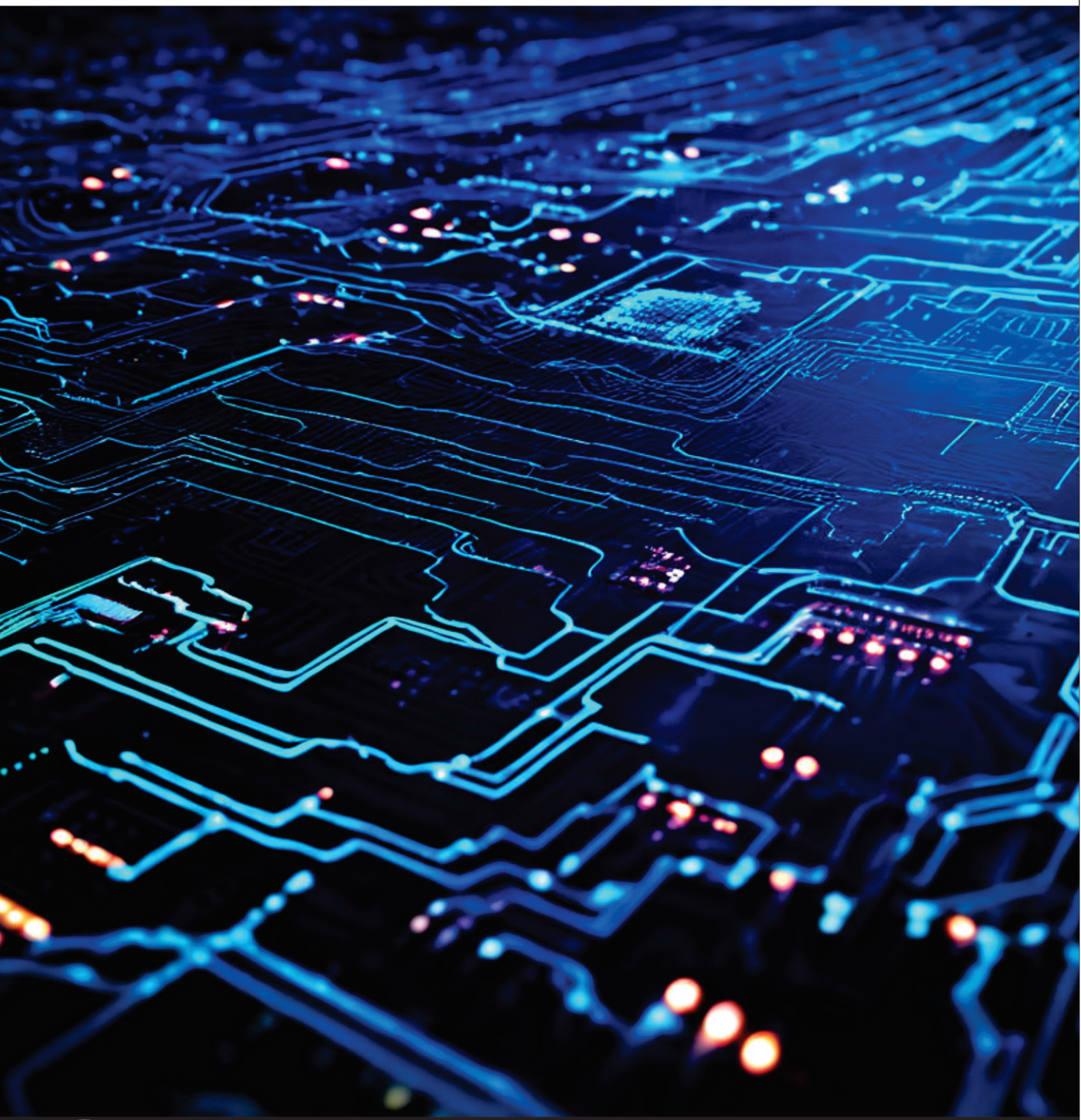
Informaticae Abstracta

Año 2024, Vol. 2, No. 2 JUL | DIC 2024



Universidad Autónoma del Estado de México

Informaticae Abstracta
Año 2024, Vol. 2, No. 2 JUL | DIC 2024

Informaticae Abstracta, año 2024, volumen 2, número 2, Julio – Diciembre 2024, es una revista de publicación semestral editada por la Universidad Autónoma del Estado de México, Instituto Literario 100 Ote., Col. Centro, Toluca, Estado de México, C.P. 50000, Tels. (722) 2140855 y 21140795 ext. 10 1217, http://informaticae.uaemex.mx, informaticae@uaemex.mx, Editora Responsable: Ruth Hernández Pérez, Facultad de Ingeniería. Reserva de Derechos al Uso Exclusivo número 04-2022-111113523100-102, ISSN: en trámite, ambos otorgados por el Instituto Nacional del Derecho de Autor. Responsable de la última edición Mercedes Lucero Chávez, de la Facultad de Ingeniería, Cerro de Coatepec s/n, Ciudad Universitaria, C.P. 50100, Toluca, Estado de México, fecha de última modificación, diciembre 2024.

Informaticae Abstracta
Año 2024, Vol. 2, No. 1 ENE | JUN 2024

Índice

Informaticae Abstracta
Año 2024, Vol. 2, No. 2, pp. 4-19 JUL | DIC 20244

Sistema de control mediante una red neuronal en motores de corriente directa

Adrian Camacho-Ramírez*; Marco Antonio Sánchez-Carmona

Información del Artículo

Recibido: 12-oct-2024
Aceptado: 17-dic-2024

Palabras clave: Control automático, red neuronal, Motor DC, ANN, PID

Resumen

Este estudio investiga la aplicación de redes neuronales artificiales en el control de velocidad de motores de corriente continua, con el objetivo de mejorar el rendimiento y la adaptabilidad en comparación con los controladores PID convencionales. La red neuronal artificial fue entrenada para ajustar dinámicamente la señal de control, logrando un seguimiento preciso de la velocidad sin necesidad de un modelo matemático detallado de la dinámica del motor, mostrando una alternativa robusta frente al control PID tradicional, el cual suele requerir un ajuste exhaustivo como su análisis de estabilidad y ajuste de las ganancias del controlador. Los resultados de simulación y experimentación confirman que el controlador basado en redes neuronales artificiales pueden mantener de manera adaptativa y precisa la velocidad deseada del motor.

1. Introducción

El control automático ha sido de gran importancia en la trasformación y evolución del ser humano debido a la constante necesidad de optimizar sus procesos o actividades en el menor tiempo posible, de manera eficiente y segura. Además, el control automático se volvió una parte importante e integral de los procesos modernos industriales y de manufactura. La automatización de procesos ha experimentado una evolución constante en los últimos años, provocando transformaciones significativas en nuestro país. En este contexto, es fundamental desarrollar procesos innovadores que modernicen y optimicen nuestros sistemas de producción automatizada, asegurando su competitividad y adaptación a las demandas tecnológicas actuales.

Los motores de corriente continua (CC) son fundamentales en la ingeniería de control y se utilizan ampliamente en aplicaciones domésticas, industriales y de investigación. Su gran versatilidad radica en su capacidad de control, ya que permiten ajustar con facilidad la velocidad, la posición y el par simplemente modificando el voltaje o la intensidad de corriente. Estas características los convierten en una opción ideal para el control de una amplia variedad de sistemas automatizados [1], [2], [3].

Aunque los motores de corriente alterna (CA), en particular los asíncronos, han ganado terreno debido a su control eficiente y costos más accesibles para los consumidores industriales [4], los motores de CC siguen siendo esenciales en aplicaciones que demandan alta potencia y precisión. Ejemplos de esto incluyen trenes, máquinas de precisión y micromotores, donde su control preciso y confiabilidad los hacen indispensables.

Existen varios métodos para controlar motores CC, una técnica común es la modulación por ancho de pulso (PWM) que es utilizado como señal de control para los controladores lineales a lazo cerrado, como los controladores PID (Proporcional-Integral-Derivativo); este último comúnmente regula el voltaje aplicado al motor, lo que resulta en un control eficiente y preciso de la velocidad. Por otro lado, los controladores PID se utilizan para mantener la velocidad o la posición deseada, compensando cualquier diferencia entre la salida real y el valor objetivo (setpoint). Esta capacidad de retroalimentación permite ajustar de manera continua la entrada al motor para corregir cualquier desviación en el comportamiento.

El uso de controladores lineales han dominado el control de motores CC, pero presentan limitaciones frente a sistemas no lineales, a la variación de parámetros y entornos cambiantes. Como resultado, la eficacia del control puede disminuir y el error aumentar bajo estas condiciones. Para abordar estos problemas, se pueden utilizar controladores robustos y no lineales, que son capaces de mejorar el rendimiento del sistema frente a dichas variaciones. Entre las técnicas de control utilizadas para resolver estos problemas podemos mencionar lógica difusa [5], control por modos deslizantes [6], aprendizaje iterativo [7], métodos de ajuste basados en inteligencia artificial [8]. La implementación de técnicas de aprendizaje profundo (deep learning), que son un conjunto de algoritmos de aprendizaje automático que intentan modelar los datos mediante el uso de componentes denominados *redes neuronales artificiales* (ANN, por sus siglas en inglés), para el control de motores CC son utilizados en [9], así como el uso de Matlab Simulink para el uso de ANN [10]. El principal componente de las ANN, lo encontramos con el término perceptrón, siendo este la ANN más antigua, y aun la más famosa [11]. En la Figura 1 se muestra la arquitectura de un perceptrón, donde se observan valores de entrada $x_1, x_2, \dots, x_n$ y un valor de salida $\overline{y}$; los valores de entrada vienen dados por la representación del conjunto de datos de entrenamiento y el valor de salida es la respuesta esperada para cada una de las muestras del conjunto de datos; para cada entrada x_i se asocia un peso w_i. El valor de salida de la neurona se calcula mediante la suma de los productos de las entradas por sus respectivos pesos, los cuales llegan mediante las conexiones de entrada aplicando una función de activación $\sigma(z)$, la cual dado un valor esta la transforma a un nuevo valor.

Si bien, el término ANN evoca la idea de máquinas que funcionan como cerebros y puede traer a la mente connotaciones de ciencia ficción, pero siendo más precisos en el término, una ANN es un conjunto interconectado de elementos de procesamiento simples, unidades o nodos, cuya funcionalidad se basa, de manera aproximada, en la neurona biológica. La capacidad de procesamiento de la red se almacena en las fortalezas de conexión entre las unidades, o pesos, que se obtienen a través de un proceso de adaptación o aprendizaje a partir de un conjunto de patrones de entrenamiento [12]. Las ANN tienen la capacidad de resolver problemas complejos en tiempo real, ya que funcionan mediante un entrenamiento previo (aprendizaje supervisado) y adaptarse continuamente a nuevas condiciones del problema. Una ventaja importante es que no es necesario desarrollar un

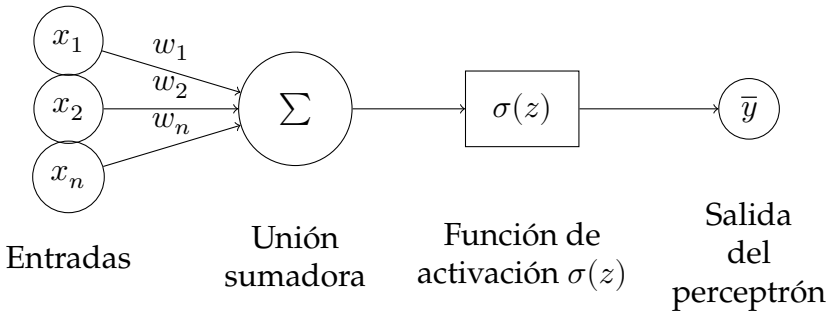


Figura 1: Estructura de un perceptrón

modelo a priori [13]. En la Figura 2 se puede observar una arquitectura básica de una ANN con tres capas; una capa de entrada con dos neuronas, una capa oculta con tres neuronas y una capa de salida con una única neurona.

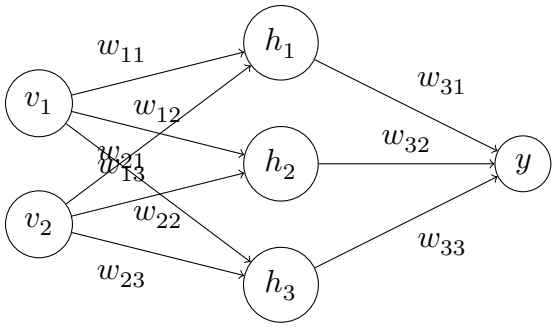


Figura 2: Arquitectura de una red neuronal artificial

Las ANN son entre todos los métodos disponibles, las más adecuadas para el reconocimiento de patrones en tiempo real, ya que operan en paralelo y actualizan todas sus instancias simultáneamente. Es fundamental señalar que esta característica se maximiza al implementar redes en hardware diseñado específicamente para el procesamiento paralelo.

Para el control automático de diferentes propósitos también se pueden utilizar ANN, incluyendo el control de fuerza para robots manipuladores [14], [15], control de postura y balanceo para robots bípedos [16], control de trayectoria para vehículo submarinos [17], para estimar la velocidad de vehículos longitudinales para aplicaciones dinámicas de control [18] y controles híbridos de posición y fuerza para controles adaptables para robots manipuladores como en [19].

Este trabajo de investigación explora la aplicación de ANN para el control de velocidad de un motor de CC, proponiendo una alternativa a los métodos tradicionales que requie-

ren un análisis exhaustivo para su diseño. El uso de esta tecnología es especialmente relevante en países emergentes como México. Las ANN se entrenarán utilizando tanto datos de simulaciones del sistema motor-control como resultados experimentales, y se espera que el control basado en ANN, entrenado para predecir el valor de control en función del error de velocidad, mejore la capacidad adaptativa y la robustez frente a perturbaciones en comparación con los controladores clásicos.

2. Materiales y Métodos

Los componentes utilizados, especificaciones y características se describen en las siguientes secciones, así como la arquitectura y componentes determinados para la ANN propuesta. También se detalla la estructura de los datos utilizados para la etapa de entrenamiento y pruebas tanto en simulación como los valores reales.

DC Motor

Se utiliza un Rotary Servo (RS), ver Figura 3, el cual es una plataforma intuitiva para experimentos de control de posición angular (grados) y control de velocidad angular (rpm, revoluciones por minuto). Es ideal para introducir conceptos y teorías de control básicos, el cual fue modificado para la adquisición de datos mediante un microcontrolador de bajo costo (Arduino Due).

Figura 3: Rotary Servo

La unidad base del RS es un sistema de servomecanismo con engranajes. Un motor CC impulsa un piñón pequeño, fijado a un engranaje de mayor diámetro que gira en torno al eje de interés. La posición angular del eje de interés se mide mediante un codificador óptico de alta resolución o encoder, ver Tabla 1. El encoder también se utiliza para estimar la velocidad angular del motor.

Tabla 1: Especificaciones del motor DC

Dimensiones	15cm x 15 cm x 18 cm
Masa	1.2 kg
Voltaje nominal	6 V
Corriente máxima	1 A
Resolución del enconder	4096 ppr (pulsos por revolución)
Velocidad máxima	63 rpm
Caja reductora	70:1

Arduino Duo

Arduino Due es una placa de desarrollo electrónica que se distingue por su potente microcontrolador SAM3X8E, basado en el núcleo ARM Cortex®-M3. Es una de las placas Arduino más avanzadas y potentes disponibles en el mercado. Con una velocidad de reloj de hasta 84 MHz y una generosa cantidad de memoria flash y RAM, el Arduino Due ofrece un rendimiento significativamente mejorado en comparación con las placas Arduino más antiguas. Esta placa es especialmente adecuada para proyectos que requieren un procesamiento de datos más rápido y una capacidad de cálculo más avanzada, como aplicaciones de control, robótica y procesamiento de señales [20].

Modulo controlador de motores L298N

El L298N es un circuito integrado comúnmente utilizado para controlar motores, actuando como interfaz entre los motores y una placa microcontroladora. Similar a otros controladores de la serie L293, el L298N está diseñado para manejar dos motores de manera simultánea. Este IC cuenta con dos puentes H (H-bridges) y tiene 15 pines, ofreciendo un mayor manejo de corriente que el L293D, lo que lo hace ideal para proyectos que requieren motores más potentes o mayores exigencias de control [21].

El driver L298N, conectado al motor, permite controlar tanto el sentido de giro como la velocidad angular a través de una entrada PWM. La señal de excitación (u) del motor depende del valor de PWM generado por el microcontrolador (Arduino Due), el cual tiene una resolución de 8 bits [0 – 255]. En este caso, el valor 255 corresponde al voltaje máximo aplicado al motor.

Arquitectura de la ANN

La ANN propuesta consta de 3 capas, en la primera capa de entrada se determinaron 10 neuronas con una entrada de 2 datos (x_1, x_2), la segunda capa, denominada oculta,

esta configurada igualmente por 10 neuronas y una tercera capa de salida es añadida a la arquitectura, esta capa consta únicamente de una neurona para dar la salida $\overline{y}$.

Función de activación

Para la capa 1 y 2 se estableció la función de activación ReLu (Rectified Linear Unit), siendo unas de las funciones de activación más comunes y utilizadas, debido a su efectividad en la aceleración del entrenamiento y en la mejora del rendimiento en la mayoría de los casos, la expresión matemática de esta función se muestra en la Ecuación 1.

$$ReLU(x) = \text{máx}(0, x) \tag{1}$$

La función ReLu devuelve el valor de entrada si es positivo y cero en caso contrario, por lo que hace que el cálculo sea eficiente en comparación con otras funciones de activación como la sigmoide o tangente hiperbólica que a diferencia de estas aplicada a la ANN, mejora la convergencia ya que se evita el problema del del desvanecimiento del gradiente, además su eficiencia computacional permite entrenar ANN de gran cantidad de capas y neuronas [22]. En la Figura 4 podemos observar el comportamiento de la función de activación ReLu, utilizada para la ANN propuesta en este trabajo.

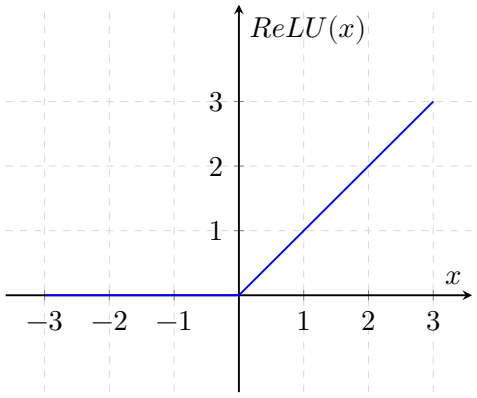


Figura 4: Gráfica de la función de activación ReLu

Optimizador

El optimizador seleccionado para la ANN es el algoritmo ADAM (Adaptative Moment Estimation), que es un algoritmo de optimización que mejora el proceso de aprendizaje de un modelo, debido a que utiliza una estimación del momento y de la magnitud de los gradientes anteriores para actualizar los parámetros, con esto ADAM adapta la tasa de aprendizaje de cada parámetro individualmente en función de su estimación del momento y la magnitud del gradiente [23]. Las principales ventajas del algoritmo ADAM conllevan a tener una adaptación de la tasa de aprendizaje, ya que ajusta dicha tasa de

cada parámetro individualmente con lo que se obtienen una mejor eficiencia, además utiliza el concepto de "corrección de sesgos"para evitar sesgos en los primeros pasos. En las Ecuaciones 2, 3 y 4 se expresan la forma de actualización de parámetros de ADAM, siendo m_t el promedio móvil de los gradientes, v_t el promedio móvil del cuadrado de los gradientes, β_1, β_2 factores de decaimiento exponencial y ϵ siendo un valor pequeño para evitar divisiones por 0.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{2}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{3}$$

$$\theta = \theta - \eta \frac{m_t}{\sqrt{v_t} + \epsilon} \tag{4}$$

Función de pérdida

La función de pérdida o costo establecida para la ANN es el Error Cuadrático Medio (MSE, por sus siglas en inglés), que puede definirse en la Ecuación 5, la ventaja de utilizar MSE para la ANN radica en que penaliza los errores más grandes que los pequeños debido a la naturaleza cuadrática de la función, con lo que prioriza la corrección donde hay grandes discrepancias entre la predicción y los valores reales [24].

$$mse = \frac{1}{n} \sum (y_n - \overline{y})^2 \tag{5}$$

Configuración para el entrenamiento

Para el entrenamiento se estableció una cantidad de épocas de $1,000$, utilizando el 80 % del conjunto de datos. Al utilizar el optimizar ADAM, se dejó la tasa de aprendizaje en $0{,}001$, dado el funcionamiento de este optimizador, la tasa de aprendizaje η se autoajusta como se observó en la Ecuación 4.

Estructura de los datos

Los datos utilizados, tanto para la etapa de entrenamiento como pruebas, se conforman de un dataset que contiene: Tiempo (medido en milisegundos), setpoint, valores de RPM, una valor de error y PWM, cada uno distribuido en una columna, los valores de tiempo, RPM y el error están almacenados como valores de punto flotante, por otro lado setpoint y PWM son valores enteros positivos. El total de registros utilizados se compone de 620 registros que representan los datos trabajados por un tiempo de 60 segundos (con una holgura). De este conjunto de datos, se utilizaron las RPM, el error como elementos de entrada X, mientras que PWM se utilizó como la salida deseada Y.

3. Implementación

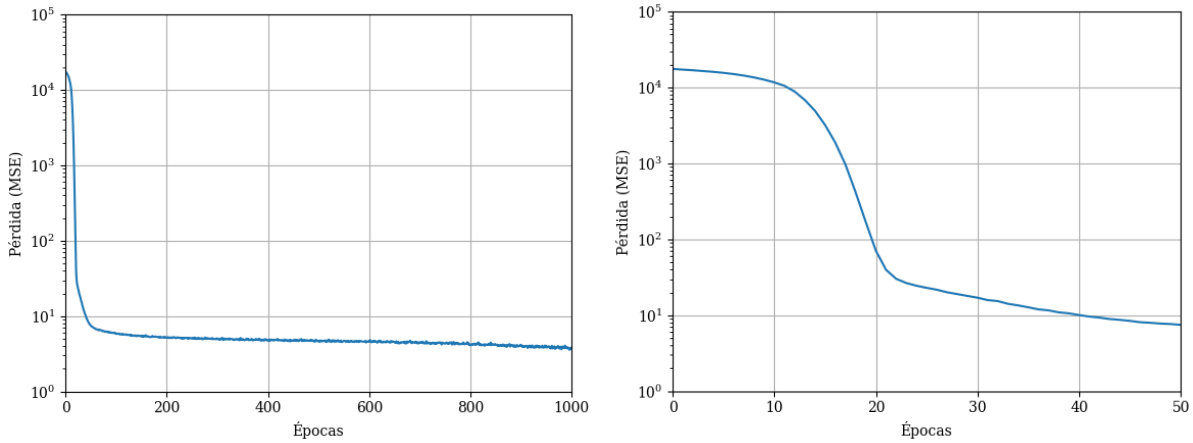
La ANN se implementó mediante la biblioteca Keras, siendo esta una biblioteca de redes neuronales de código abierto para el lenguaje Python, la ANN se creó mediante las configuraciones mencionadas, en la Tabla 2 se resume la configuración de la ANN utilizada.

Tabla 2: Resumen de la configuración de la ANN

Número de capas	3
Neuronas en la capa 1 y 2	10
Función de activación	ReLu
Optimizador	ADAM
Tasa de aprendizaje	0.001
Número de épocas	1000

Comportamiento de la ANN durante el entrenamiento

Después de la etapa de entrenamiento de la ANN propuesta, se muestra en la Figura 5 la gráfica de la función de pérdida MSE por cada época de entrenamiento, es decir, desde 1 hasta 1000 épocas. En la gráfica de la Figura 5a se observa el descenso de la función antes de llegar a la época número 200, en la gráfica de la Figura 5b se observa más a detalle este descenso en las primeras 50 épocas, ya que la curva empieza a converger con el eje X. El valor mínimo obtenido de MSE es de $0{,}0078$ que demuestra que la ANN tienen un margen de error bajo para las predicciones.



(a) Pérdida durante las 1000 épocas (b) Pérdida durante las primeras 50 épocas

Figura 5: Gráficas de la función de pérdida MSE durante el entrenamiento

Simulaciones con control ANN

Una vez entrenada, la ANN puede predecir la salida del motor en función de diferentes entradas, lo que facilita la toma de decisiones en tiempo real. La ANN puede ajustar las señales de control, minimizando el error entre la velocidad deseada y la real, mejorando así la precisión y el rendimiento del sistema.

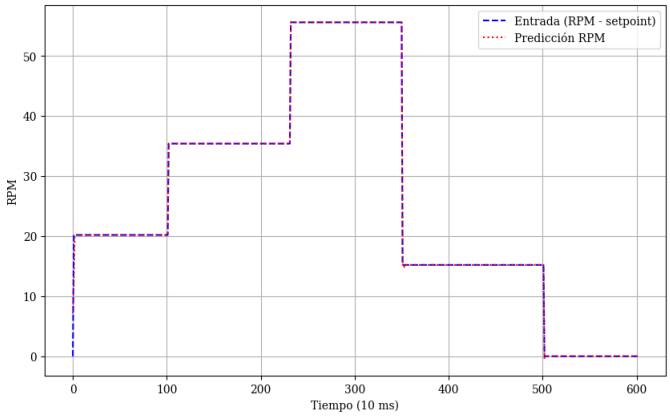


Figura 6: Predicción de la ANN para control de velocidad

Implementación con controlador PID

Para implementar el control PID, es fundamental determinar la función de transferencia del motor. Esto se logra excitando el motor con un voltaje específico y registrando su respuesta ante esta entrada. Los resultados obtenidos, que reflejan el comportamiento del sistema, se presentan en la gráfica de la Figura 7.

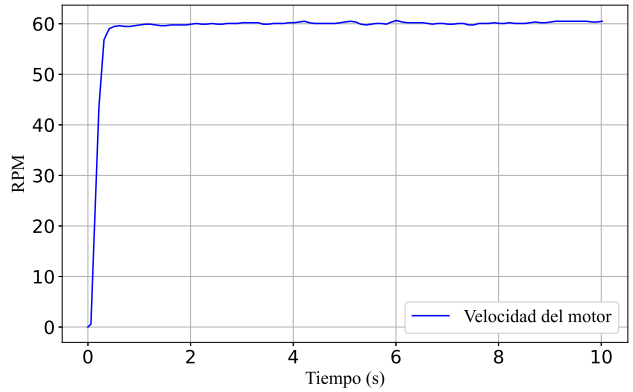


Figura 7: Curva de velocidad del motor a (PWM=255)

Los datos de entrada (voltaje) y de salida (RPM) fueron recolectados, y a través del método de System Identification en Matlab®, se obtuvo una aproximación matemática

del sistema RS en forma de una función de transferencia de primer orden, lo cual es una representación común para motores CC. Este enfoque permitió capturar con precisión la dinámica del motor, facilitando su modelado y control. La función de transferencia en la Ecuación 6 se obtuvo con entrada de control de PWM con valor de 255 y representa el modelo matemático del RS.

$$G(s) = \frac{4{,}673}{s + 4{,}676} \tag{6}$$

Un controlador PID aplicado a un motor CC permite regular su velocidad ajustando el voltaje de entrada. Esto se logra mediante el uso de un controlador L298N, que modula la señal de control generada por el Arduino a través de una señal PWM, permitiendo así el control preciso de la velocidad del motor.

La forma general del control por PID es:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt} \tag{7}$$

Donde $u(t)$ es la señal de control, $e(t)$ es el error (setpoint - valor medido). K_p, Ki, K_d son los coeficientes de ganancia de los términos proporcional, integral y derivativo, respectivamente, los cuales son los encargados de proporcionar un control eficiente y preciso de la velocidad del motor, mejorando su capacidad de respuesta y reduciendo el error.

En términos del dominio de la frecuencia (s), el controlador PID se representa utilizando la transformada de Laplace de la Ecuación 7. Obteniendo así la siguiente función de transferencia $C(s)$:

$$C(s) = K_p + \frac{K_i}{s} + K_d s \tag{8}$$

Con esta función de transferencia el control mediante PID se realiza en lazo cerrado como se ve en la Figura 8.

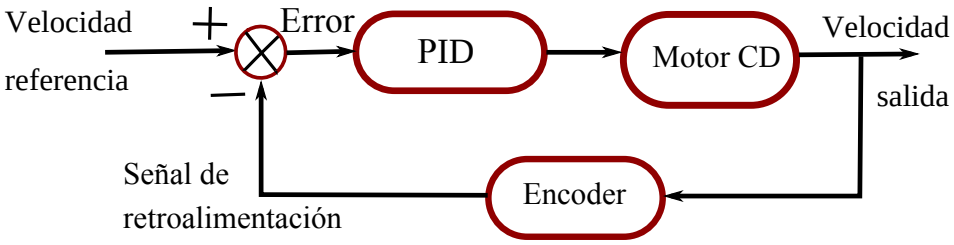


Figura 8: Diagrama de control

La implementación de este controlador al motor se realizo un análisis por medio del criterio de estabilidad Routh-Hurwitz [25], del cual se obtuvo la función de transferencia a lazo expresado en la Ecuación 9:

$$G_{PID}(s) = \frac{4{,}673(K_p s + K_i + K_d s^2)}{(s + 4{,}676)s + 4{,}673(K_p s + K_i + K_d s^2)} \tag{9}$$

La ecuación característica del sistema en lazo cerrado es:

$$(1 + 4{,}673K_d)s^2 + (4{,}676 + 4{,}673K_p)s + 4{,}673K_i = 0$$

Aplicando el criterio de Routh-Hurwitz, el sistema será estable si los valores de las ganancias K_p, K_i y K_d satisfacen las siguientes condiciones:

- $K_d \geq 0$
- $K_p > -1$
- $K_i > 0$

Estos valores aseguran que el sistema en lazo cerrado, compuesto por la función de transferencia del motor y el controlador PID, mantendrá su estabilidad.

Simulaciones con el controlador PID

Tomando como referencia estas condiciones, y mediante simulaciones en Simulink/-Matlab junto con una sintonización experimental, se determinaron las ganancias óptimas del controlador PID, las cuales se presentan en el Tabla 3.

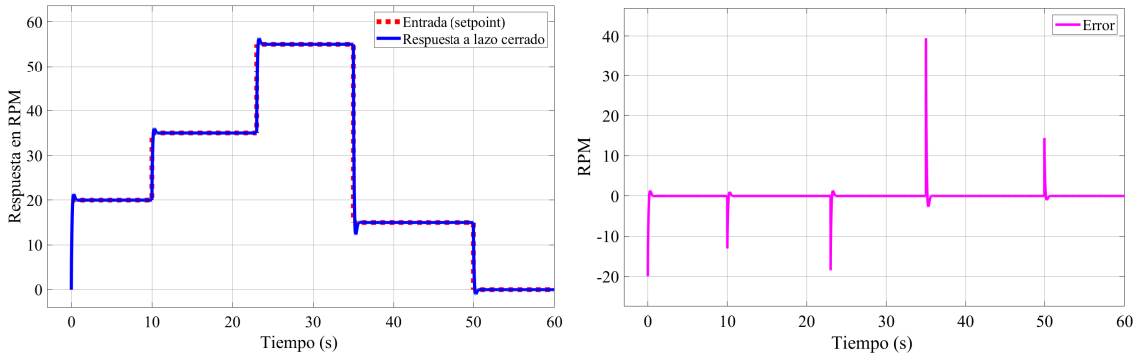
Tabla 3: Ganancias del PID

Ganancias	Valores
K_p	2.8
K_i	25
K_d	0.04

La respuesta del sistema en lazo cerrado se presenta en la Figura 9a, donde se observa en color rojo la evolución del setpoint, que varía la velocidad a lo largo del tiempo. La línea azul, por su parte, muestra el comportamiento del motor bajo el control PID, destacando cómo el sistema ajusta la velocidad del motor para seguir el valor de referencia. A medida que el setpoint cambia, el controlador PID ajusta el voltaje aplicado al motor para minimizar el error entre la velocidad deseada y la velocidad real del motor, lo que resulta en una respuesta rápida y estable. Asi mismo se muestra el error (ver Figura 9b)entre el setpoint y la velocidad de salida del motor en RPM.

4. Resultados

Los resultados experimentales en el sistema RS para la ANN y el controlador PID se basaron en los análisis discutidos en la sección anterior. Se utilizó la misma señal de referencia de velocidad propuesta en las simulaciones, con el objetivo de obtener una comparación más clara del comportamiento de ambos controladores aplicados al motor CC.



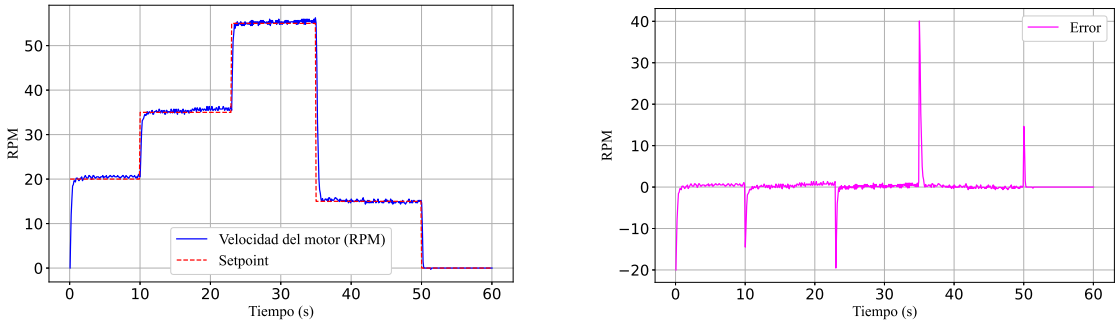
(a) Respuesta ante la entrada a lazo cerrado (b) Error

Figura 9: Simulación de control de velocidad con PID

Experimentación con ANN

La ANN ya entrenada se implementó con el microcontrolador Arduino Due, con la capacidad para realizar las predicciones en tiempo real realizadas por la ANN para ajustar el ciclo de trabajo del motor DC. Se realizó la conversión de los pesos de la ANN entrenada en Python a un formato compatible con el entorno Arduino para su uso en la predicción de PWM.

El código en Arduino se encargó de medir la velocidad del motor utilizando un encoder, calcular el error entre el setpoint y la velocidad medida, y aplicar el valor de PWM predicho por la red neuronal para controlar la velocidad del motor. La Figura 10 muestra el comportamiento del sistema RS utilizando la ANN como controlador. En la Figura 10a, se observa el seguimiento de velocidad (en azul) en comparación con el setpoint (en rojo), mientras que la Figura 10b presenta el error entre la velocidad medida y el setpoint.

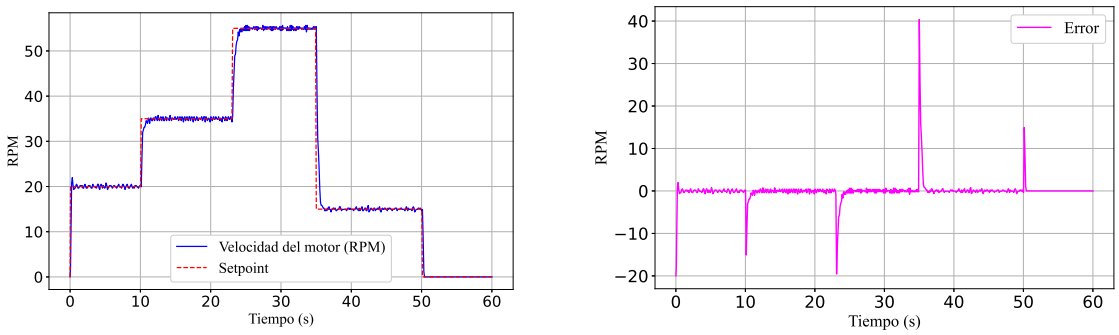


(a) Control de velocidad mediante ANN (b) Error

Figura 10: Control de velocidad del RS con ANN

Experimentación con PID

Con base en el análisis expuesto en la Sección 3, se implementó el control propuesto en el sistema RS, aplicando un ajuste de ganancias mediante una sintonización fina de forma experimental. Los valores ajustados fueron $K_d = 5$, $K_p = 15$ y $K_d = 0{,}01$, obteniendo los resultados moque se muestran en la Figura 11. En la Figura 11a, se presenta la velocidad del motor en respuesta a el setpoint determinado, mientras que la Figura 11b ilustra el error de seguimiento entre la señal de referencia y la señal medida.



(a) Control de velocidad mediante PID (b) Error

Figura 11: Control de velocidad del RS con PID

5. Conclusiones

Este estudio demuestra la efectividad de las ANN en el control de velocidad de motores de corriente continua, proporcionando una alternativa viable y adaptativa frente a métodos de control convencionales como el PID. La ANN implementada mostró un alto grado de precisión en el seguimiento de la velocidad deseada, ajustando dinámicamente la señal de control y reduciendo el error frente a cambios en el setpoint y variaciones externas. En comparación con el controlador PID, que requiere ajustes precisos de los parámetros de control, la ANN ofrece una mayor capacidad de adaptación sin depender de un modelo matemático explícito del sistema, lo cual es especialmente beneficioso para sistemas con dinámicas complejas o no lineales. Las simulaciones y experimentos realizados en el entorno de Arduino con el motor DC revelaron que la ANN y el controlador PID lograron un buen seguimiento de velocidad; sin embargo, la ANN presentó un mejor rendimiento ya que no genero sobreimpulsos en la respuesta del sistema y puede ser más robusto en presencia de perturbaciones y cambios en la dinámica del sistema. Estos resultados sugieren que, al integrar redes neuronales, los sistemas de control de motores

pueden optimizar su desempeño y fiabilidad, adaptándose mejor a variaciones en el entorno operativo. Futuros estudios podrían explorar el ajuste automático y la escalabilidad de ANN para el control de múltiples motores, así como la implementación en hardware especializado para maximizar la eficiencia en aplicaciones industriales.

Direcciones de correo de los autores

Adrian Camacho-Ramírez*: acamachor@uaemex.mx

Marco Antonio Sánchez-Carmona: msanchezc048@alumno.uaemex.mx

Referencias

[1] S. J. Hammoodi, K. S. Flayyih y A. R. Hamad, «Design and implementation speed control system of DC Motor based on PID control and matlab simulink,» en, *International Journal of Power Electronics and Drive Systems (IJPEDS)*, vol. 11, n.º 1, pág. 127, mar. de 2020, ISSN: 2722-256X, 2088-8694. DOI: 10.11591/ijpeds.v11.i1.pp127-134. visitado 9 de oct. de 2024. dirección: http://ijpeds.iaescore.com/index.php/IJPEDS/article/view/20153.

[2] M. Mahmud, S. M., A. H. y A. Nurashikin, «Control BLDC Motor Speed using PID Controller,» en, *International Journal of Advanced Computer Science and Applications*, vol. 11, n.º 3, 2020, ISSN: 21565570, 2158107X. DOI: 10.14569/IJACSA.2020.0110359. visitado 9 de oct. de 2024. dirección: http://thesai.org/Publications/ViewPaper?Volume=11&Issue=3&Code=IJACSA&SerialNo=59.

[3] S.-H. Kim, *Electric motor control: DC, AC, and BLDC motors*. Elsevier, 2017.

[4] S. Sakunthala, R. Kiranmayi y P. N. Mandadi, «A study on industrial motor drives: Comparison and applications of PMSM and BLDC motor drives,» en *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)*, IEEE, 2017, págs. 537-540.

[5] Y. A. Almatheel y A. Abdelrahman, «Speed control of DC motor using Fuzzy Logic Controller,» en *2017 International Conference on Communication, Control, Computing and Electronics Engineering (ICCCCEE)*, ene. de 2017, págs. 1-8. DOI: 10.1109/ICCCCEE.2017.7867673. visitado 9 de oct. de 2024. dirección: https://ieeexplore.ieee.org/abstract/document/7867673.

[6] A. T. Hafez, A. A. Sarhan y S. Givigi, «Brushless DC Motor Speed Control Based on Advanced Sliding Mode Control (SMC) Techniques,» en *2019 IEEE International Systems Conference (SysCon)*, ISSN: 2472-9647, abr. de 2019, págs. 1-6. DOI: 10.1109/SYSCON.2019.8836754. visitado 9 de oct. de 2024. dirección: https://ieeexplore.ieee.org/abstract/document/8836754.

[7] D. Shen, «Iterative learning control with incomplete information: a survey,» *IEEE/-CAA Journal of Automatica Sinica*, vol. 5, n.º 5, págs. 885-901, sep. de 2018, Conference Name: IEEE/CAA Journal of Automatica Sinica, ISSN: 2329-9274. DOI: 10.1109/JAS.2018.7511123. visitado 9 de oct. de 2024. dirección: https://ieeexplore.ieee.org/abstract/document/8405347.

[8] P. Kofinas y A. I. Dounis, «Fuzzy Q-learning agent for online tuning of PID controller for DC motor speed control,» *Algorithms*, vol. 11, n.º 10, pág. 148, 2018.

[9] M. Alhanjouri, «Speed Control of DC Motor Using Artificial Neural Network,» en, *International Journal of Science and Research*, vol. 6, n.º 2, 2015, ISSN: 2319-7064. DOI: 10.21275/ART20172035.

[10] H. Sridhar, P. Hemanth, Pavitra, H. V. Soumya y B. G. Joshi, «Speed Control of BLDC Motor using Soft Computing Technique,» en *2020 International Conference on Smart Electronics and Communication (ICOSEC)*, sep. de 2020, págs. 1162-1168. DOI: 10.1109/ICOSEC49089.2020.9215417. visitado 9 de oct. de 2024. dirección: https://ieeexplore.ieee.org/abstract/document/9215417.

[11] Departamento de Matemáticas e Informática de la Universidad de Barcelona, *El poder de los datos: Del big data al aprendizaje profundo*. National Geographic, 2017, ISBN: 978-84-473-8940-7.

[12] K. Gurney, *An Introduction to Neural Networks*. London: CRC Press, ene. de 2017, ISBN: 978-1-315-27357-0. DOI: 10.1201/9781315273570.

[13] Y.-c. Wu y J.-w. Feng, «Development and Application of Artificial Neural Network,» en, *Wireless Personal Communications*, vol. 102, n.º 2, págs. 1645-1656, sep. de 2018, ISSN: 1572-834X. DOI: 10.1007/s11277-017-5224-x. visitado 8 de oct. de 2024. dirección: https://doi.org/10.1007/s11277-017-5224-x.

[14] F. Lewis, «Neural network control of robot manipulators,» *IEEE Expert*, vol. 11, n.º 3, págs. 64-75, jun. de 1996, Conference Name: IEEE Expert, ISSN: 2374-9407. DOI: 10.1109/64.506755. visitado 8 de oct. de 2024. dirección: https://ieeexplore.ieee.org/abstract/document/506755.

[15] Z. Xu, S. Li, X. Zhou, S. Zhou, T. Cheng e Y. Guan, «Dynamic Neural Networks for Motion-Force Control of Redundant Manipulators: An Optimization Perspective,» *IEEE Transactions on Industrial Electronics*, vol. 68, n.º 2, págs. 1525-1536, feb. de 2021, Conference Name: IEEE Transactions on Industrial Electronics, ISSN: 1557-9948. DOI: 10.1109/TIE.2020.2970635. visitado 8 de oct. de 2024. dirección: https://ieeexplore.ieee.org/abstract/document/8984689.

[16] C. Sun, W. He, W. Ge y C. Chang, «Adaptive Neural Network Control of Biped Robots,» *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 47, n.º 2, págs. 315-326, feb. de 2017, Conference Name: IEEE Transactions on Systems, Man, and Cybernetics: Systems, ISSN: 2168-2232. DOI: 10.1109/TSMC.2016.2557223. visitado 8 de oct. de 2024. dirección: https://ieeexplore.ieee.org/abstract/document/7467542.

[17] R. Cui, C. Yang, Y. Li y S. Sharma, «Adaptive Neural Network Control of AUVs With Control Input Nonlinearities Using Reinforcement Learning,» *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 47, n.º 6, págs. 1019-1029, jun. de 2017, Conference Name: IEEE Transactions on Systems, Man, and Cybernetics: Systems, ISSN: 2168-2232. DOI: 10.1109/TSMC.2016.2645699. visitado 8 de oct. de 2024. dirección: https://ieeexplore.ieee.org/abstract/document/7812772.

[18] G. Napolitano Dell'Annunziata, V. M. Arricale, F. Farroni, A. Genovese, N. Pasquino y G. Tranquillo, «Estimation of Vehicle Longitudinal Velocity with Artificial Neural Network,» en, *Sensors*, vol. 22, n.º 23, pág. 9516, ene. de 2022, Number: 23 Publisher: Multidisciplinary Digital Publishing Institute, ISSN: 1424-8220. DOI: 10.3390/s22239516. visitado 8 de oct. de 2024. dirección: https://www.mdpi.com/1424-8220/22/23/9516.

[19] C. Cao, F. Wang, Q. Cao, H. Sun, W. Xu y M. Cui, «Neural network–based terminal sliding mode applied to position/force adaptive control for constrained robotic manipulators,» *Advances in Mechanical Engineering*, vol. 10, n.º 6, pág. 1 687 814 018 781 288, 2018.

[20] Arduino. «Arduino Due.» Accesado 19 de octubre, 2024.

[21] STMicroelectronics. «L298.» Accesado 19 de octubre, 2024.

[22] V. Nair y G. E. Hinton, «Rectified Linear Units Improve Restricted Boltzmann Machines,» en *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 2010, págs. 807-814.

[23] D. P. Kingma y J. Ba, «Adam: A method for stochastic optimization,» *arXiv preprint arXiv:1412.6980*, 2014.

[24] I. Goodfellow, Y. Bengio y A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[25] K. Ogata, «Modern control engineering,» 2020.

Informaticae Abstracta
Año 2024, Vol. 2, No. 2, pp. 20-59 JUL | DIC 20244

Redes neuronales binarias: introducción

Agustín Solís Winkler

Información del Artículo

Recibido: 12-oct-2024
Aceptado: 17-dic-2024

Palabras clave: Red neuronal, Redes neuronales binarias

Resumen

Este artículo presenta una introducción a las redes neuronales binarias. A lo largo del documento, se presentan los conceptos fundamentales del aprendizaje automático y las redes neuronales, seguido de la evolución de las técnicas de compresión de redes, los principios de la binarización, y su aplicación a redes totalmente conectadas, convolucionales y recurrentes. Tanto la cuantización como la binarización ofrecen alternativas a los métodos tradicionales de compactación de redes neuronales como la poda, la dispersión (*sparcity)* y la compartición de pesos, así como a estrategias de aceleración basadas en *hardware* como la paralelización, el uso de GPU, e implementaciones distribuidas.

El uso de valores binarios para los parámetros de la red, permite en teoría, disminuir su tamaño hasta 32 veces, mientras que el uso de operaciones a nivel de bits, como el *xnor* y el *popcount* pueden reducir el tiempo de procesamiento hasta en 50 % dependiendo del número de ciclos de reloj utilizados en operaciones de punto flotante. No obstante, la binarización lleva implícita una pérdida significativa de información, lo cual impacta directamente la exactitud de los modelos, por lo que para mitigar este problema se han desarrollado numerosas técnicas. Este artículo proporciona al lector una introducción a los fundamentos de las técnicas de binarización y ofrece una visión general de las distintas estrategias que se han utilizado para mejorar la exactitud de las redes binarias e igualarla con las redes tradicionales de punto flotante.

1. Introducción

Las redes neuronales artificiales han contribuido en los últimos años al avance de múltiples áreas como la visión por computadora y el procesamiento del lenguaje natural, debido a su capacidad para modelar dependencias y patrones no triviales, y aproximar funciones complejas aprendidas a partir de un gran volumen de datos.

Entre los factores clave que han favorecido el éxito de estos modelos de aprendizaje automático se encuentra el desarrollo de algoritmos de entrenamiento eficientes, el incremento en la capacidad y potencia de cálculo de las computadoras y la disponibilidad de grandes conjuntos de datos; lo que en suma ha permitido el desarrollo de modelos específicos para dominios particulares, como las redes convolucionales, diseñadas para trabajar con imágenes y las recurrentes, con capacidad para procesar información secuencial.

La evolución y especialización de las arquitecturas neuronales ha dado lugar a modelos con altos requerimientos computacionales tanto para su entrenamiento como para su ejecución, los cuales no eran posibles en las décadas anteriores al 2010 [1]. Modelos como LeNet-5, presentado en 1998 por LeCun, que se compone de siete capas, utiliza menos de cien mil parámetros y se ejecutaba originalmente en estaciones de trabajo de Silicon

Graphics, han dado paso en los últimos años a otros más sofisticados como GPT-4 el cual incluye 128 capas y $1{,}8 \times 10^{12}$, parámetros, requiriendo infraestructura de centros de datos con arquitectura distribuida y uso de unidades de procesamiento gráfico, el cual consume diariamente hasta $564\mathrm{m}W$ de electricidad para su operación.

Por otro lado, la computación distribuida y el uso de dispositivos de internet de las cosas, cuya relevancia se ha incrementado desde la década de 2010, ha dado lugar al modelo de cómputo perimetral (*Egde Computing*), que reduce la carga de los centros de datos especializados y busca satisfacer los requerimientos de procesamiento en tiempo real y latencia baja, utilizando *hardware* que generalmente está limitado en recursos computacionales y de consumo de energía, en comparación con los disponibles en los centros de datos.

El gran aumento en el uso de este tipo de dispositivos, ha despertado un creciente interés en modelos de aprendizaje profundo que puedan implementarse en estos entornos, impulsando el desarrollo de las técnicas de compactación de redes neuronales, con el propósito de permitir su implementación efectiva sin sacrificar su rendimiento y precisión, lo que junto con el avance en el desarrollo de procesadores, podría permitir la ejecución de modelos avanzados en escenarios restringidos en un futuro próximo.

Entre las técnicas de compresión de modelos neuronales, destacan la cuantización y la binarización, las cuales se caracterizan por la representación eficiente de pesos y activaciones utilizando formatos compactos de precisión reducida, lo cual posibilita la implementación de modelos complejos en escenarios descentralizados sin conectividad a centros de datos, donde la potencia de cómputo y la energía son restricciones significativas. La binarización y cuantización de modelos permiten reducir el tamaño y la complejidad computacional, abriendo un área de oportunidad para su implementación en dispositivos móviles, embebidos y perimetrales, sin sacrificar la capacidad de predicción y la velocidad de inferencia.

2. Estado del arte

La compresión de modelos es una área que se ha desarrollado siguiendo diferentes enfoques, que son descritos con detalle en la sección 5 Desde la introducción del concepto en 2015, el desarrollo de las redes neuronales binarias ha mantenido el objetivo de obtener modelos compactos que puedan utilizarse en dispositivos de recursos limitados, preservando la precisión, pues a pesar de ser un concepto sencillo, estas sufren de degradación en su desempeño en comparación con sus contrapartes de punto flotante [2] [3] [4] [5]. Diferentes métodos y estrategias han sido empleados desde entonces, incluyendo la reducción de errores de cuantización, la mejora de la función de pérdida y la aproximación de gradientes, la implementación de estrategias de entrenamiento específicas y el diseño de arquitecturas que consideran operaciones binarias [6] para tratar de encontrar modelos que sean amigables con el *hardware* y puedan igualar la precisión de los modelos de punto flotante.

Entre los trabajos principales en el campo de las redes binarias se pueden contar los modelos iniciales de Courbariaux como BinaryConnect [2] y BinaryNet [7], que propusieron las bases para binarizar pesos y activaciones. Modelos posteriores, como XNOR-NET de Rastegari [8] y los de Bethge [9], incorporaron factores de escala y aproximaciones probabilísticas para abordar la pérdida de información y mejorar la eficiencia. Con DoReFa-Net[10] Zhou introdujo precisión y ancho variables, y aproximación de gradientes con número limitado de bits. La tabla 1 resume algunos de los principales avances en redes binarias de 2016 a 2019, incluyendo binarización de redes convolucionales existentes como las implementadas por Tang [11] y Jacob [12], la combinación de cuantización y binarización publicada por Zhou [10], estrategias para modelado de arquitecturas binarias basadas en bloques de construcción y conexiones residuales como propone Bethge [3] [6], la red generativa adversaria de Song [13], el optimizador binario ideado por Helwegen el cual no requiere el uso de pesos latentes [14], el procedimiento de cuantización adaptable desarrollado por Razani [15] y el método iterativo de Khoram [16].

Con respecto a la binarización de redes recurrentes, se han encontrado un par de trabajos publicados; el de Zhao en 2019 [21], que presenta una red convolucional recurrente (BCRNN) compacta binarizando una memoria de largo y corto plazo LSTM siguiendo los mismos principios utilizados para las redes de propagación hacia adelante, y el de Einy en 2023 [23] donde se combinan redes convolucionales y recurrentes binarizadas en una aplicación para detección de anomalías en embriones.

La tabla 2 muestra los últimos años, comenzando en 2020 y se incluyen trabajos que consideran la eliminación de capas de normalización como el de Chen [24], un trabajo sobre binarización multinivel para redes recurrentes desarrollado por Mirsalari [25], un nuevo optimizador binario de segundo orden creado por Suárez [26], el uso de matrices de un bit para operaciones convolucionales propuesto por Redfern [27], aplicación de ramas de atención de Guo [28], estrategias para modelado de arquitecturas binarias como ampliación de los trabajos previos de Bethge [9], cuantización multinivel por Xu [29], arquitecturas multicapa con conexiones residuales del propio autor [30] y el trabajo sobre detección de objetos, publicado por Mainak [31].

La gran mayoría de los trabajos mencionados plantean entre sus líneas de investigación abiertas y trabajo futuro mejorar la precisión y radio de compresión de los modelos propuestos, explorar otros cambios de arquitectura y profundizar más en las técnicas desarrolladas.

En la amplia revisión realizada por Yuan [4] sobre el campo de las redes binarias, se ofrecen algunos ejemplos de oportunidades y retos pendientes por resolver en el campo, entre los que se mencionan:

- Decidir si una arquitectura binaria es conveniente para un problema dado.
- Utilizar búsqueda de arquitecturas para crear redes convolucionales de 2D o 3D que tengan mayor exactitud.
- Investigar la posibilidad de desarrollar modelos de transformadores basados en re-

Tabla 1: Avances en redes binarizadas y cuantizadas, 2016-2019. Elaboración ASW

Año	Autor (et al)	Modelo/Método	Contribución
2016	Courbariaux	BinaryConnect	Inferencia con pesos y activaciones binarias [2]
2016	Rastegari	XNOR-Net	Ganancia para compensar la pérdida de información [8]
2016	Zhou	Do-Re-Fa	Cuantizada y binaria, gradientes de 6 bits [10]
2017	Tang	Binary AlexNet	Binarización de AlexNet y capa de factor de escala [11] [17]
2017	Lin	ABC-Net	Bases múltiples [17]
2017	Song	BGAN	Red generativa adversaria binaria [13]
2018	Zhuang	GroupNet	Descomposición de la red en grupos binarios de precisión equivalente [18]
2018	Darabi	BNN+	Método con términos de regularización, factores de escala y mejora de la derivada de la función signo [19]
2018	Khoram	Adaptable Quant	Cuantización adaptable, algoritmo iterativo [16]
2019	Bethge	BinaryDenseNet	Conexiones residuales, recomendaciones de diseño [6]
2019	Helwegen	Bop	Optimizador binario que evita el uso de pesos latentes [14]
2019	Razani	SQ	Cuantización adaptable binaria y ternaria [15]
2019	Wang	BNN OD	Uso de BNN para detección de objetos [20]
2019	Zhao	BCRNN	Red convolucional recurrente binaria para reconocimiento de emociones en audio [21]
2019	Zheng	NLP	LSTM y Capa conectada para modelado de lenguaje [22]

des binarias para tareas de procesamiento de imágenes que puedan superar el desempeño de los basados en redes convolucionales.

Estos ejemplos dan una idea de la amplitud del campo de redes neuronales binarias y la gran cantidad de áreas y aplicaciones que se pueden explorar.

Tabla 2: Avances en redes binarizadas y cuantizadas, 2020-2023. Elaboración ASW

Año	Autor (et al)	Modelo/Método	Contribución
2020	Mirsalari	MuBiNN	Binarización multinivel para redes recurrentes [25]
2020	Bethge	MeliusNet	Arquitectura binaria con bloques densos [9]
2021	Chen	BNN-BN Free	Propone eliminar las capas de normalización y recorte de gradiente [24]
2021	Suarez Ramirez	PBOp, BOp2O	Optimizador de segundo orden para BNN [26]
2021	Redfern	BCNN	BNN convolucional con operaciones de matriz de 1 bit [27]
2023	Guo	BNext	Bloques de construcción, Maestro-alumno seguida de destilación de conocimiento [28]
2023	Xu	SLQ/MLQ	Cuantización de uno y múltiples niveles [29]
2023	Mainak	BDenseNet	Detección de peatones mediante una red neuronal binaria [31]
2023	Einy	LBCN LSTM	CRNN binaria local para detección de anomalías en embriones [23]
2023	Solis Winkler	RBMP	Perceptrón multicapa binario con conexiones residuales [30]

3. Introducción al aprendizaje automático

Antecedentes

En los primeros años de la inteligencia artificial (AI), el desarrollo se centró en algoritmos que, tomando como base hechos conocidos, podían encontrar la solución a un problema. Este tipo de métodos se conocen en AI como deductivos, porque utilizan hechos, reglas lógicas, métodos de búsqueda y razonamiento automático para obtener conclusiones [32] [33]. El enfoque deductivo tuvo su mayor auge con los sistemas expertos, que impulsaron durante la década de los 1970 y parte de los 1980 la industria de la Inteligencia Artificial [34]. Sin embargo, estos métodos requieren hipótesis bien establecidas para llegar a conclusiones que se puedan probar como correctas, y no pueden generar respuestas cuando no existen reglas de las que se pueda inferir una conclusión [32].

El enfoque de los seres humanos al aprendizaje está basado por lo general en la experiencia probatoria, a partir de observaciones del mundo real [32] [35]; los seres humanos tenemos la habilidad natural de crear hipótesis generalizadas a partir de ejemplos, que luego utilizamos para resolver problemas nuevos en situaciones similares. Esta capacidad tiene un elemento de generalización que no puede ser justificado en su totalidad por el uso

de métodos basados en la lógica. En muchos campos, el aprendizaje a través de la práctica es más eficiente que tratar de aprender las reglas de un sistema formalmente definido [32].

Esta observación llevó al desarrollo de métodos basados en aprendizaje a partir de evidencias, en los que los datos proporcionan la base para construir la hipótesis. Dentro del campo de la AI, este enfoque se conoce como aprendizaje automático [32] [34].

Fundamentos

El aprendizaje automático (ML, *Machine Learning*) es un paradigma que forma parte de los métodos de la AI, el cual se enfoca en el desarrollo de algoritmos capaces de mejorar su rendimiento a partir de datos, sin estar explícitamente programados para cada tarea [36] [34]. Esta sección sintetiza las ideas que se exponen en el texto de Shalev-Shwartz y Ben David [37].

Objetos de interés y regla de predicción

Sea $\mathcal{X}$ un conjunto arbitrario (dominio) de datos que representa objetos de interés generados por una distribución de probabilidad desconocida, $\mathcal{D}$.

Sea f una función, también desconocida (objetivo), que asigna a cada objeto en $\mathcal{X}$ un valor en un conjunto $\mathcal{Y}$, el cual representa todos los valores objetivo posibles (etiquetas) que corresponden a cada elemento de $\mathcal{X}$. Esta asignación se expresa como $f : \mathcal{X} \to \mathcal{Y}$, y cumple con que $y_i = f(x_i)$, donde $x_i \in \mathcal{X}$ y $y_i \in \mathcal{Y}$.

Entonces, el **aprendizaje automático** puede definirse como el proceso de encontrar una *función de hipótesis* o *regla de predicción*, $h : \mathcal{X} \to \mathcal{Y}$, que aproxima la función objetivo f con base en los datos observados. Para encontrar la función h, se utiliza un conjunto $\mathcal{S} \subset \mathcal{X} \times \mathcal{Y}$, el cual es una secuencia de pares ordenados de la forma $\mathcal{S} = ((x_1, y_1), \ldots, (x_m, y_m))$, que se conoce como conjunto de entrenamiento.

Una hipótesis $h_{\mathcal{S}}$ obtenida a partir de la muestra $\mathcal{S}$ forma parte de un conjunto $\mathcal{H}$, denominado *clase de hipótesis*, el cual está compuesto por todas las reglas de predicción que pueden encontrarse a partir del conjunto $\mathcal{S}$, es decir, $h_{\mathcal{S}} \in \mathcal{H}$.

Al utilizar la hipótesis $h_{\mathcal{S}}$ para predecir el valor de un punto aleatorio en $\mathcal{X}$, existe una probabilidad de que $h(x) \neq f(x)$. Esta probabilidad se conoce como el error real, pérdida o error de generalización de la regla de predicción; denotado como $L_{\mathcal{D},f}(h)$, y se mide con respecto a la distribución $\mathcal{D}$ y la función f:

$$L_{\mathcal{D},f}(h) \stackrel{\text{def}}{=} \mathbb{P}[h(x) \neq f(x)] \stackrel{\text{def}}{=} \mathcal{D}(\{x : h(x) \neq f(x)\}),$$

es decir, la probabilidad de que un punto x tomado de $\mathcal{D}$ pertenezca al conjunto de error.

Riesgo empírico

Sin embargo, dado que tanto $\mathcal{D}$ como f son desconocidos, el error real no puede calcularse directamente. Una forma de obtener una estimación del error es cuantificar el error en el que incurre la regla de predicción sobre cada par en la secuencia $\mathcal{S}$, lo cual se denomina error o *riesgo empírico*:

$$L_S(h) \stackrel{\text{def}}{=} \frac{|\{i \in [m] : h(x_i) \neq y_i\}|}{m},$$

donde $[m] = \{1, \ldots, m\}$, siendo m el número de pares en $\mathcal{S}$.

Función de pérdida

Para formalizar el concepto de error en las predicciones de la hipótesis h, se introduce el concepto de función de pérdida $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}^+$, la cual mide la discrepancia entre el valor predicho $h(x)$ y el valor real $y = f(x)$. Un ejemplo común es la *pérdida cero-uno*, definida como

$$\ell(h(x), y) = \begin{cases} 1 & \text{si } h(x) \neq y, \\ 0 & \text{si } h(x) = y. \end{cases}$$

La función de pérdida permite definir el error de generalización de la hipótesis h como el valor esperado de la pérdida en el caso general:

$$L_{\mathcal{D},f}(h) = \mathbb{E}_{x \sim \mathcal{D}}[\ell(h(x), f(x))],$$

donde $\mathbb{E}_{x \sim \mathcal{D}}$ representa la esperanza con respecto a la distribución desconocida $\mathcal{D}$. Este riesgo mide la discrepancia esperada entre las predicciones de h y los valores reales dados por f, según $\mathcal{D}$.

El riesgo empírico se puede expresar en función de la función de pérdida mediante la siguiente ecuación:

$$L_S(h) = \frac{1}{m} \sum_{i=1}^{m} \ell(h(x_i), y_i),$$

donde m es el número de muestras en $\mathcal{S}$ y $\ell(h(x_i), y_i)$ representa la pérdida incurrida por h en cada ejemplo de la muestra.

El *principio de minimización del riesgo empírico* establece que, dado el conjunto de entrenamiento $\mathcal{S}$, el objetivo del aprendizaje es encontrar una hipótesis $h \in \mathcal{H}$ que minimice $L_S(h)$, es decir:

$$h^* = \arg\min_{h \in \mathcal{H}} L_S(h).$$

Este criterio sugiere seleccionar la hipótesis que mejor se ajuste a los datos observados, con la esperanza de que esta minimización también reduzca el riesgo real en nuevos datos provenientes de la misma distribución $\mathcal{D}$.

Modelo de aprendizaje

En el contexto del aprendizaje automático, un **modelo de aprendizaje** es una función parametrizada h_θ, donde θ representa el conjunto de parámetros que determinan el comportamiento y la capacidad predictiva de la función h. Estos parámetros definen el espacio en el que el modelo busca la función de hipótesis que mejor aproxima f, en función de los datos de entrenamiento.

Los **parámetros del modelo** pueden variar ampliamente en función de su estructura:

- En un modelo de **regresión lineal**, los parámetros θ suelen ser los coeficientes de la combinación lineal que define la función de predicción.
- En **redes neuronales**, los parámetros son los pesos y sesgos de cada conexión entre nodos.

Generalizando, estos parámetros θ son los elementos ajustables que configuran la relación entre las entradas y las salidas del modelo.

Aprendizaje y minimización de la función de pérdida

Para que el modelo realice predicciones correctas, se ajustan los parámetros θ de modo que minimicen el error o el riesgo empírico $L_S(h_\theta)$, a través de un proceso de aprendizaje o entrenamiento que implemente alguna técnica de optimización. Uno de los métodos más comunes de optimización es el **descenso de gradiente** (GD, *Gradient Descent*), que actualiza los parámetros en la dirección opuesta al gradiente de $L_S(h_\theta)$, buscando el punto donde el riesgo empírico es mínimo [36]. Este proceso puede expresarse mediante:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta L_S(h_\theta)$$

donde θ representa los parámetros del modelo, $\eta > 0$ es la tasa de aprendizaje y $\nabla_\theta L_S(h_\theta)$ es el gradiente de $L_S(h_\theta)$ con respecto a θ. Para acelerar la convergencia, se utiliza la variante estocástica (SGD, *Stochastic Gradient Descent*), en la cual el gradiente se calcula en función de una muestra aleatoria del conjunto de datos.

Funciones de pérdida comunes

Error cuadrático medio (MSE) . Este es ampliamente utilizado en problemas de regresión. Esta función calcula el promedio del cuadrado de las diferencias entre las predicciones y los valores reales, penalizando fuertemente los errores grandes [38]. Se define como:

$$\text{MSE} = \frac{1}{m} \sum_{i=1}^{m} (h(x_i) - y_i)^2,$$

donde m es el número de ejemplos, $h(x_i)$ es la predicción para el ejemplo i, y y_i es el valor real.

Error absoluto medio (MAE) . Utiliza alternativamente en problemas de regresión. A diferencia de MSE, esta función calcula el promedio de los valores absolutos de las diferencias, penalizando todos los errores de manera uniforme [38]. Se define como:

$$\text{MAE} = \frac{1}{m}\sum_{i=1}^{m}|h(x_i) - y_i|.$$

MAE es menos sensible a valores atípicos comparado con MSE, ya que no eleva al cuadrado las diferencias.

Entropía cruzada (*Cross-Entropy*) . Es comúnmente utilizada en problemas de clasificación, especialmente en redes neuronales [39]. Para una clasificación binaria, la entropía cruzada se define como:

$$\text{CE} = -\frac{1}{m}\sum_{i=1}^{m}\left(y_i \log(h(x_i)) + (1-y_i)\log(1-h(x_i))\right), \tag{1}$$

donde y_i es el valor verdadero (0 ó 1) y $h(x_i)$ es la probabilidad predicha por el modelo para la clase 1. En el caso de múltiples clases, se utiliza una versión generalizada, conocida como *entropía cruzada categórica*:

$$\text{CE} = -\frac{1}{m}\sum_{i=1}^{m}\sum_{c=1}^{C} y_{i,c}\log(h_{i,c}),$$

donde C es el número de clases, $y_{i,c}$ es una variable indicadora que toma el valor 1 si el ejemplo i pertenece a la clase c, y $h_{i,c}$ es la probabilidad predicha para la clase c.

Pérdida de bisagra (*Hinge Loss*) Esta función se utiliza comúnmente en Máquinas de Vectores de Soporte (SVM) y se define para problemas de clasificación binaria. La función busca maximizar el margen de separación entre clases. La pérdida para un ejemplo (x_i, y_i), donde $y_i \in \{-1, 1\}$, es:

$$\text{Hinge Loss} = \frac{1}{m}\sum_{i=1}^{m}\max(0, 1 - y_i \cdot h(x_i)),$$

donde $h(x_i)$ es la predicción del modelo, que debe ser mayor que 1 para clasificar correctamente el ejemplo en la clase y_i.

Paradigmas de aprendizaje

Los modelos de ML se definen por medio de parámetros ajustables como se describe en 3. El proceso de aprendizaje consiste en la optimización de estos parámetros como puede verse en 3. Los algoritmos de aprendizaje se clasifican típicamente en [34]:

- **Aprendizaje supervisado:** El modelo se entrena utilizando un conjunto de datos etiquetados $(x_i, y_i)_{i=1}^m$, donde $(x_i, y_i) \in \mathcal{S} \subset \mathcal{X} \times \mathcal{Y}$, con el objetivo de minimizar la pérdida $\ell(h_\theta(x_i), y_i)$ a lo largo de todo el conjunto de datos de entrenamiento [40].
- **Aprendizaje no supervisado:** En ausencia de etiquetas y_i, el modelo intenta aprender la estructura o patrones inherentes en el conjunto de datos $\mathcal{X}$. Ejemplos comunes incluyen la agrupación (*clustering*) y la reducción de dimensiones.
- **Aprendizaje por refuerzo:** El modelo aprende a tomar decisiones mediante la interacción con un entorno, recibiendo recompensas r_t en función de sus acciones a_t. El objetivo es maximizar la suma acumulada de recompensas esperadas, formalmente $\mathbb{E}\left[\sum_{t=0}^{T} \gamma^t r_t\right]$, donde $\gamma \in [0, 1]$ es un factor de descuento.

Regularización y generalización

En ML, el entrenamiento de un modelo se realiza con un conjunto de entrenamiento (3), mientras que su capacidad de generalización se evalúa con un conjunto de prueba, que contiene datos no vistos durante el proceso de aprendizaje [37]. La *generalización* se refiere a la habilidad del modelo para desempeñarse bien con datos externos, y es un objetivo clave en el diseño de algoritmos. Un modelo que logra un rendimiento alto en el conjunto de entrenamiento, pero falla al aplicarse al conjunto de prueba, sufre de un fenómeno conocido como **sobreajuste**.

El sobreajuste indica que el modelo ha aprendido patrones específicos de los datos de entrenamiento, incluyendo ruido o detalles particulares, en lugar de capturar relaciones generales que puedan aplicarse a datos nuevos. Para mitigarlo y mejorar la generalización, se emplean técnicas de *regularización*, que introducen penalizaciones o restricciones en los parámetros del modelo [36]. Una forma regularizada de riesgo empírico es:

$$L_{S,\lambda}(h) = L_S(h) + \lambda R(h)$$

donde λ es un parámetro de regularización y $R(h)$ es una medida de complejidad del modelo h. Al minimizar $L_{S,\lambda}(h)$, el algoritmo de aprendizaje balancea la precisión en los datos de entrenamiento y la simplicidad del modelo, buscando mejorar la capacidad de generalización de h.

Funciones de salida

Dependiendo de la tarea a resolver, se eligen diferentes funciones de salida al final del modelo para adecuar la predicción de acuerdo con la naturaleza del problema. Las funciones de salida más comunes en problemas de clasificación y regresión se describen a continuación [38].

Clasificación binaria: función sigmoide. Para problemas de clasificación binaria, donde el objetivo es clasificar las muestras en una de dos categorías, se usa comúnmente la función *sigmoide* como función de salida. La función sigmoide convierte cualquier valor real en un valor entre 0 y 1, interpretado como la probabilidad de pertenecer a la clase positiva. Esta función se define como:

$$\text{sigmoide}(z) = \frac{1}{1 + e^{-z}} \tag{2}$$

Dado un valor de salida z, la función sigmoide asigna una probabilidad que permite tomar decisiones basadas en un umbral (por ejemplo, $0{,}5$) para clasificar una entrada como positiva o negativa.

Clasificación multiclase: función softmax. En problemas de clasificación multiclase, donde se desea predecir una entre varias categorías posibles, se utiliza la función *softmax* como función de salida. Esta función toma un vector de valores reales y los convierte en probabilidades, asignando una probabilidad a cada clase de manera la suma total de éstas es 1. Para una salida con n clases, la función softmax está definida como:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$$

donde z_i representa el puntaje de la clase i. El resultado de la función softmax es un vector en el que cada entrada indica la probabilidad de la clase correspondiente. Esta función es ampliamente utilizada en redes neuronales artificiales y otros modelos de clasificación multiclase, ya que convierte los valores de salida en probabilidades interpretables.

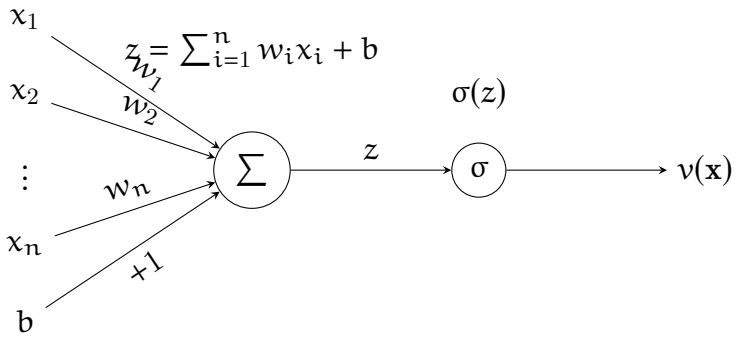
Regresión: función identidad. En tareas de regresión, el objetivo es predecir un valor continuo en lugar de una categoría discreta. En estos casos, es común utilizar la *función identidad* como función de salida. La función identidad simplemente devuelve el valor de salida sin realizar ninguna transformación adicional, permitiendo al modelo predecir cualquier valor en el conjunto de los números reales. Matemáticamente, se puede expresar como:

$$\text{identidad}(z) = z \tag{3}$$

4. Redes neuronales artificiales

Las **redes neuronales artificiales** (ANN, *Artificial Neural Network*) son modelos computacionales abstractos inspirados originalmente en el funcionamiento de las redes neuronales biológicas, y son ampliamente utilizadas en el ML para aproximar funciones complejas construidas a partir de la composición de funciones simples. Las redes neuronales están conformadas por una estructura organizada de unidades de procesamiento o neuronas

Figura 1: Neurona Artificial - (Por ASW)



conectadas entre sí [35] [38] [37], donde la salida de una neurona se utiliza como entrada de otra, y donde cada una de estas realiza transformaciones que permiten extraer características útiles de los datos [41].

Neurona artificial

El componente fundamental de una red neuronal es la unidad de procesamiento denominada **neurona artificial**. Esta es una función de la forma:

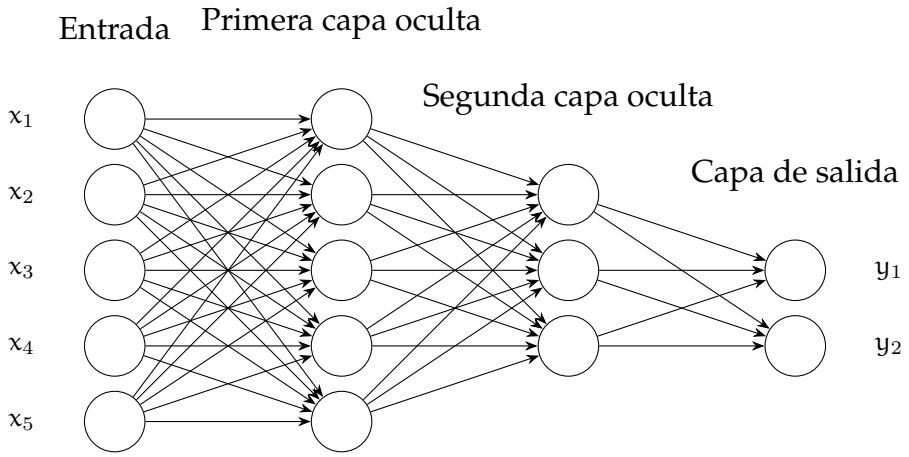
$$\mathbb{R}^d \ni \mathbf{x} \mapsto \nu(\mathbf{x}) = \sigma(\boldsymbol{w}^\intercal \mathbf{x} + b),$$

donde $\boldsymbol{w} \in \mathbb{R}^d$ se conoce como **vector de pesos**, $b \in \mathbb{R}$ se denomina **sesgo** (*bias*), y σ es una función conocida como **función de activación**. Este modelo matemático de la neurona biológica se debe a McCulloch y Pitts [42]. Si se considera que σ es la función escalón $\sigma = 1_{\mathbb{R}_+}$, $\mathbb{R}_+ := [0, \infty]$, entonces la neurona se "dispara" o activa si la suma ponderada del vector de entrada $\mathbf{x}$ sobrepasa el umbral $-b$. Si se fijan los valores de d y la función σ, entonces la neurona queda parametrizada por los $d + 1$ valores $w_1, \ldots, w_d, b \in \mathbb{R}$. [43] En la expresión que se utiliza comúnmente para para la neurona, dados los valores de entrada $\mathbf{x} = \{x_1, \ldots, x_d\} \in \mathbb{R}^d$, donde cada entrada x_i está asociada a un peso w_i que modula su contribución a la salida del modelo, como se observa en la figura 1, la neurona calcula una combinación ponderada de sus entradas sumando el producto de cada x_i por su respectivo peso w_i, agrega el sesgo b y posteriormente aplica σ [40] para calcular a salida $\nu(\mathbf{x})$:

$$\nu(\mathbf{x}) = \sigma\left(\sum_{i=1}^{n} w_i x_i + b\right), \tag{4}$$

El ajuste de los valores de los pesos y los sesgos durante el entrenamiento es lo que permite que el modelo aprenda a aproximar una función objetivo, transformando los datos de entrada en resultados que minimicen la diferencia con respecto a las salidas esperadas.

Figura 2: Perceptrón con dos capas ocultas - (Por ASW)



Red neuronal de propagación hacia adelante

Una red neuronal de propagación hacia adelante (FFNN, *Feed Forward Neural Network*) es uno de los tipos más comunes de red neuronal la cual se distingue por el hecho de que sus neuronas se encuentran organizadas en capas, en las que la entrada de la capa $\ell + 1$ consiste únicamente de las salidas de la capa ℓ. Estas redes, se conocen comúnmente como perceptrones, por extensión del concepto desarrollado por Rosenblatt [44]. Este tipo de red se puede visualizar en la fig 2 Las siguientes secciones se basan en el enfoque teórico de Petersen [43] y Shalev-Shwartz [37].

La red de propagación hacia adelante más sencilla, se denomina **red neuronal superficial** (*shallow*) y se define como una transformación afín aplicada a la salida de un conjunto de neuronas que comparten las mismas entradas y la misma función de activación. Esta transformación es un mapa $T : \mathbb{R}^p \to \mathbb{R}^q$ tal que $T(\mathbf{x}) = \boldsymbol{W}\mathbf{x} + \mathbf{b}$ para algún $\boldsymbol{W} \in \mathbb{R}^{q\times p}$, $\mathbf{b} \in \mathbb{R}^q$, donde $p, q \in \mathbb{N}$. Formalmente, la red neuronal superficial es un mapa Φ de la forma

$$\mathbb{R}^d \ni \mathbf{x} \mapsto \Phi(\mathbf{x}) = T_1 \circ \sigma \circ T_0(\mathbf{x})$$

donde T_0, T_1 son transformaciones afines y se entiende que σ se aplica a cada componente de $T_1(\mathbf{x})$.

Una **red neuronal profunda** (*deep*) se construye realizando una composición de redes neuronales superficiales, obteniendo un mapa de la forma

$$\mathbb{R}^d \ni \mathbf{x} \mapsto \Phi(\mathbf{x}) = T_{L+1} \circ \sigma \circ \cdots \circ T_1 \circ T_0(\mathbf{x})$$

donde $L \in \mathbb{N}$ y $(T_j)_{j=0}^{L+1}$ son transformaciones afines. El número de composiciones L es el número de capas o **profundidad** de la red. De manera similar a la neurona, la red neuronal puede verse como una clase de funciones parametrizada por las matrices y vectores que determinan las transformaciones afines.

Definición Formal

En la sección anterior se define la neurona ν, y se establece el concepto de red neuronal de propagación hacia adelante. Para precisar el concepto, se presenta la

Definition 1. *Sean* $L \in \mathbb{N},\ d_0, \ldots, d_{L+1} \in \mathbb{N}$ *y sea* $\sigma : \mathbb{R} \to \mathbb{R}$. *Una función* $\Phi : \mathbb{R}^{d_0} \to \mathbb{R}^{d_{L+1}}$ *se denomina* ***red neuronal de propagación hacia adelante*** *si existen matrices* $\boldsymbol{W}^{\ell} \in \mathbb{R}^{d_{\ell+1} \times d_\ell}$ *y vectores* $b^{(\ell)} \in \mathbb{R}^{d_{\ell+1}},\ \ell = 0, \ldots, L$, *tales que con*

$$\boldsymbol{x}^{(0)} := \boldsymbol{x} \tag{5}$$

$$\boldsymbol{x}^{(\ell)} := \sigma(\boldsymbol{W}^{(\ell-1)}\boldsymbol{x}^{(\ell-1)} + \boldsymbol{b}^{(\ell-1)}) \quad \textit{para } \ell \in [L] \tag{6}$$

$$\boldsymbol{x}^{(L+1)} := \boldsymbol{W}^{(L)}\boldsymbol{x}^{(L)} + \boldsymbol{b}^{(L)} \tag{7}$$

se cumple que

$$\Phi(\boldsymbol{x}) = \boldsymbol{x}^{(L+1)} \quad \textit{para todo } \boldsymbol{x} \in \mathbb{R}^{d_0}$$

y para la red Φ

L *es la* ***profundidad***,
$d_{max} = max_{\ell=1,\ldots,L} d_\ell$ *es el* ***ancho***,
σ *es la* ***función de activación***
$(\sigma; d_0, \ldots, d_{L+1})$ *la* ***arquitectura***
$\boldsymbol{W}^{(\ell)} \in \mathbb{R}^{d_{\ell+1} \times d_\ell},\ \ell = 0, \ldots, L$ *son las* ***matrices de pesos***
$\boldsymbol{b}^{(\ell)} \in \mathbb{R}^{d_{\ell+1}},\ \ell = 0, \ldots, L$ *son los* ***vectores de sesgo***

Visualización como gráfica dirigida

La arquitectura de la red neuronal FFNN se puede visualizar de manera más sencilla utilizando el concepto de gráfica dirigida acíclica [37], [32]

$$G = (V, E)$$

donde V es el conjunto de neuronas que la integran y E es el conjunto de aristas que las conectan. Las aristas tienen asociadas una función de peso $w : E \to \mathbb{R}$. Cada neurona se modela como una función escalar $\sigma : \mathbb{R} \to \mathbb{R}$ como se observa en la ecuación (4). Para simplificar la descripción se supone que las neuronas se agrupan unidades computacionales denominadas capas, las cuales se conectan entre si [35] [38] [37]. Una capa es un subconjunto disjunto no vacío de neuronas [37], donde cada capa transforma los datos de su entrada y transmite su salida a la capa siguiente. Siendo V el conjunto de neuronas que componen la red,

$$V = \bigcup_{\ell=0}^{L} V_\ell,$$

donde cada arista en E conecta algún nodo en $V_{\ell-1}$ con algún otro nodo en V_ℓ para algún $\ell \in [L]$.

La primera capa V_0 se denomina capa de entrada. Esta contiene $n+1$ neuronas siendo n la dimensión del espacio de entrada. Para cada $i \in [n]$, la salida de la neurona i en V_0 es simplemente x_i, y la última neurona es constante y siempre tiene como salida 1. Si se denota por $v_{\ell,i}$ a la i-ésima neurona de la capa ℓ y por $h_{\ell,i}(\mathbf{x})$ la salida de la neurona $v_{l,i}$ cuando la red se alimenta con el vector de entrada $\mathbf{x}$, entonces para $i \in [n]$, $h_{0,i}(\mathbf{x}) = x_i$ y $h_{0,n+1}(\mathbf{x}) = 1$. El cálculo de la salida total de la red se realiza capa por capa. Suponiendo que se ha calculado la salida de la capa ℓ, entonces la salida de la capa $\ell + 1$ se calcula como sigue: para alguna $v_{\ell+1,j} \in V_{\ell+1}$, si $a_{\ell+1}(\mathbf{x})$ es la entrada de $v_{\ell+1,j}$ cuando la red es alimentada con el vector de entrada $\mathbf{x}$, entonces,

$$a_{\ell+1,j}(\mathbf{x}) = \sum_{r:(v_{\ell,r},v_{\ell+1,j})\in E} w((v_{\ell,r}, v_{\ell+1,j}))h_{\ell,r}(\mathbf{x}),$$

y

$$h_{\ell+1,j}(\mathbf{x}) = \sigma(a_{\ell+1,j}(\mathbf{x})).$$

Que significa que la entrada a $v_{\ell+1,j}$ es la suma ponderada de las salidas de las neuronas en V_ℓ que se conectan a $v_{\ell+1}$, donde los pesos son acordes con w, y la salida de $v_{l+1,j}$ es simplemente la aplicación de la función σ sobre la entrada.

Las capas $V_1, \ldots, V_{L-1}$ se denominan capas ocultas. La última capa V_L se conoce como capa de salida. La profundidad de la red es L , siendo este el número de capas de la misma (con excepción de V_0). El tamaño de la red es $|V|$, y el ancho de la red es $\max_\ell |V_\ell|$.

Variantes

Existen muchas variantes del modelo de FFNN:

- Pueden utilizarse diferentes funciones de activación σ_ℓ en cada capa o utilizar funciones de activación distintas en cada nodo.
- Si los nodos en la capa ℓ pueden tener como entrada las capas $\mathbf{x}^{(0)}, \ldots, \mathbf{x}^{(\ell-1)}$, se dice que la red tiene **conexiones residuales**.
- Las **redes recurrentes** permiten flujo de la información hacia atrás, si la entrada a la capa $\mathbf{x}^{(\ell)}$ proviene de las capas $\mathbf{x}^{(\ell-1)}, \ldots, \mathbf{x}^{(L+1)}$, Esto crea ciclos y se introduce un índice de tiempo $t \in \mathbb{N}$, dado que la salida de un nodo en el tiempo t puede ser diferente de la salida en el tiempo $t + 1$.

Terminología

Parámetros o pesos. Se denominan parámetros todos los elemento de las matrices de pesos y los vectores de sesgo.

$$\boldsymbol{w} = ((\boldsymbol{W}^{(0)}, \mathbf{b}^{(0)}), \ldots, (\boldsymbol{W}^{(L)}, \mathbf{b}^{(L)}))$$

Estos parámetros son ajustables, y se obtienen durante el proceso de entrenamiento, lo cual determina la función específica realizada por la red.

Hiperparámetros. Son valores que definen la arquitectura de la red e influyen en el proceso de entrenamiento, aunque no son aprendidos durante este, como la profundidad, el ancho de cada capa y la función de activación utilizada. Estos se definen por anticipado, antes del entrenamiento de la red.

Modelo. Para una arquitectura fija, cada combinación de los parámetros en $\boldsymbol{w}$ que define una función específica $\boldsymbol{x} \mapsto \Phi_{\boldsymbol{x}}(\boldsymbol{x})$ se denomina modelo. En general, es cualquier parametrización de una función con el conjunto $\boldsymbol{w} \in \mathbb{R}^n, n \in \mathbb{N}$.

Operaciones básicas sobre redes neuronales

Proposition 1. *Para dos redes neuronales* Φ_1 *con arquitectura* $(\sigma; d_0^1, d_1^1 \ldots, d_{L_1+1}^1)$ *y* Φ_2 *con arquitectura* $(\sigma; d_0^2, d_1^2, \ldots, d_{L_2+1}^2)$*, se cumple que*

I. para todo $\alpha \in \mathbb{R}$ *existe una red neuronal* Φ_α *con arquitectura* $(\sigma; d_0^1, d_1^1, \ldots, d_{L_1+1}^1)$ *tal que*

$$\Phi_\alpha(\boldsymbol{x}) = \alpha\Phi_1(\boldsymbol{x}) \quad \textit{para todo } \boldsymbol{x} \in \mathbb{R}^{d_0^1},$$

II. si $d_0^1 = d_0^2 =: d_0$ *y* $L_1 = L_2 =: L$*, entonces existe una red neuonal* $\Phi_{\mathtt{paralela}}$ *con una arquitectura* $(\sigma; d_0, d_1^1 + d_1^2, \ldots, d_{L+1}^1 + d_{L+1}^2)$ *tal que*

$$\Phi_{\mathtt{paralela}}(\boldsymbol{x}) = (\Phi_1(\boldsymbol{x}), \Phi_2(\boldsymbol{x})) \quad \textit{para todo } \boldsymbol{x} \in \mathbb{R}^{d_0}$$

III. si $d_0^1 = d_0^2 =: d_0$ *y* $L_1 = L_2 =: L$*,* y $d_{L+1}^1 = d_{L+1}^2 =: d_{L+1}$*, entonces existe una red neuronal* $\Phi_{\mathtt{suma}}$ *con arquitectura* $(\sigma; d_0, d_1^1 + d_1^2, \ldots, d_L^1 + d_L^2, d_{L+1})$ *tal que*

$$\Phi_{\mathtt{suma}}(\boldsymbol{x}) = \Phi_1(\boldsymbol{x} + \Phi_2(\boldsymbol{x}) \quad \textit{para todo } \boldsymbol{x} \in \mathbb{R}^{d_0}$$

IV. si $d_{L_1+1}^1 = d_0^2$*, entonces existe* $\Phi_{\mathtt{comp}}$ *con una arquitectura* $(\sigma; d_0^1, d_1^1 \ldots, d_{L_1}^1, d_1^2, \ldots, d_{L_2+1}^2)$ *tal que*

$$\Phi_{\mathtt{comp}} = \Phi_2 \circ \Phi_1(\boldsymbol{x}) \quad \textit{para todo } \boldsymbol{x} \in \mathbb{R}^{d_0^1}$$

Tamaño de la red

Las redes neuronales proporcionan un marco para parametrizar funciones. La meta es encontrar una red neuronal que se ajuste a una relación de entrada-salida subyacente. La arquitectura (profundidad, ancho y función de activación) se escoge generalmente a priori y se considera fija. Durante el entrenamiento de la red, los parámetros (pesos y sesgos) se adaptan convenientemente mediante algún algoritmo. Dependiendo de la aplicación, puede ser deseable imponer restricciones adicionales a los pesos y sesgos. Las más comunes son:

- **compartición de pesos**. Esta es una técnica donde elementos específicos de las matrices de pesos o de los vectores de sesgo se restringen a tener el mismo valor. Formalmente, esto significa imponer condiciones de la forma $W^{(i)}_{k,l} = W^{(j)}_{s,t}$, i.e. el elemento (k, l) de la i-ésima matriz de pesos es igual al elemento (s, t) de la j-ésima matriz. Durante el entrenamiento, los pesos compartidos se actualizan al mismo tiempo, aplicando cualquier cambio de un peso de manera simultánea a los demás.

- **dispersión**. Para esta restricción, se fijan por anticipado elementos de las matrices de pesos o de los vectores de sesgo con el valor de cero $W^{(i)}_{k,l} = 0$ para algunos(k, l, i). Estos elementos con valor de cero se consideran fijos, y no se actualizan durante el entrenamiento. Esto significa que el nodo l de la capa $i-1$ no se utiliza como entrada del nodo k en la capa i, lo que en la representación gráfica equivale a no conectar estos nodos.

Ambas restricciones reducen el número de parámetros que la red neuronal tiene que aprender. El número de parámetros puede verse como una medida de la complejidad de la clase de funciones representada por la red. Este número se puede definir formalmente como

Definition 2. *Sea Φ una red neuronal de acuerdo con la definición 1. El tamaño de Φ es*

$$\textit{tamaño}(\Phi) = \left|\left(\{(i, k, l) \mid W^{(i)}_{k,l} \neq 0\} \cup \{(i, k) \mid b^{(i)}_{k} \neq 0\}\right) / \sim\right|$$

Funciones de activación

Como se explica en (4), la función de activación en combinación con el sesgo, define el umbral de "disparo" de la neurona. En los modelos más antiguos, los valores de los pesos se ajustaban de forma manual y utilizando heurísticas. Posteriormente se introdujo un algoritmo automático conocido como regla del perceptrón que utiliza una función de pérdida basada en la función signo [45]. Entre las limitaciones de estas funciones se tiene la falta de continuidad, lo que impide modelar salidas continuas o probabilísticas. La no diferenciabilidad de estas funciones impide utilizar optimización basada en el gradiente. El desarrollo del algoritmo de retropropagación [46], requiere funciones de activación diferenciables para calcular los gradientes de los pesos con respecto a la función de pérdida y realizar los ajustes a los pesos como se explica en 3

Adicionalmente, en arquitecturas profundas, el uso de funciones de activación no lineales, aumenta la flexibilidad y la capacidad de aproximación de la red, otorgándole la habilidad de modelar funciones complejas y resolver problemas más allá de aquellos que solo representan relaciones lineales. Sin una función de activación, o si se utiliza solo la función identidad (3), la red se reduciría a una combinación lineal de las entradas, lo cual limitaría su capacidad para aprender patrones no lineales y la restringiría a relaciones simples [45].

Algunas de las funciones de activación más utilizadas en la práctica, por su diferenciabilidad y capacidad de representar relaciones no lineales, incluyen [40] [38]:

- **Sigmoide**: común en problemas de clasificación binaria. La sigmoide se define en (2).
- **ReLU (Rectified Linear Unit)**: una función ampliamente usada que mitiga el problema de gradientes evanescentes.

$$\sigma(x) = \max(0, x) \tag{8}$$

- **Tanh (tangente hiperbólica**: que escala los valores entre -1 y 1, ayudando a reducir valores extremos en la salida.

$$\sigma(x) = \tanh(x)$$

Aproximación universal

Una de las propiedades fundamentales de las redes neuronales es su capacidad de aproximación de funciones. De acuerdo con uno de los teoremas de aproximación universal, una red neuronal con al menos una capa oculta y una función de activación no lineal continua puede aproximar cualquier función continua con un nivel de precisión arbitrario en un intervalo cerrado, dado un número suficiente de neuronas en la capa oculta [45] [47]. Esto se traduce en que las redes neuronales tienen el potencial de representar una gran variedad de funciones, aunque su desempeño en la práctica dependerá de la arquitectura de la red , las funciones de activación y de los métodos de entrenamiento.

Este resultado se puede expresar formalmente como sigue [48]:

Sea $f : \mathbb{R}^n \to \mathbb{R}$ una función continua en un intervalo cerrado $K \subset \mathbb{R}^n$, y sea $\epsilon > 0$. Entonces, existe una red neuronal de una capa oculta con un número finito de neuronas y una función de activación adecuada, como la función sigmoide (2), tal que la red neuronal $F(x)$ aproxima $f(x)$ en el sentido de que:

$$|f(x) - F(x)| < \epsilon, \quad \forall x \in K.$$

La red neuronal $F(x)$ con una capa oculta se puede escribir de la forma:

$$F(x) = \sum_{i=1}^{N} \alpha_i \, \sigma\left(w_i^T x + b_i\right),$$

donde:

N es el número de neuronas en la capa oculta,
$\alpha_i \in \mathbb{R}$ son los pesos de salida de la capa oculta,
$w_i \in \mathbb{R}^n$ son los vectores de peso que conectan las entradas con la i-ésima neurona de la capa oculta,

$b_i \in \mathbb{R}$ es el sesgo o *bias* de la i-ésima neurona,
σ es una función de activación no lineal, como la sigmoide o la ReLU.

Interpretación y condiciones

- Función de activación: Para que el teorema sea aplicable, la función de activación σ debe ser no lineal y continua. Ejemplos comunes incluyen la función sigmoide y la función tangente hiperbólica.
- Intervalo cerrado: El teorema se aplica a funciones definidas en intervalos compactos (acotados y cerrados), aunque extensiones del teorema han permitido aplicaciones más amplias.
- Limitaciones: Aunque el teorema garantiza la existencia de una aproximación, no establece el número mínimo de neuronas N requerido ni cómo encontrarlas en la práctica. Además, el teorema no asegura una velocidad eficiente de convergencia para el entrenamiento de la red.

Entrenamiento de las redes neuronales

El objetivo del entrenamiento de las redes neuronales es optimizar la función de pérdida a través del ajuste de los parámetros de la red $\{W^{(l)}, b^{(l)}\}$ en cada capa, los cuales determinan la función aproximada por la red y por lo tanto el comportamiento esta. A través de un algoritmo iterativo de optimización, como el descenso de gradiente explicado en 3, se mide la diferencia entre las predicciones de la red y los valores reales con el propósito de minimizar la función objetivo. La implementación de este proceso en redes neuronales profundas se realiza mediante el algoritmo de retropropagación, que permite calcular eficientemente los gradientes de la función de pérdida respecto a cada peso en la red [35].

Algoritmo de retropropagación

El algoritmo de **retropropagación** (BP, *Back Propagation*) aplica el descenso de gradiente en todas las capas de la red, calculando los gradientes de los pesos a través de la regla de la cadena y aprovechando el principio de diferenciación de funciones compuestas [45]. Este algoritmo consta de dos fases que en conjunto permiten adaptar los pesos de cada capa en función de su contribución al error total.

- **Propagación hacia adelante y cálculo de la pérdida**. Durante el entrenamiento, la entrada x de la red neuronal se propaga hacia adelante a través de las capas para obtener una predicción $\hat{y}$. Este proceso o *forward pass* implica aplicar las funciones de activación y multiplicar las entradas por los pesos en cada capa. Posteriormente, se calcula la función de pérdida $L(y, \hat{y})$, que compara la predicción $\hat{y}$ con la salida esperada y. Cuando se utiliza el SGD, el proceso se realiza sobre un *minibatch*.

- **Cálculo de gradientes y BP** Esta fase inicia con el cálculo del gradiente de la función de pérdida respecto a las salidas de la última capa de la red. Este gradiente se propaga hacia atrás, capa por capa, mediante la regla de la cadena, hasta llegar a la capa de entrada. Para una red con n capas, este proceso de BP de los gradientes permite actualizar los pesos $\boldsymbol{W}^{(i)}$ de cada capa i en función de sus contribuciones al error total.

Para una capa i, la actualización de sus pesos $\boldsymbol{W}^{(i)}$ y sesgos $\mathbf{b}^{(i)}$ mediante el descenso de gradiente se expresa como:

$$\boldsymbol{W}^{(i)} \leftarrow \boldsymbol{W}^{(i)} - \eta \frac{\partial L}{\partial \boldsymbol{W}^{(i)}}, \quad \mathbf{b}^{(i)} \leftarrow \mathbf{b}^{(i)} - \eta \frac{\partial L}{\partial \mathbf{b}^{(i)}}$$

donde η es la tasa de aprendizaje, y $\frac{\partial L}{\partial \boldsymbol{W}^{(i)}}$ y $\frac{\partial L}{\partial \mathbf{b}^{(i)}}$ son los gradientes de la función de pérdida con respecto a los pesos y sesgos de la capa i.

Optimización estocástica y retropropagación

Para reducir el costo computacional, especialmente en redes profundas, se emplea la variante de descenso de gradiente estocástico, en la cual los gradientes se calculan sobre porciones pequeñas del conjunto de entrenamiento o *minibatches* de datos en lugar de utilizar el conjunto de datos completo. Esta estrategia adicionalmente a permiter actualizaciones de los pesos de manera más frecuentes, también introduce cierto ruido que puede ayudar a evitar mínimos locales, mejorando la generalización de la red.

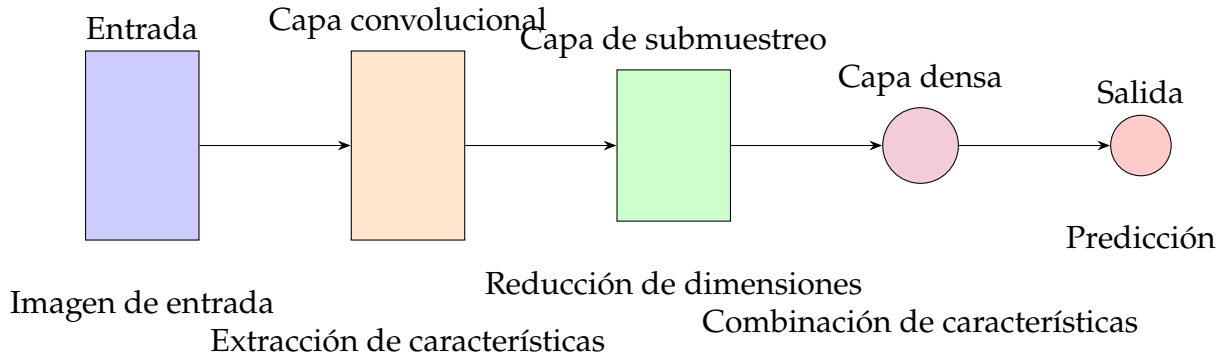
El proceso de retropropagación combinado con el descenso de gradiente estocástico se expresa en los siguientes pasos básicos:

1. Selección de un minibatch.
2. Propagación hacia adelante utilizando los datos para obtener las predicciones.
3. Cálculo del valor de la función de pérdida y sus gradientes con respecto a los pesos de cada capa mediante la retropropagación.
4. Actualización de los pesos y sesgos de cada capa utilizando los gradientes obtenidos y un parámetro denominado de tasa de aprendizaje.

Regularización del entrenamiento

Para mejorar la capacidad de generalización del modelo y mitigar el sobreajuste, se emplean generalmente técnicas de regularización, que se integran en el proceso de entrenamiento para ayudar a mantener la capacidad predictiva de este sin sacrificar precisión [45].

Figura 3: Estructúra y funciones de una red convolucional



- **Enmascaramiento aleatorio:** Durante el entrenamiento, se desactivan de manera aleatoria algunas unidades de procesamiento para reducir la dependencia entre estas y fomentar la independencia en el aprendizaje.
- **Regularización ℓ_2:** Consiste en añadir una penalización proporcional al cuadrado de los pesos, reduciendo su magnitud y mejorando la generalización del modelo.
- **Normalización por lotes:** Se aplica normalización de los datos en cada minibatch, lo que facilita la propagación del gradiente y acelera la convergencia.

Redes neuronales convolucionales

Las **redes neuronales convolucionales** (CNN, *Convolutional Neural Networks*) son una arquitectura de redes neuronales especialmente diseñada para el procesamiento de datos con una estructura matricial o de rejilla, en los que existen relaciones espaciales entre los elementos, como las imágenes, que se organizan en una matriz bidimensional de píxeles. Su capacidad para extraer y aprender características y patrones en los datos proviene de la aplicación de filtros mediante la operación de convolución. La estructura de una CNN puede verse en la figura 3, y consta de las siguientes partes:

Capas convolucionales: Las capas convolucionales son los componentes básicos de las CNN, y tienen la función de aplicar filtros o *kernels* que se pasan a modo de ventana deslizante sobre pequeñas regiones de la entrada, aplicando la operación de convolución en cada posición, lo que permite detectar patrones locales como bordes, texturas y estructuras en la imagen. Matemáticamente, una convolución en una imagen de entrada I con un filtro K se define como [36] [45]:

$$(I * K)(x, y) = \sum_i \sum_j I(x + i, y + j) \cdot K(i, j), \tag{9}$$

donde (x, y) representa una posición específica en la imagen, e i, j son índices que recorren la vecindad definida por el tamaño del filtro. El resultado de deslizar el filtro

por toda la imagen produce una *mapa de características* que destaca las características detectadas.

Función de activación: Tras la operación de convolución, se aplica una función de activación no lineal a cada valor del mapa de características, generalmente la función *ReLU*, definida por la ecuación (8), que además de introducir no linealidad en el modelo, le permite reducir la sensibilidad a valores negativos en el mapa de características.

Capas de submuestreo: Las capas de *pooling* o submuestreo se utilizan para reducir la dimensión espacial de los mapas de características, manteniendo solo las características más relevantes, reduciendo la cantidad de parámetros y el costo computacional. La técnica de *max-pooling*, que selecciona el valor máximo en una región específica, es una de las más comunes [36] [45]:

$$P(x, y) = \max_{(i,j)\in R} h^{(l)}(x + i, y + j),$$

donde R representa la región de submuestreo, y $h^{(l)}$ el valor en la posición $(x+i, y+j)$ en el mapa de características. Esta operación preserva las características importantes y contribuye a una mayor invariancia a pequeñas variaciones en la entrada.

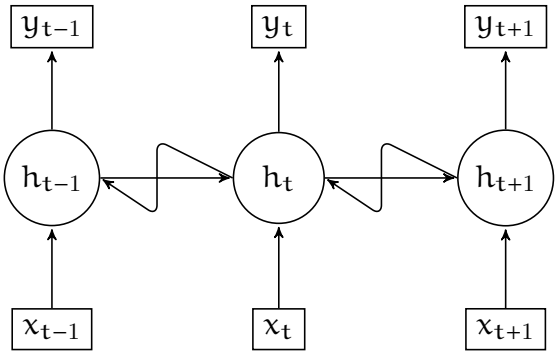
Capas densas: En la etapa final de una CNN, se incluyen una o más capas completamente conectadas (densas), que constituyen una red del tipo FFNN (1) que combina las características detectadas en las capas previas y realizan la clasificación final.

Funcionamiento de las redes convolucionales

Las CNNs procesan los datos de entrada aplicando secuencias de capas convolucionales y de submuestreo, lo que permite detectar patrones presentes en los datos. Las primeras capas suelen aprender características de bajo nivel, como bordes y texturas, mientras que las capas más profundas combinan estas características en representaciones de alto nivel, como formas y patrones específicos de objetos. Estas representaciones permiten que el modelo distinga entre diferentes categorías de objetos al capturar las características distintivas de cada clase [36].

El entrenamiento de la CNN se realiza ajustando los pesos utilizando el algoritmo de BP. Tras los pasos de convolución, activación y submuestreo las características extraídas pasan por las capas densas y finalmente a una capa de salida que genera una predicción. El error entre la predicción y la etiqueta se calcula utilizando una función de pérdida como la de entropía cruzada definida en la ecuación (1). En seguida, la retropropagación ajusta los pesos de todos los filtros y capas totalmente conectadas de la red mediante un método de optimización con el descenso de gradiente.

Figura 4: Diagrama de una RNN - (Por ASW)



Redes neuronales recurrentes

Las **redes neuronales recurrentes** (RNN, *Recurrent Neural Networks*) son un tipo de arquitectura neuronal especializada en el procesamiento de datos secuenciales. A diferencia de las redes neuronales tradicionales, que asumen independencia entre las entradas, las RNN utilizan una estructura que permite la persistencia de información a través de pasos temporales, lo cual contribuye a las tareas donde cada dato está relacionado con los anteriores, como en el procesamiento de texto, reconocimiento de voz y series temporales [34]. La figura 4 muestra una RNN.

Mecanismo de recurrencia

El *mecanismo de recurrencia*, permite que cada unidad de procesamiento conserve la información de su estado anterior y la utilice para procesar la siguiente entrada en la secuencia. Esto se logra mediante la introducción de una conexión recurrente, que alimenta la salida de la red en el tiempo $t-1$ como entrada adicional en el tiempo t [45]. Sea x_t la entrada en el tiempo t, h_t el estado oculto en el tiempo t, y y_t la salida en el tiempo t. El estado oculto h_t se actualiza de acuerdo a la ecuación [49]:

$$h_t = f(W_h \cdot x_t + U_h \cdot h_{t-1} + b_h), \tag{10}$$

en donde:

W_h y U_h son matrices de pesos para la entrada x_t y el estado anterior h_{t-1}, respectivamente,
b_h es el sesgo o *bias* de la red,
f es una función de activación, típicamente una tangente hiperbólica ($\tanh$) o la función ReLU.

La salida en el tiempo t se calcula como:

$$y_t = g(W_y \cdot h_t + b_y),$$

aquí W_y es la matriz de pesos que conecta el estado oculto con la salida y g es una función de activación adecuada para la tarea (por ejemplo, softmax para clasificación).

Entrenamiento y problema del gradiente evanescente

Durante el entrenamiento se utiliza una variación del algoritmo de retropropagación conocida como *Backpropagation Through Time* (BPTT), en la que se "desdobla" la RNN a lo largo del tiempo para conocer la influencia de cada paso en los siguientes. La retropropagación calcula los gradientes a lo largo de toda la secuencia.

Uno de los problemas principales al entrenar RNN es el desvanecimiento del gradiente, que ocurre durante la BPTT. Debido a la multiplicación sucesiva de gradientes en cada paso temporal, el gradiente puede llegar a ser muy pequeño, dificultando la actualización efectiva de los pesos, afectando las dependencias a largo plazo en secuencias extensas y limitando la capacidad de las RNN tradicionales para capturar patrones a gran escala en los datos [45].

Variantes de redes neuronales recurrentes

Para superar las limitaciones de las RNNs estándar, se han desarrollado variantes arquitectónicas como las *Long Short-Term Memory* (LSTM) y las *Gated Recurrent Units* (GRU), que integran mecanismos de compuertas para regular el flujo de información a través de la red y reducir el problema del gradiente evanescente.

Long Short-Term Memory (LSTM). Las LSTM introducen una estructura de memoria que permite conservar y olvidar información de manera controlada. Cada unidad LSTM incluye una célula de memoria que mantiene la información relevante durante pasos largos, junto con tres tipos de puertas [45]:

- **Puerta de entrada** que controla la porción de la entrada nueva que debe almacenarse en la célula de memoria,
- **Puerta de olvido** que decide qué información debe desecharse de la célula de memoria,
- **Puerta de salida** que regula que parte de la memoria interna se debe enviar como salida de la unidad.

Estas compuertas se formalizan con funciones sigmoide y tangente hiperbólica para calcular los valores que se conservan o descartan en cada paso de la secuencia.

Gated Recurrent Unit (GRU) Las GRUs son una simplificación de las LSTM que combinan las puertas de entrada y olvido en una sola, reduciendo el número de parámetros y acelerando el entrenamiento. Aunque son más simples, las GRUs suelen lograr un rendimiento similar al de las LSTM en muchas tareas, por lo que son una opción bastante utilizada en arquitecturas recurrentes [38].

Redes neuronales convolucionales recurrentes

Las redes neuronales convolucionales recurrentes (CRNN) combinan capas convolucionales con capas recurrentes, aprovechando las capacidades de extracción de características locales de las CNN junto con la habilidad de las RNN para procesar dependencias secuenciales. Esta arquitectura es útil en tareas donde los datos presentan tanto estructura espacial como temporal, como en el seguimiento de imágenes, el reconocimiento de texto documentos escaneados, la transcripción de audio en texto y la clasificación de acciones en secuencias de vídeo [49].

Arquitectura de las CRNN

Una CRNN típica se integra mediante tres componentes principales: la sección convolucional, la sección recurrente y la sección de salida.

Sección convolucional La sección convolucional de una CRNN aplica una serie de filtros (kernels) sobre la entrada, que puede ser una imagen o un espectrograma de audio, para extraer características locales. Las capas convolucionales producen mapas de características que resaltan patrones o detalles específicos en diferentes regiones de la entrada. Esta sección también puede incluir capas de submuestreo, para reducir la dimensión y resumir la información espacial, y capas de normalización que estabilizan el proceso de aprendizaje. Formalmente, el mapa de convolucional de características se calcula como se mostró en la ecuación de convolución (9).

Sección recurrente La sección recurrente toma como entrada los mapas de características generados por la sección convolucional y aplica una red recurrente (como una LSTM o GRU) para capturar dependencias temporales en la secuencia. Dado el hecho de que los mapas de características de la sección convolucional pueden considerarse como una secuencia de vectores (por ejemplo, columnas de una imagen procesada), la red recurrente procesa cada vector secuencialmente, permitiendo modelar las relaciones espaciales o temporales entre ellos. Para un mapa de características f_t en el tiempo t y una RNN con estado oculto h_t, la actualización del estado en la capa recurrente se realiza de acuerdo con la ecuación (10).

Sección de salida La sección de salida en una CRNN puede variar dependiendo de la tarea específica. En tareas de clasificación de secuencias, la salida final puede ser un vector que representa la probabilidad de cada clase; en problemas de reconocimiento de secuencias, como la transcripción de texto, la salida puede ser una secuencia de predicciones obtenida mediante una capa *softmax* aplicada a cada paso recurrente, o una técnica de alineación como la clasificación temporal conexionista (CTC, *Connectionist Temporal Classification*).

La ventaja principal de las CRNNs es su capacidad para combinar información espacial y secuencial, lo que las hace efectivas en tareas complejas. Sin embargo, su limitación reside en que suelen ser más costosas en términos de cómputo y memoria que las CNNs o RNNs individuales, lo cual puede impedir su uso en entornos de recursos limitados. Las técnicas de cuantización y binarización, como las que se exploran en esta investigación pueden hacerlas más viables en aplicaciones móviles y embebidas.

5. Compresión de redes neuronales

La compresión de redes neuronales agrupa un conjunto de métodos y técnicas utilizados para reducir la complejidad y los requerimientos de procesamiento y memoria de los modelos de redes neuronales, manteniendo al mismo tiempo su precisión y desempeño en la predicción [50] [39], haciéndolos más eficientes y viables para su implementación en dispositivos con recursos limitados, como teléfonos móviles, sensores embebidos y otros dispositivos IoT [45]. La idea básica de la compactación es reemplazar el modelo original por uno más pequeño que requiera menos memoria para almacenar sus valores y menor tiempo de ejecución para evaluar sus entradas [36].

Primeros métodos de compresión

El interés por la compactación de redes neuronales comenzó con técnicas pioneras como el método de *brain damage* de LeCun (1990) [41], que introdujo un enfoque de poda basado en la sensibilidad de los pesos. Este método elimina pesos con poco impacto en la precisión del modelo, lo cual disminuye la cantidad de operaciones y reduce su tamaño sin afectar sustancialmente el rendimiento. Posteriormente, se desarrollaron métodos similares, como *brain surgeon* de Hassibi y Stork (1993) [51], que incorporó una poda más refinada basada en la segunda derivada de la función de error.

Clasificación de los métodos de compresión

Aunque no se ha encontrado aún en la literatura un consenso universal para la forma de clasificar los métodos de compresión de redes neuronales, de acuerdo con su enfoque y objetivo, se pueden identificar varias categorías principales:

- **Poda de redes (*Pruning*)**: Son métodos mediante los cuales se eliminan conexiones o neuronas redundantes en la red neuronal, reduciendo de esta manera la complejidad del modelo sin impactar de manera significativa su desempeño. Este proceso se basa en la identificación de parámetros que tienen un impacto mínimo en el rendimiento del modelo. Algunos enfoques de poda incluyen la poda basada en magnitud de pesos, la poda estructurada (donde se eliminan bloques completos como filtros) y la poda por sensibilidad.

- **Factorización de tensores y descomposición de matrices**: Estos métodos buscan representar los tensores de pesos en una forma más compacta mediante descomposiciones como SVD (*Singular Value Decomposition*) o descomposiciones de Tucker [52]. Esto reduce la redundancia y el almacenamiento de los pesos sin eliminar por completo las conexiones en la red.

- **Cuantización de modelos**: La cuantización es particularmente efectiva para modelos en inferencia y ha mostrado mantener el rendimiento gracias a que permiten controlar el impacto en la reducción de la precisión. La cuantización se describe en la siguiente sección 6.

- **Binarización de redes**: La binarización es una forma agresiva de cuantización en la que los pesos y activaciones se representan con sólo dos valores. La binarización se trata a detalle en 7

- **Destilación de modelos**: En la destilación del conocimiento de las redes, un modelo grande y complejo (red maestra) se utiliza para entrenar un modelo más pequeño y eficiente (red estudiante) mediante aprendizaje supervisado [53]. El modelo estudiante aprende a replicar el comportamiento del modelo maestro, alcanzando un nivel de precisión similar en una estructura más compacta y eficiente. lo cual permite mantener el rendimiento mientras se reduce el tamaño y la complejidad de la red [54], y es especialmente útil donde es posible entrenar el modelo maestro en dispositivos de alto poder de procesamiento y la implementación final se hace en dispositivos con recursos limitados.

- **Optimización arquitectónica y arquitectura compacta**: Algunos métodos de compactación involucran el diseño de arquitecturas optimizadas para eficiencia, este tipo de optimización se logra mediante redes neuronales más pequeñas o estructuras especializadas, como MobileNet, publicada por Howard [55], SqueezeNet de Iandola [56] y EfficientNet de Nakod [57] diseñadas con bloques de convolución reducida y operaciones de profundidad, que operan con número de parámetros significativamente menor. Otros como Binary DenseNet [3], MeliusNet de Bethge [9], introducen conexiones residuales y bloques densos. En el modelo del propio autor [30], se implementan tres versiones de conexiones residuales para perceptrones multicapa.

6. Cuantización de redes neuronales

La cuantización es una técnica ampliamente utilizada en modelos de bajos recursos. Aplicado a las redes neuronales, es un proceso mediante el cual se reduce la precisión de los valores de los pesos y activaciones [58], de modo que puedan representarse con menor cantidad de bits [59]. Este método permite disminuir el uso de memoria y la carga computacional durante la inferencia.

Fundamentos de la cuantización

La cuantización parte de la observación de que los modelos de aprendizaje profundo entrenados suelen utilizar valores de punto flotante de 32 bits para representar pesos y activaciones, pero en muchos casos no requieren tal precisión para obtener buenos resultados. La cuantización reduce la representación de estos valores a menor precisión, usando comúnmente valores de 8 bits, 4 o incluso 2 bits [60]. En una cuantización binaria, los valores se reducen a solo dos niveles (por ejemplo, +1 y -1), mientras que la cuantización de 8 bits permite hasta $2^8 = 256$ valores distintos, lo cual aún es significativamente menor que utilizando punto flotante.

Dado un valor en punto flotante x perteneciente al rango $[x_{min}, x_{max}]$, un valor cuantizado $\tilde{x}$ se define por la función:

$$\tilde{x} = \text{round}\left(\frac{x - x_{min}}{s}\right) \cdot s + x_{min},$$

donde s es el paso de cuantización o escala, que se calcula como:

$$s = \frac{x_{max} - x_{min}}{2^b - 1},$$

siendo b la cantidad de bits de la representación cuantizada.

Enfoques a la cuantización

Existen diferentes técnicas de cuantización que se diferencian por la precisión y la forma en que distribuyen los valores cuantizados [61]:

- **Cuantización Uniforme**: La cuantización uniforme mapea todos los valores a intervalos equidistantes en el rango permitido, como se muestra en la fórmula anterior. Este tipo de cuantización es sencilla de implementar y comúnmente usada en redes neuronales debido a que permite una implementación eficiente de aritmética de punto fijo [59].

- **Cuantización No Uniforme**: En la cuantización no uniforme, los intervalos no son equidistantes, sino que se ajustan a la distribución de los valores en la red. Esto permite una representación más precisa para valores que ocurren con mayor frecuencia. Un método típico es la cuantización logarítmica, que distribuye los valores de forma logarítmica en el rango disponible.

- **Cuantización Post-Entrenamiento**: Este enfoque aplica la cuantización después de entrenar el modelo en precisión completa convirtiéndolo directamente a punto fijo sin necesidad de reentrenarlo [59]. Es un método rápido y eficiente, adecuado cuando la precisión del modelo no es extremadamente crítica, aunque en algunos casos puede causar una ligera degradación en el rendimiento.

- **Cuantización con Entrenamiento (QAT)**: En este método, la cuantización se incorpora durante el proceso de entrenamiento, permitiendo que la red neuronal aprenda a compensar los efectos de la menor precisión. Esto se logra introduciendo funciones de activación y capas de cuantización simuladas, que imitan los efectos de la cuantización en los valores intermedios y finales del modelo. La QAT ha demostrado ser muy efectivo en modelos donde la precisión es crítica [53].
- **Binarización**: La binarización se discute en la siguiente sección (7).

Aunque en teoría, la aritmética de tipos de datos más pequeños puede ser más rápida de calcular y las operaciones de punto fijo más eficientes que las de punto flotante. Sin embargo estas operaciones han sido altamente optimizadas en los CPU y GPU modernos, por lo que el uso de punto fijo puede no obtener las ventajas de velocidad de ejecución esperadas.

7. Redes neuronales binarias

La binarización de redes neuronales es un proceso de compresión extrema donde los pesos y activaciones se representan con solo dos valores, típicamente $+1$ y -1 o 0 y 1. Al reducir los datos a un nivel binario, la binarización permite una disminución significativa del uso de memoria y del costo computacional. Teóricamente, la reducción de memoria que se puede alcanzar al utilizar valores de 1 bit es de 32x en comparación con representaciones de punto flotante de 32 bits. Este enfoque es particularmente útil en aplicaciones de aprendizaje profundo en tiempo real en dispositivos móviles o embebidos [7] [4] [8].

Fundamentos de la Binarización

En una red neuronal binarizada, cada peso w y cada activación a en la red se limitan a dos valores posibles, lo cual simplifica las operaciones aritméticas. Para transformar el valor de una variable real a un valor binario, se proponen dos funciones de binarización [7]

Función signo

$$x^b = \text{signo}(x) = \begin{cases} +1, & \text{si } x \geqslant 0, \\ -1, & \text{si } x < 0 \end{cases}$$

donde para cada peso o activación original $x \in \mathbb{R}$, su valor binarizado es x^b.

Binarización estocástica

$$x^b = \begin{cases} +1, & \text{con probabilidad } p = \sigma(x), \\ -1, & \text{con probabilidad } 1 - p \end{cases}$$

donde σ es la función sigmoide dura que se define como

$$\sigma(x) = \text{clip}\left(\frac{x+1}{2}, 0, 1\right) = \max\left(0, \min\left(1, \frac{x+1}{2}\right)\right)$$

Aunque la binarización estocástica es en principio más atractiva que la función signo, el hecho de requerir de *hardware* para generar bits aleatorios para cuantizar, la hace más difícil de implementar, por cual se utiliza generalmente la función signo [2], la cual es de implementación directa y en la práctica produce buenos resultados. El uso de esta función permite reemplazar operaciones de multiplicación y suma en punto flotante por operaciones de suma y resta binarias como XNOR para la multiplicación binaria y POPCOUNT que realiza el conteo de bits, las cuales son considerablemente más eficientes en términos de computación [2] [4] [62] . Esta combinación de operaciones puede obtener reducciones de hasta 50 % en los tiempos de ejecución [63] .

Entrenamiento de Redes Binarizadas

Los primeros trabajos con redes binarizadas, mostraron que el algoritmo BP y el SGD no se pueden aplicar directamente a estas redes, pues no hay manera de realizar actualizaciones de pesos en incrementos pequeños, por lo que se utilizaban variaciones de inferencia Bayesiana para realizar el entrenamiento [62].

La metodología para entrenar BNN desarrollada por [7] es el fundamento de la mayor parte de técnicas de binarización utilizadas hoy en día. El entrenamiento de la red binaria produce mejores resultados que la binarización post-entrenamiento. En las BNN, tanto los pesos como las activaciones se binarizan, lo cual reduce la memoria necesaria y la complejidad computacional gracias al uso de operaciones entre bits.

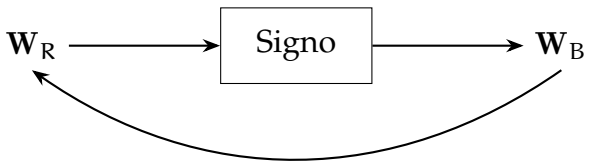
Binarización de pesos

Entrenar una BNN utilizando BP con un método basado en SGD utilizando valores binarios permite calcular una pérdida más representativa durante la optimización, en comparación con binarizar los pesos al final del entrenamiento. El cálculo del gradiente de la pérdida con respecto a los pesos binarios no presenta dificultad, sin embargo estos imposibilitan la actualización con métodos basados en SGD, pues los pequeños ajustes en los pesos que se requieren no pueden hacerse con valores binarios.

Par resolver el problema, la estrategia consiste en mantener un conjunto de pesos con valores reales o latentes $\mathbf{W}_R$, los cuales son binarizados dentro la red para obtener los pesos binarios $\mathbf{W}_B$. De esta manera, $\mathbf{W}_R$ se puede actualizar mediante retropropagación utilizando las actualizaciones incrementales de SGD. Durante la inferencia los pesos latentes $\mathbf{W}_R$ no son necesarios, por lo que solo $\mathbf{W}_B$ se almacena y utiliza. La binarización se realiza sobre todos los pesos utilizando la función signo [62]

$$\mathbf{W}_B = \text{signo}(\mathbf{W}_R)$$

Figura 5: Binarización de pesos



Lo que resulta en un tensor con valores $+1$ y -1.

Estimador de paso directo

La binarización de los pesos y activaciones introduce un problema en el cálculo del gradiente durante la BP, ya que la derivada de la función signo(x) cero en casi todos los puntos, o no está definida; haciéndola aparentemente incompatible con BP, pues el gradiente del costo con respecto a las cantidades antes de la discretización sería cero. Para solucionar esto, se utiliza el **estimador de paso directo** (STE, *Straight Trhough Estimator*), propuesto como una aproximación heurística que permite el cálculo de gradientes en redes binarizadas.

El STE permite que el gradiente fluya a través de la función de binarización al reemplazar la derivada de la función no diferenciable por una aproximación. Sea x un valor de peso en punto flotante y $x^b = \text{signo}(x)$ el valor binarizado; en lugar de calcular el gradiente con respecto a x^b, se utiliza la derivada de x como una aproximación directa [62]:

$$\frac{\partial L}{\partial x} \approx \frac{\partial L}{\partial x^b},$$

donde L es la función de pérdida.

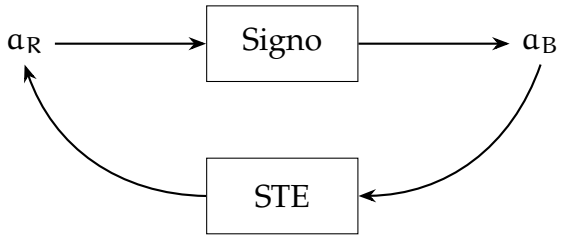
Esta aproximación del gradiente se utiliza para actualizar los pesos reales. La binarización se puede visualizar como una capa en si misma. Los pesos pasan por la capa de binarización que evalúa el signo de los valores en la propagación hacia adelante del BP y aplica la función de identidad durante la retropropagación, por lo que para una capa, el gradiente se puede verse como

$$\frac{\partial L}{\partial \mathbf{W}_R} = \frac{\partial L}{\partial \mathbf{W}_B}$$

Como las actualizaciones del gradiente afectan los valores reales $\mathbf{W}_R$ sin cambiar los valores de $\mathbf{W}_B$, como se observa en la figura 5, si los valores de $\mathbf{W}_R$ no están acotados, estos pueden acumular valores muy grandes, por esa razón los valores se recortan entre -1 y 1 para que los valores de $\mathbf{W}_R$ se mantengan cercanos a los de $\mathbf{W_B}$.

Figura 6: Binarización de activaciones

Propagación hacia adelante



Retropropagación

Binarización de las activaciones

Para la binarización de las activaciones, estas pasan a través de la función signo y se utiliza el STE durante la retropropagación, de manera similar a la binarización de los pesos. La función signo se utiliza como función de activación en la red binaria. Para obtener buenos resultados, el gradiente de la activación se necesita cancelar en la retropropagación si su valor es muy grande, utilizando

$$\frac{\partial L}{\partial a_R} = \frac{\partial L}{\partial a_B} \times 1_{|a_R| \leqslant 1}$$

donde a_R es el valor real de entrada a la función de activación y a_B es la salida binarizada de esta y $1_{|x| \leqslant 1}$ es una función indicadora que devuelve 1 si $|x| \leqslant 1$ y 0 en caso contrario. Esta expresión preserva la información del gradiente y lo cancela cuando x es muy grande. Esto permite que el gradiente fluya "a través" de la función de binarización, lo cual hace posible aplicar los métodos de optimización de gradiente descendente [62].

La derivada de $1_{|x|} \leqslant 1$ se puede ver como la propagación del gradiente a través de la función Htanh dura, por lo cual esta define al STE únicamente durante la retropropagación [5] [7] como se observa en la figura 5.

$$\text{Htanh}(x) = \text{clip}(x, -1, 1) = \max(-1, \min(1, x)) \tag{11}$$

Operaciones entre bits

Con el uso de valores binarios, el producto punto entre los pesos y las activaciones se puede reducir a operaciones entre bits. Los valores binarios son $+1$ y -1, los cuales se codifican como 1 y 0 respectivamente. De esta manera el producto de los valores binarios se puede realizar utilizando la operación lógica XNOR como se ve en la tabla 3

El producto punto, requiere la acumulación de los productos entre los valores resultado del XNOR. Con la codificacion de los valores, esto puede hacerse contando el número

Tabla 3: Equivalencia entre el producto binario y el XNOR. Elaboración ASW

Codificación	Valor	Codificación	Valor	XNOR	Producto
0	(-1)	0	(-1)	1	(+1)
0	(-1)	1	(+1)	0	(-1)
1	(+1)	0	(-1)	0	(-1)
1	(+1)	1	(+1)	1	(+1)

de bits con valor 1 en un grupo de productos del XNOR, multiplicando el valor por 2 y restando el número total de bits, lo que produce un valor entero. Los conjuntos de instrucciones de los procesadores incluyen una instrucción POPCOUNT que cuenta el número de bits en un registro de CPU.

Las operaciones entre bits son más simples de procesar que la multiplicación y adición en punto flotante o fijo, lo cual resulta en una ejecución más rápida y menor uso de recursos del procesador, por ejemplo, es posible realizar un XNOR entre dos registros de 64 bits con una sola instrucción, mientras que con estos mismos registros solo se realizarían dos multiplicaciones de 32 bits. Dependiendo del procesador, se ha observado un incremento de entre 2 hasta 23 veces en velocidad utilizando operaciones entre bits, sobre las de punto flotante.

Exactitud de las BNN

A pesar de las ventajas de las redes neuronales binarias en términos de eficiencia, la reducción de precisión inherente puede afectar su rendimiento en tareas complejas. La BNN original de [2] lograba una exactitud 3 % menor al valor de referencia para el conjunto CIFAR-10. Por ello, se han desarrollado varios métodos para mitigar esta pérdida, los cuales se pueden clasificar en varias categorías principales:

- **Minimización del error de cuantización**
  La minimización del error de cuantización busca reducir la discrepancia entre los pesos binarizados y los originales en punto flotante. Un enfoque común es ajustar los pesos en punto flotante para que sus versiones binarizadas retengan la mayor cantidad posible de información. El uso de factores de escala para las activaciones y los pesos en XNOR-Net [8], el empleo de matrices de Hadamard en HadaNet [64] o el método adaptativo de Zhao con DA-BNN o la Do-Re-Fa Net de Zhou utilizan binarización y cuantización a través de funciones de cuantización especificas.

- **Mejora de la función de pérdida**
  Las funciones de pérdida estándar pueden ser insuficientes para entrenar redes binarizadas debido a las restricciones en el espacio de valores. En respuesta, se han desarrollado funciones de pérdida adaptadas que penalizan las desviaciones causadas por la binarización. Métodos como los descritos por[11], [19], proponen agregar

distribución o regularización especial a la pérdida. Modelos como Real-to-Bin [65], o Re-Act-Net [66]incorporan un término en la pérdida y un factor de balance, que consideran el efecto de la binarización, mejorando así la precisión del modelo final.

- **Aproximación del gradiente**
  Dado que la derivada de la función de signo es cero, esto impide la actualización de los pesos en el proceso de retropropagación. El estimador de paso directo (Straight-Through Estimator, STE) es un método común para aproximar los gradientes de la función de signo. Sin embargo, el uso de STE dificulta el aprendizaje de pesos cercanos a los límites de –1 y +1, lo que reduce la capacidad de actualización en la retropropagación. GB-Net [67] utiliza un aprendizaje basado en gradientes reales para entrenar redes binarias con funciones de recorte parametrizadas (PCF) y luego sustituye estas funciones por una función de activación binaria escalada (SBAF). BNN-RBNT [19] propone una aproximación hacia atrás basada en la función sigmoide. Bi-Real-Net [68] utiliza una función escalonada polinómica para aproximar la función de signo hacia adelante. CCNN introduce un estimador derivado para aproximar su función de binarización. CBCN diseña una aproximación de gradiente basada en funciones gaussianas [4].

- **Arquitecturas optimizadas para redes binarias**
  Algunas arquitecturas se diseñan específicamente para mejorar el rendimiento de redes neuronales binarizadas, optimizando tanto su estructura como sus componentes. Ejemplos destacados incluyen las XNOR-Net [8], ABC-Net [17], Bi-Real Net [68] y Melius-Net [9], que ajustan la estructura de la red para mejorar su capacidad de representar datos binarios. La XNOR-Net, por ejemplo, utiliza operaciones especializadas para realizar convoluciones binarizadas de manera más eficiente, mientras que Bi-Real Net introduce conexiones residuales que conservan la precisión del modelo al tiempo que mantienen la eficiencia de la binarización.

- **Estrategias de Entrenamiento**
  Varias estrategias de entrenamiento han sido adaptadas para mejorar el rendimiento de redes binarizadas. Entre ellas, la binarización progresiva implementa un proceso gradual en el cual la red se entrena inicialmente con precisión completa y los pesos se binarizan progresivamente a lo largo del entrenamiento, minimizando el impacto de la binarización en el rendimiento final. Asimismo, la destilación de conocimiento se ha aplicado en redes binarizadas, utilizando una red pre-entrenada en punto flotante para guiar el entrenamiento de una versión binaria, lo que mejora la precisión de la red binarizada.

8. Conclusión

Las redes neuronales binarias proporcionan una compresión e incremento en velocidad de inferencia sobre las redes profundas tradicionales, y han generado un creciente interés porque permiten la implementación aplicaciones basadas en modelos de aprendizaje profundo en ambientes con restricciones de almacenamiento, memoria, conectividad y consumo de energía, como los dispositivos de internet de las cosas, móviles y de computación perimetral. Su desventaja principal es la falta de precisión en comparación con las de punto flotante. Sin embargo, desde 2015 se han desarrollado muchas técnicas que han ayudado a mejorar la exactitud preservando tamaños pequeños y alta velocidad de ejecución. A pesar de estos avances, las técnicas de binarización requieren ser estudiadas con mayor profundidad y mayor formalización teórica, pues aunque tienen buen desempeño en la práctica, muchas de estas están basadas en heurísticas y estrategias experimentales cuya eficacia se ha probado empíricamente en entornos computacionales, por lo que una mayor profundidad teórica proporcionaría herramientas y marcos conceptuales que permitirían mejorar el desempeño de estas redes.

Direcciones de correo de los autores

Agustín Solís Winkler: asolisw@uaemex.mx

Referencias

[1] F. Chollet, *Deep Learning with Python*. Manning Publications Company, 2018, ISBN: 9781617294433.

[2] M. Courbariaux, Y. Bengio y J.-P. David, «BinaryConnect: training deep neural Networks with binary weights during propagations,» nov. de 2015.

[3] J. Bethge, H. Yang, M. Bornstein y C. Meinel, «Back to simplicity: how to train accurate BNNs from scratch?,» jun. de 2019. dirección: http://arxiv.org/abs/1906.08637.

[4] C. Yuan y S. S. Agaian, «A comprehensive review of binary neural network,» feb. de 2022. dirección: http://arxiv.org/abs/2110.06804.

[5] H. Qin, R. Gong, X. Liu, X. Bai, J. Song y N. Sebe, «Binary neural networks: a survey,» mar. de 2020. DOI: 10.1016/j.patcog.2020.107281. dirección: http://arxiv.org/abs/2004.03333%20http://dx.doi.org/10.1016/j.patcog.2020.107281.

[6] J. Bethge, H. Yang, M. Bornstein y C. Meinel, «BinaryDenseNet: developing an architecture for binary neural networks,» inf. téc., 2019. dirección: https://github.com/hpi-xnor/BMXNet-v2.

[7] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv e Y. Bengio, «Binarized neural networks: training deep neural networks with weights and activations constrained to +1 or -1,» feb. de 2016. dirección: http://arxiv.org/abs/1602.02830.

[8] M. Rastegari, V. Ordonez, J. Redmon y A. Farhadi, «XNOR-Net: ImageNet classification using binary convolutional networks,» 2016.

[9] J. Bethge, C. Bartz, H. Yang, Y. Chen y C. Meinel, «MeliusNet: can binary neural networks achieve MobileNet-level accuracy?,» ene. de 2020.

[10] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen e Y. Zou, «DoReFa-Net: training low bitwidth convolutional neural networks with low bitwidth gradients,» jun. de 2016. dirección: http://arxiv.org/abs/1606.06160.

[11] W. Tang, G. Hua y L. Wang, «How to train a compact binary neural network with high accuracy,» en *Proceedings of the AAAI Conference on Artificial Intelligence*, 2017.

[12] B. Jacob et al., «Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference,» dic. de 2017.

[13] J. Song, «Binary Generative Adversarial Networks for Image Retrieval,» ago. de 2017.

[14] K. Helwegen, J. Widdicombe, L. Geiger, Z. Liu, K.-T. Cheng y R. Nusselder, «Latent Weights Do Not Exist: Rethinking Binarized Neural Network Optimization,» jun. de 2019.

[15] R. Razani, G. Morin, V. P. Nia y E. Sari, «Adaptive Binary-Ternary Quantization,» sep. de 2019. dirección: http://arxiv.org/abs/1909.12205.

[16] S. Khoram y J. Li, «Adaptive Quantization of Neural Networks,» ICLR, 2018.

[17] X. Lin, C. Zhao y W. Pan, «Towards Accurate Binary Convolutional Neural Network,» nov. de 2017.

[18] B. Zhuang, C. Shen, M. Tan, P. Chen, L. Liu e I. Reid, «Structured Binary Neural Networks for Image Recognition,» *International Journal of Computer Vision*, vol. 130, n.º 9, págs. 2081-2102, sep. de 2019, ISSN: 15731405. DOI: 10.1007/s11263-022-01638-0. dirección: https://arxiv.org/abs/1909.09934v4.

[19] S. Darabi, M. Belbahri, M. Courbariaux y V. P. Nia, «Regularized Binary Network Training,» dic. de 2018. dirección: http://arxiv.org/abs/1812.11800.

[20] Y. Wang, «Binary Neural Networks for Object Detection,» inf. téc., 2019. dirección: http://repository.tudelft.nl/..

[21] H. Zhao, Y. Xiao, J. Han y Z. Zhang, «Compact Convolutional Recurrent Neural Networks via Binarization for Speech Emotion Recognition,» en *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, mayo de 2019, págs. 6690-6694, ISBN: 978-1-4799-8131-1. DOI: 10.1109/ICASSP.2019.8683389.

[22] W. Zheng e Y. Tang, «Binarized Neural Networks for Language Modeling,» Stanford University, Stanford, CA, inf. téc., 2019.

[23] S. Einy, E. Sen, H. Saygin, H. Hivehchi e Y. Dorostkar Navaei, «Local Binary Convolutional Neural Networks' Long Short-Term Memory Model for Human Embryos' Anomaly Detection,» *Scientific Programming*, vol. 2023, págs. 1-11, abr. de 2023, ISSN: 1875-919X. DOI: 10.1155/2023/2426601.

[24] T. Chen, Z. Zhang, X. Ouyang, Z. Liu, Z. Shen y Z. Wang, «Training binary neural networks without batch normalization,» en *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

[25] S. A. Mirsalari, S. Sinaei, M. E. Salehi y M. Daneshtalab, «MuBiNN: Multi-Level Binarized Recurrent Neural Network for EEG signal Classification,» abr. de 2020. dirección: https://arxiv.org/abs/2004.08914v1.

[26] C. Daniel Suarez-Ramirez, M. Gonzalez-Mendoza, L. Chang, G. Ochoa-Ruiz y M. A. Duran-Vega, «A Bop and Beyond: A Second Order Optimizer for Binarized Neural Networks,» 2021.

[27] A. J. Redfern, L. Zhu y M. K. Newquist, «BCNN: A binary CNN with all matrix ops quantized to 1 bit precision,» inf. téc., 2021.

[28] N. Guo, J. Bethge, H. Guo, C. Meinel y H. Yang, «Towards Optimization-Friendly Binary Neural Network,» *Transactions on Machine Learning Research*, 2023. dirección: https://github.com/hpi-xnor/BNext..

[29] Y. Xu, Y. Wang, Zhou Aojun, W. Lin y H. Xiong, «Deep Neural Network Compression with Single and Multiple Level Quantization,» en *The Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18)*, mar. de 2023.

[30] A. S. Winkler, A. L. Chau y S. O. Baltierra, «Results on the Empirical Design of a Residual Binary Multilayer Perceptron Architecture,» en *2023 IEEE Symposium Series on Computational Intelligence (SSCI)*, IEEE, dic. de 2023, págs. 1366-1370, ISBN: 978-1-6654-3065-4. DOI: 10.1109/SSCI52147.2023.10371905.

[31] M. Bandyopadhyay y R. Baral, «Low-Memory Pedestrian Detection Using Binarized Neural Networks,» en *Lecture Notes in Networks and Systems*, vol. 725 LNNS, Springer Science y Business Media GmbH, 2023, págs. 567-581, ISBN: 9789819937332. DOI: 10.1007/978-981-99-3734-9{_}46.

[32] C. Aggarwal, *Artificial Intelligence. A textbook*, 1st. Swtzerland: Springer Nature Switzerland AG, 2021, ISBN: 978-3-030-72357-6.

[33] B. Coppin, *Artificial Intelligence Illuminated*. Jones y Bartlett Publishers, Inc, 2004, ISBN: 0-7637-3230-3.

[34] S. J. Russell y P. Norvig, *Artificial Intelligence A Modern Approach Third Edition*. Upper Saddle River, New Jersey, 2010, ISBN: 978-0-13-604259-4.

[35] T. Munakata, *Fundamentals of the New Artificial Intelligence*, 2nd, D. Gries y F. B. Schneider, eds. London: Springer-Verlag, 2008.

[36] I. Goodfellow, Y. Bengio y A. Courville, *Deep learning*. MIT Press, 2016.

[37] S. Shalev-Shwartz y S. Ben-David, *Understanding machine learning: from theory to algorithms*, First. New York: Cambridge University Press, 2014, ISBN: 978-1-107-05713-5. dirección: http://www.cs.huji.ac.il/~shais/UnderstandingMachineLearning.

[38] J. Heaton, *Artificial Intelligence for Humans. Volume 3: Deep Learning and Neural Networks*, 1.0. Heaton Research, Inc., dic. de 2015, ISBN: 978-1505714340.

[39] R. Abbasi-Asl y B. Yu, «Structural Compression of Convolutional Neural Networks with Applications in Interpretability,» *Frontiers in Big Data*, vol. 4, pág. 73, ago. de 2021, ISSN: 2624909X. DOI: 10.3389/FDATA.2021.704182/BIBTEX.

[40] N. K. Kasabov, *Foundations of neural networks, fuzzy systems, and knowledge engineering*, Second Printing. London: MIT Press, 1996, ISBN: 0262112124.

[41] Y. Le Cun, J. S. Denker y S. A. Solla, «Optimal Brain Damage,» *NIPS*, 1989.

[42] W. S. McCulloch y W. Pitts, «A Logical Calculus of the Ideas Immanent in Nervous Activity,» *BULLETIN OF MATHEMATICAL BIOPHYSICS*, vol. 5, págs. 115-133, 1943.

[43] P. Petersen y J. Zech, *Mathematical theory of deep learning*. jul. de 2024. dirección: https://arxiv.org/abs/2407.18384v2.

[44] F. Rosenblatt, «The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain,» *Psychological Review*, vol. 65, n.º 6, págs. 19-27, 1958.

[45] C. Aggarwal, *Neural Networks and Deep Learning. A textbook*. Springer International Publishing AG, 2018, ISBN: 978-3-319-94462-3.

[46] Y. LeCun, L. Bottou, Y. Bengio y P. Haffner, «Gradient-based learning applied to document recognition,» *Proceedings of the IEEE*, vol. 86, n.º 11, págs. 2278-2324, 1998, ISSN: 00189219. DOI: 10.1109/5.726791. dirección: http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf.

[47] M. Nielsen, *Neural Networks and Deep Learning*, M. Nielsen, ed. 2019. dirección: http://neuralnetworksanddeeplearning.com.

[48] K. Hornik, M. Stinchcombe y H. White, «Multilayer feedforward networks are universal approximators,» *Neural Networks*, vol. 2, n.º 5, págs. 359-366, ene. de 1989, ISSN: 0893-6080. DOI: 10.1016/0893-6080(89)90020-8.

[49] G. Keren y B. Schuller, «Convolutional RNN: an Enhanced Model for Extracting Features from Sequential Data,» feb. de 2016. dirección: http://arxiv.org/abs/1602.05875.

[50] R. R. Nakod, *Model Compression Techniques for Edge AI*, sep. de 2022. dirección: https://embeddedcomputing.com/technology/software-and-os/simulation-modeling-tools/model-compression-techniques-for-edge-ai.

[51] B. Hassibi, D. G. Stork y G. J. Wolff, «Optimal brain surgeon and general network pruning,» *1993 IEEE International Conference on Neural Networks*, págs. 293-299, 1993. DOI: 10.1109/ICNN.1993.298572.

[52] D. Bacciu y D. P. Mandic, «Tensor Decompositions in Deep Learning,» oct. de 2020. dirección: http://www.i6doc.com/en/..

[53] J. O'Neill, «A Survey of Neural Network Compression,» jun. de 2020. dirección: http://arxiv.org/abs/2006.03669.

[54] C. Bucili, R. Caruana y A. Niculescu-Mizil, «Model compression,» en *ACM KDD Conference*, 2006, págs. 535-541.

[55] A. G. Howard et al., «MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications,» abr. de 2017. dirección: https://arxiv.org/abs/1704.04861v1.

[56] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally y K. Keutzer, «SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size,» feb. de 2016. dirección: https://arxiv.org/abs/1602.07360v4.

[57] M. Tan y Q. V. Le, «EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks,» sep. de 2020.

[58] R. Mishra, H. P. Gupta y T. Dutta, «A Survey on Deep Neural Network Compression: Challenges, Overview, and Solutions,» oct. de 2020. dirección: http://arxiv.org/abs/2010.03954.

[59] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. van Baalen y T. Blankevoort, «A White Paper on Neural Network Quantization,» jun. de 2021.

[60] E. Call, «Faster Convolutional Networks,» Tesis doct., Radboud University, Nijmegen, ago. de 2017.

[61] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney y K. Keutzer, «A Survey of Quantization Methods for Efficient Neural Network Inference,» mar. de 2021. dirección: http://arxiv.org/abs/2103.13630.

[62] T. Simons y D. J. Lee, «A review of binarized neural networks,» *Electronics (Switzerland)*, vol. 8, n.º 6, jun. de 2019, ISSN: 20799292. DOI: 10.3390/electronics8060661.

[63] H. Chi, V.-K. Pham, L. Le y T.-K. Tran Thi, «XNOR-Popcount, an Alternative Solution to the Accumulation Multiplication Method for Approximate Computations, to Improve Latency and Power Efficiency,» *JTE*, vol. xx, pág. 20, 2024, ISSN: 2615-9740. DOI: 10.54644/jtexxxxxxx.

[64] Y. Akhauri, «HadaNets: Flexible Quantization Strategies for Neural Networks,» 2019.

[65] B. Martinez, J. Yang, A. Bulat y G. Tzimiropoulos, «Training Binary Neural Networks with Real-to-Binary Convolutions,» mar. de 2020.

[66] Z. Liu, Z. Shen, S. Li, K. Helwegen, D. Huang y K.-T. Cheng, «How Do Adam and Training Strategies Help BNNs Optimization?,» 2021.

[67] C. Sakr, J. Choi, Z. Wang, K. Gopalakrishnan y N. Shanbhag, «True gradient-based training of deep binary activated neural networks via continuous binarization,» en *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018.

[68] Z. Liu, B. Wu, W. Luo, X. Yang, W. Liu y K.-T. Cheng, «Bi-real net: enhancing the performance of 1-bit CNNs with improved representational capability and advanced training algorithm.,» en *Proceedings of the European conference on computer vision (ECCV)*, 2018.

Informaticae Abstracta
Año 2024, Vol. 2, No. 2, pp. 60-74 JUL | DIC 20244

Envolvente Convexa para determinar puntos terminales en Problemas de Enrutamiento de Vehículos Capacitados

Sánchez-Carmona, M.A.; Loza-Hernández, L.

Información del Artículo

Recibido: 6-nov-2024
Aceptado: 17-dic-2024

Palabras clave: envolvente convexa, rutas, enrutamiento de vehículos

Resumen

Se propone el uso de una *envolvente convexa* aplicado a cuatro diferentes escenarios para determinar el subconjunto de puntos más alejados y establecerlos como puntos *terminales*, con el fin de alimentar un modelo de programación lineal entera. Cada escenario esta conformado por un conjunto de puntos distintos con coordenadas geográficas y un punto de destino denominado *depósito*. El modelo propuesto toma de referencia los aspectos más relevantes del Problema de Enrutamiento de Vehículos para generar el conjunto rutas de transporte hacia un depósito para cada escenario. Las rutas obtenidas con este modelo y el uso de la envolvente convexa, para la determinación del subconjunto de vértices terminales, obtienen rutas óptimas dirigidas hacia el depósito en tiempos computacionales cortos.

1. Introducción

El Problema de Enrutamiento de Vehículos (VRP, por sus siglas en inglés), es conocido por ser uno de los problemas combinatorios con una gran variedad de aplicaciones en diversas áreas [1], el cual consiste en determinar rutas óptimas para un conjunto de vehículos que deben atender una serie de clientes desde un depósito, minimizando costos como distancias o tiempos [2], [3]. Se sabe que el VRP tiene sus orígenes en el clásico Problema del Agente Viajero (TSP, por sus siglas en inglés), en el cual un vendedor debe visitar un conjunto de ciudades (puntos) y regresar a la ciudad de partida; existe un costo de viaje desde una ciudad i a una ciudad j, el objetivo es visitar cada ciudad una sola vez, minimizando el costo total del viaje [3]. El TSP tienen distintas variaciones, en [2] se hace mención del denominado Problema del Agente Viajero Múltiple (mTSP), donde la formulación del problema implica tener varios vendedores que parten de un mismo punto y retornar a este, cada vendedor debe visitar distintas ciudades sin que ninguno pase por una ya visitada, por lo que la solución al problema consiste de m rutas.

Partiendo del TSP y del mTSP, el VRP se enfoca en la determinación del conjunto óptimo de rutas que debe realizar un grupo de vehículos para atender a un conjunto determinado de clientes. Es uno de los problemas de optimización combinatoria más importantes y estudiados por más de 60 años, donde su importancia e interés se debe a su relevancia práctica, así como por su dificultad [2], [4].

La envolvente convexa (convex hull), en términos referentes a teoría de gráficas [5], permite identificar la mínima cantidad de vértices que pueden encerrar al resto de ellos, sus aplicaciones en dicho campo son notorios, y bajo este concepto es factible su aplicación en el área del VRP.

El objetivo de este trabajo, es implementar una envolvente convexa al conjunto de datos geográficos en cada escenario para determinar el subconjunto de puntos en donde rutas de transporte pueden iniciar o finalizar. Esto beneficia a un modelo de VRP que determina rutas en un sentido (rutas de ida o regreso), ya que la complejidad del modelo se reduce y por ello resolverlo con métodos exactos se puede realizar en tiempos computacionales cortos.

El presente documento se estructura de la siguiente manera: en *Trabajos relacionados*, se muestran implementaciones de la envolvente convexa al contexto del VRP; en *Formulación del problema* se detalla el funcionamiento y características del VRP, la envolvente convexa, así como algunos algoritmos para obtenerla; en *Implementación de la Envolvente Convexa a un caso de CVRP* se describe la estructura de los datos usados, el método usado para calcular la envolvente convexa y el modelo matemático propuesto para determinar las rutas de transporte; en la sección *Experimentación* se menciona el comportamiento de los datos y las herramientas utilizadas para logar el objetivo descrito; en *Resultados* se muestran las rutas y las valoraciones obtenidas para cada escenario; y finalmente en *Conclusiones* se redactan los hallazgos obtenidos en este trabajo.

2. Trabajos relacionados

El uso de la envolvente convexa en el tema del VRP no es muy amplio, la mayoría de trabajos la utilizan como una técnica para valorar la calidad de la solución, S. S. Néia et al. [1] proponen una sugerencia para valorar la calidad de las rutas obtenidas en un VRP, mediante un análisis geométrico utilizando una envolvente convexa para la determinación del tamaño de un área propuesta; por otro lado, B. Kallehauge [6] da un revisión de algoritmos exactos aplicados a una variante del VRP con ventanas de tiempo, conocido como VRPTW, en donde se detalla la relación del VRP con el TSP mediante algunos métodos exactos, como la relajación Lagrangiana que utiliza una envolvente convexa de los vectores de incidencia de soluciones. T.K. Ralphs et al. [7] describen los aspectos más relevante del algoritmo *branch and cut*, ya que diseñan subrutinas para separar efectivamente un punto fraccionario dada la envolvente convexa. J. M. Belenguer et al. [8] consideraron una versión de un VRP denominado Problema de Rutas de Vehículos de Entrega Dividida (SDVRP, por sus siglas en inglés), donde definieron una solución factible al problema mediante una envolvente convexa de los vectores de incidencia asociados. Hao Tang et al. [9] propusieron un método heurístico interactivo para resolver un VRP, donde detallan la importancia del aspecto visual de las rutas en aplicaciones prácticas de trasporte de carga, evitando cruces y siendo compactos; la envolvente convexa la utilizaron como una métri-

ca para valorar la calidad de las soluciones determinadas por su método heurístico. Mir Mohammad et al. [10] desarrollaron un método heurístico de búsqueda local, basado en la inserción más cercana de construcción de la envolvente convexa para un TSP, denominado método de inserción más cercano en la búsqueda local de envolvente convexa (NICH-LS). Como se muestra en los trabajos mencionados el uso de metodologías permite determinar los espacios de solución y parámetros iniciales de búsqueda en distintas variaciones de la aplicación del VRP, por ello este artículo muestra una aplicación más de la combinación de la envolvente convexa y el VRP para determinar un conjunto de rutas generadas desde puntos ubicados en el contorno de un área hasta un depósito central, pasando por varios paradas a una distancia mínima.

3. Formulación del problema

La mayoría de implementaciones relacionadas al VRP son resueltas por métodos aproximados [9], [11], [12], lo que lleva a generar soluciones "buenas", sin embargo, no son las soluciones óptimas que puede arrojar un método exacto [6] al resolver modelos matemáticos de VRP. En las siguientes secciones se relatan los elementos asociados al VRP, los modelos de Programación Lineal Entera (ILP, por sus siglas en inglés), así como de la aplicación de una envolvente convexa para determinar rutas óptimas para cuatro destinos distintos, que se interpretan como diferentes escenarios para este trabajo.

Problema de Enrutamiento de Vehículos

Como se ha mencionado, el VRP consiste en la obtención de rutas siguiendo una serie de restricciones de transporte, algunas de las cuales pueden ser genéricas, como el de iniciar desde un punto específico (depósito) y retornar al mismo como en algunos otros casos atender una serie de peticiones de transporte como pasar por ciertos puntos o atender demandas [2]. La representación de un VRP se suele dar mediante gráficas, teniendo un conjunto de vértices (nodos) y de aristas (arcos) que conectan dichos vértices, ver Figura 1. Dada su complejidad computacional existen una gran variedad de métodos tanto exactos como aproximados para resolver un VRP, así como alguna de sus variaciones [2], [12], [13]; sin embargo los métodos exactos conllevan un mayor costo computacional por lo que se suele preferir el uso de heurísticas o metaheurísticas [14] para la determinación de una solución.

Una variación más conocida del VRP es aquella que considera una capacidad C, el cual es utilizado principalmente para el trasporte de carga [7], ya que en cada vértice se tiene un suministro o demanda; este tipo de VRP recibe la denominación de Problema de Enrutamiento de Vehículos Capacitados (CVRP, por sus siglas en inglés) y su objetivo es obtener rutas óptimas sin romper restricciones relacionadas a no rebasar la capacidad de los vehículos [2].

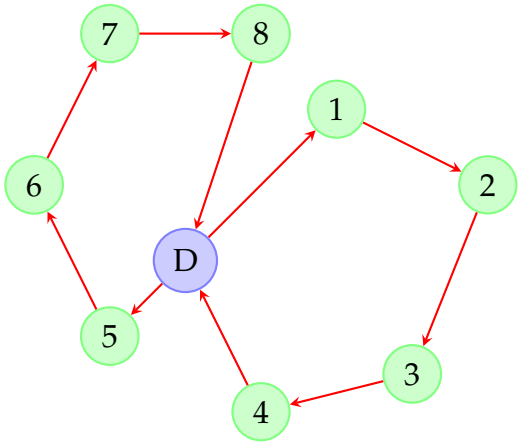


Figura 1: Ejemplo de rutas trazadas en un VRP

Para un VRP o alguna de sus variaciones, la definición de un modelo matemático que determine su comportamiento, tomando en cuenta consideraciones de trasporte, son un punto clave para la obtención de las rutas que dan solución al modelo. Este modelo suele ser formulado mediante Programación Lineal Entera (ILP, por sus siglas en inglés) la cual es una extensión de la programación lineal (LP), en la que se imponen restricciones adicionales a las variables de decisión las cuales deben tomar valores enteros en lugar de valores continuos, por lo que busca resolver problemas de optimización donde las soluciones deben ser enteras [15].

En problemas donde las variables representan cantidades discretas o indivisibles, como en la planificación de la producción, asignación de recursos, diseño de redes, entre otros, la ILP se utiliza para encontrar la mejor solución posible que satisfaga todas las restricciones dadas, mientras se maximiza o minimiza una función objetivo [15]. En la Figura 2, se muestra el modelo de ILP para un CVRP, donde la Ecuación (1) establece una función objetivo que busca el costo mínimo c_{ij} mediante la variable de decisión binaria x_{ij}, que toma el valor de 1 si la arista (i, j) es usada o 0 en caso contrario; las restricciones expresadas en las Ecuaciones (2) y (3) establecen que haya una arista de entrada y una de salida respectivamente para todos los vértices con excepción el vértice $\{0\}$ (identificado como depósito); las restricciones expresadas en las Ecuaciones (4) y (5) determinan que haya K aristas salientes y entrantes al vértice $\{0\}$, es decir, que al depósito lleguen K rutas; las Ecuaciones 6 y 7 se encarga de evitar la formación de subciclos y de establecer las restricciones de la carga acumulada mediante una variable auxiliar u_i y una demanda d_i en cada vértice; el rango de las variables se definen en las Ecuaciones(8) y (9).

Envolvente Convexa

La envolvente convexa (convex hull) de un conjunto de puntos, se define como el polígono *convexo* más pequeño. Un conjunto A es convexo si junto con dos puntos cualesquiera

Función objetivo:

$$\min \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} \tag{1}$$

Sujeto a:

$$\sum_{i \in V} x_{ij} = 1, \quad \forall j \in V \backslash \{0\} \tag{2}$$

$$\sum_{j \in V} x_{ij} = 1, \quad \forall i \in V \backslash \{0\} \tag{3}$$

$$\sum_{i \in V} x_{i0} = K, \tag{4}$$

$$\sum_{j \in V} x_{0j} = K, \tag{5}$$

$$u_i - u_j + C * x_{ij} \leqslant C - d_j \quad \forall i, j \in V \backslash \{0\}, i \neq j, \text{tal que } d_i + d_j \leqslant C \tag{6}$$

$$d_i \leqslant u_i \leqslant C, \quad i \in V \backslash \{0\} \tag{7}$$

$$x_{ij} \in \{0, 1\}, \quad \forall i, j \in V \tag{8}$$

$$u_i \geqslant 0, \quad \forall i \in V \backslash \{0\} \tag{9}$$

Figura 2: Modelo de ILP para un CVRP

p, q este contienen el segmento $s = [p, q]$, que contiene a todos los puntos del conjunto [16]; por lo que la envolvente convexa, en términos geométricos, es como "envolver" un conjunto de puntos en una "membrana elástica" que se ajusta alrededor de ellos, de tal forma que cualquier línea que conecte dos puntos dentro de la envolvente también se encuentra dentro de esta [17].

Existen varios algoritmos comunes para calcular la envolvente convexa, entre ellos:

- Algoritmo de Graham (Graham scan): Ordena los puntos por el ángulo polar respecto a un punto base y, usando un recorrido de lista, identifica los puntos que pertenecen a la envolvente convexa.
- Algoritmo de Jarvis (Gift wrapping): También conocido como "envoltura de regalo", este algoritmo selecciona puntos sucesivos del convex hull hasta cerrar el polígono.
- Algoritmo de Quickhull: Similar al algoritmo de búsqueda QuickSort [18], que implementa la estrategia "divide y conquista", para encontrar los puntos extremos en el espacio.

En la Ecuación (10) se muestra la expresión de la envolvente convexa $Conv$ para un conjunto de puntos X, donde dado k puntos $x_1, x_2, \ldots, x_k$ y teniendo los coeficientes convexos

α_i, su envolvente convexa se define como:

$$Conv(X) = \left\{ \sum_{i=1}^{k} \alpha_i x_i \;\middle|\; x_i \in X, \alpha_i \in \mathbb{R}, \alpha_i \geqslant 0, \sum_{i=1}^{k} \alpha_i = 1 \right\} \tag{10}$$

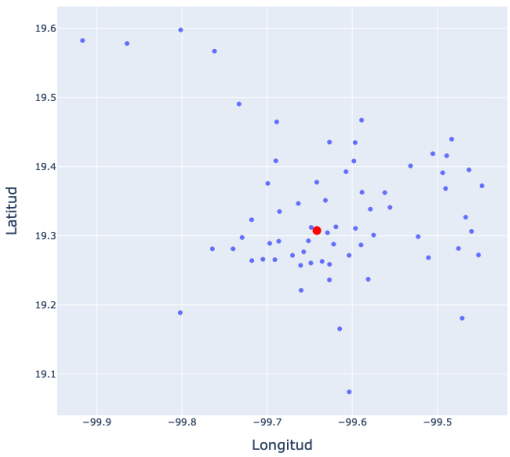
4. Implementación de la Envolvente Convexa a un caso de CVRP

La integración de la envolvente convexa al VRP se realizó mediante la aplicación de dicho método a cuatro conjunto de datos, cada uno con un destino distinto (escenarios). En las siguientes secciones se describe la estructura de los datos, así como el método para obtener la envolvente convexa y su uso para el moldeo de ILP propuesto para los cuatro escenarios.

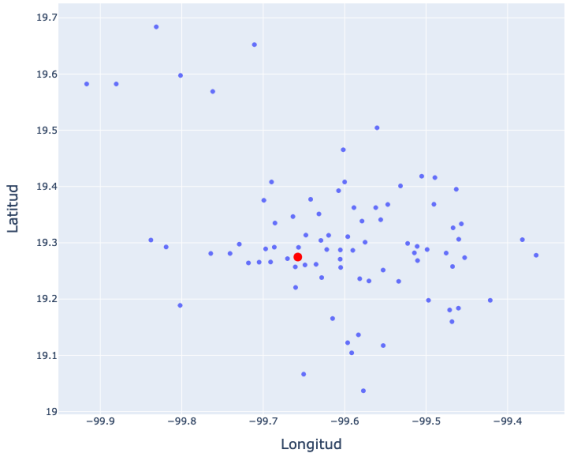
Descripción de los datos

Los datos utilizados se componen de cuatro conjunto de datos separados, en formato CSV (comma-separated values), nombrados: data1.csv, data2.csv, data3.csv y data4.csv. Los datos se construyeron con ubicaciones geográficas de la distribución de población en el Valle de Toluca, mediante uso de Códigos Postales de México [19]. Estos conjuntos de datos mantienen la misma estructura, teniendo los campos: LONGITUD, LATITUD y VOLUMEN, que representan la coordenada en x, y y la cantidad de elementos en cada punto, respectivamente. En cada conjunto de datos se estableció un depósito, siendo este el primer registro del conjunto de datos, en donde las rutas obtenidas deben converger hacia este punto; este elemento se determinó tomándolo como el punto central en cada conjunto de datos.

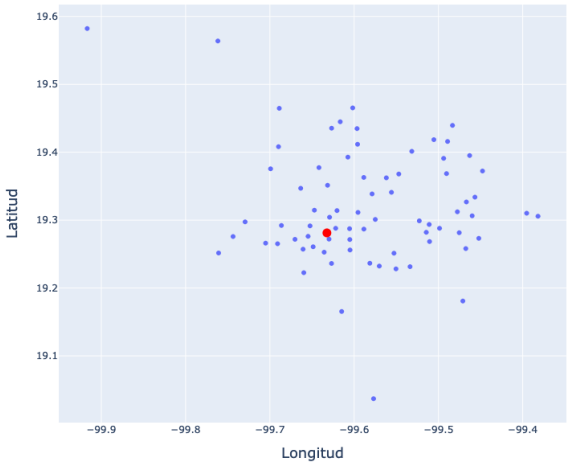
Cada uno de los conjuntos de datos graficados sobre un plano cartesiano, se muestran en la Figura 3, para cada uno la cantidad de puntos es variada; en la Figura 3a se tienen 63 vértices (contando el destino) para el escenario 1, de igual manera en la Figura 3b se tienen 80 vértices para el escenario 2, en la Figura 3c 70 vértices para el escenario 3 y finalmente para escenario 4 en la Figura 3d se tienen 22 vértices.



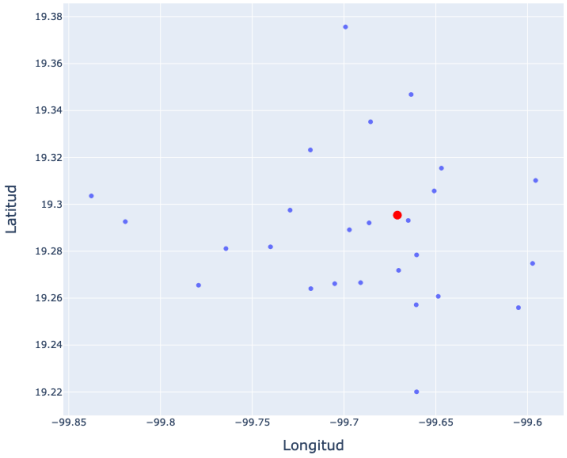
(a) Vértices para el destino 1



(b) Vértices para el destino 2



(c) Vértices para el destino 3



(d) Vértices para el destino 4

Figura 3: Conjunto de vértices para cada de los escenarios (destinos)

Aplicación de la Envolvente Convexa

La aplicación de la envolvente, se realizó mediante el método *QuickHull* [20], el cual es un método para calcular la envolvente convexa para un conjunto finito de puntos, lo cual es ideal para los escenarios propuestos, su utilidad se debe a que emplea técnicas basadas en la estrategia divide y vencerás. La complejidad de esté método para el peor de los casos se establece como $O(n^2)$, pero su complejidad promedio se estipula como $\theta(n*\log(n))$ [20].

Para el caso de dos dimensiones, el método de QuickHull realiza los siguientes pasos :

1. Se busca un par de puntos óptimos, generalmente los puntos con menor y mayor coordenada X, ya que estos siempre forman parte del cierre convexo.

2. Utilizar la línea entre ambos puntos para dividir el conjunto en dos subconjuntos que serán procesados de forma recursiva.

3. Determinar el punto situado a mayor distancia de la línea anterior y junto a los dos puntos anteriores formar un triángulo.
4. Todos los puntos situados en el interior del triángulo pueden ser descartados, ya que no formarán parte de la envolvente convexa.
5. Repetir los dos pasos anteriores en los dos lados del triángulo (no en el lado inicial).
6. Repetir hasta que no queden puntos sin clasificar; los puntos seleccionados forman la envolvente convexa.

Modelo de ILP propuesto

Siendo $G(V, A)$ una gráfica completa, con V como el conjunto de vértices y A siendo conjunto de aristas; del conjunto V se tienen tres tipos de vértices $\{0\}$, S y F de tal manera que $V = \{0\} \cup S \cup F$; donde el vértice $\{0\}$ representa al depósito, S al subconjunto de vértices denominados *paradas* y F siendo el subconjunto de vértices denominados *terminales*.

Se considera que para cada vértice $i \in F$ se inicia una ruta y en el vértice $\{0\}$ inciden la totalidad de las rutas. En cada vértice $i \in V$ hay una demanda b_i de tal manera que en cada ruta la suma de la demandas de cada uno de sus vértices no debe exceder una capacidad C. El modelo matemático de ILP se muestra a continuación:

$$\min \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} \tag{11}$$

$$\sum_{j \in V} x_{ij} = 1, \quad \forall i \in V \backslash \{0\} \tag{12}$$

$$\sum_{j \in V} x_{ji} = 0, \quad \forall i \in F \tag{13}$$

$$\sum_{j \in V \backslash \{0\}} x_{ji} = 1, \quad \forall i \in S \tag{14}$$

$$\sum_{j \in V \backslash \{0\}} x_{0j} = 0 \tag{15}$$

$$\sum_{i \in V \backslash \{0\}} x_{i0} = |F| \tag{16}$$

$$x_{ii} = 0, \quad \forall i \in V \tag{17}$$

$$u_i - u_j + C * x_{ij} \leqslant C - b_j \quad \forall i, j \in V \tag{18}$$

$$b_i \leqslant u_i \leqslant C, \quad \forall i \in V \tag{19}$$

$$x_{i,j} \in \{0, 1\}, \quad \forall i, j \in V \tag{20}$$

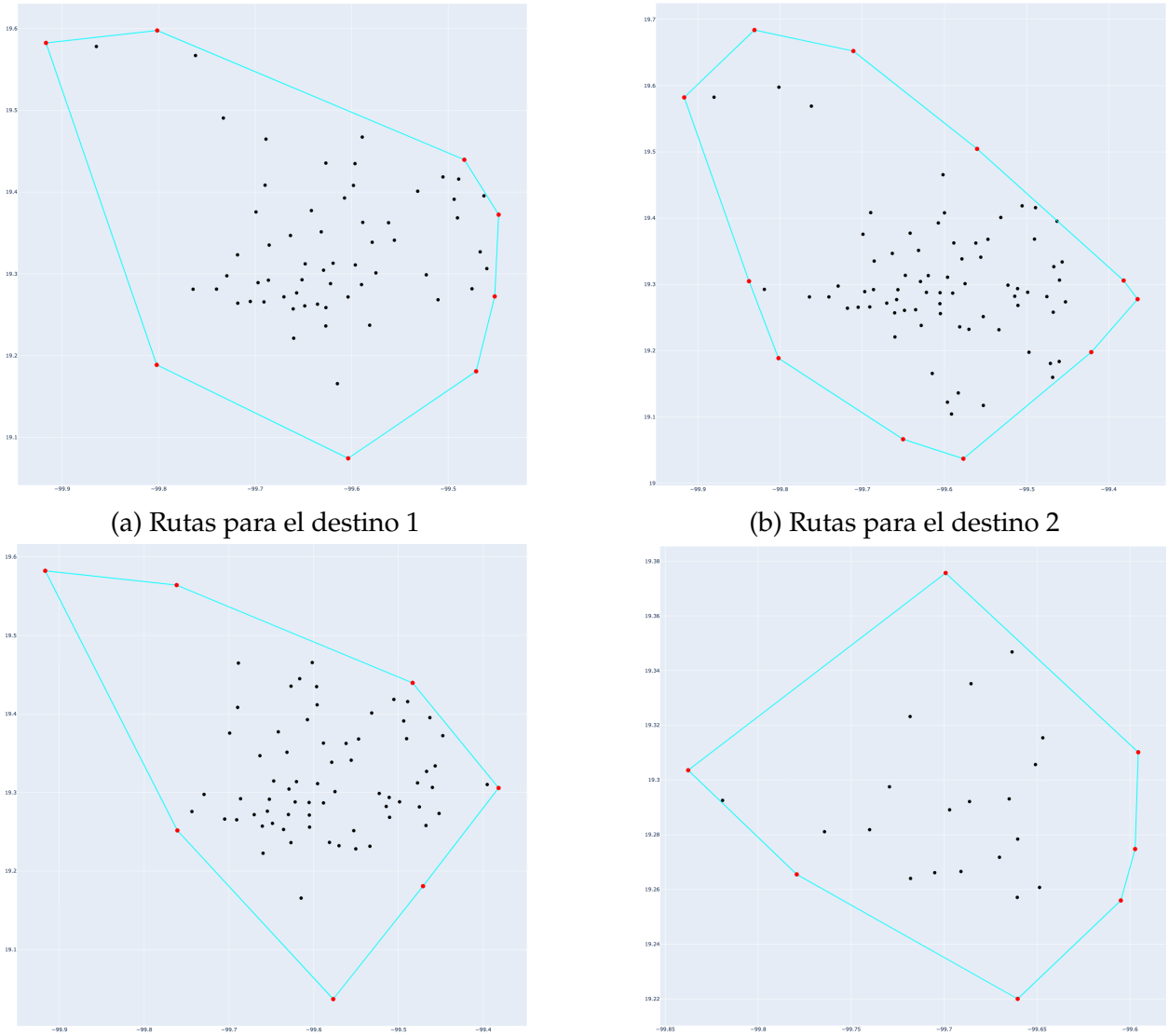
$$u_i \geqslant 0, \quad \forall i \in V, \tag{21}$$

La Ecuación (11) establece la función objetivo que busca las rutas con menor distancia; la Ecuación (12) determina que haya una sola arista que salga de todos los vértices sin considerar al depósito; la Ecuación (13) establece que para todos los vértices terminales, no haya ninguna arista entrante desde otro vértice; la Ecuación (14) señala que haya una sola arista de entrada para todos los vértices de tipo *parada* desde cualquier otro nodo sin considerar el depósito; las Ecuaciones (15) y (16) determinan la cantidad de aristas de salida y entrada para el depósito hacia los otros vértices, se señala que al depósito deben incidir tantas aristas igual el tamaño del conjunto F; la Ecuación (17) evita la generación de ciclos entre vértices iguales; las Ecuaciones (18) y (19) evitan la formación de subciclos y modelan la carga acumulada de pasajeros; la variable de decisión x_{ij} se define en (20), así como la variable auxiliar u_i definida en (21) para modelar la carga acumulada y las demandas b_i en cada vértice i.

5. Experimentación

La implementación de la envolvente convexa con el método QuickHull se realizó mediante la biblioteca *scipy.spatial* del lenguaje Python [21], [22]. Para cada uno de los escenario descritos anteriormente se aplicó dicho método, en las gráficas de la Figura 4 se observan los puntos detectados para formar la envolvente convexa; para el destino 1 se obtuvieron 8 puntos que conforman la envolvente convexa, ver Figura 4a, para el destino 2, en la Figura 4b se determinaron 11 puntos que conforman su envolvente convexa y para los destinos 3 y 4 en las Figuras 4c y 4d se encontraron 7 puntos. La aplicación de la envolvente convexa para cada escenario presentado encontró el subconjunto de vértices *terminales*, estos vértices fueron etiquetados con un valor numérico de 2, el depósito fue etiquetado con un valor de 0 y el resto de vértices se les asignó un valor de 1. Este etiquetado se almacenó en la estructura de datos de los archivos CSV en un nuevo campo denominado *TIPO*, con la finalidad de servir como una categoría para determinar los subconjuntos de datos para el modelo de ILP, denominados *depósito*, *paradas* y *terminales*.

Para el modelo de ILP propuesto, se estableció la capacidad de cada ruta mediante el parámetro C, establecido en 600 para cada escenario; este valor se utiliza en las Ecuaciones (18) y (19), con el fin de establecer un límite de capacidad que no deben rebasar las rutas obtenidas en cada escenario.



(a) Rutas para el destino 1

(b) Rutas para el destino 2

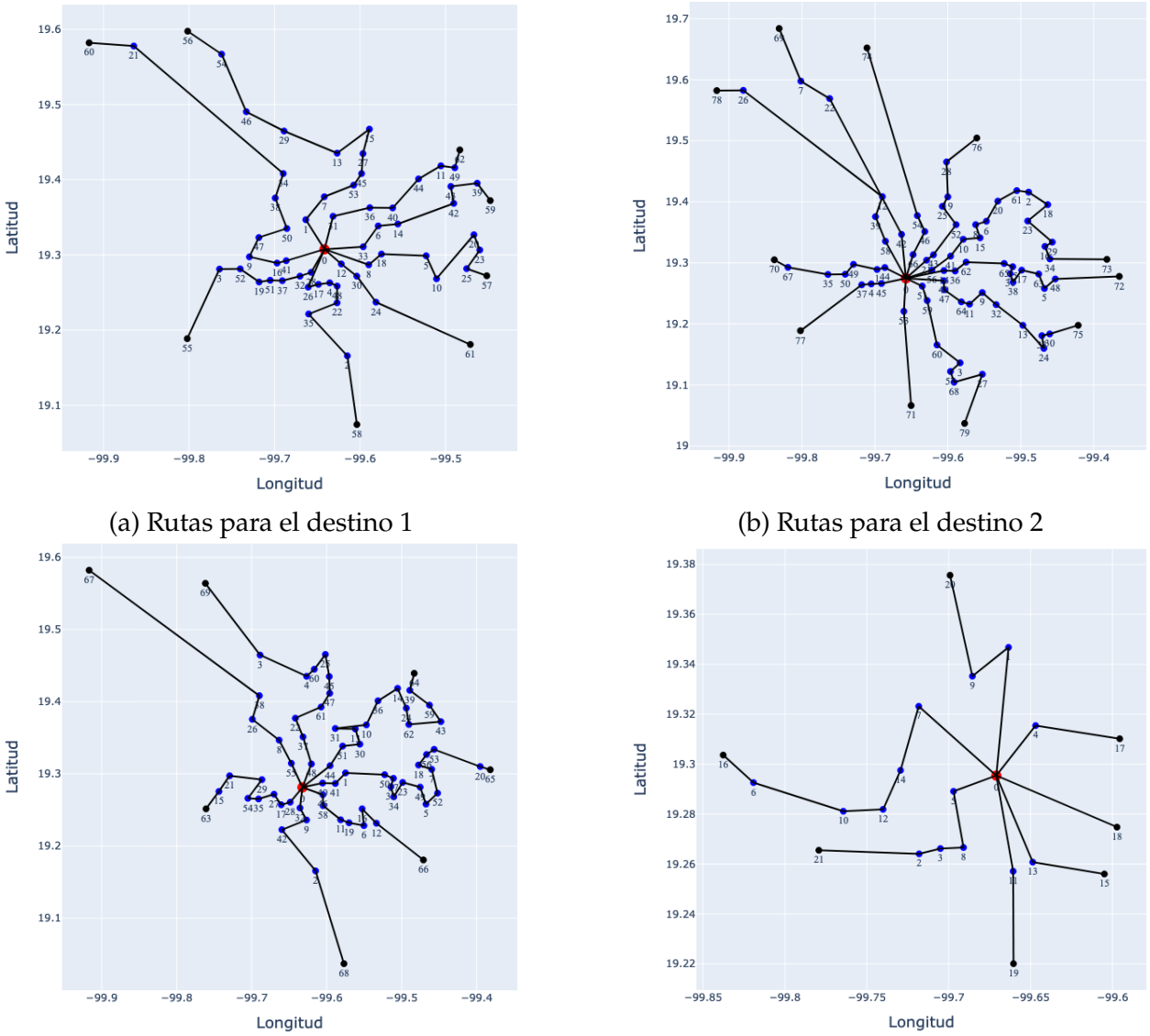
(c) Rutas para el destino 3

(d) Rutas para el destino 4

Figura 4: Rutas obtenidas con el modelo de ILP

6. Resultados

Mediante el método Quickhull para obtener la envolvente convexa y determinar el subconjunto de puntos *terminales* para cada escenario, y posteriormente alimentar el modelo de ILP propuesto; se obtuvieron las rutas mostradas en las gráficas de la Figura 5. En el escenario 1 de la Figura 5a se obtuvieron 8 rutas que inician desde las terminales para dirigirse al depósito; este escenario utilizó un total de 63 vértices. Para el conjunto de puntos del escenario 2, en la Figura 5b se observa que se determinaron 11 rutas utilizando 80 vértices. En la Figuras 5c y 5d para los escenarios 3 y 4 respectivamente, el modelo determinó un total de 7 rutas para ambos casos.



(a) Rutas para el destino 1

(b) Rutas para el destino 2

(c) Rutas para el destino 3

(d) Rutas para el destino 4

Figura 5: Rutas obtenidas con el modelo de ILP

En la Tabla 1 se muestran los valores obtenidos de la función objetivo, el número de terminales establecidas por la envolvente convexa, el total de vértices y el tiempo de ejecución requerido para resolver el modelo para cada uno de los cuatro destinos establecidos.

Tabla 1: Resultados de las métricas para valorar la calidad de los clusters

Destino	Función Objetivo	Num. terminales	Total de vértices	Tiempo de ejecución (seg)
1	2.54	8	63	8.836
2	3.59	11	80	6.704
3	2.46	7	70	7.459
4	0.76	7	22	0.7285

Con los resultados obtenidos podemos señalar que la envolvente convexa para establecer un subconjunto de puntos de inicio o fin de ruta facilita el modelo de ILP, por lo que la solución a este mediante métodos exactos se realiza en tiempos computaciones cortos. Por otro lado observamos que entre más puntos tenga el conjunto de datos, mayor será el tiempo para obtener la solución óptima del modelo

7. Conclusiones

Un método de envolvente convexa aplicado al tema del VRP, permitió determinar el subconjunto de puntos terminales de manera eficiente, por lo que para un modelo de ILP, donde se desea buscar rutas óptimas que inicien desde alguno de los vértices terminales y que incidan a los vértices del subconjunto de puntos intermedios (paradas) para finalizar en el depósito o en caso contrario, inicien desde el depósito incidan por el subconjunto de paras y finalicen en alguno de los elementos del subconjunto de terminales, permite reducir los tiempos computacionales, así como la determinación de las rutas.

Utilizar una envolvente convexa cuando no se ha establecido un conjunto de puntos específicos de inicios de rutas de transporte, permite determinarlos eficientemente; por lo que aplicar este método previo a resolver un modelo mediante métodos exactos, reduce el espacio de búsqueda de soluciones, lo que implica que los tiempos computacionales para obtener una solución óptima se ven minimizados, por lo que esta metodología se recomienda implementar en el área de transporte de carga o de pasajeros, para situaciones donde se busca encontrar rutas en un solo sentido.

Direcciones de correo de los autores

Sánchez-Carmona, M.A.: msanchezc048@alumno.uaemex.mx

Loza-Hernández, L.: llozahe@gmail.com

Referencias

[1] S. S. Néia, A. O. Artero y C. B. D. Cunha, «QUALITY ANALYSIS FOR THE VRP SOLUTIONS USING COMPUTER VISION TECHNIQUES,» en, *Pesquisa Operacional*, vol. 37, n.º 2, págs. 387-402, ago. de 2017, ISSN: 0101-7438. DOI: 10.1590/0101-7438.2017.037.02.0387. visitado 21 de nov. de 2024. dirección: http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0101-74382017000200387&lng=en&tlng=en.

[2] P. Toth y D. Vigo, eds., *The vehicle routing problem* (SIAM monographs on discrete mathematics and applications), en. Philadelphia, Pa: Society for Industrial y Applied Mathematics, 2002, ISBN: 978-0-89871-579-8 978-0-89871-498-2.

[3] R. D. Carr y N. Simonetti, *A New Integer Programming Formulation of the Graphical Traveling Salesman Problem*, 8 de jun. de 2020. visitado 15 de ago. de 2023. dirección: http://arxiv.org/abs/2006.04933.

[4] G. B. Dantzig y J. H. Ramser, «The Truck Dispatching Problem,» en, *Management Science*, vol. 6, n.º 1, págs. 80-91, 1959. dirección: http://www.jstor.org/stable/2627477.

[5] J. A. Bondy y U. S. R. Murty, *Graph Theory* (Graduate Texts in Mathematics), en. London: Springer London, 2008, vol. 244, ISBN: 978-1-84628-969-9 978-1-84628-970-5. DOI: 10.1007/978-1-84628-970-5. visitado 8 de abr. de 2024. dirección: http://link.springer.com/10.1007/978-1-84628-970-5.

[6] B. Kallehauge, «Formulations and exact algorithms for the vehicle routing problem with time windows,» en, *Computers & Operations Research*, vol. 35, n.º 7, págs. 2307-2330, jul. de 2008, ISSN: 03050548. DOI: 10.1016/j.cor.2006.11.006. visitado 21 de nov. de 2024. dirección: https://linkinghub.elsevier.com/retrieve/pii/S0305054806002929.

[7] T. Ralphs, L. Kopman, W. Pulleyblank y L. Trotter, «On the capacitated vehicle routing problem,» en, *Mathematical Programming*, vol. 94, n.º 2-3, págs. 343-359, ene. de 2003, ISSN: 0025-5610, 1436-4646. DOI: 10.1007/s10107-002-0323-0. visitado 21 de nov. de 2024. dirección: http://link.springer.com/10.1007/s10107-002-0323-0.

[8] J. M. Belenguer, M. C. Martinez y E. Mota, «A Lower Bound for the Split Delivery Vehicle Routing Problem,» en, *Operations Research*, vol. 48, n.º 5, págs. 801-810, oct. de 2000, ISSN: 0030-364X, 1526-5463. DOI: 10.1287/opre.48.5.801.12407. visitado 21 de nov. de 2024. dirección: https://pubsonline.informs.org/doi/10.1287/opre.48.5.801.12407.

[9] H. Tang y E. Miller-Hooks, «Interactive Heuristic for Practical Vehicle Routing Problem with Solution Shape Constraints,» en, *Transportation Research Record*, vol. 1964, n.º 1, págs. 9-18, 2006. DOI: 10.1177/0361198106196400102.

[10] M. M. Alipour y S. N. Razavi, «A new local search heuristic based on nearest insertion into the convex hull for solving Euclidean TSP,» en, *International Journal of Operational Research*, vol. 34, págs. 409-429, 2019, ISSN: 1745-7645, 1745-7653. DOI: 10.1504/IJOR.2019.10019739.

[11] B. Muralidharan, S. Eswaran y R. S P, «A survey of vehicle routing problem and its solutions using bio-inspired algorithms,» *International Journal of Pure and Applied Mathematics*, vol. 118, págs. 259-264, 1 de ene. de 2018.

[12] L. Ruisong y W. Ning, «Data-Driven Bus Route Optimization Algorithm Under Sudden Interruption of Public Transport,» *IEEE Access*, vol. 10, págs. 5250-5263, 2022, doi:10.1109/ACCESS.2022.3140947. DOI: 10.1109/ACCESS.2022.3140947.

[13] C. S. W. y K. M. R., «Optimisation of vehicle routing problem with time windows using Harris Hawks optimiser,» *Journal of Mechanical Engineering and Sciences*, vol. 16, n.º 39, págs. 9056-9065, 2022, doi:10.15282/jmes.16.3.2022.08.0717. DOI: 10.15282/jmes.16.3.2022.08.0717.

[14] B. Chopard y M. Tomassini, *An Introduction to Metaheuristics for Optimization* (Natural Computing Series), en. Cham: Springer International Publishing, 2018, ISBN: 978-3-319-93072-5 978-3-319-93073-2. DOI: 10.1007/978-3-319-93073-2. visitado 21 de mar. de 2024. dirección: http://link.springer.com/10.1007/978-3-319-93073-2.

[15] J. Nocedal y S. J. Wright, *Numerical optimization* (Springer series in operation research and financial engineering), Second. New York, NY: Springer, 2006, ISBN: 978-0-387-30303-1 978-1-4939-3711-0.

[16] J. Ohser y F. Mücklich, *Statistical Analysis of Microstrucutres in Materials Science*. John Wilwy & Sibs, Ltd, 2000, Capítulo 3, Estimación de densidad de Kernel, ISBN: 0-471-97486-2.

[17] M. de Berg, O. Cheong, M. van Kreveld y M. Overmars, *Computational Geometry: Algorithms and Applications*, 3rd. Springer, 2008, Capítulo 1, Envolvente convexa, ISBN: 978-3-540-77973-5.

[18] C. A. R. Hoare, «Quicksort,» *The Computer Journal*, vol. 5, n.º 1, págs. 10-16, 1962.

[19] C. P. Mexicano. «Servicio Postal Mexicano.» Consultado el 20 de noviembre de 2023. dirección: https://www.correosdemexico.gob.mx/.

[20] C. B. Barber, D. P. Dobkin y H. Huhdanpaa, «The Quickhull Algorithm for Convex Hulls,» *ACM Transactions on Mathematical Software (TOMS)*, vol. 22, n.º 4, págs. 469-483, 1996. DOI: 10.1145/235815.235821.

[21] SciPy Developers. «scipy.spatial.ConvexHull — SciPy v1.11.3 Manual.» Último acceso: 23 de noviembre de 2024. dirección: https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.ConvexHull.html.

[22] Qhull Project. «Qhull: Quickhull Algorithm for Computing Convex Hulls.» Último acceso: 23 de noviembre de 2024. dirección: http://www.qhull.org.

Informaticae Abstracta
Año 2024, Vol. 2, No. 2, pp. 75-84 JUL | DIC 20244

Inspección óptica sobre uniones de soldadura, un estudio de la aplicación de la física de la reflexión de un modelo RGB

Serna-Hernández, R.; Salas-García, J.*; Vilchis González, A.H.

Información del Artículo

Recibido: 19-nov-2024
Aceptado: 20-dic-2024

Palabras clave: uniones de soldadura, inspección, óptica reflectiva

Resumen

Para mostrar la viabilidad técnica de la inspección óptica automática, se presenta la a física de operación que sustenta su viabilidad práctica como modelo para determinar la calidad de las uniones de soldadura de componentes electrónicos. Se ilustra como la modificación de los niveles de iluminación sobre superficies especulares crean las condiciones propicias para la generación de patrones de reflexión sobre elementos de interés como lo es una unión de soldadura. Se muestra la teoría y los principios de funcionamiento de la iluminación, incidencia y reflexión de ondas y su relación práctica e implementación viable en un sistema de inspección óptica.

1. Introducción

La utilización de circuitos electrónicos se ha convertido en una parte importante del día a día de las actividades del ser humano al estar presentes en los dispositivos necesarios para desarrollar las mismas. Por su practicidad, los circuitos electrónicos en su mayoría hacen uso mayoritariamente de dispositivos con tecnología SMT (*Surface Mount Technology*), la cual es el método más común utilizado para conectar de manera permanente dispositivos SMD (*Surface Mount Devices*) hacia una *PCB (Printed Circuit Board)* [1]; que intrínsecamente lleva a tener con niveles de miniaturización y calidad elevados, elevando la complejidad de estos circuitos electrónicos.

De este hecho, surge otra necesidad para la fase de inspección y aseguramiento de la calidad de dichos circuitos, la cual debe estar a la par del nivel de complejidad de la fabricación. Por lo que realizar el control de calidad de las uniones de soldadura de una placa de circuito impreso PCB mediante una inspección visual, ha quedado totalmente rebasada e inviable para los requerimientos de la industria actual, donde se busca un método de prueba no destructivo NDT (*Non-Destructive Testing*) para detectar defectos de la soldadura aplicada en la patilla de un componente SMT, que dada su heterogeneidad y la naturaleza o razones por los cuales se producen como lo son discontinuidades, impurezas y morfología (como la concavidad o convexidad) presentan un reto poco viable para la inspección manual [2]; porque el proceso de soldadura de componentes electrónicos se lleva a cabo en masa y cualquier defecto asociado afecta directamente a la funcionalidad del producto final [3].

Este hecho lleva necesariamente a requiere un método automático de realizar esta tarea repetitiva y donde los sistemas de inspección óptica automatizada *AOI (Automated Optical Inspection)* surgen como una opción utilizando como física de operación el comportamiento de la luz visible y los fenómenos reflectivos producidos por la interacción con el medio

por el que se propaga. El presente trabajo busca ilustrar como a través del comportamiento geométrico de un haz de luz y de los fenómenos de la reflexión asociados, es posible extraer información de interés que permita generar patrones de reflexión mediante los cuales se pueda determinar la calidad de una unión de soldadura en un componente electrónico tipo SMT.

2. Fundamentos de la Física de la Luz

Tomando como referencia a la luz visible como la onda electromagnética de interés como alternativa de aplicación práctica y principio de operación de un posible sistema de inspección, se define a los colores espectrales como los colores presentes en la luz visible, también conocidos como colores monocromáticos. Estos colores tienen longitudes de onda asociadas (medidas en el vacío). La Tabla 1 muestra los intervalos de longitud de onda emitidos por los colores del espectro visible y el intervalo de frecuencias asociadas a los mismos [4].

Tabla 1: Intervalos de longitud de onda y frecuencia para colores espectrales. Unidades en nanómetros y Tera Hertz (Modificado de Carrillo [3]).

Color	Intervalo de longitud de onda λ	Intervalo de frecuencia
Rojo	$\sim$625 – 740 nm	$\sim$480 – 405 THz
Naranja	$\sim$590 – 625 nm	$\sim$510 – 480 THz
Amarillo	$\sim$565 – 590 nm	$\sim$530 – 510 THz
Verde	$\sim$500 – 565 nm	$\sim$600 – 530 THz
Cyan	$\sim$485 – 500 nm	$\sim$620 – 600 THz
Azul	$\sim$440 – 485 nm	$\sim$680 – 620 THz
Violeta	$\sim$380 – 440 nm	$\sim$790 – 680 THz

El cálculo de los intervalos de longitud de onda y su frecuencia asociada parte del análisis del movimiento de una onda; Serway et al. [5] define el al periodo de éste como el intervalo de tiempo en que una onda pase a través de un ciclo completo de su movimiento; gráficamente los valores de $x,\ y,\ v$ para la partícula/onda en el tiempo t iguala los valores de $x,\ y,\ v$ en el tiempo $t + T$. En radianes, se entiende que la fase aumenta en 2π radianes en un intervalo de tiempo T, en el cual interviene la frecuencia angular, identificada con la constante ω, la cual es la medida en $\frac{rad}{s}$ de qué tan rápido se presenta una oscilación, La ecuación que describe la frecuencia angular es:

$$\omega = \sqrt{\frac{k}{m}} \tag{1}$$

En un haz de luz visible, las variaciones de la misma siguen un comportamiento sinusoidal, por lo cual matemáticamente puede ser expresada a través de la función de onda

$$y(x,t) = A\sin\left[\frac{2\pi}{\lambda}(x - vt)\right]$$

Siendo A es la amplitud de la onda, λ la longitud de onda y v es la velocidad. La Figura 1 muestra la representación gráfica de una onda senoidal desplazándose en sentido positivo.

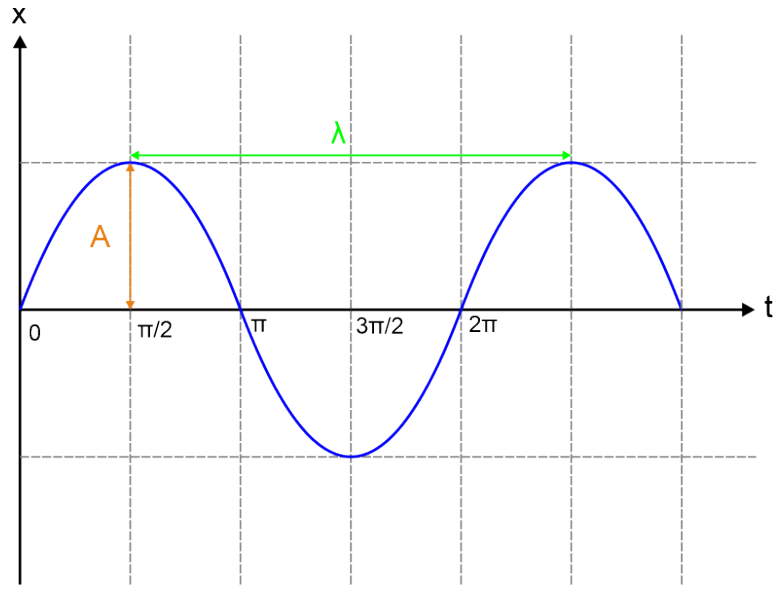


Figura 1: *Representación gráfica de una onda*

La frecuencia angular ω de una onda se puede expresar a través de su periodo T y su frecuencia f

$$\omega \equiv \frac{2\pi}{T} = 2\pi f \tag{2}$$

Matemáticamente, los campos eléctrico y magnético de una onda electromagnética sinusoidal se describen de la forma

$$E = E_{\text{máx}}\cos(\kappa x - \omega t) \tag{3}$$

$$B = B_{\text{máx}}\cos(\kappa x - \omega t) \tag{4}$$

Serway et al [4] indica que dichas ecuaciones representan soluciones especiales a las ecuaciones de onda plana para E y B, y con base en esto, la longitud y frecuencia de las ondas electromagnéticas se relacionan mediante:

$$\lambda = \frac{c}{f} = \frac{3{,}00 \times 10^8 \frac{m}{s}}{f} \tag{5}$$

3. Principios de Reflexión y Óptica Geométrica

Tippens [6] define el tratamiento de ondas electromagnéticas en forma de rayos como óptica geométrica y según el principio de Huygens estos rayos son perpendiculares a los frentes de onda que avanzan en el mismo sentido de propagación del haz. Los cambios de dirección que experimenta un rayo de luz se rigen por las leyes de reflexión. Se conocen dos tipos de reflexiones; La reflexión especular, también conocida como regular, es aquella que procede de una superficie reflectora pulida o lisa donde los rayos reflejados son paralelos entre sí. Si las variaciones de la superficie reflectores son mucho menores que la longitud de onda de la onda incidente, dicha superficie lisa se puede considerar como especular. Para la aplicación práctica del presente artículo, la reflexión difusa no es de interés particular, sin embargo; se entiende como el fenómeno que ocurre en una superficie rugosa, donde los rayos de luz son reflejados en distintas direcciones [5, 6].

Matemáticamente la ley de la reflexión se describe a través de la correspondencia, que indica que el ángulo de incidencia θ_i es igual al ángulo de reflexión θ_i respecto a la normal N.

$$\theta_r = \theta_i \tag{6}$$

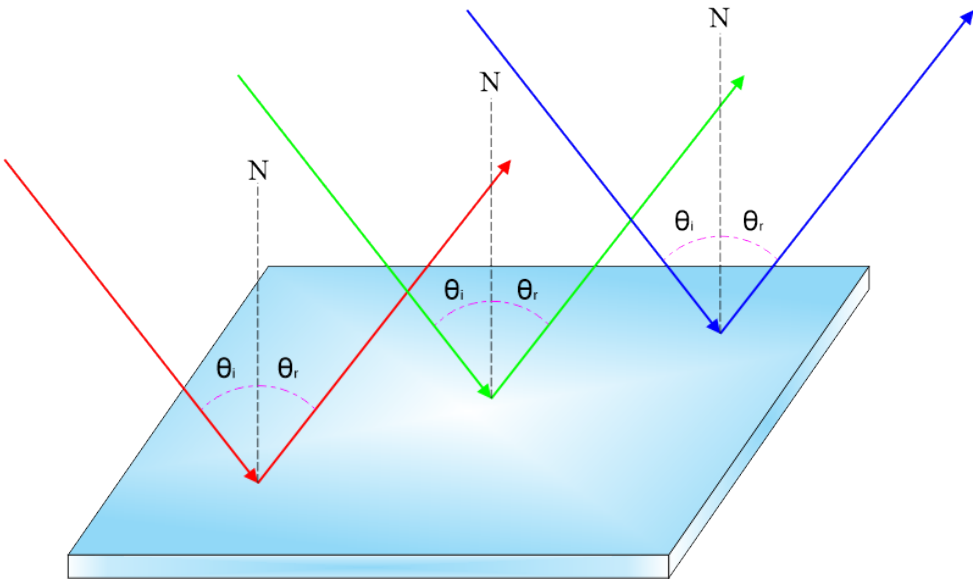


Figura 2: *Representación gráfica de la ley de reflexión.*

4. Aplicación en Superficies Especulares

Conectando la necesidad de contar con un sistema de inspección óptica para una placa de circuito impreso con el comportamiento de una superficie reflectora remite a Hao et al [7] cuyo trabajo trata sobre las superficies especulares y su relación con las patillas de un componente SMT y las uniones de soldadura, las cuales se comportan como un espejo plano, lo cual evidentemente se relaciona con la correspondencia que describe la ley de reflexión.

La caracterización específica de las superficies especulares en uniones de soldadura presenta particularidades notables que merecen un análisis detallado. La aleación de estaño-plomo (Sn-Pb) tradicionalmente utilizada, o las alternativas libres de plomo como SAC305 (Sn-Ag-Cu), exhiben propiedades reflectivas que varían según su composición y proceso de solidificación. La microestructura resultante del proceso de enfriamiento afecta directamente las propiedades ópticas de la superficie, donde los diferentes compuestos intermetálicos (Cu_6Sn_5 y Ag_3Sn) pueden generar variaciones locales en la reflectividad.

El comportamiento especular de estas superficies se ve significativamente influenciado por el proceso de reflujo térmico. La temperatura pico alcanzada durante el proceso y la velocidad de enfriamiento determinan el tamaño y distribución de los granos cristalinos en la superficie de la soldadura. Este fenómeno resulta particularmente relevante ya que la textura superficial resultante afecta directamente la calidad de la reflexión especular. Matemáticamente, esto puede expresarse mediante el coeficiente de reflexión especular (R_s):

$$R_s = \left(\frac{n_1 - n_2}{n_1 + n_2} \right)^2$$

donde n_1 y n_2 son los índices de refracción del aire y la superficie de soldadura respectivamente.

La rugosidad superficial (σ) de la soldadura juega un papel crucial en la calidad de la reflexión especular. Para que una superficie se comporte como un reflector especular efectivo, debe cumplirse la condición de Rayleigh:

$$\sigma < \frac{\lambda}{8 \cos \theta_i}$$

donde λ es la longitud de onda de la luz incidente y θ_i es el ángulo de incidencia. Esta relación explica por qué diferentes longitudes de onda del espectro *RGB* pueden revelar distintos aspectos de la calidad superficial de la soldadura.

En el contexto de la inspección automatizada, la naturaleza especular de las uniones de soldadura permite implementar técnicas de análisis multidireccional. Al variar sistemáticamente el ángulo de iluminación (θ_i), es posible construir un mapa completo de la topografía superficial. La ecuación que relaciona la intensidad de luz reflejada (I_r) con el ángulo de incidencia y las propiedades del material viene dada por:

$$I_r = I_0 R_s(\theta_i) \exp\left(-\frac{4\pi\sigma\cos\theta_i}{\lambda}\right)$$

donde I_0 es la intensidad de luz incidente y $R_s(\theta_i)$ es el coeficiente de reflexión especular en función del ángulo de incidencia.

La caracterización de defectos superficiales se beneficia particularmente de este comportamiento especular. Las discontinuidades típicas como vacíos, grietas o exceso de soldadura modifican localmente el patrón de reflexión. Para una superficie ideal, la distribución angular de la luz reflejada ($D(\theta)$) sigue una distribución gaussiana centrada en el ángulo de reflexión esperado:

$$D(\theta) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(\theta-\theta_r)^2}{2\sigma^2}\right)$$

donde σ representa la desviación estándar de la distribución angular, que está directamente relacionada con la calidad superficial de la soldadura.

Este comportamiento especular predictible permite desarrollar algoritmos de procesamiento de imagen que pueden identificar y clasificar defectos basándose en las desviaciones del patrón de reflexión ideal. La combinación de múltiples ángulos de iluminación y la captura sincronizada de imágenes facilita la reconstrucción tridimensional de la superficie de soldadura, proporcionando información crucial sobre su geometría y calidad.

5. Modelo de Color RGB y Patrones de Reflexión

Partiendo del concepto de colores espectrales, dentro del procesamiento digital de imágenes, las longitudes de onda usadas como base son los colores primarios: rojo, verde y azul, mediante los cuales se pueden generar modelos que permitan analizar la información individual de cada haz de luz que lo compone.

Cuevas et al. [8] considera los modelos RGB como un formato de color aditivo, lo que significa que la combinación de los colores se basa en la adición de los componentes individuales considerando como base el negro. Este proceso puede ser visto como el traslape de tres rayos de luz de colores rojo, verde y azul, los cuales son dirigidos hacia una superficie blanca, y cuya intensidad puede ser continuamente controlada; y donde la intensidad de los componentes de color determina tanto el tono como la iluminación de color resultante. Este principio es de vital interés para la generación de patrones de iluminación y reflexión, de los cuales es posible extraer información relevante según el comportamiento de una específica longitud de onda, en este caso, de las tres que componen el modelo RGB.

La Figura 3 muestra el modelo 2D de un patrón de reflexión sobre las uniones de soldadura de un componente SMT utilizando un modelo RGB para la fuente de iluminación.

Hasta ahora se ha explorado como la óptica geométrica se comporta en una unión de soldadura partiendo de un modelo de color especifico de los colores espectrales; el paso

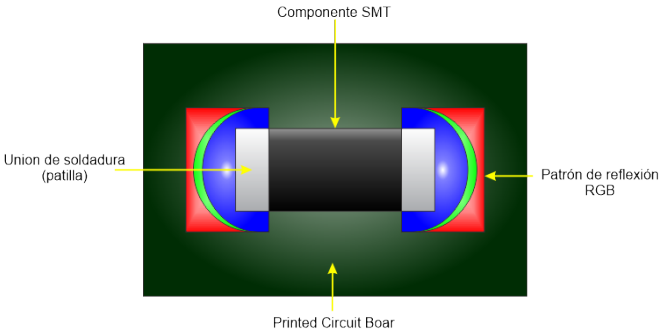


Figura 3: *Patrón de reflexión RGB sobre la unión de soldadura de un componente SMT*

natural a seguir dentro la búsqueda de la implementación práctica de estos principios físicos es el cómo es posible capturar la información producto de la reflexión de los haces de luz bajo análisis para su posterior procesamiento.

6. Implementación Práctica y Sensores

El sensor de una cámara digital es el elemento que determina la cantidad de luz que el dispositivo va a utilizar para crear una imagen. Los sensores están compuestos de receptores de luz conocidos como fotositos (en inglés, *photosite*) los cuales hacen la función de un transductor al convertir energía óptica en energía eléctrica. Esta señal eléctrica es proporcional a la intensidad de la luz captada por el fotosito.

La relación de los modelos de color RGB con el funcionamiento de los fotositos se da debido a que estos últimos contiene filtros de color basados en el modelo aditivo y la transducción de la energía óptica a eléctrica se relaciona directamente con la intensidad de la luz detectada por el sensor de cada una de estas longitudes de onda.

La Figura 4, ilustra gráficamente la implementación práctica de la ley de reflexión en la soldadura de un dispositivo electrónico SMT. Los rayos de luz incidentes en la patilla del componente (considerada como una superficie especular,) tendrán ángulos de reflexión seguirán la correspondencia previamente descrita, siendo iguales al ángulo de incidencia; los cuales serán reflejados hacia el sensor de una cámara digital (y por ende de los fotositos que éste contiene), representa un elemento idóneo para la detección de los haces de luz RGB para su posterior procesamiento.

La habilidad de una cámara para la generación de una imagen está determinada por el tipo y el tamaño del sensor con el que cuenta. La Tabla 2 muestra los diversos tipos de sensores disponibles a nivel comercial junto con los tamaños y áreas asociados a los mismos.

R. Serna [9] presentó un sistema de control óptico automático cuyo principio de operación es la capacidad de obtener información a través del ángulo de reflexión de los colores

Tabla 2: Tipos de sensores de cámaras y tamaños asociados disponibles comercialmente.

Tipo/Nombre	Tamaño del sensor	Área del Sensor
Formato Medio	53.7 × 40.2 *mm*	21.59 cm^2
Full Frame	36 × 23.9 *mm*	8.6 cm^2
APS-H	27.9 × 18.6 *mm*	5.19 cm^2
APS-C	23.6 × 15.8 *mm*	3.73 cm^2
4/3	17.3 × 13 *mm*	2.25 cm^2
1″	13.2 × 8.8 *mm*	1.16 cm^2
1/1.63″	8.38 × 5.59 *mm*	0.47 cm^2
1/2.3″	6.16 × 4.62 *mm*	0.28 cm^2
1/3.2″	4.54 × 3.42 *mm*	0.15 cm^2

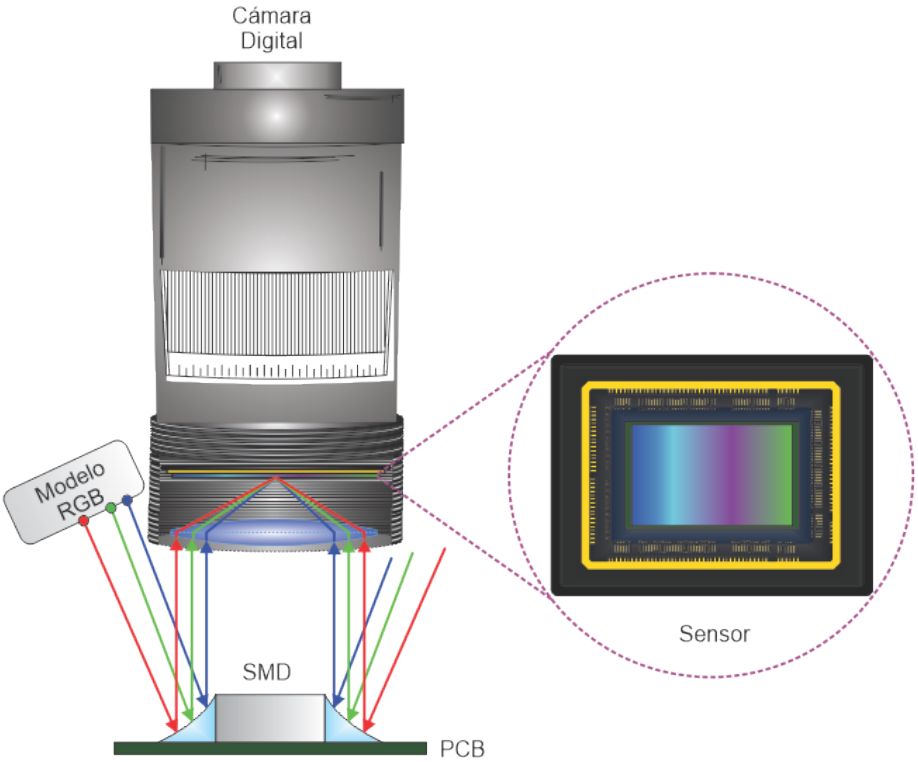


Figura 4: *Reflexión de haces de luz RGB desde la patilla/soldadura de un componente SMT hacia el sensor de una cámara digital.*

espectrales en uniones de soldadura de *PCB*. Particularmente, usando un modelo de color RGB *(Red Green Blue)*, el cual, mediante el procesamiento de imágenes y el posterior análisis de patrones de iluminación, reflexión y detección de bordes, permite a este desarrollo determinar la calidad de las uniones de soldadura de los componentes con la tecnología SMT.

7. Conclusiones

En este trabajo se ha presentado un análisis detallado de los fundamentos físicos que sustentan la viabilidad de sistemas de inspección óptica automática para la evaluación de uniones de soldadura en componentes electrónicos. Las principales conclusiones que se derivan de este estudio son:

La aplicación de los principios de la óptica geométrica, particularmente la ley de reflexión en superficies especulares, proporciona una base teórica sólida para el desarrollo de sistemas *AOI* (*Automated Optical Inspection*). Se ha demostrado que la correspondencia entre los ángulos de incidencia y reflexión ($\theta_i = \theta_r$) en las superficies de soldadura permite generar patrones de reflexión predecibles y analizables.

El uso del modelo de color *RGB* como base para la iluminación presenta ventajas significativas para la inspección de soldaduras. La capacidad de controlar independientemente las intensidades de los tres colores primarios (rojo, verde y azul) permite generar patrones de reflexión específicos que maximizan la detección de defectos en las uniones de soldadura. Esta característica es particularmente relevante dado que las longitudes de onda específicas de cada color pueden interactuar de manera diferente con las imperfecciones de la superficie.

La implementación práctica de estos principios físicos se ve favorecida por la tecnología actual de sensores en cámaras digitales. Los fotositos, al actuar como transductores de energía óptica a eléctrica y contar con filtros *RGB* integrados, proporcionan una plataforma ideal para la captura y análisis de los patrones de reflexión generados. Esta capacidad tecnológica permite una detección precisa de las variaciones en la reflexión que pueden indicar defectos en la soldadura.

El sistema propuesto representa una solución viable para los desafíos actuales en la industria electrónica, donde la inspección manual ha quedado obsoleta debido a la creciente miniaturización y complejidad de los componentes *SMT*. La capacidad de realizar inspecciones no destructivas (*NDT*) de manera automatizada responde directamente a las necesidades de control de calidad en la producción masiva de circuitos electrónicos.

La fundamentación matemática presentada, desde las ecuaciones de onda electromagnética hasta los principios de reflexión, proporciona un marco teórico robusto que permite predecir y modelar el comportamiento de la luz en las superficies de soldadura. Este entendimiento profundo facilita la optimización de los sistemas de inspección y la interpretación de los resultados obtenidos.

El trabajo establece las bases para futuras investigaciones en el campo de la inspección óptica automatizada, particularmente en el desarrollo de algoritmos de procesamiento de imágenes que aprovechen la información contenida en los patrones de reflexión *RGB* para la identificación automática de defectos en las uniones de soldadura.

Estas conclusiones demuestran que la aplicación de principios físicos fundamentales, combinada con tecnologías modernas de captura y procesamiento de imágenes, ofrece una solución práctica y efectiva para los retos actuales en la inspección de calidad de componentes electrónicos. El enfoque presentado no solo es teóricamente sólido, sino que también es técnicamente viable y alineado con las necesidades de la industria moderna.

Direcciones de correo de los autores

Serna-Hernández, R.: raulserna10@outlook.com

Salas-García, J.*: jsalasg@uaemex.mx

Vilchis González, A.H.: avilchisg@uaemex.mx

Referencias

[1] Tsai, T. (2012). *Thermal parameters optimization of a reflow soldering profile in printed circuit board assembly: A comparative study*. Applied Soft Computing, 12, 2601-2613.

[2] Zapata, J., Vilar, R., & Ruiz, R. (2010). *An adaptive-network-based fuzzy inference system for classification of welding defects*. NDT&E International, 43, 191–199.

[3] Carillo, D. (2007). *Procesamiento de Señales*. INAOE.

[4] Liukkonen, M., Havia, E., & Hiltunen, Y. (2012). *Computational intelligence in mass soldering of electronics – A survey*. Expert Systems with Applications, 39, 9928–9937.

[5] Serway, R., & Jewett, J.W. (2009). *Física para ciencias e ingeniería*. Volumen 1, 2. Cengage Learning. pp. 421, 959, 969, 982.

[6] Tippens, P. (2001). *Física, Conceptos y aplicaciones*. 7a ed. McGraw-Hill. pp. 662, 663, 664.

[7] Hao, W., Xianmin, Z., Yongcong, K., Gaofei, O., & Hongwei, X. (2010). *Solder joint inspection based on neural network combined with genetic algorithm*. Optik - International Journal for Light and Electron Optics, 124, 4110-4116.

[8] Cuevas, E., Zaldivar, D., & Pérez, M. *Procesamiento digital de imágenes usando MatLAB y Simulink*. México: Alfaomega.

[9] Serna, R. (2020). *Sistema de control por procesamiento de imágenes de una máquina CNC para soldar circuitos integrados*.

Informaticae Abstracta
Año 2024, Vol. 2, No. 2, pp. 85-94 JUL | DIC 20244

Accionamiento de un tubo disparador para crear ondas de choque

Vallejo-Palma, Braulio; González-Vallejo, Jesús Alberto; Mendioza-Clemente, Raziel Noel; Gutíerrez-García Karla; Dávila-Vilchis, Juana-Mariel*

Información del Artículo

Recibido: 10-sep-2024
Aceptado: 18-dic-2024

Palabras clave: Ondas de choque, No. Mach, Tubo disparador, Fluidos compresibles

Resumen

El estudio de las ondas de choque es fundamental en el análisis de fenómenos físicos, particularmente en el diseño aerodinámico de cuerpos sólidos sometidos a velocidades superiores a las del sonido. Es por lo que este artículo presenta el diseño y fabricación de un tubo disparador capaz de crear ondas de choque que permite entender su comportamiento en flujos supersónicos. La metodología se dividió en 4 etapas principales: 1) Análisis matemático para generar la onda de choque, 2) Modelado en CAD (Computer Aided Design) del sistema de disparo, 3) Fabricación y 4) Pruebas del disposiitvo para validar su funcionamiento. En la fase del análisis matemático se calcularon las dimensiones del disparador a fin de garantizar la generación de la onda de choque. En el Modelado se obtuvo el diseño computacional 3D del disparador. En la fabricación se seleccionaron los materiales para construir el disparador y se hizo el ensamble de todos los componentes. Finalmente, se realizaron las pruebas para generar las ondas de choque mediante la liberación controlada de aire con un compresor estacionario y utilizando dos tipos de membranas: plástico y TPU (thermopoliuretano). Los resultados obtenidos con ambos materials demostrarón que el disparador logró alcanzar flujos supersónicos para generar de las ondas de choque rectas.

1. Introducción

Durante siglos se han desarrollado diferentes armas de fuego con fines bélicos o de defensa, lo cuales son accionados por la presión generada de un propelente a alta velocidad que tienen como objetivo disparar uno o múltiples proyectiles [1]. Existe una variedad de estos dispositivos dependiendo el grado de daño deseado, alcance, o tamaño desde pistólas, rifles, fúsiles, ametralladoras o escopetas [2].

Otro tipo son los dispositivos explosivos improvisados (IED) por sus siglas en inglés, estos artefactos son construidos manualmente y son accionados de diferentes maneras pero siguen el principio de funcionamiento de las armas de fuego [3]. Por lo que, para reducir los daños causados por los ya mencionados dispositivos, se han desarrollado mecanismos de protección o herramientas que prueba como tubos de choque. Este equipo de laboratorio simula las ondas expansivas de las explosiones sin utilizar explosivos reales, lo que maximiza la seguridad durante su uso [4]. De igual manera, durante la explosión se libera rapidamente energía que genera alta presión que viaja radialmente alrededor del tubo formando una capa delgada de aire con un espesor (m), llamada onda de choque [5].

El número de Mach, Ma es fundamental para el estudio de las ondas de choque en análisis aerodinámicos ya que ocurren en flujos compresibles que tienen valores de Ma mayores a 0.3 [6], como se describe a continuación.

- Subsónico 0.3 < Ma < 0.8: los efectos de la densidad son despreciables y no presentan ondas de choque.
- Transónico 0.8 < Ma < 1.2: región de frontera donde hay posibilidad de ondas de choque.
- Supersónico 1.2 < Ma < 3.0: se presentan principalmente las ondas de choque y no hay región subsónica.
- Hipersónico Ma > 3.0: las ondas de choque y los cambios de flujo son significativos.

Las ondas de choque se forman cuando un objeto se acelera de una velocidad subsónica a una supersónica. Es decir, la velocidad del sonido de un fluido a, generalmente aire, es mayor que la velocidad de un objeto, V, relacionadas por Ma, como se muestra en la ecuación (1)[6].

$$Ma = \frac{V}{a} \tag{1}$$

Las ondas de choque se clasifican en dos tipos normales y oblicuas. Las normales se forma perpendicularmente al flujo del aire, mientras que las oblicuas se caracterizan por formar a un ángulo con respecto al fluido [8]. Una vez a la salida del objeto, las ondas presentan pérdidas de energía, sin embargo, en la onda de choque normal es mayor debido a que siempre se desacelera a una velocidad subsónica. Dentro de una sección cónica (difusor-tobera), a medida que la velocidad del objeto incrementa a velocidades supersónicas, la onda se va desplazando y forma una nueva onda de choque sucesivamente, lo que significa que la fuerza de arrastre aumente sobre el cuerpo sólido, por lo que se necesita mayor fuerza empuje para superarla [7].

Las ondas de choque se empezaron a estudiar durante la segunda guerra mundial, observando que las bombas dañaban unicamente de forma interna el objetivo [11]. Más adelante, en los años 50's se lograron generar ondas de choque por medio de fuentes electromagnéticas, lo que abrió diversas puertas para poder ser empleadas en técnicas médicas hasta el día de hoy, como la regeneración del tejido muscoloequlético, inflamación o fracturas mediante explosiones controladas que generan pulsos sónicos [9]. También, las ondas de choque se utilzan durante el diseño de aeronaves [10] o equipo balístico [12].

Por ejemplo, para el armamento corto o largo, están integrados por un cilindro metálico largo llamada cañón, que permite la propulsión del proyectil, entre más tiempo pasa dentro de éste, mayor es su aceleración radial y velocidad a la salida para alcanzar mayores distancias e incrementar la explosión [13]. Los tubos de choque funcionan bajo el mismo principio del cañón, está divido en tres zonas: 1) recámara donde se almacena el cartucho,

2) cuerpo del tubo y 3) chooke que controla la distancia del proyectil como se muestra en la Figura [1]. Normalmente, durante las pruebas de los tubos de choque se utilizan cámaras de alta velocidad para observar la trayectoria del proyectil [14] o simulaciones de fluidos computaciones (CFD) por sus siglás en inglés [15], para validar su funcionameinto.

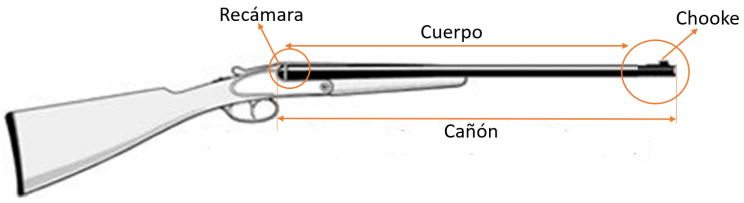


Figura 1: Partes del cañón de una escopeta.

Este artículo se centra en el diseño de un tubo de choque como (IED) que replica el funcionamiento de un arma cuando es accionada. Se incluye el modelado 3D y el cálculo del número de Mach dentro del tubo para generar las ondas de choque con base en sus dimensiones. Además, de la fabricación y ensamble de los componentes. Finalmente, se reportan los resultados obtenidos durante el accionameinto del tubo disparador y resaltan las posibles mejoras del modelo físico experimental como una aplicación didáctica de la Mecánica de Fluidos.

2. Metodología

El funcionamiento de un tubo disparador consiste en generar una diferencia de presiones, separadas por un diafragma de ruptura para poder generar ondas de choque, una vez que la capa se rompe, la onda se propaga en radialmente [16]. Para lograr este objetivo, el tubo es seccionado en tres zonas: 1) *Zona del disparador* se encarga de almacenar el aire a una presión definida. 2) *Zona de transición* está separada por una membrana delgada de tela de la zona del disparador, que cuando se rompe genera una onda de choque, debido a una diferencia de presiones P, velocidades V, temperatuas T, respectivamente y número de Ma a la entrada mayor que a su salida. 3) *Zona de pruebas* es la región donde la onda impacta al objetivo [4]. Algunos diseños incluyen un tanque de disipación que sirve para absorber la energía a la salida y disminuir la presión y velocidad del fluido [17].

Para este estudio se baso en el método de ruptura de membrana que se caracteriza por tener una capa delgada que impide el paso del aire para formar la onda de choque.

No. de *Ma* para generar la onda de choque

El número de Ma, así como su presión a la entrada y a la salida de la membrana, deben ser calculados para garantizar que el tubo disparador generará una onda de choque. Por lo tanto, para empezar con el diseño del disparador se propuso tubo de policloruro de

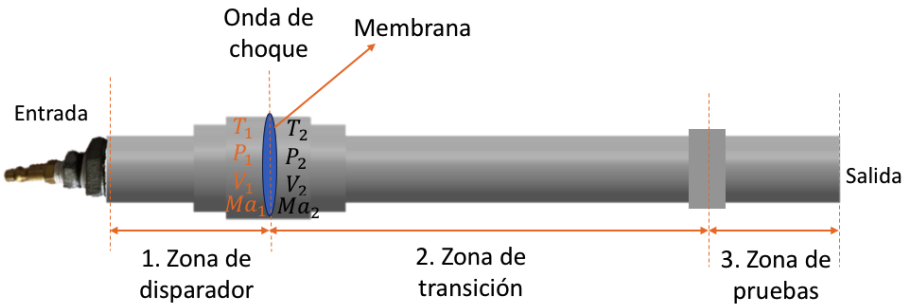


Figura 2: Zonas del tubo disparador.

Vinilo (PVC) con una longitud L (100 cm) y un diámetro, D de 1 in (2.54 cm). Autoren en [4] recomiendan que la zona de transición deba ser al menos 2 veces la longitud de la zona de disparo, para este caso se tiene una relación L/D = 39.37. Para la membrana o diafragma se utilizó 200D TPU-nylon (thermopoliuretano) de 2mm de espesor.

Despúes de seccionar cada región, se relacionaron las áreas donde ocurre la onda de choque a la entrada An_1 con la de acoplamiento o garganta A*, mediante la expresión:

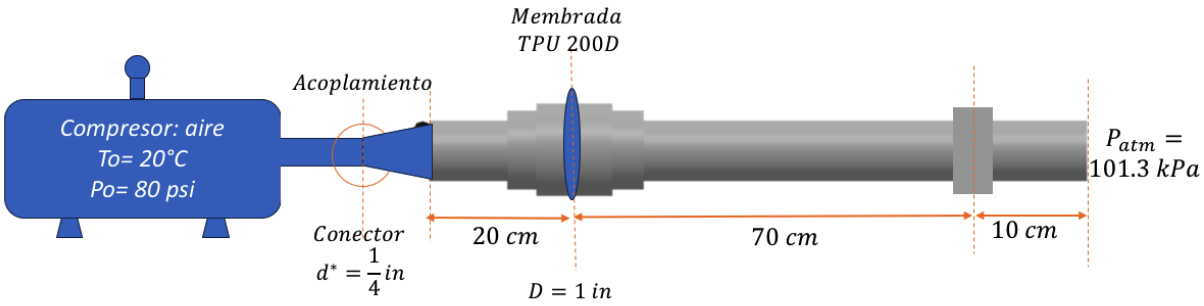
$$\frac{An_1}{A^*} \tag{2}$$

Donde a partir de las tablas para flujos compresibles se calculó Ma_{n1}, con un flujo supersónico de 2.94 para garantizar la onda de choque. De igual manera, con este valor se relacionó la presión del depósito P_o de 80 psi para encontrar la presión antes de la onda de choque P_{n1}. Una vez determinado Ma_{n1}, se utilizarn las tablas de ondas de choque [18] para encontrar el Ma_{n2} a la salida de la membrana, donde se obtuvo un valor de 0.478, el cual demuestra la desaceleración a flujo subsónico. Finalmente, se relacionó Ma_{n2} con la presión atmósferica P_{atm} a la salida del tubo, para encontrar la presión despúes de la onda de choque. Dichos valores, se muestran la Figura 3 del modelo 2D del sistema con todos sus componentes y dimensiones, donde el tubo es alimentado por un compresor mediante un conector de $1/4$ in.

En la zona de acoplamiento se montó una válvula reguradora y un conector para restringir el flujo del aire en un solo sentido y controlar el caudal para evitar golpes de ariete.

Fabricación del tubo disparador

El tubo de choque consta de un tubo de PVC como componente principal, el cual está integrado de otros componentes dependiendo de cada sección para crear la onda de choque. Durante dicho proceso, no se requirió de maquinaría sofisticada, únicamente herrameinta de presión para apretar los coples con el tubo. En la zona de disparo se utilizó un sistema de acoplamiento para conectar el tubo y el compresor para el suministro de aire. Y en la zona de transición se montaron unos coples para contener y sujetar a la me-



Depósito	Garganta	Entrada	Salida
$R = 287kJ/kgK$	$Ma^* = 1$	$Ma_{n1} = \mathbf{2.94}$	$Ma_{n2} = \mathbf{0.478}$
$T_o = 293.15\ K$	$D^* = 0.635\ cm$	$f.supersónico$	$f.subsónico$
$P_o = 551.5kPa$	$A^* = 3.16E\text{-}5\ m^2$	$D_{n1} = 2.54\ cm$	$D_{n1} = 2.54\ cm$
		$A_{n1} = 0.0005m^2$	$A_{n2} = 0.0005m^2$
		$P_{n1} = 16.38\ kPa$	$P_{n2} = 162\ kPa$

Figura 3: Vista lateral del tubo disparador con cada región acotada.

brana. A continuación se enlistan y describen cada uno de los pasos realizados y se ilustra su ensamble final en la Figura 4.

1. Macho de 1/4″: permite la conexión de la boquilla del compresor a la alimentación del tubo.
2. Enroscada 1/4″: permite la conexión del macho al reductor.
3. Reductor de 1/2.ª 1/4″: para conectar la enroscada y un conector macho.
4. Macho cople de 1.ª 1/2″: permite la conexión entre todos los elementos mencionados con el tubo PVC de 1″.
5. Válvula de globo de PVC: ayuda a regular el flujo de aire que se suministra a la zona de disparo.
6. Válvula de Check: se utilizó para direccionar el flujo en un solo sentido y evitar contraflujos.
7. Cople de 1″: sirve para unir las zonas de disparo y la de transición y contener a las membranas.
8. Conector macho 1″: se coloca nuevamente al final del tubo para unirse a la válvula de expulsión de aire.
9. Valvula de expulsión de aire: para evitar su retención de aire, o golpes de airete, producido por un aumento repentino de presión por el cambio en la velocidad del caudal, su funcionameinto es similar al escape de un automóvil.

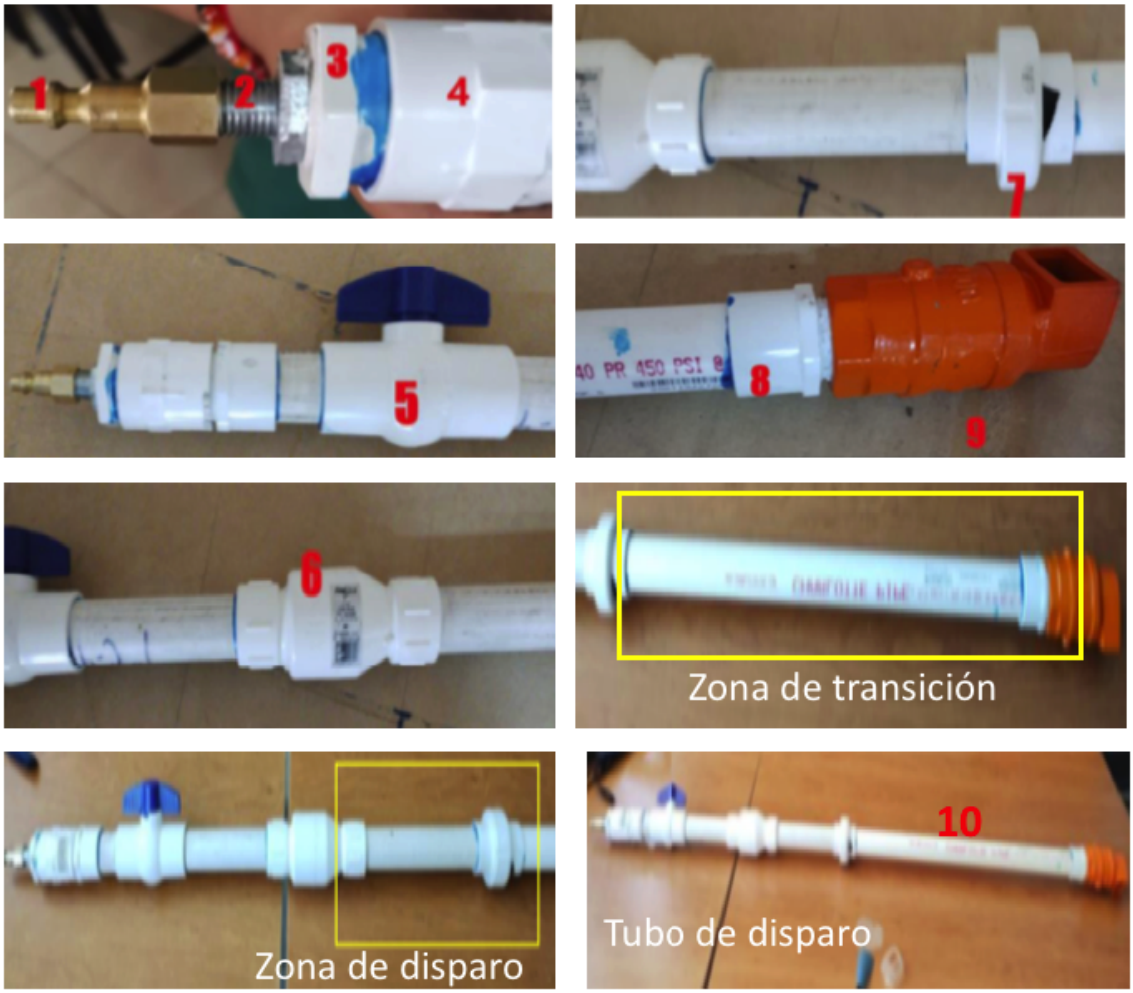


Figura 4: Ensamble del tubo disparador y sus componentes.

10. Tubo de disparador de PVC cédula 40 de 1”.

Además, se utilizó conta de Teflón, pegamento PVC para la unión de sus componentes, así como empaques de goma con el propósito de reducir posibles pérdidas de presión entre ambas partes. El diseño del cople fue propuesto para lograr un montaje y desmontaje rápido de la membrana, facilitando las pruebas repetitivas sin comprometer la funcionalidad del sistema.

3. Resultados

Para evaluar el funcionamiento del tubo se empleó un compresor comercial para alimentar la sección de disparo, de modo que al ingresar se creará la diferencia de presión para generar ondas de choque capaces de impulsar el objeto. En este estudio se realizaron pruebas con 2 tipos de membranas: 1) plástico (tear offs) y 2) TPU 200D con un espesor de

200 µm. Para ambos casos se utilizó un área de 4 cm^2, los tubos fueron accionados a partir de 80 psi ≈550 kPa donde el TPU alcanzó hasta las 110 psi sin romperse, mientras que la membrana de tear off se rompió desde los 80 psi, como se observa en la Figura 5 antes y después del disparo.

Figura 5: Membranas del tubo antes y después del disparo.

4. Discusión

Durante las pruebas de funcionamiento del tubo disparador se mostró que con la membrana de plástico se generó la onda de choque desde los 80 psi y se rompió completamente. Mientras, que la onda de choque con la membrana de TPU alcanzó hasta 110 psi, mostrando un desgaste progresivo tras cada prueba pero sin romperse, lo que significa que la fatiga del material afecta su desempeño en usos repetidos.

El diseño del cople permitió un montaje y desmontaje rápido de la membrana, facilitando las pruebas repetitivas sin comprometer la funcionalidad del sistema. A diferencia de otras propuestas, el uso de la valvúla permitió controlar del flujo del aire a través del tubo y restringiendólo en un solo sentido.

Durante las pruebas no se registraron pérdidas significativas de presión, gracias al uso de cinta de teflón en las uniones roscadas y al ajuste con pinzas. De igual menera, las dimensiones propuestas para el tubo, el uso del cople rápido de compresión, así como el par de empaques de goma ayudaron a preservar la presión obtenida en la zona de disparo. Permitiendo que la onda de choque fuera mayor y se minimizaran las pérdidas de presión entre ambas partes. Sin embargo, se detectarón pérdidas de presión en la zona de transición, lo que podría haber afectado la fuerza final del disparo.

El tubo disparador logró crear las ondas de choque y emular el funcionamiento de un arma de fuego. Visualmente, se observó un cambio brusco en la velocidad del aire

comprimido al cruzar la membrana, lo que sugiere que las ondas de choque se generaron conforme a lo esperado. Sin embargo, no se disponían de sensores para medir la velocidad exacta del aire, lo que limita la validación experimental de los cálculos de Mach. Por lo que se sugiere la implementación de instrumentos, sensores de presión, velocímetros de aire y cámaras de alta velocidad para registrar parámetros como la distancia alcanzada, la velocidad del aire comprimido durante el disparo, así como la salida del objeto, y el comportamiento de la onda de choque a lo largo de su trayectoria. Lo que podría proporcionar datos más detallados para evaluar el rendimiento y mejorar su diseño.

Las pruebas se realizaron a 80 y 110 psi, lo que restringió la evaluación de cómo diferentes niveles de presión afectan la velocidad de salida y la eficacia de las ondas de choque. Sin embargo, sería ideal realizar un rango de pruebas a presiones variables para estudiar estas relaciones. Por otro lado, la membrana funcionó como elemento de ruptura para la generación de ondas de choque, sin embargo, es importante evaluar su desgaste después de múltiples pruebas. Estudios adicionales podrían investigar materiales alternativos para mejorar su durabilidad o rendimiento bajo presiones más altas.

5. Conclusiones

El proyecto representa una aplicación didáctica de los principios de la dinámica de fluidos y de propulsión. A lo largo del desarrollo, se logró diseñar, calcular, y construir un sistema de disparo o tubo de choque, ya que aprovecha las ondas de choque para generar el impulso necesario para lanzar objetos, similar a una arma de fuego con los tres zonas principales: disparador, zona de transición, pruebas y de disipación. Tras la ruptura de la membraba se generó la onda de choque debido a la diferencia de presiones a la entrada y a la salida de la zona de prueebas y se expandio a través de la zona de transición.

También, se comprobó que el nivel de presión es un factor clave para alcanzar mayores alturas y generar la onda de choque en la membrana. La elección de materiales resistentes como el PVC, es crucial para soportar las altas presiones y fuerzas generadas en el disparo, lo que asegura la durabilidad y seguridad del sistema durante múltiples lanzamientos.

Se logró construir un sistema funcional capaz de generar ondas de choque y lanzar objetos mediante la ruptura de una membrana de plástico y TPU, este enfoque combina la ingeniería teórica y experimental. Lo que representa una combinación exitosa de diseño teórico, selección de materiales y aplicación práctica de principios de Mecánica de Fluidos.

Con mejoras en la precisión de las mediciones y la incorporación de tecnologías más avanzadas, el sistema podría adaptarse a investigaciones más complejas o aplicaciones específicas en la industria o la academia. Estas mejoras y reflexiones apuntan a continuar desarrollando el proyecto, asegurando que no solo cumpla con sus objetivos actuales, sino que también se convierta en una plataforma versátil para futuras aplicaciones experimentales.

Para recomendaciones futuras investigaciones se sugiere realizar pruebas con un rango más amplio de presiones para evaluar su impacto en la eficiencia del sistema. Incorporar simulaciones computacionales para optimizar el diseño antes de realizar pruebas físicas. Así como, experimentar con materiales alternativos para la membrana de ruptura y las conexiones del sistema.

Direcciones de correo de los autores

Vallejo-Palma, Braulio: bvallejop465@alumno.uaemex.mx

González-Vallejo, Jesús Alberto: jgonzalezv037@alumno.uaemex.mx

Mendioza-Clemente, Raziel Noel: rmendozac185@alumno.uaemex.mx

Gutíerrez-García Karla: kgutierrezg418@alumno.uaemex.mx

Dávila-Vilchis, Juana-Mariel*: mdavilav@uaemex.mx

Referencias

[1] De Celis, M. A. L. (2019). *La historia de las armas de fuego*. Clío: Revista de historia. (210), 72-77.

[2] Pineda Ríos, D. (2021). *Edad de la pólvora. Las armas de fuego en la historia del mundo*. México y la cuenca del pacífico. 10(29), 193-198.

[3] Kopp, C. (2008). *Technology of improvised explosive devices. Defence Today*, 46-49.

[4] Arias Vanegas, M. A. (2017). *Diseño y construcción de un tubo de choque*.

[5] Troya Morales, I. (2022). *Diseño e impresión 3D de un dispositivo físico para simulación de disparo con arma de fuego*.

[6] Mott, R. L. (1996). *Mecánica de fluidos aplicada. Pearson Educación*.

[7] Tolentino-Masgo, S. L. B., Parco, M. A., Caraballo, S., Lacruz, L., Marcano, V., Ferreira, J., & Mírez, J. (2021). *Análisis numérico del comportamiento del flujo en la sección de la garganta de una tobera cónica experimental*. Enfoque UTE, 12(1), 12-28.

[8] Hickel, S., Diegelmann, F., & Tritschler, V. (2015). *Mach Number Effects on Ignition and Mixing Processes in a Reacting Shock-Bubble Interaction*. In APS Division of Fluid Dynamics Meeting Abstracts (pp. G40-001).

[9] Moya, D. (2002). *Terapia por onda de choque extracorpórea para el tratamiento de las lesiones musculoesqueléticas*. Rev Asoc Argent Ortop Traumatol, 67(4), 273-86.

[10] Favorskaya, A. V. (2020). *Fall of shock wave from a supersonic aircraft into the media*. Procedia Computer Science, 176, 2546-2555.

[11] Krehl, P. O. (2008). *History of shock waves, explosions and impact: a chronological and biographical reference*. Springer Science & Business Media.

[12] Lo, K. W., and Ferguson, B. G. (2008). *A ballistic model-based method for ranging direct fire weapons using the acoustic muzzle blast and shock wave*. In 2008 International Conference on Intelligent Sensors, Sensor Networks and Information Processing. IEEE.(pp. 453-458).

[13] Settles, G. S. (2006). *Toma ultrarrápida de imágenes de ondas de choque, explosiones y disparos*. Investigación y ciencia, 75.

[14] Ligrani, P. M., McNabb, E. S., Collopy, H., Anderson, M., & Marko, S. M. (2020). *Recent investigations of shock wave effects and interactions*. Advances in Aerodynamics, 2(1), 4.

[15] Kanamori, M., & Suzuki, K. (2011).*Shock wave detection based on the theory of characteristics for CFD results*. In 20th AIAA Computational Fluid Dynamics Conference (p. 3681).

[16] Guevara Sotelo, N. S. (2019). *Diseño de un Equipo de Generación de Ondas de Choque*.

[17] Palencia Calderón, N. (2019). *Instrumentación de un equipo de generación de ondas de choque*.

[18] White, F. M., & Xue, H. (2003). *Fluid mechanics*. (Vol. 3). New York: McGraw-hill.