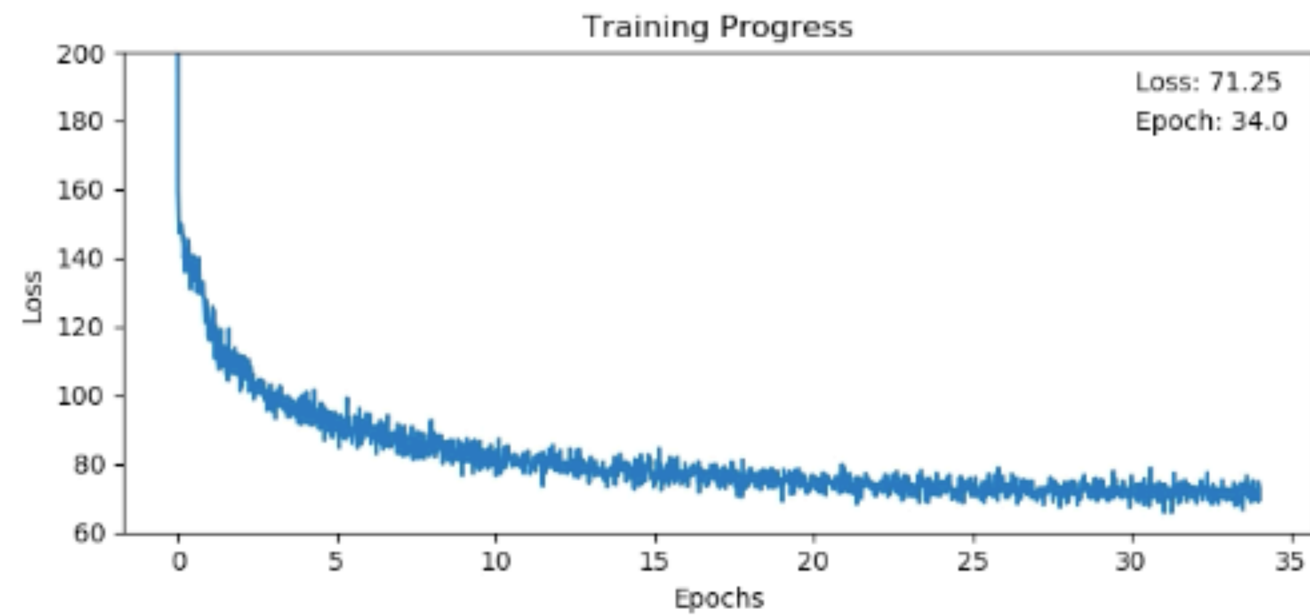
Aprendizaje profundo basado en la física

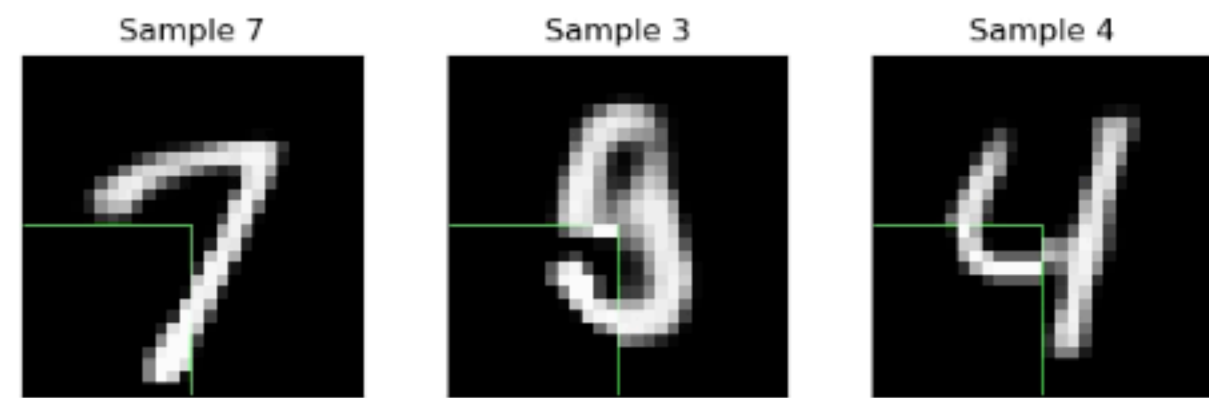
Semana 5: Modelos probabilísticos

Docente: José I. Robledo - 05/05/2026

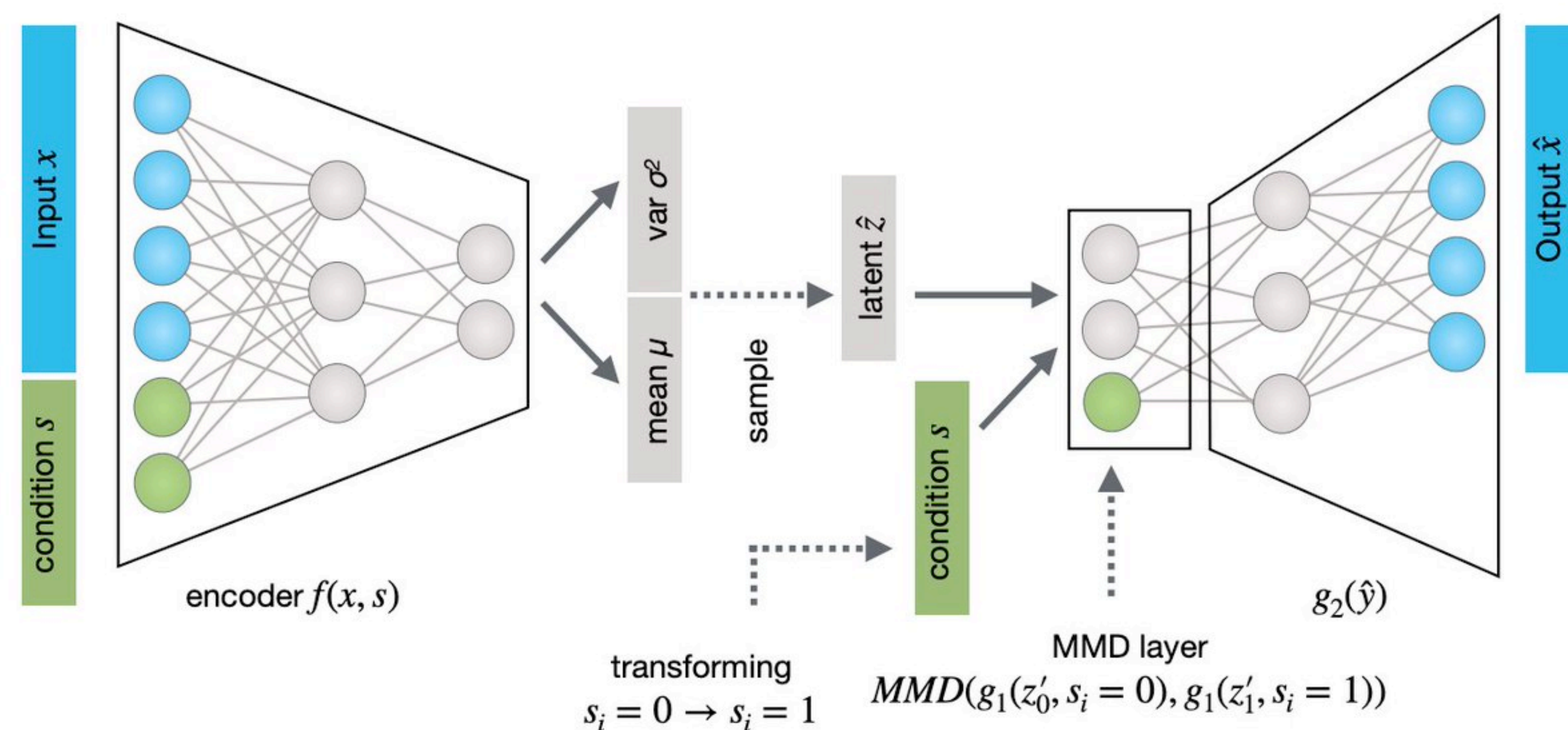
Modelos probabilísticos

Conditional VAE





$$\mathscr{L}_{VAE} = \mathbb{E}_{z \sim q_\phi(z|x)} \left[\log p_\theta(x \,|\, z) \right] - D_{KL}(q_\phi(z \,|\, x) \,||\, p(z))$$



$$\mathscr{L}_{CVAE} = \mathbb{E}_{z \sim q_\phi(z|x,y)} \left[\log p_\theta(y \,|\, x, z) \right] - D_{KL}(q_\phi(z \,|\, x, y) \,||\, p(z))$$

https://arxiv.org/pdf/2102.12037

Modelos probabilísticos

Physics Informed VAE

Podemos agregar información fisica!

PI-VAE training stage:

Generate synthetic dataset $\boldsymbol{Z}_{\text{syn}}$ using the sampling method from Algorithm 1.

Iteration For epoch $= 1$ to E_2 do

Select a mini-batch $\boldsymbol{X}_1$ from $\boldsymbol{X}_{\text{data}}$

Reconstruct the training data batch to get $\hat{\boldsymbol{X}}_1 = D_{\boldsymbol{\theta}}(E_{\boldsymbol{\phi}}(\boldsymbol{X}_1))$

Compute the VAE loss $\mathcal{L}_{\text{vae}}$ for $\boldsymbol{X}_1$ using Equation (5)

Compute the physical inconsistency loss $\mathcal{L}^{\text{d}}_{\text{phy}}$ on $\hat{\boldsymbol{X}}_1$ using Equation (27)

Select a mini-batch $\boldsymbol{Z}_2$ from $\boldsymbol{Z}_{\text{syn}}$

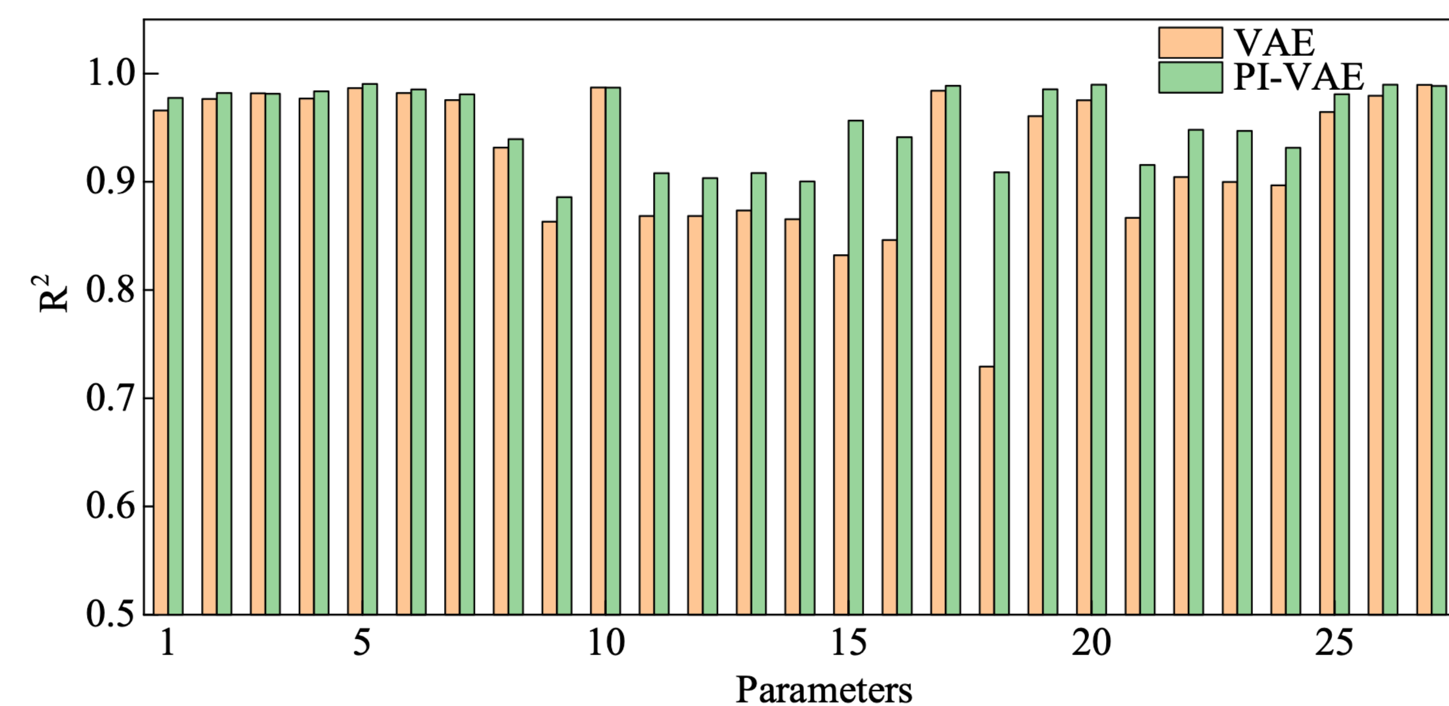
Obtain $\hat{\boldsymbol{X}}_2$ via the decoder: $\hat{\boldsymbol{X}}_2 = D_{\boldsymbol{\theta}}(\boldsymbol{Z}_2)$

Compute the physical inconsistency loss $\mathcal{L}^{\text{s}}_{\text{phy}}$ on $\hat{\boldsymbol{X}}_2$ using Equation (29)

Compute the total PI-VAE loss: $\mathcal{L}_{\text{PI-VAE}} = \mathcal{L}_{\text{VAE}} + \alpha_1 \mathcal{L}^{\text{d}}_{\text{phy}} + \alpha_2 \mathcal{L}^{\text{s}}_{\text{phy}}$

Compute the gradient $-\nabla(\mathcal{L}_{\text{PI-VAE}})$ and update $\boldsymbol{\phi}, \boldsymbol{\theta}$ by descending the gradient

Until $\mathcal{L}_{\text{PI-VAE}}$ converges, **end the iteration.**



https://www.mdpi.com/1996-1073/18/17/4742

Modelos probabilísticos

Problemas de los VAE

- ELBO es una aproximación (no optimizamos la evidencia $\log p_\theta(x)$ directamente) debido a la aproximación Variacional.
- Muchos supuestos, lo cuál puede hacer que $q_\phi(z \mid x)$ sea restrictiva.
- Muestreo suele ser borroso!

¿Podremos construir un modelo donde podamos evaluar directamente $p(x)$?

Normalizing Flows

Introducción a los Flujos Normalizadores

- Vamos a plantear un modelo que modele directamente la verosimilitud exacta.

- Modelaremos una distribución compleja objetivo $p(x)$, con $x \in \mathbb{R}^D$, **transformando una distribución simple mediante funciones invertibles.**

- Partiremos de distribución simple Gaussiana D-dimensional

$$z \sim p_z(z) = \sim \mathcal{N}_D(z \mid \mu, \sigma^2)$$

Pregunta central: **¿Cómo transforma una distribución ante un cambio de variable?**

Normalizing Flows

Transformación de distribuciones ante cambio de variable

- Supongamos transformación invertible $f : \mathbb{R}^D \to \mathbb{R}^D$. Llamemos f^{-1} a su inversa.
- Sea $\vec{z} \in \mathbb{R}^D$. Aplicamos la transformación y obtenemos una variable $y = f(\vec{z}) \in \mathbb{R}^D$. ¿Cómo cambió la densidad de probabilidad de la nueva variable aleatoria Y respecto a Z?

$$\int p_Y(\vec{y})d\vec{y} = \int p_Z(\vec{z})d\vec{z} = 1 \qquad \Longrightarrow \; p_Y(\vec{y}) = p_Z(f^{-1}(\vec{y}))\left|\det \frac{\partial f^{-1}(\vec{y})}{\partial \vec{y}}\right|$$

Jacobiano

Normalizing Flows

Transformación de distribuciones ante cambio de variable

Ejemplo $D = 1$

$$z \sim U[0,1] \qquad p_Z(z) = 1, \forall z \in [0,1]$$

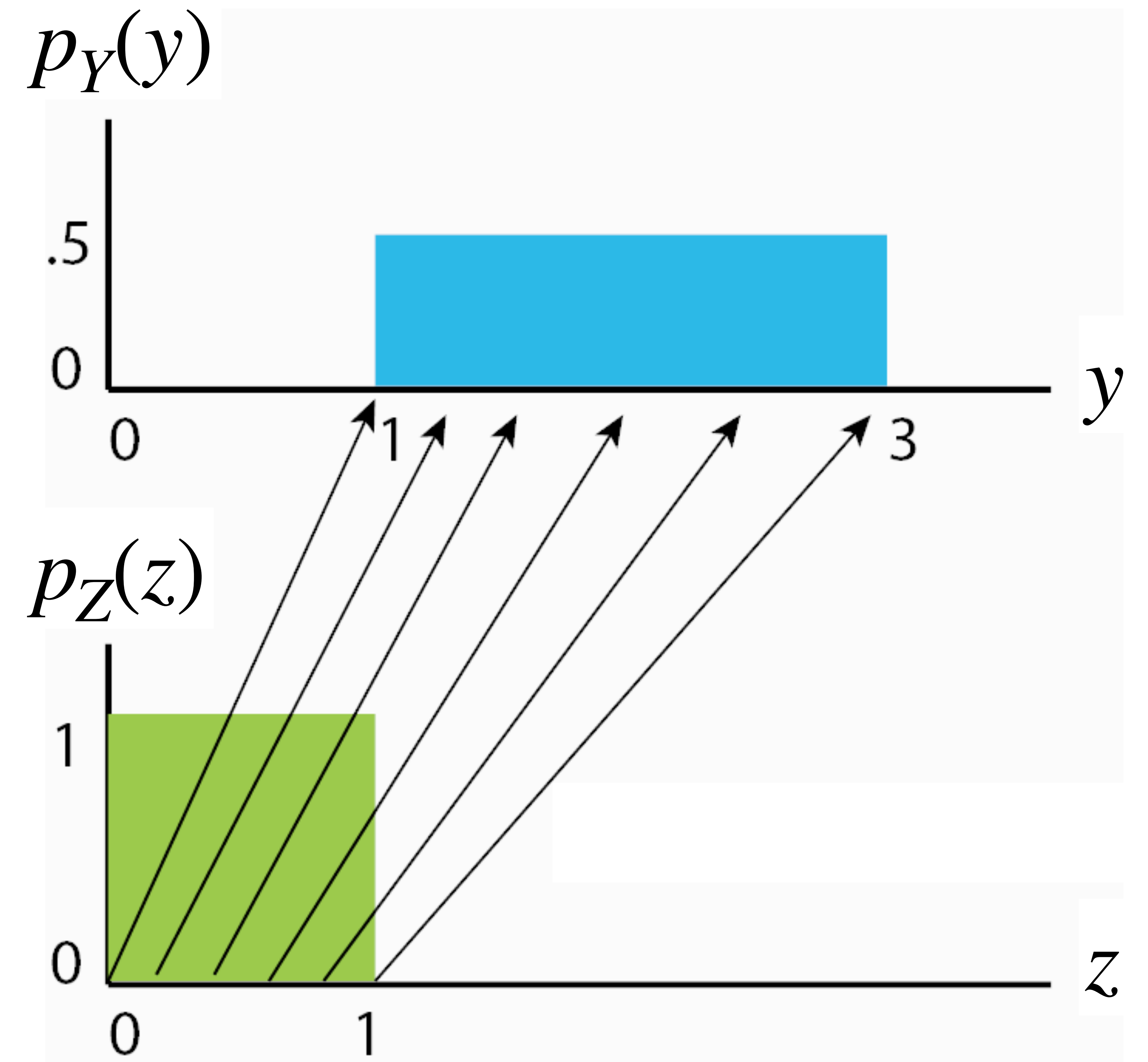
$$f(z) = 2z + 1 = y$$ **Transformación afin**

$f : \mathbb{R} \to \mathbb{R}$ es invertible ✓

$$z = f^{-1}(y) = (y - 1)/2, \qquad \frac{\partial f^{-1}(y)}{\partial y} = \frac{1}{2}$$

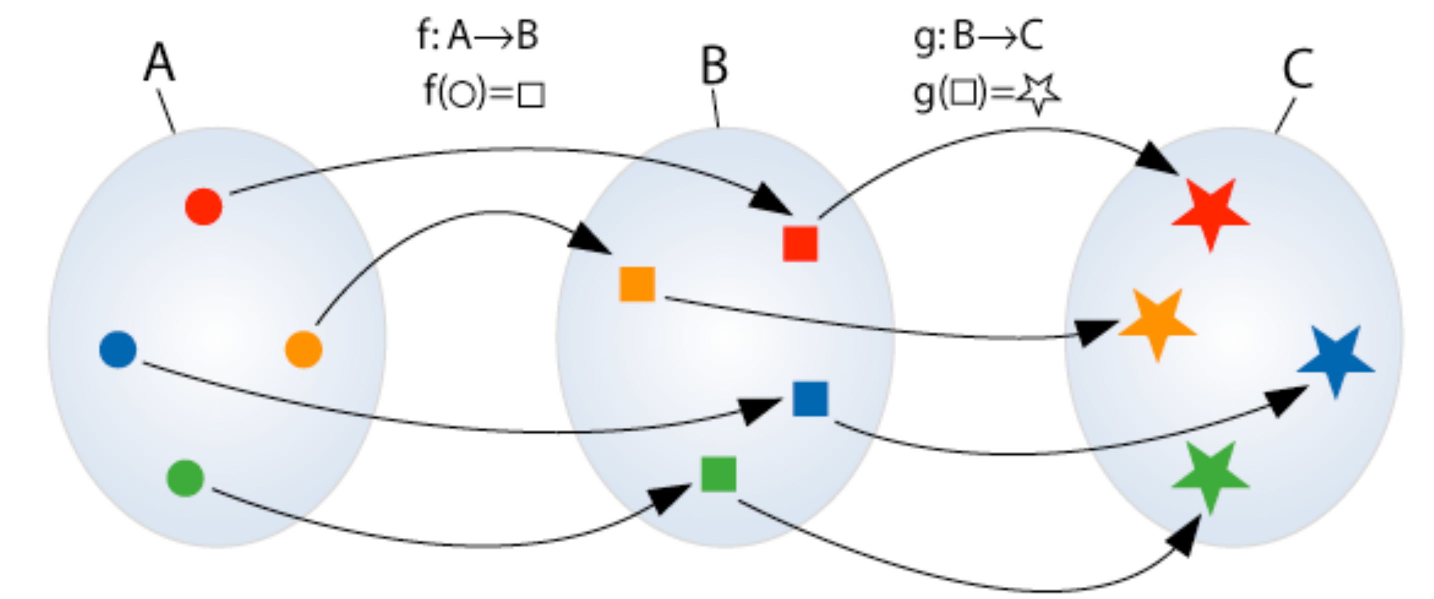
$$p_Y(y) = p_Z(f^{-1}(y)) \left| \frac{\partial f^{-1}(y)}{\partial y} \right|$$

$$\implies y \sim U[1,3] \qquad p_Y(y) = 0.5, \forall y \in [1,3]$$



Normalizing Flows

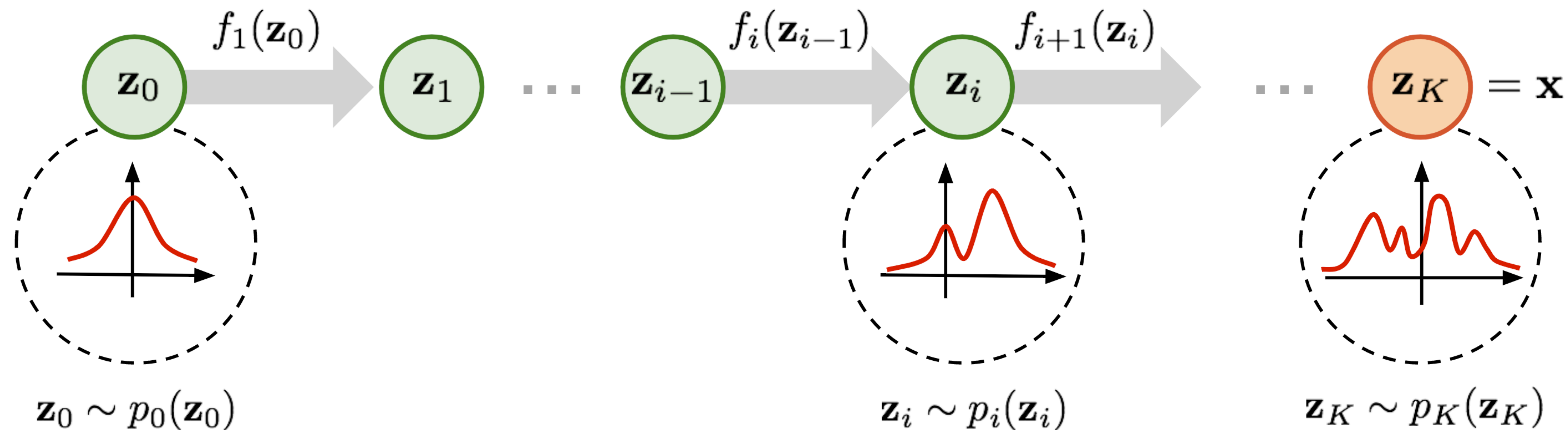
Idea central



La composición de funciones invertibles es también invertible

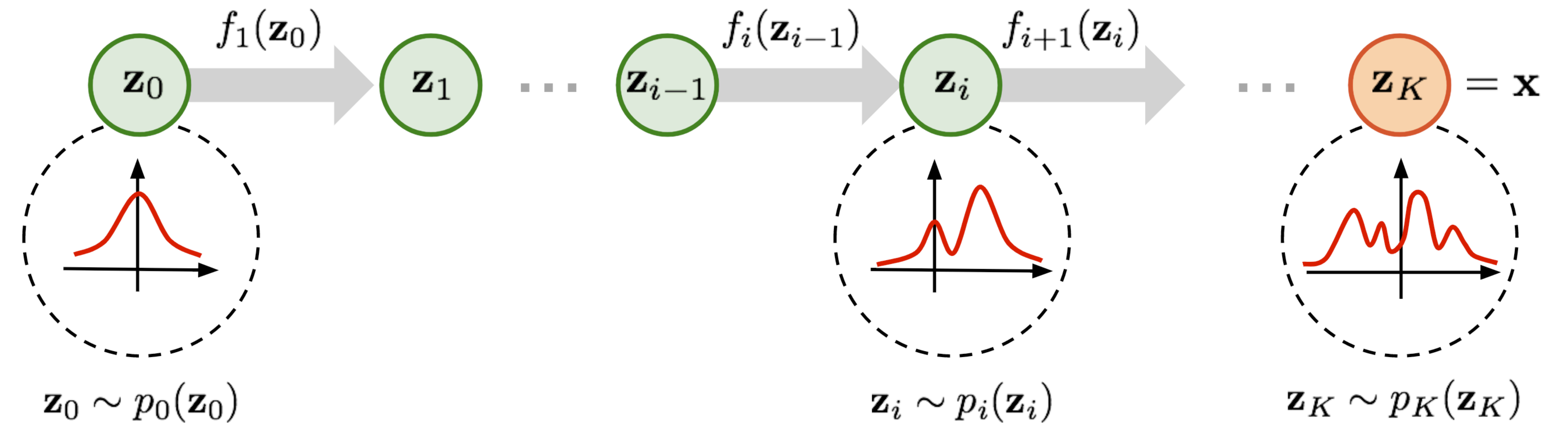
$$(f_1 \circ f_2)^{-1} = f_1^{-1} \circ f_2^{-1}$$

Un **Normalizador de flujos** o *Normalizing flow* intenta transformar la distribución $p_Z(z)$ gradualmente hacia la distribución más compleja objetivo $p_X(x)$ mediante el uso de sucesivas transformaciones invertibles.



Normalizing Flows

Como transforma la densidad



$$\vec{z}_i \sim p_i(\vec{z}_i)$$

$$\vec{z}_i = f_i(\vec{z}_{i-1}) \implies \vec{z}_{i-1} = f_i^{-1}(\vec{z}_i) \implies p_i(\vec{z}_i) = p_{i-1}(f_i^{-1}(\vec{z}_i)) \left| \det \frac{\partial f_i^{-1}(\vec{z}_i)}{\partial \vec{z}_i} \right|$$

El *teorema de la función inversa* nos dice que si $y = f(x)$, entonces

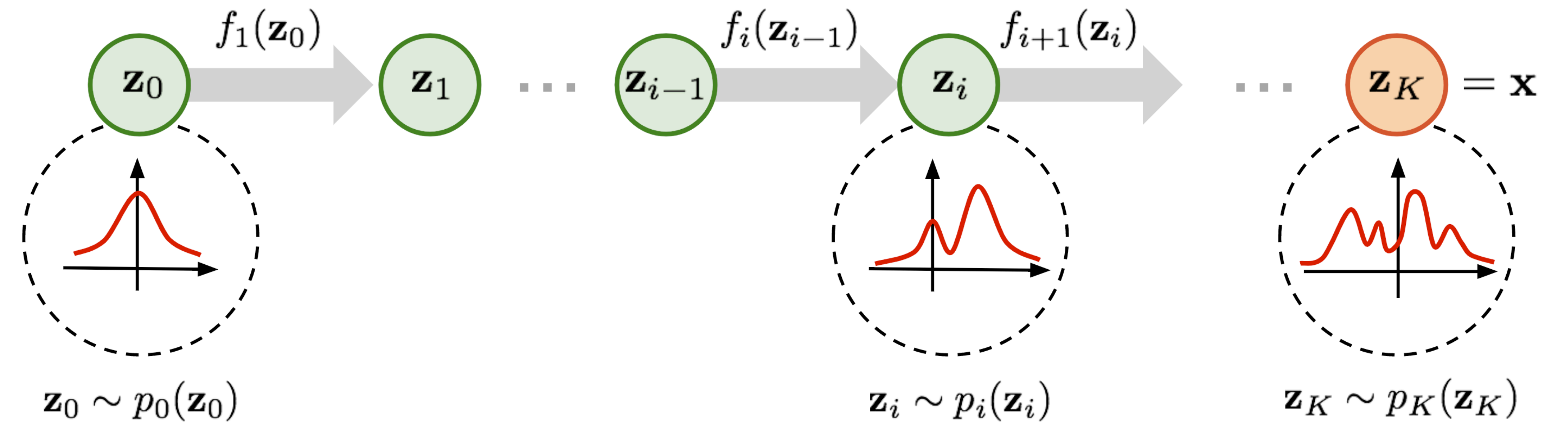
$$\frac{\partial f^{-1}(y)}{\partial y} = \left(\frac{\partial f(x)}{\partial x} \right)^{-1}$$

Fácil de ver 1D, válido para más dimensiones

$$\frac{\partial f^{-1}(y)}{\partial y} = \frac{\partial x}{\partial y} = \left(\frac{\partial y}{\partial x} \right)^{-1} = \left(\frac{\partial f(x)}{\partial x} \right)^{-1}$$

Normalizing Flows

Como transforma la densidad



$$\implies p_i(\vec{z}_i) = p_{i-1}(f_i^{-1}(\vec{z}_i)) \left| \det\left(\frac{\partial f_i(\vec{z}_{i-1})}{\partial \vec{z}_{i-1}} \right)^{-1} \right|$$

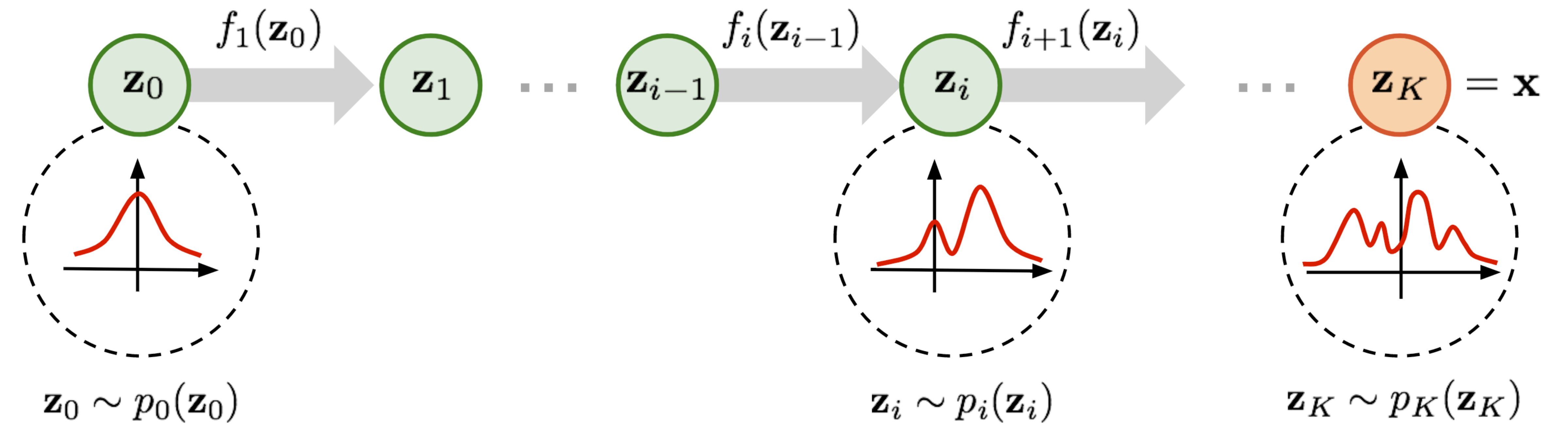
$$\det(M)\det(M^{-1}) = \det(M \cdot M^{-1}) = \det(I) = 1 \implies \det(M^{-1}) = (\det(M))^{-1}$$

$$\implies p_i(\vec{z}_i) = p_{i-1}(f_i^{-1}(\vec{z}_i)) \left| \det\left(\frac{\partial f_i(\vec{z}_{i-1})}{\partial \vec{z}_{i-1}} \right) \right|^{-1}$$

Transformación de la log probabilidad ante una función invertible

$$\log p_i(\vec{z}_i) = \log p_{i-1}(\vec{z}_{i-1}) - \log \left| \det \frac{\partial f_i(\vec{z}_{i-1})}{\partial \vec{z}_{i-1}} \right|$$

Normalizing Flows
Como transforma la densidad



Transformación de la log probabilidad ante K funciones invertibles

$$\vec{x} = \vec{z}_K = f_K \circ f_{K-1} \circ \cdots \circ f_1(\vec{z}_0)$$

$$\log p_X(\vec{x}) = \log\left(p_{Z_K}(\vec{z}_K)\right) = \log\left(p_{Z_{K-1}}(\vec{z}_{K-1})\right) - \log\left|\det \frac{\partial f_K(\vec{z}_{K-1})}{\partial \vec{z}_{K-1}}\right|$$

$$\log p_X(\vec{x}) = \log\left(p_0(\vec{z}_0)\right) - \sum_{i=1}^{K} \log\left|\det \frac{\partial f_i(\vec{z}_{i-1})}{\partial \vec{z}_{i-1}}\right|$$

Muestreo:

Tomo $\vec{z}_0 \sim p_0 = \mathcal{N}_D(0, \mathbb{I})$

$$\vec{x} = G(\vec{z}_0) = f_K \circ f_{K-1} \circ \cdots \circ f_1(\vec{z}_0)$$

Todo muy bien… pero a dónde entra el aprendizaje profundo?

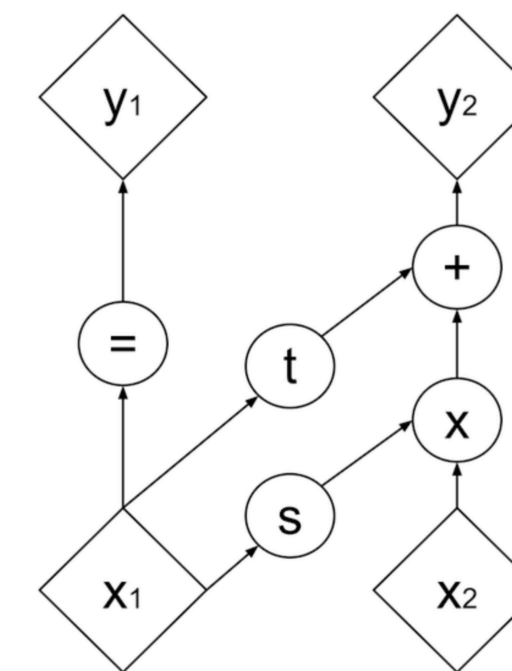
Normalizing Flows

Aprender transformaciones con NNs

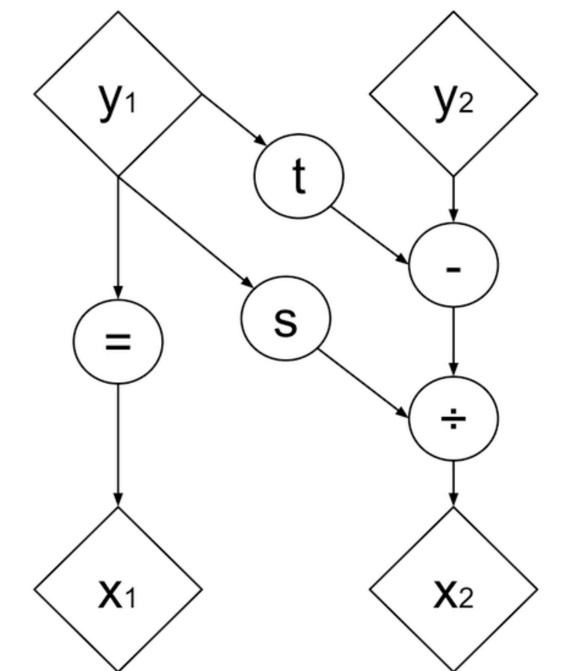
Queremos transformaciones expresivas flexibles, pero que sean invertibles.

Vimos que la transformación afin es invertible

$$z = ax + b$$



(a) Forward propagation



(b) Inverse propagation

Capa de acoplamiento afin (Coupling Layer)

Sean $s_\theta(x)$ y $t_\theta(x)$ redes neuronales parametrizadas por θ. Dividimos al vector de entrada x en dos partes: $\vec{x}_{1:d}$ y $\vec{x}_{(d+1):D}$.

$$\vec{z}_{1:d} = \vec{x}_{1:d}$$

scaling translation

Non-volume preserving (NVP)

$$\vec{z}_{(d+1):D} = \vec{x}_{(d+1):D} \odot \exp(s_\theta(\vec{x}_{1:d})) + t_\theta(\vec{x}_{1:d})$$

Normalizing Flows

Transformación de acoplamiento afin (Affine Coupling)

Transformación

scaling translation

$$\vec{z}_{1:d} = \vec{x}_{1:d}$$

$$\vec{z}_{(d+1):D} = \vec{x}_{(d+1):D} \odot \exp(s_\theta(\vec{x}_{1:d})) + t_\theta(\vec{x}_{1:d})$$

Resulta invertible gracias a que dividimos al vector en dos. Tenemos **memoria** de la entrada en la salida y las transformación depende de esa memoria.

Inversa

$$\vec{x}_{1:d} = \vec{z}_{1:d}$$

$$\vec{x}_{(d+1):D} = (\vec{z}_{(d+1):D} - t(\overrightarrow{y_{1:d}})) \odot \exp(-s(\vec{y}_{1:d}))$$

Jacobiano

$$\mathbf{J} = \begin{bmatrix} \mathbb{I}_d & 0 \\ \frac{\partial \vec{z}_{(d+1):D}}{\partial \vec{x}_{1:d}} & \text{diag}(\exp(s_\theta(\vec{x}_{1:d}))) \end{bmatrix}$$

Triangular inferior

Normalizing Flows

Transformación de acoplamiento afin

$$\mathbf{J} = \begin{bmatrix} \mathbb{I}_d & 0 \\ \frac{\partial \vec{y}_{(d+1):D}}{\partial \vec{x}_{1:d}} & \text{diag}(\exp(s_\theta(\vec{x}_{1:d}))) \end{bmatrix}$$

El determinante de una matriz triangular es el producto de los elementos diagonales!

$$\det \mathbf{J} = \exp(\sum_j s_\theta(\vec{x}_{1:d}))$$

$$\log p_X(\vec{x}) = \log\left(p_0(\vec{z}_0)\right) - \sum_{i=1}^{K} \log\left|\det \frac{\partial f_i(\vec{z}_{i-1})}{\partial \vec{z}_{i-1}}\right| \implies \log p_X(x) = \log p_0(\vec{z}_0) - \sum_{j=1}^{D-d} s_\theta(\vec{x}_{1:d})$$

Al controlar la estructura de la transformación, sigue siendo invertible, pero ganamos expresividad usando redes neuronales

Notebook Normalizing Flows - Parte 1

Normalizing Flows

Flow auto-regresivo mascarado (MAF)

Otra posibilidad sería modelar la densidad del vector $\vec{x}$ como el producto de densidades 1D condicionadas

$$p(\vec{x}) = \prod_{i=1}^{D} p(x_i | x_{1:i-1})$$

$$p(x_i | x_{1:i-1}) = \mathcal{N}(x_i | m_{i,\theta}(x_{1:i-1}), (\exp s_{i,\theta}(x_{1:i-1}))^2)$$

Cada dimensión se estandariza usando estadísticas condicionadas en las anteriores

$$z_i = (x_i - m_{i,\theta}(x_{1:i-1}))/\exp s_{i,\theta}(x_{1:i-1})$$

Estandarización autoregresiva

Inversa

$$x_i = z_i \exp s_{i,\theta}(x_{1:i-1}) + m_{i,\theta}(x_{1:i-1}),\ z_i \sim \mathcal{N}(0,1)$$

Normalizing Flows

Flow auto-regresivo mascarado (MAF)

Cómo sería el Jacobiano ante esta transformación?

$$z_i = (x_i - m_{i,\theta}(x_{1:i-1}))/\exp s_{i,\theta}(x_{1:i-1})$$

Esto define una transformación invertible con Jacobiano triangular!

$$\log|\det J| = -\sum_i s_{i,\theta}(x_{<i})$$

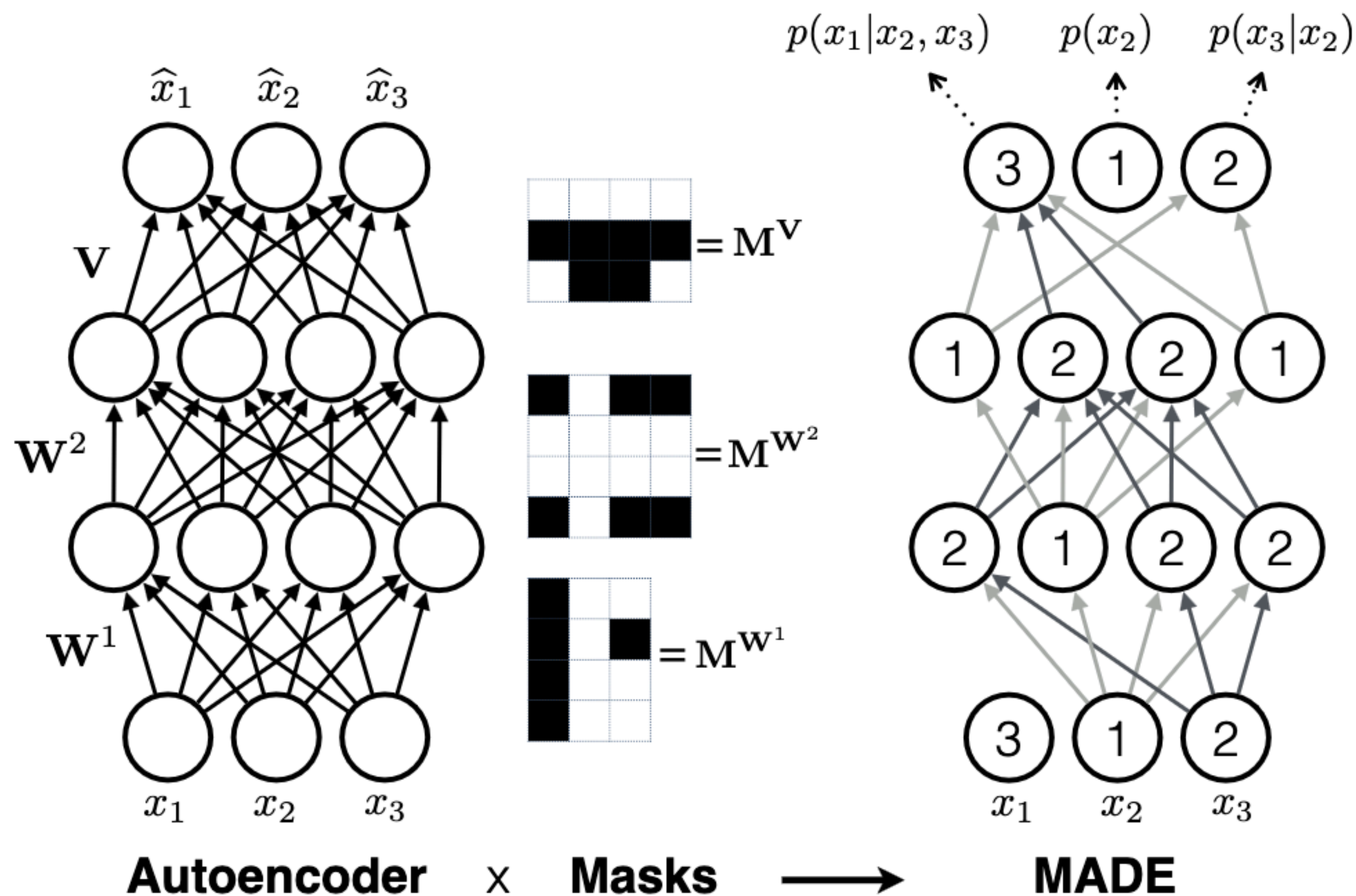
$$x_{1:i-1} = x_{<i}$$

$$\log p_X(x) = \log p_0(\vec{z}_0) + \sum_{i=1}^{D} s_{i,\theta}(\vec{x}_{<i})$$

Cómo armamos m_θ y s_θ para que cumplan con la dependencia condicional?

Normalizing Flows

Masked Autoencoder for Distribution Estimation (MADE)



Las máscaras fuerzan que cada salida i dependa de $x_{<i}$

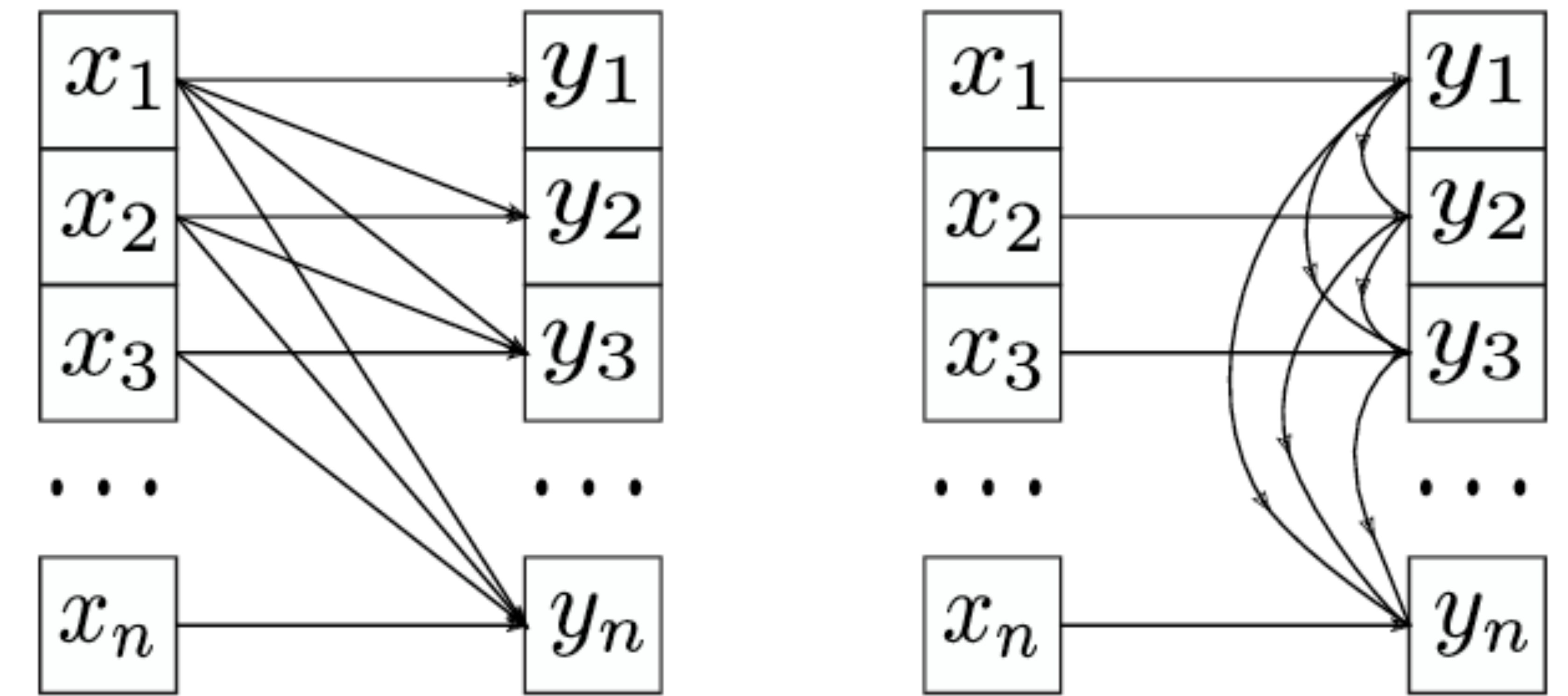
MAF combina una red autoregresiva (MADE) con el cambio de variable para construir un modelo generativo invertible

Super paralelizable!

https://arxiv.org/abs/1502.03509

Normalizing Flows

Inverse Autoregressive Flow (IAF)



MAF

Con MADE $\vec{x} \to (m_1, \ldots, m_D, s_1, \ldots, s_D)$

$$z_i = (x_i - m_{i,\theta}(x_{1:i-1}))/\exp s_{i,\theta}(x_{1:i-1})$$

Tengo todo, paralelizable! RAPIDO

Generación

$$\hat{x}_i = z_i \exp s_{i,\theta}(\hat{x}_{1:i-1}) + m_{i,\theta}(\hat{x}_{1:i-1}),\ z_i \sim \mathcal{N}(0,1)$$

Tengo que ir calculando cada $\hat{x}_{1:i-1}$ para $\hat{x}_i$

LENTO

IAF

$$z_i = (x_i - m_{i,\theta}(z_{1:i-1}))/\exp s_{i,\theta}(z_{1:i-1})$$

LENTO

Generación

$$\hat{x}_i = z_i \exp s_{i,\theta}(z_{1:i-1}) + m_{i,\theta}(z_{1:i-1}),\ z_i \sim \mathcal{N}(0,1)$$

RAPIDO

$$\log|\det J| = \sum_i s_{i,\theta}(z_{<i})$$

Notebook Normalizing Flows - Parte

Continuous Normalizing Flows

Superando limitaciones

- La arquitectura de los NFs tiene que ser invertible
- Debemos definir el número de capas

¿Por qué no pensamos en una transformación continua?

Introducimos un tiempo artificial t tal que a $t = 0$ estemos en la a priori y a tiempo $t = 1$ lleguemos a la distribución objetivo

Neural ODE

$$\frac{\partial x(t)}{\partial t} = f_\theta(x(t), t)$$

Ecuación diferencial ordinaria! (ODE)

Ahora tenemos una única red f_θ que la evaluamos en diferentes puntos en el intervalo de tiempo $t \in [0,1]$

Continuous Normalizing Flows

Jacobiano

Discreto

$$\vec{x} = \vec{z}_K = f_K \circ f_{K-1} \circ \cdots \circ f_1(\vec{z}_0)$$

Mide cambio de volumen finito global

$$\log p_X(\vec{x}) = \log\left(p_{Z_K}(\vec{z}_K)\right) = \log\left(p_{Z_{K-1}}(\vec{z}_{K-1})\right) - \log\left|\det \frac{\partial f_K(\vec{z}_{K-1})}{\partial \vec{z}_{K-1}}\right|$$

$$\log p_X(\vec{x}) = \log\left(p_0(\vec{z}_0)\right) - \sum_{i=1}^{K} \log\left|\det \frac{\partial f_i(\vec{z}_{i-1})}{\partial \vec{z}_{i-1}}\right|$$

Continuo

$$\frac{\partial x(t)}{\partial t} = f_\theta(x(t), t)$$

(Usando ecuación de continuidad y conservación de probabilidad)

Método adjunto para BP

$$\frac{d}{dt}\log p(x(t)) = -Tr\left(\frac{\partial f_\theta(x(t), t)}{\partial x}\right) \implies \log p(x(T)) = \log p(x(0)) - \int_0^T Tr\left(\frac{\partial f_\theta(x(t), t)}{\partial x(t)}\right) dt$$

divergencia

Para obtener la transformación necesitamos resolver esta ODE numéricamente

Continuous Normalizing Flows

Jacobiano

Muchas transformaciones infinitesimales $x(t+dt) = x(t) + f(x(t), t)dt$

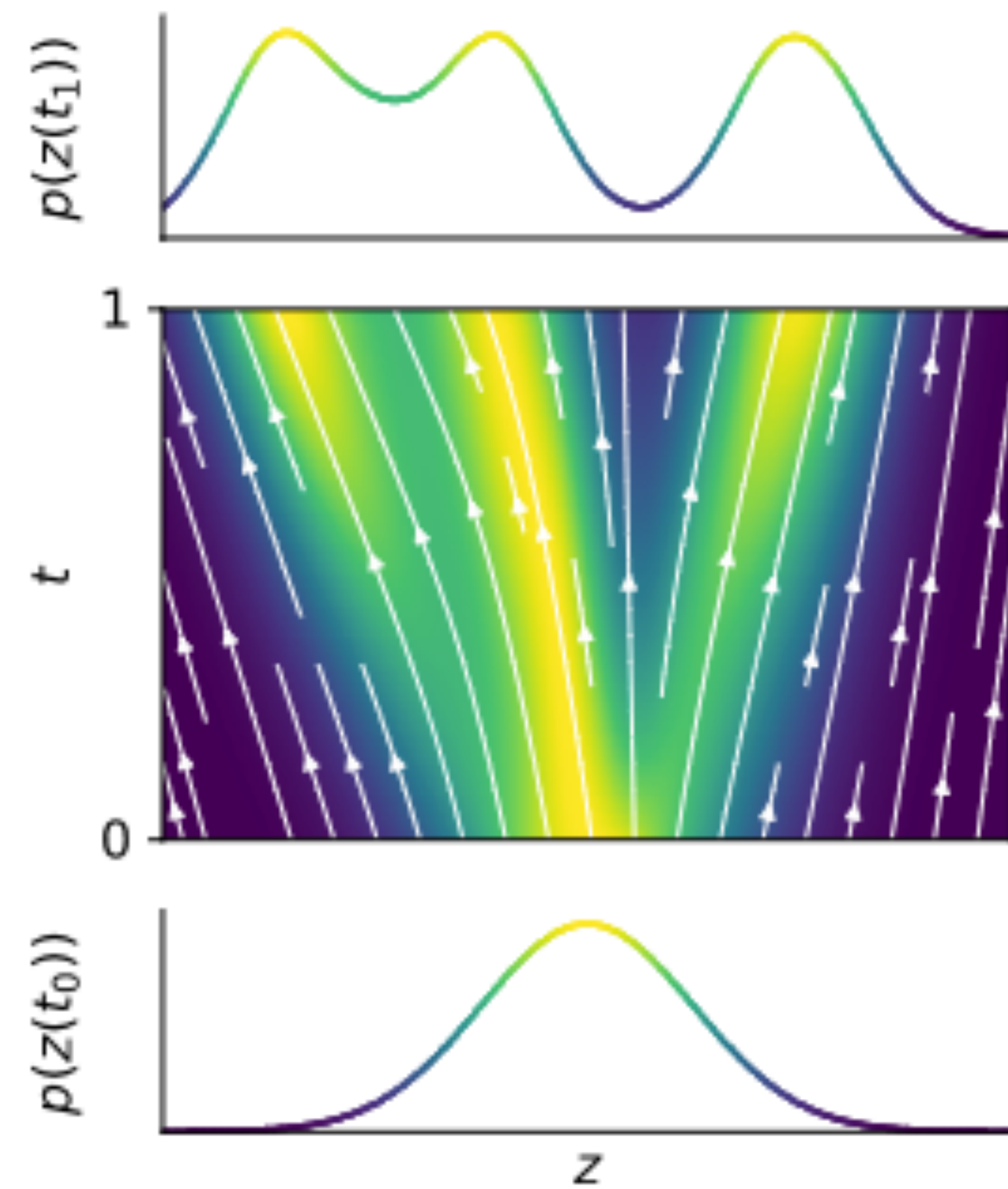
$$J = I + \frac{\partial f}{\partial x} dt$$

Usando que $\det(I + Adt) \approx 1 + Tr(A)dt$ (expansion a primer orden)

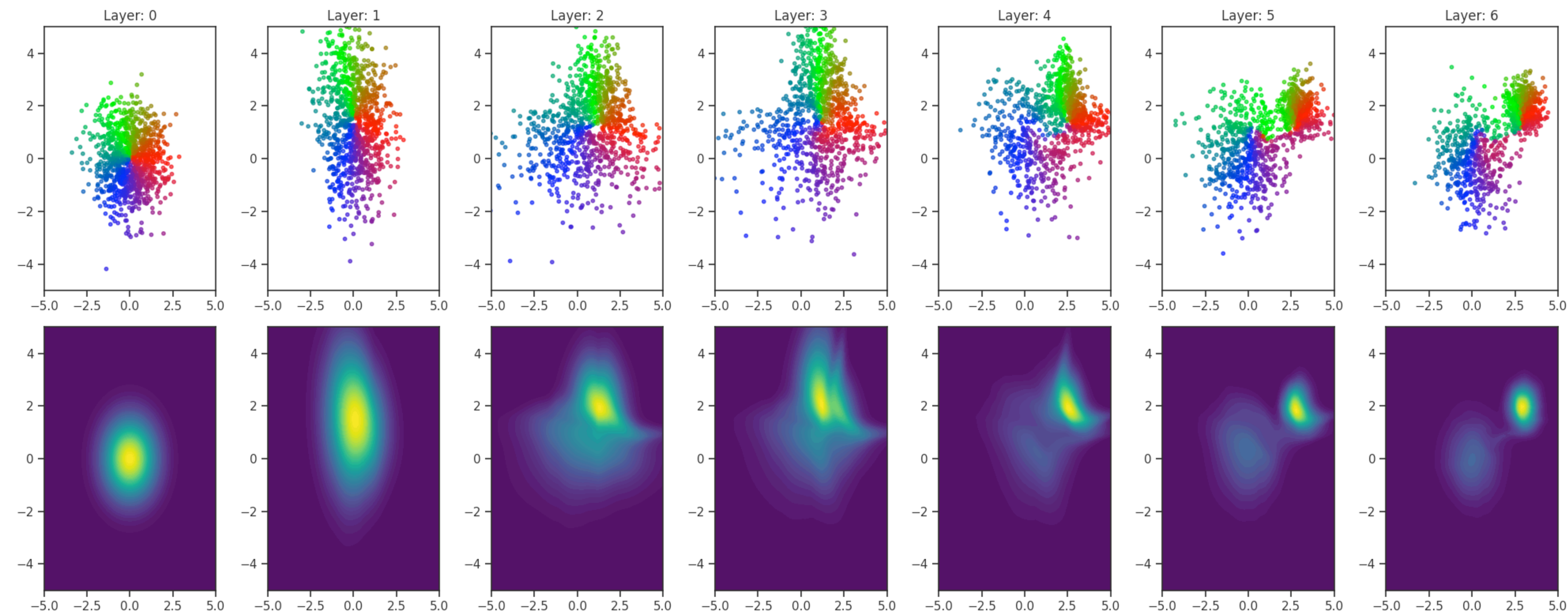
$$\log \det(J) \approx Tr\left(\frac{\partial f}{\partial x}\right) dt$$

$$\frac{d}{dt} \log p(x(t)) = -Tr\left(\frac{\partial f_\theta(x(t), t)}{\partial x}\right) \implies \log p(x(T)) = \log p(x(0)) - \int_0^T Tr\left(\frac{\partial f_\theta(x(t), t)}{\partial x(t)}\right) dt$$

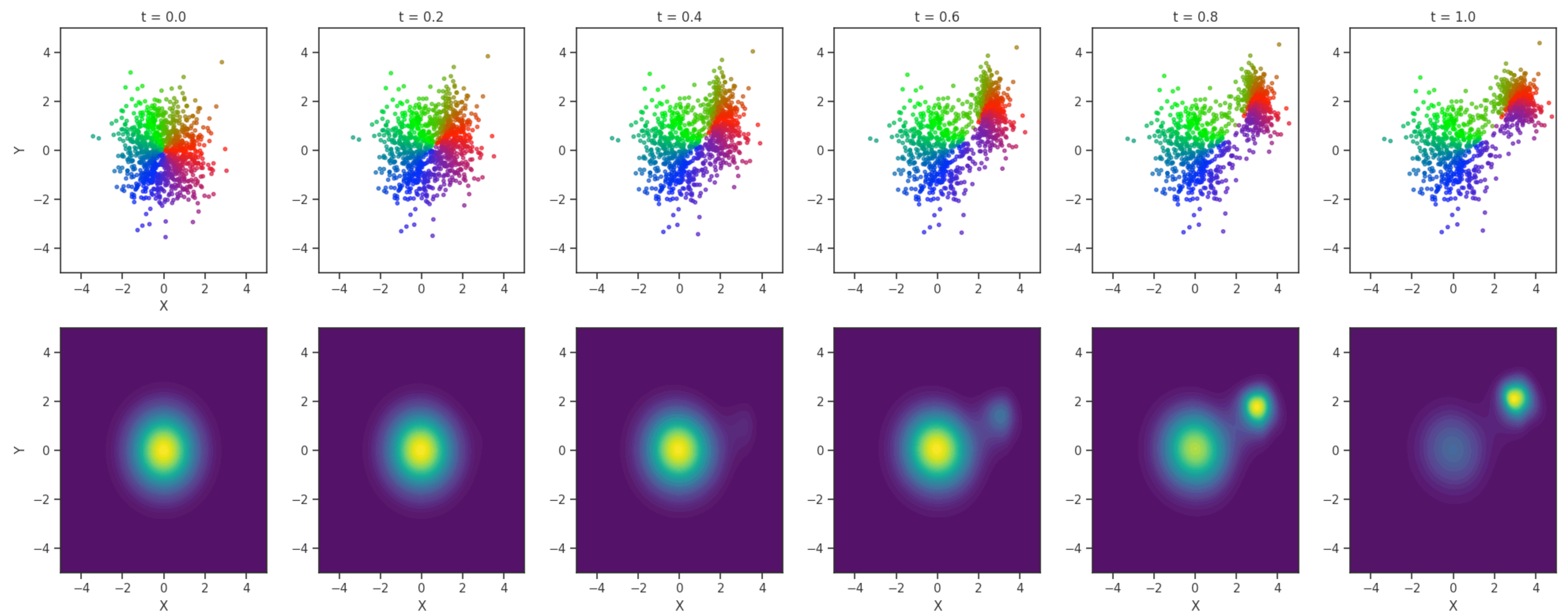
Normalizing Flows



Discreto



Continuo



La invertibilidad viene gratis

$$x(0) \rightarrow x(T)$$

$$x(T) \rightarrow x(0)$$

https://arxiv.org/pdf/1810.01367

Notebook Normalizing Flows - Parte 3

Decuantización

Distribuciones discretas (imágenes)

- Hemos visto que los NFs operan en distribuciones continuas
- Muchos datos tienen carácter discreto.
- Podemos **convertir** datos discretos en continuos agregándole un poco de ruido

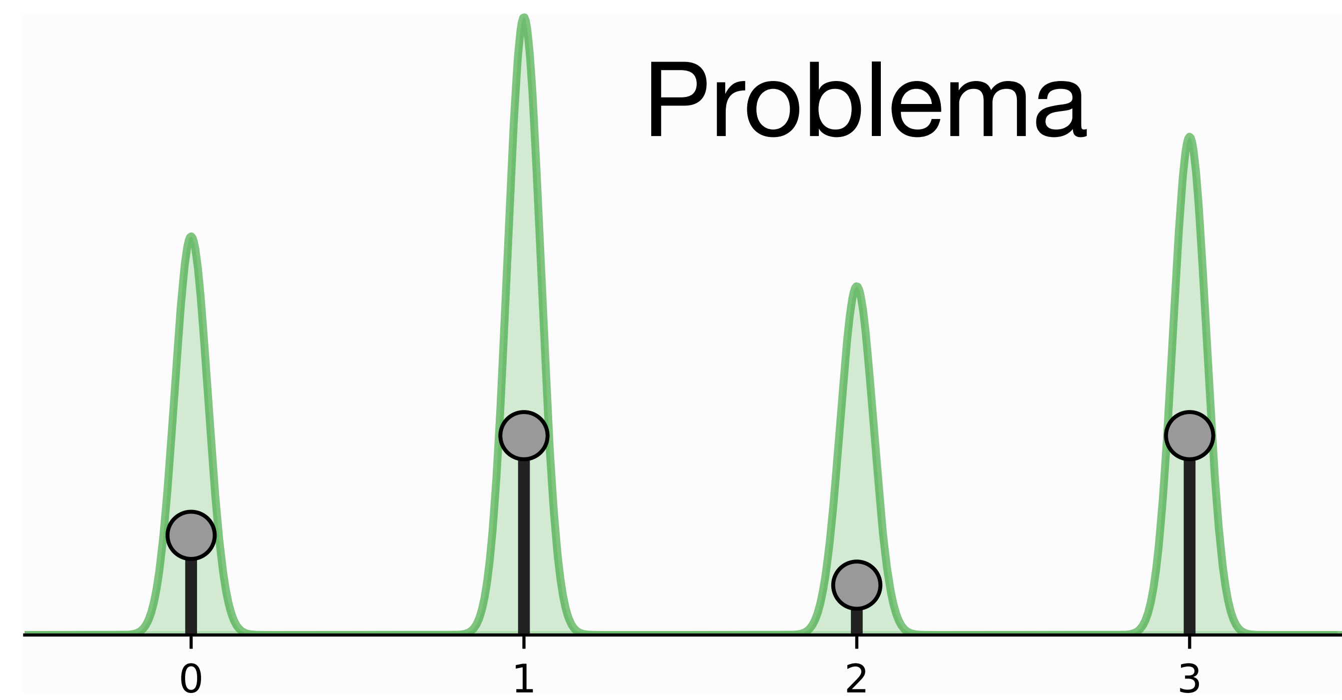
1. $v = x + u,\ u \sim U[0,1]$, donde x son los datos discretos originales

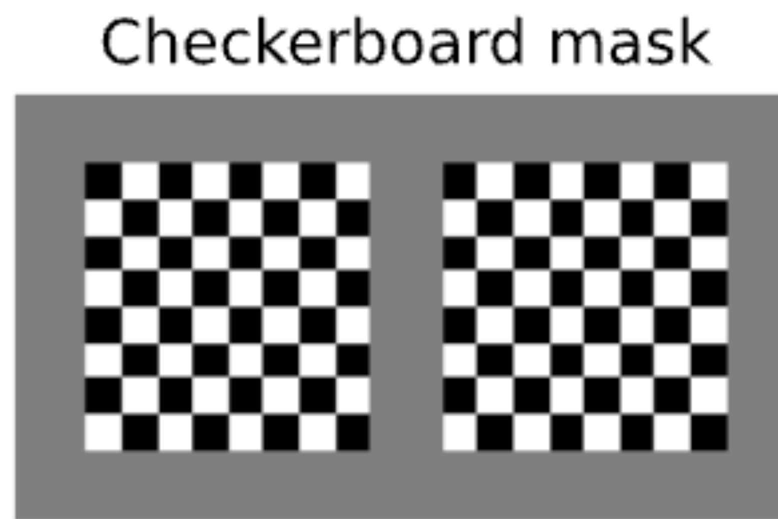
$$p(x) = \int p(x+u)du = \int \frac{q(u|x)}{q(u|x)} p(x+u)du = E_{u \sim q(u|x)} \frac{p(x+u)}{q(u|x)} = E_{u \sim U[0,1]^d} p(x+u)$$

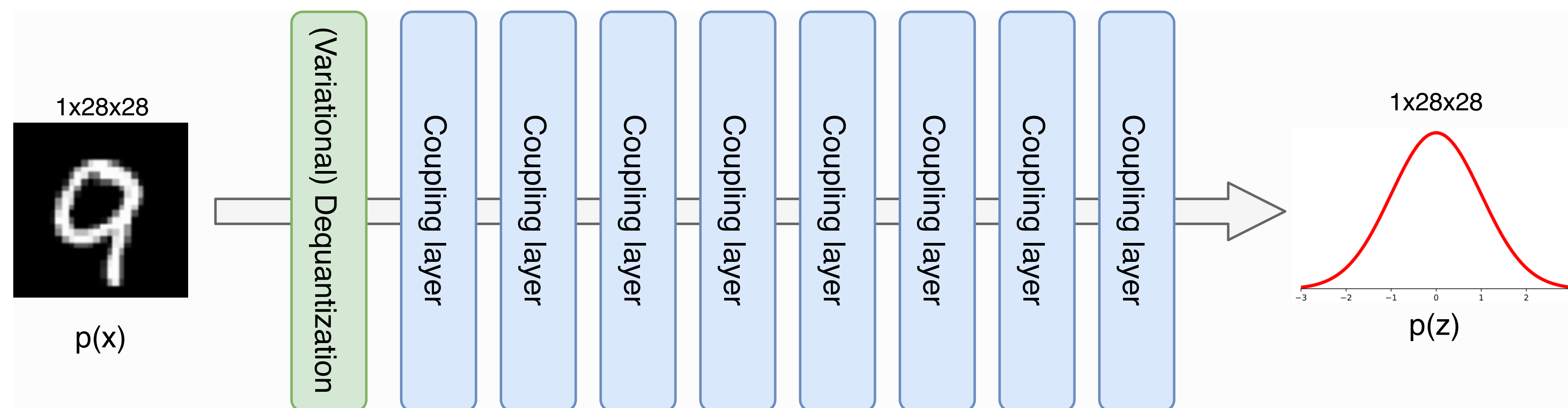
2. Decuantización Variacional: utilizamos unadistribución aprendible $q_\theta(u|x)$, modelada por un flujo normalizado adicional!

Decuantización

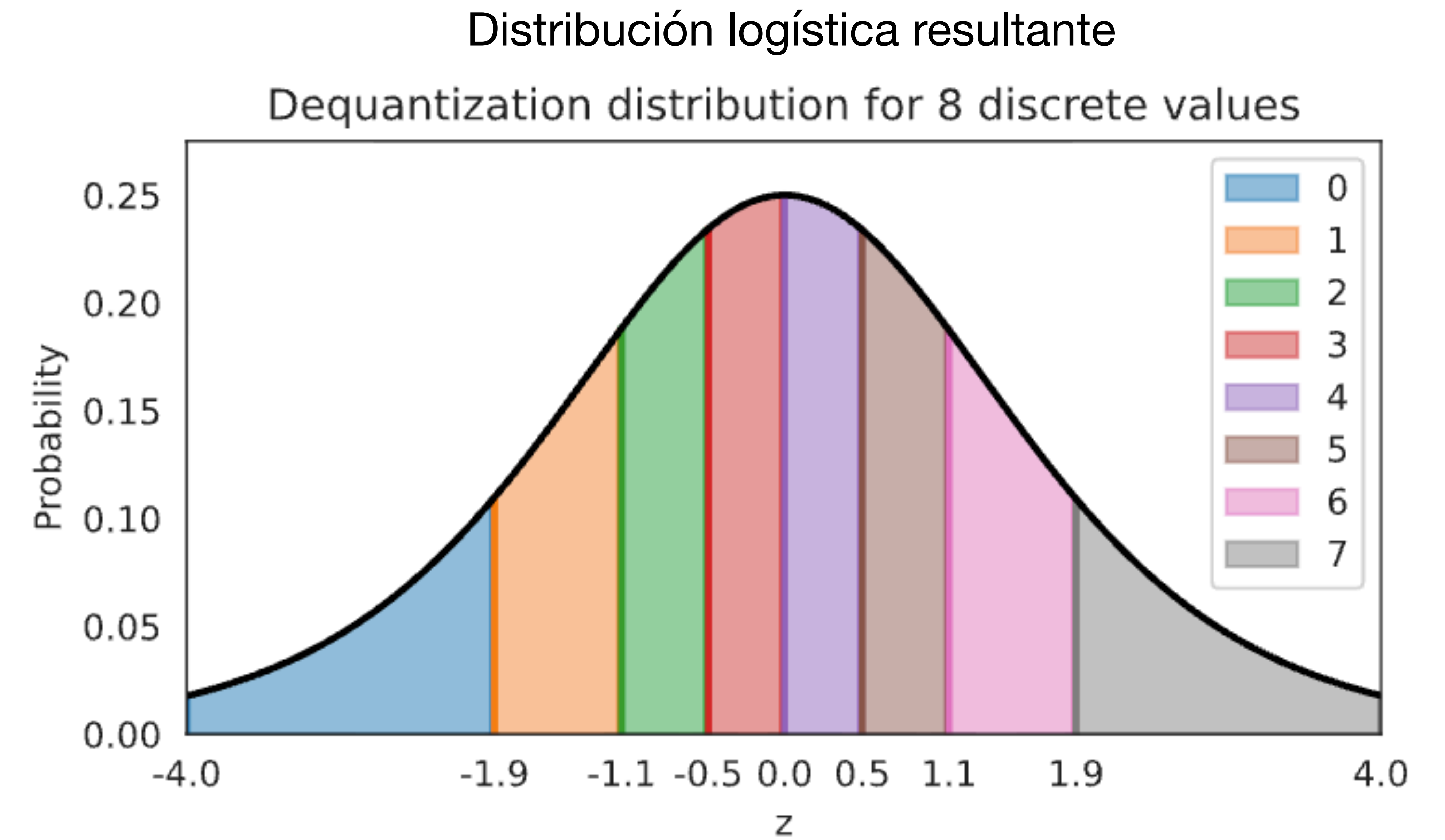
Ejemplo

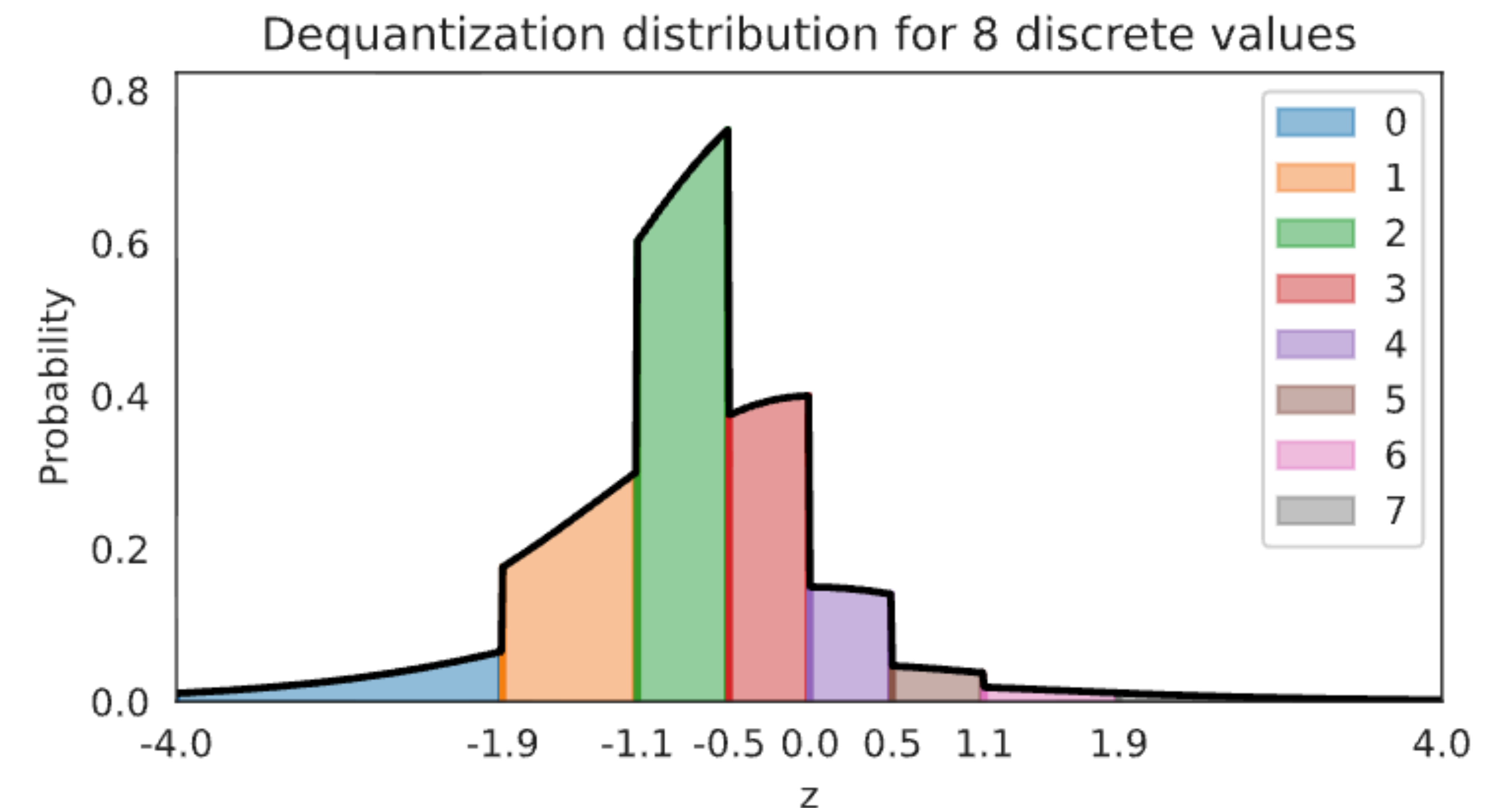






Distribución logística resultante





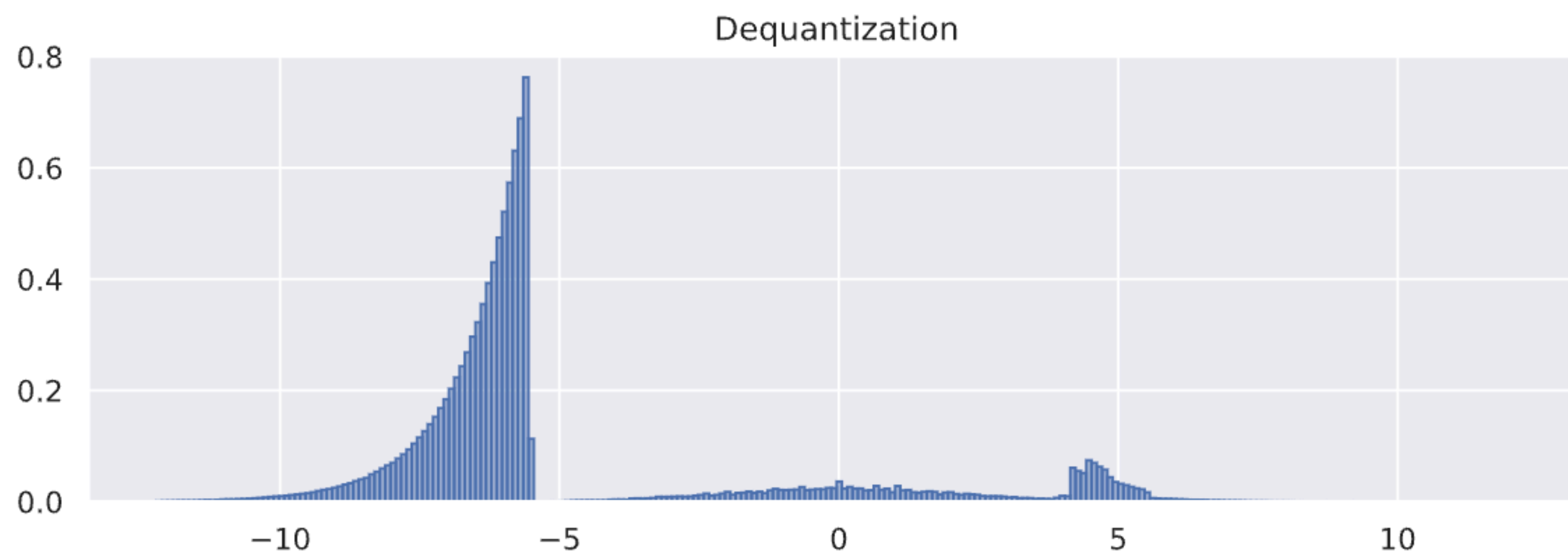
Qué sucede si no es uniforme

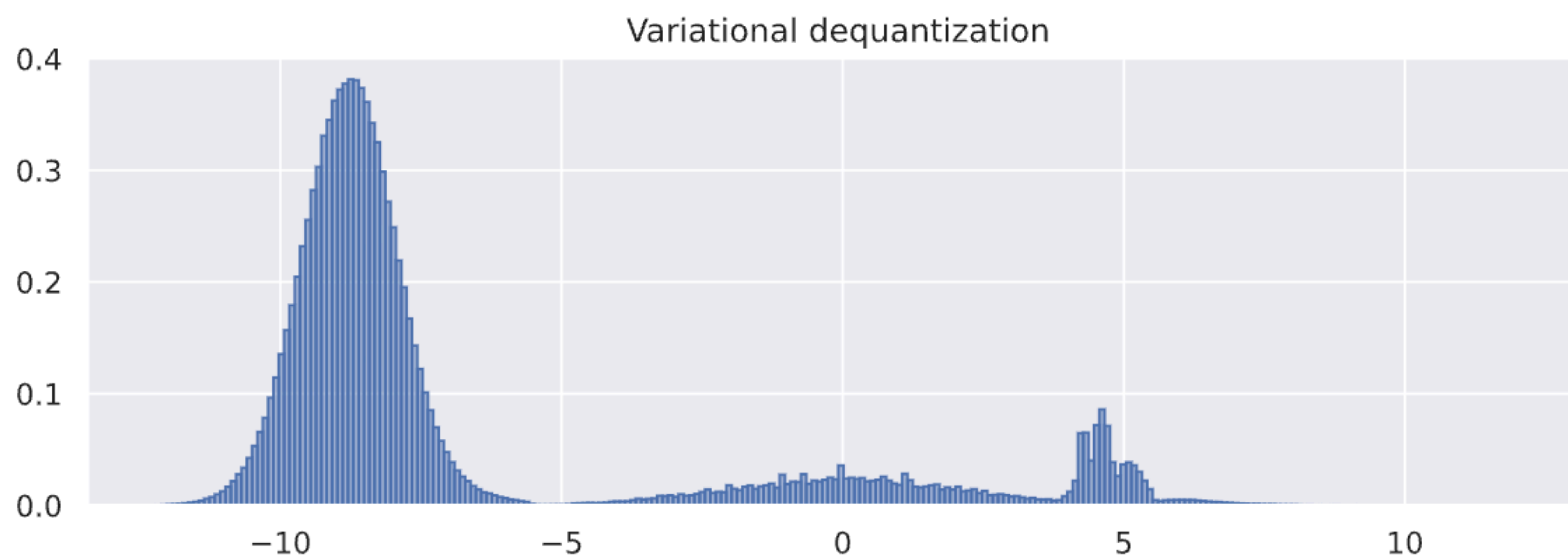
$$z = (x + u)/256,\ u \sim U[0,1] \implies z \in [0,1]$$

$$logit(z) = \log(z/(z-1)) \in \mathbb{R}$$

Decuantización

Ejemplo





Modelo de Difusión probabilístico

Denoising Diffusion Probabilstic Model (DDPM)

Modelo generativo de variable latente en dos fases **forward** y **reverse.**

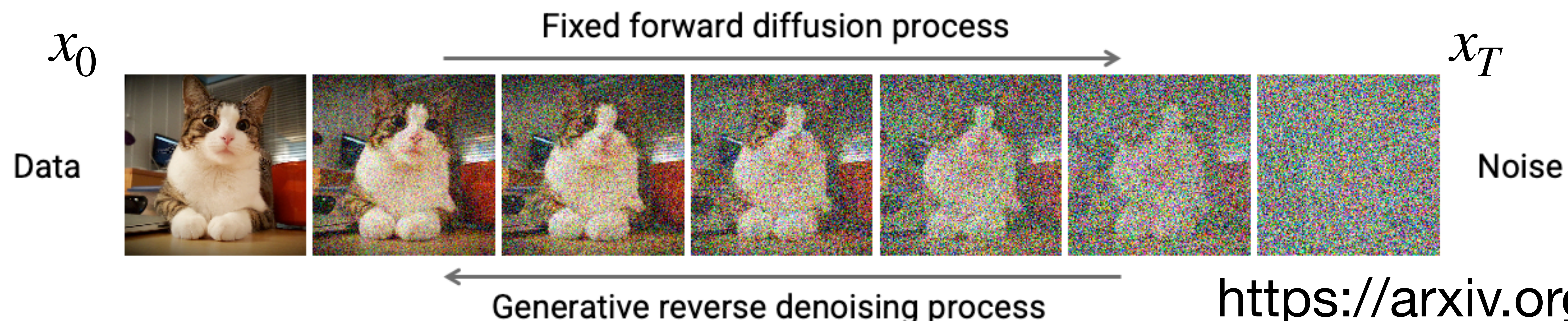
Forward: se define una cadena de Markov que va agregándole ruido en pasos pequeños

$$q(x_t | x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t} x_{t-1}, \beta_t I)$$

Con β_t una progresión de ruido pequeño.

$$x_t = \sqrt{1-\beta_t} x_{t-1} + \sqrt{\beta_t} \epsilon_t,\ \epsilon_t \sim \mathcal{N}(0, I)$$

- Cada paso hace un poco más borrosa la muestra
- Después de muchos pasos, terminas con puro ruido



https://arxiv.org/pdf/2006.11239

Modelo de Difusión probabilístico

Denoising Diffusion Probabilstic Model (DDPM)

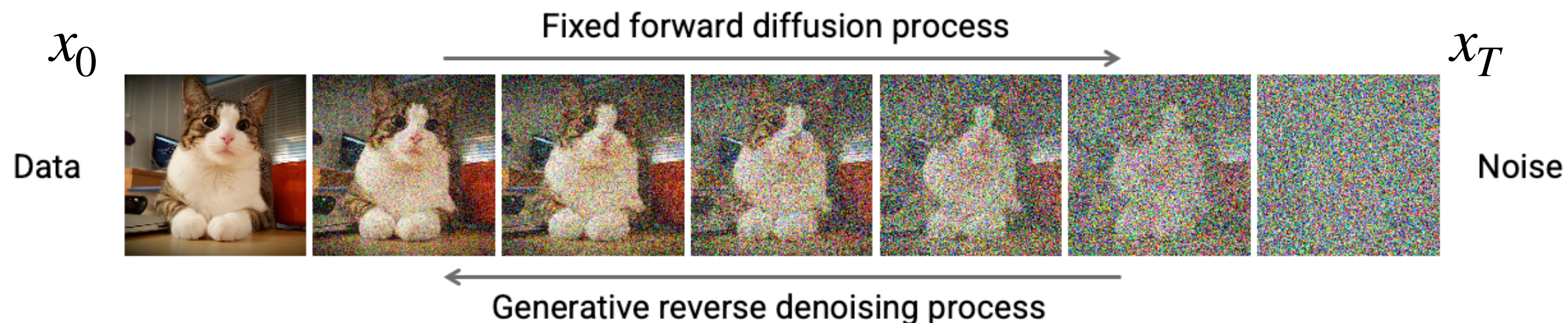
Esta cadena que lleva de $t = 0$ a $t = T$ se puede colapsar en un solo paso

$$q(x_T | x_0) = \mathcal{N}(x_T; \sqrt{\bar{\alpha}_T} x_0, (1 - \bar{\alpha}_T) I)$$

con $\bar{\alpha}_T = \prod_{t=1}^{T} (1 - \beta_t)$.

$$x_T = \sqrt{\bar{\alpha}_T} x_0 + \sqrt{1 - \bar{\alpha}_T} \epsilon, \ \epsilon \sim \mathcal{N}(0, I)$$

Permite samplear x_T directamente desde x_0 sin simular toda la cadena



Modelo de Difusión probabilístico

Invertir el proceso

Ahora queremos modelar $p_\theta(x_{t-1} | x_t)$. El modelo predice **el ruido** que se usó para generar x_T! Entrenamos red $\epsilon_\theta(x_t, t)$ tal que

$$\epsilon_\theta(x_t, t) \approx \epsilon$$

$$\nabla_x \log p_t(x) = -\frac{1}{\sqrt{1-\bar{\alpha}_t}}\epsilon$$

Predecir ϵ equivale a estimar el gradiente de la densidad

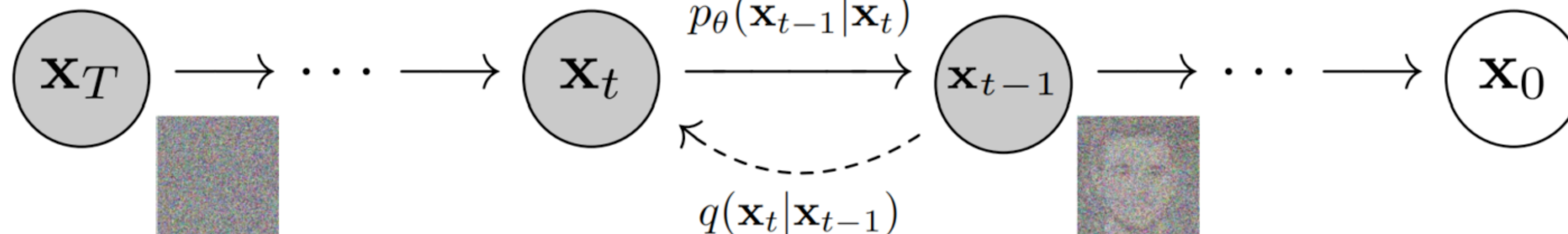
Donde ϵ es el ruido real utilizado para obtener x_t. El entrenamiento se reduce ahora a la función de costo MSE sobre ruido gaussiano

$$\mathbb{E}_{x_0,t,\epsilon}\left[|| \epsilon - \epsilon_\theta(x_t, t) ||^2 \right]$$

$x_T \sim \mathcal{N}(0,I)$

$x_{t-1} | x_t \sim \mathcal{N}(\mu_\theta(x_t), \beta_t I)$

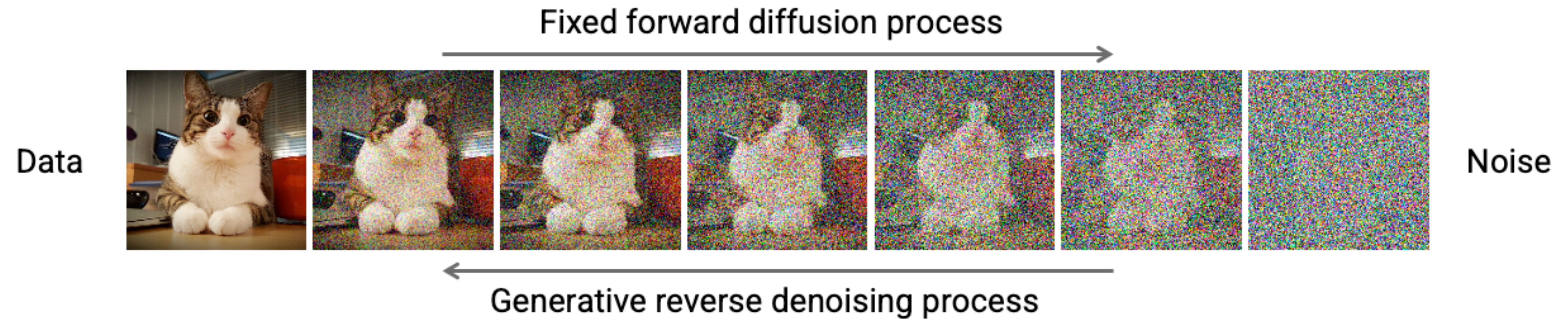
$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}}(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t))$$



Notebook Difusion

Modelo de Difusión probabilístico

Conexión con CNF



En CNFs aprendimos un campo vectorial global $f_\theta(x, t)$ y su dinámica $\frac{dx}{dt} = f_\theta(x, t)$

En DDPM aprendemos un campo que apunta hacia regiones de alta densidad. Se le llama score $s_\theta(x, t) \approx \nabla_x \log p_t(x)$ (score matching). En el límite continuo $T \to \infty$, pasos muy pequeños, tiempo $t \in [0,1]$, veremos que es una formulación análoga a neural ODE, pero con ecuación diferencial estocásticas (SDE).

$$dx = f(x, t)dt + g(t)dW_t$$

$$f_\theta(x, t) = -\frac{1}{2}g(t)^2 s_\theta(x, t)$$

$$g(t) = \sqrt{\beta}_t$$

Modelo de Difusión probabilístico

Conexión con CNF

$$x_t = \sqrt{1-\beta_t}x_{t-1} + \sqrt{\beta_t}\epsilon_t$$

Usamos desarrollo de Taylor $\sqrt{1-\beta_t} \approx 1 - \frac{1}{2}\beta_t$

$$\implies x_t \approx x_{t-1} - \frac{1}{2}\beta_t x_{t-1} + \sqrt{\beta_t}\epsilon_t$$

Que, en el limite diferencial $\Delta t \to dt,\ \epsilon \to dW_t$

$$s_\theta(x,t) \approx \nabla_x \log p_t(x)$$

$$dx = -\frac{1}{2}\beta(t)xdt + \sqrt{\beta(t)}dW_t$$

Ecuación Diferencial estocástica (SDE)

Esta SDE es el límite continuo del DDPM

Probability Flow ODE (tipo CNF)

Equivalente determinista

SDE $\quad dx = f(x,t)dt + g(t)dW_t \implies dx = [f(x,t) - \frac{1}{2}g(t)^2 s_\theta(x,t)]dt$

El ruido se remplaza por información de densidad (Focker-Planck)

Redes Generativas Adversariales

Generative Adversarial Networks (GANs)

Ian Goodfellow (2014)

Supongamos que tenemos NN parametrizada por ϕ que recibe z, muestreado de una distribución conocida, y genera una salida $\tilde{x}$. Llamamos a esta red un **generador** (G)

$$\tilde{x} = G_\phi(z)$$

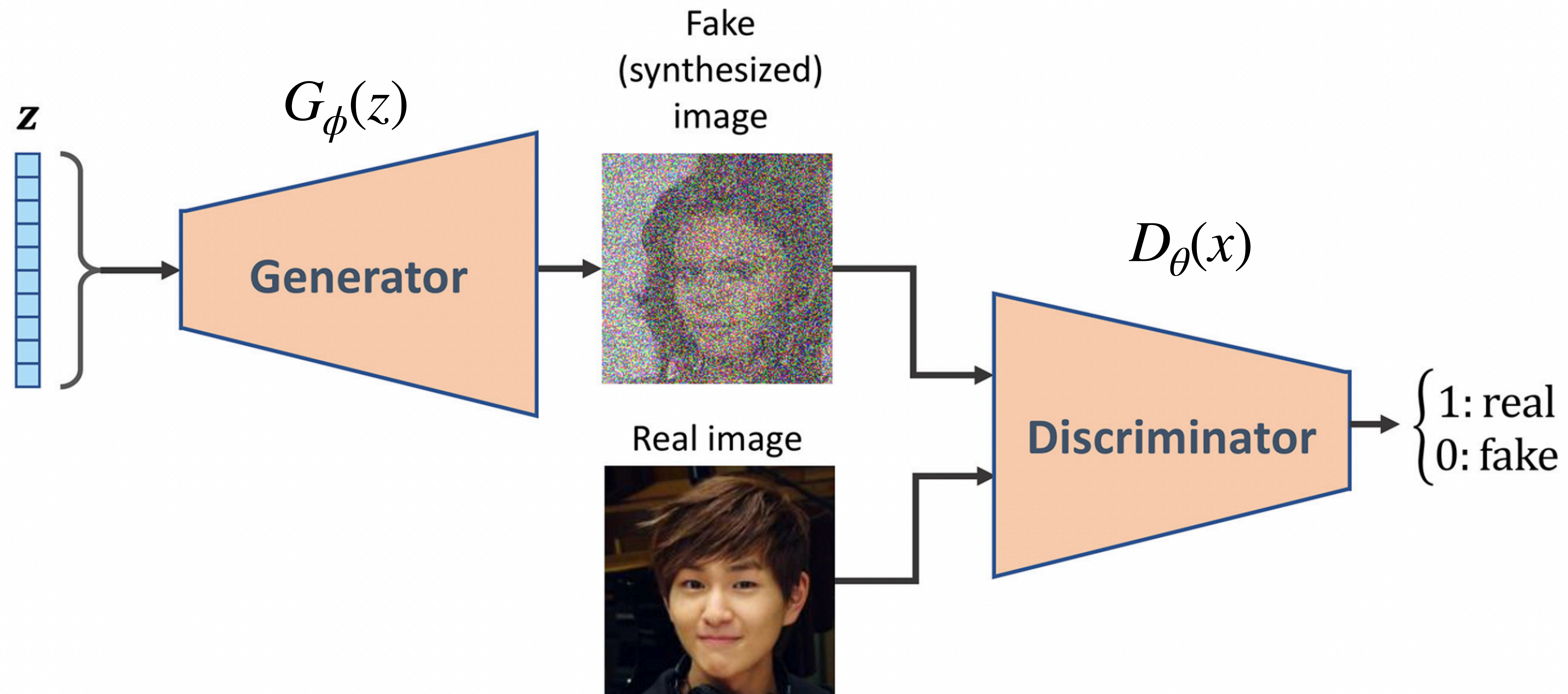
Supongamos que existe una función *evaluadora* que es capaz de evaluar la calidad de la salida generada $\tilde{x}$ en base a datos reales x. Con esta función, podríamos ajustar los pesos de la red G de manera tal de aumentar la calidad de $\tilde{x}$. Existe una función universal para esto?

Agregamos **otra** NN parametrizada por θ, llamada **discriminador** (D), que aprende a distinguir la salida generada $\tilde{x}$ de los datos reales x.

https://arxiv.org/abs/1406.2661

Redes Generativas Adversariales

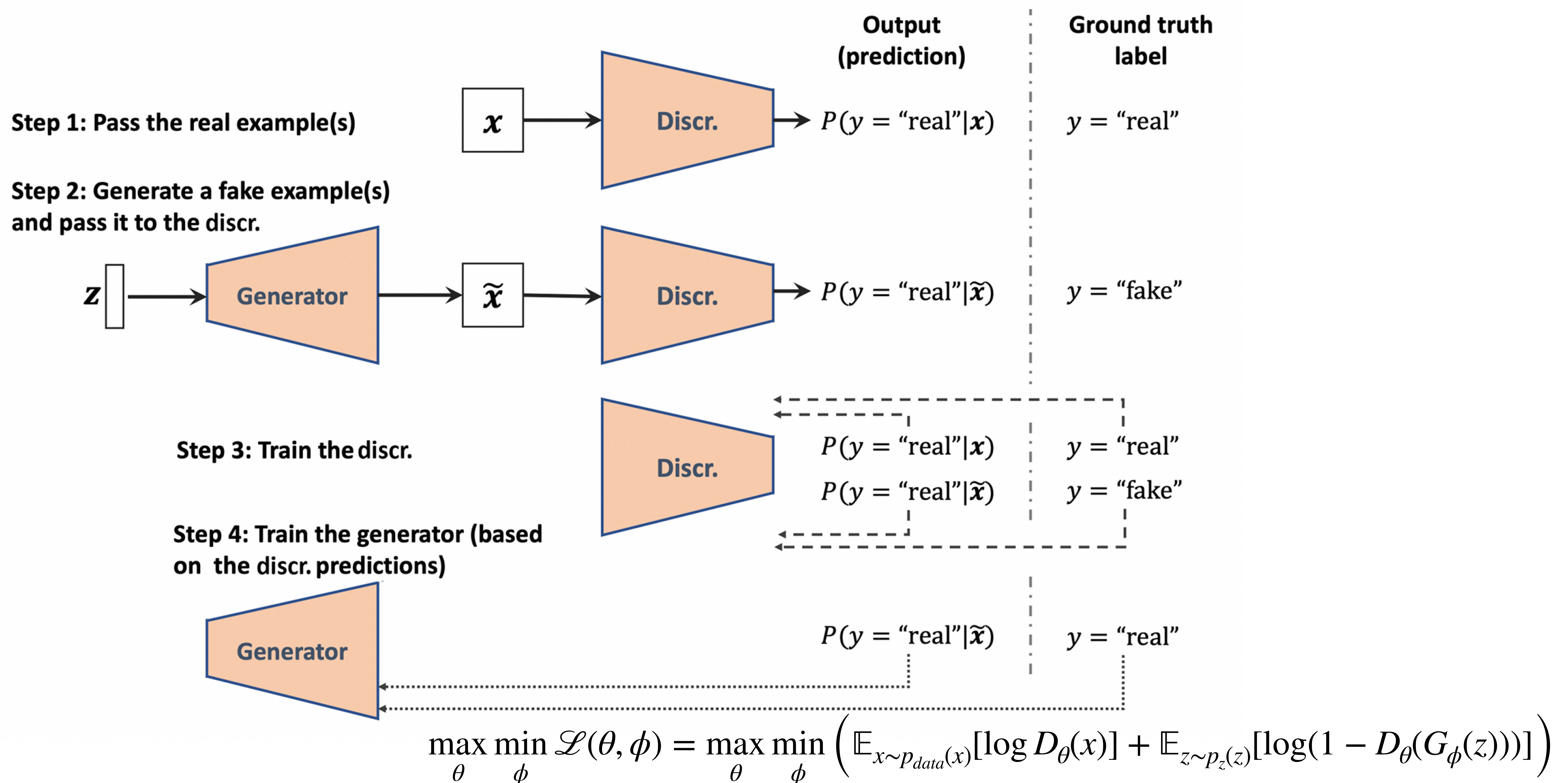
Generative Adversarial Networks (GANs)



$$\max_\theta \min_\phi \mathscr{L}(\theta, \phi) = \max_\theta \min_\phi \left(\mathbb{E}_{x \sim p_{data}(x)}[\log D_\theta(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D_\theta(G_\phi(z)))] \right)$$

Redes Generativas Adversariales

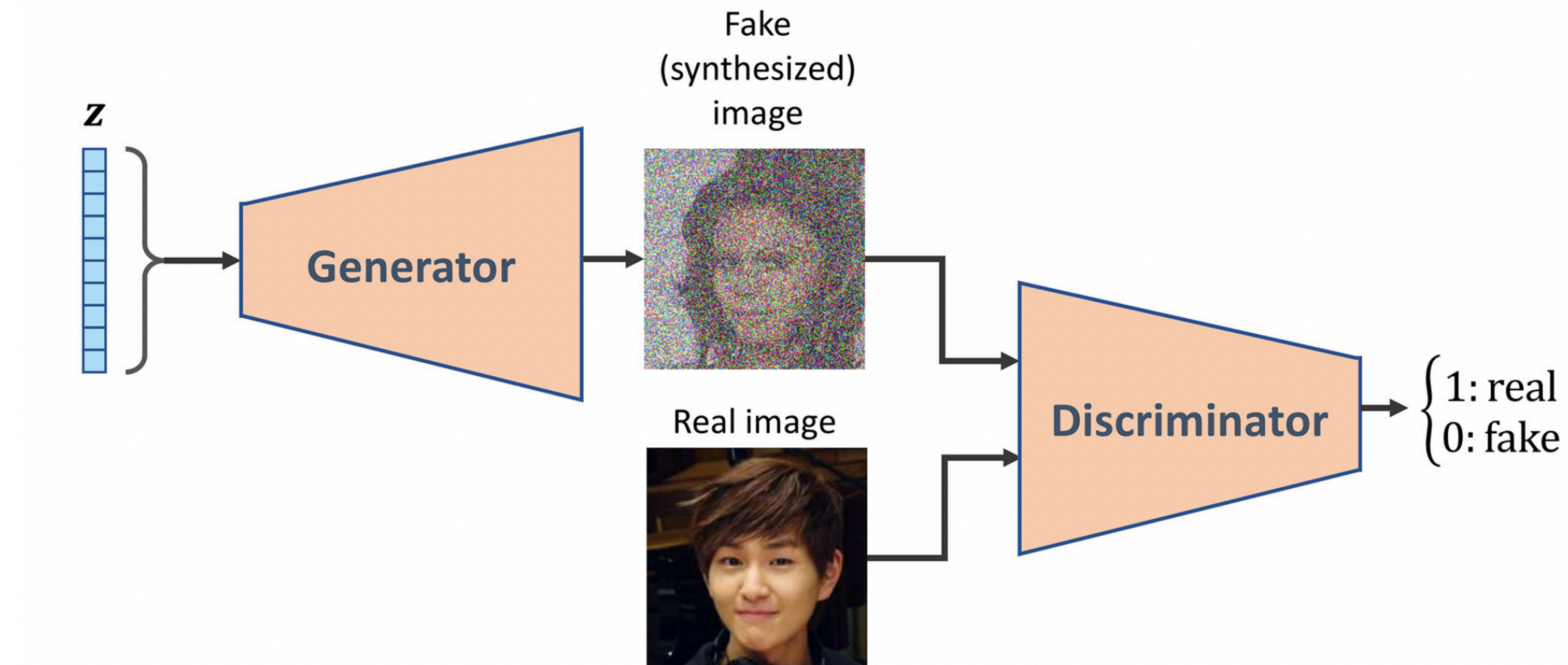
Generative Adversarial Networks (GANs)

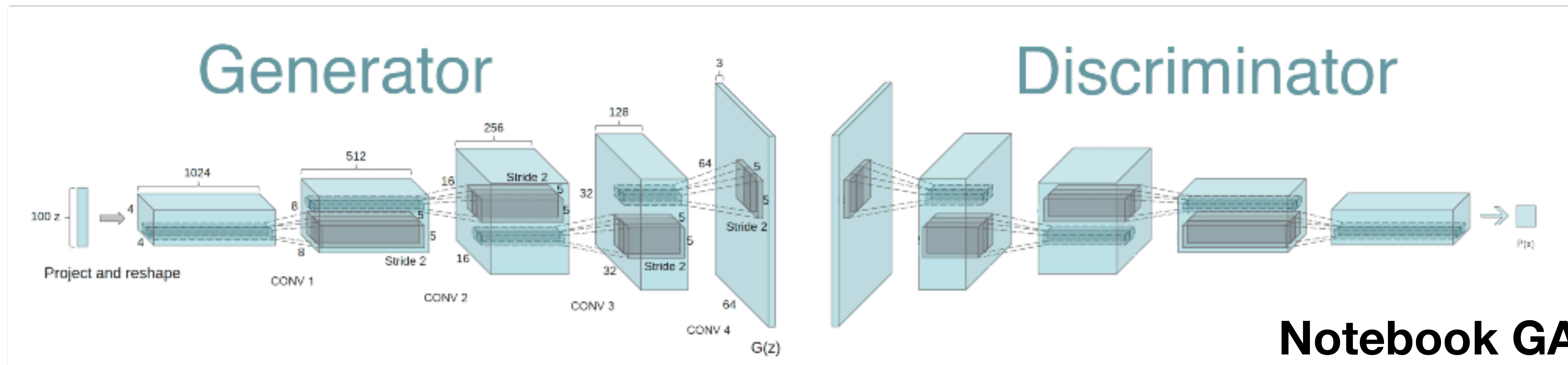


$$\max_{\theta} \min_{\phi} \mathscr{L}(\theta, \phi) = \max_{\theta} \min_{\phi} \left(\mathbb{E}_{x \sim p_{data}(x)}[\log D_\theta(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D_\theta(G_\phi(z)))] \right)$$

Redes Generativas Adversariales

Deep Convolutional Generative Adversarial Networks (DC-GANs)





Notebook GANs