Modelos Generativos 2

Material auxiliar

Máster en Inteligencia Artificial aplicada a Mercados Financieros

Jorge del Val

Índice

1. Recap
2. Variational Autoencoders
3. Generative Adversarial Networks
4. Bonus: Autoregressive Models

Recap

Recordando…

Para nosotros, cada dato x_i es una *realización* de una variable aleatoria subyacente $\mathbf{x}$, con una distribución de probabilidad $p(x)$ desconocida

$$\mathbf{x} \sim p(x)$$

- El *aprendizaje no supervisado* es el campo que intenta inferir propiedades de $\mathbf{x}$ sólo con las muestras (datos).
- Los *modelos generativos* son un subconjunto del aprendizaje no supervisado que pretende *aproximar* $\mathbf{x}$ como una combinación de variables aleatorias "simples" que se puedan muestrear:

$$\mathbf{x} \approx G_\theta(\mathbf{z_1}, \mathbf{z_2}, \dots, \mathbf{z_k}) \triangleq \hat{\mathbf{x}}$$

Entrenamiento

Queremos aproximar $\mathbf{x}$ como $\hat{\mathbf{x}} = G_\theta(\mathbf{z})$. ¿Cómo encontramos los parámetros θ óptimos?

Maximiza la verosimilitud (likelihood) de tu modelo!!

Probabilidad de que tu modelo generara x_i

$$\max_\theta \mathcal{L}(\theta | x_{train}) = \max_\theta \prod_{i=1}^{N} p_\theta(x_i)$$

$$\max_\theta \log \mathcal{L}(\theta | x_{train}) = \max_\theta \sum_{i=1}^{N} \log p_\theta(x_i)$$

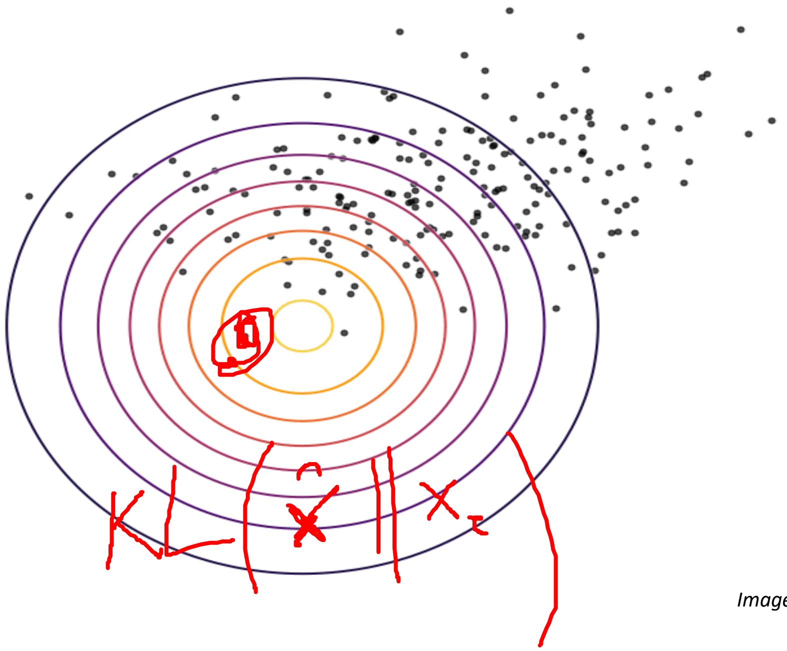
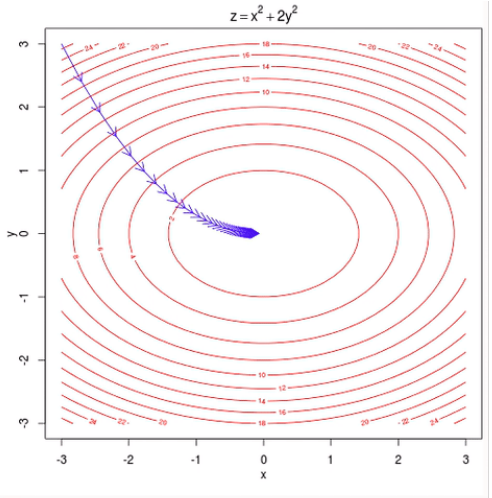


Image credit: Colin Raffel

Necesitamos $p_\theta(x)$ explícitamente!

Optimización



Optimizamos una función de coste (error)!

Stochastic Gradient Descent

$$\min_{\theta} L(\theta; \text{data}) \rightarrow \boxed{\theta_{t+1} = \theta_t - \alpha \nabla_\theta L}$$

```python
model = MyNetwork()
theta = model.parameters()
optimizer = torch.optim.Adam(theta, lr=0.001)

for x, y in dataloader:
    y_pred = model(x)
    loss = myloss(y, y_pred)
    loss.backwards()
    optimizer.step()
```

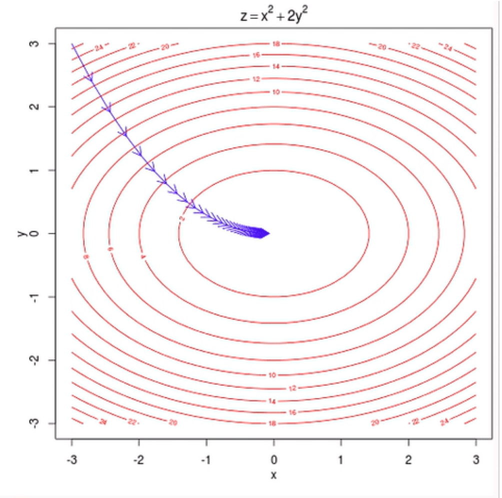
PyTorch

```python
model = MyNetwork()
theta = model.trainable_variables
optimizer = tf.train.AdamOptimizer(lr = 0.001)

for x, y in dataset:
    with tf.GradientTape() as g
        y_pred = model(x)
        loss = myloss(y, y_pred)
    grads = g.gradient(loss, theta)
    optimizer.apply_gradients(zip(grads, model.trainable_variables))
```

TensorFlow

Optimización



Optimizamos una función de coste (error)!

Stochastic Gradient Descent

$$\min_{\theta} L(\theta; \text{data}) \rightarrow \boxed{\theta_{t+1} = \theta_t - \alpha \nabla_{\theta} L}$$

```
model = MyNetwork()
theta = model.parameters()
optimizer = torch.optim.Adam(theta, lr=0.001)

for x, y in dataloader:
    y_pred = model(x)
    loss = myloss(y, y_pred)
    loss.backwards()
    optimizer.step()
```

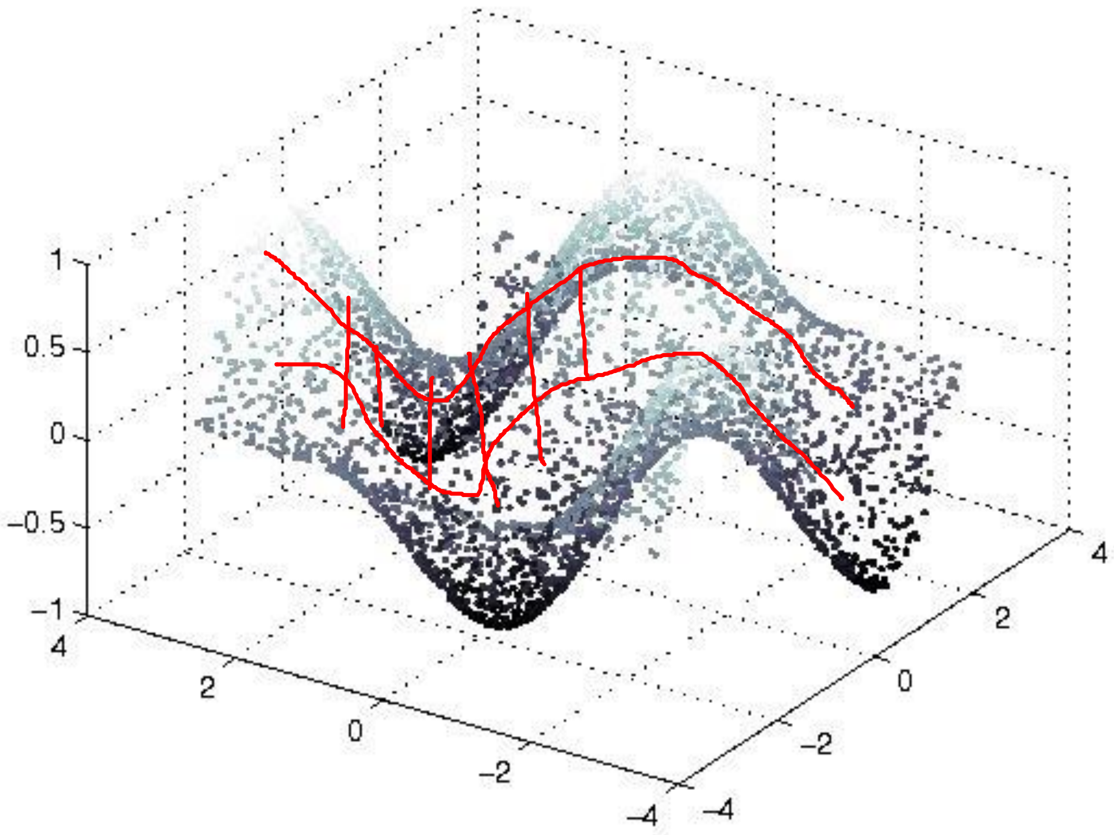
PyTorch

```
model = MyNetwork()
theta = model.trainable_variables
optimizer = tf.train.AdamOptimizer(lr = 0.001)

    loss = myloss(y, y_pred)
grads = g.gradient(loss, theta)
optimizer.apply_gradients(zip(grads, model.trainable_variables))
```

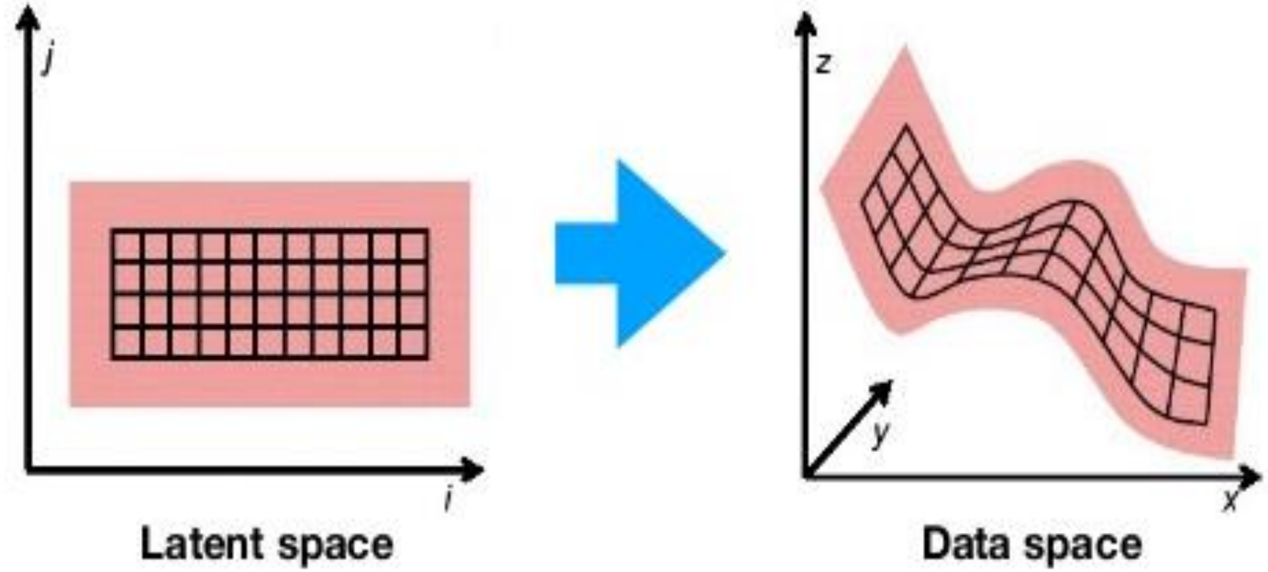
TensorFlow

Easy to gradient descent any function with current frameworks!! ☺



1
0.5
0
−0.5
−1
4
2
0
−2
−4
−4
−2
0
2
4

Dimensiones latentes de los datos

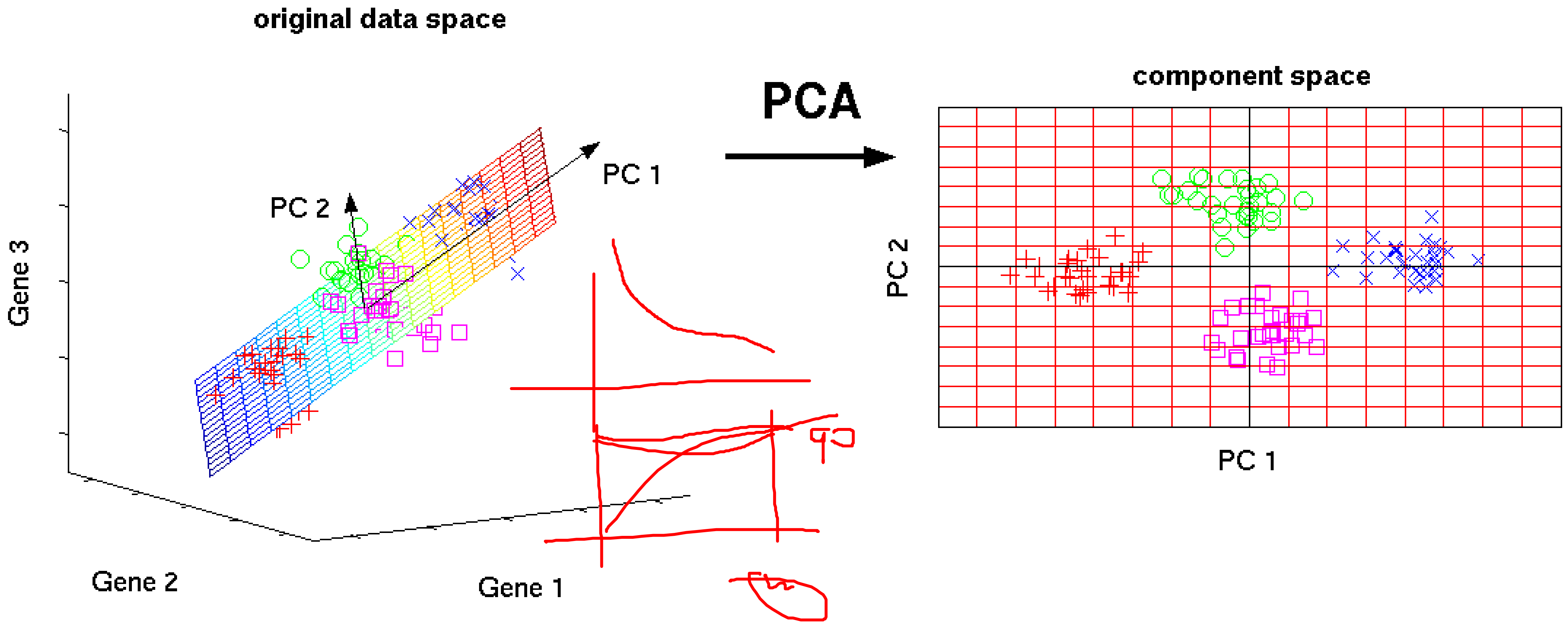


Z

X

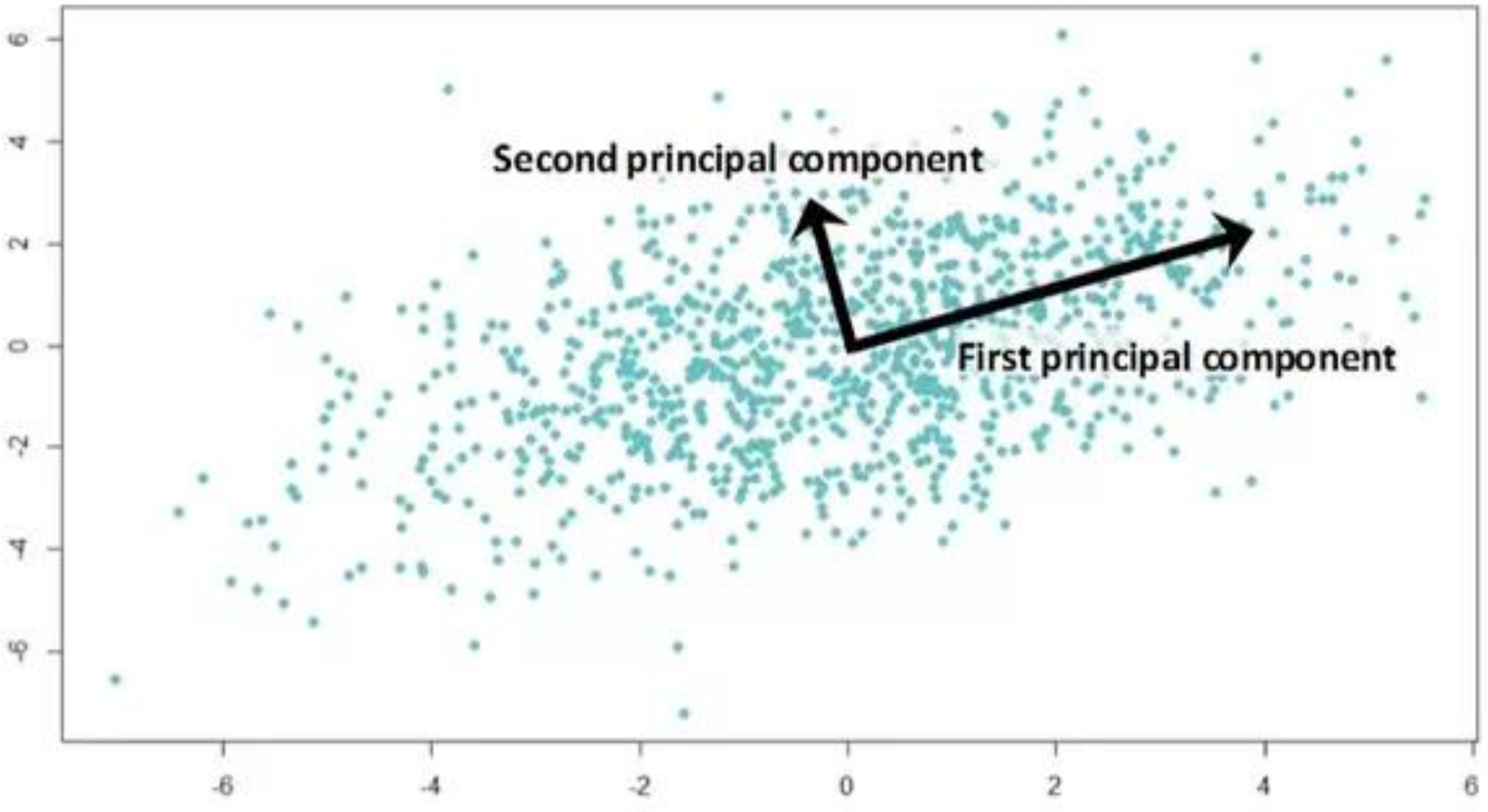
Images credit Ward A.D. et al 2007

Principal Component Analysis

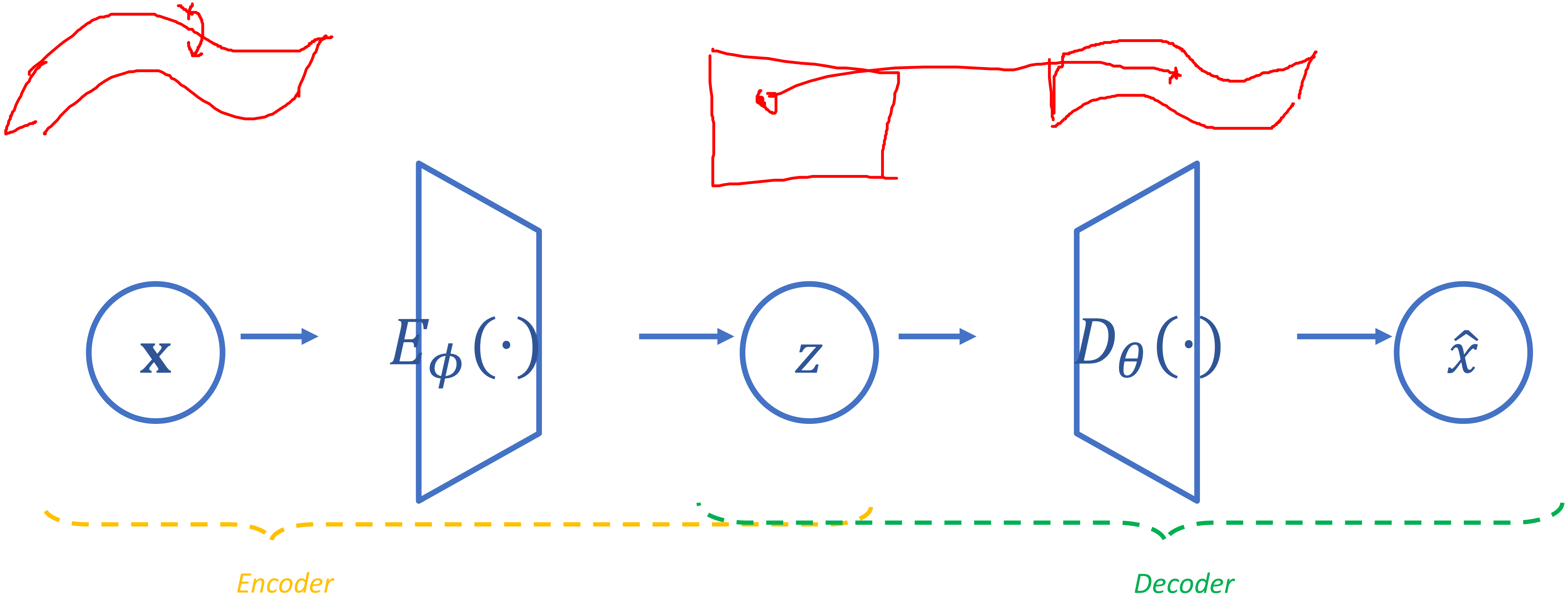


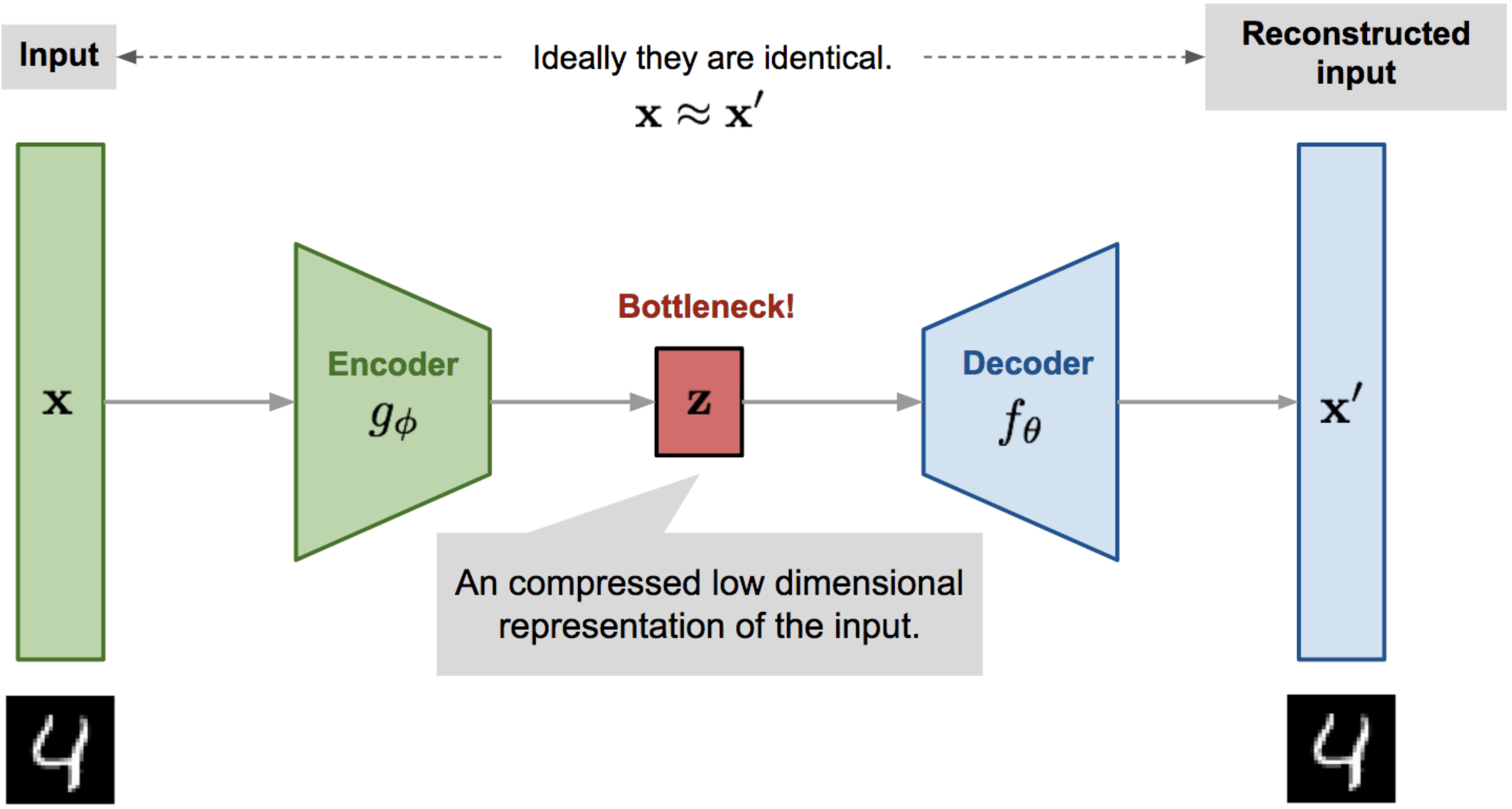
PCA asume que el manifold es lineal!

Principal Component Analysis



Autoencoders

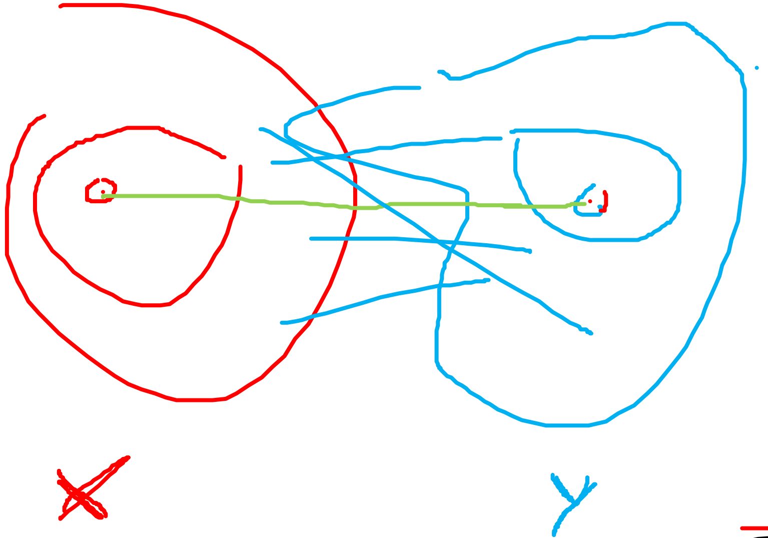




… pero no podemos usar la variable *z* para muestrear; ninguna relación con la distribución de probabilidad ☹

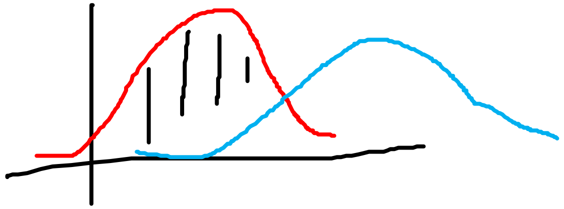
$\mathbb{E}[\ ]$

Let’s go mathy!: Divergencia de Kullback-Leibler (KL)



$\int |p(x)-p(y)|\,dx$

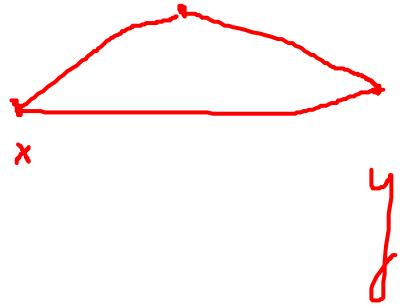
$d(x,y)$



Cómo medimos la “distancia” entre dos distribuciones?

$x \simeq$

$\mathbb{E}[x -$

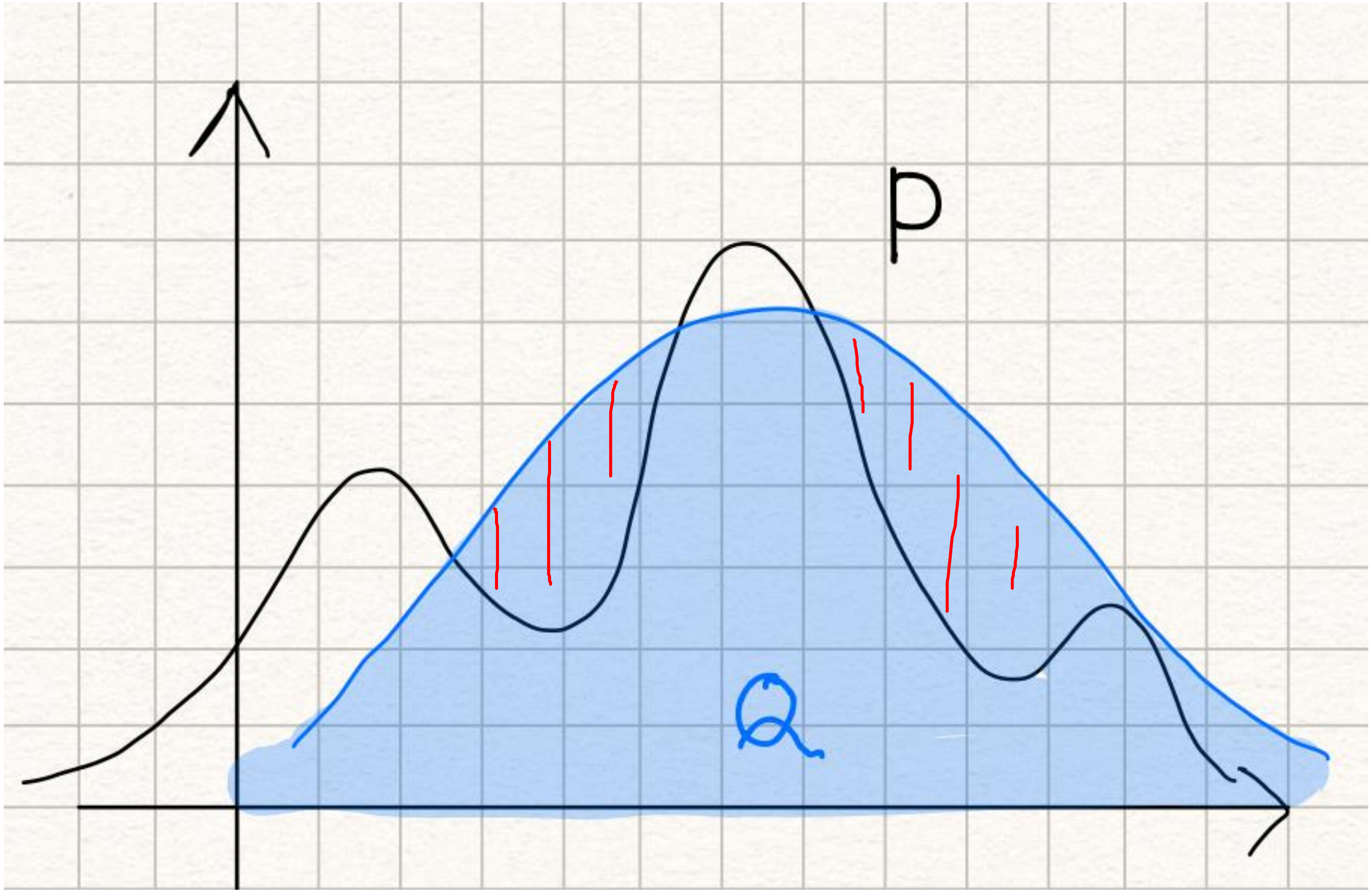


$$KL(p||q) = \mathbb{E}_{x\sim p(x)}\left[\log \frac{p(x)}{q(x)}\right] = \int_x p(x)\log\frac{p(x)}{q(x)}\,dx$$

$\log p - \log q$

$KL(p||q) \neq KL(q||p)$

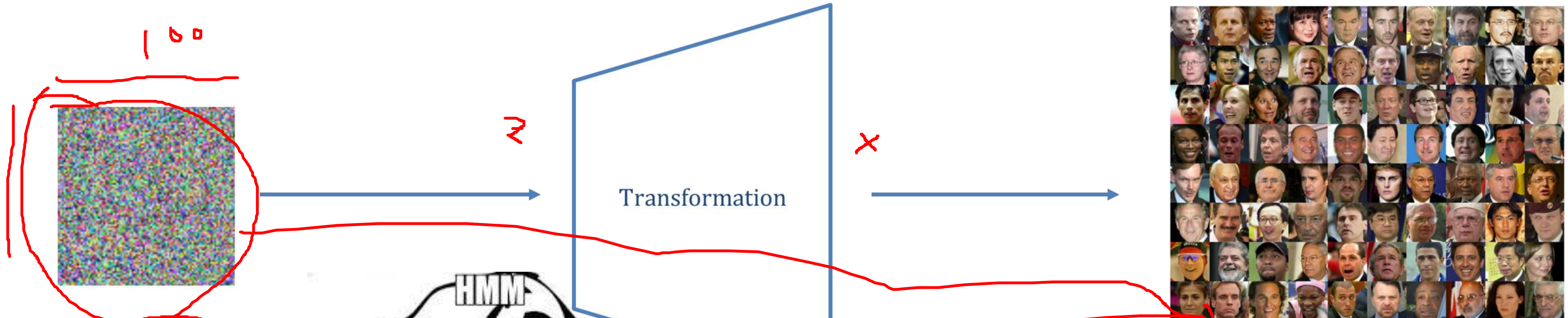
$KL \geq 0$


P
Q

Variational Autoencoders

Lo que queremos!

$z \sim N(\ , \ )$

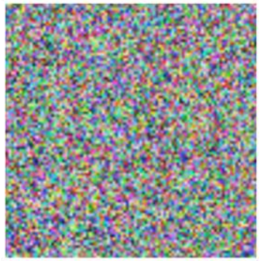


$\mathbf{z} \sim p(z)$

Prior distribution

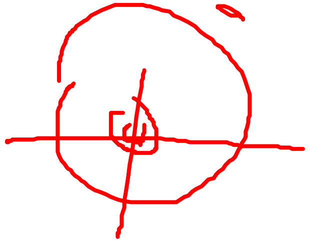
$\theta - x = G(z)$

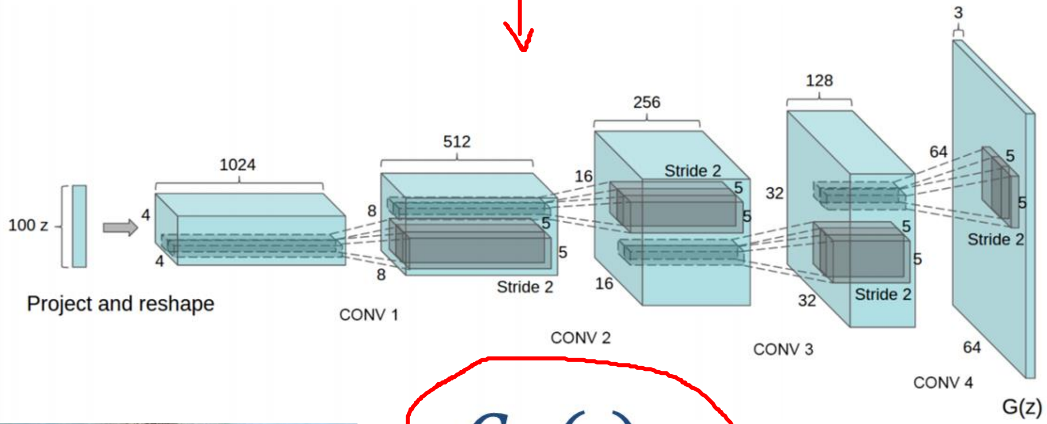
Deep Latent Variable Models



$\mathbf{z} \sim p(z)$

Prior distribution





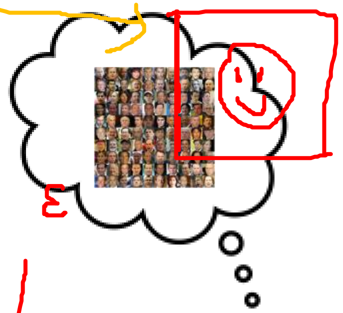
$G_\theta(\cdot)$

$\hat{\mathbf{x}} = G_\theta(\mathbf{z})$

Genial! Optimicemos la likelihood!

$\mathbf{z}$ $\longrightarrow$ $G_\theta(\cdot)$ $\longrightarrow$ $\hat{\mathbf{x}}$

$p(z)$

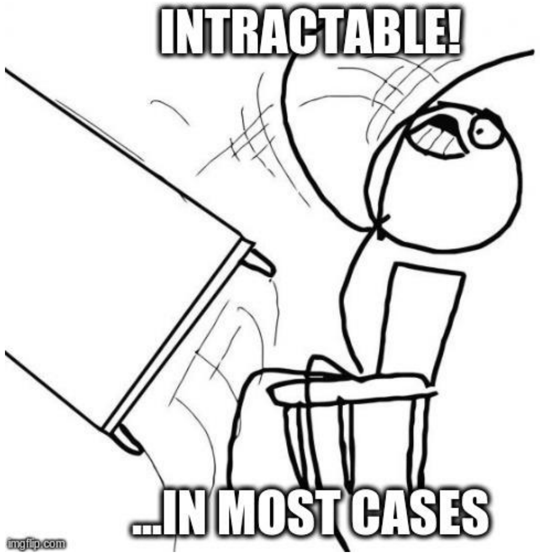


$p_\theta(x|z)$

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

$\boxed{p_\theta(x)??}$

Cuál es la probabilidad **total** de generar x?



Diferentes modelos – diferentes métodos

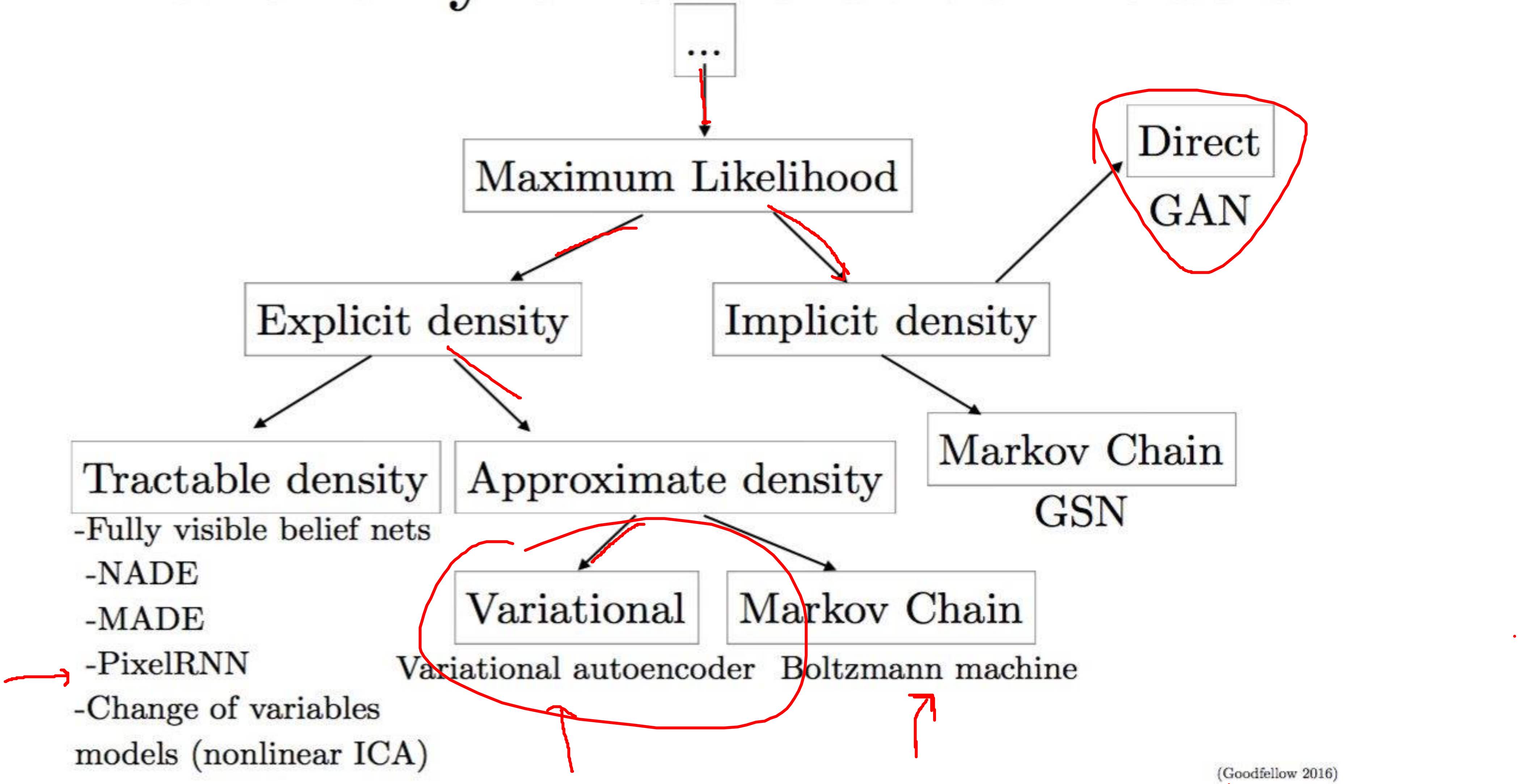
GMM

1. Tenemos $p_\theta(\hat{x})$ explícitamente: **maximizamos likelihood**.

2. $\underline{p_\theta(\hat{x})}$ es intratable: lo podemos aproximar
   - Markov Chain Monte Carlo (MCMC) methods
   - Variational methods (e.g. Variational Autoencoders)

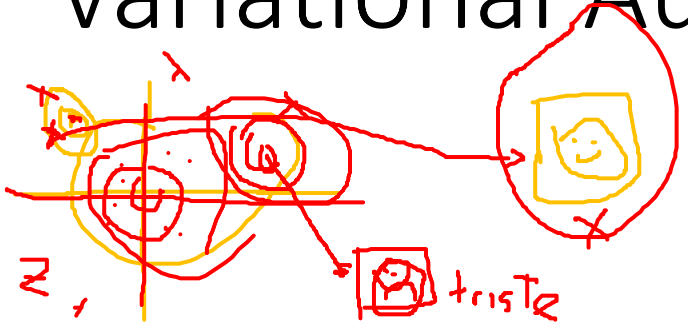
3. No necesitamos $p_\theta(\hat{x})$; está implícito!
   - Adversarial methods (e.g. GANs)

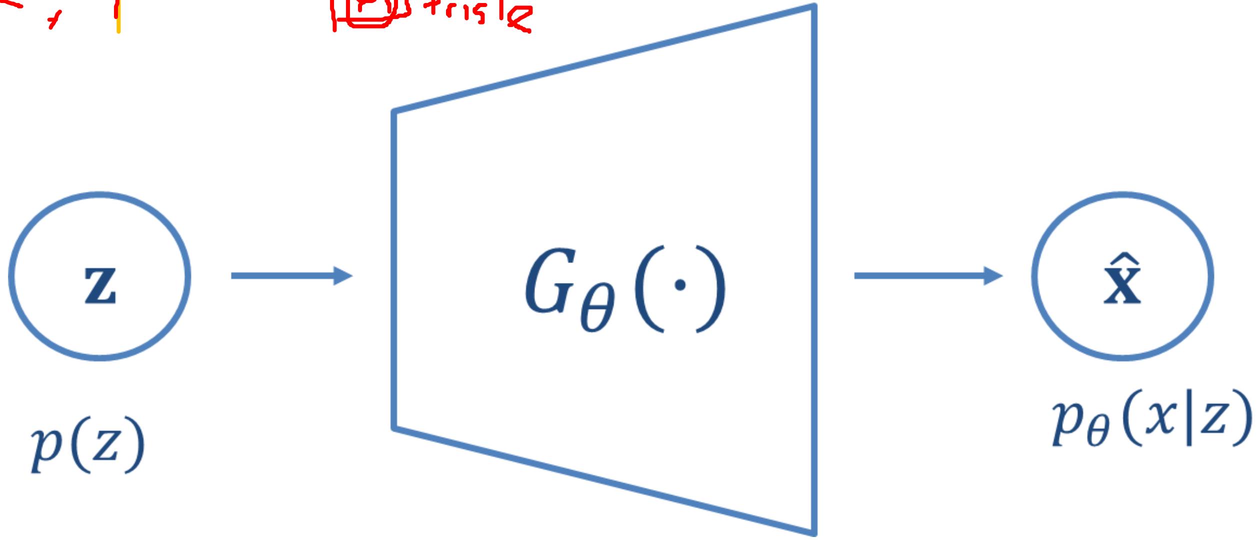
Taxonomy of Generative Models



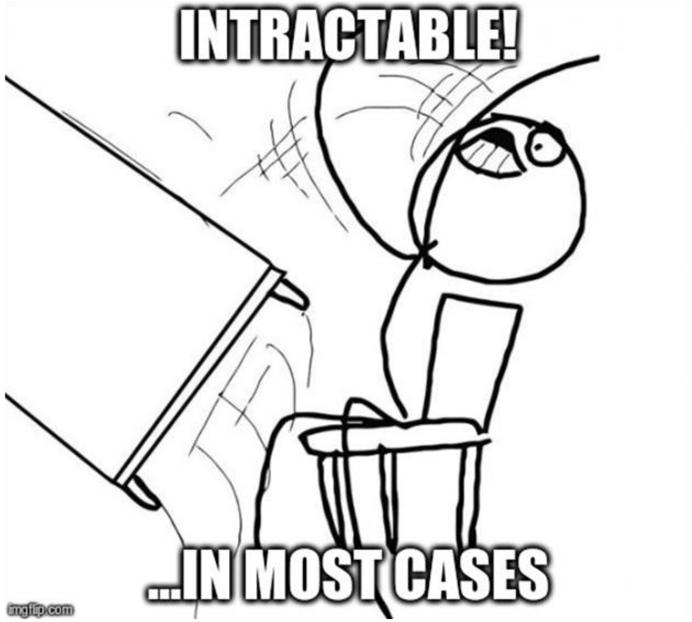
(Goodfellow 2016)

Variational Autoencoder

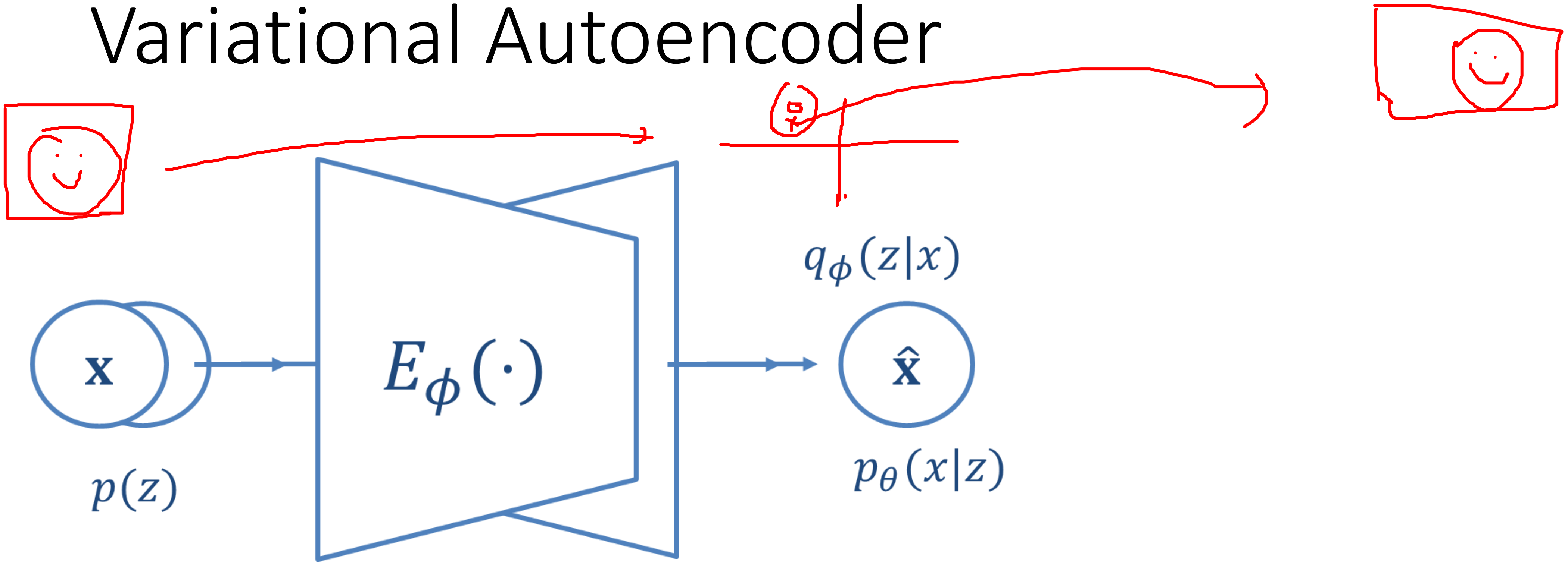




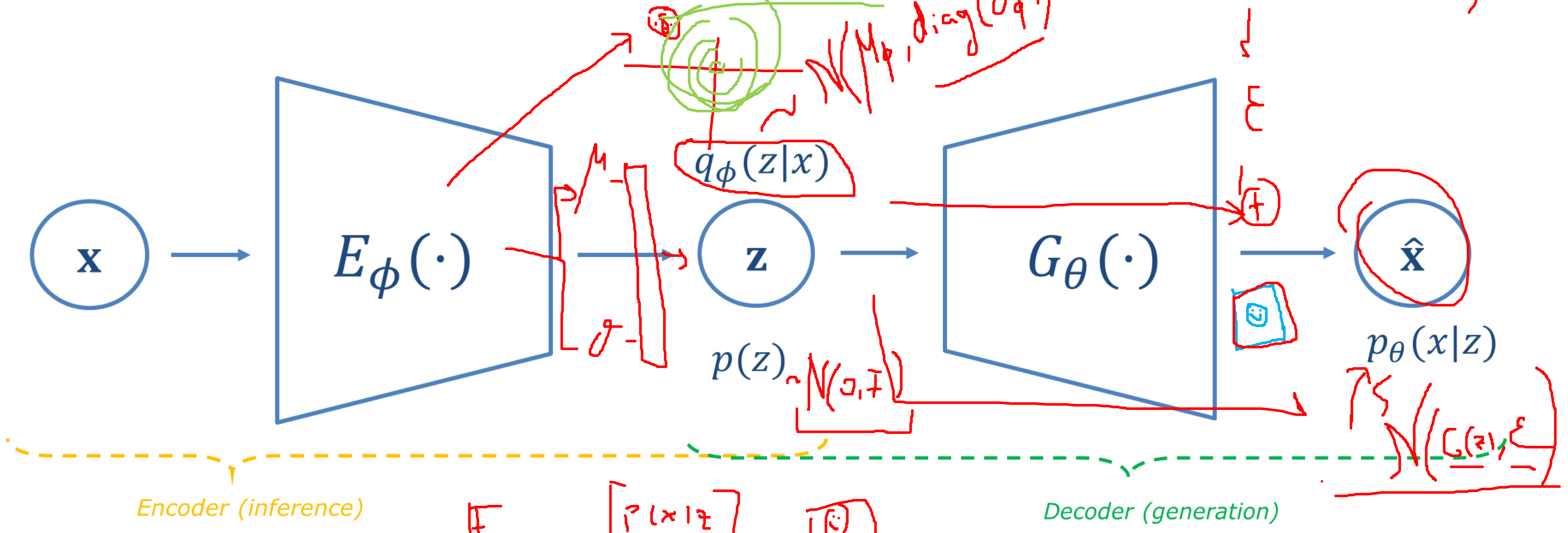
$$p_\theta(x) = \int_{Z} p_\theta(x|z)\, p(z) dz$$



Variational Autoencoder



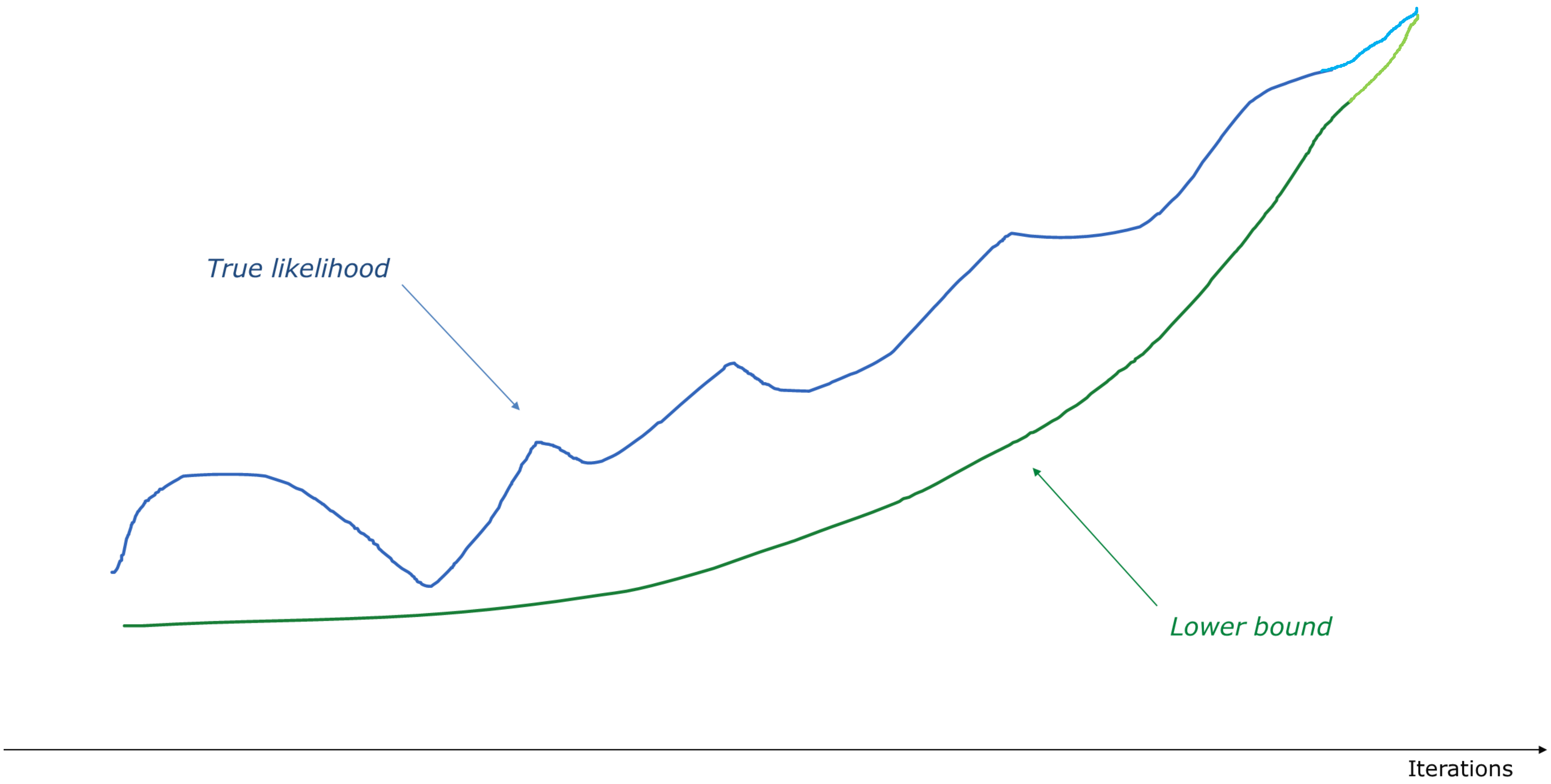
Variational Autoencoder



$$\log p_\theta(x) \geq \underbrace{\boxed{\mathbb{E}_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)] - \mathrm{KL}[q_\phi(z|x) \parallel p(z)]}}_{\text{ELBO}}$$

Maximize this instead!

$\rightarrow \|\hat{x}-x\|^2/\varepsilon$


True likelihood
Lower bound
Iterations

¿Por qué? *Nerd warning*

$$\log p_\theta(x) =$$

$$\log \int_z p_\theta(x|z)p(z)dz = \log \int_z p_\theta(x|z)p(z)\,\frac{q(z|x)}{q(z|x)}\,dz = \log \mathbb{E}_{z\sim q}\left[\frac{p_\theta(x|z)p(z)}{q(z|x)}\right] \leq$$

$$\mathbb{E}_{z\sim q}\left[\log \frac{p_\theta(x|z)p(z)}{q(z|x)}\right] = \mathbb{E}_{z\sim q}\left[\log p_\theta(x|z) - \log \frac{p(z)}{q(z|x)}\right] = \mathbb{E}_{z\sim q}[\log p_\theta(x|z)] +$$

$$\mathbb{E}_{z\sim q}\left[\log \frac{q(z|x)}{p(z)}\right] = \mathbb{E}_{z\sim q}[\log p_\theta(x|z)] - \mathrm{KL}\big(\mathrm{q(z|x)}\,||\,\mathrm{p(z)}\big)$$

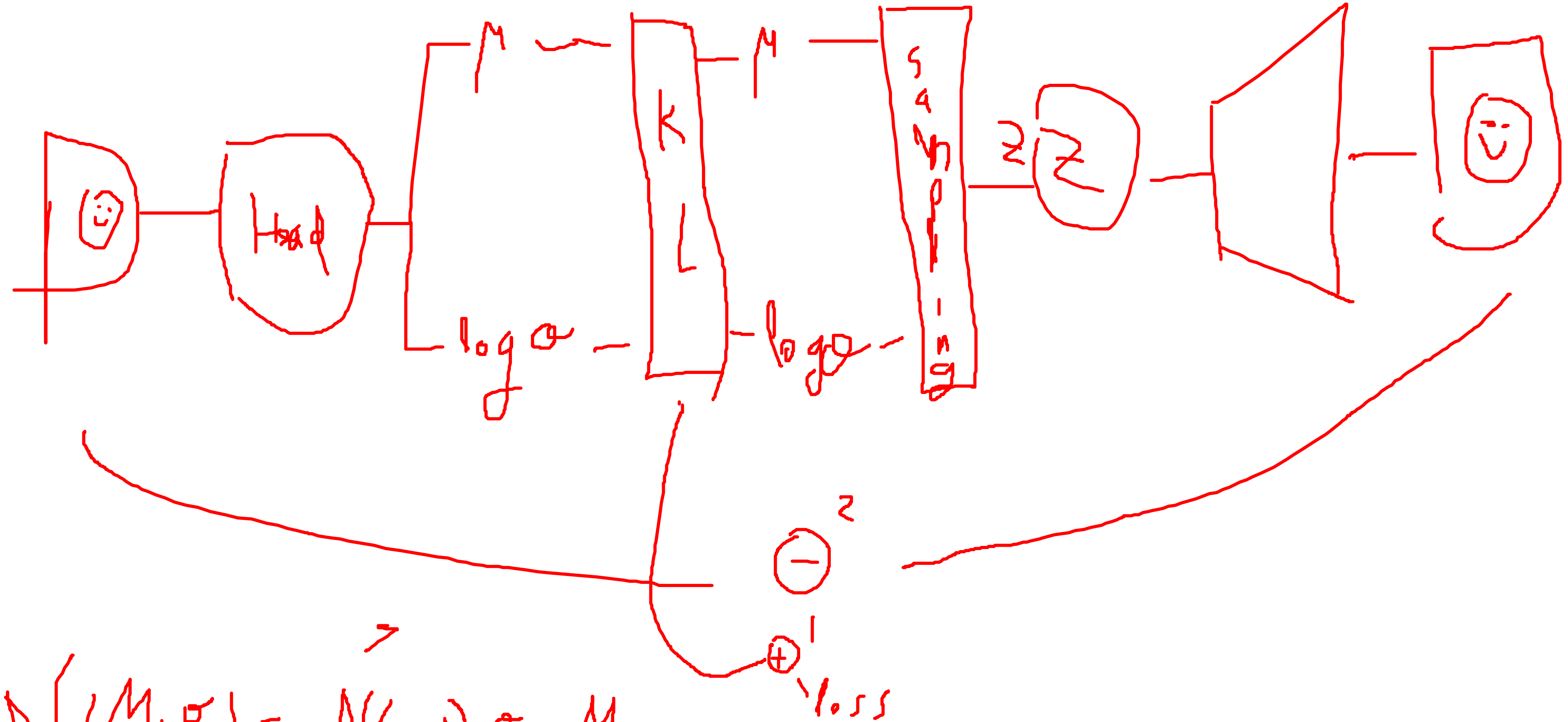
Variational Autoencoders

Pros:

- Inferencia eficiente gratis!
    - Buena herramienta para modelar la estructura interna de los datos
- Entrenamiento estable
- Buen fundamento teórico

Cons:

- No genera muy buenas muestras

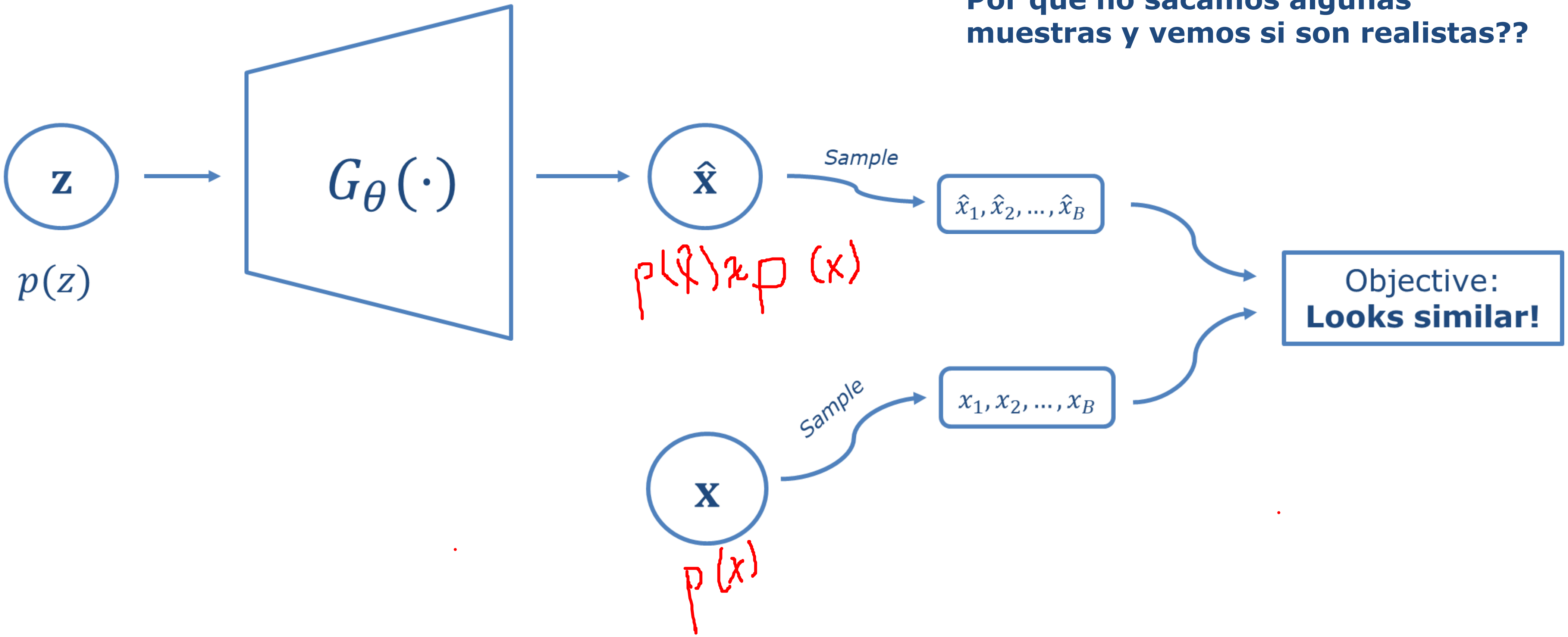


$$X \sim N(\mu, \sigma) = N(0,1)\cdot\sigma + \mu$$

Generative Adversarial Networks

Generative Adversarial Networks

Por qué no sacamos algunas muestras y vemos si son realistas??



Pero… ¿Cómo medimos similitud entre grupos de muestras?

$p(x)$ $q(x)$

$D(1,2)$?

Similitud entre muestras

Una solución: **entrenar un <u>clasificador</u>** $D_\phi(x)$ **para discriminar!**

- Si el clasificador no puede decir si una muestra es real o no, ambas distribuciones están cerca.
- Entrenamos con la *cross-entropy loss* estandar:

$$\max_{\phi} L_d(\phi) = \max_{\phi} \left(\mathbb{E}_{x_r \sim p_{real}} \log\left(D_\phi(x_r)\right) + \mathbb{E}_{x_f \sim p_{fake}} \log\left(1 - D_\phi(x_f)\right) \right)$$

Se puede probar que el coste de un clasificador *óptimo* $L_d(\phi^*)$ está relacionado con la *cercanía* entre ambas distribuciones (Jensen-Shannon divergence).

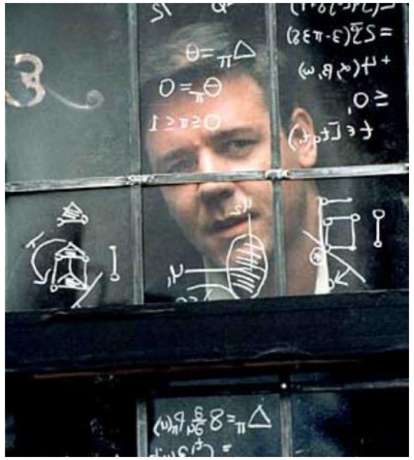
d

The GAN game

Queremos minimizar la “cercanía” entre las muestras generadas y las reales medida por el coste del discriminador:

$$\min_{\theta} \text{"closeness"}$$

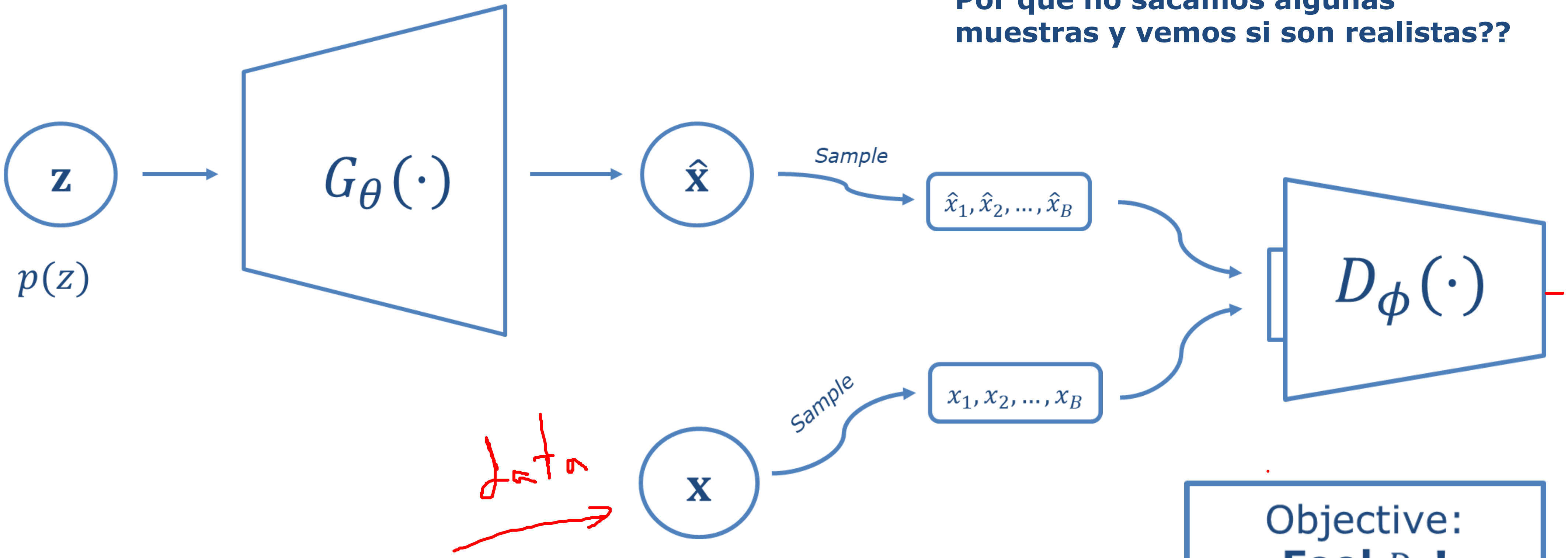
$$= \min_{\theta}\left(\max_{\phi}\left(\mathbb{E}_{x_r\sim p_{real}} \log\left(D_\phi(x_r)\right) + \mathbb{E}_{x_f\sim p_{fake}} \log\left(1 - D_\phi(x_f)\right)\right)\right)$$



Es formalmente un juego minimax de dos jugadores!

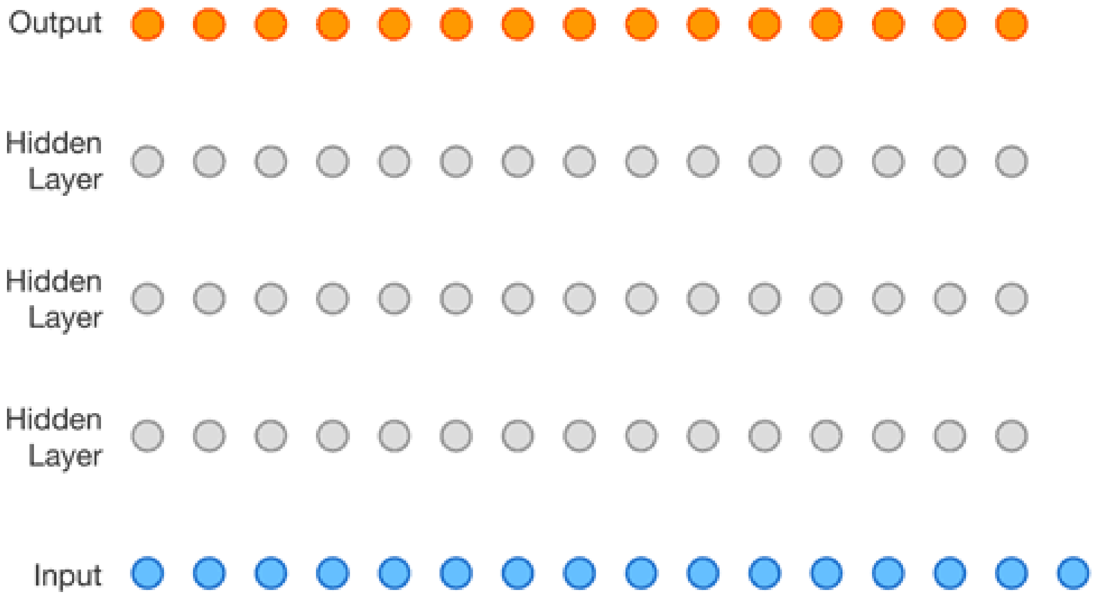
Generative Adversarial Networks

Por qué no sacamos algunas muestras y vemos si son realistas??



Objective:
Fool D_ϕ**!**

Bonus: autoregressive models



$$p_\theta(x) = \prod_{t=1}^{T} p_\theta(x_t | x_1, \ldots, x_{t-1})$$

Divide et impera!