

Modelos Generativos Profundos

UN ENFOQUE DESDE LA
CREATIVIDAD
COMPUTACIONAL

Clase 14

Dra. Wendy Aguilar

GANs III



Deep Convolutional GAN (DCGAN)

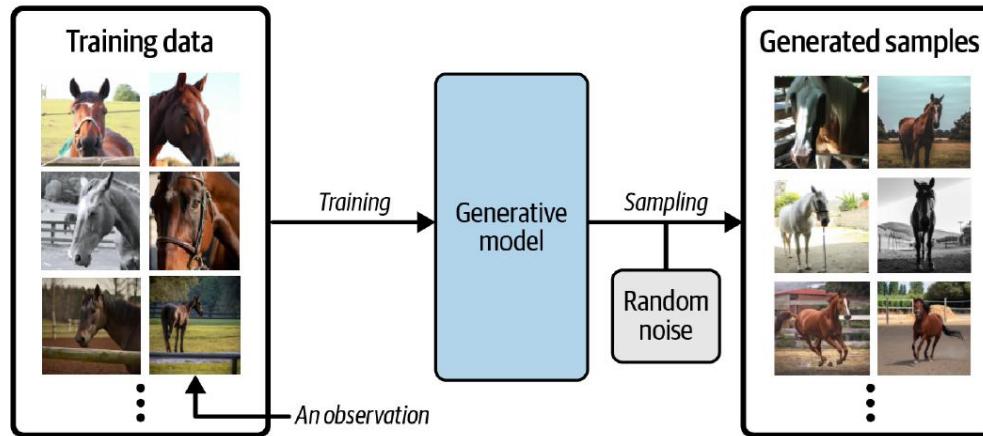
Problemas persistentes en las DCGAN

- **Inestabilidad del entrenamiento:**
Las pérdidas del generador y discriminador no reflejan el progreso real (no correlacionan con la calidad visual).
- **Modo colapso:**
El generador aprende a producir solo un subconjunto de imágenes plausibles.
- **Oscilaciones en el equilibrio adversario:**
El juego min-max puede no converger, especialmente con redes profundas.
- **Discriminador saturado:**
Si el discriminador se vuelve demasiado bueno, el gradiente del generador desaparece.

Necesitamos una métrica que mida *cuánto difieren las distribuciones*, no solo si el discriminador puede distinguirlas.

Taxonomía de modelos generativos

La meta de todos los modelos generativos es modelar la función de densidad $p_\theta(x)$ tal que podamos **muestrear ejemplos plausibles** $x \sim p_\theta(x)$.



Teóricamente:

$$p_\theta(x) = \int p_\theta(x, z) dz = \int p_\theta(x|z)p(z) dz$$

donde:

- z son variables latentes (causas ocultas).
- $p(z)$ es un prior (típicamente gaussiano).
- $p_\theta(x|z)$ es el modelo generativo

Problema:

El integral sobre z es **intractable** (no se puede resolver analíticamente), porque $p_\theta(x|z)$ y la dimensión de z son **complejas**.

No tienen una forma **analíticamente simple** (como una gaussiana o una función lineal).

Dependen de **redes neuronales profundas** con millones de parámetros no lineales.

Su cálculo involucra **altas dimensiones** (espacios latentes de 64, 128, 256... dimensiones).

Taxonomía de modelos generativos

La meta de todos los modelos generativos es modelar la función de densidad $p_\theta(x)$ tal que podamos muestrear ejemplos plausibles $x \sim p_\theta(x)$.

De manera general podemos encontrar **tres aproximaciones**:

1. Modelos explícitos (computables):

Modelan explícitamente la función de densidad $p_\theta(x)$, pero restringen el modelo de alguna manera, para que la función de densidad sea tratable (es decir, que pueda ser calculada).

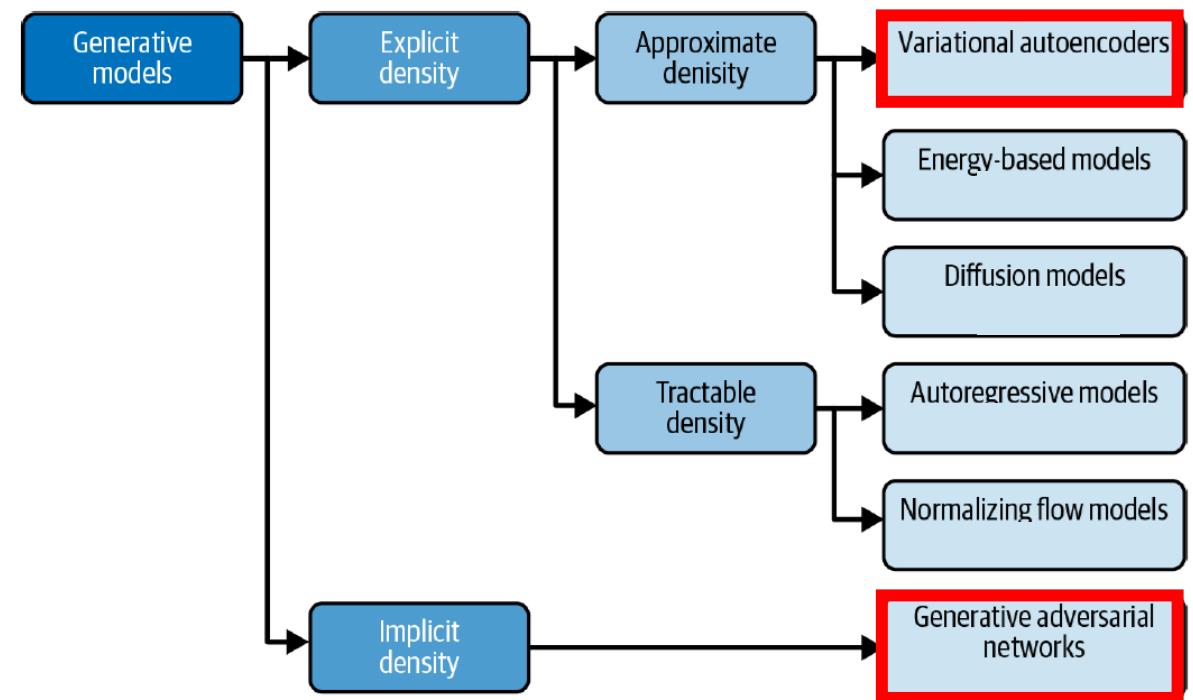
2. Modelos explícitos (aproximados):

Modelan explícitamente una aproximación tratable de la función de densidad $p_\theta(x)$.

3. Modelos implícitos:

Modelan implícitamente la función de densidad, a través de un proceso estocástico que genera datos directamente.

No modelan la densidad, sino un proceso generativo que produce muestras de $p_\theta(x)$ sin definirla explícitamente.



¿Por qué los VAEs son modelos explícitos (computables)?

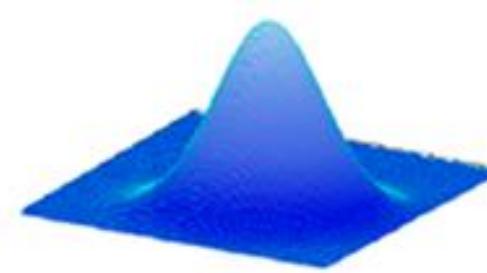
$$p_{\theta}(x) = \int p_{\theta}(x, z) dz = \int p_{\theta}(x|z)p(z) dz$$

Modelos generativos latentes

$$p(x, z) = p(x|z) p(z)$$

Prior: $p(z) = \mathcal{N}(\mathbf{0}, I)$

Un **prior** (o distribución a priori) es la **suposición inicial** que hacemos sobre cómo se distribuyen las variables latentes z , antes de ver los datos.



Likelihood (decoder): $p_{\theta}(x|z)$

La probabilidad de que la imagen del 5 corresponda al código latente del 5.

Parametrizada por una red neuronal que genera datos a partir de z .

θ son los **pesos de todas las capas** que definen el **decodificador** (densas, convolucionales, etc.).

Posterior aproximado (encoder):

$$q_{\phi}(z|x) \approx p(z|x)$$

Ya que el posterior real es intratable.

- Introducimos una distribución alternativa q_{ϕ} , donde ϕ son los parámetros del codificador.
- Elegimos una familia simple de distribuciones (ej. Normal multivariada estándar).

$$p(z|x) = \frac{p_{\theta}(x|z)p(z)}{p(x)}.$$

$$p(x) = \int p_{\theta}(x|z)p(z) dz$$

Es intratable porque esa integral sobre todos los posibles z no se puede resolver de manera exacta en espacios de alta dimensión.

¿Por qué los VAEs son modelos explícitos (computables)?

$$p_{\theta}(x) = \int p_{\theta}(x, z) dz = \int p_{\theta}(x|z)p(z) dz$$

Modelos generativos latentes

$$p(x, z) = p(x|z) p(z)$$

Así se forma un modelo **explícito y probabilístico completo** con dos componentes:

- Generativo (decodificador): $p_{\theta}(x|z)$
- Inferencial (codificador): $q_{\phi}(z|x)$

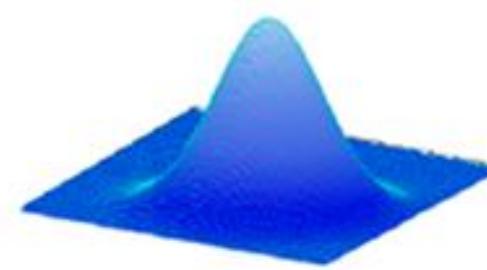
En conjunto, el VAE modela **explícitamente** las distribuciones:

$$p_{\theta}(x, z) = p_{\theta}(x|z)p(z)$$
$$q_{\phi}(z|x)$$

y busca que $q_{\phi}(z|x) \approx p_{\theta}(z|x)$.

Prior: $p(z) = \mathcal{N}(0, I)$

Un **prior** (o distribución a priori) es la **suposición inicial** que hacemos sobre cómo se distribuyen las variables latentes z , antes de ver los datos.



Likelihood (decoder): $p_{\theta}(x|z)$

La probabilidad de que la imagen del 5 corresponda al código latente del 5.

Parametrizada por una red neuronal que genera datos a partir de z .

θ son los **pesos de todas las capas** que definen el **decodificador** (densas, convolucionales, etc.).

$$p(z|x) = \frac{p_{\theta}(x|z)p(z)}{p(x)}.$$

$$p(x) = \int p_{\theta}(x|z)p(z) dz$$

Posterior aproximado (encoder):

$$q_{\phi}(z|x) \approx p(z|x)$$

Ya que el posterior real es intratable.

- Introducimos una distribución alternativa q_{ϕ} , donde ϕ son los parámetros del codificador.
- Elegimos una familia simple de distribuciones (ej. Normal multivariada estándar).

Es intratable porque esa integral sobre todos los posibles z no se puede resolver de manera exacta en espacios de alta dimensión.

¿Por qué los VAEs son modelos explícitos (computables)?

Maximizar la probabilidad de los datos $p_{\theta}(x)$

Buscamos ajustar los parámetros θ para que las imágenes generadas parezcan provenir del mismo proceso estadístico que las del conjunto de entrenamiento.

El objetivo real sería maximizar la probabilidad de los datos: $\log p_{\theta}(x) = \log \int p_{\theta}(x, z) dz = \log \int p_{\theta}(x|z) p(z) dz$

Esta es una **definición explícita** porque el modelo **declara** cómo se calcula $p_{\theta}(x)$: a través de una integral sobre el espacio latente.

... pero como la integral es intratable necesitamos una forma de optimizarla indirectamente.

Necesitamos una función alternativa y computable que se relacione con $\log p_{\theta}(x)$ de manera que maximizarla también aumente la verdadera probabilidad de los datos.

La función que sí podemos optimizar

$$\text{ELBO}(\theta, \phi) = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}(q_\phi(z|x) \| p(z))$$

Maximizamos la ELBO en lugar de $\log p_\theta(x)$

Maximizarla equivale a minimizar \mathcal{L}_{VAE}

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

Mide qué tan bien el modelo reconstruye los datos (verosimilitud o likelihood).

Regula qué tan parecida es la distribución latente aprendida al prior (normal estándar).

En la práctica hicimos:

binary cross-entropy

BCE(x, \hat{x})

$$D_{KL}[N(\mu, \sigma^2) \| N(0, 1)] = -\frac{1}{2} \sum (1 + \log(\sigma^2) - \mu^2 - \sigma^2)$$

¿Por qué los VAEs son modelos explícitos (computables)?



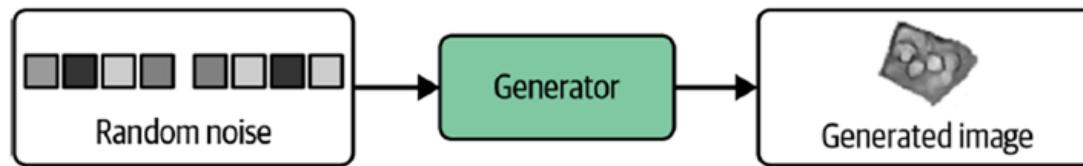
VAE (explícito):

Define $p_{\theta}(x | z)$ y $p(z)$ \Rightarrow existe $p_{\theta}(x)$; como es intratable, maximizamos ELBO (neg-ELBO = pérdida).

¿Por qué las GANs son modelos implícitos?

- No hay *likelihood* $p_\theta(x | z)$ y por lo tanto tampoco se puede calcular $p_\theta(x) = \int p_\theta(x | z)p(z)dz$

Decodificador de VAE



Fijas un prior $p(z) \mathcal{N}(0, I)$

$p(z)$ es la distribución fija
del ruido de entrada

No forma parte de una densidad
explícita sobre x como en los VAE

Ese prior solo dice **desde dónde**
muestreas dentro de \mathbb{R}^{d_z} y con qué
densidad.

El generador
aprende una
transformación

$$G_\theta: z \mapsto x.$$

Nunca define una probabilidad explícita $p_g(x)$.

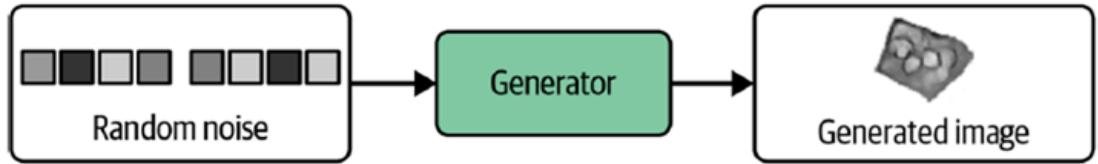
No tenemos una fórmula para $p_g(x)$, ni una
forma de evaluar su log-likelihood.

Por eso se dice que las GAN son
modelos implícitos (*implicit
generative models*).

¿Por qué las GANs son modelos implícitos?

Espacio latente: espacio matemático $\mathcal{Z} \subseteq \mathbb{R}^n$ desde el cual el generador toma vectores z para producir ejemplos sintéticos $x_{fake} = G(z)$.

- No proviene de los datos
 - No “emerge” de una inferencia probabilística como en un VAE
- No lo aprende el discriminador
- Se define arbitrariamente al diseñar la red (lo defines tú):



1. **Dominio elegido por diseño**
 - Se fija una dimensión d_z y un dominio vectorial, típicamente \mathbb{R}^{d_z}
 - Ese dominio es la **entrada del generador** $G_\theta : \mathcal{Z} \rightarrow \mathcal{X}$.
2. **Prior de muestreo $p(z)$**
 - Se escoge un prior simple para **muestrear**: $z \sim \mathcal{N}(0, I)$ o Uniforme $[-1, 1]$.
 - $p(z)$ no define una likelihood $p_\theta(x|z)$; solo determina **cómo exploras** \mathcal{Z}
3. **Semántica inducida por el generador**
 - La “geometría útil” del latente (direcciones para pose, color, estilo...) **la aprende** G durante el entrenamiento adversarial.

Así, el espacio latente es simplemente el **espacio n-dimensional** donde cada punto z es una combinación de variables aleatorias independientes.

¿Por qué las GANs son modelos implícitos?

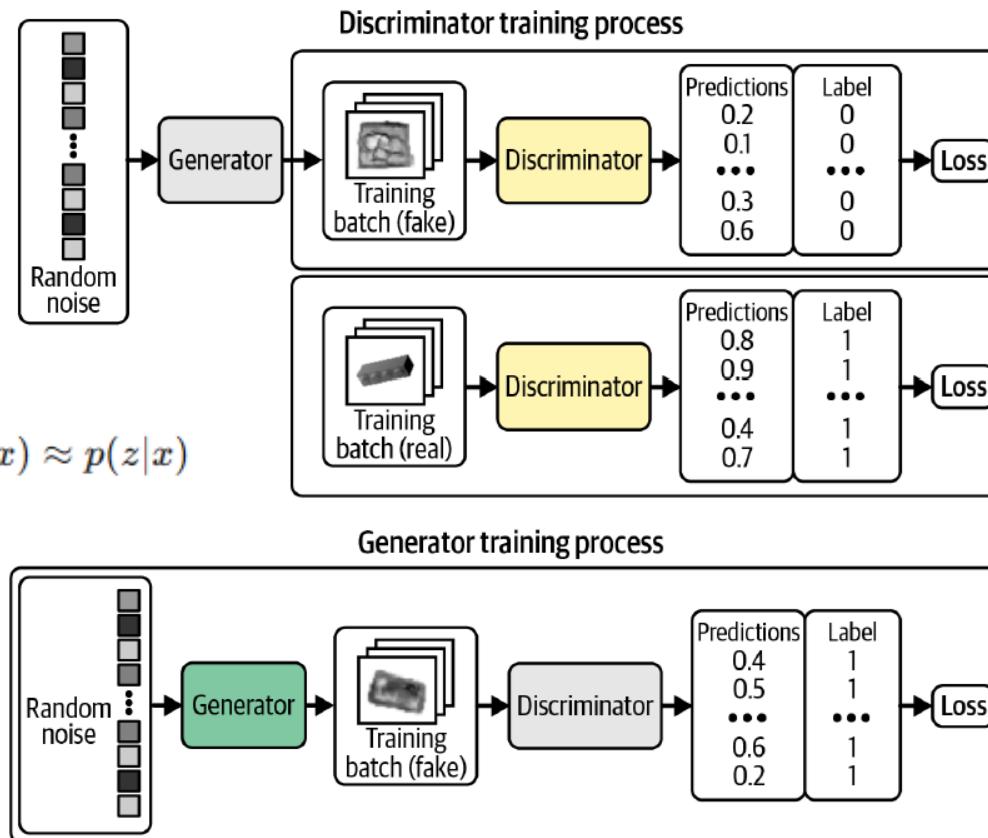
- Tampoco hay posterior $p_\theta(z | x)$ ni una cota tipo ELBO que podamos optimizar.

$$q_\phi(z|x) \approx p(z|x)$$

Se entrena usando un discriminador, que logre detectar las imágenes reales de las falsas.



En las GANs estándar no podemos usar un término KL como en VAE porque en GAN no tenemos likelihood/posterior explícito $p_\theta(x|z)$ $q_\phi(z|x) \approx p(z|x)$ para calcular $D_{KL}(q_\phi(z|x) || p(z))$.



* **Estándar:** Marco original de Goodfellow (2014) y sus variantes directas (como DCGAN) que mantienen la misma lógica de entrenamiento y la misma pérdida logística.

¿Por qué las GANs son modelos implícitos?

- Tampoco hay posterior $p_\theta(z | x)$ ni una cota tipo ELBO que podamos optimizar.

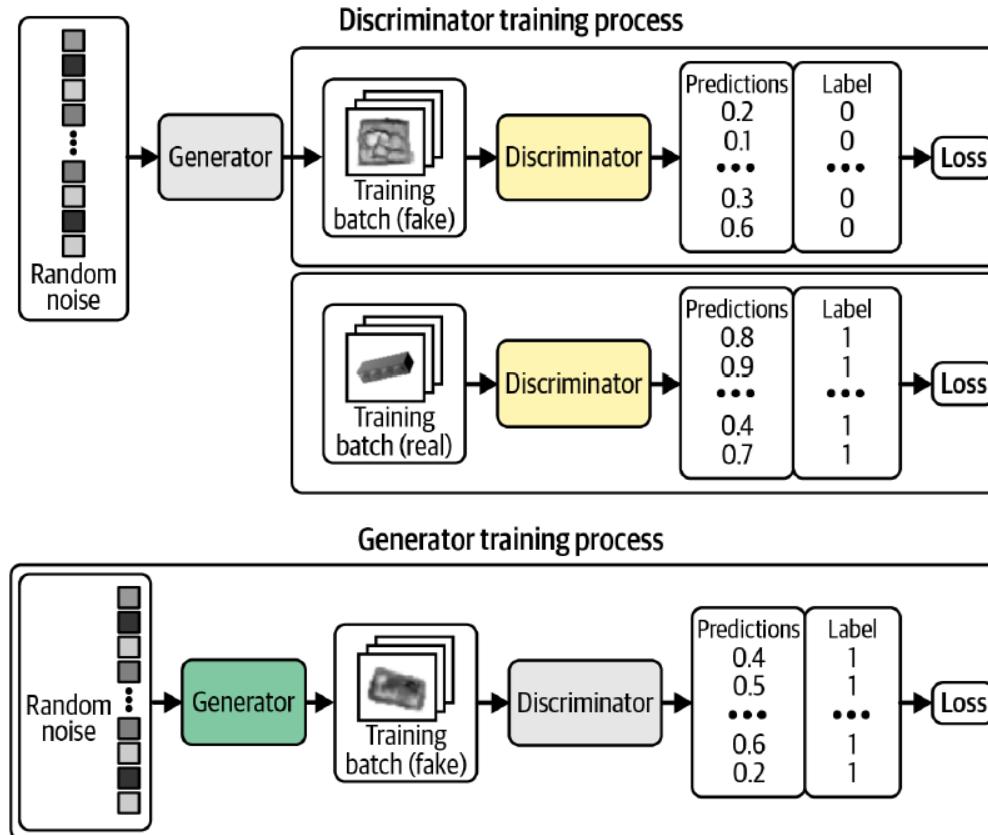
$$q_\phi(z|x) \approx p(z|x)$$

Si queremos gradientes estables, mejor correlación pérdida↔calidad y menos mode collapse ...

... no basta con saber si el discriminador distingue; necesitamos una **puntuación continua** de “qué tan real” parece una muestra.

Esa señal es más **suave y estable**, lo que ayuda a que el generador mejore sin atascarse.

Necesitamos una **métrica alternativa** entre distribuciones.



Wasserstein GAN con Penalización de Gradiente

WGAN-GP

- Fue uno de los primeros pasos hacia estabilizar el entrenamiento de las GAN.
- Con pequeños cambios, los autores lograron mostrar cómo entrenar GANs que tengan las siguientes dos propiedades:

1. Una métrica de pérdida significativa que se correlaciona de manera continua con la calidad de las muestras:

- Cuando el generador mejora, la pérdida disminuye.
- Si el generador se estanca, la pérdida deja de cambiar.

2. Estabilidad mejorada del proceso de optimización

Wasserstein GAN
2017
Martin Arjovsky¹, Soumith Chintala², and Léon Bottou^{1,2}

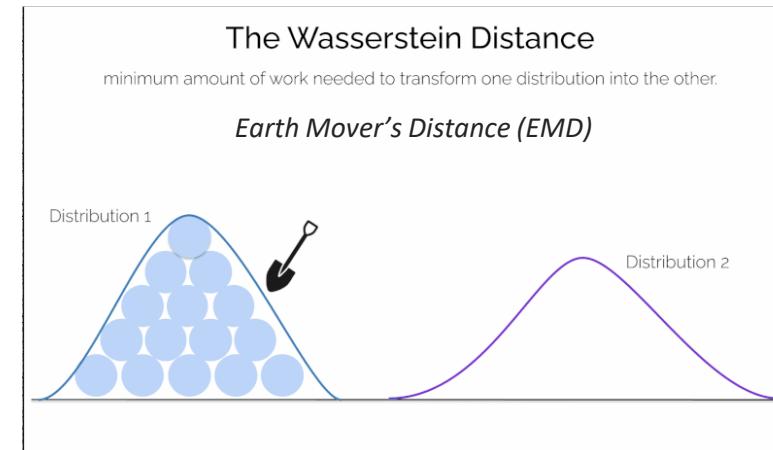
¹Courant Institute of Mathematical Sciences

²Facebook AI Research

La WGAN busca minimizar la **distanza de Wasserstein entre distribuciones**:

$$W(p_{data}, p_g)$$

Mide "cuánto trabajo cuesta" transformar una distribución en otra:



Pero esta distancia no se puede calcular directamente (porque no tenemos expresiones analíticas para p_{data} ni p_g)

Wasserstein GAN con Penalización de Gradiente

Binary Cross Entropy

$$\text{BCE} = -\frac{1}{n} \sum_{i=1}^n \left[y_i \log p_i + (1 - y_i) \log(1 - p_i) \right]$$

$y_i \in \{0, 1\}$ es la etiqueta; $p_i \in (0, 1)$ es la probabilidad predicha.

Se usa tanto en el discriminador como en el generador, pero con objetivos distintos.

Discriminador

$$\mathcal{L}_D = \underbrace{\text{BCE}(D(x_{\text{real}}), 1)}_{\text{Pérdida de BCE para las muestras reales con etiqueta objetivo 1.}} + \underbrace{\text{BCE}(D(G(z)), 0)}_{\text{Pérdida BCE aplicada a las muestras falsas que salen del generador, con etiqueta objetivo 0.}}$$

Pérdida de BCE para las muestras reales con etiqueta objetivo 1.

Pérdida BCE aplicada a las muestras falsas que salen del generador, con etiqueta objetivo 0.

Generador

$$\mathcal{L}_G = \text{BCE}(D(G(z)), 1)$$

Tratamos sus imágenes falsas $G(z)$ como si fueran **etiqueta 1 (reales)** para el discriminador.
→ Forzamos a que $D(G(z)) \rightarrow 1$.

WGAN-GP

- Se usan $y_i = 1$ y $y_i = -1$ como etiquetas, en lugar de 1 y 0.
- En lugar de "clasificar real/falso con BCE", convertimos al discriminador en un crítico que de como salida una **puntuación/calificación** en vez de una probabilidad.
- Sustituimos la función de activación sigmoide por la lineal en la última capa del discriminador, de tal forma que las predicciones p_i ya no estén restringidas a caer en el rango $[0, 1]$ sino que ahora puede ser cualquier número en el rango $(-\infty, \infty)$.
- Las **puntuaciones del crítico** se usan para **aproximar la distancia de Wasserstein** (usada como la pérdida que se optimiza),

Wasserstein GAN con Penalización de Gradiente

Aproximar la distancia de Wasserstein usando una función D (el crítico)

$$W(p_{data}, p_g) = \max_{D \in 1\text{-Lipschitz}} \underbrace{\mathbb{E}_{x \sim p_{data}} [D(x)] - \mathbb{E}_{x \sim p_g} [D(x)]}_{\text{Diferencia de expectativas}}$$

Diagrama que ilustra la fórmula:

- Distribución real de los datos**: Representado por un cuadro azul.
- Distribución generada por el modelo**: Representado por un cuadro azul.
- Crítico**: Representado por un cuadro azul.
- Expectativa del crítico sobre los datos reales**: Es decir, el promedio de los valores que D asigna a muestras reales.
- Expectativa del crítico sobre las muestras falsas (generadas)**.
- Diferencia de expectativas**: Entre las dos distribuciones: cuánto más altos son los valores del crítico en los datos reales respecto a los generados.
- Este valor es mayor cuando las distribuciones están más separadas y se acerca a 0 cuando el generador logra igualar a p_{data} .**

Wasserstein GAN con Penalización de Gradiente

Aproximar la distancia de Wasserstein usando una función D (el *crítico*)

$$W(p_{data}, p_g) = \max_{D \in 1\text{-Lipschitz}} \underbrace{\mathbb{E}_{x \sim p_{data}}[D(x)] - \mathbb{E}_{x \sim p_g}[D(x)]}$$

Se busca el **mejor crítico posible**, pero restringido a ser **1-Lipschitz**, es decir:

$$|D(x_1) - D(x_2)| \leq \|x_1 - x_2\|$$

Para cualquier par de imágenes de entrada x_1 y x_2 .

$$\|x_1 - x_2\|$$

Es la **distancia entre los dos puntos en el espacio de datos** (usualmente norma L2 o euclídea).

Los cambios en la salida de D no pueden ser más grandes que los cambios en la entrada.

- El crítico no puede tener "pendientes" arbitrariamente grandes;
- Su superficie debe variar suavemente, sin saltos bruscos ni picos pronunciados.

Wasserstein GAN con Penalización de Gradiente

Aproximar la distancia de Wasserstein usando una función D (el *crítico*)

$$W(p_{data}, p_g) = \max_{D \in 1\text{-Lipschitz}} \mathbb{E}_{x \sim p_{data}} [D(x)] - \mathbb{E}_{x \sim p_g} [D(x)]$$

Crítico D

Maximizar la diferencia de expectativas
(respetando 1-Lipschitz)

$$\max_D \mathbb{E}_{x \sim p_{data}} [D(x)] - \mathbb{E}_{x \sim p_g} [D(x)]$$

Generador G

Minimizar la distancia Wasserstein estimada

$$\min_G -\mathbb{E}_{x \sim p_g} [D(x)]$$

- El crítico D aprende una función que mide "cuánto trabajo" cuesta mover masa desde p_g hasta p_{data} .
- El valor máximo de esa diferencia de expectativas (bajo la restricción Lipschitz) es precisamente la distancia Wasserstein-1.
- Luego, el generador intenta minimizar esa distancia ajustando sus parámetros para que el crítico no pueda distinguir las muestras falsas de las reales.

Wasserstein GAN con Penalización de Gradiente

En la práctica

Función de Pérdida del crítico

Estimación por minibatch

$$\mathcal{L}_D \approx \frac{1}{m} \sum_{i=1}^m D(G(z_i)) - \frac{1}{m} \sum_{i=1}^m D(x_i)$$

Tamaño del minibatch

Qué tan altos en promedio puntúa el crítico a los falsos (queremos bajarlo).
Qué tan altos en promedio puntúa a los reales (queremos subirlo).

El crítico intenta **bajar** la media de sus puntuaciones en *fakes* (y subir la de *reales*),

Función de Pérdida del generador

Permite implementarlo como un problema estándar de minimización

$$\mathcal{L}_G = - \frac{1}{m} \sum_{i=1}^m D(G(z_i))$$

Qué tan altos en promedio puntúa el crítico a los falsos (queremos bajarlo)

El generador intenta **subir** la puntuación de sus *fakes* para "parecer más real".

Wasserstein GAN con Penalización de Gradiente

Imposición de la restricción de Lipschitz

- En el artículo original de WGAN, los autores muestran cómo es posible imponer la restricción de Lipschitz **recortando los pesos del crítico (weight clipping)** para que se mantengan dentro de un rango pequeño $[-0.01, 0.01]$, después de cada lote de entrenamiento.
 - Imagina que el crítico tiene un conjunto de pesos w
 - Despues de actualizarlo con *backpropagation*, se aplica la siguiente operación:
$$w_j \leftarrow \text{clip}(w_j, -c, c)$$
donde:
 - c es un valor pequeño (por ejemplo 0.01),
 - $\text{clip}(w_j, -c, c)$ significa:
 - si $w_j < -c$, se fija en $-c$,
 - si $w_j > c$, se fija en c ,
 - y si está dentro del rango, se deja igual.
- La capacidad del crítico para aprender se ve significativamente reducida.
- “El recorte de pesos es una forma claramente terrible de imponer una restricción de Lipschitz”.
- Un crítico sólido es fundamental para el éxito de un WGAN, ya que, sin gradientes precisos, el generador no puede aprender a ajustar sus pesos para producir mejores muestras.

Wasserstein GAN con Penalización de Gradiente

Imposición de la restricción de Lipschitz

A través de la pérdida por penalización del gradiente

Una función $D(x)$ es **1-Lipschitz** si la norma de su gradiente con respecto a la entrada nunca supera 1:

$$\|\nabla_x D(x)\|_2 = 1$$

En lugar de truncar los pesos, se añade un **término de penalización** en la función de pérdida del crítico que fuerza al gradiente a permanecer cercano a 1.

$$\mathcal{L}_D \approx \frac{1}{m} \sum_{i=1}^m D(G(z_i)) - \frac{1}{m} \sum_{i=1}^m D(x_i) + \lambda \frac{1}{m} \sum_{i=1}^m \left(\|\nabla_{\hat{x}_i} D(\hat{x}_i)\|_2 - 1 \right)^2$$

Hiperparámetro de penalización (típicamente 10)

Controla cuánto peso tiene el término de suavidad.

Puntos interpolados entre muestras reales y falsas:
Puntos donde se evalúa el gradiente.

Penalización de gradiente: mide cuánto se desvía la norma del gradiente respecto a 1.
Obliga al crítico a ser 1-Lipschitz.

$$\|\nabla_x D(x)\|_2 = \sqrt{\sum_j \left(\frac{\partial D(x)}{\partial x_j} \right)^2}$$

$\|\nabla D\|_2 > 1 \Rightarrow$ penaliza (pendiente demasiado empinada)

$\|\nabla D\|_2 < 1 \Rightarrow$ también penaliza (pendiente demasiado plana)

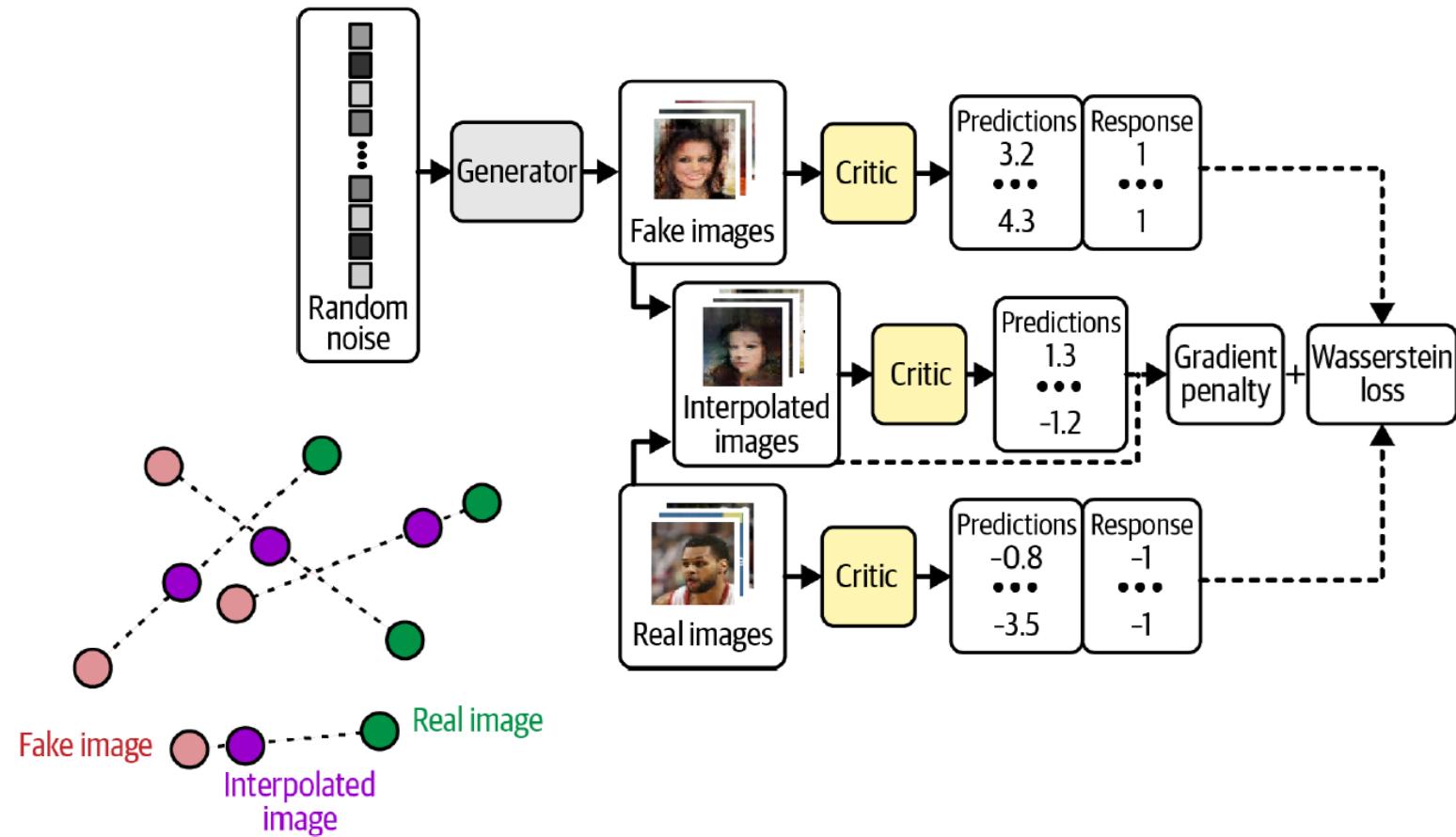
$\|\nabla D\|_2 \approx 1 \Rightarrow$ no hay penalización

Wasserstein GAN con Penalización de Gradiente

Imposición de la restricción de Lipschitz

Pérdida por penalización del gradiente

- Es inviable calcular este gradiente en todas partes durante el proceso de entrenamiento.
- En su lugar, el WGAN-GP evalúa el gradiente solo en algunos puntos.
- Para asegurar una mezcla equilibrada, utilizamos un conjunto de imágenes interpoladas que se encuentran en puntos seleccionados aleatoriamente a lo largo de las líneas que conectan el lote de imágenes reales con el lote de imágenes falsas de manera pareada.



Wasserstein GAN con Penalización de Gradiente

Imposición de la restricción de Lipschitz

Pérdida por penalización del gradiente

```
def gradient_penalty(self, batch_size, real_images, fake_images):
    alpha = tf.random.normal([batch_size, 1, 1, 1], 0.0, 1.0) ① → Cada imagen en el lote recibe un número aleatorio, entre 0 y 1, que se almacena como el vector alpha.
    diff = fake_images - real_images
    interpolated = real_images + alpha * diff ② → Se calcula un conjunto de imágenes interpoladas.

    with tf.GradientTape() as gp_tape:
        gp_tape.watch(interpolated)
        pred = self.critic(interpolated, training=True) ③ → Se le pide al crítico que califique cada una de estas imágenes interpoladas.

    grads = gp_tape.gradient(pred, [interpolated])[0] ④ → Se calcula el gradiente de las predicciones con respecto a las imágenes de entrada.
    norm = tf.sqrt(tf.reduce_sum(tf.square(grads), axis=[1, 2, 3])) ⑤ → Se calcula la norma L2 de este vector.
    gp = tf.reduce_mean((norm - 1.0) ** 2) ⑥ → La función devuelve la distancia cuadrada promedio entre la norma L2 y 1.

    return gp
```



wgan_gp.ipynb

Wasserstein GAN con Penalización de Gradiente

Entrenamiento de la WGAN-GP

- Un beneficio clave de usar la función de pérdida Wasserstein es que ya no necesitamos preocuparnos por equilibrar el entrenamiento del crítico y el generador.
- De hecho, al usar la pérdida Wasserstein, el crítico debe entrenarse hasta la convergencia antes de actualizar el generador, para asegurar que los gradientes para la actualización del generador sean precisos.
- Esto contrasta con un GAN estándar, donde es importante no dejar que el discriminador se vuelva demasiado fuerte.
- Por lo tanto, con los Wasserstein GANs, podemos simplemente entrenar al crítico varias veces entre las actualizaciones del generador, para asegurarnos de que esté cerca de la convergencia.
- Una relación típica utilizada es de tres a cinco actualizaciones del crítico por cada actualización del generador.

Wasserstein GAN con Penalización de Gradiente

Entrenamiento de la WGAN-GP

```
def train_step(self, real_images):
    batch_size = tf.shape(real_images)[0]

    for i in range(3): ❶ → Realiza tres actualizaciones del crítico.
        random_latent_vectors = tf.random.normal(
            shape=(batch_size, self.latent_dim)
        )

        with tf.GradientTape() as tape:
            fake_images = self.generator(
                random_latent_vectors, training = True
            )
            fake_predictions = self.critic(fake_images, training = True)
            real_predictions = self.critic(real_images, training = True)

            c_wass_loss = tf.reduce_mean(fake_predictions) - tf.reduce_mean(
                real_predictions
            ) ❷ → Calcula la pérdida de Wasserstein para el crítico: la diferencia entre la
                predicción promedio de las imágenes falsas y las imágenes reales.

            c_gp = self.gradient_penalty(
                batch_size, real_images, fake_images
            ) ❸ → Calcula el término de penalización del gradiente.

            c_loss = c_wass_loss + c_gp * self.gp_weight ❹ → La función de pérdida del crítico es una suma ponderada de la pérdida de
                Wasserstein y la penalización del gradiente.

            c_gradient = tape.gradient(c_loss, self.critic.trainable_variables)

            self.c_optimizer.apply_gradients(
                zip(c_gradient, self.critic.trainable_variables)
            ) ❺ → Actualiza los pesos del crítico.
```

Wasserstein GAN con Penalización de Gradiente

Entrenamiento de la WGAN-GP

```
random_latent_vectors = tf.random.normal(  
    shape=(batch_size, self.latent_dim))  
)  
with tf.GradientTape() as tape:  
    fake_images = self.generator(random_latent_vectors, training=True)  
    fake_predictions = self.critic(fake_images, training=True)  
    g_loss = -tf.reduce_mean(fake_predictions) ❻ → Calcula la pérdida de Wasserstein para el generador.  
  
gen_gradient = tape.gradient(g_loss, self.generator.trainable_variables)  
self.g_optimizer.apply_gradients(  
    zip(gen_gradient, self.generator.trainable_variables))  
) ❼ → Actualiza los pesos del generador.  
  
self.c_loss_metric.update_state(c_loss)  
self.c_wass_loss_metric.update_state(c_wass_loss)  
self.c_gp_metric.update_state(c_gp)  
self.g_loss_metric.update_state(g_loss)  
  
return {m.name: m.result() for m in self.metrics}
```

Wasserstein GAN con Penalización de Gradiente

Batch Normalization en una WGAN-GP

- Una última consideración que debemos tener en cuenta antes de entrenar un WGAN-GP es que **no se debe utilizar normalización por lotes en el crítico**.
- Esto se debe a que la normalización por lotes crea correlación entre las imágenes en el mismo lote, lo que hace que la pérdida de penalización del gradiente sea menos efectiva.
- Los experimentos han demostrado que los WGAN-GP aún pueden producir excelentes resultados incluso sin normalización por lotes en el crítico.

Wasserstein GAN con Penalización de Gradiente

Diferencias clave entre un GAN estándar y un WGAN-GP

- Un WGAN-GP utiliza la pérdida de Wasserstein.
- El WGAN-GP se entrena utilizando etiquetas de 1 para las imágenes reales y -1 para las imágenes falsas.
- No hay activación sigmoide en la capa final del crítico.
- Se incluye un término de penalización del gradiente en la función de pérdida del crítico.
- Se entrena al crítico varias veces por cada actualización del generador.
- No hay capas de normalización por lotes en el crítico.

Wasserstein GAN con Penalización de Gradiente

Algunos ejemplos de rostros generados

Época 1



Época 25



- El modelo ha aprendido los atributos importantes a nivel alto de un rostro, y no hay signos de colapso de modo.

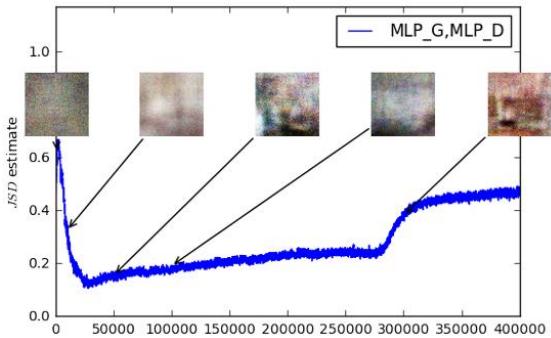
Wasserstein GAN con Penalización de Gradiente

$$\mathcal{L}_D \approx \underbrace{\frac{1}{m} \sum_{i=1}^m D(G(z_i)) - \frac{1}{m} \sum_{i=1}^m D(x_i)}_{\text{Wasserstein loss}} + \underbrace{\lambda \frac{1}{m} \sum_{i=1}^m \left(\|\nabla_{\hat{x}_i} D(\hat{x}_i)\|_2 - 1 \right)^2}_{\text{Gradient penalty}}$$

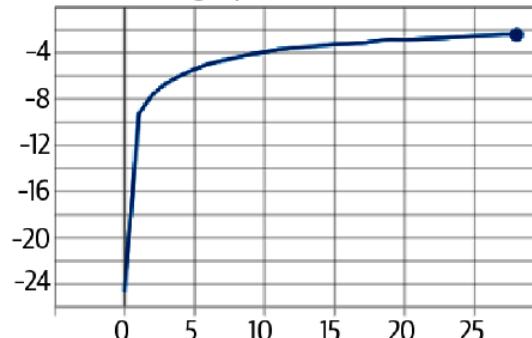
W **GP**

\mathcal{L}_D epoch_c_loss
tag: epoch_c_loss

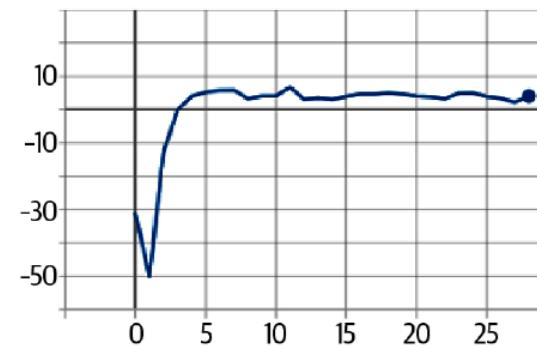
epoch_g_loss
tag: epoch_g_loss $\mathcal{L}_G = -\mathbb{E}[D(G(z))]$.



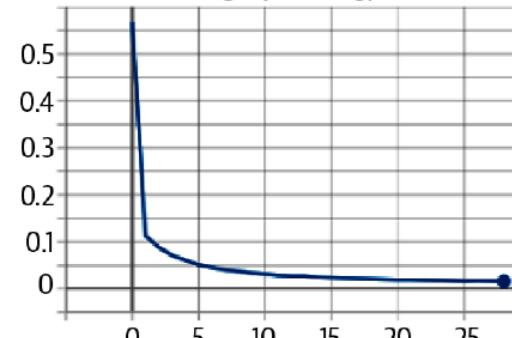
W epoch_c_wass_loss
tag: epoch_c_wass_loss



Son estables y convergentes.



GP epoch_c_gp
tag: epoch_c_gp

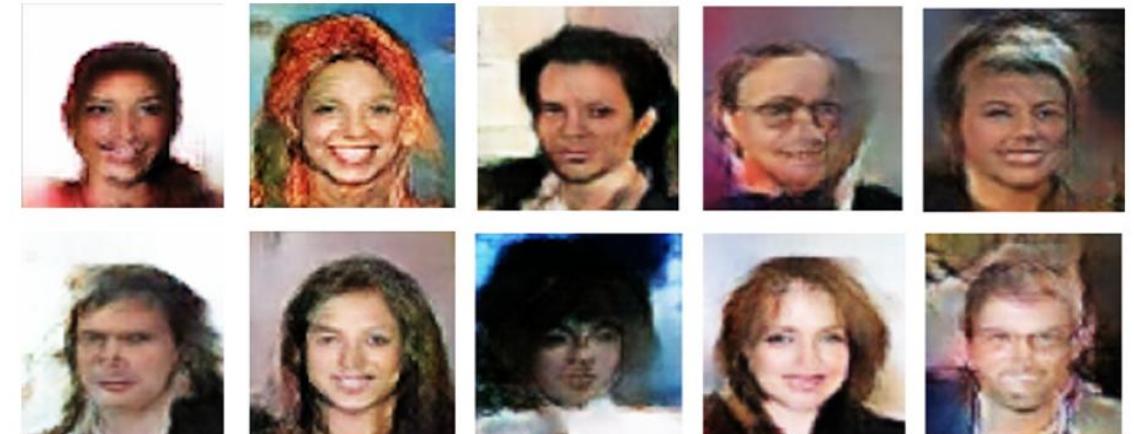


Wasserstein GAN con Penalización de Gradiente

Análisis



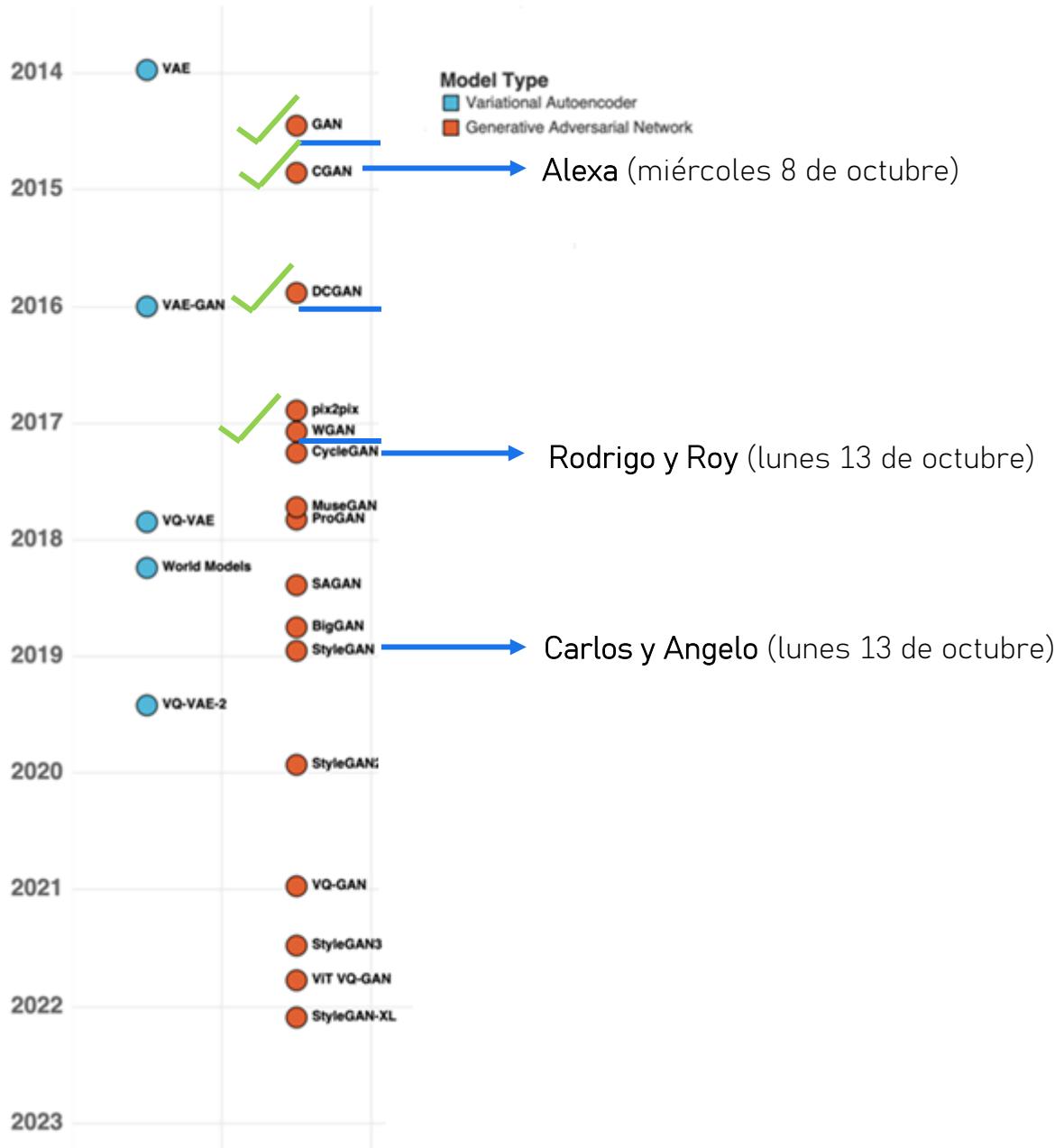
Con VAE



Con WGAN-GP

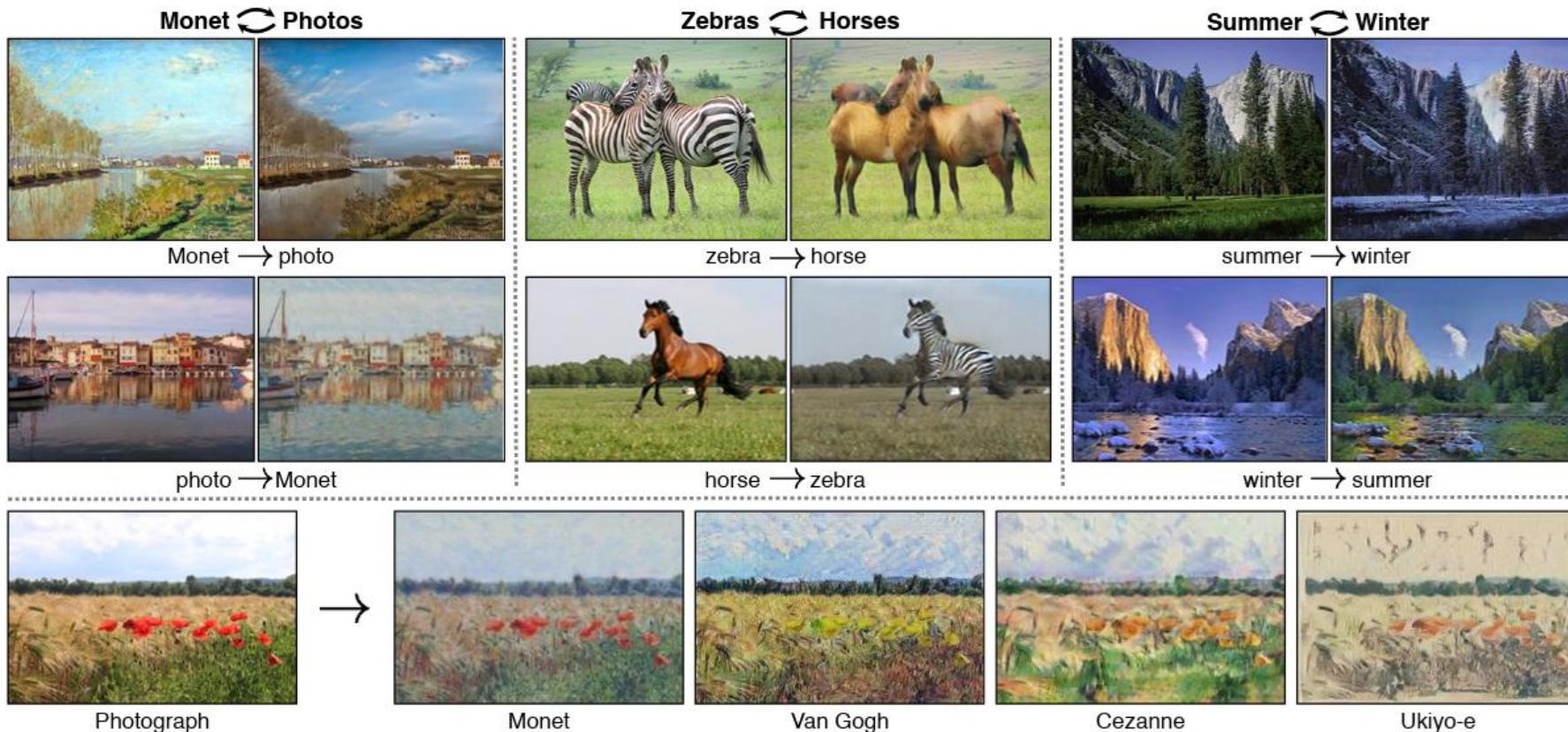
- Los VAEs tienden a producir imágenes más suaves que difuminan los límites de color, mientras que los GANs son conocidos por generar imágenes más nítidas y bien definidas.
- Los GANs son generalmente más difíciles de entrenar que los VAEs y tardan más en alcanzar una calidad satisfactoria.
- Muchos modelos generativos de vanguardia hoy en día están basados en GANs, ya que las recompensas por entrenar GANs a gran escala en GPUs durante un periodo de tiempo más largo son significativas.

Generative AI Timeline



Línea de tiempo de IA Generativa

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



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.