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.

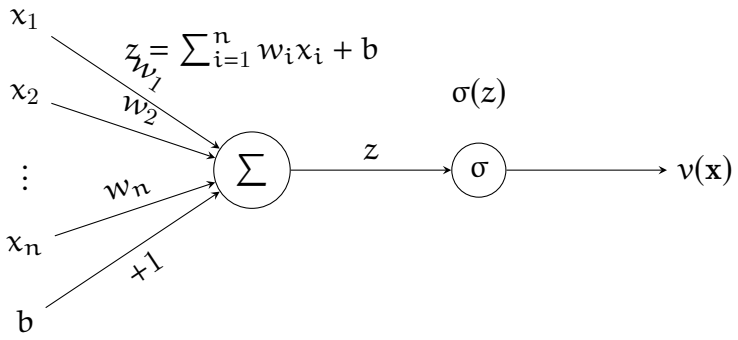
Regresión: función identidad. En tareas de regresión, el objetivo es predecir un valor continuo en lugar de una categoría discreta. En estos casos, es común utilizar la *función identidad* como función de salida. La función identidad simplemente devuelve el valor de salida sin realizar ninguna transformación 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:

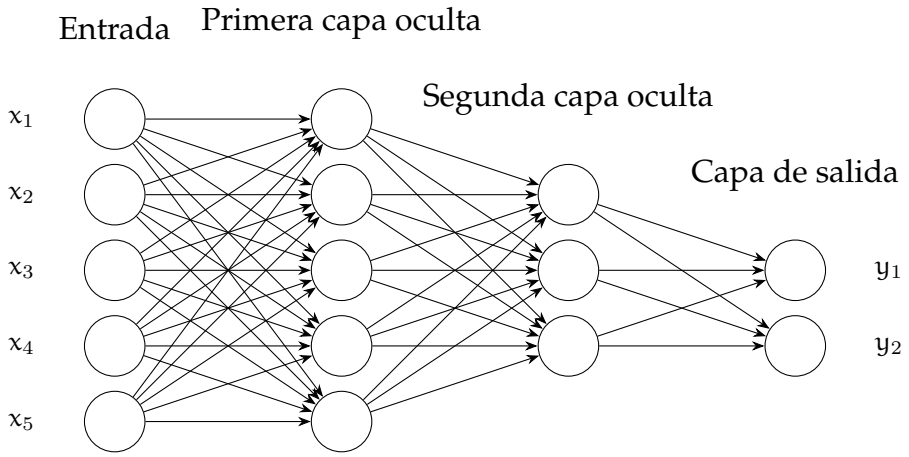
$$\mathbb{R}^d \ni \mathbf{x} \mapsto \nu(\mathbf{x}) = \sigma(\boldsymbol{w}^\intercal \mathbf{x} + b),$$

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.

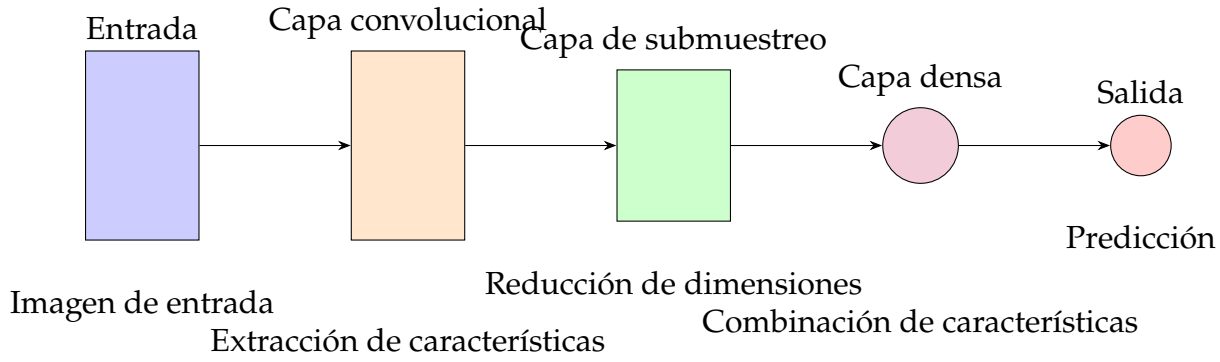
El proceso de retropropagación combinado con el descenso de gradiente estocástico se expresa en los siguientes pasos básicos:

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.

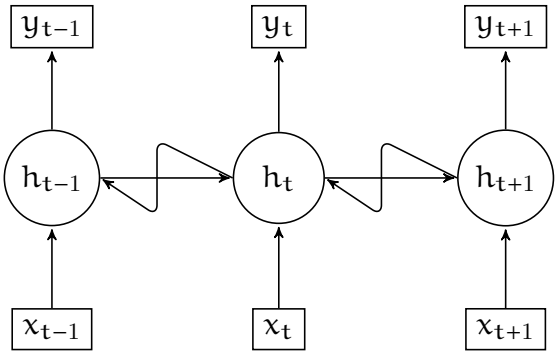
Capas densas: En la etapa final de una CNN, se incluyen una o más capas completamente conectadas (densas), que constituyen una red del tipo FFNN (1) que combina las características detectadas en las capas previas y realizan la clasificación final.

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.

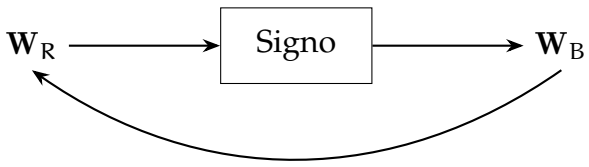
Binarización de pesos

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.

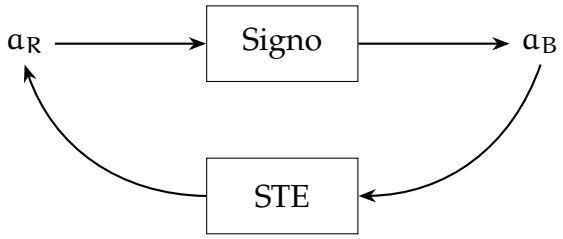
Esta aproximación del gradiente se utiliza para actualizar los pesos reales. La binarización se puede visualizar como una capa en si misma. Los pesos pasan por la capa de binarización que evalúa el signo de los valores en la propagación hacia adelante del BP y aplica la función de identidad durante la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