

Unidad 5 - Modelos generativos

UNR - TUIA - Procesamiento de Imágenes y Visión por Computadora

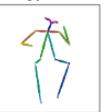
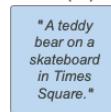
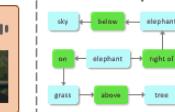
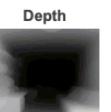
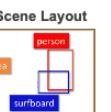
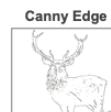
Docente teoría: Juan Pablo Manson

Docente práctica: Lucas Brugé, Constantino Ferrucci

Colab con ejemplos prácticos:

Google Colab


https://colab.research.google.com/drive/1-pgrz-tDb_Rm40YcWQ8E9lmcseWphzlo?usp=sharing

| | | | | | | | | |
|-------------------|---|---|---|---|---|--|---|---|
| Condition Types | Semantic Map | Keypoints | Sketch Map | Text (2D) | Text (3D) | Text (Video) | Audio | Scene Graph |
| |  |  |  |  |  |  |  |  |
| Synthesis Results |  |  |  |  |  |  |  |  |
| | Normal Map | Depth | Scene Layout | Hough Lines | Scribbles | Canny Edge | Brain Signal | Mouse Track |
| Condition Types |  |  |  |  |  |  |  |  |
| |  |  |  |  |  |  |  |  |

Tecnología Central de IA Generativa

Extracto del libro "IA Generativa" - Tom Taulli

Una mirada a los algoritmos clave

De 2004 a 2009, Ian Goodfellow asistió a la Universidad de Stanford, donde estudió ciencias de la computación, con énfasis en inteligencia artificial. Luego obtuvo su doctorado en aprendizaje automático en la Universidad de Montreal. A lo largo de su formación, aprendió de algunos de los mejores investigadores en IA, como Yoshua Bengio y Andrew Ng.

Goodfellow ciertamente puso esto en práctica. Coescribió el influyente libro "Deep Learning". También fue ingeniero en Google, donde ayudó a mejorar aplicaciones como Google Maps.

Pero su avance más importante sucedería por accidente. En 2014, estaba en un pub de Montreal y tuvo una discusión amplia con amigos. ¿El tema? Si el aprendizaje profundo podría crear imágenes únicas.

Debatieron los diferentes enfoques para la IA generativa. Pero la tecnología no funcionaba bien. Las imágenes solían ser borrosas o ridículas.

Para Goodfellow, esto despertó una inspiración. ¿Quizás los conceptos de la teoría de juegos podrían ayudar? En otras palabras, diferentes modelos de aprendizaje profundo esencialmente lucharían entre sí, y esto podría ayudar a crear las imágenes.

No tardó mucho en resolver esto. Cuando llegó a casa, desarrolló un prototipo de un modelo. Se conocería como una red generativa adversaria o GAN, por sus siglas en inglés. Y sí, se convertiría en una de las primeras innovaciones para la IA generativa.

Coescribió un artículo sobre esto llamado "Redes Generativas Adversarias" y lo presentó en la conferencia de Sistemas de Procesamiento de Información Neural (NeurIPS) en Montreal. Esto llevaría a un aumento en la investigación. También habría una explosión de imágenes creativas e incluso muestras de audio en Internet.

Yann LeCun diría que el GAN era "la idea más interesante en los últimos 10 años en Aprendizaje Automático".

Pero hubo consecuencias no deseadas. Los GANs se convertirían en una manera de crear deepfakes.

A pesar de todo, Goodfellow se convertiría rápidamente en una superestrella en la comunidad de IA (actualmente, tiene más de 335,000 seguidores en Twitter). Recibió ofertas multimillonarias para trabajar en varias empresas. De hecho, eventualmente Goodfellow sería conocido como el "GANfather".

Así que en este capítulo, echaremos un vistazo más a fondo a los GANs. Pero también veremos los otros sistemas fundamentales para la IA generativa.

Modelos Generativos vs. Discriminativos

En el capítulo anterior, exploramos una rama de la Inteligencia Artificial conocida como **modelado discriminativo**. Su función principal es la **clasificación de datos**, lo que nos permite realizar predicciones específicas. Por ejemplo, un modelo discriminativo puede

determinar si un correo electrónico es spam o anticipar si un cliente está a punto de cancelar un servicio. En esencia, estos modelos aprenden a *distinguir* entre diferentes categorías o a predecir un valor específico basándose en las características de entrada.

Históricamente, y al menos hasta alrededor de 2022, muchas aplicaciones comerciales de la IA se han centrado predominantemente en el modelado discriminativo. Esta preferencia se debe a la crucial necesidad que tienen las empresas de obtener pronósticos precisos, ya que errores en estas predicciones pueden acarrear consecuencias significativas.

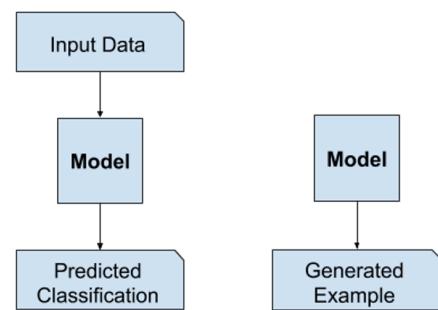
Un caso ilustrativo es el de Southwest Airlines a finales de diciembre de 2022. Los sistemas informáticos de la aerolínea se vieron superados por la magnitud y complejidad del impacto de una severa tormenta invernal que afectó a gran parte de Estados Unidos. Como resultado, Southwest se vio forzada a cancelar más de 16,700 vuelos, lo que se tradujo en pérdidas trimestrales considerables, incluyendo el reembolso de gastos a los clientes y un daño notable a su reputación. Ante esta situación, la dirección de Southwest señaló como prioridad la modernización de sus sistemas de TI, a pesar de que la compañía ya invertía cerca de mil millones de dólares anuales en tecnología. Es probable que Southwest, al igual que muchas otras empresas, explore cómo las tecnologías de IA, y en particular el modelado discriminativo, pueden ayudar a optimizar sus operaciones y prevenir colapsos similares mediante mejores sistemas de predicción y gestión de recursos.

Entonces, ¿cómo se compara el modelado discriminativo con la **IA generativa**? Aunque comparten ciertas similitudes, como la dependencia de grandes volúmenes de datos y el uso frecuente de arquitecturas de aprendizaje profundo, su diferencia fundamental radica en el **resultado que producen**.

Con el **modelado discriminativo**, el objetivo es llegar a una **predicción o clasificación**. Por ejemplo, dada una imagen, el modelo podría predecir si contiene un perro o un gato. Aprende a trazar una frontera entre las distintas clases de datos.

En contraste, con la **IA generativa**, el resultado son **datos completamente nuevos**. En lugar de simplemente clasificar o predecir, un modelo generativo aprende los patrones, las características y la estructura fundamental de los datos de entrenamiento. A partir de este profundo "entendimiento", es capaz de *generar* ejemplos originales que se asemejan a los datos con los que fue entrenado, pero que no son simples copias.

Modelos Discriminativos vs. Generativos



Así es como la IA generativa demuestra una forma de "creatividad". Por ejemplo, si se entrena un modelo generativo con una vasta colección de imágenes de perros, no solo aprenderá a reconocer perros, sino que podrá generar imágenes nuevas y realistas de perros que nunca antes existieron, pero que conservan todos los atributos característicos de un perro. El aspecto central de estos modelos es su capacidad para capturar la "esencia" o la distribución subyacente de los datos de entrada.

En cuanto a los datos, los modelos de IA generativa a menudo trabajan con **datos no estructurados** (como texto, imágenes o audio), aunque también pueden aplicarse a datos etiquetados y estructurados. Frecuentemente, este tipo de tecnología se asocia con el **aprendizaje no supervisado**, donde el modelo identifica patrones y estructuras en los datos sin necesidad de etiquetas explícitas que guíen el proceso. Esta capacidad de aprender de datos no etiquetados es una ventaja considerable, dada la abundancia de este tipo de información en el mundo.

Sin embargo, esta flexibilidad también presenta desafíos. Dado que la IA generativa crea contenido nuevo, la **salida puede ser a menudo inesperada**, lo que dificulta la evaluación directa de su "precisión" en el sentido tradicional. Los resultados pueden, en ocasiones, desviarse significativamente de lo esperado o incluso ser incorrectos o absurdos. Este ha sido el caso de modelos como ChatGPT, que a veces producen información que parece convincente pero que contiene errores factuales, un fenómeno conocido como **"alucinaciones"**.

Este problema de evaluación y la posibilidad de resultados erróneos no son tan pronunciados en el modelado discriminativo. La razón es que, al trabajar con tareas de clasificación o predicción sobre datos etiquetados, existen métricas claras y conjuntos de datos de prueba bien definidos para medir su rendimiento y precisión de forma objetiva.

Tipos de Modelos de IA Generativa

Hay varios tipos de modelos generativos.

Sin embargo, para el capítulo actual, vamos a echar un vistazo a estos modelos de IA generativa en las próximas secciones:

- Redes generativas adversarias (GANs)
- Autoencoders variacionales (VAEs)
- Modelos de difusión

Redes Generativas Adversarias (GANs)

Las GANs son una de las formas más tempranas de modelos de IA generativa, al menos en términos de aprovechar enfoques modernos como el aprendizaje profundo. Debido a esto, hay una considerable investigación en esta categoría. La GAN también se utiliza principalmente para generar imágenes, videos o ilustraciones realistas únicas. En los últimos años, la tecnología también ha demostrado ser efectiva con audio.

El modelo GAN se trata de la competencia entre dos redes neuronales. Es esencialmente un juego de suma cero. Cuando un modelo lo hace bien, el otro sufre. A través de este proceso, debería haber mejores resultados.

Para una GAN, las dos redes neuronales incluyen lo siguiente:

- Red generadora: Esta ingiere datos reales, como fotos. Pero el modelo luego creará imágenes falsas. Estas se denominan ruido.
- Red discriminadora: Este modelo intenta diferenciar las imágenes falsas de las reales. Para hacer esto, habrá una o más funciones de pérdida, como la función de pérdida

minimax, función de pérdida de Wasserstein, o función de pérdida de entropía cruzada. En las primeras etapas del entrenamiento del modelo GAN, la red discriminadora será altamente precisa. Pero a través de este proceso, el modelo generador será penalizado, lo que ayudará a mejorar los resultados para las imágenes falsas. Como resultado, la red discriminadora comenzará a declinar con el tiempo, hasta llegar a aproximadamente el 50%.

Los dos interactúan de manera "adversaria" y generalmente se implementan como una red neuronal convolucional o CNN (esto es mejor para datos de alta dimensión como imágenes). La red generadora intentará esencialmente engañar al modelo discriminador para que crea que los datos son reales. Los modelos complejos tendrán múltiples versiones de estos tipos de redes neuronales.

Existe un rico ecosistema de proyectos de código abierto para GANs, y podemos usarlos como arquitecturas para crear nuevos sistemas. Algunos de los más comunes incluyen los siguientes:

- CycleGAN: Esto es para proporcionar la transformación de imágenes con ciertos estilos. Por ejemplo, puede convertir un caballo en una cebra. O puede cambiar las estaciones para una imagen. Una aplicación popular que usa CycleGAN es FaceApp, que tiene más de 500 millones de descargas. Transforma selfies en retratos de calidad de modelo. Esto se hace con más de 60 filtros fotorealistas. Puede cambiar el color o el estilo de tu cabello, añadir una barba, proporcionar un toque de maquillaje o eliminar imperfecciones y arrugas.
- DiscoGAN: El concepto de relaciones entre dominios es donde hay conexiones entre objetos. Los humanos somos bastante buenos en esto. Con solo una mirada, podemos ver que diferentes artículos, como un bolso y zapatos, tienen un estilo similar. En cuanto a hacer esto con un sistema de IA, un GAN puede ser un buen enfoque, pero puede ser tedioso. Sin embargo, esto se simplifica con el DiscoGAN.



<https://github.com/junyanz/CycleGAN>

<https://github.com/SKTBrain/DiscoGAN>

John von Neumann fue una figura imponente durante el siglo XX. Entre sus muchos logros, desarrolló las bases fundamentales de la

computadora moderna. También sentó las bases de la teoría de juegos. Esto fue a partir de un libro que coescribió —titulado *Sobre la Teoría de los Juegos de Salón* (1928)— con el economista Oskar Morgenstern. Desde entonces, otros académicos como Emile Borel, Ernst Zermelo y John Nash (quien fue el sujeto de la película *Una Mente Maravillosa*) evolucionaron los conceptos de la teoría de juegos. Ganarían el Premio Nobel en 1994 por Ciencias Económicas.

Definitivamente hay desafíos con las GANs. Existe el problema del gradiente que desaparece. Esto sucede cuando la red discriminadora es demasiado efectiva. Esto significa que el modelo se estanca y hay pocas actualizaciones en los pesos de los parámetros. Esto puede llevar a una salida que no es muy diferente de la capa de entrada. Sin embargo, para manejar esto, se puede agregar otra función de pérdida.

Los problemas con la capa de salida pueden ser el resultado de datos de baja calidad. Esto se llama un colapso del modelo. Para esto, necesitarás agregar fuentes más diversas a los datos. Curiosamente, un modelo generativo se puede usar para crear esto.

Autoencoder Variacional (VAE)

Pero para la IA generativa, hay una variante del autoencoder tradicional. Es el autoencoder variacional o VAE. Estos toman datos, los comparan y luego los reconstruyen. Esto puede resultar en una salida interesante. Esta arquitectura también tiene un codificador y un decodificador, pero se añade algo más: El decodificador se basa en una distribución sofisticada de probabilidades, lo que permite aún más diversidad en la salida.

También está la función de pérdida de divergencia KL. Esto intenta minimizar la distancia entre el conjunto de datos y la distribución de probabilidades.

El VAE existe desde 2013. Diederik P. Kingma, científico investigador en Google Brain, y Max Welling, profesor de aprendizaje automático en la Universidad de Ámsterdam, publicaron esto en un artículo llamado "Auto-Encoding Variational Bayes".

Modelos de Difusión

En junio de 2020, los investigadores de IA Jonathan Ho, Ajay Jain y Pieter Abbeel publicaron un influyente artículo titulado "Modelos Probabilísticos de Difusión para Eliminación de Ruido". Esto estableció una innovación crítica para la IA generativa, que se ha dado a conocer como el modelo de difusión.

Los autores del artículo utilizaron esto con varios conjuntos de datos, como CIFAR10 y LSUN de 256×256 . Ciertamente, los resultados no indicaron que los modelos de difusión fueran mejores que las GANs. Pero esto no importó. Después de que se publicara el artículo, muchos otros investigadores comenzaron a enfocarse en los modelos de difusión, y los métodos mejoraron cada vez más.

Esto se basó en medidas como la puntuación FID (distancia de inicio de Fréchet). Evalúa la calidad de las imágenes de un modelo

generativo. FID compara la distribución de las imágenes generadas con las imágenes reales (<https://arxiv.org/abs/1706.08500>). Cuanto menor sea la distancia entre las dos, mejor será la calidad de la imagen. La puntuación más alta es 0.0.

Entonces, ¿cómo funciona un modelo de difusión? Es complicado, involucra matemáticas avanzadas y teoremas. Pero a un nivel alto, el proceso es realmente bastante sencillo. El modelo de difusión incluye dos pasos principales. El primero es el proceso de difusión hacia adelante. Por ejemplo, comenzaríamos con un conjunto de datos que tiene imágenes del mundo real. Cada una de ellas se conoce como una imagen base. A cada una, le añadimos ruido. Pero este no es un proceso aleatorio de un solo intento. En cambio, involucra el uso de ruido gaussiano. Esto implica una función de probabilidad que es igual a una distribución normal o curva de campana. Es una suma de las probabilidades para una variable. La distribución también se ajusta a la regla del 68-95-99.7. En otras palabras, aproximadamente el 68% de las observaciones de la curva de campana están dentro de una desviación estándar, el 95% dentro de dos desviaciones estándar y el 99.7% dentro de tres desviaciones estándar. Esta estructura es fundamental para la estadística. La razón principal es que la curva de campana a menudo se encuentra mostrando las relaciones en el mundo real, como los puntajes de IQ, la altura y el peso de una persona.

En cuanto a un modelo de difusión, se añade una pequeña cantidad de ruido gaussiano en múltiples pasos. A menudo varían de 1000 a 5000 aproximadamente, y cada uno se basa en una

cadena de Márkov. Esto significa que el modelo toma solo una de las imágenes anteriores. La suposición es que la imagen incorporará los datos de todas las demás. El resultado es que la imagen base tendrá cantidades crecientes de ruido. Al final, será solo estática.

Esto nos lleva al siguiente paso: el proceso de difusión inversa. Tomará la imagen estática y luego le hará un

denoising. Esto se hace usando un modelo de aprendizaje profundo, que se basa en el entrenamiento del proceso de difusión hacia adelante. Esto también utilizará la distribución gaussiana y la cadena de Markov. Como resultado, habrá una nueva imagen, que se parecerá a la imagen base.

A pesar de su potencia, los modelos de difusión tienen notables desventajas. Aquí hay un vistazo:

- **Suposiciones:** El uso de la distribución normal puede sesgar los resultados. No describe algunos fenómenos importantes del mundo real. Lo mismo ocurre con otros aspectos del modelo de difusión. Por ejemplo, existe la suposición de que las relaciones son lineales. Pero es ciertamente común que existan patrones no lineales o complejos en la naturaleza. Los modelos de difusión también asumen estacionariedad. Esto es donde el proceso, según se establece en la cadena de Markov, es estacionario o estático. Pero, por supuesto, la naturaleza puede ser muy dinámica.
- **Poder computacional:** Los modelos de difusión requieren enormes cantidades de procesamiento. Esto puede hacer que sea costoso construir modelos con grandes conjuntos de datos.
- **Distorsiones de imagen:** Pueden ser cómicas. ¡Incluso pueden parecerse a alguna forma de ciencia ficción! Esto podría estar bien para ciertas manipulaciones de imágenes reales. Pero puede estar muy lejos de la marca para otros. Considera que las caras de los

humanos, en modelos de difusión, están altamente distorsionadas cuando hay al menos tres. Otro problema es con el texto dentro de las imágenes. A menudo puede ser bastante malo. Sin embargo, es probable que tales problemas se aborden a medida que los modelos mejoren.

Ten en cuenta que el modelo de difusión está en el corazón de algunas de las aplicaciones más innovadoras de la IA generativa. Incluyen DALL-E 2, Imagen de Google, Midjourney y Stable Diffusion, solo por nombrar algunos.

DALL-E 2

OpenAI introdujo DALL-E en enero de 2021. El nombre es una combinación del personaje WALL-E de Pixar y Salvador Dalí, un famoso artista surrealista.

DALL-E se basa en el modelo GPT-3, que permite a un usuario escribir un mensaje que crea una imagen. En abril de 2022, OpenAI lanzó la próxima iteración de esta tecnología, llamada DALL-E 2. Sus imágenes eran mucho más realistas e inmersivas. Luego, a fines de 2022, OpenAI lanzó una API para esto.

El proceso de DALL-E 2 se basa en dos etapas principales. En primer lugar, está la fase previa. Tiene dos redes neuronales:

- Codificador de texto: Este utiliza un conjunto de datos de descripciones de texto relacionadas con imágenes.
- Codificador de imagen: Este procesa un conjunto de datos que tiene imágenes con subtítulos.

Cada uno de estos codificadores traduce las palabras e imágenes en incrustaciones, que son vectores numéricos. Se aplica el modelo CLIP ([Contrastive Language–Image Pre-training](#)). Intenta encontrar las mejores coincidencias entre el texto y la imagen.

Sin embargo, el modelo CLIP tiene desventajas. Por ejemplo, puede tener dificultades cuando se especifica un número específico de objetos o colores particulares. Una razón para esto es que no hay suficientes datos de imágenes.

A continuación, está la fase del decodificador o "unclip". Aquí se utiliza un modelo de difusión para crear las imágenes. Después de esto, puede haber manipulaciones adicionales, por ejemplo, con el remuestreo. Esto aumenta la resolución de la imagen.

La figura 4-1 muestra el flujo de trabajo general. El usuario introducirá un mensaje, como "Un astronauta montando un caballo al estilo de Andy Warhol". El sistema DALL-E 2 creará la incrustación de texto y filtrará esto para la fase previa. Luego, estos datos se traducirán en una incrustación de imagen. Este vector se convertirá en la base para crear la imagen.

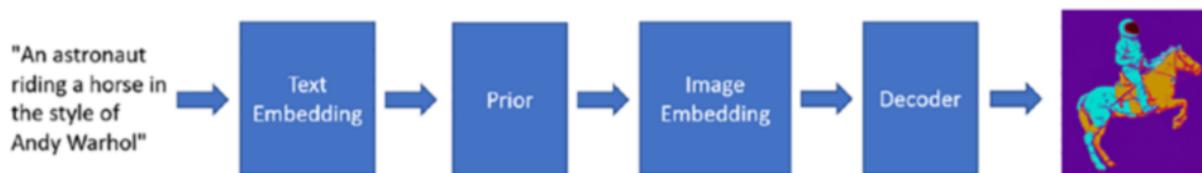


Figura 4-1. Esto muestra el flujo de trabajo básico para el modelo DALL-E 2.

Entonces, ¿qué pasaría si omitimos la incrustación de texto? ¿Qué tan importante es esto, realmente? Ten en cuenta que los investigadores que crearon el modelo para DALL-E 2 consideraron esto en su artículo "Generación de imágenes condicionada por texto jerárquico con latentes de CLIP". La conclusión fue que la incrustación de texto mejoraba mucho los resultados. Esencialmente, esto proporcionaba más contexto, lo que permitía un mejor emparejamiento.

Algo más sobre este modelo es que generalmente es ligero para los sistemas de IA generativa. Debido a esto, los generadores de imágenes pueden implementarse en dispositivos pequeños como iPhones.

Stability AI y Midjourney

Hay una variedad de sistemas de IA generativa de texto a imagen. Pero la startup que ha visto más tracción es [Stability AI](#). El CEO y cofundador es Emad Mostaque. Él nació en Jordania pero eventualmente se mudaría a Reino Unido. En cuanto a su educación universitaria, recibió su maestría en matemáticas e informática de la Universidad de Oxford en 2005. Pero continuaría su carrera en la industria de los fondos de cobertura.

Aunque esto resultó lucrativo, quería algo más significativo. En 2020, fundó Stability AI y comenzó las operaciones con recursos propios. Al principio, el enfoque estaba en aprovechar la IA para ayudar con la pandemia de COVID-19.

Pero a medida que aprendía más sobre las innovaciones con la IA generativa, hizo un cambio de dirección en la empresa. Se trataría de usar la tecnología para permitir que cualquiera creara imágenes convincentes.

Para ayudar a acelerar el crecimiento, Mostaque adoptó una estrategia de código abierto. Pensó que esto ayudaría mucho a democratizar la estrategia.

También formó asociaciones críticas. "Stable Diffusion es diferente a cualquier otro proyecto de código abierto que hemos encontrado", dijo Gaurav Gupta, quien es socio en Lightspeed Venture Partners e inversor en Stability AI. "Su historia de creación es novedosa, habiendo sido realizada en colaboración con académicos (LMU Munich), otras compañías (RunwayML) y comunidades independientes de IA (EleutherAI, LAION). Creemos que Stability AI será el socio comercial ideal para empresas, organizaciones sin fines de lucro y gobiernos que no tienen acceso a talento en IA, recursos computacionales y la capacidad de personalizar el uso."

En agosto de 2022, Stability lanzó su plataforma de código abierto llamada [Stable Diffusion](#), y la adopción fue intensa. Fue uno de los proyectos de crecimiento más rápido de la historia. Luego, la compañía lanzó una versión en la nube del sistema, [DreamStudio](#). En menos de un mes, atrajo a más de 1.5 millones de usuarios.

El enfoque de Stability AI para la IA generativa fue diferente al de DALL-E 2. Se basa en un artículo titulado "[Síntesis de Imágenes de Alta Resolución con Modelos de Difusión Latente](#)". Un modelo de difusión latente está diseñado para lidiar con los altos requisitos computacionales de la difusión tradicional. Pero mediante el uso de autoencoders en el espacio latente, donde se almacenan las características clave, hay mucha más eficiencia, sin afectar la calidad de las imágenes. Por eso alguien puede descargar Stable Diffusion y usarlo en su propio sistema de PC.

Sin embargo, los costos de infraestructura aún eran sustanciales para Mostaque debido a la enorme base de usuarios. Simplemente no tenía los recursos personales para financiar la operación por más tiempo. Por eso buscó capital de riesgo. En octubre de 2022, Stability AI anunció una inversión de 101 millones de dólares. La valoración se estableció en aproximadamente 1,000 millones de dólares.

Ahora, otra plataforma de IA generativa de texto a imagen de primer nivel es Midjourney. Antes de fundar la compañía, David Holz era el CEO y cofundador de Leap Motion. La compañía creó dispositivos para rastrear los movimientos de las manos.

Él no construyó la infraestructura para esto. En su lugar, utilizó Discord, que es un sistema de chat popular. Con esto, los usuarios pueden crear sus propios canales y mostrar sus imágenes creativas. La figura 4-2 muestra cómo se ve esto.

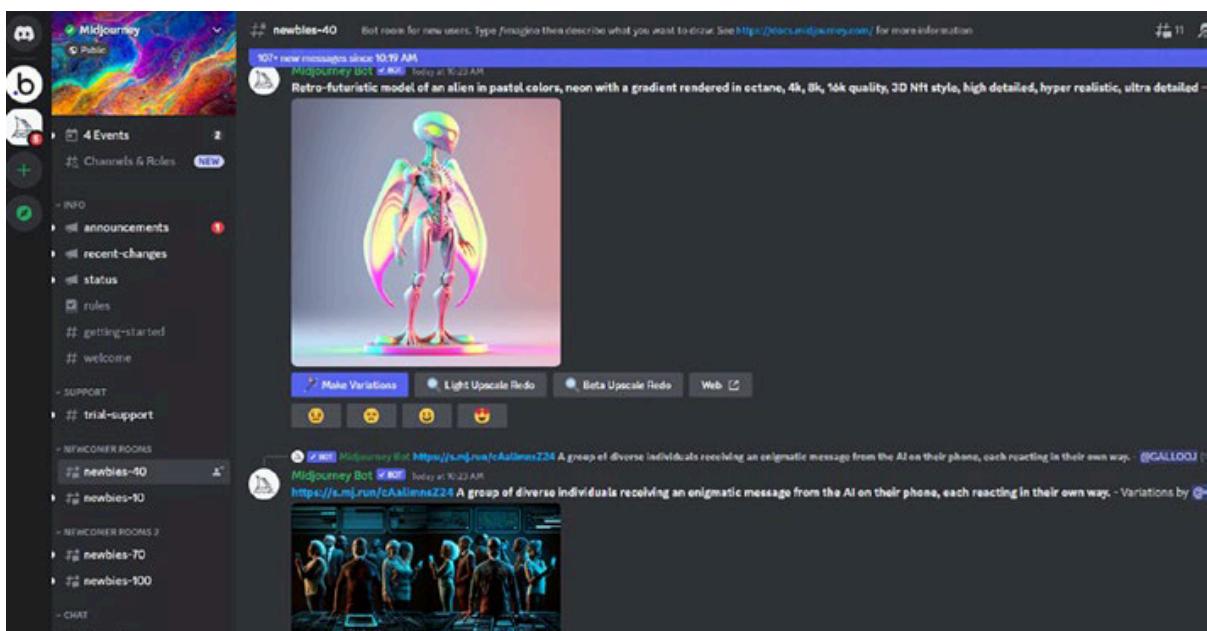


Figura 4-2. Este es un canal comunitario para creadores en Midjourney.

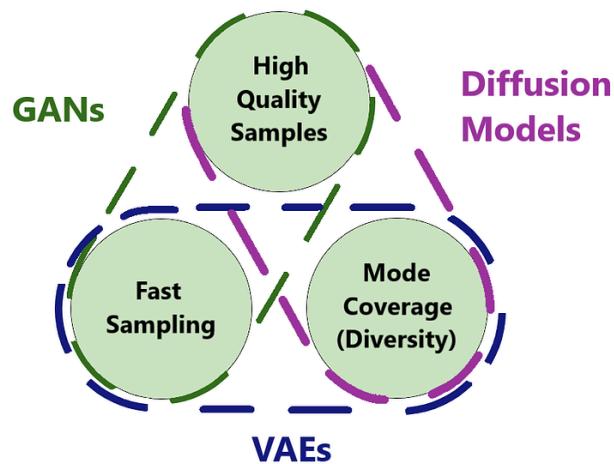
Holz ha financiado la empresa con fondos propios. Pero a medida que continúa el crecimiento, parece probable que, al igual que Stability AI, se necesitará financiamiento externo.

Nota: Durante la década de 1970, había juegos de aventura. Pero debido a la capacidad limitada de las PC, no había gráficos. En cambio, había descripciones escritas. Uno de los juegos de aventura más populares era Zork. Entonces, ¿cómo se vería esto con gráficos? Matt Walsh, que es un principal en Google, quería averiguarlo y utilizó la IA generativa para esto. Un sistema de texto a imagen, llamado Imagen, pudo proporcionar los visuales para el juego

Trilema de los Modelos de IA Generativa

Dado que la IA generativa aún está en sus inicios, hay muchos desafíos y problemas. Una forma de describirlos es a través del trilema. Esto describe los compromisos entre tres factores principales:

- Calidad de la muestra: Aquí es donde el modelo crea muestras de alta calidad. No solo deben parecer realistas, sino también ser similares a los datos de base.
- Rendimiento: Aquí, un modelo puede crear muestras rápidamente. Preferiblemente, esto significa en tiempo real o casi en tiempo real. Esto permite su uso en aplicaciones que tendrán mucha utilidad.
- Cobertura de modos: Es un modelo que puede crear muestras diversas de manera efectiva.



Dado este marco, echemos un vistazo a las GANs. Definitivamente son fuertes en términos de calidad de muestra y rendimiento. Sin embargo, la cobertura de modos puede ser insuficiente. El hecho es que las GANs generalmente producen un pequeño número de muestras. Esto puede significar que la creatividad del modelo no es tan convincente.

En cuanto a un modelo de difusión, es mejor con la cobertura de modos así como con la calidad de la muestra. Pero el problema está en el rendimiento. Como se mencionó anteriormente, puede ser intensivo en cómputo crear modelos de difusión fuertes.

Pero la buena noticia es que hay una investigación considerable sobre estos temas en todo el mundo. El ritmo de innovación también es bastante rápido.

Conclusión

En este capítulo, primero observamos las diferencias entre los modelos discriminativos y generativos. Los modelos discriminativos generalmente son para predicciones. Este tipo de IA es el más prevalente.

La IA generativa, por otro lado, también se trata de usar predicciones. Pero hay algunos otros aspectos en el modelo. Hay un elemento de probabilidad y datos aleatorios. Al hacer esto, es posible crear datos. Pero también es similar al conjunto de datos base.

La IA generativa es relativamente nueva. El primer modelo surgió en 2014 con la GAN. Fue un gran avance e impulsó una investigación considerable.

Luego, en 2020, se introdujo el modelo de difusión. Esto finalmente mostró resultados incluso mejores que las GANs. El modelo de difusión es una de las fuerzas impulsoras de las últimas innovaciones con la IA generativa.

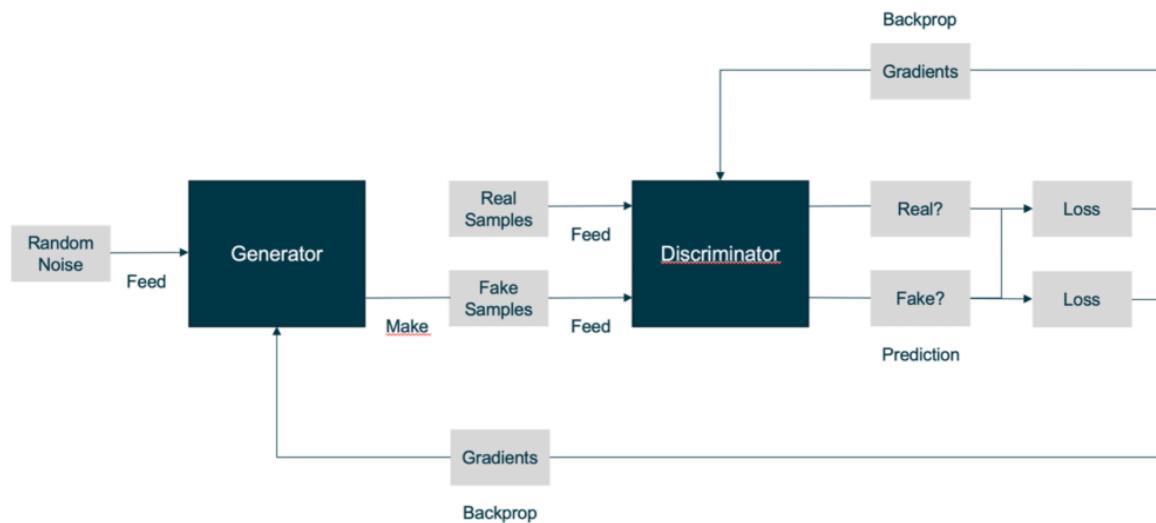
Pero todavía hay mucho por hacer. Como se ve con el problema del trilema, hay compromisos cuando se trata de modelos de IA generativa. Pero estos deberían disminuir a medida que continúa la ávida investigación.

Generative Adversarial Networks (GANs)

(Fuente: https://developers.google.com/machine-learning/gan/gan_structure?hl=es-419)

Una red generativa adversaria (GAN) tiene dos partes:

- El **generador** aprende a generar datos posibles. Las instancias generadas se convierten en ejemplos de entrenamiento negativos para el discriminante.
- El **discriminador** aprende a distinguir los datos falsos del generador de los reales. El discriminante penaliza al generador por producir resultados improbables.



Cuando comienza el entrenamiento, el generador produce datos obviamente falsos y el discriminante aprende con rapidez que es falso:



A medida que el entrenamiento avanza, el generador se acerca al resultado que puede engañar al discriminador:



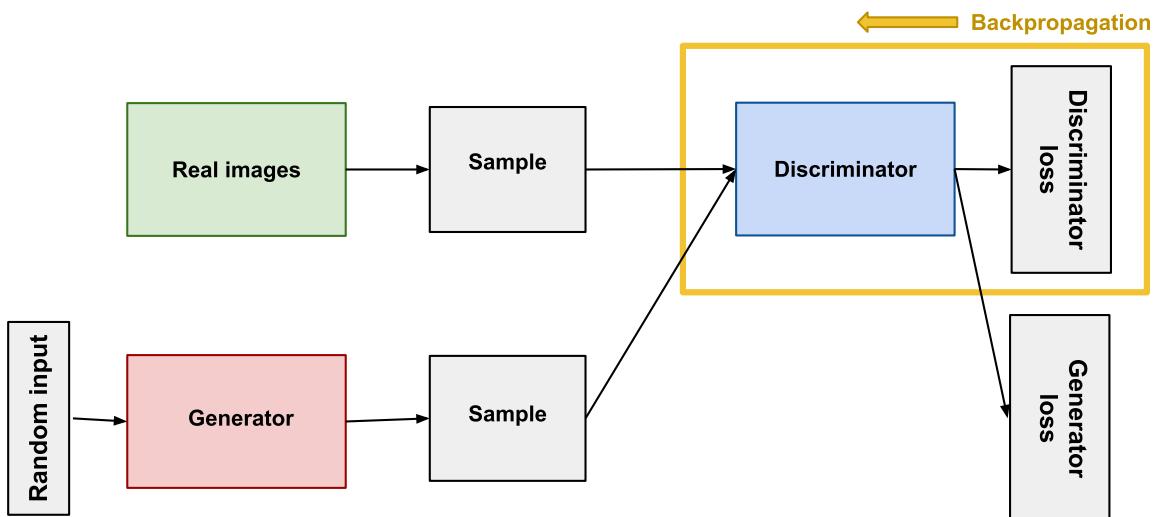
Por último, si el entrenamiento del generador funciona bien, el discriminante empeora al indicar la diferencia entre real y falso. Comienza a clasificar los datos falsos como reales y disminuye su exactitud.



El generador y el discriminante son redes neuronales independientes. El resultado del generador se conecta directamente a la entrada del discriminador. Mediante la retropropagación (backpropagation), la clasificación del discriminador proporciona una señal que el generador usa para actualizar sus pesos.

Discriminador

El discriminador en una GAN es simplemente un clasificador. Intenta distinguir datos reales de los creados por el generador. Podría usar cualquier arquitectura de red adecuada para el tipo de datos que clasifica.



Propagación inversa en el entrenamiento de discriminadores

Datos de entrenamiento sobre discriminación

Los datos de entrenamiento del discriminante provienen de dos fuentes:

- Instancias de **datos reales**, como fotos reales de personas. El discriminante usa estas instancias como ejemplos positivos durante el entrenamiento.
- Instancias de **datos falsos** creadas por el generador. El discriminante usa estas instancias como ejemplos negativos durante el entrenamiento.

En la figura, las dos cajas 'Sample' representan estas dos fuentes de datos que alimentan al discriminador. Durante el entrenamiento del discriminador, el generador no se entrena. Sus pesos permanecen constantes mientras produce ejemplos para que el discriminador se entrene con ellos.

Entrenamiento del discriminador

El discriminante se conecta a dos funciones de pérdida. Durante el entrenamiento del discriminante, este ignora la pérdida del generador y solo usa la pérdida del discriminador.

Durante el entrenamiento del discriminador:

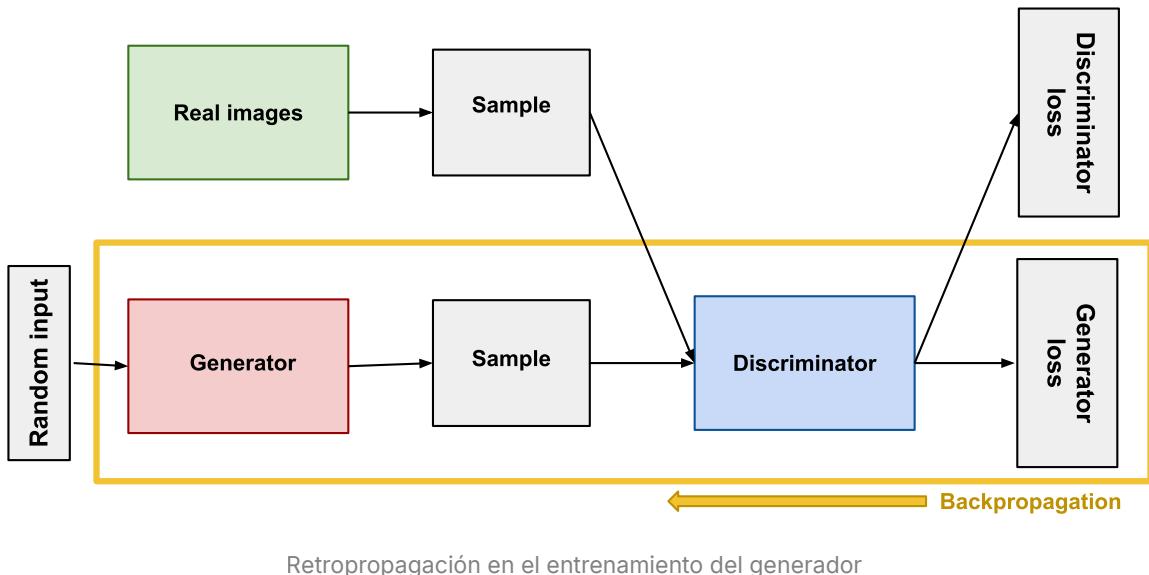
1. El discriminador clasifica tanto datos reales como datos falsos provenientes del generador.
2. La pérdida del discriminador penaliza al discriminador por clasificar incorrectamente una instancia real como falsa o una instancia falsa como real.
3. El discriminador actualiza sus pesos a través de la retropropagación desde la pérdida del discriminador a través de la red del discriminador.

Generador

El generador de una GAN aprende a crear datos falsos mediante la incorporación del feedback del discriminante. Aprende a hacer que el discriminador clasifique su resultado como real.

El entrenamiento del generador requiere una integración más estrecha entre el generador y el discriminador que el entrenamiento del discriminador. La parte de la GAN que entrena al generador incluye lo siguiente:

- entrada aleatoria
- red de generación, que transforma la entrada aleatoria en una instancia de datos
- red discriminante, que clasifica los datos generados
- salida del discriminante
- la pérdida del generador, que penaliza el generador por no engañar al discriminador



Entrada aleatoria

Las redes neuronales necesitan algún tipo de entrada. Por lo general, ingresamos datos con los que queremos hacer algo, como una instancia sobre la que queremos clasificar o hacer una predicción. Pero ¿Qué usamos como entrada para una red que genera instancias de datos completamente nuevas?

En su forma más básica, una GAN toma ruido aleatorio como su entrada. Luego, el generador transforma este ruido en un resultado significativo. Al introducir ruido, podemos hacer que la GAN produzca una amplia variedad de datos, que provienen de diferentes lugares en la distribución objetivo.

Los experimentos sugieren que la distribución del ruido no importa mucho, por lo que podemos elegir algo fácil de usar, como una distribución uniforme. Para mayor comodidad, el espacio desde el que se muestre el ruido suele ser de menor dimensión que la dimensionalidad del espacio de salida.

Cómo usar el discriminación para entrenar el generador

Para entrenar una red neuronal, alteramos los pesos de la red para reducir el error o la pérdida de su salida. Sin embargo, en nuestra GAN, el generador no está directamente conectado a la pérdida que estamos intentando afectar. El generador se alimenta en la red del discriminador, y el discriminador produce la salida que estamos intentando afectar. La pérdida del generador penaliza al generador por producir una muestra que la red del discriminador clasifica como falsa.

La retropropagación ajusta cada peso en la dirección correcta calculando el impacto del peso en la salida — cómo cambiaría la salida si cambiases el peso. Pero el impacto de un peso del generador depende del impacto de los pesos del discriminador en los que se alimenta. Por lo tanto, la retropropagación comienza en la salida y fluye hacia atrás a través del discriminador hacia el generador.

Al mismo tiempo, no queremos que el discriminador cambie durante el entrenamiento del generador. Intentar darle a un blanco móvil haría que un problema difícil sea aún más difícil

para el generador.

Entonces entrenamos al generador con el siguiente procedimiento:

1. Muestra ruido aleatorio.
2. Produce la salida del generador a partir del ruido aleatorio muestreado.
3. Obtén la clasificación "Real" o "Falso" del discriminador para la salida del generador.
4. Calcula la pérdida a partir de la clasificación del discriminador.
5. Retropropaga a través del discriminador y el generador para obtener gradientes.
6. Usa los gradientes para cambiar solo los pesos del generador.

Esto es una iteración del entrenamiento del generador. En la próxima sección veremos cómo equilibrar el entrenamiento de ambos, el generador y el discriminador.

Debido a que una GAN contiene dos redes entrenadas por separado, su algoritmo de entrenamiento debe abordar dos complicaciones:

- Las GAN deben hacer malabares con dos tipos diferentes de entrenamiento (generador y discriminante).
- La convergencia de las GAN es difícil de identificar.

Entrenamiento alternativo

El generador y el discriminador tienen diferentes procesos de entrenamiento. Entonces, ¿Cómo entrenamos la GAN en su totalidad?

El entrenamiento de la GAN procede en períodos alternos:

1. El discriminador se entrena durante una o más épocas.
2. El generador se entrena durante una o más épocas.
3. Repetir los pasos 1 y 2 para continuar entrenando las redes del generador y el discriminador.

Mantenemos el generador constante durante la fase de entrenamiento del discriminador.

Dado que el entrenamiento del discriminador intenta descubrir cómo distinguir los datos reales de los falsos, tiene que aprender a reconocer las fallas del generador. Eso es un problema diferente para un generador bien entrenado que para un generador no entrenado que produce salida aleatoria.

De manera similar, mantenemos el discriminador constante durante la fase de entrenamiento del generador. De lo contrario, el generador estaría intentando alcanzar un objetivo en movimiento y podría nunca converger.

Es este vaivén el que permite que las GAN aborden problemas generativos de otro modo intratables. Obtenemos un punto de apoyo en el difícil problema generativo comenzando con un problema de clasificación mucho más simple. A la inversa, si no puedes entrenar un clasificador para distinguir entre datos reales y generados incluso para la salida aleatoria inicial del generador, no puedes comenzar el entrenamiento de la GAN.

Convergencia

A medida que el generador mejora con el entrenamiento, el rendimiento del discriminante empeora, ya que el discriminante no puede distinguir con facilidad la diferencia entre el real y el falso. Si el generador tiene éxito, el discriminador tiene una exactitud del 50%. De hecho, el discriminante lanza una moneda para hacer su predicción.

Esta progresión plantea un problema para la convergencia de la GAN en su totalidad: los comentarios del discriminante se vuelven menos significativos con el tiempo. Si la GAN continúa entrenando después del punto en el que el discriminante brinda comentarios completamente aleatorios, el generador comienza a entrenar para recibir comentarios no deseados y su propia calidad puede contraerse.

Para una GAN, la convergencia suele ser un estado fugaz, en lugar de estable.

Funciones de pérdida

Las GAN intentan replicar una distribución de probabilidad. Por lo tanto, deben usar funciones de pérdida que reflejen la distancia entre la distribución de los datos generados por la GAN y la distribución de los datos reales.

¿Cómo puedes capturar la diferencia entre dos distribuciones en las funciones de pérdida de GAN? Esta pregunta es un área de investigación activa y se han propuesto muchos enfoques. Abordaremos dos funciones comunes de pérdida de GAN aquí, ambas implementadas en TF-GAN:

- **minimax loss:** La función de pérdida que se usó en el [artículo que introdujo GAN](#).
- **Pérdida de Wasserstein:** La función de pérdida predeterminada para los Estimadores de TF-GAN. Se describió por primera vez en un [artículo de 2017](#).

TF-GAN también implementa muchas otras funciones de pérdida.

¿Una función de pérdida o dos?

Una GAN puede tener dos funciones de pérdida: una para el entrenamiento de generadores y otra para el entrenamiento de discriminadores. ¿Cómo pueden funcionar juntas dos funciones de pérdida para reflejar una medida de distancia entre distribuciones de probabilidad?

En los esquemas de pérdida que analizaremos aquí, las pérdidas del generador y del discriminante derivan de una única medida de distancia entre las distribuciones de probabilidad. Sin embargo, en ambos esquemas, el generador solo puede afectar un término en la medida de distancia: el término que refleja la distribución de los datos falsos. Durante el entrenamiento del generador, descartamos el otro término, que refleja la distribución de los datos reales.

Las pérdidas del generador y del discriminador se ven diferentes al final, a pesar de que derivan de una sola fórmula.

Pérdida máxima

En el artículo sobre las GAN, el generador intenta minimizar la siguiente función mientras que el discriminante intenta maximizarla:

$$Ex[\log(D(x))] + Ez[\log(1-D(G(z)))]$$

En esta función:

- $D(x)$ es la estimación del discriminador de la probabilidad de que la instancia x de datos real sea real.
- E_x es el valor esperado para todas las instancias de datos reales.
- $G(z)$ es la salida del generador cuando se proporciona z de ruido.
- $D(G(z))$ es la estimación del discriminador de la probabilidad de que una instancia falsa sea real.
- E_z es el valor esperado en todas las entradas aleatorias del generador (en efecto, el valor esperado en todas las instancias falsas generadas $G(z)$).
- La fórmula deriva de la entropía cruzada entre las distribuciones reales y generadas.

El generador no puede afectar directamente el término $\log(D(x))$ en la función. Por lo tanto, para el generador, minimizar la pérdida es equivalente a minimizar $\log(1 - D(G(z)))$.

Sobre TF-GAN, podemos consultar minimax_discriminator_loss y minimax_generator_loss para ver una implementación de esta función de pérdida.

Pérdida modificada máxima

En el informe original de GAN, se indica que la función de pérdida minimax anterior puede hacer que el GAN se detenga en las primeras etapas del entrenamiento de la GAN, cuando el trabajo del discriminante es muy fácil. Por lo tanto, en el documento se sugiere modificar la pérdida del generador para que el generador intente maximizar $\log D(G(z))$.

Sobre TF-GAN, podemos consultar modified_generator_loss para ver una implementación de esta modificación.

Pérdida de Wasserstein

De forma predeterminada, TF-GAN usa la pérdida de Wasserstein.

Esta función de pérdida depende de una modificación del esquema de GAN (denominado "Wasserstein GAN") o en la que el discriminante no clasifica instancias. Para cada instancia, da como resultado un número. Este número no debe ser menor que uno o mayor que 0, por lo que no podemos usar 0.5 como límite para decidir si una instancia es real o falsa. El entrenamiento de discriminación solo trata de agrandar el resultado para instancias reales que para instancias falsas.

Debido a que no se puede discriminar entre el real y el falso, el discriminante de WGAN en realidad se denomina "crítico" en lugar de "discriminador". Esta distinción tiene una importancia teórica, pero para fines prácticos, se la puede considerar como un reconocimiento de que las entradas a las funciones de pérdida no tienen que ser probabilidades.

Las funciones de pérdida son engañosamente simples:

Pérdida de críticos: $D(x) - D(G(z))$

El discriminante intenta maximizar esta función. En otras palabras, intenta maximizar la diferencia entre su resultado en instancias reales y su resultado en instancias falsas.

Pérdida del generador: $D(G(z))$

El generador intenta maximizar esta función. En otras palabras, intenta maximizar el resultado del discriminante para sus instancias falsas.

En estas funciones:

- $D(x)$ es el resultado de un crítico para una instancia real.
- $G(z)$ es la salida del generador cuando se proporciona z de ruido.
- $D(G(z))$ es el resultado de un crítico para una instancia falsa.
- El resultado del crítico D no tiene que estar entre 1 y 0.
- Las fórmulas derivan de la distancia del desplazador de la Tierra entre las distribuciones reales y generadas.

En TF-GAN, consulta wasserstein_generator_loss y wasserstein_discriminator_loss para conocer las implementaciones.

Requisitos

La justificación teórica de la GAN (o WGAN) de Wasserstein requiere que las ponderaciones en toda la GAN se recorten para que permanezcan dentro de un rango restringido.

Beneficios

Las GAN de Wasserstein son menos vulnerables a los bloqueos que las GAN basadas en minimax y evitan los problemas con gradientes que desaparecen. La distancia del movimiento de la Tierra también tiene la ventaja de ser una métrica real: una medida de la distancia en un espacio de distribuciones de probabilidad. La entropía cruzada no es una métrica en este sentido.

Variable Autoencoders (VAEs)

Fuente: <https://towardsdatascience.com/difference-between-autoencoder-ae-and-variational-autoencoder-vae-ed7be1c038f2>

Compresión de datos

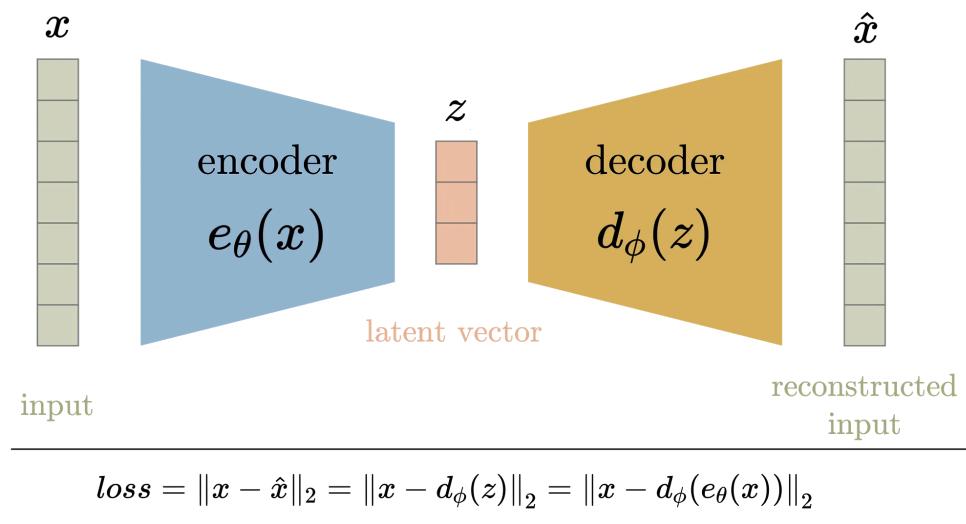
La compresión de datos es una fase esencial en el entrenamiento de una red. La idea es comprimir los datos para que la misma cantidad de información pueda ser representada por menos bits. Esto también ayuda con el problema de la maldición de la dimensionalidad. Un conjunto de datos con muchos atributos es diferente de entrenar porque tiende a sobreajustar el modelo. Por lo tanto, se deben aplicar técnicas de reducción de dimensionalidad antes de que el conjunto de datos pueda ser utilizado para el entrenamiento.

Aquí es donde entran en juego el Autoencoder (AE) y el Variational Autoencoder (VAE). Son redes de extremo a extremo que se utilizan para comprimir los datos de entrada. Tanto el Autoencoder como el Variational Autoencoder se utilizan para transformar los datos de un

espacio de mayor dimensión a un espacio de menor dimensión, logrando esencialmente la compresión.

Autoencoder (AE)

Para llegar a entender los VAE, primero debemos comprender el funcionamiento de los AE. El Autoencoder se utiliza para aprender representaciones eficientes de datos no etiquetados para una configuración de red dada. El autoencoder consta de dos partes, un codificador y un decodificador. El codificador comprime los datos de un espacio de mayor dimensión a un espacio de menor dimensión (también llamado espacio latente), mientras que el decodificador hace lo contrario, es decir, convierte el espacio latente de nuevo al espacio de mayor dimensión. El decodificador se utiliza para asegurar que el espacio latente pueda capturar la mayor parte de la información del espacio del conjunto de datos, forzándolo a que salga lo que se introdujo como entrada al decodificador.



Durante el entrenamiento, los datos de entrada x se alimentan a la función del codificador **$e_theta(x)$** . La entrada pasa a través de una serie de capas (parametrizadas por la variable theta) reduciendo sus dimensiones para lograr un vector latente comprimido z . El número de capas, el tipo y tamaño de las capas, y la dimensión del espacio latente son parámetros controlados por el usuario. Se logra la compresión si la dimensión del espacio latente es menor que la del espacio de entrada, eliminando esencialmente los atributos redundantes.

El decodificador **$d_phi(z)$** generalmente (pero no necesariamente) consiste en capas casi complementarias de las capas utilizadas en el codificador, pero en orden inverso. Una capa casi complementaria de una capa es aquella que puede usarse para deshacer las operaciones (hasta cierto punto) de la capa original, como una capa de convolución transpuesta a una capa de convolución, una de pooling a una de unpooling, totalmente conectada a totalmente conectada, etc.

Función de pérdida

Toda la arquitectura codificador-decodificador se entrena colectivamente en la función de pérdida que fomenta que la entrada se reconstruya en la salida. Por lo tanto, la función de pérdida es el error cuadrático medio entre la entrada del codificador y la salida del decodificador.

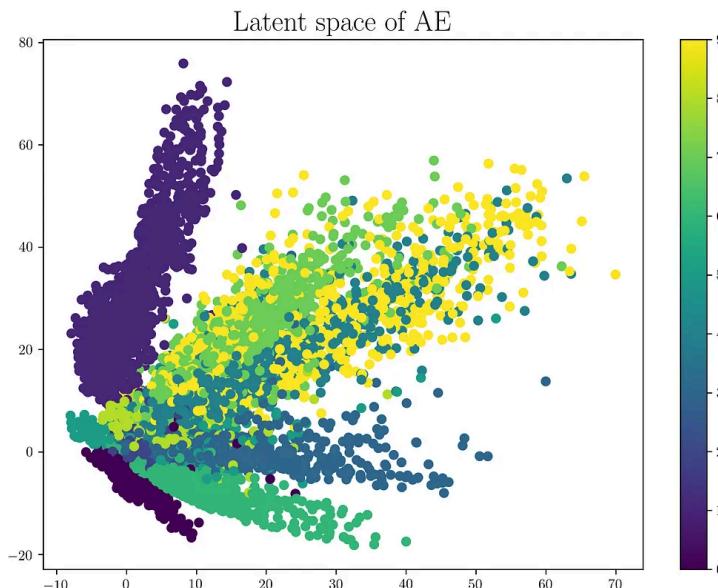
$$loss = \|x - \hat{x}\|_2 = \|x - d_\phi(z)\|_2 = \|x - d_\phi(e_\theta(x))\|_2$$

La idea es tener un espacio latente de muy baja dimensión para lograr la máxima compresión, pero al mismo tiempo, mantener el error lo suficientemente pequeño. Reducir la dimensión del espacio latente más allá de un cierto valor resultará en una pérdida significativa de información.

No hay restricciones en los valores/distribución del espacio latente. Puede ser cualquier cosa, siempre y cuando pueda reconstruir la entrada cuando se aplique la función del decodificador.

Visualización del espacio latente

A continuación se muestra un ejemplo del espacio latente generado al entrenar la red con un conjunto de datos MNIST.



Se puede observar que los mismos dígitos tienden a agruparse en el espacio latente. Otra cosa importante a notar es que hay partes del espacio latente que no corresponden a ningún punto de datos. Usar esos puntos como entradas para el codificador resultará en una salida que no se parece a ningún dígito del conjunto de datos MNIST. Esto es lo que queremos decir cuando decimos que el espacio latente no está regularizado. Dicho espacio latente solo tiene unas pocas regiones/cluster que tienen la capacidad generativa, lo que significa que muestrear cualquier punto en el espacio latente que pertenezca a un cluster generará una variación de los datos a los que pertenece el cluster. Pero el espacio latente completo no tiene la capacidad generativa. Las regiones que no pertenecen a ningún cluster generarán una salida basura. Una vez que la red está entrenada y se eliminan los datos de entrenamiento, no tenemos forma de saber si la salida generada por el decodificador a partir de un vector latente muestreado al azar es válida o no. Por lo tanto, el AE se utiliza principalmente para la compresión.



Para entradas válidas, el AE es capaz de comprimir las en menos bits, eliminando esencialmente la redundancia (Codificador), pero debido al espacio latente no regularizado del AE, el decodificador no puede usarse para generar datos de entrada válidos a partir de vectores latentes muestreados del espacio latente.

¿Por qué usar AE para la compresión?

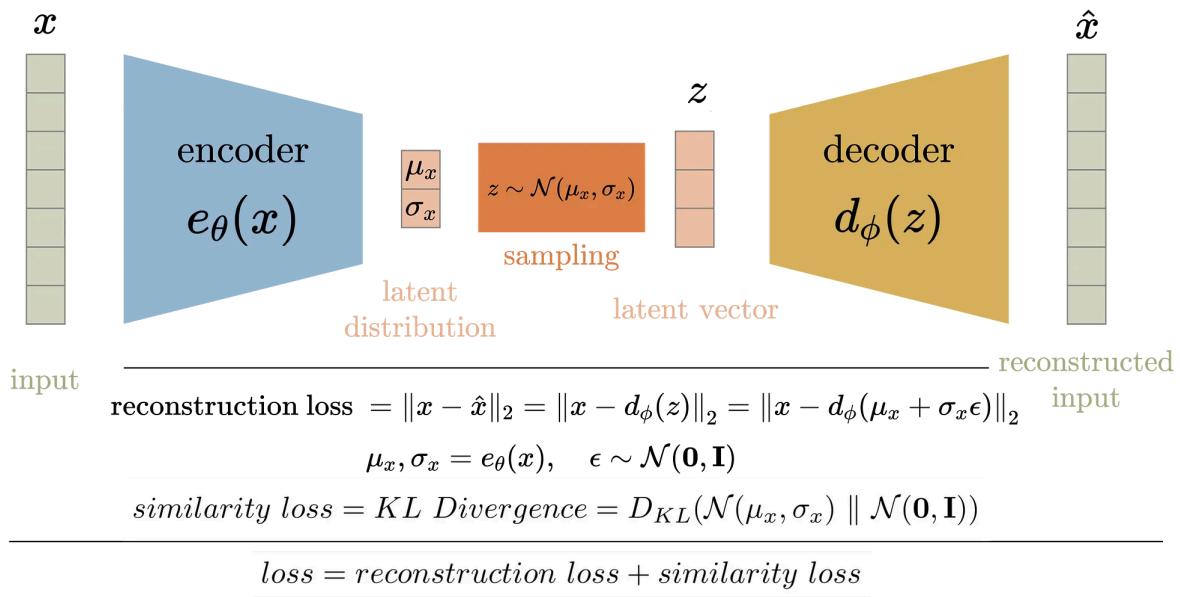
El espacio latente de un autoencoder lineal se parece mucho al espacio de eigenvectores logrado durante el análisis de componentes principales de los datos (PCA). Un autoencoder lineal con dimensión del espacio de entrada n y dimensiones del espacio latente establecidas en $m < n$ abarcará el mismo espacio vectorial que abarcan los primeros m eigenvectors de PCA. Si el AE es similar a PCA, ¿por qué usar AE? El poder del AE viene con su no linealidad. Agregar no linealidad (como funciones de activación no lineales y más capas ocultas) hace que el AE sea capaz de aprender representaciones bastante poderosas de los datos de entrada en dimensiones más bajas con mucha menos pérdida de información.

Autoencoders Variacionales (VAEs)

El VAE aborda el problema del espacio latente no regularizado en el AE y proporciona la capacidad generativa a todo el espacio. El codificador en el AE genera vectores latentes. En lugar de generar los vectores en el espacio latente, el codificador del VAE genera parámetros de una distribución predefinida en el espacio latente para cada entrada. El VAE impone una restricción en esta distribución latente, forzándola a ser una distribución normal. Esta restricción asegura que el espacio latente esté regularizado.

Diagrama de bloques

El diagrama de bloques del VAE se puede ver a continuación. Durante el entrenamiento, los datos de entrada x se alimentan a la función del codificador **e_theta(x)**. Al igual que en el AE, la entrada pasa a través de una serie de capas (parametrizadas por la variable theta), reduciendo sus dimensiones para lograr un vector latente comprimido z . Sin embargo, el vector latente no es la salida del codificador. En su lugar, el codificador genera la media y la desviación estándar para cada variable latente. El vector latente se muestra a partir de esta media y desviación estándar, que luego se alimenta al decodificador para reconstruir la entrada. El decodificador en el VAE funciona de manera similar al del AE.



Función de pérdida

La función de pérdida está definida por los objetivos del VAE. El VAE tiene dos objetivos:

1. Reconstruir la entrada.
2. El espacio latente debe estar distribuido normalmente.

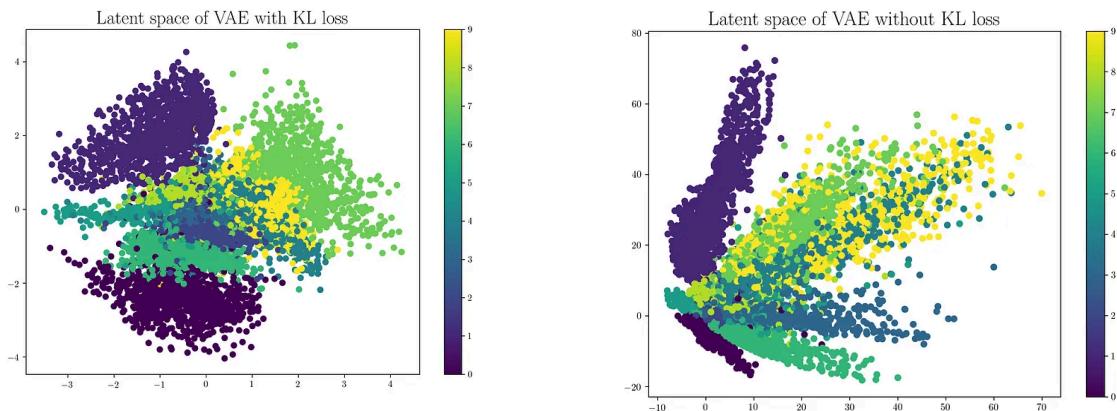
Por lo tanto, la pérdida de entrenamiento del VAE se define como la suma de la pérdida de reconstrucción y la pérdida de similitud. El error de reconstrucción, al igual que en el AE, es la pérdida cuadrática media entre la entrada y la salida reconstruida. La pérdida de similitud es la divergencia KL entre la distribución del espacio latente y la distribución gaussiana estándar (media cero y varianza unitaria). La función de pérdida es entonces la suma de estas dos pérdidas.

Como se mencionó antes, el vector latente se muestrea de la distribución generada por el codificador antes de alimentarlo al decodificador. Este muestreo aleatorio dificulta la retropropagación para el codificador, ya que no podemos rastrear los errores debido a este muestreo aleatorio. Por lo tanto, utilizamos un truco de reparametrización para modelar el proceso de muestreo, lo que hace posible que los errores se propaguen a través de la red. El vector latente

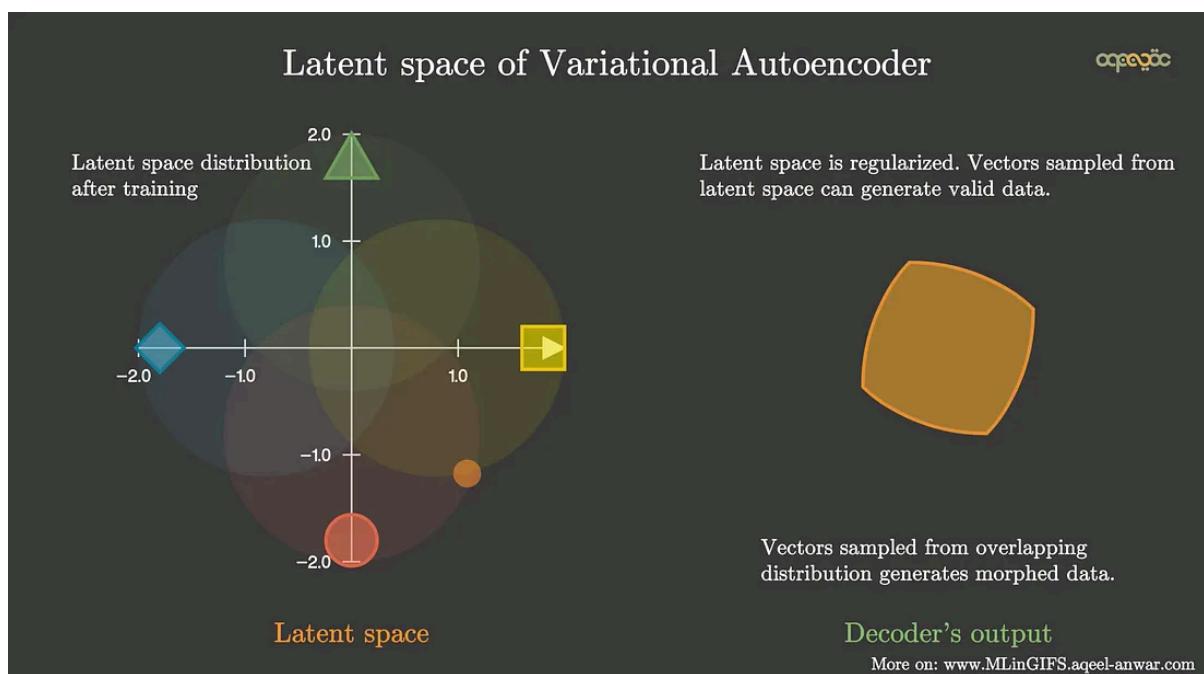
z se representa como una función de la salida del codificador.

$$z = \mu_x + \sigma_x \epsilon, \quad \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

El truco de reparametrización se utiliza para representar el vector latente z en función de la salida del codificador.



Una cosa importante a tener en cuenta es que cuando se toma una muestra del vector latente de las regiones con grupos superpuestos, obtenemos datos transformados. Obtenemos una transición suave entre la salida del decodificador cuando tomamos muestras del espacio latente que se mueve de un grupo a otro.



<https://www.youtube.com/watch?v=sV2FOdGqlX0>

Resumen

Hemos visto aquí los conceptos de Autoencoder (AE) y Autoencoder Variacional (VAE), que se utilizan principalmente para la compresión de datos y la generación de datos, respectivamente. El VAE aborda el problema del espacio latente no regularizado del AE, lo que le permite generar datos a partir de vectores muestreados aleatoriamente del espacio latente. Los puntos clave de resumen del AE y VAE son:

Autoencoder (AE)

- Se utiliza para generar una transformación comprimida de la entrada en un espacio latente.
- La variable latente no está regularizada.
- Elegir una variable latente aleatoria generará una salida basura.
- La variable latente tiene discontinuidad.
- La variable latente tiene valores determinísticos.
- El espacio latente carece de capacidad generativa.

Autoencoder Variacional (VAE)

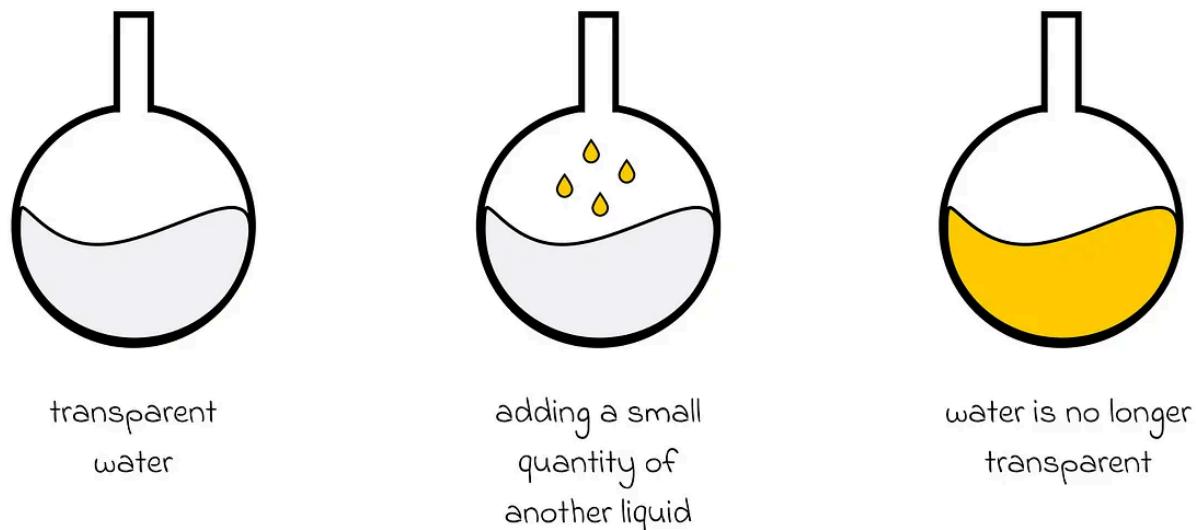
- Impone condiciones a la variable latente para que tenga una norma unitaria.
- La variable latente en forma comprimida es la media y la varianza.
- La variable latente es suave y continua.
- Un valor aleatorio de la variable latente genera una salida significativa en el decodificador.
- La entrada del decodificador es estocástica y se muestrea de una gaussiana con la media y la varianza de la salida del codificador.
- Espacio latente regularizado.
- El espacio latente tiene capacidades generativas.

Diffusion Models (Modelos de difusión)

La idea del modelo de difusión no es tan antigua. En el artículo de 2015 llamado "Deep Unsupervised Learning using Nonequilibrium Thermodynamics", los autores la describieron de la siguiente manera:

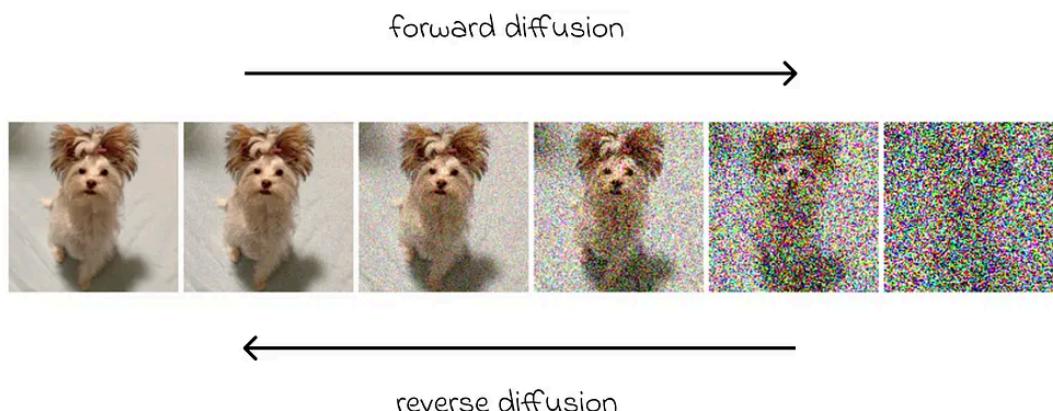
La idea esencial, inspirada en la física estadística fuera de equilibrio, es destruir sistemática y lentamente la estructura en una distribución de datos a través de un proceso de difusión hacia adelante iterativo. Luego, aprendemos un proceso de difusión inversa que restaura la estructura en los datos, produciendo un modelo generativo altamente flexible y manejable de los datos.

Imaginemos un vaso de agua transparente. ¿Qué sucede si añadimos una pequeña cantidad de otro líquido de color amarillo, por ejemplo? El líquido amarillo se esparcirá gradual y uniformemente por todo el vaso, y la mezcla resultante adquirirá un tinte amarillo ligeramente transparente.



El proceso descrito se conoce como **difusión directa** (o *forward diffusion*): alteramos el estado del entorno añadiendo una pequeña cantidad de otro líquido. Sin embargo, ¿sería igual de fácil realizar la **difusión inversa** (o *reverse diffusion*) —devolver la mezcla a su estado original? Resulta que no. En el mejor de los casos, lograrlo requeriría mecanismos muy sofisticados.

La difusión también se puede aplicar a las imágenes. Imagina una foto de alta calidad de un perro. Podemos transformar fácilmente esta imagen añadiendo gradualmente ruido aleatorio. Como resultado, los valores de los píxeles cambiarán, haciendo que el perro en la imagen sea menos visible o incluso irreconocible. Este proceso de transformación se conoce como **difusión directa** (o *forward diffusion*).



Fuente: Diffusion Models: A Comprehensive Survey of Methods and Applications

También podemos considerar la operación inversa: dada una imagen ruidosa, el objetivo es reconstruir la imagen original. Esta tarea es mucho más desafiante porque **hay muchos menos estados de imagen altamente reconocibles en comparación con el vasto número de posibles variaciones ruidosas**. Usando la misma analogía física mencionada anteriormente, este proceso se llama **difusión inversa** (o *reverse diffusion*).

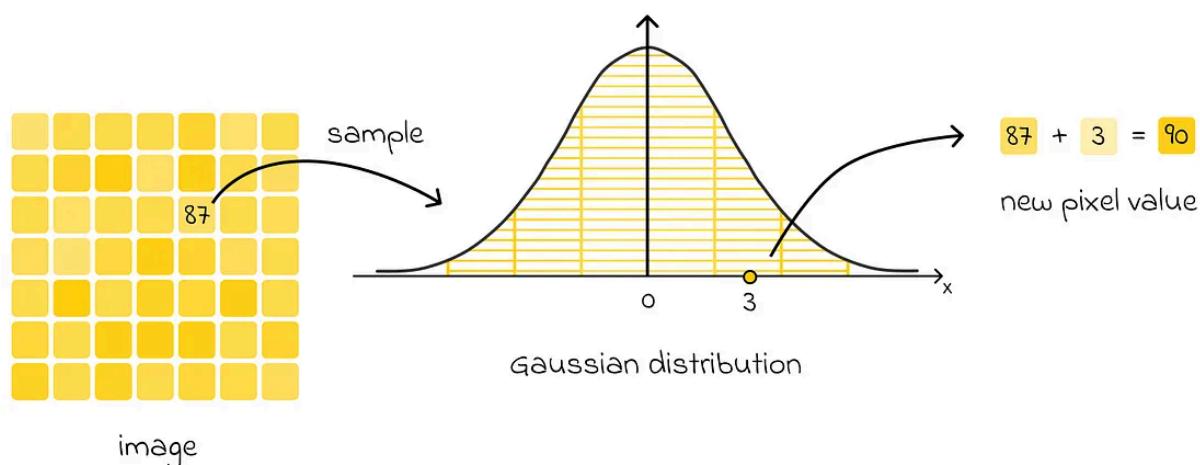
Arquitectura de los modelos de difusión

Para comprender mejor la estructura de los modelos de difusión, examinemos ambos procesos de difusión por separado.

Difusión directa

Como se mencionó anteriormente, la difusión directa implica añadir progresivamente ruido a una imagen. En la práctica, sin embargo, el proceso es un poco más matizado.

El método más común consiste en muestrear un valor aleatorio para cada píxel de una **distribución gaussiana** con una media de 0. Este valor muestreado —que puede ser positivo o negativo— se añade luego al valor original del píxel. Repetir esta operación en todos los píxeles da como resultado una versión ruidosa de la imagen original.



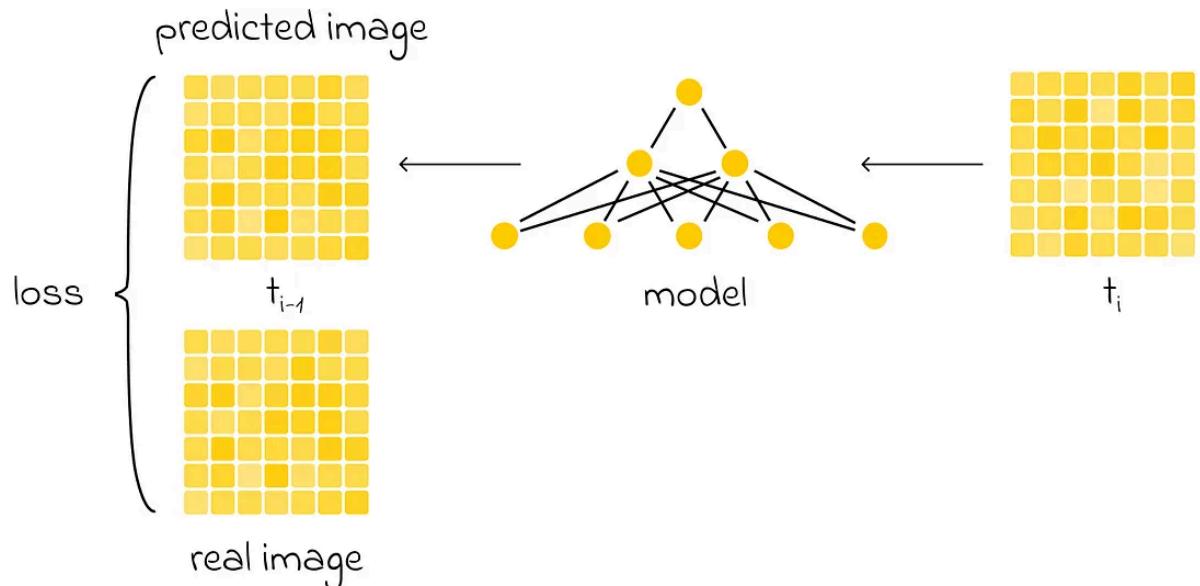
La distribución gaussiana elegida suele tener una varianza relativamente pequeña, lo que significa que los valores muestreados suelen ser pequeños. Como resultado, solo se introducen cambios menores en la imagen en cada paso.

La difusión directa es un proceso iterativo en el que se aplica ruido a la imagen múltiples veces. Con cada iteración, la imagen resultante se vuelve cada vez más diferente de la original. Después de cientos de iteraciones —lo cual es común en los modelos de difusión reales— la imagen eventualmente se vuelve indistinguible del ruido puro.

Difusión inversa

Ahora podrías preguntarte: **¿cuál es el propósito de realizar todas estas transformaciones de difusión directa?** La respuesta es que las imágenes generadas en cada iteración se utilizan para entrenar una red neuronal.

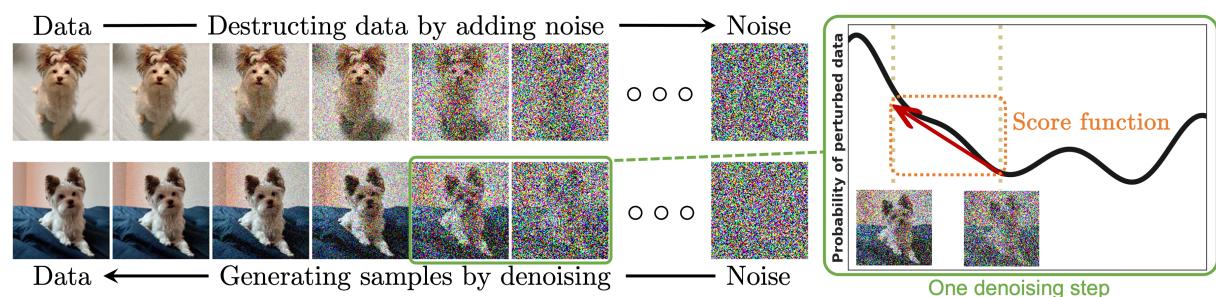
Específicamente, supongamos que aplicamos 100 transformaciones de ruido secuenciales durante la difusión directa. Podemos entonces tomar la imagen en cada paso y entrenar la red neuronal para reconstruir la imagen del paso anterior. La diferencia entre las imágenes predicha y real se calcula utilizando una función de pérdida —por ejemplo, el **Error Cuadrático Medio (ECM o MSE, por sus siglas en inglés)**, que mide la diferencia promedio píxel a píxel entre las dos imágenes.



El objetivo del modelo es detectar el ruido añadido y reconstruir la imagen anterior. La imagen predicha se compara luego con la imagen real para calcular la pérdida.

Este ejemplo muestra un modelo de difusión reconstruyendo la imagen original. Al mismo tiempo, los modelos de difusión pueden entrenarse para predecir el ruido añadido a una imagen. En ese caso, para reconstruir la imagen original, es suficiente con restar el ruido predicho de la imagen en la iteración ruidosa (o actual) para obtener la imagen del paso anterior.

Aunque ambas tareas puedan parecer similares, predecir el ruido añadido es más sencillo en comparación con la reconstrucción directa de la imagen.



Los **modelos de difusión** perturban suavemente los datos añadiendo ruido, y luego invierten este proceso para generar nuevos datos a partir del ruido. Cada paso de eliminación de ruido (o *denoising*) en el proceso inverso típicamente requiere estimar la **función de puntuación** (o *score function*; véase la figura ilustrativa a la derecha), la cual es un gradiente que apunta hacia las direcciones de los datos con mayor verosimilitud y menos ruido.

Diseño del modelo

Después de obtener una intuición básica sobre la técnica de difusión, es esencial explorar varios conceptos más avanzados para comprender mejor el diseño del modelo de difusión.

Número de iteraciones

El número de iteraciones es uno de los parámetros clave en los modelos de difusión:

- Por un lado, usar más iteraciones significa que los pares de imágenes en pasos adyacentes diferirán menos, facilitando la tarea de aprendizaje del modelo.
- Por otro lado, un mayor número de iteraciones aumenta el costo computacional.

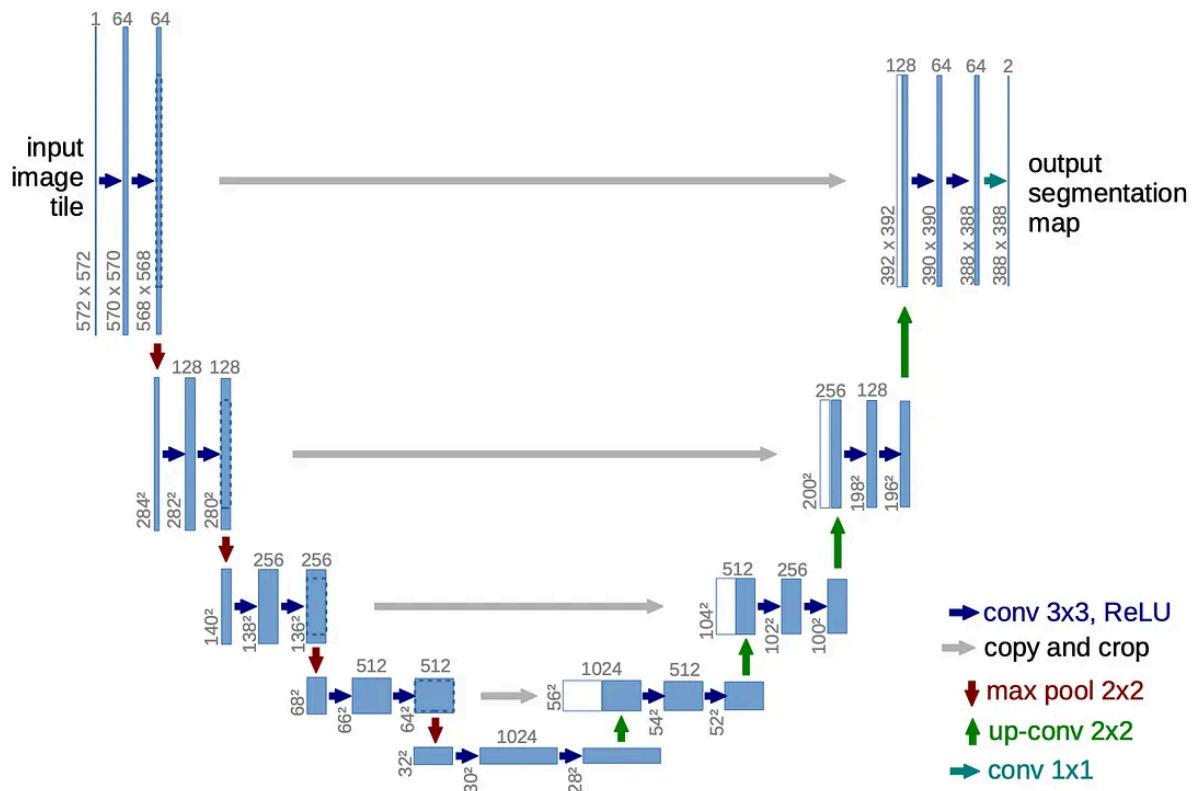
Aunque menos iteraciones pueden acelerar el entrenamiento, el modelo podría no aprender transiciones suaves entre los pasos, resultando en un rendimiento deficiente.

Típicamente, el número de iteraciones se elige entre 50 y 1000.

Arquitectura de la red neuronal

Más comúnmente, la arquitectura **U-Net** se utiliza como la estructura principal (*backbone*) en los modelos de difusión. Estas son algunas de las razones:

- U-Net conserva las dimensiones de la imagen de entrada y salida, asegurando que el tamaño de la imagen permanezca constante durante todo el proceso de difusión inversa.
- Su arquitectura de cuello de botella (*bottleneck*) permite la reconstrucción de la imagen completa después de la compresión en un **espacio latente**. Mientras tanto, las características clave de la imagen se conservan mediante **conexiones de salto** (o *skip connections*).
- Originalmente diseñada para la segmentación de imágenes biomédicas, donde la precisión a nivel de píxel es crucial, las fortalezas de U-Net se trasladan bien a las tareas de difusión que requieren la predicción precisa de los valores de los píxeles individuales.



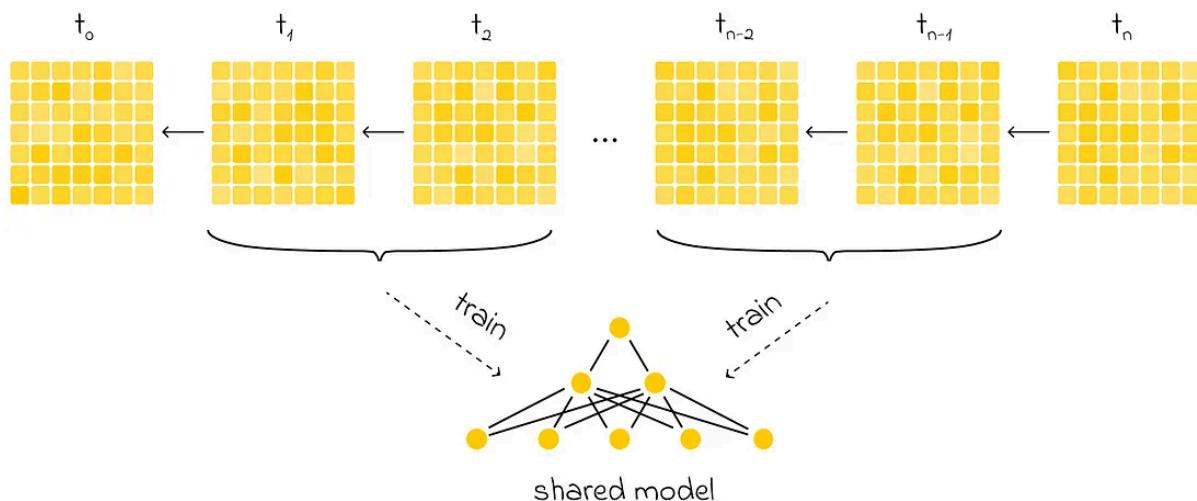
Arquitectura U-Net. Fuente: [U-Net: Convolutional Networks for Biomedical Image Segmentation](#)

Red compartida

A primera vista, podría parecer necesario entrenar una red neuronal separada para cada iteración en el proceso de difusión. Si bien este enfoque es factible y puede conducir a resultados de **inferencia** de alta calidad, es altamente ineficiente desde una perspectiva computacional. Por ejemplo, si el proceso de difusión consta de mil pasos, necesitaríamos entrenar mil modelos U-Net, una tarea extremadamente costosa en tiempo y recursos.

Sin embargo, podemos observar que **la configuración de la tarea en las diferentes iteraciones es esencialmente la misma**: en cada caso, necesitamos reconstruir una imagen de dimensiones idénticas que ha sido alterada con ruido de una magnitud similar. Esta importante observación lleva a la idea de **utilizar una única red neuronal compartida en todas las iteraciones**.

En la práctica, esto significa que utilizamos un único modelo U-Net con pesos compartidos, entrenado con pares de imágenes de diferentes pasos de difusión. Durante la inferencia, la imagen ruidosa pasa a través de la misma U-Net entrenada múltiples veces, refinándola gradualmente hasta que se produce una imagen de alta calidad.



Aunque la calidad de la generación podría deteriorarse ligeramente debido al uso de un solo modelo, la ganancia en velocidad de entrenamiento se vuelve muy significativa.

Utilidad y aplicaciones de los modelos de difusión

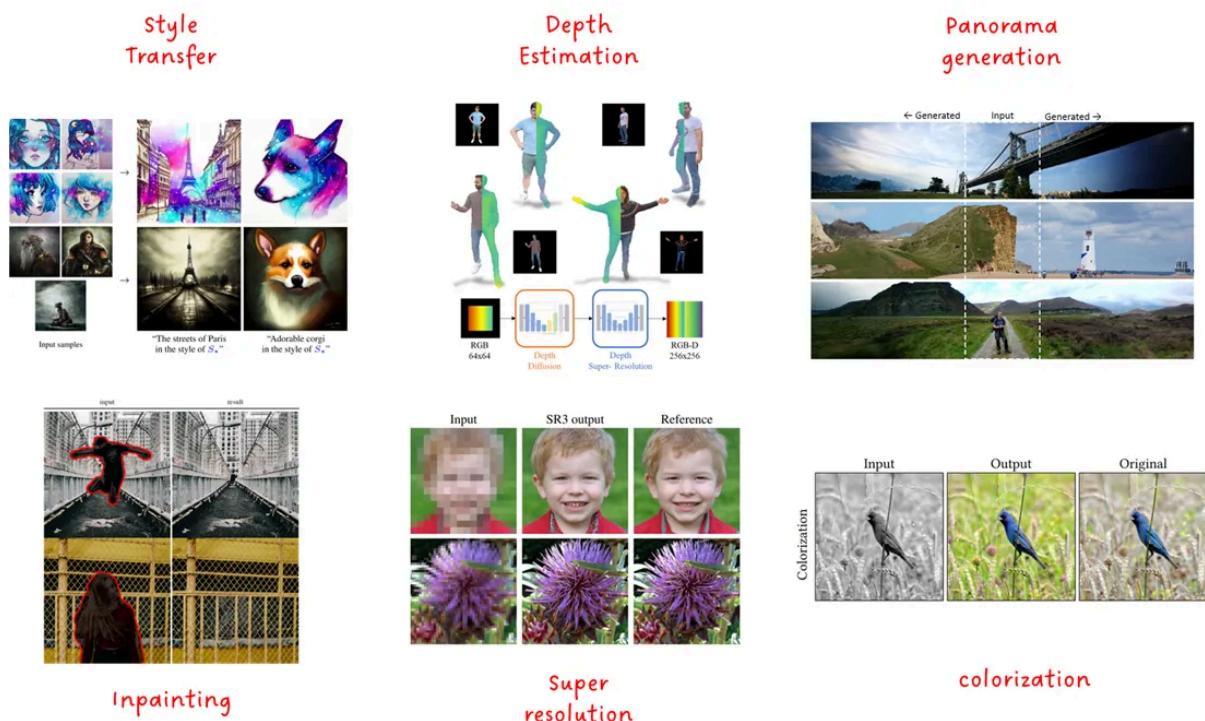
En su esencia, el proceso completo de difusión nos ofrece tres ventajas:

- 1. Generación de alta calidad:** Los modelos de difusión generan datos con una calidad y realismo excepcionales, a menudo superando a modelos generativos anteriores en muchas tareas. Esto se deriva de su capacidad para aprender la distribución de datos subyacente de manera granular a través del proceso iterativo de "desruido" (denoising). La refinación lenta y constante desde el ruido puro hasta una muestra de datos coherente resulta en salidas altamente realistas.
- 2. Versatilidad:** Los modelos de difusión son notablemente flexibles y se pueden aplicar a una amplia gama de modalidades de datos, incluyendo imágenes, audio, moléculas y más. Esta versatilidad proviene del mecanismo central del modelo de manipulación de ruido, un concepto que se puede aplicar a cualquier tipo de datos representados digitalmente. Ya

sean píxeles en una imagen, amplitudes en una onda sonora o átomos en una molécula, los modelos de difusión pueden aprender a generarlos y manipularlos. La difusión también puede ser moldeada en diferentes casos de uso.

3. **Control Paso a Paso:** El proceso de generación paso a paso en los modelos de difusión permite a los usuarios ejercer un mayor control sobre el resultado final. A diferencia de los modelos generativos tradicionales que producen un resultado de una sola vez, los modelos de difusión refinan progresivamente los datos generados desde el ruido hasta una muestra final. Esto nos da mayor transparencia y la capacidad de intervenir en el medio para llevar los experimentos en nuevas direcciones.

Aplicaciones



Diffusion Models: A Comprehensive Survey of Methods and Applications: <https://arxiv.org/pdf/2209.00796.pdf>

<https://medium.com/data-science-collective/diffusion-models-explained-simply-3a41fea596e0>

NeRF (Neural Radiance Fields)

Los modelos NeRF (Neural Radiance Fields) son un tipo de red neuronal profunda diseñada para representar escenas 3D de manera compacta y generar imágenes fotorrealistas desde cualquier ángulo de visión deseado. Los NeRF modelan la densidad de volumen y el color en un espacio 3D, aprendiendo estos atributos a partir de un conjunto de imágenes 2D de la

escena desde diferentes perspectivas. Su paper representativo es [Representing Scenes as Neural Radiance Fields for View Synthesis](https://www.matthewtancik.com/nerf) (<https://www.matthewtancik.com/nerf>).

La idea central detrás de los NeRF es utilizar una red neuronal para mapear coordenadas espaciales 3D y direcciones de visualización a valores de color y densidad. Al pasar estos valores a través de un proceso de renderizado basado en volumen, se puede generar una imagen 2D. Este proceso utiliza un algoritmo de integración de volumen para simular cómo la luz atraviesa el espacio 3D, acumulando contribuciones de color y opacidad para formar la imagen final.

Los NeRF son particularmente potentes debido a su capacidad para generar imágenes con detalles precisos y efectos de iluminación complejos, como sombras suaves y translucidez, que son desafiantes para otras técnicas de modelado 3D. Sin embargo, también tienen limitaciones, como tiempos de entrenamiento y renderización largos y la necesidad de una gran cantidad de datos de entrenamiento de alta calidad.

Este enfoque ha encontrado aplicaciones en áreas como la realidad aumentada, efectos visuales en películas y en la creación de entornos virtuales realistas para simulaciones y videojuegos.

<https://www.youtube.com/watch?v=JuH79E8rdKc>

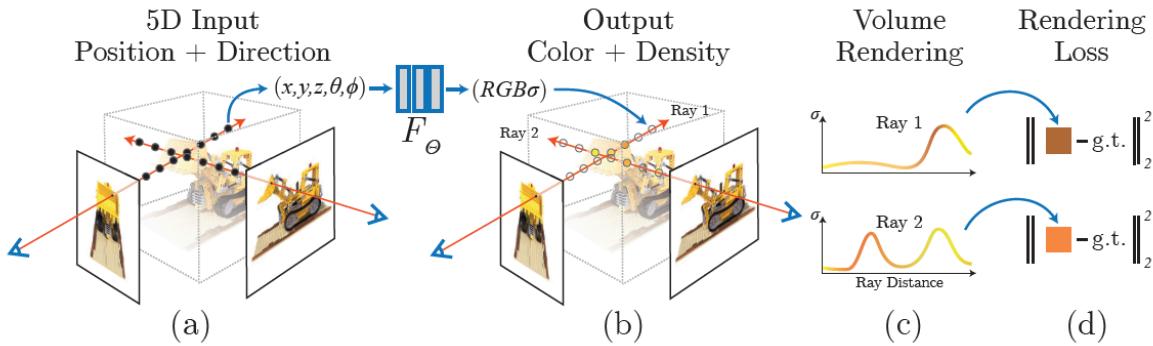
<https://github.com/bmild/nerf>

5 Dimensiones

El término "5D" se refiere a las cinco dimensiones de entrada que la red neuronal utiliza para modelar un campo de radiación. Estas cinco dimensiones son:

- 1. Coordenadas Espaciales 3D (x, y, z):** Estas tres dimensiones representan la posición en el espacio. Cada punto en el espacio 3D donde se desea estimar la densidad de volumen y el color se describe mediante estas coordenadas.
- 2. Dirección de Visualización (θ, φ):** Estas dos dimensiones adicionales representan la dirección desde la cual se observa el punto en el espacio. En práctica, esto puede ser un vector de dirección o ángulos que describen cómo está orientada la cámara respecto al punto que se está evaluando.

La combinación de estas cinco dimensiones permite al modelo NeRF entender no solo cómo es la estructura tridimensional de la escena, sino también cómo cambian la apariencia y la luminosidad de los objetos en la escena en función del ángulo desde el cual se observan. Esta capacidad es fundamental para generar imágenes fotorrealistas desde nuevas perspectivas, incorporando correctamente efectos como la vista paralaje y la reflexión especular.



Una descripción general de nuestra representación de escena del campo de radiación neuronal y el procedimiento de renderizado diferenciable. Se sintetizan imágenes muestreando coordenadas 5D (ubicación y dirección de visualización) a lo largo de los rayos de la cámara (a), introduciendo esas ubicaciones en un MLP para producir un color y una densidad de volumen (b), y utilizando técnicas de renderizado de volumen para componer estos valores en una imagen (c). Esta función de renderizado es diferenciable, por lo que podemos optimizar nuestra representación de la escena minimizando el residuo entre las imágenes sintetizadas y las imágenes observadas verdaderas (d).

Entrada del modelo:

1. Para el Entrenamiento del Modelo (Optimización de la Representación):

- **Un conjunto de imágenes 2D:** El paper menciona la utilización de "un conjunto disperso de vistas de entrada" y que "la única entrada requerida para optimizar nuestra representación es un conjunto de imágenes con poses de cámara conocidas".
- **Poses de cámara correspondientes a cada imagen:** Esto incluye la posición y orientación de la cámara para cada imagen.
- **Parámetros intrínsecos de la cámara:** El paper menciona que para cada escena se requiere "...el conjunto de imágenes RGB capturadas de la escena, las poses de cámara correspondientes y los parámetros intrínsecos, y los límites de la escena".
- **Límites de la escena (Scene bounds):** También mencionados como necesarios para el entrenamiento.

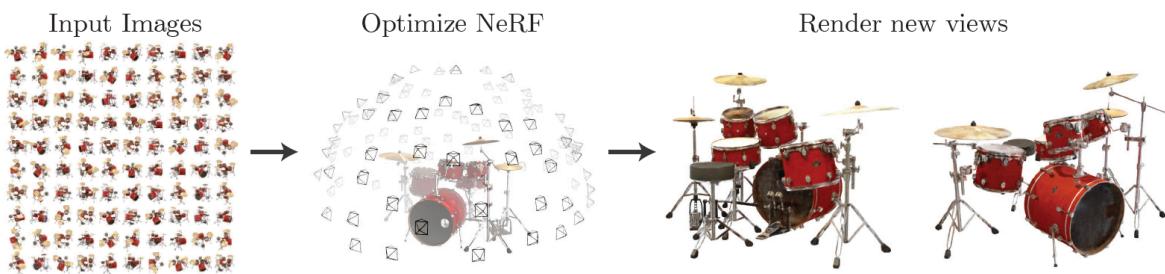
2. Para la Red Neuronal (MLP) durante el Entrenamiento y la Inferencia/Renderizado:

- La entrada directa a la red neuronal (un MLP, Perceptrón Multicapa) es una **coordenada continua 5D**. Esta se compone de:
 - **Ubicación espacial (posición) 3D:** (x, y, z) .
 - **Dirección de visualización 2D:** (θ, ϕ) . El paper también aclara que en la práctica expresan la dirección como un vector unitario Cartesiano 3D, d .

Para renderizar una nueva vista, se interrogan estas coordenadas 5D a lo largo de los rayos de la cámara. El proceso implica:

- Marchar rayos de cámara a través de la escena para generar un conjunto de puntos 3D muestreados.
- Usar esos puntos y sus correspondientes direcciones de visualización 2D como entrada a la red neuronal.

Salida del modelo:



1. Salida de la Red Neuronal (MLP) para una Coordenada 5D Específica:

- **Densidad volumétrica (σ):** Es un escalar que representa la opacidad diferencial, controlando cuánta radiancia acumula un rayo que pasa por (x, y, z) . La red predice la densidad volumétrica σ como una función únicamente de la ubicación x .
- **Radiancia emitida dependiente de la vista (Color RGB):** Es el color (R, G, B) emitido en esa ubicación espacial para esa dirección de visualización. La red permite que el color RGB c sea predicho como una función tanto de la ubicación como de la dirección de visualización.

2. Salida Final del Proceso NeRF (Renderizado de una Nueva Vista):

- **Una imagen 2D:** Se sintetizan vistas proyectando los colores y densidades de salida (de la MLP) en una imagen utilizando técnicas clásicas de renderizado volumétrico. El color esperado $C(r)$ de un rayo de cámara se calcula integrando la radiancia y la densidad a lo largo del rayo.

Google Colab

CO https://colab.research.google.com/github/bmild/nerf/blob/master/tiny_nerf.ipynb



3D Gaussian Splatting

El **3D Gaussian Splatting** es una técnica utilizada en visualización de datos y procesamiento de imágenes, particularmente en el contexto de la renderización volumétrica y la reconstrucción de imágenes. Este método involucra la representación de datos tridimensionales dispersos mediante el uso de funciones Gaussianas para suavizar y combinar puntos de datos en una representación continua en el espacio 3D. Esta técnica se presenta en el paper [3D Gaussian Splatting for Real-Time Radiance Field Rendering](https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/) (<https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/>)

El proceso de "splatting" se refiere a la proyección de un objeto, como un punto de datos, hacia una grilla o superficie, extendiéndose como una mancha. En el contexto de Gaussian Splatting, cada punto de datos se representa como una mancha Gaussiana, lo que significa que el impacto de un punto se distribuye de manera suave y decreciente desde el centro del

punto siguiendo una función Gaussiana. La intensidad y el tamaño de esta mancha están definidos por el ancho de banda de la función Gaussiana, también conocido como la desviación estándar de la distribución.

https://www.youtube.com/watch?v=T_kXY43VZnk

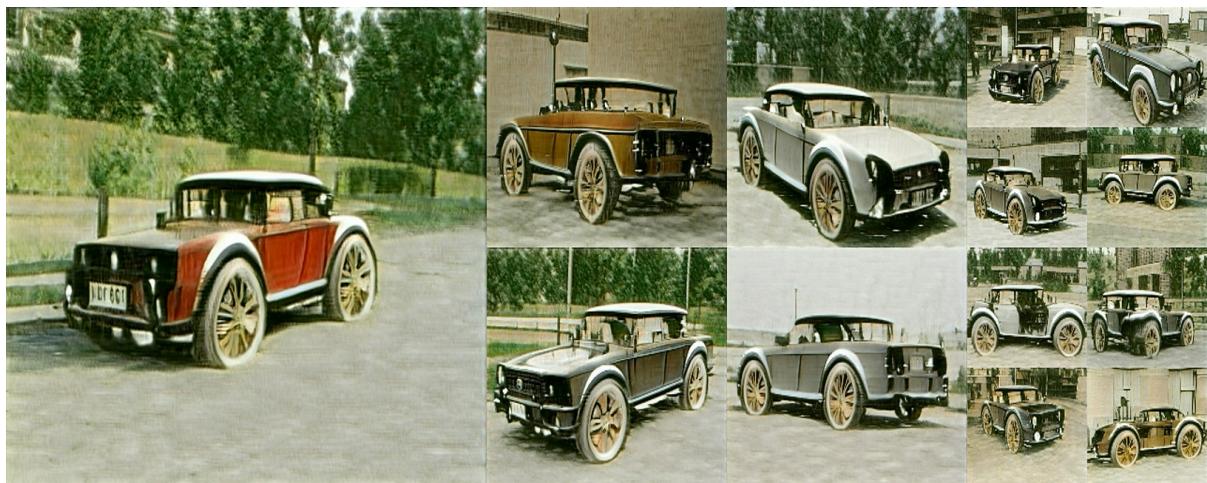
<https://github.com/graphdeco-inria/gaussian-splatting>

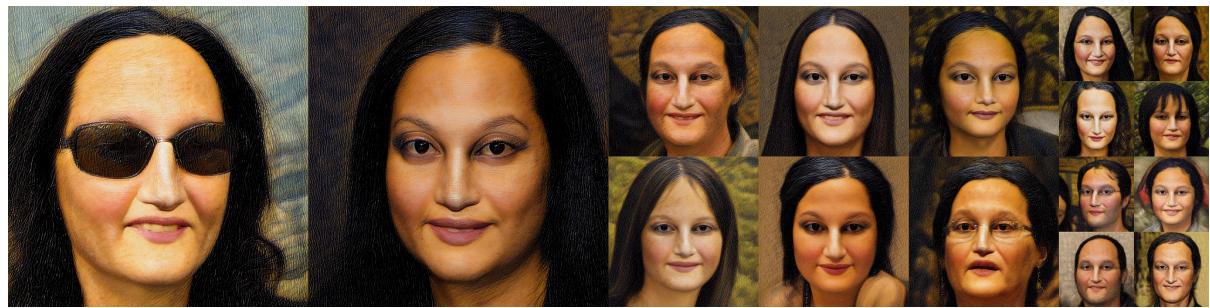
Modelos para explorar

StyleGAN-NADA: CLIP-Guided Domain Adaptation of Image Generators

Paper: <https://arxiv.org/abs/2108.00946>

<https://github.com/rinongal/StyleGAN-nada>

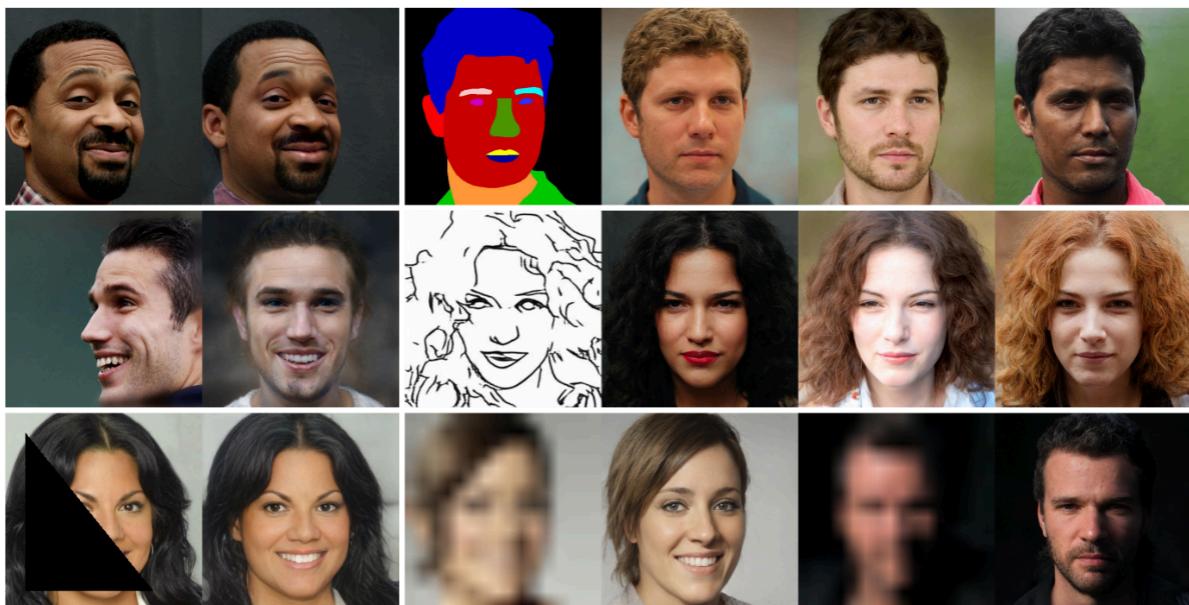




Encoding in Style: a StyleGAN Encoder for Image-to-Image Translation

Paper: <https://arxiv.org/abs/2008.00951>

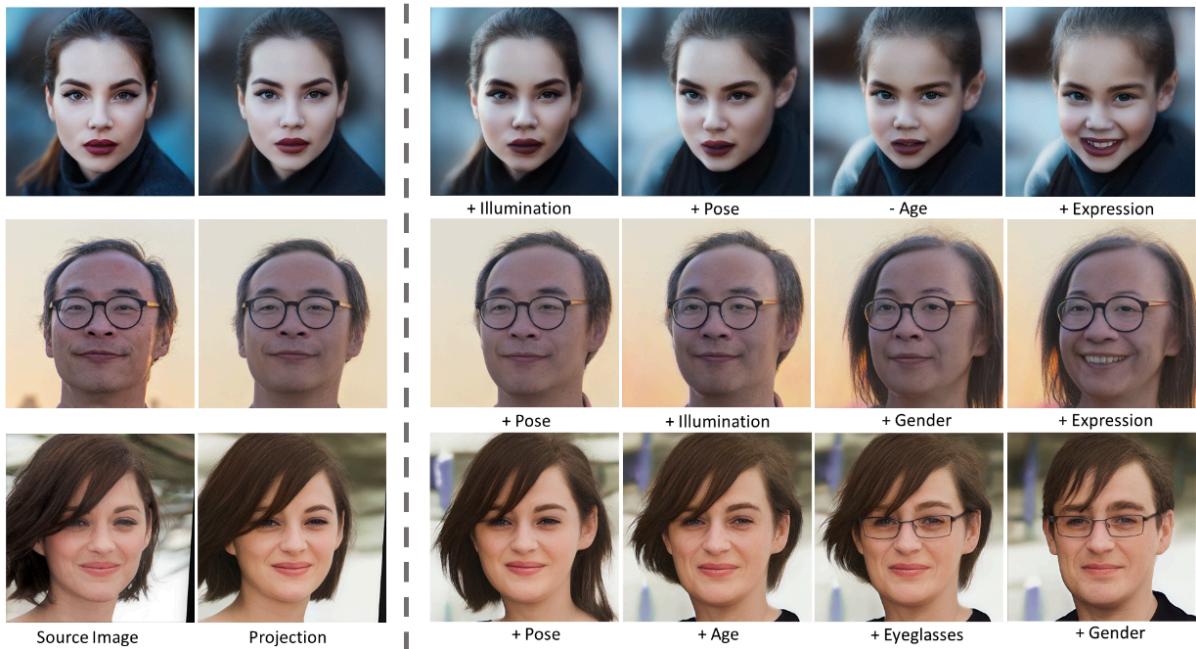
<https://github.com/eladrich/pixel2style2pixel>



StyleFlow: Attribute-conditioned Exploration of StyleGAN-generated Images using Conditional Continuous Normalizing Flows

Paper: <https://dl.acm.org/doi/10.1145/3447648>

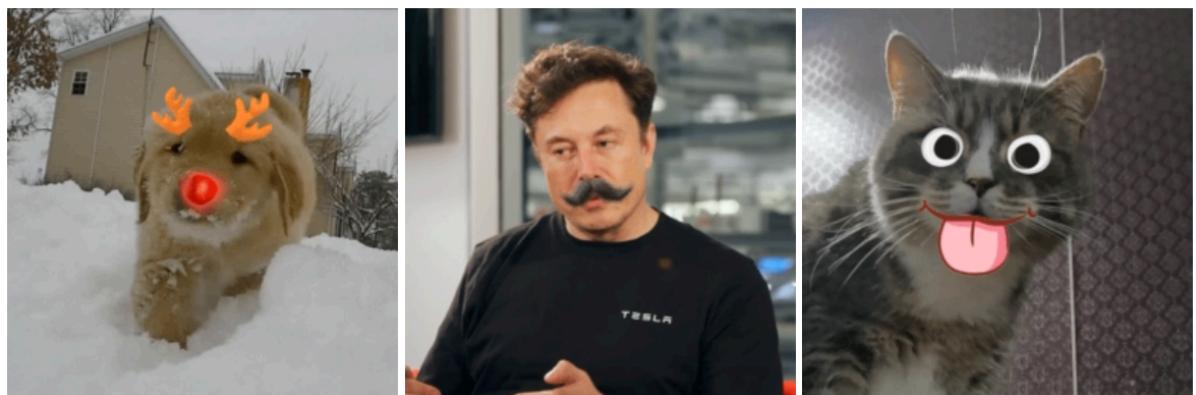
<https://github.com/RameenAbdal/StyleFlow>



GAN-Supervised Dense Visual Alignment

Paper: <https://arxiv.org/abs/2112.05143>

<https://github.com/wpeebles/gangealing>



HyperStyle: StyleGAN Inversion with HyperNetworks for Real Image Editing

Paper: <https://arxiv.org/abs/2111.15666>

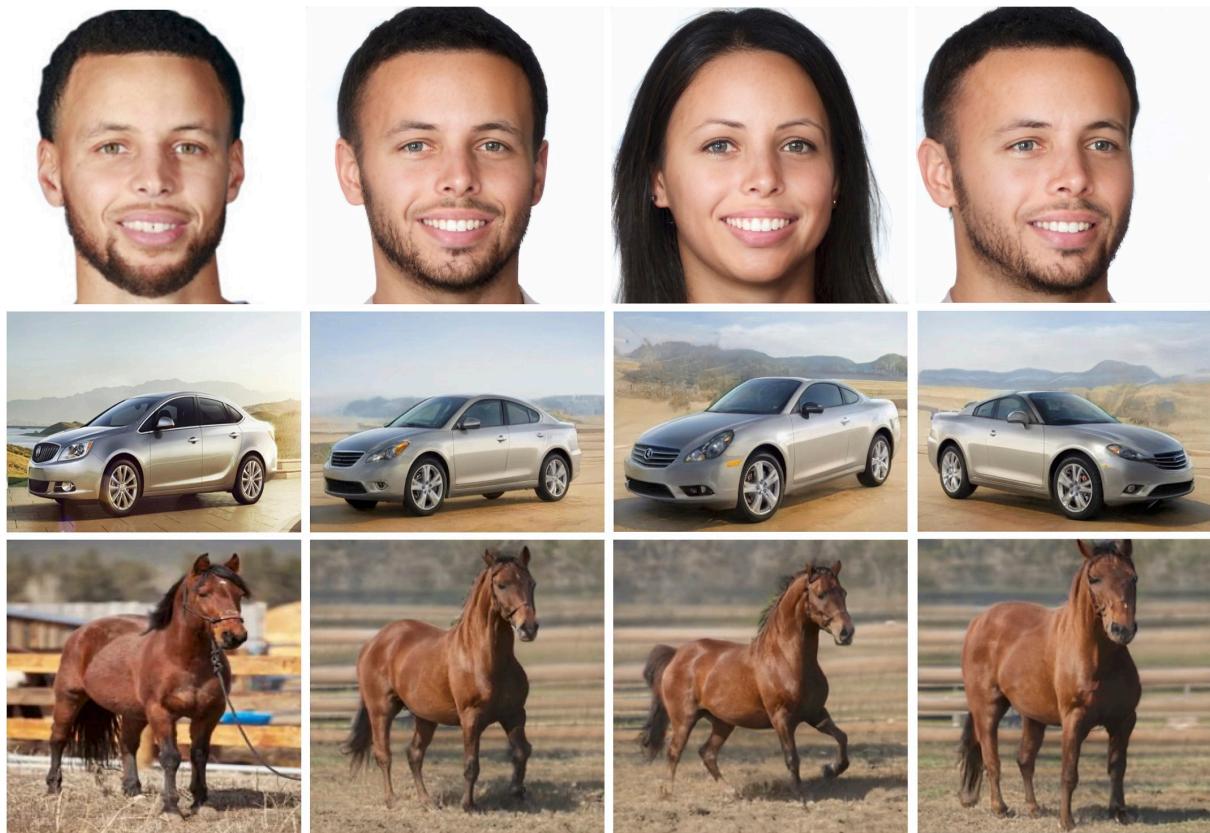
<https://github.com/yuval-alaluf/hyperstyle>



Designing an Encoder for StyleGAN Image Manipulation

Paper: <https://arxiv.org/abs/2102.02766>

<https://github.com/omertov/encoder4editing>



Third Time's the Charm? Image and Video Editing with StyleGAN3

Paper: <https://arxiv.org/abs/2201.13433>

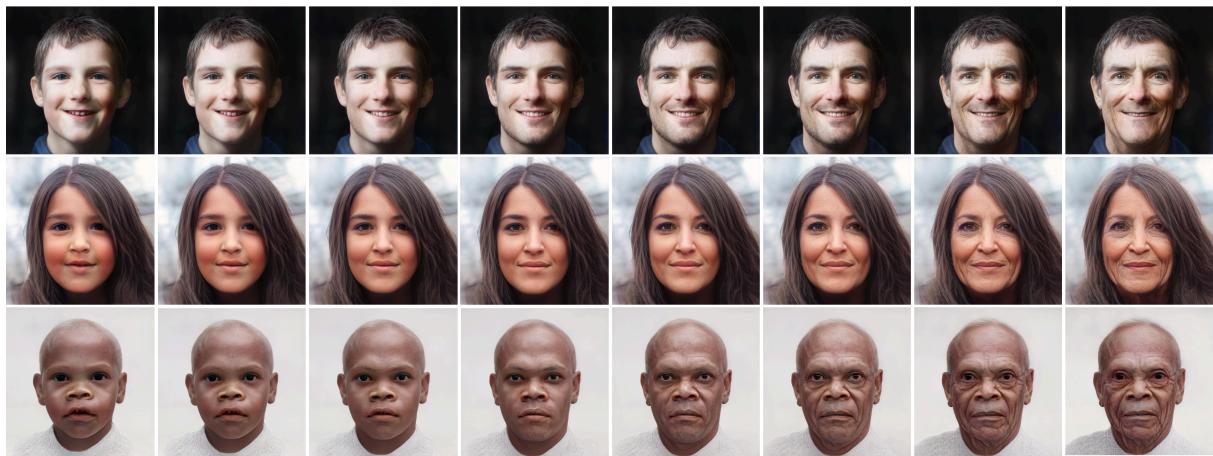
<https://github.com/yuval-alaluf/stylegan3-editing>



Only a Matter of Style: Age Transformation Using a Style-Based Regression Model

Paper: <https://arxiv.org/abs/2102.02754>

<https://github.com/yuval-alaluf/SAM>



StarGAN v2: Diverse Image Synthesis for Multiple Domains

Paper: <https://arxiv.org/abs/1912.01865>

<https://github.com/clovaai/stargan-v2>

<https://youtu.be/0EVh5Ki4dIY>

FSGAN: Subject Agnostic Face Swapping and Reenactment

Paper: <https://arxiv.org/pdf/1908.05932.pdf>

<https://github.com/YuvalNirkin/fsgan>

<https://www.youtube.com/watch?v=BsITEVX6hkE>

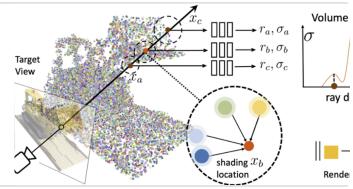
Point-NeRF: Point-based Neural Radiance Fields

Paper: <https://arxiv.org/pdf/2201.08845.pdf>

[GitHub - Xharlie/pointnerf: Point-NeRF: Point-based Neural Radiance Fields](https://github.com/Xharlie/pointnerf)

Point-NeRF: Point-based Neural Radiance Fields. Contribute to Xharlie/pointnerf development by creating an account on GitHub.

 <https://github.com/Xharlie/pointnerf?tab=readme-ov-file>



Más recursos de modelos generativos

GANs

<https://github.com/enochkan/awesome-gans-and-deepfakes>

<https://github.com/nashory/gans-awesome-applications>

<https://github.com/hindupuravinash/the-gan-zoo>

VAEs

<https://github.com/matthewvowels1/Awesome-VAEs>

Diffusion Models

<https://github.com/diffusion/Awesome-Diffusion-Models>

Nerf

<https://github.com/awesome-NeRF/awesome-NeRF>

3D Gaussian Splatting

<https://github.com/MrNeRF/awesome-3D-gaussian-splatting>

Datasets

<https://github.com/switchablenorms/CelebAMask-HQ>

<https://github.com/yumingj/DeepFashion-MultiModal>

Varios

<https://github.com/fnzhan/Generative-AI>