Fundamentos matemáticos y aplicaciones prácticas de las Redes Generativas Antagónicas (GANs)

Nicolás Vives Vicente

Universidad de Alicante

Curso 2024/2025

Índice

Preliminares

Neurona artificial

Regresión Lineal

$$\hat{y} = \omega_1 x_1 + \omega_2 x_2 + \cdots + \omega_p x_p + b$$

Neurona artificial

A raíz del modelo de regresión lineal, se define una neurona artificial como

$$\hat{y} = f(\omega_1 x_1 + \omega_2 x_2 + \cdots + \omega_p x_p + b) = f(\omega \cdot x + b)$$

f es la función de activación para romper la linealidad.

Redes Neuronales

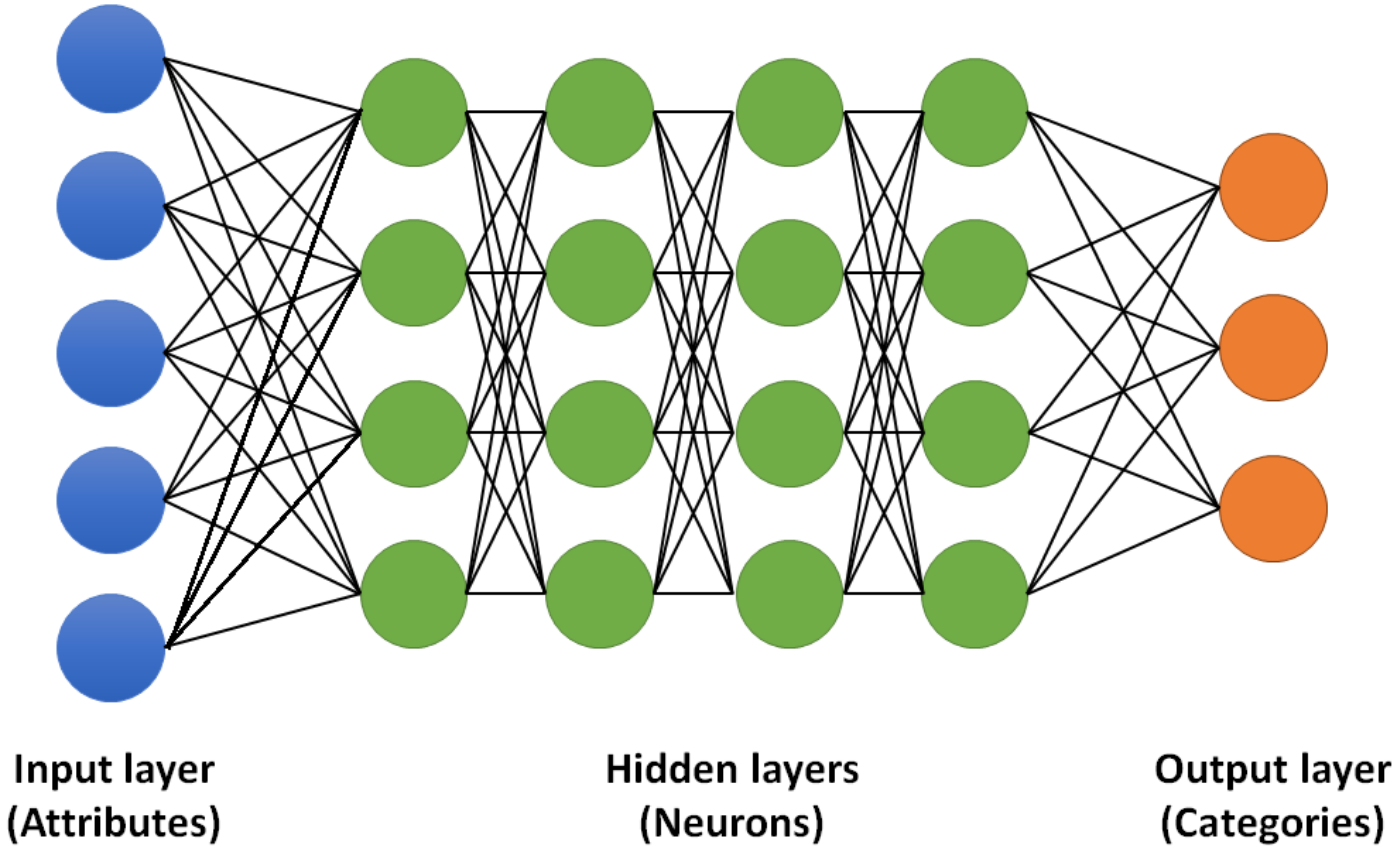


Figura: Estructura de una red neuronal artificial.

Redes Generativas Antagónicas

Modelaje

- Datos originales con distribución p_{data}
- Muestras z provenientes de una variable de con distribución p_z

Generador

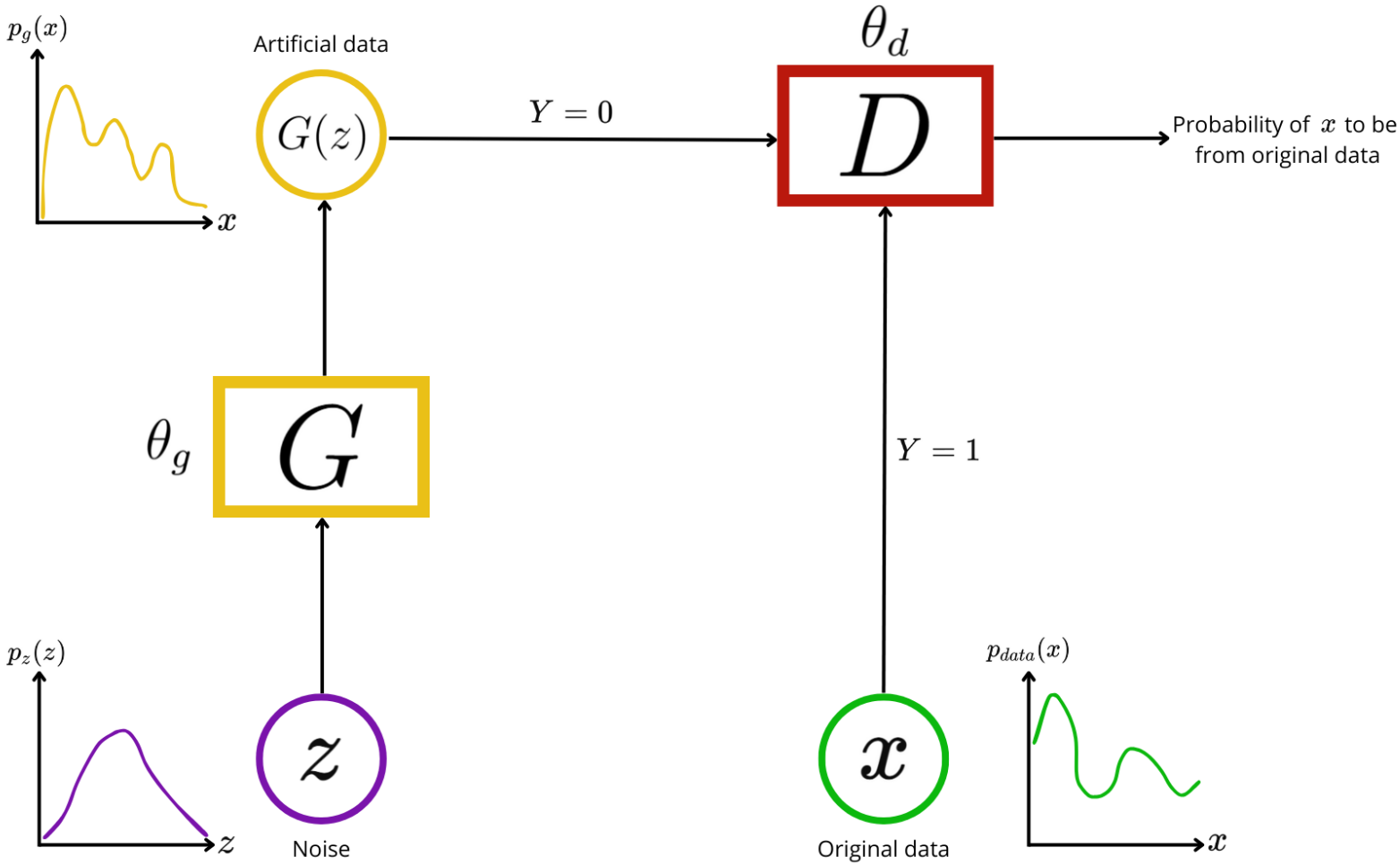
G red neuronal con parámetros $\theta_g \in \Theta_G$ que produce $G(z; \theta_g)$

- G genera una distribución p_g

Discriminador

D red neuronal con parámetros $\theta_d \in \Theta_D$ produce $D(x, \theta_d) \in [0, 1]$

Estructura de una GAN



Proceso de entrenamiento

Función de pérdida

$$\mathcal{L}(D, G) = -\mathbb{E}_{x \sim p_{data}(x)} [\log D(x)] - \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))]$$

Objetivos del discriminador:

- Si x proviene de p_{data}: se busca $D(x) \to 1$, equivalente a minimizar $-\log(D(x))$.
- Si $x = G(z)$ proviene de p_g: se busca $D(x) = D(G(z)) \to 0$, equivalente a minimizar $-\log(1 - D(G(z)))$.

Objetivos del generador:

- Si $x = G(z)$ proviene de p_g: se busca $D(x) = D(G(z)) \to 1$ equivalente a maximizar $-\log(1 - D(G(z)))$.

Proceso de entrenamiento

Problema minimax

Problema minimax

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}\left[\log D(x)\right] + \mathbb{E}_{z\sim p_z(z)}\left[\log(1-D(G(z)))\right]$$

Proceso de entrenamiento

Discriminador óptimo

Para G fijo, el discriminador D óptimo es:

$$D_G^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}$$

$$\begin{aligned} C(G) &= \max_D V(G, D) \\ &= \mathbb{E}_{x \sim p_{data}}[\log D_G^*(x)] + \mathbb{E}_{z \sim p_z}[\log(1 - D_G^*(G(z)))] \\ &= \mathbb{E}_{x \sim p_{data}}[\log D_G^*(x)] + \mathbb{E}_{x \sim p_g}[\log(1 - D_G^*(x))] \\ &= \mathbb{E}_{x \sim p_{data}}\left[\log \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}\right] + \mathbb{E}_{x \sim p_g}\left[\log \frac{p_g(x)}{p_{data}(x) + p_g(x)}\right] \end{aligned}$$

Proceso de entrenamiento

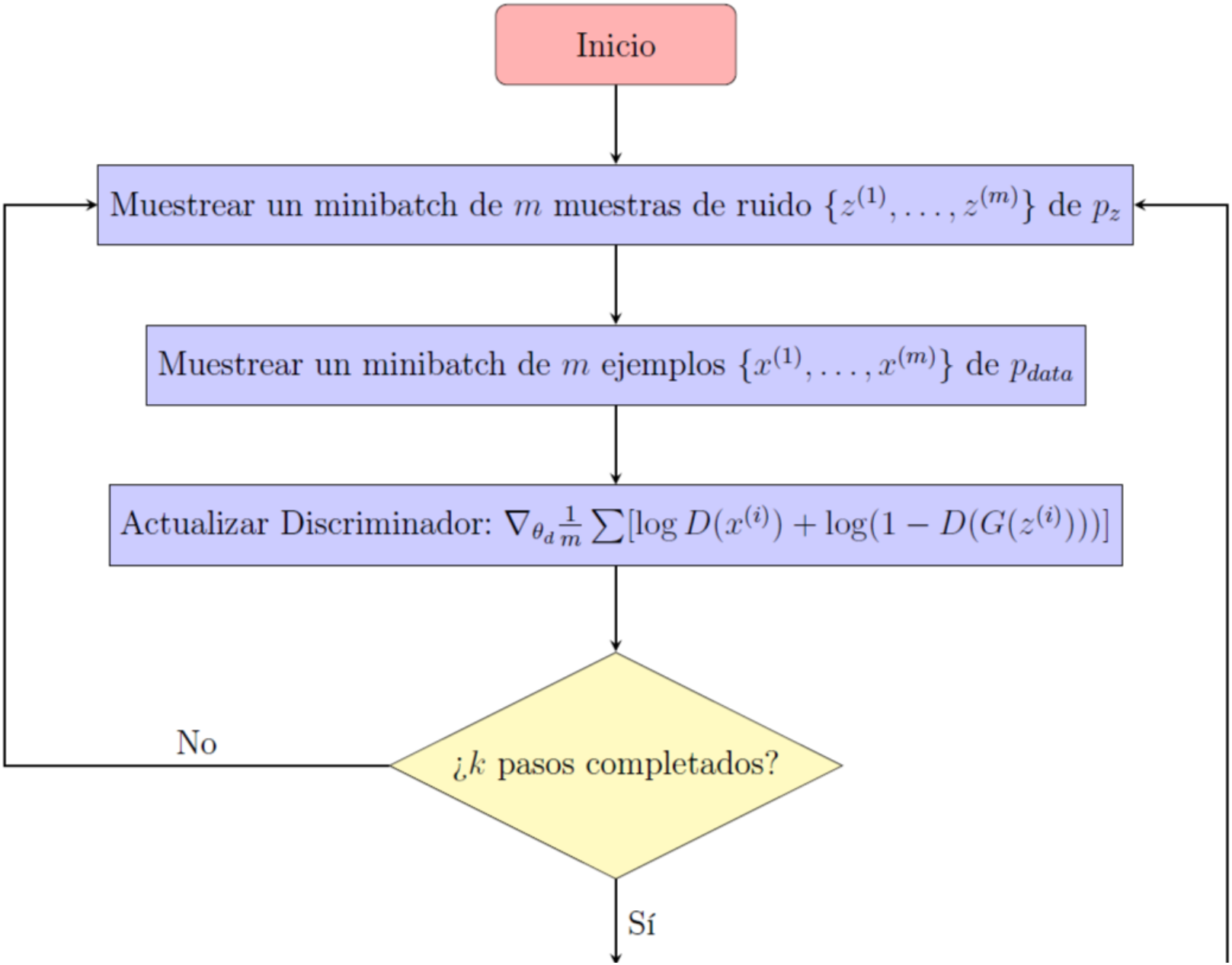
Mínimo de $C(G)$

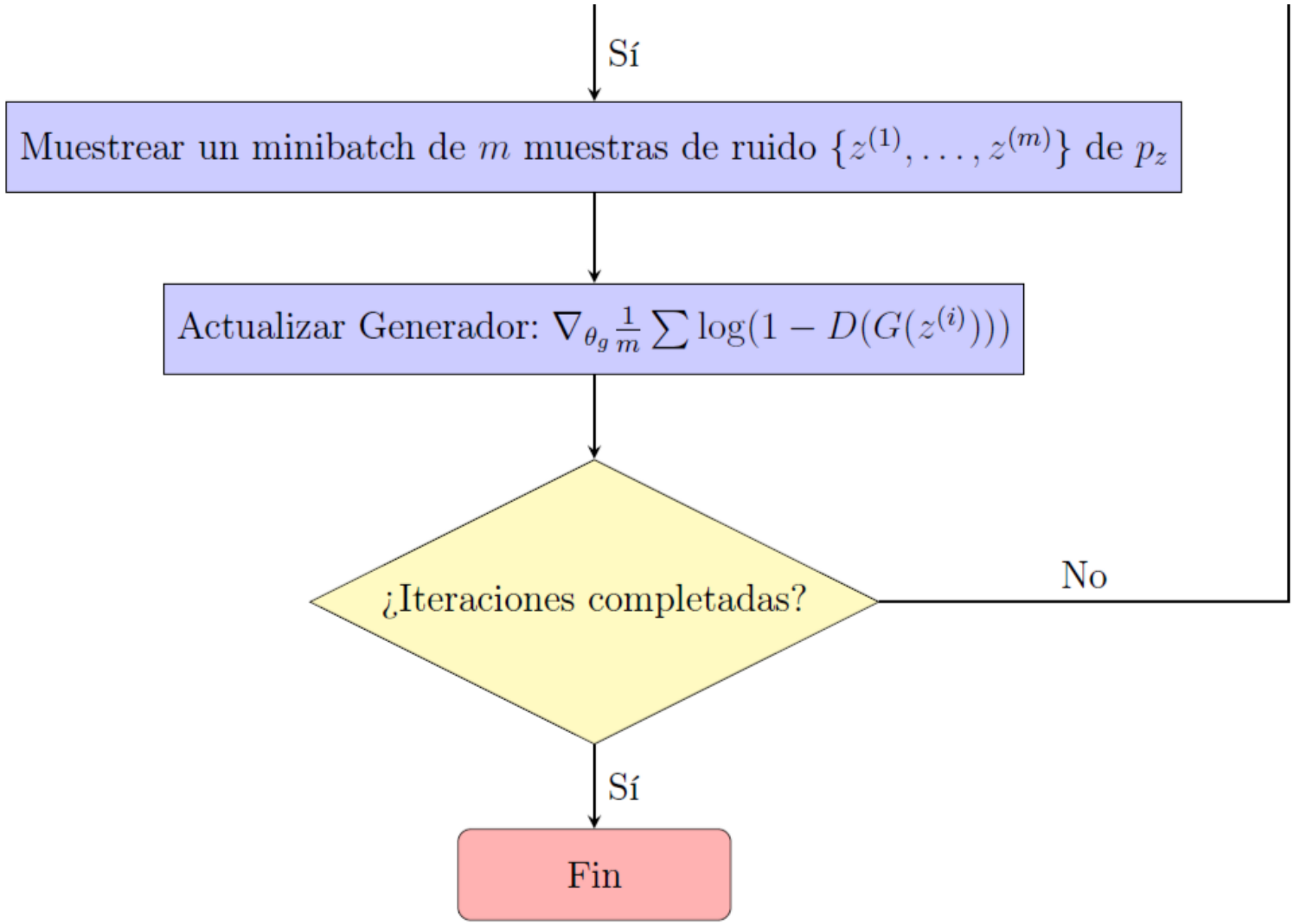
El mínimo global de $C(G)$ se alcanza si y solo si $p_g = p_{data}$. En ese punto, $C(G)$ toma el valor $-\log 4$.

En la teoría, el modelo se formula como un problema de optimización sobre las funciones G y D. Sin embargo, en la práctica:

- No exploramos todas las posibles funciones G y D para encontrar el óptimo del problema.
- La distribución p_g solo se conoce de manera implícita a través del generador.

Como $G = G(z; \theta_g)$ y $D = D(x; \theta_g)$ son redes neuronales, la optimización se lleva a cabo ajustando los parámetros $\theta_g \in \Theta_G$ y $\theta_d \in \Theta_D$.


Inicio
Muestrear un minibatch de m muestras de ruido $\{z^{(1)}, \ldots, z^{(m)}\}$ de p_z
Muestrear un minibatch de m ejemplos $\{x^{(1)}, \ldots, x^{(m)}\}$ de p_{data}
Actualizar Discriminador: $\nabla_{\theta_d} \frac{1}{m} \sum [\log D(x^{(i)}) + \log(1 - D(G(z^{(i)})))]$
¿k pasos completados?
No
Sí


Sí
Muestrear un minibatch de m muestras de ruido $\{z^{(1)}, \ldots, z^{(m)}\}$ de p_z
Actualizar Generador: $\nabla_{\theta_g} \frac{1}{m} \sum \log(1 - D(G(z^{(i)})))$
¿Iteraciones completadas?
No
Sí
Fin

Convergencia del Algoritmo

Decimos que un modelo tiene suficiente capacidad cuando es capaz de representar cualquier función.

Convergencia del Algoritmo

Supongamos que se cumplen las hipótesis:

a) Los modelos de G y D tienen suficiente capacidad.

b) En cada paso del Algoritmo, el discriminador es capaz de alcanzar su valor óptimo dado cualquier G.

c) La distribución p_g se actualiza con el fin de minimizar la expresión

$$\mathbb{E}_{x\sim p_{data}}[\log D_G^*(x)] + \mathbb{E}_{x\sim p_g}[\log(1 - D_G^*(x))]$$

Entonces, p_g converge a p_{data}, tomando así la distribución generada la forma de la distribución de los datos.

Limitaciones y problemas en la práctica

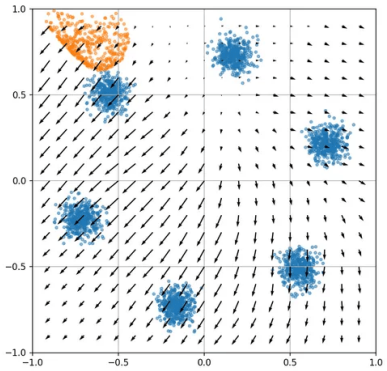
Desvanecimiento de gradientes (*Vanishing gradients*)

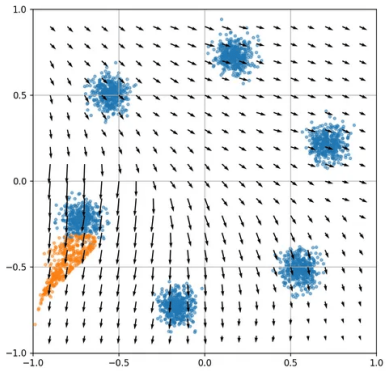
- **Problema**: Cuando D es mucho mejor que G se tiene que

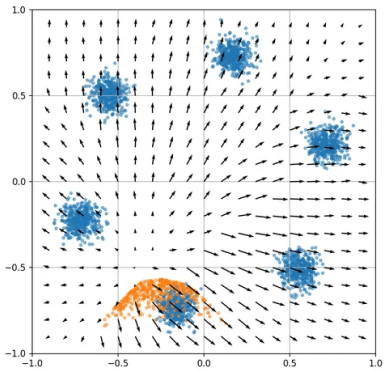
$$\left\| \nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(1 - D(G(z^{(i)}))) \right\| \to 0$$

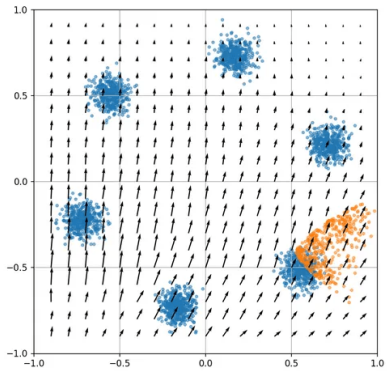
- **Consecuencias**:
  - Actualizaciones de θ_g insignificantes
  - G y D no consiguen mejorar
- **Posibles soluciones**:
  - Maximizar $\log(D(G(z)))$ en lugar de minimizar $\log(1 - D(G(z)))$
  - Ajustar el hiperparámetro k y las *learning rates*
- **Conclusiones**:
  - D debe ser bueno pero no perfecto
  - No es un problema fácil de solucionar y continúa apareciendo en modelos de GANs actuales.

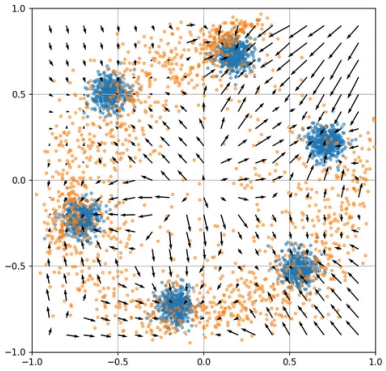
Colapso de modo (*Mode Collapse*)

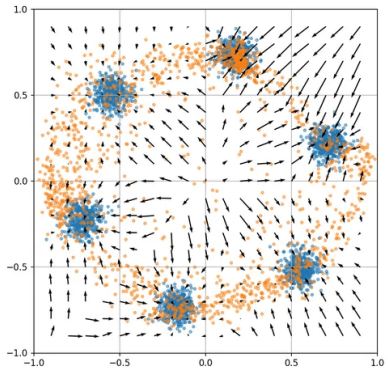

Iteration 1000

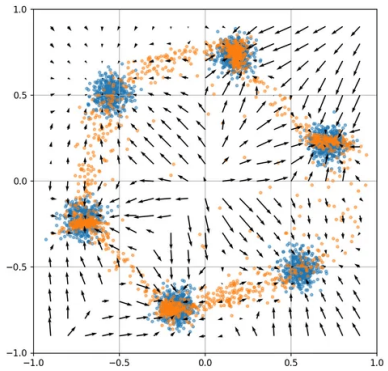

Iteration 1500

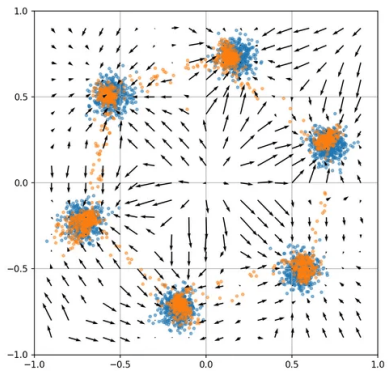

Iteration 2000


Iteration 2500


Iteration 1000


Iteration 2000


Iteration 3000


Iteration 4000

Colapso de modo (*Mode Collapse*)

- **Problema**: G transforma cualquier vector de ruido z en elementos prácticamente idénticos del espacio de datos.
- **Causas**:
  - Capacidad insuficiente del modelo
  - Sobreentreno del discriminador
  - Ausencia de variedad en el *dataset*
  - Función de pérdida y parámetros inadecuados
- **Soluciones**:
  - Aumentar la complejidad del generador
  - Ajustar correctamente al discriminador
  - Correcta elección de la función de pérdida e hiparámetros

DCGAN

Convolución

Llamamos Kernel de tamaño $2N+1$ y Kernel no centrado de tamaño N a las matrices:

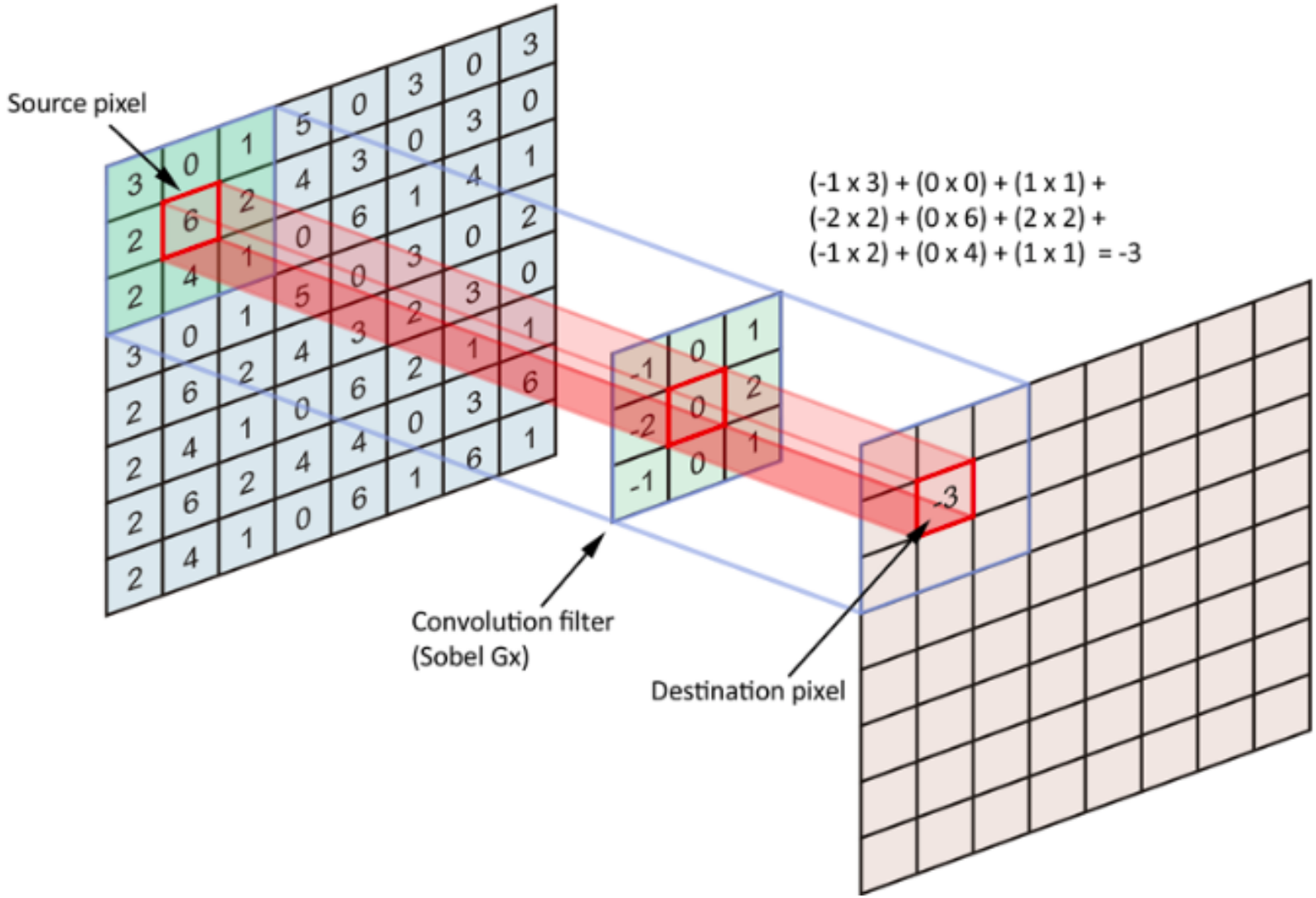
$$\begin{bmatrix} k_{-N,-N} & \cdots & k_{-N,N} \\ \vdots & \ddots & \vdots \\ k_{N,-N} & \cdots & k_{N,N} \end{bmatrix} \qquad \begin{bmatrix} k_{1,1} & \cdots & k_{1,N} \\ \vdots & \ddots & \vdots \\ k_{N,1} & \cdots & k_{N,N} \end{bmatrix}$$

Convolución entre M y K

$$Y(i,j) = \sum_{u=-N}^{N} \sum_{v=-N}^{N} M(i-u, j-v) K(u,v)$$

- $M(i,j)$ representa la entrada en la posición (i,j) de la matriz M.
- $K(s,t)$ es la posición (s,t) del kernel K. Centrado en $k_{0,0}$.
- $Y(i,j)$ es la salida en la posición (i,j).

Convolución



Convolución traspuesta entre Y y K

- Y una matriz de tamaño $m \times n$
- $Y(i,j)$ la entrada en la posición (i,j) de la matriz Y.
- K kernel no centrado de tamaño N
- $K(u,v)$ la posición (u,v) del kernel K.
- s el valor del hiperparáemtro *stride*

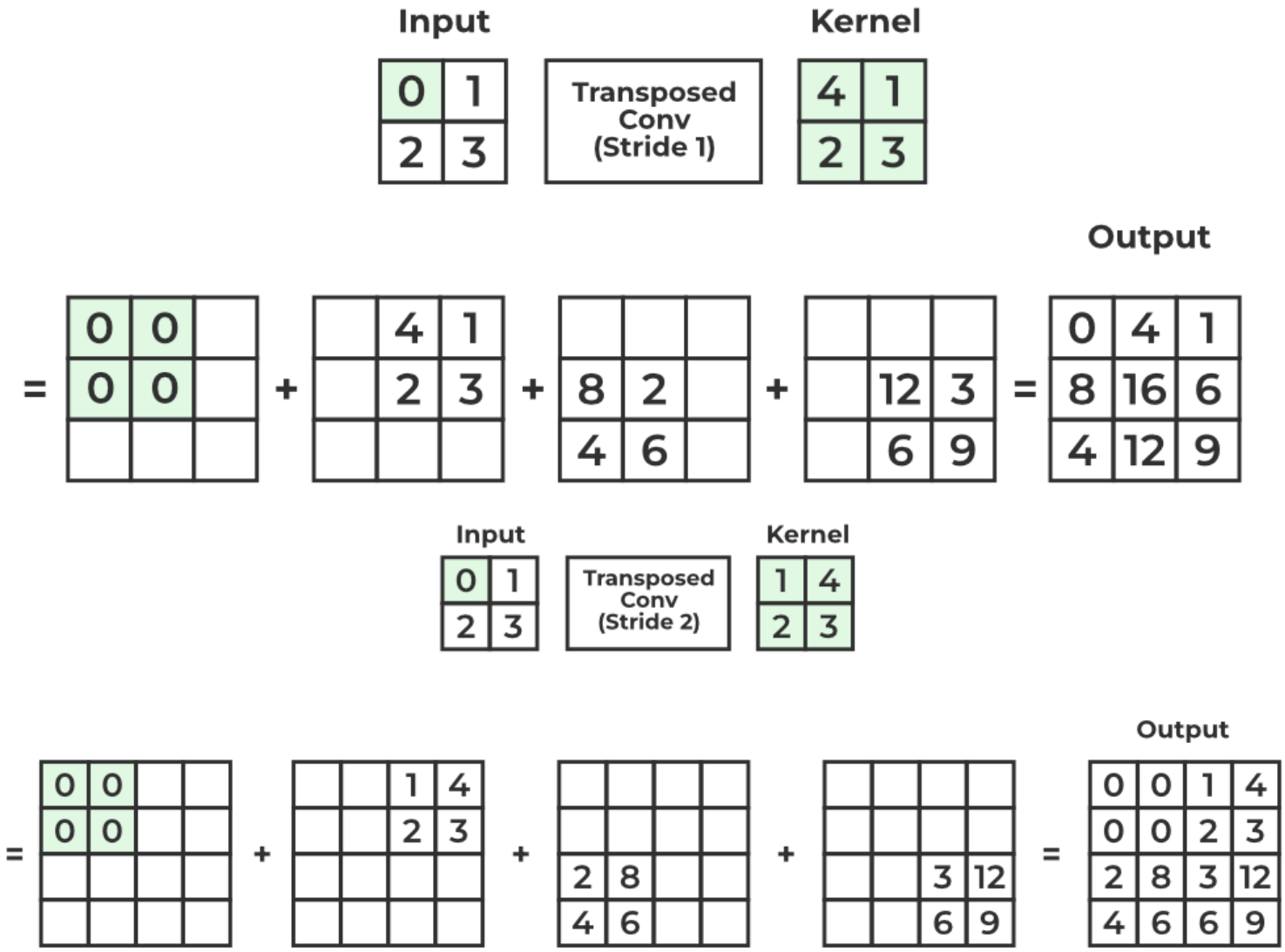
Para cada $Y(i,j)$ se define M_q de dimensión $s(m-1)+N \times s(n-1)+N$:

$$M_q(s(i-1)+u, s(j-1)+v) = \sum_{u=1}^{N}\sum_{v=1}^{N} Y(i,j)K(u,v)$$

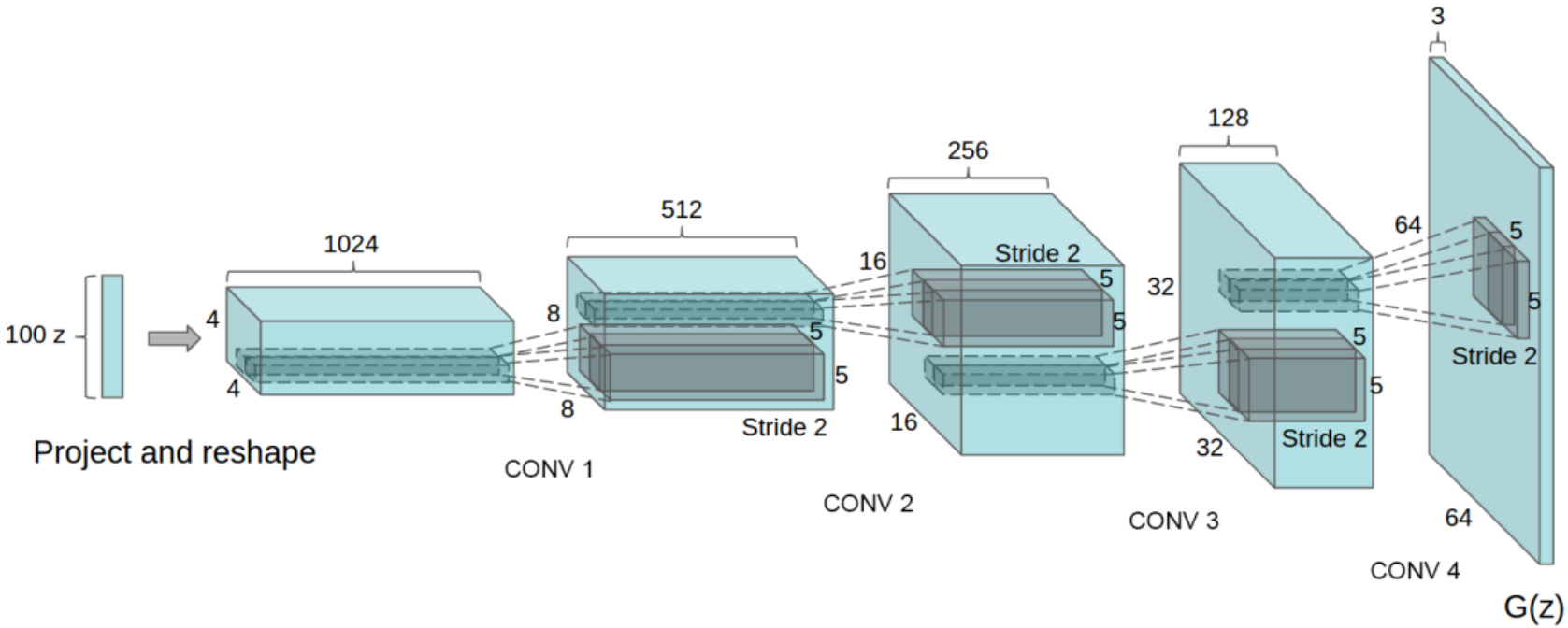
los elementos de la matriz no definidos por esta fórmula toman el valor 0.

$$M = \sum_{q=1}^{m\times n} M_q$$

Convolución traspuesta



Estructura del Generador de una DCGAN



Desarrollo en python de una GAN

Modelo y *dataset*

- **Modelo:** DCGAN (Deep Convolutional GAN)
- **Dataset:** Fashion-MNIST
  - 70.000 imágenes (60k entrenamiento, 10k test)
  - Resolución: 28×28 píxeles (escala de grises, 1 canal)
  - 10 categorías: camiseta, pantalón, suéter, vestido, abrigo, sandalia, camisa, zapatilla deportiva, bolso, botín.

Estructura del generador

Tipo de capa	Dimensiones del Output	Parámetros
Dense	6272	633 472
Reshape	(7, 7, 128)	0
Batch Normalization	(7, 7, 128)	512
Traspose Convolution	(14, 14, 64)	204 864
Batch Normalization	(14, 14, 64)	256
Traspose Convolution	(28, 28, 1)	1 601

Estructura del discriminador

Tipo de capa	Dimensiones del Output	Parámetros
Convolution	(14, 14, 64)	1 664
LeakyReLU	(14, 14, 64)	0
Dropout	(14, 14, 64)	0
Convolution	(7, 7, 128)	204 928
LeakyReLU	(7, 7, 128)	0
Dropout	(7, 7, 128)	0
Flatten	6272	0
Dense	1	6 273

Datos de entrenamiento

- **Número de *epochs***: 20
- **Tamaño del *batch***: 32
- ***Batches* totales**: 37 500
- **Tiempo de entrenamiento**: 55 minutos

Resultados

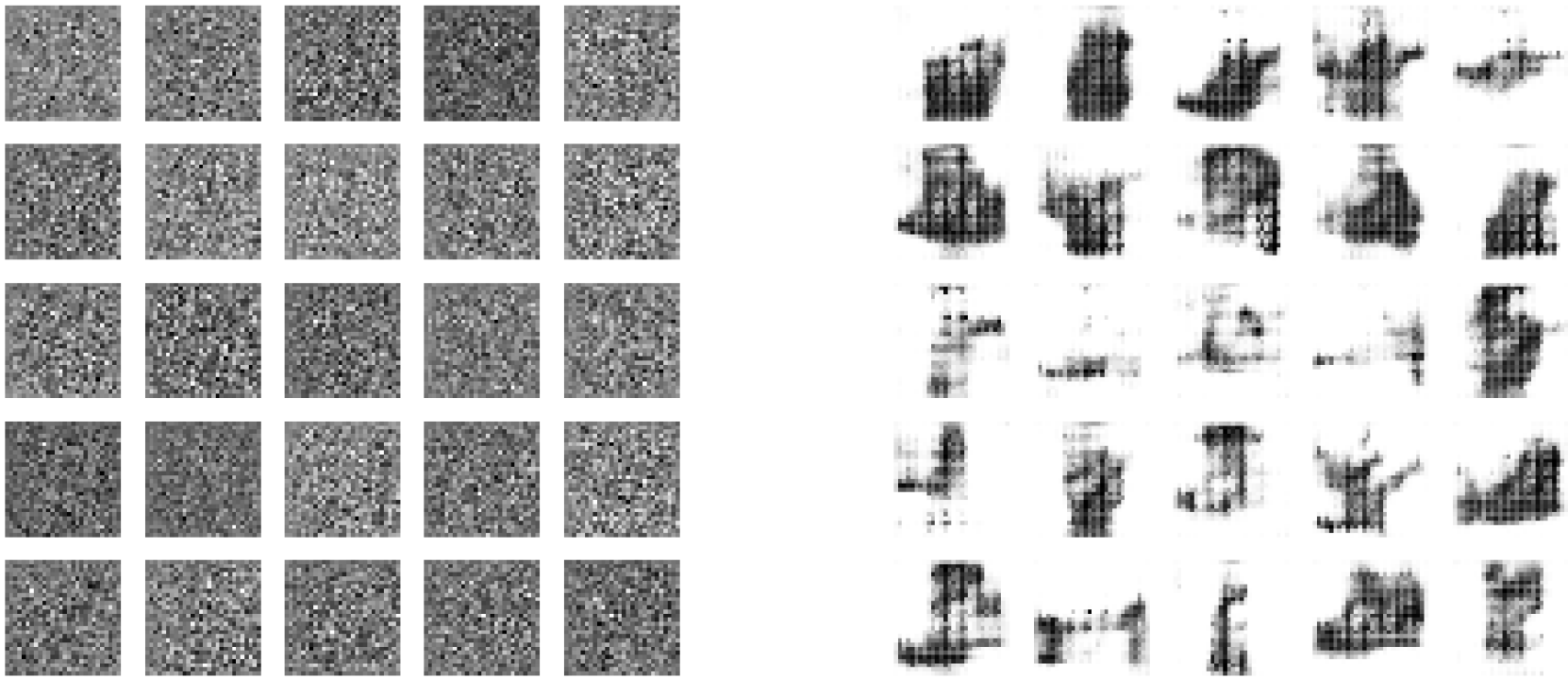


Figura: Ruido inicial e imágenes producidas por la GAN tras el primer *epoch*.

Resultados

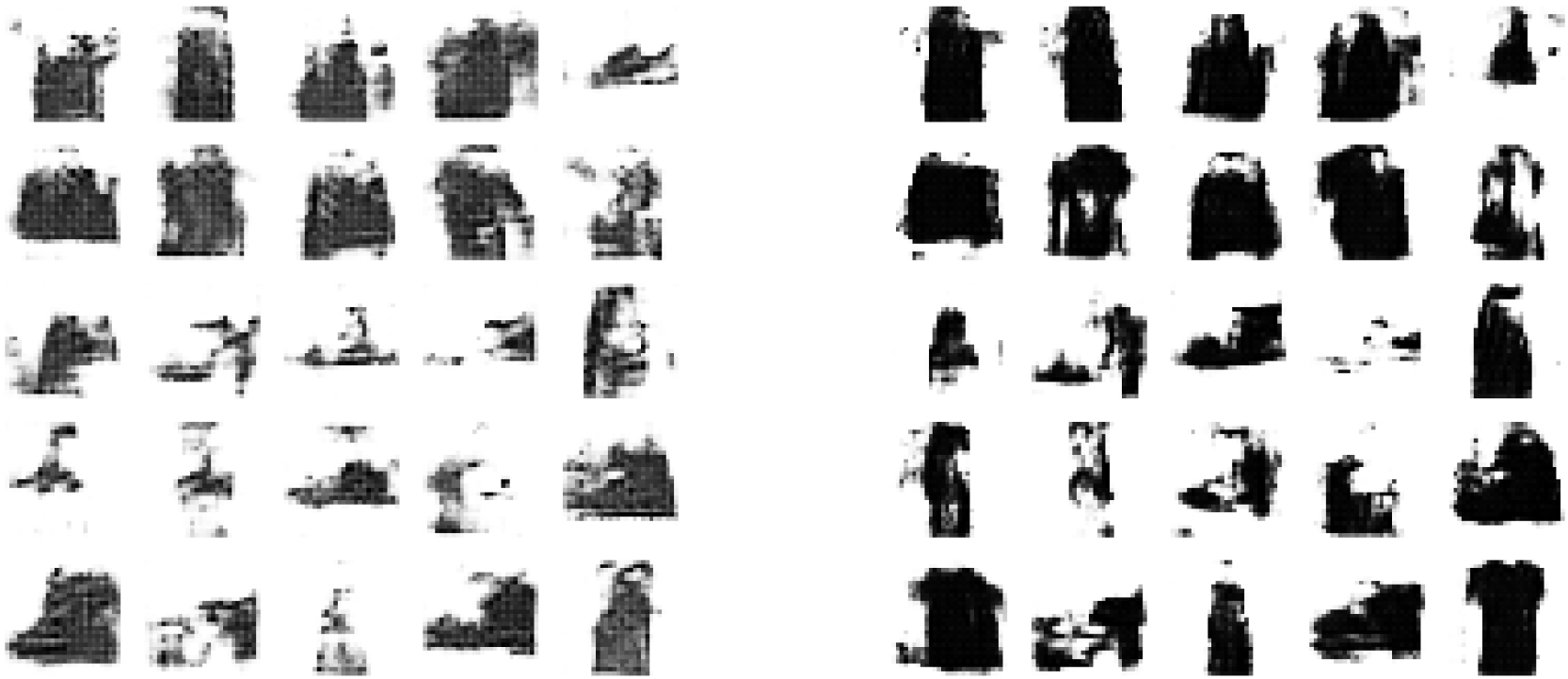


Figura: Imágenes generadas tras los *epochs* número 2 y 3 respectivamente.

Resultados

Figura: Imágenes generadas tras los *epochs* número 5 y 10 respectivamente.

Resultados

Figura: Imágenes generadas tras los *epochs* número 15 y 20 respectivamente.

Resultados

Evolución de la loss por batch

Figura: Evolución de la función de pérdida (entropía binaria cruzada) en función del *batch* de entrenamiento.

Conclusiones

Conclusiones

- En este trabajo se ha abordado el estudio de la teoría matemática tras las Redes Generativas Antagónicas.
- Se ha visto que la estructura GAN alcanza un equilibrio de Nash cuando $p_g = p_{data}$ y $D(x) = \frac{p_{data}(x)}{p_{data}(x)+p_g(x)}$.
- Se ha entrenado una DCGAN en Python, lo que ha permitido observar cómo una GAN aprende progresivamente a transformar ruido en imágenes. Durante este proceso, se han evitado problemas comunes, como el desvanecimiento de gradientes y el colapso de modo.
- Este trabajo abre la puerta al estudio de modelos generativos más actuales, como los modelos difusivos y los transformadores, que, si bien ofrecen avances significativos en la generación de datos, también presentan sus propios desafíos e inconvenientes.

Conclusiones

- Este trabajo me ha permitido integrar conocimientos adquiridos durante el grado para acercarme más al campo de la inteligencia artificial que tanto me interesa
- La experiencia obtenida con el entrenamiento y el análisis de modelos GAN me sirve como base sólida para investigar y desarrollar otros modelos generativos más avanzados en futuros trabajos de investigación o un posible TFM.
- En definitiva, el trabajo me ha permitido profundizar en el conocimiento de los modelos generativos y me motiva a seguir investigando en esta área en el futuro.

Gracias