

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!!

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

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

Probabilidad de que tu modelo generara x_i

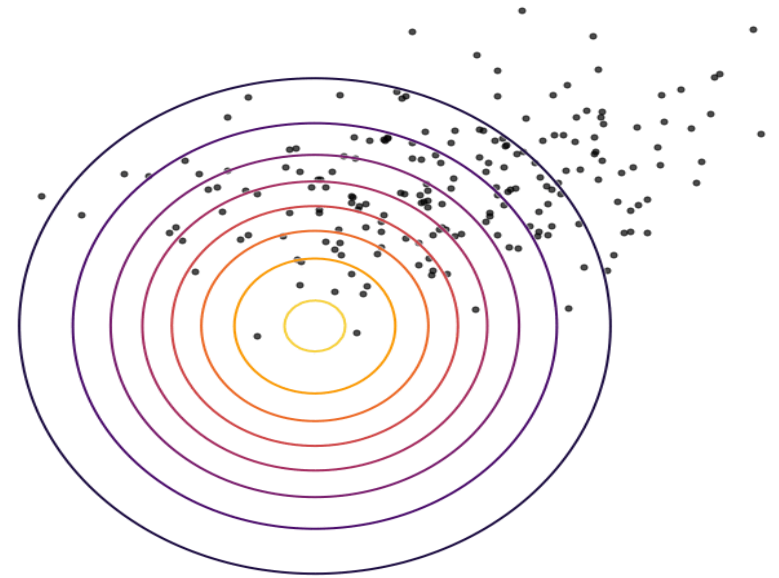


Image credit: Colin Raffel

Necesitamos $p_{\theta}(x)$ explícitamente!

Optimización

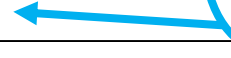
Optimizamos una función de coste (error)!

Stochastic Gradient Descent

$$\min_{\theta} L(\theta; \text{data})$$



$$\theta_{t+1} = \theta_t - \alpha \nabla_{\theta} L$$



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

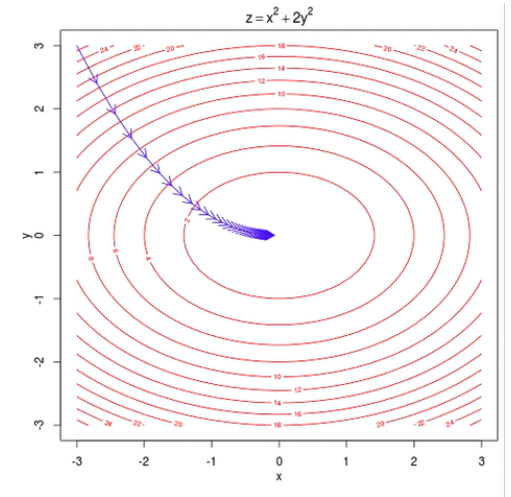
for x, y in dataloader:
    y_pred = model(x)
    loss = myloss(y, y_pred)
    loss.backward()
    optimizer.step()
```

 PyTorch

```
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

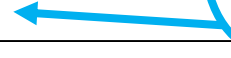
Optimizamos una función de coste (error)!

Stochastic Gradient Descent

$$\min_{\theta} L(\theta; \text{data})$$



$$\theta_{t+1} = \theta_t - \alpha \nabla_{\theta} L$$



```
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.backward()
    optimizer.step()
```

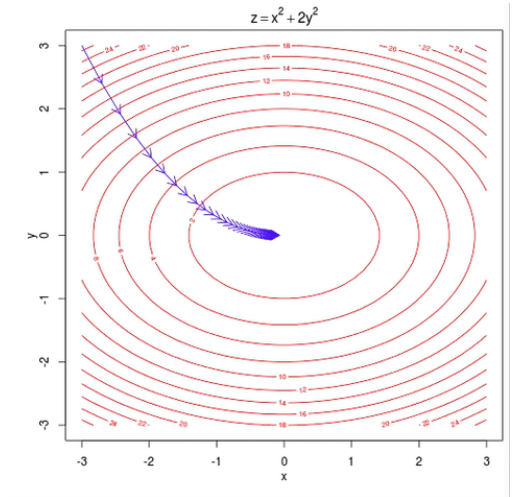
 PyTorch

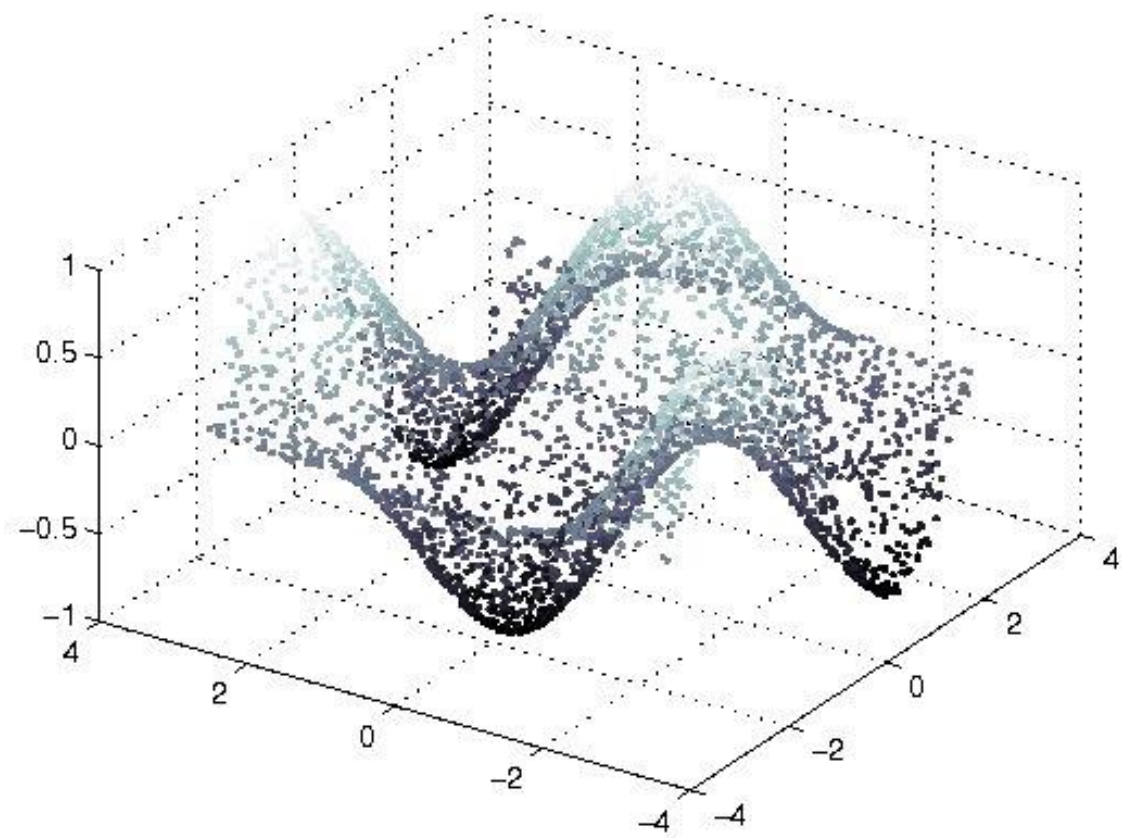
```
model = MyNetwork()
theta = model.trainable_variables
optimizer = tf.train.AdamOptimizer(lr=0.001)
```

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

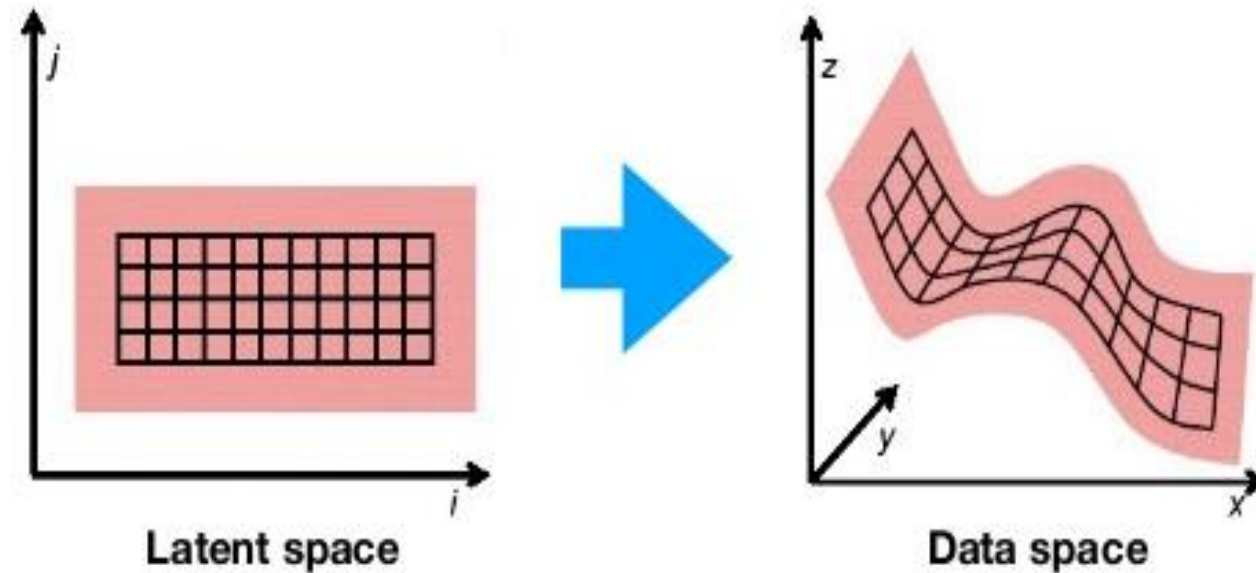
 TensorFlow

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

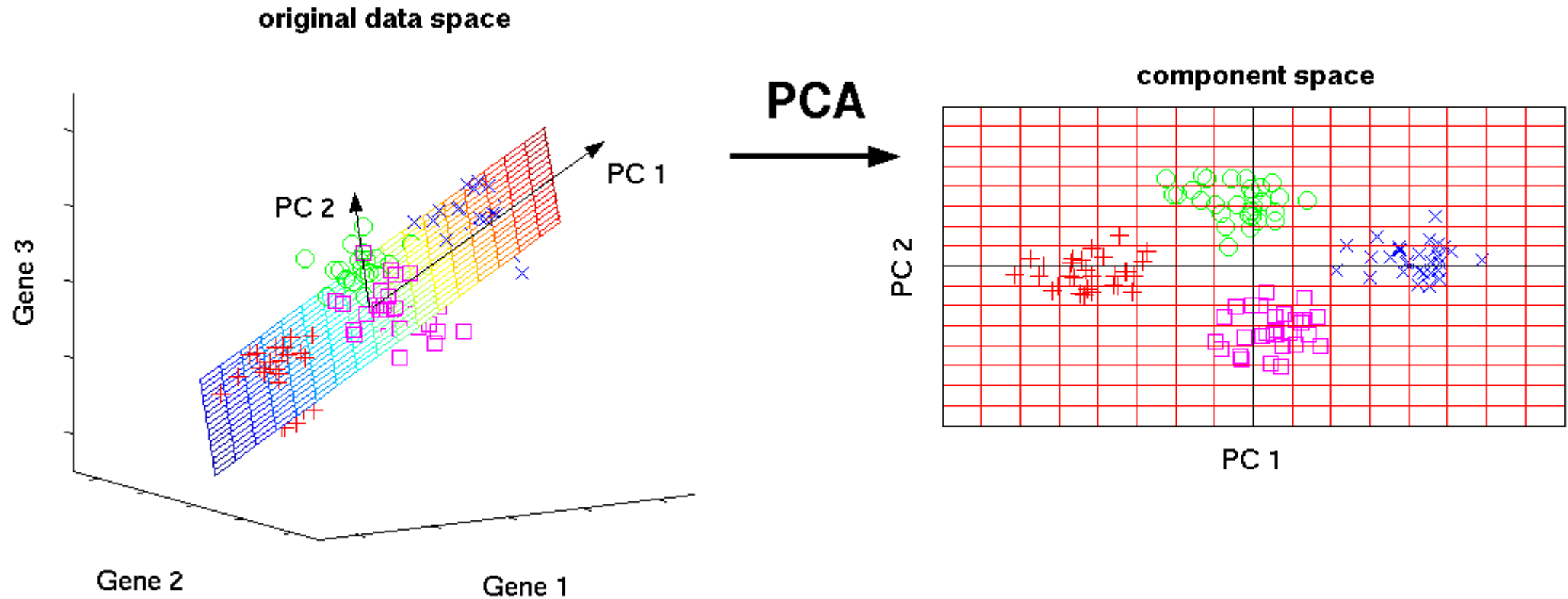




Dimensiones latentes de los datos

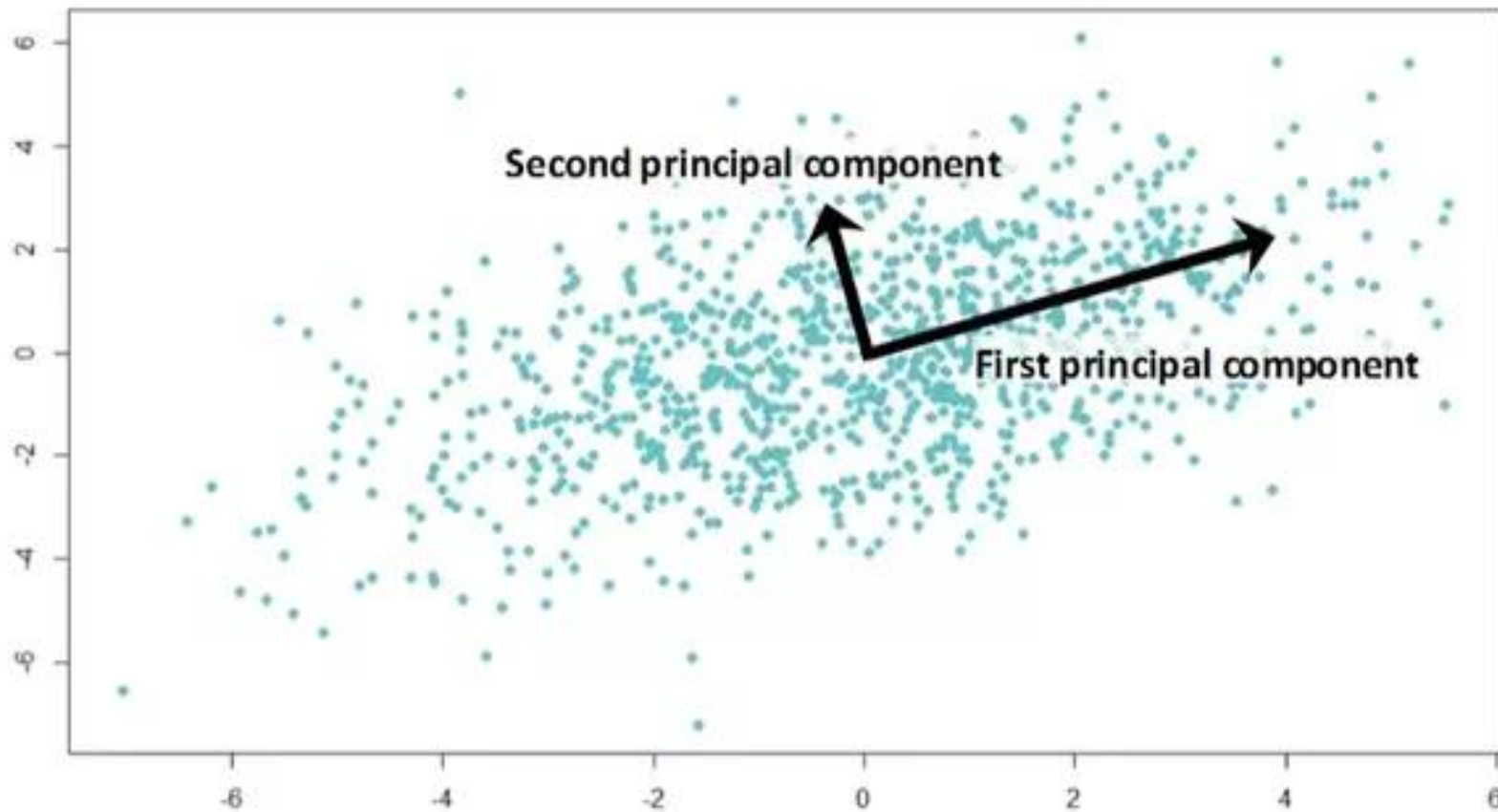


Principal Component Analysis

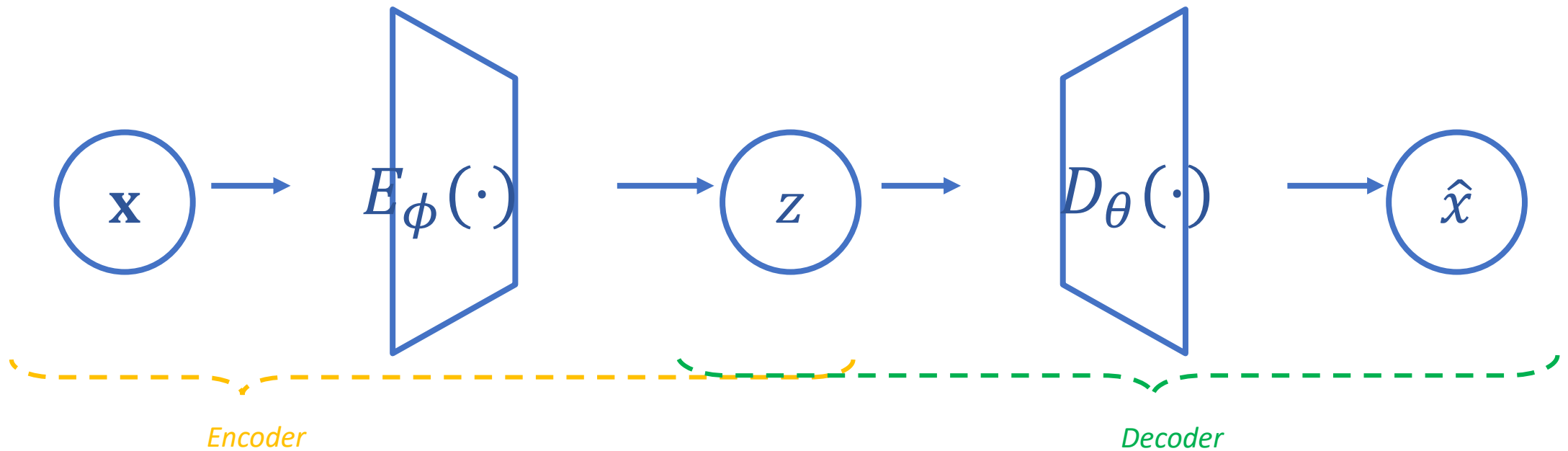


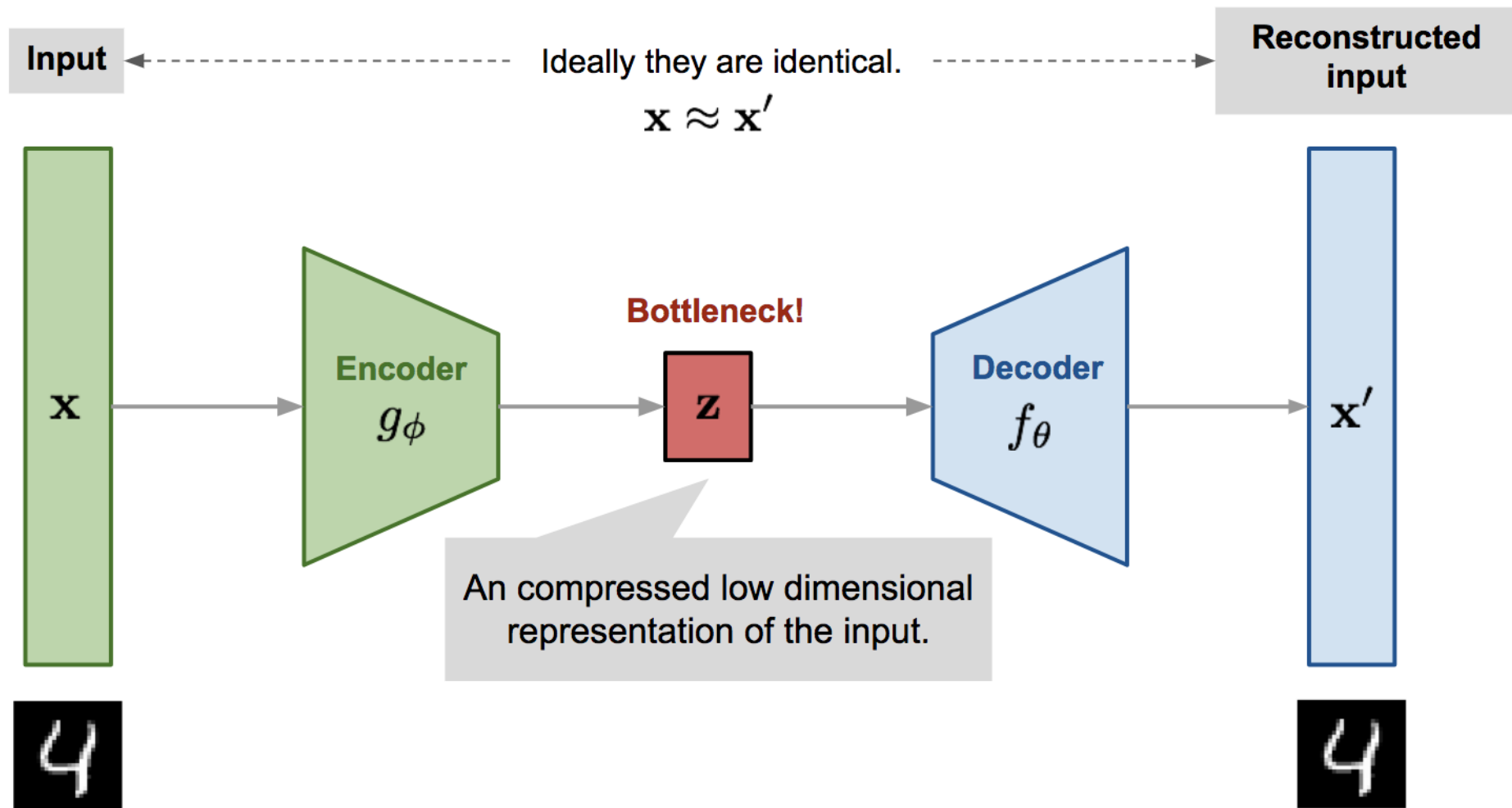
PCA asume que el manifold es lineal!

Principal Component Analysis



Autoencoders





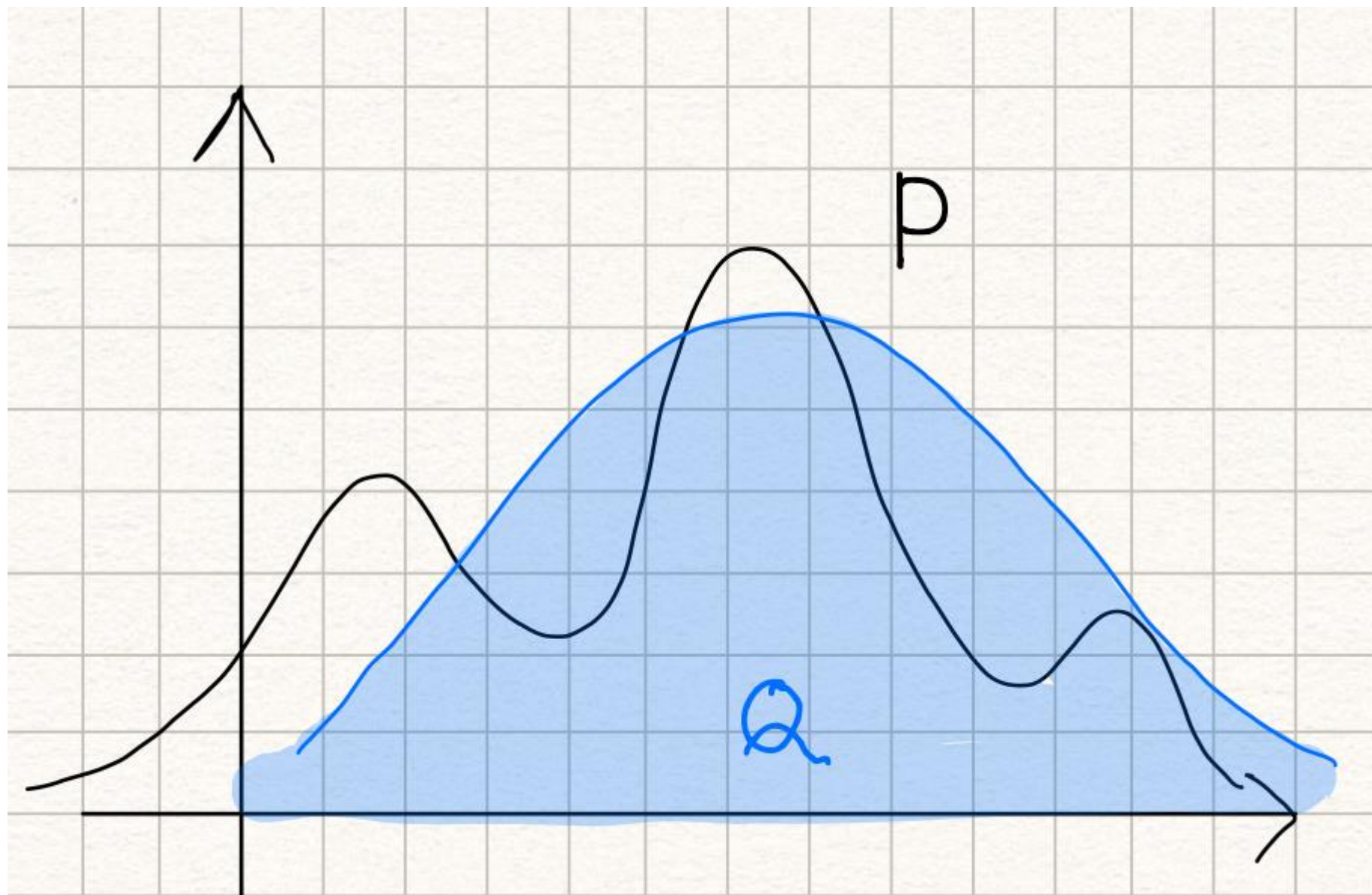
... pero no podemos usar la variable z para muestrear; ninguna relación con la distribución de probabilidad ☹

Let's go mathy!:

Divergencia de Kullback-Leibler
(KL)

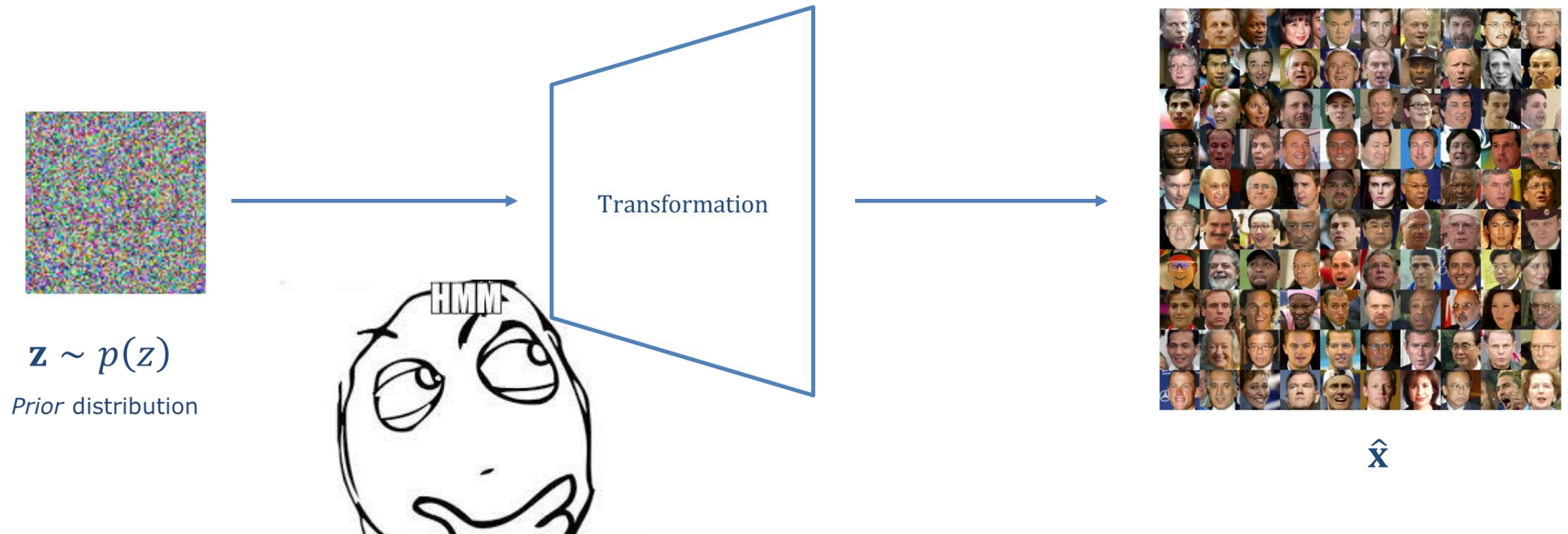
Cómo medimos la “distancia” entre dos distribuciones?

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

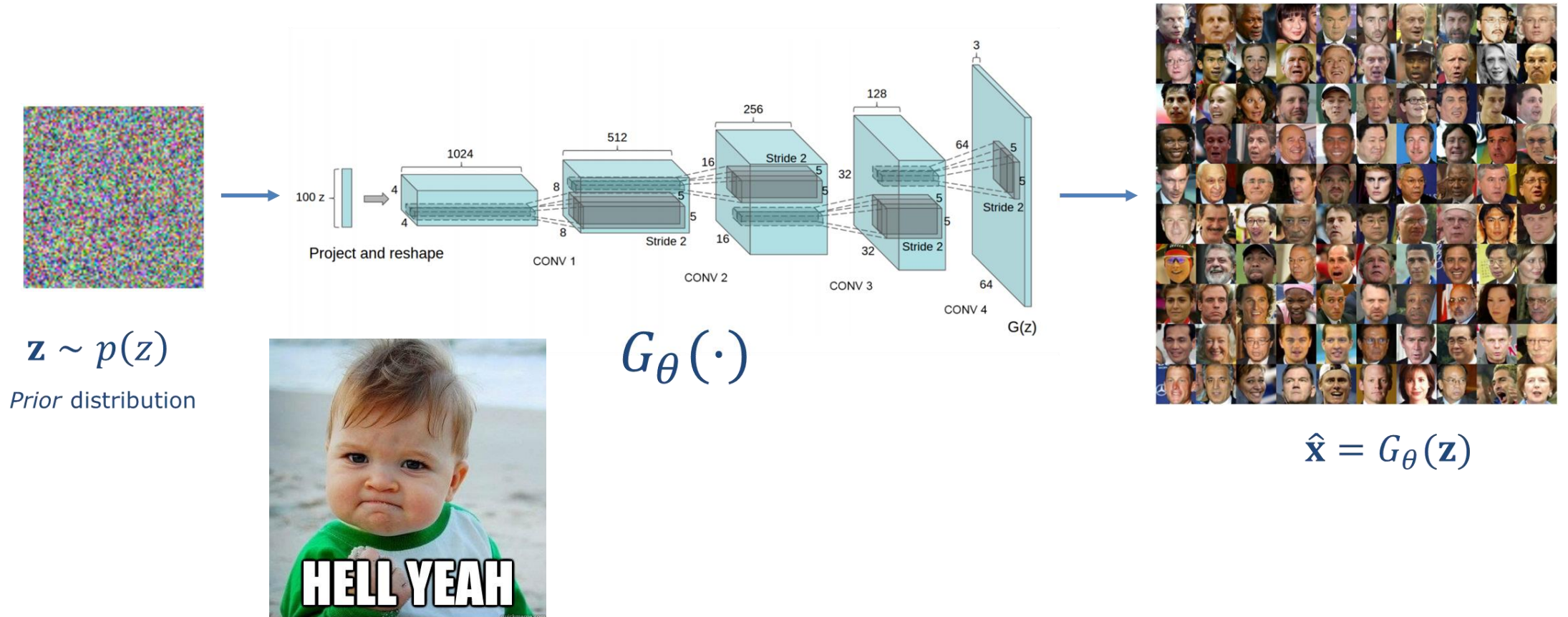


Variational Autoencoders

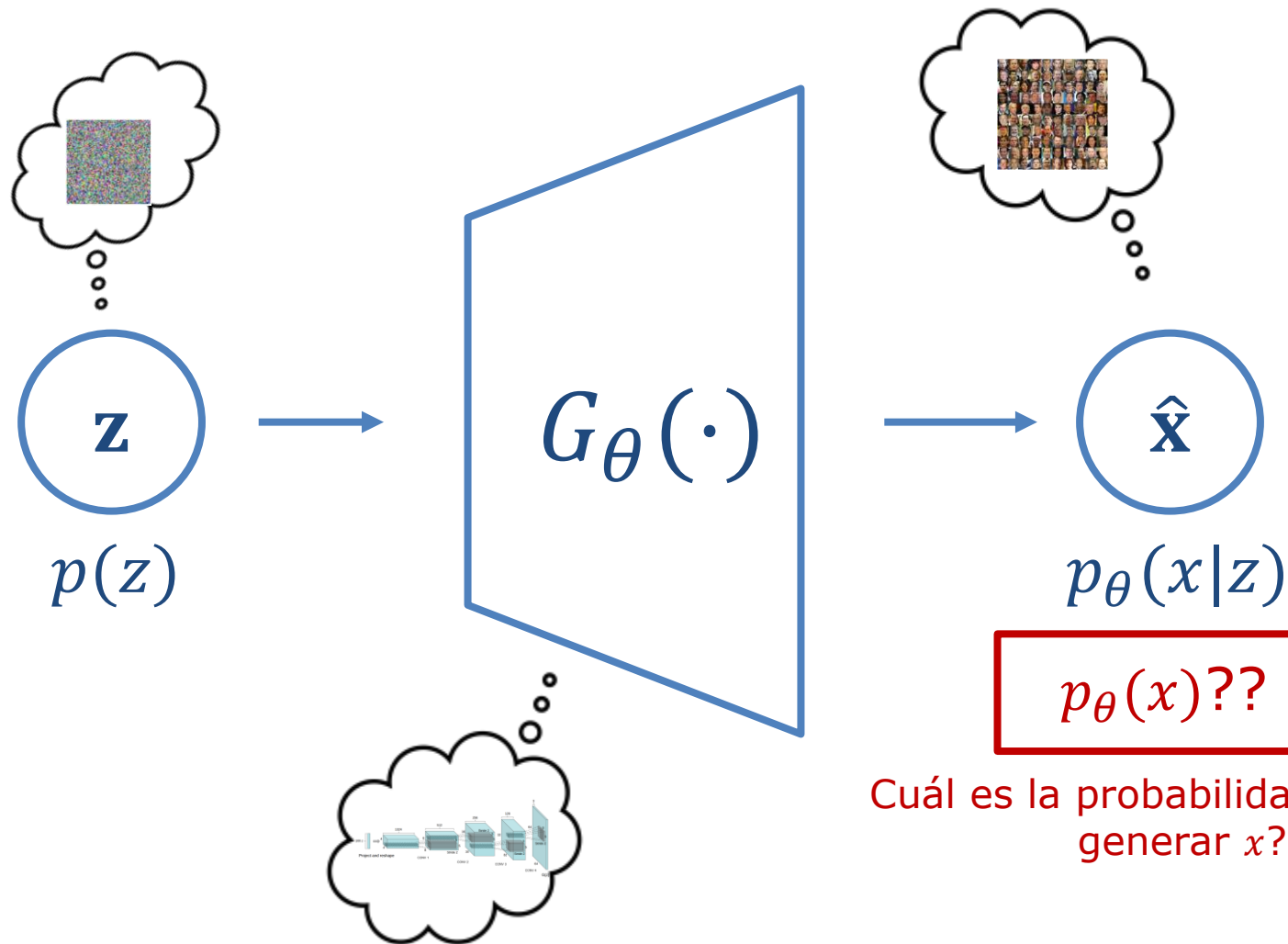
Lo que queremos!



Deep Latent Variable Models

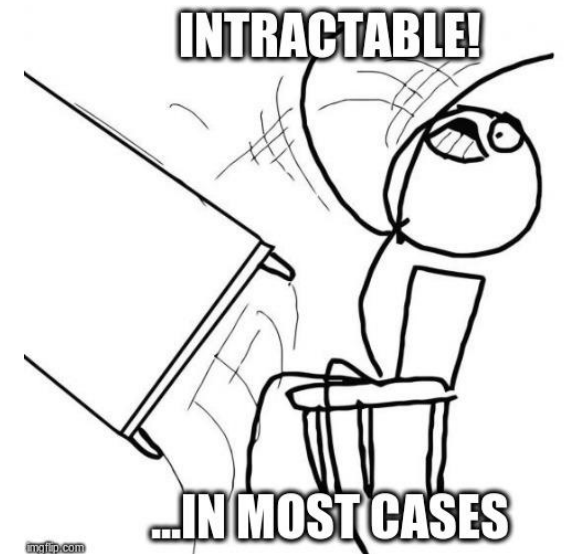


Genial! Optimicemos la likelihood!



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

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



Diferentes modelos – diferentes métodos

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

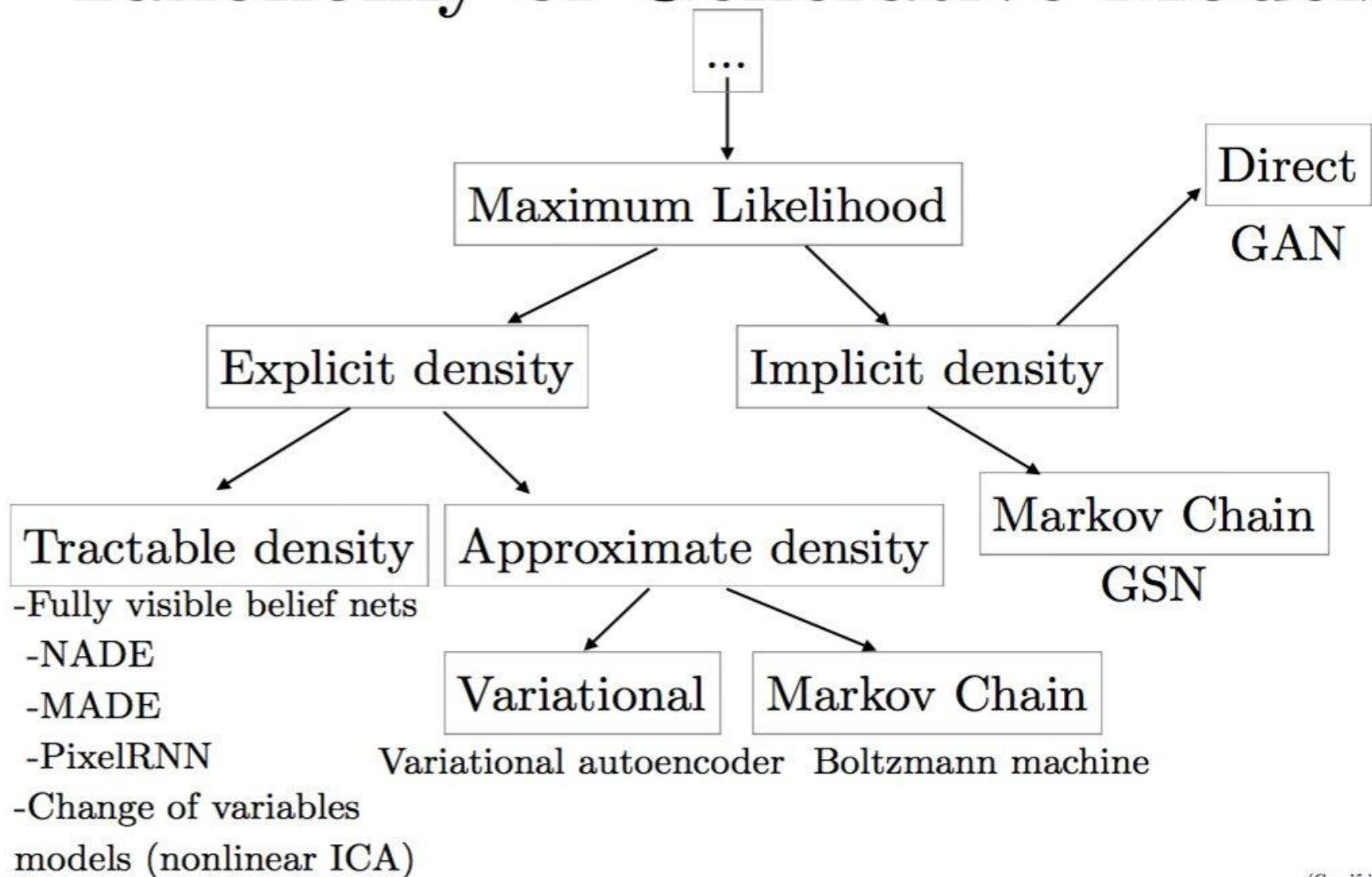
2. $p_{\theta}(\hat{x})$ es intratable: lo podemos aproximar

- Markov Chain Monte Carlo (MCMC) methods
- Variational methods (e.g. Variational Autoencoders)

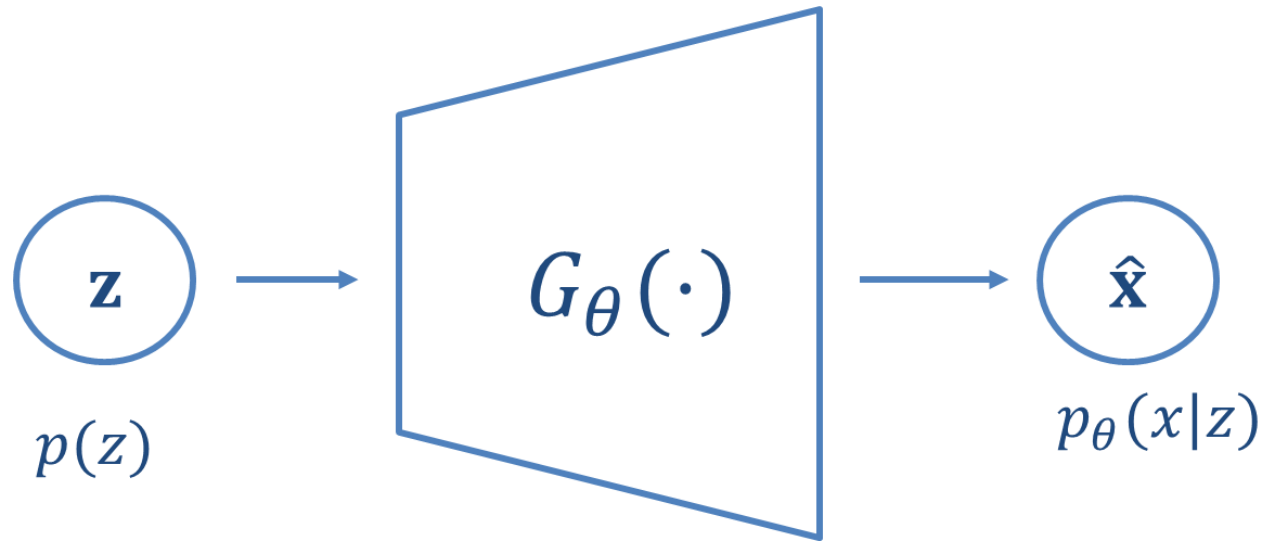
3. No necesitamos $p_{\theta}(\hat{x})$; está implícito!

- Adversarial methods (e.g. GANs)

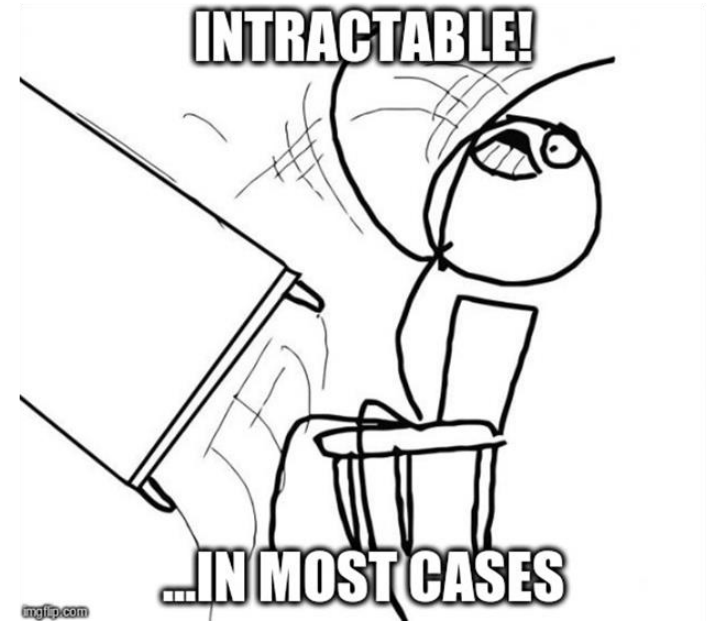
Taxonomy of Generative Models



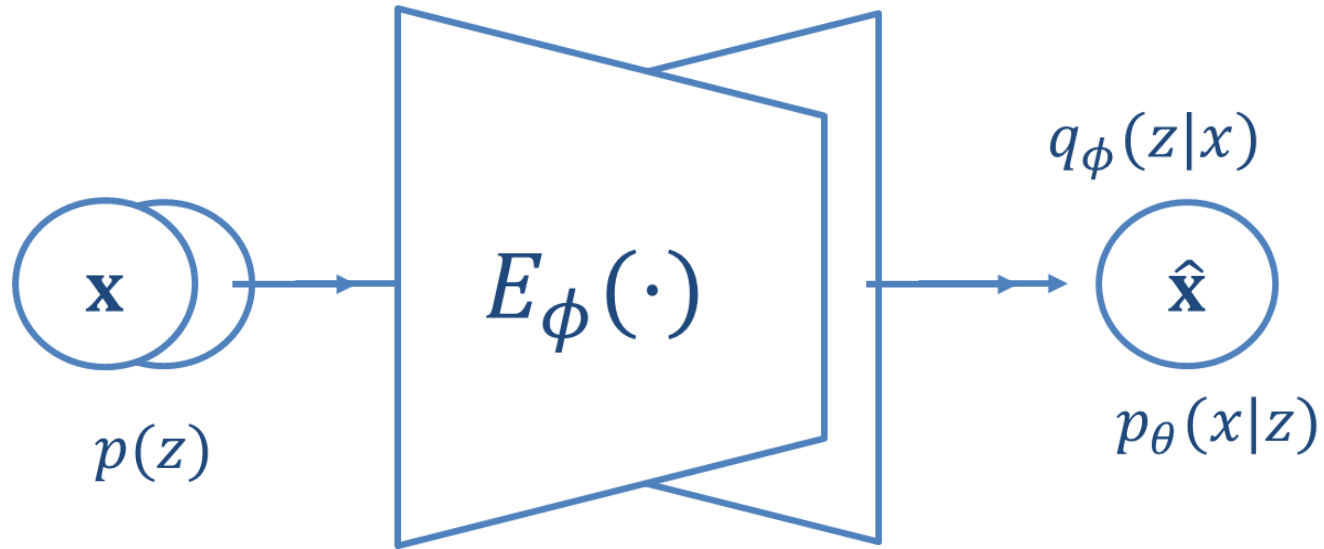
Variational Autoencoder



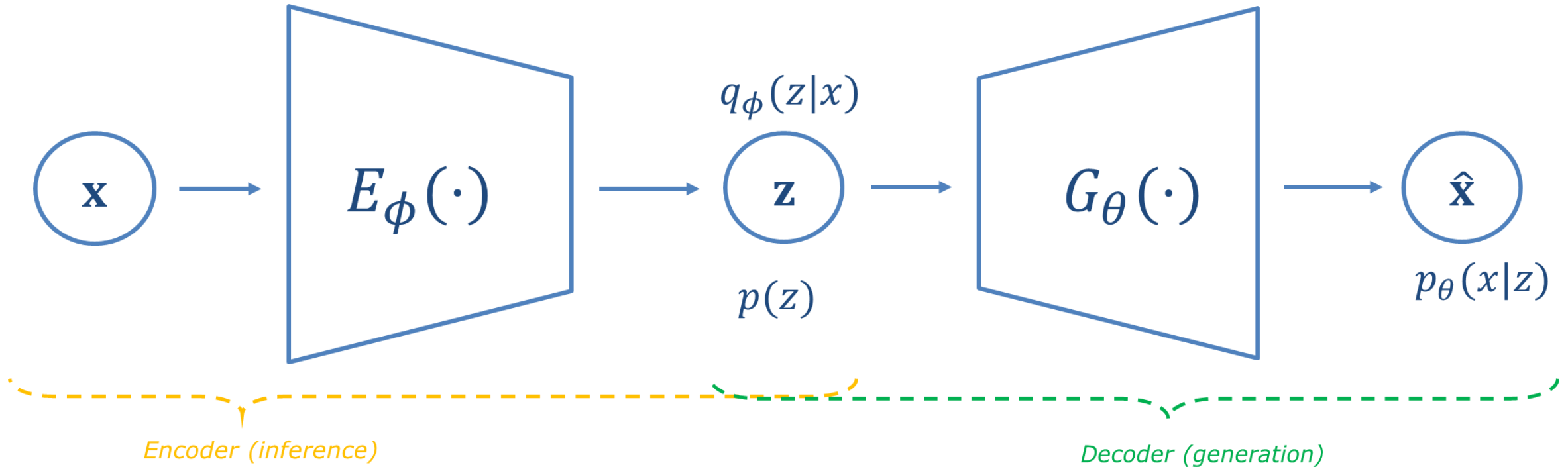
$$p_{\theta}(x) = \int p_{\theta}(x|z) p(z) dz$$



Variational Autoencoder

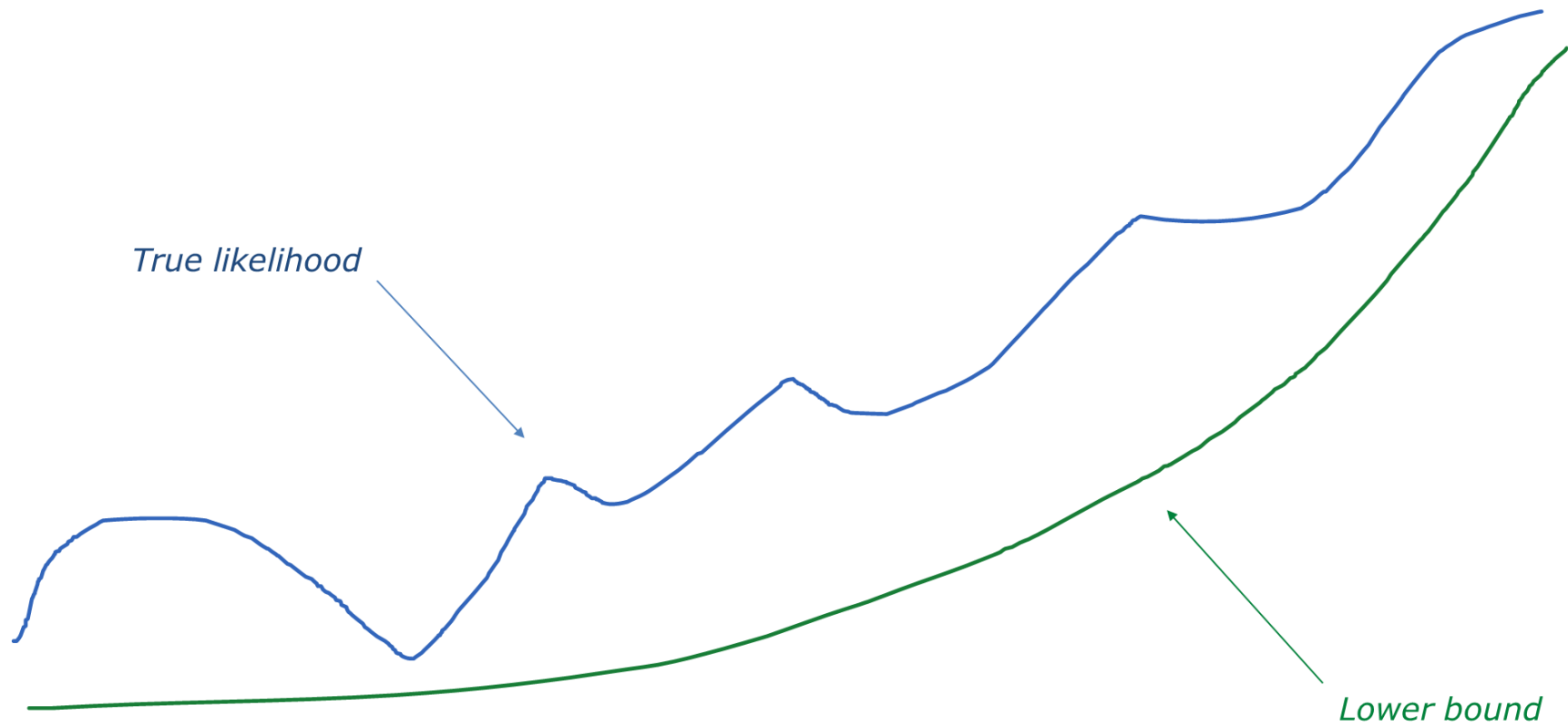


Variational Autoencoder



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

Maximize
this instead!



Iterations

¿Por qué? *Nerd warning*

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

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

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

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

Variational Autoencoders

Pros:

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

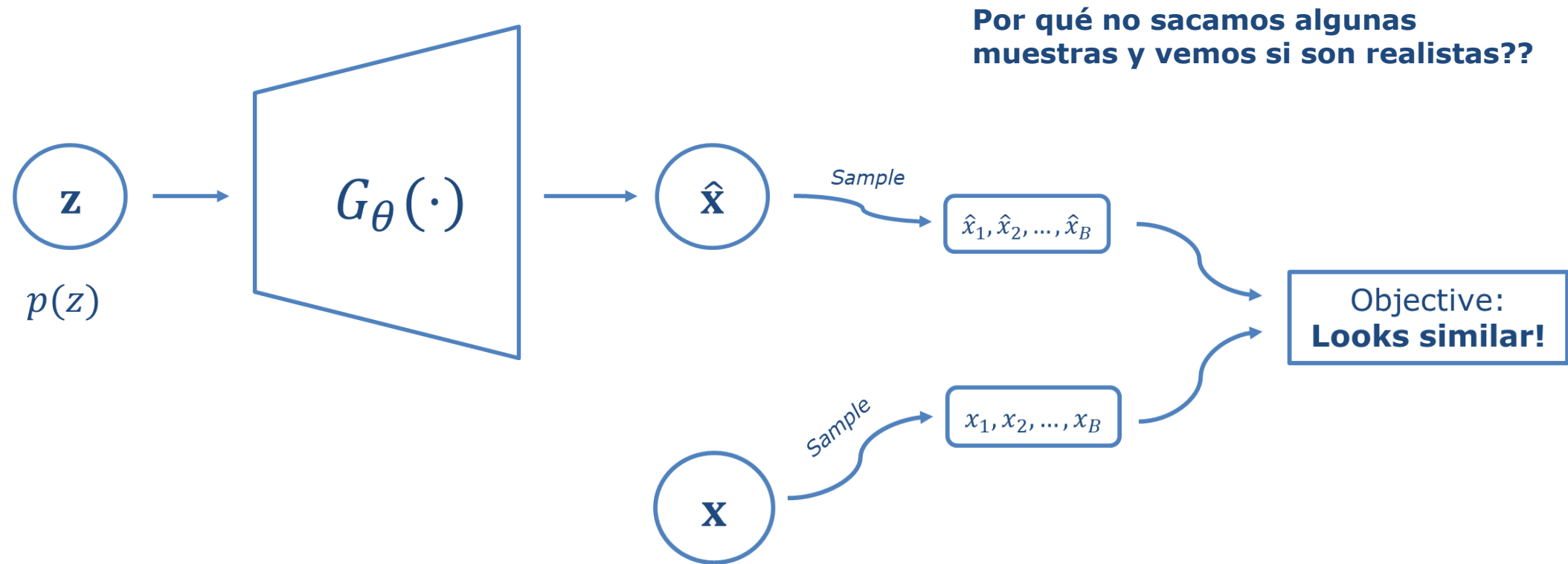
Cons:

- No genera muy buenas muestras



Generative Adversarial Networks

Generative Adversarial Networks



Pero... ¿Cómo medimos similitud entre grupos de muestras?

Similitud entre muestras

Una solución: **entrenar un clasificador $D_\phi(x)$ 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).

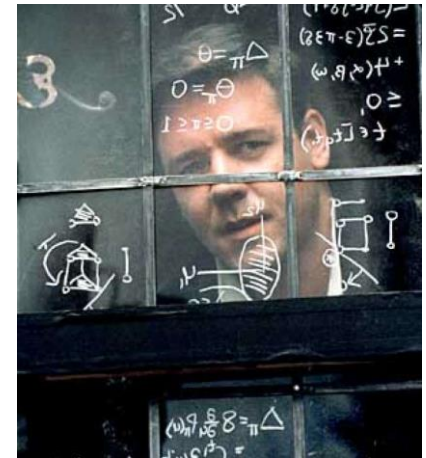
The GAN game

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

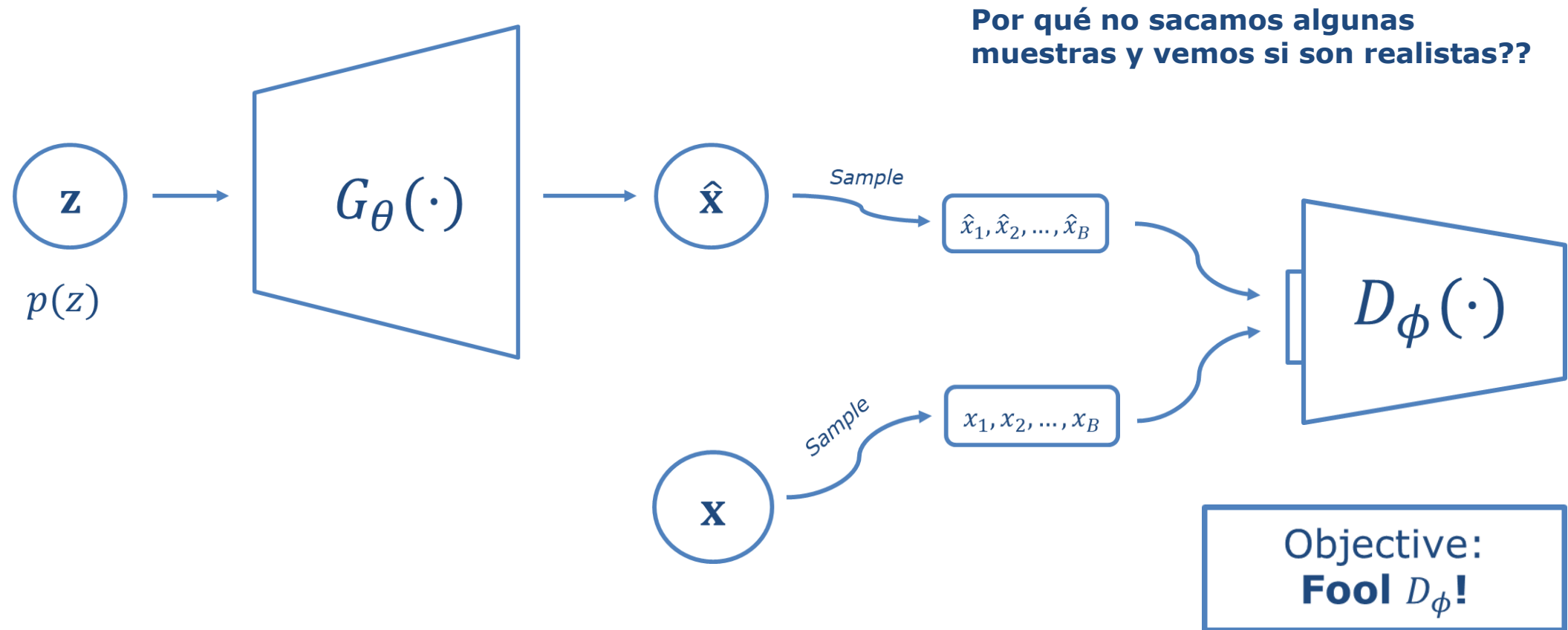
\min_{θ} "closeness"

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

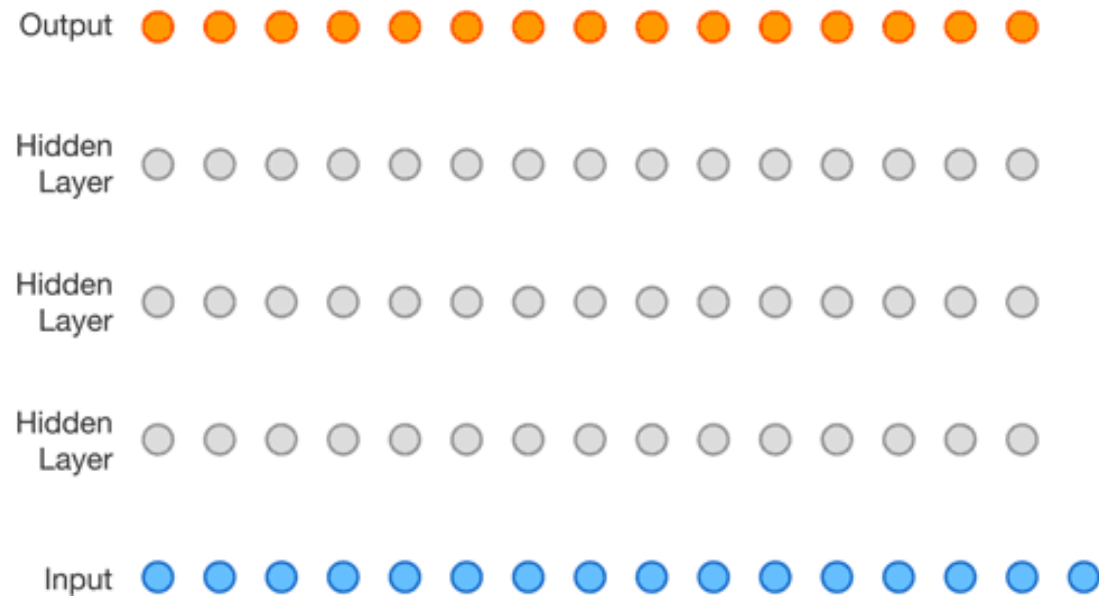
Es formalmente un juego minimax de dos jugadores!



Generative Adversarial Networks



Bonus: autoregressive models



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



Divide et impera!