Paradigmas de la Inteligencia Artificial

DIEGO JAVIER BASTIDAS LOGROÑO

MARCO IVÁN CADENA ASTUDILLO

FLAVIO RAMIRO CADENA ASTUDILLO

DIEGO OMAR GUILCAPI LUNAVICTORIA

ROSA CARMEN ORDOÑEZ MASACO

MÓNICA ALEXANDRA VILLAVICENCIO CHÁVEZ

MIRIAM GRACIELA GAONA AMBULUDÍ

VENANCIO YOBANY GALLEGOS CUENCA

ELISA VIVIANA GALLEGOS CUENCA

FREDDY OSWALDO ELIZALDE SARANGO

SEGUNDO ISAÍAS QUEZADA VALENCIA

EDICIONES GESICAP

Paradigmas de la inteligencia artificial

Diego Javier Bastidas Logroño
Instituto Superior Tecnológico Quito.
ORCID: https://orcid.org/0000-0003-3924-7468.
Correo: diego.bastidas@itq.edu.ec

Marco Iván Cadena Astudillo
Instituto Superior Tecnológico Quito.
ORCID: https://orcid.org/ https://orcid.org/0009-0002-8785-174X.
Correo: ivan.cadena@itq.edu.ec

Flavio Ramiro Cadena Astudillo
Instituto Superior Tecnológico Quito.
ORCID: https://orcid.org/0009-0008-7018-5713 .
Correo: favio.cadena@itq.edu.ec

Diego Omar Guilcapi Lunavictoria
Universidad Nacional de Educación.
ORCID: https://orcid.org/0009-0001-4550-1736.
Correo: diego.guilcapi@unae.edu.ec

Rosa Carmen Ordoñez Masaco
Distrito22D01 Joya de los Sachas, Ministerio de Educación de Ecuador.
ORCID: https://orcid.org/0009-0007-7399-1529.
Correo: rosac.ordonez@educacion.gob.ec

Mónica Alexandra Villavicencio Chávez
Distrito22D01 Joya de los Sachas, Ministerio de Educación de Ecuador.
ORCID: https://orcid.org/0009-0000-6859-8780.
Correo monica.villavicencio@educacion.gob.ec

Paradigmas de la inteligencia artificial

Miriam Graciela Gaona Ambuludí
Distrito22D01 Joya de los Sachas, Ministerio de Educación de Ecuador.
ORCID: https://orcid.org/0009-0003-7961-0740.
Correo: miriam.gaona@educacion.gob.ec

Venancio Yobany Gallegos Cuenca
Distrito22D01 Joya de los Sachas, Ministerio de Educación de Ecuador.
ORCID: https://orcid.org/0009-0004-9855-116.
Correo: venancio.gallegos@educacion.gob.ec

Elisa Viviana Gallegos Cuenca
Distrito21D03 Lago Agrio, Ministerio de Educación de Ecuador.
ORCID: https://orcid.org/0009-0007-5479-9574.
Correo: elisa.gallegos@educacion.gob.ec

Freddy Oswaldo Elizalde Sarango
Universidad Nacional de Educación.
ORCID: https://orcid.org/0009-0008-6117-182X
Correo: freddy.elizalde@unae.edu.ec

Segundo Isaías Quezada Valencia
Instituto Superior Tecnológico Sudamericano.
ORCID: https://orcid.org/0009-0009-0074-0897
Correo: iquiss@gmail.com

Texto arbitrado bajo la modalidad doble par ciego.
El Carmen, Manabí, Ecuador
https://edicionesgesicap.com
ISBN: 978-9942-626-24-0

DEPÓSITO LEGAL:

1ra Edición: Ediciones Gesicap, Calle 24 de julio y Ave. 3 de julio, El Carmen, Manabí, Ecuador.
Derechos de autor © octubre de 2024.

CÓMO CITAR ESTE LIBRO:

Bastidas Logroño, D.J.; Cadena Astudillo, M.I.; Cadena Astudillo, F.R.; Guilcapi Llunavictoria, D.O.; Ordoñez Masaco, R.C.; Villavicencio Chávez, M.A.; Gaona Ambuludí, M.G.; Gallegos Cuenca, V.Y.; Gallegos Cuenca, E.V.; Elizalde Sarango, F.O.; Quezada Valencia, S.I. (2024). Paradigmas de la inteligencia artificial. Ediciones GESICAP. 79 pp.

EQUIPO EDITORIAL:

Edición y Maquetación: Sergio Alejandro Rodríguez Hernández
Revisión y Corrección: Xenia Pedraza González
Diseño de Portada: Sergio Alejandro Rodríguez Hernández.

Índice de Contenidos

Agradecimientos

Agradezco a Elohim, Jehova por darme la vida y el albedrío, lo que la inteligencia artificial no lo tendrá, a mis padres Galo y Monserratt, a mis hijas Cristina y Sariah, a mis abuelitas Blanca Piedad y María Cristina + , a mi profesor universitario de las cátedras de inteligencia artificial, sistemas inteligentes y base de conocimientos Mg. Wladimir Castro Salazar, familiares y amigos.

Diego Javier Bastidas Logroño

Soy agradecido con la vida por las oportunidades que me da para aprender. Soy agradecido con Dios, porque me da la vida.

Marco Iván Cadena Astudillo

Este libro no habría sido posible sin el apoyo incondicional de las personas más importantes en mi vida. A mi querida esposa, Fabiola, por su amor, paciencia y constante ánimo. Gracias por creer en mí incluso en los momentos más difíciles y por ser mi mayor fuente de inspiración. Tu compañía y comprensión han sido fundamentales en cada etapa de este proyecto. A mi hijo, Favio, cuya alegría y curiosidad inagotable me han recordado la importancia de perseguir nuestros sueños. Tu energía y entusiasmo me han impulsado a seguir adelante cuando más lo necesitaba. A mi hermano, Iván, por su incansable apoyo y sabios consejos. Gracias por estar siempre dispuesto a escuchar mis ideas y por ofrecerme tu perspectiva única, que ha enriquecido profundamente este trabajo. A todos ustedes, les debo mi más sincero agradecimiento. Este libro es tanto suyo como mío.

Flavio Ramiro Cadena Astudillo

Agradecimientos

Quiero expresar mi más sincero agradecimiento a Dios a mis padres, a mis hermanos, sobrinos, cuñados a Mónica y a mis hijos Omar y Odalys por su apoyo continuo. Cada uno de ustedes ha contribuido de manera invaluable y a superar los desafíos.

Diego Omar Guilcapi Llunavictoria

Gracias a Dios por permitirme cumplir mis anhelos, objetivos personales y profesionales en mi vida, a mis padres por darme el don de vida e inculcar aquellos valores para ser mejor persona en la sociedad, a mi hija Damaris por ser mi compañera de vida en cada paso que doy, siendo la fortaleza e impulso de superación personal y por alentarme a seguir un camino hacia el éxito.

Rosa Carmen Ordoñez Masaco

Quiero expresar mi más sincero agradecimiento a Dios a mis padres, a mi esposo Diego e hijos Omar y Odalys por su apoyo continuo. Cada uno de ustedes ha contribuido de manera invaluable a este éxito, me ha permitido crecer y superar los desafíos que hemos enfrentado juntos.

Mónica Alexandra Villavicencio Chávez

Hoy quiero tomarme un momento para expresar a mi esposo, mi más profundo agradecimiento por el inmenso apoyo que me has brindado durante el proceso de elaboración de mi libro científico, tus palabras de ánimo han sido mi motivación constante cuando enfrentaba los desafíos y obstáculos propios de este proceso creativo, quiero agradecerte profundamente por tu contribución económica, tu creencia inquebrantable en mí y en mi capacidad para lograr mis metas ha sido mi mayor fortaleza.

Miriam Graciela Gaona Ambuludí

Agradecimientos

Mi gratitud se extiende a nuestro creador, por su amor incondicional que me demuestra día tras día, por llenar mi vida de sabiduría, templanza y fortaleza. A mis padres, hermanos e hijos quienes alimentan mi corazón y espíritu, a mis amigos, lo que permite lograr que esos sueños se vuelvan realidad.

Venancio Yobany Gallegos Cuenca

En primer lugar, agradezco a Dios por la vida, la salud y la sabiduría, que me han permitido alcanzar esta meta. A mis padres y hermanos, por su paciencia, amor y comprensión en cada etapa de mi vida y en las nuevas metas a ser alcanzadas. Su apoyo incondicional ha sido una fuente constante de motivación y fortaleza. En especial, a mi hijo Cristopher: tu presencia y amor han sido una inspiración y una fuerza constante para mí. A mis amigos, amigas y compañeros de trabajo, quienes siempre estuvieron dispuestos a colaborar y brindar sus ideas y opiniones constructivas. Su apoyo y ánimo han sido invaluables en este viaje.

Elisa Viviana Gallegos Cuenca

Mi gratitud se extiende a nuestro creador, por su amor incondicional que me demuestra día tras día, por llenar mi vida de sabiduría, templanza y fortaleza. A mis padres, hermanos e hijos quienes alimentan mi corazón y espíritu, lo que permite lograr que esos sueños se vuelvan realidad.

Freddy Oswaldo Elizalde Sarango

Agradezco a Dios por darme batallas y ponerme a prueba en cada una de ellas, permitiéndome crecer y fortalecerme. A mi familia, por su amor y apoyo constante, por mantenerme firme en este mundo y brindarme la fortaleza para alcanzar mis sueños. Su presencia ha sido mi luz en el camino y su fe en mí, mi mayor inspiración.

Segundo Isaías Quezada Valencia

Prólogo

La inteligencia artificial es una rama de las ciencias de la computación que se encarga de estudiar mecanismos y metodologías que permitan simular el comportamiento humano en dispositivos de middleware, se encarga de desarrollar herramientas que simulen la inteligencia humana, como consecuencia directa de esto también ayuda a fortalecer el estudio de la mente humana y la forma que el cerebro produce el pensamiento, es una nueva generación de tecnología informática caracterizada no solo por su arquitectura (hardware), sino también por sus capacidades.

La humanidad se encuentra en el umbral de una nueva era educativa, impulsada por la innovación tecnológica y el avance exponencial de la inteligencia artificial (IA). Esta revolución tecnológica está transformando la manera en que aprendemos, enseñamos y concebimos la educación en su totalidad. La IA, una vez relegada a la ciencia ficción, se ha convertido en una herramienta poderosa y omnipresente que promete remodelar los cimientos de los sistemas educativos en todo el mundo.

El énfasis de generaciones previas fueron las computadoras numéricas para aplicaciones científicas o de negocios, la nueva generación de tecnología informática incluye además la manipulación simbólica, con el objetivo de emular el comportamiento inteligente y la computación en paralelo para conseguir resultados en tiempo real, la capacidad predominante es superar en ciertas funciones inteligentes a las funciones del ser humano.

Los sistemas de inteligencia artificial adquieren conocimiento a partir de la educación o de la experiencia, la computadora obtiene el conocimiento de forma general a partir de uno o varios experimentos humanos en determinada rama del saber ya sean estos conceptos, teorías, relaciones y procedimientos, esta información que ha sido organizada y analizada para ser aplicada en la efectiva solución de problemas y en la toma de decisiones.

Que este viaje sea una fuente de inspiración y conocimiento, y que nos ayude a todos a construir un futuro educativo más brillante, inclusivo y justo. Bienvenidos a una exploración profunda y apasionante de la educación con inteligencia artificial.

Diego Javier Bastidas Logroño

I.

El conocimiento

El componente esencial de cualquier sistema de inteligencia artificial radica en el conocimiento que adquiere a través del aprendizaje o de experiencias previas. Generalmente, una computadora desarrolla este conocimiento basándose en uno o varios experimentos realizados por humanos en áreas específicas del conocimiento. Este está compuesto por datos, teorías, conceptos, relaciones y métodos, los cuales constituyen información que ha sido estructurada y evaluada para su uso en la resolución de problemas y la toma de decisiones.

1.1 Breve reseña histórica de la inteligencia artificial

En el año de 1956 se utiliza por primera vez el término de inteligencia artificial para expresar potencialidades de la computadora. (Moor, 2006).

Desde el año 1955 a 1960 se demuestra la posibilidad de programar aperturas de ajedrez. (Newell, 1958)

En 1957 John McCarthy crea el lenguaje LISP, que es un lenguaje propio de la inteligencia artificial (McCarthy, 1996).

En 1965 a 1969 se lleva a cabo el proyecto RENDRAL que es el primer sistema experto el cual determinaba la estructura molecular a partir de la densidad espectral y ensayos magnéticos moleculares. (Smith, 2020).

En 1966 se crea el sistema MACSYMA (segundo Sistema Experto) que ayudaba a resolver problemas matemáticos, algebraicos, de integrales vectoriales, de inecuaciones de series (Moses, 1983).

De 1967 a 1969 se exponen las posibilidades de los sistemas expertos en los congresos de inteligencia artificial de los Estados Unidos de América, en el cual se expone el proyecto MYCIN que determina el ente probable de una infección sanguínea y prescribe el tratamiento (Buchanan, 1984).

De 1970 a 1974 Alain Colmeraver, crea el lenguaje prolog (programación lógica). (Colmerauer, 1992)

En 1972 se crea el sistema SHRDLU (Sistema de comprensión de lenguaje natural interactivo para mover bloques con un brazo mecánico) (Winograd, 1972).

En 1975 se crea el sistema PARRY (Simulación de un paranoico) (Colby, 1975).

De 1975 a 1980 se crea el sistema ELISA (Simulación de un psicoterapeuta). (Weizenbaum, 1966).

En la década de los 80 se desarrollan los sistemas de juegos de video con elementos de inteligencia artificial (Rollings, 2003).

En los 90 el impulso lo da la realidad virtual y la robótica (Rollings, 2003).

En la actualidad la inteligencia artificial puede prevenir muertes prematuras (Infosalus, 2019).

La inteligencia artificial (IA) puede considerarse en parte como ingeniería y en parte como ciencia. Como ingeniería, el objetivo de la IA es resolver problemas reales actuando como un conjunto de ideas acerca de cómo representar y utilizar el conocimiento, como desarrollar sistemas informáticos. Como ciencia, el objetivo de la IA es buscar la explicación de diversas clases de inteligencia, a través de la representación del conocimiento y de la aplicación de este en los sistemas informáticos desarrollados. El cambio de milenio trajo consigo avances en el aprendizaje automático (machine learning) y el procesamiento de grandes volúmenes de datos (big data). Las mejoras en la capacidad de procesamiento y el almacenamiento de datos permitieron la creación de modelos de aprendizaje más complejos y precisos. (Fernandez, 2006).

Partes de un sistema de inteligencia artificial

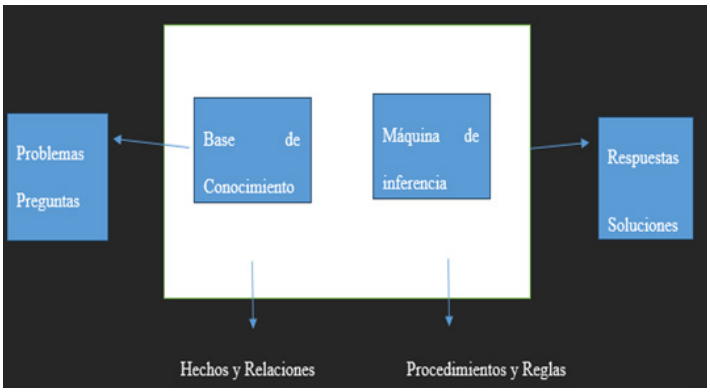

Ilustración 1: Partes de un sistema de inteligencia artificial. Fuente: Los autores

Un sistema de inteligencia artificial consta de dos partes fundamentales que son:

B.C.= Base de conocimiento

M.I. = Máquina de inferencia

La base de conocimientos está formada por hechos y relaciones, la máquina de inferencia tiene los procedimientos y reglas para trabajar con la base del conocimiento.

A partir de la creación de la base de conocimiento la computadora está en condiciones de usar las técnicas de inteligencia artificial para pensar y razonar.

En un programa de inteligencia artificial, se instruye a la computadora sobre el problema específico a resolver. Estos programas operan mediante la representación y manipulación de símbolos. Un símbolo puede ser una letra, una palabra o un número que se utiliza para representar objetos, procesos y las relaciones entre ellos.

1.1.1. La inteligencia artificial vs. la computación tradicional

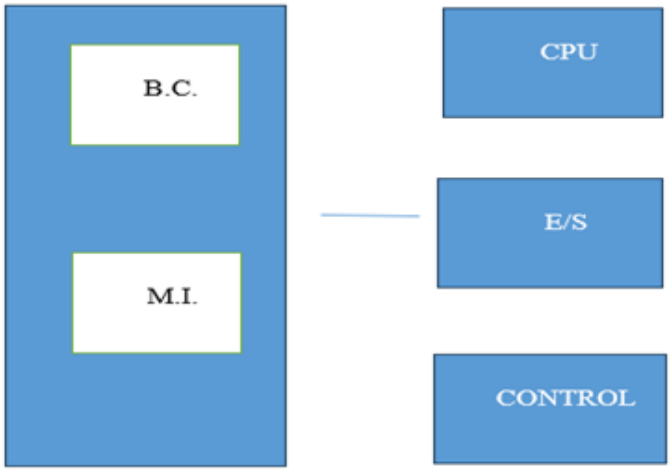

Ilustración 2: Inteligencia artificial. Fuente: Los autores

Los programas tradicionales se fundamentan en algoritmos que dictan cada paso a seguir. En dichos programas, se refieren a las computadoras en cómo abordar y resolver un problema específico. La inteligencia artificial y la computación convencional constituyen dos metodologías complementarias dentro del campo de la informática. La computación convencional es esencial para ejecutar tareas con reglas claras y bien establecidas, mientras que la inteligencia artificial se destaca por su capacidad de adaptación y flexibilidad en situaciones dinámicas y complejas. Estos dos enfoques seguirán desarrollándose y enriqueciéndose mutuamente, fomentando avances tecnológicos y transformando variados ámbitos de la sociedad.

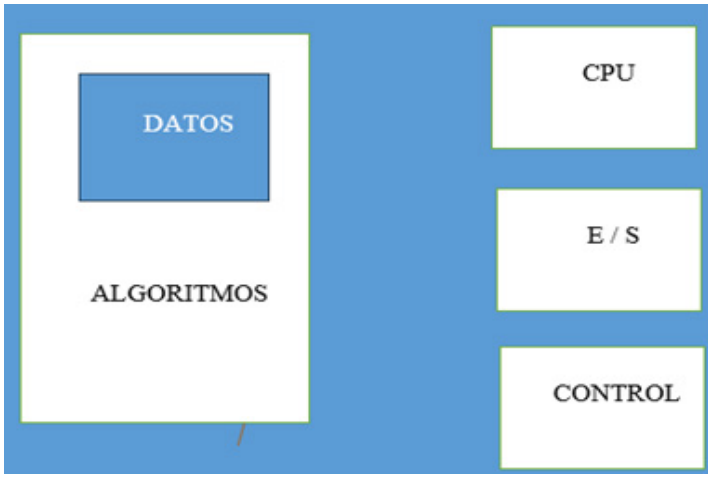

Ilustración 3: Computación tradicional.
Fuente: Los autores

1.1.1.1. Test de Turing

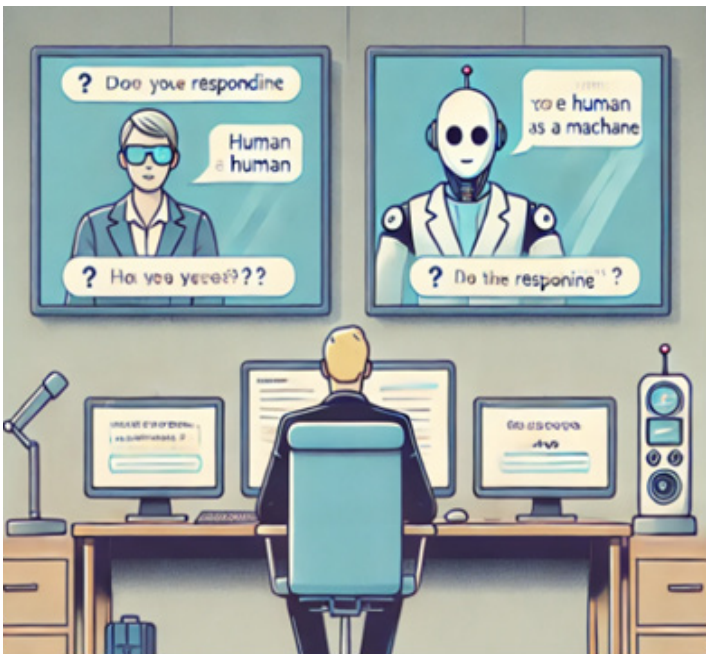

Ilustración 4: Test de Turing. Fuente: Chatgpt4o.

Evaluar la inteligencia de un programa de inteligencia artificial es una tarea desafiante. De manera similar, medir la inteligencia humana es complicado debido a la gran cantidad de factores subjetivos y al hecho de que el cerebro humano aún es objeto de profundo estudio. En el campo de la inteligencia artificial, se han ideado diversas pruebas y exámenes para determinar si un programa o computadora posee capacidades inteligentes, entre ellos se encuentra el test de Turing que se puede describir de la siguiente manera:

- Una computadora conecta a un operador y a una máquina con inteligencia artificial.

- En la computadora hay un operador emitiendo preguntas constantemente sin saber de qué máquina proviene la respuesta.

- Si al final el test del operador piensa que las respuestas que obtuvo venían de un ser humano, se dice entonces que la máquina con inteligencia artificial es inteligente.

Todos los test tienen una efectividad limitada porque es complicado calificar la inteligencia (Merino E., 2024).

1.1.1.2. El razonamiento de modelos Transformers

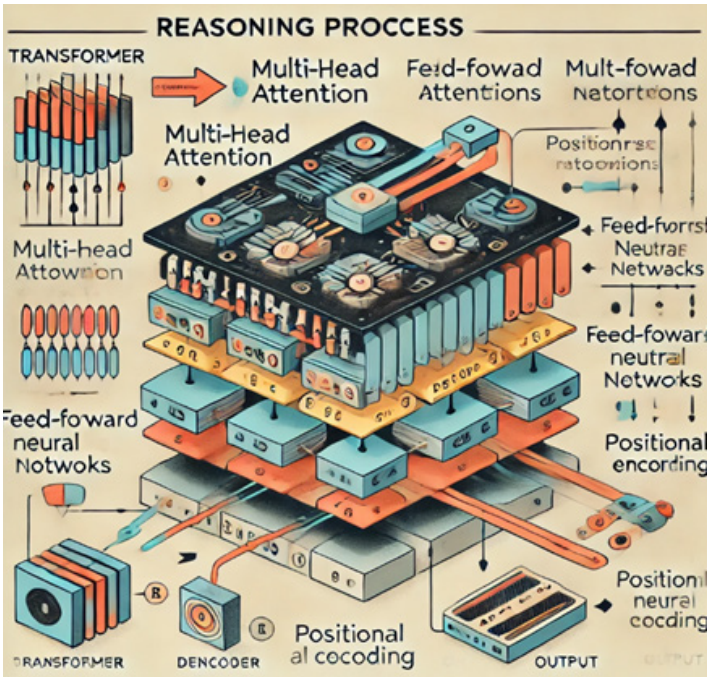

Ilustración 5: Modelos Transformers.
Fuente: Chatgpt4o

Los Transformers constituyen una de las arquitecturas más aplicadas en el aprendizaje automático. Aunque las redes neuronales han avanzado significativamente, aún no alcanzan la robustez y flexibilidad de los humanos al procesar simultáneamente una variedad de estímulos, esto se debe a sus limitaciones en generalización y en la capacidad de utilizar su experiencia previa para gestionar con precisión entradas inesperadas y naturales. (Moreira, 2023).

Los transformadores y las redes neuronales han avanzado considerablemente gracias a los desarrollos en hardware, en particular las unidades de procesamiento gráfico (GPU) de NVIDIA. Este avance ha sido crucial para acelerar la computación en el campo de la inteligencia artificial, donde NVIDIA ha jugado un papel fundamental al proporcionar la infraestructura necesaria para entrenar y desplegar modelos de transformadores a gran escala (Beyaz, 2023).

GPU de NVIDIA:

CUDA: La arquitectura CUDA de NVIDIA permite la computación paralela masiva, esencial para entrenar grandes modelos de transformadores (Reinders, 2023).

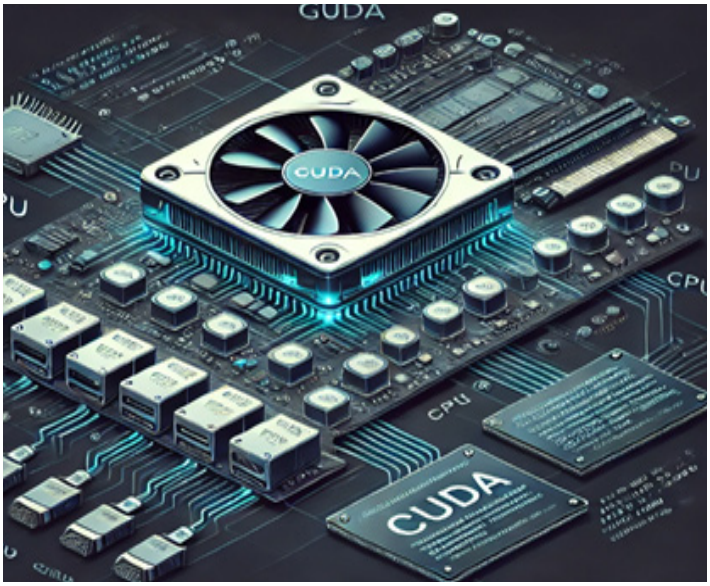

Ilustración 6: Cuda. Fuente: Chatgpt4o

Tensor Cores:

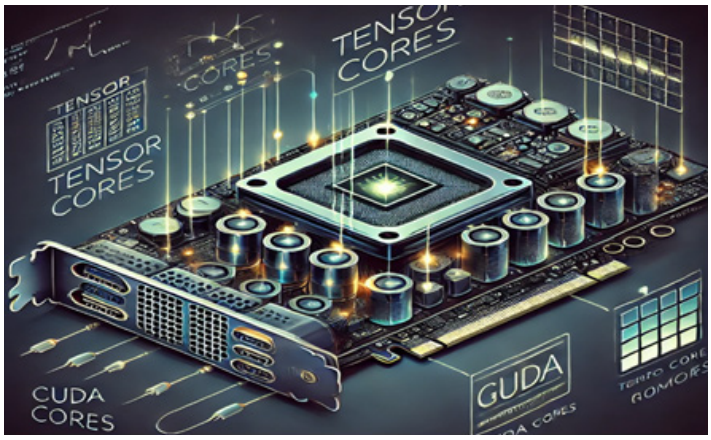

Ilustración 7: Tensor Cores. Fuente: Chatgpt4o

Introducidos en las GPU Volta, Tensor Cores están diseñados para acelerar las operaciones de aprendizaje profundo, permitiendo un entrenamiento más rápido de redes neuronales (Markidis, 2018).

Plataforma NVIDIA AI:

NVIDIA DGX Systems: Sistemas de supercomputación diseñados específicamente para IA, que incluyen múltiples GPU de alto rendimiento.
NVIDIA A100: La GPU A100, basada en la arquitectura Ampere, ofrece un rendimiento sin precedentes para cargas de trabajo de IA, siendo especialmente eficiente en el entrenamiento y la inferencia de modelos de transformadores. (Reinders, 2023).

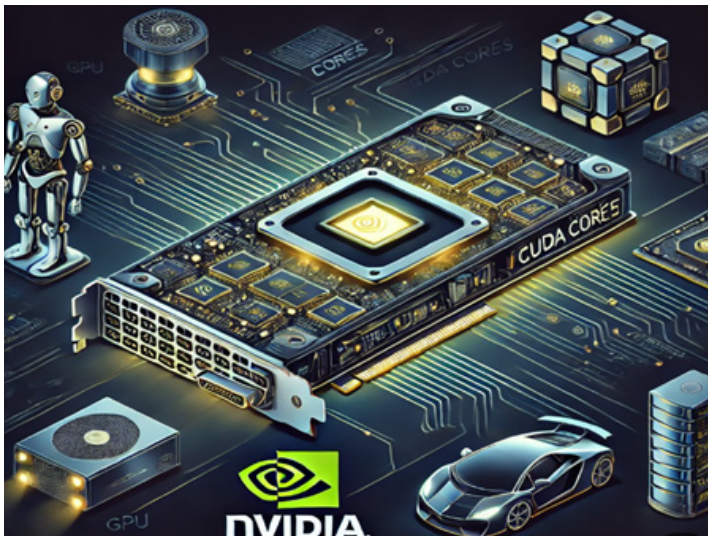

Ilustración 8: Nvidia revoluciona la IA. Fuente: Chatgpt4o

Software y Herramientas:

NVIDIA CUDA-X AI: Un conjunto de bibliotecas optimizadas para IA y análisis de datos que se ejecutan en GPU NVIDIA.

NVIDIA TensorRT: Una biblioteca de inferencia de alto rendimiento para desplegar modelos de aprendizaje profundo en producción.

NVIDIA Triton Inference Server: Un servidor de inferencia de aprendizaje profundo que facilita el despliegue y la gestión de modelos en producción. (Reinders, 2023).

Ejemplos de Aplicación de Transformadores con Tecnología NVIDIA

Entrenamiento de Modelos a Gran Escala:

GPT-3: Este modelo, que cuenta con cientos de miles de millones de parámetros, demanda una cantidad considerable de capacidad computacional. Las GPU de NVIDIA facilitan el entrenamiento distribuido de modelos como GPT-3 en periodos de tiempo prácticos.

GPT-4 y NVIDIA: GPT-4 es un modelo de inteligencia artificial diseñado específicamente para el procesamiento del lenguaje natural. Por su parte, NVIDIA proporciona el hardware y software necesarios que potencian una amplia gama de aplicaciones de IA, incluyendo el entrenamiento de modelos avanzados como GPT-4.

BERT y Sus Variantes: Modelos como BERT, RoBERTa, entre otros, se aprovechan de la aceleración que ofrecen las GPU de NVIDIA. Esta tecnología permite el entrenamiento de modelos de gran tamaño sobre extensos conjuntos de datos de manera eficiente.

Inferencia Rápida y Eficiente:

Inferencia en Tiempo Real: Herramientas como TensorRT y Triton Inference Server permiten realizar inferencias rápidas en aplicaciones de producción, crucial para servicios en tiempo real como chatbots y asistentes virtuales.

Optimización de Modelos: TensorRT optimiza los modelos de transformadores para maximizar la eficiencia en las GPU, reduciendo la latencia y el consumo de energía.

Investigación y Desarrollo:

Modelos Innovadores: Los investigadores pueden experimentar con arquitecturas de transformadores más complejas y con mayor cantidad de parámetros, gracias a la capacidad de cómputo proporcionada por las GPU de NVIDIA.

Simulación y Análisis: La capacidad de realizar simulaciones y análisis complejos en áreas como la biología computacional y la física, donde los transformadores se están comenzando a aplicar.

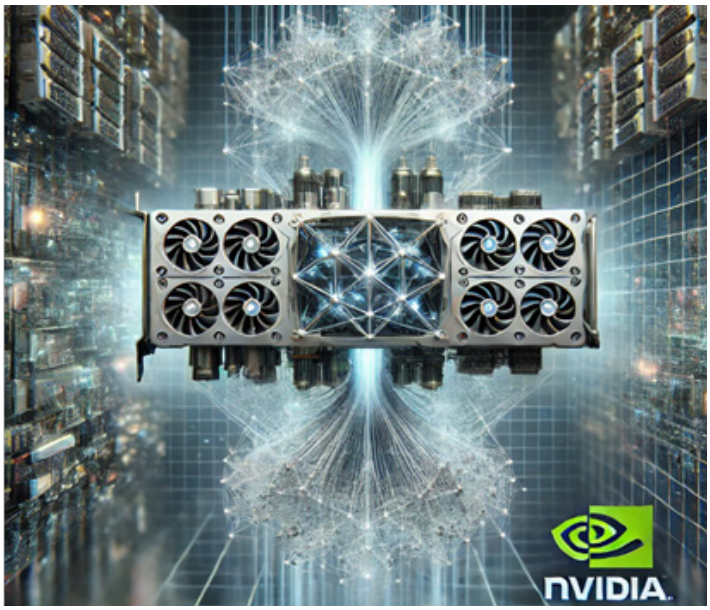

Ilustración 9: Megatron Lm. Fuente: Los Autores

Caso de Estudio: Megatron-LM

Megatron-LM es un proyecto de NVIDIA que aprovecha el potencial de sus GPU para entrenar modelos de lenguaje de gran escala. Este modelo de transformador, diseñado para ser escalable, puede gestionar modelos con hasta cientos de miles de millones de parámetros. Esto ilustra la capacidad de la infraestructura de NVIDIA para soportar algunas de las tareas computacionales más exigentes en el ámbito de la inteligencia artificial. (Shoeybi, 2020).

1.1.2. Modelos de representación del conocimiento.

Para incorporar conocimiento en una computadora, es necesario desarrollar un modelo de conocimiento que luego se traduce en los símbolos requeridos para su completa representación. La inteligencia artificial dispone de diversos modelos de representación de conocimiento que los desarrolladores de software inteligente han empleado para construir sus aplicaciones. (Yu, 2020).

1.1.2.1. Lógica

La lógica es comúnmente utilizada tanto en el lenguaje coloquial como en el científico, con expresiones habituales como "esto no es lógico" o "de acuerdo con la lógica", que se asocian a lo razonable. En este sentido, se podría decir que todos los seres humanos son lógicos, ya que razonan de manera espontánea y natural. Por lo tanto, existe una lógica natural, que es la tendencia innata de la razón humana para actuar correctamente en todas sus manifestaciones (Bastidas, 2024). Sin embargo, la lógica como ciencia tiene una definición más precisa, centrada en el estudio de la estructura del pensamiento y establece las metodologías adecuadas mediante las cuales la razón puede evitar errores y acceder a la verdad. Esta disciplina formaliza el pensamiento intuitivo y examina los procesos de razonamiento, así como los sistemas de reglas y procedimientos que se aplican en el mismo, constituyendo el marco de cualquier razonamiento lógico (sistema lógico). (Yu, 2020).

1.1.2.2. Juicios:

Son aquellos elementos simples para un razonamiento, concepto que se tienen del medio, es el hecho en el cual se afirma o se niega una idea.

1.1.2.3. Clasificación:

Por la calidad:

Afirmativos: Son aquellos que indican la relación positiva de sus elementos. Ejemplo: Diego es hombre.

Negativos: Establecen negación en la relación de sus elementos. Ejemplo: Mariel no es estudiante

Por la cantidad:

Universales: Cuando incluyen a todos los elementos que se están hablando. Ejemplo: Todas las aves ponen huevos, todos los hombres duermen.

Particulares: Cuando se incluye a un grupo de la totalidad de elementos de los que estamos hablando. Ejemplo: Algunos peces son de agua dulce, algunas aves vuelan.

Singulares: Cuando se habla de un elemento específico de la generalidad Ejemplo: Javier es alto.

1.1.2.4. Razonamiento

Proceso de inferencia basado en el procesamiento de premisas (juicios) mediante mecanismos inherentes a la inteligencia humana.

Tipos de Razonamiento en IA

Razonamiento Deductivo: Parte de premisas generales para llegar a conclusiones específicas. Es lógico y siempre produce conclusiones ciertas si las premisas son verdaderas. Ejemplo: "Todos los humanos son mortales. Sócrates es humano. Por lo tanto, Sócrates es mortal."

Razonamiento Inductivo: Parte de observaciones específicas para llegar a conclusiones generales. Es probabilístico y las conclusiones pueden no ser siempre ciertas. Ejemplo: "He observado que el sol sale por el este todos los días. Por lo tanto, el sol siempre saldrá por el este."

Razonamiento Abductivo: Parte de una observación y busca la mejor explicación posible. Es comúnmente utilizado en diagnóstico médico y detección de fallos (Venegas, 2023). Ejemplo: "La planta está marchita. La mejor explicación es que necesita agua."

II.
La inteligencia artificial no es sólo ChatGPT

La inteligencia artificial (IA) posee una amplia gama de tecnologías y aplicaciones más allá de ChatGPT. (Vega, 2023)

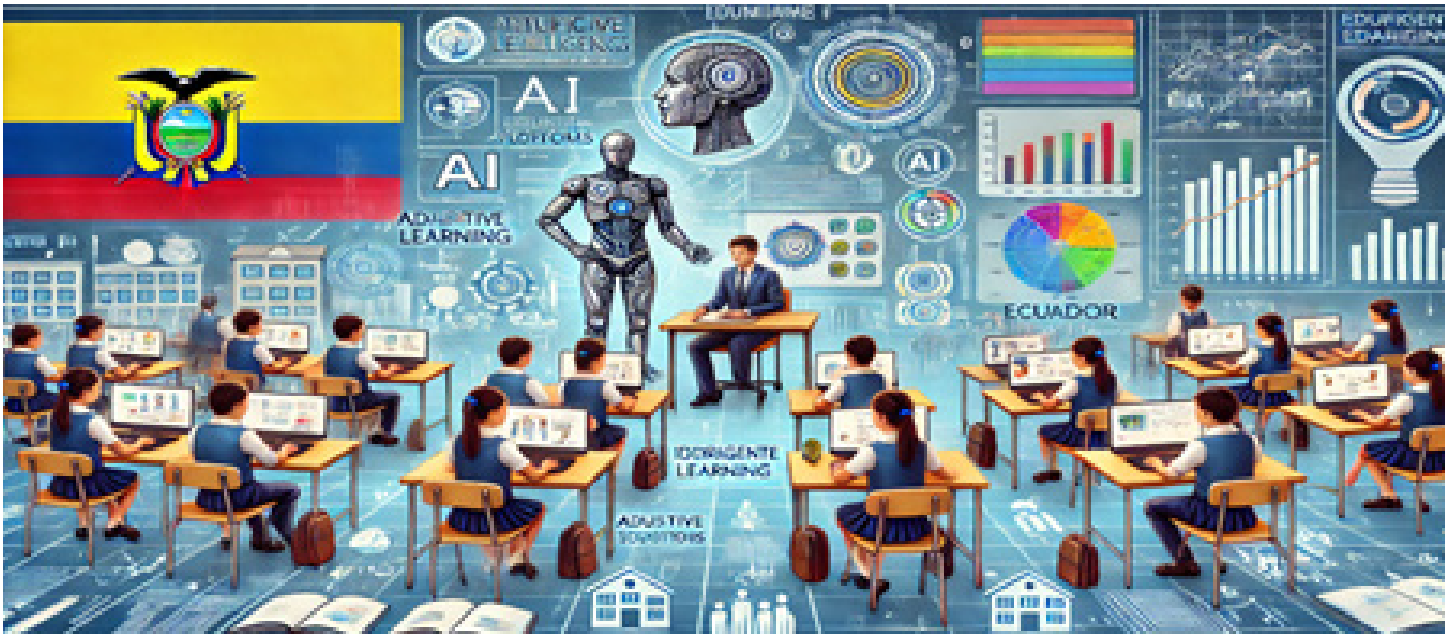

Ilustración 10. La inteligencia artificial no solo es Chatgpt. Fuente: Chatgpt4o

2.1 Inteligencia Artificial Generativa

Es una especialización dentro de la inteligencia artificial dedicada a la creación de contenido original basándose en datos preexistentes. Esta tecnología emplea algoritmos avanzados y redes neuronales para analizar y aprender de diversos tipos de datos, como textos e imágenes. A partir de este aprendizaje, la tecnología es capaz de producir nuevo contenido que presenta características análogas a los datos utilizados durante el entrenamiento. (Vega, 2023).

Ilustración 11. Proyección de una sala de control de alta tecnología. Fuente: Chatgpt4o

La inteligencia artificial generativa describe los sistemas de IA diseñados para crear contenido nuevo y original, incluyendo texto, imágenes, música y otros tipos de datos, a partir de los patrones identificados en los datos de entrenamiento. Estos modelos generativos emplean diversas técnicas avanzadas, tales como redes neuronales generativas adversariales (GANs), redes neuronales recurrentes (RNNs), transformadores y otras tecnologías, para facilitar la producción de contenido innovador basado en el aprendizaje automático. (Vega, 2023).

2.1.1. Agentes

Los agentes de IA generativa representan un avance en los grandes modelos de lenguaje (LLM), transformándolos de simples asistentes a entidades más sofisticadas capaces de tomar decisiones e interactuar con datos y aplicaciones empresariales. Las características distintivas de estos agentes incluyen la habilidad de consultar y extraer información de múltiples fuentes de datos empresariales usando lenguaje natural. Además, generan respuestas dinámicas en tiempo real basadas en la información actualizada en lugar de depender de contenido pregenerado. Estos agentes se integran con aplicaciones empresariales para automatizar tareas y procesos, ofreciendo una interfaz

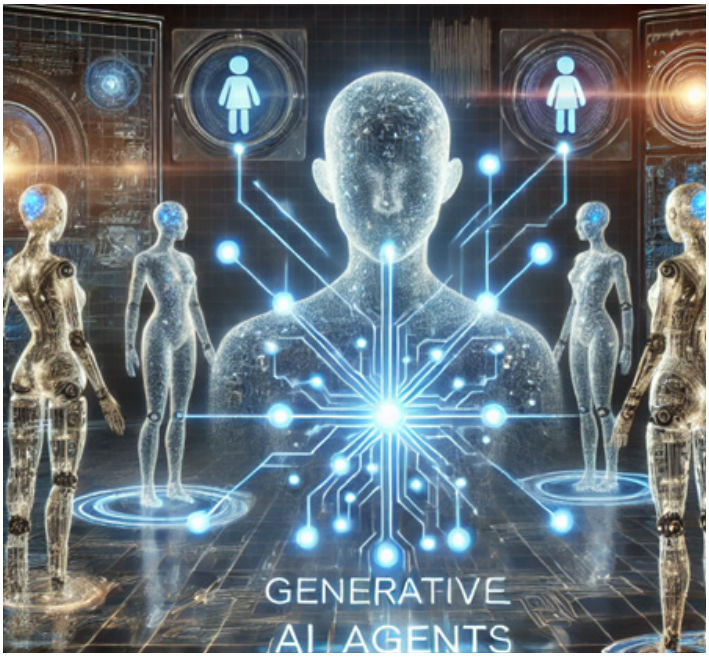

Ilustración 12. Agentes de inteligencia artificial generativos. Fuente: Chatgpt4o

conversacional que facilita el acceso al conocimiento y datos empresariales a cualquier usuario. Utilizan la generación de recuperación aumentada (RAG) para integrar modelos de lenguaje con datos estructurados, logrando resultados más coherentes y precisos. Esta tecnología tiene el potencial de revolucionar cómo las organizaciones acceden a la información y toman decisiones basadas en datos. (Yu, 2020).

Algunos ejemplos de cómo los agentes de IA generativa pueden ser utilizados incluyen:

Servicio al Cliente: Estos agentes pueden proporcionar a los representantes de servicio al cliente acceso rápido al historial de los clientes y facilitar una resolución más eficiente de las consultas.

Recursos Humanos: Permiten a los profesionales de RRHH buscar y seleccionar candidatos potenciales fácilmente al consultar las bases de datos de empleados utilizando lenguaje natural.
Finanzas: Facilitan a los equipos financieros la obtención de información sobre tendencias y pronósticos de compras simplemente conversando con un agente de inteligencia artificial, en lugar de tener que ejecutar informes complicados.

Para implementar estos agentes, las empresas pueden utilizar servicios como los ofrecidos por Oracle Cloud Infrastructure (OCI) de Oracle, que integran grandes modelos de lenguaje (LLM) con fuentes de datos empresariales utilizando la tecnología de generación de recuperación aumentada (RAG). La emergencia de los agentes de IA generativa marca un cambio significativo en la manera en que los humanos interactúan y se benefician de los sistemas de inteligencia artificial, evolucionando de ser simples asistentes pasivos a convertirse en agentes activos y capacitados que pueden tener un impacto tangible en los negocios. (Coloma, 2020).

2.1.2. Q lora (adaptación cuantificada de bajo rango)

Ilustración 13. Quantum Logic Operations for AI. Fuente: Chatgpt4o

Es Q-LoRA es una técnica de ajuste avanzada para modelos de lenguaje grandes (LLM) que se fundamenta en los principios de LoRA (adaptación de bajo rango). Esta técnica tiene como objetivo optimizar la eficiencia y el rendimiento de los LLM para tareas o dominios específicos. Las principales características de Q-LoRA son:

Descomposición de Rango Bajo: Al igual que LoRA, Q-LoRA implementa una descomposición de rango bajo en las matrices de peso de modelos de lenguaje previamente entrenados. Esta estrategia

permite actualizaciones de parámetros más eficientes durante el ajuste fino, reduciendo así los requerimientos computacionales y de memoria en comparación con un ajuste fino completo.

Cuantización: Además de la descomposición de bajo rango, Q-LoRA aplica cuantización a los parámetros del modelo. Esto disminuye la precisión de las representaciones numéricas, lo que resulta en una reducción del tamaño del modelo y de la latencia durante la inferencia.

Los beneficios que Q-LoRA aporta a las aplicaciones de IA generativa incluyen:

Eficiencia: La reducción en el número de parámetros y el tamaño del modelo facilita un ajuste y una inferencia más rápidos, adaptando de manera más práctica los LLM a casos de uso específicos.

Escalabilidad: Q-LoRA posibilita el ajuste de modelos de lenguaje grandes y potentes en hardware con recursos más limitados, como dispositivos de borde, ampliando las posibilidades de implementación de la IA generativa en una gama más diversa de aplicaciones.

Accesibilidad: La reducción de los requisitos computacionales y de memoria hace que la IA generativa sea más accesible para organizaciones y desarrolladores con menos recursos.

Sostenibilidad: La menor demanda de energía y la reducción en la huella de carbono asociadas con Q-LoRA promueven prácticas de inteligencia artificial más respetuosas con el medio ambiente. (Yu, 2020).

2.1.3. Big Gan

BigGAN es una avanzada arquitectura de red generativa adversarial (GAN) que ha logrado progresos significativos en la creación de imágenes de alta resolución y gran fidelidad. Las características principales de BigGAN incluyen:

Ampliación del tamaño del modelo y del lote: BigGAN utiliza modelos más grandes, con un aumento de 2 a 4 veces en el número de parámetros, y tamaños de lote hasta 8 veces mayores que las GANs anteriores. Esto facilita la generación de imágenes de mucha mayor calidad.

Innovaciones arquitectónicas: BigGAN incorpora modificaciones como módulos de autoatención y conexiones omitidas desde la entrada latente hacia las capas generadoras, además de un truco de truncamiento durante la inferencia. Estas innovaciones mejoran el rendimiento y la estabilidad de la red.

Generación condicional de clase: Diseñado para la generación de imágenes condicionales de clase, BigGAN permite la creación de imágenes basadas tanto en un vector latente como en una etiqueta de clase, ofreciendo control sobre las categorías de las imágenes generadas.

Rendimiento de última generación: Entrenado en ImageNet a una resolución de 128x128, BigGAN ha establecido nuevos estándares en métricas como el Inception Score y la Fréchet Inception Distance. Además, es capaz de generar imágenes de alta calidad en resoluciones de 256x256 y 512x512. (Coloma, 2020).

Ilustración 14. Big gan. Fuente: Chatgpt4o.

BigGAN, con su capacidad para generar imágenes de alta resolución y fidelidad, ofrece un amplio abanico de aplicaciones en sectores como los videojuegos, el cine, la arquitectura y las imágenes médicas, donde la calidad de las imágenes es crucial. Sin embargo, el uso de esta tecnología también suscita preocupaciones éticas significativas,

```
pip install torch torchvision
```

2. Código de Implementación

```python
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms

# Definición del Generador
class Generator(nn.Module):
    def __init__(self, latent_dim, img_size, channels):
        super(Generator, self).__init__()
        self.init_size = img_size // 4
        self.l1 = nn.Sequential(nn.Linear(latent_dim, 128 * self.init_size ** 2))

        self.conv_blocks = nn.Sequential(
            nn.BatchNorm2d(128),
            nn.Upsample(scale_factor=2),
            nn.Conv2d(128, 128, 3, stride=1, padding=1),
            nn.BatchNorm2d(128, 0.8),
            nn.ReLU(inplace=True),
            nn.Upsample(scale_factor=2),
            nn.Conv2d(128, 64, 3, stride=1, padding=1),
            nn.BatchNorm2d(64, 0.8),
            nn.ReLU(inplace=True),
            nn.Conv2d(64, channels, 3, stride=1, padding=1),
            nn.Tanh(),
        )

    def forward(self, z):
        out = self.l1(z)
        out = out.view(out.shape[0], 128, self.init_size, self.init_size)
        img = self.conv_blocks(out)
```

Ilustración 15. Código conceptos clave Big Gan en phytorch. Fuente: Los autores

especialmente en relación con la producción de imágenes falsas que parecen reales. Estas preocupaciones abarcan desde la desinformación hasta el uso indebido en contextos sensibles como la política y la seguridad, además se destaca como un logro significativo en la inteligencia artificial generativa, demostrando el impacto que pueden tener el aumento del tamaño de los modelos y de los conjuntos de datos. Este avance no solo mejora las capacidades técnicas de las GANs, sino que también plantea desafíos importantes para la regulación y el uso ético de estas tecnologías. Es fundamental abordar estas cuestiones éticas para asegurar que el desarrollo y la implementación de tecnologías como BigGAN se realicen de manera responsable y con consideración de sus posibles impactos negativos. grandes (Sharma, 2024).

2.1.4. Transfer Learning (Aprendizaje por transferencia)

El aprendizaje por transferencia es una técnica efectiva que ha progresado considerablemente en la inteligencia artificial (IA) generativa. Consiste en reutilizar un modelo entrenado previamente en una tarea para ajustarlo a una nueva tarea relacionada. Esto permite que los modelos se desarrollen con mayor eficiencia, utilizando menos datos y recursos computacionales. Algunas estrategias clave incluyen el entrenamiento de confrontación de dominio, donde una red discriminadora ayuda a ajustar un modelo para generar datos alineados con el dominio objetivo. También se incluye el aprendizaje profesor-alumno, que transfiere conocimientos de un modelo "maestro" más grande a un modelo "estudiante" más pequeño para una implementación eficiente. El desenredo de funciones, que separa elementos como contenido y estilo, permite su manipulación independiente. Además, la transferencia intermodal aprovecha las representaciones aprendidas en diferentes modalidades, como texto e imágenes. (Sharma, 2024).

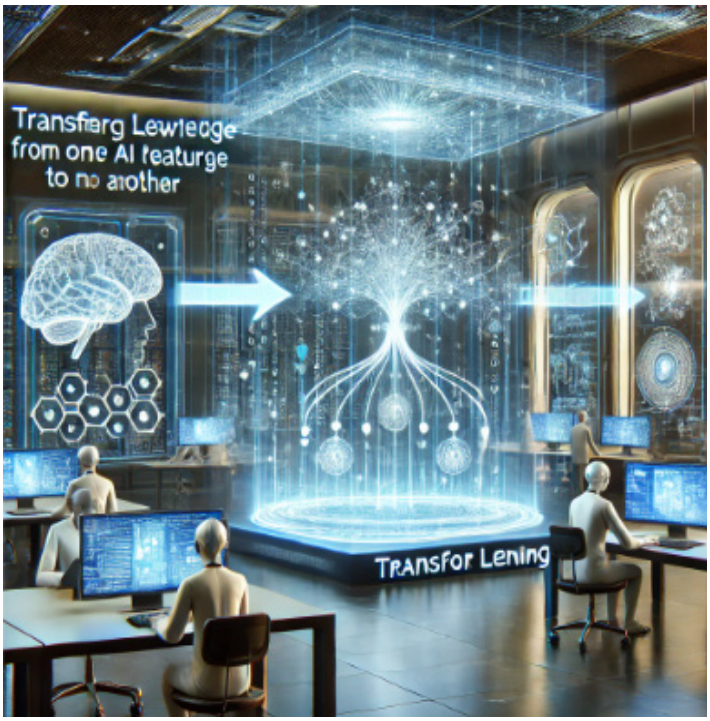

Ilustración 16. Aprendizaje por transferencia. Fuente: Chatgpt4o.

El aprendizaje de cero y con pocas oportunidades permite generar contenido nuevo a partir de un número limitado de ejemplos de entrenamiento. Plataformas como AWS (Amazon Web Services) con SageMaker y JumpStart ofrecen acceso a modelos y herramientas generativas previamente entrenados, lo que facilita el aprendizaje por transferencia y acelera el desarrollo y la implementación de aplicaciones personalizadas. (Bolpur, 2024).

```python
import tensorflow as tf
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.models import Model

# Cargar el modelo preentrenado ResNet50 sin la capa de clasificación superior
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Congelar las capas del modelo base
for layer in base_model.layers:
    layer.trainable = False

# Añadir nuevas capas de clasificación
x = Flatten()(base_model.output)
x = Dense(1024, activation='relu')(x)
x = Dense(10, activation='softmax')(x)  # 10 clases de ejemplo

# Crear el nuevo modelo
model = Model(inputs=base_model.input, outputs=x)

# Compilar el modelo
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Entrenar el modelo con el nuevo conjunto de datos
# Nota: X_train, y_train deben ser los datos específicos de la nueva tarea
model.fit(X_train, y_train, epochs=10, batch_size=32)
```

Ilustración 17. Código Ejemplo (Usando Keras y TensorFlow para Transfer Learning en Visión por Computadora). Fuente: Los Autores.

2.1.5. One shot learning (Aprendizaje de una sola vez)

Ilustración 18. Aprendizaje de una sola vez. Fuente: Chatgpt4o.

El aprendizaje de una sola vez es una técnica poderosa que permite a los modelos generativos de inteligencia artificial (IA) aprender y crear contenido nuevo a partir de un solo ejemplo de entrenamiento. A diferencia de los métodos tradicionales de aprendizaje automático, que requieren grandes volúmenes de datos para ser efectivos, el aprendizaje único se destaca por su eficiencia. En el campo de la IA generativa, esta técnica tiene aplicaciones clave, como en la generación de imágenes, donde permite que modelos como DALL-E 2 produzcan nuevas imágenes basadas en un solo ejemplo de un objeto, escena o estilo particular, y luego


```python
import tensorflow as tf
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.models import Model

# Cargar el modelo preentrenado ResNet50 sin la capa de clasificación superior
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Congelar las capas del modelo base
for layer in base_model.layers:
    layer.trainable = False

# Añadir nuevas capas de clasificación
x = Flatten()(base_model.output)
x = Dense(1024, activation='relu')(x)
x = Dense(10, activation='softmax')(x)  # 10 clases de ejemplo

# Crear el nuevo modelo
model = Model(inputs=base_model.input, outputs=x)

# Compilar el modelo
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Entrenar el modelo con el nuevo conjunto de datos
# Nota: X_train, y_train deben ser los datos específicos de la nueva tarea
model.fit(X_train, y_train, epochs=10, batch_size=32)
```


Ilustración 19. Modelo de una red siames usando phytorch. Fuente: Chatgpt4o.

generalicen este conocimiento para crear variaciones únicas. En la generación de texto, el aprendizaje de una sola vez permite que los modelos de lenguaje generen texto coherente y relevante en contexto, como historias, poemas o código, utilizando solo unos pocos ejemplos del resultado deseado. Asimismo, en la generación de audio o música, los modelos de IA pueden emplear esta técnica para crear nuevas composiciones a partir de un único ejemplo. (Bolpur, 2024).

Los beneficios del aprendizaje de una sola vez en la IA generativa incluyen:

Eficiencia de datos: disminuye la dependencia de grandes conjuntos de datos de entrenamiento, lo que hace que la IA generativa sea más accesible y fácil de implementar.

Adaptación rápida: los modelos generativos pueden ajustarse rápidamente a nuevos dominios o estilos aprendiendo a partir de un número limitado de ejemplos.

Flexibilidad: el aprendizaje de una sola vez permite la creación de contenido diverso y original, superando la simple combinación o interpolación de los datos de entrenamiento.

Técnicas como el aprendizaje por transferencia y el meta-aprendizaje se utilizan comúnmente para habilitar las capacidades de aprendizaje de una sola vez en modelos de IA generativa. Estos enfoques permiten que los modelos identifiquen y utilicen características y patrones relevantes a partir de datos limitados. En general, el aprendizaje único es una capacidad transformadora que impulsa el avance de la IA generativa, facilitando una generación de contenido más eficiente, adaptable y creativa en diversas aplicaciones. (Lee, 2024).

2.1.6. Inteligencia artificial multimodal (Multimodal AI)

Ilustración 20. Multimodal IA. Fuente: Chatgpt4o.

Los aspectos clave de la IA generativa multimodal incluyen:

Capacidad para procesar e integrar diversas fuentes de datos: durante el entrenamiento y la inferencia, la IA multimodal puede combinar texto, imágenes y audio, generando productos que abarcan múltiples modalidades, como descripciones de texto para imágenes o videos creados a partir de indicaciones de texto.

Mejora de la comprensión y generación contextual: al aprovechar las relaciones intermodales, se potencia la creación de contenido, así como el desarrollo de asistentes virtuales e interfaces de usuario multimodales.

```python
import tensorflow as tf
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.models import Model

# Cargar el modelo preentrenado ResNet50 sin la capa de clasificación superior
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Congelar las capas del modelo base
for layer in base_model.layers:
    layer.trainable = False

# Añadir nuevas capas de clasificación
x = Flatten()(base_model.output)
x = Dense(1024, activation='relu')(x)
x = Dense(10, activation='softmax')(x)  # 10 clases de ejemplo

# Crear el nuevo modelo
model = Model(inputs=base_model.input, outputs=x)

# Compilar el modelo
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Entrenar el modelo con el nuevo conjunto de datos
# Nota: X_train, y_train deben ser los datos específicos de la nueva tarea
model.fit(X_train, y_train, epochs=10, batch_size=32)
```

Ilustración 21. Algoritmo de modelo multimodal que combina texto e imágenes para realizar una tarea de clasificación. Fuente: Los autores.

Algunos ejemplos destacados de modelos de IA generativa multimodal son:

GPT-4 de OpenAI, que acepta entradas tanto de texto como de imágenes.

ImageBind de Meta, que integra seis modalidades, incluyendo texto, imágenes, audio y datos de profundidad.
Sora de OpenAI, un modelo diseñado para la generación de texto a video.

La IA generativa multimodal supone un avance notable en comparación con los modelos unimodales anteriores, ya que facilita interacciones entre humanos e IA que son más naturales, inmersivas e inteligentes. A medida que este campo sigue evolucionando, tiene un enorme potencial para revolucionar diversas industrias, desde el entretenimiento hasta la atención médica. (Bakker, 2023).

2.1.7. Alucinaciones de inteligencia artificial (Hallucination AI)

Ilustración 22. La inteligencia artificial de alucinación (Hallucination AI), mostrando un sistema avanzado generando contenido visual vívido e imaginativo que no existe en el mundo real. Fuente: Chatgpt4o.

Las alucinaciones de inteligencia artificial se refieren a situaciones en las que un modelo de IA generativo, como un modelo de lenguaje grande (LLM) que impulsa un chatbot, produce información incorrecta, sesgada o sin sentido, pero la presenta como si fuera verdadera. Estas alucinaciones se deben a limitaciones y sesgos en los datos de entrenamiento y en los algoritmos empleados para desarrollar los modelos de IA. Esto puede llevar a que el modelo haga asociaciones erróneas y genere contenido falso o engañoso. Ejemplos de estas alucinaciones incluyen un chatbot que ofrece información incorrecta, un modelo de reconocimiento de imágenes que identifica objetos de manera equivocada, o una herramienta de generación de contenido que inventa hechos. Las alucinaciones de IA son preocupantes porque pueden facilitar la rápida propagación de desinformación y afectar negativamente los procesos de toma de decisiones. (Diaz, 2024).

```
Entrenamiento
r epoch in range(epochs):
  for i, (imgs, _) in enumerate(dataloader):

      # Adversarial ground truths
      valid = torch.ones(imgs.size(0), 1, requires_grad=False)
      fake = torch.zeros(imgs.size(0), 1, requires_grad=False)

      # Configurar imágenes reales
      real_imgs = imgs

      # -----------------
      #  Entrenar Generador
      # -----------------

      optimizer_G.zero_grad()

      # Generar un batch de imágenes
      z = torch.randn(imgs.shape[0], latent_dim)
      gen_imgs = generator(z)

      # Calcular pérdida del generador
      g_loss = adversarial_loss(discriminator(gen_imgs), valid)

      g_loss.backward()
      optimizer_G.step()

      # ---------------------
      #  Entrenar Discriminador
      # ---------------------
```

Ilustración 23. Algoritmo entrenamiento de alucinación. Fuente: Los autores.

Para prevenir las alucinaciones en la inteligencia artificial, se pueden adoptar varias estrategias: utilizar datos de entrenamiento diversos y representativos para reducir los sesgos, fundamentar el modelo con fuentes de datos relevantes y de alta calidad, ajustar la temperatura del modelo para equilibrar la creatividad con la precisión, e implementar la supervisión humana y la validación de los resultados generados por la IA. (Díaz, 2024).

2.1.8. Grandes modelos de lenguaje (LLM(Largelanguaje models))

Los grandes modelos de lenguaje (LLM) están entrenados con vastas cantidades de datos de texto para desarrollar la capacidad de comprender el lenguaje y

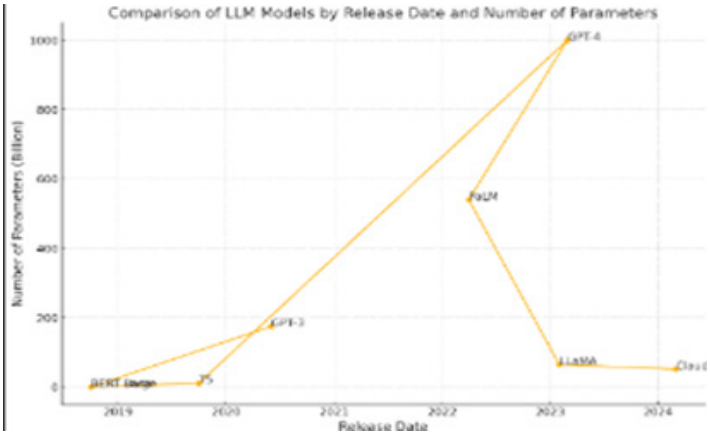

Ilustración 24. Comparando diferentes modelos de LLM en función de sus fechas de lanzamiento y la cantidad de parámetros. Fuente: Los autores.

generar texto que se asemeje al producido por humanos. Ejemplos de estos modelos incluyen GPT, BERT y LLaMA. Los sistemas de inteligencia artificial generativa a menudo utilizan LLM como motor principal para la generación de texto, proporcionando las capacidades esenciales de modelado del lenguaje. Sin embargo, el sistema general de IA generativa integra el LLM con otros componentes. Más allá del texto, la inteligencia artificial generativa puede crear contenido en otras modalidades, como imágenes y audio, combinando LLM con otros modelos y técnicas especializadas. Ejemplos destacados de IA generativa incluyen ChatGPT, DALL-E y Stable Diffusion, todos los cuales utilizan grandes modelos de lenguaje como una tecnología subyacente clave. (Meléndez, 2023).

Sin embargo, los LLM también pueden reflejar sesgos e inexactitudes presentes en los datos con los que fueron entrenados, lo que puede provocar problemas potenciales como alucinaciones en los resultados generados.

2.1.9. Redes generativas adversas (Generative adversial networks GANs).

Las GAN (Generative Adversarial Networks) son un marco destacado en la IA generativa, compuesto por dos redes neuronales: un generador y un discriminador, que compiten entre sí en un juego de suma cero. El generador tiene como objetivo crear datos suficientemente realistas como para engañar al discriminador, mientras que el discriminador se encarga de diferenciar con precisión entre los datos reales y los generados. El generador aprende a transformar ruido aleatorio en muestras de datos realistas, mientras que el discriminador clasifica

Ilustración 25. Dos redes neuronales, el generador y el discriminador, conectadas a través de un flujo de datos y algoritmos, frente a un paisaje cibernético con flujos de datos brillantes y patrones de circuitos intrincados. Fuente: Chatgpt4.

estas muestras como reales o falsas. Las GAN pueden generar contenido diverso, como imágenes, audio y texto, que imita las características estadísticas de los datos de entrenamiento. (Perez, 2024).

```
python                                                    Copiar código

for each training iteration:
    # Train Discriminator
    for each discriminator step:
        Sample real data samples {x^(1), ..., x^(m)} from training data
        Sample noise vectors {z^(1), ..., z^(m)} from prior distribution p_z(z)
        Generate fake data samples: G(z^(i)) for i in {1, ..., m}
        Update discriminator D by ascending its stochastic gradient:
            ∇_θ_D [ (1/m) Σ [ log D(x^(i)) + log(1 - D(G(z^(i)))) ] ]

    # Train Generator
    for each generator step:
        Sample noise vectors {z^(1), ..., z^(m)} from prior distribution p_z(z)
        Generate fake data samples: G(z^(i)) for i in {1, ..., m}
        Update generator G by descending its stochastic gradient:
            ∇_θ_G [ (1/m) Σ log D(G(z^(i))) ]
```

Ilustración 26. Pseudocódigo entrenamiento Gan´s. Fuente: Los autores

El proceso de entrenamiento adversarial permite a las GAN aprender distribuciones de datos complejos de manera no supervisada. Las GAN se han utilizado en tareas como la super resolución de imágenes, la síntesis de texto a imagen y la conversión de voz. No obstante, enfrentan desafíos como el colapso del modo, la inestabilidad durante el entrenamiento y la falta de diversidad en las muestras generadas. Además, plantean preocupaciones sobre el posible uso indebido de los medios generados para engañar. (Perez, 2024).

III.
El aprendizaje profundo (Deep Learning)

El aprendizaje profundo es un subconjunto del aprendizaje automático que emplea redes neuronales artificiales con múltiples capas para aprender y hacer predicciones a partir de datos. El término “profundo” se refiere al gran número de capas en la red, donde cada una transforma ligeramente los datos, permitiendo al modelo aprender representaciones progresivamente más complejas. (Veisi , 2024).

3.1 Representaciones de codificador bidireccional de Transformers (Bidirectional Encoder Representations from Transformers(BERT))

BERT es un poderoso modelo de lenguaje desarrollado por Google que ha logrado avances significativos en el campo del procesamiento del lenguaje natural (PLN). Aunque BERT se utiliza principalmente para comprender y procesar el lenguaje natural, también puede ser aprovechado para tareas de IA generativa. BERT es un modelo previamente entrenado que se puede ajustar para diversas tareas de programación neurolingüística PLN, como la clasificación de texto, la respuesta a preguntas y el reconocimiento de entidades nombradas. Para utilizar BERT en tareas de generación, generalmente se combina con un módulo decodificador, formando una arquitectura codificador-decodificador, similar a modelos como GPT. Su entrenamiento bidireccional permite a BERT capturar información contextual tanto del lado izquierdo como del derecho de un token, lo que es especialmente útil para tareas de generación que requieren una comprensión completa del contexto. (Veisi , 2024).

Ilustración 27. BERT (Representaciones de codificador bidireccional de Transformers) en un entorno futurista de alta tecnología. Fuente: Chatgpt4.

Aunque BERT no es un modelo de IA generativa pura, sus fuertes capacidades para comprender el lenguaje lo hacen un componente valioso en muchos sistemas de IA generativa, especialmente cuando se combina con módulos de generación especializados. BERT se basa en la arquitectura de Transformers, específicamente en los encoders bidireccionales. Esto le permite, a diferencia de los modelos unidireccionales, analizar el contexto completo de una palabra considerando tanto el texto que está a la izquierda como a la derecha de la secuencia.

```python
# Load pre-trained BERT model and tokenizer
model = BertForSequenceClassification.from_pretrained('bert-base-uncased')
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

# Tokenize and encode input text
inputs = tokenizer(text, return_tensors='pt', padding=True, truncation=True)

# Fine-tune BERT model on labeled data
outputs = model(**inputs, labels=labels)
loss = outputs.loss
logits = outputs.logits

# Backpropagation and optimization
loss.backward()
optimizer.step()
```

Ilustración 28. Pseudocódigo simplificado para ajustar BERT en una tarea de clasificación de texto. Fuente: Los autores.

La flexibilidad de la arquitectura Transformer permite que BERT se adapte tanto para tareas de comprensión como de generación.

3.2. Las redes neuronales convolucionales (Convolutional neural networks (CNN)

Las redes neuronales convolucionales (CNN) son un modelo de aprendizaje profundo especialmente adecuado para procesar y entender imágenes visuales. Son un tipo especializado de red neuronal diseñada para trabajar con datos en formato de cuadrícula, como las imágenes. Las CNN se basan en tres ideas arquitectónicas clave: conexiones locales, pesos compartidos y agrupación. La capa convolucional es el componente central de una CNN. Las aplicaciones de las CNN incluyen:

Reconocimiento de Imágenes: Clasificación de objetos en imágenes.

Detección de Objetos: Identificación y localización de objetos en una imagen.

Segmentación de Imágenes: División de una imagen en regiones de interés.

Reconocimiento de Rostros: Identificación de individuos en imágenes o videos.

Análisis de Imágenes Médicas: Detección de enfermedades en imágenes médicas, como radiografías y resonancias magnéticas.

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
from torchvision import datasets, transforms

# Define CNN architecture
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=6, kernel_size=5)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.conv2 = nn.Conv2d(in_channels=6, out_channels=16, kernel_size=5)
        self.fc1 = nn.Linear(16 * 4 * 4, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 4 * 4)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

Ilustración 29. Algoritmo de redes neuronales convencionales. Fuente: Los autores.

Las redes neuronales convolucionales (CNN) aplican un conjunto de filtros (o núcleos) que se pueden aprender a la imagen de entrada, lo que permite a la red extraer automáticamente funciones directamente de los datos. Estas redes tienen una estructura jerárquica, donde cada capa extrae características de nivel superior a partir de las características capturadas en la capa anterior. Las primeras capas detectan elementos de bajo nivel, como bordes y formas, mientras que las capas posteriores reconocen patrones más complejos. Los componentes clave de una CNN incluyen:

Capas convolucionales: encargadas de la extracción de características.

Capas de agrupación: reducen la dimensionalidad de las características extraídas.

Capas completamente conectadas: realizan tareas como clasificación, detección de objetos y segmentación semántica.

En comparación con los enfoques tradicionales de aprendizaje automático, las CNN pueden aprender automáticamente características relevantes a partir de datos de imágenes sin procesar, eliminando la necesidad de ingeniería de características manuales. (Zhu, 2022).

```
# Initialize model, loss function, and optimizer
model = CNN()
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)

# Load data
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5,), (0.5,))])
trainset = datasets.MNIST(root='./data', train=True, download=True, transform=transform)
trainloader = DataLoader(trainset, batch_size=32, shuffle=True)

# Training loop
for epoch in range(10):
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        inputs, labels = data

        optimizer.zero_grad()

        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        if i % 100 == 99:
            print(f'Epoch {epoch + 1}, Batch {i + 1}, Loss: {running_loss / 100:.3f}')
            running_loss = 0.0

print('Finished Training')
```

Ilustración 30. Pseudocódigo de entrenamiento del modelo de redes neuronales convencionales. Fuente: Los autores.

Ilustración 31. Entrada que se procesa a través de múltiples capas de filtros convolucionales, funciones de activación y capas de agrupación, y cada capa extrae diferentes características como bordes, texturas y patrones, en un contexto de alta tecnología. Fuente: Chatgpt4o

La estructura jerárquica y el uso compartido de parámetros en las CNN las hacen computacionalmente eficientes y les permiten generalizar bien a nuevos datos. Además, son un componente central en muchas aplicaciones de aprendizaje profundo, como vehículos autónomos, reconocimiento facial, análisis de imágenes médicas y otras áreas clave.

3.3. Redes neuronales recurrentes (Recurrent Neural Networks (RNN))

Son un tipo de modelo de aprendizaje profundo diseñado específicamente para manejar datos secuenciales, como texto, voz y series de tiempo.

A diferencia de las redes neuronales tradicionales, las redes neuronales recurrentes (RNN) tienen la capacidad de mantener un estado interno, o "memoria", que les permite procesar entradas secuenciales y utilizar información previa para influir en la salida actual. Su arquitectura única, que incluye un bucle que permite la persistencia de la información, les permite modelar comportamientos temporales dinámicos y capturar dependencias entre elementos en una secuencia. Esto es crucial para tareas como el modelado de lenguaje, la traducción automática y el reconocimiento de voz. Las RNN utilizan el mismo conjunto de pesos en cada paso de la secuencia, lo que las hace eficientes en términos de parámetros y capaces de generalizar bien. Sin embargo, las RNN estándar pueden enfrentar el problema del gradiente que desaparece o explota, lo que complica su entrenamiento en secuencias largas. (Ma, 2024).

Ilustración 32. Red Neuronal Recurrente (RNN) en un entorno de investigación futurista de alta tecnología. Fuente: Chatgpt4o

```python
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader

# Define RNN architecture
class RNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(RNN, self).__init__()
        self.hidden_size = hidden_size
        self.rnn = nn.RNN(input_size, hidden_size, batch_first=True)
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        h0 = torch.zeros(1, x.size(0), self.hidden_size)
        out, _ = self.rnn(x, h0)
        out = self.fc(out[:, -1, :])
        return out

# Initialize model, loss function, and optimizer
input_size = 10
hidden_size = 20
output_size = 1
model = RNN(input_size, hidden_size, output_size)
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Load data
# Assuming we have a DataLoader 'trainloader' that provides input sequences and targets
trainloader = DataLoader(dataset, batch_size=32, shuffle=True)
```

Ilustración 33. Pseudocódigo de entrenamiento de RNN. Fuente: Los autores.

Para abordar el problema del gradiente que desaparece o explota en las RNN estándar, se desarrollaron variantes como las Memorias a Largo Plazo (LSTM) y las Unidades Recurrentes Cerradas (GRU), que introducen mecanismos de activación para controlar mejor el flujo de información. Las RNN bidireccionales (BiRNN) también ofrecen mejoras al procesar secuencias en ambas direcciones, hacia adelante y hacia atrás, permitiendo aprovechar tanto el contexto pasado como el futuro. Estas RNN se combinan con otras arquitecturas de aprendizaje profundo, como las redes neuronales convolucionales (CNN), para crear modelos híbridos de gran potencia. En el aprendizaje profundo, las RNN y sus variantes se aplican en tareas como modelado de lenguaje, traducción automática, reconocimiento de voz, subtitulado de imágenes y pronóstico de series temporales. (Ma, 2024).

3.4. Épocas (Epoch)

En el aprendizaje profundo, una *época* se refiere a un ciclo completo en el que todo el conjunto de datos de entrenamiento se pasa a través del modelo una vez. Durante una época, se utiliza todo el conjunto de datos de entrenamiento para actualizar los parámetros internos del modelo, lo que contribuye al proceso de aprendizaje.

```
# Initialize the model, loss function, and optimizer
model = SimpleNN()
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.01)

# Load the training data
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5,), (0.5,
trainset = datasets.MNIST(root='./data', train=True, download=True, transform=transform)
trainloader = DataLoader(trainset, batch_size=32, shuffle=True)

# Training loop
epochs = 10
for epoch in range(epochs):
    running_loss = 0.0
    for inputs, labels in trainloader:
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()

    print(f'Epoch {epoch + 1}/{epochs}, Loss: {running_loss / len(trainloader):.4f}')

print('Finished Training')
```

Ilustración 34. Pseudocódigo de entrenamiento de época (epoch). Fuente: Los autores.

Durante una época, el modelo procesa todas las muestras de entrenamiento, calcula la pérdida y ajusta los pesos del modelo en consecuencia. El número de épocas es un hiperparámetro que determina cuántas veces el modelo recorrerá el conjunto de datos de entrenamiento completo durante el proceso de aprendizaje.

Épocas vs Iteraciones:

Época: Una época se refiere a un ciclo completo a través de todo el conjunto de datos de entrenamiento.

Iteración: Una iteración se refiere al procesamiento de un único lote de muestras de entrenamiento. El número de iteraciones por época depende del tamaño del lote: a mayor tamaño del lote, menor será el número de iteraciones por época. Por ejemplo, si se tiene un conjunto de datos con 1000 muestras y un tamaño de lote de 100, habrá 10 iteraciones en cada época. (Ma, 2024).

Importancia de las épocas:

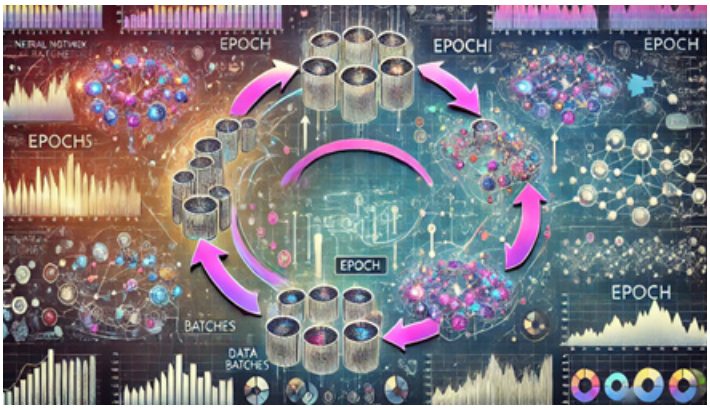

Ilustración 35. Épocas (epoch) con distintos bucles. Fuente: Chatgpt4o

Aumentar el número de épocas permite al modelo captar patrones más complejos en los datos. No obstante, usar demasiadas épocas puede causar sobreajuste, por lo que es esencial ajustar este hiperparámetro adecuadamente. En esencia, una época en el aprendizaje profundo corresponde a un recorrido completo por todo el conjunto de datos de entrenamiento a través del modelo, lo cual es vital para que el modelo mejore su rendimiento. Las épocas son un hiperparámetro crucial que debe ser optimizado, permite que la red neuronal aprenda y ajuste sus parámetros. Un mayor número de épocas puede conducir a una mejor convergencia y mayor precisión del modelo. Sin embargo, entrenar el modelo durante un exceso de épocas puede resultar en sobreajuste, donde el modelo se adapta demasiado a los datos de entrenamiento y tiene problemas para generalizar a datos nuevos. Las épocas definen el ciclo de aprendizaje del modelo y su capacidad para mejorar en función de los datos de entrenamiento. Elegir el número adecuado de épocas es fundamental para equilibrar el aprendizaje y prevenir el sobreajuste. (Ma, 2024).

3.5. GPT (Transformador generativo pre entrenado)

Es una familia de grandes modelos de lenguaje desarrollados por Open AI que se han convertido en un avance significativo en el aprendizaje profundo y la inteligencia artificial.

Ilustración 36. Arquitectura Gpt. Fuente: Chatgpt4o.

Los modelos GPT son arquitecturas de redes neuronales basadas en transformadores que sobresalen en tareas de generación y procesamiento del lenguaje natural. El primer modelo, GPT-1, fue lanzado en 2018, seguido por GPT-2 en 2019, GPT-3 en 2020 y GPT-4 en 2024. Estos modelos se entrenan con grandes volúmenes de datos textuales para aprender patrones y generar texto que imite el lenguaje humano. Utilizan una arquitectura de transformadores, que emplea un mecanismo de atención para valorar diferentes partes de la entrada, permitiendo captar dependencias a largo plazo en el texto. A diferencia de las redes neuronales recurrentes anteriores, los modelos GPT se entrenan de manera no supervisada con extensos corpus textuales, desarrollando una comprensión general del lenguaje, y luego pueden ser ajustados para tareas específicas con conjuntos de datos etiquetados más pequeños. (Gutierrez, 2023).

Los modelos GPT son capaces de llevar a cabo diversas tareas relacionadas con el procesamiento del lenguaje natural, incluyendo la generación de texto, la respuesta a preguntas, la creación

```python
import torch
import torch.nn as nn
import torch.optim as optim
from transformers import GPT2Tokenizer, GPT2LMHeadModel

# Load pre-trained GPT model and tokenizer
model = GPT2LMHeadModel.from_pretrained('gpt2')
tokenizer = GPT2Tokenizer.from_pretrained('gpt2')

# Define training parameters
learning_rate = 5e-5
epochs = 3
optimizer = optim.AdamW(model.parameters(), lr=learning_rate)

# Sample training data
train_data = ["Example sentence 1.", "Example sentence 2."]

# Training loop for pre-training
for epoch in range(epochs):
    for sentence in train_data:
        inputs = tokenizer(sentence, return_tensors='pt')
        outputs = model(**inputs, labels=inputs['input_ids'])
        loss = outputs.loss

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    print(f'Epoch {epoch + 1}/{epochs}, Loss: {loss.item()}')

print('Finished Pre-training')
```

Ilustración 37. Modelo de pre entrenamiento con gpt. Fuente: Los autores.

de resúmenes y la traducción. Estos modelos se basan en la arquitectura de Transformers, que utiliza mecanismos de autoatención para manejar secuencias de datos, permitiendo que el modelo tenga en cuenta el contexto completo de cada palabra dentro de una oración y, así, mejorar la coherencia y precisión. Inicialmente, los modelos GPT se entrenan con grandes volúmenes de texto sin etiquetar, durante los cuales aprenden patrones lingüísticos, gramática y conocimientos generales. (Gutierrez, 2023).

IV. Redes Neuronales (Neural Networks)

Las redes neuronales son una técnica de aprendizaje automático que se inspira en la estructura y funcionamiento del cerebro humano. Estas redes consisten en nodos interconectados, o neuronas artificiales, que aprenden a realizar tareas al procesar datos y reconocer patrones. (Wurzberger, 2023).

4.1. Las redes neuronales de función de base radial (Radial Basis Function Networks (RBFN).

Las redes neuronales de función de base radial (RBF) son un tipo de red neuronal artificial que utiliza funciones de base radial como funciones de activación.

Las redes neuronales se emplean frecuentemente en tareas como aproximación de funciones, clasificación, predicción de series temporales y control de sistemas.

Ilustración 38. Arquitectura de una RBFN. Fuente: Chatgpt4o

A continuación, se describen las características clave de las redes neuronales de base radial (RBF):

Arquitectura:

- Compuestas por tres capas: capa de entrada, capa oculta y capa de salida.
- La capa oculta aplica funciones de base radial, generalmente gaussianas, a las entradas.
- La capa de salida combina linealmente las salidas de la capa oculta.

Funcionamiento:

- Las redes RBF ubican una o más neuronas RBF en el espacio definido por las variables predictoras. Se calcula la distancia euclidiana entre el punto evaluado y el centro de cada neurona. A esta distancia se le aplica una función de base radial (como la gaussiana) para determinar el peso o influencia de cada neurona. La salida se obtiene combinando linealmente las salidas de las funciones RBF, multiplicadas por pesos.

Capacitación:

- El entrenamiento generalmente se realiza en dos fases:
  1. Determinar los centros y anchos de las funciones RBF (por ejemplo, mediante agrupación K-medias).
  2. Determinar los pesos que conectan la capa oculta con la de salida (por ejemplo, usando regresión lineal).
- Las redes RBF suelen tener un entrenamiento más rápido en comparación con otras redes neuronales como los perceptrones multicapa (MLP).

```python
import numpy as np
from sklearn.cluster import KMeans

class RBFN:
    def __init__(self, num_centers, sigma=1.0):
        self.num_centers = num_centers
        self.sigma = sigma
        self.centers = None
        self.weights = None

    def _rbf(self, x, c):
        return np.exp(-np.linalg.norm(x - c) ** 2 / (2 * self.sigma ** 2))

    def fit(self, X, y):
        # Step 1: Initialize centers using k-means clustering
        kmeans = KMeans(n_clusters=self.num_centers)
        kmeans.fit(X)
        self.centers = kmeans.cluster_centers_

        # Step 2: Compute the hidden layer output matrix
        hidden_output = np.zeros((X.shape[0], self.num_centers))
        for i in range(X.shape[0]):
            for j in range(self.num_centers):
                hidden_output[i, j] = self._rbf(X[i], self.centers[j])

        # Step 3: Compute the weights using least squares
        self.weights = np.linalg.pinv(hidden_output).dot(y)

    def predict(self, X):
        hidden_output = np.zeros((X.shape[0], self.num_centers))
        for i in range(X.shape[0]):
            for j in range(self.num_centers):
                hidden_output[i, j] = self._rbf(X[i], self.centers[j])
        return hidden_output.dot(self.weights)
```

Ilustración 39. Pseudocódigo de pre entrenamiento RFBN. Fuente: Los autores

Ventajas:

- Diseño sencillo y buena capacidad de generalización.
- Alta tolerancia al ruido en los datos de entrada.
- Requiere solo una capa oculta.
- Entrenamiento más rápido en comparación con otras redes neuronales.

Las redes neuronales RBF son una clase potente y versátil de redes neuronales que sobresalen en tareas de aproximación, clasificación y predicción de funciones. Su arquitectura distintiva y proceso de entrenamiento las hacen una opción atractiva para diversas aplicaciones en aprendizaje automático. (Wurzberger, 2023).

4.2. Redes de Hopfield (Hopfield Networks)

Las redes de Hopfield son un tipo de red neuronal recurrente empleadas para tareas de optimización y memoria asociativa.

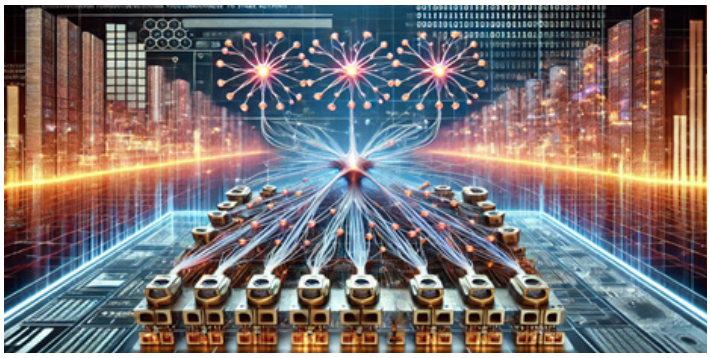

Ilustración 40. Redes Hopfield. Fuente: Chatgpt4o

Las redes de Hopfield son redes neuronales recurrentes con una sola capa de neuronas completamente conectadas, donde cada neurona tiene salidas tanto inversoras como no inversoras. Las neuronas están conectadas entre sí, pero no a sí mismas, y los pesos de las conexiones son simétricos (wij = wji), sin autoconexiones (wii = 0).

Capacitación:

- Los pesos se actualizan según los patrones de entrada usando ecuaciones específicas.

- Para patrones binarios, los pesos se inicializan para almacenar los patrones deseados. La red se ejecuta iterativamente hasta alcanzar un estado estable, o atractor.

Tipos:

- Las redes de Hopfield pueden ser discretas o continuas. Las redes discretas tienen salidas binarias o bipolares, y ambas versiones se utilizan en tareas de memoria asociativa y optimización.

Aplicaciones:

- Autoasociación y finalización de patrones: Permiten recuperar patrones completos a partir de fragmentos o versiones ruidosas.

- Problemas de optimización: Como el problema del viajante.

- Restauración de imágenes e identificación de sistemas.

Las redes de Hopfield son efectivas para la memoria asociativa y la optimización debido a su capacidad para converger a estados estables que representan patrones almacenados o soluciones óptimas. Las variantes discretas y continuas permiten manejar diversos tipos de datos y problemas, haciéndolas útiles para aplicaciones que requieren la recuperación de información completa a partir de datos parciales. (Nicolini, 2024).

```python
import numpy as np

class HopfieldNetwork:
    def __init__(self, num_neurons):
        self.num_neurons = num_neurons
        self.weights = np.zeros((num_neurons, num_neurons))

    def train(self, patterns):
        num_patterns = len(patterns)
        for pattern in patterns:
            self.weights += np.outer(pattern, pattern)
        self.weights /= num_patterns
        np.fill_diagonal(self.weights, 0)

    def recall(self, pattern, steps=10):
        state = pattern.copy()
        for _ in range(steps):
            for i in range(self.num_neurons):
                h_i = np.dot(self.weights[i], state)
                state[i] = 1 if h_i >= 0 else -1
        return state

# Example usage
patterns = np.array([[1, -1, 1, -1], [-1, 1, -1, 1]])
hn = HopfieldNetwork(num_neurons=4)
hn.train(patterns)

test_pattern = np.array([1, -1, 1, 1])  # Noisy version of the first pattern
recalled_pattern = hn.recall(test_pattern)
```

Ilustración 41. Pseudocódigo de pre entrenamiento redes hopfield. Fuente: Los autores.

4.3. Perceptrón multicapa (Multiplayer Perceptrons)

Un perceptrón multicapa (MLP) es una red neuronal artificial compuesta por varias capas de neuronas interconectadas, lo que le permite capturar relaciones complejas en los datos. Su estructura incluye:

- Capa de entrada: Esta capa recibe los datos iniciales. El número de neuronas en esta capa coincide con la cantidad de características de los datos de entrada.

- Capas ocultas: Los MLP pueden tener una o más capas ocultas. En estas capas, las neuronas procesan las entradas de la capa anterior y transmiten sus salidas a la siguiente capa. La cantidad de neuronas y capas ocultas puede ajustarse según la complejidad de la tarea.

- Capa de salida: La capa final genera la salida de la red. Su configuración depende de la tarea específica, como la clasificación binaria.

En un MLP, cada neurona en una capa está conectada a todas las neuronas en la capa siguiente, y estas conexiones están ponderadas para determinar el grado de influencia de una neurona sobre otra.

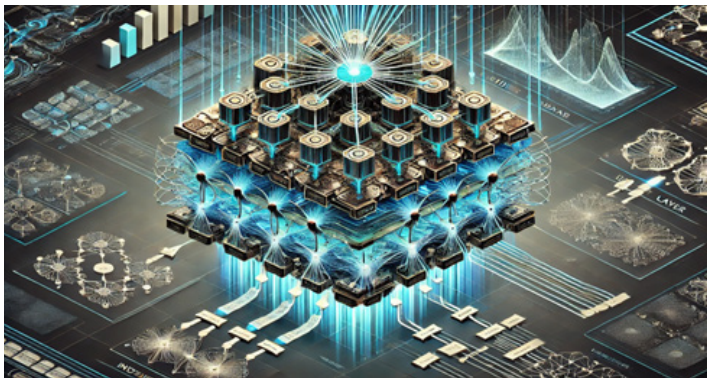

Ilustración 42. Arquitectura de un perceptrón multicapa MLP. Fuente: Chatgpt4o

Los MLP se entrenan mediante el algoritmo de retropropagación, que ajusta los pesos de las conexiones según el error entre la salida de la red y el resultado esperado. Este proceso incluye:- Cálculo de errores: Se determina el error en cada nodo de salida y se calcula el error total en todos los nodos de salida.- Ajuste de peso: Utilizando el descenso de gradiente, los pesos se actualizan para reducir el error. El cambio en cada peso es proporcional al error y a la entrada correspondiente.- Funciones de activación no lineales: Funciones como sigmoide o ReLU permiten a los MLP aprender relaciones no lineales en los datos, lo que les capacita para resolver problemas que no son separables linealmente.

Aplicaciones:

- Reconocimiento de imágenes: Los MLP pueden clasificar y reconocer patrones en imágenes.

- Procesamiento del lenguaje natural: Se utilizan para tareas como análisis de sentimientos y traducción de idiomas.

- Reconocimiento de voz: Ayudan a convertir el lenguaje hablado en texto. (Tong, 2024).

```python
import numpy as np

# Activation functions and their derivatives
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def sigmoid_derivative(x):
    return x * (1 - x)

# Initialize weights and biases
def initialize_parameters(layer_sizes):
    parameters = {}
    for l in range(1, len(layer_sizes)):
        parameters['W' + str(l)] = np.random.randn(layer_sizes[l], layer_sizes[l-1]) * 0.
        parameters['b' + str(l)] = np.zeros((layer_sizes[l], 1))
    return parameters

# Forward propagation
def forward_propagation(X, parameters):
    cache = {'A0': X}
    L = len(parameters) // 2
    for l in range(1, L + 1):
        Z = np.dot(parameters['W' + str(l)], cache['A' + str(l-1)]) + parameters['b' + str
        A = sigmoid(Z)
        cache['Z' + str(l)] = Z
        cache['A' + str(l)] = A
    return A, cache
```

Ilustración 43. Pseudocódigo de entrenamiento MLP. Fuente: Los autores

4.4. Redes Neuronales Modulares (Modular Neural Networks)

Las redes neuronales modulares (MNN) son una arquitectura avanzada en redes neuronales artificiales que emplea múltiples redes neuronales funcionando como módulos interconectados para resolver problemas complejos. Este enfoque facilita la descomposición de un problema en partes más pequeñas y manejables, con cada módulo especializado en un aspecto específico de la tarea general. Un integrador coordina estos módulos y combina sus resultados para producir el resultado final. (Tong, 2024).

Ilustración 44. Módulos MNN. Fuente: Chatgpt4o.

Características de las redes neuronales modulares (MNN):

- Modularidad: Cada módulo en una red neuronal modular opera de manera semi independiente, lo que permite el procesamiento en paralelo. Esta estructura está inspirada en las redes neuronales biológicas, que presentan regiones especializadas para diferentes funciones.
- Aprendizaje conjunto: El enfoque modular se basa en el principio de aprendizaje conjunto, donde una colección de modelos más simples puede superar a un modelo único más complejo. Esto es especialmente útil en escenarios donde diversos módulos pueden especializarse en diferentes aspectos del espacio de entrada.
- Eficiencia y robustez: Las MNN pueden mejorar la eficiencia computacional al reducir la cantidad de conexiones y pesos en comparación con redes monolíticas. Esta reducción resulta en tiempos de entrenamiento más rápidos y una mayor tolerancia a fallas, ya que la falla de un módulo no compromete necesariamente todo el sistema.

Aplicaciones:

- Fusión de datos y promedio de predicción: Las MNN pueden integrar información de múltiples fuentes de manera efectiva, mejorando la precisión de las predicciones.
- Sistemas adaptativos: Son adecuadas para entornos donde las tareas evolucionan, permitiendo que los módulos se entrenen y adapten de manera independiente, facilitando así el aprendizaje y la adaptación continua. (Leoshchenko, 2024) .

```
# Initialize modules
feature_extractor = FeatureExtractor()
object_detector = ObjectDetector(input_dim=64*6*6, output_dim=10)
classifier = Classifier(input_dim=10, num_classes=10)

# Initialize the integrated modular network
model = IntegrationModule(feature_extractor, object_detector, classifier)

# Define loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Load data
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5,), (0.5,)
trainset = datasets.MNIST(root='./data', train=True, download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=32, shuffle=True)

# Training loop
epochs = 10
for epoch in range(epochs):
    running_loss = 0.0
    for inputs, labels in trainloader:
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()

    print(f'Epoch {epoch + 1}/{epochs}, Loss: {running_loss / len(trainloader):.4f}')

print('Finished Training')
```

Ilustración 45. Pseudocódigo de pre entrenamiento modular neural networks. Fuente: Los autores.

A pesar de sus ventajas, las redes neuronales modulares (MNN) enfrentan desafíos, especialmente en la integración de los resultados de varios módulos. Es fundamental asegurar que los módulos funcionen de manera coherente y que comuniquen eficazmente sus resultados para lograr el éxito de la red. En resumen, las MNN representan un avance significativo en el diseño de redes neuronales, proporcionando flexibilidad, eficiencia y robustez para abordar problemas complejos mediante un enfoque colaborativo entre módulos especializados. (Leoshchenko, 2024).

4.5. Máquinas de Boltzman (Boltzmann Machines)

Una máquina de Boltzmann es una red neuronal estocástica que está compuesta por unidades conectadas de manera simétrica, y cada unidad puede estar en estado “encendido” o “apagado”. Esta arquitectura permite a la máquina de Boltzmann modelar distribuciones complejas sobre datos binarios, haciéndola una herramienta poderosa para el aprendizaje no supervisado. (Leoshchenko, 2024) .

Ilustración 46. Boltzman machines. Fuente: Chatgpt4o

Estructura de las Máquinas de Boltzmann:

- Nodos visibles: Representan los datos de entrada y son observables.
- Nodos ocultos: No se observan directamente, pero capturan los patrones subyacentes en los datos.
- Naturaleza estocástica: Las decisiones de los nodos son probabilísticas, lo que permite a la red explorar diversas configuraciones de estados. Este comportamiento estocástico es clave para aprender la distribución de probabilidad de los datos de entrada.
- Modelo basado en energía: La máquina de Boltzmann funciona minimizando una función de energía asociada con los estados de la red. Los estados de menor energía son más probables.

Mecanismo de aprendizaje: El proceso de aprendizaje en las máquinas de Boltzmann incluye dos fases:

- Fase positiva: Las unidades visibles se configuran en un estado binario específico según los datos de entrenamiento.
- Fase negativa: La red opera libremente, actualizando los estados de las unidades ocultas en función de la configuración actual.

El objetivo es ajustar los pesos para que la distribución de los estados visibles en la fase positiva coincida con la distribución de los estados visibles generados en la fase negativa.

Variantes:

- Máquinas de Boltzmann Restringidas (RBM): En las RBM, las unidades ocultas no están conectadas entre sí, lo que simplifica el proceso de capacitación y permite un aprendizaje eficiente de las características a partir de los datos.

```python
import numpy as np

# Sigmoid function
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# Boltzmann Machine class
class BoltzmannMachine:
    def __init__(self, num_visible, num_hidden, learning_rate=0.1):
        self.num_visible = num_visible
        self.num_hidden = num_hidden
        self.learning_rate = learning_rate

        # Initialize weights and biases
        self.weights = np.random.randn(num_visible, num_hidden) * 0.01
        self.visible_bias = np.zeros(num_visible)
        self.hidden_bias = np.zeros(num_hidden)

    def train(self, data, epochs=1000):
        for epoch in range(epochs):
            for sample in data:
                # Positive phase
                pos_hidden_probs = sigmoid(np.dot(sample, self.weights) + self.hidden_bias
                pos_hidden_states = (pos_hidden_probs > np.random.rand(self.num_hidden)).a

                # Negative phase
                neg_visible_probs = sigmoid(np.dot(pos_hidden_states, self.weights.T) + se
                neg_visible_states = (neg_visible_probs > np.random.rand(self.num_visible)
                neg_hidden_probs = sigmoid(np.dot(neg_visible_states, self.weights) + self
```

Ilustración 47. Pseudocódigo de pre entrenamiento con la máquina de Boltzmann. Fuente: Los autores.

- Redes de Creencias Profundas (DBN): Compuestas por múltiples capas de RBM apiladas, donde la salida de una RBM sirve como entrada para la siguiente. Esta estructura jerárquica permite el modelado de distribuciones de datos complejas.
- Máquinas Deep Boltzmann (DBM): Similares a las DBN, pero con conexiones no dirigidas entre capas, lo que permite interacciones más complejas entre ellas.
- Aprendizaje de funciones: Pueden aprender funciones complejas a partir de conjuntos de datos binarios, siendo útiles en tareas como la reducción de dimensionalidad.
- Modelado generativo: Son capaces de generar nuevos puntos de datos que se asemejan a los datos de entrenamiento, lo cual es beneficioso para la generación de imágenes y el filtrado colaborativo.
- Problemas de optimización: Su naturaleza estocástica permite tomar muestras de distribuciones complejas, lo que las hace aplicables en diversas tareas de optimización. (Leoshchenko, 2024) .

4.6. Mapas Auto organizados (Self Organizing Maps Autoencoders)

Los mapas auto organizados (SOM) y los codificadores automáticos son dos tipos distintos de arquitecturas de redes neuronales que se pueden combinar para aprovechar sus puntos fuertes en el aprendizaje no supervisado.

Ilustración 48. Arquitectura de un SOM. Fuente: Chatgpt4o

Modelos Híbridos que combinan SOM y Codificadores Automáticos:

- Mapa Autoorganizado de Codificador Automático Híbrido: Este modelo utiliza un codificador automático para aprender representaciones comprimidas de los datos y luego emplea un SOM para organizar estas representaciones topológicamente. Este enfoque mejora el rendimiento en tareas como la clasificación y compresión de imágenes, especialmente en conjuntos de datos complejos como CIFAR-10.
- Mapa Autoorganizado del Codificador Automático de Eliminación de Ruido (DASOM): Este modelo integra codificadores automáticos diseñados para eliminar ruido con SOM, permitiendo un aprendizaje no supervisado más efectivo. El codificador automático de eliminación de ruido mejora la robustez de la extracción de características al limpiar los datos de entrada antes de que sean procesados por el SOM, lo que resulta en una mejor calidad de las representaciones aprendidas.

```python
import numpy as np

class SOM:
    def __init__(self, grid_size, input_dim, learning_rate=0.1, radius=1.0, decay=0.99):
        self.grid_size = grid_size
        self.input_dim = input_dim
        self.learning_rate = learning_rate
        self.radius = radius
        self.decay = decay
        self.weights = np.random.rand(grid_size[0], grid_size[1], input_dim)

    def train(self, data, epochs):
        for epoch in range(epochs):
            for sample in data:
                bmu_idx = self.find_bmu(sample)
                self.update_weights(sample, bmu_idx, epoch, epochs)

    def find_bmu(self, sample):
        distances = np.linalg.norm(self.weights - sample, axis=2)
        return np.unravel_index(np.argmin(distances), self.weights.shape[:2])

    def update_weights(self, sample, bmu_idx, epoch, epochs):
        learning_rate = self.learning_rate * (1 - epoch / epochs)
        radius = self.radius * (1 - epoch / epochs)

        for x in range(self.grid_size[0]):
            for y in range(self.grid_size[1]):
                dist_to_bmu = np.linalg.norm(np.array([x, y]) - np.array(bmu_idx))
                if dist_to_bmu <= radius:
                    influence = np.exp(-dist_to_bmu / (2 * (radius**2)))
                    self.weights[x, y] += influence * learning_rate * (sample - self.weigh
```

Ilustración 47. Pseudocódigo de pre entrenamiento con la máquina de Boltzmann. Fuente: Los autores.

La integración de Mapas Autoorganizados (SOM) y Codificadores Automáticos puede llevar a:

- Aprendizaje de Representación Mejorado: Combinar la preservación topológica de los SOM con las capacidades de compresión de los codificadores automáticos permite aprender representaciones más significativas de datos complejos.
- Agrupación Mejorada: La organización topológica proporcionada por los SOM mejora la agrupación de las características aprendidas, lo que es beneficioso para tareas como la segmentación y clasificación de imágenes.
- Robustez Frente al Ruido: Modelos híbridos como DASOM, que integran codificadores automáticos de eliminación de ruido con SOM, aumentan el rendimiento en aplicaciones del mundo real, especialmente en escenarios donde los datos pueden ser ruidosos o incompletos. (Kollasch, 2024).

4.7. La teoría de la resonancia adaptativa (Adaptive Resonance Theory)

La teoría de la Resonancia Adaptativa (ART), desarrollada por Stephen Grossberg y Gail Carpenter,

es una teoría cognitiva y neuronal que explica cómo el cerebro aprende a categorizar, reconocer y predecir objetos y eventos en un entorno dinámico. Esta teoría aborda el dilema estabilidad-plasticidad, que es el desafío de integrar nueva información sin olvidar los conocimientos previamente adquiridos.

Características de las redes ART:

- Estabilidad-Plasticidad: Las redes ART pueden incorporar nueva información sin olvidar las categorías aprendidas anteriormente, manteniendo así una estabilidad en el aprendizaje a lo largo del tiempo.
- Estrategia de Coincidencia (Matching): Utilizan una estrategia de coincidencia para clasificar y reconocer patrones. Esto permite que los modelos ART se adapten a datos dinámicos y variables.
- Parámetro de Vigilancia: Controla la precisión con la que se deben reconocer los patrones. Un valor alto del parámetro de vigilancia permite un reconocimiento más preciso, pero puede resultar en un mayor número de categorías (clases), mientras que un valor bajo puede llevar a una menor precisión y menos categorías.

Esta teoría proporciona un marco para entender cómo los sistemas cognitivos y neuronales pueden equilibrar la necesidad de aprender nuevas categorías mientras preservan el conocimiento previamente adquirido. (Petrenko, 2023).

Ilustración 50. Arquitectura de una red ART. Fuente: Chatgpt4o

```python
import numpy as np

class ART1:
    def __init__(self, num_features, num_categories, vigilance=0.75):
        self.num_features = num_features
        self.num_categories = num_categories
        self.vigilance = vigilance
        self.weights = np.random.rand(num_categories, num_features)
        self.weights = self.weights / np.linalg.norm(self.weights, axis=1, keepdims=True)

    def train(self, data):
        for pattern in data:
            pattern = pattern / np.linalg.norm(pattern)  # Normalize input pattern
            match_found = False

            for category in range(self.num_categories):
                similarity = np.dot(self.weights[category], pattern)
                if similarity >= self.vigilance:
                    match_found = True
                    self.update_weights(category, pattern)
                    break

            if not match_found:
                self.create_new_category(pattern)

    def update_weights(self, category, pattern):
        self.weights[category] = pattern

    def create_new_category(self, pattern):
        for category in range(self.num_categories):
            if np.linalg.norm(self.weights[category]) == 0:
                self.weights[category] = pattern
                break

# Example usage
data = np.array([[1, 0, 1, 0], [0, 1, 0, 1], [1, 1, 0, 0], [0, 0, 1, 1]])  # Example bina
art = ART1(num_features=4, num_categories=4, vigilance=0.75)
art.train(data)
print("Weights after training:", art.weights)
```

Ilustración 51. Pseudocódigo modelo art 1. Fuente: Los autores.

Características clave de la teoría de Resonancia Adaptativa (ART):

- Procesamiento de Arriba Hacia Abajo y de Abajo Hacia Arriba: ART propone que el reconocimiento de objetos se produce a través de la interacción entre expectativas de arriba hacia abajo (plantillas de memoria) y entradas sensoriales de abajo hacia arriba. El modelo compara estas señales para determinar la pertenencia a una categoría, permitiendo la identificación de nuevos patrones mientras conserva el conocimiento existente.
- Parámetro de Vigilancia: Este parámetro es crucial en ART, ya que controla el grado de similitud necesario para clasificar nuevas entradas con las categorías existentes. Un valor alto de vigilancia lleva a una categorización más específica, mientras que un valor bajo permite una categorización más general.
- Aprendizaje Incremental: Las redes ART son capaces de aprender de forma continua y

adaptativa, integrando nueva información sin alterar las categorías previamente establecidas. Esta característica es esencial para aplicaciones que requieren aprendizaje y adaptación en tiempo real.

Tipos de modelos ART:

ART1: El formulario más simple, diseñado para datos de entrada binarios.

ART2: una extensión que admite entradas de valores continuos.

Fuzzy ART: Incorpora lógica difusa para manejar la incertidumbre en los datos de entrada.

ARTMAP: una versión supervisada de ART que aprende asociaciones entre patrones de entrada y resultados deseados.

TopoART: combina ART difuso con aprendizaje de topología, mejorando la reducción de ruido y las capacidades de agrupación.

Aplicaciones de la Teoría de Resonancia Adaptativa (ART):

- Reconocimiento Facial: Permite adaptarse a cambios en las expresiones faciales para una identificación precisa y consistente.
- Robótica: Facilita el control de movimiento en tiempo real en entornos dinámicos, adaptándose a nuevas situaciones y condiciones.
- Diagnóstico Médico: Ayuda en el reconocimiento de patrones en datos médicos para fines diagnósticos, mejorando la precisión y efectividad de los diagnósticos.
- Agrupación y Minería de Datos: Eficiente en la categorización de grandes conjuntos de datos, manteniendo la integridad del conocimiento previamente aprendido.

Limitaciones de ART:

- Sensibilidad al Orden de los Datos: Algunos modelos ART, como Fuzzy ART y ART1, pueden ser sensibles al orden de los datos de entrenamiento, lo que puede resultar en resultados inconsistentes.
- Modelos Avanzados: Modelos más avanzados como TopoART buscan abordar estos problemas al agrupar categorías en grupos que dependen menos del orden de presentación de los datos. (Petrenko, 2023).

V. Aprendizaje Automático (Machine Learning)

5.1. La regresión lineal y la regresión logística (Linear Logistic Regression)

La regresión lineal y la regresión logística son algoritmos fundamentales en el aprendizaje automático y el análisis de datos, y juegan un papel crucial en la ciencia de datos y la inteligencia artificial.

Regresión Lineal: Se utiliza para predecir una variable continua basándose en una o más características independientes. El modelo ajusta una línea (o un hiperplano en dimensiones más altas) que minimiza la diferencia entre las predicciones y los valores reales. Es útil para entender la relación entre variables y para hacer predicciones numéricas.

Regresión Logística: Se emplea para predecir una variable categórica, típicamente binaria, utilizando características independientes. El modelo estima la probabilidad de que una instancia pertenezca a una categoría en particular mediante una función sigmoide que mapea las predicciones a un rango de 0 a 1. Es fundamental para tareas de clasificación, como la detección de fraudes o la identificación de spam.

Ambos métodos son herramientas esenciales en la ciencia de datos para modelar relaciones y realizar predicciones, facilitando el análisis y la toma de decisiones basada en datos. artificial (Okoye, 2024).

Ilustración 52. Arquitectura de un modelo de regresión logística, incluidas las características de entrada, los pesos y la función logística (sigmoidea). Fuente: Chatgpt4o

Función de Ecuación y Activación:

Regresión Lineal: Utiliza una ecuación lineal simple: \(y = \beta_0 + \beta_1x_1 + \beta_2x_2 + \ldots + \beta_nx_n \).

Regresión Logística: Emplea la función logística (sigmoidea) para transformar la ecuación lineal en una probabilidad: \(y = \frac{1}{1 + e^{-(\beta_0 + \beta_1x_1 + \beta_2x_2 + \ldots + \beta_nx_n)}} \).

Función de Pérdida:

Regresión Lineal: Minimiza la suma de los cuadrados de las diferencias entre los valores previstos y los valores reales (error cuadrático medio).

Regresión Logística: Minimiza la pérdida logística (también conocida como pérdida logarítmica o entropía cruzada) para ajustar el modelo.

Suposiciones:

Regresión Lineal: Asume una relación lineal entre las variables independientes y la variable

```
import numpy as np

class LogisticRegression:
    def __init__(self, learning_rate=0.01, epochs=1000):
        self.learning_rate = learning_rate
        self.epochs = epochs
        self.weights = None
        self.bias = None

    def sigmoid(self, z):
        return 1 / (1 + np.exp(-z))

    def train(self, X, y):
        num_samples, num_features = X.shape
        self.weights = np.zeros(num_features)
        self.bias = 0

        for epoch in range(self.epochs):
            linear_model = np.dot(X, self.weights) + self.bias
            predictions = self.sigmoid(linear_model)

            # Compute gradients
            dw = (1 / num_samples) * np.dot(X.T, (predictions - y))
            db = (1 / num_samples) * np.sum(predictions - y)

            # Update weights and bias
            self.weights -= self.learning_rate * dw
            self.bias -= self.learning_rate * db

            # Compute and print the loss every 100 epochs
            if epoch % 100 == 0:
                loss = -np.mean(y * np.log(predictions) + (1 - y) * np.log(1 - predictions
                print(f'Epoch {epoch}, Loss: {loss:.4f}')
```

Ilustración 53. Pseudocódigo de regresión lineal y regresión logística. Fuente: Los autores.

dependiente, residuos normalmente distribuidos, y varianza constante de los residuos.

Regresión Logística: No asume una relación lineal directa entre las variables independientes y la variable dependiente. Solo asume una relación lineal entre las variables independientes y las probabilidades logarítmicas de la variable dependiente.

Sensibilidad a Valores Atípicos:

Regresión Lineal: Es más sensible a los valores atípicos, ya que estos pueden influir significativamente en la línea de ajuste.

Regresión Logística: Es más robusta frente a los valores atípicos porque se centra en la predicción de probabilidades, no en valores numéricos exactos.

Ambos algoritmos son herramientas fundamentales en el aprendizaje automático y el análisis de datos, cada uno adaptado a diferentes tipos de problemas y datos.

Exactamente, aquí está el resumen de cómo elegir entre regresión lineal y regresión logística:

Regresión Lineal:

Uso: Pronósticos, análisis de tendencias, estimación de efectos de variables.

Variable Dependiente: Continua (por ejemplo, precios, temperaturas, ingresos).

Modelo: Relación lineal entre las variables independientes y la dependiente.

Regresión Logística:

Uso: Clasificación, predicción de riesgos, estimación de probabilidades.

Variable Dependiente: Categórica (generalmente binaria, como sí/no, spam/no spam).

Modelo: Probabilidad de ocurrencia de un evento, transformada en una escala entre 0 y 1 mediante la función sigmoide.

-Se debe elegir regresión lineal, cuando se tenga una variable dependiente continua y quiera modelar una relación lineal.

- Se debe elegir regresión logística cuando se tenga una variable dependiente categórica y desee modelar la probabilidad de que ocurra un evento específico. (Okoye, 2024).

5.2. Árbol de decisiones (Decision trees)

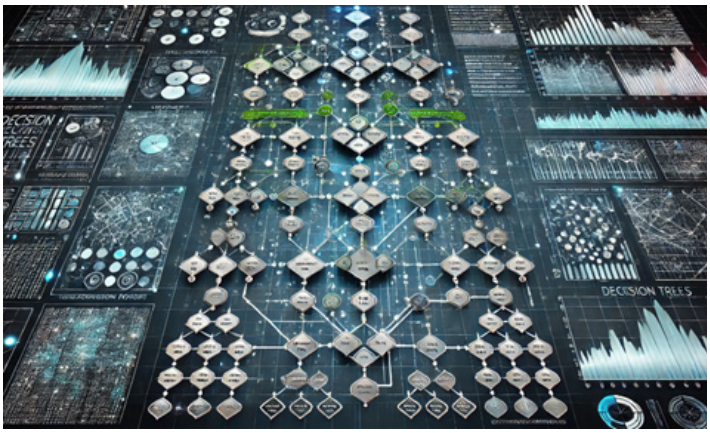

Ilustración 54. La estructura de un árbol de decisión con nodos, ramas y hojas. Fuente: Chatgpt4o.

Un árbol de decisiones es un modelo jerárquico utilizado para respaldar las decisiones y que traza visualmente diferentes decisiones y sus posibles resultados.

Consta de tres elementos principales:

Nodos de decisión (cuadrados): representan una decisión a tomar.

Nodos de probabilidad (círculos): representan resultados o eventos inciertos.

Nodos finales (triángulos): representan los resultados finales.

El árbol comienza con un nodo raíz que contiene la decisión principal, que se ramifica en decisiones alternativas y eventos aleatorios. Cada camino desde la raíz hasta un nodo final representa un posible escenario con sus costos, probabilidades y resultados asociados. Los beneficios clave del uso de árboles de decisión incluyen la división de decisiones complejas en una serie de preguntas más simples, proporciona un marco visual para explorar sistemáticamente todas las opciones y resultados, ayuda a cuantificar el valor esperado de cada resultado en función de

probabilidades y permite la comparación de diferentes escenarios para determinar el curso de acción óptimo.

```python
import numpy as np

class DecisionTree:
    def __init__(self, max_depth=None, min_samples_split=2):
        self.max_depth = max_depth
        self.min_samples_split = min_samples_split
        self.tree = None

    def fit(self, X, y):
        self.tree = self._build_tree(X, y)

    def _build_tree(self, X, y, depth=0):
        num_samples, num_features = X.shape
        if num_samples >= self.min_samples_split and (self.max_depth is None or depth < s
            best_split = self._find_best_split(X, y, num_features)
            if best_split:
                left_subtree = self._build_tree(*best_split["left"], depth + 1)
                right_subtree = self._build_tree(*best_split["right"], depth + 1)
                return Node(best_split["feature"], best_split["threshold"], left_subtree,

        leaf_value = self._most_common_label(y)
        return Leaf(leaf_value)

    def _find_best_split(self, X, y, num_features):
        best_split = {}
        best_gain = -1
        for feature in range(num_features):
            thresholds = np.unique(X[:, feature])
            for threshold in thresholds:
                left_mask = X[:, feature] <= threshold
                right_mask = X[:, feature] > threshold
                if len(np.unique(y[left_mask])) > 1 and len(np.unique(y[right_mask])) > 1
                    gain = self._information_gain(y, y[left_mask], y[right_mask])
```

Ilustración 55. Pseudocódigo de árboles de decisión. Fuente: Los autores.

Los Árboles de Decisión:

Aplicaciones Comunes:

- Investigación de operaciones
- Estrategia comercial
- Análisis de riesgos
- Aprendizaje automático

Ventajas:

- Ofrecen un enfoque visual y estructurado para la toma de decisiones.
- Ayudan a analizar escenarios complejos y cuantificar resultados.
- Facilitan la identificación de las mejores alternativas y decisiones informadas.

Desventajas:

- Sensibles a pequeños cambios en los datos, lo que puede afectar su estabilidad.
- Pueden no ser siempre el modelo predictivo más preciso comparado con otras técnicas.
- Tienen tendencia a sobreajustarse a los datos de entrenamiento, especialmente si el árbol es profundo con muchas ramas.

Técnicas de Mejora:

- Poda (Pruning): Se utiliza para reducir el sobreajuste, eliminando ramas del árbol que aportan poco valor predictivo.

En resumen, los árboles de decisión son herramientas valiosas para la toma de decisiones y análisis en diversos campos, pero es importante gestionar el sobreajuste y considerar su precisión en comparación con otros métodos predictivos. (Merino E. , 2024).

5.3. Naive Bayes Clasification

La clasificación Naive Bayes es un algoritmo de aprendizaje automático supervisado basado en la aplicación del teorema de Bayes para clasificar datos. Hace predicciones calculando la probabilidad de que una instancia determinada pertenezca a cada clase y luego seleccionando la clase con la mayor probabilidad.

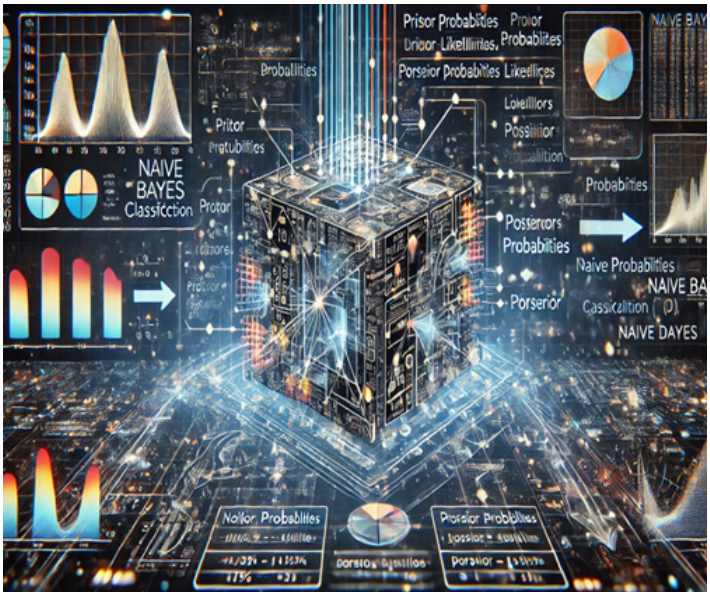

Ilustración 54. La estructura de un árbol de decisión con nodos, ramas y hojas. Fuente: Chatgpt4o.

Se utiliza para calcular probabilidades condicionales.

Formula: \(P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)} \)

```python
import numpy as np

class GaussianNaiveBayes:
    def fit(self, X, y):
        self.classes = np.unique(y)
        self.means = {}
        self.variances = {}
        self.priors = {}

        for c in self.classes:
            X_c = X[y == c]
            self.means[c] = np.mean(X_c, axis=0)
            self.variances[c] = np.var(X_c, axis=0)
            self.priors[c] = X_c.shape[0] / X.shape[0]

    def predict(self, X):
        posteriors = []

        for x in X:
            posteriors.append(self._calculate_posteriors(x))

        return np.array([self.classes[np.argmax(p)] for p in posteriors])

    def _calculate_posteriors(self, x):
        posteriors = []

        for c in self.classes:
            prior = np.log(self.priors[c])
            class_conditional = np.sum(np.log(self._pdf(c, x)))
            posterior = prior + class_conditional
            posteriors.append(posterior)

        return posteriors

    def _pdf(self, class_idx, x):
        mean = self.means[class_idx]
        var = self.variances[class_idx]
```

Ilustración 57. Pseudocódigo de ejemplo con el algoritmo de Bayes. Fuente: Los autores.

Naive Bayes:

Suposición Clave: Cada característica es independiente de las demás dentro de cada clase (supuesto de independencia "ingenua").

Clasificación Probabilística: Genera probabilidades para cada clase. La clase con la mayor probabilidad se selecciona como la predicción final.

Aplicaciones: Filtrado de spam, análisis de opiniones, clasificación de documentos.

Ventajas:

Simplicidad y Velocidad: Fácil de implementar y entrenar; eficiente en términos de tiempo y recursos.

Manejo de Datos de Alta Dimensión: Funciona bien con grandes conjuntos de datos con muchas características.

Desventajas:

Supuesto de Independencia: La suposición de que las características son independientes puede no ser realista, afectando la precisión en algunos casos.

Desempeño en Datos Complejos: Puede tener un rendimiento deficiente si las características están correlacionadas.

Variantes de Naive Bayes:

Gaussian Naive Bayes: Para características continuas, asumiendo una distribución normal.

Multinomial Naive Bayes: Para características discretas, especialmente útil en texto.

Bernoulli Naive Bayes: Para características booleanas.

El clasificador Naive Bayes es un método poderoso y simple que utiliza el teorema de Bayes para realizar clasificaciones probabilísticas. Aunque sufre del supuesto de independencia entre características, sigue siendo muy eficaz y ampliamente utilizado en diversas aplicaciones del mundo real. (Sinaga, 2024).

5.4. K vecinos más cercanos (K nearest neighbors)

k-Nearest Neighbor (k-NN) es un algoritmo de aprendizaje supervisado simple pero potente que se utiliza para tareas de clasificación y regresión. Opera según el principio de que situaciones similares probablemente produzcan resultados

Ilustración 58. K-Vecinos más cercanos (KNN). Fuente: Chatgpt4o

similares. El aprendizaje basado en instancias: k-NN significa que no aprende el modelo explícitamente, sino que recuerda las instancias de entrenamiento. Cuando es necesario clasificar una nueva instancia, el algoritmo la compara con las instancias almacenadas. (Jetawat, 2024).

Métrica de Distancia:

Base en Distancia: El algoritmo utiliza una métrica de distancia, comúnmente la distancia euclidiana, para determinar qué tan cerca están las instancias entre sí.

Cálculo de Distancia: Se calcula la distancia entre la nueva instancia y todas las instancias del conjunto de entrenamiento.

Elegir el Valor de \(k \):
- Número de Vecinos (\(k \)): Representa el número de vecinos más cercanos a considerar al realizar una predicción.

Valor Bajo de \(k \): Puede hacer que el algoritmo sea sensible al ruido y a las variaciones en los datos.

Valor Alto de \(k \): Generalmente proporciona un límite de decisión más suave, pero puede incluir puntos irrelevantes, lo que puede diluir la precisión.

Proceso de Clasificación:

1. Cálculo de Distancia: Para una instancia de prueba, se calcula la distancia a todas las instancias de entrenamiento.

2. Identificación de Vecinos: Se seleccionan las \(k \) instancias de entrenamiento más cercanas a la instancia de prueba.

3. Mecanismo de Votación:

Clasificación: La etiqueta de clase más común entre los \(k \) vecinos se asigna a la instancia de prueba.

Regresión: Se calcula el promedio de los valores de los \(k \) vecinos más cercanos para predecir el valor de la instancia de prueba.

Ventajas:

Simplicidad: El algoritmo es fácil de entender e implementar, sin requerir un modelo complejo.

Sin Fase de Entrenamiento: k-NN es un "alumno perezoso", lo que significa que no necesita una fase de entrenamiento, haciendo la configuración inicial rápida.

Versatilidad: Puede ser utilizado tanto para clasificación como para regresión.

Desventajas:

Intensivo Computacionalmente: A medida que aumenta el tamaño del conjunto de datos, los cálculos de distancia pueden volverse costosos en términos de tiempo y recursos, ralentizando las predicciones.

```python
import numpy as np
from collections import Counter

class KNearestNeighbors:
    def __init__(self, k=3):
        self.k = k
        self.X_train = None
        self.y_train = None

    def fit(self, X_train, y_train):
        self.X_train = X_train
        self.y_train = y_train

    def predict(self, X_test):
        predictions = [self._predict(x) for x in X_test]
        return np.array(predictions)

    def _predict(self, x):
        # Compute distances between x and all examples in the training set
        distances = [np.linalg.norm(x - x_train) for x_train in self.X_train]
        # Sort by distance and return the indices of the first k neighbors
        k_indices = np.argsort(distances)[:self.k]
        # Extract the labels of the k nearest neighbor training samples
        k_nearest_labels = [self.y_train[i] for i in k_indices]
        # Return the most common class label
        most_common = Counter(k_nearest_labels).most_common(1)
        return most_common[0][0]

# Example usage
X_train = np.array([[1, 2], [2, 3], [3, 4], [6, 7], [7, 8]])  # Example features
y_train = np.array([0, 0, 1, 1, 1])                           # Example labels

model = KNearestNeighbors(k=3)
model.fit(X_train, y_train)
```

Ilustración 59. Pseudocódigo con el algoritmo KNN.
Fuente: Los autores.

Maldición de la Dimensionalidad: En espacios de alta dimensión, el concepto de distancia se vuelve menos significativo, lo que puede degradar el rendimiento del algoritmo.

Sensibilidad al Ruido: El algoritmo puede verse afectado por datos ruidosos y características irrelevantes, especialmente con valores bajos de \(k \).

En resumen, el algoritmo k-NN es útil para una variedad de tareas debido a su simplicidad y flexibilidad. Sin embargo, su eficiencia puede verse afectada por el tamaño del conjunto de datos, la dimensionalidad de los datos y la presencia de ruido con un tamaño pequeño. (Jetawat, 2024).

k-NN se usa ampliamente en varios dominios, que incluyen a los sistemas de recomendación: Para sugerir productos basados en similitudes de los usuarios. Reconocimiento de imágenes: para clasificar imágenes en función de las características extraídas de ellas. Diagnóstico Médico: Para predecir enfermedades basándose en los datos del paciente. En resumen, k-Nearest Neighbors es un algoritmo fundamental en el aprendizaje automático, valorado por su simplicidad y eficacia en diversas aplicaciones, a pesar de algunas limitaciones en escalabilidad y sensibilidad a la calidad de los datos (Jetawat, 2024).

5.5. El Análisis de Componentes Principales (Principal Component Analysis (PCA))

El Análisis de Componentes Principales (PCA) es una técnica estadística ampliamente utilizada para la reducción de dimensionalidad y el análisis de datos. Transforma un gran conjunto de variables en un conjunto más pequeño de variables no correlacionadas, conocidas como componentes principales, preservando al mismo tiempo la mayor variación posible del conjunto de datos original.

PCA Análisis de Componentes Principales (PCA):

Reducción de Dimensionalidad: PCA reduce el número de variables en un conjunto de datos mientras conserva la mayor parte de la información importante. Facilita la visualización y el análisis de datos complejos.

Ilustración 60. Análisis de Componentes Principales (PCA). Fuente: Chatgpt4o.

Componentes Principales:

Definición: Son combinaciones lineales de las variables originales que capturan la máxima varianza en los datos.

Primer Componente Principal: Explica la mayor cantidad de variación en los datos.

Segundos Componentes Principales: Capturan las varianzas restantes en orden descendente.

Ortogonalidad: Los componentes principales son ortogonales entre sí, es decir, no están correlacionados.

Proceso de PCA:

1. Estandarización: Escalar los datos para que tengan una media de cero y una desviación estándar de uno. Esto asegura que cada variable tenga el mismo peso en el análisis.

2. Cálculo de la Matriz de Covarianza: Determinar cómo se relacionan entre sí las variables del conjunto de datos.

3. Cálculo de Valores y Vectores Propios:

Valores Propios: Indican la cantidad de varianza capturada por cada componente.

Vectores Propios: Representan las direcciones de los componentes principales.

4.Creación de Vectores de Características: Seleccionar los \(k \) vectores propios principales

basados en sus valores propios para formar un nuevo espacio de características.

5. Transformación de Datos: Proyectar los datos originales en el nuevo espacio definido por los componentes principales, resultando en un conjunto de datos reducido.

Aplicaciones:

Análisis de Datos Exploratorios: Visualización de datos de alta dimensión para identificar patrones o grupos.

Preprocesamiento para Machine Learning: Reducción de dimensionalidad antes de aplicar otros algoritmos, mejorando el rendimiento y reduciendo el sobreajuste.

Compresión de Imágenes: Reducción de la cantidad de variables en datos de imágenes conservando las características esenciales.

Genómica y Bioinformática: Análisis de datos biológicos complejos, como perfiles de expresión genética.

Ventajas:

Simplificación: Facilita la visualización e interpretación de datos complejos.

Reducción de Costo Computacional: Menor número de dimensiones reduce el costo computacional.

Mitigación de la Maldición de la Dimensionalidad: Reduce la complejidad del espacio de datos.

Limitaciones:

Suposición de Linealidad: PCA asume relaciones lineales entre variables, que pueden no siempre ser válidas.

Sensibilidad a Valores Atípicos: Los valores atípicos pueden sesgar los resultados y afectar la calidad del análisis.

PCA es una técnica poderosa para reducir la dimensionalidad y simplificar los datos, lo que facilita su análisis y visualización. Sin embargo, sus supuestos y sensibilidad a los valores atípicos deben ser considerados al aplicar la técnica.

```python
import numpy as np

class PCA:
    def __init__(self, n_components):
        self.n_components = n_components
        self.components = None
        self.mean = None

    def fit(self, X):
        # Step 1: Standardize the data
        self.mean = np.mean(X, axis=0)
        X_centered = X - self.mean

        # Step 2: Compute the covariance matrix
        covariance_matrix = np.cov(X_centered, rowvar=False)

        # Step 3: Compute eigenvalues and eigenvectors
        eigenvalues, eigenvectors = np.linalg.eig(covariance_matrix)

        # Step 4: Sort eigenvalues and eigenvectors
        sorted_indices = np.argsort(eigenvalues)[::-1]
        eigenvalues = eigenvalues[sorted_indices]
        eigenvectors = eigenvectors[:, sorted_indices]

        # Step 5: Select the top n_components eigenvectors
        self.components = eigenvectors[:, :self.n_components]

    def transform(self, X):
        # Step 6: Project the data onto the principal components
        X_centered = X - self.mean
        return np.dot(X_centered, self.components)

# Example usage
X = np.array([[1, 2], [3, 4], [5, 6], [7, 8], [9, 10]])  # Example data

pca = PCA(n_components=1)
pca.fit(X)
X_transformed = pca.transform(X)
print("Transformed Data:", X_transformed)
```

Ilustración 61. Pseudocódigo del algoritmo PCA.
Fuente: Los autores

Los componentes principales pueden no siempre ofrecer una interpretación clara en relación con las variables originales. En resumen, el análisis de componentes principales (PCA) es una herramienta valiosa para reducir la cantidad de dimensiones en un conjunto de datos mientras se mantiene la información relevante, lo que la convierte en un recurso esencial para diversas aplicaciones en análisis de datos y aprendizaje automático. Esta técnica es fundamental en el campo de la inteligencia artificial y el aprendizaje automático para simplificar la dimensionalidad, facilitando el análisis y el procesamiento de grandes

volúmenes de datos. Sin embargo, es crucial aplicar PCA con precaución para minimizar la pérdida de información y asegurarse de que las relaciones lineales sean adecuadas para los datos en cuestión. (Bharathi, 2023).

5.6. Detección de anomalías (Anomaly detection)

La identificación de anomalías en el aprendizaje automático es una técnica clave para detectar elementos, eventos o observaciones que se alejan notablemente del patrón general de los datos. Estas irregularidades pueden señalar problemas importantes, como fraudes, fallos en el equipo o brechas de seguridad. Este proceso utiliza diversas metodologías y puede dividirse en enfoques supervisados y no supervisados. (Bharathi, 2023).

Ilustración 62. Detección de anomalías. Fuente: Chatgpt4o.

Tipos de anomalías

1. **Anomalía de punto:** un único punto de datos que es significativamente diferente del resto del conjunto de datos.

2. **Anomalía Contextual:** Una observación que se considera anómala en un contexto específico pero que puede no serlo en otro.

3. **Anomalía colectiva:** un conjunto de puntos de datos que colectivamente exhiben un comportamiento anómalo, incluso si los puntos individuales no lo hacen.

Técnicas para la detección de anomalías

1. **Detección de anomalías supervisada:** Este enfoque requiere datos de entrenamiento etiquetados que incluyan instancias normales y anómalas. Las técnicas comunes incluyen:

   Redes neuronales: Se utilizan para modelar relaciones complejas en los datos.

   Máquinas de vectores de soporte (SVM): particularmente SVM de clase única, que identifica un límite alrededor de puntos de datos normales.

   K-Vecinos más cercanos (KNN): clasifica las instancias según la distancia a sus vecinos más cercanos.

2. **Detección de anomalías no supervisadas**

Este método no requiere datos etiquetados y se basa en el supuesto de que las anomalías son raras en comparación con los casos normales. Las técnicas incluyen:

Mapas autoorganizados (SOM): se utilizan para agrupar y reducir la dimensionalidad.

K-Means Clustering: agrupa datos en grupos e identifica puntos que quedan fuera de estos grupos.

Bosque de aislamiento: un método de conjunto que aísla anomalías dividiendo aleatoriamente los datos.

Factor de valores atípicos locales (LOF): Mide la desviación de densidad local de un punto de datos en comparación con sus vecinos.

```
# Example usage
X_train = np.array([[1, 2], [2, 3], [3, 4], [8, 8], [9, 9], [10, 10]])  # Example data

model = IsolationForest(n_estimators=100, max_samples=3, max_depth=10)
model.fit(X_train)

X_test = np.array([[2, 3], [9, 9]])
scores = model.score_samples(X_test)
predictions = model.predict(X_test, threshold=0.5)
print("Scores:", scores)
print("Predictions:", predictions)
```

Ilustración 63 . Pseudocódigo detección de anomalías. Fuente: Los autores

Aplicaciones

Detección de fraude: identificación de transacciones fraudulentas en banca y finanzas.

Seguridad de la red: detección de intrusiones o patrones anormales en el tráfico de la red.

Mantenimiento Predictivo: Monitoreo de maquinaria para predecir fallas antes de que ocurran.

Atención médica: identificación de patrones inusuales en datos de pacientes o imágenes médicas.

Desafíos

La detección de anomalías enfrenta varios desafíos, tales como:

Alta dimensionalidad: Detectar anomalías puede ser complicado en espacios con muchas dimensiones.

Datos desequilibrados: Las anomalías suelen ser raras, lo que dificulta el aprendizaje eficaz por parte de los modelos.

Ruido y valores atípicos: Separar las verdaderas anomalías del ruido puede hacer que la detección sea más complicada. La detección de anomalías es esencial en el aprendizaje automático para identificar patrones inusuales en los datos, lo que facilita la toma de medidas preventivas en diversas aplicaciones. La elección de la técnica más adecuada dependerá de la disponibilidad de datos etiquetados y de los requisitos específicos de la aplicación. aplicación (Wei, 2024).

5.7. Random Forest

Es un algoritmo de aprendizaje conjunto popular que se utiliza tanto para tareas de clasificación como de regresión. Combine múltiples árboles de decisión para mejorar la precisión general y la solidez del modelo.

Random Forest genera un grupo de árboles de decisión, cada uno entrenado con un subconjunto aleatorio de datos y características. Utiliza el

Ilustración 64. Random Forest. Fuente: Chatgpt4o.

método de bagging (agregación por arranque) para formar múltiples árboles de decisión, entrenando cada uno en una muestra aleatoria de los datos de entrenamiento. Además, emplea una selección aleatoria de características, de manera que cada árbol solo considera un subconjunto aleatorio de atributos al dividir los nodos, lo que aumenta la aleatoriedad y disminuye la correlación entre los árboles. En tareas de clasificación, el resultado final se obtiene mediante el voto mayoritario de los árboles, mientras que en regresión, se calcula la media de las predicciones de los árboles. Este método es muy escalable y puede manejar grandes volúmenes de datos con eficiencia. También es resistente al sobreajuste y puede capturar relaciones no lineales complejas en los datos. El algoritmo requiere ajustar pocos parámetros, como el número de árboles, la profundidad máxima y el número mínimo de muestras por hoja. Es una técnica eficaz y flexible que mejora la precisión y robustez de las predicciones mediante la combinación de múltiples árboles de decisión. Es adecuada para diversas aplicaciones, desde la clasificación de imágenes hasta la detección de fraudes, y es especialmente valiosa cuando se necesita un modelo preciso y generalizable. No obstante, es crucial considerar el costo computacional y la interpretabilidad al utilizar este modelo en aplicaciones prácticas.

La relevancia de las características en Random Forest se puede determinar mediante la evaluación de la reducción promedio en la precisión a través de todos los árboles. Este método se aplica ampliamente en áreas como la modelización del riesgo crediticio, la predicción de la retención de clientes, el reconocimiento de imágenes y la bioinformática. Como un algoritmo robusto y versátil, Random Forest utiliza el aprendizaje en conjunto para ofrecer predicciones precisas y confiables. Su habilidad para

```python
import numpy as np
from collections import Counter
from sklearn.tree import DecisionTreeClassifier

class RandomForest:
    def __init__(self, n_trees=100, max_features='sqrt', max_depth=None):
        self.n_trees = n_trees
        self.max_features = max_features
        self.max_depth = max_depth
        self.trees = []

    def fit(self, X, y):
        n_samples, n_features = X.shape
        if self.max_features == 'sqrt':
            max_features = int(np.sqrt(n_features))
        elif self.max_features == 'log2':
            max_features = int(np.log2(n_features))
        else:
            max_features = n_features

        self.trees = []
        for _ in range(self.n_trees):
            tree = DecisionTreeClassifier(max_features=max_features, max_depth=self.max_de
            indices = np.random.choice(n_samples, n_samples, replace=True)
            X_sample = X[indices]
            y_sample = y[indices]
            tree.fit(X_sample, y_sample)
            self.trees.append(tree)

    def predict(self, X):
        tree_predictions = np.array([tree.predict(X) for tree in self.trees])
        return np.apply_along_axis(lambda x: Counter(x).most_common(1)[0][0], axis=0, arr

# Example usage
X_train = np.array([[1, 2], [2, 3], [3, 4], [4, 5], [5, 6]])  # Example features
y_train = np.array([0, 0, 1, 1, 1])                           # Example labels

model = RandomForest(n_trees=10, max_features='sqrt', max_depth=None)
model.fit(X_train, y_train)
```

Ilustración 65. Pseudocódigo de Random Forest.
Fuente: Los autores

gestionar diferentes tipos de datos, su capacidad de escalabilidad y su resistencia al sobreajuste lo hacen una opción excelente para resolver numerosos problemas de aprendizaje automático. (Ghosh, 2024).

5.8. Razonamiento automatizado (Automatic reasoning)

El razonamiento automatizado es un campo de la inteligencia artificial dedicado a la creación de algoritmos y sistemas que pueden llevar a cabo tareas de inferencia, deducción y resolución de problemas lógicos, que generalmente requieren inteligencia humana. Utiliza diversas técnicas para generar conclusiones a partir de premisas específicas, permitiendo así que las máquinas simulen los procesos de razonamiento humano.

Tipos de razonamiento: El razonamiento automatizado abarca diversas formas, entre las que se incluyen:

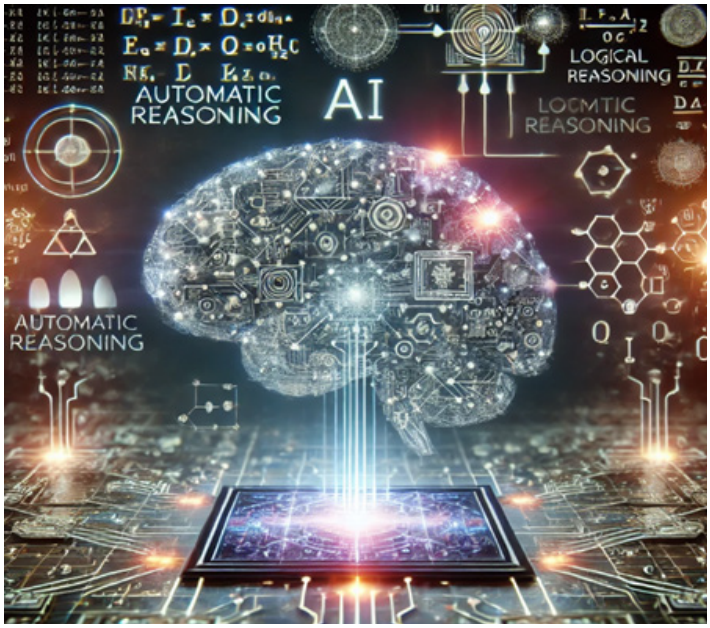

Ilustración 66. Automatic reasoning.
Fuente: Chatgpt4o

Razonamiento Deductivo: Llegar a conclusiones específicas a partir de principios generales.

Razonamiento Inductivo: Inferir normas generales basadas en observaciones específicas.

Razonamiento Abductivo: Proponer la mejor explicación posible para fenómenos observados.

Razonamiento Analógico: Hacer comparaciones entre diferentes situaciones para sacar conclusiones.

Marco: Los sistemas de razonamiento automatizado generalmente comprenden:

Definición del Dominio del Problema: Especificar claramente los problemas que se deben resolver.

Representación del Conocimiento: Utilizar lenguajes y estructuras formales para representar información y sus relaciones.

Mecanismos de Inferencia: Aplicar reglas lógicas para generar nueva información o verificar la existente.

Aplicaciones:

Verificación de Software: Asegurar la corrección y fiabilidad de sistemas de software mediante pruebas formales.

Demostración de Teoremas: Validar automáticamente teoremas matemáticos o verificar la validez de pruebas existentes.

Representación del Conocimiento: Mejorar la recuperación de información y la toma de decisiones en sistemas complejos.

Inteligencia Artificial: Optimizar sistemas inteligentes, como sistemas expertos y tutores inteligentes, para tomar decisiones informadas.

Ventajas:

Eficiencia: Puede resolver problemas complejos más rápidamente que el razonamiento humano.

Precisión: Minimiza los errores humanos, generando resultados más fiables.

Escalabilidad: Capaz de manejar grandes volúmenes de datos y estructuras lógicas complejas.

Limitaciones:

Altos recursos computacionales: Requiere una considerable capacidad de procesamiento y conocimiento especializado.

Manejo de la Incertidumbre: Puede enfrentar dificultades con información ambigua o incierta, que a menudo requiere intuición humana.

Interpretabilidad: Los resultados pueden ser difíciles de interpretar o aplicar por los humanos.

El razonamiento automatizado es crucial en inteligencia artificial, ya que mejora la habilidad de los sistemas para realizar deducción lógica y resolver problemas. Su integración en distintas aplicaciones está en constante desarrollo, promoviendo el avance de sistemas inteligentes que pueden tomar decisiones basadas en datos. A medida que la investigación en este campo avanza, se espera que las técnicas de razonamiento automatizado se vuelvan más eficientes y adaptables, superando los retos actuales en escenarios reales. Este ámbito es muy prometedor dentro de la inteligencia artificial, con aplicaciones que van desde el cuidado de la salud hasta la

```python
import torch
import torch.nn as nn
import torch.optim as optim

# Definir la red neuronal
class ReasoningNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(ReasoningNN, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        out = self.fc1(x)
        out = self.relu(out)
        out = self.fc2(out)
        return out

# Parámetros
input_size = 3   # Número de hechos de entrada
hidden_size = 5  # Tamaño de la capa oculta
output_size = 1  # Número de conclusiones (meta)

# Crear el modelo
model = ReasoningNN(input_size, hidden_size, output_size)
```

Ilustración 67. Pseudocódigo con phytorch de razonamiento automatizado. Fuente: Los autores

automatización de procesos industriales, ofreciendo una forma efectiva y avanzada de tomar decisiones fundamentadas en datos y lógica. (Sayin, 2023) .

5.9. K means clustering

La agrupación en clústeres K-means es un algoritmo de aprendizaje automático no supervisado ampliamente utilizado que divide un conjunto de datos en k clústeres distintos según la similitud de características.

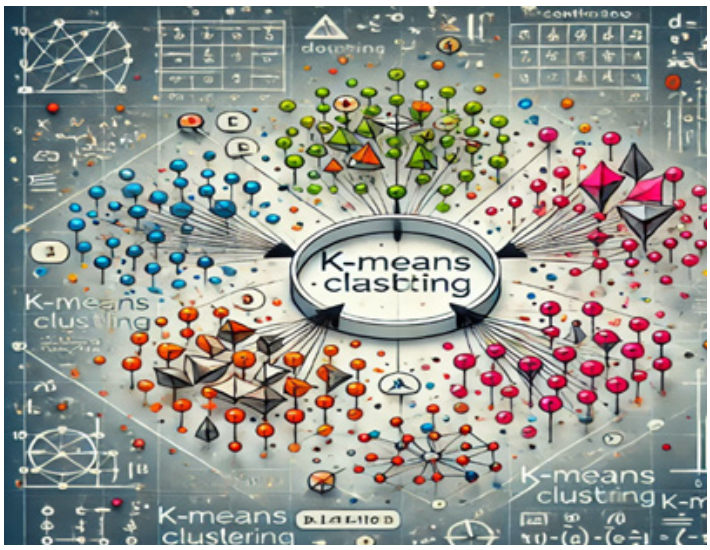

Ilustración 68. Kmeans clustering. Fuente: Chatgpt4o.

El algoritmo K-means se especializa en agrupar datos similares mientras maximiza la separación entre distintos grupos. Este método es una técnica de aprendizaje no supervisado, ideal para el análisis exploratorio cuando se conoce el número de grupos a formar. Cada grupo se define por un centroide, que

es el promedio de todos los puntos en ese grupo. K-means ajusta estos centroides iterativamente para reducir la distancia entre los puntos y sus respectivos centroides.

Proceso de K-means:

1. **Inicialización:** Selecciona aleatoriamente \(k\) centroides iniciales del conjunto de datos.

2. **Asignación:** Asigna cada punto al centroide más cercano, creando así \(k\) grupos.

3. **Actualización:** Recalcula los centroides tomando el promedio de los puntos en cada grupo.

4. **Iteración:** Repite los pasos de asignación y actualización hasta que los centroides se estabilicen o se alcance un número máximo de iteraciones.

K-means se enfoca en minimizar la suma de las distancias al cuadrado dentro de cada grupo (WCSS), buscando clusters que sean lo más compactos posible.

Determinación del número de clusters (\(k\)):

Elegir el número óptimo de clusters puede ser complicado. Dos enfoques comunes para esto son:

Método del codo: Se grafica el WCSS en función de diferentes valores de \(k\) y se busca un punto donde la reducción del WCSS se estabiliza, conocido como el "codo".

Puntuación de silueta: Evalúa la similitud de un punto con su propio grupo en comparación con otros grupos, ofreciendo una forma de medir la calidad del agrupamiento.

Ventajas:

Simplicidad: El método es fácil de entender y aplicar, lo que lo hace popular para tareas de agrupación.

Eficiencia: Es eficiente en términos computacionales, especialmente para grandes conjuntos de datos.

Escalabilidad: Puede manejar un gran número de puntos y grupos.

Limitaciones:

Sensibilidad a la inicialización: Los resultados pueden variar según la selección inicial de los centroides, lo que puede influir en los resultados finales.

Supuesto de esfericidad: K-means asume que los clusters son esféricos y tienen tamaño uniforme, lo cual puede no ser representativo de los datos reales.

Sensibilidad a valores atípicos: Los valores atípicos pueden afectar de manera desproporcionada la ubicación de los centroides, distorsionando los resultados.

Aplicaciones:

Segmentación de mercado: Agrupa clientes según sus patrones de compra para desarrollar estrategias de marketing específicas.

Compresión de imágenes: Reduce el número de colores en una imagen agrupando colores similares.

Agrupación de documentos: Organiza documentos en grupos basados en la similitud de contenido para mejorar la recuperación de información.

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans

# Generar datos de ejemplo
X, y = make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=0)

# Visualizar los datos
plt.scatter(X[:, 0], X[:, 1], s=50)
plt.show()

# Aplicar K-means
kmeans = KMeans(n_clusters=4)
kmeans.fit(X)
y_kmeans = kmeans.predict(X)

# Visualizar los clusters formados
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=50, cmap='viridis')

# Marcar los centroides
centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='red', s=200, alpha=0.75)
plt.show()
```

Ilustración 69. Algoritmo de k- medias en phyton.
Fuente: Los autores

K-means es una técnica esencial en el aprendizaje no supervisado, valorada por su capacidad para agrupar datos similares de manera eficiente y efectiva. Aunque presenta algunas limitaciones, como la sensibilidad a la inicialización y los valores atípicos, sigue siendo una herramienta popular para diversas aplicaciones.

5.10. El aprendizaje por refuerzo (Reinforcement learning)

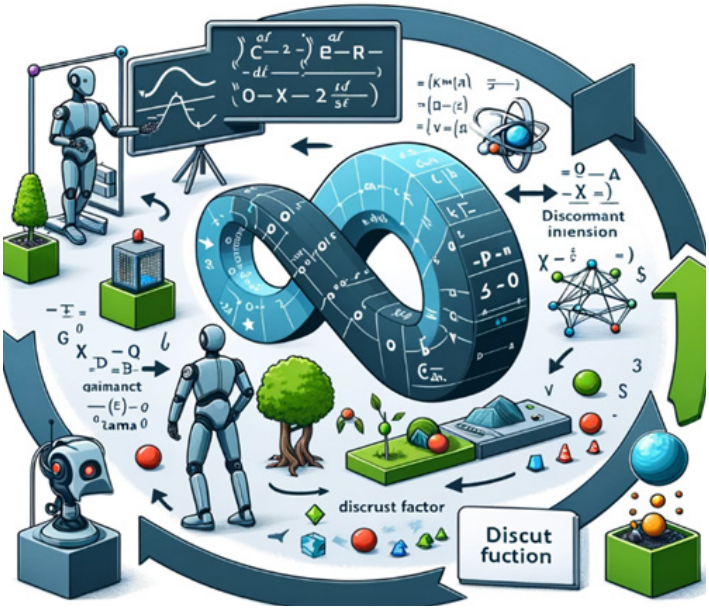

Ilustración 70. Reinforcment learning. Fuente: Chatgpt4o.

El aprendizaje por refuerzo (RL) es una técnica de aprendizaje automático en la que un agente aprende a ejecutar tareas a través de la experimentación y la retroalimentación continua con un entorno cambiante. (Umapathy, 2023).

El objetivo del agente es maximizar una métrica de recompensa tomando acciones que conduzcan a los resultados más favorables.

- **Agente:** El alumno o quien toma las decisiones.

- **Entorno:** Todo con lo que interactúa el agente.

- **Acciones:** Qué puede hacer el agente

- **Recompensas:** señales de retroalimentación escalar que el agente intenta maximizar con el tiempo

- **Estado:** La condición actual del medio ambiente

El agente se educa explorando el entorno, llevando a cabo diversas acciones y observando las recompensas o penalizaciones resultantes. Gradualmente, el agente descubre qué acciones resultan en mayores recompensas acumulativas con el tiempo. A diferencia del aprendizaje supervisado, el aprendizaje por refuerzo no necesita datos etiquetados para el entrenamiento; en cambio, el agente aprende a partir de sus propias experiencias e interacciones con el entorno.

- Capacidad para resolver problemas complejos.

- Corregir errores durante el entrenamiento.

- Obtención de datos de entrenamiento mediante interacción directa.

- Manejo de entornos no deterministas.

```python
import numpy as np
import gym

# Inicializar el entorno de Gym (por ejemplo, FrozenLake)
env = gym.make('FrozenLake-v1')

# Inicializar la Q-table
state_size = env.observation_space.n
action_size = env.action_space.n
Q = np.zeros((state_size, action_size))

# Parámetros del algoritmo
alpha = 0.1    # Tasa de aprendizaje
gamma = 0.99   # Factor de descuento
epsilon = 1.0  # Epsilon inicial para la política epsilon-greedy
epsilon_decay = 0.995
epsilon_min = 0.01
episodes = 1000

# Algoritmo Q-Learning
for episode in range(episodes):
    state = env.reset()
    done = False
```

Ilustración 71 Reinforcmente learning phyton. Fuente: Los autores.

Las aplicaciones del aprendizaje por refuerzo incluyen robótica, juegos, asignación de recursos y sistemas de control adaptativos. En resumen, el aprendizaje por refuerzo permite a los agentes aprender comportamientos óptimos interactuando con su entorno y recibiendo retroalimentación en forma de recompensas o castigos. Es una técnica poderosa para entrenar sistemas autónomos para que tomen decisiones secuenciales que maximicen las recompensas a largo plazo (Umapathy, 2023).

5.11. Métodos de conjunto (Ensemble methods)

Los métodos de conjunto son técnicas de aprendizaje automático que integran múltiples modelos para elevar el rendimiento general, la robustez y la exactitud. La premisa central es que al combinar las predicciones de varios modelos base, el conjunto puede ofrecer una mejor generalización en comparación con cualquier modelo individual.

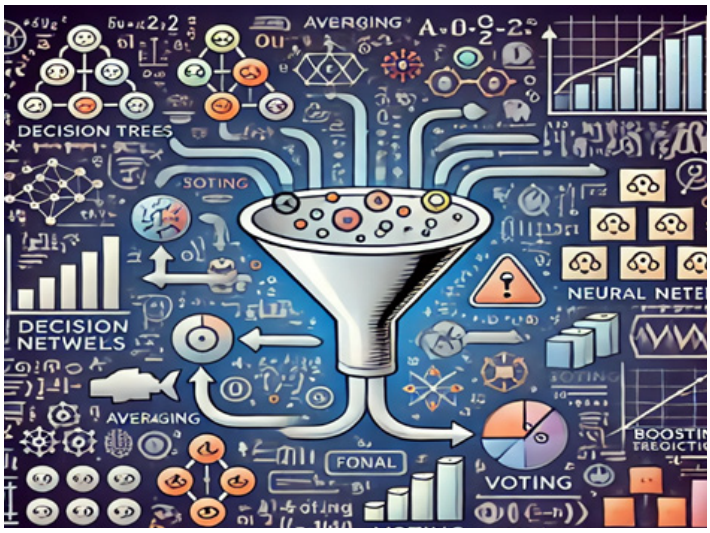

Ilustración 72. Ensemble methods. Fuente: Chatgpt4o

Los tipos clave de métodos de conjunto incluyen:

Embolsado (agregación Bootstrap)
- Implica entrenar múltiples modelos en diferentes subconjuntos de datos de entrenamiento, creados mediante muestreo aleatorio con reemplazo.

- Cada modelo se entrena de forma independiente y sus predicciones se agregan.

- Ejemplo: Random Forest, que crea múltiples árboles de decisión y agrega sus predicciones.

Impulsando
- Entrena modelos secuencialmente, donde cada nuevo modelo se enfoca en corregir los errores cometidos por los anteriores.

- Las predicciones se combinan de forma ponderada, dando más peso a las predicciones de los modelos que tuvieron un rendimiento deficiente.

- Ejemplos: AdaBoost, aumento de gradiente y XGBoost.

Apilamiento
- Combina diferentes tipos de modelos (aprendices base) y utiliza otro modelo (metaaprendiz) para aprender cómo combinar mejor sus predicciones.

- Los alumnos base se entrenan con el mismo conjunto de datos y sus predicciones se utilizan como entrada para el metaaprendiz.

Algunas ventajas clave de los métodos de conjunto incluyen:

Precisión mejorada: los métodos de conjunto a menudo superan a los modelos individuales al reducir la varianza (en el embolsado) y el sesgo (en el impulso).

Robustez: Son menos sensibles al ruido y al sobreajuste, ya que la combinación de varios modelos puede suavizar los errores.

Versatilidad: Se puede aplicar a una amplia gama de algoritmos y problemas, lo que los hace adecuados para diversas tareas.

```
models = {
    'Random Forest': rf,
    'AdaBoost': adaboost,
    'Stacking': stacking
}

# Evaluar y mostrar los resultados de cada modelo
for name, model in models.items():
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"{name} Accuracy: {accuracy:.4f}")

# Visualizar la importancia de las características con Random Forest
importances = rf.feature_importances_
indices = np.argsort(importances)[::-1]

plt.figure(figsize=(10, 6))
plt.title("Feature Importances - Random Forest")
plt.bar(range(X.shape[1]), importances[indices], align="center")
plt.xticks(range(X.shape[1]), indices)
plt.xlabel("Feature Index")
plt.ylabel("Importance")
plt.show()
```

Ilustración 73. Resultados con métodos de ensamble. Fuente: Los autores

Los métodos de conjunto encuentran aplicaciones extensivas en sectores como finanzas, atención médica, marketing y reconocimiento de imágenes. Aunque son menos comunes en el aprendizaje no supervisado, como en la agrupación en clústeres, también se han creado enfoques para aplicar técnicas de conjunto en estos algoritmos. En resumen, los métodos de conjunto maximizan las fortalezas de diversos modelos para optimizar el rendimiento predictivo y la robustez, haciendo de ellos una herramienta potente en el aprendizaje automático. Su habilidad para aumentar la precisión y disminuir el sobreajuste los ha hecho muy populares en una amplia gama de aplicaciones (Zhang, 2024).

5.12.Las máquinas de soporte de vectores (Support vector machines)

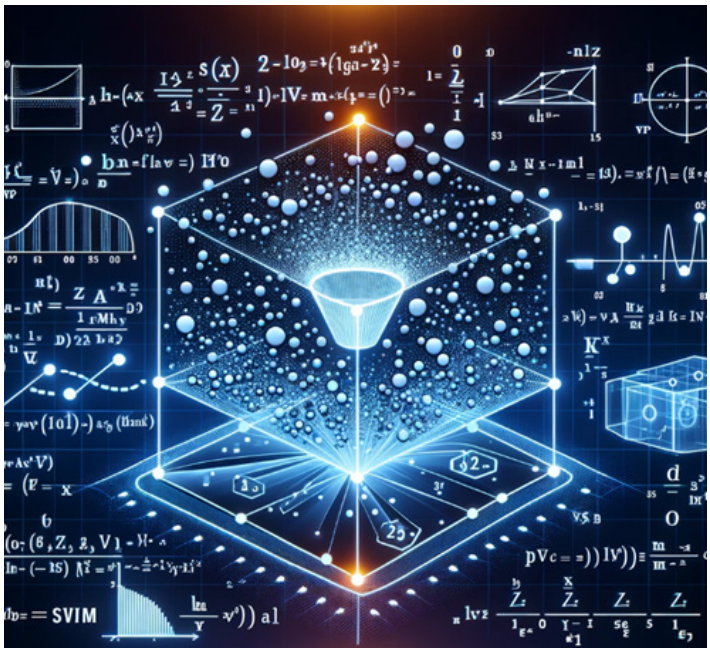

Ilustración 74 Support vector machines. Fuente: Chatgpt4o.

Las máquinas de vectores de soporte (SVM) son un robusto algoritmo de aprendizaje supervisado, principalmente utilizado para problemas de clasificación, aunque también puede adaptarse a tareas de regresión. El concepto fundamental de SVM es identificar el hiperplano óptimo que divide los puntos de datos de distintas clases en un espacio de alta dimensión. (Zhang, 2024).

Conceptos clave

-Hiperplano: En SVM, un hiperplano es un límite de decisión que separa diferentes clases. En dos dimensiones, ésta es una línea; en tres dimensiones es un plano; y en dimensiones superiores, se le conoce como hiperplano.

- Vectores de soporte: Estos son los puntos de datos más cercanos al hiperplano y son críticos para definir su posición. El algoritmo SVM se centra en estos vectores de soporte para determinar el hiperplano óptimo.

- Margen: El margen es la distancia entre el hiperplano y los vectores de soporte más cercanos de cualquier clase. SVM tiene como objetivo maximizar este margen para mejorar la capacidad de generalización del modelo.

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Generar datos de ejemplo
X, y = make_classification(n_samples=1000, n_features=20, n_classes=2, random_state=42)

# Dividir los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Crear y entrenar el modelo SVM
svm = SVC(kernel='linear', C=1.0)
svm.fit(X_train, y_train)

# Realizar predicciones
y_pred = svm.predict(X_test)

# Evaluar el modelo
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")

# Mostrar la matriz de confusión y el reporte de clasificación
print("Confusion Matrix:")
```

Ilustración 73. Resultados con métodos de ensamble. Fuente: Los autores

Cómo funciona SVM

1. Representación de datos: SVM transforma los datos de entrada en un espacio de dimensiones superiores (si es necesario) utilizando una función del núcleo, lo que permite la separación de datos separables no linealmente.

2. Encontrar el hiperplano: el algoritmo identifica el hiperplano que maximiza el margen entre las clases.

3. Clasificación: una vez establecido el hiperplano, los nuevos puntos de datos se pueden clasificar según el lado del hiperplano en el que se encuentran.

Funciones del núcleo

SVM utiliza funciones del kernel para transformar datos en dimensiones superiores sin calcular explícitamente las coordenadas en ese espacio. Las funciones comunes del kernel incluyen:

Núcleo lineal: Adecuado para datos separables linealmente.

Núcleo polinomial: captura interacciones entre características.

Núcleo de función de base radial (RBF): eficaz para datos no lineales, mapeando puntos en un espacio de dimensión infinita.

Ventajas de SVM

Efectivo en dimensiones altas: SVM funciona bien en espacios de altas dimensiones y es efectivo cuando el número de dimensiones excede el número de muestras.

Robusto al sobreajuste: particularmente en espacios de alta dimensión, SVM es menos propenso al sobreajuste, especialmente con la elección correcta de los parámetros de regularización y del kernel.

VI. Inteligencia Artificial

6.1. Visual Perception

En el ámbito de la inteligencia artificial (IA), la percepción visual se centra en crear sistemas y algoritmos que emulan el procesamiento visual humano para interpretar y comprender información visual. Este campo es esencial para aplicaciones como la visión por computadora, el reconocimiento de imágenes y los sistemas autónomos. (Zhang, 2024).

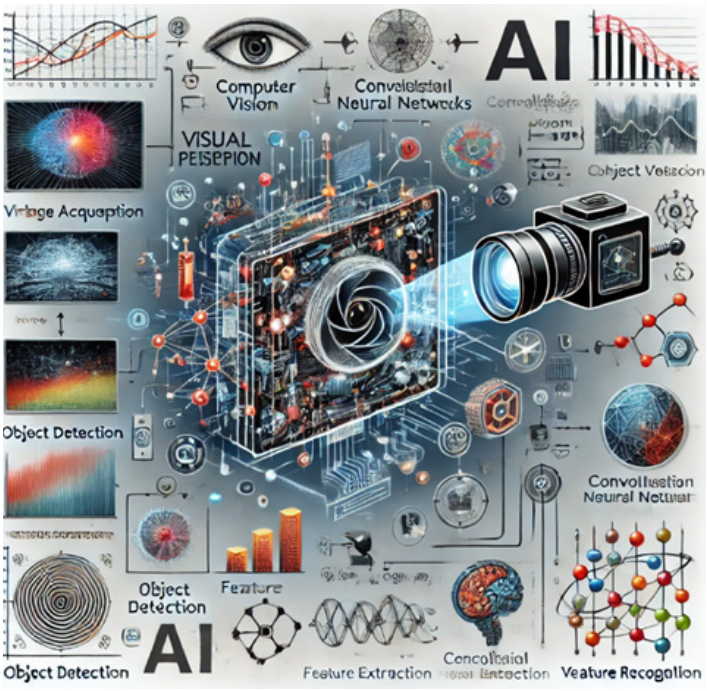

Ilustración 76. Visual perception. Fuente: Chatgpt4o

Aspectos clave de la percepción visual en la IA

La percepción de conjunto se refiere a la habilidad de extraer y resumir información sobre grupos de objetos, lo que facilita una interpretación eficiente de escenas visuales. Este concepto es crucial en la IA, ya que optimiza el procesamiento y análisis de la información visual. Los estudios sugieren que el sistema visual humano tiene la capacidad de sintetizar las características de varios objetos (como tamaño, color y orientación) sin necesidad de identificar cada uno de forma individual. Este enfoque puede ser utilizado en IA para mejorar la eficacia en el procesamiento visual. (Leoshchenko, 2024).

Técnicas de aprendizaje automático:

En la inteligencia artificial, se emplean diversas técnicas de aprendizaje automático, incluido el aprendizaje profundo, para analizar datos visuales. Las redes neuronales convolucionales (CNN) son especialmente efectivas en la clasificación de imágenes y la detección de objetos, ya que tienen la capacidad de aprender características jerárquicas a partir de datos de píxeles sin necesidad de procesamiento manual. También se utilizan métodos de conjunto, que combinan las predicciones de varios modelos para mejorar la precisión general. Estos enfoques contribuyen a una mayor robustez y capacidad de generalización al integrar los resultados de diferentes modelos entrenados con datos visuales.

La percepción visual en la IA tiene numerosas aplicaciones, entre ellas:

- **Vehículos autónomos:** permiten que los vehículos interpreten su entorno, reconozcan obstáculos y tomen decisiones de conducción.

Reconocimiento facial: identificación y verificación de individuos en función de los rasgos faciales, lo que se basa en la comprensión de las propiedades del conjunto de las estructuras faciales.

Imágenes médicas: ayuda en el análisis de imágenes médicas (p. ej., resonancias magnéticas, tomografías computarizadas) para detectar anomalías o enfermedades al resumir la información visual de manera efectiva.

Uno de los principales retos en emular la percepción visual humana es gestionar la variabilidad en la información visual, como variaciones en la iluminación, obstrucciones y cambios en la perspectiva. Los sistemas de inteligencia artificial deben ser diseñados para ser robustos ante estas variaciones, asegurando que la precisión se mantenga a pesar de los cambios en las condiciones visuales.

Además, entender los procesos cognitivos que subyacen a la percepción visual humana puede mejorar el diseño de sistemas de inteligencia artificial, facilitando interacciones más naturales y efectivas

```
# Visualizar algunas predicciones
for i in range(5):
    plt.imshow(X_test[i].reshape(28, 28), cmap='gray')
    plt.title(f'Prediction: {np.argmax(predictions[i])}, True label: {np.argmax(y_test[i])}')
    plt.show()
```

Ilustración 77. Visualización de predicciones. Fuente: Los autores.

entre humanos y máquinas. La percepción visual en IA implica desarrollar y aplicar algoritmos que permiten a las máquinas interpretar y comprender datos visuales de manera análoga a cómo lo hacen los seres humanos. Aprovechando conceptos como la percepción de conjunto y utilizando técnicas avanzadas de aprendizaje automático, los sistemas de inteligencia artificial pueden alcanzar capacidades destacadas en diversas áreas, desde la conducción autónoma hasta el diagnóstico médico. La investigación continua en este ámbito busca perfeccionar la precisión, eficiencia y robustez de los sistemas de percepción visual, con el objetivo final de reducir la distancia entre la forma en que los humanos y las máquinas procesan la información visual. (Leoshchenko, 2024) .

6.2. Planificación y Programación (Planning and Scheduling)

La planificación y programación en inteligencia artificial (IA) se centra en crear algoritmos y sistemas que capaciten a las máquinas para tomar decisiones sobre cómo asignar recursos, ejecutar tareas y gestionar el tiempo en diferentes aplicaciones. Estas actividades son cruciales para mejorar la eficiencia operativa en áreas como la robótica, la logística, la manufactura y la gestión de proyectos.

Ilustración 78 . Planning and Sheduling. Fuente: Chatgpt4o.

Planificación: Este proceso implica definir una serie de acciones que deben tomarse para alcanzar objetivos específicos. En IA, se elabora un modelo del entorno y se establecen metas para crear un plan detallado que indique los pasos necesarios para lograrlas. Este enfoque puede ser jerárquico, organizando las acciones según sus relaciones de dependencia, o temporal, estableciendo el momento adecuado para cada acción.

Programación: Contrariamente, la programación se enfoca en la asignación eficiente de recursos a lo largo del tiempo. Esto incluye decidir el momento y el lugar para llevar a cabo cada tarea con el objetivo de optimizar el uso de recursos y cumplir con los plazos establecidos. Los algoritmos de programación en IA son cruciales para manejar sistemas complejos, como la coordinación de múltiples robots o la optimización de procesos en entornos industriales.

Técnicas en Planificación y Programación de IA:

Algoritmos de Búsqueda: Herramientas como la búsqueda en amplitud, la búsqueda en profundidad y la búsqueda A* se utilizan para explorar diferentes secuencias de acciones y encontrar los planes más efectivos.

Satisfacción de Restricciones: Esta técnica asegura que las restricciones definidas, como la disponibilidad de recursos, las dependencias entre tareas y los límites de tiempo, sean cumplidas durante la programación.

Métodos Heurísticos: Las heurísticas son utilizadas para guiar el proceso de planificación y programación, mejorando la eficiencia al enfocar la búsqueda en áreas prometedoras del espacio de soluciones.

Aprendizaje por Refuerzo: En entornos que cambian, el aprendizaje por refuerzo permite que los sistemas aprendan de manera adaptativa para mejorar la planificación y la programación mediante la retroalimentación del entorno.

Aplicaciones:

Robótica: La planificación y programación en IA son fundamentales para la coordinación de robots en actividades como ensamblaje, navegación y colaboración entre robots.

Gestión de Logística y Cadena de Suministro: La IA optimiza la programación de entregas, la gestión de inventarios y la asignación de recursos, mejorando la eficiencia y reduciendo costos.

Fabricación: En la producción industrial, la IA se encarga de programar operaciones, gestionar flujos de trabajo y minimizar el tiempo de inactividad de las máquinas.

Gestión de Proyectos: La IA puede asistir en la programación de proyectos analizando dependencias, estimando la duración de tareas y optimizando la asignación de recursos para asegurar que los proyectos se completen a tiempo.

```
# Parámetros del problema
num_tasks = 10
num_machines = 3
task_durations = [random.randint(1, 10) for _ in range(num_tasks)]

# Parámetros del Algoritmo Genético
pop_size = 20
generations = 100
mutation_rate = 0.01

# Ejecutar el Algoritmo Genético
best_schedule, best_fitness = genetic_algorithm(task_durations, num_tasks, num_machines,
print("Mejor plan encontrado:", best_schedule)
print("Tiempo total de finalización:", best_fitness)
```

Ilustración 79. Parámetros del Algoritmo genético en phyton. Fuente: Los autores.

Desafíos en Planificación y Programación en IA

Complejidad: Los problemas de planificación y programación pueden volverse muy complejos, especialmente a medida que aumentan el número de tareas y recursos involucrados. Esta complejidad puede llevar a que las soluciones requieran una gran cantidad de recursos computacionales para ser procesadas de manera efectiva.

Entornos Dinámicos: En situaciones del mundo real, los entornos pueden experimentar cambios impredecibles, lo que plantea un desafío para las estrategias de planificación y programación. Los sistemas deben ser capaces de adaptarse rápidamente a estas variaciones para mantener su efectividad.

Incertidumbre: La incertidumbre relacionada con la duración de las tareas, la disponibilidad de recursos y factores externos puede complicar significativamente los procesos de planificación y programación. Manejar estas variables inciertas requiere enfoques flexibles y estrategias que puedan adaptarse a nuevas informaciones y cambios en las condiciones.

6.3. La robótica Inteligente (Intelligent Robotics).

La robótica inteligente es un campo que combina la inteligencia artificial y la robótica para crear robots capaces de percibir, aprender y adaptarse a su entorno. Implica desarrollar robots que puedan realizar tareas de forma autónoma, tomar decisiones e interactuar con humanos y otros robots de manera inteligente. (Asencio, 2023).

Ilustración 80. La robótica inteligente Fuente: Chatgpt4o.

Los aspectos clave de la robótica inteligente incluyen:

Percepción y Sensación

Los robots deben poder percibir e interpretar su entorno mediante sensores como cámaras, sensores táctiles y sensores de proximidad, se utilizan técnicas de visión por computadora y

reconocimiento de objetos para identificar y localizar objetos, obstáculos y otras características relevantes, la fusión de sensores combina datos de múltiples sensores para crear una representación más completa y precisa del entorno.

Razonamiento y toma de decisiones

Los robots inteligentes utilizan técnicas de inteligencia artificial como el aprendizaje automático, la representación del conocimiento y el razonamiento para tomar decisiones y planificar acciones, el aprendizaje por refuerzo permite a los robots aprender comportamientos óptimos mediante interacciones de prueba y error con el entorno, los algoritmos de planificación se utilizan para generar secuencias de acciones para lograr objetivos específicos teniendo en cuenta limitaciones e incertidumbres.

Aprendizaje y Adaptación

Los robots pueden aprender nuevas habilidades y adaptarse a los cambios en su entorno mediante técnicas como el aprendizaje por imitación, el aprendizaje activo y el aprendizaje por transferencia, la robótica del desarrollo tiene como objetivo crear robots que puedan adquirir de forma autónoma nuevas habilidades y conocimientos a lo largo de su vida, de forma similar a cómo aprenden los humanos y los animales, la robótica evolutiva utiliza algoritmos evolutivos para optimizar el diseño y control de los robots.

Interacción humano-robot

Los robots inteligentes deben poder interactuar con los humanos de forma natural e intuitiva, utilizando técnicas como el procesamiento del lenguaje natural, el reconocimiento de gestos y la interacción social, pueden ayudar y colaborar con los humanos en diversas tareas, como la fabricación, la atención sanitaria y la educación, las consideraciones éticas son importantes a la hora de desarrollar robots inteligentes que puedan tomar decisiones autónomas e interactuar con los humanos.

Las aplicaciones de la robótica inteligente incluyen:

- Robots de servicio para tareas personales y domésticas.
- Robots industriales para fabricación y montaje.
- Robots quirúrgicos para procedimientos mínimamente invasivos.
- Robots de exploración para entornos espaciales, submarinos y peligrosos.
- Robots de asistencia para personas mayores y personas con discapacidad.

```python
import numpy as np
import gym

# Crear el entorno FrozenLake
env = gym.make('FrozenLake-v1')

# Inicializar la Q-table con ceros
state_size = env.observation_space.n
action_size = env.action_space.n
Q = np.zeros((state_size, action_size))

# Parámetros del algoritmo
alpha = 0.1    # Tasa de aprendizaje
gamma = 0.99   # Factor de descuento
epsilon = 1.0  # Epsilon inicial para la política epsilon-greedy
epsilon_decay = 0.995
epsilon_min = 0.01
episodes = 1000

# Algoritmo Q-Learning
for episode in range(episodes):
    state = env.reset()
    done = False

    while not done:
```

Ilustración 81. Algoritmo de robótica inteligente en phyton. Fuente: Los autores.

Los principales desafíos en la robótica inteligente incluyen:

Incertidumbre: Los robots deben gestionar la incertidumbre inherente en la percepción y el entorno, donde los datos pueden ser imprecisos o incompletos. Esto demanda sistemas robustos que puedan adaptarse a información variable y tomar decisiones confiables.

Entornos Dinámicos: Los robots necesitan ajustarse rápidamente a cambios en su entorno. Los entornos reales suelen ser impredecibles y pueden cambiar repentinamente, por lo que los robots deben ser lo suficientemente versátiles para manejar tales variaciones.

Seguridad y Fiabilidad: La operación segura y fiable de robots es fundamental, especialmente en áreas críticas como la atención médica y la producción industrial. Los sistemas deben

garantizar un funcionamiento continuo sin fallos y prevenir cualquier riesgo potencial.

Ética en la Inteligencia Artificial: Desarrollar sistemas de inteligencia artificial que sean éticos y transparentes es esencial. Los robots deben operar de manera justa, evitando sesgos y tomando decisiones que respeten principios morales.

La investigación en robótica inteligente se enfoca en superar estos desafíos para crear robots más adaptables e inteligentes. Aprovechando avances en IA, sensores y actuadores, los robots pueden llevar a cabo tareas complejas y adaptarse a entornos variados con mayor eficacia. eficiente (Asencio, 2023).

6.4. Reconocimiento de voz (Speech recognition)

El reconocimiento de voz, o reconocimiento automático de voz (ASR), es una tecnología fundamental en inteligencia artificial (IA) que convierte el habla humana en texto escrito. Esta capacidad ha revolucionado la interacción entre las personas y las máquinas, permitiendo una comunicación más fluida y natural mediante comandos de voz y otras aplicaciones. (Asencio, 2023).

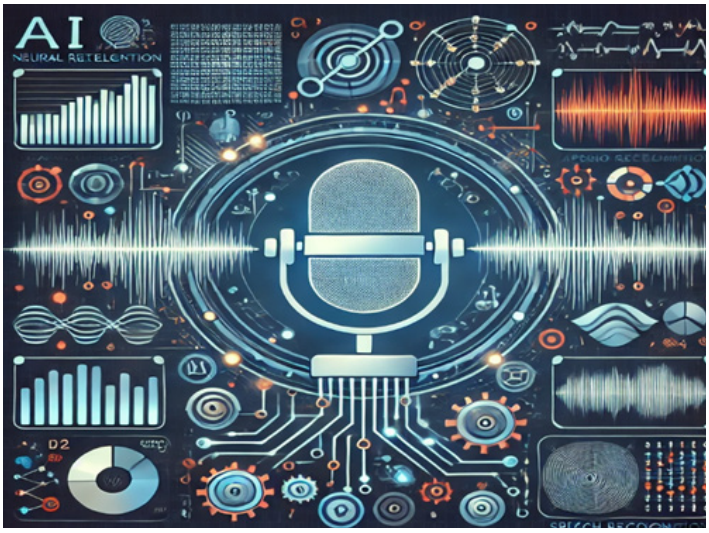

Ilustración 82. Reconocimiento de voz. Fuente: Chatgpt4o.

Cómo funciona el reconocimiento de voz

El reconocimiento de voz sigue un proceso detallado que incluye:

Captura y Procesamiento de Audio: Se inicia con la captura de la señal de audio, que luego se procesa para reducir el ruido y convertir la señal analógica en una digital.

Extracción de Características: Se extraen propiedades importantes de la señal de voz, como patrones de frecuencia y amplitud, que son esenciales para el análisis del habla.

Identificación de Patrones: Los algoritmos de aprendizaje automático examinan las características extraídas para reconocer palabras o frases, a menudo entrenando modelos con extensos conjuntos de datos de voz para mejorar la precisión.

Decodificación: Se combinan los modelos acústicos, que traducen fonemas en señales de audio, con los modelos de lenguaje, que ayudan a predecir secuencias de palabras, para convertir el audio en texto.

Ajuste Final: El texto generado puede ser ajustado para corregir errores gramaticales, puntuación y formato para asegurar una mejor presentación.

Aplicaciones del reconocimiento de voz

La tecnología de reconocimiento de voz se aplica ampliamente en diversas industrias y aplicaciones, que incluyen:

Asistentes virtuales: sistemas como Siri, Alexa y Google Assistant utilizan el reconocimiento de voz para comprender y responder a los comandos del usuario.

Servicios de transcripción: conversión del lenguaje hablado en texto escrito para reuniones, entrevistas y procedimientos legales.

Interfaces activadas por voz: permite el control manos libres de dispositivos en aplicaciones de automoción, atención sanitaria y hogar inteligente.

Servicio al Cliente: Automatizar respuestas en call center y mejorar las interacciones de los usuarios a través del reconocimiento de voz.

Desafíos en el reconocimiento de voz

A pesar de los importantes avances, la tecnología de reconocimiento de voz enfrenta varios desafíos:

Acentos y dialectos: las variaciones en la pronunciación pueden afectar la precisión del reconocimiento, lo que dificulta que los sistemas comprendan a diferentes hablantes.

Ruido de fondo: los sonidos ambientales pueden interferir con la claridad del habla y provocar errores de transcripción.

Aprendizaje continuo: a medida que el lenguaje evoluciona, los sistemas de reconocimiento de voz deben actualizarse periódicamente con nuevo vocabulario y patrones de uso para mantener la precisión.

Preocupaciones por la privacidad: el manejo y almacenamiento de datos de voz plantea problemas relacionados con la privacidad del usuario y la seguridad de los datos.

Futuro del reconocimiento de voz

```python
import speech_recognition as sr
import pyttsx3

# Inicializar motores de STT y TTS
recognizer = sr.Recognizer()
tts_engine = pyttsx3.init()

# Definir un conjunto simple de Skills
skills = {
    "hello": "Hola, ¿cómo estás?",
    "time": "Lo siento, no puedo decirte la hora en este momento."
}

# Función para convertir texto a voz
def speak(text):
    tts_engine.say(text)
    tts_engine.runAndWait()

# Función para reconocer la intención del usuario
def recognize_intent(text):
    for keyword in skills.keys():
        if keyword in text.lower():
            return skills[keyword]
    return "Lo siento, no entendí eso."
```

Ilustración 83. Mycroft. Fuente: Mycroft.

El campo del reconocimiento de voz está evolucionando rápidamente y hay investigaciones en curso centradas en mejorar la precisión y la usabilidad. Los avances en el aprendizaje profundo y las redes neuronales están mejorando las capacidades de los sistemas de reconocimiento de voz, permitiéndoles comprender mejor el contexto y los matices del habla humana. (Paiva, 2024).

Este ejemplo proporciona una base para un asistente de voz simple. Mycroft es mucho más avanzado y modular, utilizando componentes intercambiables y un ecosistema de Skills que pueden ser desarrollados y compartidos por la comunidad de software libre https://github.com/MycroftAI.

6.5. La programación automatizada (Automated programming)

La programación automatizada, o síntesis de programas, es un área de la inteligencia artificial que busca crear sistemas capaces de generar código de software automáticamente a partir de especificaciones o ejemplos dados. Este enfoque tiene como meta desarrollar herramientas que puedan escribir programas sin intervención humana directa en el proceso de codificación. (Paiva, 2024).

Ilustración 83. Mycroft. Fuente: Mycroft.

Aspectos Clave de la Programación Automatizada

Síntesis del Programa

- Generación Automática de Programas: Consiste en crear código que cumpla con especificaciones dadas, como ejemplos de entrada-salida o descripciones en lenguaje natural. Las técnicas empleadas incluyen la síntesis basada en restricciones, la búsqueda exhaustiva y la síntesis de programas utilizando redes neuronales.

Inducción al Programa

- Aprendizaje a Partir de Ejemplos: Se refiere a la capacidad de aprender a programar a partir de un conjunto de ejemplos, imitando el aprendizaje humano. Las técnicas usadas en este proceso incluyen la generación de bosquejos de programas, la transformación de programas y la inducción de programas a través de redes neuronales.

Lenguajes Específicos del Dominio

- Creación de Lenguajes Especializados: Desarrollar lenguajes diseñados para aplicaciones específicas que faciliten la generación de código. Ejemplos de esto son expresiones regulares, SQL y plantillas HTML.

Asistentes Inteligentes

- Herramientas para Programadores: Utilización de herramientas impulsadas por IA para asistir a los programadores humanos en la automatización de tareas repetitivas, ofrecer sugerencias de código y asistencia en la depuración. Esto incluye características como la auto completación de código, la generación de código a partir de comentarios y la refactorización automatizada.

Aplicaciones de la Programación Automatizada

Ingeniería de Software: Automatización de tareas repetitivas de programación, generación de código estándar y apoyo en el mantenimiento de software.

Ciencia de Datos: Creación automática de flujos de procesamiento de datos y código analítico basado en especificaciones de alto nivel.

Educación: Provisión de entornos interactivos que facilitan la enseñanza de conceptos de programación y ofrecen retroalimentación.

Programación para Usuarios Finales: Permitir a personas sin experiencia en programación desarrollar aplicaciones simples mediante interfaces intuitivas y ejemplos prediseñados. (Paiva, 2024).

```python
from transformers import GPT2LMHeadModel, GPT2Tokenizer

# Cargar el modelo y el tokenizador preentrenados
model_name = 'gpt2'  # Usar 'gpt2' como ejemplo; para GPT-3 se requeriría un API diferent
model = GPT2LMHeadModel.from_pretrained(model_name)
tokenizer = GPT2Tokenizer.from_pretrained(model_name)

# Función para generar código
def generate_code(description, max_length=100):
    # Tokenizar la descripción de la tarea
    inputs = tokenizer.encode(description, return_tensors='pt')

    # Generar código usando el modelo
    outputs = model.generate(inputs, max_length=max_length, num_return_sequences=1)

    # Decodificar la secuencia de salida en código legible
    code = tokenizer.decode(outputs[0], skip_special_tokens=True)

    return code

# Loop principal
while True:
    # Obtener la descripción de la tarea de programación del usuario
    description = input("Describe la tarea de programación (o 'salir' para terminar): ")
    if description.lower() == 'salir':
        break

    # Generar código
    generated_code = generate_code(description)
```

Ilustración 85. Hugging Face Transformers con phyton. Fuente: Los autores

Los retos de la programación automatizada abarcan la gestión de especificaciones complejas, la producción de código que sea tanto eficiente como fácil de entender, y la garantía de que el código generado sea correcto y seguro. La investigación actual busca superar estos obstáculos y hacer que la programación automatizada sea más práctica y de amplio uso. Aunque sigue siendo un campo en desarrollo, esta tecnología tiene el potencial de aumentar considerablemente la productividad de los programadores, democratizar el proceso de desarrollo de software y habilitar nuevas aplicaciones de inteligencia artificial en la creación y modificación de códigos. (Paiva, 2024).

6.6. Natural Languaje Processing (NLP)

El procesamiento del lenguaje natural (PLN) es una especialidad de la inteligencia artificial que se dedica a facilitar la comunicación entre las computadoras y los seres humanos utilizando el lenguaje natural. Su meta principal es permitir que las máquinas entiendan, interpreten y produzcan el lenguaje humano de manera que sea relevante y funcional. (Paiva, 2024).

Ilustración 86. NLP. Fuente: Chatgpt4o.

Componentes esenciales del procesamiento del lenguaje natural (PLN):

1. Comprensión del Lenguaje Natural (NLU): Se enfoca en descifrar el significado detrás del texto. Incluye actividades como el análisis de sentimientos, el reconocimiento de entidades y la identificación de intenciones.

2. Generación de Lenguaje Natural (NLG): Consiste en crear texto que sea coherente y relevante en contexto, basándose en entradas específicas. Se utiliza en aplicaciones como chatbots, generación automática de informes y creación de contenido.

3. Reconocimiento de Voz: Transforma el lenguaje hablado en texto, siendo crucial para sistemas activados por voz y asistentes virtuales.

4. Clasificación de Texto: Asigna etiquetas predefinidas a datos textuales, lo cual es útil para tareas como la identificación de correos electrónicos no deseados y el análisis de opiniones.

5. Traducción Automática: Permite traducir texto de un idioma a otro sin necesidad de intervención humana, como en herramientas como Google Translate.

Técnicas y enfoques

Métodos Estadísticos: Inicialmente, los sistemas de procesamiento del lenguaje natural (PLN) dependían de reglas manuales complejas. Con los avances en aprendizaje automático, los métodos estadísticos han ganado relevancia, permitiendo a los sistemas aprender de los datos de manera más eficiente.

Aprendizaje Profundo: La PNL contemporánea utiliza ampliamente técnicas de aprendizaje profundo, especialmente redes neuronales, para manejar y analizar grandes cantidades de datos textuales. Este enfoque ha mejorado notablemente la precisión y la eficacia de las aplicaciones de PLN.

Técnicas de Preprocesamiento: Los procedimientos comunes de preprocesamiento incluyen la tokenización (dividir el texto en palabras o frases), la derivación (reducir las palabras a su forma base) y la eliminación de palabras vacías (remover palabras frecuentes que aportan poco significado).

Aplicaciones de la PNL

Servicio al cliente: Implementación de chatbots y asistentes virtuales que entienden y responden a las preguntas de los usuarios.

Atención médica: Análisis de registros y notas clínicas para extraer datos relevantes y mejorar la calidad de la atención.

Monitoreo de redes sociales: Evaluación de las percepciones del público sobre marcas o productos a través del análisis de datos en redes sociales.

Recomendación de contenido: Ofrecimiento de sugerencias personalizadas de contenido según las preferencias y comportamientos del usuario.

Desafíos en la PNL

Aunque se han logrado avances significativos, la PNL enfrenta varios retos, tales como:

Ambigüedad: El lenguaje natural puede ser ambiguo, lo que requiere una comprensión precisa del contexto para una interpretación correcta.

Variabilidad: Los distintos dialectos, jergas y usos coloquiales pueden dificultar el procesamiento del lenguaje.

Calidad de los datos: La eficacia de los modelos de PNL depende de datos de alta calidad, cuya obtención puede ser costosa y laboriosa.

```python
from transformers import pipeline

# Cargar el modelo preentrenado de clasificación de texto
classifier = pipeline("text-classification", model="distilbert-base-uncased-finetuned-sst-

# Función para la clasificación de texto
def classify_text(text):
    # Obtener la predicción de clasificación del modelo
    result = classifier(text)
    # Extraer la etiqueta de clasificación y la puntuación
    label = result[0]['label']
    score = result[0]['score']
    return label, score

# Loop principal
while True:
    # Obtener texto de entrada del usuario
    user_input = input("Ingresa el texto para clasificar (o 'salir' para terminar): ")
    if user_input.lower() == 'salir':
        break

    # Clasificar el texto
    classification, confidence = classify_text(user_input)

    # Mostrar la clasificación al usuario
    print(f"\nClasificación: {classification}, Confianza: {confidence:.2f}\n")
```

Ilustración 87.NLP. Fuente: Los autores.

El procesamiento del lenguaje natural (PNL) en inteligencia artificial abarca tareas como el análisis de sentimientos, la traducción automática y la clasificación de textos. A continuación, se proporciona un pseudocódigo para un algoritmo de análisis de sentimientos y una implementación básica en Python utilizando la biblioteca Hugging Face Transformers. (Surdeanu, 2024).

En este ejemplo, usamos la función `pipeline` de la biblioteca Hugging Face Transformers para cargar un modelo preentrenado para clasificación de texto. El modelo específico, `distilbert-base-uncased-finetuned-sst-2-english`, está optimizado para análisis de sentimientos.

Función para Clasificar Texto: La función `classify_text` toma un texto como entrada, lo procesa con el modelo de clasificación de texto y devuelve la etiqueta de clasificación junto con la puntuación de confianza (Surdeanu, 2024).

Bucle Principal: El programa solicita al usuario que ingrese un texto. Si el usuario escribe 'salir', el programa finaliza. De lo contrario, se llama a la función `classify_text` para analizar el texto proporcionado y se muestra el resultado al usuario. Este ejemplo sirve como base para implementar un algoritmo de clasificación de texto utilizando modelos de NLP disponibles en código abierto. Hugging Face Transformers ofrece una amplia variedad de modelos y funciones que pueden aplicarse a diversas tareas de procesamiento del lenguaje natural. Se puede visitar . https://huggingface.co/welcome para interactuar.

6.7. La representación del conocimiento (Knowledge representation)

La representación del conocimiento en inteligencia artificial (IA) se refiere a cómo las máquinas organizan, almacenan y usan la información para comprender y razonar sobre el mundo. Este proceso es fundamental porque permite que los sistemas de IA realicen tareas complejas, aprendan de la información y tomen decisiones informadas.

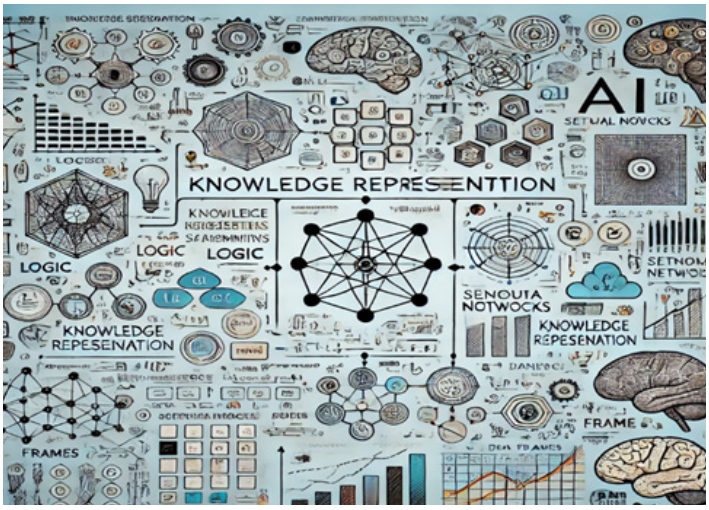

Ilustración 88. La representación del conocimiento. Fuente: Chatgpt4o.

1. Tipos de Conocimiento:

- **Conocimiento Declarativo:** Información factual que puede ser expresada explícitamente, como "La Luna es un satélite de la Tierra".

- **Conocimiento Procedimental:** Información sobre cómo realizar tareas o procedimientos, como algoritmos para resolver problemas.

- **Conocimiento Estructural:** Información sobre las relaciones entre diferentes conceptos o entidades.

- **Meta Conocimiento:** Conocimiento sobre cómo manejar otros tipos de conocimiento, facilitando su uso y comprensión.

```python
from rdflib import Graph, Literal, RDF, URIRef, Namespace

# Crear un nuevo grafo RDF
g = Graph()

# Definir un namespace
ex = Namespace("http://example.org/")

# Añadir triples al grafo
g.add((ex.Perro, RDF.type, ex.Animal))
g.add((ex.Perro, ex.hace, Literal("Ladrar")))
g.add((ex.Gato, RDF.type, ex.Animal))
g.add((ex.Gato, ex.hace, Literal("Maullar")))

# Definir una consulta SPARQL
query = """
PREFIX ex: <http://example.org/>

SELECT ?animal ?sound
WHERE {
    ?animal ex:hace ?sound .
}
"""

# Ejecutar la consulta y mostrar los resultados
for row in g.query(query):
    print(f"{row.animal} hace {row.sound}")
```

Ilustración 89. Pseudocódigo en phyton definición de un grafo RDF y realizar consultas SPARQL. Fuente: Los autores

- **Conocimiento Heurístico:** Reglas generales o principios basados en la experiencia que guían la solución de problemas.

2. Técnicas de Representación:

- **Redes Semánticas:** Diagramas que representan el conocimiento con nodos (conceptos) y aristas (relaciones).

- **Marcos:** Estructuras de datos que definen un concepto con sus atributos y valores asociados.

- **Reglas de Producción:** Condiciones tipo "si-entonces" que guían las acciones en función de ciertas condiciones.

- **Ontologías:** Modelos formales que representan conceptos y sus relaciones en un dominio específico, facilitando el intercambio y reutilización del conocimiento.

3. Razonamiento:

- **Deductivo:** Derivar conclusiones específicas a partir de principios generales.
- **Inductivo:** Generalizar a partir de observaciones específicas.
- **Abductivo:** Inferir la mejor explicación posible para un conjunto de observaciones.

Aplicaciones

La representación del conocimiento es crucial en varias áreas de IA, tales como:

- **Procesamiento del Lenguaje Natural (PNL):** Para entender y generar lenguaje humano, se necesita una representación estructurada del conocimiento.

- **Sistemas Expertos:** Imita la toma de decisiones humana en áreas específicas usando representación del conocimiento para almacenar y razonar información.

- **Robótica:** Los robots utilizan la representación del conocimiento para interpretar su entorno, planificar y tomar decisiones basadas en datos sensoriales.

Desafíos

Complejidad: Representar el conocimiento de manera precisa puede ser complejo debido a la naturaleza rica y ambigua del lenguaje.

Escalabilidad: A medida que se acumula más conocimiento, gestionar y actualizar las bases de conocimiento se convierte en un reto.

Integración del Aprendizaje: Combinar la representación del conocimiento con técnicas de aprendizaje para adaptarse a nueva información presenta desafíos significativos.

6.8. Las estrategias de búsqueda y resolución de problemas

En inteligencia artificial (IA), los conceptos clave se enfocan en cómo los agentes resuelven problemas complejos mediante la exploración de diferentes estados y acciones. Un problema se caracteriza por su estado inicial, el estado deseado y las acciones disponibles para mover entre estos estados. El espacio de estados comprende todas las posibles configuraciones del problema. Para resolver problemas de manera efectiva, es crucial entender bien la estructura del problema, incluyendo qué define una solución y las reglas que dictan cómo se realizan

las transiciones de estado. Las estrategias de búsqueda no informadas funcionan sin información adicional más allá de la definición del problema. Un ejemplo de algoritmo de búsqueda no informada es la búsqueda en amplitud (BFS), que examina todos los nodos en el nivel actual de profundidad antes de proceder a los nodos del siguiente nivel. (Surdeanu, 2024).

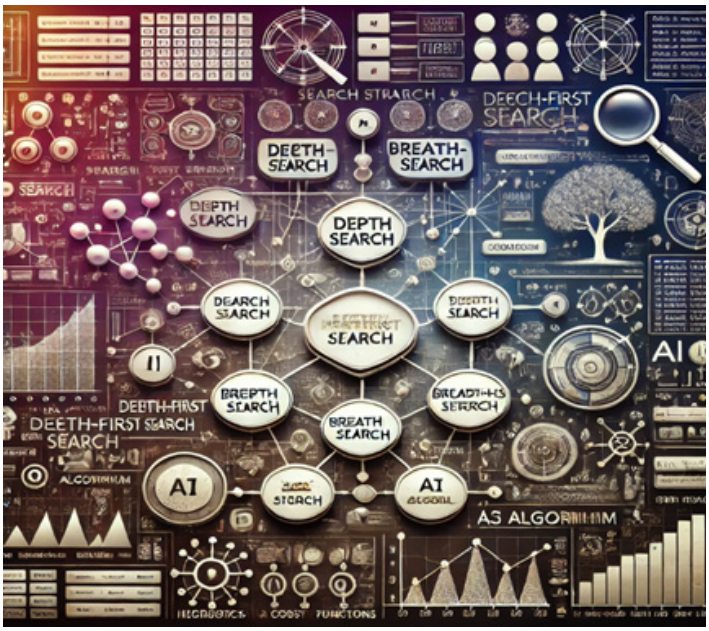

Ilustración 88. La representación del conocimiento. Fuente: Chatgpt4o.

- Búsqueda primero en profundidad (DFS): explora una rama lo más abajo posible antes de retroceder.

- Búsqueda de costo uniforme: expande el nodo con el costo de ruta más bajo.

- Estrategias de búsqueda informada: estas estrategias utilizan información adicional (heurística) para guiar el proceso de búsqueda de manera más eficiente. Ejemplos incluyen:

- A Búsqueda: Combina el costo para llegar a un nodo y un costo estimado para llegar a la meta desde ese nodo.

- Búsqueda codiciosa de mejor primero: expande el nodo que parece estar más cerca del objetivo según una heurística.

VII. Conclusiones

- Es menester que estos conceptos sean socializados con todo el mundo ya que la inteligencia artificial no solo es Chat GPT, y se necesita partir desde un desarrollo desde cero combinando cualquier tipo de software de preferencia con software de código abierto y al momento con hardware de nvidia ya que realiza procesamientos en paralelo.

- En la antigüedad la frase que todos debían programar era muy usada en la educación, en la actualidad se ha mejorado ya que el programar te llevará lejos, sin embargo, el programar con inteligencia artificial no tiene límites.

- Si la educación de nivel inicial, medio y superior, adopta estos conceptos y los profundiza, cualquier país estará en pocos años liderando esta rama en la ciencia en el mundo, en pocas palabras no habría fronteras para las personas dedicadas a esta rama ya que puede desarrollar procesos en todas las áreas existentes del conocimiento.

- El uso de software de código abierto en inteligencia artificial (IA) ha democratizado el acceso a herramientas y conocimientos avanzados. Cualquier persona con interés y acceso a internet puede aprender, experimentar y contribuir al campo de la IA sin incurrir en altos costos de licencias.

- El código abierto es una herramienta educativa poderosa. Los profesores y estudiantes de nivel primarios, secundarios y universitarios, pueden estudiar y aprender de los mejores ejemplos y prácticas de la industria directamente del código. Además, muchos proyectos de código abierto vienen con documentación, tutoriales y comunidades de soporte que facilitan el proceso de aprendizaje.

- El acceso abierto a herramientas de IA promueve una mayor discusión sobre la ética y el impacto social de la inteligencia artificial. Los profesores y estudiantes de nivel primarios, secundarios y universitarios, tienen la responsabilidad de considerar las implicaciones éticas de sus contribuciones y trabajar hacia el desarrollo de IA que beneficie a la sociedad de manera equitativa y justa.

- Muchas industrias están adoptando soluciones de código abierto para integrar IA en sus operaciones. La adopción de estas herramientas no solo reduce costos, sino que también permite a las empresas mantenerse competitivas mediante la implementación de tecnología de punta, lo que brindará un trabajo estable al implementar las bases de la programación en inteligencia artificial a la educación en todas las áreas de las ciencias.

VIII. Recomendaciones

- Abordar tanto los conceptos teóricos como las aplicaciones prácticas de la inteligencia artificial en la educación, proporcionando estrategias innovadoras de desarrollo de inteligencia artificial y no solo de un uso de aplicaciones ya hechas.

- Adoptar estos conocimientos desde el inicio de una vida estudiantil, ya que de esa manera el mundo avanzará a pasos agigantados en todas las áreas de las ciencias.

- Proporcionar consejos prácticos y accesibles para los educadores que desean incorporara a la inteligencia artificial en sus prácticas de enseñanza.

- Aprovecha los numerosos ejemplos y tutoriales disponibles en línea. Sitios como Kaggle, GitHub y los propios repositorios de las bibliotecas suelen tener notebooks y ejemplos que puedes seguir para entender cómo se aplican las técnicas de IA a problemas reales.

- Busca compañeros de estudio o mentores que compartan tus intereses. Colaborar con otros puede acelerar tu aprendizaje y brindarte nuevas perspectivas sobre los problemas que enfrentas.

VII. Bibliografía

Bibliografía

Abadi, M. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems.* Recuperado el Enero de 2021, de https://research.google/pubs/pub45166/

Asencio, C. (2023). Simulación y control: el secreto detrás de los robots seguidores de línea destreza. *EcoSur Innovación Tecnología y Desarrollo Sostenible de América Latina*, 1-19.

Bakker, N. (2023). Effectiveness and cost-effectiveness of a multimodal, physiotherapist-led, vocational intervention in people with inflammatory arthritis: study protocol of the Physiotherapy WORKs trial. *BMC Rheumatology*, 1-13.

Bastidas, D. (2024). Paradigmas para la enseñanza de inteligencia artificial en la educación ecuatoriana ok. *Pentaciencias* , 49-56.

Beyaz, A. (13 de 11 de 2023). *Sugar Beet Seed Classifcation for Production Quality Improvement.* Obtenido de Sugar Beet Seed Classifcation for Production Quality Improvement: https://www.researchgate.net/publication/379522170_Sugar_Beet_Seed_Classification_for_Production_Quality_Improvement_by_Using_YOLO_and_NVIDIA_Artificial_Intelligence_Boards

Bharathi, A. (2023). A Comparative Analysis of PCA and IA for Assortment of Features in Hyper Spectral Images. *International Conference on Power Energy, Environment & Intelligent Control (PEEIC)*, 1674-1679.

Bolpur. (2024). A Study on the Detection and Classification of Lung Cancer using Ensemble Learning and Transfer Learning of Artificial Intelligence. *Journal of Emerging Technologies and Innovative Research (JETIR)*, 8-21.

Coloma, J. (2020). Inteligencia artificial, sistemas inteligentes, agentes inteligentes. *Recimundo*, 16-30.

Díaz, M. (2024). AI Hallucination in the Wake of GenAI. *AMPLIFY: ANTICIPATE, INNOVATE, TRANSFORM*, 32-37.

Diaz, M. P. (2024). AI Hallucination in the Wake of GenAI. *AIHALLUCINATIONINTHEWAKEOFGENA*, 32-37.

Ghosh, S. (2024). Comparing Regular Random Forest Model with Weighted Random Forest Model for Classification Problem. International Journal of Statistics and Applications 2024, 7-12.

Gutierrez, E. (2023). Chat GPT en el espacio antropológico . Revista de materialismo filosófico, 55-63.

Infosalus. (2019). La inteligencia artificial podría predecir la muerte prematura. Infosalus , 50-62.

Jetawat, A. (2024). Predicting Lung Cancer with K-Nearest Neighbors (KNN): A Computational Approach. Indian Journal of Science and Techno, 2199-2206.

Kollasch, F. (2024). UltraPINK - New possibilities to explore Self-Organizing Kohonen. LicenseCC BY-NC-ND 4.0, 1-4.

Lee, D. (2024). One-shot learning for seismic velocity estimation from well-logging data. Conference image 2024 , Houston , 1-8.

Leoshchenko. (2024). USING MODULAR NEURAL NETWORKS AND MACHINE LEARNING. Radio Electronics, Computer Science, Control, 71-81.

Enero de 2021, de https://www.luisllamas.es/protocolos-de-comunicacion-para-iot/

Ma, Z. (2024). DB-RNN: An RNN for Precipitation Nowcasting Deblurring. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 1-16.

Markidis, S. (11 de 03 de 2018). NVIDIA Tensor Core Programmability, Performance & Precision. Obtenido de NVIDIA Tensor Core Programmability, Performance & Precision: https://www.researchgate.net/publication/323722776_NVIDIA_Tensor_Core_Programmability_Performance_Precision

Matich, J. D. (2001). Redes Neuronales, Conceptos básicos y aplicaciones. Rosario.

Meléndez, N. (2023). METODOLOGÍA DE EVALUACIÓN CUALITATIVA DE ENSAYOS EN EDUCACIÓN SUPERIOR UTILIZANDO INTELIGENCIA ARTIFICAL (IA): MODELOS LINGUISTICOS AVANZADOS (LLM). Centro de Estudios de Estrategias Digitales (CEED), 1-8.

Merino, E. (2024). La era pedagógica de la IA. Obtenido de La era pedagógica de la IA: https://www.researchgate.net/publication/382542651_La_era_pedagogica_de_la_IA

Moor, J. (2006). The Dartmouth College Artificial Intelligence Conference: The Next Fifty Years. Recuperado el 06 de 04 de 2023, de The Dartmouth College Artificial Intelligence Conference: The Next Fifty Years.: https://www.researchgate.net/publication/220605256_The_Dartmouth_College_Artificial_Intelligence_Conference_The_Next_Fifty_Years

Moreira, F. (11 de 2023). Use of Machine Learning Models of the “Transformers” Type in the Construction of Services in a. Obtenido de Use of Machine Learning Models of the “Transformers” Type in the Construction of Services in a: https://www.researchgate.net/publication/375609782_Use_of_Machine_Learning_Models_of_the_Transformers_Type_in_the_Construction_of_Services_in_a

Muller, A. (2016). Introduction to Machine Learnign with Pyton. (D. Schanafelt, Ed.) O’Reilly Media. Obtenido de https://www.pdfdrive.com/introduction-to-machine-learning-with-python-e58337749.html

Muñoz, G. (mayo de 2016). Deep Learning con Tensorflow. Obtenido de https://es.scribd.com/document/398956166/seminario-tensorflowGabi-pdf

Newell, A. (1958). Chess-Playing_Programs_And_The_Problem_Of_Complexity. Obtenido de Chess Playing Programs And The Problem Of Complexity: https://bitsavers.org/pdf/rand/ipl/P-1319_

Chess-Playing_Programs_And_The_Problem_Of_Complexity_Sep58.pdf

Nicolini, C. (2024). Hopfield Networks for Asset Allocation. LicenseCC BY 4.0, 1-12.

Okoye, K. (2024). Regression Analysis in R: Linear Regression and Logistic Regression. R programming.

Paiva, J. (2024). Clustering source code from automated assessment of programming assignments. International Journal of Data Science and Analytics, 1-13.

Perez, B. (2024). Arquitecturas para Deteccion de Anomal ´ ´ıas: Fusion de GAN, U-Net y Transformers. Comité Español de Automática , 1-6.

Petrenko, S. (2023). Analyzing Biomedical Datasets with Symbolic Tree Adaptive. LicenseCC BY 4.0, 1-25.

Reinders, J. (10 de 2023). Migrating CUDA Code. Obtenido de Migrating CUDA Code: https://www.researchgate.net/publication/374436868_Migrating_CUDA_Code

Rollings, A. &. (2003). Andrew Rollings and Ernest Adams on Game Design. Indianapolis: New Riders.

Rubin, K. (2013). O´Reilly Media, Inc. Recuperado el Enero de 2021, de Essential Scrum: https://learning.oreilly.com/library/view/essential-scrum-a/9780321700407/copyright.html

Sayin, A. (2023). Automatic item generation for non-verbal reasoning itemsAutomatic item generation for non-verbal reasoning items. International Journal of Assessment Tools in Education, 131-147.

Sharma, P. (2024). Robust GAN-Based CNN Model as Generative AI Application for Deepfake Detection. EAI Endorsed Transactions on Internet of Things, 1-8.

Shoeybi, M. (13 de 03 de 2020). Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. Obtenido de Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism: https://arxiv.org/abs/1909.08053

Simeone, O. (2018). A very brief introduction to machine learning with applications to communication systems. Standford.

Sinaga, T. (2024). Penerapan Algoritma Naive Bayes dalam Pemrosesan Bahasa Alamiah. 1-8.

Smith, J. (2020). The History of Rendral. New York: Academic Press.

Soto, B. R. (2019). Desarrollo de un prototipo domotico utilizando tecnologìa inalambrica para el ahorro de energia, confort y seguridad. Universidad Populae del Cesar, 17.

Sternberg, A. (2013). Multi-Epoch High-Spectral-Resolution Observations of Neutral Sodium in 14 Type Ia Supernovae. 20-27.

Surdeanu, M. (2024). Using Transformers with the Hugging Face Library. In book: Deep Learning for Natural Language Processing: A Gentle Introduction, 1-10.

Tong, W. (2024). MLPs Learn In-Context. Harvard University, Cambridge, MA 02138, 1-29.

Umapathy, C. (2023). FRAMEWORK IN PROTECTING CYBER ATTACK WITH INTEGRATED RNN AND REINFORCMENT LEARNING. Content uploaded by Chandrasekar Umapathy, 1-8.

Vega, E. (2023). Inteligencia Artificial Generativa e Investigación Científica. Universidad Nacional Federico Villarreal, Perú, 1-10.

Veisi , H. (2024). KuBERT: Central Kurdish BERT Model and Its Application for Sentiment Analysis. University of Kurdistan Hewlêr, 1-34.

Venegas, P. (2023). iA (razonamiento artificioso). Panambí Revista de Investigaciones Artísticas, 3-4.

Wei, D. (2024). An anomaly detection model for multivariate time series with anomaly perception. PeerJ Computer Science, 10-18.

Weizenbaum, J. (1966). A Computer Program For the Study of Natural Language Communication Between Man and Machine. New York: Communications of the ACM.

Winograd, T. (1972). Understanding Natural Language. New York: Academic Press.

Wurzberger, F. (2023). Learning in Deep Radial Basis Function Networks. Entropy, 1-13.

Yu, Y. (2020). Multi-Agent Q-Learning Algorithm for Dynamic Power and Rate Allocation in LoRa Networks. IEEE Explore , 1-5.

Zhang, M. (2024). DRML-Ensemble: drug repurposing method based on feature construction of multi-layer ensemble. Journal of Molecular Modeling, 50-62.

Zhu, H. (2022). IA-Mask R-CNN: Improved Anchor Design Mask R-CNN for Surface Defect Detection of Automotive Engine Parts. Applied Sciences, 2-15.

Paradigmas de la Inteligencia Artificial

La inteligencia artificial es una rama de las ciencias de la computación que se encarga de estudiar mecanismos y metodologías que permitan simular el comportamiento humano en dispositivos de middleware, se encarga de desarrollar herramientas que simulen la inteligencia humana, como consecuencia directa de esto también ayuda a fortalecer el estudio de la mente humana y la forma que el cerebro produce el pensamiento, es una nueva generación de tecnología informática caracterizada no solo por su arquitectura (hardware), sino también por sus capacidades.

ISBN: 978-9942-626-24-0

9 789942 626240

EDICIONES GESICAP