Autocodificadores III

Clase 12

Dra. Wendy Aguilar

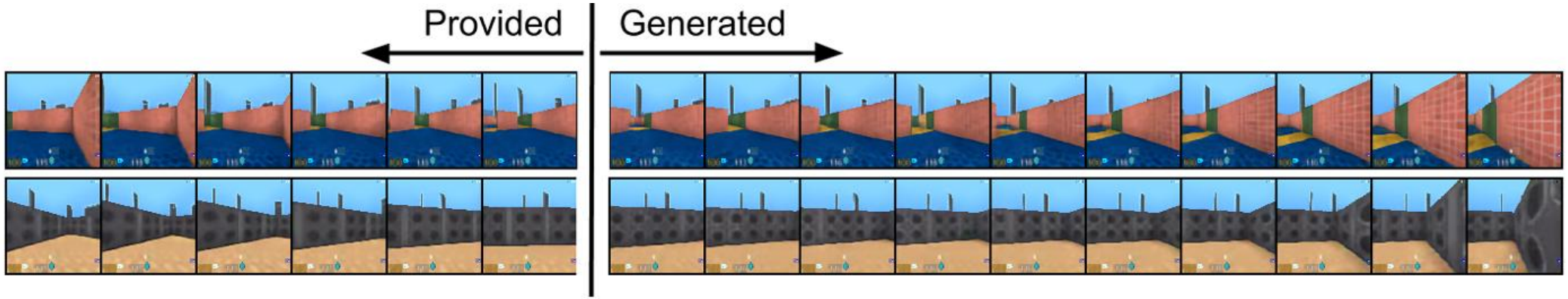
Modelos Generativos Profundos

UN ENFOQUE DESDE LA CREATIVIDAD COMPUTACIONAL

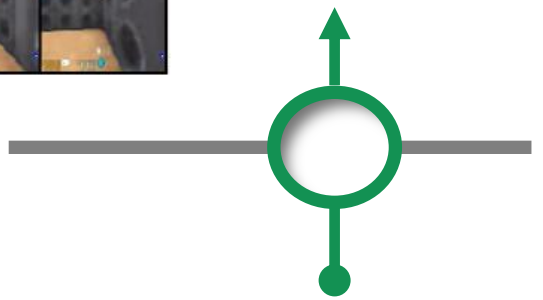
VQ-VAEs

Proyecto de profundización

En equipo



2017



VQ-VAE (Vector Quantized VAE)

Modelo que combina las ideas de los VAEs con una **codificación discreta** basada en **vector quantization**. A diferencia de los VAE tradicionales, que utilizan variables latentes continuas, **VQ-VAE aprende una representación latente discreta**, lo cual es útil para tareas como síntesis de audio, compresión de imágenes y modelado secuencial. Esta representación discreta también permite usar modelos autoregresivos potentes (como PixelCNN) sobre los códigos latentes, mejorando la calidad de las muestras generadas.

van den Oord, A., Vinyals, O., & Kavukcuoglu, K. (2017). *Neural Discrete Representation Learning*. arXiv preprint arXiv:1711.00937.

https://keras.io/examples/generative/vq_vae/

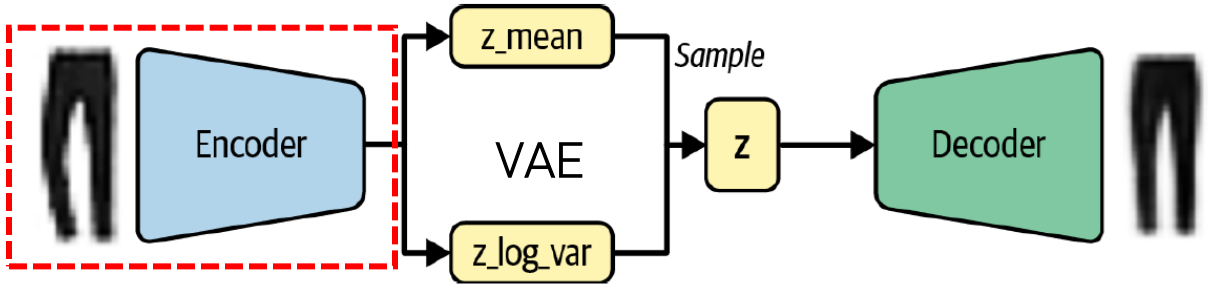
Principales motivaciones de los VQ-VAEs

- Superar las limitaciones de los VAEs continuos (reconstrucciones borrosas y dificultad para modelar estructuras secuenciales),

- Querían un modelo que produjera representaciones discretas más cercanas a cómo funcionan los datos reales (símbolos, fonemas, palabras).

- Y querían un prior aprendido, en vez de forzar siempre un gaussiano fijo.

¿Qué cambios hicieron con respecto al VAE?

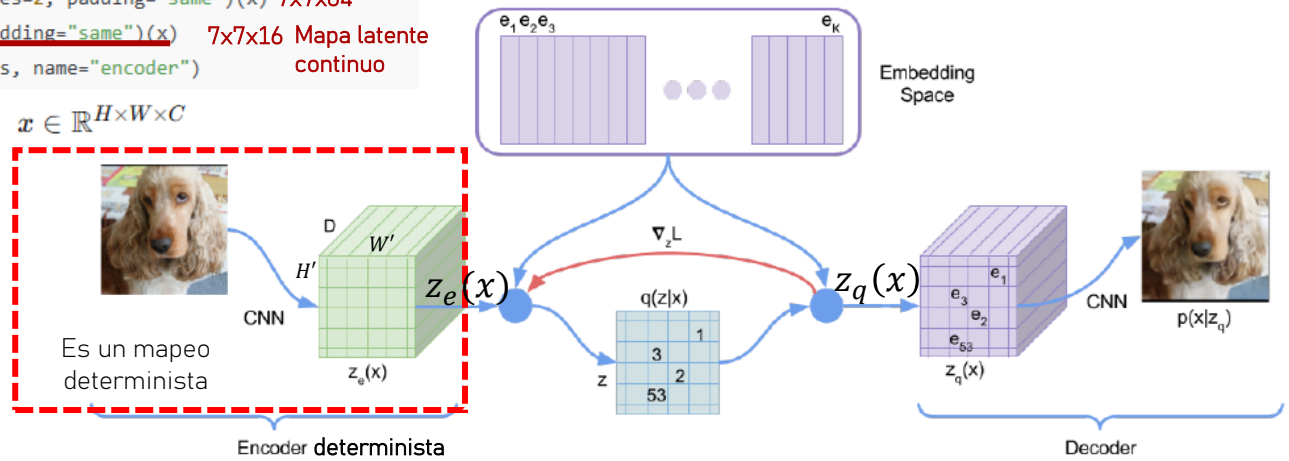


Dada una imagen $\boldsymbol{x}$, produce una distribución $\mathcal{N}(0, I)$ particular para esa

```
latent_dim=16

encoder_inputs = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(32, 3, activation="relu", strides=2, padding="same")(encoder_inputs) 14x14x32
x = layers.Conv2D(64, 3, activation="relu", strides=2, padding="same")(x) 7x7x64
encoder_outputs = layers.Conv2D(latent_dim, 1, padding="same")(x)   7x7x16  Mapa latente continuo
return keras.Model(encoder_inputs, encoder_outputs, name="encoder")
```

VQ-VAE



Dimensiones reducidas de la imagen, obtenidas tras convoluciones con *stride* o *pooling*.

Dada una entrada $\boldsymbol{x}$, produce un mapa latente continuo

$$z_e(x) = \text{Enc}(x) \in \mathbb{R}^{H' \times W' \times D}$$

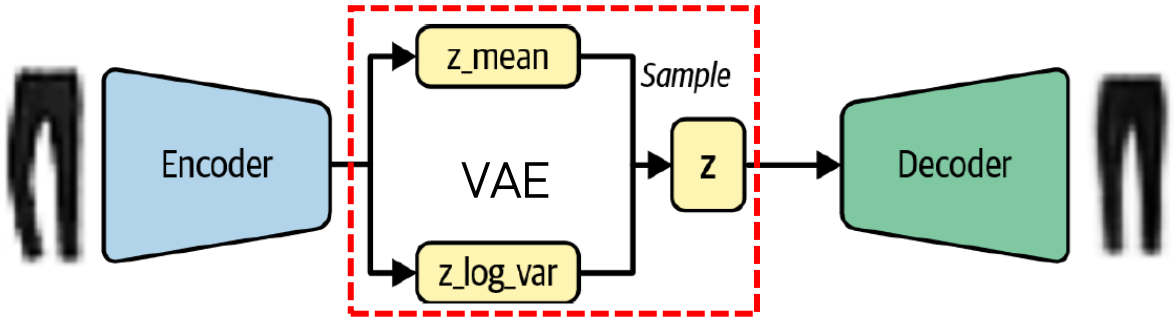
encoder

Número de **filtros** de la última capa convolucional del encoder.
Cada celda (h', w') tiene asociado un vector latente en $\mathbb{R}^D$.

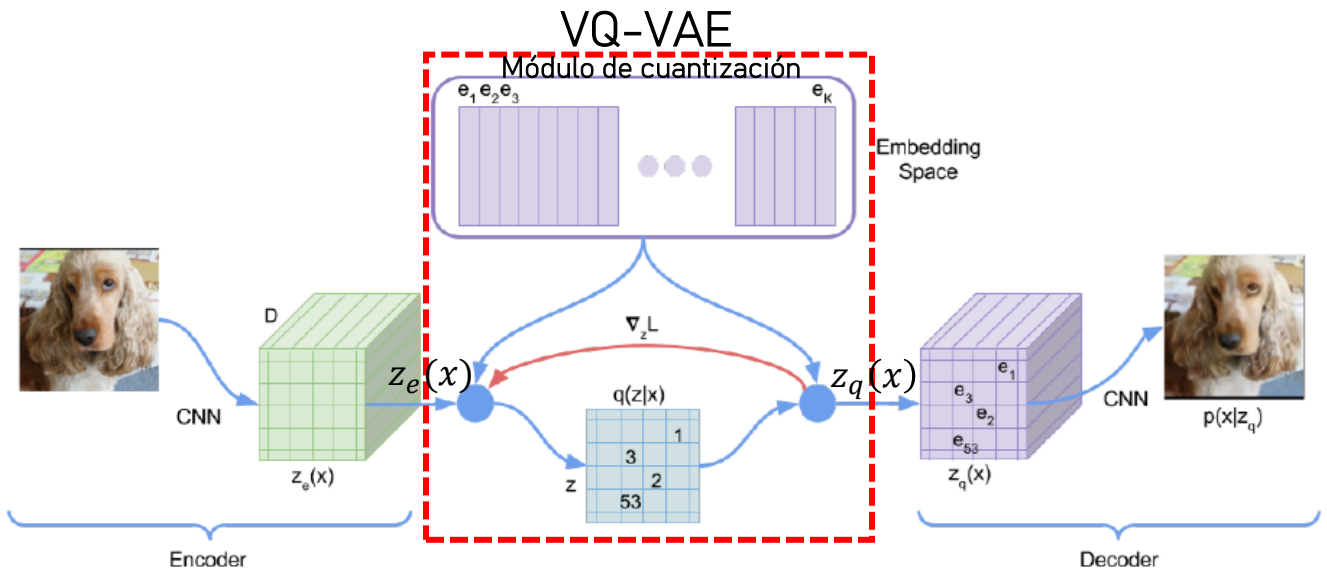
Piénsalo como:

- un **descriptor que resume texturas, bordes y patrones detectados por el encoder convolucional** de la región de la imagen cubierta por el campo receptivo de esa posición.

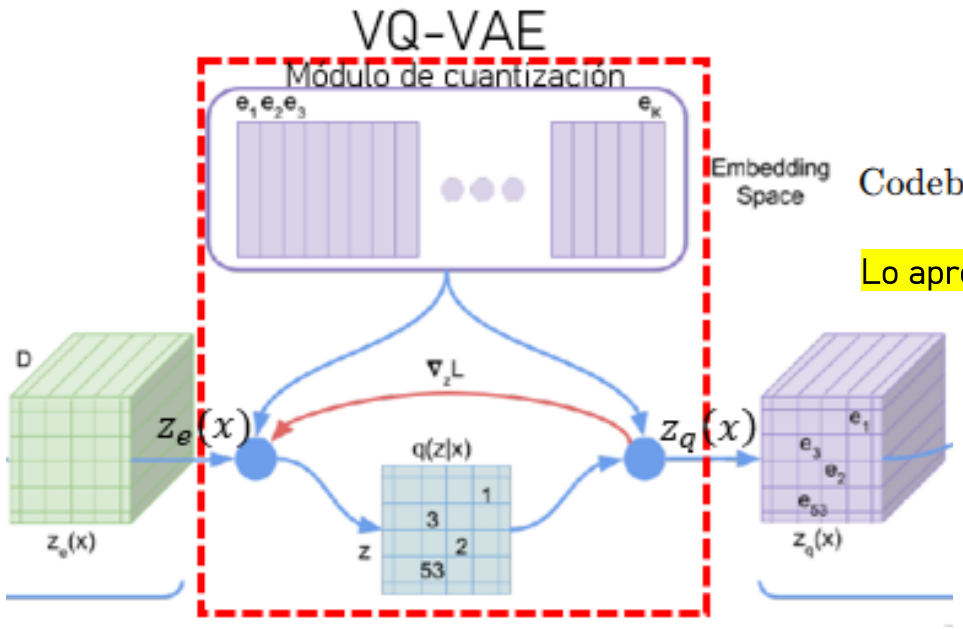
¿Qué cambios hicieron con respecto al VAE?



Se muestrea aleatoriamente un vector latente continuo z de la distribución entregada por el encoder.



Transforma el mapa latente continuo $z_e(x) \in \mathbb{R}^{H' x W' x D}$ en un mapa latente discreto $z_q(x) \in \mathbb{R}^{H' x W' x D}$.



$\text{Codebook} \in \mathbb{R}^{K \times D}$

K= tamaño del vocabulario

Lo aprende durante el entrenamiento.

Ejemplo: Codebook con $K = 4, D = 3$

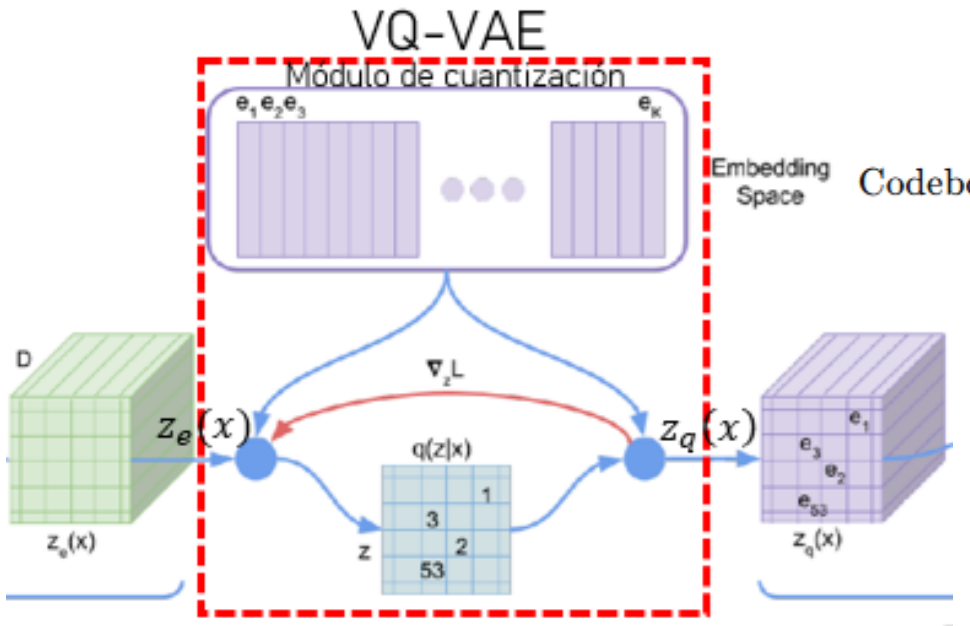
$$\text{Codebook} = \begin{matrix} e_1 & e_2 & e_3 & e_4 \end{matrix} \begin{bmatrix} 0.2 & 1.3 & -0.8 & 0.0 \\ -1.1 & 0.5 & 0.9 & -0.3 \\ 0.7 & -0.2 & 1.1 & 0.4 \end{bmatrix}$$

Piensa en cada $e_1$como:

- un **prototipo aprendido** de una "pieza visual" recurrente.
- Es como una **palabra visual** del vocabulario del modelo.

Por ejemplo:

- e_{37} podría representar el trazo vertical grueso que aparece en muchos dígitos.
- e_{142} podría representar una curvatura típica de bordes redondeados.
- e_{311} podría codificar una combinación de borde + textura.

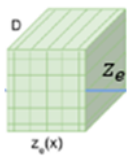


$\text{Codebook} \in \mathbb{R}^{K \times D}$

K= tamaño del vocabulario

Ejemplo: Codebook con $K = 4$, $D = 3$

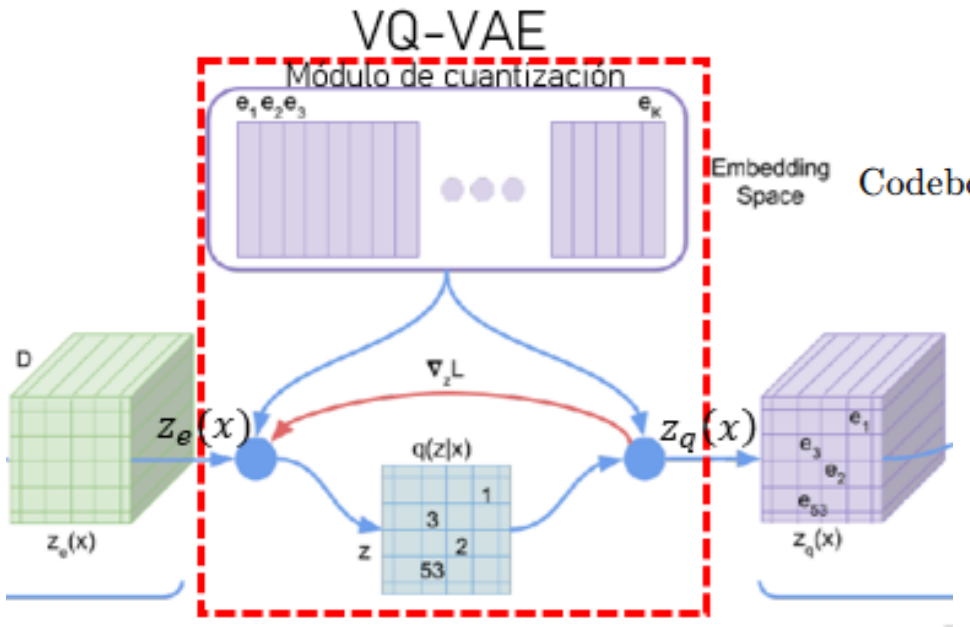
$$\text{Codebook} = \begin{matrix} \\ \\ \\ \end{matrix} \begin{matrix} e_1 & e_2 & e_3 & e_4 \end{matrix} \begin{bmatrix} 0.2 & 1.3 & -0.8 & 0.0 \\ -1.1 & 0.5 & 0.9 & -0.3 \\ 0.7 & -0.2 & 1.1 & 0.4 \end{bmatrix}$$



1. **Recibe** el mapa latente continuo $z_e(x) \in \mathbb{R}^{H' \times W' \times D}$.

Supongamos que el encoder produce en cierta posición (h', w'):

$$z_e(h', w') = (0.1,\ -1.0,\ 0.6)$$



$\text{Codebook} \in \mathbb{R}^{K \times D}$

K= tamaño del vocabulario

Ejemplo: Codebook con $K = 4$, $D = 3$

$$\text{Codebook} = \begin{array}{c} \begin{matrix} e_1 & e_2 & e_3 & e_4 \end{matrix} \\ \begin{bmatrix} 0.2 & 1.3 & -0.8 & 0.0 \\ -1.1 & 0.5 & 0.9 & -0.3 \\ 0.7 & -0.2 & 1.1 & 0.4 \end{bmatrix} \end{array}$$

1. **Recibe** el mapa latente continuo $z_e(x) \in \mathbb{R}^{H' \times W' \times D}$.

2. **Calcula** la distancia entre cada vector $z_e(h', w')$ y todos los embeddings del codebook $\{e_k\}_{k=1}^{K}$.

$$d_k = \|z_e(h', w') - e_k\|^2$$

distancia euclídea al cuadrado

Supongamos que el encoder produce en cierta posición (h', w'):

$$z_e(h', w') = (0.1,\ -1.0,\ 0.6)$$

1. Con $e_1 = (0.2, -1.1, 0.7)$:

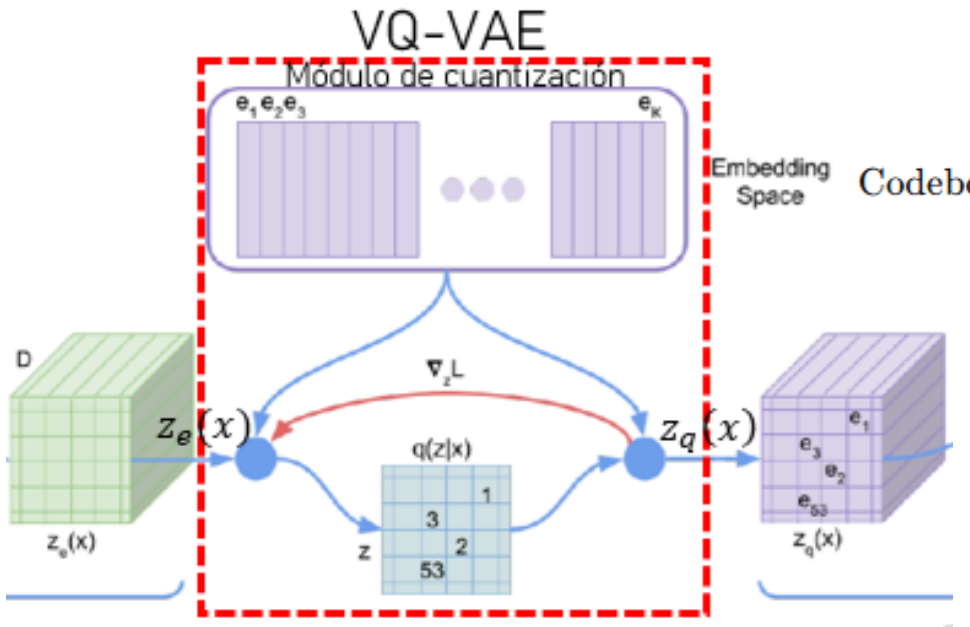
$$d_1 = (0.1 - 0.2)^2 + (-1.0 - (-1.1))^2 + (0.6 - 0.7)^2$$

$$d_1 = (-0.1)^2 + (0.1)^2 + (-0.1)^2 = 0.01 + 0.01 + 0.01 = \underline{0.03}$$

2. Con $e_2 = (1.3, 0.5, -0.2)$: $d_2 = \underline{4.33}$

3. Con $e_3 = (-0.8, 0.9, 1.1)$: $d_3 = \underline{4.67}$

4. Con $e_4 = (0.0, -0.3, 0.4)$: $d_4 = \underline{0.54}$



$\text{Codebook} \in \mathbb{R}^{K \times D}$

K= tamaño del vocabulario

Ejemplo: Codebook con $K = 4$, $D = 3$

$$\text{Codebook} = \begin{array}{c} \begin{matrix} e_1 & e_2 & e_3 & e_4 \end{matrix} \\ \begin{bmatrix} 0.2 & 1.3 & -0.8 & 0.0 \\ -1.1 & 0.5 & 0.9 & -0.3 \\ 0.7 & -0.2 & 1.1 & 0.4 \end{bmatrix} \end{array}$$

1. **Recibe** el mapa latente continuo $z_e(x) \in \mathbb{R}^{H' \times W' \times D}$.

2. **Calcula** la distancia entre cada vector $z_e(h', w')$ y todos los embeddings del codebook $\{e_k\}_{k=1}^{K}$.

$$d_k = \|z_e(h', w') - e_k\|^2$$

distancia euclídea al cuadrado

3. **Selecciona** el embedding más cercano e_k

$e_1 = 0.03$

Supongamos que el encoder produce en cierta posición (h', w'):

$$z_e(h', w') = (0.1,\ -1.0,\ 0.6)$$

1. Con $e_1 = (0.2, -1.1, 0.7)$:

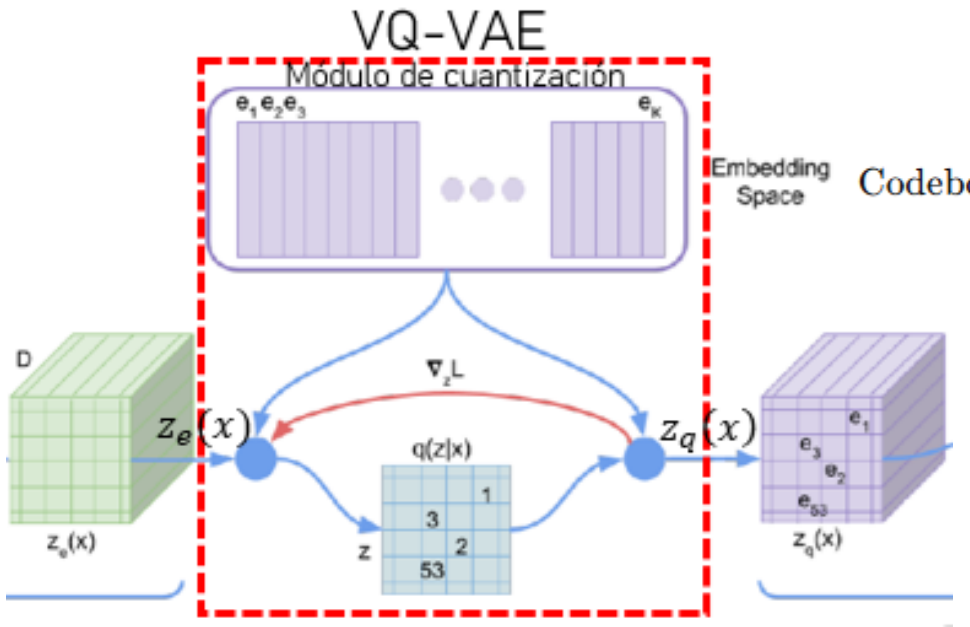
$$d_1 = (0.1 - 0.2)^2 + (-1.0 - (-1.1))^2 + (0.6 - 0.7)^2$$

$$d_1 = (-0.1)^2 + (0.1)^2 + (-0.1)^2 = 0.01 + 0.01 + 0.01 = 0.03$$

2. Con $e_2 = (1.3, 0.5, -0.2)$: $d_2 = 4.33$

3. Con $e_3 = (-0.8, 0.9, 1.1)$: $d_3 = 4.67$

4. Con $e_4 = (0.0, -0.3, 0.4)$: $d_4 = 0.54$



$\text{Codebook} \in \mathbb{R}^{K \times D}$

K= tamaño del vocabulario

Ejemplo: Codebook con $K = 4$, $D = 3$

$$\text{Codebook} = \begin{array}{c} \begin{matrix} e_1 & e_2 & e_3 & e_4 \end{matrix} \\ \begin{bmatrix} 0.2 & 1.3 & -0.8 & 0.0 \\ -1.1 & 0.5 & 0.9 & -0.3 \\ 0.7 & -0.2 & 1.1 & 0.4 \end{bmatrix} \end{array}$$

1. **Recibe** el mapa latente continuo $z_e(x) \in \mathbb{R}^{H' \times W' \times D}$.

2. **Calcula** la distancia entre cada vector $z_e(h', w')$ y todos los embeddings del codebook $\{e_k\}_{k=1}^{K}$.

$$d_k = \|z_e(h', w') - e_k\|^2$$

distancia euclídea al cuadrado

3. **Selecciona** el embedding más cercano e_k

4. **Reemplaza** cada vector continuo por su embedding más cercano.

$$z_q(h', w') = e_{k^*(h', w')} = e_1$$

Supongamos que el encoder produce en cierta posición (h', w'):

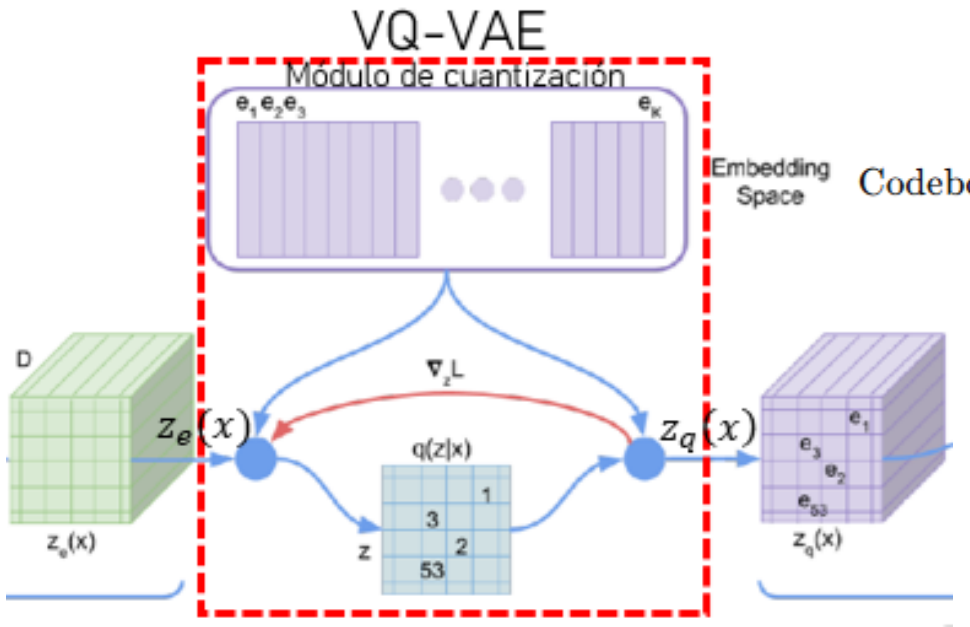
Antes (continuo): $z_e(h', w') = (0.1,\ -1.0,\ 0.6)$

1. Con $e_1 = (0.2, -1.1, 0.7)$:

$$d_1 = (0.1 - 0.2)^2 + (-1.0 - (-1.1))^2 + (0.6 - 0.7)^2$$

$$d_1 = (-0.1)^2 + (0.1)^2 + (-0.1)^2 = 0.01 + 0.01 + 0.01 = 0.03$$

Después del reemplazo (discreto): $z_q(h', w') = (0.2,\ -1.1,\ 0.7)$



$\text{Codebook} \in \mathbb{R}^{K \times D}$

K= tamaño del vocabulario

Ejemplo: Codebook con $K = 4$, $D = 3$

$$\text{Codebook} = \begin{matrix} \\ \\ \\ \end{matrix} \begin{matrix} e_1 & e_2 & e_3 & e_4 \end{matrix} \begin{bmatrix} 0.2 & 1.3 & -0.8 & 0.0 \\ -1.1 & 0.5 & 0.9 & -0.3 \\ 0.7 & -0.2 & 1.1 & 0.4 \end{bmatrix}$$

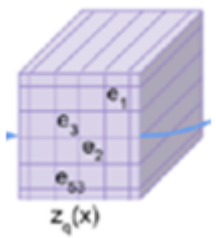
1. **Recibe** el mapa latente continuo $z_e(x) \in \mathbb{R}^{H' \times W' \times D}$.

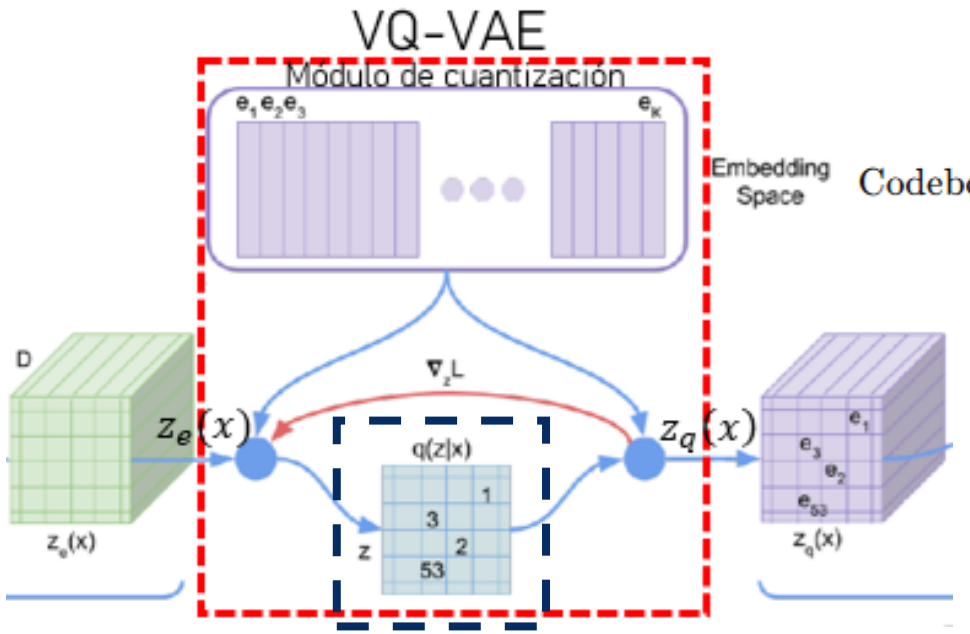
2. **Calcula** la distancia entre cada vector $z_e(h', w')$ y todos los embeddings del codebook $\{e_k\}_{k=1}^{K}$.

3. **Selecciona** el embedding más cercano e_k

4. **Reemplaza** cada vector continuo por su embedding más cercano.

5. **Genera** el mapa latente discreto $z_q(x) \in \mathbb{R}^{H' \times W' \times D}$.



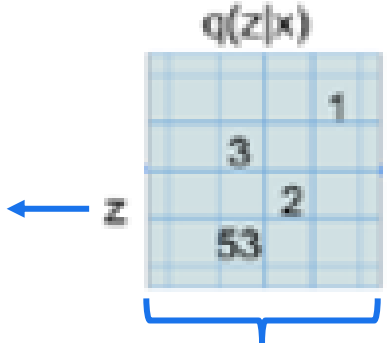


$\text{Codebook} \in \mathbb{R}^{K \times D}$

K= tamaño del vocabulario

Cada renglón corresponde a una posición en el mapa latente.

Es una tabla de decisiones: "dice qué código del codebook corresponde a cada vector continuo del codificador".



- Cada fila de la matriz es **"one-hot"** (un 1 en la posición del índice elegido, 0 en las demás).
- Para simplificar la ilustración, en lugar de dibujar el vector one-hot completo, los autores ponen directamente el **número del índice** (el entero entre 0 y $K-1$).

Las probabilidades de asignar ese vector a cada entrada del codebook.

$q(z|x)$ se interpreta como una distribución categórica degenerada:

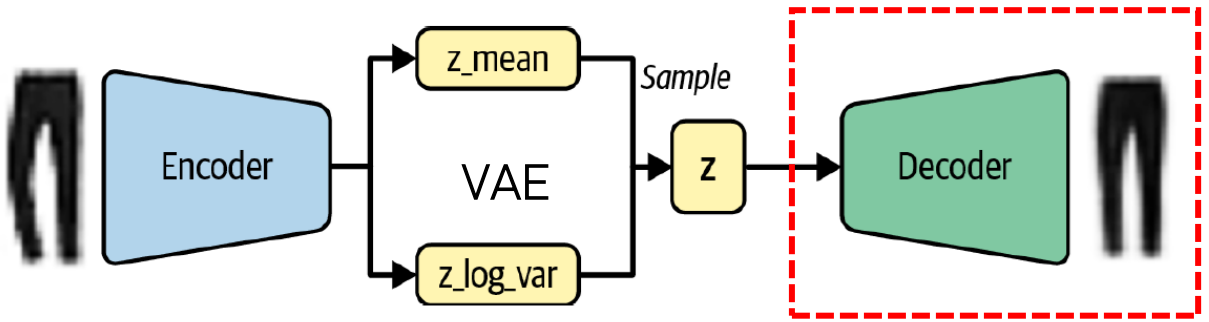
- Para un x dado, la probabilidad de elegir el vector latente más cercano es 1, y 0 para los demás.
- En la práctica, simplemente "elige el índice del codebook más cercano a $z_e(x)$." usando la **distancia euclidiana (L2)**

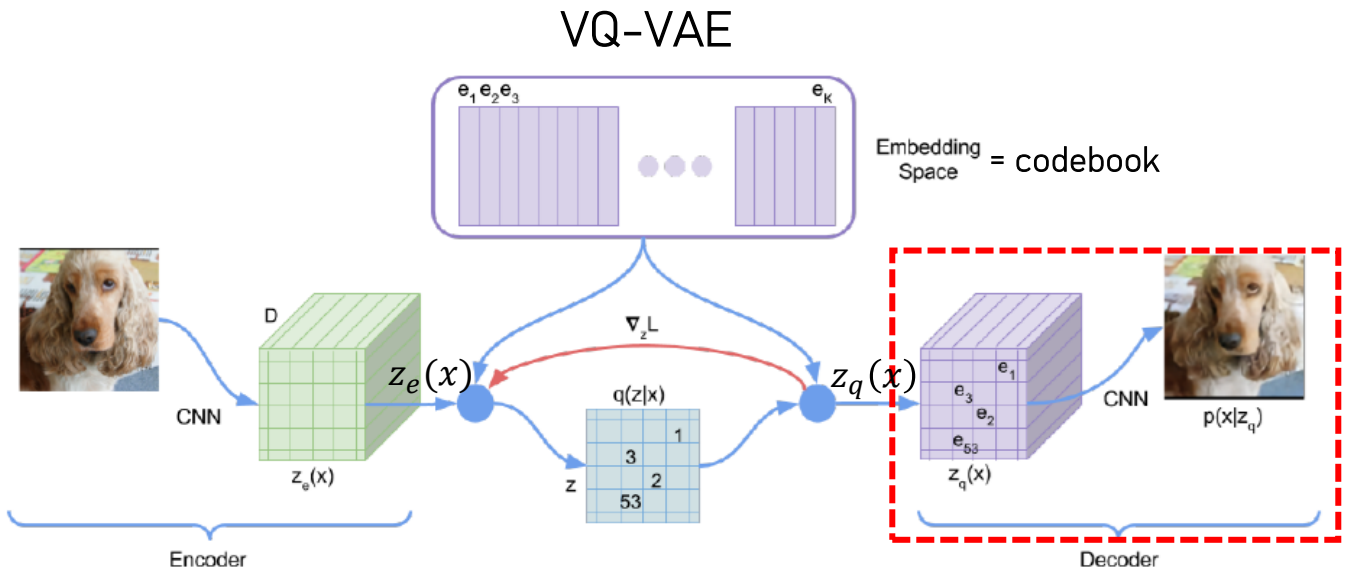
$$q(z=k|x) = \begin{cases} 1 & \text{for } k = \text{argmin}_j \|z_e(x) - e_j\|_2, \\ 0 & \text{otherwise} \end{cases}$$

1. **Recibe** el mapa latente continuo $z_e(x) \in \mathbb{R}^{H' \times W' \times D}$.
2. **Calcula** la distancia entre cada vector $z_e(h', w')$ y todos los embeddings del codebook $\{e_k\}_{k=1}^{K}$.
3. **Selecciona** el embedding más cercano e_k
4. **Reemplaza** cada vector continuo por su embedding más cercano.
5. **Genera** el mapa latente discreto $z_q(x) \in \mathbb{R}^{H' \times W' \times D}$.

Formalmente, esto corresponde a calcular la distribución $q(z \mid x)$.

¿Qué cambios hicieron con respecto al VAE?

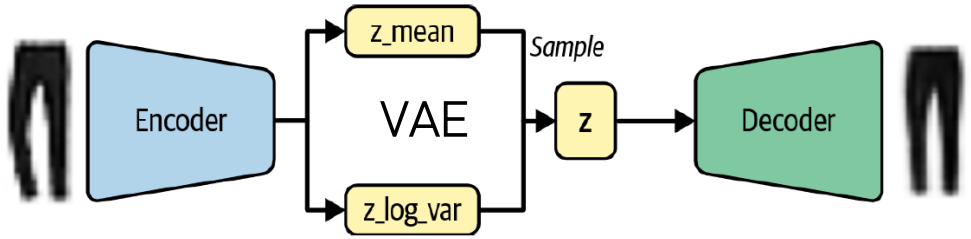




- Es esencialmente la misma (red neuronal que mapea un vector latente a una reconstrucción de datos).
- Lo que cambia no es el diseño del decoder, sino el tipo de representación latente que recibe como entrada.

¿Qué cambios hicieron con respecto al VAE?

Función de pérdida

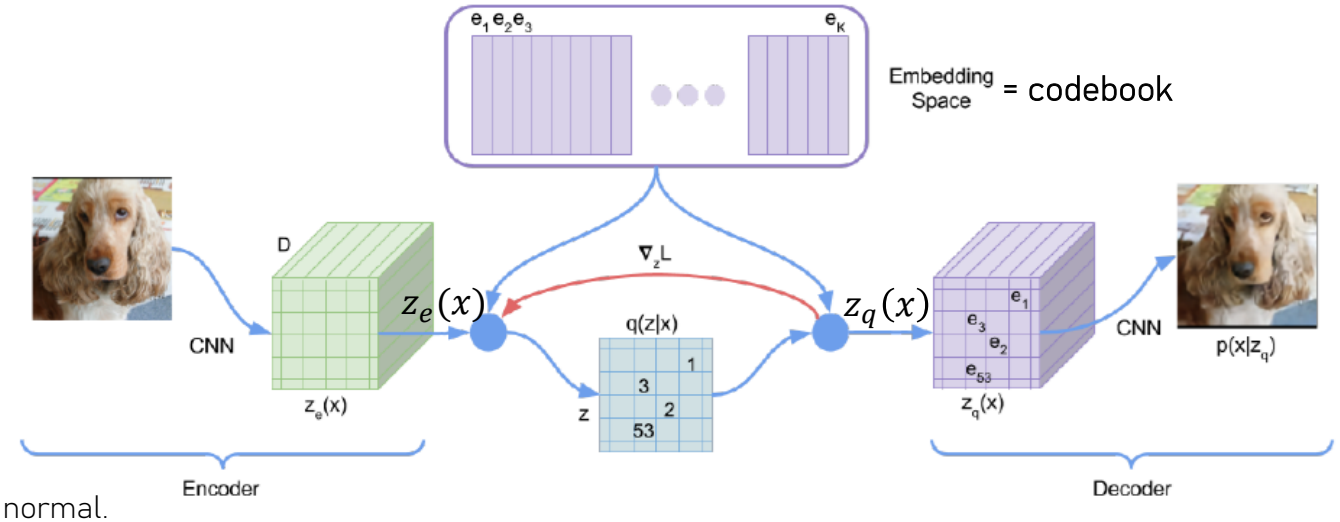


$$\mathcal{L}_{VAE} = \underbrace{-\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)]}_{\text{Pérdida de reconstrucción}} + \underbrace{D_{KL}(q_\phi(z|x) \,\|\, p(z))}_{\text{Regularización KL}}$$

El VAE busca dos objetivos:

- **Reconstrucción fiel** de la entrada.
- **Regularización** del espacio latente para que sea "suave" y muestreable.

VQ-VAE



$$\mathcal{L} = \underbrace{\log p\big(x \mid z_q(x)\big)}_{\text{Reconstrucción}} + \underbrace{\|\text{sg}[z_e(x)] - e_k\|^2}_{\text{Codebook loss}} + \underbrace{\beta \|z_e(x) - \text{sg}[e]\|^2}_{\text{Commitment loss}}$$

El VQ-VAE busca tres objetivos:

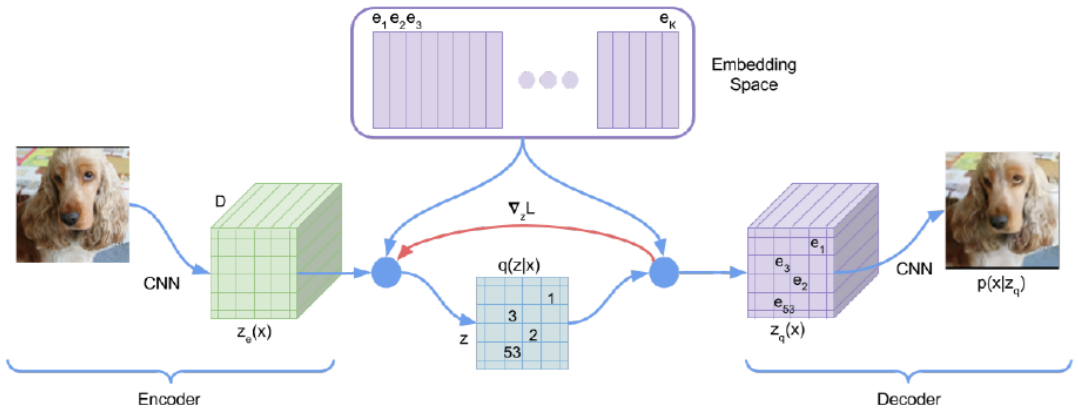
- **Reconstrucción fiel** de la entrada.
- **Aprender un codebook útil** (que el diccionario se adapte al dataset y capture sus regularidades).
- **Evitar que el encoder ignore el diccionario** (para que los vectores continuos no se alejen y realmente usen el vocabulario).

$\text{sg}[\cdot]$ significa **stop-gradient.**

- En el forward se deja pasar el valor normal.
- En el backward se bloquea el gradiente (como si fuera una constante).

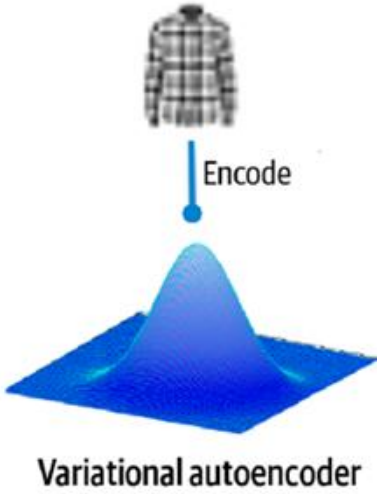
¿Cómo aprende VQ-VAE?

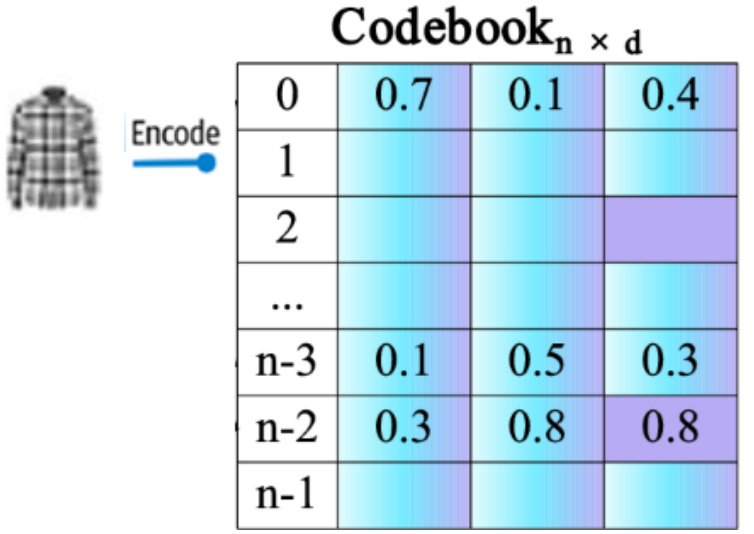
Aprende entrenando en conjunto el encoder, el codebook y el decoder



1. **Codificación**
   - El encoder convolucional transforma la imagen x en un mapa continuo $z_e(x)$.
   - Cada celda (h', w') es un vector en $\mathbb{R}^D$.
2. **Cuantización**
   - Cada vector $z_e(h', w')$ se reemplaza por su prototipo más cercano en el **codebook**.
   - Se obtiene $z_q(x)$, un mapa latente discreto de embeddings.
3. **Decodificación**
   - El decoder toma $z_q(x)$ y reconstruye una imagen $\hat{x}$.
4. **Cálculo de pérdidas**
   - **Reconstrucción:** mide qué tan bien $\hat{x}$ se parece a x.
   - **Codebook loss:** mueve los embeddings e_k hacia los vectores z_e que los usan.
   - **Commitment loss:** empuja al encoder a no desviarse demasiado del embedding seleccionado.
5. **Actualización de parámetros**
   - El **encoder** aprende a producir vectores que caigan cerca de los embeddings del diccionario.
   - El **codebook** aprende a mover sus vectores para representar mejor patrones frecuentes.
   - El **decoder** aprende a convertir esos embeddings en imágenes realistas.

¿Cómo se generan nuevas imágenes en VQ-VAE?





Un VAE tiene un prior explícito $p(z) = \mathcal{N}(0, I)$ que permite **muestrear nuevos puntos en el espacio latente** y generar imágenes directamente.

- Interpolaciones
- Edición en el espacio latente
- CVAE

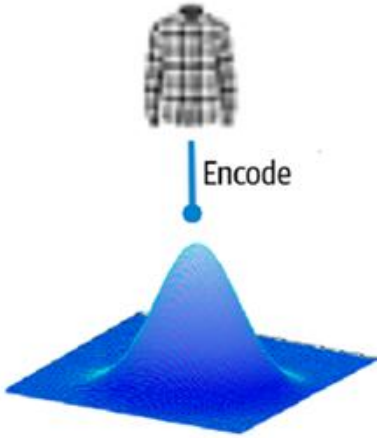
Un VQ-VAE no define un prior sobre los índices del *codebook*.

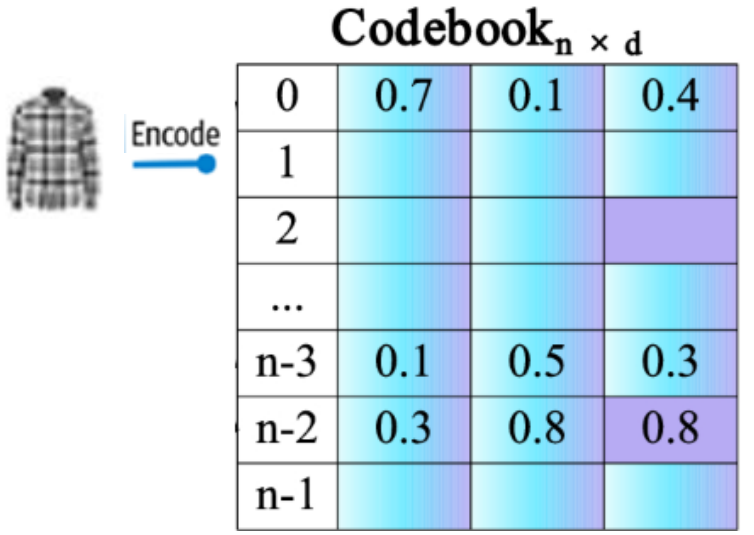
El *codebook* te da un vocabulario discreto, pero no dice cómo combinarlos para formar imágenes plausibles.

- Sin prior, puedes tomar índices al azar del codebook y pasarlos al decoder → se generarán imágenes.

- Pero… esas imágenes casi siempre son incoherentes, porque los índices aleatorios no siguen la distribución de los datos.

¿Cómo se generan nuevas imágenes en VQ-VAE?





Un VAE tiene un prior explícito $p(z) = \mathcal{N}(0, I)$ que permite **muestrear nuevos puntos en el espacio latente** y generar imágenes directamente.

- Interpolaciones
- Edición en el espacio latente
- CVAE

- Sin prior – No es directamente generativo.
- Para producir imágenes nuevas, requiere un **modelo adicional** (p.ej. PixelCNN o Transformer) que muestree secuencias de códigos del diccionario.
- Una vez generados, el decodificador los traduce en imágenes realistas.

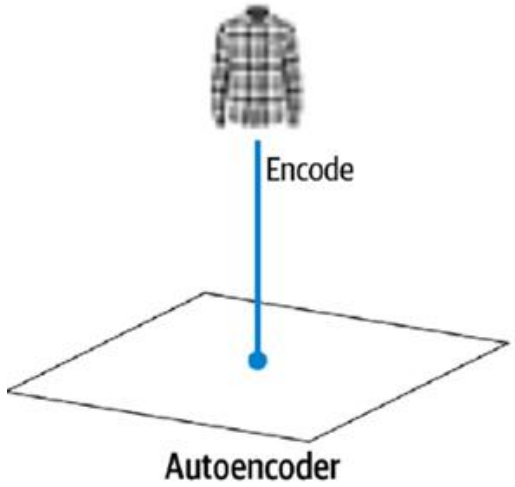
¿Qué lograron?

- Pudieron **modelar imágenes con mayor nitidez** que un VAE clásico.
- **Representaciones de audio** cercanas a fonemas sin supervisión.
- Pusieron las bases de modelos modernos como **VQ-VAE-2, VQ-GAN y DALL·E**, donde las representaciones discretas se tratan como "tokens visuales".

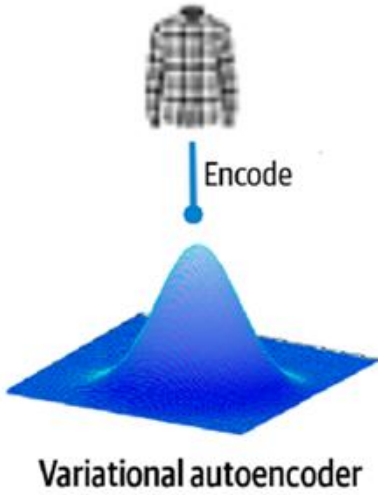
Resumen

Autocodificador determinista vs VAE vs VQ-VAE

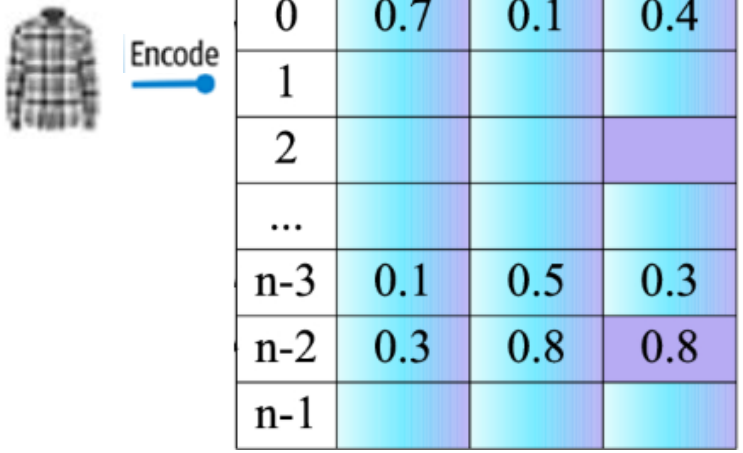
1. Representación latente



AE determinista:
Cada imagen se codifica como un **vector continuo fijo** z.



VAE:
Cada imagen se representa como una **distribución gaussiana** (μ, σ), y el vector $z$se obtiene por muestreo.

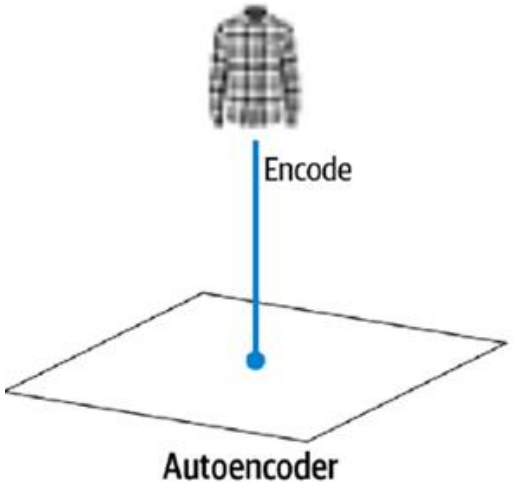


VQ-VAE:
Cada imagen se codifica como un índice discreto, seleccionado de un diccionario finito de "códigos latentes" (codebook).

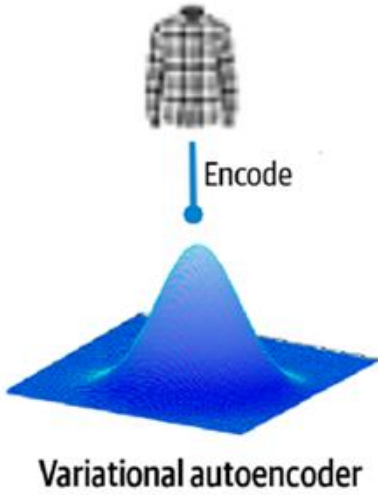
Resumen

Autocodificador determinista vs VAE vs VQ-VAE

2. Regularización del espacio latente



AE: No hay regularización → el espacio latente puede estar disperso y no sigue una distribución conocida.



VAE: Regulariza el latente hacia una **normal estándar** $\mathcal{N}(0, I)$ gracias al término KL.

$\text{Codebook}_{n \times d}$

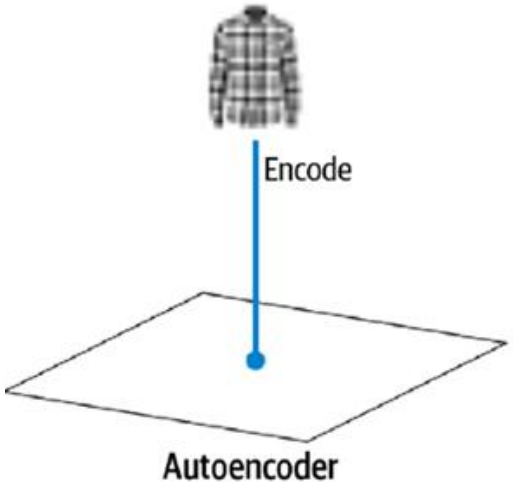
0	0.7	0.1	0.4
1			
2			
...			
n-3	0.1	0.5	0.3
n-2	0.3	0.8	0.8
n-1			

VQ-VAE: El latente se ajusta a un **conjunto discreto de prototipos** (los vectores del codebook), lo que fuerza una estructura organizada.

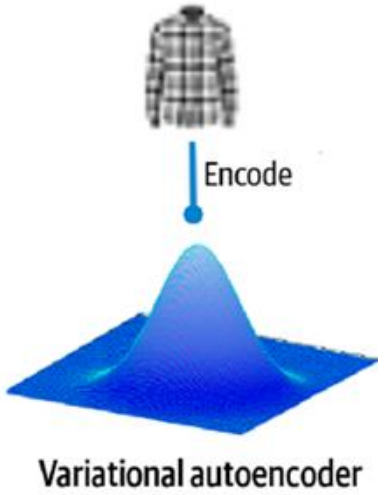
Resumen

Autocodificador determinista vs VAE vs VQ-VAE

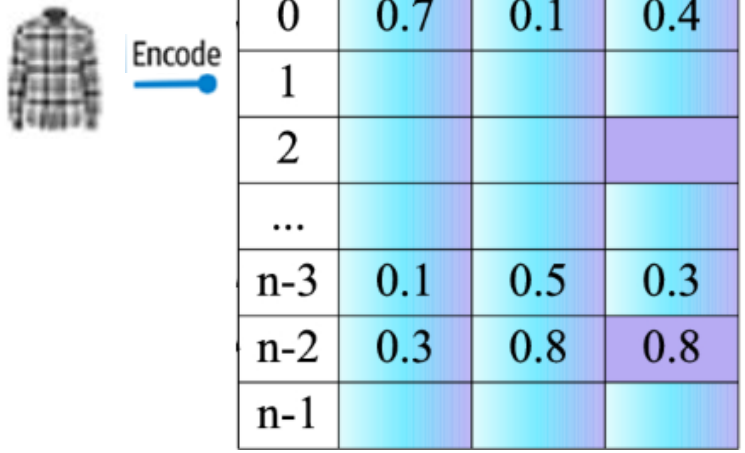
3. Poder generativo



AE: Limitado. Puede decodificar, pero muestrear z al azar da imágenes irreales.



VAE: Generativo. Cualquier $z \sim \mathcal{N}(0, I)$ produce imágenes plausibles.

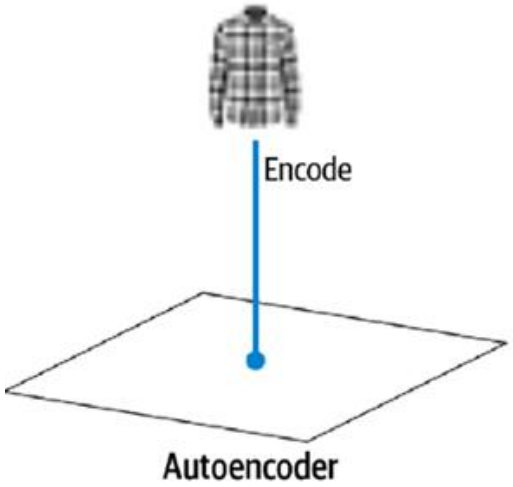


VQ-VAE por sí mismo solo reconstruye imágenes. Para generar imágenes nuevas y realistas necesita un prior adicional (PixelCNN, Transformer o Diffusion) que modele la distribución de los códigos.

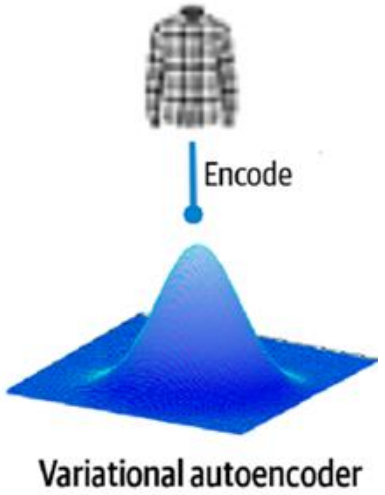
Resumen

Autocodificador determinista vs VAE vs VQ-VAE

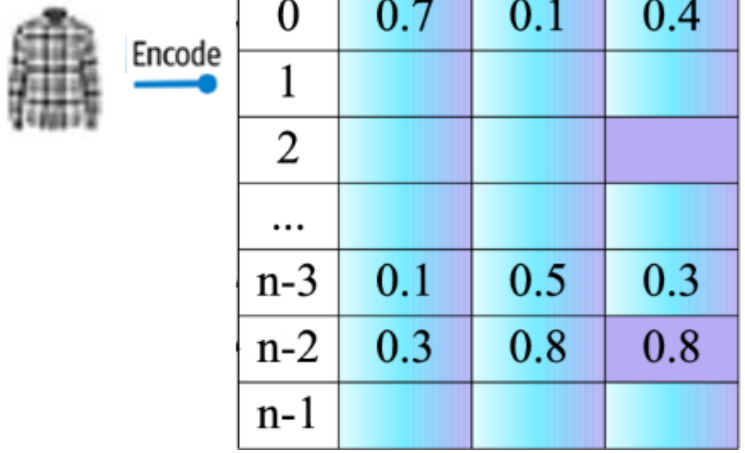
4. Calidad de las reconstrucciones



AE: Reconstrucciones nítidas, porque no sacrifica fidelidad.

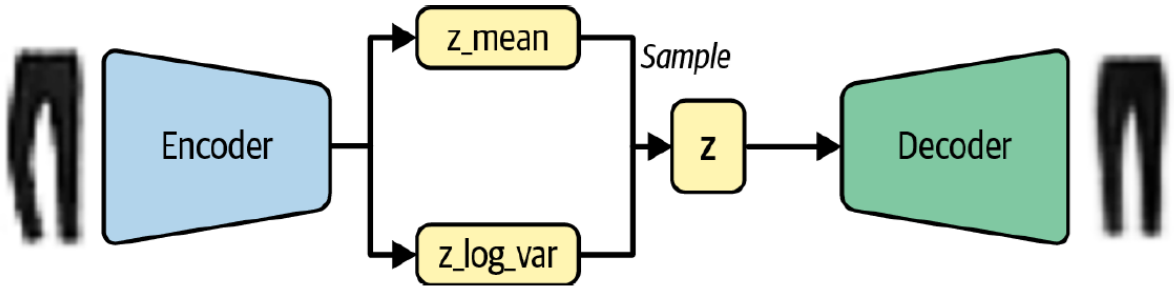


VAE: Reconstrucciones más borrosas, porque el modelo equilibra reconstrucción con regularización probabilística.



VQ-VAE: Reconstrucciones de calidad, comparables a las del AE, ya que evita el "blur" del VAE (usa latentes discretos en lugar de distribuciones).

Limitaciones del paradigma basado en reconstrucción



- Autoencodificadores deterministas , VAEs y VQ-VAEs aprenden mediante **reconstrucción supervisada**.
- La medida de éxito depende de la distancia entre la salida y la entrada.
- Esto limita el realismo visual: las imágenes tienden a verse borrosas o promedio.

Necesitamos una forma de **enseñar al modelo qué "parece real"** sin usar el error de píxeles.

GANs

Clase 12

Dra. Wendy Aguilar

Modelos Generativos Profundos

UN ENFOQUE DESDE LA CREATIVIDAD COMPUTACIONAL

Nueva perspectiva: aprendizaje adversarial

De reconstruir entradas conocidas → a engañar.

2014 **Generative Adversarial Nets**

NeurIPS (Neural Information Processing Systems)

Ian J. Goodfellow[*], **Jean Pouget-Abadie**[†], **Mehdi Mirza**, **Bing Xu**, **David Warde-Farley**, **Sherjil Ozair**[‡], **Aaron Courville**, **Yoshua Bengio**[§]
Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC H3C 3J7

Abstract

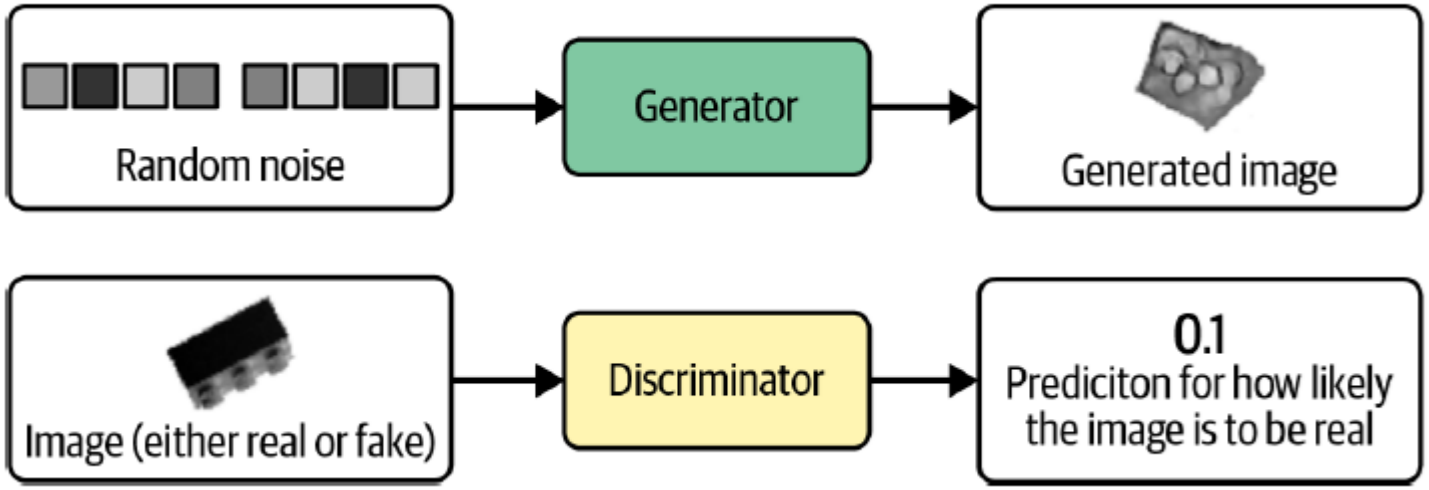
We propose a new framework for estimating generative models via an adversarial process, in which we simultaneously train two models: a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather than G. The training procedure for G is to maximize the probability of D making a mistake. This framework corresponds to a minimax two-player game. In the space of arbitrary functions G and D, a unique solution exists, with G recovering the training data distribution and D equal to $\frac{1}{2}$ everywhere. In the case where G and D are defined by multilayer perceptrons, the entire system can be trained with backpropagation. There is no need for any Markov chains or unrolled approximate inference networks during either training or generation of samples. Experiments demonstrate the potential of the framework through qualitative and quantitative evaluation of the generated samples.

GANs

Una GAN es una batalla entre dos adversarios: el generador y el discriminador.

El generador intenta convertir ruido aleatorio en observaciones que parezca que fueron muestreadas del conjunto de datos original.

El discriminador intenta predecir si una observación proviene del conjunto de datos original o si es uno creado por el generador,



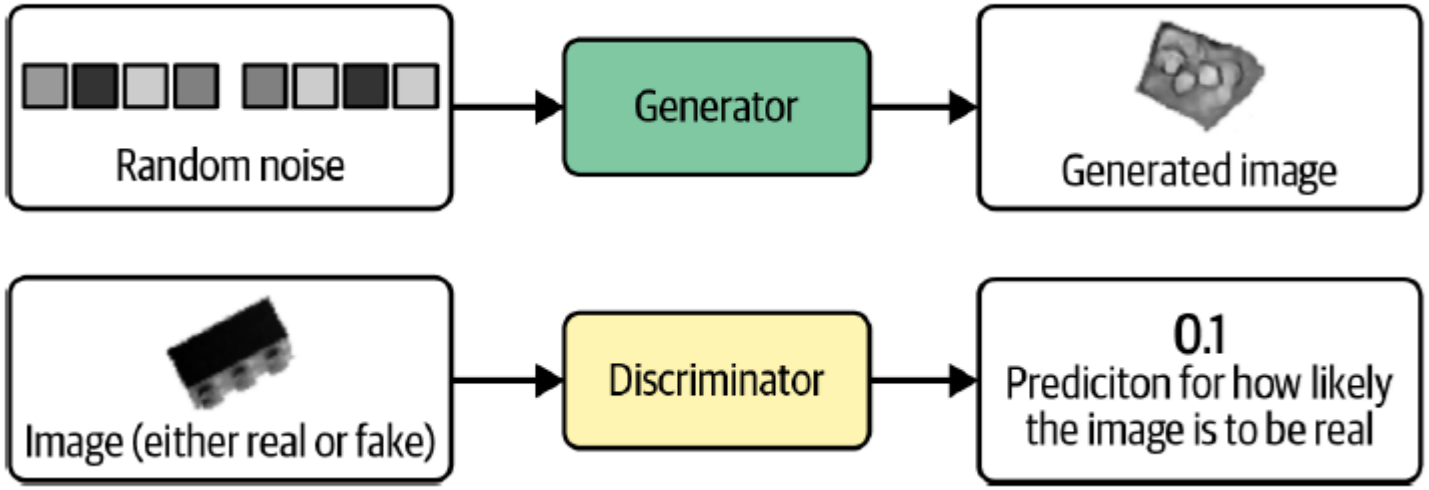
Entradas y salidas de las dos redes: el generador y el discriminador

GANs

Una GAN es una batalla entre dos adversarios: el generador y el discriminador.

Al comienzo del proceso, el generador produce imágenes ruidosas.

Al comienzo del proceso el discriminador predice aleatoriamente.

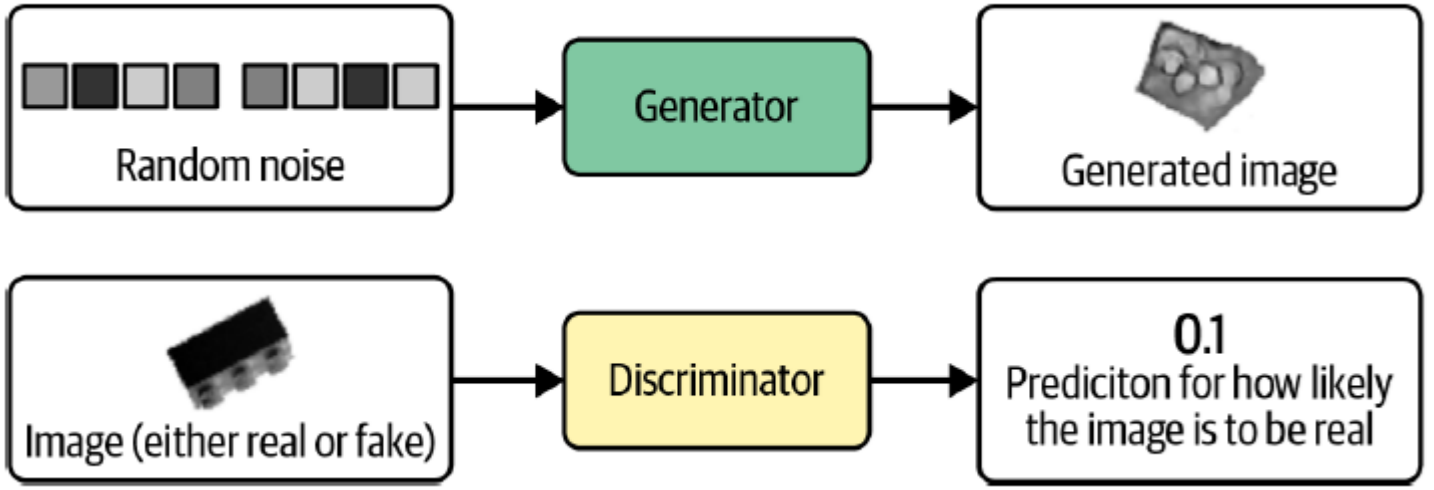


Entradas y salidas de las dos redes: el generador y el discriminador

GANs

La clave de las GANs radica en cómo alternamos el entrenamiento de las dos redes, de tal forma que:

- a medida que el generador se vuelve más hábil para engañar al discriminador, este último debe adaptarse para mantener su capacidad de identificar correctamente qué observaciones son falsas.
- Esto impulsa al generador a encontrar nuevas formas de engañar al discriminador, y así el ciclo continúa.



GANs

Entrenamiento en forma de **juego minimax**:

El entrenamiento de una GAN se plantea como un **juego de suma cero** entre dos redes neuronales:

situación en la que **lo que uno gana, el otro lo pierde** en la misma proporción.

$$\text{Ganancia de G} + \text{Ganancia de D} = 0$$

Si el Discriminador (D) mejora su capacidad para distinguir falsos → el Generador (G) "pierde".

Si el Generador (G) logra engañar a D → entonces D "pierde".

El Generador G **busca minimizar** esa misma función, es decir, hacer que D se equivoque.

El Discriminador D **busca maximizar** la probabilidad de clasificar correctamente:
·reales como reales,
·falsas como falsas.

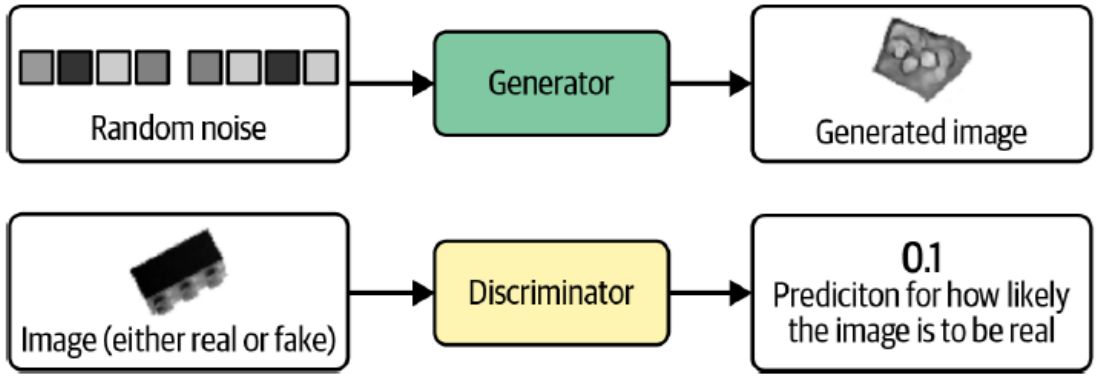
$$\min_G \max_D V(D, G)$$

GANs

Necesitamos una forma de **enseñar al modelo qué "parece real"** sin usar el error de píxeles.

¿Qué hace a una imagen parecer real?"

En las GANs, la respuesta a esta pregunta no está codificada en una función matemática...
sino **aprendida por el discriminador a través de la competencia**.



GANs

- La GAN original (Goodfellow et al., 2014) introdujo la idea revolucionaria del juego minimax adversarial entre Generador y Discriminador.

- Sin embargo, su implementación práctica era inestable:
  - Gradientes explosivos o que se desvanecen.
  - Colapso de modo (mode collapse).

    El generador produce siempre imágenes muy similares, perdiendo variedad.

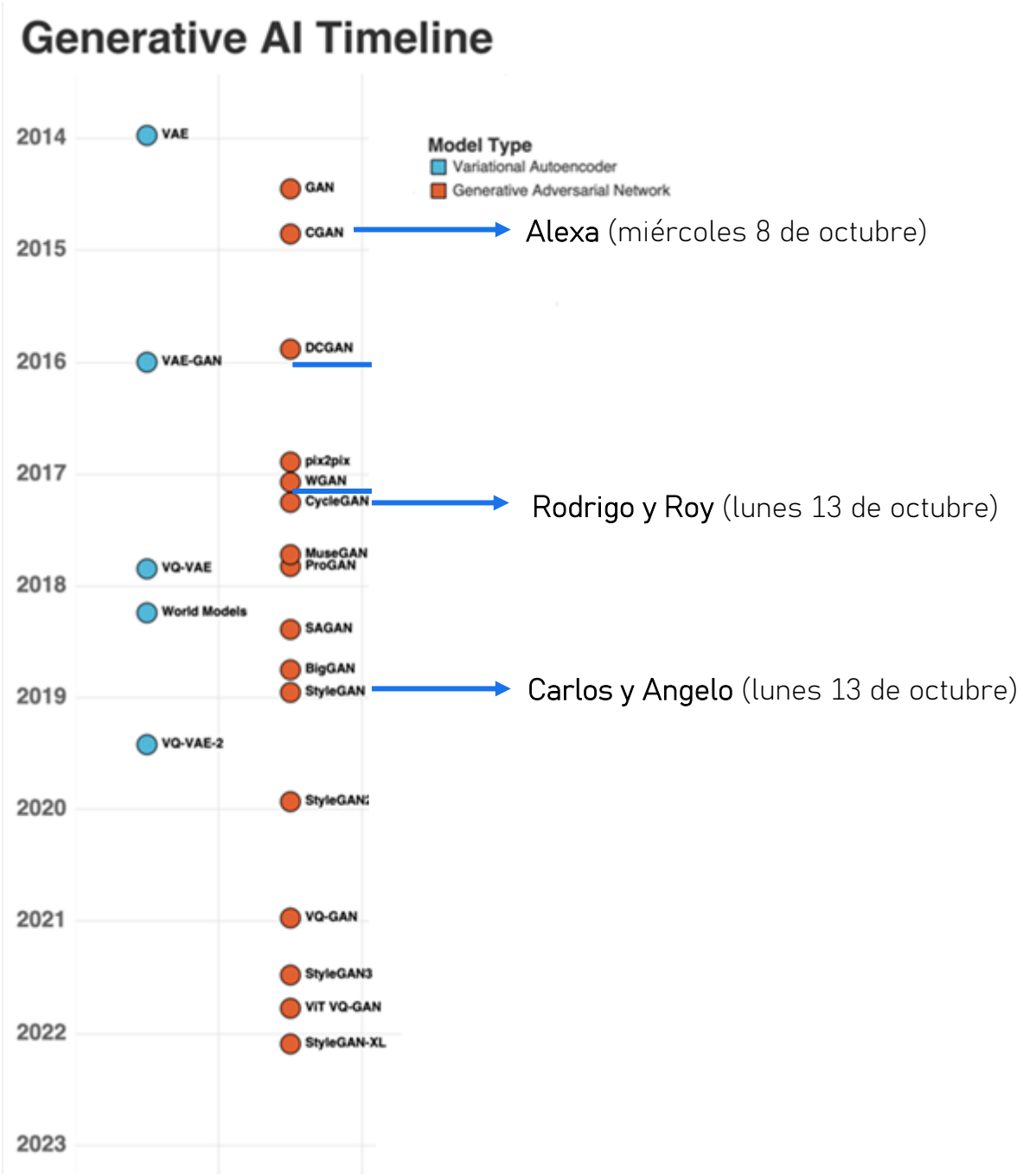
    El modelo se estanca generando un único patrón que engaña al discriminador.

- Entrenamiento altamente inestable → difícil mantener el equilibrio G–D.
- Aun así, demostraron que la competencia podía generar muestras plausibles.

MNIST

TFD (Toronto Face Database)

CIFAR-10


Generative AI Timeline
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
VAE
GAN
CGAN
DCGAN
VAE-GAN
pix2pix
WGAN
CycleGAN
MuseGAN
ProGAN
VQ-VAE
World Models
SAGAN
BigGAN
StyleGAN
VQ-VAE-2
StyleGAN2
VQ-GAN
StyleGAN3
ViT VQ-GAN
StyleGAN-XL
Model Type
Variational Autoencoder
Generative Adversarial Network
Alexa (miércoles 8 de octubre)
Rodrigo y Roy (lunes 13 de octubre)
Carlos y Angelo (lunes 13 de octubre)

Línea de tiempo de IA Generativa

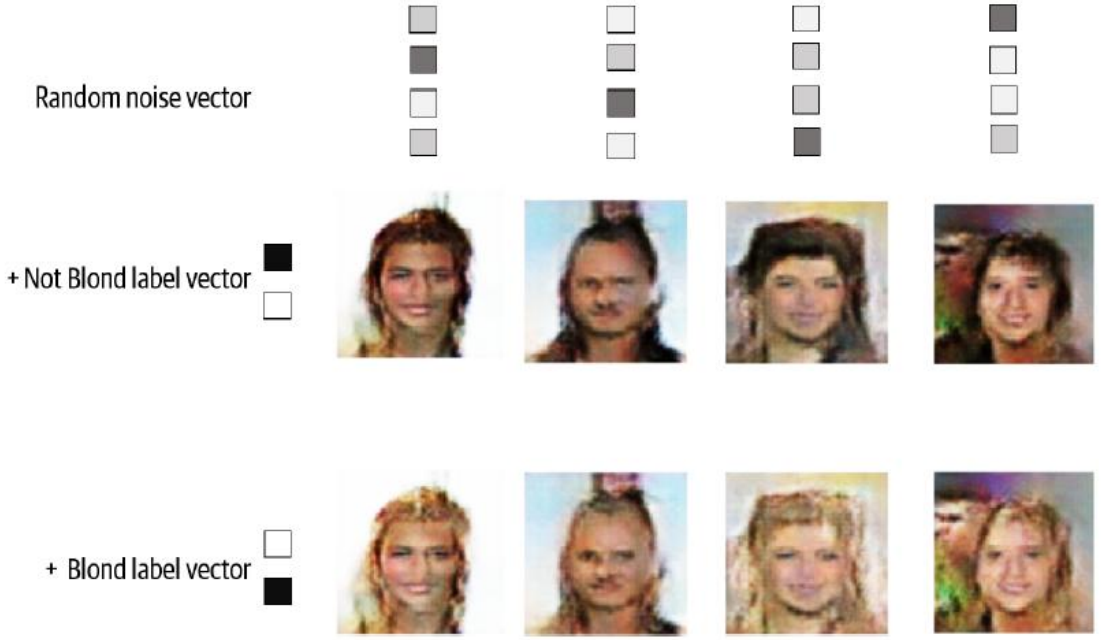
1. La era de las VAEs y GANs (2013-2017)

CGAN (Conditional GAN)

Extensión de la GAN original para permitir el control explícito sobre la salida del generador mediante condiciones, como etiquetas de clase. Esto marca el inicio del enfoque condicional en modelos generativos, donde se puede guiar la generación con información adicional.

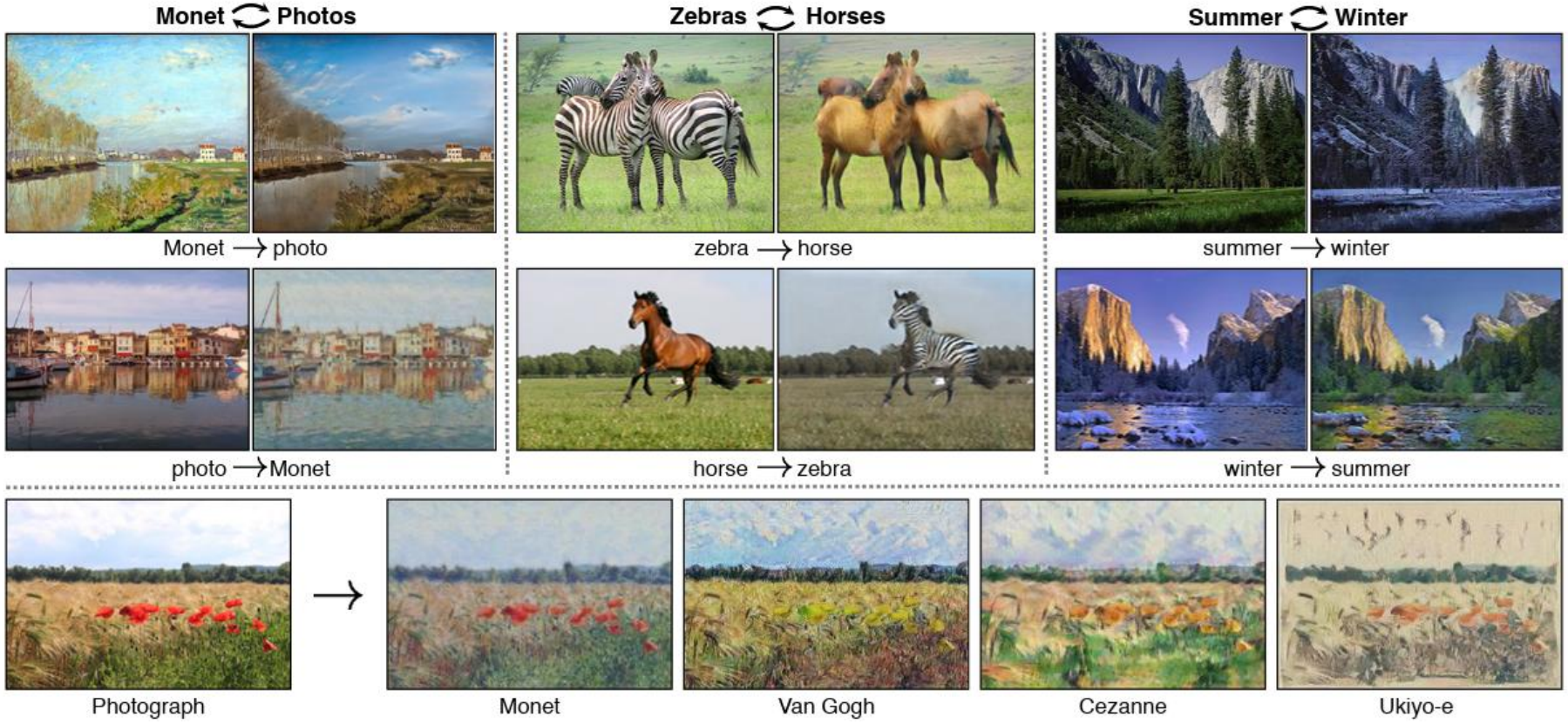
2014

Mirza, M., & Osindero, S. (2014). *Conditional generative adversarial nets*. arXiv preprint arXiv:1411.1784.

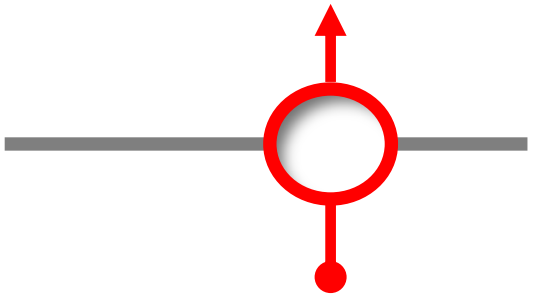


Línea de tiempo de IA Generativa

1. La era de las VAEs y GANs (2013-2017)



2017



CycleGAN

Enfoque innovador de traducción de imágenes de un dominio a otro *sin necesidad de pares de imágenes alineadas*. Introduce la **consistencia cíclica** como restricción para garantizar que una imagen traducida pueda revertirse a su forma original, lo que permitió realizar tareas como transformar caballos en cebras o paisajes de verano en paisajes de invierno con notable calidad visual.

Zhu, J. Y., Park, T., Isola, P., & Efros, A. A. (2017). *Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks*. In Proceedings of the IEEE International Conference on Computer Vision (ICCV), 2223–2232.

Línea de tiempo de IA Generativa

1. La era de las VAEs y GANs (2013-2017)

StyleGAN

Introduce un nuevo enfoque en la arquitectura del generador al incorporar un "mecanismo de estilo" que permite un control más preciso y jerárquico sobre atributos visuales en la imagen generada. Este modelo logra un nivel sin precedentes de realismo y control sobre rasgos faciales y estructuras globales, marcando un hito en la generación de rostros sintéticos.

2019

·A partir de una sola imagen de una persona, puedes generar **versiones más jóvenes**, **con barba**, **con gafas** o **con diferente iluminación** sin necesidad de volver a entrenar la red.
·Retocar el **cabello**, el **maquillaje** o la **expresión facial** de una persona real conservando la coherencia visual.
·Puedes **mezclar estilos** de distintas imágenes: por ejemplo, la pose de una, la textura de piel de otra, y el peinado de una tercera.

Karras, T., Laine, S., & Aila, T. (2019). A Style-Based Generator Architecture for Generative Adversarial Networks. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 2019), 4401–4410.
https://arxiv.org/abs/1812.04948