UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

> Reinventa el mundo <

Sesión 5.2

Flow-based generative models

Glow, CNF, Flow Matching

TRANSFORMATEC

UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

1.

Flow-based *Models*

TRANSFORMATEC

Normalizing *Flows (NF)*

Normalizing Flow es una transformación invertible y diferenciable que permite mapear una variable aleatoria z con una distribución conocida $p_z(z)$ a otra variable aleatoria x con una distribución más compleja $p_x(x)$, mediante una función determinística $f: \mathbb{R}^d \rightarrow \mathbb{R}^d$:

$$x = f(z)$$

Dado que f es invertible, podemos expresar la variable latente en términos de la variable observada: $x = f^{-1}(z)$

Normalizing *Flows (NF)*

Normalizing Flow es una transformación invertible y diferenciable que permite mapear una variable aleatoria z con una distribución conocida $p_z(z)$ a otra variable aleatoria x con una distribución más compleja $p_x(x)$, mediante una función determinística $f: \mathbb{R}^d \rightarrow \mathbb{R}^d$:

$$x = f(z)$$

Dado que f es invertible, podemos expresar la variable latente en términos de la variable observada: $x = f^{-1}(z)$

Para calcular la densidad de probabilidad de x, usamos la regla del cambio de variable en distribuciones:

$$p_x(x) = p_z(z)\left|\det J_f\,(z)\right|^{-1}$$

donde:

- $p_z(z)$ es la densidad de la variable latente z.
- $J_f(z) = \frac{\partial f}{\partial z}$ es la matriz Jacobiana de la transformación f.
- $\det J_f\,(z)$ es el determinante del Jacobiano, que ajusta la densidad para reflejar el cambio de volumen bajo la transformación.

Es importante que $\boldsymbol{f}$ **sea invertible** porque garantiza que el **Jacobiano** $\boldsymbol{J_f(z)}$ **sea invertible**.

Normalizing *Flows (NF)*

Normalizing Flow es una transformación invertible y diferenciable que permite mapear una variable aleatoria z con una distribución conocida $p_z(z)$ a otra variable aleatoria x con una distribución más compleja $p_x(x)$, mediante una función determinística $f: \mathbb{R}^d \to \mathbb{R}^d$:

$$x = f(z)$$

Dado que f es invertible, podemos expresar la variable latente en términos de la variable observada: $x = f^{-1}(z)$

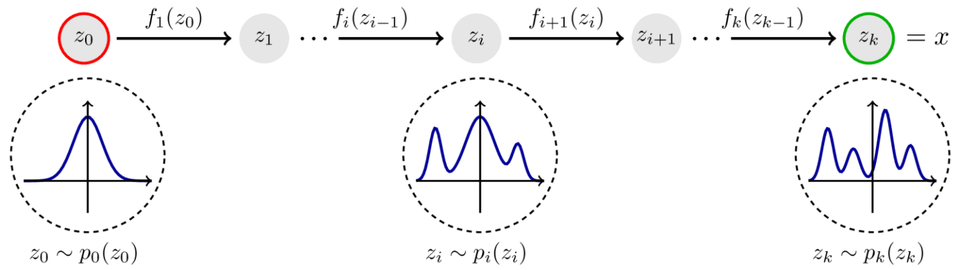
En términos generales, Normalizing Flow define una secuencia de K transformaciones $f_1, f_2, \dots, f_K$, de modo que:

$$z_0 \sim p_{z_0}(z_0), \quad z_k = f_k(z_{k-1}) \quad \text{para} \quad k = 1,2,\dots,K$$

La densidad final se obtiene mediante la cascada de cambios de variable:

$$p_x(x) = p_{z_0}(z_0) \prod_{k=1}^{K} \left|\det J_{f_k}(z_{k-1})\right|^{-1}$$

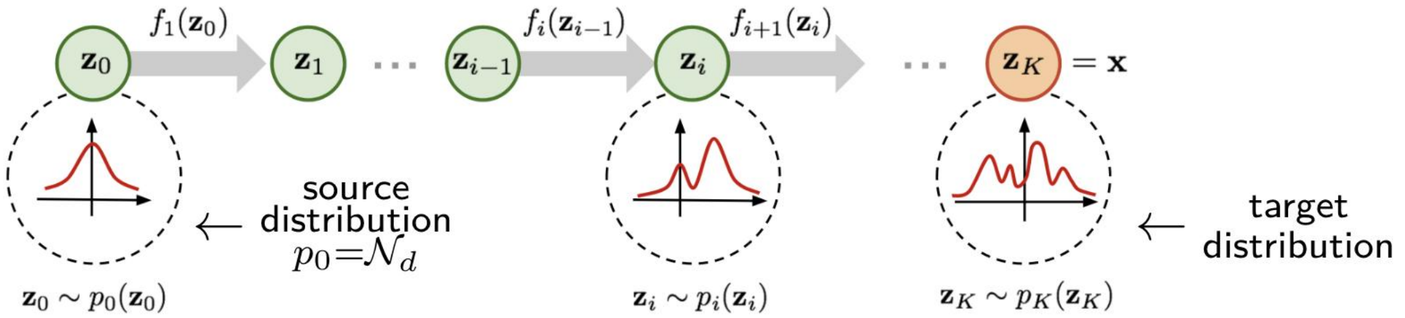
donde cada f_k es una transformación invertible.



Variational Inference *with Normalizing Flows*

En la inferencia variacional tradicional se elige una familia simple de distribuciones para modelar el posterior, lo que puede limitar la capacidad de capturar estructuras complejas (como multi-modalidad, correlaciones, etc.). Normalizing flow parte de una distribución base $q_0(z_0)$ y se le aplica una serie de transformaciones invertibles y diferenciables. Cada transformación moldea la distribución, de modo que, al final de la secuencia, se obtiene una distribución $q_K(z_K)$ mucho más flexible.

La regla de cambio de variable: $$q(z') = q(z)\left|\det\frac{\partial f^{-1}(z')}{\partial z'}\right| = q(z)\left|\det\frac{\partial f(z)}{\partial z}\right|^{-1}$$



Aplica la regla de cambio de variable de manera secuencial, acumulando términos de log-determinante del Jacobiano en cada paso.

$$\log q_K(z_K) = \log q_0(z_0) - \sum_{k=1}^{K} \log\left|\det\frac{\partial f_k(z_{k-1})}{\partial z_{k-1}}\right|$$

Danilo Jimenez Rezende and Shakir Mohamed (2015) "Variational Inference with Normalizing Flows". International conference on machine learning (pp. 1530-1538). PMLR.

Variational Inference *with Normalizing Flows*

Planar Flows

$f(z) = z + u \cdot h(w^\top z + b)$ donde u, w y b son parámetros y $h(\cdot)$ es una función no lineal suave (por ejemplo, la tanh).

Esta transformación realiza un ajuste local en la dirección del vector u, condicionado por la proyección $w^\top z + b$.
Es decir, modifica la densidad de forma localizada, realizando expansiones o contracciones dependiendo del valor de h y de su derivada h'.

El determinante del Jacobiano: $\det \frac{\partial f}{\partial z} = 1 + u^\top \psi(z)$

donde $\psi(z) = h'(w^\top z + b)w$. Este cálculo es de costo $O(D)$, siendo D la dimensión de z.

Danilo Jimenez Rezende and Shakir Mohamed (2015) "Variational Inference with Normalizing Flows". International conference on machine learning (pp. 1530-1538). PMLR.

Variational Inference *with Normalizing Flows*

Planar Flows

$f(z) = z + u \cdot h(w^\top z + b)$ donde u, w y b son parámetros y $h(\cdot)$ es una función no lineal suave (por ejemplo, la tanh).

Esta transformación realiza un ajuste local en la dirección del vector u, condicionado por la proyección $w^\top z + b$.
Es decir, modifica la densidad de forma localizada, realizando expansiones o contracciones dependiendo del valor de h y de su derivada h'.

El determinante del Jacobiano: $\det \frac{\partial f}{\partial z} = 1 + u^\top \psi(z)$

donde $\psi(z) = h'(w^\top z + b)w$. Este cálculo es de costo $O(D)$, siendo D la dimensión de z.

Radial Flows

Estas transformaciones están diseñadas para expandir o contraer la densidad alrededor de un punto de referencia z_0. La forma presentada es:

$$f(z) = z + \beta \cdot h(\alpha, r) \cdot (z - z_0) \qquad \text{donde } r = \|z - z_0\| \text{ y } h(\alpha, r) = \frac{1}{\alpha + r}.$$

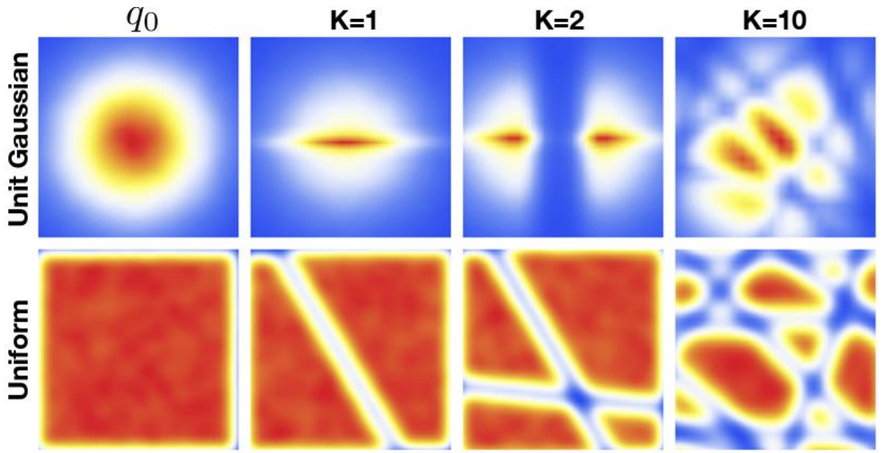
Con esta transformación se modifica la densidad de manera radial: se pueden inducir contracciones (si se quiere concentrar la densidad) o expansiones (si se desea dispersarla), de manera que se adapta a la estructura del posterior que se desea aproximar.

Danilo Jimenez Rezende and Shakir Mohamed (2015) "Variational Inference with Normalizing Flows". International conference on machine learning (pp. 1530-1538). PMLR.

Variational Inference *with Normalizing Flows*

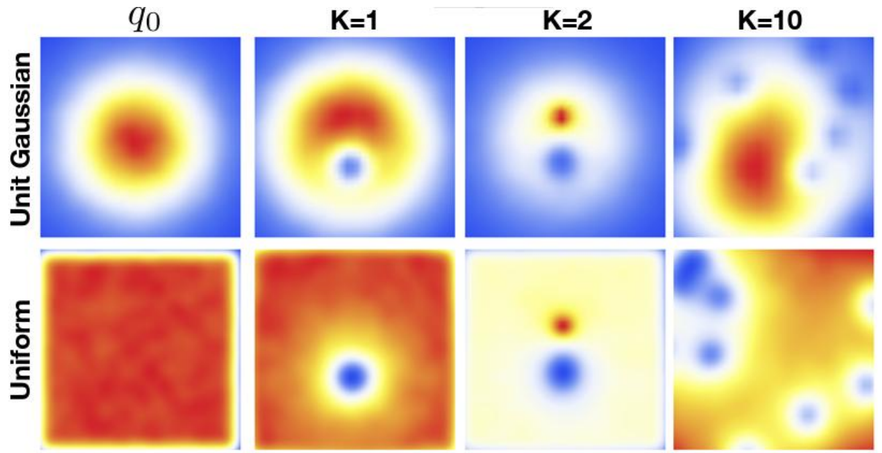
Planar Flows

$$f(z) = z + u \cdot h(w^{\top} z + b)$$



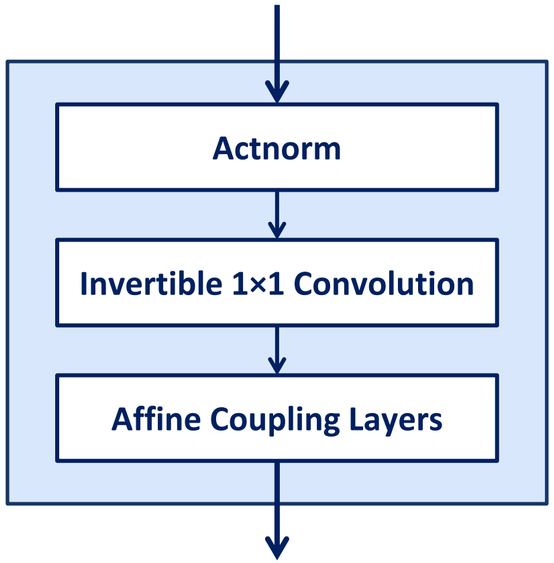
Radial Flows

$$f(z) = z + \beta \cdot h(\alpha, r) \cdot (z - z_0)$$



Danilo Jimenez Rezende and Shakir Mohamed (2015) "Variational Inference with Normalizing Flows". International conference on machine learning (pp. 1530-1538). PMLR.

Glow



Actnorm

Es una normalización de activaciones que se inicializa de manera dependiente a los datos. Se encarga de centrar y escalar las activaciones, ayudando a estabilizar el entrenamiento.

$y_c = s_c \cdot x_c + b_c$ donde:

- x_c es la entrada del canal c.
- s_c es un parámetro de escala aprendido.
- b_c es un parámetro de bias aprendido.

Invertible 1×1 Convolution

Transformación lineal invertible que se aplica a los canales de cada ubicación espacial. Consideremos en cada posición espacial la entrada es un vector $x \in \mathbb{R}^c$, donde c es el numero de canales.

$y = Wx$ donde:

- W es una matriz cuadrada de dimensión $C \times C$, aprendida por backpropagation.
- x, y son los valores de los canales en una posición espacial fija.

Affine Coupling Layers

Las capas de acoplamiento permiten transformar la entrada de manera invertible sin necesidad de calcular un Jacobiano costoso. Divide la entrada x en dos partes (x_1, x_2), y transforma una parte usando parámetros condicionados en la otra.

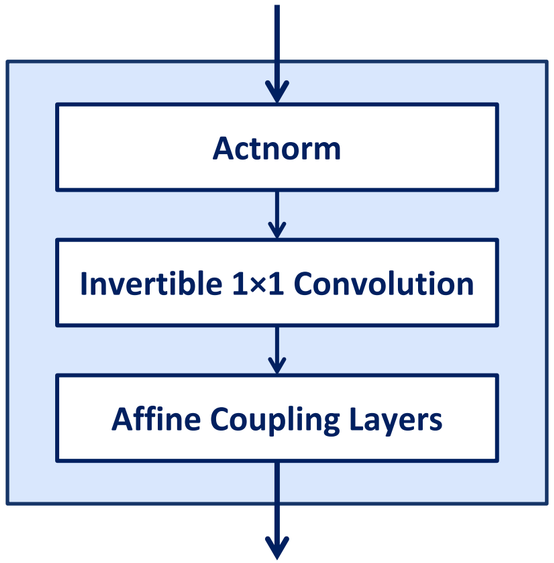
$y_1 = x_1$
$y_2 = x_2 \cdot \exp\big(s(x_1)\big) + t(x_1)$

donde:

- $s(x_1)$ y $t(x_1)$ son funciones aprendidas de x_1 (MLP).
- $\exp\big(s(x_1)\big)$ asegura que la transformación sea invertible.

Diederik P. Kingma and Prafulla Dhariwal (2018) "Glow: Generative Flow with Invertible 1x1 Convolutions". Advances in neural information processing systems, 31.

Glow



Diederik P. Kingma and Prafulla Dhariwal (2018) "Glow: Generative Flow with Invertible 1x1 Convolutions". Advances in neural information processing systems, 31.

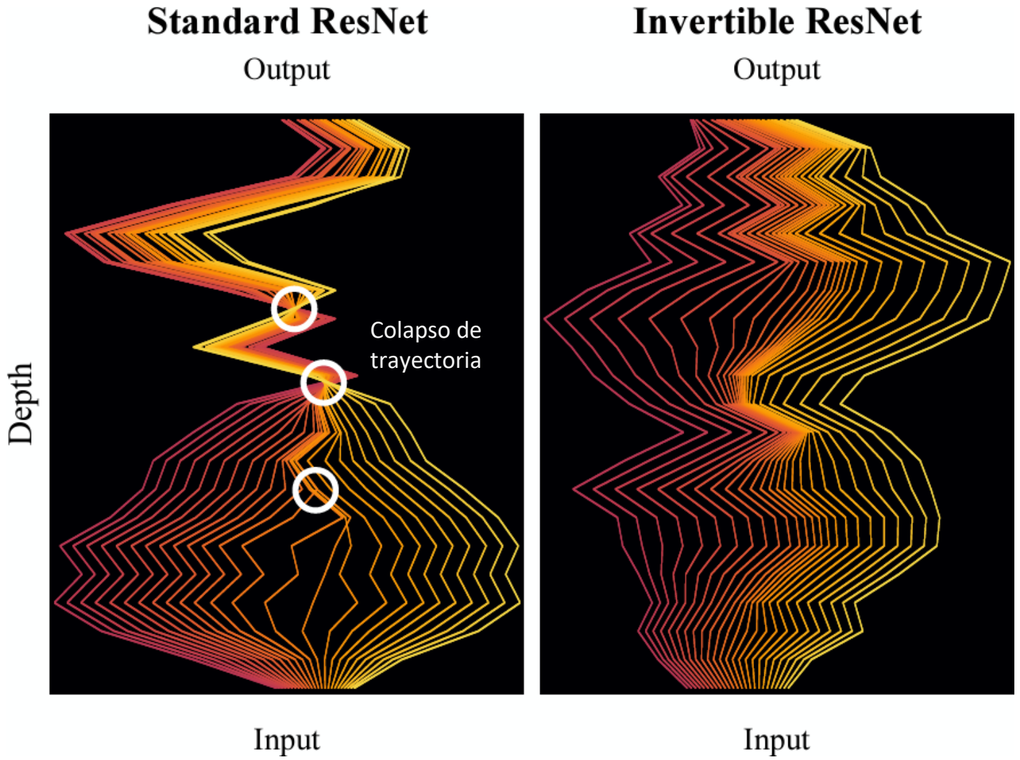
i-ResNets

Propone una modificación de las ResNets tradicionales para hacerlas invertibles, permitiendo que la misma arquitectura sea utilizada tanto para clasificación como para modelado generativo basado en máxima verosimilitud.

El modelo se basa en la observación de que un bloque residual estándar se define como:

$$y = x + g(x)$$

donde g(x) es la función residual.



Jens Behrmann et al. (2019) "Invertible Residual Networks". International conference on machine learning (pp. 573-582). PMLR.

i-ResNets

Propone una modificación de las ResNets tradicionales para hacerlas invertibles, permitiendo que la misma arquitectura sea utilizada tanto para clasificación como para modelado generativo basado en máxima verosimilitud.

El modelo se basa en la observación de que un bloque residual estándar se define como:

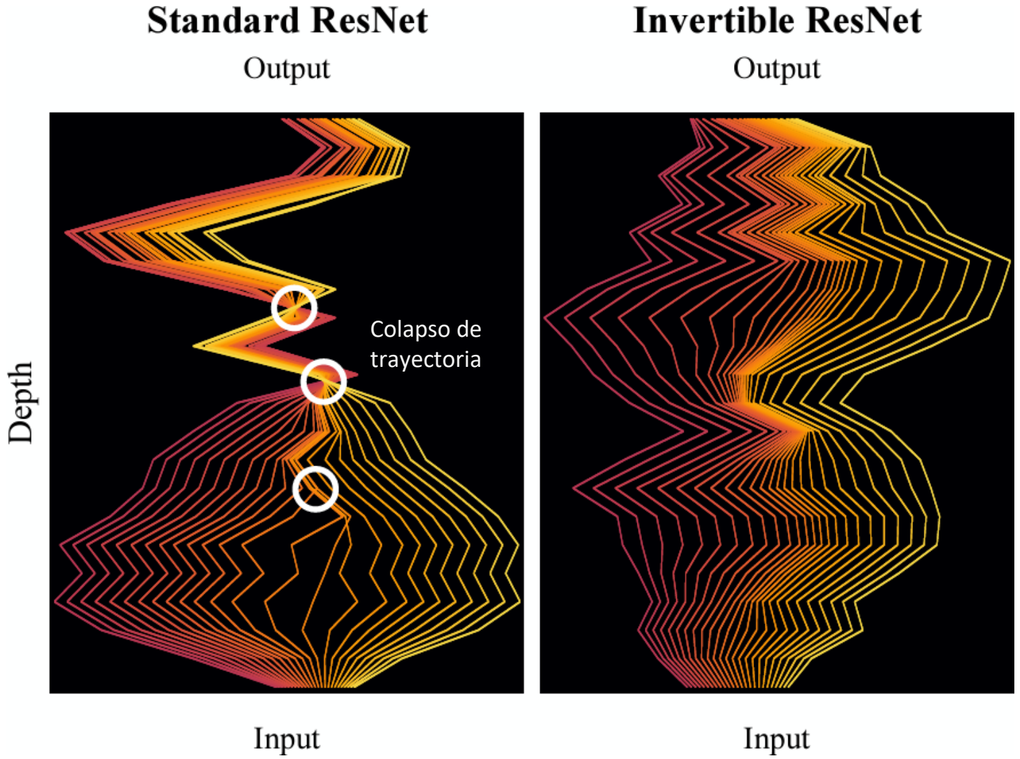
$$y = x + g(x)$$

donde g(x) es la función residual.

La red es invertible si $g(x)$ es una función contractiva, es decir, su constante de Lipschitz L:

$$\text{Lip}(g) < 1$$

Esto garantiza que la transformación $x \mapsto x + g(x)$ sea biyectiva y que la inversa pueda calcularse mediante iteración de punto fijo (demostrado en el paper).



Jens Behrmann et al. (2019) "Invertible Residual Networks". International conference on machine learning (pp. 573-582). PMLR.

i-ResNets

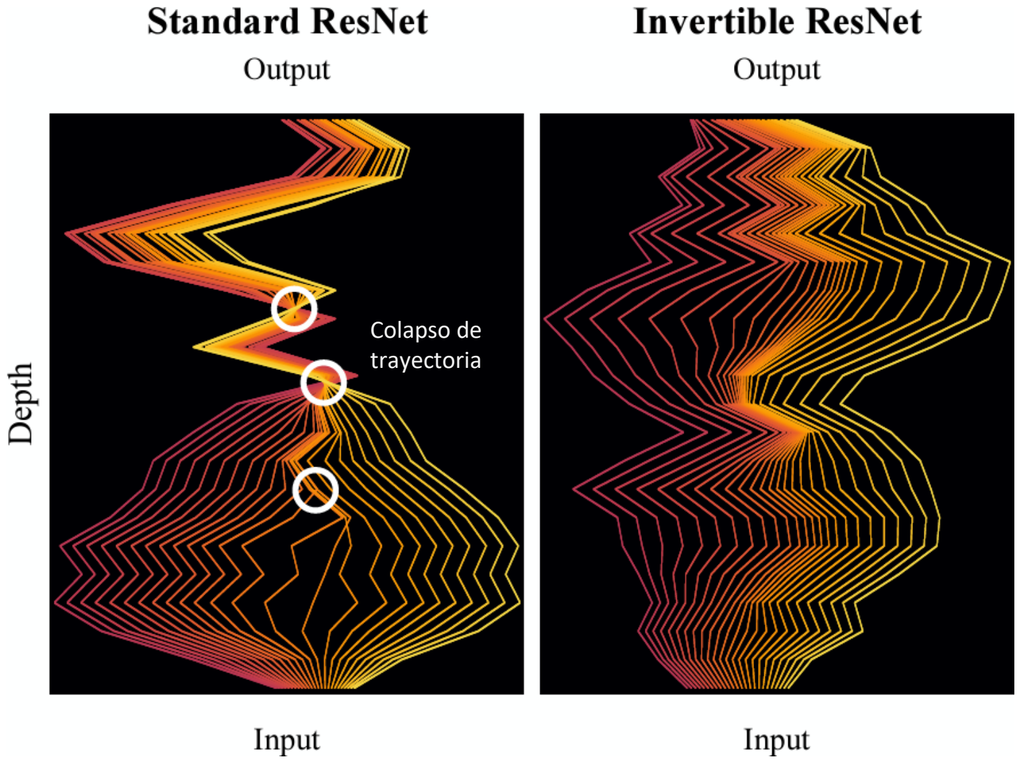
$g(x)$ en i-ResNets

Se emplea la siguiente configuración:

$$g(x) = W_2 ReLU(W_1 x)$$

donde:

- W_1, W_2 son convoluciones (usualmente 3×3).
- Se restringe W_1, W_2 para que su norma espectral sea menor que 1.



Jens Behrmann et al. (2019) "Invertible Residual Networks". International conference on machine learning (pp. 573-582). PMLR.

i-ResNets

$g(x)$ en i-ResNets

Se emplea la siguiente configuración:

$$g(x) = W_2 ReLU(W_1 x)$$

donde:

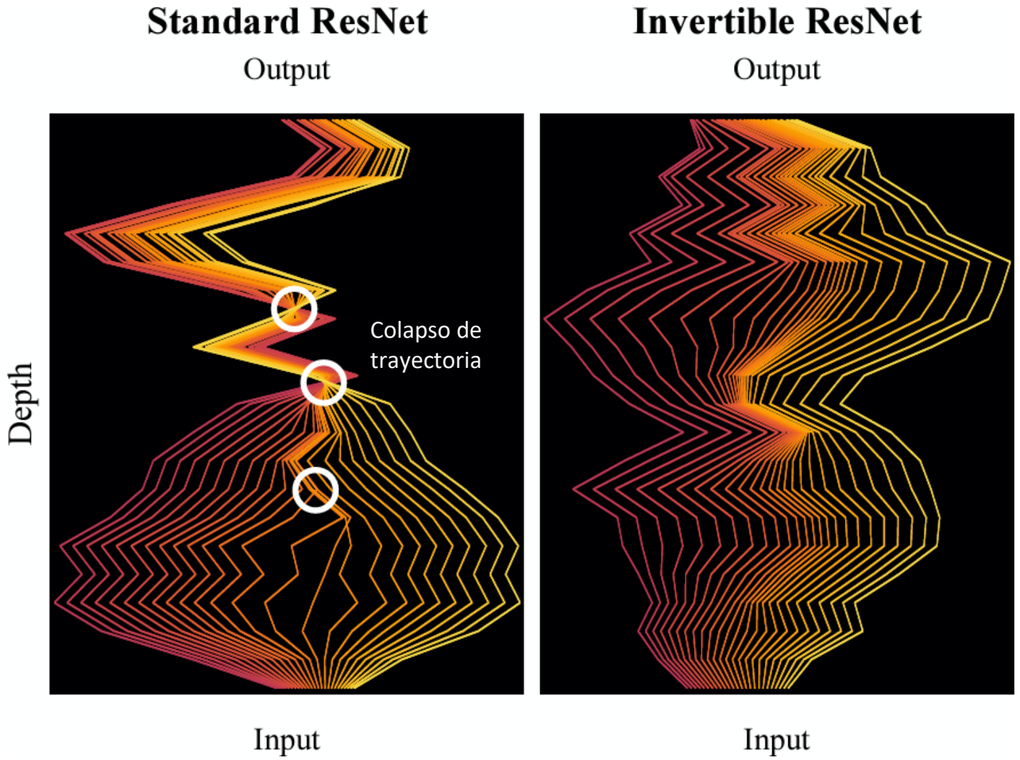
- W_1, W_2 son convoluciones (usualmente 3×3).
- Se restringe W_1, W_2 para que su norma espectral sea menor que 1.

Para asegurar la condición de Lipschitz, se emplea normalización espectral:

$$\widetilde{W} = \frac{\lambda W}{\sigma_{max}(W)}$$

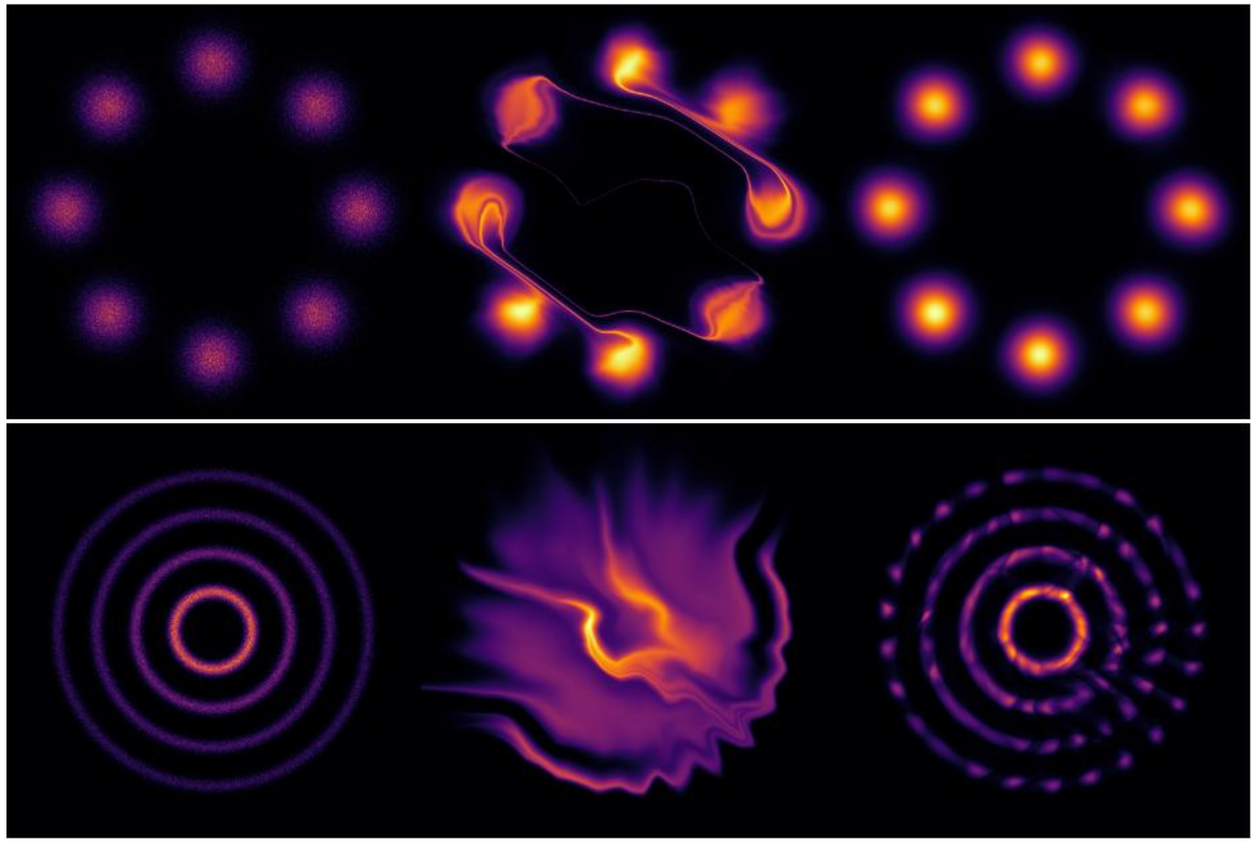
donde:
- $\sigma_{max}(W)$ el mayor valor singular de W.
- $\lambda < 1$ es un hiperparámetro que controla la contracción.

Esto se hace en cada capa convolucional, asegurando que todas las convoluciones de $g(x)$ respeten la condición $\text{Lip}(g) < 1$.



Jens Behrmann et al. (2019) "Invertible Residual Networks". International conference on machine learning (pp. 573-582). PMLR.

i-ResNets



Data Samples Glow i-ResNet

Jens Behrmann et al. (2019) "Invertible Residual Networks". International conference on machine learning (pp. 573-582). PMLR.

UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

2.

Continuous Normalizing *Flows (CNF)*

TRANSFORMATEC

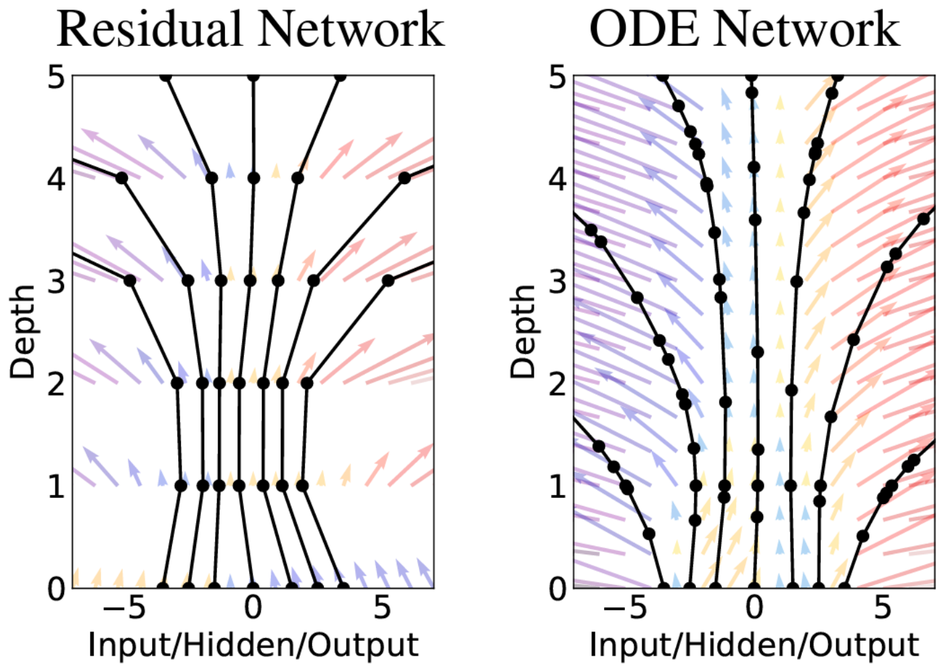
Neural Ordinary Differential *Equations (Neural ODEs)*

Neural ODEs son una clase de modelos de aprendizaje profundo que representan la evolución de estados ocultos a través de ecuaciones diferenciales ordinarias (ODEs), en lugar de una secuencia discreta de transformaciones de capas, como en las redes neuronales tradicionales.

Red neuronal estándar: $\boldsymbol{h}_{t+1} = \boldsymbol{h}_t + f(\boldsymbol{h}_t, \theta_t)$

ODE network: $\frac{d\boldsymbol{h}(t)}{dt} = f(\boldsymbol{h}(t), t, \theta)$

donde:
- $h(t) \in R^D$ es el estado oculto en el tiempo t
- $f: \mathbb{R}^D \times \mathbb{R} \to \mathbb{R}^D$ es una función diferenciable parametrizada por una red neuronal con parámetros θ
- θ son los pesos de la red neuronal f



Ricky T. Q. Chen et al. (2018) "Neural ordinary differential equations". Advances in neural information processing systems, 31.

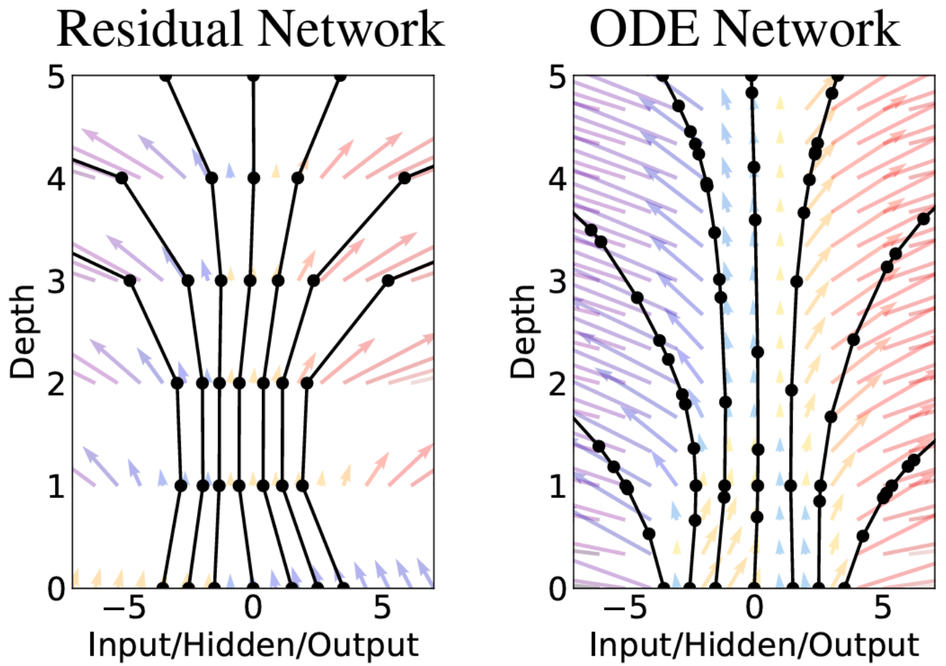
Neural Ordinary Differential *Equations (Neural ODEs)*

Neural ODEs son una clase de modelos de aprendizaje profundo que representan la evolución de estados ocultos a través de ecuaciones diferenciales ordinarias (ODEs), en lugar de una secuencia discreta de transformaciones de capas, como en las redes neuronales tradicionales.

Red neuronal estándar: $\boldsymbol{h}_{t+1} = \boldsymbol{h}_t + f(\boldsymbol{h}_t, \theta_t)$

ODE network: $\frac{d\boldsymbol{h}(t)}{dt} = f(\boldsymbol{h}(t), t, \theta)$

- **Control de memoria:** se pueden entrenar sin guardar todos los estados intermedios, usando técnicas de sensibilidades adjuntas (adjoint method).
- **Adaptación en el tiempo de cómputo:** el número de evaluaciones de la red se ajusta dinámicamente según la complejidad local de la trayectoria.
- **Regularización natural:** los parámetros de la dinámica están compartidos de forma continua en lugar de repetirse capa a capa.



Ricky T. Q. Chen et al. (2018) "Neural ordinary differential equations". Advances in neural information processing systems, 31.

Neural Ordinary Differential *Equations (Neural ODEs)*

Neural ODEs son una clase de modelos de aprendizaje profundo que representan la evolución de estados ocultos a través de ecuaciones diferenciales ordinarias (ODEs), en lugar de una secuencia discreta de transformaciones de capas, como en las redes neuronales tradicionales.

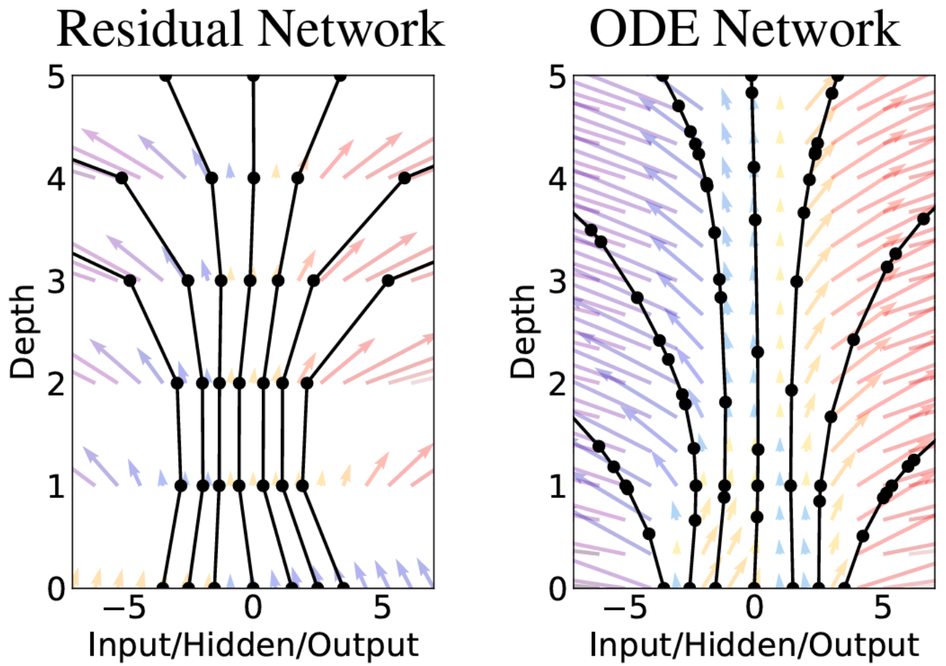
Red neuronal estándar: $\boldsymbol{h}_{t+1} = \boldsymbol{h}_t + f(\boldsymbol{h}_t, \theta_t)$

ODE network: $\frac{d\boldsymbol{h}(t)}{dt} = f(\boldsymbol{h}(t), t, \theta)$

Dado un estado inicial $h(t_0)$, el valor del estado en un tiempo posterior t_1 se obtiene resolviendo la ODE:

$$h(t_1) = h(t_0) + \int_{t_0}^{t_1} f(h(t), t, \theta) dt$$

Aquí se puede emplear métodos numéricos de ecuaciones diferenciales, como por ejemplo Runge-Kutta o Adam-Bashforth.



TRANSFORMATEC

Ricky T. Q. Chen et al. (2018) "Neural ordinary differential equations". Advances in neural information processing systems, 31.

Continuous Normalizing *Flows (CNF)*

Queremos minimizar una función de pérdida $\mathcal{L}$, la cual depende del estado oculto $h(T)$ en el tiempo final T:

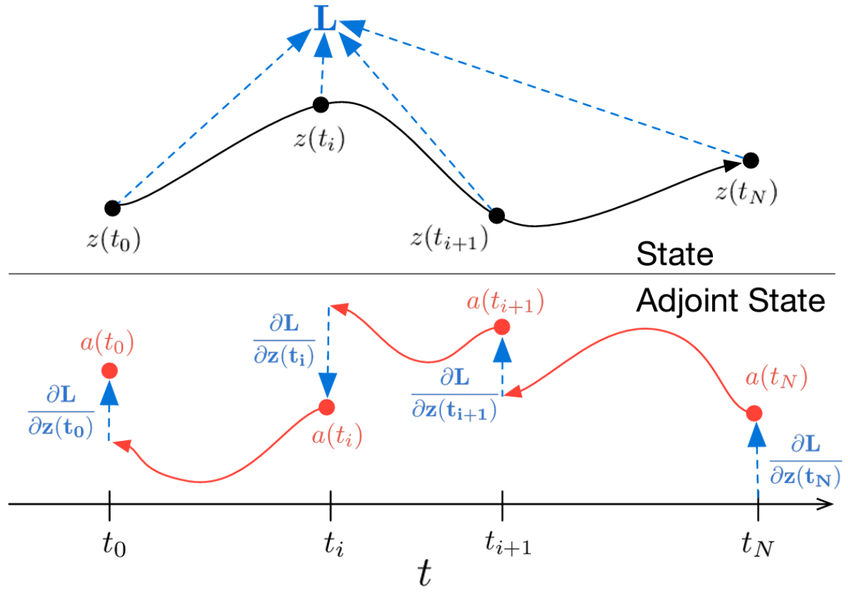
$$\min_{\theta} \mathcal{L}(h(T), y)$$

donde:

- $h(T)$ es la salida de la Neural ODE, obtenida resolviendo la ecuación:

$$h(T) = h(0) + \int_0^T f(h(t), t, \theta) dt$$

- $f(h, t, \theta)$ es la red neuronal que define la dinámica del estado oculto.
- y es la etiqueta objetivo en un problema supervisado.



Ricky T. Q. Chen et al. (2018) "Neural ordinary differential equations". Advances in neural information processing systems, 31.

Continuous Normalizing *Flows (CNF)*

Queremos minimizar una función de pérdida $\mathcal{L}$, la cual depende del estado oculto $h(T)$ en el tiempo final T:

$$\min_{\theta} \mathcal{L}(h(T), y)$$

donde:
- $h(T)$ es la salida de la Neural ODE, obtenida resolviendo la ecuación:

$$h(T) = h(0) + \int_0^T f(h(t), t, \theta) dt$$

- $f(h, t, \theta)$ es la red neuronal que define la dinámica del estado oculto.
- y es la etiqueta objetivo en un problema supervisado.

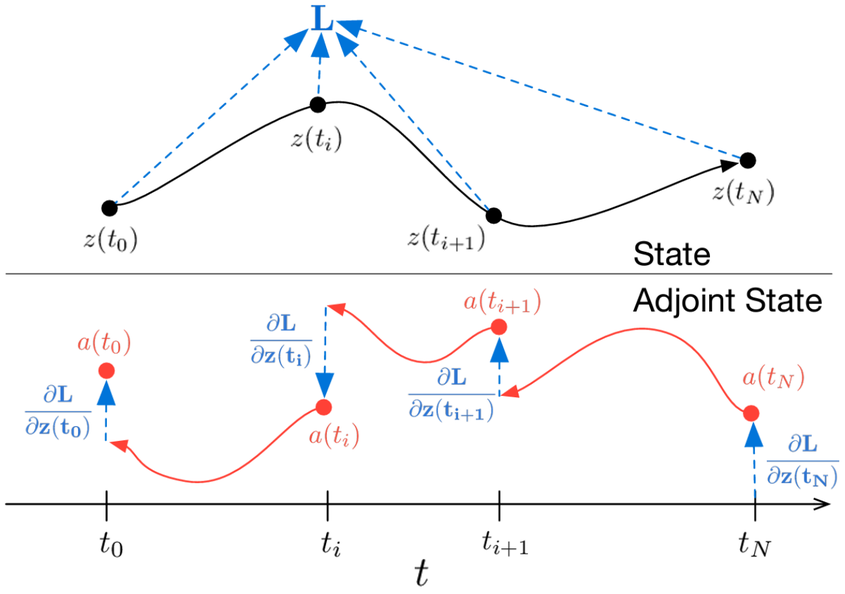
Para actualizar θ empleamos **adjoint sensitivity**, el cual permite calcular los gradientes sin necesidad de almacenar toda la trayectoria.

Definimos la variable adjunta: $a(t) = \dfrac{\partial \mathcal{L}}{\partial h(t)}$

que mide la sensibilidad de la pérdida $\mathcal{L}$ respecto al estado oculto $h(t)$.

Se demuestra que $a(t)$ sigue una ecuación diferencial en sentido inverso al de la ODE original.

$$\frac{da(t)}{dt} = -a(t)^T \frac{\partial f}{\partial h}(h(t), t, \theta)$$



Ricky T. Q. Chen et al. (2018) "Neural ordinary differential equations". Advances in neural information processing systems, 31.

Continuous Normalizing *Flows (CNF)*

Queremos minimizar una función de pérdida $\mathcal{L}$, la cual depende del estado oculto $h(T)$ en el tiempo final T:

$$\min_{\theta} \mathcal{L}(h(T), y)$$

La variable adjunta: $a(t) = \frac{\partial \mathcal{L}}{\partial h(t)}$

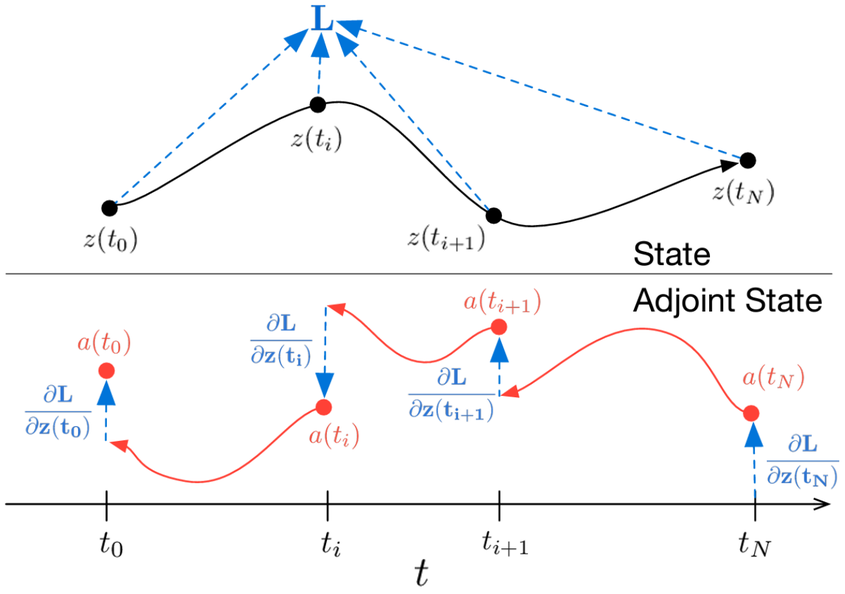
Se demuestra que $a(t)$ sigue una ecuación diferencial en sentido inverso al de la ODE original.

$$\frac{da(t)}{dt} = -a(t)^T \frac{\partial f}{\partial h}(h(t), t, \theta)$$

El aspecto fundamental de este método es cómo se obtiene la información de $h(t)$ necesaria para calcular $\frac{\partial f}{\partial h}$ durante el backpropagation. En lugar de almacenar todos los valores intermedios de $h(t)$ durante la pasada hacia adelante, se opta por re-integrar la ODE durante la pasada hacia atrás. Esto significa que, partiendo del estado final $h(T)$, se vuelve a resolver la ecuación diferencial en dirección inversa para reconstituir los valores de $h(t)$ cuando se necesitan.

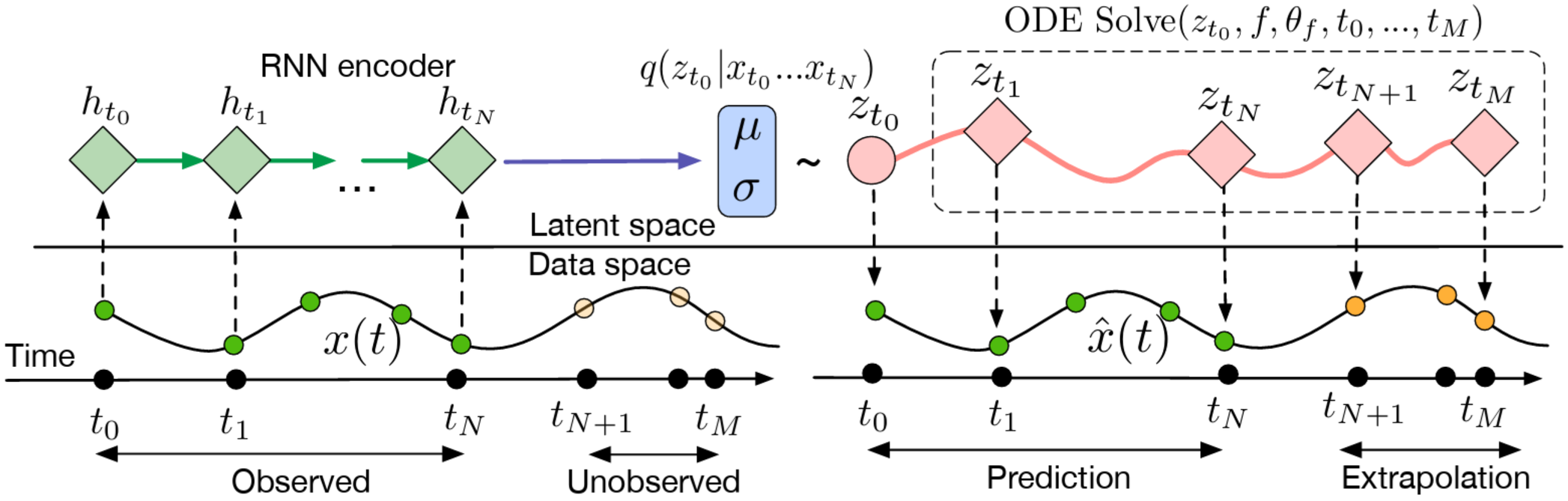
Entonces: $$\frac{dL}{d\theta} = \int_{t_1}^{t_0} a(t)^T \frac{\partial f}{\partial \theta}(h(t), t, \theta) dt$$

Esta estrategia permite mantener un uso de memoria constante, ya que no es necesario almacenar la trayectoria completa de $h(t)$ en memoria, a cambio de volver a computar la trayectoria durante el backpropagation.



Ricky T. Q. Chen et al. (2018) "Neural ordinary differential equations". Advances in neural information processing systems, 31.

Continuous Normalizing *Flows (CNF)*



TRANSFORMATEC

Ricky T. Q. Chen et al. (2018) "Neural ordinary differential equations". Advances in neural information processing systems, 31.

Continuous Normalizing *Flows (CNF)*

$$L(\mathbf{z}(t_1)) = L\left(\mathbf{z}(t_0) + \int_{t_0}^{t_1} f(\mathbf{z}(t), t, \theta)dt\right) = L\left(\text{ODESolve}(\mathbf{z}(t_0), f, t_0, t_1, \theta)\right)$$

```python
import torch
import torch.nn as nn
import torch.optim as optim
from torchdiffeq import odeint  # Solver de ODEs

# Función f(h, t, θ)
class ODEFunc(nn.Module):
    def __init__(self):
        super(ODEFunc, self).__init__()
        self.net = nn.Sequential(
            nn.Linear(2, 50),
            nn.Tanh(),
            nn.Linear(50, 2)
        )

    # ODE depende de t, pero no lo usamos aquí
    def forward(self, t, h):
        return self.net(h)

# Modelo Neural ODE
class NeuralODE(nn.Module):
    def __init__(self):
        super(NeuralODE, self).__init__()
        self.odefunc = ODEFunc()

    def forward(self, h0, t):
        return odeint(self.odefunc, h0, t)  # Solver de ODE
```

```python
# Crear datos sintéticos
t = torch.linspace(0, 1, 10)    # Intervalo de tiempo
h0 = torch.randn(1, 2)          # Estado inicial aleatorio
target = torch.randn(10, 1, 2) # Datos de referencia

model = NeuralODE()
optimizer = optim.Adam(model.parameters(), lr=0.01)

# Entrenamiento
for epoch in range(100):
    optimizer.zero_grad()
    h_pred = model(h0, t)  # Forward pass resolviendo la ODE
    loss = torch.mean((h_pred - target) ** 2)  # MSE Loss
    loss. backward()
    optimizer.step()
```


Ricky T. Q. Chen et al. (2018) "Neural ordinary differential equations". Advances in neural information processing systems, 31.

Continuous Normalizing *Flows (CNF)*

Problema

En Continuous Normalizing Flows, la densidad de un punto $x(t)$ evoluciona según la ecuación:

$$\frac{d}{dt}\log p\left(x(t)\right) = -\operatorname{tr}\left(\frac{\partial f}{\partial x}\left(x(t)\right)\right)$$

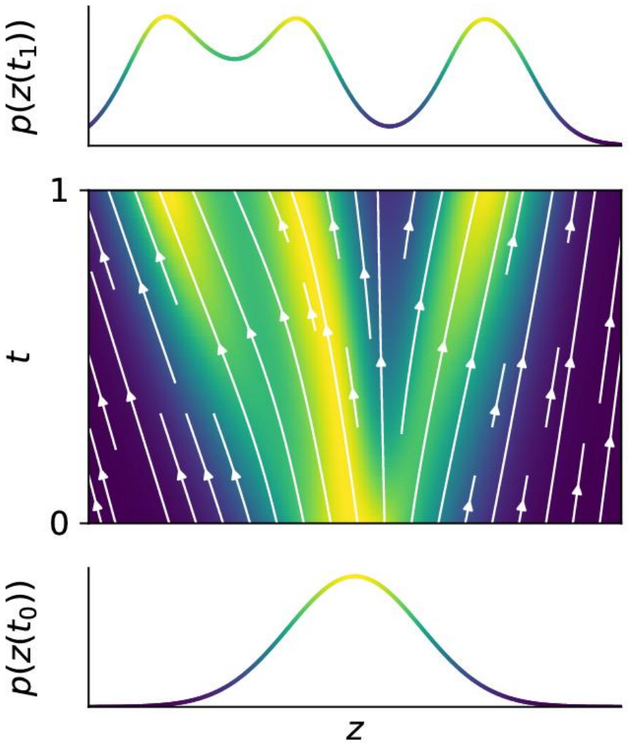
Traza de la matriz Jacobiana

donde f es el campo vectorial que define la ODE $\frac{dx}{dt} = f\left(x(t)\right)$.

Describimos qué tan probable es observar $x(T)$ después de que la variable x ha sido transformada hasta el tiempo T.

$$\log p\left(x(T)\right) = \log p\left(x(0)\right) - \int_0^T \operatorname{tr}\left(\frac{\partial f}{\partial x}\left(x(t)\right)\right) dt$$

En alta dimensión D, calcular esa traza tiene un costo de $O(D^2)$, lo cual puede volverse prohibitivo.



Ricky T. Q. Chen et al. (2018) "Neural ordinary differential equations". Advances in neural information processing systems, 31.

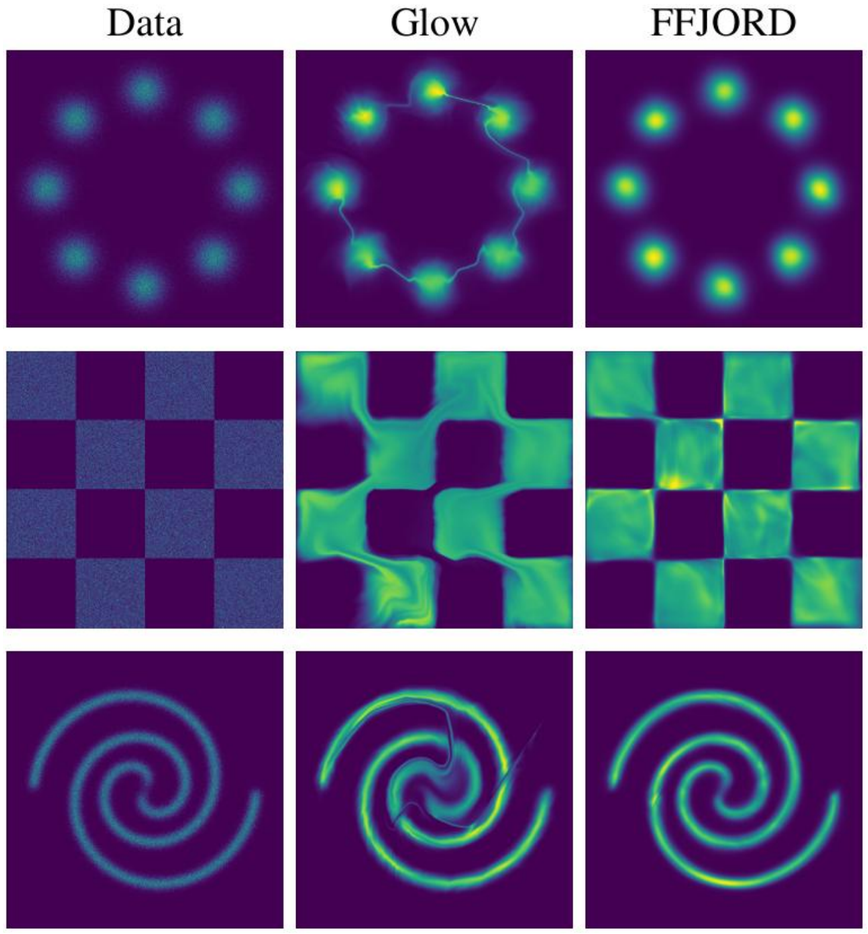
FFJORD

(Free-form Jacobian of Reversible Dynamics)

Para resolver el problema de la traza, FFJORD introduce un estimador estocástico basado en Hutchinson. Dado que la traza se puede estimar con:

$$\mathrm{Tr}(A) = \mathbb{E}_{\varepsilon}[\varepsilon^T A \varepsilon]$$

con ε una variable aleatoria de distribución $\mathcal{N}(0, I)$, obtenemos un estimador no sesgado en $O(D)$.



Will Grathwohl et al. (2018) "FFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models". arXiv preprint arXiv:1810.01367.

FFJORD

(Free-form Jacobian of Reversible Dynamics)

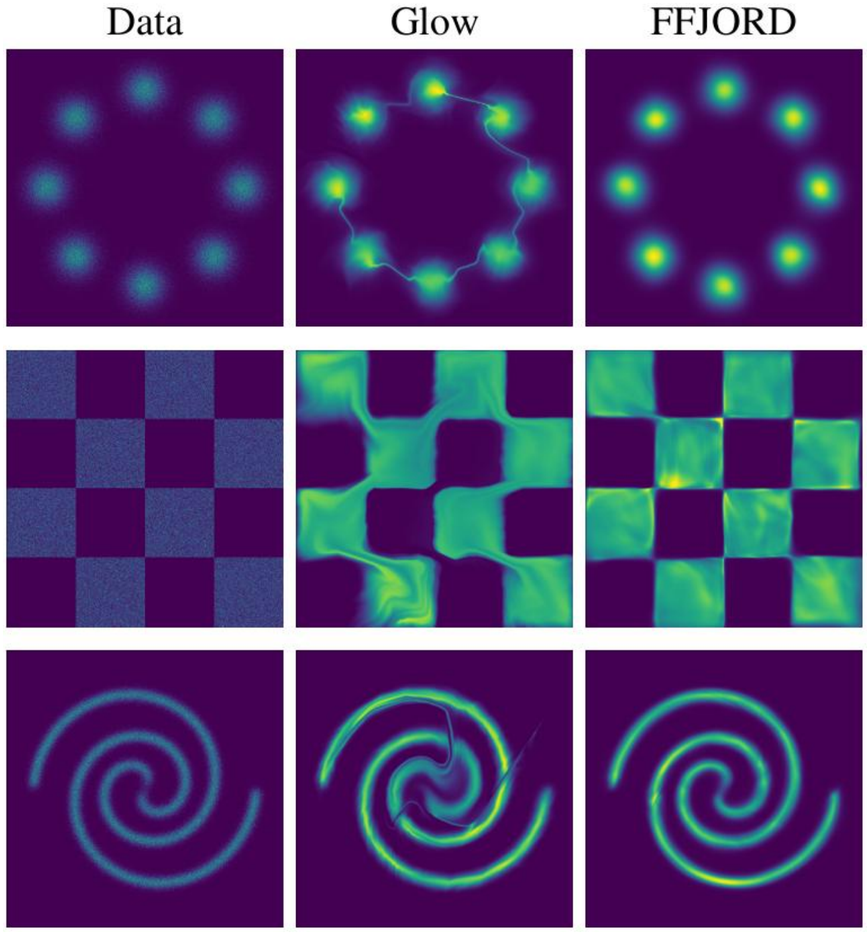
Para resolver el problema de la traza, FFJORD introduce un estimador estocástico basado en Hutchinson. Dado que la traza se puede estimar con:

$$\mathrm{Tr}(A) = \mathbb{E}_{\varepsilon}[\varepsilon^T A \varepsilon]$$

con ε una variable aleatoria de distribución $\mathcal{N}(0, I)$, obtenemos un estimador no sesgado en $O(D)$.

En la práctica, se toma un único ε por integración (o se fija a lo largo de la integración para mantener el determinismo) y se estima:

$$\mathrm{Tr}\left(\frac{\partial f}{\partial h}\right) \approx \varepsilon^T \frac{\partial f}{\partial h} \varepsilon$$



Will Grathwohl et al. (2018) "FFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models". arXiv preprint arXiv:1810.01367.

FFJORD

(Free-form Jacobian of Reversible Dynamics)

Para resolver el problema de la traza, FFJORD introduce un estimador estocástico basado en Hutchinson. Dado que la traza se puede estimar con:

$$\mathrm{Tr}(A) = \mathbb{E}_{\varepsilon}[\varepsilon^T A \varepsilon]$$

con ε una variable aleatoria de distribución $\mathcal{N}(0, I)$, obtenemos un estimador no sesgado en $O(D)$.

En la práctica, se toma un único ε por integración (o se fija a lo largo de la integración para mantener el determinismo) y se estima:

$$\mathrm{Tr}\left(\frac{\partial f}{\partial h}\right) \approx \varepsilon^T \frac{\partial f}{\partial h} \varepsilon$$

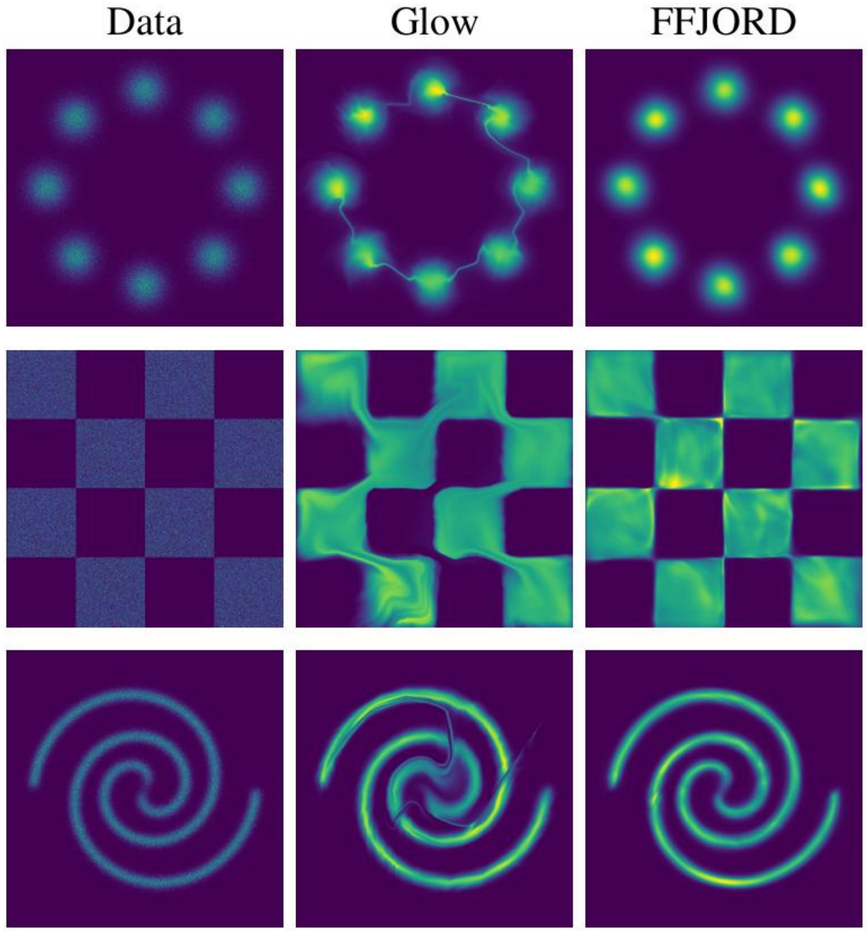
Reemplazando:

$$\log p\left(h(t_1)\right) \approx \log p\left(h(t_0)\right) - \int_{t_0}^{t_1} \varepsilon^T \frac{\partial f}{\partial h}(h(t), t, \theta)\varepsilon \, dt$$

“Simulación”

Luego aplicamos backpropagation mediante adjoint sensitivity:

$$\frac{da(t)}{dt} = -a(t)^T \frac{\partial f}{\partial h}(h(t), t, \theta)$$



Will Grathwohl et al. (2018) “FFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models”. arXiv preprint arXiv:1810.01367.

UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

3.

Flow *Matching*

TRANSFORMATEC

Flow *Matching*

El problema de entrenar CNFs de forma tradicional es que se requiere la simulación numérica de la ODE, lo cual puede tener un alto costo computacional. Flow Matching consigue entrenar CNFs sin emplear simulaciones.

La idea es definir una trayectoria de probabilidad $p_t(x)$ que conecte la distribución inicial p_0 con la distribución de datos $q(x_1)$ al final del tiempo $t = 1$. Para ello, se construye esta trayectoria a partir de trayectorias condicionales $p_t(x|x_1)$ asociadas a cada muestra x_1 de q:

$$p_t(x) = \int p_t(x|x_1)\ q(x_1)\, dx_1$$

Cada trayectoria condicional se diseña de forma que, en $t = 0$, $p_0(x|x_1) = p_0(x)$ (ruido gaussiano, por ejemplo) y en $t = 1$ se concentre alrededor de x_1:

$$p_1(x|x_1) = \mathcal{N}(x; x_1, \sigma^2 I)$$

con σ suficientemente pequeño.

Para cada trayectoria condicional existe un campo vectorial $u_t(x|x_1)$ que la genera. El siguiente paso es promediar estos campos condicionales para obtener el campo vectorial marginal $u_t(x)$ que genera la trayectoria global $p_t(x)$:

$$u_t(x) = \frac{\int u_t(x|x_1)\ p_t(x|x_1)\, q(x_1)\, dx_1}{p_t(x)}$$

Aunque en principio $u_t(x)$ resulta intractable, esta formulación permite construir una aproximación a través de las trayectorias condicionales.

Yaron Lipman et al. (2022) "Flow Matching for Generative Modeling". The Eleventh International Conference on Learning Representations.

Flow *Matching*

Flow Matching (FM) loss:

El objetivo de entrenamiento propuesto consiste en emparejar el campo vectorial aprendido $v_t(x)$ (parametrizado por θ) con el campo objetivo $u_t(x)$:

$$\mathcal{L}_{FM}(\theta) = E_{t\sim U(0,1),\, x\sim p_t(x)} \| v_t(x) - u_t(x) \|^2$$

donde:

- $t \sim U(0,1)$. Toma la esperanza respecto a t que se muestrea de una distribución uniforme en el intervalo $[0,1]$.
- $x \sim p_t(x)$. Trayectoria de densidad de probabilidad en el tiempo t. Esta trayectoria conecta la distribución inicial p_0 (ruido) con la distribución objetivo p_1 (aproximación de la distribución de datos).

Sin embargo, dado que $u_t(x)$ es difícil de calcular de manera directa.

Yaron Lipman et al. (2022) "Flow Matching for Generative Modeling". The Eleventh International Conference on Learning Representations.

Flow *Matching*

Flow Matching (FM) loss:

El objetivo de entrenamiento propuesto consiste en emparejar el campo vectorial aprendido $v_t(x)$ (parametrizado por θ) con el campo objetivo $u_t(x)$:

$$\mathcal{L}_{FM}(\theta) = E_{t\sim U(0,1),\, x\sim p_t(x)} \| v_t(x) - u_t(x) \|^2$$

donde:

- $t \sim U(0,1)$. Toma la esperanza respecto a t que se muestrea de una distribución uniforme en el intervalo $[0,1]$.
- $x \sim p_t(x)$. Trayectoria de densidad de probabilidad en el tiempo t. Esta trayectoria conecta la distribución inicial p_0 (ruido) con la distribución objetivo p_1 (aproximación de la distribución de datos).

Sin embargo, dado que $u_t(x)$ es difícil de calcular de manera directa.

Conditional Flow Matching (CFM) loss:

$$\mathcal{L}_{CFM}(\theta) = E_{t\sim U(0,1),\, x_1\sim q(x_1),\, x\sim p_t(x|x_1)} \| v_t(x) - u_t(x|x_1) \|^2$$

donde:

- $x_1 \sim q(x_1)$. Distribución de datos, esto implica que para cada dato real x_1 se define una trayectoria condicional.
- $x \sim p_t(x|x_1)$. Para cada x_1 se toma x según la distribución condicional $p_t(x|x_1)$.

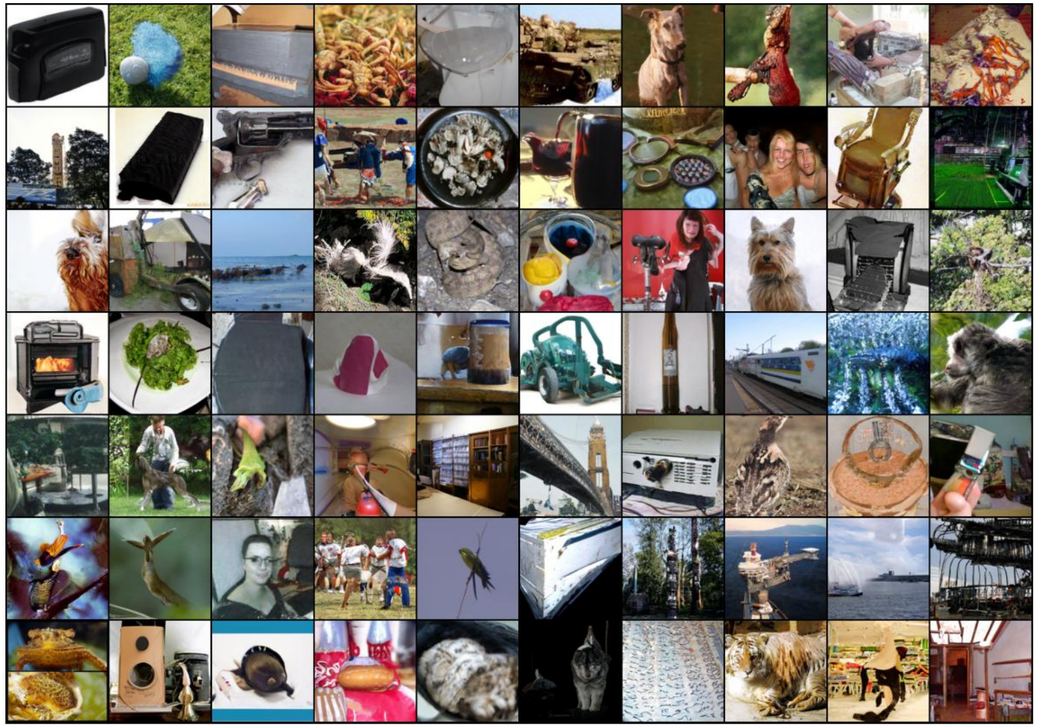
  Esta trayectoria condicional está diseñada para que:

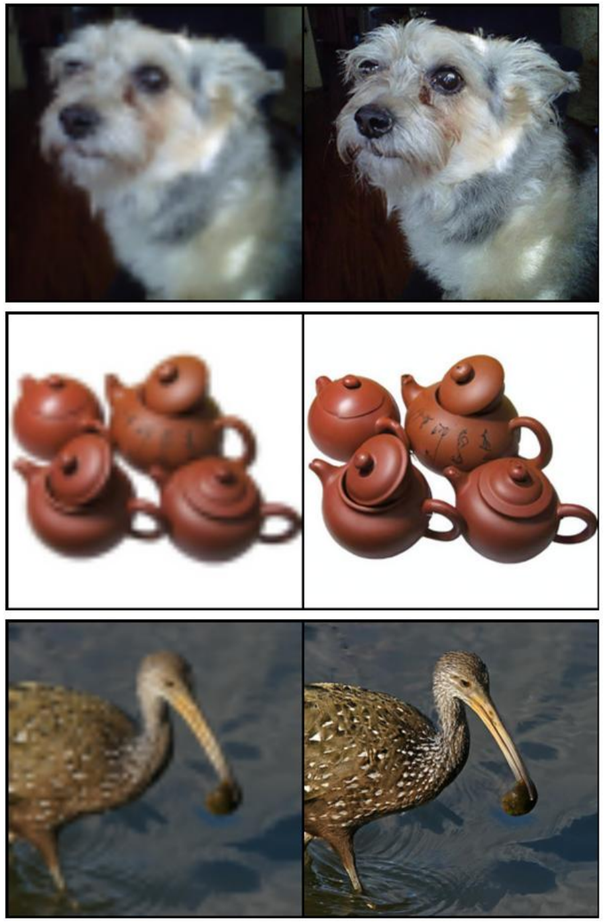
  - En $t = 0$, $p_0(x|x_1)$ sea igual a una distribución de ruido.
  - En $t = 1$, $p_1(x|x_1)$ se concentre alrededor de x_1.

Se demuestra teóricamente que los gradientes de $\mathcal{L}_{CFM}$ son equivalentes a los de $\mathcal{L}_{FM}$, lo que permite entrenar el CNF sin necesidad de estimar $u_t(x)$ de manera directa.

Yaron Lipman et al. (2022) "Flow Matching for Generative Modeling". The Eleventh International Conference on Learning Representations.

Flow *Matching*





Yaron Lipman et al. (2022) "Flow Matching for Generative Modeling".
The Eleventh International Conference on Learning Representations.

UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

4.

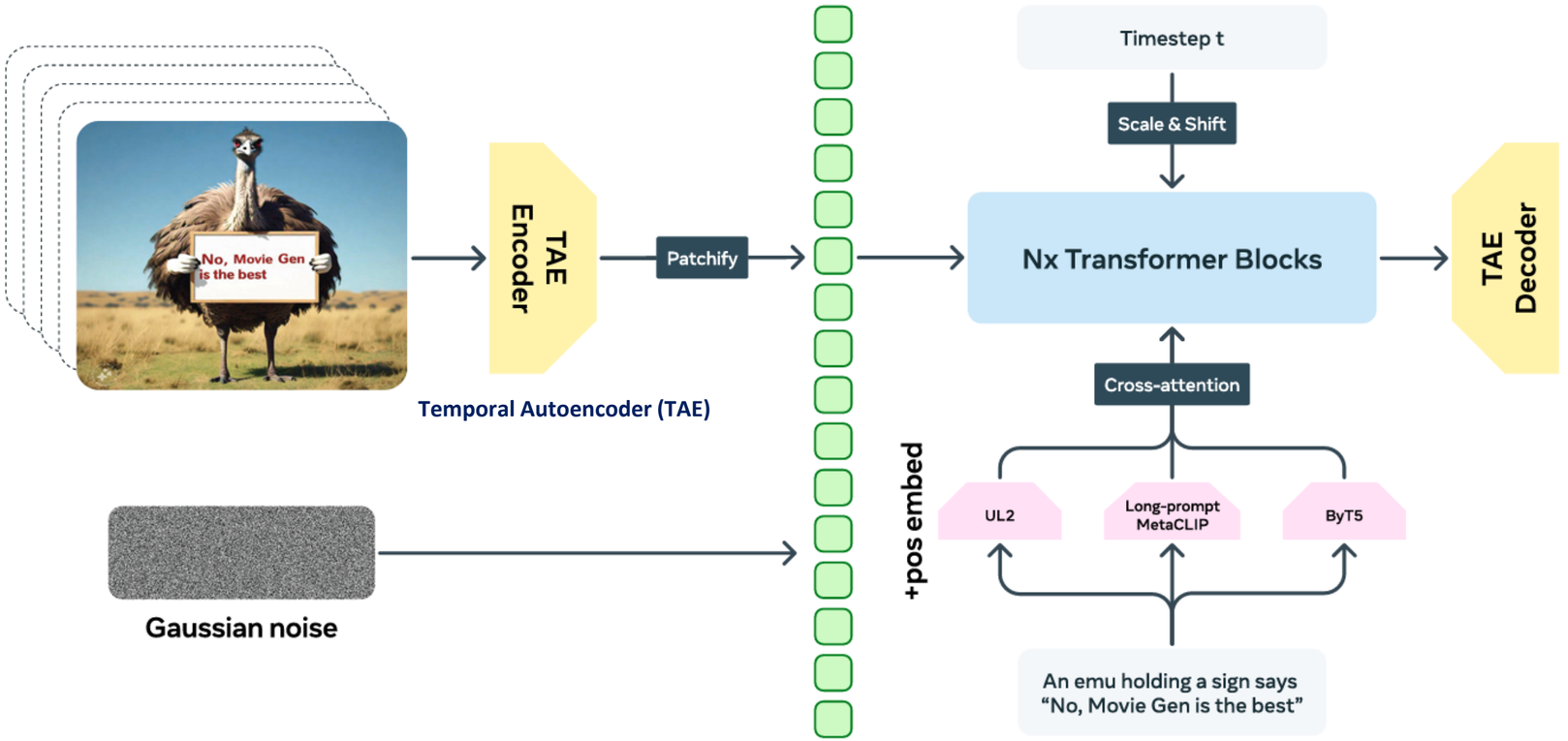
MovieGen

TRANSFORMATEC

MovieGen

Adam Polyak et al. (2024) "MovieGen: A Cast of Media Foundation Models".
arXiv preprint arXiv:2410.13720.

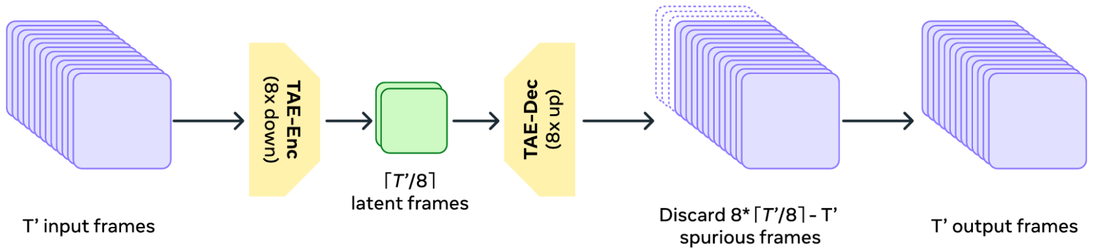
MovieGen



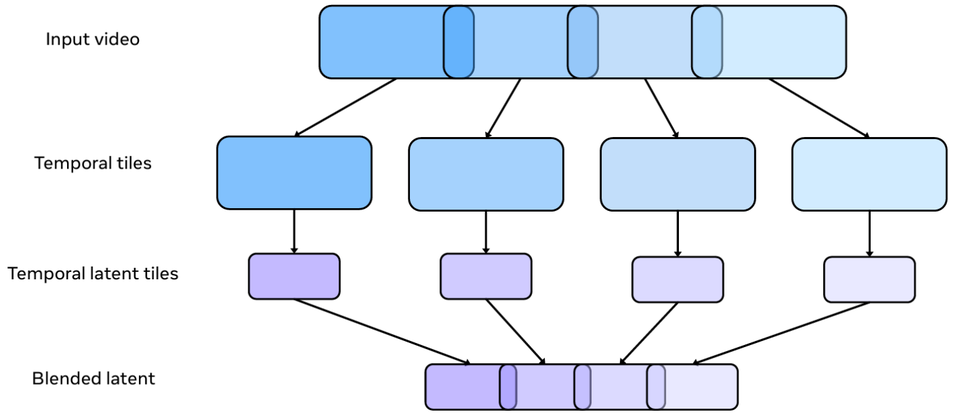
Adam Polyak et al. (2024) "MovieGen: A Cast of Media Foundation Models". arXiv preprint arXiv:2410.13720.

MovieGen

Variable length video encoding and decoding using the TAE

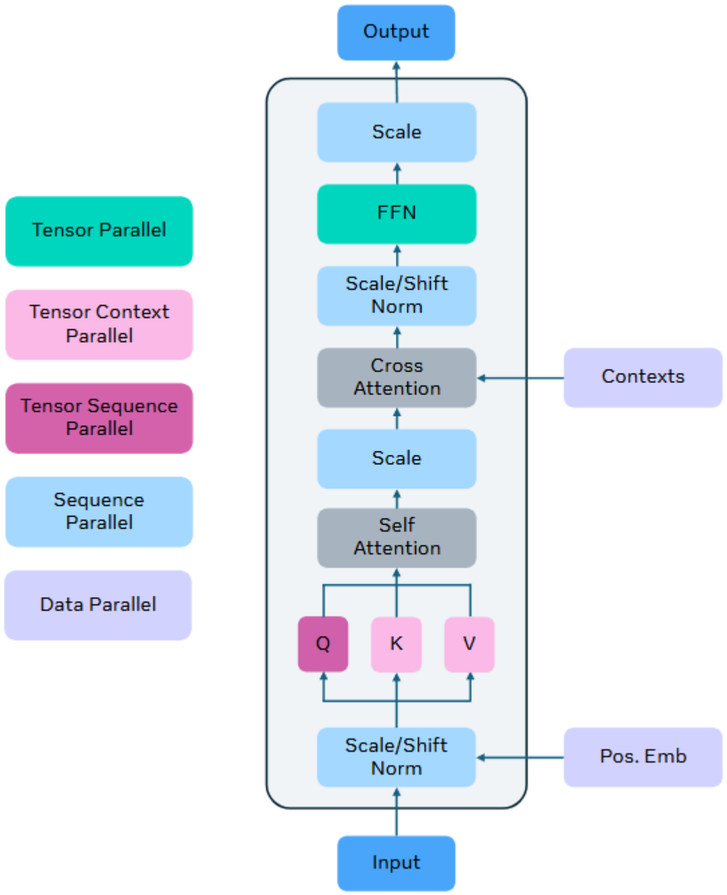


iled inference using the TAE



Adam Polyak et al. (2024) "MovieGen: A Cast of Media Foundation Models".
arXiv preprint arXiv:2410.13720.

MovieGen



Feature dimensions during 768 px video training

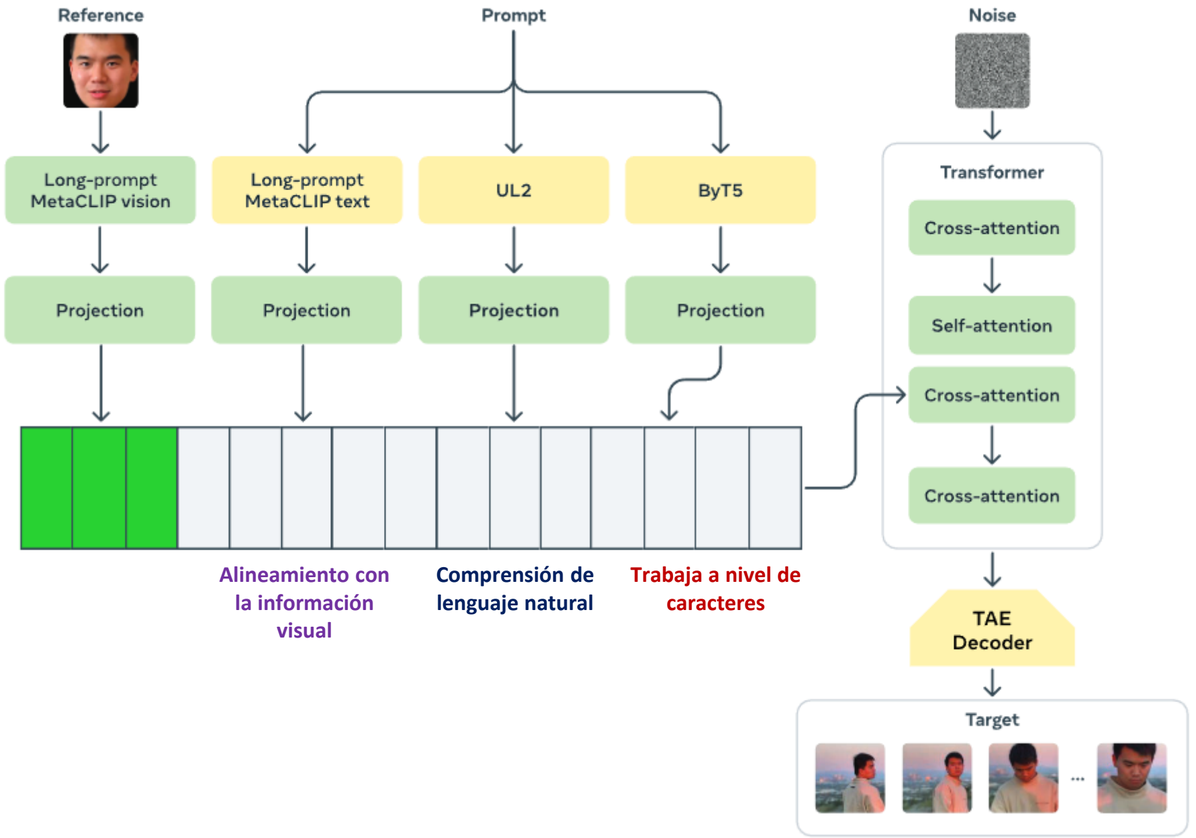
- Sequence length: 73728
- Number of heads: 48
- Hidden dimension: 6144

Output	(73728, 6144)
Contexts	UL2: (4096, 6144) Long-prompt MetaCLIP: (1280, 6144) ByT5: (1472, 6144)
Q K V	(73728, 48, 128)
Scale/Shift Norm	(1, 6144)
Pos. Emb	(73728, 6144)
Input	(73728, 6144)

Adam Polyak et al. (2024) "MovieGen: A Cast of Media Foundation Models". arXiv preprint arXiv:2410.13720.

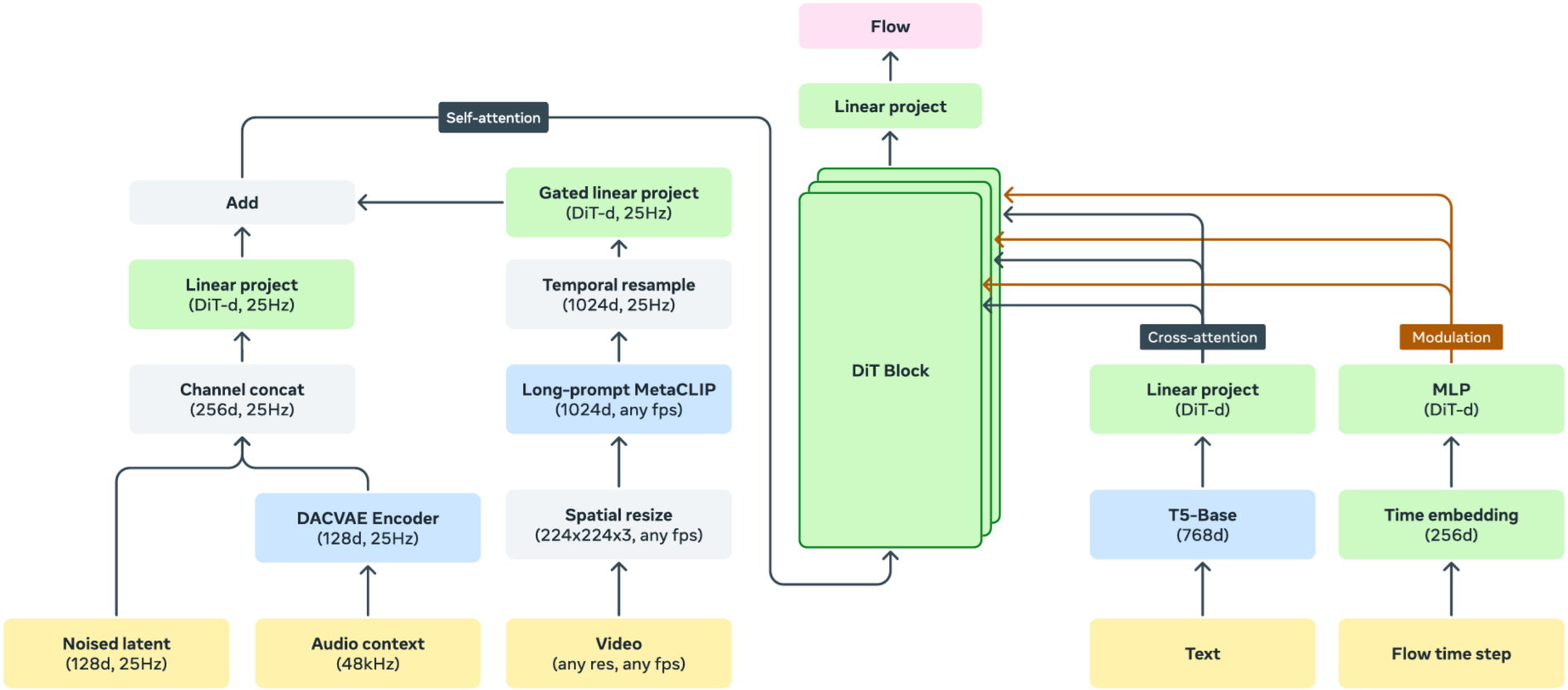
MovieGen

rchitecture and inference pipeline of the Personalized Movie Gen Video (PT2V) model



Adam Polyak et al. (2024) "MovieGen: A Cast of Media Foundation Models". arXiv preprint arXiv:2410.13720.

MovieGen

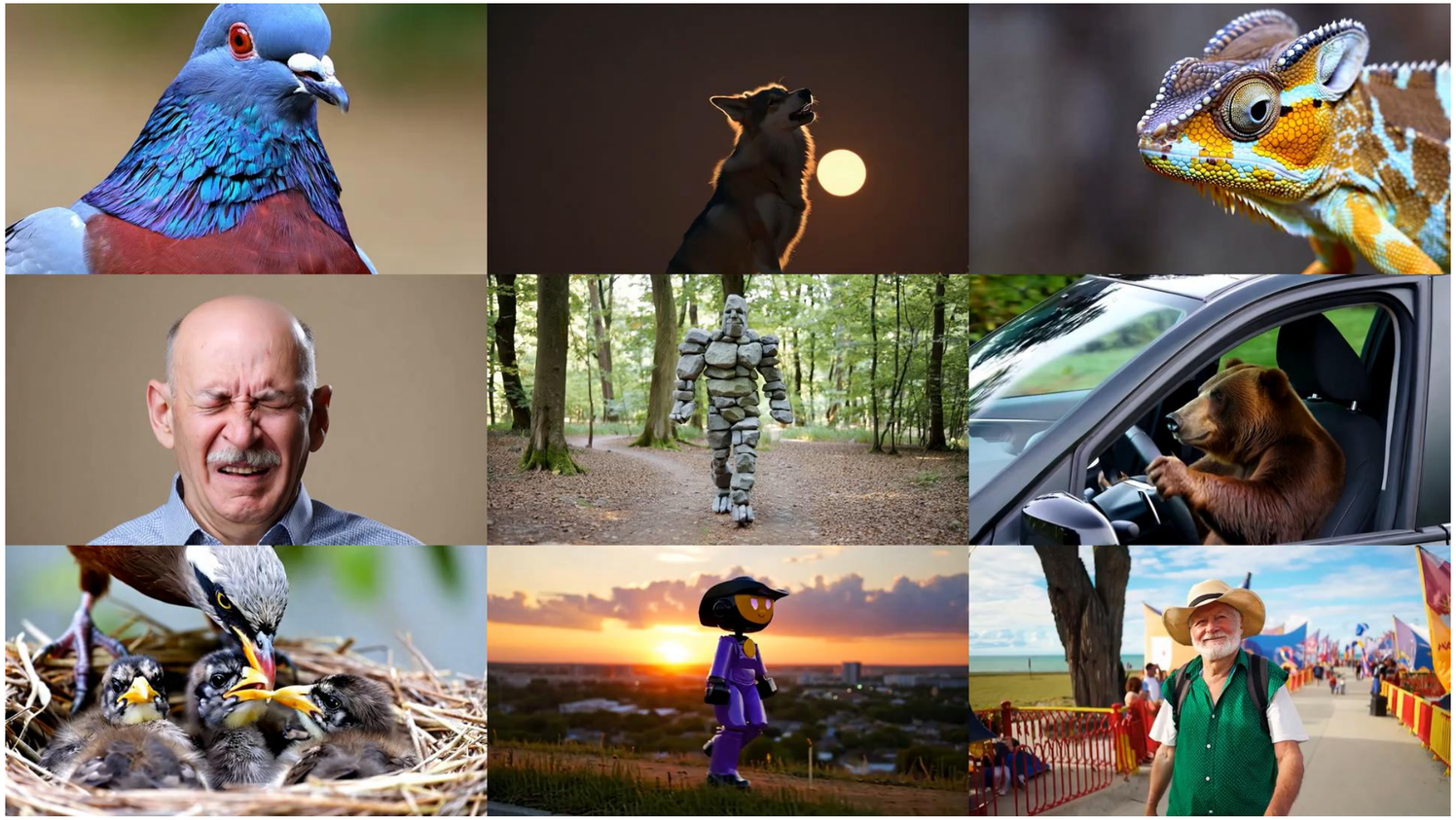


Adam Polyak et al. (2024) "MovieGen: A Cast of Media Foundation Models".
arXiv preprint arXiv:2410.13720.

MovieGen

Adam Polyak et al. (2024) "MovieGen: A Cast of Media Foundation Models". arXiv preprint arXiv:2410.13720.

Goku



Shoufa Chen et al. (2025) "Goku: Flow Based Video Generative Foundation Models". arXiv preprint arXiv:2502.04896

UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

> Reinventa el mundo <

GRACIAS

Victor Flores Benites

TRANSFORMATEC