INSTITUTO TECNOLOGICO DE TUXTLA GUTIERREZ.

RESIDENCIA PROFESIONAL

Alumna: **Alejandra Pérez Gómez**

No. De Control: 09270489

Sistema Integral de Alerta Temprana en Cuenca Hidrológica en la prevención de Desastres Naturales con Base en Redes Neuronales.

Asesor: Dr. Alejandro Medina Santiago.

Tuxtla Gutiérrez, Chiapas.

INDICE

INTRODUCCION ----------3

- Planteamiento del Problema ----------3
- Objetivo----------4
- Justificación----------4
- Hipótesis----------4

MARCO REFERENCIAL ----------4

MARCO TEORICO----------5

METODOLOGIA ---------- 17

RESULTADOS----------23

CONCLUSION ---------- 25

BIBLIOGRAFIA---------- 25

ANEXOS---------- 26

INTRODUCCION

El agua es uno de los recursos naturales más valiosos de cualquier país, debido a los beneficios que se derivan de su consciente explotación; sin embargo, junto con las ventajas existen también las desventajas tales como las inundaciones y sequías. A nivel mundial las inundaciones están aumentando más rápidamente que ningún otro desastre. Uno de los cuatro estados más vulnerable es: Chiapas, debido al desfogue de las centrales hidroeléctricas, y el rio Grijalva, es uno de los más caudalosos del sureste de México, por lo cual el monitoreo de este debe ser constante, para la prevención temprana a la ciudadanía.

El presente proyecto se basa en desarrollar e implementar un sistema integral de alerta temprana en cuenca hidrológica para la prevención de desastres naturales; a través del procesamiento de datos en paralelo con base en redes neuronales. Partiendo de los parámetros que CENAPRED (Centro Nacional de Prevención de Desastres), y del Instituto de Protección Civil de Chiapas.

Utilizaremos un Sensor Ultrasónico con comunicación I2C, a través de un arduino, la red neuronal determinara el nivel de alerta a mandar para le prevención de dicho evento. Esto como una primera etapa del proyecto.

- PLANTEAMIENTO DEL PROBLEMA

El río Sabinal, que recorre de poniente a oriente la capital del estado, es objeto cada año de trabajos de desazolve y limpieza para evitar inundaciones. Sin embargo, no deja de representar un riesgo para los ciudadanos. Un evento extremo de precipitaciones aunado a las condiciones topográficas y al crecimiento desordenado de la mancha urbana podría desencadenar inundaciones en más de 90 puntos considerados como críticos a lo largo del afluente. El sistema de alerta que CONAGUA instalo en el rio Sabinal, no es de gran prevención para los ciudadanos ni para el mismo Instituto de Protección Civil de Chiapas, pues genera costos para el cambio de batería que requiere la actual alerta y en temporada de lluvia tiene que ir personal humano a verificar la altura del afluente.

Con los recursos necesarios, el sistema integral será bastante económico, además que la energía a utilizar será renovable. Alguna de las cuestiones a presentarse será las condiciones donde se harán las mediciones.

- OBJETIVO
    1) Investigar los sistemas de alertas tempranas en cuencas hidrológicas, pudiendo ampliar conocimientos y seguimiento que se le ha dado.
    2) Investigar sensores de nivel, precisos, económicos y autosustentables, que se asocien a dicho proyecto.
    3) Conocimiento y caracterización del sensor, escala de medición, adaptación a pruebas físicas.
    4) Diseño de un sistema de captación de datos que guarde las mediciones predeterminadas para determinación de las alertas del sistema.
    5) Presentación de Resultados.

- JUSTIFICACION

El Sistema Integral de Alerta Temprana, tendrá un semáforo de Alerta lo cual será visible para que la sociedad pueda estar pendiente del nivel que está activado, creando una cultura de la prevención. Será de gran beneficio tanto para las instituciones en materia de prevención, que podrán evacuar a la población de manera anticipada. Así como para las escuelas, comercios, casas y sociedad en general que viva o esté cerca de la cuenca o de sus afluentes, resguardándose a sí mismo o sus bienes materiales.

Sera de bajo costo en instalación y mantenimiento pues el Arduino Mega2560, y el Sensor Ultrasónico son muy accesibles y fáciles de utilizar, así como actualizar constantemente el sistema para futuras mejorías. Además de tener precisión en el sistema de alerta con la Red Neuronal. Se podrá tener un sistema de Energía Renovable que no necesitara el cambio constante de batería. Se tendrá de manera digital los datos, para futuras estadísticas que nos permitan visualizar el comportamiento del mimo.

- HIPOTESIS

El sistema de integral de alerta temprana será capaz de prevenir a la sociedad acerca del nivel de la Cuenca Hidrológica con el Semáforo de Alerta, el cual será visible a la sociedad. Este sistema estará integrado por medio de Redes Neuronales basado en el modelo de backpropagation, el cual generara una salida para continuar como entrada de la siguiente capa de la Red, para así obtener un margen mínimo de error y determinar la salida deseada. También con el apoyo de un Sensor ultrasónico será capaz de medir el nivel de la cuenca en un tiempo aproximado de respuesta de 2.5 Segundos, para tener mejor monitoreo de la cuenca, avisando a los autoridades correspondientes, y que tomen las medidas pertinentes de protección y seguridad.

MARCO REFERENCIAL

Sin duda alguna la Prevención de Desastres Naturales, es algo que nos incluye a toda la sociedad, pero el constante monitoreo de las cuencas queda a cargo del Instituto de Protección Civil. Las alertas tempranas es una situación que se declara a través de instituciones, organizaciones e individuos responsables ya previamente identificados, que permite la información adecuada, precisa y efectiva, previo a la manifestación de un fenómeno peligroso en un área y tiempo determinado, con el fin de que los organismos de emergencia activen procedimientos de acción preestablecidos y la población tome precauciones específicas para evitar o reducir el riesgo. En la cuenca “Rio Sabinal”, que atraviesa la capital del Estado de Chiapas, se ha tratado de mantener un monitoreo, que CONAGUA realizo en el año de 2004. Sin embargo no ha sido tan efectivo para ciudadanía.

Pretendemos realizar un Sistema de Alerta Temprana para la Cuenca Hidrológica “Rio Sabinal”. No solo para que el instituto de Protección Civil pueda ver el monitoreo, si no para la sociedad pueda estar informada de dicho monitoreo de la cuenca hidrológica en el punto del cual le interese.

Hoy en día existen sistemas de alertas tempranas, como lo es en el Estado de Veracruz, Cuenca del Plata en el estado de México, y el vecino país de Guatemala, pero que aún no incluyen a la sociedad en el sistema.

El Rio Sabinal cuenta con 14 Afluentes: 24 de Junio, Arroyos Centro Sur, Cerro Hueco, Chacona, El Poti, Patria Nueva, Pomarrosa, Poc – Poc, San Agustín, San Francisco, San José "El Arenal", San Roque, Santa Ana, Totoposte

Vamos a tomar 3 puntos críticos de la cuenca hidrológica (CONAGUA) para dicha relación con nuestro sistema.

Estaciones de Nivel	Altura(Nivel de Piso)	Relación Sensor-RNA	
Terán- Aeropuerto	5.6 M	[1.1 2.2 3.3 4.4 5.5]	[1 0.8 0.6 0.4 0.2]
Joyyo Mayu	5.4 M	[1 2.1 3.2 4.3 5.4]	[1 0.8 0.6 0.4 0.2]
Parque del Oriente	4.9 M	[0.9 1.8 2.7 3.8 4.9]	[1 0.8 0.6 0.4 0.2]

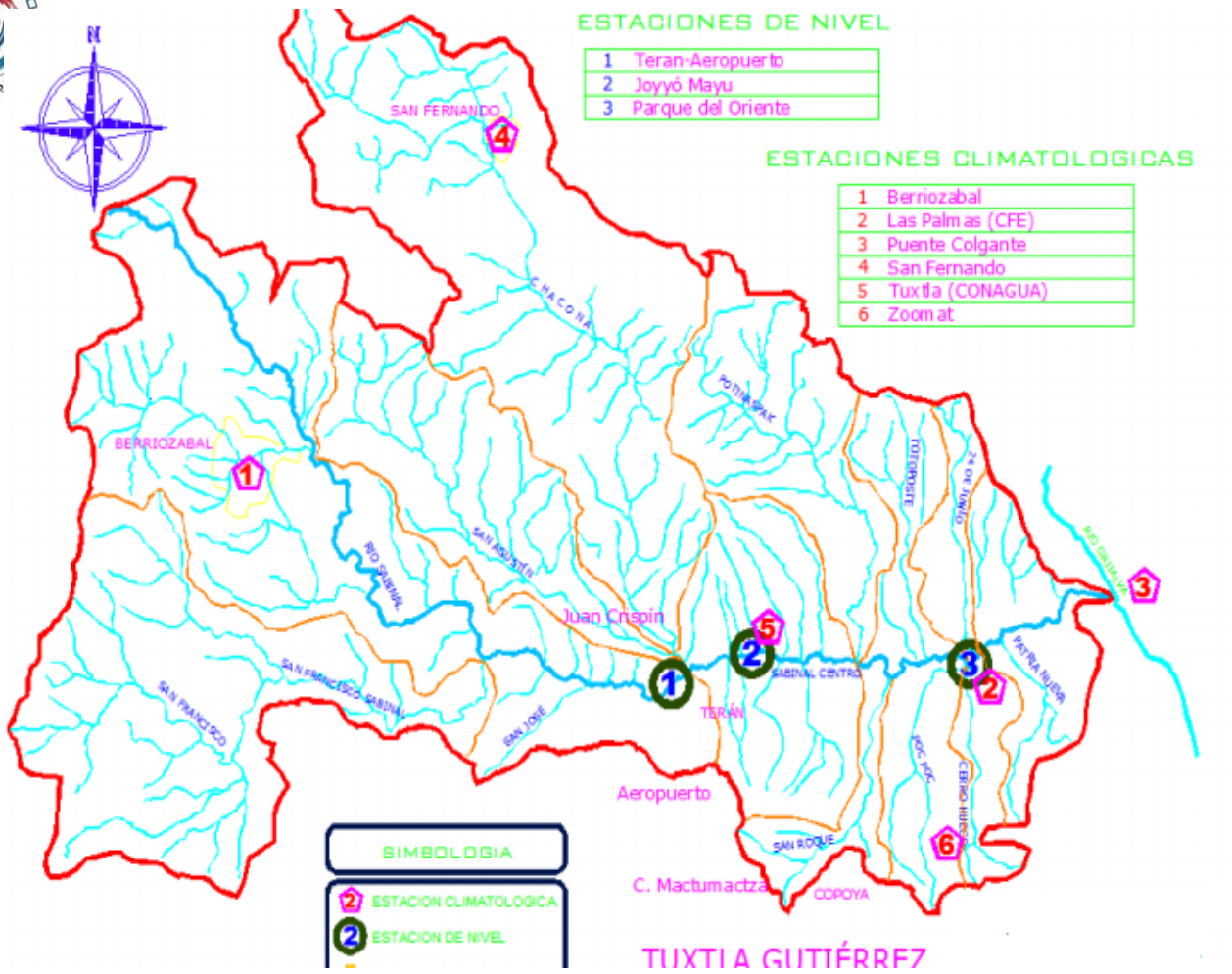


Figura 1. Esquema Hidrométrico de la Cuenca Hidrológica "Sabinal"

MARCO TEORICO

La aplicación de Redes Neuronales es muy amplia, sin embargo utilizaremos la red dentro del nuestro procesador que será un Arduino Mega, la cual hará las acciones que la red decida hacer, para ello, se requiere del Sensor Ultrasónico SSRF02, para el manejo de las distancias del agua a la altura del puente. Este sistema estará entrenado anteriormente en Matlab para observaciones.

Las redes de neuronas artificiales

Denominada habitualmente como RNA, son un paradigma de aprendizaje y procesamiento automático inspirado en la forma en que funciona el sistema nervioso de los animales. Se trata de un sistema de interconexión de neuronas que colaboran entre sí, para producir un estímulo de salida. En inteligencia

artificial es frecuente referirse a ellas como redes de neuronas o redes neuronales.

Propiedades

Una red neuronal se compone de unidades llamadas neuronas. Cada neurona recibe una serie de entradas a través de interconexiones y emite una salida. Esta salida viene dada por tres funciones:

1. Una función de propagación (también conocida como función de excitación), que por lo general consiste en el sumatorio de cada entrada multiplicada por el peso de su interconexión (valor neto). Si el peso es positivo, la conexión se denomina exitatoria; si es negativo, se denomina inhibitoria.
2. Una función de activación, que modifica a la anterior. Puede no existir, siendo en este caso la salida la misma función de propagación.
3. Una función de transferencia, que se aplica al valor devuelto por la función de activación. Se utiliza para acotar la salida de la neurona y generalmente viene dada por la interpretación que queramos darle a dichas salidas. Algunas de las más utilizadas son la función sigmoidea (para obtener valores en el intervalo [0,1]) y la tangente hiperbólica (para obtener valores en el intervalo [-1,1]).

Estructura

Al igual que el cerebro, una RNA se compone de un conjunto masivamente paralelo de unidades de proceso muy simples y es en las conexiones entre estas unidades donde reside la inteligencia de la red. Sin embargo, en términos de escala, un cerebro es muchísimo mayor que cualquier RNA creada hasta la actualidad, y las neuronas artificiales también son más simples que su contrapartida animal.

Biológicamente, un cerebro aprende mediante la reorganización de las conexiones sinápticas entre las neuronas que lo componen. De la misma manera, las RNA tienen un gran número de procesadores virtuales interconectados que de forma simplificada simulan la funcionalidad de las neuronas biológicas. En esta simulación, la reorganización de las conexiones sinápticas biológicas se modela mediante un mecanismo de pesos, que son ajustados durante la fase de aprendizaje. En una RNA entrenada, el conjunto de los pesos determina el conocimiento de esa RNA y tiene la propiedad de resolver el problema para el que la RNA ha sido entrenada.

Por otra parte, en una RNA, además de los pesos y las conexiones, cada neurona tiene asociada una función matemática denominada función de transferencia. Dicha función genera la señal de salida de la neurona a partir de las señales de entrada. La entrada de la función es la suma de todas las señales de entrada por el peso asociado a la conexión de entrada de la señal. Algunos ejemplos de entradas son la función escalón, lineal o mixta, la sigmoide y la función gaussiana, recordando que la función de transferencia es la relación entre la señal de salida y la entrada.

Ventajas

Las redes neuronales artificiales (RNA) tienen muchas ventajas debido a que está basado en la estructura del sistema nervioso, principalmente el cerebro.

- Aprendizaje: Las RNA tienen la habilidad de aprender mediante una etapa que se llama etapa de aprendizaje. Esta consiste en proporcionar a la RNA datos como entrada a su vez que se le indica cuál es la salida (respuesta) esperada.
- Auto organización: Una RNA crea su propia representación de la información en su interior, descargando al usuario de esto.
- Tolerancia a fallos: Debido a que una RNA almacena la información de forma redundante, ésta puede seguir respondiendo de manera aceptable aun si se daña parcialmente.
- Flexibilidad: Una RNA puede manejar cambios no importantes en la información de entrada, como señales con ruido u otros cambios en la entrada (por ejemplo si la información de entrada es la imagen de un objeto, la respuesta correspondiente no sufre cambios si la imagen cambia un poco su brillo o el objeto cambia ligeramente).
- Tiempo real: La estructura de una RNA es paralela, por lo cual si esto es implementado con computadoras o en dispositivos electrónicos especiales, se pueden obtener respuestas en tiempo real.

Modelos:

- Perceptrón
- Adaline
- Perceptrón multicapa
- Memorias asociativas
- Propagación hacia atrás (backpropagation)
- Redes de Elman

- Redes de Hopfield
- Red de contra propagación
- Redes de neuronas de base radial
- Redes de neuronas de aprendizaje competitivo
- Mapas Autoorganizados (RNA) (Redes de Kohonen)
- Crecimiento dinámico de células
- Gas Neuronal Creciente

Aprendizaje

Una segunda clasificación que se suele hacer es en función del tipo de aprendizaje de que es capaz (si necesita o no un conjunto de entrenamiento supervisado). Para cada tipo de aprendizaje encontramos varios modelos propuestos por diferentes autores:

- Aprendizaje supervisado: necesitan un conjunto de datos de entrada previamente clasificado o cuya respuesta objetivo se conoce. Ejemplos de este tipo de redes son: el perceptrón simple, la red Adaline, el perceptrón multicapa, red backpropagration, y la memoria asociativa bidireccional.
- Aprendizaje no supervisado o auto organizado: no necesitan de tal conjunto previo. Ejemplos de este tipo de redes son: las memorias asociativas, las redes de Hopfield, la máquina de Boltzmann y la máquina de Cauchy, las redes de aprendizaje competitivo, las redes de Kohonen o mapas auto organizados y las redes de resonancia adaptativa (ART).
- Redes híbridas: son un enfoque mixto en el que se utiliza una función de mejora para facilitar la convergencia. Un ejemplo de este último tipo son las redes de base radial.
- Aprendizaje reforzado: se sitúa a medio camino entre el supervisado y el auto organizado.

Tipo de entrada

Se pueden clasificar las RNA según sean capaces de procesar información de distinto tipo en:

- Redes analógicas: procesan datos de entrada con valores continuos y, habitualmente, acotados. Ejemplos de este tipo de redes son: Hopfield, Kohonen y las redes de aprendizaje competitivo.
- Redes discretas: procesan datos de entrada de naturaleza discreta; habitualmente valores lógicos booleanos. Ejemplos de este segundo tipo

de redes son: las máquinas de Boltzmann y Cauchy, y la red discreta de Hopfield.

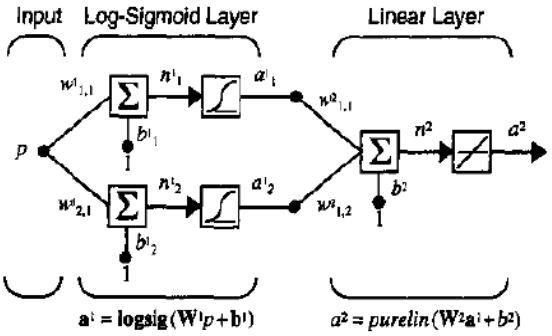


Figura 2. Función de aproximación de una Red.

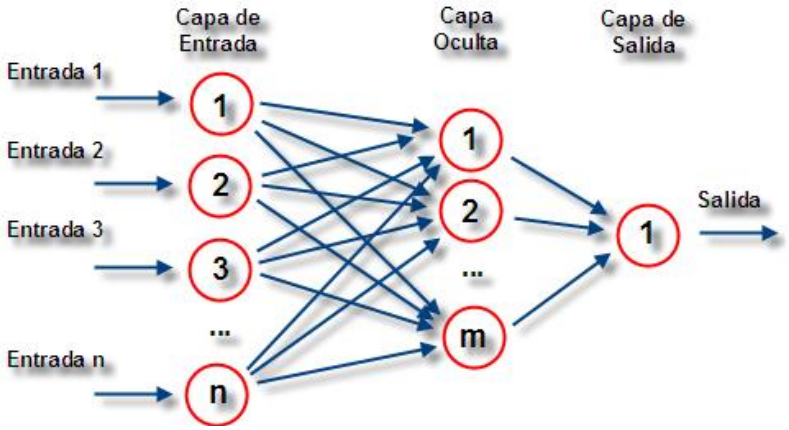


Figura 3. Ejemplo de una red backpropagration

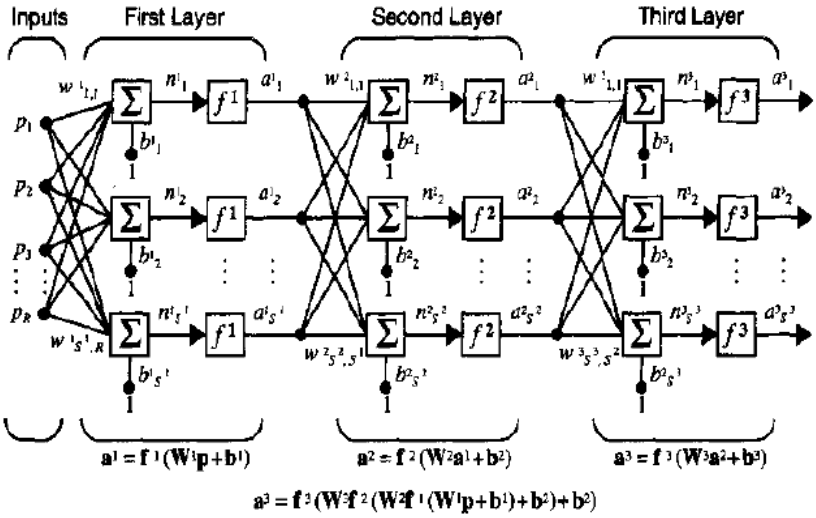


Figura 4. Función backpropagration (Sumatoria de Capas)

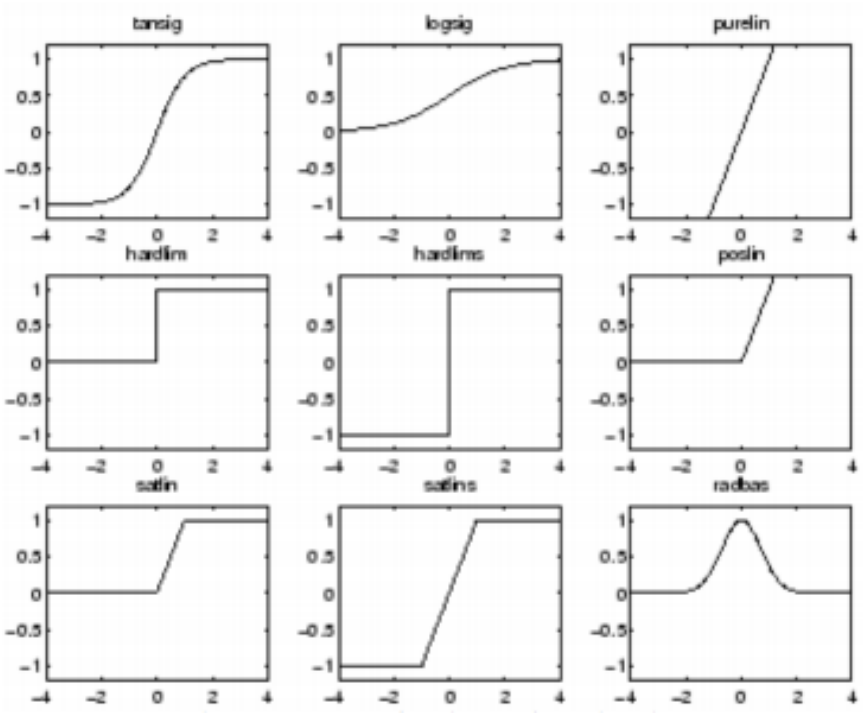


Figura 5. Funciones de activación para las Redes Neuronales.

SRF02 SENSOR DE DISTANCIAS POR ULTRASONIDOS

El sensor de distancias por ultrasonidos SRF02 es 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 de datos y un bit de parada. El nivel de tensión es a nivel TTL lo que permite conectarlo a cualquier microcontrolador del mercado. El modo del rango de medidas es de 15 cm a 800 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 4 mA. Medidas: 24 x 20 x 17 mm de altura. Peso: 4,6 g.

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. La distancia mínima que es capaz de medir es alrededor de 15cm (6"). Como otros sensores de distancia, el SRF02 puede medirse en cm o pulgadas.

Modo I2C

Para seleccionar el modo I2C deberá 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. 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. Esto quiere decir que solo son necesarias dos resistencias en todo el bus, no dos por cada circuito.

La dirección I2C del medidor SRF02 por defecto es 0xE0 pero 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, 0xFE.

Se tiene que tener en cuenta que el Modo Pin debe estar conectada a (0V) para que el SRF02 actúe como modo serie.

El pin (RX) es una entrada por la que se recibe la dirección y el comando a ejecutar y debería estar conectado a el pin transmisor (TX).

El pin (TX) es el que se encarga de transmitir el resultado del comando ejecutado.

Mientras tengan diferentes direcciones se pueden conectar varios sonar SRF02 a un mismo bus, para ello solo deberá conectar la señal TX a todos los pin RX disponibles, igual que se conecta todos los pin TX disponible a una única señal RX del controlador maestro. No debería haber problemas en lo que a la interferencia de señales TX se refiere ya que todas las señales se mantienen en impedancia alta excepto cuando una de ellas tenga que transmitir información.

Arduino Mega

Microcontrolador	ATmega1280
Voltaje de funcionamiento	5V
Voltaje de entrada (recomendado)	7-12V
Voltaje de entrada (limite)	6-20V
Pines E/S digitales	54 (14 proporcionan salida PWM)
Pines de entrada analógica	16
Intensidad por pin	40 mA
Intensidad en pin 3.3V	50 mA
Memoria Flash	128 KB de las cuales 4 KB las usa el gestor de arranque(bootloader)
SRAM	8 KB
EEPROM	4 KB
Velocidad de reloj	16 MHz

Alimentación

El Arduino Mega puede ser alimentado vía la conexión USB o con una fuente de alimentación externa. La placa puede trabajar con una alimentación externa de entre 6 a 20 voltios. Si el voltaje suministrado es inferior a 7V el pin de 5V puede proporcionar menos de 5 Voltios y la placa puede volverse inestable, si se usan más de 12V los reguladores de voltaje se pueden sobrecalentar y dañar la placa. El rango recomendado es de 7 a 12 voltios.
Los pines de alimentación son los siguientes:

- VIN. La entrada de voltaje a la placa Arduino cando se está usando una fuente externa de alimentación (en opuesto a los 5 voltios de la conexión USB). Se puede proporcionar voltaje a través de este pin, o, si se está alimentado a través de la conexión de 2.1mm. acceder a ella a través de este pin.

- 5V La fuente de voltaje estabilizado usado para alimentar el microcontrolador y otros componentes de la placa. Esta puede provenir de VIN a través de un regulador integrado en la placa, o proporcionada directamente por el USB ó otra fuente estabilizada de 5V.
- 3V3. Una fuente de voltaje a 3.3 voltios generada en el chip FTDI integrado en la placa. La corriente máxima soportada 50mA.
- GND. Pines de toma de tierra.

Memoria

El ATmega1280 tiene 128KB de memoria flash para almacenar código (4KB son usados para el arranque del sistema (bootloader).El ATmega1280 tiene 8 KB de memoria RAM. El ATmega1280 tiene 4KB de EEPROM , que puede a la cual se puede acceder para leer o escribir con la [Reference/EEPROM |librería EEPROM]].

Comunicaciones

EL Arduino Mega facilita en varios aspectos la comunicación con el ordenador, otro Arduino u otros microcontroladores. El ATmega1280 proporciona cuatro puertos de comunicación vía serie UART TTL (5V). Un chip FTDI FT232RLintegrado en la placa canaliza esta comunicación serie a traes del USB y los drivers FTDI (incluidos en el software de Arduino) 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. Los LEDS RX y TX de la placa parpadearan cuando se detecte comunicación transmitida través del chip FTDI y la conexión USB (no parpadearan si se usa la comunicación serie a través de los pines 0 y 1). Para el uso de la comunicación SPI, mira en la hoja de especificaciones (datasheet) del ATmega1280. El ATmega1280 también soporta la comunicación I2C (TWI) y SPI. El software de Arduino incluye una librería Wire para simplificar el uso el bus I2C.

Programación

El Arduino Mega se puede programar con el software Arduino. El ATmega1280 en el Arduino Mega viene precargado con un gestor de arranque (bootloader) que permite cargar nuevo código sin necesidad de un programador por hardware externo. Se comunica utilizando el protocolo STK500 original (referencia, archivo de cabecera C).
También te puedes utilizar los pines de 20 (SDA), 21 (SCL) para la comunicación por I2C.

Netbook Gateway LT27

Capacidad de Almacenamiento (GB): 250
Velocidad Procesador (MHz): 1660 Mhz
Memoria RAM (MB): 1024 MB
Tipo de Memoria RAM: DDR2 SDRAM
Tecnología Intel Centrino: No
Sistema Operativo: Windows 7 Starter
Media Center Edition: No
Tamaño (Pulgadas): 10.1 pulgadas
Resolución del Proyector: 1024 x 600
Formato de Reproducción: 16:9
Retroiluminación LED: Edge LED
Grabador de DVD: No
Tipo de Disco Duro: disco duro
Mando a Distancia: No
UMTS: Sí
Tarjeta TV: No
Tarjeta de Edición de Vídeo: No
Cámara Web Incorporada: Sí
LAN Inalámbrico: Sí
Salida TV: No
Firewire: No
Número de Puertos USB: 3
Lector de Tarjeta de Memoria: Lector 2 en 1
Marca del Procesador: Intel
Procesador: ATOM
Número de Procesador: N450
Número de Procesadores: 1
Marca Chipset: Intel
Tipo de Chipset: NM10
Marca del Chip de la Tarjeta Gráfica: Intel
Modelo del Chip de Tarjeta Gráfica: GMA3150
Ancho (Centímetros): 37 cm
Tipo de Ordenador: Portátil
Netbook: Sí

MATLAB

Es una herramienta de software matemático que ofrece un entorno de desarrollo integrado(IDE) con un lenguaje de programación propio (lenguaje M). Está disponible para las plataformas Unix, Windows, Mac OS X y GNU/Linux

Entre sus prestaciones básicas se hallan: la manipulación de matrices, la representación de datos y funciones, la implementación de algoritmos, la creación de interfaces de usuario (GUI) y la comunicación con programas en otros lenguajes y con otros dispositivos hardware.

Tiene aplicaciones en ingeniería y ciencias de cualquier tipo Por ejemplo, resuelve problemas de álgebra lineal, electrónica, finanzas. Hay herramientas de procesamiento de imágenes, optimización, manejo de simulaciones, etc.

PROTEUS

Proteus es un software de diseño electrónico desarrollado por Labcenter Electronics que consta de dos módulos, ARES e ISIS.

ISIS: Mediante este programa podemos diseñar el circuito que deseemos con componentes muy variados, desde una simple resistencia hasta algún que otro microprocesador o micro controlador, incluyendo fuentes de alimentación, generadores de señales y muchas otras prestaciones. Los diseños realizados en Isis pueden ser simulados en tiempo real.

ARES: Es la herramienta de rutado de Proteus, se utiliza para la fabricación de placas de circuito impreso, esta herramienta puede ser utilizada de manera manual o dejar que el propio programa trace las pistas.

METODOLOGIA

El sistema se enfocara en la detección del sistema de alerta, para ello utilizaremos la red neuronal (Backpropagation) con el fin de tener un margen de error muy pequeño, e ir decrementandolo, esto queremos obtener a través de Matlab, donde entrenaremos nuestra red, para obtener nuestros pesos y bias, que nos serviran para construir nuestro programa en Arduino.

Utilizaremos tres sensores a nivel de calle del "RIO SABINAL", esto se debe a la corriente y turbulencia del agua pueden variar de extremo a extremo el nivel en la cuenca, se vio también necesario porque aún hay árboles dentro de la Cuenca que impiden el camino del agua, y se va a las orillas. El sistema Integral se propone de la siguiente manera:

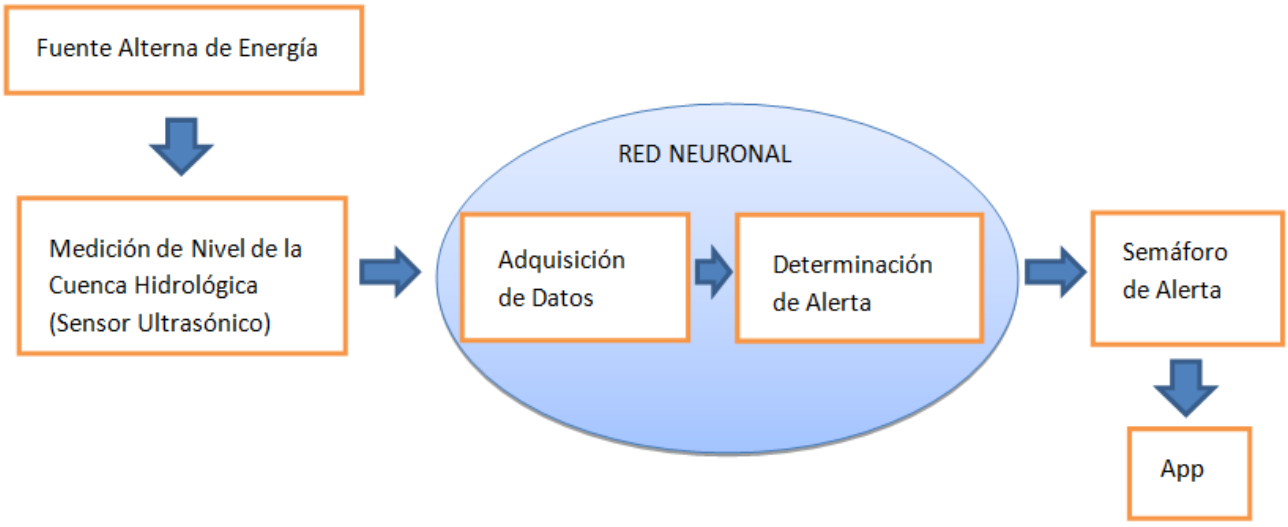


Figura 6. Esquema Propuesto para Sistema de Alerta Temprana

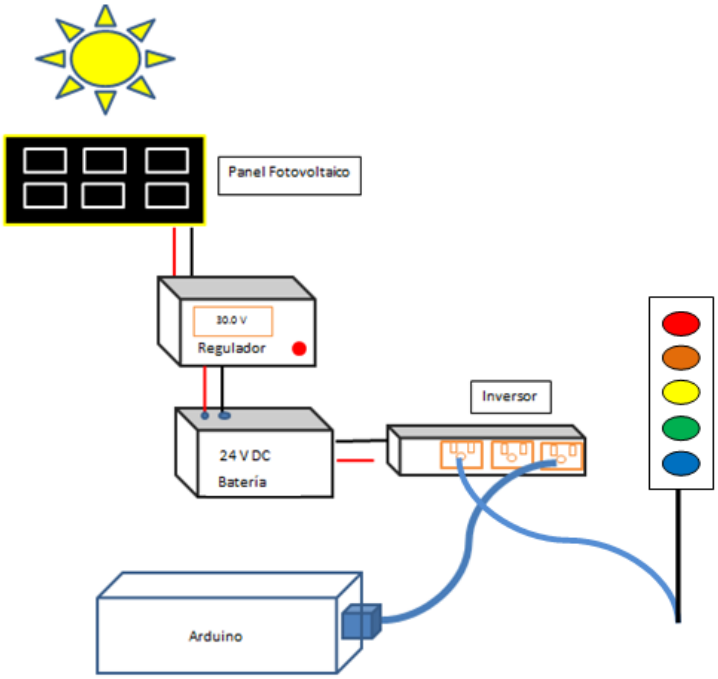


Figura 7. Esquema de propuesta del Sistema

Como parte de la primera etapa del proyecto, se ubica la Red Neuronal como parte importante. Es por ello, es que nuestro enfoque va más al procesamiento y manejo de datos en la red neuronal.

Se propone como Fuente Alterna de Energía, celdas solares. Con las cuales se podrá alimentarse el Arduino así como el Semáforo de Alerta.

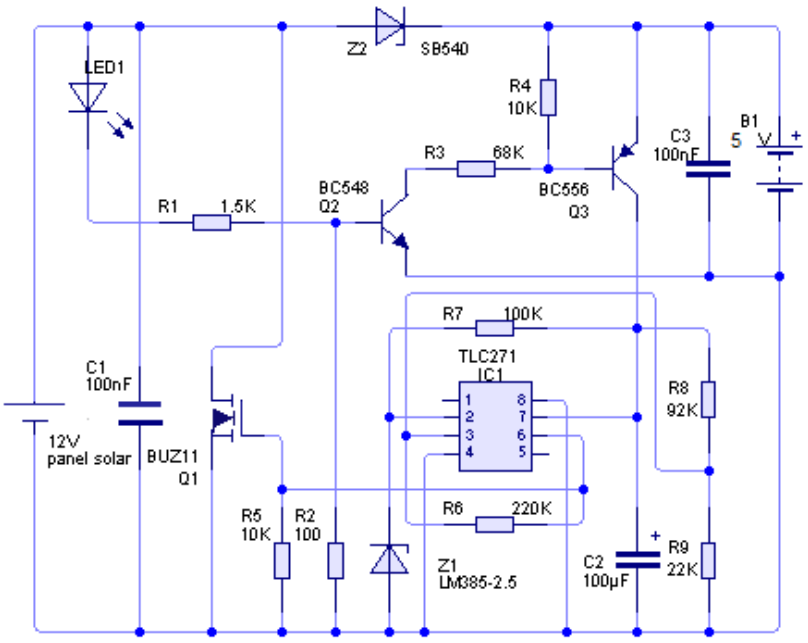


Figura 8. Diagrama Fuente de Energía

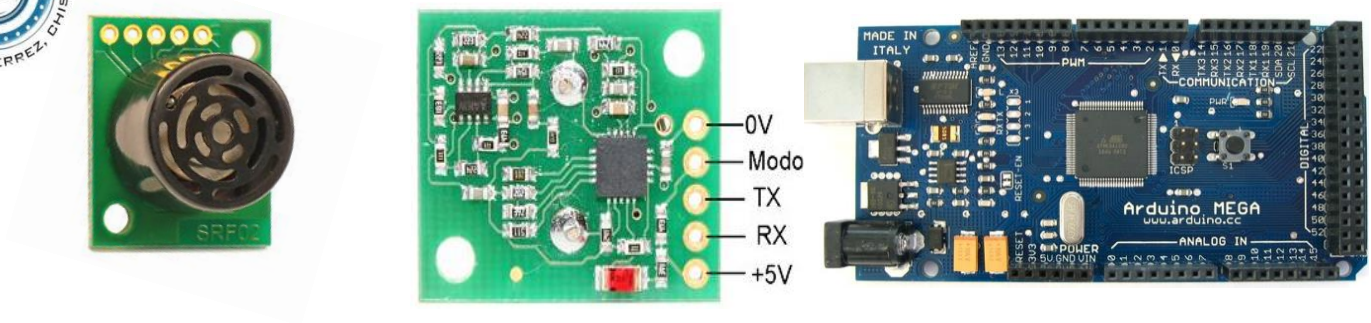


Figura 9. Imágenes del Sensor SRF02 y Arduino Mega

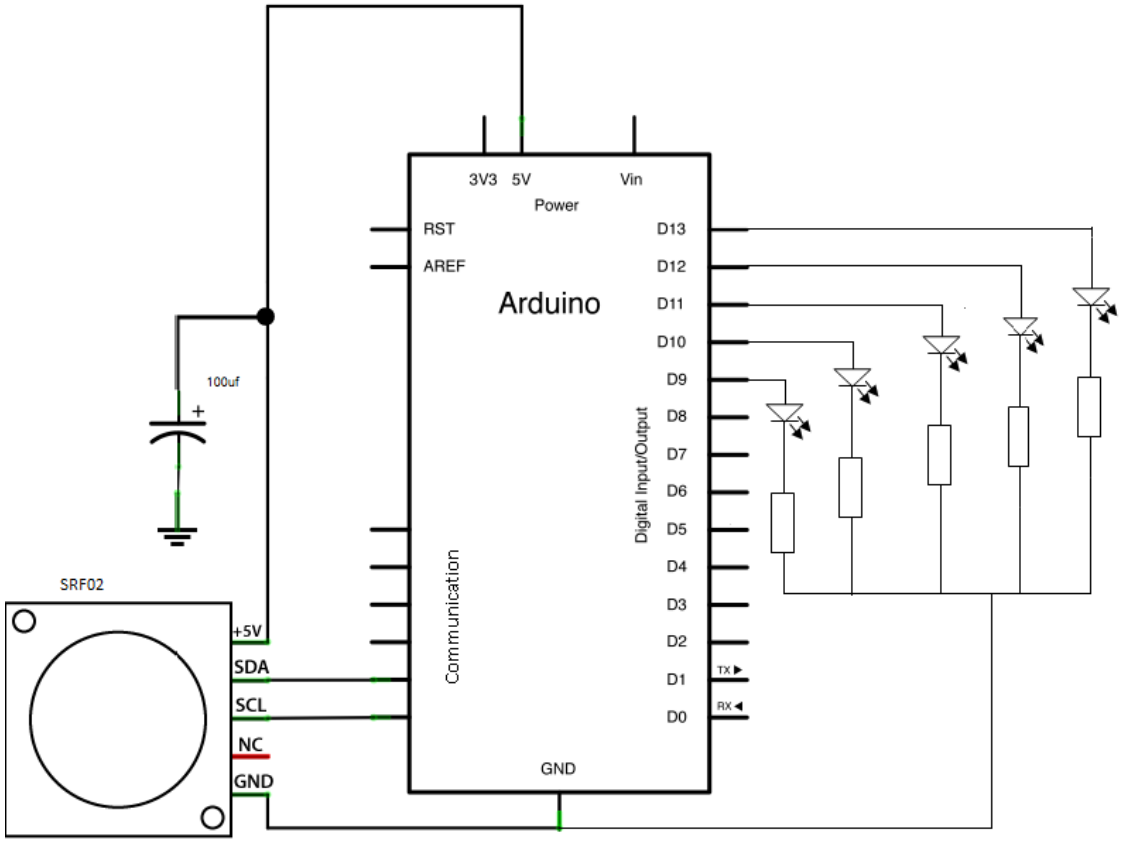


Figura 10. Conexión del Sensor1 (Entrada) al Arduino y Salidas (led).

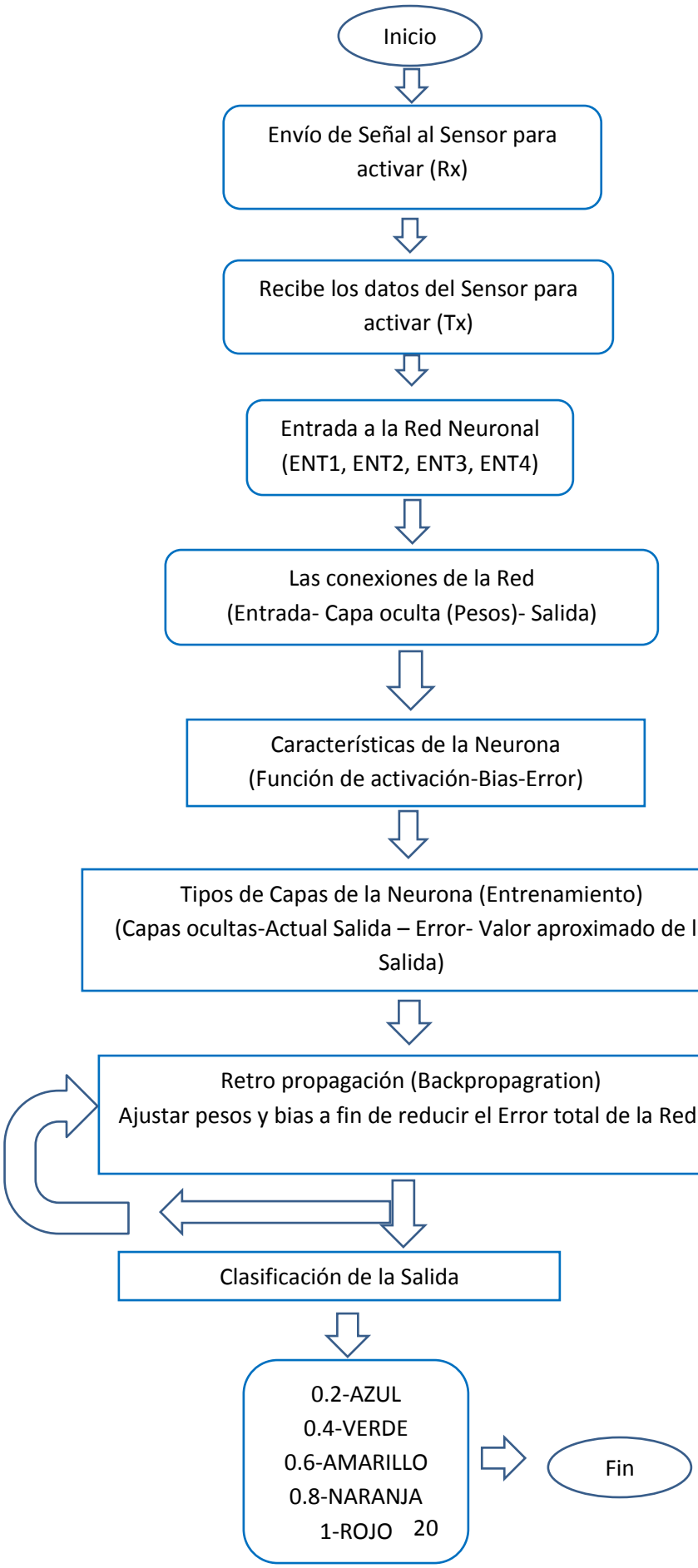


Figura 11. Diagrama de la Red Neuronal

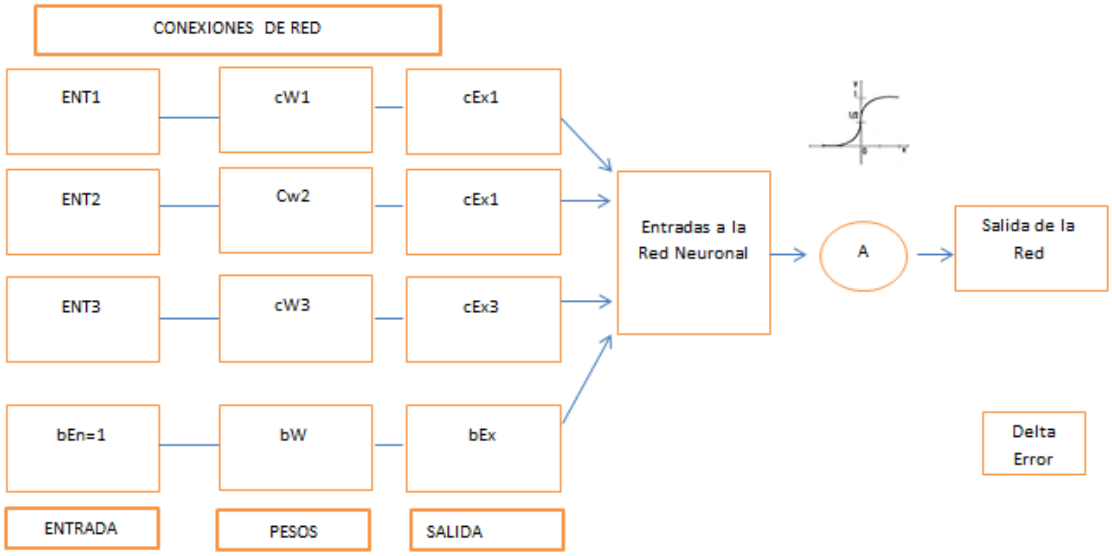


Figura 12. Esquema de la Red Neuronal (Conexión de Red)

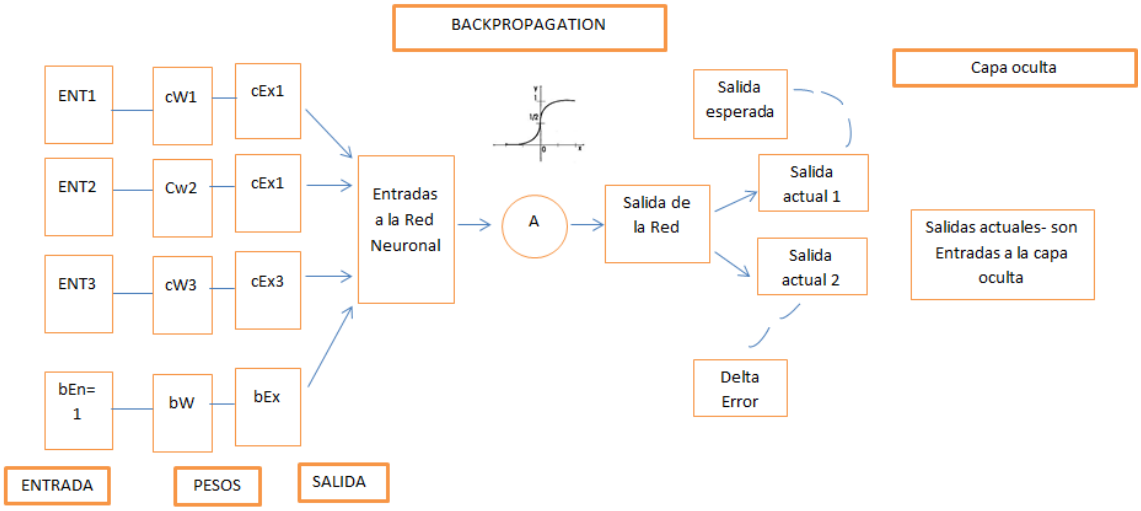


Figura 13. Esquema de la Red- Proceso de Retro propagación.

Nuestra red entrenada, con una 1 entrada de nuestro sensor, lo cual nos arroja los pesos, bias necesarios, así como los resultados esperados.

```
{u= [0 2 4 6 8];
net= newff ([0 1],[4 8 1],{'logsig','logsig','purelin'}, 'trainlm');
output=sim(net,u)
red.b{1}=0.5;
bias=red.b{1};
target = [1 0.8 0.6 0.4 0.2];
net.IW{1,1}
net.LW{2,1}
net = train(net,u,target);
output2 = sim(net,u)
plot(u,target,u,output2,'o')}
```

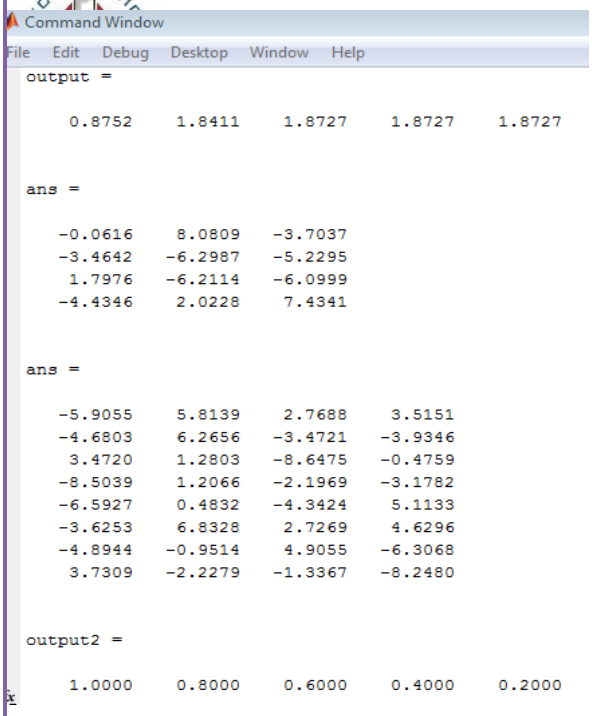


Figura 14. Entrenamiento de la Red Neuronal en Matlab

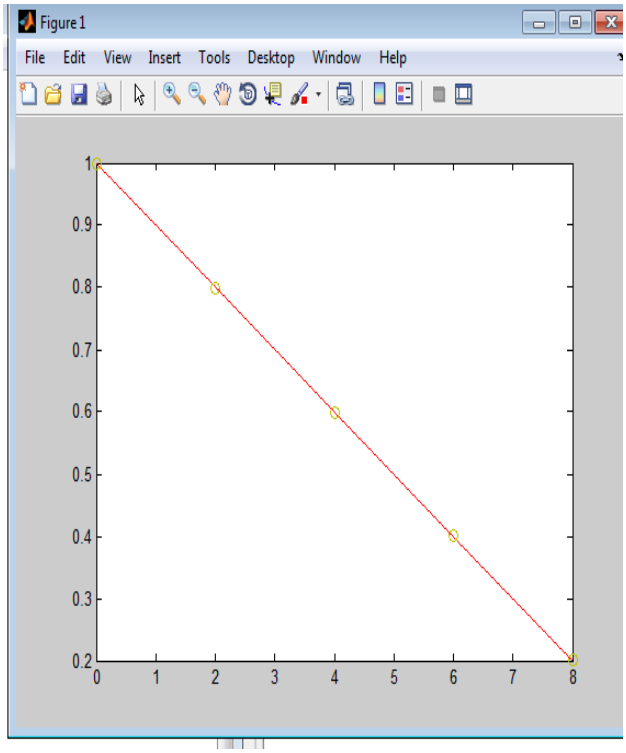


Figura 15. Grafica de la RNA. Línea Roja representa la entrada con respecto mi salida deseada. Y los círculos representan mi entrada con respecto a las salidas de la RNA, después de entrenarse

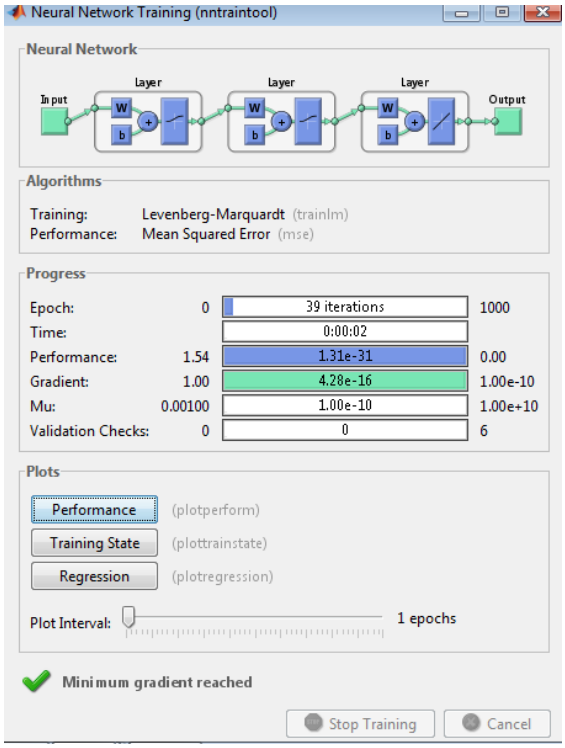


Figura 16. Datos anexos al entrenamiento de la Red.

Este fue el Entrenamiento realizado en Matlab, lo cual nos arroja en primero una salida sin entrenar con resultados lejos de nuestro objetivo. Sin embargo al entrenar la red con nuestra entrada y salida deseadas el programa da valores aleatorios con el fin de reducir el error de salida cada vez menos para que al final nuestra salida sea la que esperamos. Nuestra primera matriz de números nos indica los pesos en la capa oculta que obtuvo la red durante este entrenamiento, y la segunda matriz, nos indica que valores de bias obtuvo la red, así como las iteraciones necesarias para que el error de salida fuera menos. Obteniendo nuestra salida deseada.

Resultados

Estos resultados obtenidos, fueron mediante las siguientes condiciones:

Punto Crítico:	Caña Hueca
Altura (Nivel de Piso):	6 metros
1 Sensor:	SRF-02
1 Laptop	Gateway-LT27
1 Arduino	Mega 2560
5 Leds	Indicadores

Con base a los resultados obtenidos, que fueron lo más cercanos a nuestro objetivo, el error mínimo es casi 0, por lo que se aprecia que el resultado de la red llega a ser bastante preciso por sus capas ocultas. Se plantea hacer un ajuste de medida para el nivel de alerta, ya que el punto crítico donde se midió (Caña Hueca), solo consta de 6 Metros, sobre nivel de calle.

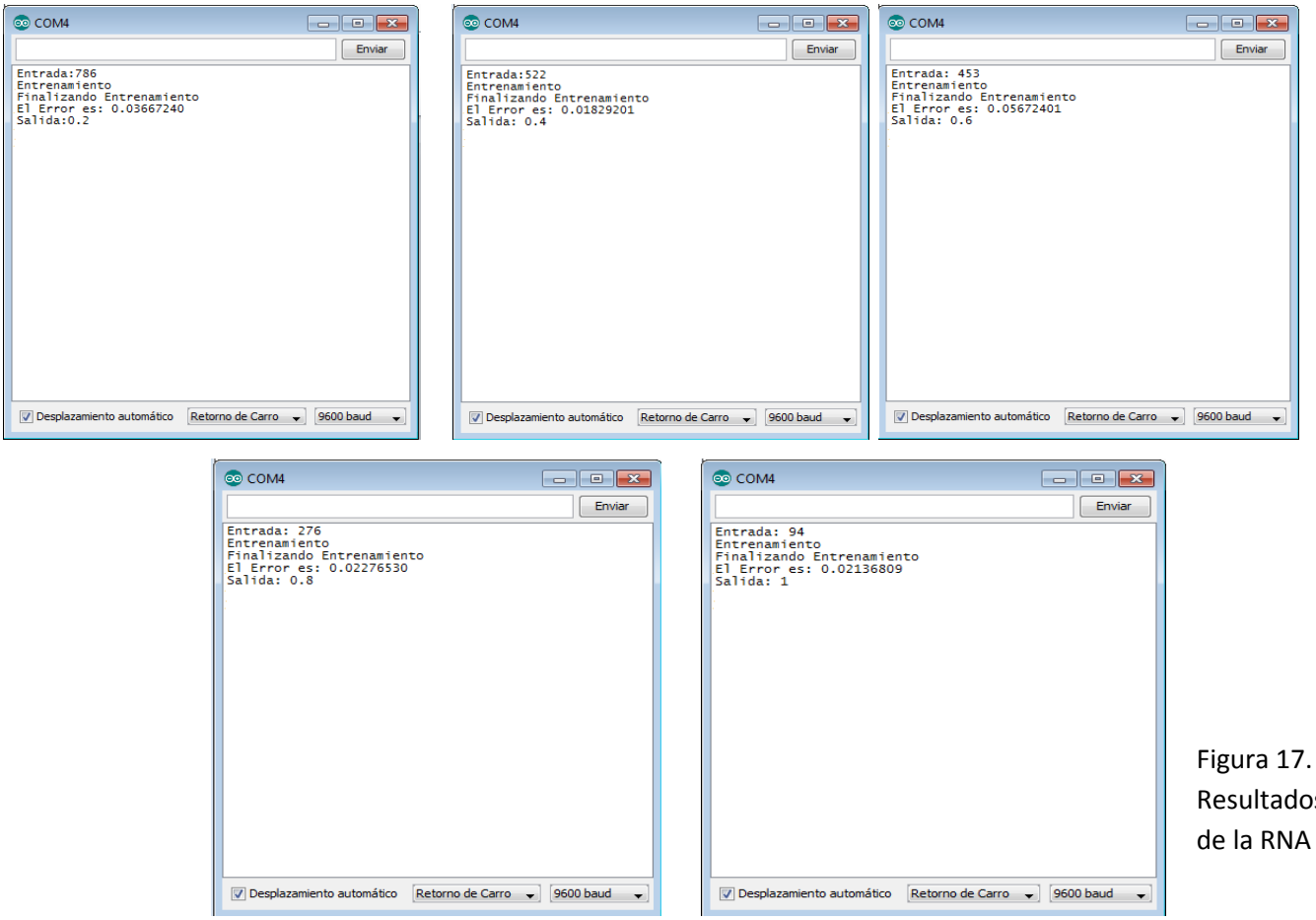
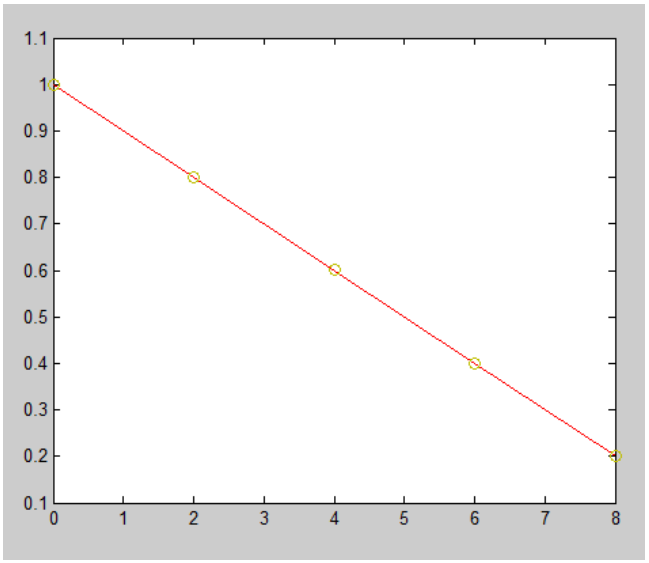


Figura 17. Resultados de la RNA

Además se vio la necesidad de introducir otros dos sensores al sistema de alerta temprana, pues de acuerdo a la naturaleza de la cuenca, los arboles obstruyen el paso del agua, arrojándolo a las orillas el cauce. Por lo cual se llegó hacer el entrenamiento sin llevar a prueba física.

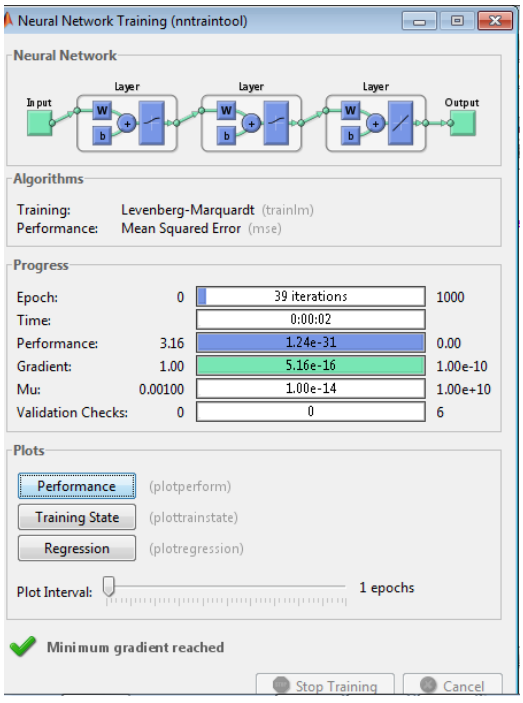
Entrenamiento con los 3 sensores de entrada:

```
u= [0 2 4 6 8; 0 2 4 6 8; 0 2 4 6 8];
net= newff ([0 1; 0 1; 0 1],[4 7 1],{'logsig','logsig','purelin'},
'trainlm');
output=sim(net,u)
red.b{1}=0.5;
bias=red.b{1};
target = [1 0.8 0.6 0.4 0.2];
net.IW{1,1}
net.LW{2,1}
net = train(net,u,target);
output2 = sim(net,u)
plot(u,target,u,output2,'o')
```




```
output =

   -1.4243   -1.0850   -1.0480   -1.0480   -1.0480


ans =

   -8.1264    1.1431   -3.4171
   -6.3608   -6.2055    0.2316
    6.3102    6.0323   -1.6778
    1.5802    2.4198   -8.4065


ans =

   -3.1884    5.4527    3.4357    5.5918
   -8.5836   -0.2082   -2.5126    1.7138
   -4.7951   -0.1630   -3.9185   -6.6781
   -5.6862   -3.5484   -2.1011   -5.7993
   -1.2715    6.1481   -6.2018   -2.2567
   -6.0580   -1.7589   -4.9507    4.3207
    4.2540   -4.1070    4.6695   -5.1189


output2 =

    1.0000    0.8000    0.6000    0.4000    0.2000
```

Figura 18. Datos del 2° Entrenamiento de la RNA con las 3 entradas.



Sin embargo en cada lugar a aplicarse el sistema, tendrá que ser ajustado dependiendo la altura (nivel de piso) para mayor precisión.

CONCLUSION

El Sistema de Alerta Temprana realizado es capaz de prevenir a la sociedad de posibles inundaciones cerca de la Cuenca Hidrológica "Sabinal", lo cual se probó en el entrenamiento con Matlab con salidas [0.2 0.4 .6 0.8 1] con respecto a la altura (nivel de piso) y lo cual se comprobó al hacer pruebas físicas obteniendo márgenes de error muy pequeño y aproximándonos a los resultados deseados de dicho entrenamiento. El sensor ultrasónico utilizado para determinar la altura (nivel de piso) presenta una característica estable y con buenos tiempos de respuesta. Esto facilita considerablemente al monitoreo de la Cuenca Hidrológica.

Sin embargo para elevar la calidad del mismo sistema, sería recomendable hacer un ajuste en las escalas de medición, pues la altura (nivel de piso) varía en los puntos críticos de la Cuenca Hidrológica, probarlo físicamente con los 3 sensores la Red Neuronal, así como ver los tiempos de monitoreo en temporada de lluvias, incluir pluviómetros, y estadísticas de puntos críticos.

BIBLIOGRAFIA:

- Centro Nacional de Prevención de Desastres, Secretaria de Gobernación. Inundaciones. Series Fascículos, 1° Edición, 2004.
- Centro Nacional de Prevención de Desastres, Secretaria de Gobernación. Guía de Prevención de Desastres. Series Fascículos, 3° Edición, 2013.
- Comisión Nacional del Agua, Gerencia de Protección a la Infraestructura y Atención de Emergencias, Plan de Emergencia de Inundación, Corrientes problemáticas, RIO SABINAL, Estado de Chiapas, 2009.
- T. Hagan Martin, B. Dcmuth Howard, Beale Mark. Neural Network Design, Thomson.
- http://www.accioncontraelhambre.org/centroamerica/phocadownload/guia_funcionamiento_y%20mantenimiento_%20sat.pdf
- http://www.aic.uniovi.es/ssii/P10/P10-RedesNeuronales.pdf
- http://www.csva.gob.mx/sah/Material/4NivelSAH.pdf

ANEXOS

Figura 19. Imagen actual de la Cuenca Hidrológica "Sabinal". (Parque Tuxtlan)

Figura 20. Imagen de la Cuenca Hidrológica "Sabinal" (Caña Hueca)

Figura 21. Imagen ancho (6° Poniente Norte)

Figura 22. Imagen altura (6° Poniente Norte)

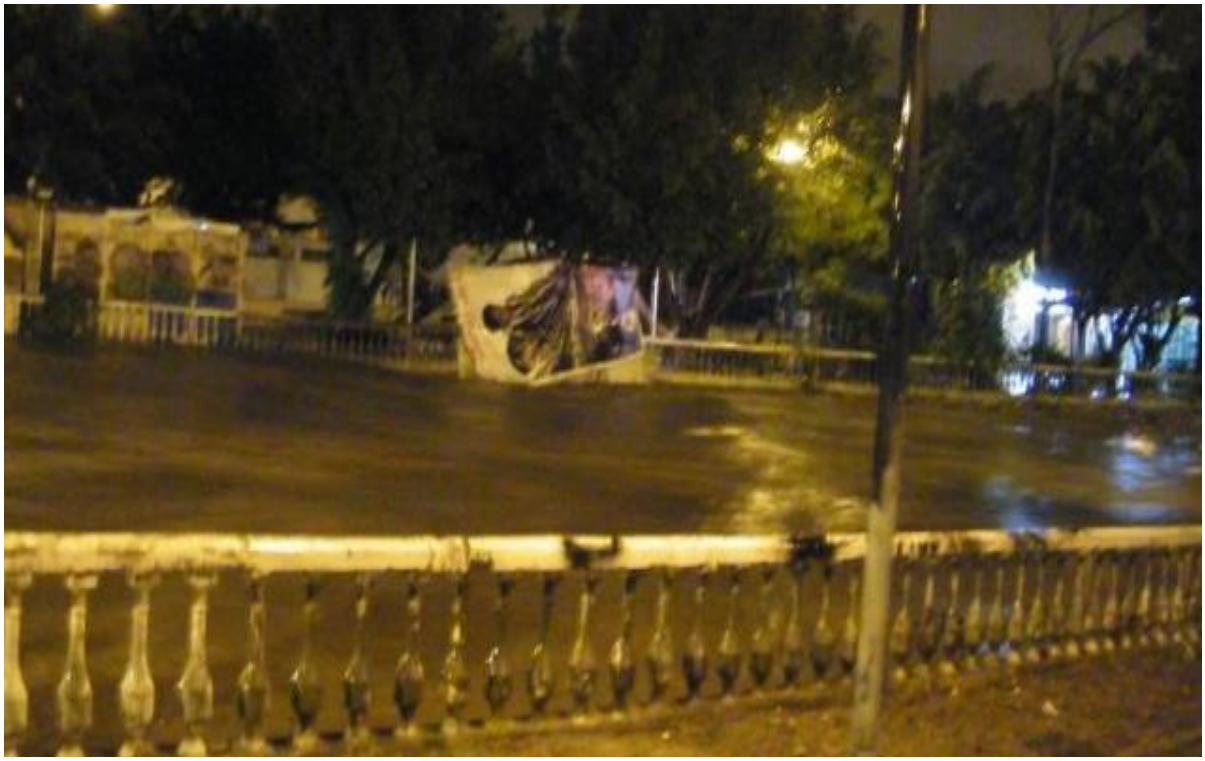


Figura 23. Inundación 2004 Tuxtla Gutiérrez.

Figura 24. Inundación Tuxtla Gutiérrez (4° Poniente norte)

Figura 25 .Inundación 2004 Centro de Tuxtla Gutiérrez

Figura 26. Afluente "El Poti" 2004.

Figura 27. Estado actual de alerta CONAGUA

Figura 28. Sistema de Medición CONAGUA (Caña Hueca)

Figura 29. Sistema de Alimentación (CONAGUA)

Figura 30. Centro de Carga (Sin utilización Actual)

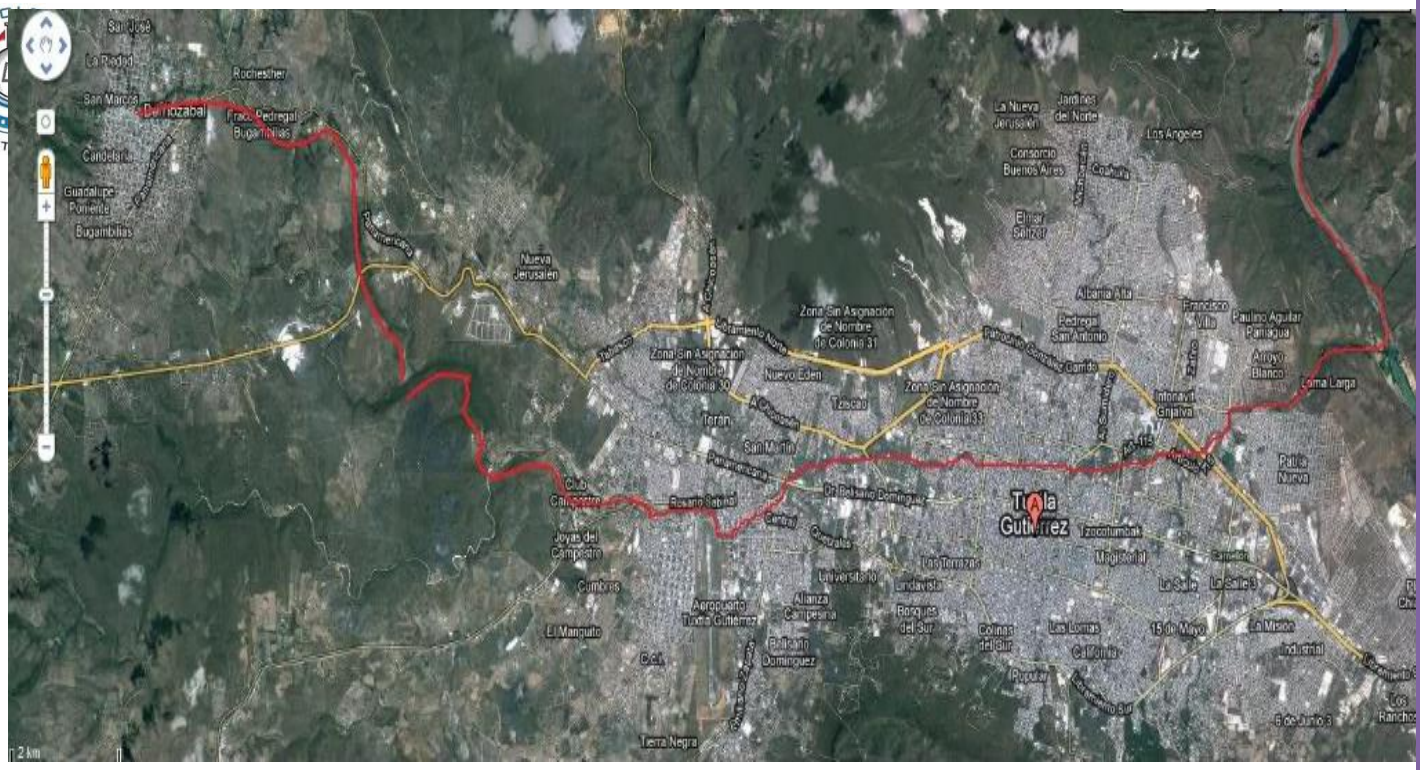


Figura 31. Inicio y Fin de la Cuenca Hidrológica "EL SABINAL"

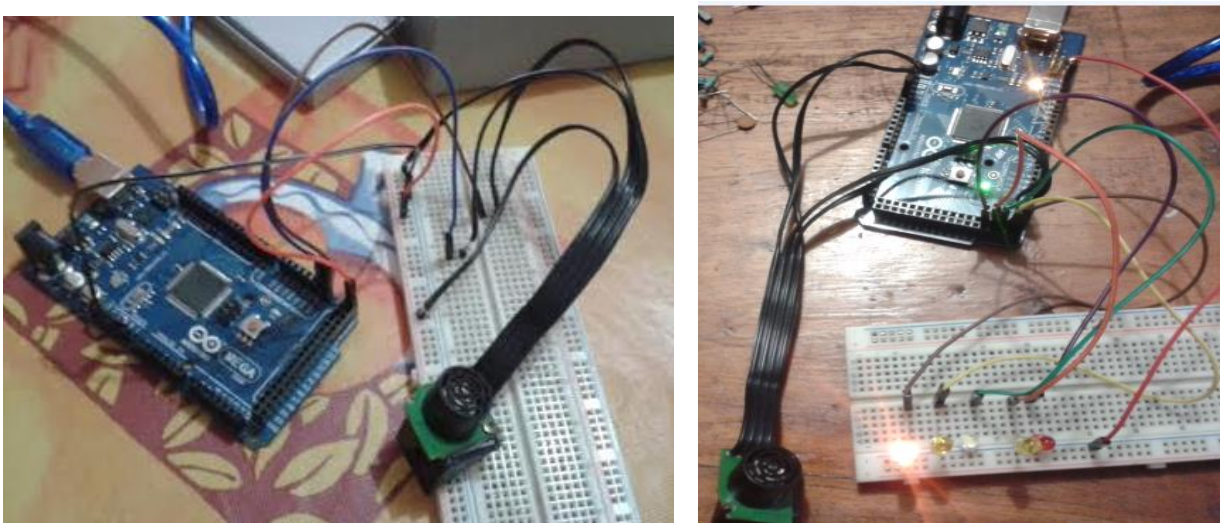


Figura 32. Conexión para Pruebas

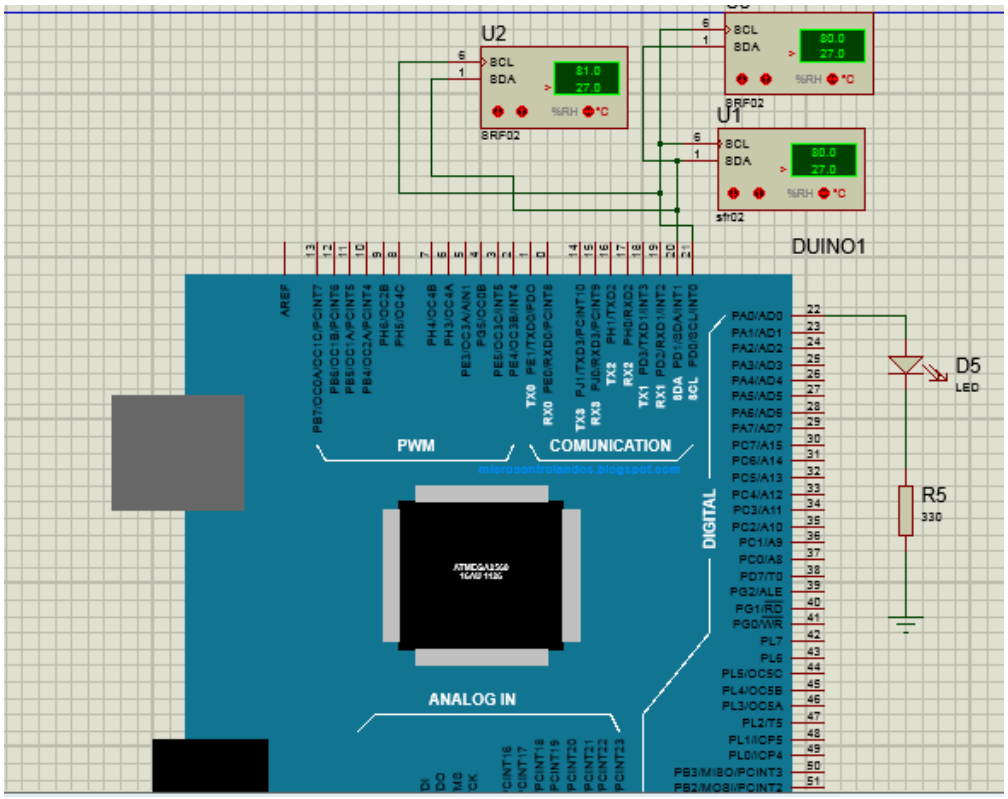


Figura 33. Conexión en Proteus – (Sensor de entrada I2C, salida)

Costo Del Sistema de Alerta Temprana

Cantidad	Nombre del Producto	Costo
1	Arduino Mega 2560	$400
3	Sensor SRF02	$445
1	Netbook Gateway LT27	$3500
30	Jumbo Leds	$3.50
TOTAL=		$5350

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

  int reading=0;
  int sar=22;

  myArray1[]= new myTrainingInput;
  myArray2[]= new myTrainingputput;

  float[] ENT1={};
  float[] ENT2={};
  float[] ENT3={};

  float[] SAL1={};

void setup()
{

  pinMode(sar, OUTPUT);

  Wire.begin();
  Serial.begin(9600);

  myArray1[]= new myTrainingInput;
  myArray2[]= new myTrainingputput;

  float[] ENT1={};
  float[] ENT2={};
  float[] ENT3={};

  float[] SAL1={};


 Serial.print("Datos:");
  Serial.print("-----------------------------------------");
  myTrainingInputs.add(ENT1);
  myTrainingOutputs.add(SAL1);
  Serial.print("Entrada= " + ENT1[] + ", " + ENT1[1] + "; Salida = " + SAL1[]);
  myTrainingInputs.add(ENT2);
  myTrainingOutputs.add(SAL1);
  Serial.print("Entrada= " + ENT2[] + ", " + ENT2[1] + "; Salida = " + SAL1[]);
  myTrainingInputs.add(ENT3);
  myTrainingOutputs.add(SAL1);
  Serial.print("Entrada= " + ENT3[] + ", " + ENT3[] + "; Salida  = " +SAL1[]);

  NeuralNetwork NN = new NeuralNetwork();
  NN.addLayer(2,2);
  NN.addLayer(2,1);

  Serial.print("Entrenamiento");
  float[] myInputDataA1={0,0};
```

```
  NN.processInputsToOutputs(myInputDataA1);
  float[] myOutputDataA1={};
  myOutputDataA1=NN.getOutputs();
  Serial.print("Feed Forward:  INPUT = 0; OUTPUT=" + myOutputDataA1[]);

  float[] myInputDataB1={};
  NN.processInputsToOutputs(myInputDataB1);
  float[] myOutputDataB1={};
  myOutputDataB1=NN.getOutputs();
  Serial.print("Feed Forward:  INPUT = 0,1; OUTPUT=" + myOutputDataB1[]);

  float[] myInputDataC1={};
  NN.processInputsToOutputs(myInputDataC1);
  float[] myOutputDataC1={};
  myOutputDataC1=NN.getOutputs();
  Serial.print("Feed Forward:  INPUT = 1,0; OUTPUT=" + myOutputDataC1[]);



  Serial.print("");
  Serial.print("-----------------------------------------");
  Serial.
  Serial.print("Entrenamiento");
  NN.autoTrainNetwork(myTrainingInputs,myTrainingOutputs,0.0001,500000);
  Serial.print("");
  Serial.print("Finalizando Entrenamiento");
  Serial.print("Prueba de la RNA");
  float[] myInputDataA2={0,0};
  NN.processInputsToOutputs(myInputDataA2);
  float[] myOutputDataA2={};
  myOutputDataA2=NN.getOutputs();
  Serial.print("Feed Forward:  INPUT = 0,0; OUTPUT=" + myOutputDataA2[]);

  float[] myInputDataB2={0,1};
  NN.processInputsToOutputs(myInputDataB2);
  float[] myOutputDataB2={};
  myOutputDataB2=NN.getOutputs();
  Serial.print("Feed Forward:  INPUT = 0,1; OUTPUT=" + myOutputDataA2[]);

  float[] myInputDataC2={};
  NN.processInputsToOutputs(myInputDataC2);
  float[] myOutputDataC2={};
  myOutputDataC2=NN.getOutputs();
  Serial.print("Feed Forward:  INPUT = 1,0; OUTPUT=" + myOutputDataC2[]);
}


void loop()
{
  Wire.beginTransmission(112);
  Wire.write(byte(0x00));
```

```
  Wire.write(byte(0x50));
  Wire.endTransmission();

  delay(70);


  Wire.beginTransmission(112);
  Wire.write(byte(0x02));
  Wire.endTransmission();
  Wire.requestFrom(112, 2);

  if(2 <= Wire.available())
  {
    reading = Wire.read();
    reading = reading << 8;
    reading |= Wire.read();
    Serial.print(reading);
  }

  delay(250);
}

////////////////////////////////////////////////////////////////////////////////////////////
class Conexion{
  float conent;
  float peso;
  float consal;

  Conexion(){
    randomisepesot();
  }
  Conexion(float temppeso){
    setpeso(temppeso);
  }
  void setpeso(float temppeso){
    peso=temppeso;
  }

  void randomisepeso(){
    setpeso(random(2)-1);
  }

  float calcConsal(float tempent){
    conent = tempInput;
    consal = conent * peso;
    return consal;
  }
}
 ////////////////////////////////////////////////////////////////////////////////////////////

class Neurona
```

```
{
 Conexion[] conexiones={};
 float bias;
 float neuronInputValue;
 float neuronOutputValue;
 float deltaError;

 Neurona()
 {

 Neurona(int numOfConexiones){
  randomiseBias();
  for(int i=0; i<numOfConexiones; i++){
   Conexion conn = new Conexion();
   addConnection(conn);
  }
 }

 void addConnection(Conexion conn){
    conexiones = (Conexion[]) append(conexiones, conn);
 }
 int getConnectionCount(){
    return connections.length;
 }

 void setBias(float tempBias)
{
  bias = tempBias;
 }
 void randomiseBias()
{
  setBias(random(1));
 }
 float getNeuronOutput(float[] connEntryValues){
  if(connEntryValues.length!=getConnectionCount()){
   Serial.print("Error no. Invalido");
   exit();
  }

  neuronInputValue=0;

  for(int i=0; i<getConnectionCount(); i++){
   neuronInputValue+=connections[i].calcConnExit(connEntryValues[i]);
  }

  neuronInputValue+=bias;
  neuronOutputValue=Activation(neuronInputValue);
  return neuronOutputValue;
 }

 float Activation(float x){
```

```
    float activatedValue = 1 / (1 + exp(-1 * x));
    return activatedValue;
  }

}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

class Layer{
  Neuron[] neurons = {};
  float[] layerINPUTs={};
  float[] actualOUTPUTs={};
  float[] expectedOUTPUTs={};
  float layerError;
  float learningRate;

  Layer(int numberConnections, int numberNeurons){

    for(int i=0; i<numberNeurons; i++){
      Neuron tempNeuron = new Neuron(numberConnections);
      addNeuron(tempNeuron);
      addActualOUTPUT();
    }
  }
  void addNeuron(Neuron xNeuron){
       neurons = (Neuron[]) append(neurons, xNeuron);
  }

  int getNeuronCount(){
    return neurons.length;
  }

  void addActualOUTPUT(){
     actualOUTPUTs = (float[]) expand(actualOUTPUTs,(actualOUTPUTs.length+1));
  }

  void setExpectedOUTPUTs(float[] tempExpectedOUTPUTs){
    expectedOUTPUTs=tempExpectedOUTPUTs;
  }

  void clearExpectedOUTPUT(){
      expectedOUTPUTs = (float[]) expand(expectedOUTPUTs, 0);
  }

  void setLearningRate(float tempLearningRate){
    learningRate=tempLearningRate;
  }

  void setInputs(float[] tempInputs){
    layerINPUTs=tempInputs;
  }
```

```
  void processInputsToOutputs()
{
    int neuronCount = getNeuronCount();

    if(neuronCount>0) {
      if(layerINPUTs.length!=neurons[0].getConnectionCount()){
        Serial.print("Error en el numero de conexiones de la capa");
        exit();
      } else {

        for(int i=0; i<neuronCount;i++){
          actualOUTPUTs[i]=neurons[i].getNeuronOutput(layerINPUTs);
        }
      }
    }else{
     Serial.print("Error el no. de capas");
      exit();
    }
  }
  float getLayerError(){
    return layerError;
  }

  void setLayerError(float tempLayerError){
    layerError=tempLayerError;
  }

  void increaseLayerErrorBy(float tempLayerError){
    layerError+=tempLayerError;
  }

void setDeltaError(float[] expectedOutputData){
    setExpectedOUTPUTs(expectedOutputData);
    int neuronCount = getNeuronCount();
    setLayerError(0);
     for(int i=0; i<neuronCount;i++){
       neurons[i].deltaError = actualOUTPUTs[i]*(1-
actualOUTPUTs[i])*(expectedOUTPUTs[i]-actualOUTPUTs[i]);


       increaseLayerErrorBy(abs(expectedOUTPUTs[i]-actualOUTPUTs[i]));
     }
  }

  void trainLayer(float tempLearningRate)
  {
    setLearningRate(tempLearningRate);

    int neuronCount = getNeuronCount();

    for(int i=0; i<neuronCount;i++){
```

```
      neurons[i].bias += (learningRate * 1 * neurons[i].deltaError);

      for(int j=0; j<neurons[i].getConnectionCount(); j++){
         neurons[i].connections[j].weight += (learningRate *
neurons[i].connections[j].connEntry * neurons[i].deltaError);
      }
    }
  }
}

////////////////////////////////////////////////////////////////////////////////////////////////

class NeuralNetwork
{
  Layer[] layers = {};
  float[] arrayOfInputs={};
  float[] arrayOfOutputs={};
  float learningRate;
  float networkError;
  float trainingError;
  int retrainChances=0;

  NeuralNetwork()
  {
    learningRate=0.1;
  }
  void addLayer(int numConnections, int numNeurons){
    layers = (Layer[]) append(layers, new Layer(numConnections,numNeurons));
  }

  int getLayerCount(){
     return layers.length;
  }
  void setLearningRate(float tempLearningRate){
    learningRate=tempLearningRate;
  }


  void setInputs(float[] tempInputs){
    arrayOfInputs=tempInputs;
  }

  void setLayerInputs(float[] tempInputs, int layerIndex){
    if(layerIndex>getLayerCount()-1){
      Serial.print("Error en la Entrada de Capas" + layerIndex + " Limite excedido " +
(getLayerCount()-1));
    } else {
      layers[layerIndex].setInputs(tempInputs);
    }
  }
```

```
 void setOutputs(float[] tempOutputs){
  arrayOfOutputs=tempOutputs;
 }

 float[] getOutputs(){
  return arrayOfOutputs;
 }

 void processInputsToOutputs(float[] tempInputs){
  setInputs(tempInputs);

  if(getLayerCount()>0){
   if(arrayOfInputs.length!=layers[0].neurons[0].getConnectionCount()){
    Serial.print("Error en no. de capas no coincide");
    exit();
   } else {

    for(int i=0; i<getLayerCount(); i++){

     if(i==0){
      setLayerInputs(arrayOfInputs,i);
     } else {
      setLayerInputs(layers[i-1].actualOUTPUTs, i);
     }

     layers[i].processInputsToOutputs();
    }
    setOutputs(layers[getLayerCount()-1].actualOUTPUTs);
   }
  }else{
   Serial.print("Error no hay capas en la Red");
   exit();
  }
 }
 void trainNetwork(float[] inputData, float[] expectedOutputData){

  processInputsToOutputs(inputData);

  for(int i=getLayerCount()-1; i>-1; i--){
   if(i==getLayerCount()-1){
    layers[i].setDeltaError(expectedOutputData);
    layers[i].trainLayer(learningRate);
    networkError=layers[i].getLayerError();
   } else {
    for(int j=0; j<layers[i].getNeuronCount(); j++){

     layers[i].neurons[j].deltaError=0;
     for(int k=0; k<layers[i+1].getNeuronCount(); k++){
      layers[i].neurons[j].deltaError += (layers[i+1].neurons[k].connections[j].weight *
layers[i+1].neurons[k].deltaError);
     }
```

```
          layers[i].neurons[j].deltaError *= (layers[i].neurons[j].neuronOutputValue * (1-
layers[i].neurons[j].neuronOutputValue));
        }

        layers[i].trainLayer(learningRate);
        layers[i].clearExpectedOUTPUT();
      }
    }
  }

 void trainingCycle(ArrayList trainingInputData, ArrayList trainingExpectedData,
Boolean trainRandomly){
      int dataIndex;

      trainingError=0;
       for(int i=0; i<trainingInputData.size(); i++){
        if(trainRandomly){
          dataIndex=(int) (random(trainingInputData.size()));
        } else {
          dataIndex=i;
        }

        trainNetwork((float[]) trainingInputData.get(dataIndex),(float[])
trainingExpectedData.get(dataIndex));
        trainingError+=abs(networkError);
      }
  } else
   {
      Serial.print("No se ah conseguido seguir readaptando " + retrainChances + " El
Error es: " + trainingError);
    }
  }
}
```