

# Generative Adversial Networks

Deep Learning



Marco Teran  
Universidad Sergio Arboleda

2023

# Contenido

- 1 Introducción
- 2 Generative Adversarial Networks (GANs)
  - Entrenando GANs
- 3 GANs: avances recientes
- 4 Autoencoders
  - Estructura y Función de un Autoencoder
  - Stacked Autoencoders
  - Entrenamiento no supervisado con Autoencoders Apilados
  - Autoencoders en Redes Convolucionales
  - Denoising Autoencoders
- 5 Sparse Autoencoders
- 6 Variational Autoencoders (VAEs)
- 7 Resumen

# ¿Cuál rostro es real?



# **Aprendizaje supervisado vs. No supervisado**

# Aprendizaje supervisado vs. No supervisado

## Aprendizaje supervisado

- **Datos:**  $(x, y)$   
donde  $x$  es un dato,  $y$  es una etiqueta
- **Objetivo:** Aprender la función de mapeo  
 $x \rightarrow y$
- **Aplicaciones:** Clasificación, regresión,  
detección de objetos, segmentación  
semántica, etc.

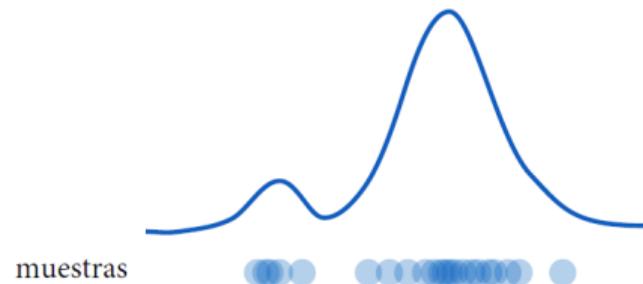
## Aprendizaje no supervisado

- **Datos:**  $x$   
 $x$  es un dato, no hay etiquetas
- **Objetivo:** Aprender alguna estructura  
oculta o subyacente de los datos  
 $x \rightarrow y$
- **Aplicaciones:** Agrupación, reducción de  
características o dimensionalidad, etc.

# Modelado generativo

**Objetivo:** Tomar como entrada muestras de entrenamiento de alguna distribución y aprender un modelo que represente esa distribución Estimación de la densidad

Estimación de la densidad



Generación de muestras



Input samples

Generated samples

Training data  $\sim P_{data}(x)$

Generated  $\sim P_{model}(x)$

¿Cómo podemos aprender la  $P_{model}(x)$  similar a la  $P_{data}(x)$ ?

# ¿Por qué modelos generativos? Debiasing

Capacidad de descubrir las variables latentes subyacentes en un conjunto de datos



VS



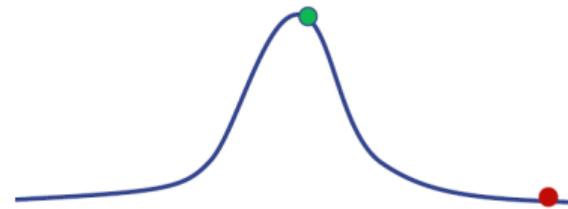
Color de piel homogéneo, postura

Color de piel diverso, postura, iluminación

¿Cómo podemos usar las distribuciones latentes para crear conjuntos de datos justos y representativos?

# ¿Por qué modelos generativos? Detección de valores atípicos (outliers)

- **Problema:** ¿Cómo podemos detectar cuando nos encontramos con algo nuevo o raro?
  - **Estrategia:** Apalancar la generación de modelos, detectan valores atípicos en la distribución
  - Utilizan valores atípicos durante el entrenamiento para mejorar aún más
- El 95% de los datos de conducción:**
- (1) soleado, (2) autopista, (3) carretera recta



Detectar valores atípicos para evitar comportamientos impredecibles durante el entrenamiento



Casos extremos



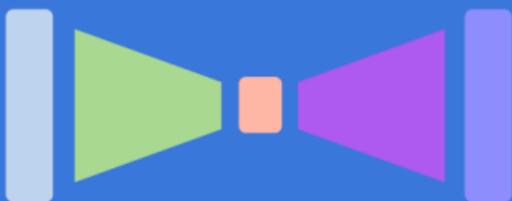
Mal clima



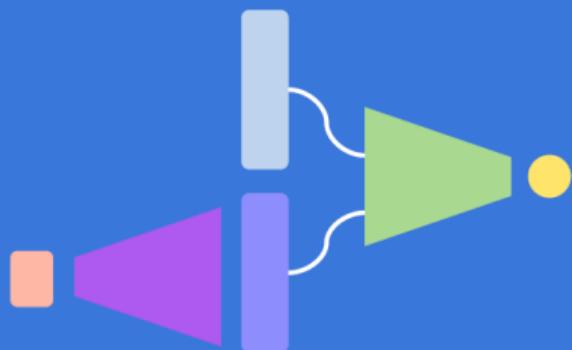
Peatones

# Modelos variables latentes

Autoencoders and Variational  
Autoencoders (VAEs)



Generative Adversarial  
Networks (GANs)



# ¿Qué es una variable latente?

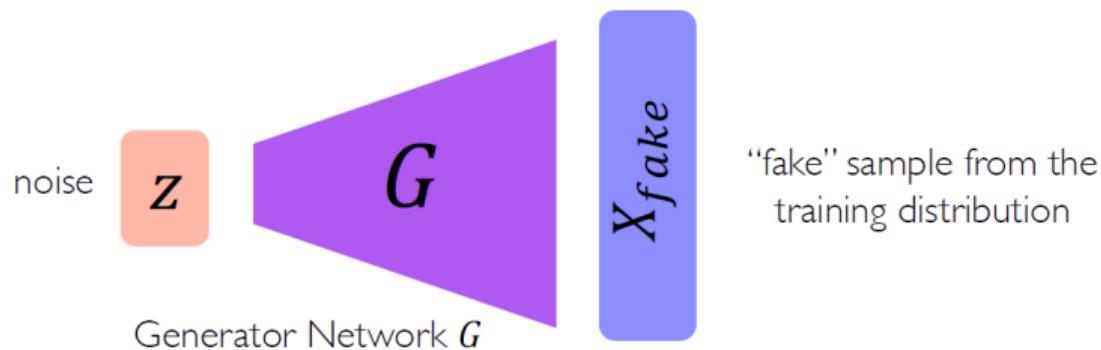


¿Podemos aprender **verdaderos factores explicativos**, por ejemplo, las variables latentes, sólo a partir de los datos observados?

# **Generative Adversarial Networks (GANs)**

# ¿Y si sólo queremos tomar una muestra?

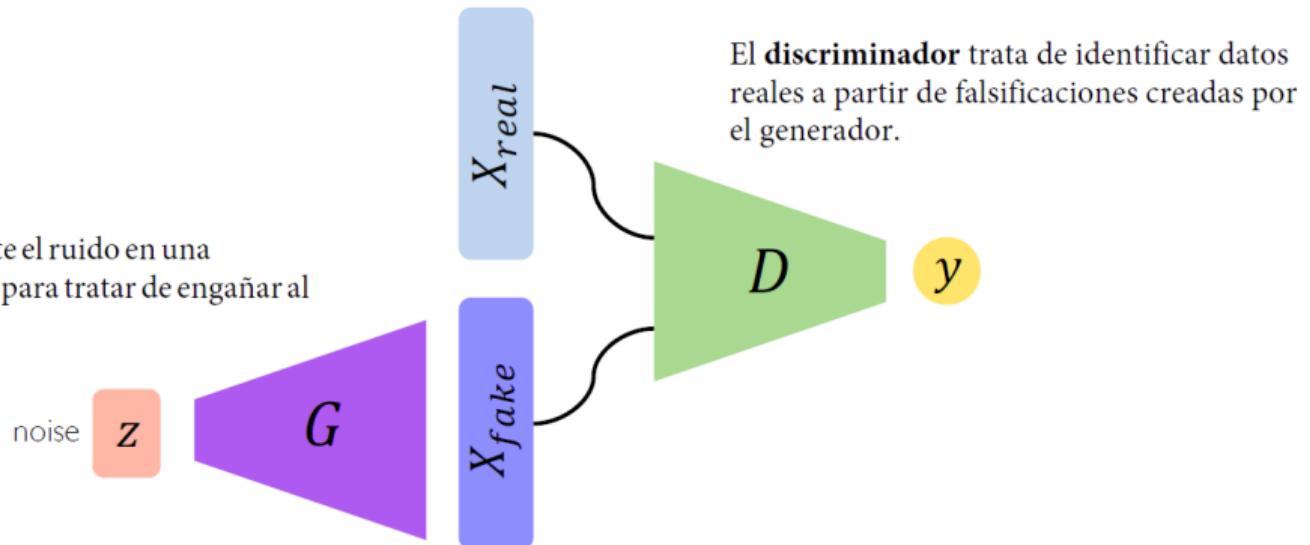
- **Idea:** no modelar explícitamente la densidad, y en su lugar sólo muestrear para generar nuevas instancias.
- **Problema:** querer tomar muestras de una distribución compleja... ¡no se puede hacer esto directamente!
- **Solución:** muestrear a partir de algo simple (ruido), aprender una transformación a la distribución de la formación.



# Generative Adversarial Networks (GANs)

Las **Redes Generativas Adversas (GANs)** son una forma de hacer un modelo generativo haciendo que dos redes neuronales compitan entre sí.

El **generador** convierte el ruido en una imitación de los datos para tratar de engañar al discriminador.



# Intuición detrás de las GANs

El **generador** parte del ruido para intentar crear una imitación de los datos.

Generator



Fake data

# Intuición detrás de las GANs

El **discriminador** mira tanto los datos reales como los falsos creados por el generador.



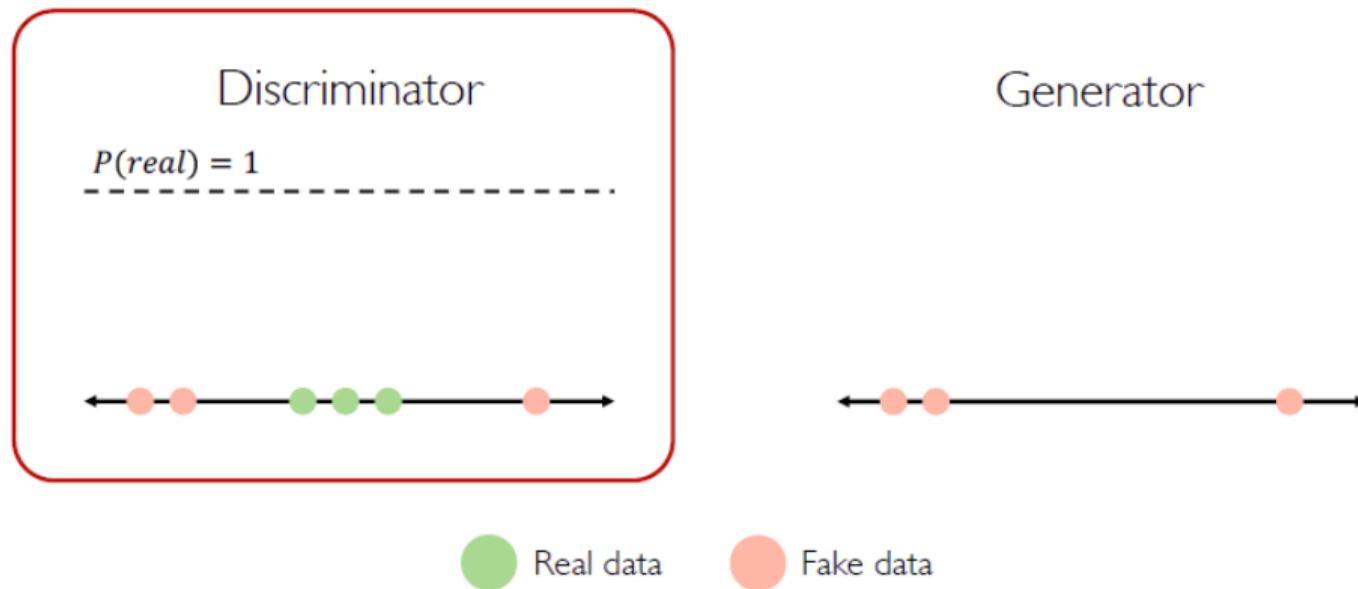
# Intuición detrás de las GANs

El **discriminador** mira tanto los datos reales como los falsos creados por el generador.



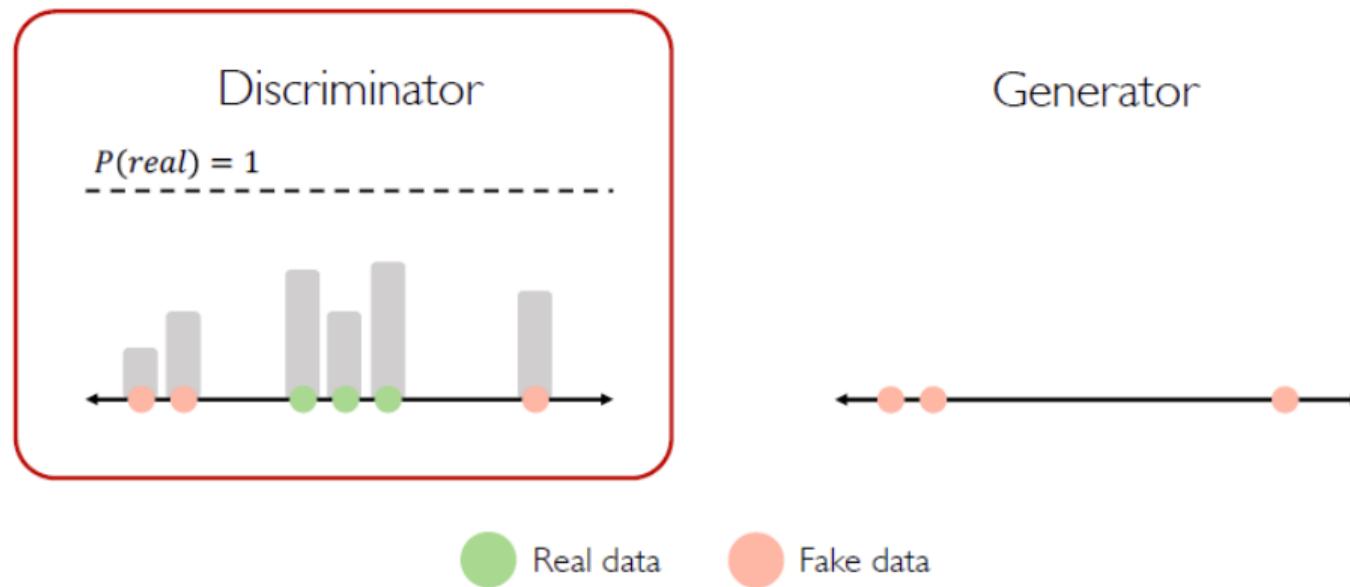
# Intuición detrás de las GANs

El **discriminador** trata de predecir lo que es real y lo que es falso.



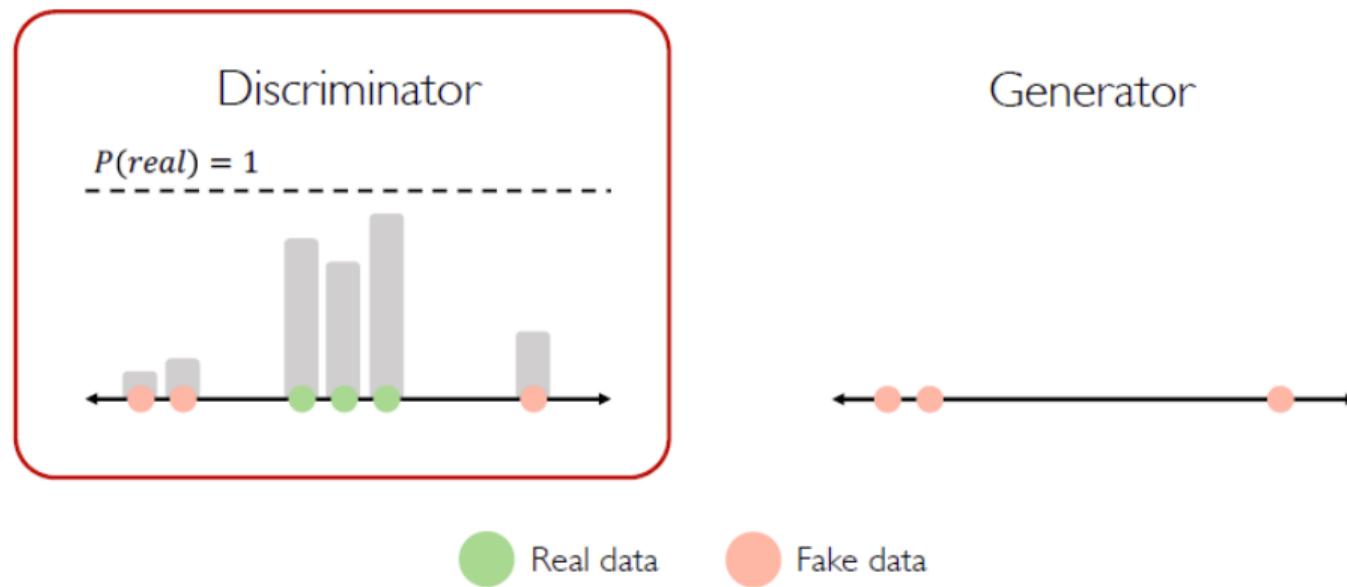
# Intuición detrás de las GANs

El **discriminador** trata de predecir lo que es real y lo que es falso.



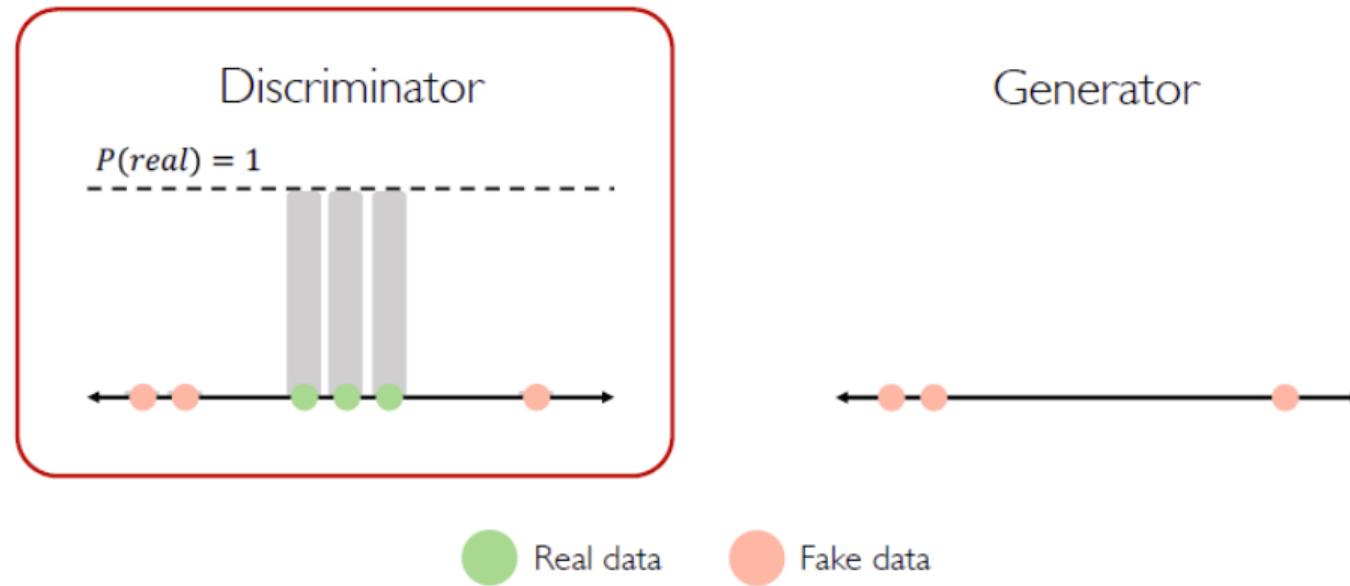
# Intuición detrás de las GANs

El **discriminador** trata de predecir lo que es real y lo que es falso.



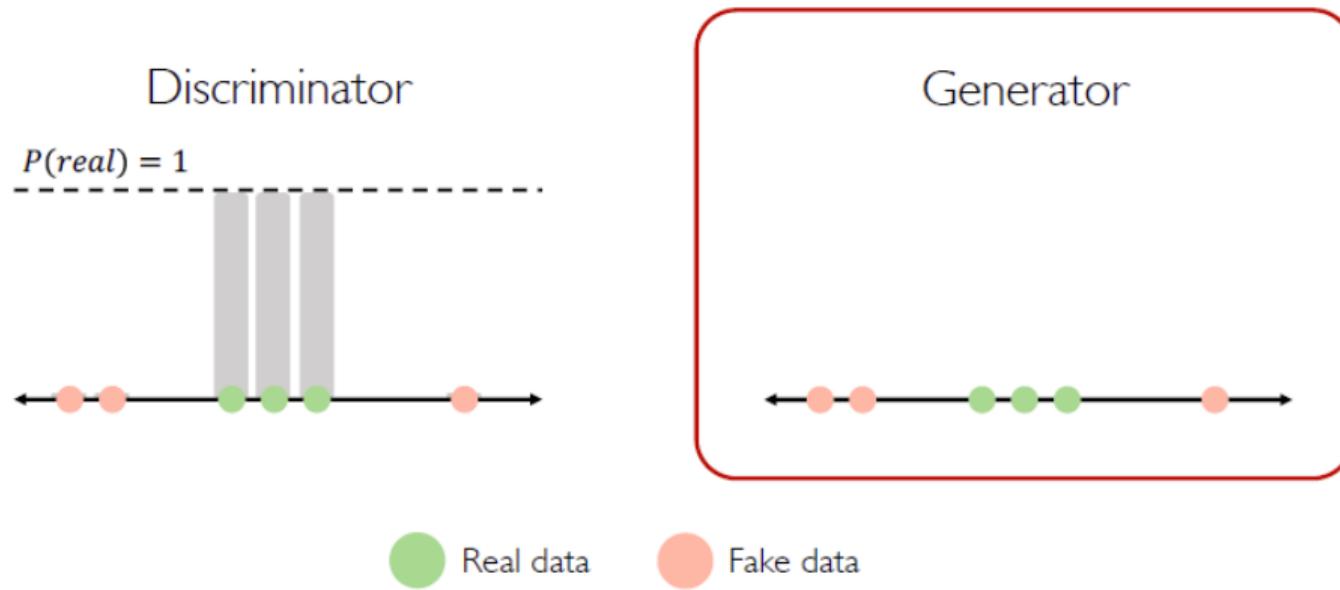
# Intuición detrás de las GANs

El **discriminador** trata de predecir lo que es real y lo que es falso.



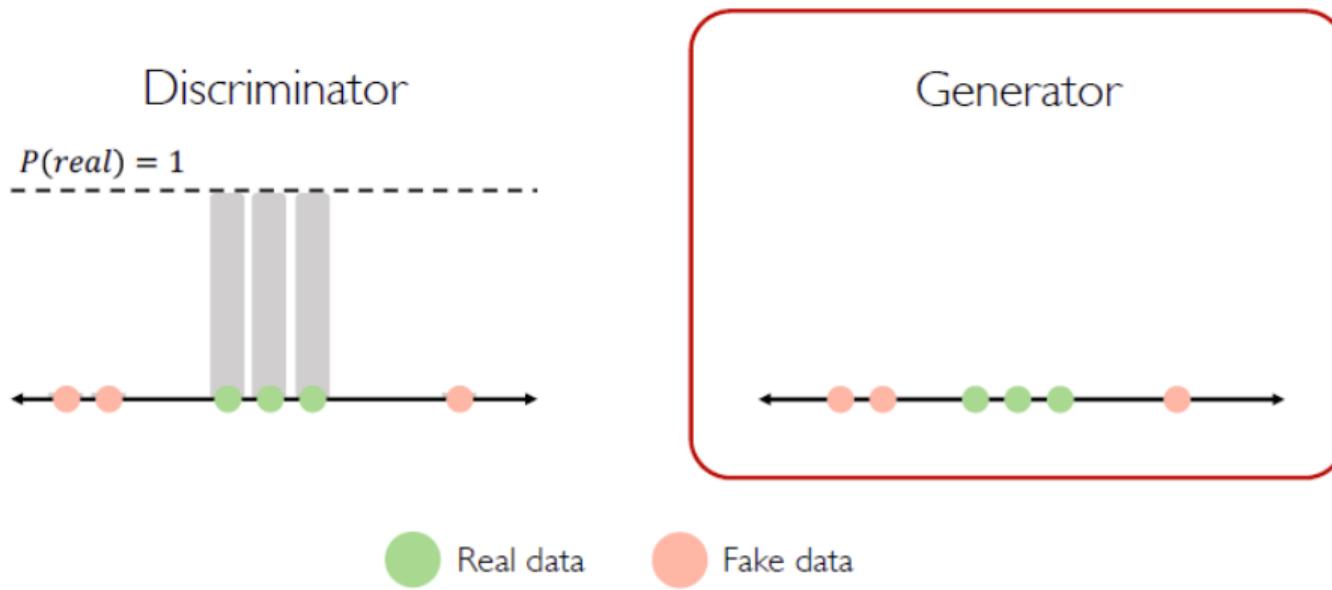
# Intuición detrás de las GANs

El **generador** trata de mejorar su imitación de los datos.



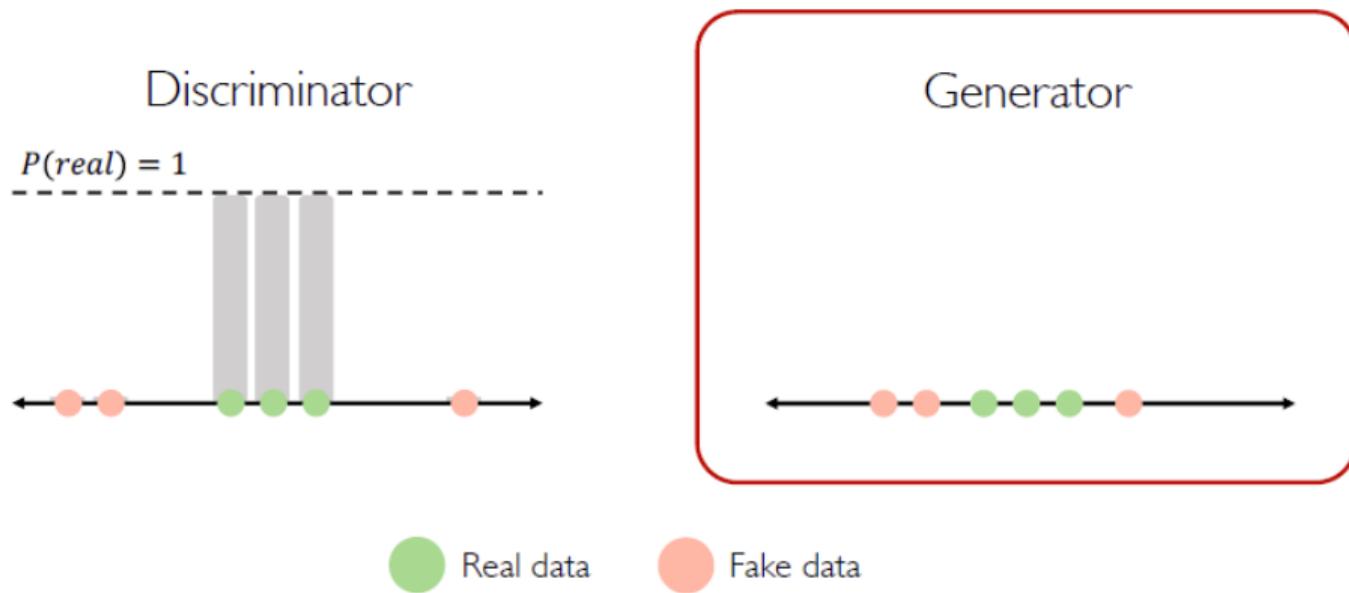
# Intuición detrás de las GANs

El **generador** trata de mejorar su imitación de los datos.



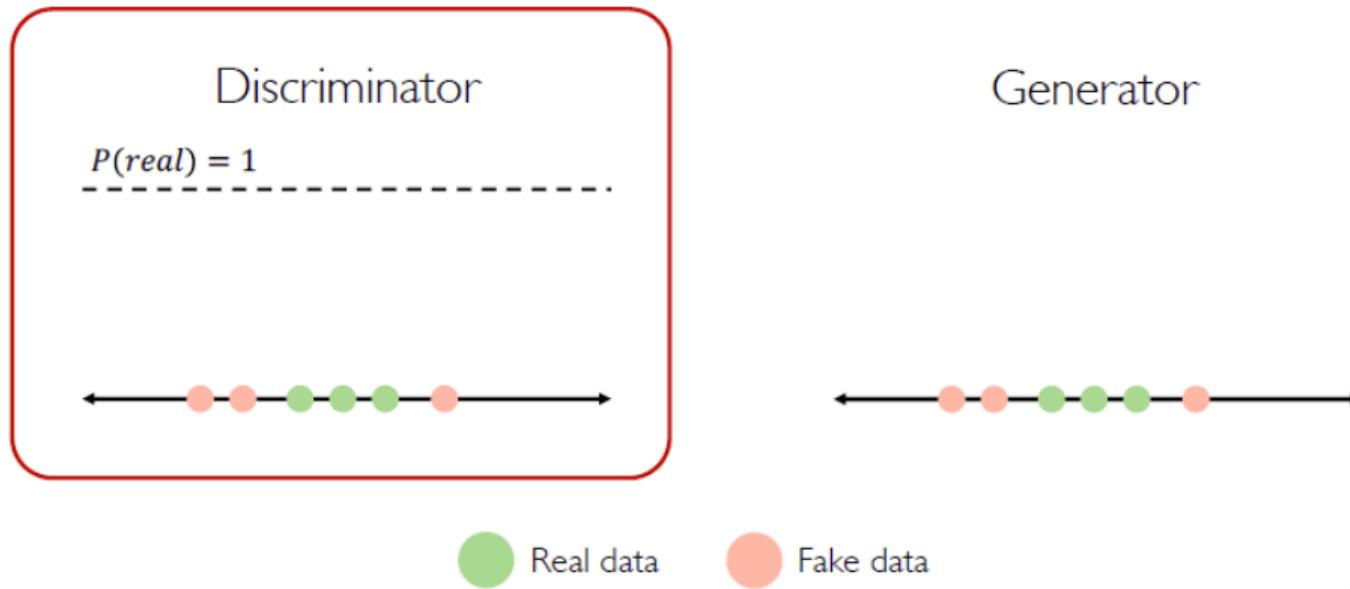
# Intuición detrás de las GANs

El **generador** trata de mejorar su imitación de los datos.



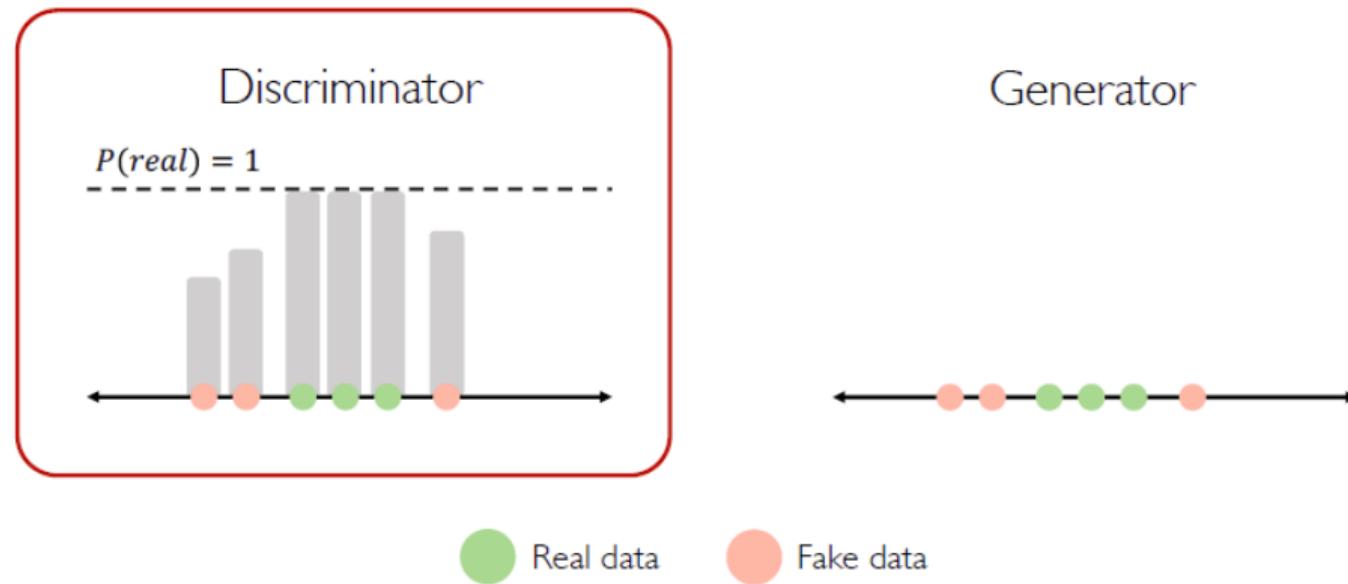
# Intuición detrás de las GANs

El **discriminador** trata de predecir lo que es real y lo que es falso.



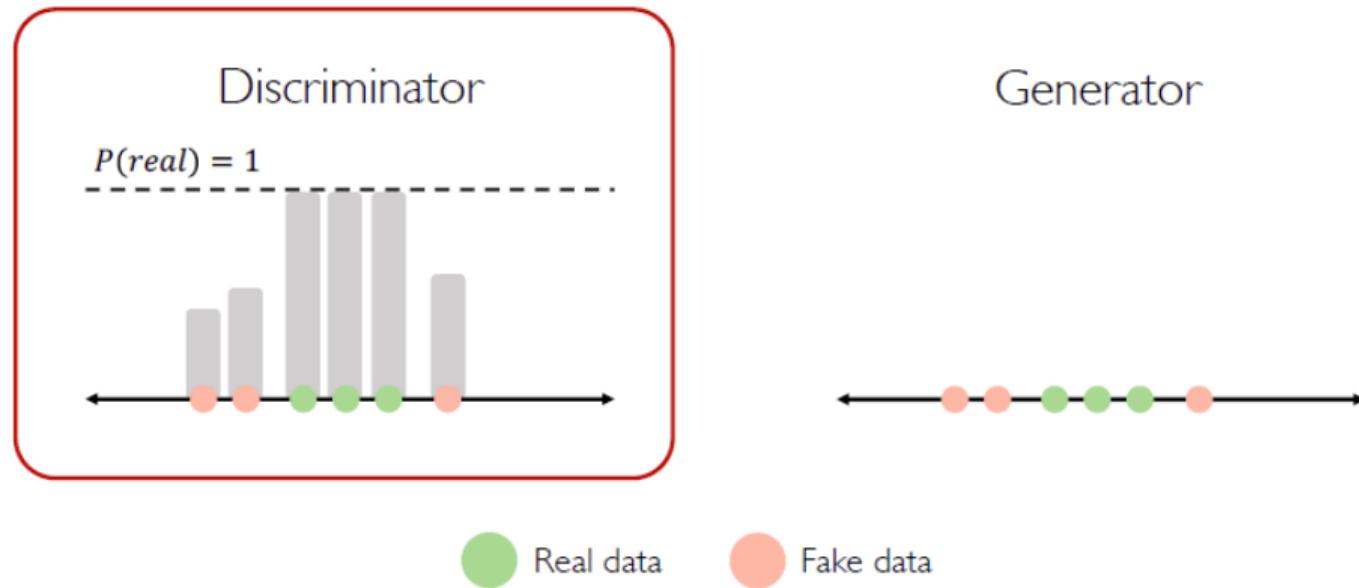
# Intuición detrás de las GANs

El **discriminador** trata de predecir lo que es real y lo que es falso.



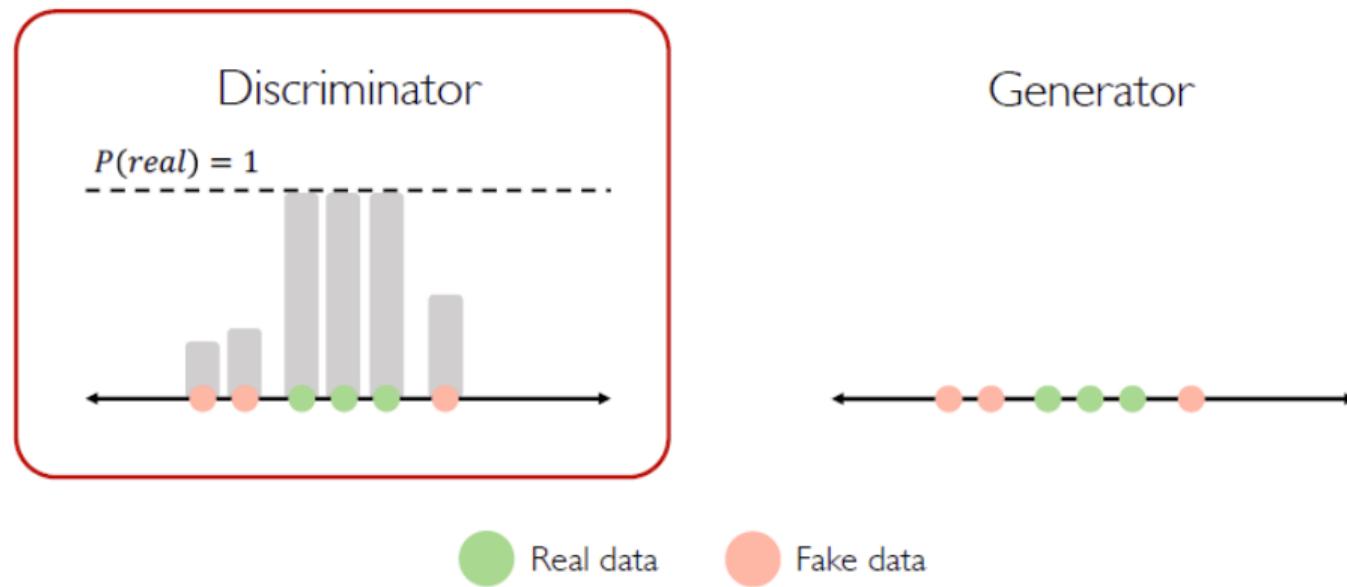
# Intuición detrás de las GANs

El **discriminador** trata de predecir lo que es real y lo que es falso.



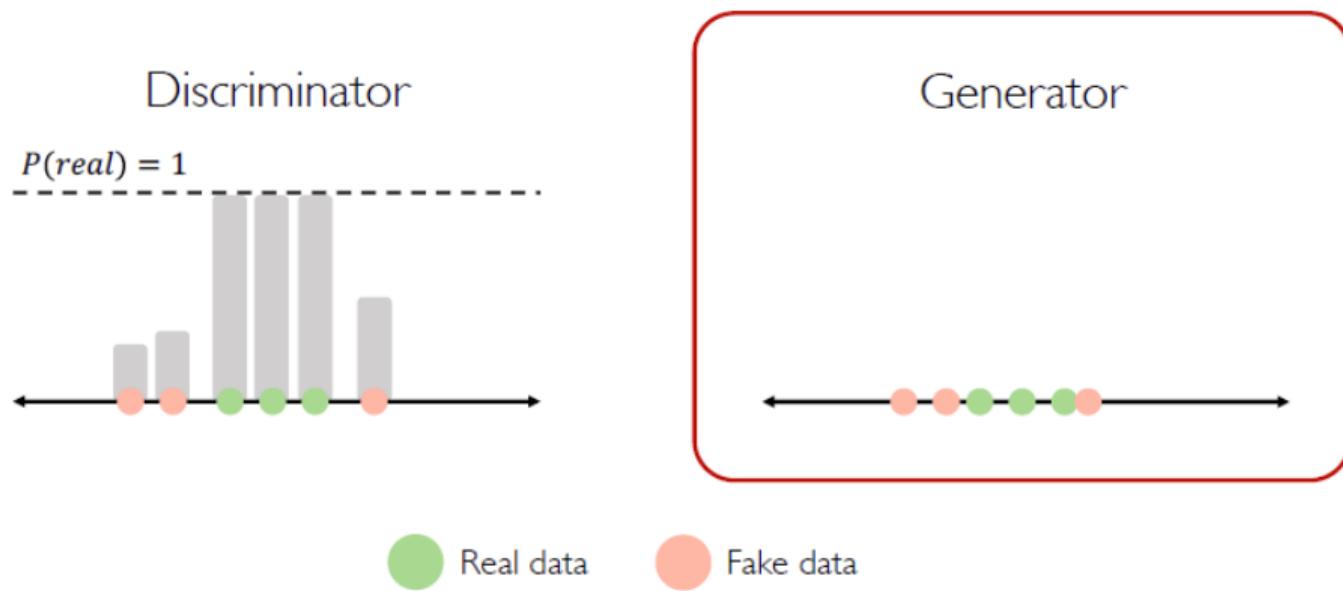
# Intuición detrás de las GANs

El **discriminador** trata de predecir lo que es real y lo que es falso.



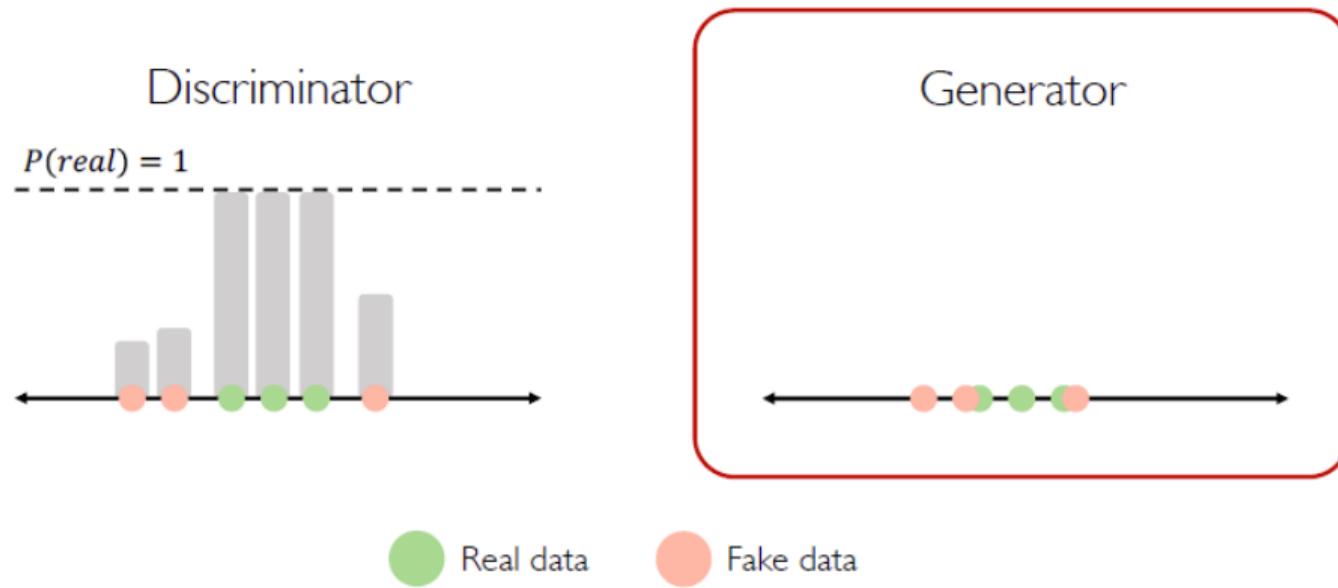
# Intuición detrás de las GANs

El **generador** trata de mejorar su imitación de los datos.



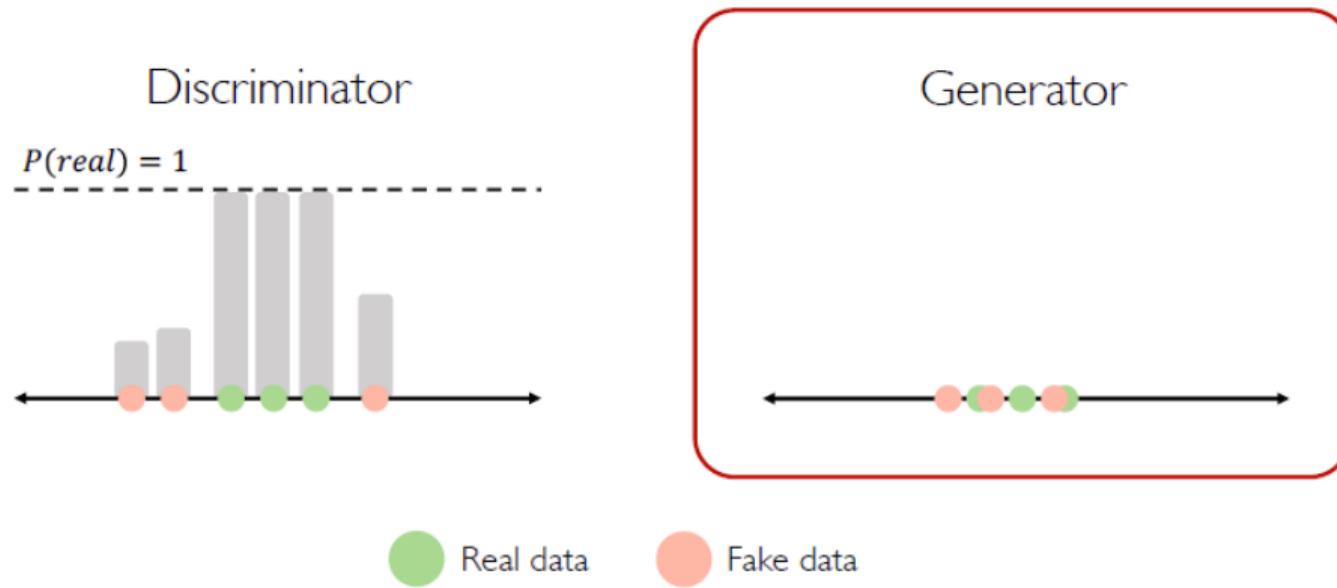
# Intuición detrás de las GANs

El **generador** trata de mejorar su imitación de los datos.



# Intuición detrás de las GANs

El **generador** trata de mejorar su imitación de los datos.



# Intuición detrás de las GANs

El **discriminador** trata de identificar los datos reales de las falsificaciones creadas por el generador.

El **generador** trata de crear imitaciones de datos para engañar al discriminador.



# Entrenando GANs

El **discriminador** trata de identificar los datos reales de las falsificaciones creadas por el generador.

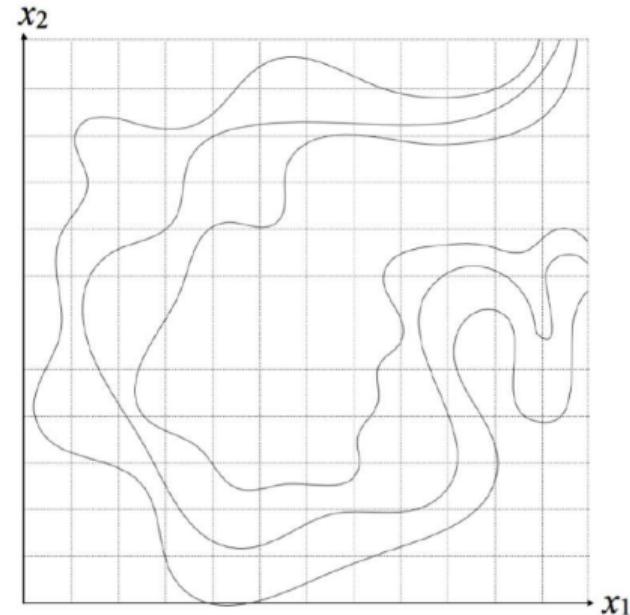
El **generador** trata de crear imitaciones de datos para engañar al discriminador.

**Entrenar** una GAN se realiza a través de un juego de **minimax**:

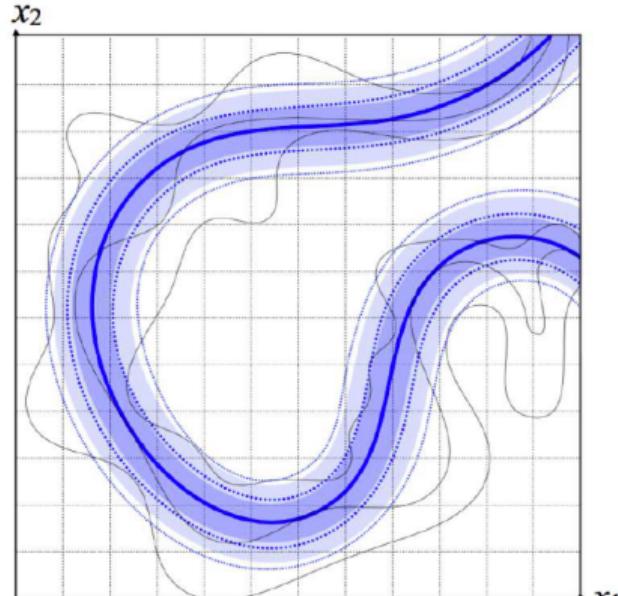
$$\min_{\Theta_g} \max_{\Theta_d} \left[ \mathbb{E}_{x \sim p_{data}} \log D_{\Theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log (1 - D_{\Theta_d}(G_{\Theta_g}(z))) \right]$$

- El **discriminador** quiere maximizar el objetivo  $D(x)$  cerca de 1,  $D(G(z))$  cerca de 0.
- El **generador** quiere minimizar el objetivo  $D(G(z))$  cerca de 1.

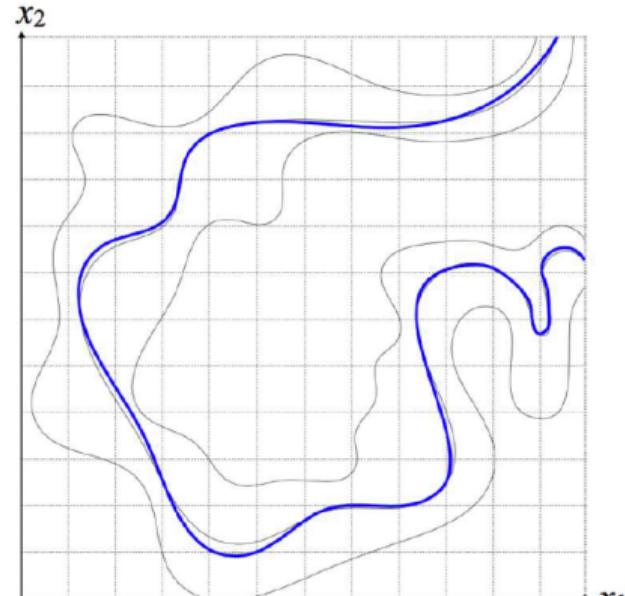
# ¿Por qué GSNs



# ¿Por qué GSNs



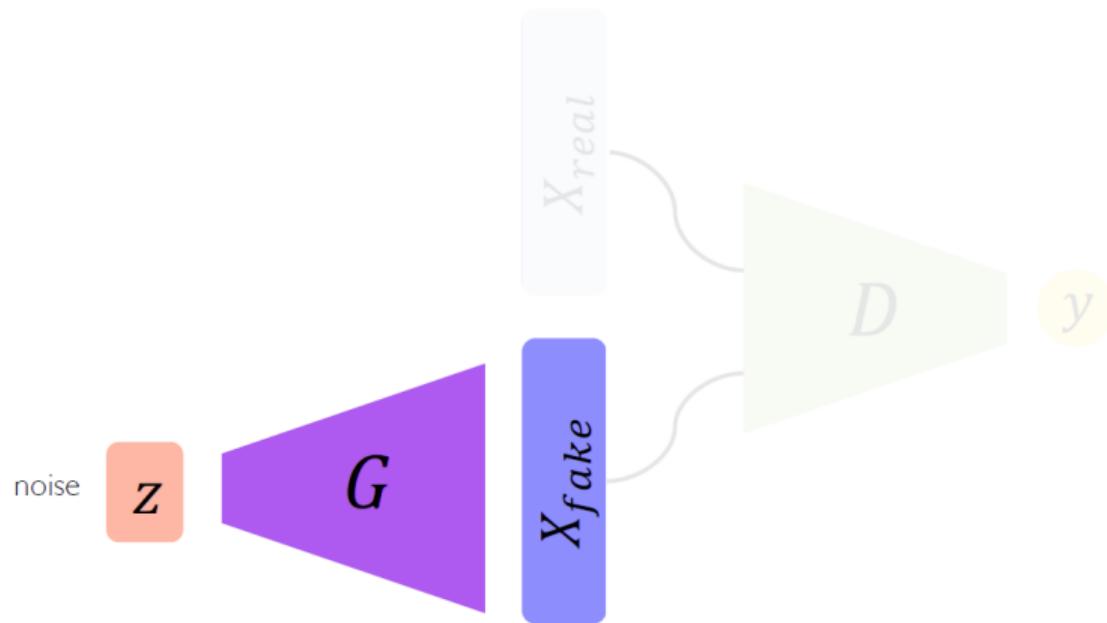
more traditional max-likelihood approach



GAN

# Generando nuevos datos con GANs

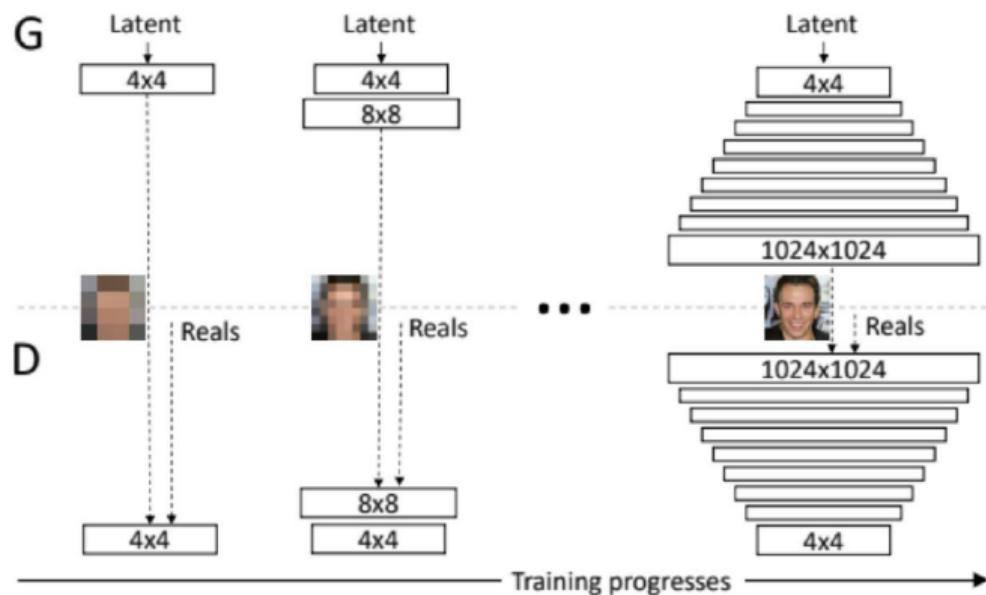
Después del entrenamiento, usa la red de generadores para crear nuevos datos que nunca se han visto antes.



# GANs: avances recientes

# Progressive growing of GANs (NVIDIA)

Crecimiento progresivo de las GAN



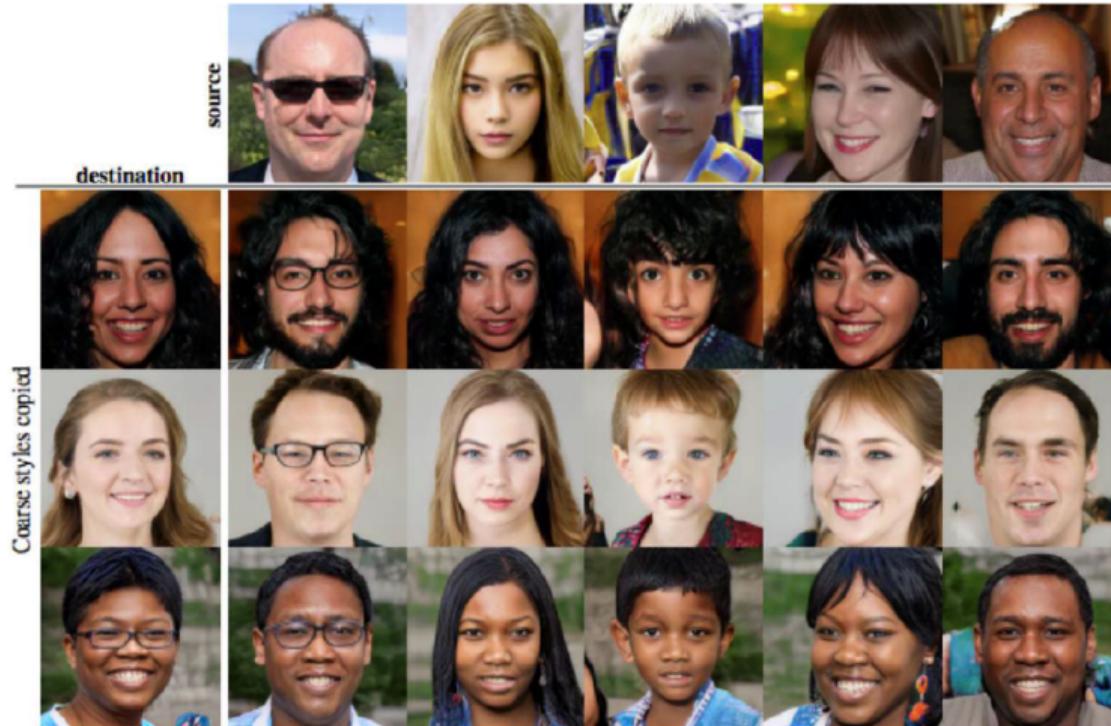
# Progressive growing of GANs: results



# Generador basado en el estilo: resultados

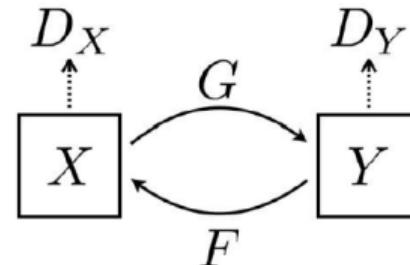


# Transferencia basada en el estilo: resultados



# CycleGAN: transformación del dominio

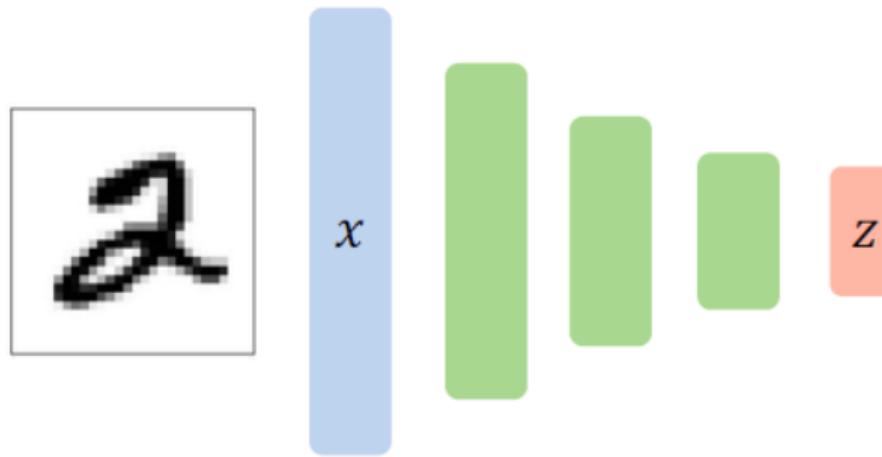
**CycleGAN** aprende las transformaciones a través de los dominios con datos no emparejados.



# Autoencoders

# Autoencoders: background

Enfoque no supervisado para aprender una representación de características de menor dimensión a partir de datos de entrenamiento no etiquetados

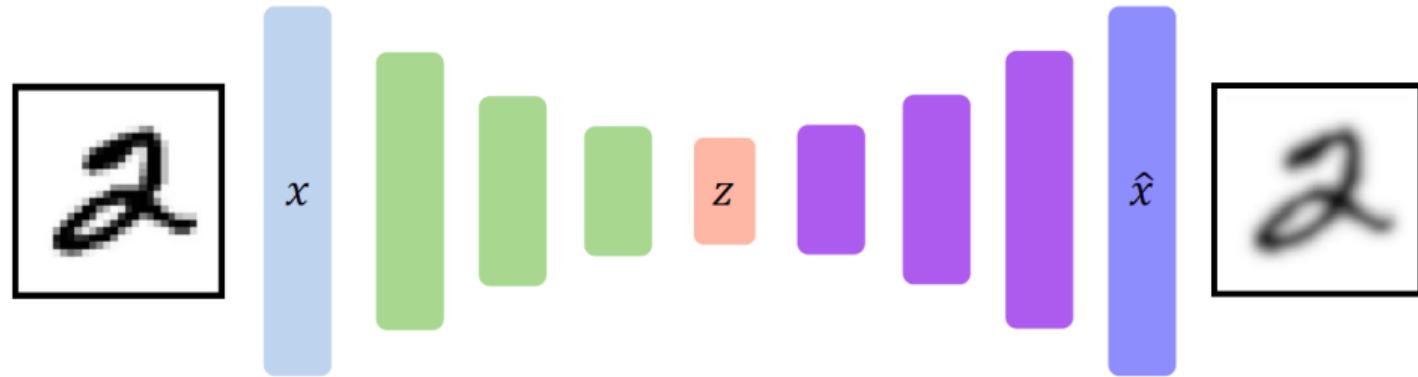


El **Codificador** (*encoder*) aprende el mapeo de los datos,  $x$ , a un espacio latente de baja dimensión,  $z$

# Autoencoders: background

¿Cómo podemos aprender este espacio latente?

Entrena al modelo para que use estas características para **reconstruir los datos originales**

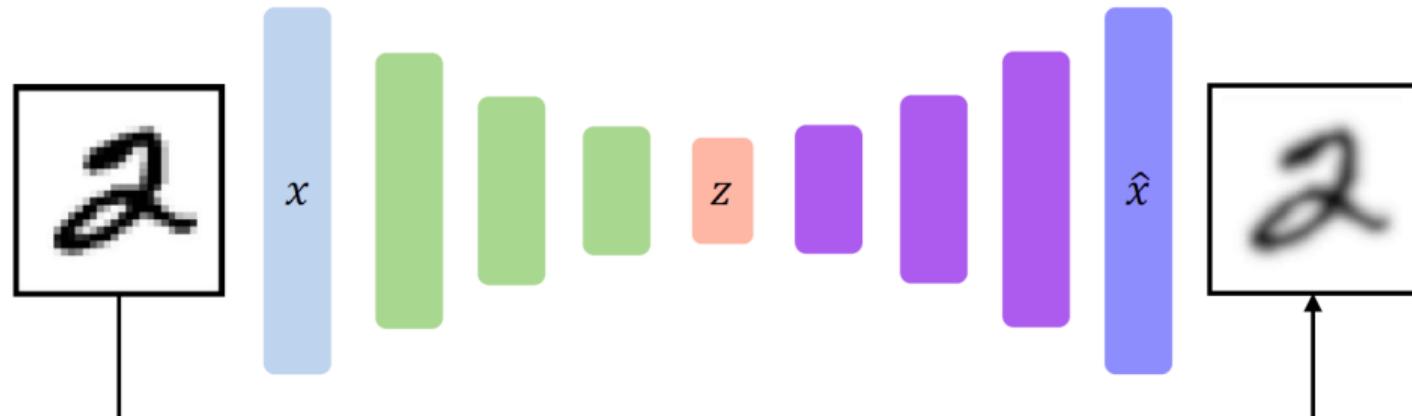


El **Decodificador** (*decoder*) aprende a mapear desde la latente,  $z$ , hasta una observación reconstruida,  $\hat{x}$

# Autoencoders: background

¿Cómo podemos aprender este espacio latente?

Entrena al modelo para que use estas características para **reconstruir los datos originales**



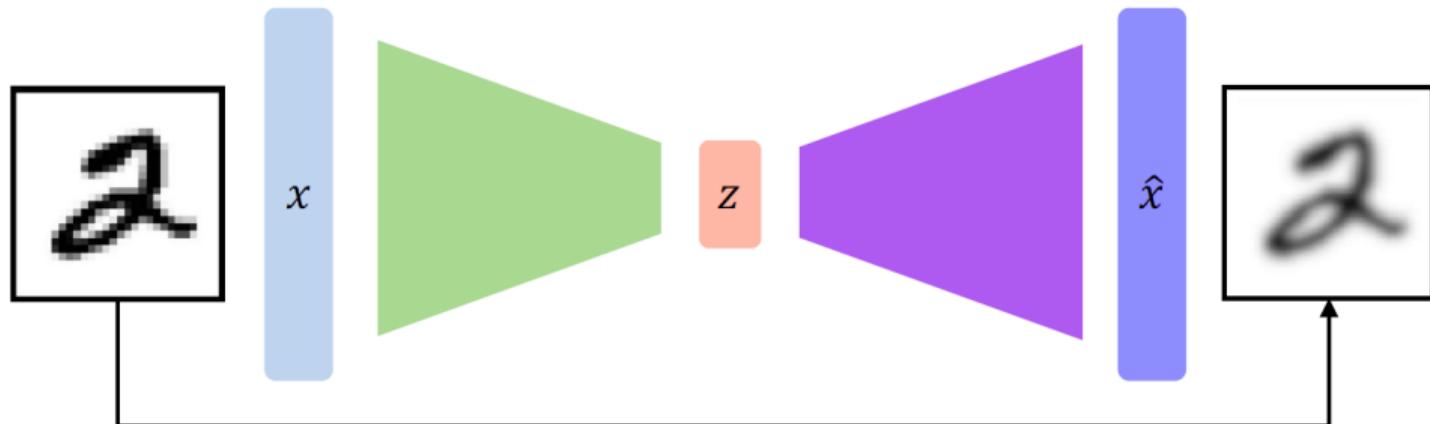
$$\mathcal{L}(x, \hat{x}) = \|x - \hat{x}\|^2$$

La función de pérdida no usa  
ninguna etiqueta!!

# Autoencoders: background

¿Cómo podemos aprender este espacio latente?

Entrena al modelo para que use estas características para **reconstruir los datos originales**



$$\mathcal{L}(x, \hat{x}) = \|x - \hat{x}\|^2$$

La función de pérdida no usa  
ninguna etiqueta!!

# Autoencoders

## Autoencoders

Los autoencoders son redes neuronales artificiales capaces de aprender **representaciones densas** de datos de entrada, conocidas como *representaciones latentes*, sin supervisión. Estas codificaciones generalmente tienen una **dimensionalidad mucho menor** que los datos de entrada, lo que los hace útiles para la **reducción de dimensionalidad**.

# Autoencoders

- Útiles para reducción de dimensionalidad y visualización.
- Actúan como detectores de características.
- Pueden ser utilizados para preentrenamiento no supervisado de redes neuronales profundas.
- Algunos autoencoders son modelos generativos: pueden generar nuevos datos similares a los de entrenamiento.

# La dimensionalidad del espacio latente → calidad de la reconstrucción

- ¡La autocodificación es una forma de compresión!
- Un espacio latente más pequeño forzará un mayor cuello de botella de entrenamiento

2D latent space	5D latent space	Ground Truth
		

# Autocodificadores para el aprendizaje por representación

- La **capa oculta cuello de botella** obliga a la red a aprender una comprimida representación latente
- La **pérdida por reconstrucción** obliga a la representación latente a capturar (o *codificar*) tanta **información** sobre los datos como sea posible
- Autocodificación = **Codificación** automática de datos

# Representaciones Eficientes de Datos

# Representaciones Eficientes de Datos

**¿Cuál de las siguientes secuencias numéricas encuentra más fácil de memorizar?**

- 40, 27, 25, 36, 81, 57, 10, 73, 19, 68
- 50, 48, 46, 44, 42, 40, 38, 36, 34, 32, 30, 28, 26, 24, 22, 20, 18, 16, 14

# Representaciones Eficientes de Datos

- Las secuencias más cortas no siempre son más fáciles de memorizar.
- Reconocer un patrón en los datos puede simplificar la memorización.
- Las restricciones durante el entrenamiento de un autoencoder lo empujan a descubrir y explotar patrones en los datos.

# Representaciones Eficientes de Datos

**¿Cuál de las siguientes secuencias numéricas encuentra más fácil de memorizar?**

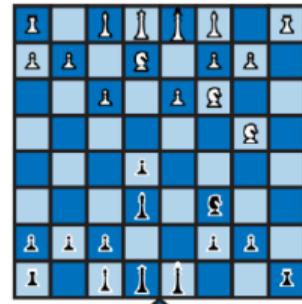
- 40, 27, 25, 36, 81, 57, 10, 73, 19, 68
- 50, 48, 46, 44, 42, 40, 38, 36, 34, 32, 30, 28, 26, 24, 22, 20, 18, 16, 14

Una secuencia que muestra números pares decrecientes desde 50 hasta 14 es más fácil de recordar que una secuencia aleatoria de números, debido al patrón reconocible.

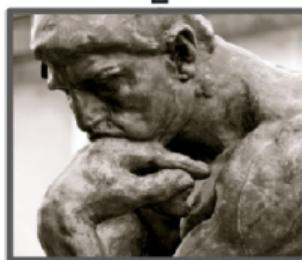
# Estudios de Chase y Simon

Los expertos en ajedrez memorizan posiciones de piezas rápidamente si estas provienen de partidas reales. La experiencia ayuda a reconocer patrones y almacenar información de manera eficiente.

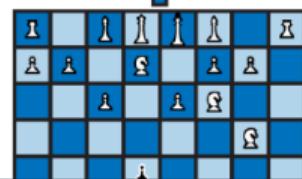
# Relación entre Memoria, Percepción y Correspondencia de Patrones



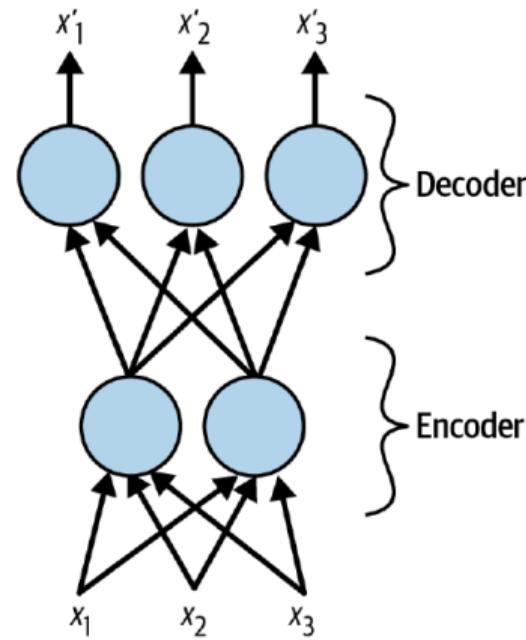
Outputs  
(≈inputs)



Latent representation



Inputs



# Relación entre Memoria, Percepción y Correspondencia de Patrones

- La percepción de patrones es crucial para una memorización efectiva.
- Los expertos no tienen mejor memoria, pero reconocen patrones con mayor facilidad.
- Un autoencoder convierte entradas a una representación latente eficiente.

# Estructura y Función de un Autoencoder

# Estructura y Función de un Autoencoder

Compuesto por dos partes:

- Un codificador que convierte las entradas en una representación latente
- Un decodificador que convierte esta representación en salidas.

# Estructura y Función de un Autoencoder

- Arquitectura similar a un perceptrón multicapa.
- El número de neuronas en la capa de salida es igual al número de entradas.
- El autoencoder trata de reconstruir las entradas.
- Los autoencoders incompletos.<sup>a</sup>prenden características importantes de los datos.

Un autoencoder con una representación interna de menor dimensión que los datos de entrada aprende a destacar y retener solo las características más importantes de los datos.

# Stacked Autoencoders

# Stacked Autoencoders

## Autoencoders Apilados

Al igual que otras redes neuronales, los autoencoders pueden tener múltiples capas ocultas. En este caso, se les denomina autoencoders apilados o profundos. Estos aprenden codificaciones más complejas al agregar más capas.

# Stacked Autoencoders

- Pueden tener múltiples capas ocultas.
- Aprender codificaciones más complejas.
- No deben ser demasiado potentes para evitar sobreajuste.
- La arquitectura es típicamente simétrica respecto a la capa central.
- Su diseño se asemeja a un sándwich.

# Arquitectura Simétrica

La arquitectura de un autoencoder apilado es generalmente simétrica en relación con la capa oculta central, conocida como la capa de codificación.

# Arquitectura Simétrica

- Diseño similar a un sándwich.
- La capa central es la capa de codificación.
- Las capas anteriores y posteriores a la capa central son simétricas.
- La entrada y salida tienen dimensiones similares.

# Stacked autoencoder

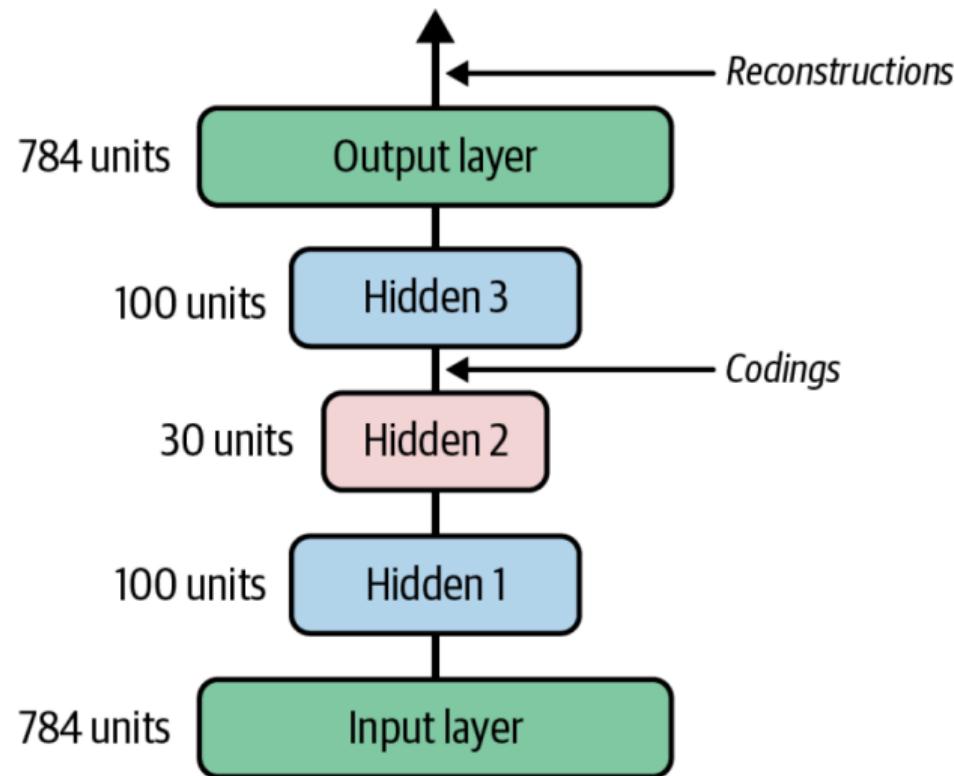
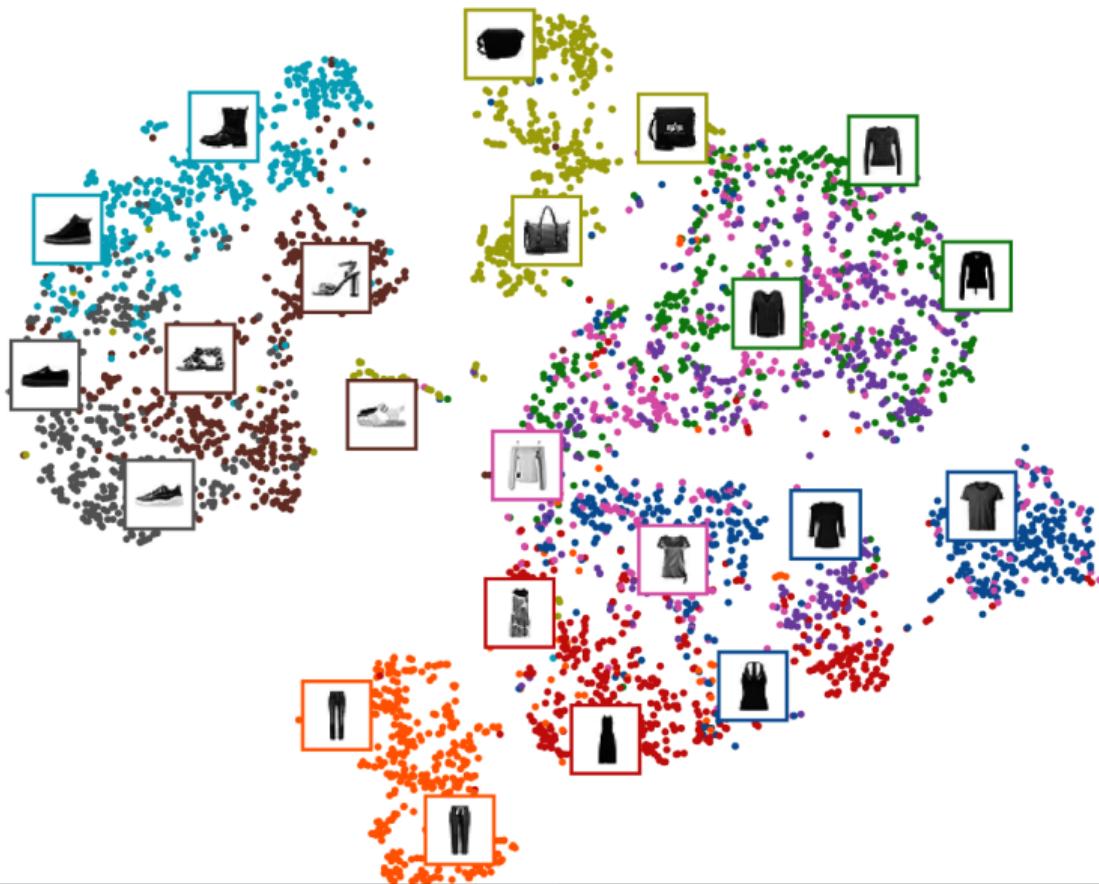


Figura 2: Stacked autoencoder

# Stacked autoencoder



# Entrenamiento no supervisado con Autoencoders Apilados

# Entrenamiento no supervisado con Autoencoders Apilados

## Preentrenamiento no supervisado

Si se enfrenta a una tarea supervisada compleja con pocos datos etiquetados, se puede reutilizar las capas inferiores de una red neuronal que realice una tarea similar. Esto permite entrenar un modelo de alto rendimiento con pocos datos, reutilizando detectores de características de la red existente.

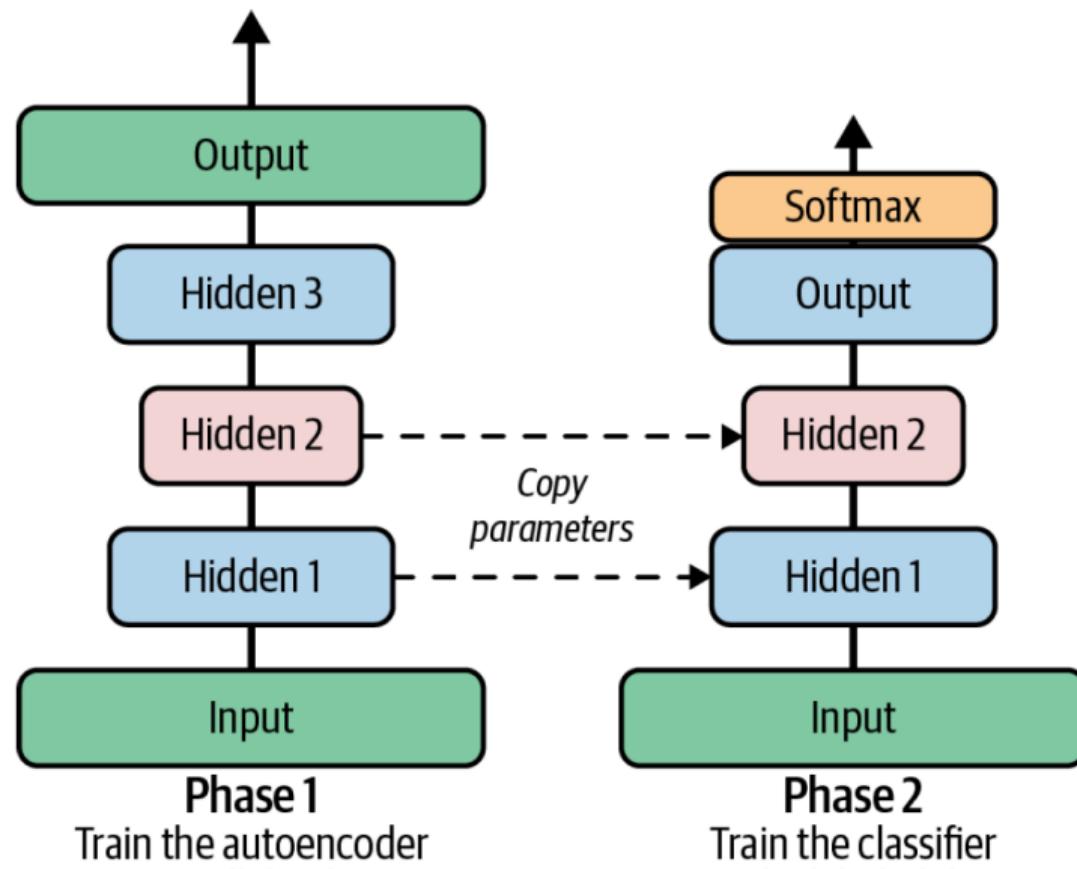
# Entrenamiento no supervisado con Autoencoders Apilados

- Reutilización de capas inferiores para tareas similares.
- El modelo no necesita aprender todas las características de bajo nivel.
- Los autoencoders apilados pueden ser entrenados con datos no etiquetados.
- Se pueden congelar las capas preentrenadas para mejorar la formación.
- El preentrenamiento no supervisado es útil para redes neuronales de clasificación.

# Entrenamiento no supervisado con Autoencoders Apilados

Utilizar un autoencoder apilado para preentrenamiento no supervisado en una red neuronal de clasificación. Si hay pocos datos etiquetados, congelar las primeras capas del autoencoder.

# Stacked autoencoder



# Training one autoencoder at a time

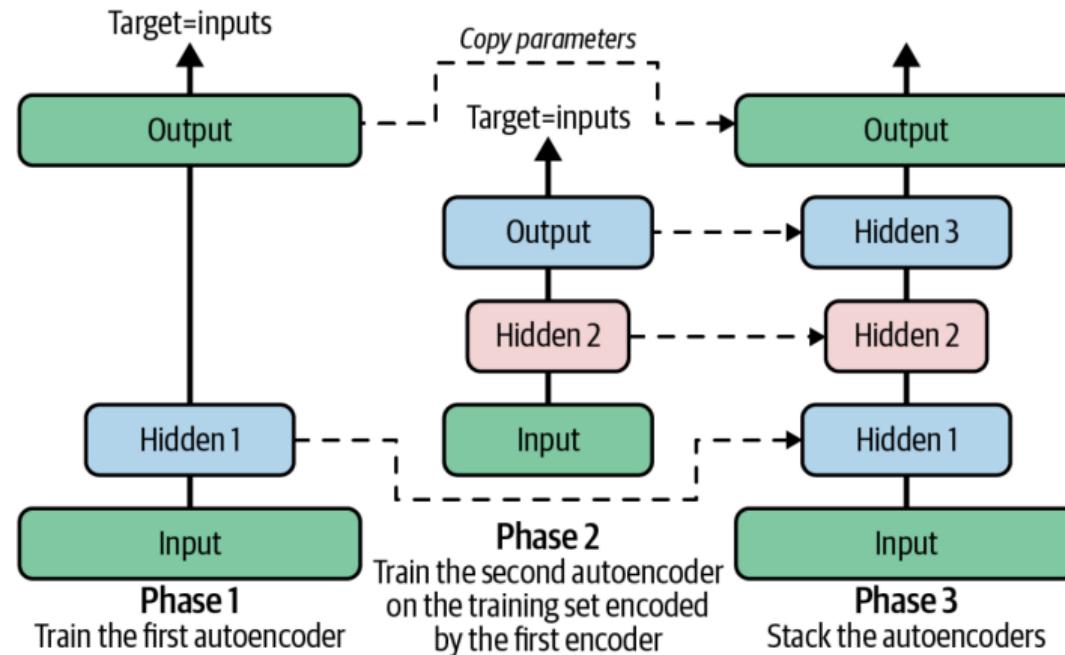


Figura 5: Training one autoencoder at a time

# Convolutional Autoencoders

# Autoencoders en Redes Convolucionales

Los autoencoders no se limitan a redes densas; también es posible construir autoencoders convolucionales, que utilizan convoluciones para detectar patrones espaciales.

## Definición de Autoencoders Convolucionales

Si se trabaja con imágenes, los autoencoders tradicionales no serán eficientes a menos que las imágenes sean pequeñas. Para imágenes, es preferible usar autoencoders convolucionales, diseñados usando redes neuronales convolucionales (CNN).

## Autoencoders en Redes Convolucionales

- Los autoencoders convolucionales funcionan con datos estructurados espacialmente.
- Utilizan operaciones de convolución para la compresión y deconvolución para la reconstrucción.
- Permiten la detección de características jerárquicas en imágenes y otros datos espaciales.
- Pueden ser entrenados capa por capa o de una sola vez.
- El codificador (encoder) es una CNN típica con capas convolucionales y de agrupación.
- Reduce la dimensionalidad espacial mientras aumenta la profundidad (número de mapas de características).
- El decodificador realiza el proceso inverso utilizando capas convolucionales transpuestas.

# Autoencoders en Redes Convolucionales

En el procesamiento de imágenes, un autoencoder convolucional puede aprender a reconocer patrones como bordes y texturas en sus capas inferiores.

# Tipos Avanzados de Autoencoders

Más allá de los autoencoders básicos, existen variantes avanzadas como autoencoders de ruido, autoencoders dispersos y autoencoders variacionales que se adaptan a diferentes necesidades y aplicaciones.

# Tipos Avanzados de Autoencoders

- Autoencoders de ruido: Introducen ruido en la entrada durante el entrenamiento.
- Autoencoders dispersos: Usan una penalización de dispersión para activaciones raras.
- Autoencoders variacionales: Añaden una capa de aleatoriedad para modelar distribuciones de datos.

# Denoising Autoencoders

# Introducción a Autoencoders de Denoising

## Definición de Autoencoders de Denoising

Una forma de forzar a los autoencoders a aprender características útiles es agregar ruido a sus entradas y entrenarlos para recuperar las entradas originales sin ruido. Esta idea ha estado presente desde la década de 1980.

# Introducción a Autoencoders de Denoising

- Yann LeCun mencionó esta idea en su tesis de maestría de 1987.
- Pascal Vincent et al., en 2008, demostraron que los autoencoders pueden usarse para extracción de características.
- En 2010, Vincent et al. presentaron autoencoders de denoising apilados.

# Tipos de Ruido en Autoencoders de Denoising

El ruido puede ser ruido gaussiano puro añadido a las entradas, o pueden ser entradas apagadas aleatoriamente, similar al dropout.

- Ruido Gaussiano: Ruido continuo añadido a la entrada.
- Dropout: Apaga aleatoriamente una proporción de las entradas.
- Ambas opciones están visualizadas en la Figura 17-8.

# Tipos de Ruido en Autoencoders de Denoising

Imagina añadir un ruido gaussiano aleatorio a una imagen de un gato, haciendo que algunas partes de la imagen sean poco claras. El autoencoder deberá aprender a "limpiar.<sup>esta</sup> imagen.

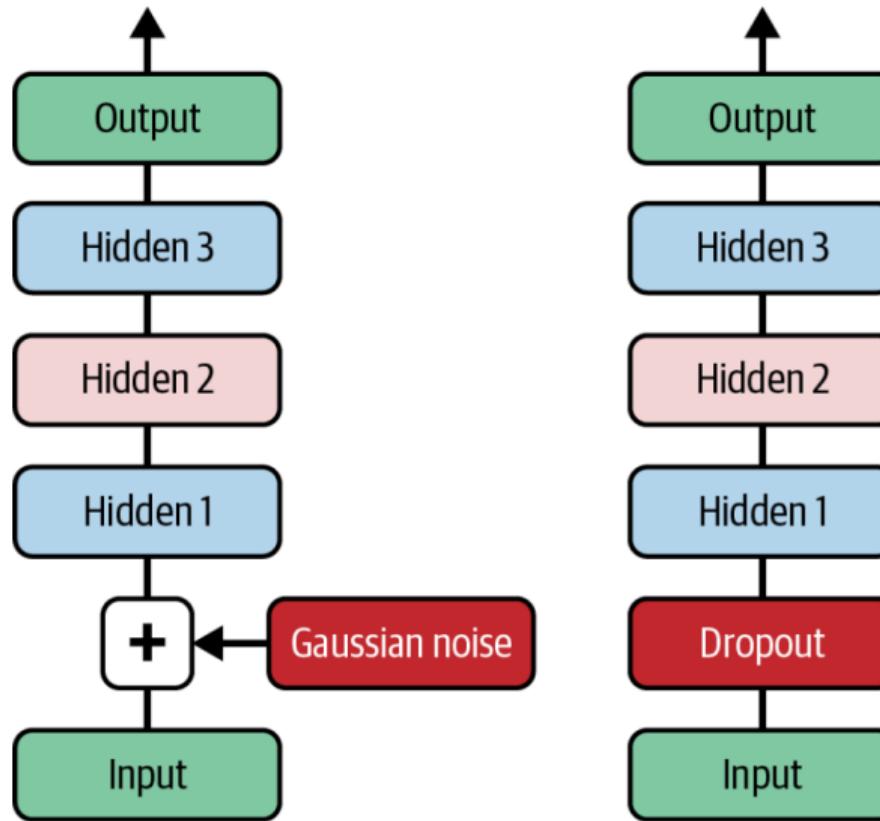
# Implementación de Autoencoders de Denoising

La implementación es directa: es un autoencoder apilado regular con una capa adicional de Dropout aplicada a las entradas del codificador. Alternativamente, se puede usar una capa de GaussianNoise.

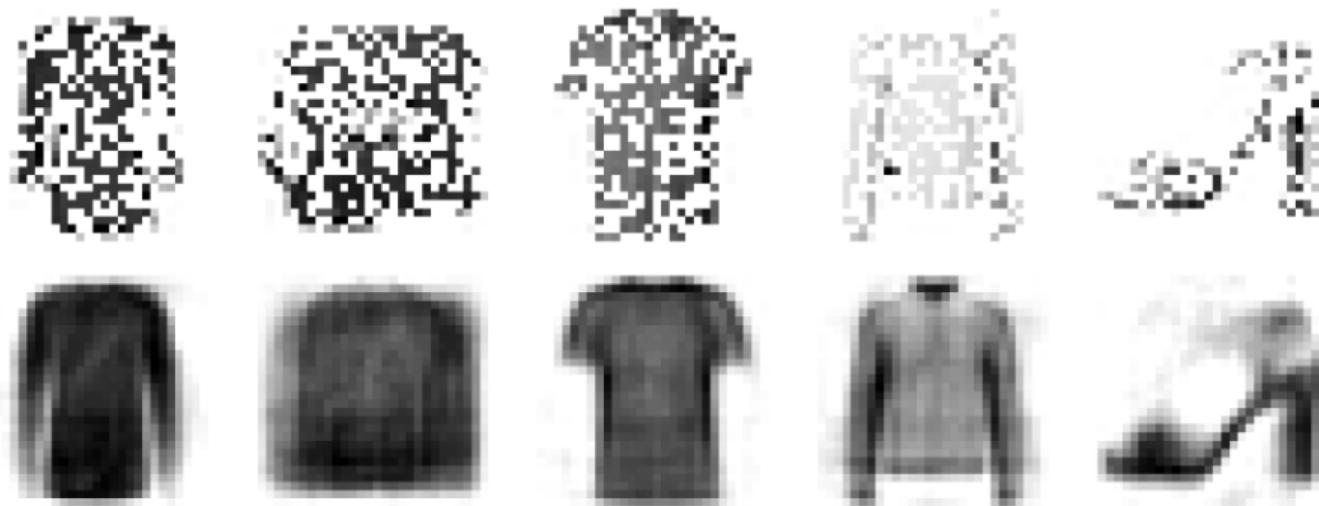
# Implementación de Autoencoders de Denoising

- La capa de Dropout solo está activa durante el entrenamiento.
- Igualmente, la capa GaussianNoise solo se activa durante el entrenamiento.
- Estas capas introducen ruido solamente en la fase de entrenamiento.

# Denoising autoencoders



# Denoising autoencoders



**Figura 7:** Noisy images (top) and their reconstructions (bottom)

# Sparse Autoencoders

# Autoencoders Dispersos (Sparse Autoencoders)

## Definición de Autoencoders Dispersos

Un tipo de restricción que a menudo conduce a una buena extracción de características es la dispersión. Mediante la adición de un término adecuado a la función de coste, se impulsa al autoencoder a reducir el número de neuronas activas en la capa de codificación.

## Autoencoders Dispersos (Sparse Autoencoders)

- La restricción de dispersión lleva a la representación de cada entrada con un pequeño número de activaciones.
- Se puede usar la función de activación sigmoide en la capa de codificación para limitar los codings entre 0 y 1.
- Se añade una regularización  $\ell_1$  a las activaciones de la capa de codificación.
- Otra técnica es medir la dispersión real de la capa de codificación en cada iteración de entrenamiento.
- La divergencia de Kullback–Leibler (KL) es una herramienta eficaz para este propósito.

## Autoencoders Dispersos (Sparse Autoencoders)

### Divergencia Kullback–Leibler:

$$DKL(P\|Q) = \sum_i P_i \log \frac{P_i}{Q_i}$$

Divergencia KL entre la dispersión objetivo p y la dispersión real

$$DKL(p\|q) = p \log \frac{p}{q} + (1-p) \log \frac{1-p}{1-q}$$

**Ejemplo:** Suponga una capa de codificación que tiene en promedio el 30% de sus neuronas activas. Si se establece un objetivo de dispersión del 10%, mediante técnicas como la divergencia KL, se puede penalizar y ajustar este comportamiento para acercarse al objetivo.

# Sparse Autoencoders

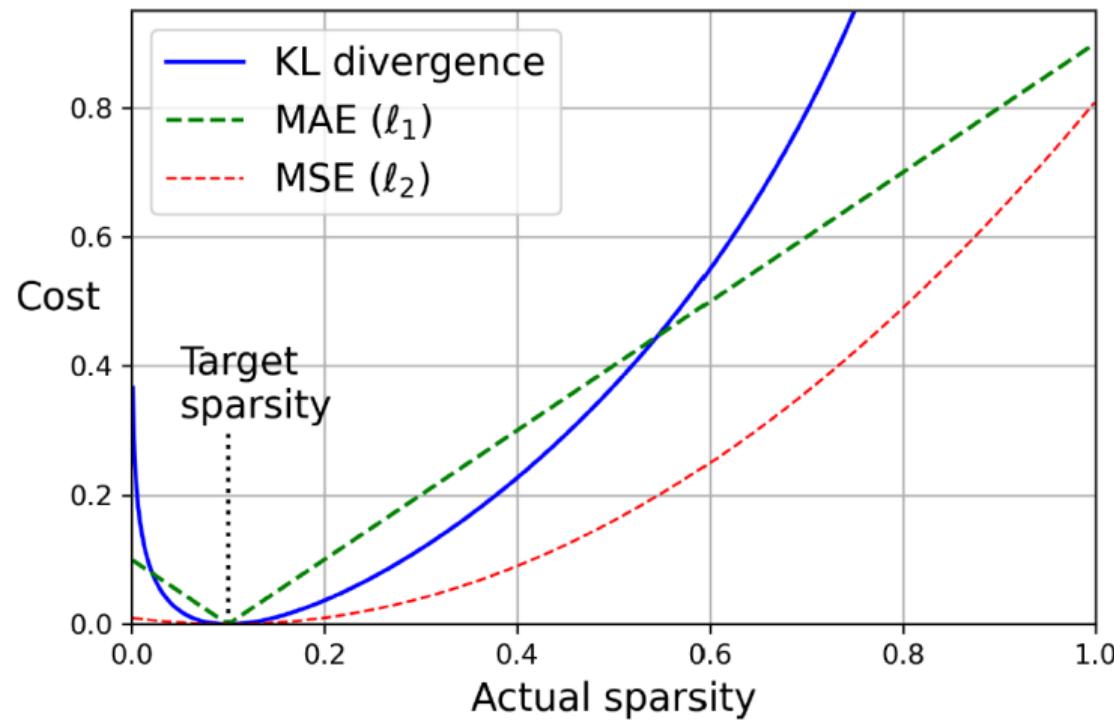


Figura 8: Sparsity loss

# Variational Autoencoders (VAEs)

# Introducción a las Redes Generativas Adversarias (GANs)

## Definición

Las GANs fueron propuestas en un artículo de 2014 por Ian Goodfellow y colaboradores. Estas hacen competir redes neuronales entre sí, con la esperanza de que esta competencia las impulse a sobresalir.

# Introducción a las Redes Generativas Adversarias (GANs)

- Propuesto en 2014 por Ian Goodfellow.
- Hace competir a dos redes neuronales.
- Compuesto por un Generador y un Discriminador.



Figura 9: Ian Goodfellow

# Componentes de una GAN

## Generador:

- Toma una distribución aleatoria (usualmente Gaussiana) como entrada.
- Produce datos, típicamente imágenes.
- Similar a un decodificador en un autoencoder variacional.

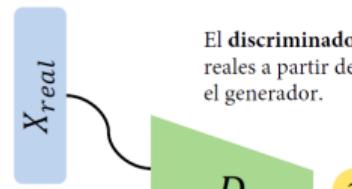
## Discriminador:

- Toma imágenes falsas del generador o imágenes reales del conjunto de entrenamiento.
- Adivina si la imagen es falsa o real.

# Entrenamiento de GANs

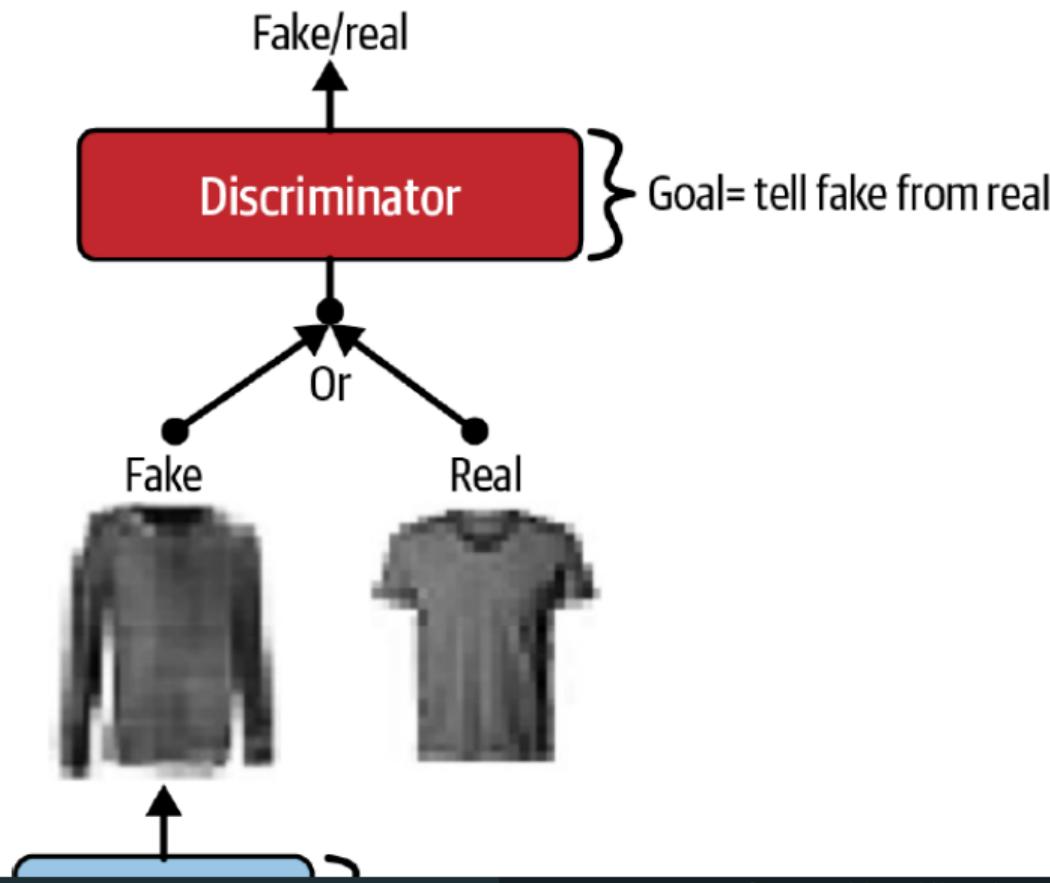
- **Primera fase:** Entrenamos al discriminador. Usamos imágenes reales y falsas. Etiquetamos como 0 las falsas y 1 las reales.
- **Segunda fase:** Entrenamos al generador. Producimos imágenes falsas. Todas las etiquetas se establecen en 1.
- Solo se optimizan los pesos del discriminador en la primera fase.
- Solo se optimizan los pesos del generador en la segunda fase.
- El generador nunca ve imágenes reales.
- El discriminador mejora la información sobre las imágenes reales.

El **generador** convierte el ruido en una



El **discriminador** trata de identificar datos reales a partir de falsificaciones creadas por el generador.

# Entrenamiento de GANs

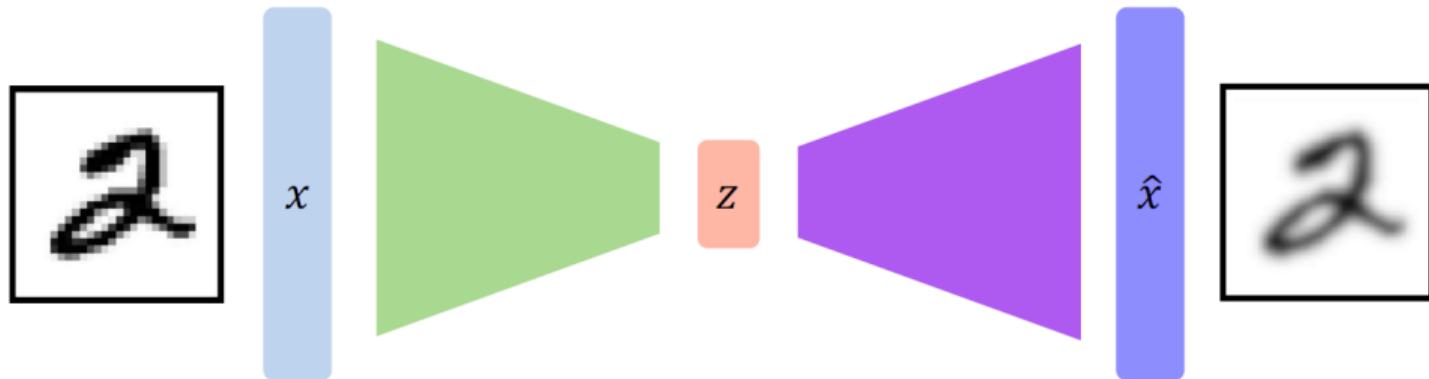


# Entrenamiento de GANs

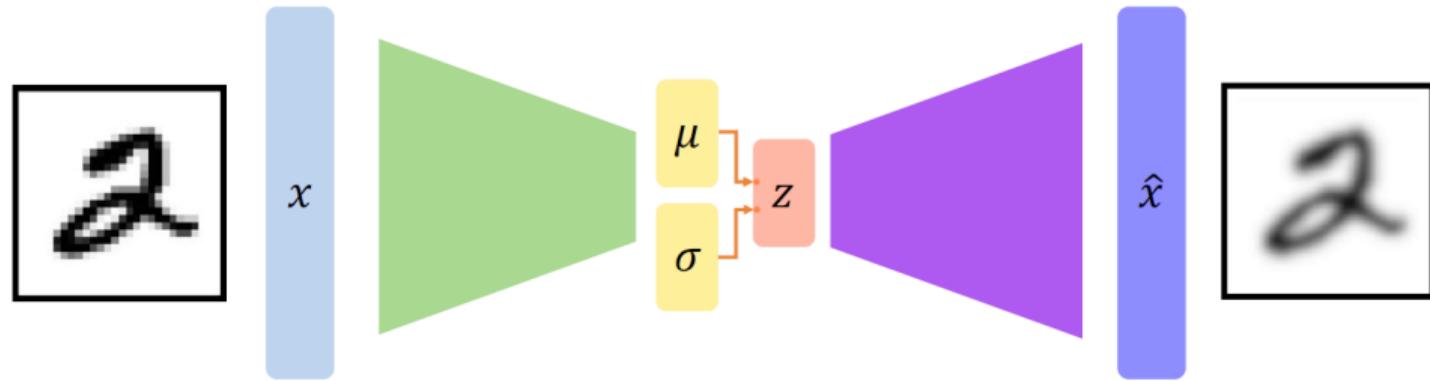


Figura 11: Images generated by the GAN after one epoch of training

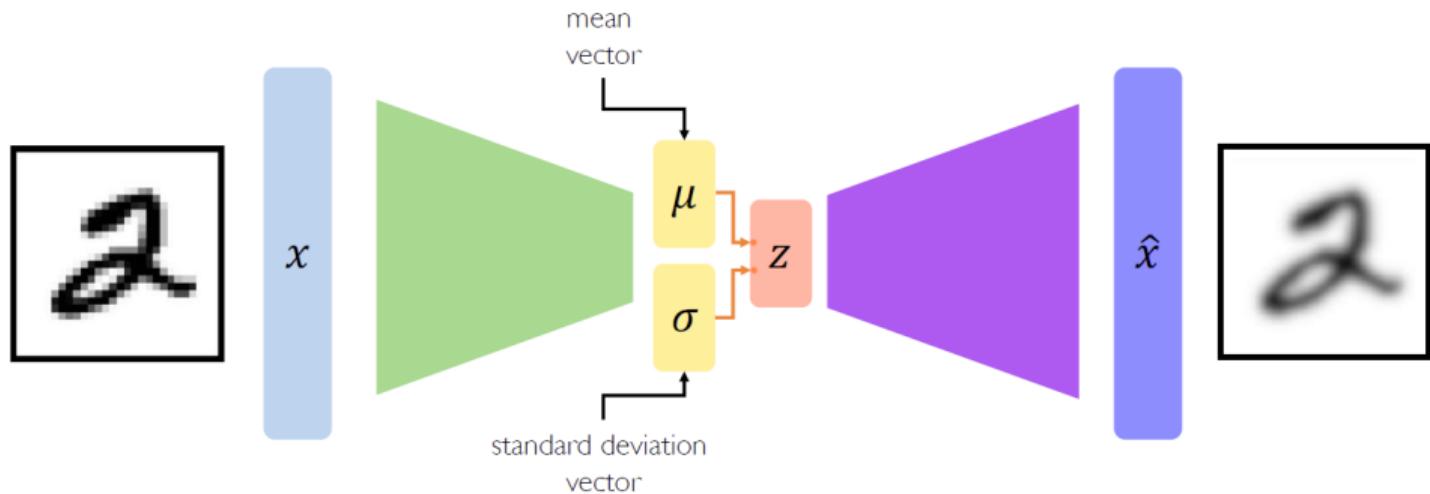
# VAEs: diferencia clave con el autoencoder tradicional



# VAEs: diferencia clave con el autoencoder tradicional

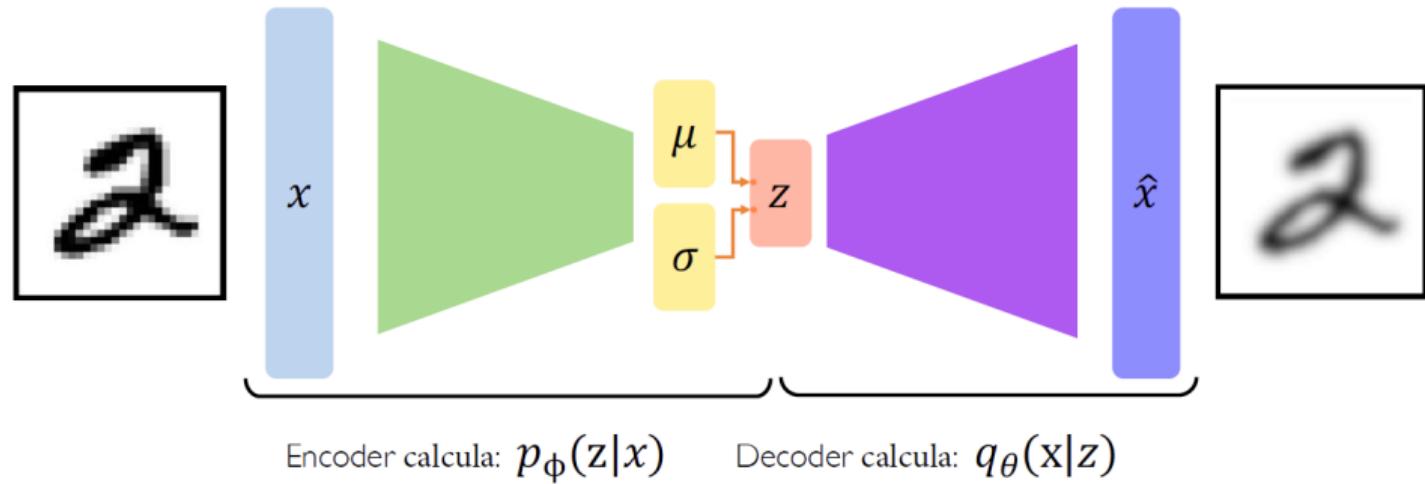


# VAEs: diferencia clave con el autoencoder tradicional

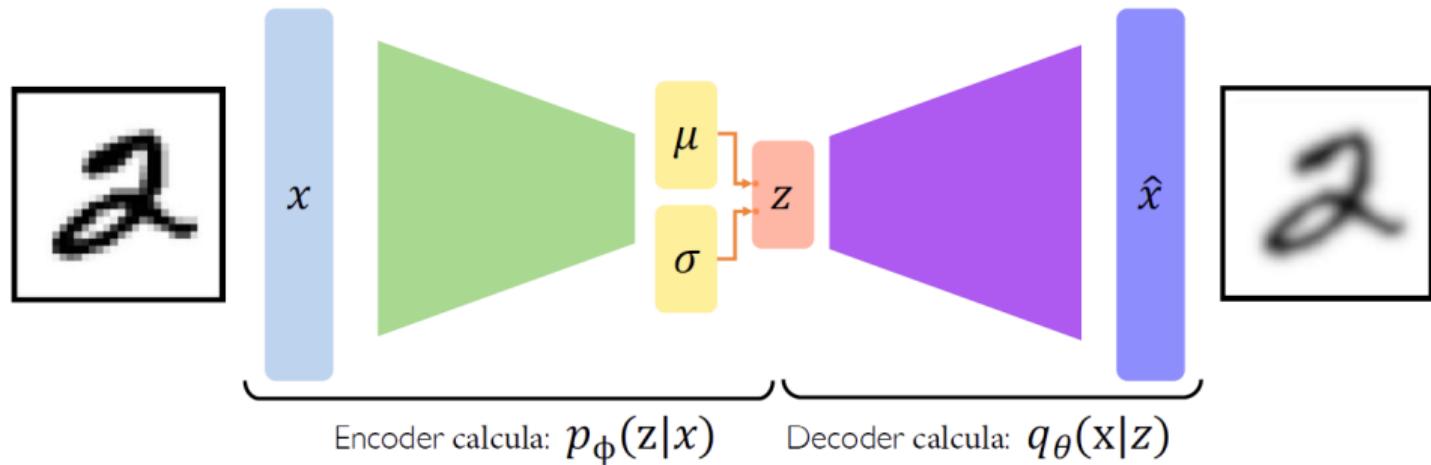


- ¡Los autoencoders variables son un giro probabilístico de los autocodificadores!
- Muestra de la media y la desviación estándar para calcular la muestra latente

# Optimización de la VAE

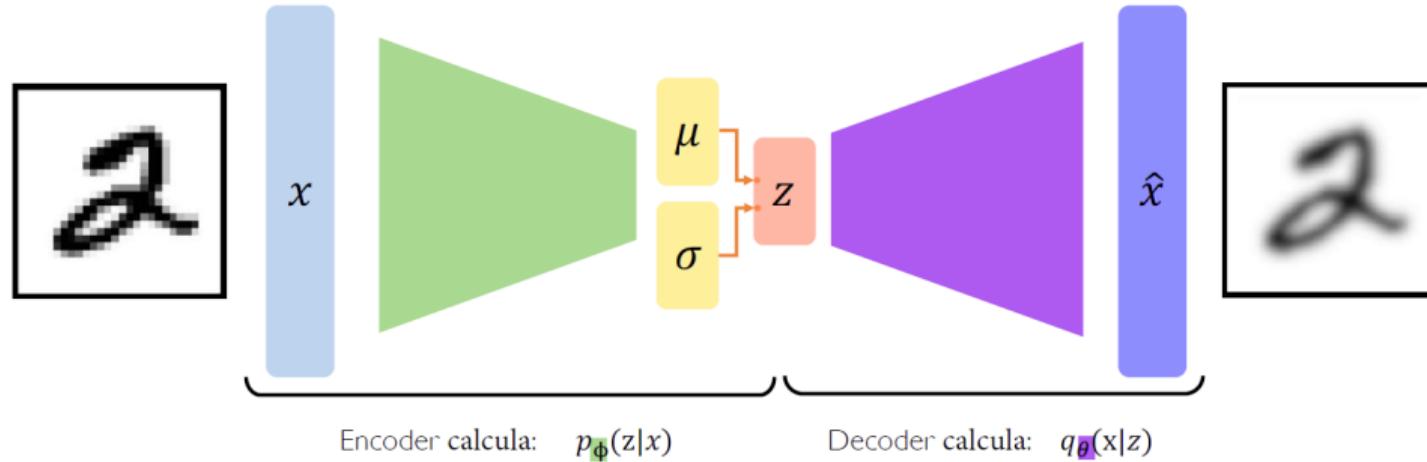


# Optimización de la VAE



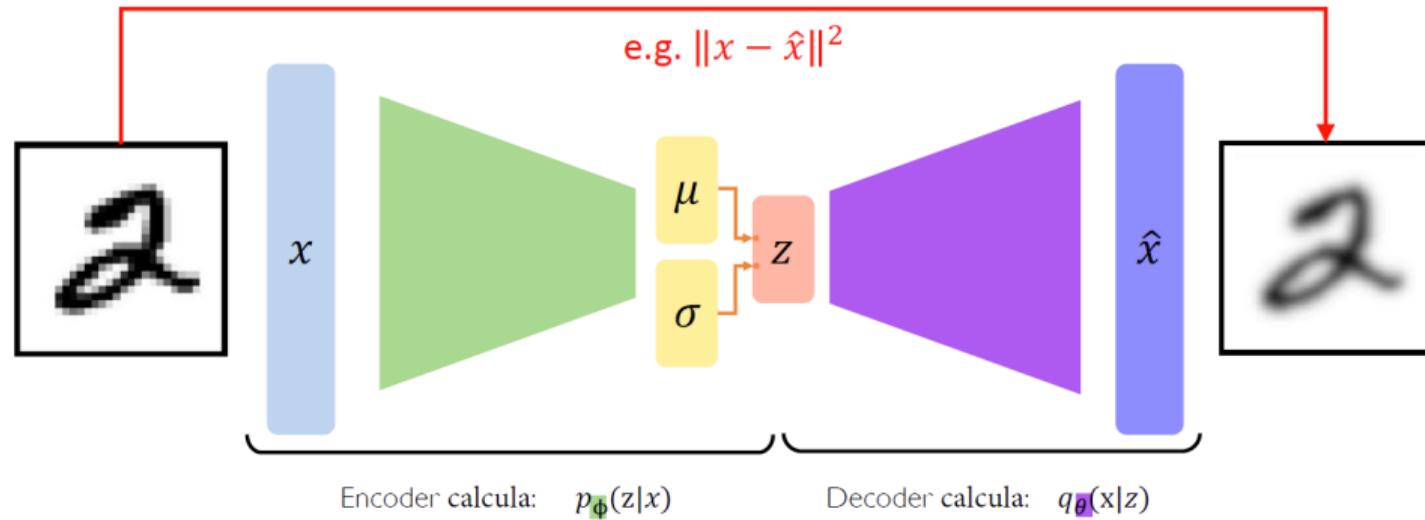
$$\mathcal{L}(\phi, \theta) = (\text{reconstruction loss}) + (\text{termino de regularización})$$

# Optimización de la VAE



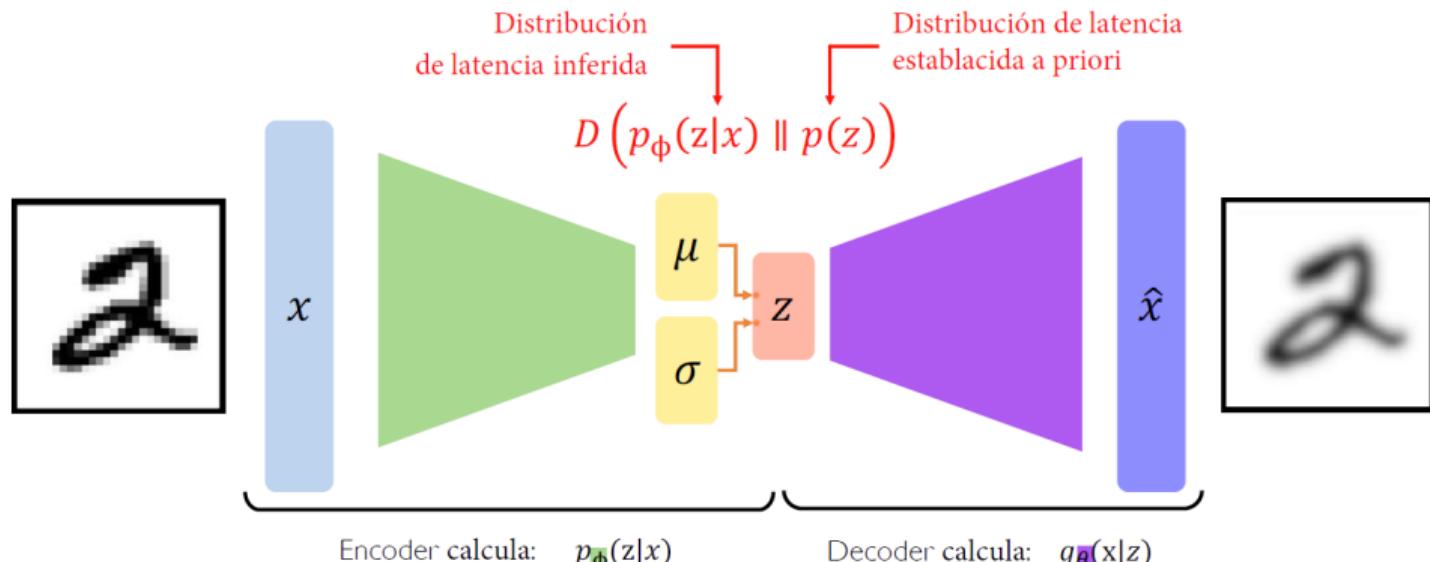
$$\mathcal{L}(\phi, \theta, x) = (\text{reconstruction loss}) + (\text{termino de regularización})$$

# Optimización de la VAE



$$\mathcal{L}(\phi, \theta, x) = \text{(reconstruction loss)} + \text{(termino de regularización)}$$

# Optimización de la VAE



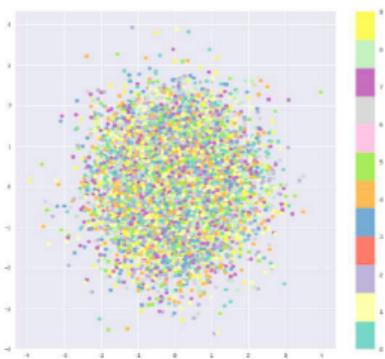
$$\mathcal{L}(\phi, \theta, x) = \text{(reconstruction loss)} + \text{(termino de regularización)}$$

# Los *a priors* de la distribución latente

$$D(p_{\phi}(z|x) \parallel p(z))$$

↑                           ↑  
Distribución              Distribución de latencia  
de latencia inferida    establecida a priori

## La elección común del prior:



$$p(z) = \mathcal{N}(\mu = 0, \sigma^2 = 1)$$

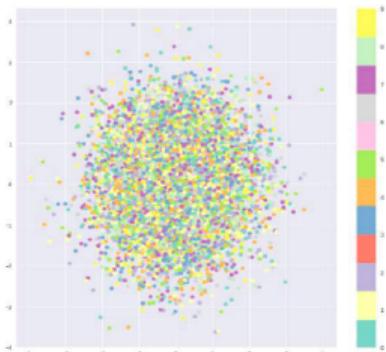
- Alienta a las codificaciones a distribuirlas uniformemente alrededor del centro del espacio latente
- Penaliza a la red cuando intente engañar agrupando puntos en regiones específicas (es decir, memorizando los datos)

# Los *a priors* de la distribución latente

$$\begin{aligned} D(p_{\phi}(z|x) \parallel p(z)) \\ = -\frac{1}{2} \sum_{j=0}^{k-1} (\sigma_j + \mu_j^2 - 1 - \log \sigma_j) \end{aligned}$$

KL-divergencia entre las dos distribuciones

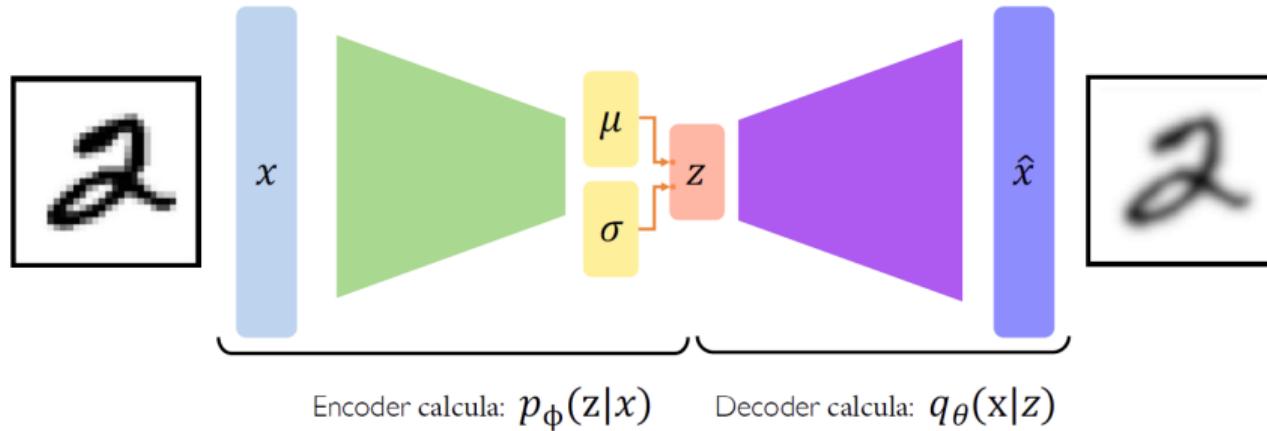
La elección común del prior:



$$p(z) = \mathcal{N}(\mu = 0, \sigma^2 = 1)$$

- Alienta a las codificaciones a distribuir las uniformemente alrededor del centro del espacio latente
- Penaliza a la red cuando intente engañar agrupando puntos en regiones específicas (es decir, memorizando los datos)

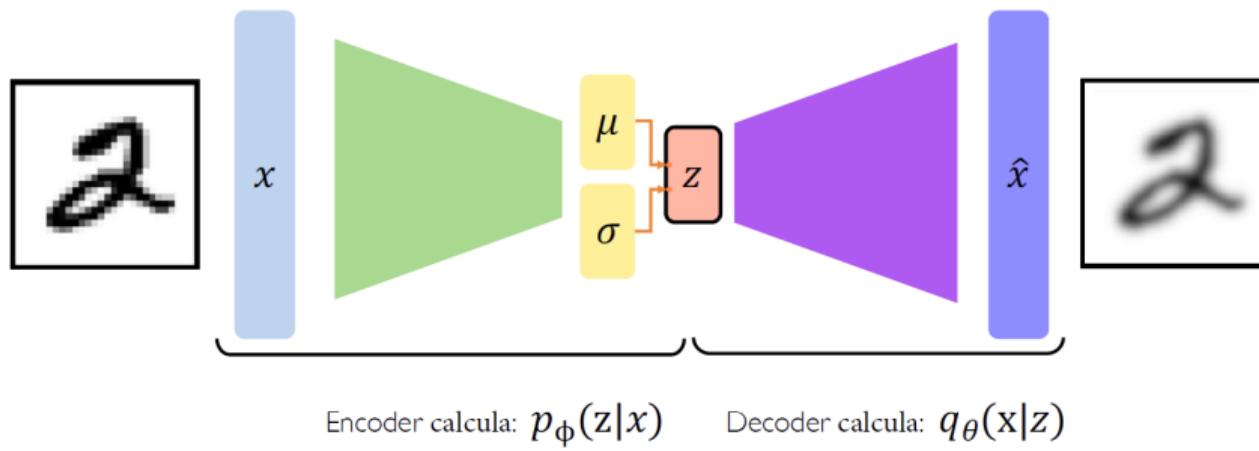
# Gráfo de cálculo de la VAE



$$\mathcal{L}(\phi, \theta) = (\text{reconstruction loss}) + (\text{termino de regularización})$$

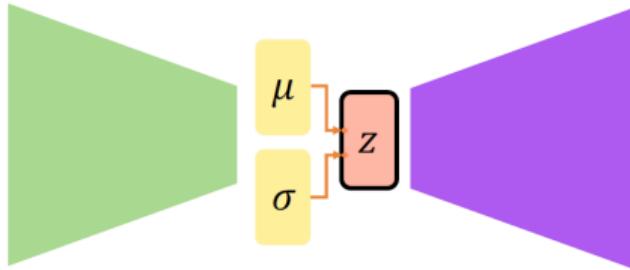
# Gráfo de cálculo de la VAE

**Problema:** No podemos retropropagar los gradientes a través de las capas de muestreo



$$\mathcal{L}(\phi, \theta) = (\text{reconstruction loss}) + (\text{termino de regularización})$$

# Reparametrizando la capa de muestreo



Idea clave:

$$z \sim \mathcal{N}(\mu = 0, \sigma^2 = 1)$$

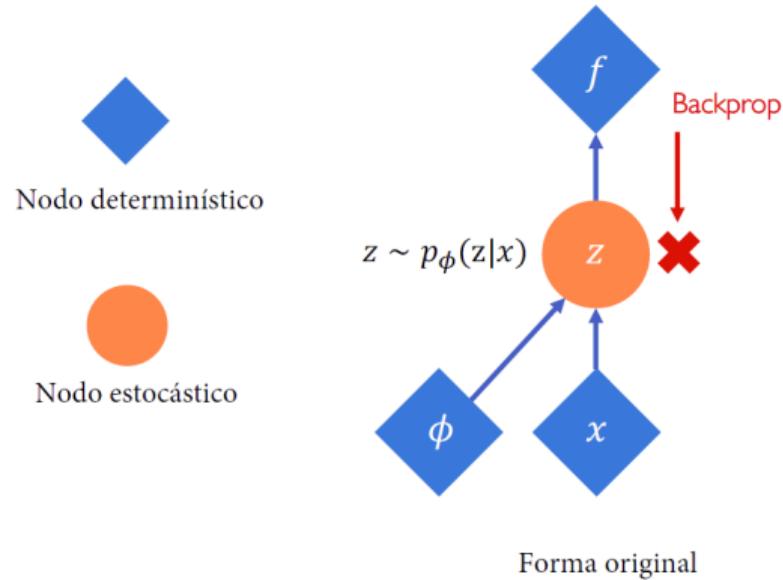
Considerar el vector latente muestreado como una suma de

- un vector  $\mu$  fijo
- y un vector  $\sigma$  fijo, escalado por constantes aleatorias extraídas de la distribución *a prior*

$$z = \mu + \sigma \odot \epsilon$$

$$\text{donde } \epsilon \sim \mathcal{N}(0, 1)$$

# Reparametrizando la capa de muestreo



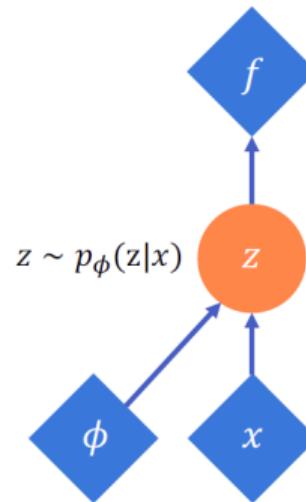
# Reparametrizando la capa de muestreo



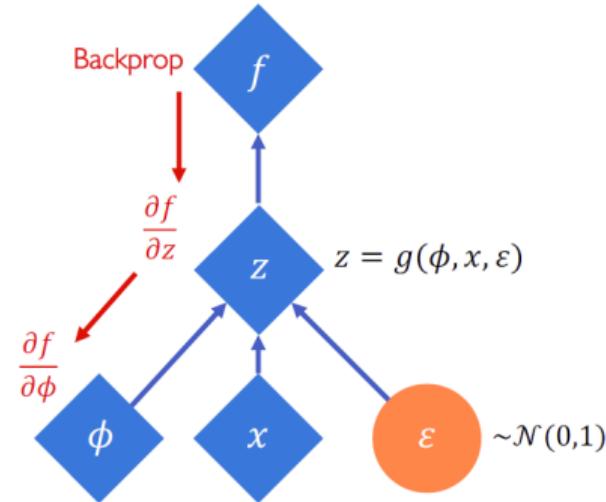
Nodo determinístico



Nodo estocástico



Forma original



Forma reparametrizada

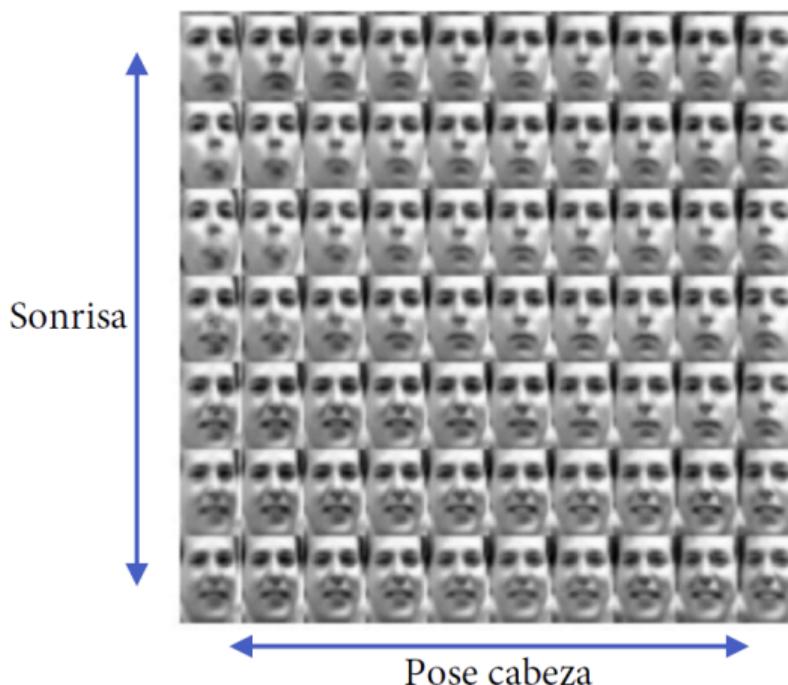
# VAEs: Perturbación latente

- Aumentar o disminuir lentamente una **sola variable latente**
- Mantener todas las demás variables fijas



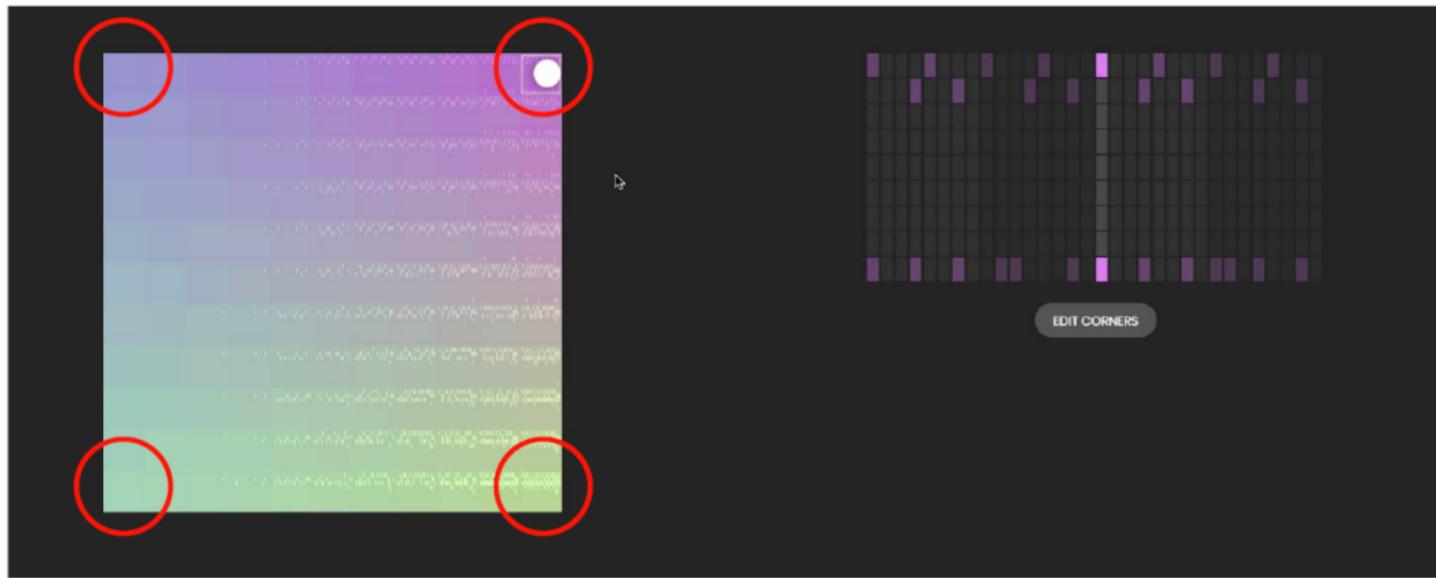
Diferentes dimensiones de  $z$  codifica **diferentes características latentes interpretables**

# VAEs: Perturbación latente



- Idealmente, queremos variables latentes que no estén correlacionadas entre sí
- Aplicar el priorato diagonal a las variables latentes para fomentar independencia
- **Desenredo** (*Disentanglement*)

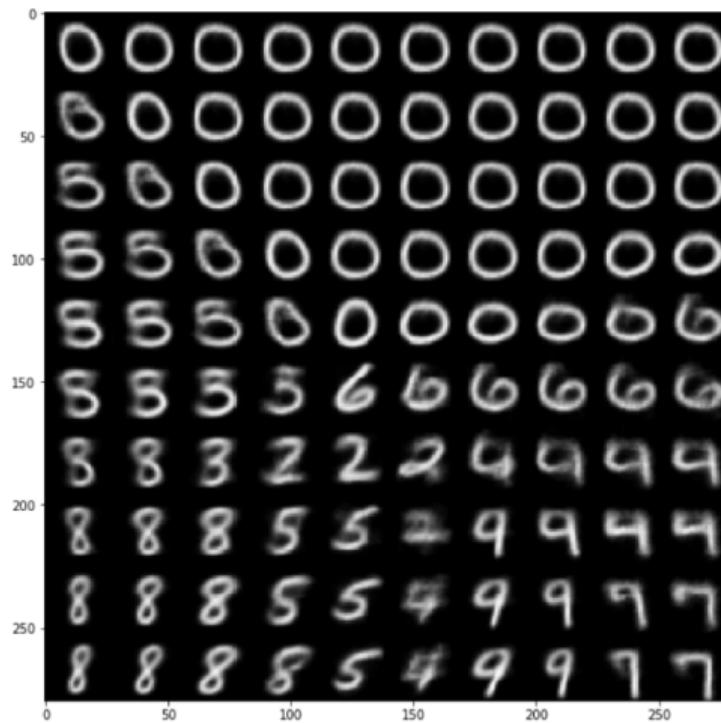
# VAEs: Perturbación latente



Google BeatBlender

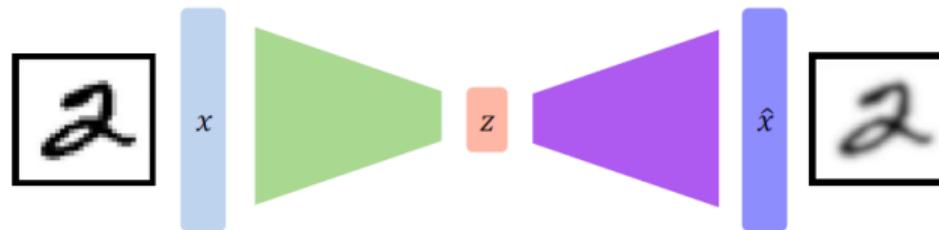
► AI Experiments: Beat Blender

# VAEs: Perturbación latente



# Resumen de las VAEs

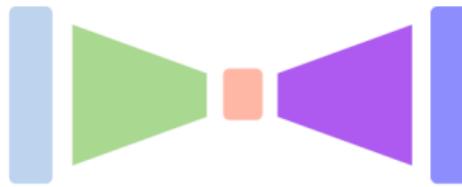
- 1 Comprimir la representación del mundo a algo que podamos usar para aprender
- 2 La reconstrucción permite un aprendizaje no supervisado (¡sin etiquetas!)
- 3 El truco de la reparameterización para entrenar de extremo a extremo
- 4 Interpretar las variables latentes ocultas utilizando la perturbación
- 5 Generar nuevos ejemplos



# Modelado Generativo Profundo: Resumen

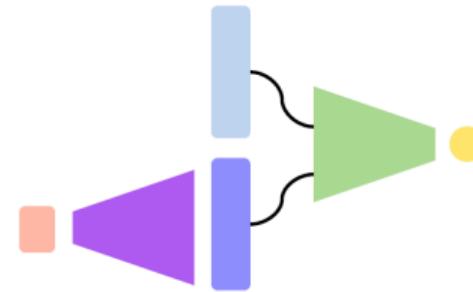
## Autoencoders and Variational Autoencoders (VAEs)

- Aprende el espacio latente de **menor dimensión** y toma **muestras** para generar reconstrucciones de entrada



## Generative Adversarial Networks (GANs)

- Redes de generadores y discriminadores en competencia



# ¡Muchas gracias por su atención!

*¿Preguntas?*



Contacto: Marco Teran  
webpage: [marcoteran.github.io/](https://marcoteran.github.io/)  
e-mail: [marco.teran@usa.edu.co](mailto:marco.teran@usa.edu.co)

