UNIVERSIDAD MAYOR DE SAN ANDRÉS

FACULTAD DE CIENCIAS PURAS Y NATURALES

CARRERA DE INFORMÁTICA

DEEP LEARNING

Universitario(a)s:

- Choque Vega Alex Franco
- Flores Ninachoque Jhoselyn Daniela
- Quisbert Saavedra Levit Adrian

Materia: INF-354 “Inteligencia Artificial”

Docente: Toledo Paz Miguel

Fecha: 26 de octubre de 2024

La Paz - Bolivia

1. Introducción al Deep Learning

El deep learning es un subconjunto del machine learning que utiliza redes neuronales multicapa, llamadas redes neuronales profundas, para simular el complejo poder de toma de decisiones del cerebro humano. Algunas formas de deep learning impulsan la mayoría de las aplicaciones de inteligencia artificial (IA) en nuestra vida actual.

La principal diferencia entre el deep learning y el machine learning es la estructura de la arquitectura de red neuronal subyacente. Los modelos tradicionales de machine learning "no profundos" utilizan redes neuronales simples con una o dos capas computacionales. Los modelos de deep learning utilizan tres o más capas, pero normalmente cientos o miles de capas, para entrenar los modelos.

Mientras que los modelos de aprendizaje supervisado requieren datos de entrada estructurados y etiquetados para obtener resultados precisos, los modelos de deep learning pueden utilizar el aprendizaje no supervisado. Con el aprendizaje no supervisado, los modelos de deep learning pueden extraer las características, los rasgos y las relaciones que necesitan para obtener resultados precisos a partir de datos brutos y no estructurados. Además, estos modelos pueden incluso evaluar y refinar sus resultados para aumentar la precisión.

El deep learning es un aspecto de la ciencia de datos que impulsa muchas aplicaciones y servicios que mejoran la automatización, realizando tareas analíticas y físicas sin intervención humana. Esto permite muchos productos y servicios cotidianos, como asistentes digitales, controles remotos de TV habilitados para voz, detección de fraudes con tarjetas de crédito, automóviles autónomos e IA generativa.

1.1. Historia de la Deep Learning

La historia del deep learning está asociada a la creación de las redes neuronales artificiales. **Walter Pitts** y **Warren McCulloch** fueron quienes, en la década de 1940, dieron a conocer un estudio acerca de cómo el cerebro humano genera patrones de gran complejidad a través de la interconexión de neuronas. Estos científicos establecieron una comparación entre las células cerebrales y la lógica booleana, creando un modelo informático donde la unidad de procesamiento se basa en algoritmos para imitar el comportamiento neuronal.

Partiendo de la investigación de **Pitts** y **McCulloch**, se desarrollaron diversos enfoques de abordaje de las redes neuronales: mientras que una tendencia apuntó a los procesos de la biología, otra se orientó a aplicar estos preceptos al desarrollo de la inteligencia artificial.

El **perceptrón** una neurona artificial ideada por **Frank Rosenblatt** en los ´**50**, la **retropropagación** (**backpropagation**) propuesta por **Paul Werbos** en los ´**70** y el entrenamiento de algoritmos que postuló **Yann LeCun** en los ´**80** contribuyeron al avance de las redes neuronales.

El deep learning, en este contexto, se potenció sobre todo en el **siglo XXI**, cuando se incrementó la potencia de cómputo y crecieron los datos disponibles (apareciendo la noción de **big data**). Un protagonista de esta expansión fue **Geoffrey Hinton**, responsable de arquitecturas y algoritmos como las **máquinas Boltzmann** (**Boltzmann machines**) y las **redes neuronales convolucionales**.

Ya a partir de **2010**, el deep learning se enriqueció con las llamadas **redes neuronales profundas** que multiplicaron las aplicaciones. Se optimizó, de este modo, el procesamiento de lenguaje natural, la visión por computadora y el reconocimiento de patrones para identificar imágenes.

2. Aplicaciones de Deep Learning

2.1. Usos de la Deep Learning

El deep learning es una de las bases de la inteligencia artificial (AI) y el interés actual en el deep learning se debe en parte al auge que tiene ahora la inteligencia artificial. Las técnicas de deep learning han mejorado la capacidad de **clasificar, reconocer, detectar y describir**, en una palabra, entender.

Por ejemplo, el deep learning se utiliza para clasificar imágenes, reconocer el habla, detectar objetos y describir contenido. Sistemas como Siri y Cortana son potenciados, en parte, por el aprendizaje a fondo.

El deep learning es un campo que se encuentra en pleno desarrollo. No obstante, este recurso ya se emplea en numerosos procesos y técnicas.

- **Reconocimiento de imágenes**: Gracias al deep learning, un sistema puede describir escenas y añadir subtítulos a una filmación. El reconocimiento de imágenes también es importante en los vehículos de conducción autónoma que están equipados con cámaras de 360º.
- **Reconocimiento del habla**: Asistentes virtuales como **Siri** y **Alexa;** software como **Skype;** consolas como **Xbox**; y diversos **chatbots** usan deep learning para reconocer patrones de voz de los usuarios, lo que se menciona en inglés como **speech recognition**.
- **Procesamiento del lenguaje natural**: Como una especialización del **data mining**, este tipo de procesamiento puede hallar patrones en reportes de información, contenidos médicos o hasta reclamos de clientes.
- **Sistemas de recomendación**: Plataformas como **Netflix** y **Spotify** ofrecen recomendaciones personalizadas gracias al deep learning ya que aprenden de los intereses y las preferencias mostradas por los usuarios en sus hábitos de navegación.

2.2. Tipos de Modelos Deep Learning

Los algoritmos de deep learning son increíblemente complejos y hay diferentes tipos de redes neuronales para abordar problemas o conjuntos de datos específicos. Aquí hay cinco. Cada uno tiene sus propias ventajas y se presentan aquí aproximadamente en el orden de su desarrollo, con cada modelo sucesivo ajustándose para superar una debilidad de un modelo anterior.

Redes neuronales recurrentes (RNN)

Se utilizan normalmente en aplicaciones de lenguaje natural y reconocimiento de voz, ya que utilizan datos secuenciales o de series temporales. Las RNN se pueden identificar por sus bucles de comentarios. Estos algoritmos de aprendizaje se utilizan principalmente cuando se utilizan datos de series temporales para hacer predicciones sobre resultados futuros. Los casos de uso incluyen predicciones bursátiles o de ventas, o problemas ordinales o temporales, como traducción del idioma, procesamiento del lenguaje natural (PLN), reconocimiento de

voz y leyendas de imágenes. Estas funciones a menudo se incorporan en aplicaciones populares como Siri, búsqueda por voz y Google Translate.

Las RNN utilizan su "memoria", ya que toman información de entradas anteriores para influir en la entrada y la salida actuales. Si bien las redes neuronales profundas tradicionales asumen que las entradas y las salidas son independientes entre sí, la salida de los RNN depende de los elementos anteriores de la secuencia. Mientras que las redes neuronales profundas tradicionales asumen que las entradas y las salidas son independientes entre sí, la salida de las RNN depende de los elementos anteriores dentro de la secuencia.

Las RNN comparten parámetros en cada capa de la red y tienen el mismo parámetro de peso dentro de cada capa de la red, con los pesos ajustados mediante los procesos de retropropagación y descenso de gradiente para facilitar el aprendizaje por refuerzo.

Las RNN utilizan un algoritmo de retropropagación en el tiempo (BPTT) para determinar los gradientes, que es ligeramente diferente de la retropropagación tradicional, ya que es específica para los datos secuenciales. Los principios de la BPTT son los mismos que los de la retropropagación tradicional, en la que el modelo se entrena a sí mismo calculando los errores de su capa de salida a su capa de entrada. BPTT difiere del enfoque tradicional en que BPTT suma errores en cada paso de tiempo, mientras que las redes feedforward no necesitan sumar errores ya que no comparten parámetros en cada capa.

Una ventaja sobre otros tipos de redes neuronales es que las RNN utilizan tanto el procesamiento de datos binarios como la memoria. Las RNN pueden planificar múltiples entradas y producciones de modo que, en lugar de ofrecer un único resultado para una sola entrada, las RMM pueden producir salidas de una a varias, de varias a una o de varias a varias.

También hay opciones dentro de las RNN. Por ejemplo, la red de memoria a corto plazo (LSTM) es superior a las RNN simples al aprender y actuar sobre las dependencias a largo plazo.

Sin embargo, las RNN tienden a enfrentarse a dos problemas básicos, conocidos como gradientes de explosión y gradientes de desaparición. Estos problemas se definen por el tamaño del gradiente, que es la pendiente de la función de pérdida a lo largo de la curva de error.

- Cuando el gradiente *desaparece* y es demasiado pequeño, sigue reduciéndose y se actualizan los parámetros de peso hasta que se vuelven insignificantes, es decir: cero (0). Cuando eso ocurre, el algoritmo ya no está aprendiendo.
- *La explosión de* gradientes sucede cuando el gradiente es demasiado grande, lo que crea un modelo inestable. En este caso, las ponderaciones del modelo crecen demasiado y acabarán representándose como NaN (not a number). Una solución a estos problemas es reducir el número de capas ocultas de la red neuronal, lo que elimina parte de la complejidad de los modelos RNN.

Algunas desventajas finales: las RNN también pueden requerir un largo tiempo de entrenamiento y ser difíciles de usar en grandes conjuntos de datos. La optimización de RNN agrega complejidad cuando tienen muchas capas y parámetros.

Autocodificadores y autocodificadores variacionales (VAE)

El deep learning permitió ir más allá del análisis de datos numéricos, añadiendo el análisis de imágenes, voz y otros tipos de datos complejos. Entre la primera clase de modelos en lograrlo se encuentran los autocodificadores variacionales (VAE). Fueron los primeros modelos de deep learning que se utilizaron ampliamente para generar imágenes y voz realistas, lo que potenció el modelado generativo profundo al facilitar el escalado de los modelos, que es la piedra angular de lo que consideramos la IA generativa.

Los autocodificadores codifican los datos no etiquetados en una representación comprimida y luego los descodifican en su forma original. Los autocodificadores *simples* se utilizaban para diversos fines, incluida la reconstrucción de imágenes corruptas o borrosas. Los autocodificadores *variacionales* añadían la capacidad crucial no solo de reconstruir los datos, sino también de generar variaciones de los datos originales.

Esta capacidad de generar nuevos datos dio lugar a una rápida sucesión de nuevas tecnologías, desde las redes generativas antagónicas (GAN) hasta los modelos de difusión, capaces de producir imágenes cada vez más realistas, aunque falsas. De esta manera, los VAE preparan el escenario para la IA generativa actual.

Los autocodificadores se construyen a partir de bloques de codificadores y decodificadores, una arquitectura que también sustenta los modelos de lenguaje de gran tamaño actuales. Los codificadores comprimen un conjunto de datos en una representación densa, organizando puntos de datos similares más juntos en un espacio abstracto. Los decodificadores toman muestras de este espacio para crear algo nuevo al tiempo que conservan las características más importantes del conjunto de datos.

La mayor ventaja de los autocodificadores es su capacidad para gestionar grandes lotes de datos y mostrar los datos de entrada de forma comprimida, por lo que destacan los aspectos más significativos, como la detección de anomalías y las tareas de clasificación. Esto también acelera la transmisión y reduce los requisitos de almacenamiento. Los autocodificadores se pueden entrenar con datos sin etiquetar para que puedan usarse cuando los datos etiquetados no estén disponibles. Cuando se utiliza el entrenamiento no supervisado, existe una ventaja de ahorro de tiempo: los algoritmos de deep learning aprenden automáticamente y ganan precisión sin necesidad de ingeniería manual de características. Además, los VAE pueden generar nuevos datos de muestra para la generación de texto o imágenes.

Los autocodificadores tienen sus desventajas. El entrenamiento de estructuras profundas o intrincadas puede ser una pérdida de recursos computacionales. Y durante el entrenamiento no supervisado, el modelo puede pasar por alto las propiedades necesarias y, en su lugar, simplemente replicar los datos de entrada. Los autocodificadores también pueden pasar por alto vínculos de datos complejos en datos estructurados, de modo que no identifiquen correctamente las relaciones complejas.

Redes generativas antagónicas (GAN)

Son redes neuronales que se utilizan tanto dentro como fuera de la inteligencia artificial (IA) para crear nuevos datos que se parezcan a los datos de entrenamiento originales. Pueden incluir imágenes que parezcan rostros humanos, pero son generadas, no tomadas de personas reales. La parte "antagónica" del nombre proviene del vaivén entre las dos partes del GAN: un generador y un discriminador.

- El generador crea *algo*: imágenes, vídeo o audio y, a continuación, produce una salida con un toque diferente. Por ejemplo, un caballo puede transformarse en una cebra con cierto grado de precisión. El resultado depende de la entrada y de lo bien entrenadas que estén las capas en el modelo generativo para este caso de uso.

- El discriminador es el adversario, en el que el resultado *generativo* (imagen falsa) se compara con las imágenes *reales del conjunto* de datos. El discriminador trata de distinguir entre las imágenes, el vídeo o el audio reales y falsos.

Las GAN se entrenan solas. El generador crea falsificaciones mientras el discriminador aprende a detectar las diferencias entre las falsificaciones del generador y los ejemplos reales. Cuando el discriminador es capaz de marcar la falsificación, se penaliza al generador. El bucle de retroalimentación continúa hasta que el generador logra producir una salida que el discriminador no puede distinguir.

La principal ventaja de las GAN es la creación de resultados realistas que pueden ser difíciles de distinguir de los originales, lo que a su vez puede utilizarse para entrenar más modelos de machine learning. Configurar un GAN para que aprenda es sencillo, ya que se entrenan utilizando datos sin etiquetar o con un etiquetado menor. Sin embargo, la desventaja potencial es que el generador y el discriminador podrían ir y venir compitiendo durante mucho tiempo, creando un gran drenaje del sistema. Una limitación del entrenamiento es que puede ser necesaria una enorme cantidad de datos de entrada para obtener un resultado satisfactorio. Otro problema potencial es el "colapso de modo", cuando el generador produce un conjunto limitado de salidas en lugar de una variedad más amplia.

Modelo de Difusión

Son modelos generativos que se entrenan utilizando el proceso de difusión directa e inversa de adición progresiva de ruido y eliminación de ruido. Los modelos de difusión generan datos (la mayoría de las veces imágenes) similares a los datos con los que se entrenan, pero luego sobrescriben los datos utilizados para entrenarlos. Añaden gradualmente ruido gaussiano a los datos de entrenamiento hasta hacerlos irreconocibles y, a continuación, aprenden un proceso de "eliminación de ruido" inverso que puede sintetizar la salida (normalmente imágenes) a partir de la entrada de ruido aleatorio.

Un modelo de difusión aprende a minimizar las diferencias de las muestras generadas frente al objetivo deseado. Se cuantifica cualquier discrepancia y se actualizan los parámetros del modelo para minimizar la pérdida, entrenando al modelo para que produzca muestras muy parecidas a los datos de entrenamiento auténticos.

Más allá de la calidad de la imagen, los modelos de difusión tienen la ventaja de no requerir entrenamiento de adversarios, lo que acelera el proceso de aprendizaje y también ofrece un control estricto del proceso. El entrenamiento es más estable que con las GAN y los modelos de difusión no son tan propensos al colapso de modos.

Pero, en comparación con las GAN, los modelos de difusión pueden requerir más recursos informáticos para entrenarse, incluido un mayor ajuste. IBM Research también ha descubierto que esta forma de IA generativa puede ser secuestrada con puertas traseras ocultas, lo que da a los atacantes el control sobre el proceso de creación de imágenes para que los modelos de difusión de IA puedan ser engañados para generar imágenes manipuladas.

Modelos de Transformadores

Combinan una arquitectura de codificador-decodificador con un mecanismo de procesamiento de texto y han revolucionado la forma en que se entrenan los modelos de lenguaje. Un codificador convierte el texto sin procesar, sin anotar, en representaciones conocidas como incrustaciones; el descodificador toma estas incrustaciones junto con los resultados anteriores del modelo y predice sucesivamente cada palabra de una frase.

Mediante adivinanzas para completar espacios en blanco, el codificador aprende cómo se relacionan entre sí las palabras y las frases, construyendo una potente representación del lenguaje sin tener que etiquetar las partes de la oración y otros rasgos gramaticales. Los transformadores, de hecho, se pueden entrenar previamente desde el principio sin una tarea particular en mente. Una vez que se aprenden estas poderosas representaciones, los modelos se pueden especializar, con muchos menos datos, para realizar una tarea solicitada.

Varias innovaciones lo hacen posible. Los transformadores procesan las palabras de una oración simultáneamente, lo que permite procesar el texto en paralelo y acelera el entrenamiento. Las técnicas anteriores, incluidas las redes neuronales recurrentes (RNN), procesaban las palabras una por una. Los transformadores también aprendieron las posiciones de las palabras y sus relaciones: este contexto les permite inferir significados y desambiguar palabras como "eso" en frases largas.

Al eliminar la necesidad de definir una tarea por adelantado, los transformadores hicieron que fuera práctico entrenar previamente los modelos de lenguaje en grandes cantidades de texto sin formato, lo que les permitió crecer dramáticamente en tamaño. Anteriormente, los datos etiquetados se recopilaban para entrenar un modelo en una tarea específica. Con los transformadores, un modelo entrenado con una gran cantidad de datos se puede adaptar a múltiples tareas ajustándose en una pequeña cantidad de datos etiquetados específicos de la tarea.

Hoy en día, los transformadores de lenguaje se utilizan para tareas no generativas, como la clasificación y la extracción de entidades, así como para tareas generativas, como la traducción automática, el resumen y la respuesta a preguntas. Los transformadores han sorprendido a muchas personas con su capacidad para generar diálogos, ensayos y otros contenidos convincentes.

Los transformadores de procesamiento del lenguaje natural (PLN) proporcionan una potencia notable, ya que pueden ejecutarse en paralelo, procesando múltiples partes de una secuencia simultáneamente, lo que acelera enormemente el entrenamiento. Los transformadores también rastrean las dependencias a largo plazo en el texto, lo que les permite entender el contexto general con mayor claridad y crear resultados superiores. Además, los transformadores son más escalables y flexibles para personalizarlos según las tareas.

En cuanto a las limitaciones, debido a su complejidad, los transformadores requieren enormes recursos computacionales y un largo tiempo de entrenamiento. Además, los datos de entrenamiento deben ser precisos, imparciales y abundantes para producir resultados exactos.

3. Frameworks

3.1. Frameworks populares

Los frameworks son herramientas fundamentales en deep learning, ya que permiten a los desarrolladores e investigadores crear, entrenar y probar redes neuronales con mayor facilidad. Los tres frameworks más populares son TensorFlow, Keras y PyTorch.

- **TensorFlow**

  TensorFlow es sin duda el framework de deep learning más reconocido y ampliamente utilizado. Se destaca por su facilidad de aprendizaje y la gran comunidad de soporte que lo respalda. Además, cuenta con una herramienta propia de visualización llamada TensorBoard, que facilita la monitorización y la depuración de modelos de deep learning. La asociación con el stack completo de Google Cloud Platform brinda a TensorFlow la capacidad de desplegar proyectos en la nube de manera eficiente. Estas características, sumadas a su eficiencia y popularidad, lo convierten en una opción sólida para desarrollar proyectos de deep learning.

  Ventajas:

  Amplitud de Herramientas: TensorFlow tiene una amplia gama de herramientas y bibliotecas, incluyendo TensorFlow Extended (TFX) para producción y TensorFlow Lite para dispositivos móviles.

  Compatibilidad con TPUs: TensorFlow permite usar TPUs (unidades de procesamiento tensorial), hardware especializado en Google Cloud, que acelera el entrenamiento de modelos complejos.

  Gran Comunidad y Recursos: Al ser tan popular, existen muchos recursos de aprendizaje, tutoriales y foros de soporte, lo cual facilita su uso y resolución de problemas.

  Escalabilidad: TensorFlow es ideal para aplicaciones en producción y escalables. Grandes compañías y aplicaciones en la nube, como Google Cloud y Amazon AWS, soportan TensorFlow.

  Desventajas:

  Curva de Aprendizaje Empinada: TensorFlow es más complejo y menos intuitivo, lo que puede dificultar su aprendizaje y uso para principiantes.

  Sintaxis Compleja: En comparación con PyTorch, TensorFlow tiene una sintaxis más complicada y menos natural, lo que puede hacer que el desarrollo sea más lento.

  Desventajas de Versiones Previas: En sus versiones iniciales, TensorFlow tenía problemas de incompatibilidad entre versiones. Aunque se ha mejorado mucho, todavía puede presentar algunas limitaciones en compatibilidad.

- **Keras:**

  Keras es una API basada en Python para redes neuronales de alto nivel. Este kit de herramientas de código abierto se basa en CNTK, TensorFlow y Theano y se puede utilizar para experimentar rápidamente con redes neuronales profundas. Keras hace hincapié en la modularidad, la facilidad de uso y la extensibilidad. No maneja

cálculos de bajo nivel; En su lugar, los pasa al backend, que es una biblioteca separada.

A mediados de 2017, se adoptó Keras y se incorporó a TensorFlow. El tf. El módulo Keras da acceso a los usuarios a él. Por otro lado, la biblioteca Keras puede seguir funcionando de forma independiente.
Por ejemplo, en el código siguiente se muestra cómo definir una red convolucional simple en Keras.

Ventajas:

Fácil de Usar y Aprender: Keras es conocida por su simplicidad y claridad, lo que la convierte en una excelente opción para principiantes y para prototipos rápidos.

Tiempo de Desarrollo Rápido: Debido a su API de alto nivel, Keras permite construir modelos complejos con pocas líneas de código.

Integración en TensorFlow: Al estar integrada en TensorFlow, Keras puede aprovechar las ventajas de la escalabilidad y el soporte de TensorFlow.

Desventajas:

Menor Flexibilidad: Al estar diseñada para simplificar el proceso de desarrollo, Keras no es tan flexible ni tan potente como otros frameworks como PyTorch para tareas de investigación avanzadas.

Rendimiento Limitado: Keras puede ser menos eficiente en términos de tiempo de entrenamiento y rendimiento en producción, especialmente en comparación con frameworks de bajo nivel.

Limitada para Modelos Muy Complejos: Aunque es adecuada para muchas aplicaciones, algunos tipos de modelos avanzados pueden ser más difíciles de implementar exclusivamente con Keras.

- **PyTorch**

PyTorch, desarrollado por el grupo de investigación FAIR de Facebook, ha ganado popularidad en la industria y es utilizado por empresas como Salesforce, Facebook y Twitter. Se destaca por su facilidad de uso y por tener una comunidad de soporte sólida, especialmente en el ámbito de la investigación. PyTorch permite un desarrollo rápido y flexible, lo que lo convierte en una elección popular entre los investigadores y aquellos que buscan experimentar con modelos de deep learning. Sin embargo, su prevalencia en aplicaciones de producción es algo más limitada.

Ventajas:

Flexible e Intuitivo: PyTorch es muy popular en la investigación debido a su diseño dinámico, que se asemeja a la programación estándar en Python. La estructura de código es fácil de leer y permite el uso de depuradores comunes de Python.

Desarrollo en Tiempo Real: A diferencia de TensorFlow, PyTorch permite la creación de grafos computacionales dinámicos. Esto significa que puedes cambiar la estructura del modelo en tiempo real, lo cual es útil para modelos experimentales.

Comunidad Activa y Expansión Rápida: PyTorch ha desarrollado una comunidad activa y crece rápidamente. Su popularidad está aumentando también en la industria.

Desventajas:

Menor Soporte en Producción: En comparación con TensorFlow, PyTorch tiene menos herramientas de producción, aunque esto ha mejorado con TorchServe y su soporte en AWS.

Incompatibilidad con TPUs: PyTorch no tiene un soporte oficial de TPUs, lo que puede limitar su uso en aplicaciones que requieran esta clase de hardware.

Menor Documentación y Recursos que TensorFlow: Aunque la comunidad ha crecido, TensorFlow todavía tiene una mayor cantidad de recursos y documentación, lo que puede ser una ventaja para los principiantes.

3.2. Comparación de Frameworks

En conclusión, la elección del framework dependerá de las necesidades del proyecto. TensorFlow es ideal para proyectos en producción y de alto rendimiento, Keras es excelente para el desarrollo rápido y el prototipado, mientras que PyTorch es una opción flexible y ampliamente adoptada en el campo de la investigación.

4. Desafios y avances recientes en Deep Learning

4.1. Desafios actuales

Deep Learning ha evolucionado rápidamente en los últimos años, pero aún enfrenta varios desafíos importantes que limitan su aplicación y efectividad en diversos campos. Aquí te comento algunos de los desafíos más relevantes en la actualidad:

- **Dependencia de Datos Grandes y Etiquetados**

  Los modelos de Deep Learning suelen requerir grandes volúmenes de datos etiquetados, lo cual no siempre es viable, especialmente en dominios donde la recolección de datos es costosa o difícil (por ejemplo, en la medicina). Esto ha impulsado la investigación en técnicas como el aprendizaje auto-supervisado y el aprendizaje de pocos disparos (few-shot learning).

- **Consumo computacional elevado**

  Requisitos de Hardware: Entrenar modelos complejos, como redes neuronales profundas, requiere hardware especializado (GPUs y TPUs). Este tipo de hardware es caro y consume mucha energía, lo que hace que entrenar y desplegar estos modelos sea costoso.

  Energía y Sostenibilidad: El entrenamiento de modelos grandes consume cantidades significativas de electricidad. Esto no solo aumenta el costo, sino que también tiene un impacto ambiental. Por ejemplo, el entrenamiento de modelos como GPT-3 y otros modelos de lenguaje a gran escala emiten toneladas de CO_2.

Accesibilidad: No todas las empresas o investigadores individuales pueden acceder a los recursos necesarios. Esto crea una brecha, donde las organizaciones grandes, como Google, Amazon y Facebook, tienen una ventaja significativa frente a otras debido a su acceso a infraestructuras de computación avanzadas.

- **Interpretabilidad y Explicabilidad**

  Muchos modelos de Deep Learning se consideran "cajas negras" debido a la dificultad para entender cómo toman decisiones. En aplicaciones críticas (como medicina o justicia), es esencial poder explicar las decisiones del modelo. Esto ha dado lugar a técnicas como el aprendizaje interpretable y métodos de explicabilidad (explainable AI).

- **Generalización y Robustez**

  Los modelos de Deep Learning tienden a fallar cuando se enfrentan a datos fuera de su distribución de entrenamiento (out-of-distribution) o a datos con ruido. Aumentar la robustez y mejorar la capacidad de generalización sigue siendo un reto, especialmente para aplicaciones en el mundo real, donde los datos pueden variar mucho.

- **Sesgo y Equidad**

  Los modelos de Deep Learning pueden amplificar sesgos presentes en los datos, lo cual es problemático en aplicaciones sensibles. Eliminar estos sesgos y asegurar que los modelos sean justos y éticos es un gran desafío, especialmente cuando se utilizan en contextos sociales, legales o de contratación.

- **Escalabilidad en el Aprendizaje por Refuerzo**

  En el campo del aprendizaje por refuerzo (reinforcement learning), uno de los retos principales es la escalabilidad para aplicaciones complejas. Entrenar agentes en entornos complejos, como videojuegos o robótica, sigue siendo costoso y lento, y la transferencia de aprendizaje entre diferentes entornos aún está en desarrollo.

- **Seguridad contra Ataques Adversariales**

  Los ataques adversariales, en los que se manipulan ligeramente los datos de entrada para engañar al modelo, son un riesgo importante, especialmente en aplicaciones de seguridad y sistemas autónomos. Desarrollar modelos robustos ante este tipo de ataques es un área de investigación activa.

- **Limitaciones en el Aprendizaje Auto-supervisado y Transferencia de Conocimientos**

  Aunque el aprendizaje auto-supervisado ha mostrado potencial en ciertas áreas (como el procesamiento de lenguaje natural), aún enfrenta desafíos en términos de adaptabilidad y generalización en otras aplicaciones. Igualmente, la transferencia de conocimientos entre diferentes dominios y tareas es limitada.

4.2. Avances recientes y tendencia futuras

- **Modelos mas eficiente y ligeros.**

Se están desarrollando modelos optimizados que son mas pequeños pero igual de precisos como MobileNet o distilBERT. Esto permite su uso en dispositivos mobiles y entonos con recursos limitados

Reducción de Tamaño sin Sacrificar Precisión: Modelos como MobileNet, EfficientNet y DistilBERT están diseñados para mantener una alta precisión con menos parámetros y un tamaño de modelo más pequeño. Estos modelos permiten el uso de deep learning en dispositivos móviles y entornos con limitaciones de recursos, lo cual es ideal para aplicaciones en el "Internet de las Cosas" (IoT).

Técnicas de Compresión y Cuantización: Métodos como la compresión de modelos y la cuantización (reducir la precisión de los pesos y las activaciones) permiten que los modelos sean menos exigentes computacionalmente, facilitando su uso en dispositivos con menos capacidad de procesamiento, como teléfonos y dispositivos IoT.

Reducción de Inferencia: Los modelos más ligeros también aceleran el proceso de inferencia, lo cual es esencial para aplicaciones en tiempo real como el procesamiento de imágenes y el reconocimiento de voz.

- **Aprendizaje auto-supervisado y no supervisado**

  Nuevas técnicas permiten a los modelos aprender a partir de grandes cantidades de datos sin necesidad de etiquetas. Esto reduce la dependencia de datos etiquetados y mejora el rendimiento en tareas donde los datos son escasos

  Auto-supervisión: Esta técnica permite que los modelos se entrenen usando datos sin etiquetas. Un ejemplo clave es el modelo SimCLR de Google, que aprende representaciones a partir de imágenes no etiquetadas. Los modelos auto-supervisados se entrenan identificando patrones en los datos de manera automática, reduciendo la necesidad de grandes volúmenes de datos etiquetados.

  Aprendizaje No Supervisado: Métodos de aprendizaje no supervisado como las redes generativas adversarias (GANs) y los modelos de lenguaje no supervisados han permitido avances en el procesamiento de lenguaje natural, generación de imágenes y otras áreas.

  Impacto en la Escasez de Datos: Estas técnicas ayudan a mejorar el rendimiento en tareas donde los datos etiquetados son escasos o costosos, permitiendo que las aplicaciones de deep

  learning se extiendan a áreas previamente inaccesibles, como idiomas minoritarios o imágenes médicas.

- **Fusión con otras ramas de la IA**

  El depp learning se esta combinando con técnicas de razonamiento simbólico o algoritmos de búsqueda para crear sistemas mas inteligentes, capaces de resolver problemas complejos

  Razonamiento Simbólico y Redes Neuronales: La combinación del deep learning con el razonamiento simbólico permite que los modelos resuelvan problemas de lógica y

comprensión que no se pueden abordar solo con redes neuronales. Esta integración da lugar a sistemas más inteligentes y versátiles.

Meta-aprendizaje: El meta-aprendizaje, o "aprendizaje para aprender", permite que los modelos de deep learning adquieran la capacidad de adaptarse rápidamente a nuevas tareas con pocos datos. Un ejemplo es el modelo MAML (Model-Agnostic Meta-Learning), que aprende patrones en las tareas y se adapta a tareas nuevas con un entrenamiento mínimo.

IA Multimodal: Los modelos multimodales, que combinan texto, imagen, audio y otros tipos de datos, permiten una comprensión más rica y un análisis integral. Un ejemplo reciente es CLIP de OpenAI, que asocia texto con imágenes para crear un modelo que puede entender e interpretar el contexto de manera más profunda.

Futuras Tendencias en Deep Learning

- **Aumento de la Interpretabilidad y Transparencia**

  Nuevas técnicas de interpretación y visualización están ayudando a mejorar la transparencia de los modelos de deep learning. Esto es clave para aplicaciones críticas, como la medicina y las finanzas, donde la explicabilidad de los modelos es esencial para la confianza y el cumplimiento de normativas.

- **Edge AI**

  La capacidad de ejecutar modelos de deep learning directamente en dispositivos edge, como teléfonos, cámaras de seguridad o sensores IoT, se está volviendo más viable gracias a los modelos ligeros. Esto permite procesar datos localmente, mejorando la privacidad, reduciendo el tiempo de latencia y bajando el consumo de ancho de banda.

- **IA Ética y Sostenible**

  La IA sostenible se está convirtiendo en un objetivo clave para reducir el impacto ambiental de los grandes modelos de deep learning. También se están desarrollando normas para garantizar la ética en el desarrollo y el uso de IA, evitando sesgos y protegiendo la privacidad de los usuarios.

En conjunto, estos avances y desafíos definen el estado actual del deep learning, orientando su desarrollo hacia un futuro más accesible, eficiente y ético.

5. Fundamentos de Deep Learning

El Deep Learning es una rama del aprendizaje automático basada en redes neuronales profundas. Estas redes emulan el funcionamiento del cerebro humano, permitiendo que las máquinas aprendan de grandes volúmenes de datos, descubran patrones complejos y realicen predicciones.

5.1 Redes Neuronales

Cómo Funciona

Una red neuronal está formada por neuronas artificiales conectadas entre sí. Cada neurona recibe señales de entrada, las procesa mediante una función matemática y envía una señal de salida.

Neurona

Cada neurona tiene conexiones de entrada que representan características de los datos. Cada entrada es multiplicada por un peso específico, que define la importancia de esa entrada para la neurona. Estas señales ponderadas se suman y pasan por una función de activación para generar la salida de la neurona.

Capas

- Capa de entrada: Recibe los datos en crudo (características de los datos de entrada).
- Capas ocultas: Procesan los datos mediante múltiples neuronas y aplican funciones de activación no lineales.
- Capa de salida: Genera el resultado final de la red.

Funciones de Activación

Las funciones de activación introducen no linealidad en la red, permitiendo que aprenda patrones complejos.
- Sigmoide: Transforma los valores en un rango entre 0 y 1.
- ReLU (Rectified Linear Unit): Activa solo valores positivos ($f(x) = \max(0, x)$), útil en redes profundas.
- Tanh: Similar a la sigmoide, pero con un rango de -1 a 1, lo cual permite trabajar mejor con valores negativos.

5.2 Tipos de Redes Neuronales

Redes Neuronales Artificiales (ANN)

Las ANN son redes básicas compuestas por capas de neuronas conectadas de forma densa. Se aplican principalmente en problemas de clasificación y regresión.

Redes Neuronales Convolucionales (CNN)

Las CNN están diseñadas para reconocer patrones espaciales en imágenes mediante operaciones de convolución. Usan filtros para detectar características locales (bordes, texturas) en imágenes, y son efectivas para tareas de visión artificial.

Redes Neuronales Recurrentes (RNN)

Las RNN manejan secuencias de datos mediante conexiones retroalimentadas, lo que les permite conservar información de entradas previas. Son adecuadas para procesar datos secuenciales, como el lenguaje y la música.

5.3 Entrenamiento de Modelos

Función de Pérdida

La función de pérdida mide la diferencia entre la predicción de la red y el valor real. El objetivo del entrenamiento es minimizar esta función de pérdida.

Algoritmos de Optimización

Los algoritmos de optimización ajustan los pesos de la red para reducir la función de pérdida. Uno de los métodos más comunes es el Descenso de Gradiente Estocástico (SGD), que realiza ajustes de manera iterativa sobre lotes de datos.

Overfitting y Underfitting

- Overfitting: Ocurre cuando la red se ajusta demasiado a los datos de entrenamiento, perdiendo capacidad de generalización.
- Underfitting: Sucede cuando la red no es capaz de capturar patrones de los datos, lo que indica un modelo poco entrenado o demasiado simple.

Fundamentos de Deep Learning

5.4 Aplicación Práctica de Redes Neuronales Convolucionales (CNN)

Esta sección describe una aplicación práctica de una Red Neuronal Convolucional (CNN) en un entorno Django para la clasificación de imágenes.
Se implemento este modelo de CNN en el lenguaje de Python usando las bibliotecas TensorFlow y Keras, lo cual nos permitira distinguir entre dos tipos de imágenes (perro y gato):
La estructura del código sigue un flujo de procesamiento dividido en entrenamiento del modelo y predicción sobre nuevas imágenes.

Preparación de los Datos

En este ejemplo, los datos de entrenamiento y prueba están separadas en carpetas dentro de un entorno Django.
La clase ImageDataGenerator de Keras se utiliza para generar lotes de imágenes aumentadas en tiempo real, lo que mejorará generalización del modelo al variar las imágenes (escalado, rotación, zoom y reflejo horizontal).

Las imágenes se redimensionan a 64x64 píxeles, adaptándose al tamaño de entrada requerido por la arquitectura de la CNN y de esa manera todas las imágenes serán convertidas a un mismo formato

Arquitectura de la CNN

La arquitectura de la CNN en este ejemplo consta de múltiples capas, diseñadas para extraer características relevantes de las imágenes:

- *Capa Convolucional (Conv2D)*: Extrae características espaciales relevantes aplicando filtros sobre la imagen.
- *MaxPooling (MaxPool2D)*: Reduce la dimensionalidad espacial, enfocándose en las características más importantes.
- *Capa Densa (Dense)*: Conecta las características extraídas en un vector y permite tomar la decisión final con una capa de salida con activación sigmoide.

Compilación y Entrenamiento del Modelo

Para compilar el modelo, se utilizo el optimizador 'Adam' y la función de pérdida 'binary_crossentropy', adecuada para clasificación binaria.

El modelo es entrenado durante 25 épocas, comparando su rendimiento en los datos de entrenamiento y prueba en cada época.

Por último, el modelo se guarda como un archivo .h5, lo que permite cargarlo posteriormente para realizar predicciones.

Predicción con el Modelo Entrenado

En la fase de predicción, el modelo carga una imagen enviada por el usuario, la redimensiona y normaliza para que coincida con el tamaño de entrada de la red. y de esa manera realiza la predicción asignando una etiqueta basada en un umbral de probabilidad.

Un valor mayor a 0.5 clasifica la imagen como "perro" y menor como "gato". En este ejemplo, los umbrales también pueden ajustarse para mejorar la precisión según los resultados obtenidos. para este ejemplo básico solo se tiene las opciones de: ‘es un perro’ y ‘es un gato’, si quisieramos añadir algun tipo de mensaje más, como por ejemplo: ‘no se identifico como perro ni gato’ podríamos hacerlo pero se deberá entrenar el modelo con aún más imagenes para que sea certero.

Ejemplo de Código

A continuación se presenta el código de ejemplo para la implementación de esta CNN en un entorno Django.

```
from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework import status
import os
from django.conf import settings
```

```
import tensorflow as tf
from keras.preprocessing import image
import numpy as np
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from PIL import Image

class TrainModelView(APIView):
        def post(self, request, *args, **kwargs):
        try:
        path_training = os.path.join(settings.BASE_DIR, 'classifier/data/rnc/training_set')
        path_test = os.path.join(settings.BASE_DIR, 'classifier/data/rnc/test_set')
        train_datagen = ImageDataGenerator(rescale=1./255, shear_range=0.2, zoom_range=0.2,
horizontal_flip=True)
        test_datagen = ImageDataGenerator(rescale=1./255)
        training_set = train_datagen.flow_from_directory(path_training, target_size=(64, 64),
batch_size=32, class_mode='binary')
                test_set = test_datagen.flow_from_directory(path_test, target_size=(64, 64),
batch_size=32, class_mode='binary')
        model = tf.keras.models.Sequential()
                model.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu',
input_shape=[64, 64, 3]))
        model.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))
        model.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu'))
        model.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))
        model.add(tf.keras.layers.Flatten())
        model.add(tf.keras.layers.Dense(units=128, activation='relu'))
        model.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
        model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
        model.fit(training_set, validation_data=test_set, epochs=25)
        model.save(os.path.join(settings.BASE_DIR, 'classifier/data/rnc/models/model.h5'))
        return Response({"message": "Model trained and saved successfully"},
status=status.HTTP_200_OK)
        except Exception as e:
        return Response({"error": str(e)}, status=status.HTTP_500_INTERNAL_SERVER_ERROR)
class RNCPredictionView(APIView):
        def post(self, request, *args, **kwargs):
        cnn = tf.keras.models.load_model(os.path.join(settings.BASE_DIR,
'classifier/data/rnc/models/model.h5'))
        if 'image' not in request.FILES:
        return Response({"error": "No file provided."}, status=status.HTTP_400_BAD_REQUEST)
        img_file = request.FILES['image']
        try:
        img = Image.open(img_file).resize((64, 64))
        test_image = image.img_to_array(img) / 255.0
        test_image = np.expand_dims(test_image, axis=0)
        result = cnn.predict(test_image)
        prediction = 'Es un perro' if result[0][0] > 0.5 else 'Es un gato'
        except Exception as e:
        return Response({"error": "Error during prediction: " + str(e)},
status=status.HTTP_500_INTERNAL_SERVER_ERROR)
        return Response({'prediction': prediction}, status=status.HTTP_200_OK)
```