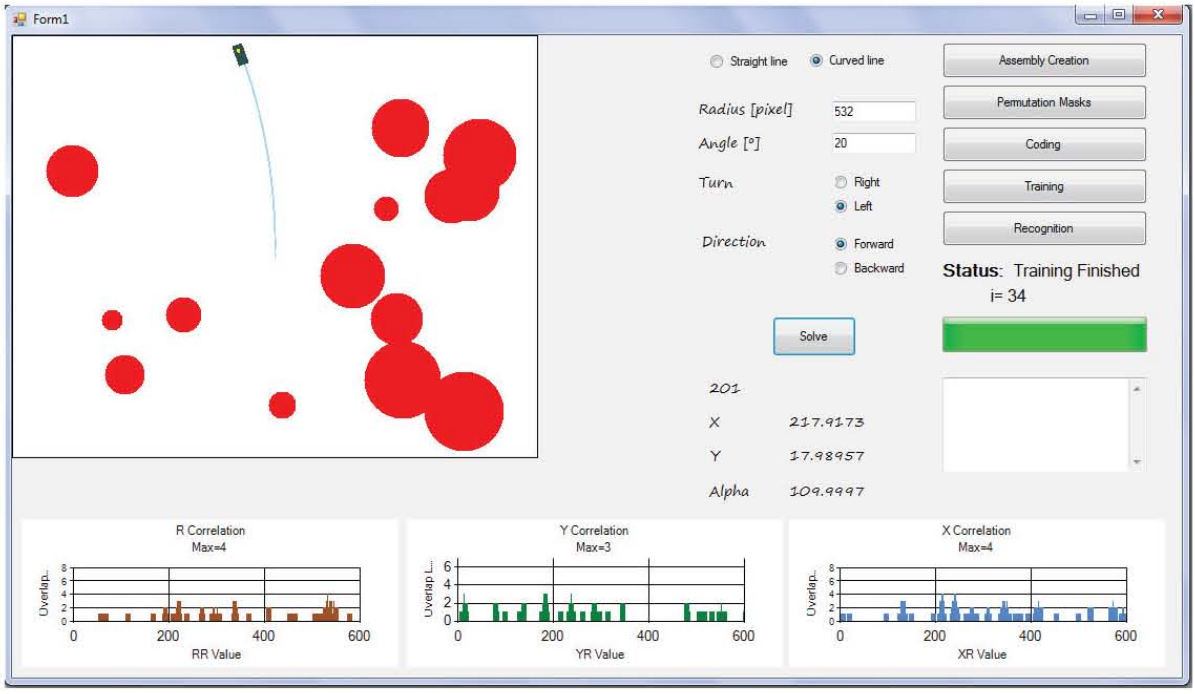


Figura 5.13: Maniobra ante situación desconocida

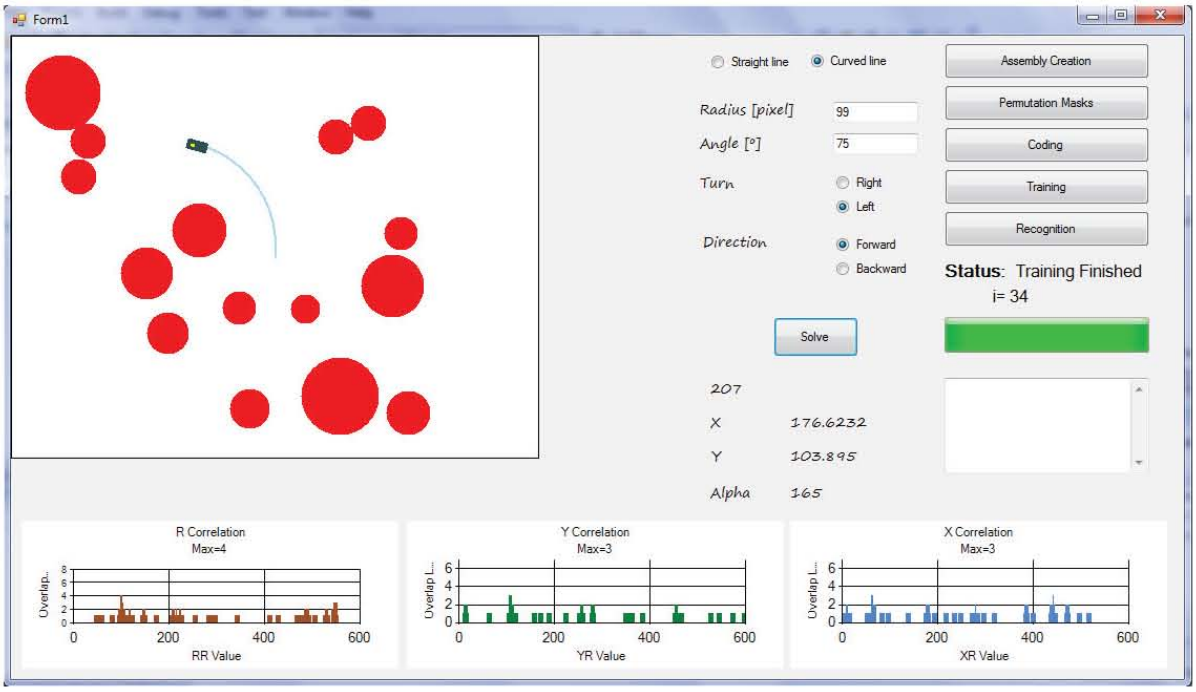


Figura 5.14: Maniobra ante situación desconocida

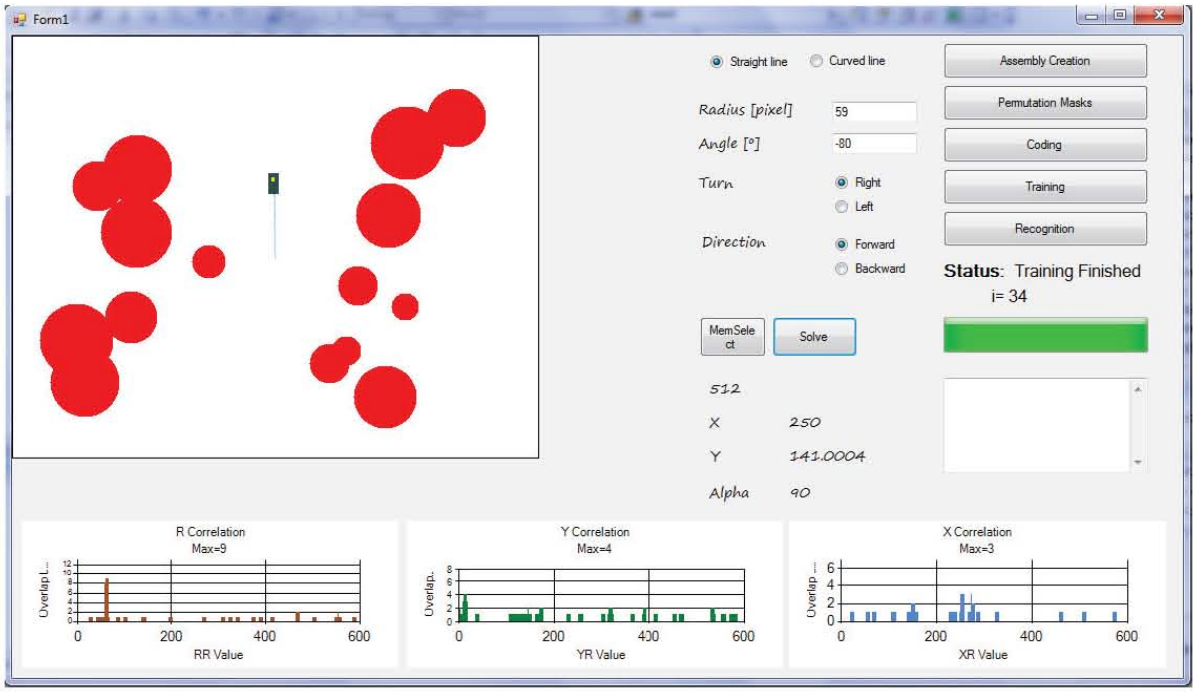


Figura 5.15: Maniobra ante situación desconocida

Este tipo de maniobras son el resultado de realizar combinaciones de las maniobras presentes en el entrenamiento para formular nuevas maniobras ante las situaciones desconocidas. Para corre-

girlas y mejorar el sistema, es necesario realizar secuencialmente el análisis de las situaciones para obtener maniobras consecutivas, sin embargo, este trabjo se deja para un futuro en el desarrollo de esta aplicación e investigación.

La reducción de la situación a tres obstáculos fue realizada bajo la división de una tarea mayor en pequeñas subtareas. Posteriormente se podrán resolver situaciones más complejas y hasta con obstáculos dinámicos.

Como primera aproximación consideramos nuestro sistema como una solución práctica adecuada, práctica y con futuro desarrollo para obtener un mejor desempeño.

Las gráficas de nivel de traslape para la decodificación de los valores de las maniobras ante las situaciones desconocidas nos muestran niveles aun débiles de traslape, si bien así se memorizaron más ensambles, es necesario incrementar estos niveles aumentando el tamaño de los ensambles y a su vez el tamaño de la red neuronal.

Conclusiones

El proyecto de investigación que se desarrollo como proyecto de tesis de la maestría tiene resultados muy concretos y satisfactorios. Se realizó una investigación acerca de las estructuras de ensambles en redes neuronales: se estudiaron sus fundamentos teóricos basados en la biología, se analizó la formación de los mismos dentro de las redes neuronales empleando una regla de aprendizaje Hebbiana y se describió el funcionamiento de la restauración de la actividad en los ensambles ante información incompleta y/o ruidosa.

Las estructuras de ensambles se emplearon de forma inicial para analizar la capacidad informativa de la red neuronal para ensambles estadísticamente independientes y debido a estos resultados prometedores se implementó esta metodología para ensambles estadísticamente dependientes en la tarea de evasión de obstáculos de un robot móvil.

Debido a la naturaleza de los datos del mundo real, ante estas situaciones se presenta correlación entre la información, tal es el caso de la base de datos recolectada.

Para implementar la aplicación de la red neuronal en el robot móvil se desarrolló un simulador gráfico del robot móvil y a partir de diversas simulaciones se generó una base de datos. Estos datos fueron empleados para la generación de ensambles estadísticamente dependientes y así entrenar a la red neuronal con información específica de la aplicación.

Se desarrolló un método de codificación/decodificación basado en estructuras de ensambles de neuronas, los cuales representan información real de la base de datos y permiten al sistema no sólo reconocer cualitativamente los datos, sino que también le provee la capacidad de obtener información cuantitativa. Además este método de codificación distribuye la información en la red neuronal permitiendo que exista similitud para valores cercanos y eliminándola para valores lejanos.

La metodología empleada presenta cualidades deseadas para manejar la red neuronal como una memoria asociativa útil en la aplicación mostrada en este proyecto.

Los resultados demuestran que la red neuronal puede restaurar ensambles a partir de información parcial y con ruido, para ensambles correlacionados. Además con la información almacenada es capaz de proveer al sistema la capacidad de solución de situaciones desconocidas, aprovechando la propiedad de la red como memoria asociativa.

Aún es necesario continuar en esta línea de investigación para generar un sistema con un mejor desempeño, esto se puede lograr básicamente con las siguientes tares: incrementar la capacidad de los programas desarrollados para poder manejar redes neuronales de mayor tamaño y ensambles neuronales de mayor tamaño. Es posible modificar la regla de entrenamiento por otras que han mostrado mejores resultados como son los métodos de covariancia [43] o bien la regla Bayesiana de máxima verosimilitud [44].

Si bien el trabajo a futuro es basto, los resultados obtenidos hasta el momento son satisfac-

torios e incluso prometedores para una gran variedad de sistemas más robustos. En esta tesis se desarrolló una aplicación muy específica, sin embargo en combinación con el método de codificación/decodificación, se puede pensar en una amplia gama de aplicaciones.

Este trabajo se limita a simulaciones por medio de programas de computadora, sin embargo, presenta una validez completa por la naturaleza de la información y los sistemas desarrollados, pero es deseable implementar en un futuro el sistema de forma física con sensores, actuadores y sistemas digitales como microcontroladores.

En resumen, la investigación integra varios conceptos de redes neuronales, de codificación, de programación entre otros, para analizar, describir e implementar un sistema que cuenta con una aplicación particular aquí presentada y con posibilidad de mejora para lograr un mejor desempeño.

Bibliografía

[1] P. D. Wassermann, *Neural Computing*, Van Nostrand Reinhold, Nueva York, 1989.

[2] K. S. Lasheley, *In Search of the engram*, Society of experimental Biology Symposium, N. 4, Physiological Mechanisms in Animal Behavior, Ambridge University Press, recopilado en Neurocomputing: Foundations of Research, pp. 59-63, MIT Press, 1989.

[3] D. O. Hebb, *The Organization of Behavior.* Nueva York: Wiley, 1949.

[4] J. Von-Neumann, *The Computer and the Brain*, Yale University Press, recopilado en Neurocomputing: Foundations of Research, pp. 83-87, MIT Press, 1989.

[5] O. G. Selfridge "Pandemonium: a paradigm for learning", Mechanisation of Thought processes: Proceedings of a Symposium Held at th National Physical laboratory, Noviembre 1958, recapitulado en Neurocomputing, pp. 117-120, MIT Press, 1989.

[6] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain", *Psychological Review*, Vol. 65, No. 6, 1958.

[7] B. Widrow, M. E. Hoff "Adaptive Switching Circuits", *Neurocomputing: Foundations of Research*, pp. 126-134, MIT Press, 1989.

[8] S. Haykin, *Adaptive Filter Theory*, Prentice Hall, 1996.

[9] M. Minsky, S. Papert, "Perceptrons", recopilado en Neurocomputing: Foundations of Research, pp. 161-173, MIT Press, 1989.

[10] J. A. Anderson, "A simple neural network generating an interactive memory", recopilado en Neurocomputing: Foundations of Research, pp. 181-192, MIT Press, 1989.

[11] T. Kohonen, "Correlation matrix memories", recopilado en Neurocomputing: Foundations of Research, pp. 174-180. MIT Press, 1989.

[12] S. Grossberg, "How does a brain build a cognitive code", recopilado en Neurocomputing: Foundations of Research, pp. 349-399. MIT Press, 1989.

[13] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities", *Proc. Nat. Acad. Sci. USA*, vol 79, 1982, pp. 2554-2558..

[14] T. Kohonen, “Self-organized formation of topologically correct feature maps”, recopilado en Neurocomputing: Foundations of Research, pp. 349-399. MIT Press, 1989.

[15] K. Fukushima, “Neocognitron: a self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position,” *Biological Cybernetics*, Springer Verlag, 36, 1980, pp. 193-202.

[16] D. E. Rumelhart, G. E. Hinton, R. J. Williams, “Learning internal representations by error propagation”, recopilado en neurocomputing: Foundations of Research, pp. 349-399. MIT Press, 1989.

[17] D. S. Brommhead, D. lowe, “Multivariable functional interpolation and adaptive networks”, *Complex Systems*, vol 2, 321-355.

[18] T. Poggio, F. Girosi, “Networks for Approximation and Learning“, *Proceedings IEEE*, vol 78, N 9, pp. 1481-1497, Spetiembre 1990.

[19] V. M. Vapnik, *“The Nature of Statistical Learning Theory”*, Springer Verlag, 1996.

[20] E. Wan, “Finite impulse response neural networks with applications in time series prediction”, PhD Disertation, Universidad de Stanford, 1993.

[21] J. A. Anderson, *An Introduction to Neural Networks,* Massachusetts Institute of Technology Press 1995.

[22] W. S. McCulloch, W. H. Pitts, “A logical calculus of the ideas immanent in nervous activity”, *Bulletin of Mathematical Biophysics*, pp. 115-133, 1943.

[23] R. Rojas, *Neural Networks a Systematic Introduction*, Springer Verlag, Berlín, 1996.

[24] C. E. Shannon, “A mathematical theory of communication”, *Bell System Technical Journal*, Vol. 27, pp. 379–423, 623–656, 1948.

[25] B. Knight, “Dynamic of encoding in a population of neurons”, *Journal of general Physiology*, pp. 734-766, 1972.

[26] J. Proakis, D. Manolakis, *Tratamiento Digital de Señales*, Prentice Hall, 1997.

[27] H. K. Hartline, F. Ratliff, “Spatial summation of inhibitory influences in the eye of limulus and the mutual inreaction of receptor units”, *Journal of General Physiology*, pp. 1049-1066, 1958.

[28] H. B. Barlow, *Single units and sensations: A neuron based doctrine for perceptual psychology?* Perception, 1972.

[29] D. Calderon, T. Baidyk, E. Kussul, “Information coding with neural networks for a mobile robot”, IEEE, *International Joint Conference on Neural Networks*, San Jose, California, USA, Agosto 2011.

[30] R. J. McEliece, E. C. Posner, E. R. Rodemich and Santosh Venkatesh, "The capacity of the Hopfield associative memory," *IEEE Transactions on Information Theory* Vol. IT-33, No. 4, July 1987.

[31] D. Calderon, T. Baydyk, E. Kussul, "Ensembles' Structures in Neural Networks", *1st International Congress on Instrumentation and Applied Sciences*, Cancún Quintana Roo, México, Octubre 26 - 29, 2010.

[32] D. Golomb, N. Rubin and H. Sompolinsky, "Willshaw model: associative memory with sparse coding and low firing rates," *Physical review A.*, February 1990, Vol. 4 No. 41 pp. 1843-1854.

[33] E. Kussul, O. Makeyev, T. Baidyk, D. Calderon, "Neural Network with Ensembles", *IEEE World Congress on Computational Intelligence 2010*, Barcelona España, Julio 18-23 2010, pp. 2955-2961.

[34] G. Palm and F.T. Sommer, "Information capacity in recurrent McCulloch-Pitts networks with sparsely coded memory states," *Network 3*, 1992, pp.177-186.

[35] M. Oubbati, W. Holoch and G. Palm, "Neural Fields for Complex Behavior Generation on Autonomous Robots," *International Joint Conference on Neural Networks*, Atlanta, Georgia, USA, June 2009, pp. 2388-2393.

[36] C. G. Atkeson and D. J. Reinkensmeyer, "Using associative content-addressable memories to control robots," *Proceedings of the 27th Conference on Decision and Control*, 1988, pp. 729-797.

[37] S. Jockel, M. Mendes, J. Zhang, A. P. Coimbra and M. Crisóstomo, "Robot navigation and manipulation based on a predictive associative memory," *IEEE 8th Conference on Development and Learning*, 2009, pp. 1-7.

[38] E. Kussul, T. Baydyk, D. Wunsch, "Image recognition systems with permutative coding," *International Joint Conference on Neural Networks*, Montreal, July 2005, pp. 1788-1793.

[39] E. Kussul, T. Baidyk, D. Wunsch, *Neural Networks and Micro Mechanics,* Springer Verlag, 2010, ISBN 978-3-642-02534-1.

[40] Kussul E., Baidyk T., "LIRA Neural classifier for handwritten digit recognition and visual controlled microassembly", *Neurocomputing*, Vol. 69, Issue 16-18, ISSN: 0925-2312, 2006, pp.2227-2235

[41] E. Kussul, T. Baidyk and O. Makeyev, "Pairwise permutation coding neural classifier," *International Joint Conference on Neural Networks*, Orlando, Florida, USA, August 12-17, 2007, pp. 1847-1852.

[42] A. Knoblauch, "Neural associative memory for brain modeling and information retrieval", *Information Processing Letters* 95, pp 537-544, 2005.

[43] P. Dayan and D. Willshaw, "Optimising synaptic learning rules in linear associative memory," *Biological Cybernetics* vol. 65, 1991, pp. 253-265.

[44] A. Knoblauch, "Optimal synaptic learning in non-linear associative memory," *International Joint Conference on Neural Networks (IJCNN)*, IEEE World Congress on Computational Intelligence (WCCI), Barcelona, Spain 2010, pp. 3205-3211.

Anexos

A. Código de programa para realizar red neuronal con ensambles MatLab

```
clear
NSize=400; %Neural Network Size
N=NSize;
ASize=30; %Assembly Size
MM=ASize;
ANum=50;  %Assembly Number
K=ANum;
Rec=1;     %Recalculation Cycles
NL=0.5;    %Noise Level
SUT=0.9*ASize;   %Control for neural
Network
mi=1; %display vector
As=zeros(ANum,NSize);
Asn=zeros(ANum,NSize);
%Creation of the assemblies
for i=1:ANum
    k=0;
    while(k<ASize)
        Pos=randi(NSize,1,1);
        if (As(i,Pos)~=1)
            As(i,Pos)=1;
            k=k+1;
        end
    end
end
%Training of network by extern product
M=zeros(NSize,NSize);
for i=1:ANum
    M=M+As(i,:)'*As(i,:);
end
for i=1:NSize
    for j=1:NSize
        if(M(i,j)>16)
            M(i,j)=16;
        end
    end
end
%Values in diagonal
crash=zeros(1,NSize);
for i=1:NSize
    crash(1,i)=M(i,i);
end
diagonal=zeros(NSize,1);
for i=1:NSize
    diagonal(i)=M(i,i);
end
%Adding Noise to Assemblies
for i=1:ANum
    k=1;
    mPos=zeros(ASize,1);
    %Extract positions with 1's
    for j=1:NSize
        if(As(i,j)==1)
            mPos(k)=j;
            Asn(i,j)=1;
            k=k+1;
        end
    end
    M2=eye(NSize);
    M2=1-M2;
    M2=M2.*M;
    %Set 1's with half of ASize (noise)
    k=0;
    while(k<(ASize*NL))
        Pos=randi(NSize,1,1);
        if (Asn(i,Pos)~=1)
            Asn(i,Pos)=1;
            k=k+1;
        end
    end
    %Change the half of 1's
    k=0;
    zPos=zeros(ASize*NL,1);
    while(k<(ASize*NL))
        Pos=randi(ASize,1,1);
        if(mPos(Pos)~=0)
            zPos(k+1)=mPos(Pos);
            mPos(Pos)=0;
            k=k+1;
        end
    end
    %Set 1's with half of ASize
    for j=1:ASize
        if(mPos(j)~=0)
            Asn(i,mPos(j))=1;
        end
    end
    %Set 0's to the half ASize
    for j=1:(ASize*NL)
        Asn(i,zPos(j))=0;
    end

end
overl=zeros(ANum,1);
for i=1:ANum
    overl(i)=As(i,:)*Asn(i,:)';
end
trasl=zeros(ANum,1);
%Recognition of Assemblies
for i=1:ANum
    E=Asn(i,:);
    for r=1:Rec
        E=E*M;
        hist=zeros(10000*NSize,1);
        for j=1:NSize
            k=E(j);
            hist(k+1)=hist(k+1)+1;
        end
        summ=0;
        j=length(hist);
        while((summ<SUT)&&(j>0))
            summ=summ+hist(j);
            j=j-1;
        end
        thres=j;
        for j=1:NSize
            if(E(j)>thres)
                E(j)=1;
            else
                E(j)=0;
            end
        end
        if(i==mi)
            figure(4)
            stem(E)
        end
```

```
    end
    tras1(i)=E*As(i,:)';
end
figure(1)
stem(tras1/ASize);
recog=0;
for i=1:ANum
    if(tras1(i)>=(0.9*ASize))
        recog=recog+1;
    end
end
recog
figure(3)

 salida=Asn(1,:)*M;
 stem(salida)
 title('Salida');
original=As(mi,:);

ruido=Asn(mi,:);

 comun=original.*ruido;
 restaura=original-comun;
 atenua=ruido-comun;


%Estadistica de los pesos sinápticos
M1=eye(NSize,NSize);
M1=1-M1;
M3=M.*M1;
k=1;
j=1;
for i=1:(NSize*NSize)
    triangulos(i)=M3(k,j);
    k=k+1;
    if(k>=NSize)
        k=1;
        j=j+1;
        if(j>=NSize)
            j=1;
        end
    end
end


trig1=zeros(max(crash)+1,1);
trig2=zeros(max(triangulos)+1,1);
for i=1:NSize

trig1(crash(i)+1)=trig1(crash(i)+1)+1;
end
for i=1:(NSize*NSize)

trig2(triangulos(i)+1)=trig2(triangulos
(i)+1)+1;
end
trig2(1)=trig2(1)-N;
trig1=trig1/NSize;
trig2=trig2/(NSize*NSize-N);

aqui=17-size(trig1);
trig1=[trig1' zeros(1,aqui)];

figure(1)
ni=0:16;
stem(ni,trig1)
figure(2)
aqui=17-size(trig2);
trig2=[trig2' zeros(1,aqui)];
stem(ni,trig2)
vardiag=var(crash)
vartriag=var(triangulos)
meandiag=mean(crash)
meantriag=sum(triangulos)/(NSize*NSize-
NSize)


for ni=0:15
mio1(ni+1)=(((N-MM)/N)^(K-
ni))*((MM/N)^ni)*(factorial(K)/(factori
al(ni)*factorial(K-ni)));
end
mio1(17)=1-sum(mio1(1:16));

A=(MM*MM-MM)/(N*N-N);
B=(N*N-N-MM*MM+MM)/(N*N-N);
for ni=0:15
mio2(ni+1)=(A^ni)*(B^(K-
ni))*(factorial(K)/(factorial(ni)*facto
rial(K-ni)));
end
mio2(17)=1-sum(mio2(1:16));

figure(1)
hold
ni=0:16;
stem(ni,mio1,'*','black')
xlabel('i')
ylabel('f Vd')
title('Pesos de la diagonal')

mmed1=sum(ni.*mio1)
mvar1=sum(mio1.*(ni-mmed1).^2)



figure(2)
hold
ni=0:16;
stem(ni,mio2,'*','black')
xlabel('i')
ylabel('f Vnd')
title('Pesos fuera de la diagonal')

mmed2=sum(ni.*mio2)
mvar2=sum(mio2.*(ni-mmed2).^2)



figure(7)
 stem(comun.*salida)
 title('comun');
 figure(8)
 stem(restaura.*salida);
```

```
 title('restaura');
 figure(9)
 stem(atenua.*salida);
 title('atenua');
 figure(10)
 stem((1-comun-atenua-
restaura).*salida)
 title('ruido')


 Ediag=mmed1; Etriag=mmed2;
 Eruido=MM*Etriag
 elruido=(1-comun-atenua-
restaura).*salida;
 Er=sum(elruido);
 torpe=0;
 for mudo=1:length(elruido)
     if(elruido(mudo)>0)
         torpe=torpe+1;
     end
 end
 Er=Er/torpe

 Eicorrecta=Ediag+(1+Etriag)*(1-
NL)*(MM-1)+NL*(Etriag)*(MM-1)
  elcorr=comun.*salida;
 Eic=sum(elcorr);
 torpe=0;
 for mudo=1:length(elcorr)
     if(elcorr(mudo)>0)
         torpe=torpe+1;
     end
 end
 Eic=Eic/torpe

Eirestau=(1+Etriag)*(MM-1)*(1-
NL)+(Etriag)*(MM-1)*NL
  elresta=restaura.*salida;
 Eir=sum(elresta);
 torpe=0;
 for mudo=1:length(elresta)
     if(elresta(mudo)>0)
         torpe=torpe+1;
     end
 end
 Eir=Eir/torpe

Eiatenua=Etriag*(MM-1)+Ediag
 elate=atenua.*salida;
 Eia=sum(elate);
 torpe=0;
 for mudo=1:length(elate)
     if(elate(mudo)>0)
         torpe=torpe+1;
     end
 end
 Eia=Eia/torpe
```

B. Código de programa para realizar red neuronal con ensambles C++

Variables Globales

```
const unsigned long NetSize=18000;                  //size of Neural Network
const unsigned long AssemblySize=50;               //size of Neuron Assembly
const unsigned long AssemblyNumber=67900;          //number of Neuron Assemblies
const unsigned long MatSize1=NetSize/8;
int SUT=(int)AssemblySize*0.9;          //SUT level
unsigned char Wmax=16;                  //synaptic weight maximum
unsigned char buff[NetSize];                       //working array
unsigned char buff1[NetSize];                      //working array
unsigned char buff2[NetSize];                      //working array
int buff3[NetSize];                       //working array
float Noise=0.5;                          //probability of bit value changing in the
binary vector of activity during recognition process
const unsigned long MatSize=(NetSize*NetSize)/8;      //size of the matrix
unsigned char *matr0 = new unsigned char [MatSize];   //synaptic matrix 0
unsigned char *matr1 = new unsigned char [MatSize];   //synaptic matrix 1
unsigned char *matr2 = new unsigned char [MatSize];   //synaptic matrix 2
unsigned char *matr3 = new unsigned char [MatSize];   //synaptic matrix 3
unsigned char *matr4 = new unsigned char [MatSize];   //synaptic matrix 4
unsigned char *matr5 = new unsigned char [MatSize];   //synaptic matrix 5
unsigned char *matr6 = new unsigned char [MatSize];   //synaptic matrix 6
unsigned char *matr7 = new unsigned char [MatSize];    //synaptic matrix 7
unsigned long ActiveNumber[AssemblySize];  //numbers of active neurons
int error=0;                               //number of recall errors
int OverlapLevel=(int)AssemblySize*0.9;          //acceptable level of overlap of
restored assembly with stored assembly
unsigned long Excitation[NetSize];              //array of neuron excitations
const unsigned Xscale=5;                        //histogram scale in X-direction
const unsigned Yscale=10;                        //histogram scale in Y-direction
const int RecCycle=5;                          //Number of recalculation cycles
int randi=0;
```

Assembly Creation

```
register int i,j,k,p,q;
FILE *fp ;
fp=fopen("assemblies.dat","wb");
this->label2->Text = L"Creating Assembly...";
this->label2->Refresh();
this->progressBar1->Maximum=AssemblyNumber-1;
for(i=0;i<AssemblyNumber;i++)
{
      this->label3->Text = System::String::Concat(L"i= ",i.ToString()) ;
      this->label3->Refresh();
      this->Show();
      this->progressBar1->Value=i;
      for(j=0;j<NetSize;j++)
      {
            buff[j]=0;
      }
      for(k=0;k<AssemblySize;k++)
      {
```

```
            q=1;
            while(q!=0)
            {
                  p=Random(NetSize);
                  q=buff[p];
            }
            buff[p]=1;
      }
      k=0;
      for(j=0;j<NetSize;j++)
      {
            if(buff[j]!=0)
            {
                  ActiveNumber[k++]=j;
            }
      }
      if (k!=AssemblySize)
            {
                  this->label3->Text= L"Error1";
            }

      if(fwrite(ActiveNumber,4,AssemblySize,fp)!=AssemblySize)
      {
            this->label3->Text= L"Cannot wrote Assembly";
      }
}
fclose(fp);
this->label2->Text = L" Creation Finished";
```

Training

```
int i,j,m,n,p,q,excess,E,B;
unsigned v,k;
unsigned char ch1;
double MatrSum;
float Wmean;
FILE *fp;
FILE *fi;
fi=fopen("assemblies.dat","rb");
if(fi==NULL)
{
      this->label2->Text= L"Cannot open assemblies.dat";
}
this->label2->Text=L"Training...";
this->label2->Refresh();
memset(matr0,0,MatSize1*NetSize);
memset(matr1,0,MatSize1*NetSize);
memset(matr2,0,MatSize1*NetSize);
memset(matr3,0,MatSize1*NetSize);
memset(matr4,0,MatSize1*NetSize);
memset(matr5,0,MatSize1*NetSize);
memset(matr6,0,MatSize1*NetSize);
memset(matr7,0,MatSize1*NetSize);
excess=0;
this->progressBar1->Maximum=AssemblyNumber-1;
for(i=0;i<AssemblyNumber;i++)
```

```
{
	this->label3->Text = System::String::Concat(L"i= ",i.ToString()) ;
	this->label3->Refresh();
	this->Show();
	this->progressBar1->Value=i;
	if(fread(ActiveNumber,4,AssemblySize,fi)!=AssemblySize)
	{
		this->label2->Text= L"Cannot read assemblies.dat";
		goto end1;
	}
	for(m=0;m<AssemblySize;m++)
	{
		p=ActiveNumber[m];
		E=p/MatSize1;
		B=p-E*MatSize1;
		for(n=0;n<AssemblySize;n++)
		{
			q=ActiveNumber[n];
			if(E==0)
				ch1=matr0[B*NetSize+q];
			if(E==1)
				ch1=matr1[B*NetSize+q];
			if(E==2)
				ch1=matr2[B*NetSize+q];
			if(E==3)
				ch1=matr3[B*NetSize+q];
			if(E==4)
				ch1=matr4[B*NetSize+q];
			if(E==5)
				ch1=matr5[B*NetSize+q];
			if(E==6)
				ch1=matr6[B*NetSize+q];
			if(E==7)
				ch1=matr7[B*NetSize+q];
			if(ch1<Wmax)
				ch1++;
		else
			excess++;
			if(E==0)
				matr0[B*NetSize+q]=ch1;
			if(E==1)
				matr1[B*NetSize+q]=ch1;
			if(E==2)
				matr2[B*NetSize+q]=ch1;
			if(E==3)
				matr3[B*NetSize+q]=ch1;
			if(E==4)
				matr4[B*NetSize+q]=ch1;
			if(E==5)
				matr5[B*NetSize+q]=ch1;
			if(E==6)
				matr6[B*NetSize+q]=ch1;
			if(E==7)
				matr7[B*NetSize+q]=ch1;
		}
	}
}
```

```
end1:
fclose(fi);
this->label2->Text= L"Training Finished";
```

Recognition

```
int i,j,k,m,n,p,q;
FILE *fp;
FILE *fi;
fi=fopen("assemblies.dat","rb");
if(fi==NULL)
{
      this->label2->Text= L"Cannot open assemblies.dat";
      goto end1;
}
this->label2->Text=L"Recognition...";
this->label2->Refresh();
error=0;
this->progressBar1->Maximum=AssemblyNumber-1;
for(i=0;i<AssemblyNumber;i++)
{
      this->progressBar1->Value=i;
      this->label3->Text = System::String::Concat(L"i= ",i.ToString()) ;
      this->label3->Refresh();
      this->Show();
      if(fread(ActiveNumber,4,AssemblySize,fi)!=AssemblySize)
      {
            this->label2->Text= L"Cannot read assemblies.dat";
            goto end1;
      }
      for(j=0;j<NetSize;j++)
            buff[j]=0;
      for(j=0;j<AssemblySize;j++)
      {
            p=ActiveNumber[j];
            buff[p]=1;
      }
      for(j=0;j<NetSize;j++)
             [j]=buff[j];
      Mutation();
      for(k=0;k<RecCycle;k++)
      {
            Recalculation();
      }
      if(Overlap()<OverlapLevel)
            error++;
      this->label4->Text = System::String::Concat(L"error= ",error.ToString()) ;
      this->label4->Refresh();
}
this->label2->Text= L"Recognition Finished";
end1:
fclose(fi);
```

Mutation Function

```
void Mutation (void)
      {
            int i,j,k,m,n,p,q;
            int ChangeLevel;

            ChangeLevel=1000*Noise;
            k=0;
            for(i=0;i<NetSize;i++)
            {
                  if(buff1[i]!=0)
                  {
                        if ((Random(1000))<ChangeLevel)
                        {
                              buff1[i]=0;
                              k++;
                        }
                  }
            }
            for(j=0;j<k;j++)
            {
                  q=1;
                  while(q!=0)
                  {
                        p=Random(NetSize);
                        q=buff1[p];
                  }
                  buff1[p]=1;
            }

}
```

Recalculation Function

```
void Recalculation (void)
      {
            int i,j,k,m,n,p,q,hs,hs1,thresh,sum,E,B;
            long unsigned na=0,nn=0;
            unsigned hist[AssemblySize*12];

            hs=AssemblySize*12;

            for(i=0;i<NetSize;i++)
                  Excitation[i]=0;

            for(j=0;j<NetSize;j++)
                  if(buff1[j]!=0)
                  {
                        E=j/MatSize1;
                        B=j-E*MatSize1;
                        for(k=0;k<NetSize;k++)
                        {
```

```
                        if(E==0)
                              Excitation[k]+=matr0[B*NetSize+k];
                        if(E==1)
                              Excitation[k]+=matr1[B*NetSize+k];
                        if(E==2)
                              Excitation[k]+=matr2[B*NetSize+k];
                        if(E==3)
                              Excitation[k]+=matr3[B*NetSize+k];
                        if(E==4)
                              Excitation[k]+=matr4[B*NetSize+k];
                        if(E==5)
                              Excitation[k]+=matr5[B*NetSize+k];
                        if(E==6)
                              Excitation[k]+=matr6[B*NetSize+k];
                        if(E==7)
                              Excitation[k]+=matr7[B*NetSize+k];

                  }
            }

      for(i=0;i<AssemblySize*12;i++)
            hist[i]=0;

      for(i=0;i<NetSize;i++)
      {
            p=Excitation[i];
            hist[p]++;
      }
      sum=0;
      i=hs-1;
      while(sum<SUT)
      {
            sum+=hist[i--];
            if(i<0)
                  break;
      }
      thresh=i;
      for(i=0;i<NetSize;i++)
      {
            if(Excitation[i]>thresh)
            {
                  buff1[i]=1;
                  na++;
            }
            else
                  buff1[i]=0;
      }
      na;
      k=0;
}
```

Overlap function

```
int Overlap (void)
	{
		int i,j,k,sum;

		sum=0;
		for(i=0;i<NetSize;i++)
			if((buff[i]!=0)&&(buff1[i]!=0))
				sum++;

		return sum;
}
```

Random Function

```
long unsigned Random(long unsigned xmax)
	{
		static bool First = true;
		unsigned int mynum;
		long double n;
		// Initialises random generator for first call
		if (First)
		{
			First = false;
			srand(GetTickCount());
		}
		rand_s(&mynum);
		n=(long double)mynum/(long double)(UINT_MAX );
		n=xmax*n;
		return n;
}
```

C. Código de programa para realizar simulador gráfico del robot móvil C++

Variables Globales

```
const unsigned long NetSize=20000; //size of Neural Network
const unsigned DataBase=43;
const unsigned MIF=1;                //percentage for joint vector
const int RecCycle=1;                        //Number of recalculation cycles
const unsigned Tcycles=1;//number of training cycles
const long int AssemblySize=200;  //48            //size of Neuron Assembly
const unsigned long AssemblyNumber=DataBase;        //number of Neuron Assemblies
const unsigned long MatSize1=NetSize/8;
int SUT=(int)AssemblySize*1.1;          //SUT level
const unsigned MaxValue=600;
const unsigned ShiftRange=10;  //correlation gap
const unsigned ShiftNum=MaxValue/ShiftRange;//250;
const unsigned BCN=1;
long int myhs;
unsigned char Wmax=16;                  //synaptic weight maximum
unsigned char buff[NetSize];                     //working array
unsigned char buff1[NetSize];                     //working array
unsigned char buff2[NetSize];                     //working array
unsigned wb1[NetSize]={0};
unsigned wb2[NetSize]={0};
int buff3[NetSize];                       //working array
float Noise=0.5;  //probability of bit value changing in the binary vector of activity during
recognition process
const unsigned long MatSize=(NetSize*NetSize)/8;      //size of the matrix
unsigned char *matr0 = new unsigned char [MatSize];   //synaptic matrix 0
unsigned char *matr1 = new unsigned char [MatSize];   //synaptic matrix 1
unsigned char *matr2 = new unsigned char [MatSize];   //synaptic matrix 2
unsigned char *matr3 = new unsigned char [MatSize];   //synaptic matrix 3
unsigned char *matr4 = new unsigned char [MatSize];   //synaptic matrix 4
unsigned char *matr5 = new unsigned char [MatSize];   //synaptic matrix 5
unsigned char *matr6 = new unsigned char [MatSize];   //synaptic matrix 6
unsigned char *matr7 = new unsigned char [MatSize];   //synaptic matrix 7
unsigned long ActiveNumber[AssemblySize];  //numbers of active neurons
int error=0;                                 //number of recall errors
int OverlapLevel=(int)AssemblySize*0.9;          //acceptable level of overlap of restored assembly
with stored assembly
unsigned long Excitation[NetSize];              //array of neuron excitations
const unsigned Xscale=1;                        //histogram scale in X-direction
const unsigned Yscale=10;                        //histogram scale in Y-direction
//variables for shift
unsigned OD1,OA1,OR1,OD2,OA2,OR2,OD3,OA3,OR3,RX,RY,RA,RR,RP; //decoding reading file
long unsigned *MShiftOD=new long unsigned[ShiftNum*NetSize];
long unsigned *MShiftOA=new long unsigned[ShiftNum*NetSize];
long unsigned *MShiftOR=new long unsigned[ShiftNum*NetSize];
long unsigned *MShiftRX=new long unsigned[ShiftNum*NetSize];
long unsigned *MShiftRY=new long unsigned[ShiftNum*NetSize];
long unsigned *MShiftRR=new long unsigned[ShiftNum*NetSize];
long unsigned *Shift=new long unsigned[ShiftNum*NetSize];
//long unsigned Shift[ShiftNum*NetSize]={0};
long unsigned misuma[ShiftNum*NetSize]={0};
const int numbfile=3;                                    //Number of obstacle to save
const int numbmov=1;
const int mWidth=500;                                    //Width of area
const int mHeight=400;                                   //Height of area
int centerX=(int)(mWidth/2);                          //CenterX of robot posotion
int centerY=(int)(mHeight/2);                         //CenterY of robot position
//dimension of robot
const int DimW=10;                   //Width of robot
const int DimH=20;                   //Height of robot
const int steps=200;                 //number of steps bye movement
#define pi 3.14159
const int Obst = 15;                 //number of the obstackles
const int Rmin = 10;                 //minimal radius of the obstackle
const int Rmax = 40;                 //maximal radius of the obstackle
int seed = 211;                             //initial value for random numbers
static int firstt=0;
static float totalx,totaly,totala;
int tofile[numbfile*3+5];
float distances[Obst][3]={0};
int dist1=0;
float dist2=0;
float dist3=0;
float dist4=0;
```

Simulador gráfico

```
float distances1[Obst][3]={0};
float pointcx;         //this is the x coordinate of the center for the movement
float pointcy;         //this is the y coordinate of the center for the movement
float pointx2;         //final position of movement
float pointy2;         //final position of movement
double radio;          //radio for the movement
double angle;          //angle for the movement
float PosX;                   //X coordinate of robot
float PosY;                   //Y coordinate of robot
int i,n=steps;
float delta;
int f1,f2,f3;
float radiorobot;
float dx,dy;
float mindist,phi,alpha,gamma,subangle; //to calculate minimal distance to limits
float xa,ya,xb,yb;
//Variables for Obstacles
System::Random ^R1=gcnew System::Random(seed);      //Random numbers
int CenterPointX, CenterPointY, j ,k, Radius,x,y,x0,y0,r;
float x2,y2,r2,auxx,auxy;
static int CenterPointArray[Obst][2];      //center points of the obstackles
static int RadiusArray[Obst];
radiorobot=2*DimH;                    //to clean area for initial position of robot
Pen^ myPen=gcnew Pen(Brushes::Black,1);
System::Drawing::Drawing2D::Matrix^ Position = gcnew System::Drawing::Drawing2D::Matrix;
System::Drawing::Rectangle myrobot=System::Drawing::Rectangle(0,0,DimW,DimH);
System::Drawing::Rectangle myfront=System::Drawing::Rectangle(DimW/3,DimH/5,DimW/3,DimH/5);
System::Drawing::Rectangle rect= System::Drawing::Rectangle(200,100,50,50);
System::Drawing::SolidBrush^ color = gcnew
System::Drawing::SolidBrush(System::Drawing::Color::DarkSlateGray);
System::Drawing::SolidBrush^ eraser = gcnew
System::Drawing::SolidBrush(System::Drawing::Color::White);
System::Drawing::SolidBrush^ colorobs = gcnew
System::Drawing::SolidBrush(System::Drawing::Color::Red);
System::Drawing::SolidBrush^ front = gcnew
System::Drawing::SolidBrush(System::Drawing::Color::Yellow);
System::Drawing::SolidBrush^ track = gcnew
System::Drawing::SolidBrush(System::Drawing::Color::LightBlue);
//reseting:
//Identify if the Button is pushed by first time
       if(firstt==0)
                {
                       dist1=0;
                       dist2=0;
                       dist3=0;
                       for (i=0;i<=Obst;i++)
                       {
                              distances[i][0]=i;
                       }
                       for (i=0;i<=Obst;i++)
                       {
                              distances1[i][0]=distances[i][0];
                              distances1[i][1]=distances[i][1];
                              distances1[i][2]=distances[i][2];
                       }
                       //Draw initial position of the robot
                       this->button2->Text= L"Move";
                       formGraphics->FillRectangle(System::Drawing::Brushes::White ,
System::Drawing::Rectangle(1,1,mWidth,mHeight) );
                       formGraphics-
>DrawRectangle(myPen,System::Drawing::Rectangle(0,0,mWidth+2,mHeight+2));
                       Position->Translate(centerX-DimW/2,centerY-
DimH/2,System::Drawing::Drawing2D::MatrixOrder::Append);
                       this->Position2->Translate(centerX-DimW/2,centerY-
DimH/2,System::Drawing::Drawing2D::MatrixOrder::Append);
                       this->Robot->Transform=Position2;
                       Robot->FillRectangle(color,myrobot);
                       Robot->FillRectangle(front,myfront);
                       firstt=1;
                       //Generate Obstacles
                       for(i=0;i<Obst;i++)
                       {
again:
```

```
                                RadiusArray[i] = R1->Next(Rmin,Rmax);
//generate random radius
                                CenterPointArray[i][0] = R1->Next(Rmax,mWidth-Rmax);
//generate random CenterPoints
                                CenterPointArray[i][1] = R1->Next(Rmax,mHeight-Rmax);        //generate
random CenterPoints
                        //check for obstacle in center
                                if(((CenterPointArray[i][0]+RadiusArray[i])>(centerX-
radiorobot))&&((CenterPointArray[i][0]-RadiusArray[i])<(centerX+radiorobot)))
                                {
                                        if(((CenterPointArray[i][1]+RadiusArray[i])>(centerY-
radiorobot))&&((CenterPointArray[i][1]-RadiusArray[i])<(centerY+radiorobot)))
                                                goto again;
                                }
                        }
                        //Draw obstacles
                        for(i=0;i<Obst;i++)
                        {
                              formGraphics->FillEllipse(colorobs,CenterPointArray[i][0]-
RadiusArray[i],CenterPointArray[i][1]-RadiusArray[i],RadiusArray[i]*2,RadiusArray[i]*2);
                        }

                        //define the inital coordinates of the robot
                        totala=90;
                        totalx=centerX;
                        totaly=centerY;
                        //Show initial coordinates
                        this->label9->Text= totalx.ToString();
                        this->label10->Text= totaly.ToString();
                        this->label11->Text= totala.ToString();
                        this->label9->Refresh();
                        this->label10->Refresh();
                        this->label11->Refresh();
                        goto ending_start;
                }
                //if it is not the first time read input data from user
                //erase actual position
                Robot->FillRectangle(eraser,myrobot);
                //Read Data from User
                radio=System::Convert::ToDouble  (this->textBox1->Text);
                angle=System::Convert::ToDouble  (this->textBox2->Text);

                if (this->radioButton3->Checked)//clockwise
                        f1=1;
                else
                        f1=-1;
                if (this->radioButton1->Checked)//right movement
                      f2=1;
                else
                        f2=-1;
                if (this->radioButton6->Checked)//straigth line
                      f3=0;
                else
                      f3=1;


                // movement by n steps


               delta=angle/n;
                for(i=1;i<=n;i++){
                        if(f3==1)
                        {
                        //calculate center point of circle of movement
                        pointcx=f2*radio+PosX;
                        pointcy=PosY;
                        //calculate ending point
                       pointx2=pointcx-f2*radio*cos(pi*delta/180);
                       pointy2=PosY-f1*radio*sin(pi*delta/180);
                        //Calculation of delta for coordinates
                        dx=f2*(radio-radio*cos(pi*delta/180));
                        dy=f1*radio*sin(pi*delta/180);
                        //upgrade possible new coordinates
                        auxx=totalx+dy*cos((totala)*pi/180)+dx*cos((totala-90)*pi/180);
                        auxy=totaly-dy*sin((totala)*pi/180)+dx*sin((totala-90)*pi/180);
                        }
                        else
                       {
```

```
                        delta=radio/n;
                        subangle=totala;
                        dx=f1*delta*cos(pi*totala/180);
                        dy=-1*f1*delta*sin(pi*totala/180);
                                auxx=totalx+dx;
                                auxy=totaly+dy;
                        }

                        //before translating it must be checked if there would be any collision
                        for(j=0;j<Obst;j++)
                        {
                                        //we localize the position obstacle by obstacle
                                        x0 = CenterPointArray[j][0];
                                        y0 = CenterPointArray[j][1];
                                        r = RadiusArray[j];
                                        x2=(auxx-(x0))*(auxx-(x0));
                                        y2=(auxy-(y0))*(auxy-(y0));
                                        r2=sqrt(x2+y2);
                                        x2=sqrt(x2);
                                        y2=sqrt(y2);
                                        //upgrade next possible angle, and always between 0-360º
                                        subangle=totala-f2*f1*delta;
                                        if(subangle>=360)
                                                        subangle=subangle-360;
                                        if(subangle<0)
                                                        subangle=360+subangle;
                                        //calculate minimal distance
                                        subangle=subangle*pi/180;
                                        //point for vector to the front of robot
                                        xa=abs(cos(subangle)*DimH/2);
                                        ya=abs(sin(subangle)*DimH/2);
                                        //vector to center of obstacle
                                        xb=abs(x0-auxx);
                                        yb=abs(y0-auxy);
                                        //angle of treshold, corner of robot rectanle
                                        gamma=atan((float)(DimW)/(float)(DimH));
//angle between front vector and vector to center of obstacle by internal prodcut of vectors
phi=acos((xa*xb+ya*yb)/((sqrt(xa*xa+ya*ya))*(sqrt(xb*xb+yb*yb))));
                                        //distance in direction to the obstacle from center of robot
                                        if(phi<=gamma)
                                                mindist=DimH/(2*cos(phi));
                                       if(phi>gamma)
                                                mindist=DimW/(2*sin(phi));
                                        if(phi<0.001)
                                                mindist=DimH/2;
                                        if(phi>(pi/2-0.001))
                                                mindist=DimW/2;
                                         mindist=mindist+2;
                                       phi=phi*180/pi;
                                        if(i==1)
                                                {
                                                        int mdx,mdy,aaa;
                                                        distances[j][1]=r2-r-mindist;
                                                        mdx=x0-totalx;
                                                        mdy=y0-totaly;
                                                        aaa=distances[j][1];

                                                        aaa=CenterPointArray[j][0];
                                                        aaa=totalx;
                                                        if(mdx>0)//obstacle is right
                                                                if(mdy<0) //obstacle is up
                                                                        distances[j][2]=360-phi;//*180/pi;
                                                                else //obstacle is down
                                                                        distances[j][2]=180+phi;
                                                        else //obstacle is left
                                                                if(mdy<0) //obstacle is up
                                                                        distances[j][2]=phi;//*180/pi;
                                                                else//obstacle is down
                                                                        distances[j][2]=180-phi;

                                                }

                                        if(((r2-r))<mindist)//checking distance to obstacle
                                        {
                                       //if the step cross minimal distance, this step ist not taken
                                                        goto letsgo;
                                        }
                        }
```

```
                        if(i==1)
                        {
                        for (j=0;j<=Obst;j++)
                        {
                               distances1[j][0]=distances[j][0];
                               distances1[j][1]=distances[j][1];
                               distances1[j][2]=distances[j][2];
                        }
                       j=0;
                        //3 closest objects
                        for(int jj=0;jj<Obst-1;jj++)
                        {
                               dist1=distances[jj][1];
                               for(j=jj+1;j<Obst;j++)
                               {
                                        if(distances[j][1]<distances[jj][1])
                                       {
                                              //change
                                              dist2=distances[j][0];
                                              dist3=distances[j][1];
                                              dist4=distances[j][2];

                                              distances[j][0]=distances[jj][0];
                                              distances[j][1]=distances[jj][1];
                                              distances[j][2]=distances[jj][2];

                                              distances[jj][0]=dist2;
                                              distances[jj][1]=dist3;
                                              distances[jj][2]=dist4;
                                       }

                               }
                        }
                        for (int ij=0;ij<=Obst;ij++)
                                       {
                                              distances1[ij][0]=distances[ij][0];
                                              distances1[ij][1]=distances[ij][1];
                                              distances1[ij][2]=distances[ij][2];
                                       }
                        //prepare for file
                        for(int ij=0;ij<numbfile;ij++)
                        {
                               dist1=distances[ij][0];
                               tofile[3*ij]=distances1[dist1][1];//distance between robot and
obstacle//CenterPointArray[dist1][0];
                               tofile[3*ij+1]=distances1[dist1][2];//angle between robot and
obstacle//CenterPointArray[dist1][1];
                               tofile[3*ij+2]=RadiusArray[dist1];

                        }
                        int helpi[numbfile*3+5];
                        for (int ij=0;ij<=numbfile*3+5;ij++)
                        {
                               helpi[ij]=tofile[ij];
                        }
                        }

                        x0=auxx+10;   //x point vector for distance to limit
                        y0=auxy;              //y point vector for distance to limit
                        subangle=subangle*pi/180;
                        xa=abs(cos(subangle)*DimH/2);
                        ya=abs(sin(subangle)*DimH/2);
                        xb=abs(x0-auxx);
                        yb=abs(y0-auxy);
                        gamma=atan((float)(DimW)/(float)(DimH));
                        phi=acos((xa*xb+ya*yb)/((sqrt(xa*xa+ya*ya))*(sqrt(xb*xb+yb*yb))));
                        if(phi<=gamma)
                               mindist=DimH/(2*cos(phi));
                        if(phi>gamma)
                               mindist=DimW/(2*sin(phi));
                        if(phi<0.001)
                               mindist=DimH/2;
                        if(phi>(pi/2-0.001))
                               mindist=DimW/2;
                        mindist=mindist+2;
                        if(((auxx-mindist)<0)||((auxx+mindist)>mWidth))//checking distance to
horizontal limits  <-- o -->
                        {
```

```
                                //don't move
                                goto letsgo;
                        }
                        //for the other limits
                        x0=auxx;
                        y0=auxy-10;
                        subangle=subangle*pi/180;
                        xa=abs(cos(subangle)*DimH/2);
                        ya=abs(sin(subangle)*DimH/2);
                        xb=abs(x0-auxx);
                        yb=abs(y0-auxy);
                        gamma=atan((float)(DimW)/(float)(DimH));
                        phi=acos((xa*xb+ya*yb)/((sqrt(xa*xa+ya*ya))*(sqrt(xb*xb+yb*yb))));
                        if(phi<=gamma)
                                mindist=DimH/(2*cos(phi));
                        if(phi>gamma)
                                mindist=DimW/(2*sin(phi));
                        if(phi<0.001)
                                mindist=DimH/2;
                        if(phi>(pi/2-0.001))
                                mindist=DimW/2;
                        mindist=mindist+6;
                        if(((auxy-mindist)<0)||((auxy+mindist)>mHeight))//checking distance to
vertical limits
                        {
                                //don't move
                                goto letsgo;
                        }
                       //If the code reaches this point, the robot moves on and upgrade coordinates
                        if(f3==1)
                        {
                               //new coordinates
                               totalx=totalx+dy*cos((totala)*pi/180)+dx*cos((totala-90)*pi/180);
                               totaly=totaly-dy*sin((totala)*pi/180)+dx*sin((totala-90)*pi/180);
                               totala=totala-f2*f1*delta;//new angle of robot
                               if(totala>=360)
                                       totala=totala-360;
                               if(totala<0)
                                       totala=360+totala;
                               //Translate robot
                               Position2->Translate(f2*radio-f2*radio*cos(pi*delta/180),-
f1*radio*sin(pi*delta/180));
                               this->Robot->Transform=Position2;
                               //Rotate robot
                               Position2->RotateAt(f2*f1*delta,System::Drawing::Point(DimW/2,DimH/2));
                               Robot->Transform=Position2;
                        }
                        else
                        {
                                //this->label2->Text=mindist.ToString();
                                totalx=totalx+dx;
                                totaly=totaly+dy;
                                //Translate robot
                                Position2->Translate(0,-f1*delta);
                                this->Robot->Transform=Position2;
                                //Draw Trayectory
                                Robot->FillEllipse(track,DimW/2,DimH,2,2);
                                Robot->FillEllipse(track,DimW/2,1,2,2);

                       }

                       //Draw Trayectory
                       Robot->FillEllipse(track,DimW/2,DimH,2,2);
                       Robot->FillEllipse(track,DimW/2,1,2,2);
               }
letsgo:
               //Draw robot in final position
               Robot->FillRectangle(color,myrobot);//body of robot
               Robot->FillRectangle(front,myfront);//mark on front


               tofile[numbfile*3]=totalx;//deltax//centerX;//x initial
               tofile[numbfile*3+1]=totaly;//deltay//centerY;//y initial
               tofile[numbfile*3+2]=radio;//90;//alfa initial
               //tofile[numbfile*3+3]=radio;//r movement
               //tofile[numbfile*3+4]=totala;//alfa final

               int helpiu[numbfile*3+3];
```

```
                for (int ij=0;ij<=numbfile*3+3;ij++)
                {
                        helpiu[ij]=tofile[ij];
                }
                        //save file
                         //File for data
                         FILE *fp ;
                     fp=fopen("datarobotx.dat","ab");
                         //fwrite(RadiusArray,4,Obst,fp);
                         fwrite(tofile,4,numbfile*3+3,fp);
                         fclose(fp);
                seed=seed+1;
                this->label5->Text=seed.ToString();
                firstt=0;
                Position2->Reset();
                this->button2->Text= L"Start";
                //goto reseting;


ending_start:
                //final coordinates are schown
                this->label9->Text= totalx.ToString();
                this->label10->Text= totaly.ToString();
                this->label11->Text= totala.ToString();
                this->label9->Refresh();
                this->label10->Refresh();
                this->label11->Refresh();
```

Creación de máscaras

```
//creation of masks
                 long unsigned i,k,q;
                 FILE *fp ;

                 this->progressBar1->Maximum=NetSize-1;
                 ////////////////////////////////////////////////////////////////////
                 //permutation obj d
                 long unsigned mybuff[NetSize];
                 this->progressBar1->Value=0;
                for(i=0;i<NetSize;i++) //clean buffer
                        buff[i]=0;
                for(i=0;i<NetSize;i++) //index for permutations
                {
                        this->progressBar1->Value=i;
                        this->label3->Text = System::String::Concat(L"i= ",i.ToString()) ;
                        this->Show();
                        k=1;
                        while(k!=0)
                        {
                                q=Random(NetSize);
                                k=buff[q];
                        }
                        buff[q]=1;
                        mybuff[i]=q;
                }
                fp=fopen("ShiftOD.dat","wb");
                if(fwrite(mybuff,4,NetSize,fp)!=NetSize)
                {
                        this->label3->Text= L"Cannot wrote Assembly";
                }
                fclose(fp);
                this->label3->Text= L"ShiftOD created";
                this->Show();
                ////////////////////////////////////////////////////////////////////////////////
                //permutation obj a
                this->progressBar1->Value=0;
                for(i=0;i<NetSize;i++) //clean buffer
                        buff[i]=0;
                for(i=0;i<NetSize;i++) //index for permutations
                {
                        this->progressBar1->Value=i;
                        this->Show();
                        k=1;
                        while(k!=0)
                        {
```

```
                        q=Random(NetSize);
                        k=buff[q];
                }
                buff[q]=1;
                mybuff[i]=q;
        }
        fp=fopen("ShiftOA.dat","wb");
        if(fwrite(mybuff,4,NetSize,fp)!=NetSize)
        {
                this->label3->Text= L"Cannot wrote Assembly";
        }
        long unsigned auxb[NetSize];
        for (i=0;i<NetSize;i++)
                auxb[i]=buff1[i];
        fclose(fp);
        this->label3->Text= L"ShiftOA created";
        this->Show();
        ////////////////////////////////////////////////////////////////////////////////
        //permutation obj r
        this->progressBar1->Value=0;
        for(i=0;i<NetSize;i++) //clean buffer
                buff[i]=0;
        for(i=0;i<NetSize;i++) //index for permutations
        {
                this->progressBar1->Value=i;
                this->Show();
                k=1;
                while(k!=0)
                {
                        q=Random(NetSize);
                        k=buff[q];
                }
                buff[q]=1;
                mybuff[i]=q;
        }
        fp=fopen("ShiftOR.dat","wb");
        if(fwrite(mybuff,4,NetSize,fp)!=NetSize)
        {
                this->label3->Text= L"Cannot wrote Assembly";
        }
        fclose(fp);
        this->label3->Text= L"ShiftOR created";
        this->Show();
////////////////////////////////////////////////////////////////////////////////
        //permutation rob x
        this->progressBar1->Value=0;
        for(i=0;i<NetSize;i++) //clean buffer
                buff[i]=0;
        for(i=0;i<NetSize;i++) //index for permutations
        {
                this->progressBar1->Value=i;
                this->Show();
                k=1;
                while(k!=0)
                {
                        q=Random(NetSize);
                        k=buff[q];
                }
                buff[q]=1;
                mybuff[i]=q;
        }
        fp=fopen("ShiftRX.dat","wb");
        if(fwrite(mybuff,4,NetSize,fp)!=NetSize)
        {
                this->label3->Text= L"Cannot wrote Assembly";
        }
        fclose(fp);
        this->label3->Text= L"ShiftRX created";
        this->Show();
////////////////////////////////////////////////////////////////////////////////
        //permutation rob y
        this->progressBar1->Value=0;
        for(i=0;i<NetSize;i++) //clean buffer
                buff[i]=0;
        for(i=0;i<NetSize;i++) //index for permutations
        {
                this->progressBar1->Value=i;
                this->Show();
```

```
                        k=1;
                        while(k!=0)
                        {
                                q=Random(NetSize);
                                k=buff[q];
                        }
                        buff[q]=1;
                        mybuff[i]=q;
                }
                fp=fopen("ShiftRY.dat","wb");
                if(fwrite(mybuff,4,NetSize,fp)!=NetSize)
                {
                        this->label3->Text= L"Cannot wrote Assembly";
                }
                fclose(fp);
                this->label3->Text= L"ShiftRY created";
                this->Show();
////////////////////////////////////////////////////////////////////////////
                //permutation rob r
                this->progressBar1->Value=0;
                for(i=0;i<NetSize;i++) //clean buffer
                        buff[i]=0;
                for(i=0;i<NetSize;i++) //index for permutations
                {
                        this->progressBar1->Value=i;
                        this->Show();
                        k=1;
                        while(k!=0)
                        {
                                q=Random(NetSize);
                                k=buff[q];
                        }
                        buff[q]=1;
                        mybuff[i]=q;
                }
                fp=fopen("ShiftRR.dat","wb");
                if(fwrite(mybuff,4,NetSize,fp)!=NetSize)
                {
                        this->label3->Text= L"Cannot wrote Assembly";
                }
                fclose(fp);
                this->label3->Text= L"ShiftRR created";
                this->Show();
```

Codificación

```
                 long unsigned i,j;
                 unsigned p,q;
                 FILE *fi;
                 FILE *fp;
                long unsigned mybuff[NetSize];
                 //implementing permutation
//permutation Ox//////////////////////////////////////////////////////////////
                 for(j=0;j<ShiftNum;j++) //vector initialization
                 {
                        for(i=0;i<NetSize;i++)
                        {
                                MShiftOD[j*NetSize+i]=0;
                        }
                 }
                 fp=fopen("ShiftOD.dat","rb");
                 if(fp==NULL)
                 {
                         this->label2->Text= L"Cannot read ShiftOD1.dat";
                         goto endp;
                 }
                 if(fread(mybuff,4,NetSize,fp)!=NetSize)
                 {
                        this->label2->Text= L"Cannot read ShiftOD2.dat";
                        goto endp;
                 }
                 for(i=0;i<NetSize;i++)
                 {
                        MShiftOD[i]=mybuff[i];
                 }
                 for(j=0;j<ShiftNum-1;j++)
                 {
                        for(i=0;i<NetSize;i++)
                        {
                                q=MShiftOD[i];
```

```
                                p=MShiftOD[j*NetSize+q];
                                MShiftOD[(j+1)*NetSize+i]=p;
                        }
                }
                fclose(fp);
//permutation OA//////////////////////////////////////////////////////////////
                for(j=0;j<NetSize;j++)
                        mybuff[j]=0;
                for(j=0;j<ShiftNum;j++) //vector initialization
                {
                        for(i=0;i<NetSize;i++)
                        {
                                MShiftOA[j*NetSize+i]=0;
                        }
                }
                fp=fopen("ShiftOA.dat","rb");
                if(fp==NULL)
                {
                        this->label2->Text= L"Cannot read ShiftOA1.dat";
                        goto endp;
                }
                if(fread(mybuff,4,NetSize,fp)!=NetSize)
                {
                        this->label2->Text= L"Cannot read ShiftOA2.dat";
                        goto endp;
                }
                for(i=0;i<NetSize;i++)
                {
                        MShiftOA[i]=mybuff[i];
                }
                for(j=0;j<ShiftNum-1;j++)
                {
                        for(i=0;i<NetSize;i++)
                        {
                                q=MShiftOA[i];
                                p=MShiftOA[j*NetSize+q];
                                MShiftOA[(j+1)*NetSize+i]=p;
                        }
                }
                fclose(fp);
//permutation OR//////////////////////////////////////////////////////////////
                for(j=0;j<NetSize;j++)
                        mybuff[j]=0;
                for(j=0;j<ShiftNum;j++) //vector initialization
                {
                        for(i=0;i<NetSize;i++)
                        {
                                MShiftOR[j*NetSize+i]=0;
                        }
                }
                fp=fopen("ShiftOR.dat","rb");
                if(fp==NULL)
                {
                        this->label2->Text= L"Cannot read ShiftOR1.dat";
                        goto endp;
                }
                if(fread(mybuff,4,NetSize,fp)!=NetSize)
                {
                        this->label2->Text= L"Cannot read ShiftOR2.dat";
                        goto endp;
                }
                for(i=0;i<NetSize;i++)
                {
                        MShiftOR[i]=mybuff[i];
                }
                for(j=0;j<ShiftNum-1;j++)
                {
                        for(i=0;i<NetSize;i++)
                        {
                                q=MShiftOR[i];
                                p=MShiftOR[j*NetSize+q];
                                MShiftOR[(j+1)*NetSize+i]=p;
                        }
                }
                fclose(fp);

//permutation RX//////////////////////////////////////////////////////////////
                for(j=0;j<NetSize;j++)
                        mybuff[j]=0;
                for(j=0;j<ShiftNum;j++) //vector initialization
                {
                        for(i=0;i<NetSize;i++)
                        {
                                MShiftRX[j*NetSize+i]=0;
                        }
                }
                fp=fopen("ShiftRX.dat","rb");
                if(fp==NULL)
                {
```

```
                        this->label2->Text= L"Cannot read ShiftRX1.dat";
                        goto endp;
                }
                //fread(mybuff,2,NetSize,fp);
                if(fread(mybuff,4,NetSize,fp)!=NetSize)
                {
                        this->label2->Text= L"Cannot read ShiftRX2.dat";
                        goto endp;
                }
                for(i=0;i<NetSize;i++)
                {
                        MShiftRX[i]=mybuff[i];
                }
                for(j=0;j<ShiftNum-1;j++)
                {
                        for(i=0;i<NetSize;i++)
                        {
                                q=MShiftRX[i];
                                p=MShiftRX[j*NetSize+q];
                                MShiftRX[(j+1)*NetSize+i]=p;
                        }
               }
               fclose(fp);
//permutation RY//////////////////////////////////////////////////////////////
                for(j=0;j<NetSize;j++)
                        mybuff[j]=0;
                for(j=0;j<ShiftNum;j++) //vector initialization
                {
                        for(i=0;i<NetSize;i++)
                        {
                                MShiftRY[j*NetSize+i]=0;
                        }
                }
                fp=fopen("ShiftRY.dat","rb");
                if(fp==NULL)
                {
                        this->label2->Text= L"Cannot read ShiftRY1.dat";
                        goto endp;
                }
                if(fread(mybuff,4,NetSize,fp)!=NetSize)
                {
                        this->label2->Text= L"Cannot read ShiftRY2.dat";
                        goto endp;
                }
                for(i=0;i<NetSize;i++)
                {
                        MShiftRY[i]=mybuff[i];
                }
                for(j=0;j<ShiftNum-1;j++)
                {
                        for(i=0;i<NetSize;i++)
                        {
                                q=MShiftRY[i];
                                p=MShiftRY[j*NetSize+q];
                                MShiftRY[(j+1)*NetSize+i]=p;
                        }
               }
               fclose(fp);

//permutation RR//////////////////////////////////////////////////////////////
                for(j=0;j<NetSize;j++)
                        mybuff[j]=0;
                for(j=0;j<ShiftNum;j++) //vector initialization
                {
                        for(i=0;i<NetSize;i++)
                        {
                                MShiftRR[j*NetSize+i]=0;
                        }
                }
                fp=fopen("ShiftRR.dat","rb");
                if(fp==NULL)
                {
                        this->label2->Text= L"Cannot read ShiftRR1.dat";
                        goto endp;
                }
                if(fread(mybuff,4,NetSize,fp)!=NetSize)
                {
                        this->label2->Text= L"Cannot read ShiftRR2.dat";
                        goto endp;
                }
                for(i=0;i<NetSize;i++)
                {
                        MShiftRR[i]=mybuff[i];
                }
                for(j=0;j<ShiftNum-1;j++)
                {
                        for(i=0;i<NetSize;i++)
                        {
                                q=MShiftRR[i];
```

```
                                p=MShiftRR[j*NetSize+q];
                                MShiftRR[(j+1)*NetSize+i]=p;
                        }
                }
                fclose(fp);
//permutating
                 //read datarobot
                long unsigned ent,rest,Pos,part,n;
                long unsigned mbuff[14]={0};
                long unsigned nbuff[AssemblySize]={0};
                long unsigned nbuff1[AssemblySize]={0};
                long unsigned nbuff2[AssemblySize]={0};
                long unsigned z=0;
                FILE *fg;
                FILE *fp2;
                 fg=fopen("datarobotx.dat","rb");
                 if(fg==NULL)
                 {
                         this->label2->Text= L"Cannot read datarobot1.dat";
                         goto endp;
                 }
                 fp=fopen("assemblies.dat","wb");
                 if(fp==NULL)
                 {
                        this->label2->Text= L"Cannot read assembliesmasks1.dat";
                        goto endp;
                 }
                 fp2=fopen("assembliesS.dat","wb");
                 if(fp==NULL)
                 {
                        this->label2->Text= L"Cannot read assembliesmasks1.dat";
                        goto endp;
                 }
        FILE *decode;
        decode=fopen("decoded.dat","wb");
        unsigned decoded[3];
///////for DataBase cycles
for(z=0;z<DataBase;z++)
{
        unsigned cbuff[NetSize]={0};
                unsigned hbuff[NetSize]={0};
                unsigned hbuff2[NetSize]={0};
                unsigned hbuff3[NetSize]={0};
                unsigned hbuff4[NetSize]={0};
                unsigned hbuff5[NetSize]={0};
         if(fread(mbuff,4,12,fg)!=12)
         {
                this->label2->Text= L"Cannot read datarobot.dat";
                goto endp;
         }
         //decode values
        OD1=(unsigned)(mbuff[0]);
        OA1=(unsigned)(mbuff[1]);
        OR1=(unsigned)(mbuff[2]);
        OD2=(unsigned)(mbuff[3]);
        OA2=(unsigned)(mbuff[4]);
        OR2=(unsigned)(mbuff[5]);
        OD3=(unsigned)(mbuff[6]);
        OA3=(unsigned)(mbuff[7]);
        OR3=(unsigned)(mbuff[8]);
        RX=(unsigned)(mbuff[9]);
        RY=(unsigned)(mbuff[10]);
        RR=(unsigned)(mbuff[11]);
        decoded[0]=RX;
        decoded[1]=RY;
        decoded[2]=RR;
        //save values to campare in decodification
        if(fwrite(decoded,4,3,decode)!=3)
        {
                this->label3->Text= L"Cannot wrote Decoded";
        }
        //read masks assemblies
         fi=fopen("assembliesmasks.dat","rb");
         if(fi==NULL)
         {
                this->label2->Text= L"Cannot read assembliesmasks1.dat";
                goto endp;
         }
        //nbuff1 read mask of OD
        if(fread(nbuff1,4,AssemblySize,fi)!=AssemblySize)
        {
                this->label2->Text= L"Cannot read assembliesmasks A.dat";
                goto endp;
        }
        //Copy the original mask of OD
        for(n=0;n<AssemblySize;n++)
        {
                nbuff[n]=0;
                nbuff[n]=nbuff1[n];
```

```
        }

        Setpos(nbuff,cbuff,AssemblySize);
//Permutation for OD1//////////////////////////////////////
        Perm(nbuff1,nbuff,OD1,1);
         // set position after 1st permutation in hbuff
        Setpos(nbuff,hbuff3,AssemblySize);
//Permutation for OD2//////////////////////////////////////
        Perm(nbuff1,nbuff,OD2,1);
        Setpos(nbuff,hbuff3,AssemblySize);
//Permutation for OD3//////////////////////////////////////
 Perm(nbuff1,nbuff,OD3,1);
 Setpos(nbuff,hbuff3,AssemblySize);
//Read next mask for OA
         //read mask for OA in nbuff1
         if(fread(nbuff1,4,AssemblySize,fi)!=AssemblySize)
        {
                this->label2->Text= L"Cannot read assembliesmasks B.dat";
                goto endp;
        }
//Permutation for OA1//////////////////////////////////////
         Perm(nbuff1,nbuff,OA1,2);
         Setpos(nbuff,hbuff4,AssemblySize);
//Permutation for OA2//////////////////////////////////////
         Perm(nbuff1,nbuff,OA2,2);
         Setpos(nbuff,hbuff4,AssemblySize);
//Permutation for OA3//////////////////////////////////////
         Perm(nbuff1,nbuff,OA3,2);
         Setpos(nbuff,hbuff4,AssemblySize);
//Read next mask for OR
         if(fread(nbuff1,4,AssemblySize,fi)!=AssemblySize)
        {
                this->label2->Text= L"Cannot read assembliesmasks C.dat";
                goto endp;
        }
 //Permutation for OR1//////////////////////////////////////
         Perm(nbuff1,nbuff,OR1,3);
         Setpos(nbuff,hbuff5,AssemblySize);
//Permutation for OR2//////////////////////////////////////
         Perm(nbuff1,nbuff,OR2,3);
         Setpos(nbuff,hbuff5,AssemblySize);
  //Permutation for OR3//////////////////////////////////////
         Perm(nbuff1,nbuff,OR3,3);
         Setpos(nbuff,hbuff5,AssemblySize);
//Read next mask for RX
         if(fread(nbuff,4,AssemblySize,fi)!=AssemblySize)
        {
                this->label2->Text= L"Cannot read assembliesmasks D.dat";
                goto endp;
        }
//Permutation for RX//////////////////////////////////////
         Perm(nbuff,nbuff,RX,4);
         Setpos(nbuff,hbuff2,AssemblySize);
//Read next mask for RY
         if(fread(nbuff,4,AssemblySize,fi)!=AssemblySize)
        {
                this->label2->Text= L"Cannot read assembliesmasks E.dat";
                goto endp;
        }
//Permutation for RY//////////////////////////////////////
         Perm(nbuff,nbuff,RY,5);
         Setpos(nbuff,hbuff2,AssemblySize);
//Read next mask for RR
         if(fread(nbuff,4,AssemblySize,fi)!=AssemblySize)
        {
                this->label2->Text= L"Cannot read assembliesmasks F.dat";
                goto endp;
        }
//Permutation for RR//////////////////////////////////////
         Perm(nbuff,nbuff,RR,6);
         Setpos(nbuff,hbuff2,AssemblySize);
fclose(fi);
         //Binding //////////////////////////////////////Normalization
         //Normalization for OR3
         Norm(hbuff5,AssemblySize);
         //Normalization for Angles
         Norm(hbuff4,AssemblySize);

         //Normalization for Distances
         Norm(hbuff3,AssemblySize);
        for(n=0;n<NetSize;n++)
        {
                if(hbuff3[n]==1)
                        hbuff[n]=1;
                if(hbuff5[n]==1)
                        hbuff[n]=1;
                if(hbuff4[n]==1)
                        hbuff[n]=1;
        }
```

```
        //Normalization for parameter of Obstacles
        Norm(hbuff,AssemblySize*2);
        for(i=0;i<AssemblySize;i++)
                nbuff2[i]=0;
        i=Xpos(hbuff,nbuff2); //extract position from hbuff->nbuff2
        //fill 0's with the first active neuron
        if(i<AssemblySize)
        {
                for(n=i;n<AssemblySize;n++)
                        nbuff2[n]=nbuff2[0];
        }
        //save data for recognition
         if(fwrite(nbuff2,4,AssemblySize,fp2)!=AssemblySize)
        {
                this->label3->Text= L"Cannot wrote AssemblyS";
        }
        //second normalization for robot parameters
        Norm(hbuff2,MIF*AssemblySize);
        //Subjunction bt1 and bt2
         for(n=0;n<NetSize;n++)
         {
                hbuff2[n]=hbuff[n]||hbuff2[n];
         }
        //third normalization
         Norm(hbuff2,AssemblySize);
        i=Xpos(hbuff2,nbuff);
        //fill 0's with the first active neuron
        if(i<AssemblySize)
        {
                for(n=i;n<AssemblySize;n++)
                        nbuff[n]=nbuff[0];
        }
        //save data for training
         if(fwrite(nbuff,4,AssemblySize,fp)!=AssemblySize)
        {
                this->label3->Text= L"Cannot wrote Assembly";
        }
}
        fclose(fg);
        fclose(decode);
        //fclose(fi);
        fclose(fp);
        fclose(fp2);
endp:
                 this->label3->Text= L"Files Coded";
```

Reconocimiento

```
        long unsigned i,j,k,m,n,p,q;
        long unsigned decox[AssemblySize]={0};
        long unsigned decoy[AssemblySize]={0};
        long unsigned decor[AssemblySize]={0};
        long unsigned nbuff[AssemblySize]={0};
        long unsigned hbuff3[AssemblySize*2]={0};
        long unsigned ActiveNumber2[AssemblySize]={0};
        long unsigned in=0;
        long unsigned xent=0;
        long unsigned Pos=0;
        unsigned mmsuma[ShiftNum]={0};
        long unsigned mimax=0,mimax2=0;
        unsigned decoded1[DataBase*3]={0};
        unsigned decoded[DataBase*3]={0};
        int rest[DataBase*3]={0};
        unsigned post=0;
        chart1->Visible=true;
        chart3->Visible=true;
        chart2->Visible=true;
        FILE *fp;
        FILE *fi;
        FILE *deco;
        FILE *fe;
        deco=fopen("decoded.dat","rb");
        if(fread(decoded1,4,DataBase*3,deco)!=DataBase*3)
                {
                        this->label2->Text= L"Cannot read deco.dat";
                        goto end1;
                }
        fclose(deco);
        fi=fopen("assembliesS.dat","rb");
        if(fi==NULL)
        {
                this->label2->Text= L"Cannot open assemblies.dat";
                goto end1;
        }
        fe=fopen("assemblies.dat","rb");
        if(fe==NULL)
        {
```

```
        this->label2->Text= L"Cannot open assemblies.dat";
        goto end1;
}
this->label2->Text=L"Recognition...";
this->label2->Refresh();
error=0;
this->progressBar1->Maximum=AssemblyNumber-1;
for(i=0;i<AssemblyNumber;i++)
{
        this->progressBar1->Value=i;
        this->label3->Text = System::String::Concat(L"i= ",i.ToString()) ;
        this->label3->Refresh();
        this->Show();
        if(fread(ActiveNumber,4,AssemblySize,fi)!=AssemblySize)
        {
                this->label2->Text= L"Cannot read assemblies.dat";
                goto end1;
        }
        if(fread(ActiveNumber2,4,AssemblySize,fe)!=AssemblySize)
        {
                this->label2->Text= L"Cannot read assemblies.dat";
                goto end1;
        }
        unsigned compb1[NetSize]={0};
        unsigned compb2[NetSize]={0};
        for(j=0;j<NetSize;j++)
        {
                compb1[j]=0;
                compb2[j]=0;
        }
        Setpos(ActiveNumber2,compb2,AssemblySize);
        //Setpos(
        long unsigned auxilio[AssemblySize]={0};
        for(j=0;j<AssemblySize;j++)
                auxilio[j]=ActiveNumber[j];

        for(j=0;j<NetSize;j++)
                buff[j]=0;
        in=0;
        for(j=0;j<AssemblySize;j++)
        {
                p=ActiveNumber[j];
                buff[p]=1;
                in++;
        }

        for(j=0;j<NetSize;j++)
                buff1[j]=buff[j];
        for(k=0;k<RecCycle;k++)
        {
                     Sleep(2);
                Recalculation();
        }

        for(j=0;j<NetSize;j++)
                if(buff1[j]!=0)
                        compb1[j]=1;

        long unsigned overl=0;
        overl=OL(compb1,compb2);
        if(overl<OverlapLevel)
                error++;
        this->label4->Text = System::String::Concat(L"error= ",error.ToString()) ;
        this->label4->Refresh();
        //Decoding
        unsigned working1[NetSize]={0};
        //read masks assemblies
        for(n=0;n<AssemblySize;n++)
                hbuff3[n]=0;
        in=0;
        for(n=0;n<NetSize;n++)
        {
                if(buff1[n]!=0)
                {
                        hbuff3[in]=n;
                        working1[n]=1;
                        in++;
                }
        }
        fp=fopen("assembliesmasks.dat","rb");
        if(fp==NULL)
        {
                this->label2->Text= L"Cannot read assembliesmasks1.dat";
        }
        if(fread(nbuff,4,AssemblySize,fp)!=AssemblySize)//mask for od
        {
                this->label2->Text= L"Cannot read assembliesmasks2.dat";
        }
        if(fread(nbuff,4,AssemblySize,fp)!=AssemblySize)//mask for oa
```

```
		{
			this->label2->Text= L"Cannot read assembliesmasks3.dat";
		}
		if(fread(nbuff,4,AssemblySize,fp)!=AssemblySize)//mask for or
		{
			this->label2->Text= L"Cannot read assembliesmasks4.dat";
		}
		if(fread(nbuff,4,AssemblySize,fp)!=AssemblySize)//mask for RX
		{
			this->label2->Text= L"Cannot read assembliesmasks5.dat";
		}
		long unsigned maximal=0;
		long unsigned maximal2=0;
		unsigned corr=0;
		long unsigned nwork[AssemblySize]={0};
		//Decoding RX
		maximal=0;
		maximal2=0;
		chart1->Series->Clear();
		chart1->Series->Add("Serie");
		chart1->Titles->Clear();
		chart1->Titles->Add("X Correlation");
		chart1->ChartAreas["ChartArea1"]->AxisY->Title="Overlap Level";
		chart1->ChartAreas["ChartArea1"]->AxisX->Title="XR Value";
		chart2->ChartAreas["ChartArea1"]->AxisY->Title="Overlap Level";
		chart2->ChartAreas["ChartArea1"]->AxisX->Title="YR Value";
		chart3->ChartAreas["ChartArea1"]->AxisY->Title="Overlap Level";
		chart3->ChartAreas["ChartArea1"]->AxisX->Title="RR Value";
		for(xent=0;xent<MaxValue;xent++)
		{
			unsigned working2[NetSize]={0};
			Perm(nbuff,nwork,xent,4);
			Setpos(nwork,working2,AssemblySize);
			corr=OL(working1,working2);
			if(corr>maximal)
			{
				maximal=corr;
				maximal2=xent;
			}
			chart1->Series["Serie"]->Points->AddXY((double)(xent+1),(double)(corr));
		}
		chart1->Series["Serie"]->IsVisibleInLegend=false;
		chart1->ChartAreas["ChartArea1"]->AxisY->Maximum= maximal+4;
		chart1->Titles->Add(System::String::Concat(L"Max=",maximal.ToString()));
		chart1->Update();
		chart1-
>SaveImage(System::String::Concat(i.ToString(),L"X.png"),System::Windows::Forms::DataVisualization::Charting::ChartI
mageFormat::Png);
		decox[i]=maximal2;

		if(fread(nbuff,4,AssemblySize,fp)!=AssemblySize)//mask for RY
		{
			this->label2->Text= L"Cannot read assembliesmasks5.dat";
		}
		//Permutation for RY//////////////////////////////////////
		//Decoding RY
		maximal=0;
		maximal2=0;
		chart2->Series->Clear();
		chart2->Series->Add("Serie");
		chart2->Titles->Clear();
		chart2->Titles->Add("Y Correlation");
		for(xent=0;xent<MaxValue;xent++)
		{

			unsigned working2[NetSize]={0};
			Perm(nbuff,nwork,xent,5);
			Setpos(nwork,working2,AssemblySize);
			corr=OL(working1,working2);
			if(corr>maximal)
			{
				maximal=corr;
				maximal2=xent;
			}
			chart2->Series["Serie"]->Points->AddXY((double)(xent+1),(double)(corr));
		}
		chart2->Series["Serie"]->IsVisibleInLegend=false;
		chart2->ChartAreas["ChartArea1"]->AxisY->Maximum= maximal+4;
		chart2->Titles->Add(System::String::Concat(L"Max=",maximal.ToString()));
		chart2->Update();
		chart2-
>SaveImage(System::String::Concat(i.ToString(),L"Y.png"),System::Windows::Forms::DataVisualization::Charting::ChartI
mageFormat::Png);
		decoy[i]=maximal2;
//decoding Rr/////////////////////////////////////////////////////////////
		if(fread(nbuff,4,AssemblySize,fp)!=AssemblySize)//mask for RR
		{
			this->label2->Text= L"Cannot read assembliesmasks5.dat";
		}
```

```
                //Permutation for RR/////////////////////////////////////
                maximal=0;
                maximal2=0;
                chart3->Series->Clear();
                chart3->Series->Add("Serie");
                chart3->Titles->Clear();
                chart3->Titles->Add("R Correlation");
                for(xent=0;xent<MaxValue;xent++)
                {
                        unsigned working2[NetSize]={0};
                        Perm(nbuff,nwork,xent,6);
                        Setpos(nwork,working2,AssemblySize);
                        corr=OL(working1,working2);
                        if(corr>maximal)
                        {
                                maximal=corr;
                                maximal2=xent;
                        }
                        chart3->Series["Serie"]->Points->AddXY((double)(xent+1),(double)(corr));
                }
                chart3->Series["Serie"]->IsVisibleInLegend=false;
                chart3->ChartAreas["ChartArea1"]->AxisY->Maximum= maximal+4;
                chart3->Titles->Add(System::String::Concat(L"Max=",maximal.ToString()));
                chart3->Update();
                chart3-
>SaveImage(System::String::Concat(i.ToString(),L"R.png"),System::Windows::Forms::DataVisualization::Charting::ChartI
mageFormat::Png);
                decor[i]=maximal2;

                decox[i];
                decoy[i];
                fclose(fp);
        }
        for(n=0;n<DataBase;n++)
        {
                decoded[3*n]=decox[n];
                decoded[3*n+1]=decoy[n];
                decoded[3*n+2]=decor[n];
        }
        float ecm=0;
        for(n=0;n<(DataBase*3-1);n++)
        {
                rest[n]=decoded1[n]-decoded[n];//*ShiftRange;
                ecm=ecm+rest[n]*rest[n];
                this->textBox3->Text=System::String::Concat(this->textBox3->Text,rest[n].ToString()) ;
                this->textBox3->Text=System::String::Concat(this->textBox3->Text,Char(13));
                this->textBox3->Text=System::String::Concat(this->textBox3->Text,Char(10));
        }
        ecm=ecm/(DataBase*3);
        ecm=sqrt(ecm);
        int numerr2=0;
        for(n=0;n<(DataBase*3-1);n++)
                if(abs(rest[n])>=ShiftRange)
                        numerr2++;
        this->textBox3->Text=System::String::Concat(this->textBox3->Text,L" error=  ");

        this->textBox3->Text=System::String::Concat(this->textBox3->Text,numerr2.ToString()) ;
        this->textBox3->Text=System::String::Concat(this->textBox3->Text,L" ecm=  ");
        this->textBox3->Text=System::String::Concat(this->textBox3->Text,ecm.ToString()) ;
        this->label2->Text= L"Recognition Finished";
        end1:
        fclose(fi);
        fclose(fe);
```