



UNIVERSIDAD NACIONAL
DEL NORDESTE

Modulo 6:

Modelos generativos

Objetivos

- ▶ Máquinas de Boltzmann.
- ▶ GAN
- ▶ Flujos normalizantes
- ▶ Variational autoencoders
- ▶ Modelos difusivos

Modelos generativos



Imágenes de caras “sintéticas” no reales generadas por StyleGAN2 (NVIDIA)

Tomadas del sitio: <http://thispersondoesnotexist.com>

Modelos generativos

- ▶ No tenemos targets. Aprendizaje no-supervisado.
- ▶ Queremos aprender la función densidad de probabilidad.

Esto va mucho mas allá del problema de “estimación de función/al”

Queremos caracterizar a todo el conjunto de los datos con una densidad de probabilidad.

Generación de muestras sintéticas (no existen en los datos).

Aplicaciones:

- ▶ Generación de caras “nuevas”.
- ▶ Generación de textos (escribir un poema).
- ▶ Generación de una voz sintética.
- ▶ Tomadores de decisiones.

Modelos generativos

Métodos clásicos para estimar una PDF son el histograma y kernel density estimation (KDE).

- ▶ Curso de la dimensionalidad. No sirven para dimensiones mayores a 10.
- ▶ No existe una metodología eficiente para generar muestras de distribuciones complejas. (E.g. MCMC, importance sampling).

Alternativa: Aprender una transformación de los datos a un espacio latente de baja dimensionalidad.

Que los datos se representen con distribuciones sencillas (e.g. la normal).

Si tengo el mapa desde el espacio latente al espacio observacional, entonces dada una muestra en el espacio latente puedo encontrar una muestra en el espacio observacional.

GANs: Generative adversarial Networks.

Técnica para generar muestras que son indistinguibles de los datos.

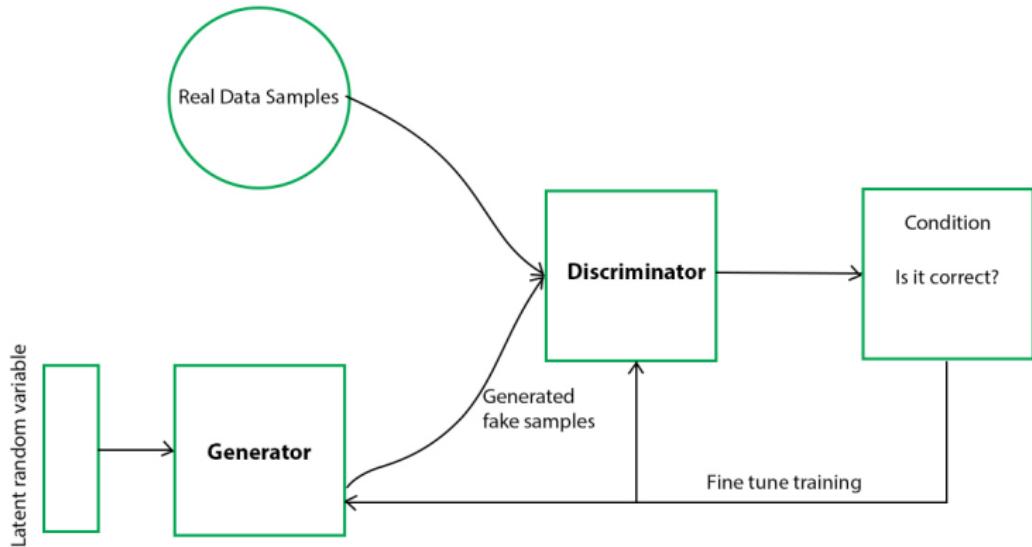
Para generar una muestra sintética, \mathbf{x}^* :

- ▶ Se muestrea a partir de una distribución normal en el espacio latente \mathbf{z}^*
- ▶ Dado \mathbf{z}^* la transformamos con una red al espacio observacional $\mathbf{x}^* = g(\mathbf{z}^*, \theta)$. Esta red es denominada **generador**.
- ▶ Una segunda red $d(\mathbf{x}^*, \phi)$ que actua como **discriminador**, intenta clasificar si la muestra es una real o es ficticia (no pertenece a los datos).

Juego del policía y ladrón.

CONS: No nos genera una distribución de probabilidad de los datos por lo que no podemos evaluar si un dato pertenece o no al conjunto.

Estructura de la GAN



Función de pérdida de las GANs

Para el discriminador, como es un clasificador vamos a poner $y_i = 1$ si pensamos que la muestra es real, y $y_i = 0$ si clasifica a la muestra como ficticia.

$$J(\phi) = - \sum_i (1 - y_i) \log (1 - \text{sig}(f(\mathbf{x}_i, \phi))) + y_i \log (\text{sig}(f(\mathbf{x}_i, \phi)))$$

Si tenemos n_D, n_G muestras reales y ficticias, entonces:

$$J(\phi) = - \sum_j^{n_G} \log (1 - \text{sig}(f(\mathbf{x}_j^*, \phi))) - \sum_i^{n_D} \log (\text{sig}(f(\mathbf{x}_i, \phi)))$$

Un término “pesa” las muestras ficticias y el otro las reales. La idea sería encontrar el ϕ los parámetros de la red que nos dan una **J mínima**.

Función de pérdida de las GANs

Si las muestras las genero con el generador, $\mathbf{x}_j^* = g(\mathbf{z}_j, \theta)$,

$$J(\theta, \phi) = - \sum_j^{n_G} \log (1 - \text{sig}(f(g(\mathbf{z}_j, \theta), \phi))) - \sum_i^{n_D} \log (\text{sig}(f(\mathbf{x}_i, \phi)))$$

Pero ahora necesito maximizar la J para “engaños” al discriminador que se piense que son muestras reales.

Es decir que para encontrar los θ requiero maximizar la J o minimizar la $-J$.

En el caso de θ solo depende de un término de la J , lo que tenemos que minimizar es:

$$J_G(\theta) = \sum_j^{n_G} \log (1 - \text{sig}(f(g(\mathbf{z}_j, \theta), \phi)))$$

Entrenamiento de las GANs

Entrenamiento en dos pasos, en uno minimizamos la J con respecto a ϕ para que el discriminador pueda distinguir las muestras sintéticas de las reales.

En otro paso, hacemos la maximización de J , o minimización de J_G , con respecto a θ busca que las muestras sintéticas se parezcan reales.

Esto es un **minimax game**.

Para cada paso de entrenamiento:

- ▶ Generamos n_G muestras aleatorias latentes $\{z_j, j = 1 : n_G\}$ y las transformamos, $\{x_j^* = g(z_j, \theta), j = 1 : n_G\}$
- ▶ Tomamos n_D muestras de los datos.
- ▶ Las $N_G + N_D$ muestras se las pasamos al discriminador.

Entrenamiento de las GANs

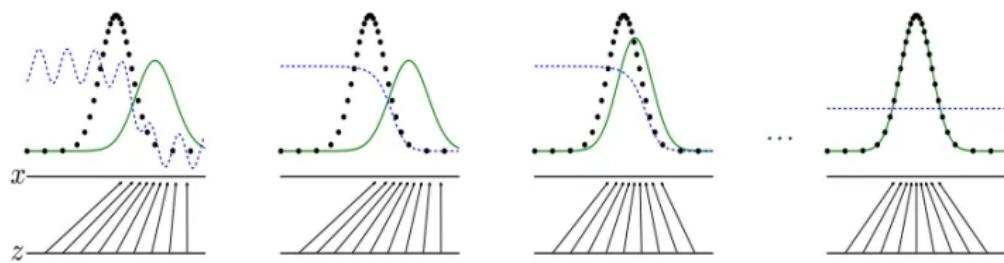
Para el descenso de gradientes hacemos primero el paso de J y luego el paso de J_G :

$$\phi_k = \phi_{k-1} - \eta \nabla J(x_{1:N_G}^*, x1 : N_D, \phi_{k-1}, \theta_{k-1})$$

$$\theta_k = \theta_{k-1} - \eta_G \nabla J_G(x_{1:N_G}^*, x1 : N_D, \phi_k, \theta_{k-1})$$

donde N_G, N_D es un minibatch, eg. $N_G = N_D = 64$.

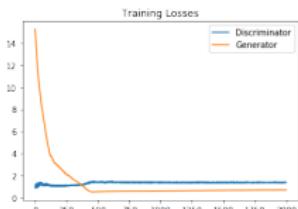
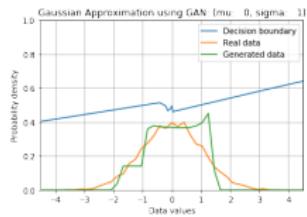
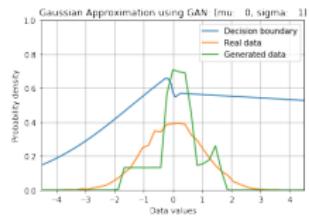
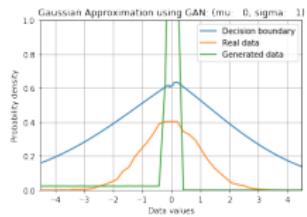
Ejemplo de una Gaussiana



Distribución discriminativa (línea azul a trazos). Distribución de los datos p_x (línea a puntos negros) y Distribucion generada por la GAN (curva verde).

A partir de puntos de una Gaussiana se transforman a la verde.
Notar que a lo ultimo el discriminador no puede distinguir los puntos (convergencia total).

Convergencia de las GANs



Convergencia de las GANs

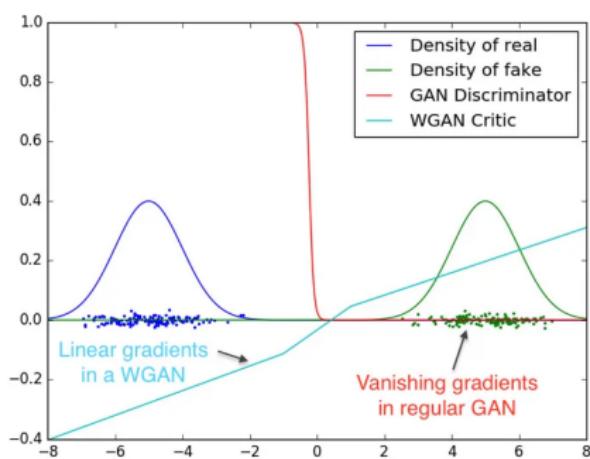
El problema de convergencia de las GANs es un equilibrio de Nash.

- ▶ Colapso en la moda.
- ▶ Pérdida de modas.
- ▶ Inestabilidad y no-convergencia.
- ▶ Alta sensibilidad a las métricas e hiperparámetros.

Queremos aprender no solo la media pero tambien el ruido/la incerteza del sistema

Wasserstein GAN

La medida que hay detrás de las GANs es la divergencia de Jensen-Shannon se puede usar la métrica de Wasserstein.



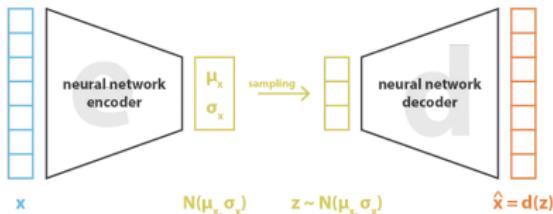
Wasserstein GAN:

- ▶ Mejora la estabilidad del aprendizaje.
- ▶ Previene el colapso a las modas.
- ▶ Permite optimización de los hiperparámetros (por la mejora de la estabilidad).

Arjovsky, M. et al. Wasserstein Generative Adversarial Networks. ICML 2017.

Variational Autoencoders

Los autocodificadores buscan producir replicas de la entrada pasando por un espacio latente de mucho menor dimensionalidad. Ejemplo MNIST.



- ▶ Modelo generativo: Queremos determinar la distribución de probabilidad.
- ▶ Asumimos que la distribución es Gaussiana.

$$\text{loss} = \|x - \hat{x}\|^2 + \text{KL}[N(\mu_x, \sigma_x), N(0, I)] = \|x - d(z)\|^2 + \text{KL}[N(\mu_x, \sigma_x), N(0, I)]$$

- ▶ Autoencoder que va a tratar de **inferir la media y la varianza en el espacio latente**.
- ▶ Luego debería producir muestras Gaussianas con $z \sim \mathcal{N}(\mu, \sigma)$.

Marco probabilístico de un variational autoencoder



Queremos inferir una distribución $p(z|x)$ de probabilidad en el espacio latente.

La reconstrucción $d(z)$ son muestras aleatorias condicionadas a la entrada.

Suponemos la inferencia de un modelo paramétrico, $q_\phi(z|x)$, el codificador.

La red del codificador debe predecir los parámetros de la distribución normal,

$$\mu, \sigma = \text{NeuralNet}_\phi(\mathbf{x})$$

$$q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; \mu, \sigma)$$

donde σ es la diagonal de la covarianza Σ .

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

ELBO

La inferencia variacional se encarga de optimizar los parámetros de la distribución. La log-verosimilitud de los datos es:

$$\log p_{\theta}(\mathbf{x}) = \mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x})} \left(\log \frac{p_{\theta}(\mathbf{x}, \mathbf{z})}{q_{\phi}(\mathbf{z}|\mathbf{x})} \right) + \mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x})} \left(\log \frac{q_{\phi}(\mathbf{z}|\mathbf{x})}{p_{\theta}(\mathbf{z}|\mathbf{x})} \right)$$

$$\log p_{\theta}(\mathbf{x}) = \mathcal{L}_{\theta, \phi}(\mathbf{x}) + \mathcal{D}_{KL}(q_{\phi}(\mathbf{z}|\mathbf{x}) || p_{\theta}(\mathbf{z}|\mathbf{x}))$$

Notando que $\mathcal{D}_{KL}(q_{\phi}(\mathbf{z}|\mathbf{x}) || p_{\theta}(\mathbf{z}|\mathbf{x})) \geq 0$,

$$\mathcal{L}_{\theta, \phi}(\mathbf{x}) \geq \log p_{\theta}(\mathbf{x})$$

La función de pérdida utilizada es la ELBO (evidence lower bound). Si maximizamos la ELBO con respecto a θ, ϕ estamos:

- ▶ maximizando la verosimilitud $p_{\theta}(\mathbf{x})$
- ▶ minimizando la KLD, i.e. $q_{\phi}(\mathbf{z}|\mathbf{x})$ se aproxima a $p_{\theta}(\mathbf{z}|\mathbf{x})$.

El truco de la reparametrización

En la práctica la ELBO la estimamos con una muestra de Monte Carlo,

$$\mathcal{L}_{\theta, \phi}(\mathbf{x}) = \log p_{\theta}(x_i|z) + \log p(z) - \log q_{\phi}(z|x_i)$$

Si realizamos muestras de $z = \mathcal{N}(\mu, \sigma)$, no podemos volver desde \mathbf{z} a θ , en términos del gradiente ya que las z son muestras.

Necesitamos independizar la parte aleatoria de la parte de los parámetros.
Hacemos una transformación:

$$z = \mu + \sigma \odot \epsilon$$

donde $\epsilon = \mathcal{N}(0, 1)$, muestra de una normal estandar.

Entonces estamos obligando que el espacio latente tenga media 0 y varianza 1, mientras μ y σ pasan a ser parte de los parámetros del codificador y el decodificador.

VAE en pytorch

```
class VAE(nn.Module):
    def __init__(self, shape, nhid = 16):
        super(VAE, self).__init__()
        self.dim = nhid
        self.encoder = Encoder(shape, nhid)
        self.decoder = Decoder(shape, nhid)

    def sampling(self, mean, logvar):
        eps = torch.randn(mean.shape).to(device)
        sigma = 0.5 * torch.exp(logvar)
        return mean + eps * sigma

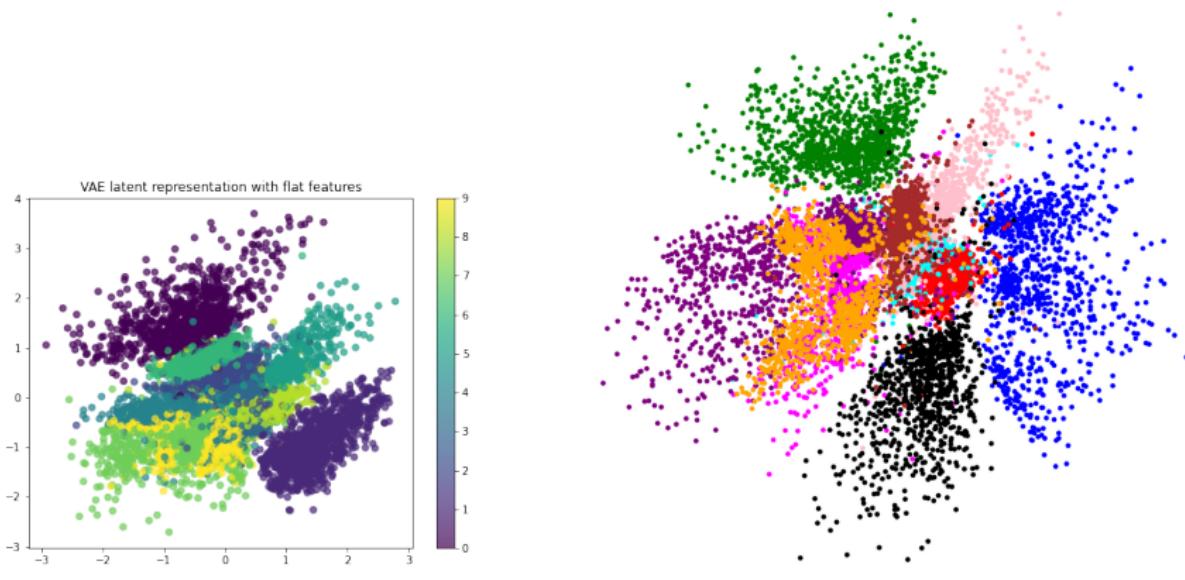
    def forward(self, x):
        mean, logvar = self.encoder(x)
        z = self.sampling(mean, logvar)
        return self.decoder(z), mean, logvar

    def generate(self, batch_size = None):
        z = torch.randn((batch_size, self.dim)).to(device)
        return self.decoder(z)
```



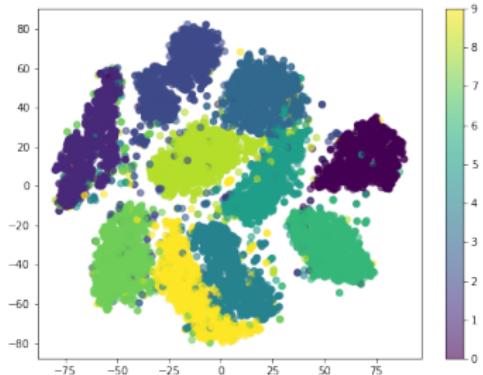
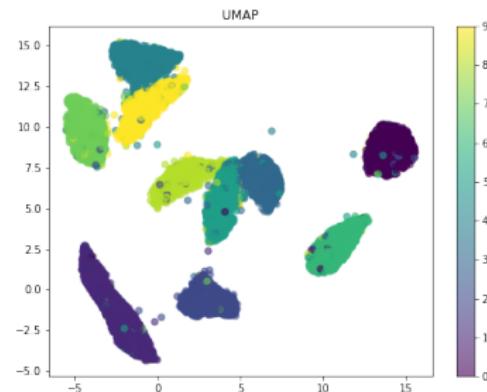
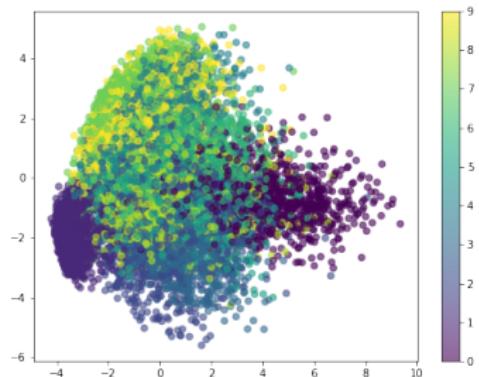
```
def loss_function(x, x_hat, mean, log_var):
    reproduction_loss = nn.functional.binary_cross_entropy(x_hat, x, reduction='sum')
    KLD = - 0.5 * torch.sum(1+ log_var - mean.pow(2) - log_var.exp())
    return reproduction_loss + KLD
```

Representación en el espacio latente



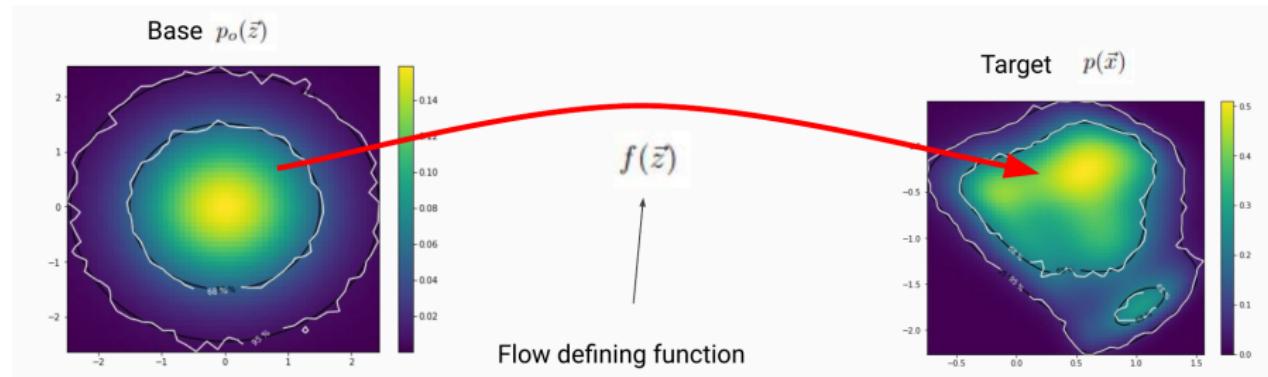
Esto podría pensarse como una reducción de la dimensionalidad de imágenes de 48x48 a 2 dimensiones.

Reducción de la dimensionalidad en MNIST.



PCA, UMAP y TNSE

Flujos normalizantes



Transformación de una distribución simple (normal) a la distribución de los datos

Rezende D, Mohamed S. Variational inference with normalizing flows. ICML 2015.

Esteban Takak (Courant Institute).

Cristina Turner (FaMAF, UNC).

Flujos normalizantes

- ▶ Trabaja con distribuciones arbitrariamente complejas.
- ▶ Funciona en alta dimensionalidad y sobre manifolds.
- ▶ Permite evaluar analiticamente la probabilidad.
- ▶ Genera muestras diferenciables y por lo tanto las esperanzas.

Tabak, E.G. and Turner, C.V., 2013. A family of nonparametric density estimation algorithms. CPAM.

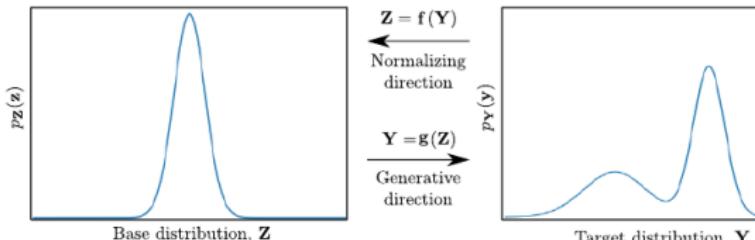
Pulido M., and P. J. vanLeewen, 2019: Sequential Monte Carlo with kernel embedded mappings: The mapping particle filter. JCP.

Flujos invertibles

Proponemos una transformación de variables $x = f_\phi(z)$.

Producido una transformación de densidades:

$$p_\phi(x) = \left| \det \frac{\partial f_\phi(z)}{\partial z} \right|^{-1} p(z)$$



Si tenemos esta transformación luego $z^* \sim p(z) \rightarrow x^* = f_\phi(z^*)$

Requerimos la inversa para evaluar la verosimilitud $z = f_\phi^{-1}(x)$.

Función biyectiva para que exista la inversa + Función diferenciable → Diffeomorfismo.

$$p(x) = p_0(f^{-1}(x)) \left| \det \frac{\partial f_\phi^{-1}(x)}{\partial x} \right|$$

Entrenamiento

Dada las muestras el negativo de la log-likelihood:

$$nll(x) = \sum_{n=1}^N \left[\log \left| \det \frac{\partial f_\phi(z_n)}{\partial z} \right| - \log p(z_n) \right]$$

donde $z_n = f_\phi^{-1}(x_n)$ son la transformación de cada una de las muestras.

En general se asume que $f_\phi(z)$ es una red neuronal con múltiples capas k , luego

$$x = f_K(f_{K-1}(\dots f_1(z)))$$

Potenciales capas de la red:

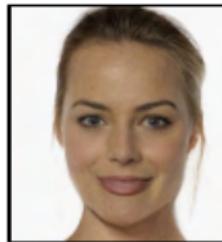
- ▶ Flujos lineales
- ▶ Flujos por componentes nolineales.
- ▶ Flujos autoregresivos
$$h_d = g[h_d, \phi(h_{1:d-1})]$$
- ▶ Flujos residuales

$$z = f_\phi^{-1}(x) = f_1^{-1}(f_2^{-1}(\dots f_K^{-1}(x) \dots))$$

$$\left| \frac{\partial f_\phi(z_n)}{\partial z} \right| = \left| \frac{\partial f_k}{\partial f_{k-1}} \right| \cdot \left| \frac{\partial f_{k-1}}{\partial f_{k-2}} \right| \dots \left| \frac{\partial f_1}{\partial z} \right|$$

Aplicación de flujos normalizantes

Generación de imágenes sintéticas

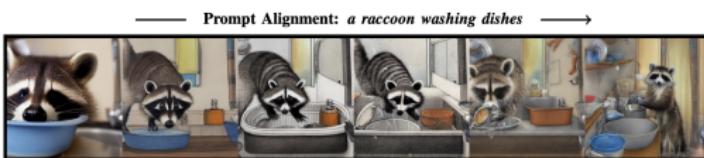


Modelos difusivos

Estado del arte en modelos generativos.

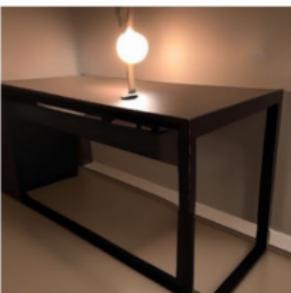
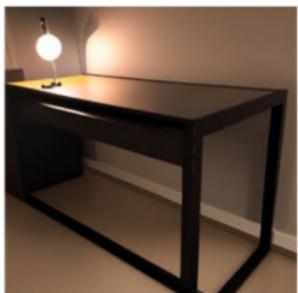
DALL-E (openai), StableDiffusion (StabilityAI), Imagen (google),
Midjourney, DreamFusion (google), Magic3D (Nvidia), Virtual reality

- ▶ Generación de imágenes a partir de texto,
- ▶ Generación de videos de música
- ▶ Imágenes para eventos y sitios web.
- ▶ Generación de imágenes artísticas.



Tomadas de Chen et al. 2024.

Edicion de imagenes 3D!

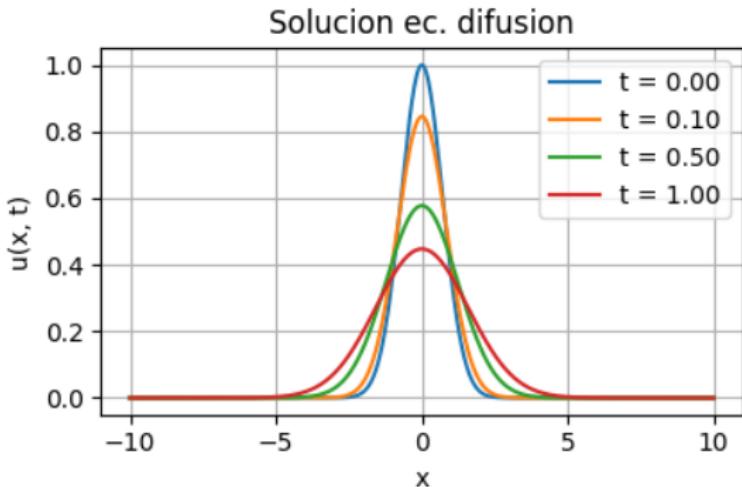


Pandey et al. Diffusion Handles Enabling 3D Edits for Diffusion Models by Lifting Activations to 3D, CVPR 2024

Ecuación de difusión en física

Ecuación de difusión:

$$\partial_t \phi - \nabla^2 \phi = 0.$$



Solución usando el método de la función de Green:

$$\phi(\mathbf{x}, t) = \frac{1}{(2\pi t)^{N/2}} \int \phi(\mathbf{x}', 0) \exp\left(-\frac{|\mathbf{x} - \mathbf{x}'|^2}{2t}\right) d\mathbf{x}'$$

Procesos difusivos en física

Definición: Un proceso difusivo es un proceso de Markov con trayectorias de muestras continuas gobernado por la ecuación de Fokker-Planck.

$$\frac{\partial p(x,t)}{\partial t} = -\frac{\partial}{\partial x} [v(x,t)p(x,t)] + \frac{\partial^2}{\partial x^2} [D(x,t)p(x,t)]$$

Ecuación diferencial estocástica:

$$dx = v(x,t) dt + \sqrt{2D(x,t)} dW_t$$

donde dW_t es un proceso de Wiener.

Ejemplo: **Movimiento Browniano.** Trayectoria de una partícula en un flujo dinámico sujeta a desplazamientos aleatorios debido a las colisiones con otras partículas del fluido.

Modelos probabilísticos de difusión y eliminación de ruido

Proceso de difusión e inclusión de ruido



Tengo una cadena de Markov de procesos secuenciales:

$$p(\mathbf{x}_{0:T}) = \prod_{i=0}^T p(\mathbf{x}_t | \mathbf{x}_{t-1}),$$

Asumo que en cada paso le agrego ruido blanco, cada una de estas transiciones viene dada por

$$p(\mathbf{x}_t | \mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; (1 - \beta_{t-1})\mathbf{x}_{t-1}, \beta_{t-1}\mathbf{I})$$

En cada paso para imagen i hago:

$$\mathbf{x}_1^{(i)} = (1 - \beta_0)\mathbf{x}_0^{(i)} + \boldsymbol{\epsilon}_0$$

donde $\boldsymbol{\epsilon}_0 \sim \mathcal{N}(0, \beta_0\mathbf{I})$.

kernel de difusión

Si hacemos un gran número de pasos, generalmente $T = 1000$, $p(\mathbf{x}_T) = \mathcal{N}(0, \mathbf{I})$. Convergencia a una normal estandar.

En teoría converge a la normal para $T \rightarrow \infty$ y β_t es muy pequeño.
Componiendo varios pasos a la vez:

$$p(\mathbf{x}_2|\mathbf{x}_0) = \int p(\mathbf{x}_2, \mathbf{x}_1|\mathbf{x}_0) d\mathbf{x}_1 = \int p(\mathbf{x}_2|\mathbf{x}_1)p(\mathbf{x}_1|\mathbf{x}_0) d\mathbf{x}_1$$

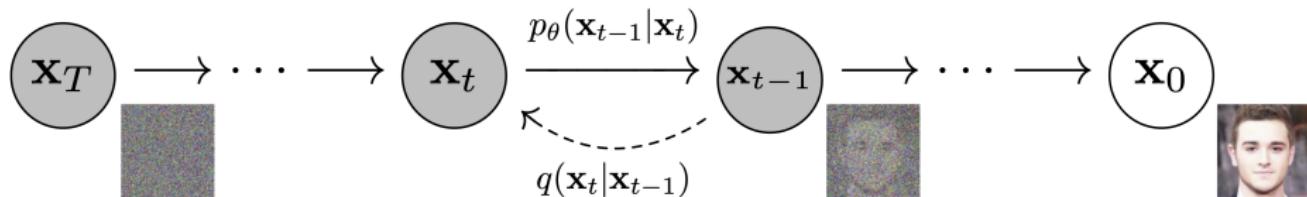
Una propiedad interesante de los procesos gaussianos: el producto de dos gaussianas es una nueva gaussiana.

$$p(\mathbf{x}_{t+1}|\mathbf{x}_0) = \mathcal{N}\left(\alpha_t^{1/2}\mathbf{x}_0, [1 - \alpha_t]\mathbf{I}\right)$$

donde $\alpha_t = \prod_{s=0}^t (1 - \beta_s)$ este es denominado el *kernel de difusión*.

Usando el kernel de difusión podemos muestrear directamente a \mathbf{x}_t sin tener que realizar toda la cadena de Markov desde $t = 0$.

Proceso reverso: denoising



Ahora vamos hacia atrás de t a $t - 1$

$$p_{\theta}(\mathbf{x}_{0:T}) = \prod_{t=1}^T p_{\theta_t}(\mathbf{x}_{t-1} | \mathbf{x}_t) p(\mathbf{x}_T)$$

Cada pasito lo representamos por una gaussiana parametrizada con $\mu(\mathbf{x}_t, \theta)$,

$$p_{\theta_t}(\mathbf{x}_{t-1} | \mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu(\mathbf{x}_t, \theta_t), \Sigma(\mathbf{x}_t, \theta_t)),$$

θ son los parametros de una red neuronal

Entrenamiento

En un modelo de difusión solo tenemos una red neuronal para producir el denoising.

¿Como hacemos para estimar sus parámetros?

Los datos son $\mathbf{x} = \mathbf{x}_0$ y luego todas las otras son variables latentes y las tenemos que marginalizar:

$$p(\mathbf{x}_0 | \boldsymbol{\theta}) = \int p(\mathbf{x}_0, \mathbf{x}_{1:T} | \boldsymbol{\theta}) d\mathbf{x}_{1:T}$$

Luego tenemos que maximizar la verosimilitud con respecto a los parámetros

En principio entonces la solución viene dada por

$$\boldsymbol{\theta} = \operatorname{argmax} \sum_i^{N_D} \log(p(\mathbf{x}^{(i)} | \boldsymbol{\theta}))$$

ELBO: Evidence Lower Bound

La verosimilitud es intratable, no podemos representar la integral de todas las variables latentes.

Similar al caso de los VAEs:

$$\log p_{\theta}(\mathbf{x}) = \mathcal{L}_{\theta}(\mathbf{x}) + \mathcal{D}_{KL}(q(\mathbf{z}|\mathbf{x}) || p_{\theta}(\mathbf{z}|\mathbf{x}))$$

Notando que $\mathcal{D}_{KL} \geq 0$ se tiene que

$$\log p_{\theta}(\mathbf{x}) = \mathcal{L}_{\theta}(\mathbf{x})$$

Entonces lo que se quiere hacer es maximizar la ELBO y de esta manera estamos obteniendo cotas que se acerca al maximo de la verosimilitud.

En lugar de predecir la imagen predecimos el ruido

Ho, et al (2020) hicieron una transformacion que permite en lugar de predecir con la red la imagen latente en cada paso lo que predice es el ruido que fue adicionado en cada paso:

$$x_0 = \frac{1}{\sqrt{\alpha_t}} \mathbf{x}_t - \frac{\sqrt{1 - \alpha_t}}{\sqrt{\alpha_t}} \epsilon_t$$

La red ahora predice el ruido: $\epsilon(\mathbf{x}_t, \theta, t)$.

En un primer paso se tiene que

$$\log p(\mathbf{x}|x_1, \theta) = -\frac{1}{2(1 - \beta_1)} \|\epsilon(\mathbf{x}_1, \theta, t=1) - \epsilon_1\|^2$$

Finalmente sumando todos los terminos la funcion de perdida, la ELBO, para el entrenamiento despreciando los factores

$$\mathcal{L}_\theta(\mathbf{x}) = - \sum_{t=1}^T \|\epsilon(\sqrt{\alpha_t} \mathbf{x} + \sqrt{1 - \alpha_t} \epsilon_1, \theta, t) - \epsilon_1\|^2$$

se relaciona directamente a los datos. En cada paso estimamos el

Algoritmo de entrenamiento

Para cada iteración de un total de N_I :

1. Muestrear un dato: $\mathbf{x}_0 \sim p(\mathbf{x}_0)$, es $p(\mathbf{x}_0)$ la distribución de los datos.
2. Tomar un paso aleatorio de la difusión: $t \sim \text{Uniform}(\{1, \dots, T\})$.
3. Tomo una muestra del ruido: $\epsilon \sim \mathcal{N}(0, I)$.
4. Genera la imagen con ruido de \mathbf{x}_0 :

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon$$

5. Se predice el ruido usando la red neuronal: $\epsilon_\theta(\mathbf{x}_t, t)$.
6. Calcula la fn de pérdida:

$$\mathcal{L} = \mathbb{E}_{x_0, \epsilon, t} [\|\epsilon - \epsilon_\theta(\mathbf{x}_t, t)\|^2]$$

Algoritmo de inferencia

Asumiendo ya tenemos entrenada la red neuronal $\epsilon_\theta(\mathbf{x}_t, t)$.

Proceso de difusión reverso:

1. Comenzando de un campo/imagen de puro ruido: $\mathbf{x}_T \sim \mathcal{N}(0, I)$.
2. Para $t = T, T - 1, \dots, 2$:
 - ▶ Se predice el ruido con la red: $\hat{\epsilon} = \epsilon_\theta(x_t, t)$.
 - ▶ Se calcula la media de la densidad posterior:

$$\boldsymbol{\mu}_t = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1 - \bar{\alpha}_t}} \hat{\epsilon} \right)$$

- ▶ Muestrear $z \sim \mathcal{N}(0, I)$ y calcular:

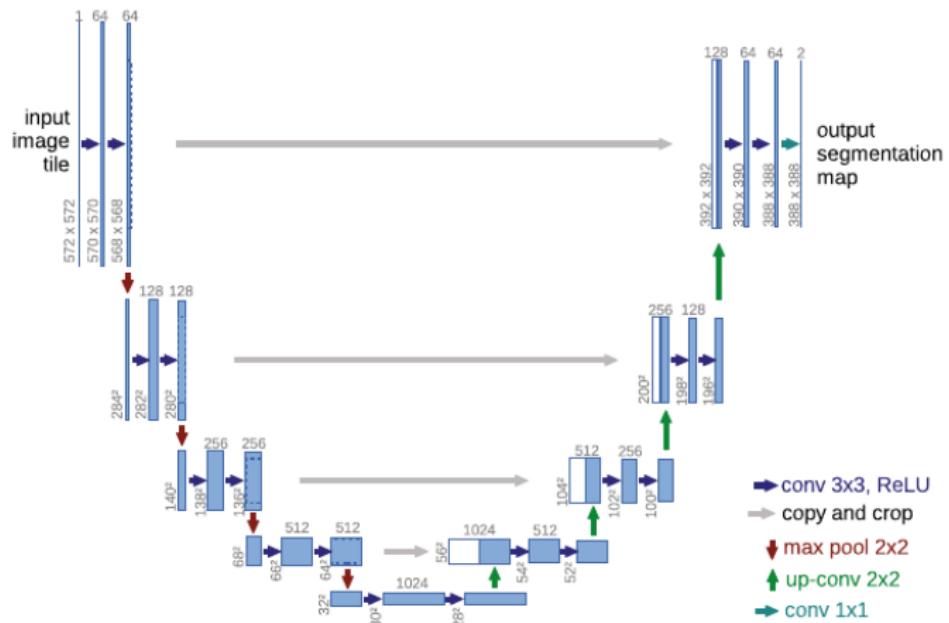
$$\mathbf{x}_{t-1} = \boldsymbol{\mu}_t + \sqrt{\tilde{\beta}_t} \mathbf{z}$$

Para el último tiempo, $t = 1$

$$\mathbf{x}_0 = \boldsymbol{\mu}_1$$

Arquitectura de la red utilizada en modelos difusivos

Para generación de imágenes en la cual necesitamos una imagen de entrada y una imagen de salida de la red, que estructura usamos?



Difusión guiada

Hasta el momento la técnica solo permite generar una muestra sintética al azar a partir de haber aprendido la distribución de los datos.

Como debería hacer si lo que requiero es graficar dentro de la base de datos ImageNet un perro sintético.

$$p(\mathbf{x}|\mathbf{c})$$

Otra posibilidad es que \mathbf{c} sea un texto que nos permita generar imágenes de acuerdo a lo que requiera el usuario.

⇒ se podría meter el \mathbf{c} también como variable de entrada.

Difusión guiada

Score matching o matcheado de Stein: $s(\mathbf{x}) = \nabla \log p(\mathbf{x})$

En el caso de la guiada:

$$s(\mathbf{x}, \mathbf{c}) = \nabla \log p(\mathbf{x}|\mathbf{c}) = \nabla \log p(\mathbf{x}) + \nabla \log p(|\mathbf{x}) - \nabla \log p(|) \approx$$

Para controlar la convergencia hacia el guiado

$$s(\mathbf{x}, \mathbf{c}) = \nabla \log p(\mathbf{x}) + \lambda \nabla \log p(|\mathbf{x})$$

Mas alla que podemos apurar la convergencia con λ grande, hay que tener cuidado porque estamos disminuyendo la diversidad (la creatividad).

Para modelar el segundo término requerimos de otra red neuronal para clasificar además de la propia de difusión (primer término).

Difusión guiada libre

$$s(\mathbf{x}, \mathbf{c}) = \lambda \nabla \log p(\mathbf{x}) + (1 - \lambda) \nabla \log p(\mathbf{x}|\mathbf{c})$$



Ho y Salimans 2022 proponen usar una sola red sin clasificador, lo único que hace es cuando corresponde no ponerles clasificador $\mathbf{c} = 0$.

El método fusión guiada libre de clasificador es el más utilizado para generación por text2image y para image2image.

Ejemplos: DALL-E 2 e Imagen.