RESUMEN

El proyecto propuesto consiste en diseñar e implementar algoritmos basados en inteligencia artificial, e implementado en la plataforma Arduino, para robots que trabajen de manera colaborativa en un ambiente no controlado, de tal manera que exista una comunicación entre ellos y realicen un trabajo en conjunto.

El diseño de los algoritmos logrará hacer que el robot trabaje de forma autónoma, esquivando obstáculos que se puedan presentar en su trayecto. Este desplazamiento será enviado a los demás, al igual que posibles obstáculos que pueda encontrarse el robot principal, para que los robots siguientes ya tengan un conocimiento previo de la trayectoria que deberán seguir.

Así, el robot ira recolectando datos de su exterior, que le servirán para decidir hacia donde debe continuar su recorrido. El uso de algoritmos inteligentes permitirá a los robots tomar decisiones propias para no errar en su recorrido, y de acuerdo a su análisis del lugar donde se encuentre, y como se encuentre el entorno, decidirá hacia donde debe continuar.

De esta forma, los robots podrán trabajar de forma colaborativa, resolviendo el problema de la trayectoria que deban recorrer, evadiendo obstáculos y mandando esa información, e incluso sobre posibles sustancias o tóxicos que pueda encontrar en su camino el primer robot.

PALABRAS CLAVE

Algoritmo, Robot, Inteligencia Artificial, Trabajo colaborativo, Arduino, Neurona.

1. INTRODUCCIÓN

Actualmente, los robots se usan de manera extensa en la industria, siendo un elemento indispensable en una gran parte de los procesos de manufactura. Impulsados principalmente por el sector automotriz. Los robots han dejado de ser máquinas misteriosas propias de la ciencia ficción, para ser un elemento más de muchos de los talleres y líneas de producción.

Por su propia definición el robot industrial es multifuncional, puede ser aplicado a un sin número de funciones en la industria. No obstante, la práctica ha demostrado que su adaptación es óptima en determinados procesos en los que hoy en día, el robot es sin duda alguna la solución más rentable.

Junto a estas aplicaciones ya arraigadas, hay otras novedades que si bien la utilización del robot no se realiza a gran escala, si se justifica su aplicación por las condiciones intrínsecas del medio de trabajo (ambientes contaminados, salas asépticas, construcción, etc.) o la elevada exigencia en cuanto a calidad de los resultados (medicina, etc.).

Los robots son muy útiles en muchas áreas:

- Industria
- Laboratorios médicos
- Institutos de investigación
- Actividad espacial

Actualmente, se está produciendo un gran desarrollo en el campo de los robots cooperativos. La robótica colectiva busca diseñar sistemas compuestos de varios robots capaces de resolver problemas conjuntamente. La robótica en enjambre o colmena, es una rama de la inteligencia artificial en la que se trabaja desde hace años algunos de los más prestigiosos expertos del mundo y que trata de copiar el comportamiento colectivo observado en las colmenas o colonias de hormigas.

Bajo este principio, los robots forman parte de un sistema multi-robot, siendo simples en términos de diseño y control, y menos costosos que los sistemas de un solo robot especializado. Estos sistemas, están orientados a resolver problemas en los cuales la participación de un solo robot no es suficiente o resulta ser muy costosa en términos de diseño y tiempo, como por ejemplo, el transporte de un objeto voluminoso, material peligroso, exploración y cobertura de terreno.

Los proyectos de vida artificial robótica siempre han atraído a los investigadores, principalmente cuando intentan emular insectos o seres vivos como colmena. La robótica sirve para crear máquinas útiles, pero también criaturas artificiales que pueden ser o no parecidas a los seres vivos que nos rodean, y mejor si tienen sentido artístico o estético.

El comportamiento de colmena ha intrigado a los expertos durante años, y la robótica se ha convertido en una herramienta muy importante para explorarlo. Para ello, ha sido necesario entrar en una rama de las ciencias de la computación, denominada Inteligencia Artificial, que se ha desempeñado en hacer “máquinas inteligentes”.

El empleo de la inteligencia artificial, nos trae ventajas como solucionar errores y defectos propios del ser humano, es decir, el desarrollo de sistemas expertos que hoy en día se están utilizando con éxito en los campos de la medicina, geología y aeronáutica, entre otras ciencias.

Esta relación entre la robótica con la inteligencia artificial consiste en que la primera se encarga de diseñar y construir máquinas que mediante programas creados por la inteligencia artificial imiten el comportamiento y la comprensión humana, y sean capaces de realizar tareas reiterativas o peligrosas para el ser humano.

Así, nuestra meta está enfocada en conseguir un sistema no natural que resuelva problemas de la misma manera que resuelven los humanos de forma colaborativa. Si bien existen muchas técnicas que pueden utilizarse en sistemas basados en robots de trabajo colaborativo, podemos destacar 3 principalmente: Redes Neuronales Artificiales, Algoritmos genéticos y Lógica Difusa.

En conclusión, el comportamiento de robots con trabajo colaborativo es un área de mucho interés y aplicación para los investigadores que actualmente desarrollan sistemas inteligentes que realicen tareas complejas al trabajar cooperativamente.

1.1 JUSTIFICACIÓN

Este proyecto se llevó a cabo con la finalidad de desarrollar los algoritmos más adecuados, basados en métodos de inteligencia artificial, que mejor resolvieran la problemática que se tiene en un sistema comprendido por robots con trabajo colaborativo que deban alcanzar un objetivo predeterminado, evadiendo obstáculos que se presentaran en el recorrido del robot.

Si bien existen muchos métodos de programación que resuelven este tipo de problemas, se optó por el uso de Redes Neuronales Artificiales, no solo porque mejora la efectividad del sistema, sino que también mejora el rendimiento del mismo, al ser capaz de encontrar relaciones (patrones) de forma inductiva por medio de los algoritmos de aprendizaje basados en los datos existentes; es un sistema tolerante a fallos, pues cualquiera que ocurriese en alguna neurona de la red, no altera significativamente la respuesta total de la red; y al ser un sistema distribuido no lineal, permitirá la simulación de sistemas no lineales y caóticos, lo cual no era posible con sistemas clásicos lineales..

Además, se trabajó dentro de la plataforma Arduino, ya que brinda una completa flexibilidad en su uso, tanto a nivel físico, como a nivel software. Dicho entorno nos facilita el uso tanto de sensores, comunicación inalámbrica, e incluso la implementación de la red dentro de la misma plataforma, pues cuenta con un lenguaje de programación Open Source bastante amigable, sencillo de comprender, y lo mejor de este sistema, cuenta con microcontroladores AVR, los cuales superan por mucho a los comúnmente conocidos PIC de Microchip, tanto en memoria, como en puertos analógicos, seriales, etc.

1.2 OBJETIVO

Desarrollar e implementar algoritmos inteligentes basados en inteligencia artificial para la toma de decisiones en el movimiento del robot: avance, retroceso, izquierda, derecha y paro, a partir de señales provenientes de sensores ultrasónicos y biométricos.

1.3 OBJETIVOS ESPECÍFICOS

Estudiar, analizar y elegir el tipo de algoritmo que será el adecuado para implementarlo en el robot, basado en Redes Neuronales Artificiales, Lógica Difusa, o Algoritmos Genéticos.

Diseñar y simular distintos algoritmos inteligentes en Matlab, haciendo uso de las herramientas precargadas en el software, para crear el algoritmo que posteriormente se llevara al sistema de control.

Comprobar los algoritmos, eligiendo cuál sea el más adecuado para realizar la toma de decisiones del robot.

Implementar el algoritmo en un sistema basado en la plataforma Arduino para controlar al robot y la toma de decisiones, que tendrá la capacidad de evadir obstáculos en un entorno no controlado.

1.4 ALCANCES Y LIMITACIONES

El algoritmo diseñado para el robot tendrá la capacidad de decidir cuándo detenerse, avanzar, dar un giro a la derecha o izquierda, o retroceder, dependiendo del tipo de obstáculo que encuentre. Cada vez que encuentre un obstáculo, podrá enviar la posición GPS de dicho obstáculo, o sustancia toxica detectada en la trayectoria del robot.

El desplazamiento del robot será enviado de forma inalámbrica al robot siguiente, para que este sepa que ruta siguió el primer robot que ya evadió los obstáculos que encontró en su trayectoria. La toma del dato GPS se enviara siempre y cuando la red esté disponible, en caso de que la red no permita conocer dicha posición, solo se enviará los datos del recorrido que hizo el robot. Los robots podrán comunicarse en una distancia no máxima de 100 mts., pues los dispositivos utilizados para la comunicación inalámbrica no superan esa distancia, esta puede disminuirse dependiendo de la interferencia que pueda haber entre los robots. Debido al diseño de los robots, no podrá subir ni bajar escaleras, ni transitar terrenos no pavimentados, o con pendientes mayores a los 30°.

2. MARCO TEÓRICO

2.1 ROBOT

Un robot es una entidad virtual o mecánica artificial. En la práctica, esto es por lo general un sistema electromecánico que, por su apariencia o sus movimientos, ofrece la sensación de tener un propósito propio. Puede moverse, hacer funcionar un brazo mecánico (fig. 1), sentir y manipular su entorno, y mostrar un comportamiento inteligente, especialmente si ese comportamiento imita al de los humanos o a otros animales.

Fig. 1 Robot en industria Automotriz (automatizado)

Actualmente podría considerarse que un robot es una computadora con la capacidad y el propósito de movimiento que en general es capaz de desarrollar múltiples tareas de manera flexible según su programación; así que podría diferenciarse de algún electrodoméstico específico. Los robots comerciales e industriales, son ampliamente utilizados y realizan tareas de forma más exacta que un humano, e incluso resulta más barato.

También se les utiliza en trabajos demasiado sucios, peligrosos o tediosos para los humanos. Los robots son muy utilizados en plantas de manufactura, montaje y embalaje, en transporte, en exploraciones en la tierra y en el espacio, cirugía, armamento investigación en laboratorios y en la producción en masa de bienes industriales o de consumo.

Otras aplicaciones incluyen la limpieza de residuos tóxicos, minería, búsqueda y rescate de personas y localización de minas terrestres. Es de notar un gran incremento en su

aceptación como herramienta educativa y de investigación. Su valor didáctico experimental los ha llevado a ámbitos académicos, universitarios, escuelas de enseñanza, escuelas técnicas y hogares, e incluso en el espacio (fig. 2 y fig. 3).

Existe una gran variedad de robots y máquinas autónomas que simplifican el trabajo diario. Se los encuentran en formas variadas, desde cajeros automáticos, expendedoras de alimentos, lava-autos, hasta brazos robots de ensamblaje, pintura o soldadura, presentes en la industria electrónica y automotriz desde hace tiempo.

Fig. 2 Robot rover (NASA)

Si bien, es necesario que el robot sea capaz de percibir el entorno donde se encuentra, analizarlo y luego con estos datos realizar un plan de acción. Al aplicar este plan de acción también debe realimentarse de las variaciones que afecten el entorno y de manera recursiva hacer las correcciones necesarias hasta alcanzar su objetivo. Haciendo una analogía con los seres humanos, se le podría llamar “el sentido del robot”.

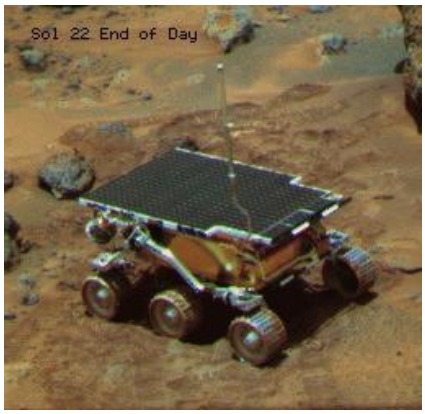


Fig. 3 Robot Pahfinder (NASA)

2.1.1 ROBOT DE TRABAJO COLABORATIVO

Actualmente en el campo de la robótica, se está produciendo un gran desarrollo en el campo de los robots colaborativos. Algunos problemas pueden llegar a ser demasiado difíciles para un único robot. La robótica colectiva busca diseñar sistemas compuestos de varios robots capaces de resolver problemas conjuntamente. Los robots que forman parte de un sistema multi-robot son simples en términos de diseño y control, y menos costosos que los sistemas de un solo robot especializado, obsérvese la figura 4.

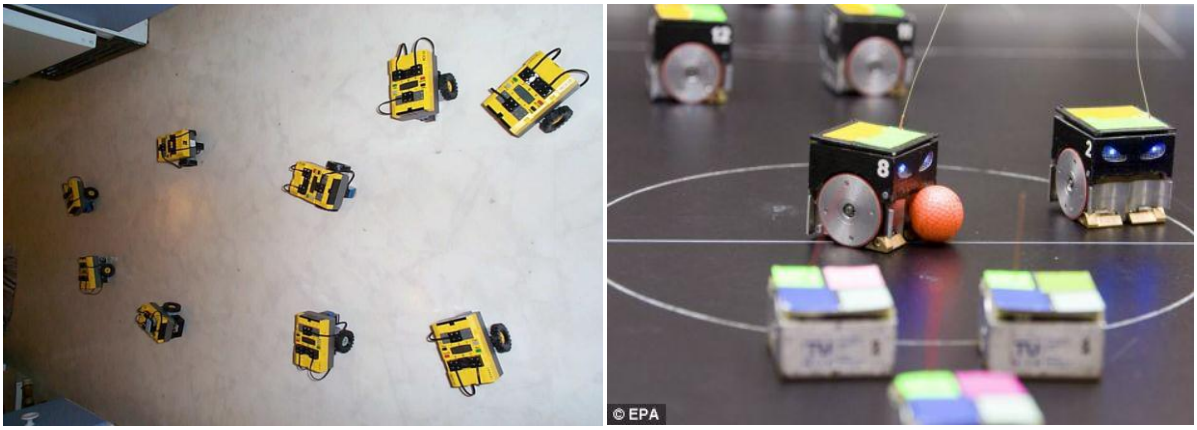


Fig. 4 Robots de trabajo colaborativo a) enjambre b) robots en juego de futbol

Los sistemas multi-robot están orientados a resolver problemas en los cuales la participación de un solo robot no es suficiente o resulta ser muy costosa, en términos de diseño y tiempo, como por ejemplo, el transporte de objetos voluminosos, el manejo de material peligroso, la exploración y cobertura de terreno, entre otras aplicaciones.

Los robots colaborativos son entidades virtuales o mecánicas que por distintos medios y protocolos de comunicación pueden intercambiar información entre ellos para actuar de manera conjunta en el loro de distintos objetivos. Esta comunicación e interacción permite un comportamiento parecido al que se presenta en cardúmenes de peces, enjambres o parvadas.

La forma de comunicarse entre robots colaborativos es comúnmente la comunicación inalámbrica o sin cables en la que extremos de la comunicación (emisor/receptor) no se encuentran unidos por un medio de conexión físico, sino que se utiliza la modulación de ondas electromagnéticas a través del espacio.

En este sentido, los dispositivos físicos de comunicación solo están presentes en los emisores y receptores de la señal, entre los cuales encontramos: antenas, computadores portátiles, PDA, teléfonos móviles, blootooth, wifi, infrarrojo, micrófonos inalámbricos, USB inalámbrico vía satélite, microondas, etc. Para cualquier robot móvil, la posibilidad de

navegar en su entorno es una de las capacidades más importantes de todas. Permanecer operacionales, es decir, evitar situaciones peligrosas, como colisiones y mantenerse dentro de las condiciones de operación segura: la temperatura, la radiación, la exposición a la intemperie, etc., pero si las tareas que se van a realizar se refieren a lugares específicos en el entorno del robot, la navegación es una necesidad.

- **Tipos de control**

CENTRALIZADO: tenemos varios robots que dependen de una unidad central. Los robots adquieren datos a través de sus sensores, y toda la información se la envían a la unidad central, que es la encargada de tomar las decisiones. La principal ventaja de este sistema, es que como la unidad central almacena mucha información, es más eficiente a la hora de obtener soluciones.

DISTRIBUIDO: cada robot decide que hacer por su cuenta, sin órdenes de ninguna entidad. Con este tipo de control el sistema es mucho más dinámico y rápido, pues no es necesario ni el envío ni el almacenamiento de información. Si uno de los robots llegase a averiarse, el sistema seguiría funcionando independientemente (fig. 5). Es difícil diseñar los algoritmos de programación, porque aunque son muy simples, deben de funcionar correctamente, es decir, es difícil encontrar un algoritmo sencillo que funcione, pero una vez encontrado, es muy fácil de aplicar e implementar.

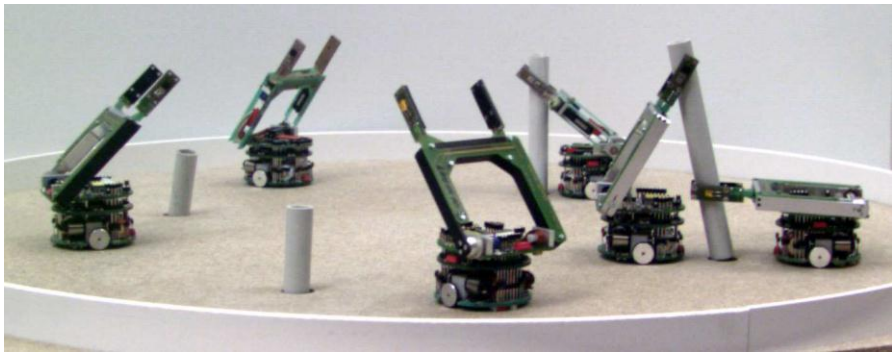


Fig. 5 Sistema distribuido de robots

- **Tipos de sistemas**

MUY ACOPLADOS (Colectivos): todos los individuos tienen su propia inteligencia pero además tienen la capacidad de comunicación, y por lo tanto, de cooperación y coordinación. De esta manera se tiene que la inteligencia colectiva es la suma de la inteligencia de cada individuo. Cada robot debe tener la capacidad de transferir información a otro compañero y tener la suficiente inteligencia para pedir ayuda y coordinar los movimientos con los demás individuos. Son sistemas pocos robustos, pues si falla u robot, falla todo el sistema.

POCO ACOPLADOS (manadas): son sistemas en que los robots funcionan independientemente del resto, sin tener en cuenta los movimientos o decisiones de los demás. No hay una inteligencia central sino que todo el control es distribuido, cada elemento simplemente sigue las reglas. Este tipo de comportamiento trata de imitar sistemas biológicos como el de los insectos sociales.

Este tipo de sistemas, tienen tolerancia a fallos, pues si se diera en uno de los robots, puede ser subsanado por el resto del equipo. Otra de las ventajas, se da en la rapidez y eficiencia que obtiene el sistema, pues si tuviésemos un solo robot, este único tendría que encargarse de todas las tareas, tomar decisiones, reconocer el medio, enviar la información, y controlar sus movimientos. Al dividir las tareas, el sistema se vuelve más eficiente en todos los aspectos. Dependiendo del tipo de programación y control de los robots que forman el sistema, la interferencia entre ellos puede llegar a ser un problema, como golpes, choques, distracciones.

En los últimos años, han surgido nuevas tecnologías, muchas de ellas tratando de imitar comportamientos que ya han sido desarrollados con gran efectividad por la naturaleza. El estudio de los insectos sociales, ejemplos naturales de sistemas descentralizados, es un punto de partida importante para entender cómo controlar sistemas complejos. Los insectos sociales son capaces de realizar tareas globales basándose en reglas simples y percepción local.

Los robots colaborativos son una alternativa superior para muchas aplicaciones. Los costes se pueden reducir, los tiempos de procesamiento y simplificar procesos. El uso de un sistema colaborativo constituye una solución altamente eficiente, particularmente para tareas difíciles. El desarrollo óptimo y la adaptación de esta tecnología requieren un buen proceso de conocimiento y una comprensión detallada de la aplicación.

2.2 INTELIGENCIA ARTIFICIAL

En la actualidad, la inteligencia artificial se ha consolidado como una disciplina científica contenida dentro de las ciencias de la computación, esta ha brindado significativos aportes a la humanidad moderna. Desde etapas muy tempranas de la humanidad, la inteligencia artificial ha ido evolucionando contribuyendo al surgimiento de varios métodos que dentro de esta rama, han creado sistemas inteligentes artificiales dotados de aprendizaje, y capaces de razonar.

La inteligencia artificial es considerada una rama de la computación y relaciona un fenómeno natural con una analogía artificial a través de programas de computador. La inteligencia artificial puede ser tomada como ciencia si se enfoca hacia la elaboración de

programas basados en comparaciones con la eficiencia del hombre, contribuyendo a un mayor entendimiento de conocimiento humano. Si bien por otro lado es tomada como ingeniería, basada en una relación de entrada-salida para sintetizar un programa de computador. El resultado es un programa de alta eficiencia que funciona como una poderosa herramienta para quien la utiliza.

A través de la inteligencia artificial se han desarrollado los sistemas expertos que pueden imitar la capacidad mental del hombre y relacionan reglas de sintaxis del lenguaje hablado y escrito sobre la base de la experiencia, para luego hacer juicios acerca de un problema, cuya solución se logra con mejores juicios y más rápidamente que el ser humano. En la medicina tiene gran utilidad al acertar el 85% de los casos de diagnóstico.

Si bien, la inteligencia artificial ha tenido que hacer frente a una serie de problemas:

- Los computadores no tienen autoconciencia
- Un computador solo puede hacer aquello para lo que está programado
- Las máquinas no pueden pensar realmente

La inteligencia artificial, comprende distintas ramas, entre las cuales destacamos las siguientes: Lógico-Matemática, Psicología, Informática y Simulación. A pesar de los avances que se obtiene en estas distintas ramas de la inteligencia artificial, científicos aseguran que una máquina de estados discretos, no puede emular al hombre, pues este no tiene un conjunto de reglas para regir su vida y las máquinas funcionan precisamente con reglas.

Para esto, cabe mencionar que la discretización no supone limitaciones, y las neuronas tiene un comportamiento rígido y de ellas surge comportamiento flexible inteligente, lo cual nos da la posibilidad de seguir creando máquinas “inteligentes”, que imiten el comportamiento humano, y lleguen a comprenderlo, desde una perspectiva más abierta.

La búsqueda de la imitación de las funciones lógicas mediante el empleo de ordenadores, ha llevado a la utilización de patrones de lógica clásica en la inteligencia artificial, creando sistemas artificiales de representación y recuperación del conocimiento. Existen distintos métodos empleados en estos sistemas inteligentes, como lo son las Redes Neuronales Artificiales, la Lógica Difusa, y los Algoritmos Genéticos.

Con la ayuda de estas técnicas de programación, es posible hacer que un robot perciba el exterior, reconozca formas, tenga visión, razone e inclusive hable. Estos comportamientos se han logrado gracias a los avances que han hecho los investigadores desde hace más de 40 años. Basándose en la naturaleza del pensamiento humano, podemos mencionar principalmente las Redes Neuronales Artificiales, que como su nombre lo dice, se basan del concepto que conocemos como neurona.

Existen varias categorías, las cuales son las siguientes:

- Sistemas que piensan como humanos: tratan de emular el pensamiento humano (RNA), como la toma de decisiones, resolución de problemas, etc.
- Sistemas que actúan como humanos: tratan de actuar como humanos, es decir, imitan el comportamiento humano (robótica).
- Sistemas que piensan racionalmente: con lógica, tratan de imitar o emular el pensamiento lógico racional del ser humano.
- Sistemas que actúan racionalmente: tratan de emular en forma racional el comportamiento humano, está relacionado con conductas inteligentes.

Si bien las técnicas que existen actualmente para crear máquinas inteligentes han dado pasos muy avanzados, para clasificar las máquinas como "pensantes", es necesario definir que es inteligencia y que grado de inteligencia implica resolver problemas matemáticos complejos, hacer generalizaciones o relaciones, percibir y comprender. Los estudios en las áreas del aprendizaje, del lenguaje y de la percepción sensorial han ayudado a los científicos a definir a una máquina inteligente.

Características de la inteligencia artificial

1. Una característica fundamental que distingue a los métodos de inteligencia artificial de los métodos numéricos es el uso de símbolos no matemáticos, aunque no es suficiente para distinguirlo completamente. Otros tipos de programas como los compiladores y sistemas de bases de datos, también procesan símbolos y no se considera que usen técnicas de inteligencia artificial.
2. El comportamiento de los programas no es descrito explícitamente por el algoritmo. La secuencia de pasos seguidos por el programa es influenciado por el problema particular presente. El programa especifica cómo encontrar la secuencia de pasos necesarios para resolver un problema dado.
3. Las conclusiones de un programa declarativo no son fijas y son determinadas parcialmente por las conclusiones intermedias alcanzadas durante las consideraciones al problema específico. Los lenguajes orientados al objeto comparten esta propiedad y se han caracterizado por su afinidad con la inteligencia artificial.
4. El razonamiento basado en el conocimiento, implica que estos programas incorporan factores y relaciones del mundo real y del ámbito del conocimiento en que ellos operan. Al contrario de los programas para propósito específico, como los de contabilidad y cálculos científicos; los programas de inteligencia artificial pueden distinguir entre el programa de razonamiento o motor de inferencia y base de conocimientos dándole la capacidad de explicar discrepancias entre ellas.

5. Aplicabilidad a datos y problemas mal estructurados, sin las técnicas de inteligencia artificial los programas no pueden trabajar con este tipo de problemas. Un ejemplo es la resolución de conflictos en tareas orientadas a metas como en planificación, o el diagnostico de tareas en un sistema del mundo real: con poca información, con una solución cercana y no necesariamente exacta.

Diferentes metodologías

1. La lógica difusa: permite tomar decisiones bajo condiciones de incerteza.
2. Redes Neuronales: esta tecnología es poderosa en ciertas tareas como la clasificación y el reconocimiento de patrones. Está basada en el concepto de "aprender" por agregación de un gran número de muy simples elementos.

Experiencia, habilidades y conocimiento

Los tipos de experiencia que son de interés en los sistemas basados en conocimiento, pueden ser clasificados en tres categorías: asociativa, motora y teórica. Los sistemas basados en conocimiento son excelentes para representar conocimiento asociativo. Este tipo de experiencia refleja la habilidad heurística o el conocimiento que es adquirido mayoritariamente, a través dela observación. Puede ser que no se comprenda exactamente lo que ocurre al interior de un sistema, pero se pueden asociar entradas o estímulos con salidas o respuestas, para resolver problemas que han sido previamente conocidos.

La experiencia motora es más física que cognitiva. La habilidad se adquiere fundamentalmente a través del ejercicio y la práctica física constante. Los sistemas basados en conocimiento no pueden emular fácilmente este tipo de experiencia, principalmente por la limitada capacidad de la tecnología robótica. La experiencia teórica y el conocimiento profundo permiten que los humanos puedan resolver problemas que no se han visto antes, es decir, no existe una posibilidad asociativa.

El conocimiento teórico y profundo se adquiere a través de estudio y entrenamiento formal, así como por medio de la resolución directa de problemas.

2.2.1 REDES NEURONALES ARTIFICIALES

Una red neuronal artificial es un procesador distribuido en paralelo de forma masiva que tiene una tendencia natural para almacenar conocimiento de forma experimental y lo hace disponible para su uso. Son capaces de encontrar relaciones (patrones) de forma inductiva por medio de los algoritmos de aprendizaje basado en los datos existentes más

que requerir la ayuda de un modelador para especificar la forma funcional y sus interacciones. Se parece al cerebro humano en dos aspectos:

- El conocimiento es adquirido por la red a través de un proceso de aprendizaje
- Los pesos sinápticos o fuerza con que están interconectadas las neuronas se utilizan para almacenar la información.

Otras definiciones son:

- Una nueva forma de computación, inspirada en modelos biológicos.
- Un modelo matemático compuesto por un gran número de elementos procesales organizados en niveles.
- Un sistema de computación hecho por un gran número de elementos simples, elementos de proceso interconectados, los cuales procesan información por medio de su estado dinámico como respuesta a entradas externas.
- Redes neuronales artificiales son redes interconectadas masivamente en paralelo de elementos simples (usualmente adaptativos) y con organización jerárquica, las cuales intentan interactuar con los objetos del mundo real del mismo modo que lo hace el sistema nervioso biológico.
- Sistema de procesado de la información que hacen uso de algunos de los principios que organizan la estructura del cerebro humano.

En casi todos los textos sobre redes neuronales se establece una analogía entre estos elementos y los componentes básicos de un ordenador. Las puertas de silicio. En órdenes de velocidad las neuronas son en varios órdenes de magnitud más lentas que las puertas lógicas de silicio. No obstante, el cerebro suple esta menor velocidad con un mayor número de interconexiones. También hay que destacar la eficiencia del cerebro desde un punto de vista energético; a pesar del gran número de operaciones realizadas el cerebro no necesita de un ventilador como las modernas CPU´s.

Esta capacidad de operar en paralelo le permite realizar tareas que necesitan una gran cantidad de cálculos y tiempo en potentes ordenadores. Un ejemplo cotidiano de esta característica es el reconocimiento de una cara en una fotografía; aunque se haya tomado mal y la persona esté un poco girada, la identificación de dicha persona no nos puede llevar mucho tiempo, sin embargo, este giro puede poner en un serio aprieto a un ordenador.

Dentro de estas redes, se encuentran las neuronas, que están interconectados entre sí mediante una serie de conexiones que, siguiendo con la analogía biológica, se conocen como pesos sinápticos. Estos pesos, varían con el tiempo mediante un proceso que se conoce como aprendizaje. Así pues podemos definir el aprendizaje de una red como el

proceso por el cual modifica las conexiones entre neuronas, pesos sinápticos, para realizar la tarea deseada.

Las neuronas, se agrupan en capas o niveles y poseen un alto grado de conectividad, que es ponderada por los pesos sinápticos, lo que sustenta su capacidad cognitiva. Es importante señalar que la propiedad más importante de las redes neuronales artificiales es su capacidad de aprender a partir de un conjunto de patrones de entrenamiento, es decir, es capaz de encontrar un modelo que ajuste los datos.

Analogía con el cerebro

La neurona es la unidad fundamental del sistema nervioso y en particular del cerebro. Cada neurona es una simple unidad procesadora que recibe y combina señales desde y hacia otras neuronas. Si la combinación de entradas es suficientemente fuerte la salida de la neurona se activa. El tamaño y forma de las neuronas es variable, pero todas tienen los mismos componentes; el soma o cuerpo de la célula, el axón y las dendritas; las conexiones entre las neuronas son materializadas por medio de la sinapsis. En la fig. 6 se muestra las partes que constituyen una neurona.

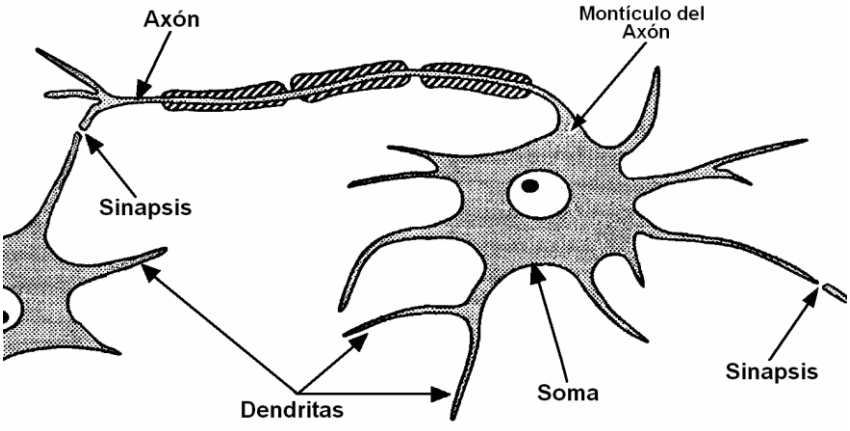


Fig. 6 Neurona biológica

El soma: es la porción central o cuerpo celular, que contiene el núcleo y tiene una o más estructuras denominadas dendritas o axones, estas son extensiones de la célula y están implicadas en la recepción y envió de las señales; en el soma se realiza la suma de todas las señales provenientes de otras neuronas.

El axón: es una fibra larga y delgada que lleva la señal desde su montículo hasta otras neuronas; generalmente existe un gran número de ramificaciones en el extremo del axón, para permitir que las señales de una neurona sean transmitidas a muchas otras.

Las dendritas: son extensiones de la célula, que se encargan de la recepción de señales provenientes de otras neuronas. Son un grupo de fibras nerviosas muy ramificadas y de forma irregular que están conectadas directamente al soma, cada neurona posee un gran número de dendritas que permiten recibir información de muchas otras neuronas. Las dendritas transmiten un potencial eléctrico hasta el soma, este potencial es de magnitud variable y puede ser excitador o inhibidor, según contribuya o no a la generación de un potencial de acción en el axón.

La sinapsis: la unión existente entre dos neuronas se denomina unión sináptica o sinapsis, esta unión consiste en un punto de contacto entre el axón de una neurona y la dendrita de otra neurona. En la unión sináptica se produce una conversión entre una señal eléctrica a una química. La señal eléctrica consiste en una diferencia de potencial entre el axón y el fluido extracelular, esta diferencia de potencial llamado potencial de acción, hace que la membrana presináptica, libere unas sustancias llamadas neurotransmisores.

Modelo de una Neurona

El modelo computacional de una neurona artificial es una imitación del proceso de una neurona biológica. En todo modelo artificial de neurona, se tienen cuatro elementos básicos, como se muestra en la **fig. 7**:

- Un conjunto de conexiones, pesos o sinapsis, que determinan el comportamiento de la neurona. Estas conexiones pueden ser excitadoras (presentan signo positivo) o inhibidoras (conexiones negativas).
- Un sumador que se encarga de sumar todas las entradas multiplicadas por las respectivas sinapsis (pesos).
- Una función de activación no lineal para limitar la amplitud de la salida de la neurona.
- Un umbral (bias) que determina el umbral por encima del cual la neurona se activa.

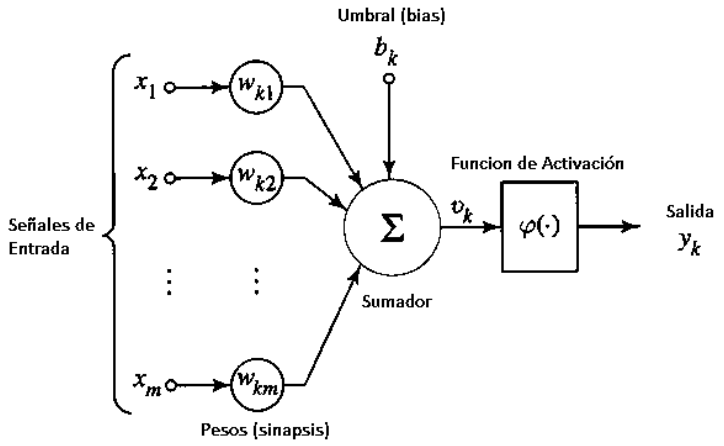


Fig.7 Modelo matemático de una neurona

Pesos sinápticos: constituyen bloques de ganancia aplicados a las señales de entrada. Al igual que en las células neuronales, estos pueden ser excitadores o inhibidores, dependiendo de su signo. Los valores de los pesos son asignados en la etapa de aprendizaje utilizado, algunos pesos sinápticos pueden tener un gran valor respecto a los demás y otros podrían anularse.

Función de activación: es aquella función asociada a cada neurona, que transforma la entrada total en la respuesta de la neurona. En cada neurona artificial, los productos de las entradas por sus pesos sinápticos, son sumados, esta suma constituye la entrada total a la neurona. La suma de las entradas ponderadas o entrada total, es transformada por la función de activación, para obtener la salida de la neurona. Esto puede representarse matemáticamente mediante la siguiente expresión:

$$salida = f(w_1x_1 + w_2x_2 + w_3x_3 + \cdots + w_nx_n)$$

En donde x_1, x_2, x_3 son las entradas, y w_1, w_2, w_3 son los pesos sinápticos de las entradas y $f()$ es la función de activación de la neurona.

Salida: la salida de cada neurona se obtiene de la función de activación, esa es análoga al axón de las celular neuronales. La salida de una neurona puede ser aplicada a una entrada de muchas otras neuronas o bien puede ser tomada como salida de la red.

Arquitecturas Neuronales

Los elementos básicos neuronales se pueden conectar entre sí para dar lugar a las estructuras neuronales o modelos conexionistas que podríamos clasificar de diferentes formas según el criterio usado. Estos se clasifican según su número de capas, su tipo de conexiones, o su grado de conexión.

- Redes Neuronales Monocapas (fig. 8): se corresponde con la red neuronal más sencilla, ya que se tiene una capa de neuronas que proyectan las entradas a una capa de neuronas de salida donde se realizan diferentes cálculos.

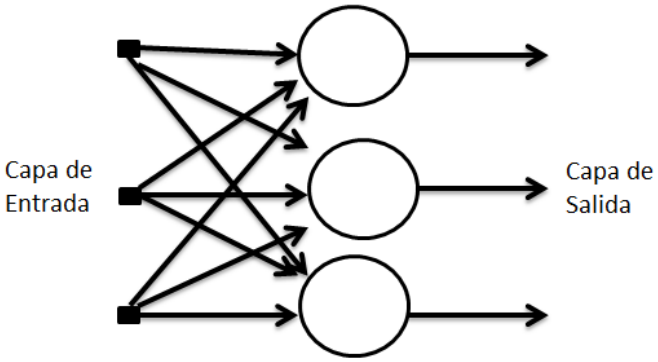


Fig. 8 Red Neuronal Monocapa

- Redes Neuronales Multicapa (fig. 9): es una generalización de la anterior, existiendo un conjunto de capas intermedias entre la entrada y la salida (capas ocultas). Este tipo de red puede estar total o parcialmente conectada.

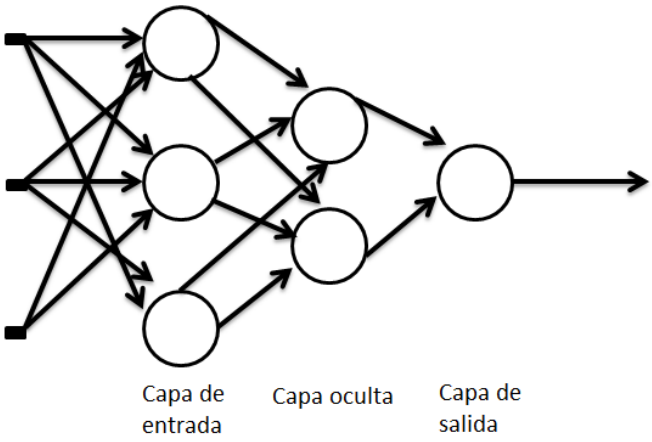


Fig. 9 Red Neuronal Multicapa

- Redes Neuronales No Recurrentes: en esta red la propagación de las señales se produce en un sentido solamente, no existiendo la posibilidad de realimentaciones.
- Redes Neuronales Recurrentes (fig. 10): esta red viene caracterizada por la existencia de lazos de realimentación. Estos lazos pueden ser entre neuronas de diferentes capas, neuronas de la misma capa, o entre una misma neurona. Esta estructura la hace especialmente adecuada para estudiar la dinámica de sistemas no lineales.

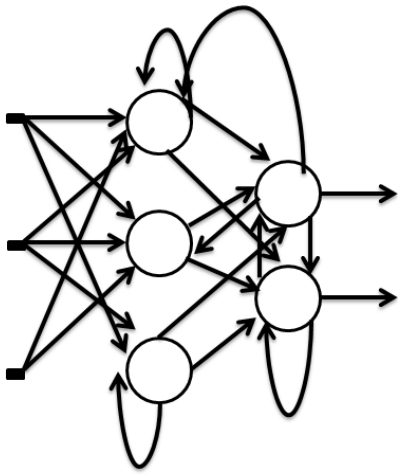


Fig. 10 Red Neuronal Recurrente

- Redes Neuronales Totalmente Conectadas: es este caso todas las neuronas de una capa se encuentran conectadas con las de la capa siguiente (redes no recurrentes) o con las de la anterior (redes recurrentes).
- Redes Neuronales Parcialmente Conectadas: en este caso no se da la conexión total entre neuronas de diferentes capas.

Tipos de aprendizaje

El proceso de aprendizaje de la red, puede ser supervisado o no supervisado. El aprendizaje supervisado consiste en entrenar la red a partir de un conjunto de datos o patrones de entrenamiento compuesto por patrones de entrada y salida. El objetivo del algoritmo de aprendizaje es ajustar los pesos de la red $\underline{w}$ de tal manera que la salida generada por la RNA sea lo más cercanamente posible a la verdadera salida dada una cierta entrada.

Es decir, la red neuronal trata de encontrar un modelo al proceso desconocido que género la salida. Este aprendizaje se conoce como supervisado pues se conoce el patrón de salida, el cual hace el papel de supervisor de la red. En cambio, en el aprendizaje no supervisado, se presenta solo un conjunto de patrones a la RNA, y el objetivo del algoritmo de aprendizaje es ajustar los pesos de la red de manera tal que la red encuentre alguna estructura o configuración presente en los datos.

2.2.1.1 PERCEPTRÓN

La primera red neuronal conocida, desarrollada en 1943 por el neurofisiólogo Warren McCulloch y el matemático Walter Pitts, esta consistía en la suma de las señales de entrada, multiplicadas por unos valores de pesos escogidos aleatoriamente. La entrada total era comparada con un valor preestablecido para determinar la salida de la red. Si en la comparación, la suma de las entradas multiplicadas por los pesos era mayor o igual que el valor preestablecido, la salida de la red era uno; en caso contrario la salida era cero.

La arquitectura del perceptrón, llamado mapeo de patrones, aprende a clasificar modelos mediante un aprendizaje supervisado. Los modelos que clasifica suelen ser generalmente vectores con valores binarios (0, 1) y las categorías de la clasificación se expresan mediante vectores binarios. El perceptrón presenta dos capas de unidades procesadoras, y solo una de ellas presenta la capacidad de adaptar o modificar los pesos de las conexiones.

La arquitectura del perceptrón simple admite capas adicionales pero éstas no disponen la de capacidad de modificar sus propias conexiones. La **figura 11** muestra la unidad procesadora básica del perceptrón. Las entradas llegan por la parte izquierda y cada conexión con la neurona j tiene asignada un peso de valor w_{ji}.

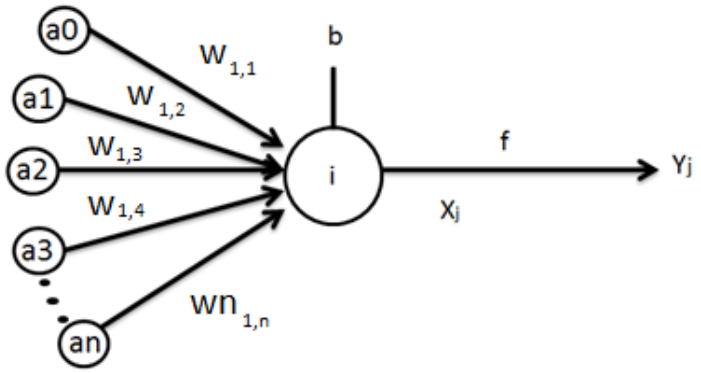


Fig. 11 Perceptrón

La unidad procesadora del perceptrón realiza la suma ponderada de las entradas según la ecuación siguiente:

$$y_j = f(\sum w_{ij}\, a_i + b)$$

Un aspecto común en muchas de la RNA es la entrada especial llamada “bias” representada en la parte superior de la figura 7.2.1.1. Esta entrada siempre presenta un valor fijo, y funciona como una masa en un circuito eléctrico donde no varía de valor (se puede utilizar como un valor constante de referencia).

El perceptrón comprueba si la suma de las entradas ponderadas es mayor o menor que un cierto valor umbral y genera la salida "y_j" según la siguiente ecuación:

$$\text{Si } x_j > 0 \; entonces \; y_j = 1$$

$$x_j \leq 0 \; entonces \; y_j = 0$$

La salida y_j es transmitida a lo largo de la línea de salida y constituye uno de los componentes del vector de salida de la red. Las redes perceptrón de dos capas, tienen una capa de entrada y una capa de salida de unidades procesadoras que constituyen la capa de salida (**fig. 12**).

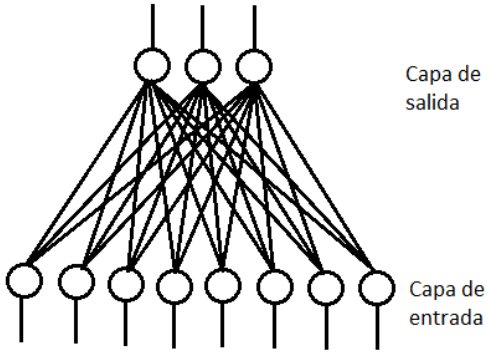


Fig. 12 Red perceptrón de dos capas

Las topologías con tres o más capas se caracterizan porque la regla de aprendizaje del perceptrón solo adapta los pesos o valores de las conexiones de una capa. Una aplicación típica de un sistema de tres capas es la que muestra la **figura 13** donde la entrada es la imagen de la letra E y la salida es la caracterización de la entrada en dos clases.

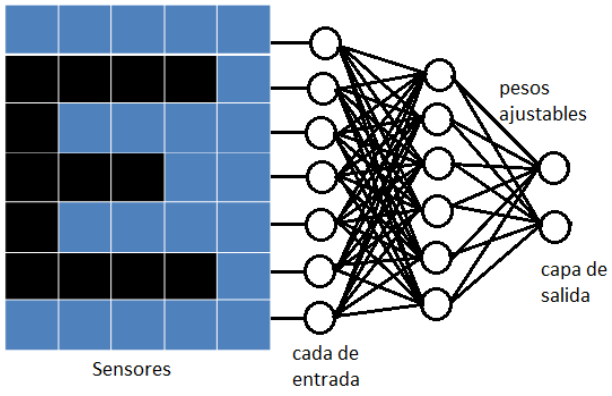


Figura 13 Red perceptrón de 3 capas

El entrenamiento del perceptrón consiste en presentar a la red todos los elementos del conjunto de entrenamiento constituido por parejas de vectores (entrada y salida deseada) de forma secuencial. El objetivo del entrenamiento es llegar a un conjunto de valores de

los pesos de la red de forma que responda correctamente a todo el conjunto de entrenamiento. Después del entrenamiento los pesos no son ya modificados y la red está ya en disposición de responder adecuadamente a las entradas que se le presente. La adaptación de los pesos se puede realizar mediante diferentes reglas. Una de las reglas más simples de aprendizaje del perceptrón se indica en la ecuación siguiente:

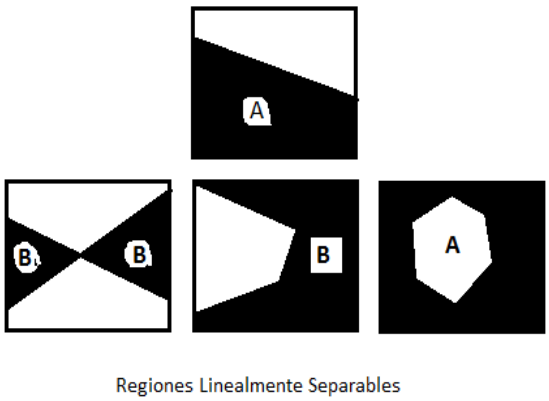
$$w_{ji\ nuevo} = w_{ij\ viejo} + C(t_j * y_j)a_i$$

Siendo t_j el valor de la salida deseada, y_j el valor de salida producida por la unidad procesadora, a_i el valor de la entrada i y C el coeficiente de aprendizaje. En todo proceso de entrenamiento el comportamiento de la red inicialmente va mejorando hasta que llega a un punto en el que se estabiliza y se dice que la red ha convergido. Esta convergencia tiene dos posibilidades, la primera consiste en que la red haya aprendido correctamente el conjunto de entrenamiento o la segunda se trata de que la red no haya aprendido todas las respuestas correctas.

- **Separación lineal**

El mayor inconveniente del Perceptrón, a pesar del éxito que ha tenido en muchas aplicaciones de clasificación de patrones es la imposibilidad de adaptar los pesos de todas las capas. En los años en los que se realizó el perceptrón, los investigadores no fueron capaces de diseñar un algoritmo que propagara las correcciones de los pesos a través de redes multicapa.

La principal limitación funcional de perceptrón es que una unidad de salida solo puede clasificar patrones linealmente separables. La **figura 14** muestra el concepto general de separabilidad lineal, es decir, las clases de patrones que pueden separarse en dos clases mediante una línea. Este concepto se puede extender a tres o más dimensiones, simplemente separando dos clases mediante planos e hiperplanos.



Regiones Linealmente Separables

Fig. 14 Separabilidad lineal

Afortunadamente, para la computación neuronal surgieron nuevas reglas de aprendizaje para redes multicapa y nuevas arquitecturas, entre ellas la más popular Back-Propagation, que resolvieron entre otros los problemas de clasificación de patrones no separables linealmente.

2.2.1.2 BACKPROPAGATION

La invención del algoritmo BackPropagation ha desempeñado un papel vital en el surgimiento del interés de las redes neuronales artificiales. Backpropagation es un método de entrenamiento de redes multicapa. Su potencia reside en su capacidad de entrenar capas ocultas y de este modo supera las posibilidades restringidas de las redes de una única capa.

- **Arquitectura de la red Backpropagation**

La unidad procesadora básica de la red Backpropagation se representa en la fig. 15**.** Las entradas se muestran a la izquierda, y a la derecha se encuentran unidades que reciben la salida de la unidad procesadora situada en el centro de la figura.

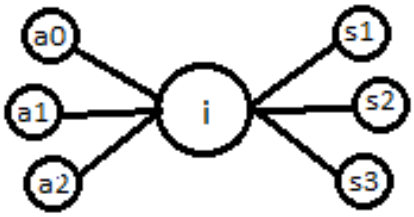


Fig. 15 Backpropagation

Normalmente, la Backpropagation utiliza tres o más capas de neuronas. La figura 16 muestra una topología Backpropagation típica de tres capas. La capa inferior es la capa de entrada, y se caracteriza por ser la única capa cuyas neuronas reciben entradas desde el exterior. Sirven como puntos distribuidores, no realizan ninguna operación de cálculo.

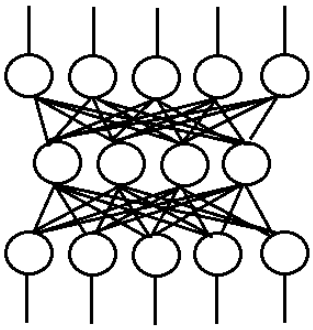


Fig. 16 Red Backpropagation arquitectura 5 4 5

Las neuronas de las demás capas procesan las señales como se indica en la figura 16. La siguiente capa superior es la capa oculta, y todas sus neuronas están interconectadas con la capa inferior y con la capa superior. La capa superior es la capa de salida, que presenta la respuesta de la red.

- **Algoritmo de entrenamiento**

Las redes Backpropagation tienen un método de entrenamiento supervisado. A la red se le presenta parejas de patrones, un patrón de entrada emparejado con un patrón de salida deseada. Por cada presentación los pesos son ajustados de forma que disminuya el error entre la salida deseada y la respuesta de la red. El algoritmo de aprendizaje Backpropagation conlleva una fase de propagación hacia adelante y otra fase de propagación hacia atrás. Ambas fases se realizan por cada patrón presentado en la sesión de entrenamiento.

- Propagación hacia adelante

Esta fase se inicia cuando se presenta un patrón en la capa de entrada de la red. Cada unidad de la entrada se corresponde con un elemento del vector patrón de entrada. Las unidades de entrada toman un valor de su correspondiente elemento del patrón de entrada y se calcula el valor de activación o nivel de salida de la primera capa. A continuación las demás capas realizarán la fase de propagación hacia adelante que determina el nivel de activación de las otras capas. La neurona obtiene la cantidad S_j según la siguiente ecuación:

$$S_j = \sum i\ ai\ wji$$

Y genera la salida o nivel de activación siguiente:

$$\text{Salida} = f(S_j)$$

La función de transferencia f es una función Sigmoidea, y también destaca la función hiperbólica. El valor de salida de la neurona i es enviado o transmitido a lo largo de todas las conexiones de salida de dicha unidad. En la **figura 17** se muestra la fase de propagación hacia adelante.

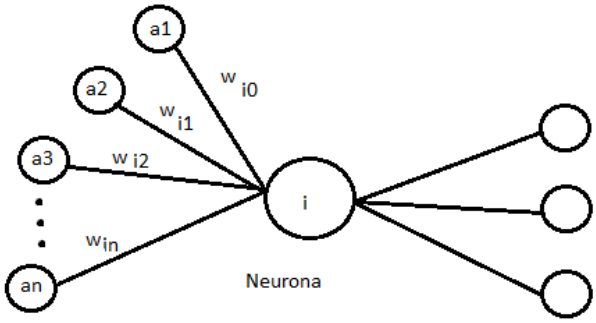


Fig. 17 Propagación hacia adelante

Por otro lado, algunas redes Backpropagation utilizan unidades llamadas bias como parte de cualquiera de las capas ocultas y de la capa de salida. Estas unidades representan constantemente un nivel de activación de valor 1. Además esta unidad está conectada a todas las unidades de la capa inmediatamente superior y los pesos asociados a dichas conexiones son ajustables en el proceso de entrenamiento. La utilización de esta unidad tiene un doble objetivo, mejorar las propiedades de convergencia de la red y ofrecer un nuevo efecto umbral sobre la unidad que opera.

- Propagación hacia atrás

Una vez completada la fase de propagación hacia adelante, se inicia la fase de corrección o fase de propagación hacia atrás. Los cálculos de las modificaciones de todos los pesos de las conexiones empiezan por la capa de salida y continua hacia atrás a través de todas las capas de la red hasta la capa de entrada. Dentro de los tipos de ajuste de pesos de puede clasificar dos grupos, ajuste de neuronas de la capa de salida y ajuste de neuronas de las capas ocultas.

Ajuste de pesos de la capa de salida*:* el ajuste de pesos es sencillo, debido a que existe y se conoce el valor deseado para cada una de las neuronas de la capa de salida. Cada unidad de la capa de salida produce un número real como salida y se compara con el valor deseado especificado en el patrón del conjunto de entrenamiento. A partir del resultado de la comparación se calcula un valor de error, y para cada neurona de la capa de salida.

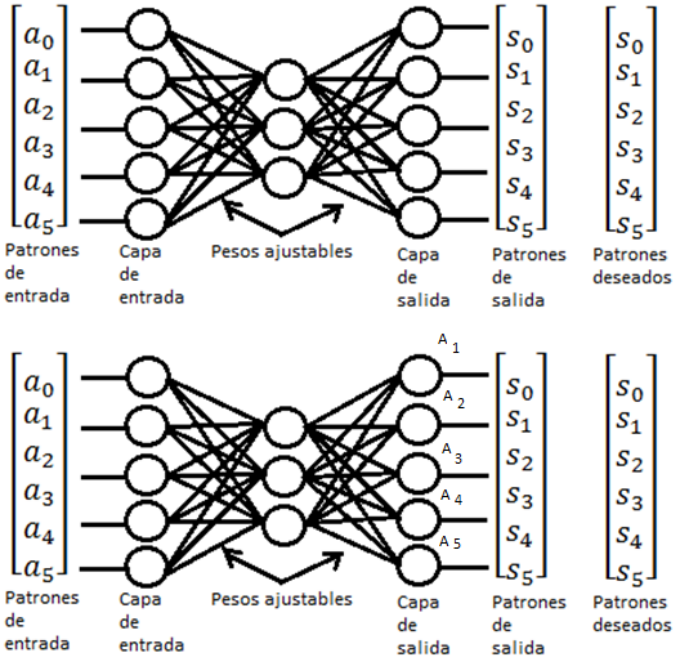


Fig. 18 Pesos la capa de entrada y salida

***Ajuste de los pesos de las capas ocultas**:* estas capas no tienen un vector de salidas deseadas, y por tanto no se puede seguir el método de propagación de error mencionado en el caso de las neuronas de la capa de salida. El valor de error calculado para este tipo de neuronas se obtiene de la ecuación siguiente, y la **figura 18** representa la obtención a partir de la misma:

$$\delta_j = [\Sigma_k \delta_k \, w_{kj}] \, f\_(S_j)$$

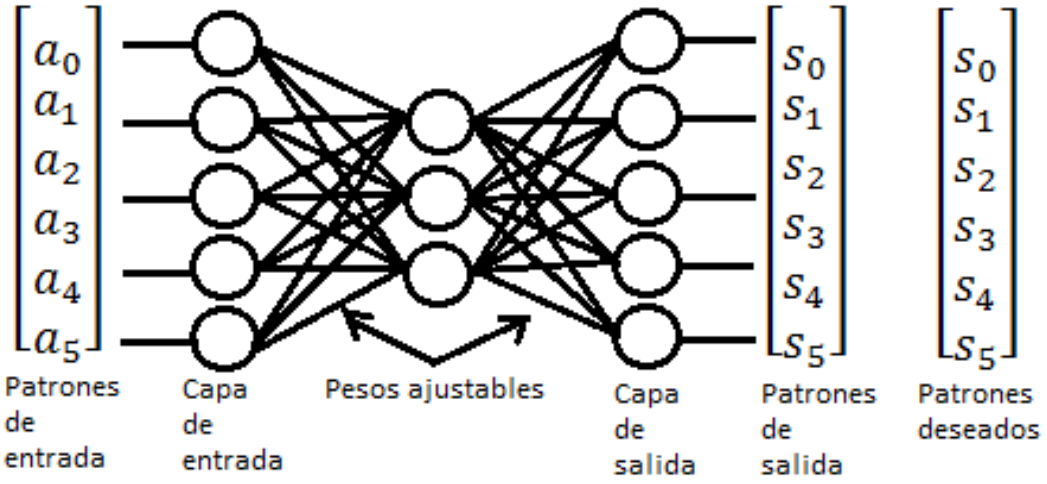


Fig. 19 Cálculos de los valores de las capas ocultas

Convergencia: en el proceso de entrenamiento o aprendizaje de la Backpropagation es frecuente medir cuantitativamente el aprendizaje mediante el valor RMS (Root Mean Square) del error de la red. Esta media refleja el modo en el que la red está logrando respuestas correctas; a medida que la red aprende, su valor RMS decrece.

Debido a que los valores de salida de la red y los valores de salidas deseadas son valores reales, es necesario definir un parámetro de corte o un valor umbral del valor RMS del error de la red que permita decir que la red se aproxima a la salida deseada y considerar que la respuesta es correcta.

La convergencia es un proceso en el que el valor RMS del error de la red tiende cada vez más al valor 0. La convergencia no siempre es fácil de conseguirla porque a veces el proceso puede requerir un tiempo excesivo o bien porque la red alcanza un mínimo local y deja de aprender.

2.3 ARDUINO

Es una plataforma de electrónica abierta para la creación de prototipos basada en software y hardware flexibles y fáciles de usar. Se creó para artistas, diseñadores, aficionados y cualquiera interesado en crear entornos u objetos interactivos. Arduino puede tomar información del entorno a través de sus pines de entrada de toda una gama de sensores y puede afectar aquello que le rodea controlando luces, motores y otros actuadores.

El microcontrolador en la placa Arduino se programa mediante el lenguaje de programación Arduino (basado en Wiring) y el entorno de desarrollo Arduino (basado en processing). En la **figura 20** puede observarse físicamente la placa Arduino Duemilanove.

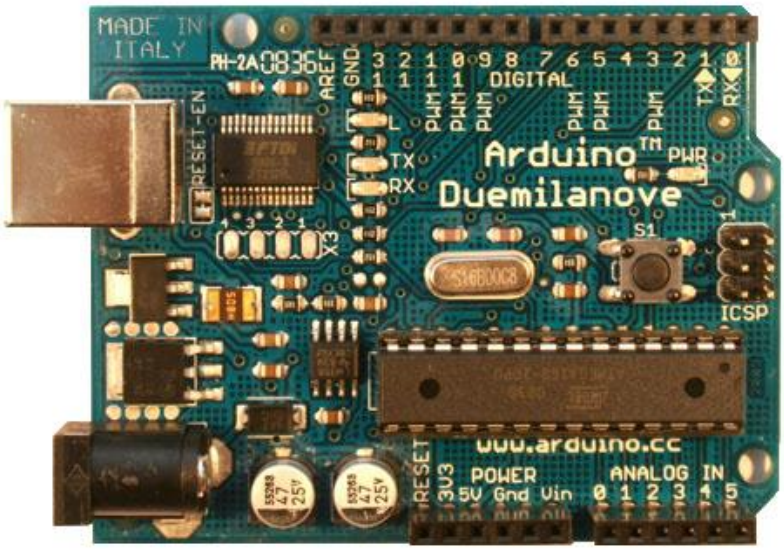


Fig. 20 Arduino Duemilanove

Los proyectos hechos con Arduino pueden ejecutarse sin necesidad de conectar a un ordenador, si bien tienen la posibilidad de hacerlo y comunicar con diferentes tipos de software (flash, processing., MaxMSP, etc). El software de Arduino consiste en un entorno de desarrollo (IDE) y las bibliotecas de núcleo.

El IDE está escrito en Java y basado en el entorno de desarrollo Processing las bibliotecas están escritas en C y C++ y compilado usando AVR-GCC y librería AVR. El código fuente para Arduino está alojado en GITHUB.

2.3.1 CONCEPTOS BÁSICOS

Arduino consiste en una placa con un microcontrolador Atmel AVR y puertos de entrada/salida. Los microcontroladores más usados son el Atmega168, Atmega328, Atmega1280, Atmega8 por su sencillez y bajo costo. Cuenta con un oscilador de 16Mhz, conexión USB, una cabecera ISCP y un botón de reset. El software consiste en un entorno de desarrollo que implementa el lenguaje de programación Processing/Writing y el cargador de arranque (bootloader) que corre en la placa.

Arduino se puede utilizar para desarrollar objetos interactivos autónomos o puede ser conectado a software del ordenador (por ejemplo: Macromedia Flash, Processing, Max/MSP, Pure Data.), simplemente se conecta al ordenador vía USB. Las placas se pueden montar a mano o adquirirse. El entorno de desarrollo integrado libre se puede descargar gratuitamente.

Al ser open-hardware, tanto su diseño como su distribución, son libres. Es decir, puede utilizarse libremente para el desarrollo de cualquier tipo de proyecto sin haber adquirido ninguna licencia.

- Entradas y salidas

Como podemos observar en la tabla 1, consta de 14 entradas digitales configurables como entrada o salida, que operan a 5V. Cada pin puede proporcionar o recibir como máximo 40mA. Los pines 3, 5, 6, 8, 10 y 11 pueden proporcionar una salida PWM (modulación por ancho de pulso). Si se conecta cualquier dispositivo a los pines 0 y 1, eso interferirá con la comunicación USB. También cuenta con 6 entradas analógicas que proporcionan una resolución de 10 bits. Por defecto miden de 0V hasta 5V, aunque es posible cambiar el nivel más alto, utilizando el pin Aref, y un código de bajo nivel.

Tabla 1 Especificaciones

	Atmega168	Atmega328	Atmega1280
Voltaje operativo	5V	5V	5V
Voltaje recomendado	7-12V	7-12V	7-12V
Pines In/Out	14 (6 PWM)	14 (6 PWM)	54 (14 PWM)
Pines Analógicos	6	6	16
Intensidad de corriente	40mA	40mA	40mA
Memoria Flash	16KB	32KB	128KB
EEPROM	512 bytes	1KB	4KB

2.3.2 LENGUAJE DE PROGRAMACIÓN

El lenguaje Arduino está basado en C/C++ y soporta todas las construcciones de C estándar y algunas funcionalidades de C++. Vincula la librería AVR Libc y permite el uso de todas sus funciones. Es una implementación de WIring, una plataforma de computación física parecida, que a su vez se basa en Processing, un entorno de programación multimedia. Arduino está basado en C y soporta todas las funciones del estándar C y algunas de C++. A continuación se muestra un resumen con todas las estructuras del lenguaje Arduino.

Estructuras de control:
- setup(): inicialización
- loop(): bucle
- if, if...else, switch case
- for, while, do... while
- break, continue, return, goto

Sintaxis
- Delimitadores: ;, {}
- Comentarios: //, /* */
- Operadores aritméticos: +, -, *, /, %
- Asignación: =
- Comparación: ==, !=, <, >, <=, >=
- Booleanos: &&, ||, !
- Acceso a punteros: *, &
- Bits: &, |, ^, ~, <<, >>
- Cabeceras: #define, #include
- Incremento/decremento: ++, --
- Asignación y operación: +=, -=, *=, /=, &=, |=

Variables

Constantes
- HIGH/LOW
- INPUT/OUTPUT
- True/false

Tipos de datos
- void, boolean, char, unsigned char, byte, int, unsigned int, word, long, unsigned long, float, double, string, array

Conversion entre tipos
- char(), byte(), int(), word(), long(), float()

Comunicación por puerto serie
- begin(), available(), read(), flush(), print(), println(), write()

Interrupciones
interrupts(), noInterrupts()

Funciones

E/S Digital
- pinMode(pin, modo)
- digitalWrite(pin, valor)
- int digitalRead(pin)

E/S Analógica
- analogReference(tipo)
- int analogRead(pin)
- analogWrite(pin, valor)

E/S Avanzada
- shiftOut(dataPin, clockPin, bitOrder, valor)
- unsigned long pulseIn(pin, valor)

Tiempo
- unsigned long millis()
- unsigned long micros()
- delay(ms)
- delayMicroseconds(microsegundos)

Matemáticas
- min(x, y), max(x, y), abs(x), constrain(x, a, b), map(valor, fromLow, fromHigh, toLow, toHigh), pow(base, exponente), sqrt(x)

Trigonometría
- sin(rad), cos(rad), tan(rad)

Números aleatorios
- randomSeed(semilla), long random(máx), long random(mín, máx)

Bits y Bytes
- lowByte(), highByte(), bitRead(), bitWrite(), bitSet(), bitClear(), bit()

Interrupciones externas
- attachInterrupt(interrupción, función, modo)
- detachInterrupt(interrupción)

- **Entorno**

El entorno de desarrollo Arduino está constituido por un editor de texto para escribir el código, un área de mensajes, una consola de texto, una barra de herramientas con botones para las funciones comunes y una serie de menús. Permite la conexión con el hardware de Arduino para cargar los programas y comunicarse con ellos. Arduino utiliza para escribir el software lo que denomina "Sketch (programa)" (fig. 21). Estos programas son escritos en el editor de texto. Existe la posibilidad de cortar/pegar y buscar/reemplazar texto. En el área de mensajes se muestra información mientras se cargan los programas y también muestra errores. La consola muestra el texto de salida para el entorno Arduino incluyendo los mensajes de error completos y otras informaciones. La barra de herramientas permite verificar el proceso de carga, creación, apertura y guardado de programas, y la monitorización serie:

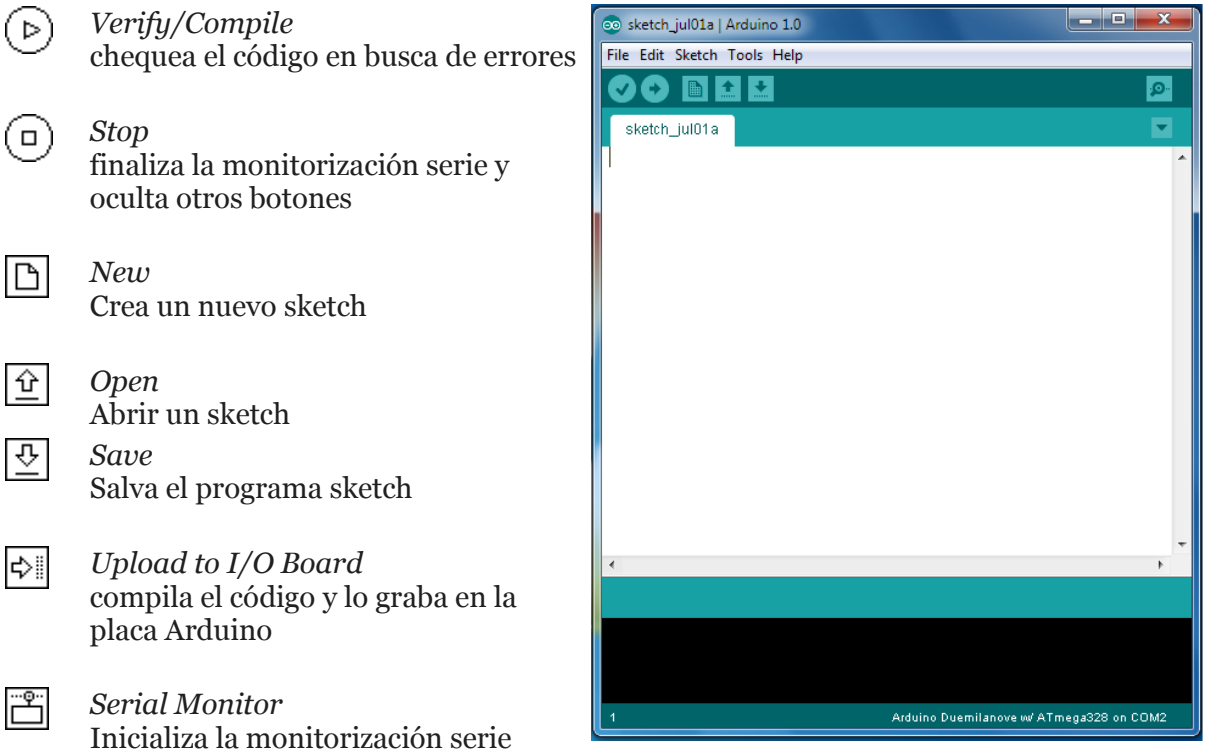


Fig. 21 Entorno de Arduino

2.3.3 LIBRERÍAS Y COMUNICACIÓN

Las librerías proporcionan funcionalidad extra para la utilización en “sketches”, por ejemplo para trabajar hardware o manipular datos. Para utilizar una librería “sketch”, seleccione Sketch>Import Library. Esto insertará una o más sentencias #include al principio del “sketch” y compilara la librería con su “sketch”. Debido a que las librerías se vuelcan a la placa junto con su ”sketch”, incrementan la ocupación del espacio disponible. Si un “sketch” no precisa de una librería, simplemente borra su sentencia #include en la parte inicial del código.

Algunas librerías están incluidas en el software Arduino, otras pueden ser descargadas desde una gran variedad de fuentes.

- **Librerías**

EEPROM – para leer y escribir en memorias “permanentes”.
Ethernet – para conectar a internet usando el Ethernet Shield.
Firmata – Para comunicarse con aplicaciones en la computadora usando un protocolo estándar serial.
LiquidCrystal – para controlar Displays de cristal líquido (LCD).
Servo – para controlar servomotores.
SoftwareSerial – para la comunicación serial de cualquier pin digital.
Stepper – para controlar motores paso a paso.
Wire – interfaz de dos cables, o Two Wire Interface (TWI, I2C), para enviar y recibir datos a través de una red de dispositivos y sensores.

- **Comunicación**

Arduino facilita en varios aspectos la comunicación con el ordenador, otro Arduino, u otros microcontroladores. Tanto el Atmega328 como el Atmega168 proporcionan comunicación vía serie UART TTL (5V), disponible a través de los pines digitales 0 (TX) y 1 (RX). Un chip FTDI FT232 integrado en la placa canaliza esta comunicación serie a través del USB y los drivers FTDI proporcionan un puerto serie virtual en el ordenador.

El software incluye un monitor de puerto serie que permite enviar y recibir información textual de la placa Arduino. La librería SoftwareSerial permite comunicación serie con otros dispositivos. Tanto el Atmega168 y Atmega328 también soportan la comunicación I2C (TWI) y SPI. El software de Arduino incluye una librería Wire para simplificar el uso del bus I2C, y para el uso de la comunicación SPI.

2.4 MATLAB

Matlab es una abreviatura de la frase Matrix Laboratory. Es un entorno informático de análisis numérico y representación gráfica de fácil manejo. Originalmente fue escrito para la enseñanza de algebra lineal, aunque actualmente es, al mismo tiempo, un entorno y un lenguaje de programación.

También permite crear funciones propias y programas especiales (denominados archivos .m) en código Matlab, que se pueden agrupar en las llamadas Toolboxes: colección especializada de archivos .M para trabajar en distintos tipos de problemas, por ejemplo de optimización, de estadística, de ecuaciones diferenciales parciales, etc.

Se puede considerar a Matlab, como una plataforma para el desarrollo de aplicaciones y para la resolución de problemas en múltiples áreas de aplicación.

Entre sus utilidades, encontraremos:

- Calculo matricial y algebra lineal
- Polinomios e interpolación
- Regresión y ajuste de funciones
- Ecuaciones diferenciales ordinarias
- Integración
- Funciones y gráficos en dos y tres dimensiones

El elemento básico de Matlab es una matriz rectangular de elementos reales o complejos. Matlab incluso considera los escalares como matrices. Además todas las variables utilizadas en línea de comandos son almacenadas por Matlab.

- Archivos .M

Se puede colocar órdenes en un simple archivo de texto (o ascii) y a continuación, hacer que Matlab lo abra y evalué las órdenes exactamente como si hubiesen sido escritas desde la línea de comandos. Estos archivos se llaman archivos script o archivos –M, y deben finalizar con la expresión ".m".

Para crear un archivo –M se escoge New del menú File, y seleccionamos M-file. Una vez guardado este archivo-M, Matlab ejecutara las órdenes en dicho archivo simplemente escribiendo su nombre (sin extensión) en la línea de comandos. Normalmente, las órdenes leídas desde el archivo-M no se visualizan cuando se evalúan. La orden "echo on" le dice a Matlab que visualice o efectué un eco de las órdenes en la ventana de orden cuando se leen y evalúan. También existe la función "echo off".

2.4.1 TOOLBOX NEURAL NETWORK

El Neural Network Toolbox de Matlab modela solo abstracciones simples de los modelos de redes biológicas. Comúnmente se entrenan por aprendizaje supervisado, aunque también soporta el aprendizaje no supervisado y el diseño directo. Contiene una colección de funciones construidas predefinidas en un ambiente numérico de cómputo de Matlab.

Estas funciones predefinidas pueden ser llamadas por el usuario para simular diferentes tipos de modelos neuronales. Neural Network es una herramienta útil en la industria, educación e investigación. Como herramienta para el desarrollo y entrenamiento de redes neuronales más populares bajo el ambiente de Matlab, se encuentran las redes:

- Perceptrón
- Adaline
- Backpropagation
- Redes de base radial
- Som
- Elman
- Hopfield
- Lvq

Dentro de las herramientas de Matlab, podremos encontrar las 2 redes más comunes para el trabajo de diseño de redes neuronales, las cuales se explican a continuación:

2.4.1.1 PERCEPTRÓN

Dentro de las herramientas que contiene precargado el software Matlab, encontramos el toolbox Neural Network, el cual integra comandos y funciones que nos ayudan a crear módelos de redes neuronales biológicas, y llevarlos a la simulación. Uno de ellos es el perceptron, cuya arquitectura es muy sencilla, pues estamos hablando de una neurona simple, como se muestra en la **fig. 22** (a), el cual se introduce dentro de Matlab, y crea su

modelo virtual, como se muestra en la figura 7.4.1 (b).

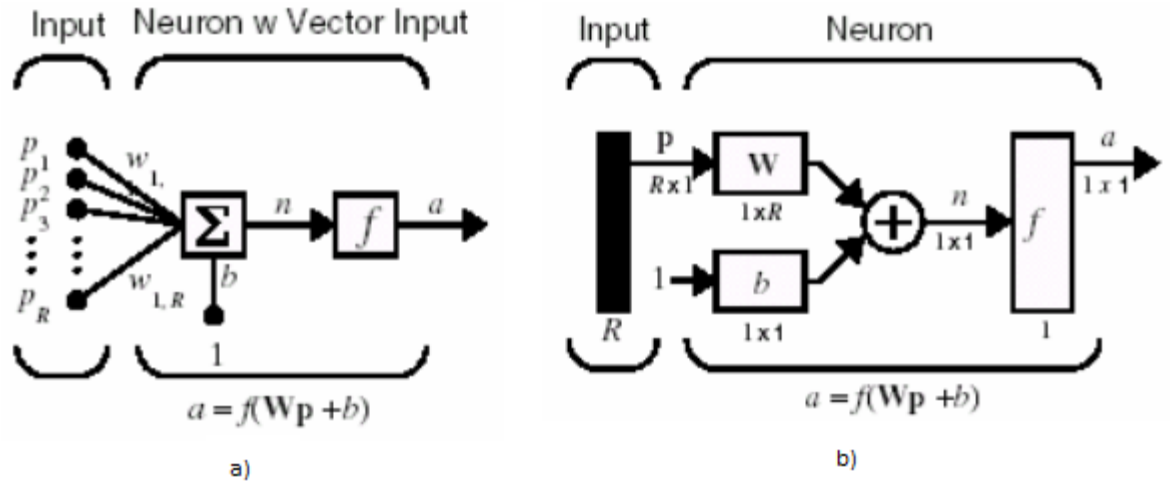


Fig. 22 Arquitectura del Perceptrón (modelo matemático y modelado en Matlab)

Una neurona Perceptrón, utiliza una función de transferencia *hardlim*, cada entrada es ponderada con un peso "w_{ij}" y la suma de las entradas ponderadas se envían a dicha función de transferencia, la cual devuelve un 0 o un 1. La función de transferencia, da a un perceptrón la capacidad de clasificar vectores de entrada, al dividir el espacio de entrada en dos regiones en el plano, **figura 23**.

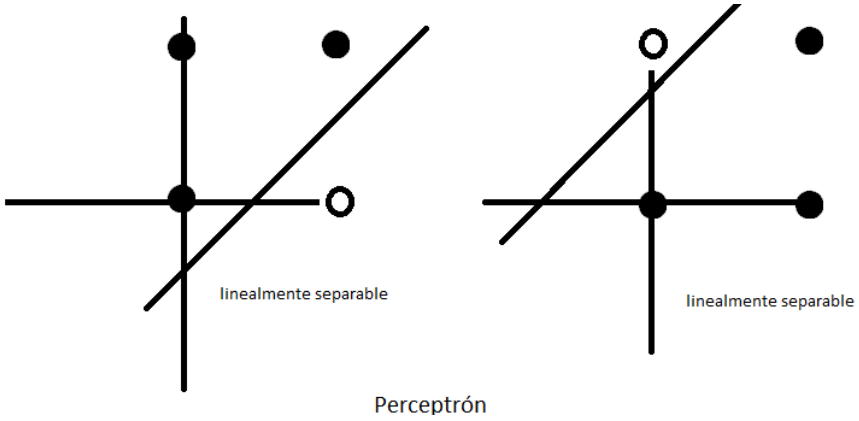


Fig. 23 Separabilidad lineal

- Creación de un Perceptrón

Un Perceptrón, se puede crear con la función newp.

Net=newp(P,T);

Donde los argumentos de entrada son los siguientes:

*P es una matriz de R por Q, Q vectores de entrada, R elementos de cada uno.
*T es una matriz de S por Q, Q vectores objetivo, S elementos de cada uno.

Los comandos siguientes crean una red perceptrón con un vector de entrada única, de un elemento con los valores 0 y 2, y una neurona con salidas que pueden ser 0 o 1.

P=[0 2];
T=[0 1];
net=newp(p,t);

Se puede ver los pesos que la red ha creado ejecutando el siguiente comando:

Inputweights=net.inputweights{1,1}

La función de transferencia por defecto es *hardlim,* y la función de aprendizaje es *learnp,* que genera el producto de la matriz de vector de entradas y el peso, y añade el bias para calcular la entrada neta. La inicialización por defecto es la función *initzero* y se utiliza para establecer los valores iniciales de los pesos a cero. Similarmente

Biases=net.biases{1}

Se puede observar que la inicialización del bias es también 0. La regla de aprendizaje *learnp* calcula los cambios deseados a los pesos del perceptrón y los bias, dado que un vector de entrada *p* y el error asociado *e*. El vector de salidas deseadas *t* debe contener valores de 0 o 1, pues el perceptrón solo puede tener como salida esos 2 valores.

Cada vez que se ejecuta *learnp* el perceptrón tiene una mejor oportunidad de producir las salidas correctas. La regla de aprendizaje ha demostrado que convergen en una solución en un número finito de iteraciones, si existe una solución. Si el bias no se utiliza, *learnp* trabaja para encontrar una solución mediante la alternación de solo el vector de pesos *w* para apuntar hacia vectores de entrada para ser clasificados como 1 y lejos de vectores para ser clasificados como 0. Esto resulta en una frontera de decisión que es perpendicular a w y que clasifica adecuadamente los vectores de salida.

Hay tres condiciones que pueden ocurrir de una sola neurona una vez que un vector de entrada p se presenta y la respuesta de la red uno se calcula:

Caso 1. Si un vector de entrada se presenta y la salida de la neurona es correcta (a = t y e = t – a = 0), entonces el vector de pesos w no se altera.

Caso 2. De la neurona de salida es 0 y debería haber sido 1 (a = 0 y t = 1, y e = t – a = 1), el vector de entrada p se añade al peso vector w. esto hace que el punto de vector de peso más cerca de la vector de entrada, aumentando la posibilidad de que el vector de entrada se clasifica como un 1 en el futuro.

Caso 3. Si la salida es 1 neurona y debería haber sido 0 (a = 1 y t = 0, y e = t - a = -1), el vector de entrada p se resta del peso vector w. esto hace que el punto de vector de peso más lejos del vector de entrada, aumentando la posibilidad de que el vector de entrada se clasifica como un 0 en el futuro.

La regla de aprendizaje perceptrón puede escribirse en términos del error e = t-a y el cambio que deben introducirse en el vector de peso dw.

Caso 1. Si E = 0, luego hacer un cambio w iguales.
Caso 2. Si e = 1, el hacer un cambio igual a w p ^ t.
Caso 3. Si e = -1, el hacer un cambio igual a w-p ^ t.

Para un ejemplo sencillo, se debe comenzar con una única neurona que tiene un vector de entradas con solo dos elementos, al igual que sus posibles salidas:

```
Net=newp([-2 2;-2 2],0 1[]);
```

Para simplificar las cosas, se establece el bias igual a 0, y los pesos en 1 y 0.8:

```
Net.b{1}=[0];
w=[1 0.8];
net.iw(1,1)=w;
```

El par de entrada de destino está dada por:

```
P=[1;2];
t=[1];
```

Se puede calcular la salida y el error con:

```
A=sim(net,p)
e=t-A;
```

Y utilizar la función *learnp* para encontrar el cambio de los pesos:

```
Dw=learnp(w,p,[],[],[],[],e,[],[],[] )
```

Y los nuevos pesos se obtienen como:

$$W=w+wd$$

$$w=2\ 1.2$$

El proceso de encontrar nuevos pesos y bias se puede repetir hasta que no haya errores. Ay que recordar que la regla de aprendizaje del perceptrón está garantizada a converger en un número finito de pasos a seguir para todos los problemas que pueden ser resueltos por un perceptrón. Éstos incluyen todos los problemas de clasificación que son linealmente separables. Los objetos para ser clasificados en tales casos pueden ser separados por una sola línea.

- Training

La función de "train" lleva a cabo como un bucle de cálculo. en cada paso la función de "entrenar" el producto a través de la secuencia especificada de insumos, el cálculo de la salida, el error, y el ajuste de red para cada vector de entrada en la secuencia que las entradas se presentan.

Hay que tener en cuenta que entrenar no garantiza que la red resultante hace su trabajo. Se debe comprobar los nuevos valores de la "w" y "b" mediante el cálculo de la salida de red para cada vector de entrada, para ver si todos los objetivos se alcanzan. Si una red no funciona correctamente se puede entrenar aún más llamando al "train" de nuevo con los nuevos pesos y bias de las tarjetas de entrenamiento más, o se puede analizar el problema para ver si se trata de un problema apropiado para el perceptrón.

Para ilustrar el procedimiento de formación, trabajar a través de un simple problema. Considerar un perceptrón una neurona con un vector de entrada única que tiene dos elementos: esta red, y el problema que está a punto a considerar, son bastante simples que usted puede seguir a través de lo que se hace con los cálculos a mano, si quieres.

Utilizar los pesos iniciales y las bias. Denotan las variables en cada paso de este cálculo mediante el uso de un número entre paréntesis después es la variable. por lo tanto, más arriba, los valores iniciales son W (0) y B (0).

$$W(0)= [0\ \ 0]\ \ b(0)= 0$$

El siguiente código define un perceptrón con los pesos iniciales y los valores de polarización de 0.

```
net = newp ([-2 2 ; -2 2] , [0 1])
```

Considerar la aplicación de una sola entrada

p = [2 ; 2];

Teniendo el objetivo

t = [0];

Establece las épocas a 1, por lo que el tren pasa a través de los vectores de entrada (sólo una aquí) sólo una vez.

net.trainparam.epochs=1;
net=train(net,p,t);

Los nuevos pesos y los prejuicios son

w= net.iw{1,1}, b=net.b{1}
w= -2 -2
b= -1

Por lo tanto, los pesos iniciales y las bias son 0, y después de entrenar sólo en el primer vector, tienen los valores de [-2 -2] y -1. Ahora aplicamos el segundo vector de entrada "P2". Ahora entrenar a la red con los nuevos pesos y los prejuicios son finalmente, simular la red entrenada para cada una de las entradas.

2.4.1.2 BACKPROPAGATION

La red neuronal feedforward multicapa es el caballo de batalla de la Caja de herramientas de software de neural network. Se puede utilizar para el montaje y función tanto problemas de reconocimiento de patrones. Con la adición de una línea de retardo con tomas, también puede ser utilizado para los problemas de predicción. El flujo de trabajo para el proceso general de diseño de la red neuronal tiene siete pasos principales:

Recopilación de datos
Creación dela red
Configuración dela red
Inicializar los pesos y sesgos
Capacitar a la red
Validarla red (después del entrenamiento de análisis)
Utilice la red

- **Modelo neuronal (logsig, tansig, purelin)**

Una neurona primaria con R entradas se muestra en la **figura 24**. Cada entrada es ponderada con un w apropiado. La suma de las entradas ponderadas y el sesgo de la forma la entrada a la función f transferencia. Las neuronas se pueden utilizar cualquier función de transferencia f diferenciable para generar su producción.

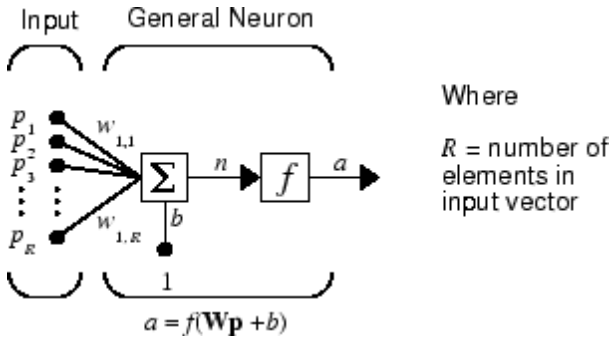
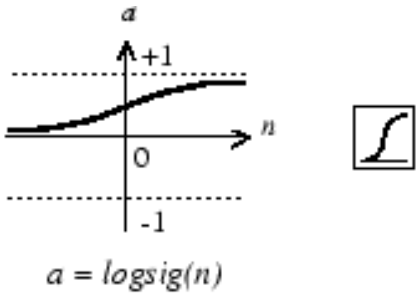


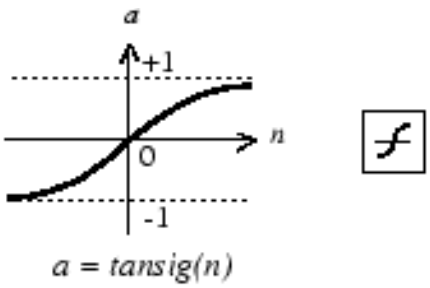
Fig. 24 Arquitectura neuronal

Redes multicapa pausan a menudo la transferencia logsiglog-sigmoi de función.



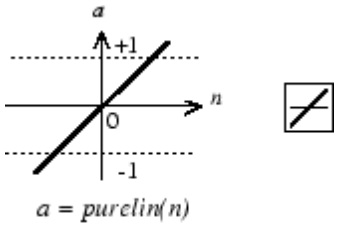
Log-Sigmoid Transfer Function

La función logsig genera salidas entre 0 y 1 como entrada neta de la neurona va de negativo a infinito positivo. Por otra parte, las redes de múltiples capas pueden utilizar la transferencia tansig de función Tangente-sigmoidal.



Tan-Sigmoid Transfer Function

Las neuronas de salida sigmoidal se utilizan a menudo para los problemas de reconocimiento de patrones, mientras que las neuronas de salida lineales se utilizan para problemas de la función de ajuste. La función de transferencia lineal purelin, se muestra a continuación.



$a = purelin(n)$

Linear Transfer Function

Las tres funciones de transferencia descritas aquí son las funciones de transferencia más comúnmente utilizados para las redes multicapa, pero otras funciones de transferencia diferenciables se pueden crear y utilizar si se desea.

Recoger y preparar los datos

Antes de comenzar el proceso de diseño de la red, primero recolectar y preparar los datos de la muestra. Después de que los datos han sido recogidos, hay dos pasos que deben llevarse a cabo antes de que los datos se utilizan para entrenar a la red: los datos deben ser pre-procesados, y tienen que ser divididos en subgrupos.

Pesos de inicialización (init)

Antes de la formación de una red feedforward, debe inicializar los pesos y bias. El comando configura automáticamente, inicializa los pesos, pero es posible que desee para reiniciar. Esto se hace con el comando init. Esta función toma un objeto de red como entrada y devuelve un objeto de red con todos los pesos y bias inicializados. Así es como una red se inicializa:

net =init (neto);

Algoritmos de entrenamiento

Una lista de los algoritmos de formación que están disponibles en el software de red neuronal de herramientas, y que utilizan gradiente o métodos basados en jacobiana, se muestra en la tabla 2.

Tabla 2. Algoritmos de entrenamiento

Function	Algorithm
trainlm	Levenberg-Marquardt
trainbr	Bayesian Regularization
trainbfg	BFGS Quasi-Newton
trainrp	Resilient Backpropagation
trainscg	Scaled Conjugate Gradient
traincgb	Conjugate Gradient with Powell/Beale Restarts
traincgf	Fletcher-Powell Conjugate Gradient
traincgp	Polak-Ribiére Conjugate Gradient
trainoss	One Step Secant
traingdx	Variable Learning Rate Gradient Descent
traingdm	Gradient Descent with Momentum
traingd	Gradient Descent

La función más rápida generalmente es trainlm, y es la función de formación predeterminada para feedforwardnet. El método cuasi-Newton, trainbfg, también es bastante rápido. Ambos de estos métodos tienden a ser menos eficaces para redes grandes (con miles de pesos), ya que requieren más memoria y más tiempo de cálculo para estos casos. Además, trainlm tiene un mejor rendimiento sobre el ajuste de la función (regresión lineal) problemas de reconocimiento de patrones.

Tabla 3 Parámetros de entrenamiento

Parameter	Stopping Criteria
min_grad	Minimum Gradient Magnitude
max_fail	Maximum Number of Validation Increases
time	Maximum Training Time
goal	Minimum Performance Value
epochs	Maximum Number of Training Epochs (Iterations)

La capacitación se detendrá si se pulsa el botón de parada de formación en la ventana de la formación. Es posible que desee hacer esto si la función de rendimiento no disminuye significativamente a lo largo de muchas iteraciones. Desde la ventana de entrenamiento,

puede acceder a cuatro parcelas: el rendimiento, la formación del estado, histograma de error y de regresión.

2.5 SENSOR SRF02

El SRF02 es un medidor ultrasónico de distancia, es decir, un sonar de pequeño tamaño y que funciona mediante un solo transductor (fig. 25). Entre las características más destacadas de este transductor se encuentra sin duda el hecho de utilizar un único transductor tanto para transmitir la ráfaga ultrasónica como para recibir el eco de la misma y poder así medir la distancia. Esta característica hace que la distancia mínima medida sea mayor que la de otros medidores del mercado que utilizan para ello dos transductores diferentes.

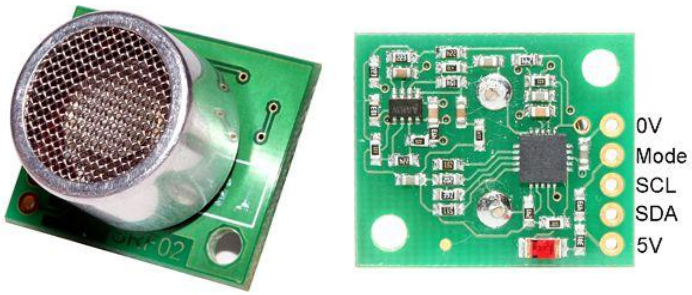


Fig. 25 SRF02

El sensor de distancia por ultrasonidos SRF02 es un nuevo sensor de pequeño tamaño y mínimo consumo que destaca por tener interfaz serie e interfaz I2C. El interfaz serie tiene un formato estándar de 9600 baudios, un bit de comienzo, ocho datos y un bit de parada. El nivel de tensión es a nivel TTL lo que permite conectarlo a cualquier microcontrolador del mercado. En ambos modos el rango de medidas es de 15 cm a 600 cm. Cada sensor tiene su propia dirección interna, aunque esta se puede cambiar de forma que se pueden tener hasta 16 módulos SRF02 en el mismo bus, ya sea serie o I2C. Las medidas pueden ser en centímetros, pulgadas o microsegundos. La alimentación es de 5V y el consumo medio de 4mA.

- **Modo I2C**

Para seleccionar el modo I2C debe dejar sin conectar el pin Modo del SRF02. El pin SDA corresponde a la señal de datos y el pin SCL a la señal de reloj, como se muestra en la figura 7.5.1. Ambas señales se deben polarizar a 5Vcc a través de dos resistencias de polarización positiva que normalmente se encuentran en el circuito maestro del bus I2C que controla los dispositivos I2C esclavos. La dirección I2C del medidor SRF02 por defecto es 0xE0, pero se puede elegir cualquiera de las otras 16 siguientes para conectar otros

sensores: 0xE0, 0xE2, 0xE4, 0xE6, 0xE8, 0xEA, 0xECC, 0xEE, 0xF0, 0xF4, 0xF6, 0xF8, 0xFA, 0xFC y 0xFE.

El SRF02 está compuesto por un juego e 6 registros:

Registros Nº	Modo de lectura	Modo de Escritura
0	Revisión de software interno	Registros de comandos
1	No usado (se lee 0x80)	No disponible
2	Byte alto de la medidad realizada	No disponible
3	Byte bajo de la medidad realizada	No disponible
4	Byte alto del valor mínimo de distancia	No disponible
5	Byte bajo del valor mínimo de distancia	No disponible

El único registro que se puede escribir es el 0, ya que este es el que se utiliza para empezar un nuevo cálculo. Las medidas tardan 65ms en llevarse a cabo y mientras se realizan no responden a ninguna otra operación que se realice mediante el bus I2C. Los registros 2 y 3 son el resultado de la última medida en un valor de 16 bits medidos en pulgadas, centímetros o microsegundos según el comando que se haya utilizado. Un valor 0 indica que el sensor no ha detectado ningún objeto. Los registros 4 y 5 son el resultado del valor aproximado de la distancia mínima que el sonar puede medir en un valor de 16 bits.

- **Comandos**

Los 3 primeros comandos de la **tabla 4**, del 80 al 82, se utilizan para iniciar una nueva medición en pulgadas, centímetros o microsegundos. Los otros 3 siguientes, del 86 al 88, son parecidos pero no transmiten ninguna ráfaga por lo que no miden la distancia a ningún objeto. Estos comandos le pueden servir para detectar las ráfagas de otros medidores ultrasónicos. El comando 96 reinicia el SRF02 realizando un ciclo de autoajuste. Es como si se conectara la alimentación. Los últimos 160, 165 y 170 son los encargados de cambiar la dirección I2C de los SRF02.

Tabla 4 Comandos

Comandos		Descripción
Decimal	Hexadecimal	
80	0x50	Iniciar una nueva medición real. Resultado en pulgadas
81	0x51	Iniciar una nueva medición real. Resultado en centímetros
82	0x52	Iniciar una nueva medición real. Resultado en microsegundos
86	0x56	Iniciar una nueva medida falsa. Resultado en pulgadas
87	0x57	Iniciar una nueva medida falsa. Resultado en centímetros
88	0x58	Iniciar una nueva medida falsa. Resultado en microsegundos
92	0x5C	Transmite una ráfaga de 8 ciclos de 40khz- no hace cálculos de medición
96	0x60	Fuerza un reinicio del sonar SRF02 realizando un ciclo de autoajuste.
160	0xA0	1º comando de la secuencia para cambiar la dirección I2C
165	0xA5	3º comando de la secuencia para cambiar la dirección I2C
170	0xAA	2º comando de la secuencia para cambiar la dirección I2C

2.6 SENSORES DE GAS MQ-4 Y MQ-7

- **MQ-4 (sensor de gas metano)**

Este es un sensor muy sencillo de usar para gas natural comprimido (CNG), ideal para medir concentraciones de gas natural (compuesto en su mayor parte de metano [CH4] en el aire. El MQ-4 puede detectar concentraciones desde 200 hasta 10000ppm. Este sensor es de alta sensibilidad y con un tiempo de respuesta rápido. Su salida es una resistencia analógica. El circuito para operarlo es bastante simple, lo único que se necesita es alimentar el devanado calefactor con %V, añadir una resistencia de carga y conectar la salida a un ADC. Este sensor tiene un empaquetado bastante similar al sensor de alcohol MQ-3.En la **figura 26** puede observarse su forma física.

Fig. 26 MQ-4

- **MQ-7 (sensor de monóxido de carbono)**

Este sensor de fácil uso, se usa para a detección de Monóxido de Carbono (CO), ideal para sensar concentraciones de CO en el aire. El MQ-7 puede detectar concentraciones en el rango de 20 a 2000ppm. Este sensor tiene una alta sensibilidad y un corto tiempo de respuesta. La salida de este sensor es una resistencia analógica, al igual que el MQ-4 y el MQ-6, y puede operarse con tan solo proporcionarle 5V al igual que en los dos anteriores.

3. METODOLOGÍA

3.1 DESARROLLO

Para poder comprender la complejidad, y los distintos tipos de soluciones del sistema, iniciamos haciendo lectura a 3 importantes métodos de inteligencia artificial:

- Redes Neuronales Artificiales
- Lógica Difusa
- Algoritmos Genéticos

El diseño del algoritmo inteligente, se realizó tomando como base estos métodos basados en inteligencia artificial, teniendo en cuenta que se busca la mejor opción de éstas para resolver la problemática que tiene el robot para resolver la toma de decisiones al explorar un entorno no controlado.

De estos métodos, se optó por utilizar el método de Redes Neuronales Artificiales, que muestra las siguientes ventajas:

- Son sistemas distribuidos lineales
- Son sistemas tolerantes a fallos
- Adaptabilidad
- Establecen relaciones no lineales entre datos
- Posibilidad de implementación en VLSI

La forma más práctica de implementar la RNA en el sistema es utilizando un computador convencional, pues esto permite visualizar y modificar fácilmente diversas características de la red. A pesar de ser un sistema de procesamiento secuencial, el computador puede ser utilizado para implementar una RNA, ya que se puede programar para realizar cualquier tipo de proceso algorítmico, incluyendo la simulación de un sistema de procesamiento en paralelo.

En este trabajo se utilizó el software Matlab para diseñar y simular la red neuronal, y posteriormente fue exportada a la plataforma Arduino, para ser implementada en el sistema en tiempo real controlando los movimientos y la toma de decisiones del robot.

3.2 DIAGRAMA A BLOQUES DEL FUNCIONAMIENTO DEL SISTEMA

En la **figura 27** se observa el diagrama a bloques, donde observamos los dispositivos que se encuentran conectados a nuestra placa Arduino, la cual tiene conectado a sus entradas los sensores ultrasónicos y analógicos, y a su único puerto serial el modulo GPS GM862. A las salidas, se puede observar que solo se encuentran conectados los drivers que controlan a los motores del robot, debido a que este Driver (L298D) es un puente H, y necesita 2 señales digitales para poder utilizarlo.

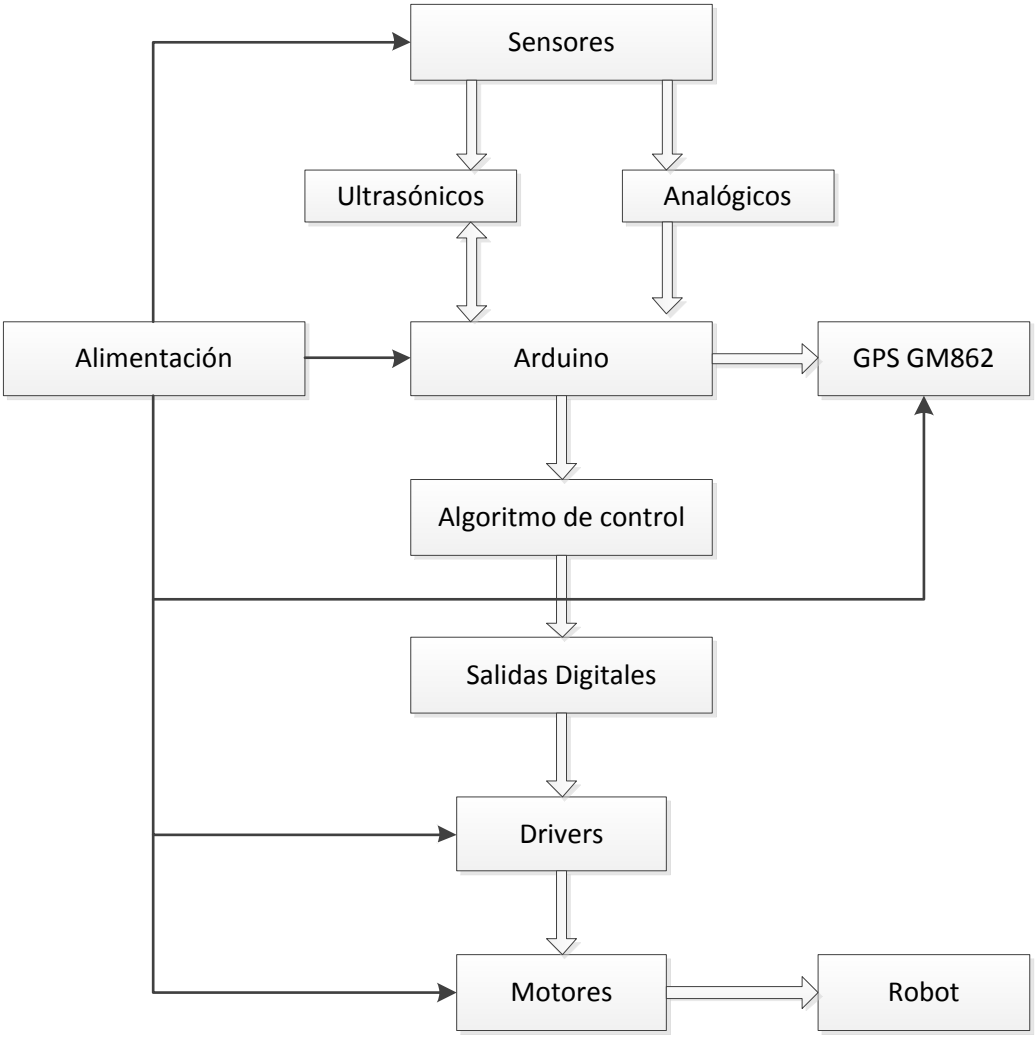


Fig. 27 Diagrama a bloques

3.3 ANÁLISIS DE ALGORITMOS

Antes de comenzar a hacer la programación, es necesario hacer un análisis del tipo de red que utilizará el Robot para resolver la toma de decisiones, si bien ya elegimos el método de redes neuronales artificiales, existen muchos tipos de redes que podemos implementar, para ello necesitaremos lo siguiente:

- La red debe recibir todos los datos de entrada, y usar todos de forma "paralela" dentro del algoritmo, para ello, la capa de entrada deberá constar de 7 entradas, ya que estamos usando 3 sensores ultrasónicos y 4 analógicos.
- Al usar 2 puentes H para controlar las 4 llantas del motor, será necesario que la red disponga de una capa de salida constituida por 4 neuronas, para controlar cada par de llantas.
- Debido a lo mencionado anteriormente, podemos observar que será necesario crear una red multicapa, ya que con una red Monocapa no sería posible resolver esta problemática, debido a la cantidad de entradas y salidas de la red.

Teniendo en cuenta estas necesidades, fue necesario crear una red Perceptrón multicapa de 3 entradas y 3 salidas, para primero crear un algoritmo que solo trabaje con los sensores ultrasónicos. Una vez hecho esto, se procedió a trabajar con una red Perceptrón multicapa de 4 entradas 4 salidas, que corresponden a la lectura de los sensores analógicos, esto solamente como prueba, para observar cómo responde el sistema.

Una vez listas las 2 simulaciones, ya podremos trabajar con la red final de 7 entradas y 4 salidas. La finalidad de esta actividad fue comprender como la red neuronal resuelve la problemática que se le está presentando, y separa los patrones de otros para resolver el problema que se le plantea.

Como estamos trabajando con un entrenamiento supervisado, es más fácil para el programador trabajar con la red, ya que es más factible poder hacer cambios en la red, tanto en las entradas como en las salidas, cambios en el número de capas de la red, así como el número de neuronas de cada capa, e incluso si fuera necesario, la respuesta de la red y sus condiciones a cada salida respectivamente.

Posteriormente, procederemos a simular una red de tipo Backpropagation de 7 entradas y 4 salidas, definiendo cuantas capas ocultas tendrá después de haber hecho la simulación de las anteriores. Esta red nos dará mejores resultados, ya que los cálculos se hacen en las capas ocultas, y la conectividad de la red es mayor que la de una red Monocapa.

3.4 SIMULACIONES Y PRUEBAS EN MATLAB

Una vez comprendidas las funciones de los tipos de redes adecuadas para este sistema, procedemos a hacer simulación en el software Matlab, basándonos en los 2 tipos de redes mencionados: el Perceptrón y la red Backpropagation.

Creación de un perceptrón simple 1 entrada

Para crear el perceptrón simple, utilizaremos el comando *newp* incluido en el Neural Network Toolbox de Matlab. Este comando, crea un perceptrón, el cual será necesario introducirle otros datos, que son el vector de entradas de la red, y la salida deseada. La función de transferencia y el método de aprendizaje Matlab los toma por default como hardlim y learnp respectivamente, así que no los modificamos. Dicho comando se escribe de la siguiente manera:

>>net=newp (p,t)

Donde *p* es el vector de entradas, *t* es la salida deseada, y *net* es la red. En esta primera simulación, tendremos una sola entrada, que corresponderá a un solo sensor ultrasónico, por lo cual solo tendremos una sola entrada que ira de los valores 20 a 600, y la salida de la red será 0 o 1, 0 cuando un objeto no se haya detectado a 30 cm o menos del sensor, y 1 cuando haya un objeto dentro de este rango. En la ventana de comando de Matlab se hace de la siguiente forma:

>>p=[20 600];	//se crea el vector de entrada
>>t=[1 0];	//se crea el vector de salida deseada
>>net=newp(p,t);	//se crea la neurona, tomando los valores de p y t
>>ney=init(net);	//se inicializan los pesos y las bias de las neuronas
>>net.iw, net.b	//se observa los valores de pesos y bias iguales a cero
>>net.trainparam.epochs=1000;	//indicamos que serán 100 los bucles de entrenamiento
>>net=train(net,p,t);	//se entrena la red, con los valores de p y t
>>net.iw, net.b	//se observa como después del entrenamiento cambiaron los valores de los pesos y las bias
>>y=sim(net,p)	//se simula la red, y esta arrojara la salida del sistema en el vector y.

Esto puede hacerse tambien desde un Edit, como se observa en la figura 28. en este caso es necesario guardar el Edit, y despues, solo se llama desde la ventana de comandos con el nombre del archivo guardado.

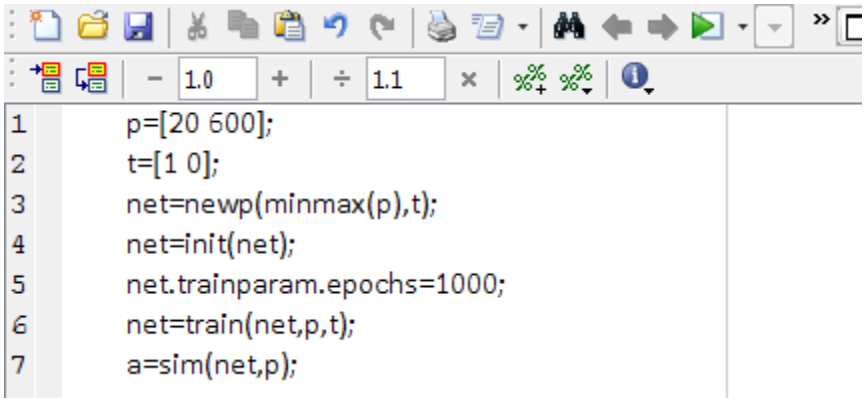


Figura 28 Codigo en Matlab

En la figura 29 observamos como Matlab lleva a cabo el entrenamiento de la red, el ajuste de pesos, la simulación, y nos da la respuesta del sistema. Esto, para un solo sensor.

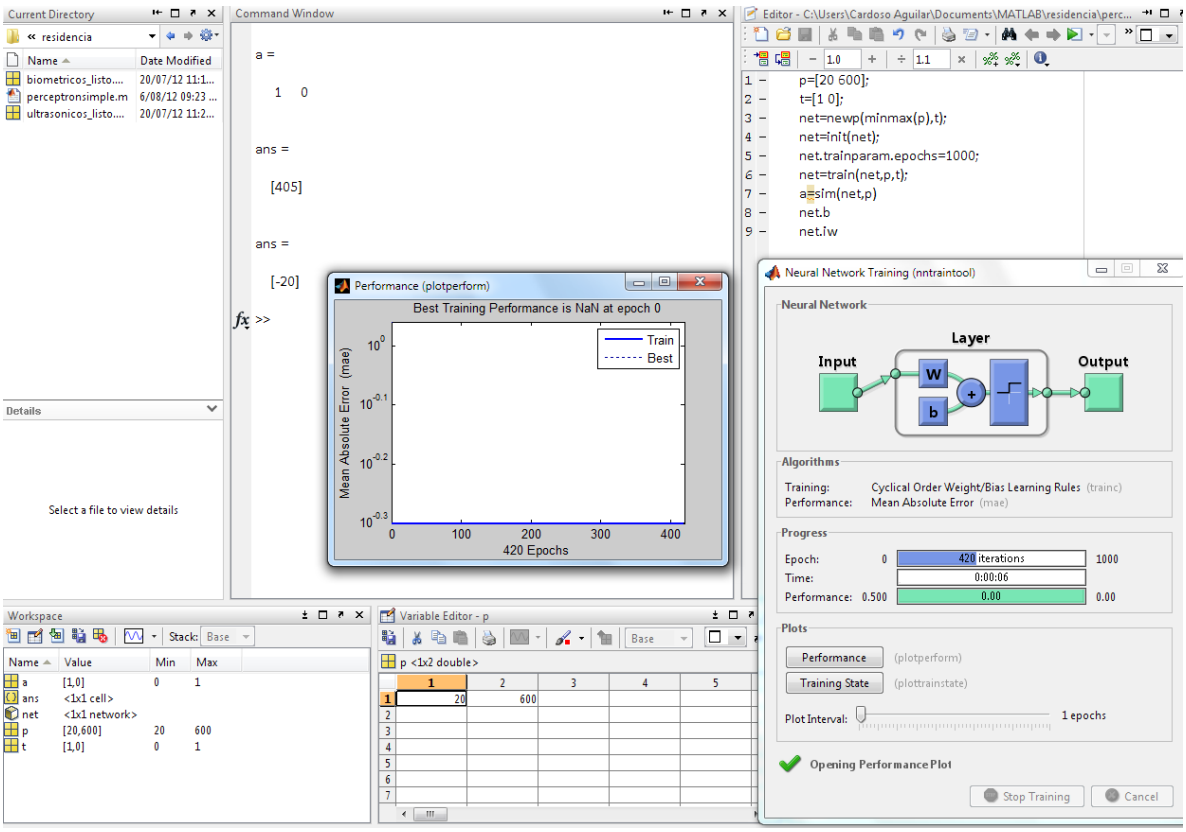


Fig. 29 Entrenamiento de la red neuronal

Creación de una capa de Perceptrones de 3 entradas

Si bien es muy sencillo crear un Perceptrón simple, con una capa de Perceptrones hay que tomar en consideración otros aspectos muy importantes, como lo son configurar cada salida deseada para cada neurona de forma independiente, por ello, se considera una red más compleja y eficiente en comparación de un Perceptrón simple.

Para este ejercicio, tendremos que crear 3 perceptrones, los cuales compartirán las mismas 3 entradas provenientes de los sensores ultrasónicos. Con esto, será necesario crear 3 vectores de entrada, o en su caso, una matriz que contenga a los 3 vectores de entrada, que simularan los datos que se están recibiendo de los 3 sensores ultrasónicos conectados a la red neuronal.

Así mismo, es necesario ingresar 3 vectores distintos para las salidas deseadas, pues cada neurona tendrá una salida distinta cuando reciba las mismas señales de entrada. Y por consiguiente, crear 3 neuronas, las cuales comparten las mismas entradas, pero no tienen la misma salida, como ya se ha mencionado anteriormente. Como ya sabemos, los valores que entrarán a la red irán desde 20 a 600, por lo cual debemos ingresar los vectores que contengan las combinaciones principales, para este caso, que puedan darse en la lectura de los sensores ultrasónicos. Los comandos escritos en Matlab serían los siguientes:

```
>>p=[600 600 600 20 20;          //creación de la matriz de entradas
      600 20 20 20 20;
      600 20 600 600 20];
>>t1=[0 1 1 0 0];                //creación de las salidas deseadas por neurona
>>t2=[1 0 0 0 0];
>>t3=[0 0 0 1 0];
>>net1=newp(p,t1);               //creación de las neuronas net1, net2 y net3
>>net2=newp(p,t2);
>>net3=newp(p,t3);
>>net1=init(net1);               //inicialización de pesos y bias
>>net2=init(net2);
>>net3=init(net3);
>>net1.trainparam.epochs=10;     //configuración del número de épocas de cada
>>net2.trainparam.epochs=1000;     neurona
>>net3.trainparam.epochs=100;
>>net1=train(net1,p,t1);         //entrenamiento de las 3 neuronas
>>net2=train(net2,p,t2);
>>net3=train(net3,p,t3);
>>a1=sim(net1,p)                 //simulación de las 3 neuronas,
>>a2=sim(net2,p)
>>a3=sim(net3,p)
>> net1.iw{1,1}, net2.iw{1,1}, net3.iw{1,1}  //obtención de pesos y bias de cada neurona
>> net1.b, net2.b, net3.b
```

En el Edit, nuestro código sería el siguiente (**figura 30**):

```
clear,clc
p=[600 600 600 20 20;
   600 20 20 20 20;
   600 20 600 600 20];
t1=[0 1 1 0 0];
t2=[1 0 0 0 0];
t3=[0 0 0 1 0];
net1=newp(p,t1); net2=newp(p,t2); net3=newp(p,t3);
net1=init(net1); net2=init(net2); net3=init(net3);
net1.trainparam.epochs=10;
net2.trainparam.epochs=1000;
net3.trainparam.epochs=1000;
net1=train(net1,p,t1); net2=train(net2,p,t2); net3=train(net3,p,t3);
a1=sim(net1,p), a2=sim(net2,p), a3=sim(net3,p)
net1.iw{1,1}, net2.iw{1,1}, net3.iw{1,1},
net1.b, net2.b, net3.b
```

Fig. 30 Capa de Perceptrones

Al ejecutar nuestro programa, se observa como cada neurona tiene una respuesta distinta, pues cada una tiene un objetivo diferente, por lo cual la neurona 1 puede resolver el problema planteado en 2 épocas, mientras que neurona 2 lo hace en 847 épocas. Esto se observa en la **figura 31**.

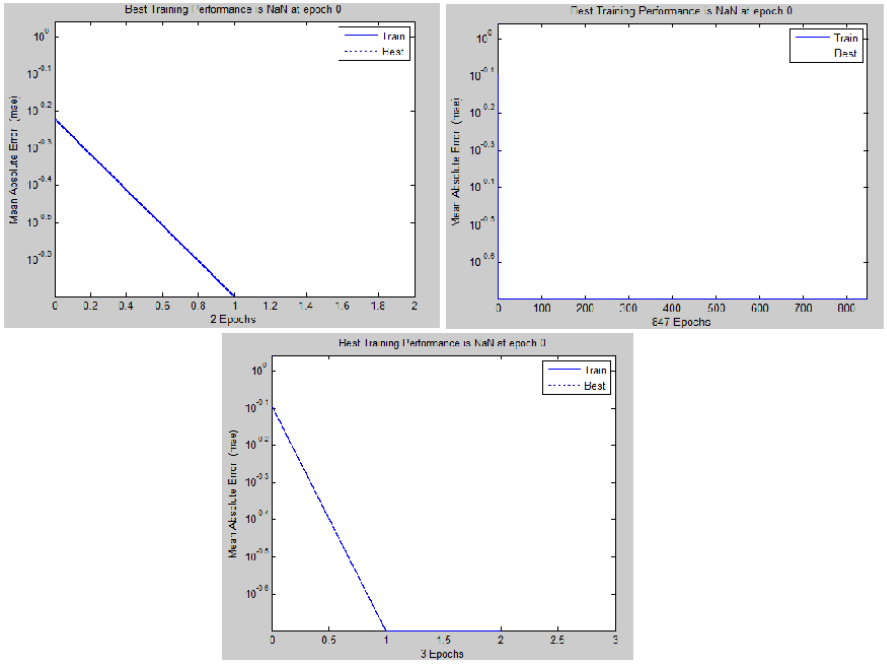


Fig. 31 Entrenamientos de la capa de neuronas

3.4.1 CREACIÓN DE UNA RED BACKPROPAGATION DE 3 ENTRADAS (SENSORES SRF02)

En nuestro caso, haremos uso solamente del comando *newff* con las configuraciones siguientes: *P* es un vector de entradas, T es el vector de salidas deseadas, S es el vector de tamaño de las N-1 capas ocultas, TF es el vector que contiene las funciones de transferencia de cada capa, BTF es la función de entrenamiento de la red, ya que para las demás, Matlab les asigna un valor por default, el cual no es necesario cambiar debido al tipo de red que utilizaremos y a su complejidad.

>>net= newff (P, T, S, TF, BTF, BLF, PF, IPF, OPF, DDF)

Para la red que crearemos, tendremos una capa de entrada de 3 neuronas, las capas ocultas, y una capa de salida de 4 neuronas, debido a que esta red controlará los motores del robot, y ellos disponen de 4 señales lógicas (2 por driver). Para ello, comenzamos creando la matriz de los datos de entrada, y así mismo, la matriz de las salidas deseadas.

>>p=[0 0 0 0 1 1 1 1;0 0 1 1 0 0 1 1;0 1 0 1 0 1 0 1];

>>t=[1.5 0 0 0 1.5 1.5 1.5 1.5;0 1.5 1.5 1.5 0 0 0 0;0 0 0 0 1.5 0 1.5 1.5];

Estas dos matrices, corresponden cada una a cada conjunto de posibles entradas, con su respectiva salida deseada. En nuestro caso, tenemos 8 posibles combinaciones con las 3 entradas de los sensores ultrasónicos, por lo cual tendremos 3 filas con 8 columnas cada una para cumplir con las salidas deseadas de cada combinación. Posteriormente, al crear la red, tomamos en cuenta los parámetros más adecuados para obtener los mejores resultados, los cuales después de un análisis, son los siguientes, tanto en tipo de entrenamiento como en número de capas y neuronas por capa:

>>net=newff(minmax(p),[3 5 4],{'tansig','tansig','purelin'},'traincgp');

Una vez creada la red, pasamos a configurar el entrenamiento, para lo cual hay que primeramente inicializar los pesos, configurar la cantidad de épocas, y el error permitido. Después de ello proseguimos a llevar a cabo el entrenamiento, y observar los resultados obtenidos, para esto, obtendremos una ventana como la de la **figura 32**:

>>net=init(net);

>>net.trainparam.epochs=1000;

>>net.trainparam.goal=1e-3;

>>net=train(net,p,t);

>>a=sim(net,p)

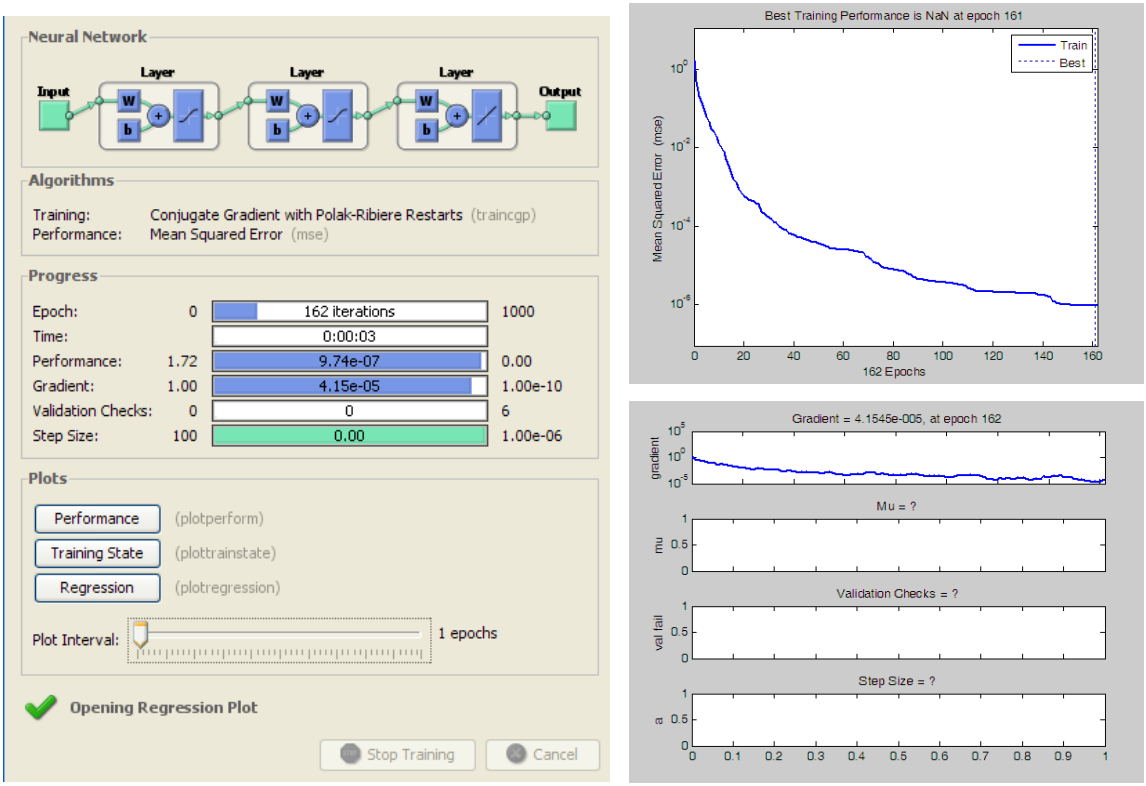


Fig. 32 Entrenamiento de la red

Podemos observar como el toolbox logró obtener el resultado convergente al llegar a las 162 iteraciones. Se muestra después en la **figura 33** la gráfica correspondiente.

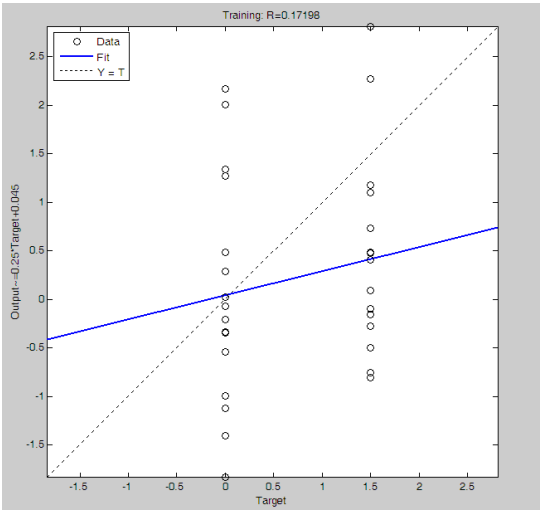


Fig. 33 Separabilidad de la red

Como podemos observar el sistema puede lograr resolver el problema con las condiciones y parámetros que se implementaron, pues estamos hablando de un problema no lineal, el cual la red Backpropagation es capaz de resolver.

Al trabajar con una red Backpropagation, tendremos mayores valores de pesos y bias, los cuales se encontraran guardados en vectores y matrices dentro de la variable llamada "net", y podemos obtenerlos con el comando *net.b*, ya que serán los que implementaremos dentro del código del arduino. Dicho comando nos devolverá un vector de 3 filas 1 columna, y cada una de ellas, tendrá los valores de las bias de cada capa. La fila 1 del vector, corresponde a la primera capa de neuronas de la red, la fila 2 corresponde a la segunda capa de la red, y la fila 3 corresponde a la tercera capa de la red. Para llamarlos solo basta con escribir el comando, especificando la fila y columna del vector.

```
>>net.b{1,1}
>>net.b{2,1}
>>net.b{3,1}
```

Así mismo, para obtener los pesos sinápticos de la red, tendremos que hacer llamado a la matriz de pesos de la primera capa con el comando *net.iw{1,1}*. La cual tiene guardados los pesos de la primera capa, y para obtener los pesos sinápticos de la capa oculta y se los pesos de la capa oculta a la capa de salida, debe usarse el comando siguiente:

```
>>net.lw{2,1}
>>net.lw{3,2}
```

Con esto, ya tendremos todos los datos necesarios para poder implementarlos en el código del Arduino, y ya podremos implementar la red neuronal físicamente. Los resultados de los pesos sinápticos y bias obtenidos de la red se muestran en la **figura 34**.

```
>> net.b{1.1},net.b{2,1},net.b{3,1}
ans =
   5.2585
   1.4385
  -1.0167
ans =
  -2.3891
   1.4314
  -0.5222
   1.0958
   2.4701
ans =
   1.2990
  -0.7316
   0.2269
   0.6312
```

```
>> net.iw{1,1}
ans =
   0.0588  -2.7382  -2.2898
  -3.1601  -2.2137   1.7397
   2.2931   0.3145   3.5053
>> net.lw{2,1}
ans =
   1.1021   1.2825  -1.9854
  -0.5501   1.8527  -2.0468
   0.4322  -0.2474  -2.5017
  -0.2658  -2.0624  -2.2511
   1.7401   1.1111   1.2254
>> net.lw{3,2}
ans =
   1.0819  -1.0897   0.0658  -0.1232   0.3510
  -1.0865   1.0899  -0.0620   0.1241   0.5818
  -0.2292  -0.0606   1.0508  -0.9711   1.1635
   0.2262   0.0607  -1.0482   0.9717  -0.5215
```

Fig. 34 Bias y pesos sinápticos

3.4.2 CREACION DE UNA RED BACKPROPAGATION DE 4 ENTRADAS (SENSORES MQ-4 Y MQ-7)

En el caso de una red de 4 entradas, aumenta el número de combinaciones, debido a que estamos hablando de la lectura de los 4 sensores analógicos del sistema, por lo cual el número de neuronas será mayor. Debido a que tenemos 4 vectores que contienen los datos de cada entrada, será necesario crear una matriz de 4 filas, donde se encontraran todas las combinaciones posibles para este problema.

Para ello es necesario crear los vectores nombrados p1, p2, p3, p4, los cuales posteriormente, deber ser guardados en una matriz llamada "p", que será la cual se deberá ingresar a la red. Así mismo, se debe crear una matriz de salidas deseadas, de la misma forma que se hizo con la matriz de entradas. Dichos resultados podemos observarlos en la **figura 35**.

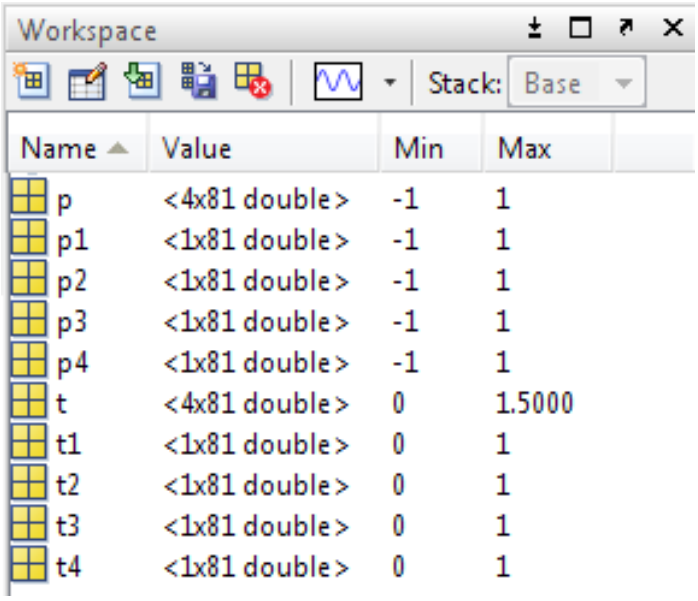


Fig. 35 Variables de entrada y salidas deseadas

Al tener estos datos listos, ay que crear la red tomando los valores de p y t, usando el comando *newff* y debido a la complejidad del problema, es necesario crear una red más compleja, por lo cual ésta cuenta con una capa de entrada de 4 neuronas, una primer capa oculta de 15 neuronas, una segunda capa oculta de 9 neuronas, y una capa de salida de 4 neuronas. Las funciones de transferencia correspondientes deben ser tansig, tansig, tansig y purelin. El método de entrenamiento más efectivo para este caso es el traincgp, al igual que en la red de 3 entradas. El comando final para crear la red es el siguiente:

>>net=newff(minmax(p),[4 15 9 4],{'tansig','tansig','tansig','purelin'},'traincgp');

Una vez creada la red neuronal, hay que inicializar los pesos, y configurar el entrenamiento que se debe llevar a la red para poder conseguir los pesos sinápticos y bias adecuados para resolver el problema planteado. Los comandos siguientes son los encargados de dicha tarea:

```
>>net=init(net);
>>net.trainparam.epochs=1000;
>>net.trainparam.goal=1e-3;
>>net=train(net,p,t);
```

Una vez llevado a cabo dicho entrenamiento, obtendremos las gráficas siguientes, que muestran como la red obtuvo los pesos y bias adecuados hasta la iteración número 562, y se observa en la **figura 36** la separabilidad del sistema, así que disponemos de una red que podrá resolver el problema planteado.

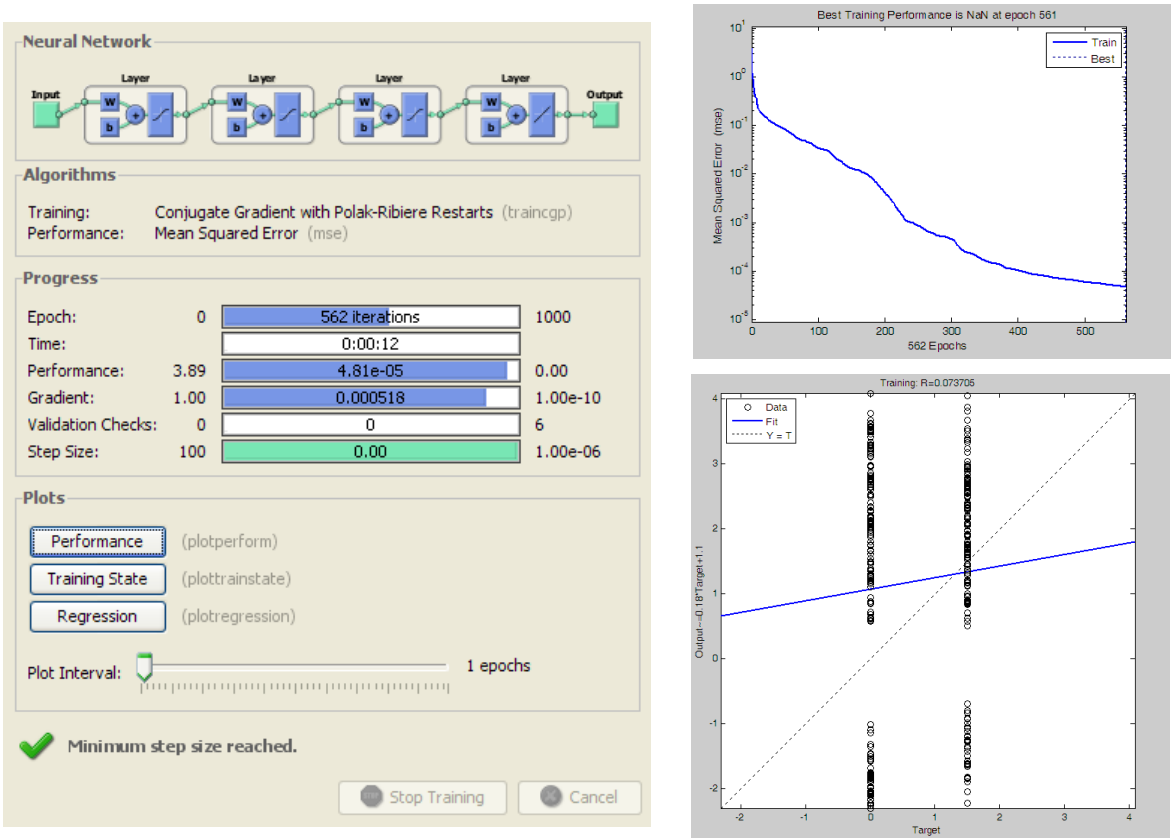


Fig. 36 Entrenamiento de la red y separabilidad del sistema

Una vez obtenidas las salidas deseadas, ya podemos extraer los datos de las bias y pesos sinápticos de la red, de la capa de entrada, capas ocultas, y capa de salida, con los comandos siguientes, los cuales nos entregaran los datos que llevaremos a nuestro código en el arduino.

```
>>net.b{1,1},net.b{2,1},net.b{3,1},net.b{4,1}
>> net.iw{1,1},net.lw{2,1},net.lw{3,2},net.lw{4,3}
```

3.5 PROGRAMACIÓN DEL ALGORITMO EN ARDUINO

Después de simular las redes neuronales de distintas entradas, y con distintas arquitecturas, es necesario llevar la red al Arduino para poder hacer pruebas físicas, por lo cual es necesario crear un "sketch" con el software Arduino, del cual se dispone de la versión 1.0.1, y de un Arduino Duemilanove. Teniendo en cuenta que estamos haciendo uso de sensores analógicos, comunicación serial de tipo I2C, comunicación inalámbrica por el puerto Serial, y salidas digitales, es necesario configurar el Arduino, incluir las librerías necesarias, y crear las variables que vallan a utilizarse en el programa final.

Para ello, primero se debe hacer pruebas a cada uno de los dispositivos, y en base a ello, unir todo el proyecto en conjunto, para tener un programa final que contenga implementados todos los sensores y dispositivos previamente mencionados.

3.5.1 Backpropagation de 3 entradas (Sensores SRF02)

En el caso de la red Backpropagation de 3 entradas, tomamos los datos que nos entregó Matlab, y proseguimos a crear la red, creando variables para las neuronas de la red, las salidas, y las lecturas de los sensores ultrasónicos y las entradas de la red neuronal. Así mismo, es necesario incluir la librería math.h y Wire.h, debido que dentro de la red, es necesario hacer uso de las funciones de transferencia, y de la comunicación I2C respectivamente, lo cual podemos observar en el siguiente código:

.

```
#include <math.h>      //arq 3 5 4, tansig tansig purelin, 3 ultrasonics sensors
#include <Wire.h>

byte p1,p2,p3;
float n1,n2,n3,n4,n5,n6,n7,n8,n9,n10,n11,n12,s1,s2,s3,s4,s5,s6,s7,s8;
unsigned int ultra1,ultra2,ultra3;
byte dir1=112,dir2=114,dir3=115;

void setup()
{
 pinMode(8,OUTPUT);
 pinMode(9,OUTPUT);
 pinMode(10,OUTPUT);
 pinMode(11,OUTPUT);
 Wire.begin();
}
```

En la mismo código anterior podemos observar la parte del “setup”, en donde se puede observar que configuramos solamente 4 salidas digitales, debido a que es lo único que necesitamos en esta prueba para probar el funcionamiento de la red únicamente con los sensores ultrasónicos. Fue necesario crear funciones para la lectura de los sensores ultrasónicos, por lo cual se creó una función llamada “ultrasonic”, la cual se encarga de recibir la dirección del sensor que se desea leer, y entrega la lectura del sensor en centímetros. Asi mismo, se creó una función para interpretar dicha lectura, llamada “in_ultra”, y una función llamada “tansig”, la cual nos servirá como operador, pues es la que llevara a cabo el trabajo de la función de transferencia. Esto podemos observar en el siguiente código.

```
unsigned int ultrasonic(int x)
{
 unsigned int ultra;
 Wire.beginTransmission(x);
 Wire.write(byte(0x00));
 Wire.write(byte(0x51));
 Wire.endTransmission();
 delay(100);
 Wire.beginTransmission(x);
 Wire.write(byte(0x02));
 Wire.endTransmission();
 Wire.requestFrom(x,2);
 if(2 <= Wire.available())
 {
  ultra = Wire.read();
  ultra = ultra << 8;
  ultra |=Wire.read();
 }
 return ultra;
}

byte in_ultra(int x)
{byte ultra;
 if(x<=40)
  {ultra=1;}
 else
  {ultra=0;}
 return ultra;
}

float tansig(float x)
{
 float a;
 a=(pow(2.7182,x)-pow(2.7182,-x))/(pow(2.7182,x)+pow(2.7182,-x));
 return a;
}
```

Una vez terminado todo lo anterior, implementamos la red, tomando los datos que nos entregó Matlab, y podemos observar la parte de la red neuronal dentro del código en siguiente:

```
void loop()
{
 ultra1=ultrasonic(dir1);
 ultra2=ultrasonic(dir2);
 ultra3=ultrasonic(dir3);
 p1=in_ultra(ultra1);
 p2=in_ultra(ultra2);
 p3=in_ultra(ultra3);

n1=(0.0588*p1)+(-2.7382*p2)+(-2.2898*p3)+(5.2586);
n2=(-3.1601*p1)+(-2.2137*p2)+(1.7397*p3)+(1.4386);
n3=(2.2931*p1)+(0.3145*p2)+(3.5053*p3)+(-1.0167);
s1=tansig(n1);
s2=tansig(n2);
s3=tansig(n3);

n4=(1.1021*s1)+(1.2825*s2)+(-1.9864*s3)+(-2.3891);
n5=(-0.5501*s1)+(1.8527*s2)+(-2.0468*s3)+(1.4314);
n6=(0.4322*s1)+(-0.2474*s2)+(-2.5017*s3)+(-0.5222);
n7=(-0.2658*s1)+(-2.0624*s2)+(-2.2511*s3)+(1.0958);
n8=(1.7401*s1)+(1.1111*s2)+(1.2254*s3)+(2.4701);
s4=tansig(n4);
s5=tansig(n5);
s6=tansig(n6);
s7=tansig(n7);
s8=tansig(n8);

n9=(1.0819*s4)+(-1.0897*s5)+(0.0658*s6)+(-0.1232*s7)+(0.3510*s8)+(1.2990);
n10=(-1.0865*s4)+(1.0899*s5)+(-0.062*s6)+(0.1241*s7)+(0.5818*s8)+(-0.7316);
n11=(-0.2292*s4)+(-0.0606*s5)+(1.0508*s6)+(-0.9711*s7)+(1.1635*s8)+(0.2269);
n12=(0.2262*s4)+(0.0607*s5)+(-1.0482*s6)+(0.9717*s7)+(-0.5215*s8)+(0.6312);

 digitalWrite(8,n9);
 digitalWrite(9,n10);
 digitalWrite(10,n11);
 digitalWrite(11,n12);
}
```

En donde se observa la lectura de los 3 sensores ultrasónicos, la entrada a la red, y todo el procesamiento a través de los pesos y las bias, hasta llegar a la capa de salida donde los valores se entregan a las salidas de cada pin respectivamente.

3.5.2 Backpropagation de 4 entradas (Sensores MQ-4 Y MQ-7)

En el caso de los sensores analógicos, el código cambia en la parte de la cabecera y configuración, pues no se hará uso de la comunicación serial de tipo I2C, sino solamente de la librería math.h, por lo cual nuestra cabecera y setup quedaron de la siguiente forma, como se muestra:

```
#include <math.h>        //arq 4 15 9 4
byte p1,p2,p3,p4;
unsigned int mq7_i,mq4_i,mq7_d,mq4_d;
float n1,n2,n3,n4,n5,n6,n7,n8,n9,n10,n11,n12,n13,n14,n15,n16,n17,n18,n19,n20,n21,n22,n23,n24,n25,n26,n27,n28,n29,n30,n31,n32;
float s1,s2,s3,s4,s5,s6,s7,s8,s9,s10,s11,s12,s13,s14,s15,s16,s17,s18,s19,s20,s21,s22,s23,s24,s25,s26,s27,s28;

void setup()
{
 pinMode(8,OUTPUT);
 pinMode(9,OUTPUT);
 pinMode(10,OUTPUT);
 pinMode(11,OUTPUT);
}
```

Como se puede observar, esta red en Matlab fue más compleja, consta de 32 neuronas, con una arquitectura 4 15 9 4, tiene un mayor número de variables. Para la lectura de los sensores analógicos, fue necesario crear funciones distinta, en las cuales se hiciera lectura de las entradas analógicas, y tanto para los sensores MQ-7, como los MQ-4, se crearon sus propias funciones, que acondicionaran nuestra lectura e ingresaran a la red como un dato de entrada como se observa:

```
void read_analog()
{
 mq7_i = analogRead(A0);
 mq4_i = analogRead(A1);
 mq7_d = analogRead(A2);
 mq4_d = analogRead(A3);
 mq4_i = (mq4_i*9.481)+300;
 mq7_i = (mq7_i*1.935)+20;
 mq4_d = (mq4_d*9.481)+300;
 mq7_d = (mq7_d*1.935)+20;
}
```

```
byte in_analog7(int x)
{byte ana;
 if(x>20 && x<500)
 {ana=-1;}
 else
 {if(x>=500 && x<1000)
   {ana=0;}
  else
   {if(x>1000&&x<=2000)
     {ana=1;}}}
 return ana;
}
```

```
byte in_analog4(int x)
{byte ana;
 if(x>300 && x<1000)
 {ana=-1;}
 else
 {if(x>=1000 && x<5000)
  {ana=0;}
  else
  {if(x>=5000 && x<10000)
   {ana=1;}}}
 return ana;
}
```

En este programa, también se hizo uso de la función de transferencia “tansig”, por lo cual se incluye la función “tansig” usada en la red anterior.

```
float tansig(float x)
{float a;
 a=(pow(2.7182,x)-pow(2.7182,-x))/(pow(2.7182,x)+pow(2.7182,-x));
 return a;
}
```

Y por lo tanto, la red neuronal de los sensores analógicos es la que podremos observar en el anexo 1 [página 70].

Como observamos, la red neuronal es mucho más compleja que la anterior red neuronal implementada en los sensores ultrasónicos, debido a que se le presento un problema mucho más complejo, pues los sensores ultrasónicos tienen lecturas entre 20 y 600, valores que no rebasan los mil datos, mientras que los sensores analógicos, entregan valores de 20 a 2,000 partes por millón, en el caso del MQ-7, mientras que para el MQ-4 son valores entre 300 y 10,000 partes por millón, por ello la complejidad de la red neuronal.

3.6 MODELOS MATEMÁTICOS DE LA RED NEURONAL

Una vez concluida la red neuronal el software Arduino y realizadas las simulaciones, y pruebas físicas del robot trabajando con la red neuronal, se obtuvo por medio de MATLAB los modelos matemáticos de las salidas de la red neuronal trabajando con 3 entradas correspondientes a cada salida entregada por los sensores ultrasónicos SRF02.

Dichas entradas, fueron ingresadas a MATLAB de forma simbólica, como podemos observar en el **anexo 4**, para poder hacer un cálculo de la ecuación de cada salida de las neuronas de la capa de salida de la red neuronal, donde cada una tiene todos los elementos, desde las entradas, hasta los pesos sinápticos y salidas de cada capa.

Podemos observar en el **anexo 5**, como se muestra el modelo matemático para la red neuronal de 3 entradas, correspondiente a los sensores ultrasónicos, cada uno con su respectiva ecuación, pues tenemos 4 salidas que son las que controlan directamente la lógica de la toma de decisiones del robot en cuanto se encuentra trabajando en tiempo real:

En donde:

- ultra1, ultra2 y ultra3 son los valores que proporcionan los sensores ultrasónicos.
- w_{ijk} corresponden a los pesos sinápticos de los cuales, sus subíndices indican:
    - i = neurona
    - j = entrada
    - k = capa de la red neuronal
- b_x = bias correspondiente a cada neurona
- s_x = salida de cada neurona

En el uso de herramientas como es la plataforma Arduino, obtuvimos resultados muy satisfactorios, pues gracias al diseño de la placa Arduino Duemilanove (fig. 37), no fue necesario crear placas para conectar el microcontrolador, ni interfaz entre el Microcontrolador y los distintos tipos de dispositivos que utilizamos, tanto xbee como sensores, etc.

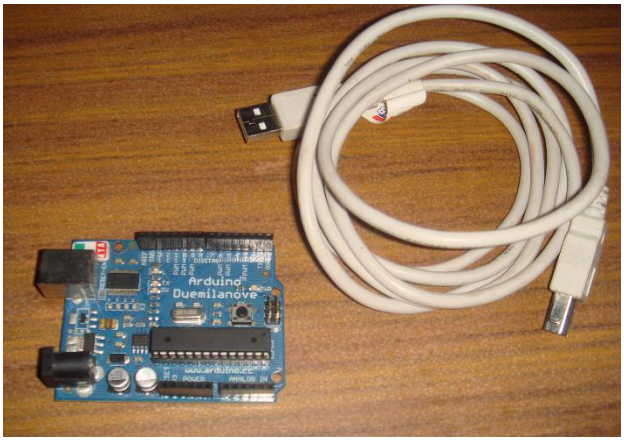


Fig. 37 Arduino Duemilanove

El contar con entradas analógicas, salidas digitales, comunicación Serial I2C, puerto serial, y Serial-USB en un solo sistema, nos permite ahorrar trabajo, tiempo, dinero y espacio en la elaboración del proyecto, pues es un sistema bastante amigable con el programador/diseñador electrónico que desea realizar proyectos basados en esta plataforma. En la figura 38 podemos observar el sistema, conectado a los dispositivos de entrada y salida correspondientes:

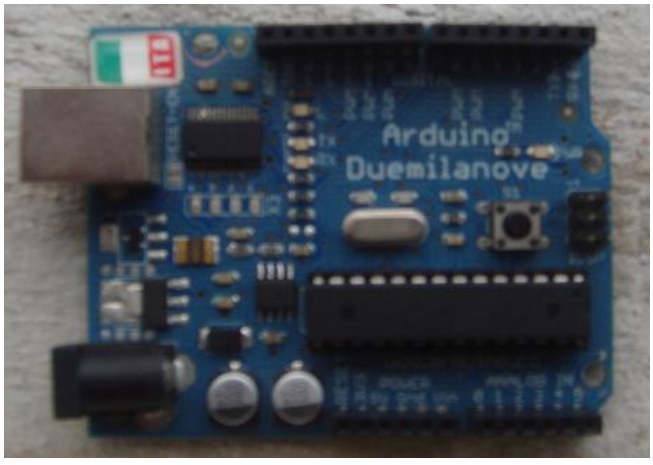


Fig. 38 Arduino Duemilanove - puertos

No se tuvo ningún problema con el uso del sistema Arduino, pues tanto las entradas digitales como analógicas, nos fueron de mucha ayuda, al igual que los puertos de comunicación serial, y gracias a ello el sistema del robot obtuvo buenos resultados; el conexionado de dispositivos digitales y analógicos no proporciono problema alguno gracias a la flexibilidad del uso del sistema, figura 39.

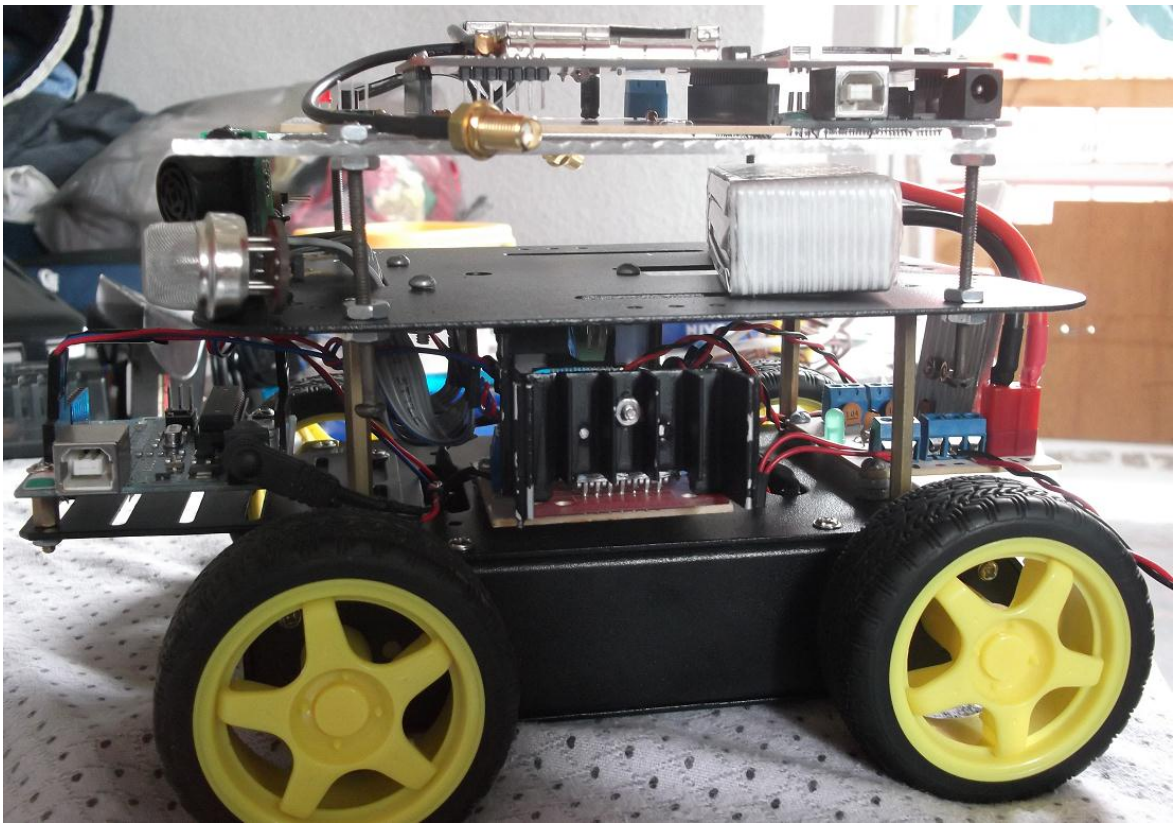


Fig. 39 Sistema arduino implementado en el robot

Por su parte, como pudimos observar en las simulaciones de Matlab, la herramienta precargada para el diseño de redes neuronales nos facilitó el diseño de los algoritmos, pues sin ello, diseñar la red neuronal seria mucho muy complicado. Las gráficas que Matlab arroja al entrenar las redes neuronales de los métodos de entrenamientos fueron muy efectivas, pues nos dan una noción de cómo se está comportando la red, si está llegando a la solución del problema, o estamos cometiendo errores en el diseño de la red neuronal, figura 40.

Gracias a ello, pudimos modelar el funcionamiento de una neurona extremadamente simple, y sin embargo, con una gran capacidad al interconectarse con más neuronas y formar capas de neuronas.

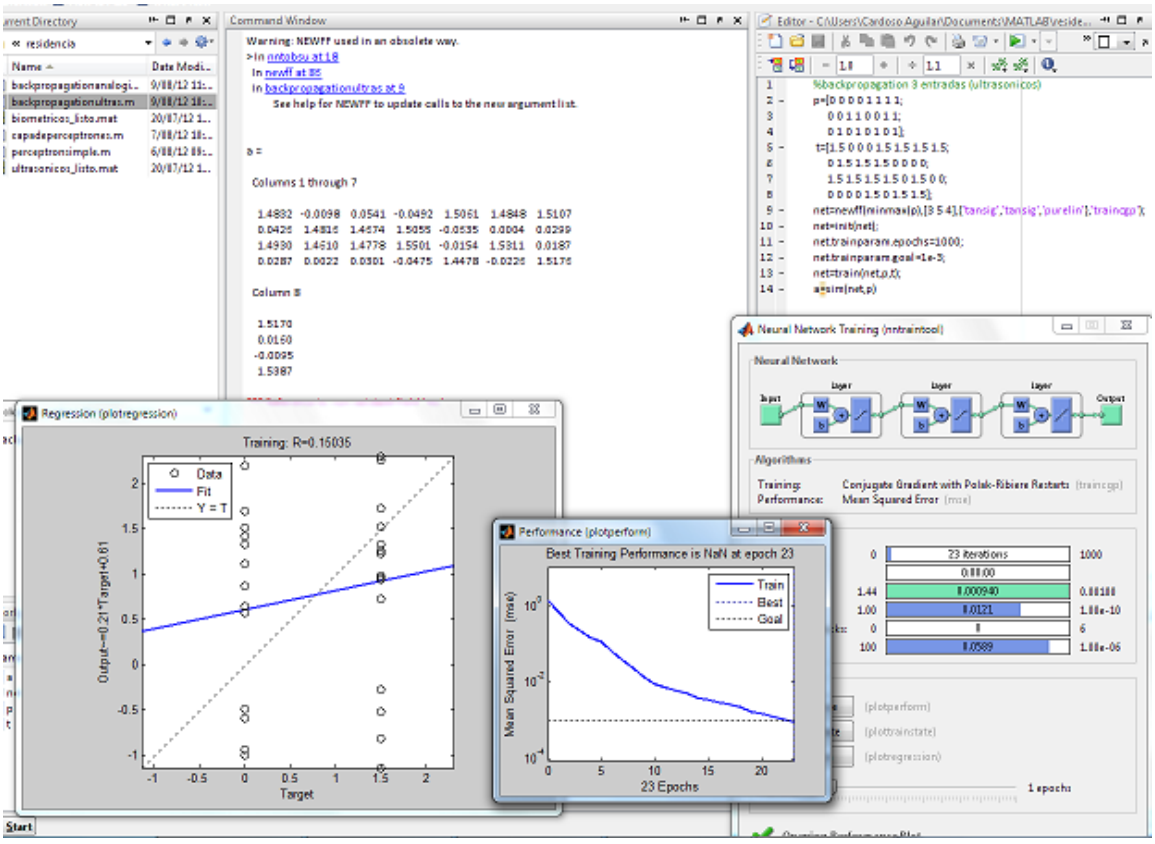


Fig. 40 Simulación en Matlab

Un aspecto importante, fue el elegir la cantidad de capas ocultas, y las neuronas de cada capa, pues con la práctica, se entendió como al aumentar el número de neuronas en cualquier capa oculta, aumentaba notablemente la conectividad de la red neuronal, y en consecuencia, el sistema podía converger en mayor tiempo, no obstante, era capaz de resolver problemas más complejos, lo cual en algunas ocasiones, redes muy simples no podían hacer.

El sistema presenta un aspecto muy peculiar, que es la tolerancia a fallos. Esto en otros sistemas es un problema que no nos permite tener un diseño que nos convenza, en cambio, las redes neuronales permiten este tipo de situaciones, y se adaptan al problema, pues relacionan patrones, con las salidas del sistema lo cual permite que el sistema no se colapse si alguna entrada fallase, o entregara lecturas fuera del rango que se tiene establecido.

Gracias al uso de la red Backpropagation, pudimos procesar gran cantidad de datos en tiempo real, y en ningún momento obtuvimos alguna lentitud por parte del sistema, resultado que nos demuestra que tanto el sistema arduino, como la red neuronal, no tienen ningún problema al ser implementadas.

En el programa final, fue necesario hacer una serie de modificaciones, debido a que la red se volvió más compleja al contar con 7 entradas y 4 salidas, mientras que los datos de las lecturas, se debían enviar a través del puerto serial, a un dispositivo xbee, el cual envía la información leída a una computadora, por lo cual se incluyeron funciones para la lectura de datos, y envió de la información recabada por el robot. Dicha información, es visualizada a través de Proteus, con la ayuda de la herramienta "Compim", en la cual observamos como los datos llegan a la computadora de forma inalámbrica, y nos entregan cantidades de gas CO, metano, distancia de objetos cercanos al robot, y posición GPS de cualquier obstáculo cercano.

La cabecera del programa final, muestra las librerías agregadas al programa, las variables destinadas a entradas analógicas, entradas y direcciones de los sensores ultrasónicos, entradas y salidas de las neuronas, así como la configuración de salidas digitales, velocidad de transmisión para el puerto serial, y habilitación de la comunicación I2C. Posteriormente, tendremos todas las funciones para las lecturas de sensores, conversión de unidades, y limitaciones de lecturas, así como la función de transferencia (anexo 1 pag. 73):

```
#include <math.h>
#include <Wire.h>

byte dir1=112,dir2=114,dir3=115;     //Address of each sensor
unsigned int ultra1,ultra2,ultra3;   //Variables for the ultrasonic sensor readings
byte p1,p2,p3;                       //Variables for the input of network of ultrasonic sensor
float n1,n2,n3,n4,n5,n6,n7,n8,n9,n10,n11,n12;  //Variables for the neural network
float s1,s2,s3,s4,s5,s6,s7,s8;       //Neural network architecture 3 5 4, Backpropagation

unsigned int mq7_i,mq4_i,mq7_d,mq4_d;//Variables for the analog sensor readings
byte p4,p5,p6,p7;                    //Variables for the input of network of analog sensor
float n21,n22,n23,n24,n25,n26,n27,n28,n29,n30,n31,n32,n33,n34,n35,n36,n37,n38,n39,n40;
float n41,n42,n43,n44,n45,n46,n47,n48,n49,n50,n51,n52;
float s21,s22,s23,s24,s25,s26,s27,s28,s29,s30,s31,s32,s33,s34,s35,s36,s37,s38,s39,s40;
float s41,s42,s43,s44,s45,s46,s47,s48;

float x1,x2,x3,x4;

void setup()
{
 pinMode(8,OUTPUT);                          //Left engine  = y1
 pinMode(9,OUTPUT);                          //Left engine  = y2
 pinMode(10,OUTPUT);                         //Right engine = y3
 pinMode(11,OUTPUT);                         //Right engine = y4
 Wire.begin();                               //I2C
 Serial.begin(9600);                         //Baud rate
}
```

El envió de datos, se hace por medio de la función siguiente, llamada send_data, la cual envía dato por dato que va tomando el Arduino de cada sensor, separados por un punto y un salto de línea entre analógicos y ultrasónicos, para poderlos interpretar en la ventana de Matlab, o bien en el "VIRTUAL TERMINAL" de Proteus, como se muestra:

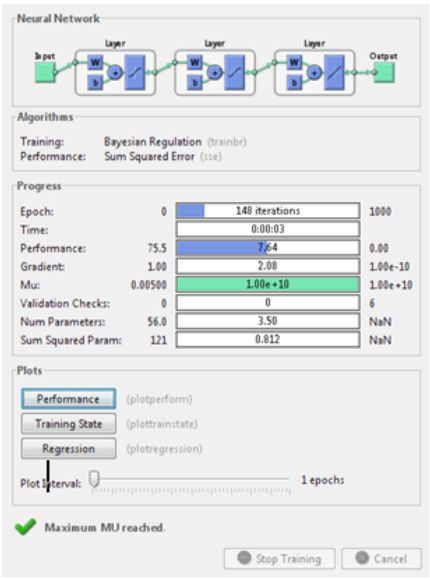
```
void send_data()          //Function for sending sensor data
{
 Serial.print(ultra1);
 Serial.print('.');
 Serial.print(ultra2);
 Serial.print('.');
 Serial.print(ultra3);
 Serial.println('.');
 Serial.print(mq7_i);
 Serial.print('.');
 Serial.print(mq7_d);
 Serial.print('.');
 Serial.print(mq4_i);
 Serial.print('.');
 Serial.println(mq4_d);
}
```

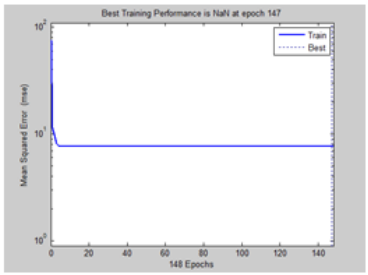
Una vez listas todas las configuraciones necesarias para lecturas, envió de datos, función de transferencia, acondicionamientos de datos, y conversiones, la red neuronal se creó de acuerdo a los datos tomados de las simulaciones de Matlab, de las cuales se crearon 44 neuronas para la red, con 7 neuronas de entrada, y 4 de salida. La red Backpropagation consta de funciones de transferencia tansig, y purelin, las cuales fueron las ideales y básicas para poder llegar a la solución del problema. El código final podemos observarlo en el anexo 2 [página 75].

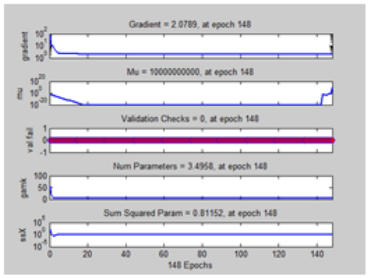
Comparación de los tipos de entrenamiento

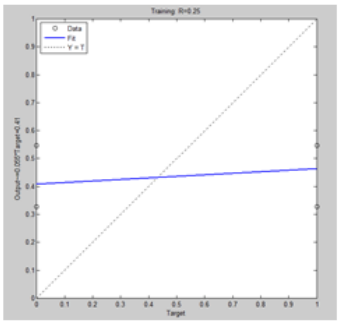
Un aspecto muy importante en el diseño de la red neuronal fue el elegir el tipo de entrenamiento que debía llevar a cabo la red con la cual estuviéramos trabajando, para lo cual se hicieron pruebas y simulaciones, las cuales nos dieron los siguientes resultados:

- **Trainbr**

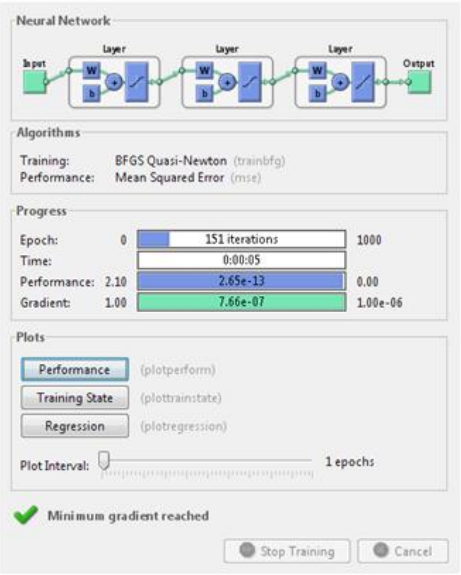


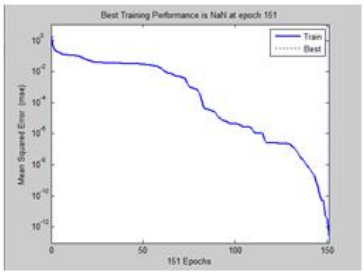


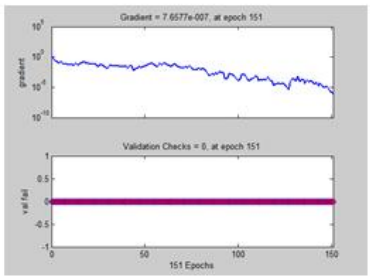


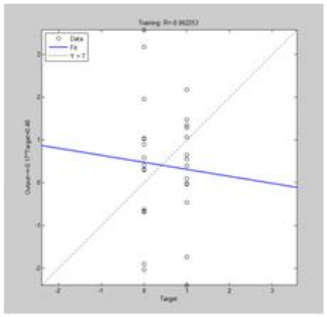


- **Trainbfg**

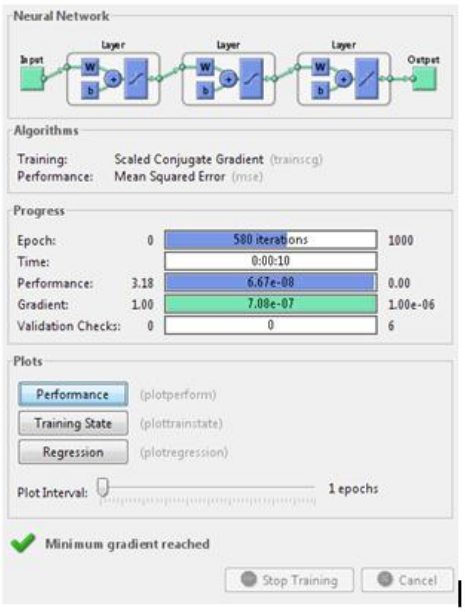


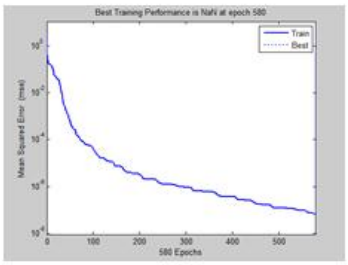


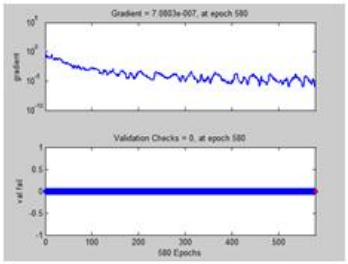


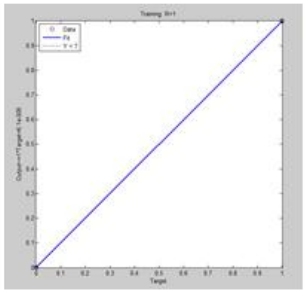


- **Trainscg**

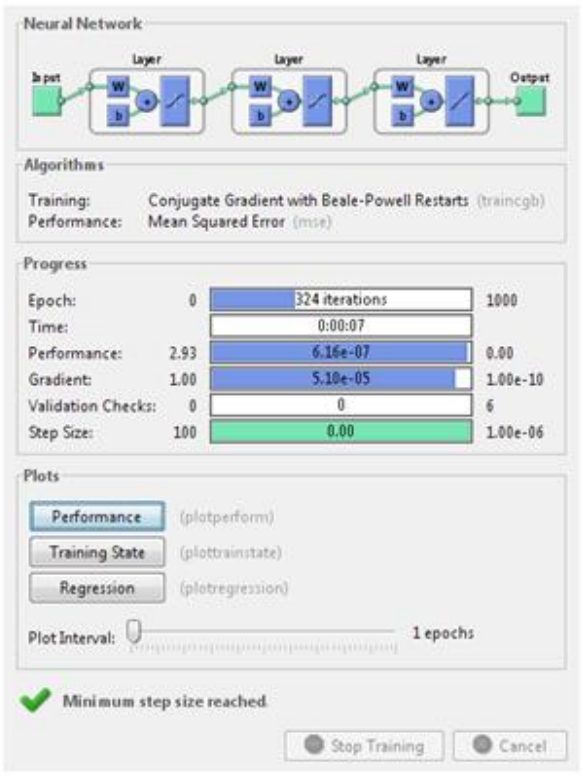


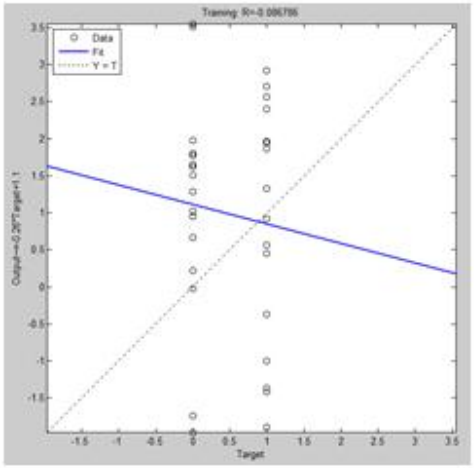




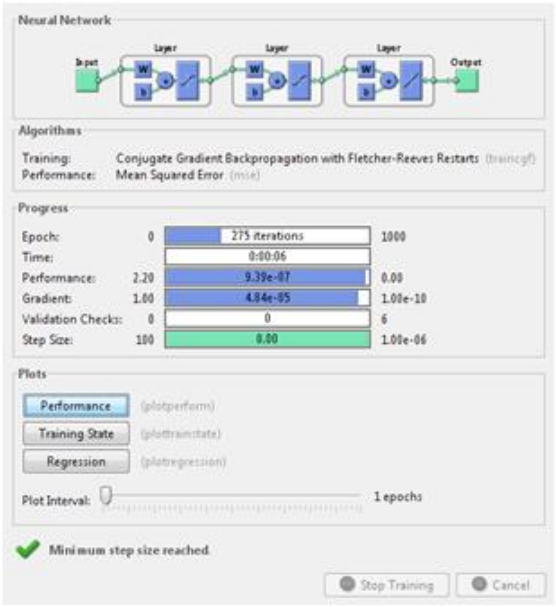


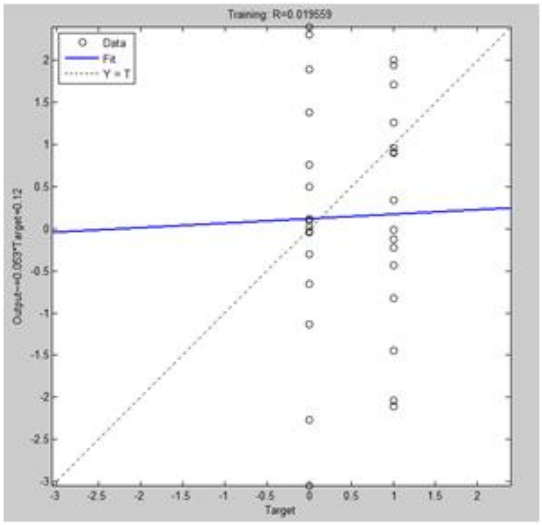
- **Traincgb**



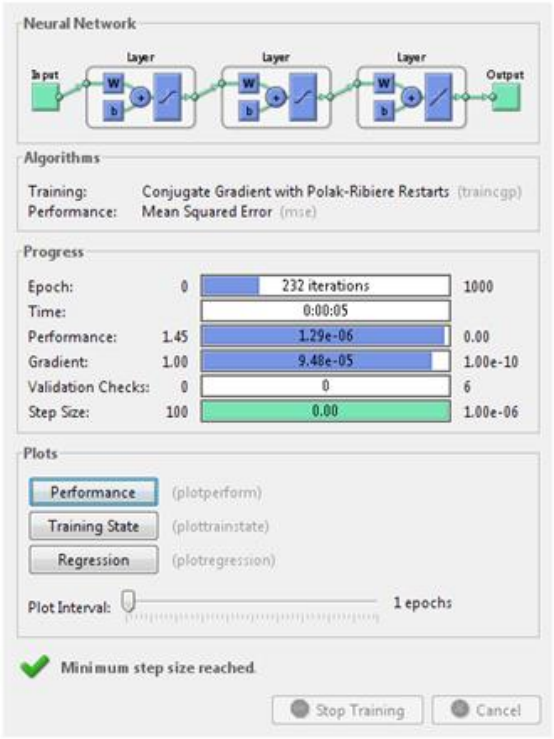


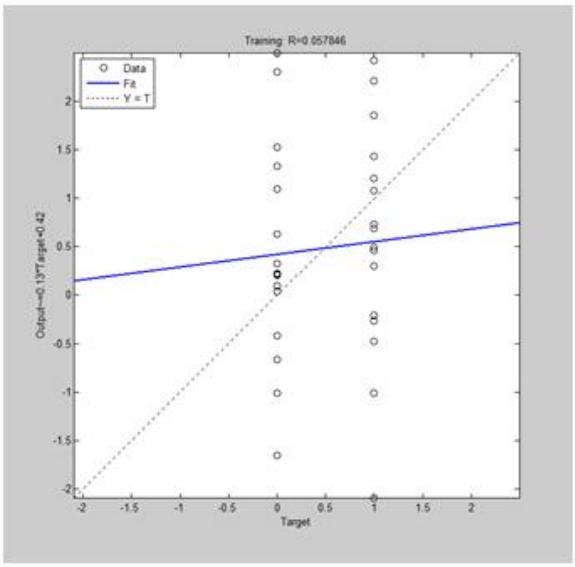
- **Traincgf**





- **Traincgp**





Gracias a estos resultados, pudimos observar como el entrenamiento Traincgf y Traincgp, fueron los más adecuados para diseñar nuestra red, debido a la rapidez en que llega a la solución del problema planteado.

CONCLUSIONES

En este trabajo se ha explorado la utilización de las redes neuronales artificiales como una nueva herramienta para la solución de la problemática de la toma de decisiones de un robot explorador con trabajo colaborativo, del cual se obtuvieron resultados favorables, gracias a las herramientas que hoy en la actualidad, facilitan la simulación de sistemas basados en células biológicas en ordenadores convencionales.

Dentro de los resultados obtenidos, podemos destacar que el uso de la herramienta Neural Network de Matlab, me fue indispensable en la implementación de redes neuronales, pues facilitó el diseño de la red, gracias a las herramientas precargadas en el software. Así, el trabajo desarrollado nos permitió llegar a implementar un sistema basado en la arquitectura de una red neuronal biológica, y comprender como el cerebro humano procesa información, reconoce patrones, y relaciona datos, para interpretarlos de forma inteligente.

Con ello, se comprobó que las redes neuronales artificiales son un método de resolución de problemas, que en lo personal, demuestran cómo no es necesario tener conocimientos en medicina, para poder implementar un modelo biológico en un sistema electrónico. Gracias al trabajo realizado se comprendieron nuevos conceptos no vistos anteriormente en materias como programación, lo cual fue un resultado satisfactorio, pues este tipo de programación esta recientemente implementada en los métodos convencionales de programación para robots en general.

El uso de redes neuronales, no se limita a la robótica, también puede ser aplicado a la medicina, a la estadística, a la física, a la psicología, entre otras ciencias, pues es un método muy efectivo para el análisis de sistemas o resolución de problemas muy complejos. Con ello, podremos crear un sistema inteligente, que pueda procesar cantidades enormes de información de una manera eficiente, con tolerancia a errores gracias a la clasificación de patrones de los datos que procesa la red.

En general, el trabajo realizado en estos 6 meses en inteligencia artificial, específicamente en redes neuronales artificiales, me sirvió para actualizar mis conocimientos en la rama de la programación y la robótica. Si bien ya existen métodos basados lógica digital, las redes neuronales demuestran como la solución de un problema a nivel lógico puede llevarse a un algoritmo matemático basado en una célula biológica, que es capaz de resolver el mismo problema. Gracias a esto, me es posible implementarlo en infinidad de proyectos, no específicamente del área de electrónica como mencionaba, pero si con igual o mejor rendimiento que los métodos ya conocidos.

Anexo 1. Código del programa de los sensores analógicos:

```
void loop()
{
 read_analog();
 p1=in_analog7(mq7_i);
 p2=in_analog4(mq4_i);
 p3=in_analog7(mq7_d);
 p4=in_analog4(mq4_d);

n1=(2.4815*p1)+(-0.7246*p2)+(-0.6590*p3)+(-0.0023*p4)+(-2.3144);
n2=(1.5090*p1)+(0.3343*p2)+(-1.5075*p3)+(-2.4554*p4)+(1.4404);
n3=(0.4051*p1)+(3.2376*p2)+(0.5918*p3)+(-0.1719*p4)+(-0.9546);
n4=(0.0684*p1)+(-1.3652*p2)+(0.5040*p3)+(-0.8228*p4)+(2.6570);
 s1=tansig(n1);
 s2=tansig(n2);
 s3=tansig(n3);
 s4=tansig(n4);

n5=(0.2334*s1)+(1.5458*s2)+(-0.8810*s3)+(-2.0643*s4)+(-2.7881);
n6=(1.8065*s1)+(-0.2228*s2)+(1.4260*s3)+(-1.4955*s4)+(-2.3770);
n7=(1.5228*s1)+(-2.0960*s2)+(1.7719*s3)+(2.0598*s4)+(-2.4744);
n8=(-1.8585*s1)+(0.6085*s2)+(0.4623*s3)+(-2.3271*s4)+(1.3321);
n9=(-0.7229*s1)+(-1.3865*s2)+(-1.8252*s3)+(1.1760*s4)+(1.5503);
n10=(-0.7322*s1)+(-1.9802*s2)+(2.1156*s3)+(0.8705*s4)+(1.3755);
n11=(-0.0731*s1)+(2.1850*s2)+(-1.6272*s3)+(0.9569*s4)+(0.4515);
n12=(0.5029*s1)+(1.5586*s2)+(1.1487*s3)+(2.1200*s4)+(0.6152);
n13=(-1.2741*s1)+(2.3883*s2)+(-1.6573*s3)+(-0.6803*s4)+(-0.8653);
n14=(1.8415*s1)+(1.6053*s2)+(-1.2004*s3)+(-1.2810*s4)+(0.5852);
n15=(1.6509*s1)+(-2.9183*s2)+(0.4519*s3)+(1.8618*s4)+(2.0014);
n16=(-1.9653*s1)+(-1.0371*s2)+(-1.9518*s3)+(1.5510*s4)+(-1.7264);
n17=(1.8789*s1)+(2.6070*s2)+(3.2265*s3)+(-1.5011*s4)+(1.2978);
n18=(1.2219*s1)+(2.7109*s2)+(0.9116*s3)+(-0.9258*s4)+(2.9241);
n19=(1.9649*s1)+(1.3901*s2)+(-0.9918*s3)+(0.8634*s4)+(3.0912);
```

```
s5=tansig(n5);
s6=tansig(n6);
s7=tansig(n7);
s8=tansig(n8);
s9=tansig(n9);
s10=tansig(n10);
s11=tansig(n11);
s12=tansig(n12);
s13=tansig(n13);
s14=tansig(n14);
s15=tansig(n15);
s16=tansig(n16);
s17=tansig(n17);
s18=tansig(n18);
s19=tansig(n19);
```

n20=(-0.697*s5)+(-0.8827*s6)+(-0.8817*s7)+(-0.1299*s8)+(0.3795*s9)+(-0.4231*s10)+(0.0403*s11)+(-0.1264*s12)+(-0.0689*s13)+(-0.3544*s14)+(0.0748*s15)+(0.1520*s16)+(-0.0395*s17)+(-0.0427*s18)+(-0.1231*s19)+(1.8866);

n21=(0.5658*s5)+(0.9414*s6)+(2.3135*s7)+(0.2977*s8)+(-0.078*s9)+(-0.6103*s10)+(-0.9822*s11)+(0.1613*s12)+(0.1556*s13)+(0.2337*s14)+(-0.1221*s15)+(-0.337*s16)+(0.5004*s17)+(1.3506*s18)+(1.1748*s19)+(-1.7384);

n22=(0.4260*s5)+(0.8648*s6)+(0.1647*s7)+(0.2311*s8)+(-1.0628*s9)+(-1.9819*s10)+(0.1899*s11)+(-0.2635*s12)+(0.3742*s13)+(0.7068*s14)+(-2.9499*s15)+(0.4235*s16)+(-1.5714*s17)+(0.4692*s18)+(-0.6709*s19)+(0.1797);

n23=(-0.0162*s5)+(0.9740*s6)+(0.3380*s7)+(-0.2034*s8)+(0.6281*s9)+(0.9782*s10)+(-0.5273*s11)+(-0.0758*s12)+(-0.1906*s13)+(0.7065*s14)+(0.9756*s15)+(-0.3145*s16)+(-0.6221*s17)+(-1.9826*s18)+(-0.2226*s19)+(0.1455);

n24=(0.0357*s5)+(-1.1608*s6)+(1.1327*s7)+(-0.888*s8)+(0.7409*s9)+(-0.2927*s10)+(-0.3032*s11)+(0.2778*s12)+(-0.1247*s13)+(0.1262*s14)+(-1.3414*s15)+(2.5315*s16)+(-3.6989*s17)+(-0.6468*s18)+(0.1101*s19)+(0.3805);

n25=(0.1202*s5)+(0.7136*s6)+(1.3828*s7)+(0.5007*s8)+(-0.4557*s9)+(0.7143*s10)+(-0.1011*s11)+(-1.1103*s12)+(-1.8025*s13)+(-0.065*s14)+(0.3191*s15)+(0.4564*s16)+(-1.4954*s17)+(-0.5851*s18)+(0.0178*s19)+(0.3036);

n26=(-0.1609*s5)+(-0.7228*s6)+(-0.3451*s7)+(-0.056*s8)+(0.0994*s9)+(0.3029*s10)+(0.8047*s11)+(-0.0031*s12)+(0.183*s13)+(-0.5519*s14)+(0.1291*s15)+(0.052*s16)+(-0.1021*s17)+(-0.0148*s18)+(-0.5256*s19)+(1.5552);

n27=(0.0157*s5)+(0.4986*s6)+(0.6802*s7)+(-0.1281*s8)+(0.1939*s9)+(-0.5615*s10)+(-0.0923*s11)+(-0.6456*s12)+(-0.4044*s13)+(0.4459*s14)+(-0.6792*s15)+(0.018*s16)+(0.2763*s17)+(0.0296*s18)+(0.2147*s19)+(-1.5782);

n28=(0.0372*s5)+(-0.47*s6)+(-0.5858*s7)+(0.4636*s8)+(0.2538*s9)+(0.6849*s10)+(-0.1673*s11)+(0.4300*s12)+(0.2972*s13)+(0.471*s14)+(0.1231*s15)+(0.4839*s16)+(-0.4514*s17)+(0.4259*s18)+(0.1476*s19)+(1.8334);

```
s20=tansig(n20);
s21=tansig(n21);
s22=tansig(n22);
s23=tansig(n23);
s24=tansig(n24);
s25=tansig(n25);
s26=tansig(n26);
s27=tansig(n27);
s28=tansig(n28);
```

n29=(-0.2038*s20)+(-0.7043*s21)+(0.5507*s22)+(0.9462*s23)+(-0.5487*s24)+(-0.8956*s25)+(-0.0640*s26)+(-0.1733*s27)+(-0.1475*s28)+(0.5903);

n30=(-0.2303*s20)+(-0.9128*s21)+(-0.552*s22)+(-0.9457*s23)+(0.5468*s24)+(0.8954*s25)+(-0.2537*s26)+(0.8212*s27)+(-0.3697*s28)+(0.7083);

n31=(0.2687*s20)+(-0.5057*s21)+(-0.0005*s22)+(-0.0016*s23)+(0.5008*s24)+(0.0002*s25)+(-0.2192*s26)+(-0.1973*s27)+(-0.5429*s28)+(0.2895);

n32=(0.8263*s20)+(-1.1277*s21)+(0.0032*s22)+(0.0022*s23)+(-0.5025*s24)+(-0.0013*s25)+(-0.9629*s26)+(-0.9097*s27)+(-0.7474*s28)+(-0.6503);

```
digitalWrite(8,n29);
digitalWrite(9,n30);
digitalWrite(10,n31);
digitalWrite(11,n32);
}
```

Anexo 2. Funciones del programa final

```
unsigned int ultrasonic(int x)  //Function of ultrasonic sensor reading
{
 unsigned int ultra;
 Wire.beginTransmission(x);
 Wire.write(byte(0x00));
 Wire.write(byte(0x51));
 Wire.endTransmission();
 delay(100);
 Wire.beginTransmission(x);
 Wire.write(byte(0x02));
 Wire.endTransmission();
 Wire.requestFrom(x,2);
 if(2 <= Wire.available())
 {
  ultra = Wire.read();
  ultra = ultra << 8;
  ultra |=Wire.read();
 }
 return ultra;
}
byte in_ultra(int x)            //Function limitation ultrasonic sensors
{byte ultra;
 if(x<=40)
  {ultra=1;}
 else
  {ultra=0;}
 return ultra;
}
```

```
void read_analog()              //Function analogue sensor reading
{
 mq7_i=analogRead(A0);
 mq4_i=analogRead(A1);
 mq7_d=analogRead(A2);
 mq4_d=analogRead(A3);
 mq4_i=(mq4_i*9.481)+300;
 mq7_i=(mq7_i*1.935)+20;
 mq4_d=(mq4_d*9.481)+300;
 mq7_d=(mq7_d*1.935)+20;
}
byte in_analog7(int x)          //Function limitation of analog sensors MQ7
{
 int ana;
 if(x>20 && x<500)
 {ana=-1;}
 else
 {if(x>=500 && x<1000)
   {ana=0;}
  else
   {if(x>1000&&x<=2000)
     {ana=1;}}}
 return ana;
}
```

```
byte in_analog7(int x)
{
 int ana;
 if(x>20 && x<500)
 {ana=-1;}
 else
 {if(x>=500 && x<1000)
   {ana=0;}
  else
   {if(x>1000&&x<=2000)
     {ana=1;}}}
 return ana;
}
```

```
byte in_analog4(int x)
{int ana;
 if(x>300 && x<1000)
 {ana=-1;}
 else
 {if(x>=1000 && x<5000)
  {ana=0;}
  else
  {if(x>=5000 && x<10000)
   {ana=1;}}}
 return ana;
}
```

Anexo 3. Código final del programa en Arduino.

```
void loop()
{
 ultra1=ultrasonic(dir1);
 ultra2=ultrasonic(dir2);
 ultra3=ultrasonic(dir3);
 read_analog();
 p1=in_ultra(ultra1);
 p2=in_ultra(ultra2);
 p3=in_ultra(ultra3);
 p4=in_analog7(mq7_i);
 p5=in_analog4(mq4_i);
 p6=in_analog7(mq7_d);
 p7=in_analog4(mq4_d);

n1=(0.0588*p1)+(-2.7382*p2)+(-2.2898*p3)+(5.2586);
n2=(-3.1601*p1)+(-2.2137*p2)+(1.7397*p3)+(1.4386);
n3=(2.2931*p1)+(0.3145*p2)+(3.5053*p3)+(-1.0167);
s1=tansig(n1);
s2=tansig(n2);
s3=tansig(n3);
n4=(1.1021*s1)+(1.2825*s2)+(-1.9864*s3)+(-2.3891);
n5=(-0.5501*s1)+(1.8527*s2)+(-2.0468*s3)+(1.4314);
n6=(0.4322*s1)+(-0.2474*s2)+(-2.5017*s3)+(-0.5222);
n7=(-0.2658*s1)+(-2.0624*s2)+(-2.2511*s3)+(1.0958);
n8=(1.7401*s1)+(1.1111*s2)+(1.2254*s3)+(2.4701);
s4=tansig(n4);
s5=tansig(n5);
s6=tansig(n6);
s7=tansig(n7);
s8=tansig(n8);
n9=(1.0819*s4)+(-1.0897*s5)+(0.0658*s6)+(-0.1232*s7)+(0.3510*s8)+(1.2990);
n10=(-1.0865*s4)+(1.0899*s5)+(-0.062*s6)+(0.1241*s7)+(0.5818*s8)+(-0.7316);
n11=(-0.2292*s4)+(-0.0606*s5)+(1.0508*s6)+(-0.9711*s7)+(1.1635*s8)+(0.2269);
n12=(0.2262*s4)+(0.0607*s5)+(-1.0482*s6)+(0.9717*s7)+(-0.5215*s8)+(0.6312);
```

```
n21=(2.4815*p4)+(-0.7246*p5)+(-0.6590*p6)+(-0.0023*p7)+(-2.3144);
n22=(1.5090*p4)+( 0.3343*p5)+(-1.5075*p6)+(-2.4554*p7)+( 1.4404);
n23=(0.4051*p4)+( 3.2376*p5)+( 0.5918*p6)+(-0.1719*p7)+(-0.9546);
n24=(0.0684*p4)+(-1.3652*p5)+( 0.5040*p6)+(-0.8228*p7)+( 2.6570);
s21=tansig(n21);
s22=tansig(n22);
s23=tansig(n23);
s24=tansig(n24);
n25=( 0.2334*s21)+( 1.5458*s22)+(-0.8810*s23)+(-2.0643*s24)+(-2.7881);
n26=( 1.8065*s21)+(-0.2228*s22)+( 1.4260*s23)+(-1.4955*s24)+(-2.3770);
n27=( 1.5228*s21)+(-2.0960*s22)+( 1.7719*s23)+( 2.0598*s24)+(-2.4744);
n28=(-1.8585*s21)+( 0.6085*s22)+( 0.4623*s23)+(-2.3271*s24)+( 1.3321);
n29=(-0.7229*s21)+(-1.3865*s22)+(-1.8252*s23)+( 1.1760*s24)+( 1.5503);
n30=(-0.7322*s21)+(-1.9802*s22)+( 2.1156*s23)+( 0.8705*s24)+( 1.3755);
n31=(-0.0731*s21)+( 2.1850*s22)+(-1.6272*s23)+( 0.9569*s24)+( 0.4515);

n32=( 0.5029*s21)+( 1.5586*s22)+( 1.1487*s23)+( 2.1200*s24)+( 0.6152);
n33=(-1.2741*s21)+( 2.3883*s22)+(-1.6573*s23)+(-0.6803*s24)+(-0.8653);
n34=( 1.8415*s21)+( 1.6053*s22)+(-1.2004*s23)+(-1.2810*s24)+( 0.5852);
n35=( 1.6509*s21)+(-2.9183*s22)+( 0.4519*s23)+( 1.8618*s24)+( 2.0014);
n36=(-1.9653*s21)+(-1.0371*s22)+(-1.9518*s23)+( 1.5510*s24)+(-1.7264);
n37=( 1.8789*s21)+( 2.6070*s22)+( 3.2265*s23)+(-1.5011*s24)+( 1.2978);
n38=( 1.2219*s21)+( 2.7109*s22)+( 0.9116*s23)+(-0.9258*s24)+( 2.9241);
n39=( 1.9649*s21)+( 1.3901*s22)+(-0.9918*s23)+( 0.8634*s24)+( 3.0912);

s25=tansig(n25);
s26=tansig(n26);
s27=tansig(n27);
s28=tansig(n28);
s29=tansig(n29);
s30=tansig(n30);
s31=tansig(n31);
s32=tansig(n32);
s33=tansig(n33);
s34=tansig(n34);
s35=tansig(n35);
s36=tansig(n36);
s37=tansig(n37);
s38=tansig(n38);
s39=tansig(n39);
```

n40=(-0.6970*s25)+(-0.8827*s26)+(-0.8817*s27)+(-0.1299*s28)+(0.3795*s29)+(-0.4231*s30)+(0.0403*s31)+(-0.1264*s32)+(-0.0689*s33)+(-0.3544*s34)+(0.0748*s35)+(0.1520*s36)+(-0.0395*s37)+(-0.0427*s38)+(-0.1231*s39)+(1.8866);

n41=(0.5658*s25)+(0.9414*s26)+(2.3135*s27)+(0.2977*s28)+(-0.0780*s29)+(-0.6103*s30)+(-0.9822*s31)+(0.1613*s32)+(0.1556*s33)+(0.2337*s34)+(-0.1221*s35)+(-0.3370*s36)+(0.5004*s37)+(1.3506*s38)+(1.1748*s39)+(-1.7384);

n42=(0.4260*s25)+(0.8648*s26)+(0.1647*s27)+(0.2311*s28)+(-1.0628*s29)+(-1.9819*s30)+(0.1899*s31)+(-0.2635*s32)+(0.3742*s33)+(0.7068*s34)+(-2.9499*s35)+(0.4235*s36)+(-1.5714*s37)+(0.4692*s38)+(-0.6709*s39)+(0.1797);

n43=(-0.0162*s25)+(0.9740*s26)+(0.3380*s27)+(-0.2034*s28)+(0.6281*s29)+(0.9782*s30)+(-0.5273*s31)+(-0.0758*s32)+(-0.1906*s33)+(0.7065*s34)+(0.9756*s35)+(-0.3145*s36)+(-0.6221*s37)+(-1.9826*s38)+(-0.2226*s39)+(

0.1455);
n44=(0.0357*s25)+(-1.1608*s26)+(1.1327*s27)+(-0.8880*s28)+(0.7409*s29)+(-0.2927*s30)+(-0.3032*s31)+(0.2778*s32)+(-0.1247*s33)+(0.1262*s34)+(-1.3414*s35)+(2.5315*s36)+(-3.6989*s37)+(-0.6468*s38)+(0.1101*s39)+(0.3805);
n45=(0.1202*s25)+(0.7136*s26)+(1.3828*s27)+(0.5007*s28)+(-0.4557*s29)+(0.7143*s30)+(-0.1011*s31)+(-1.1103*s32)+(-1.8025*s33)+(-0.0650*s34)+(0.3191*s35)+(0.4564*s36)+(-1.4954*s37)+(-0.5851*s38)+(0.0178*s39)+(0.3036);
n46=(-0.1609*s25)+(-0.7228*s26)+(-0.3451*s27)+(-0.0560*s28)+(0.0994*s29)+(0.3029*s30)+(0.8047*s31)+(-0.0031*s32)+(0.1830*s33)+(-0.5519*s34)+(0.1291*s35)+(0.0520*s36)+(-0.1021*s37)+(-0.0148*s38)+(-0.5256*s39)+(1.5552);
n47=(0.0157*s25)+(0.4986*s26)+(0.6802*s27)+(-0.1281*s28)+(0.1939*s29)+(-0.5615*s30)+(-0.0923*s31)+(-0.6456*s32)+(-0.4044*s33)+(0.4459*s34)+(-0.6792*s35)+(0.0180*s36)+(0.2763*s37)+(0.0296*s38)+(0.2147*s39)+(-1.5782);
n48=(0.0372*s25)+(-0.4700*s26)+(-0.5858*s27)+(0.4636*s28)+(0.2538*s29)+(0.6849*s30)+(-0.1673*s31)+(0.4300*s32)+(0.2972*s33)+(0.4710*s34)+(0.1231*s35)+(0.4839*s36)+(-0.4514*s37)+(0.4259*s38)+(0.1476*s39)+(1.8334);

```
s40=tansig(n40);
s41=tansig(n41);
s42=tansig(n42);
s43=tansig(n43);
s44=tansig(n44);
s45=tansig(n45);
s46=tansig(n46);
s47=tansig(n47);
s48=tansig(n48);
```

n49=(-0.2038*s40)+(-0.7043*s41)+(0.5507*s42)+(0.9462*s43)+(-0.5487*s44)+(-0.8956*s45)+(-0.0640*s46)+(-0.1733*s47)+(-0.1475*s48)+(0.5903);
n50=(-0.2303*s40)+(-0.9128*s41)+(-0.5520*s42)+(-0.9457*s43)+(0.5468*s44)+(0.8954*s45)+(-0.2537*s46)+(0.8212*s47)+(-0.3697*s48)+(0.7083);
n51=(0.2687*s40)+(-0.5057*s41)+(-0.0005*s42)+(-0.0016*s43)+(0.5008*s44)+(0.0002*s45)+(-0.2192*s46)+(-0.1973*s47)+(-0.5429*s48)+(0.2895);
n52=(0.8263*s40)+(-1.1277*s41)+(0.0032*s42)+(0.0022*s43)+(-0.5025*s44)+(-0.0013*s45)+(-0.9629*s46)+(-0.9097*s47)+(-0.7474*s48)+(-0.6503);

```
send_data();

x1=n9+n49;
x2=n10+n50;
x3=n11+n51;
x4=n12+n52;

 digitalWrite(8,x1);
 digitalWrite(9,x2);
 digitalWrite(10,x3);
 digitalWrite(11,x4);
 delay(500);
}
```

Anexo 4. Modelos matemáticos, simulación de forma simbólica en Matlab

```
%%Trabajo para Memoria de residencia profesional
%%Diseño de algoritmos inteligentes para robots con trabajo colaborativo

%%entradas de los sensores ultrasonicos
syms ultra1 ultra2 ultra3
%%neuronas
syms n1 n2 n3 n4 n5 n6 n7 n8 n9 n10 n11 n12
%%pesos de la red neuronal
syms w111 w121 w131 w211 w221 w231 w311 w321 w331
syms w112 w122 w132 w212 w222 w232 w312 w322 w332 w412 w422 w432 w512 w522 w532
syms w113 w123 w133 w143 w153 w213 w223 w233 w243 w253 w313 w323 w333 w343 w353
syms w413 w423 w433 w443 w453 w513 w523 w533 w543 w553
%%bias de la red neuronal
syms b1 b2 b3 b4 b5 b6 b7 b8 b9 b10 b11 b12
%%salidas de cada neurona
syms s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12
 %%capa de entrada, 3 neuronas 3 entradas
 n1=((w111*ultra1)+(w121*ultra2)+(w131*ultra3)+b1);
 n2=((w211*ultra1)+(w221*ultra2)+(w231*ultra3)+b2);
 n3=((w311*ultra1)+(w321*ultra2)+(w331*ultra3)+b3);
 s1=tanh(n1);
 s2=tanh(n2);
 s3=tanh(n3);

 %%capa oculta, 5 neuronas
 n4=((w112*s1)+(w122*s2)+(w132*s3)+b4);
 n5=((w212*s1)+(w222*s2)+(w232*s3)+b5);
 n6=((w312*s1)+(w322*s2)+(w332*s3)+b6);
 n7=((w412*s1)+(w422*s2)+(w432*s3)+b7);
 n8=((w512*s1)+(w522*s2)+(w532*s3)+b8);
 s4=tanh(n4);
 s5=tanh(n5);
 s6=tanh(n6);
 s7=tanh(n7);
 s8=tanh(n8);

 %%capa de salida, 4 neuronas
 n9=((w113*s4)+(w123*s5)+(w133*s6)+(w143*s7)+(w153*s8)+b9);
 n10=((w213*s4)+(w223*s5)+(w233*s6)+(w243*s7)+(w253*s8)+b10);
 n11=((w313*s4)+(w323*s5)+(w333*s6)+(w343*s7)+(w353*s8)+b11);
 n12=((w413*s4)+(w423*s5)+(w433*s6)+(w443*s7)+(w453*s8)+b12);
 s9=n9
 s10=n10
 s11=n11
 s12=n12
```

Anexo 5. Modelos matemáticos de cada salida de la red neuronal

s9 = b9 + w113*tanh(b4 + w112*tanh(b1 + ultra1*w111 + ultra2*w121 + ultra3*w131) + w122*tanh(b2 + ultra1*w211 + ultra2*w221 + ultra3*w231) + w132*tanh(b3 + ultra1*w311 + ultra2*w321 + ultra3*w331)) + w123*tanh(b5 + w212*tanh(b1 + ultra1*w111 + ultra2*w121 + ultra3*w131) + w222*tanh(b2 + ultra1*w211 + ultra2*w221 + ultra3*w231) + w232*tanh(b3 + ultra1*w311 + ultra2*w321 + ultra3*w331)) + w133*tanh(b6 + w312*tanh(b1 + ultra1*w111 + ultra2*w121 + ultra3*w131) + w322*tanh(b2 + ultra1*w211 + ultra2*w221 + ultra3*w231) + w332*tanh(b3 + ultra1*w311 + ultra2*w321 + ultra3*w331)) + w143*tanh(b7 + w412*tanh(b1 + ultra1*w111 + ultra2*w121 + ultra3*w131) + w422*tanh(b2 + ultra1*w211 + ultra2*w221 + ultra3*w231) + w432*tanh(b3 + ultra1*w311 + ultra2*w321 + ultra3*w331)) + w153*tanh(b8 + w512*tanh(b1 + ultra1*w111 + ultra2*w121 + ultra3*w131) + w522*tanh(b2 + ultra1*w211 + ultra2*w221 + ultra3*w231) + w532*tanh(b3 + ultra1*w311 + ultra2*w321 + ultra3*w331))

s10 = b10 + w213*tanh(b4 + w112*tanh(b1 + ultra1*w111 + ultra2*w121 + ultra3*w131) + w122*tanh(b2 + ultra1*w211 + ultra2*w221 + ultra3*w231) + w132*tanh(b3 + ultra1*w311 + ultra2*w321 + ultra3*w331)) + w223*tanh(b5 + w212*tanh(b1 + ultra1*w111 + ultra2*w121 + ultra3*w131) + w222*tanh(b2 + ultra1*w211 + ultra2*w221 + ultra3*w231) + w232*tanh(b3 + ultra1*w311 + ultra2*w321 + ultra3*w331)) + w233*tanh(b6 + w312*tanh(b1 + ultra1*w111 + ultra2*w121 + ultra3*w131) + w322*tanh(b2 + ultra1*w211 + ultra2*w221 + ultra3*w231) + w332*tanh(b3 + ultra1*w311 + ultra2*w321 + ultra3*w331)) + w243*tanh(b7 + w412*tanh(b1 + ultra1*w111 + ultra2*w121 + ultra3*w131) + w422*tanh(b2 + ultra1*w211 + ultra2*w221 + ultra3*w231) + w432*tanh(b3 + ultra1*w311 + ultra2*w321 + ultra3*w331)) + w253*tanh(b8 + w512*tanh(b1 + ultra1*w111 + ultra2*w121 + ultra3*w131) + w522*tanh(b2 + ultra1*w211 + ultra2*w221 + ultra3*w231) + w532*tanh(b3 + ultra1*w311 + ultra2*w321 + ultra3*w331))

s11 = b11 + w313*tanh(b4 + w112*tanh(b1 + ultra1*w111 + ultra2*w121 + ultra3*w131) + w122*tanh(b2 + ultra1*w211 + ultra2*w221 + ultra3*w231) + w132*tanh(b3 + ultra1*w311 + ultra2*w321 + ultra3*w331)) + w323*tanh(b5 + w212*tanh(b1 + ultra1*w111 + ultra2*w121 + ultra3*w131) + w222*tanh(b2 + ultra1*w211 + ultra2*w221 + ultra3*w231) + w232*tanh(b3 + ultra1*w311 + ultra2*w321 + ultra3*w331)) + w333*tanh(b6 + w312*tanh(b1 + ultra1*w111 + ultra2*w121 + ultra3*w131) + w322*tanh(b2 + ultra1*w211 + ultra2*w221 + ultra3*w231) + w332*tanh(b3 + ultra1*w311 + ultra2*w321 + ultra3*w331)) + w343*tanh(b7 + w412*tanh(b1 + ultra1*w111 + ultra2*w121 + ultra3*w131) + w422*tanh(b2 + ultra1*w211 + ultra2*w221 + ultra3*w231) + w432*tanh(b3 + ultra1*w311 + ultra2*w321 + ultra3*w331)) +

w353*tanh(b8 + w512*tanh(b1 + ultra1*w111 + ultra2*w121 + ultra3*w131) + w522*tanh(b2 + ultra1*w211 + ultra2*w221 + ultra3*w231) + w532*tanh(b3 + ultra1*w311 + ultra2*w321 + ultra3*w331))

s12 = b12 + w413*tanh(b4 + w112*tanh(b1 + ultra1*w111 + ultra2*w121 + ultra3*w131) + w122*tanh(b2 + ultra1*w211 + ultra2*w221 + ultra3*w231) + w132*tanh(b3 + ultra1*w311 + ultra2*w321 + ultra3*w331)) + w423*tanh(b5 + w212*tanh(b1 + ultra1*w111 + ultra2*w121 + ultra3*w131) + w222*tanh(b2 + ultra1*w211 + ultra2*w221 + ultra3*w231) + w232*tanh(b3 + ultra1*w311 + ultra2*w321 + ultra3*w331)) + w433*tanh(b6 + w312*tanh(b1 + ultra1*w111 + ultra2*w121 + ultra3*w131) + w322*tanh(b2 + ultra1*w211 + ultra2*w221 + ultra3*w231) + w332*tanh(b3 + ultra1*w311 + ultra2*w321 + ultra3*w331)) + w443*tanh(b7 + w412*tanh(b1 + ultra1*w111 + ultra2*w121 + ultra3*w131) + w422*tanh(b2 + ultra1*w211 + ultra2*w221 + ultra3*w231) + w432*tanh(b3 + ultra1*w311 + ultra2*w321 + ultra3*w331)) + w453*tanh(b8 + w512*tanh(b1 + ultra1*w111 + ultra2*w121 + ultra3*w131) + w522*tanh(b2 + ultra1*w211 + ultra2*w221 + ultra3*w231) + w532*tanh(b3 + ultra1*w311 + ultra2*w321 + ultra3*w331))

REFERENCIAS BIBLIOGRÁFICAS Y VIRTUALES

[1] “CONSTRUCCIÓN DE ROBOTS AUTÓNOMOS COLABORATIVOS”, TESIS MAESTRÍA EN INFORMÁTICA, FACULTAD DE TECNOLOGÍA, UNIVERSIDAD ABIERTA INTERAMERICANA, Tesista: Ing. Néstor Adrián Balich, Director: Dr. Marcelo De Vincenzi.

[2] “REDES NEURONALES ARTIFICIALES. FUNDAMENTOS, MODELOS Y APLICACIONES”, J. R. HILERA, Ed. Rama, 1995.

[3] “REDES NEURONALES ARTIFICIALES. UN ENFOQUE PRÁCTICO”, PEDRO ISASI VIÑUETA, EDIT. PEARSON PRENTICE HALL, 2003.

[4] HAYKIN S. *NEURAL NETWORKS. A COMPREHENSIVE FOUNDATION,* SECOND EDITION PEARSON PRENTICE HALL, 1999.

[5] “NEURAL NETWORK TOOLBOX”, MARTIN T. HAGAN, MATHWORKS, R2011B, USER GUIDE, PAG. 53 - 84

REFERENCIAS VIRTUALES

[6] https://www.youtube.com/watch?feature=player_embedded&v=9KfqmXQtRGo

[7] http://phys.org/news/2011-11-kilobots-tiny-collaborative-robots.html

[8] http://www.swarmanoid.org/

[9] http://www.swarm-bots.org/

[10] http://arduino.cc/es/Tutorial/HomePage

[11] http://arduino.cc/es/Reference/HomePage

[12] http://www.ehu.es/ehusfera/zientzia-astea/2009/11/13/multi-robots-colaborativos/

[13] http://conecti.ca/2011/10/27/se-comercializa-en-mexico-el-primer-robot-humanoide/

[14] http://dimencionesdescubiertas.blogspot.mx/2012/01/robots-inteligentes-que-piensan-antes.html

[15] http://ro-botica.com/Arduino.asp

[16] http://arduino.cc/es/Main/ArduinoBoardDuemilanove