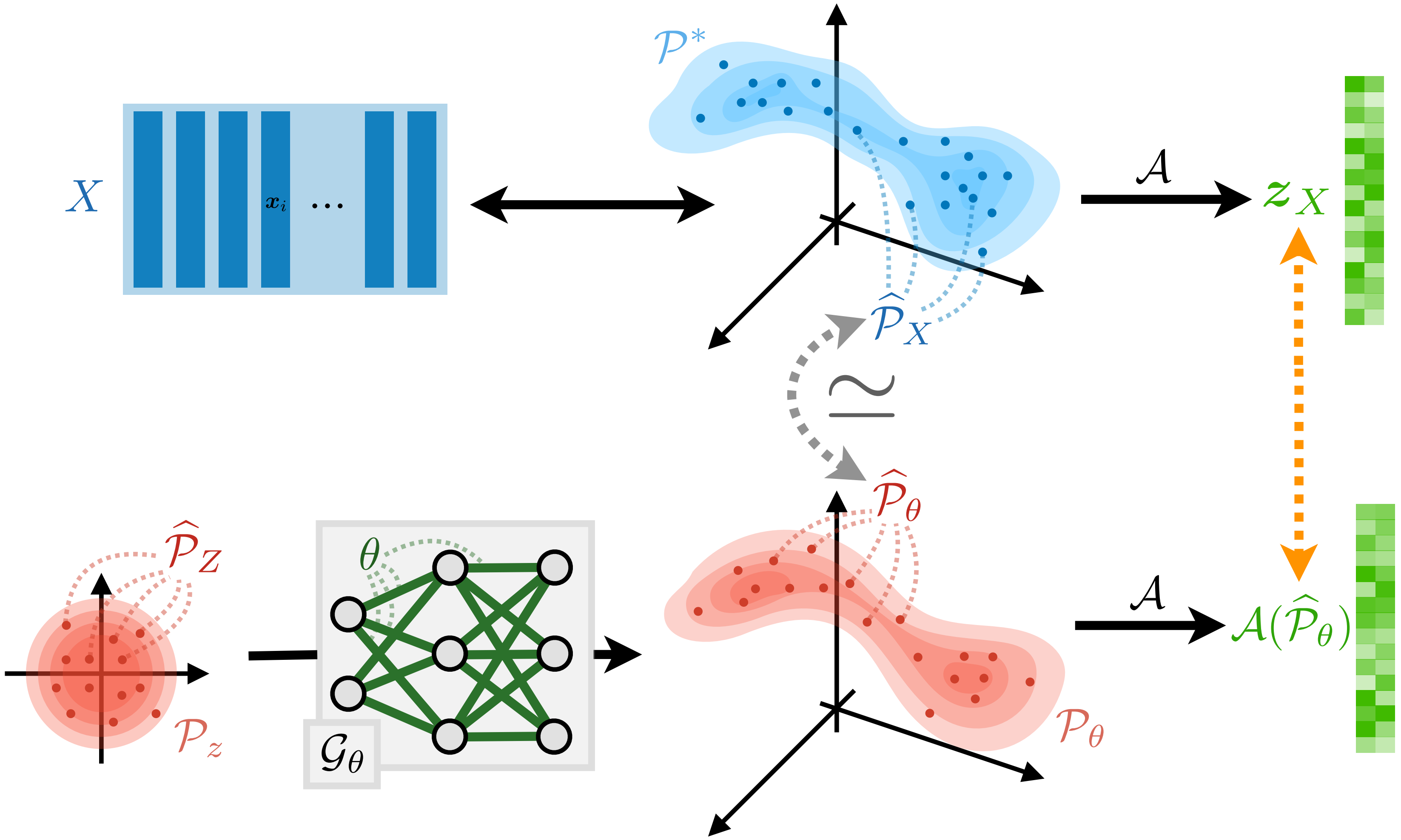
Generative models as data-driven priors: how to learn them efficiently?

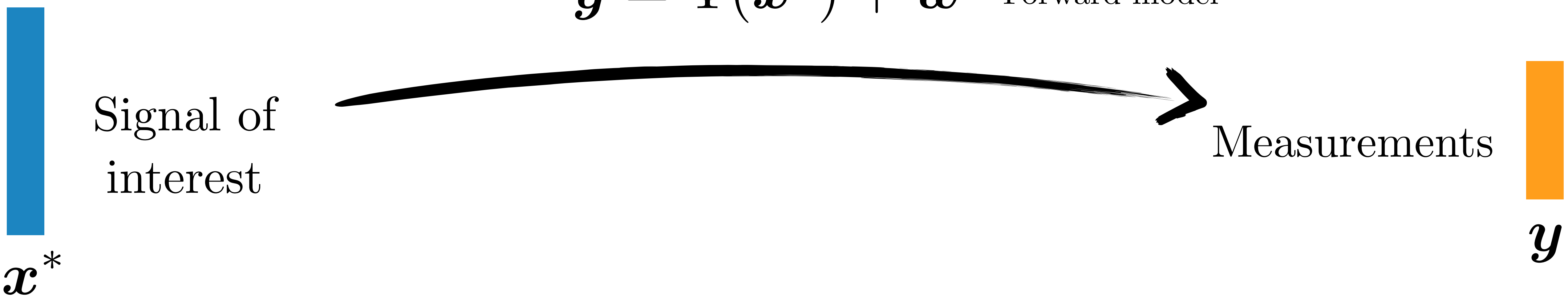
Vincent Schellekens & Laurent Jacques

UCLouvain



Motivation: inverse problems

$$\boldsymbol{y} = \Phi(\boldsymbol{x}^*) + \boldsymbol{w}$$ Forward model



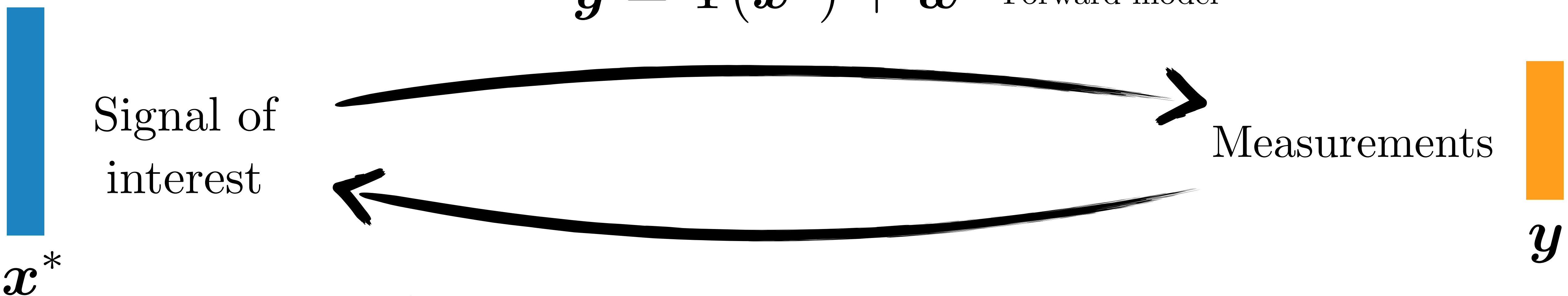
Signal of interest

$\boldsymbol{x}^*$

Measurements

$\boldsymbol{y}$

Motivation: inverse problems



$$\boldsymbol{y} = \Phi(\boldsymbol{x}^*) + \boldsymbol{w}$$ Forward model

Signal of interest

$\boldsymbol{x}^*$

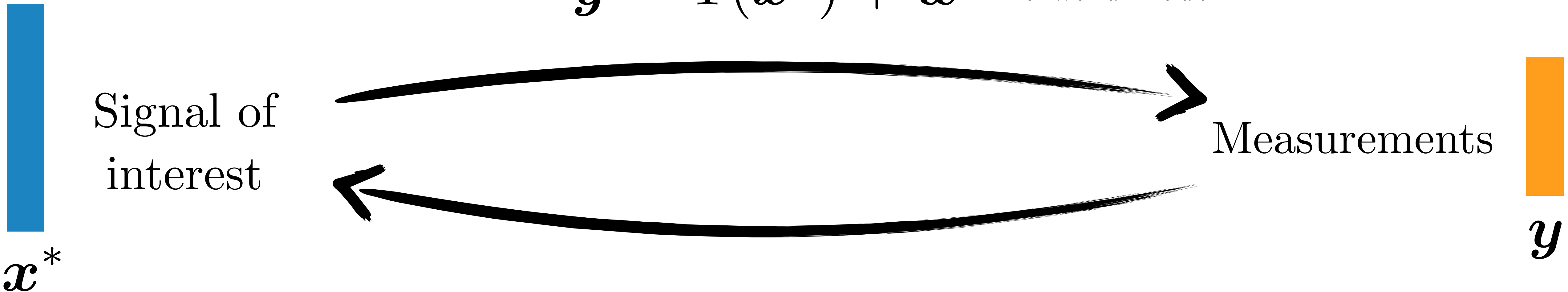
Measurements

$\boldsymbol{y}$

$$\widehat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} \; l(\boldsymbol{x}; \boldsymbol{y}) \; + \; \lambda r(\boldsymbol{x})$$ Variational formulation

Motivation: inverse problems

$\boldsymbol{y} = \Phi(\boldsymbol{x}^*) + \boldsymbol{w}$ Forward model

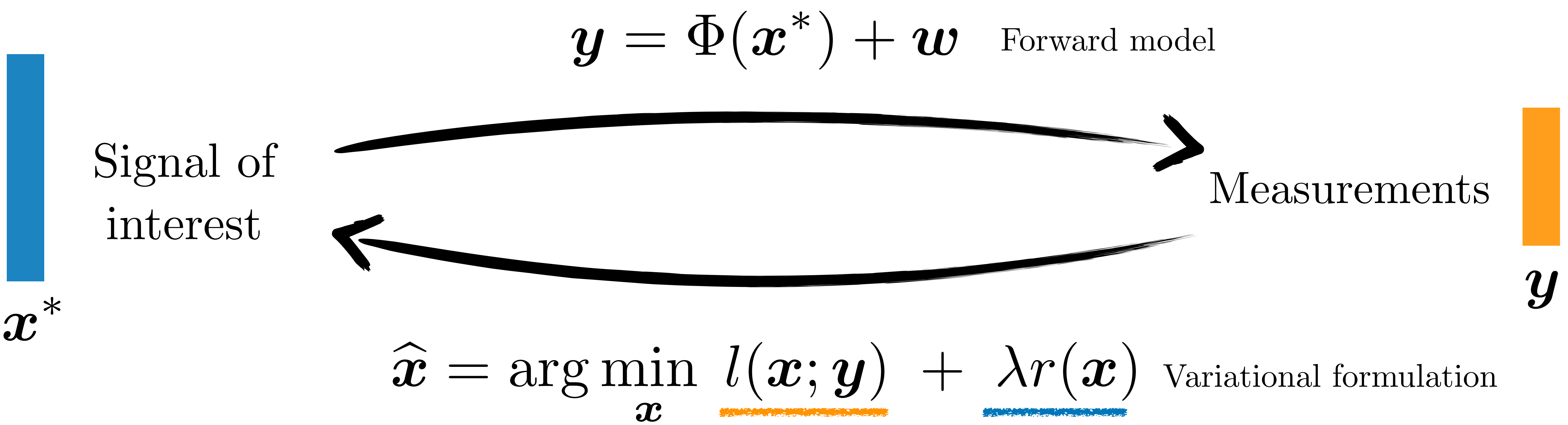


$\widehat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} \underline{l(\boldsymbol{x}; \boldsymbol{y})} + \lambda r(\boldsymbol{x})$ Variational formulation

Data-fidelity loss

- Euclidean $l(\boldsymbol{x}; \boldsymbol{y}) = \|\boldsymbol{y} - \Phi(\boldsymbol{x})\|_2^2$
- ...

Motivation: inverse problems



$$\boldsymbol{y} = \Phi(\boldsymbol{x}^*) + \boldsymbol{w}$$ Forward model

Signal of interest

$\boldsymbol{x}^*$

Measurements

$\boldsymbol{y}$

$$\widehat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} \ \underline{l(\boldsymbol{x}; \boldsymbol{y})} \ + \ \underline{\lambda r(\boldsymbol{x})}$$ Variational formulation

Data-fidelity loss

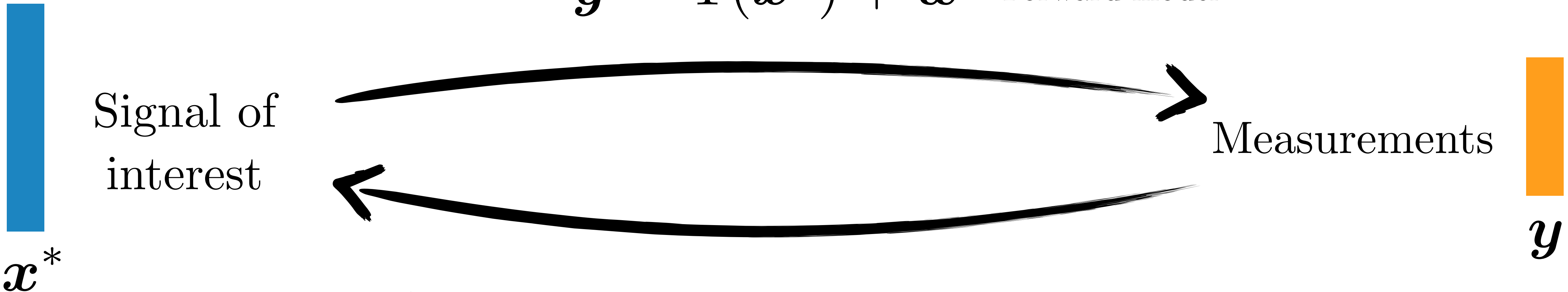
- Euclidean $l(\boldsymbol{x}; \boldsymbol{y}) = \|\boldsymbol{y} - \Phi(\boldsymbol{x})\|_2^2$
- …

Regularization (= prior)

- Tikhonov $r(\boldsymbol{x}) = \|\boldsymbol{x}\|_2^2$
- TV-norm $r(\boldsymbol{x}) = \|\boldsymbol{x}\|_{TV}$
- Sparsity (in wavelets) $r(\boldsymbol{x}) = \|\Psi \boldsymbol{x}\|_1$
- Sparsity (in dictionary) $r(\boldsymbol{x}) = \|D\boldsymbol{x}\|_1$
- Generative priors

Motivation: inverse problems

$$\boldsymbol{y} = \Phi(\boldsymbol{x}^*) + \boldsymbol{w}$$ Forward model



Signal of interest

$\boldsymbol{x}^*$

Measurements

$\boldsymbol{y}$

$$\widehat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} \underline{l(\boldsymbol{x}; \boldsymbol{y})} + \underline{\lambda r(\boldsymbol{x})}$$ Variational formulation

Data-fidelity loss

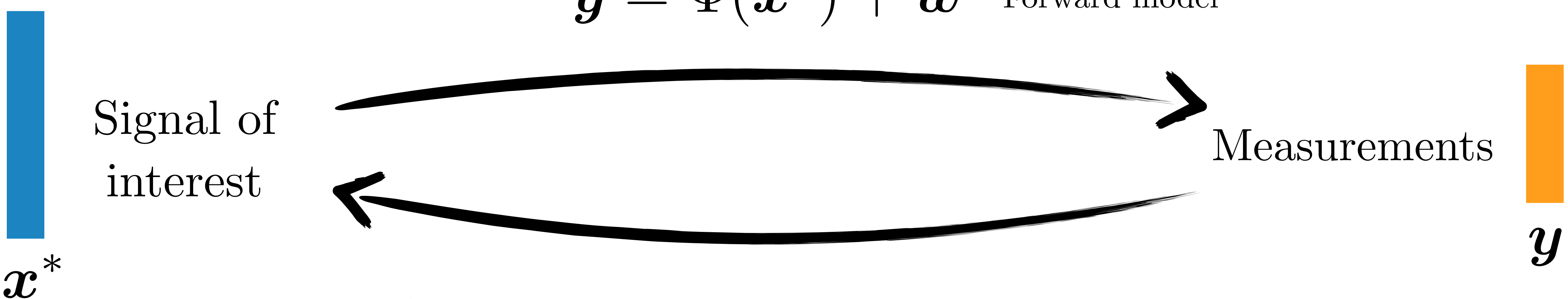
- Euclidean $l(\boldsymbol{x}; \boldsymbol{y}) = \|\boldsymbol{y} - \Phi(\boldsymbol{x})\|_2^2$
- ...

Regularization (= prior)

- Tikhonov $r(\boldsymbol{x}) = \|\boldsymbol{x}\|_2^2$
- TV-norm $r(\boldsymbol{x}) = \|\boldsymbol{x}\|_{TV}$
- Sparsity (in wavelets) $r(\boldsymbol{x}) = \|\Psi \boldsymbol{x}\|_1$
- Sparsity (in dictionary) $r(\boldsymbol{x}) = \|D\boldsymbol{x}\|_1$
- Generative priors

From model-based
to data-driven priors!

Motivation: inverse problems



$$\boldsymbol{y} = \Phi(\boldsymbol{x}^*) + \boldsymbol{w}$$ Forward model

$$\widehat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} \underline{l(\boldsymbol{x};\boldsymbol{y})} + \underline{\lambda r(\boldsymbol{x})}$$ Variational formulation

Data-fidelity loss

- Euclidean $l(\boldsymbol{x};\boldsymbol{y}) = \|\boldsymbol{y} - \Phi(\boldsymbol{x})\|_2^2$
- ...

Regularization (= prior)

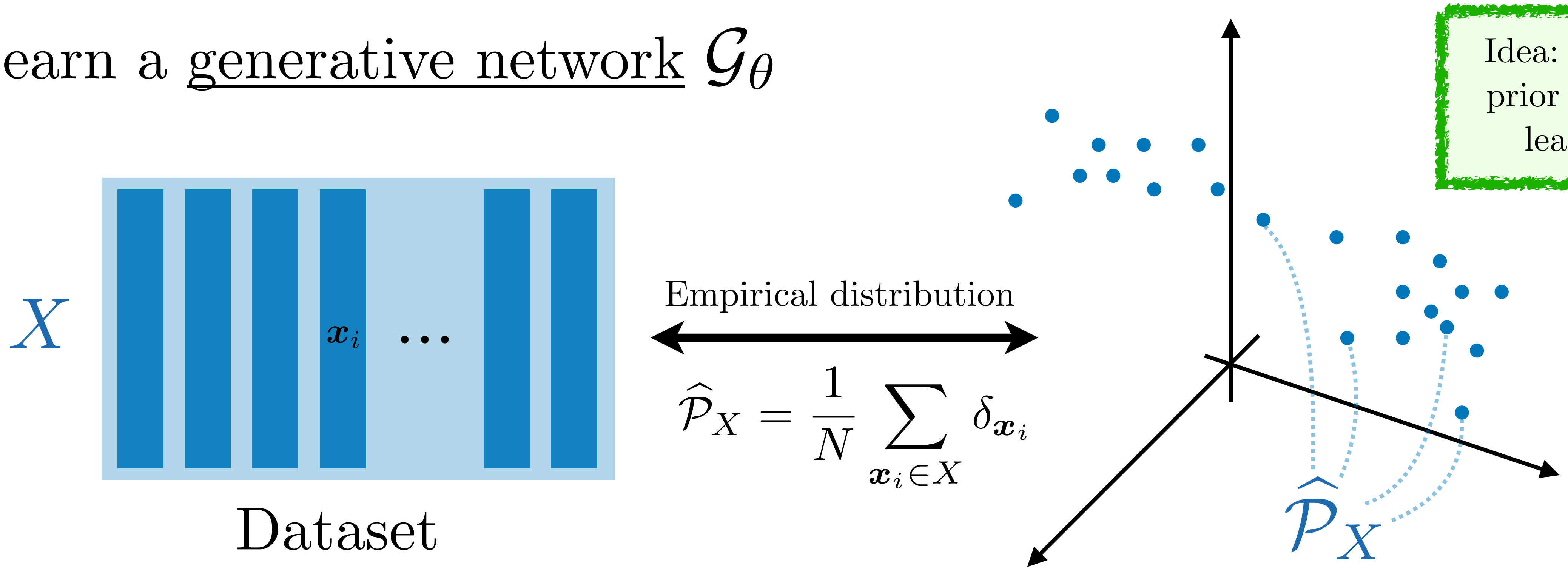
- Tikhonov $r(\boldsymbol{x}) = \|\boldsymbol{x}\|_2^2$
- TV-norm $r(\boldsymbol{x}) = \|\boldsymbol{x}\|_{TV}$
- Sparsity (in wavelets) $r(\boldsymbol{x}) = \|\Psi\boldsymbol{x}\|_1$
- Sparsity (in dictionary) $r(\boldsymbol{x}) = \|D\boldsymbol{x}\|_1$
- Generative priors

From model-based
to data-driven priors!

Idea: approximate true prior distribution from learning examples

Generative priors in inverse problems (1)

1) Learn a generative network $\mathcal{G}_\theta$



$$\widehat{\mathcal{P}}_X = \frac{1}{N} \sum_{\boldsymbol{x}_i \in X} \delta_{\boldsymbol{x}_i}$$

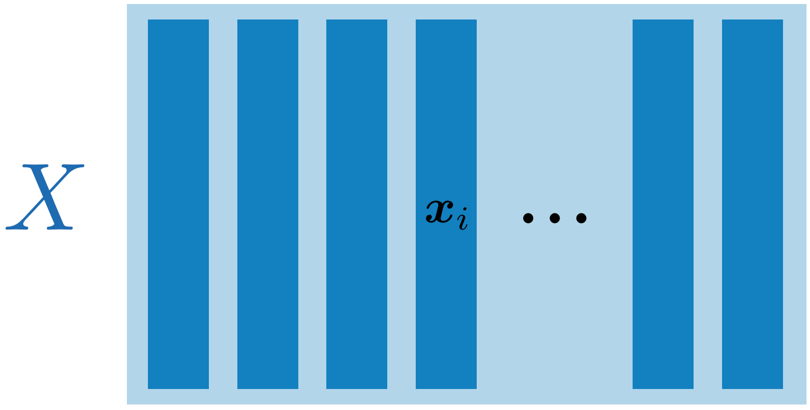
Dataset

We have **samples** (signals)

Idea: approximate true prior distribution from learning examples

Generative priors in inverse problems (1)

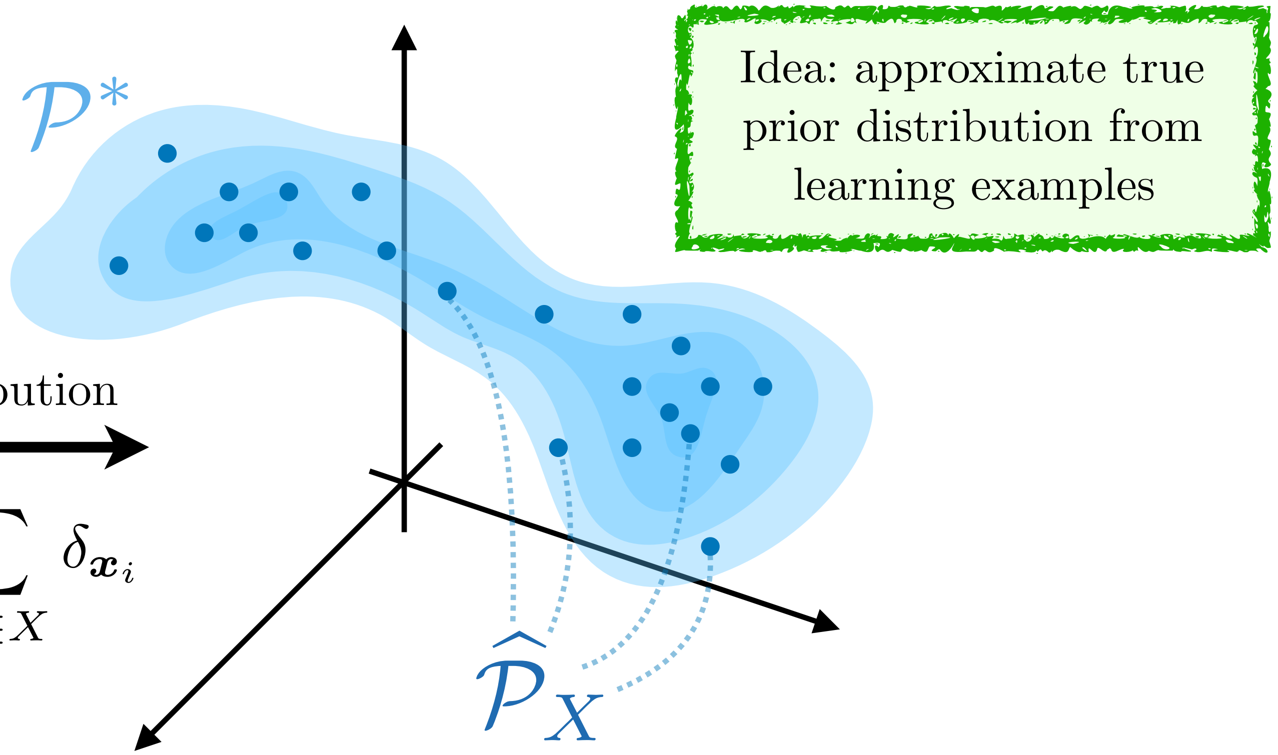
1) Learn a generative network $\mathcal{G}_\theta$



Dataset

Empirical distribution

$$\widehat{\mathcal{P}}_X = \frac{1}{N} \sum_{\boldsymbol{x}_i \in X} \delta_{\boldsymbol{x}_i}$$



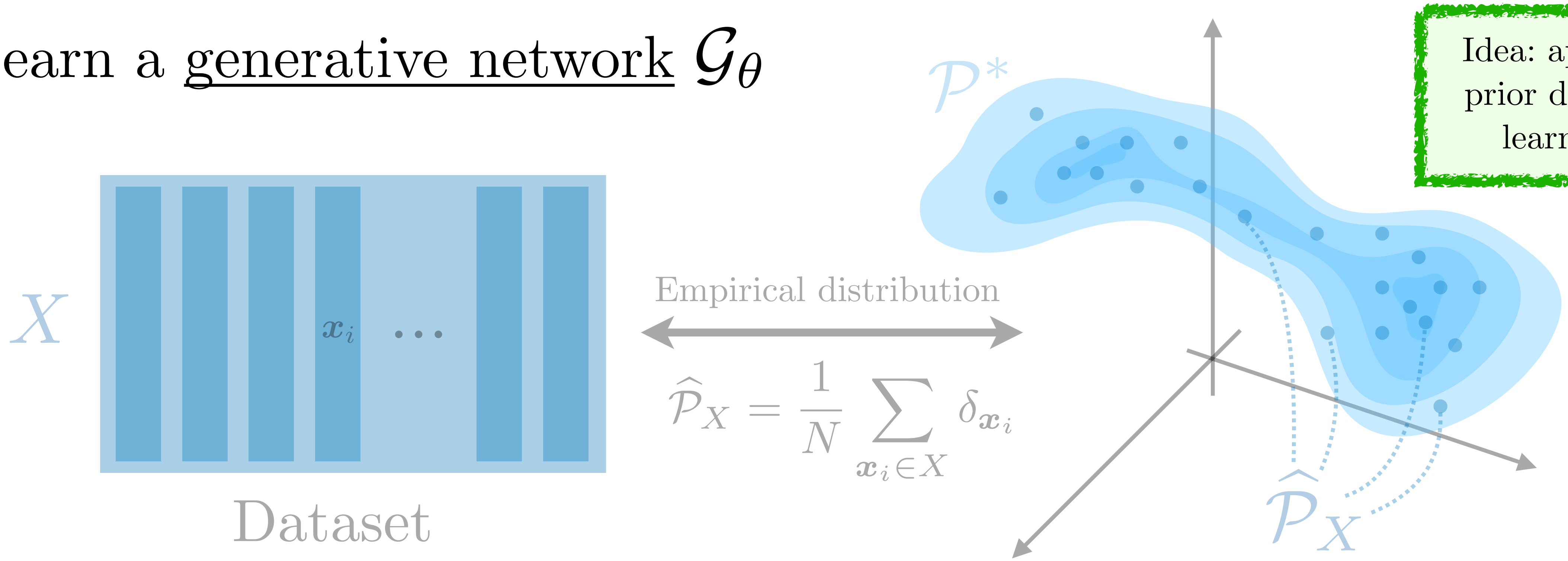
Idea: approximate true prior distribution from learning examples

We have **samples** (signals) from a high-dimensional **distribution**...

Generative priors in inverse problems (1)

1) Learn a <u>generative network</u> $\mathcal{G}_\theta$

Idea: approximate true prior distribution from learning examples

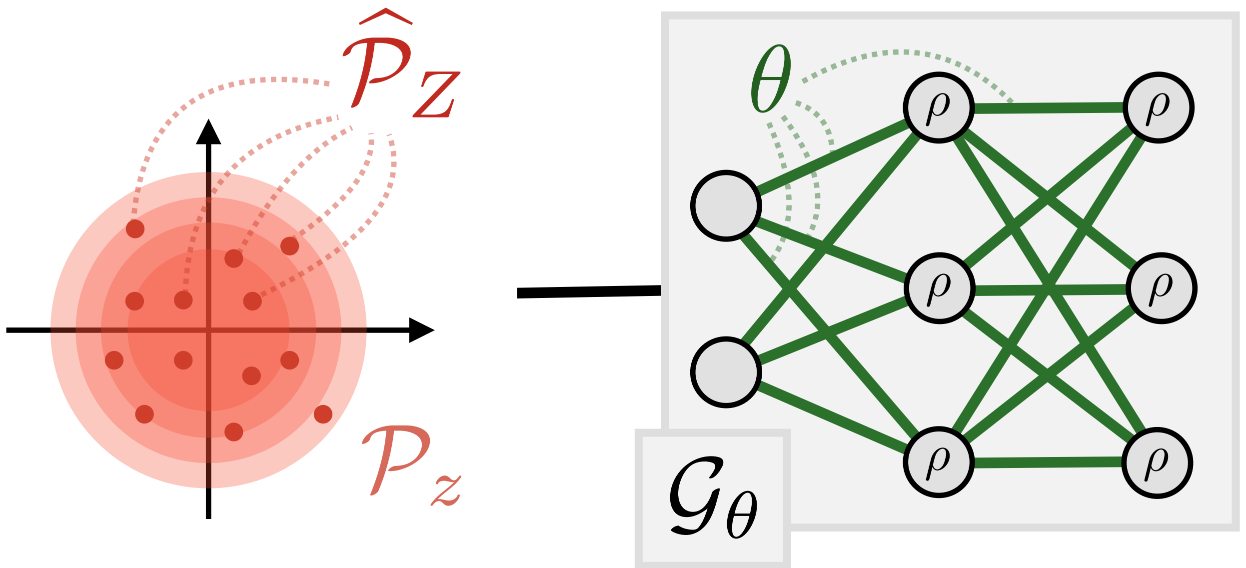


$$\widehat{\mathcal{P}}_X = \frac{1}{N} \sum_{\boldsymbol{x}_i \in X} \delta_{\boldsymbol{x}_i}$$

We have **samples** (signals) from a high-dimensional **distribution**...

... and a way to generate "artificial" samples...

$$\mathcal{G}_\theta(\boldsymbol{z}) = \rho(\Theta_L \cdot \rho(\Theta_{L-1} \cdots \rho(\Theta_2 \cdot \rho(\Theta_1 \cdot \boldsymbol{z}))))$$



Latent space

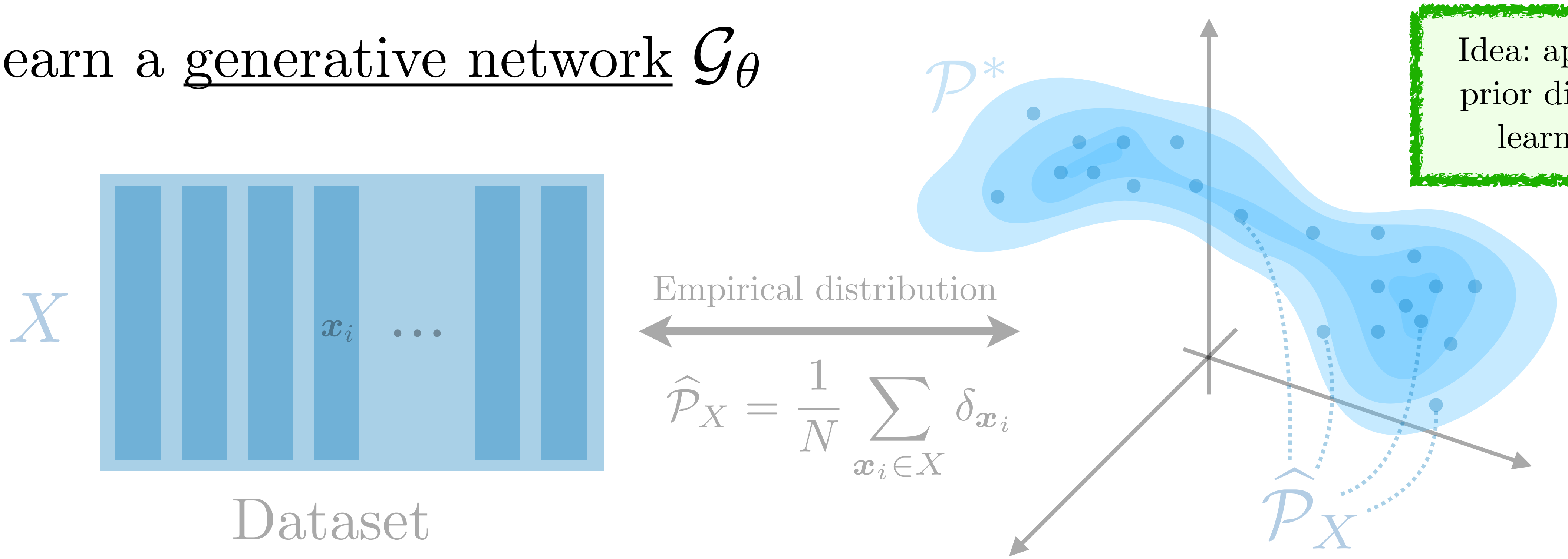
Random "noise"

Generative network

Generative priors in inverse problems (1)

1) Learn a generative network $\mathcal{G}_\theta$

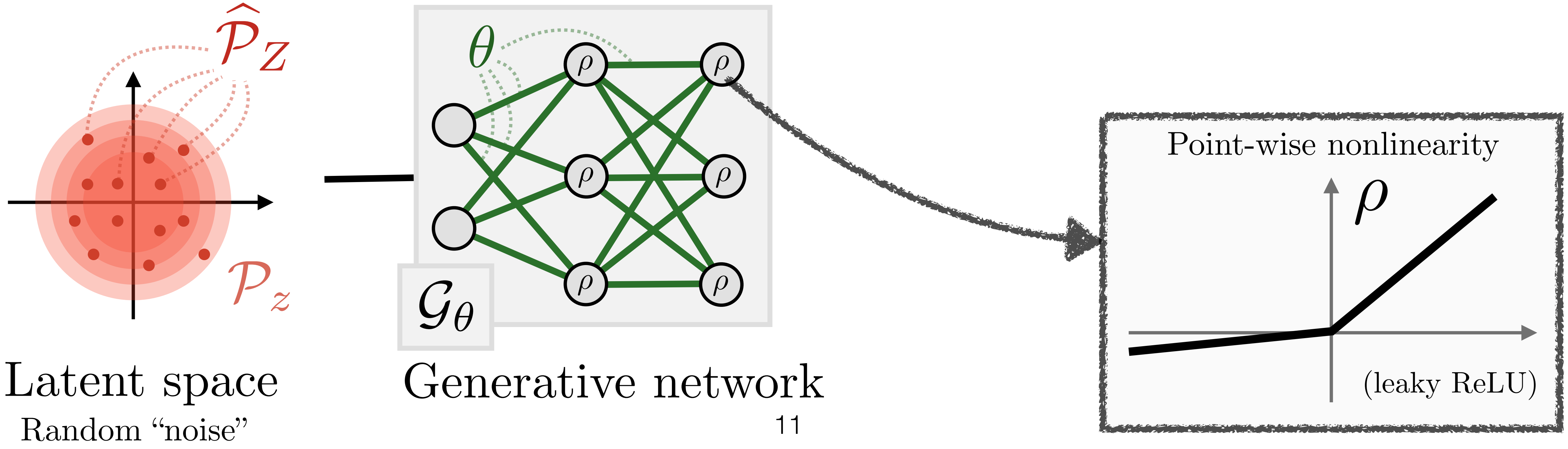
Idea: approximate true prior distribution from learning examples



We have **samples** (signals) from a high-dimensional **distribution**...

... and a way to generate “artificial” samples...

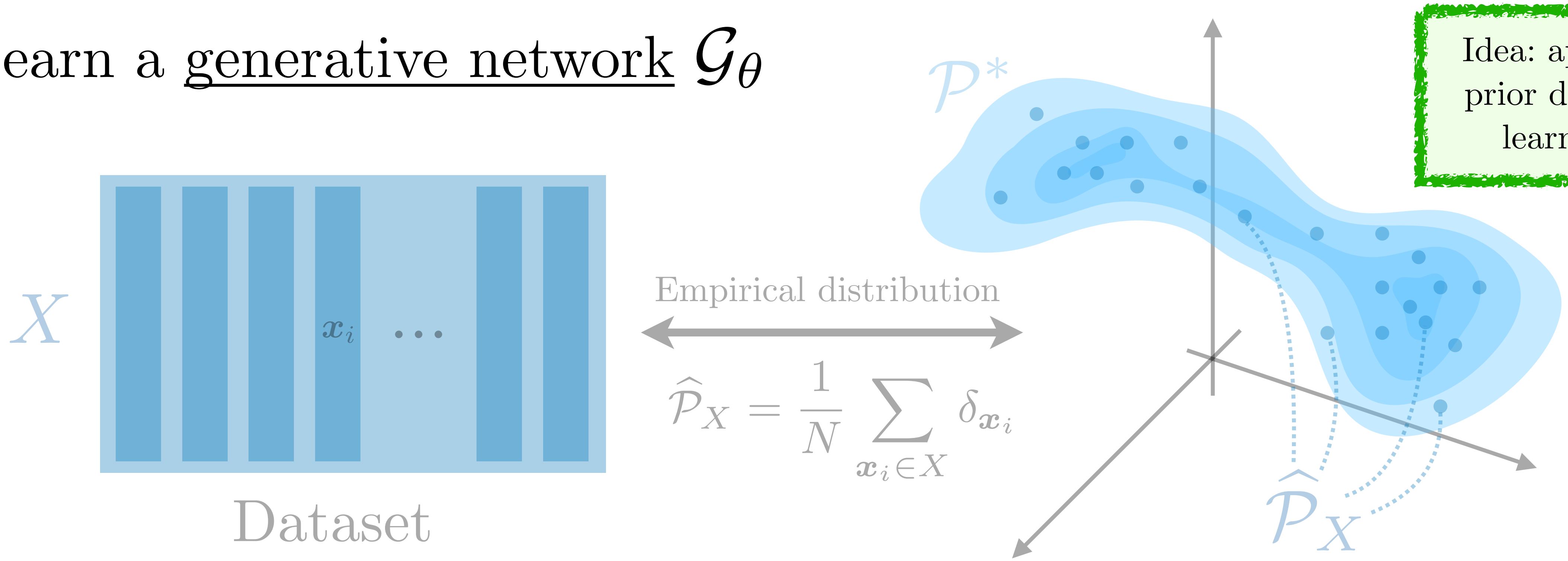
$$\mathcal{G}_\theta(\boldsymbol{z}) = \rho(\Theta_L \cdot \rho(\Theta_{L-1} \cdots \rho(\Theta_2 \cdot \rho(\Theta_1 \cdot \boldsymbol{z}))))$$



Generative priors in inverse problems (1)

1) Learn a generative network $\mathcal{G}_\theta$

Idea: approximate true prior distribution from learning examples



Dataset

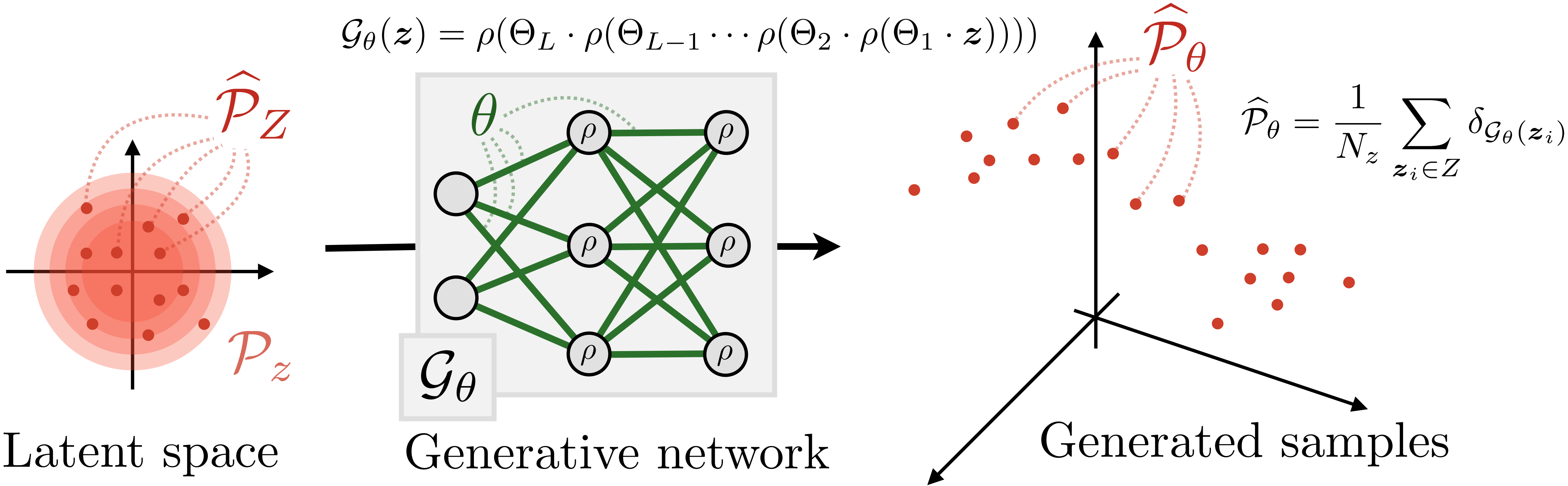
Empirical distribution

$$\widehat{\mathcal{P}}_X = \frac{1}{N} \sum_{x_i \in X} \delta_{x_i}$$

We have **samples** (signals) from a high-dimensional **distribution**...

... and a way to generate “artificial” samples...

$$\mathcal{G}_\theta(z) = \rho(\Theta_L \cdot \rho(\Theta_{L-1} \cdots \rho(\Theta_2 \cdot \rho(\Theta_1 \cdot z))))$$



$$\widehat{\mathcal{P}}_\theta = \frac{1}{N_z} \sum_{z_i \in Z} \delta_{\mathcal{G}_\theta(z_i)}$$

Latent space

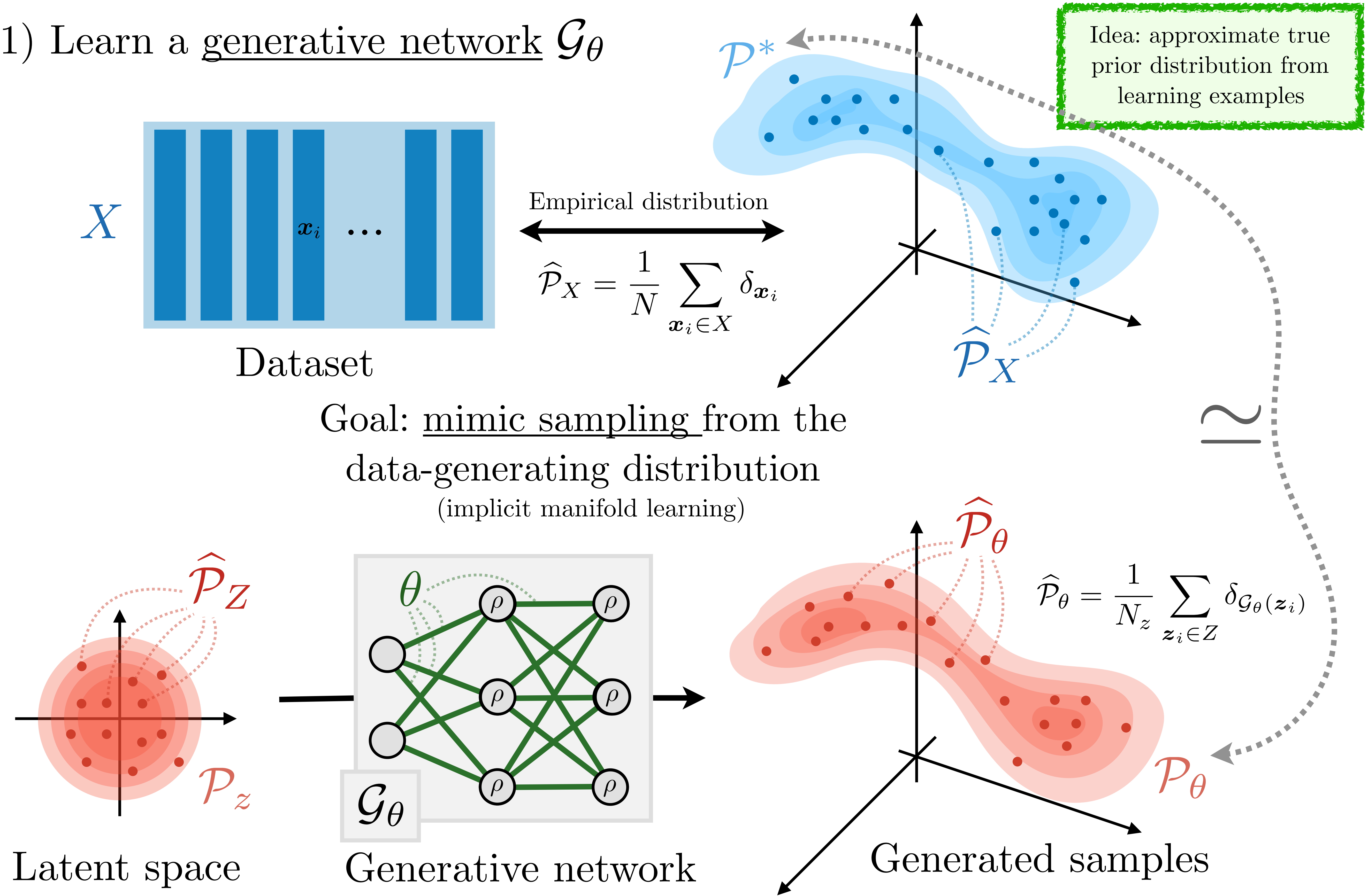
Random “noise”

Generative network

Generated samples

Generative priors in inverse problems (1)

1) Learn a generative network $\mathcal{G}_\theta$



Idea: approximate true prior distribution from learning examples

Goal: mimic sampling from the data-generating distribution

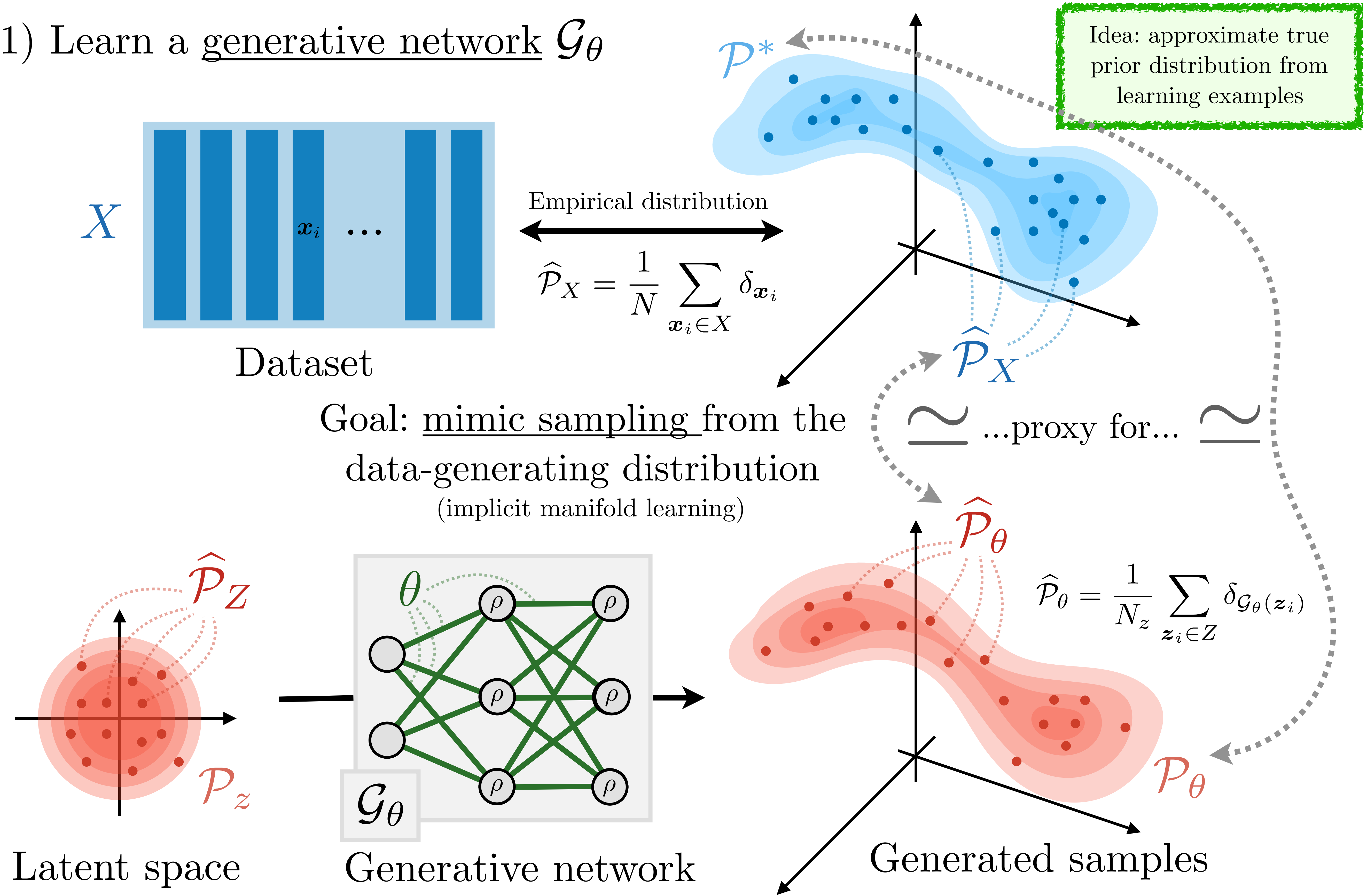
(implicit manifold learning)

$$\widehat{\mathcal{P}}_X = \frac{1}{N} \sum_{\boldsymbol{x}_i \in X} \delta_{\boldsymbol{x}_i}$$

$$\widehat{\mathcal{P}}_\theta = \frac{1}{N_z} \sum_{\boldsymbol{z}_i \in Z} \delta_{\mathcal{G}_\theta(\boldsymbol{z}_i)}$$

Generative priors in inverse problems (1)

1) Learn a generative network $\mathcal{G}_\theta$



Idea: approximate true prior distribution from learning examples

Dataset

Empirical distribution

$$\widehat{\mathcal{P}}_X = \frac{1}{N}\sum_{\boldsymbol{x}_i \in X} \delta_{\boldsymbol{x}_i}$$

...proxy for...

Goal: mimic sampling from the data-generating distribution

(implicit manifold learning)

$$\widehat{\mathcal{P}}_\theta = \frac{1}{N_z}\sum_{\boldsymbol{z}_i \in Z} \delta_{\mathcal{G}_\theta(\boldsymbol{z}_i)}$$

Latent space

Random “noise”

Generative network

Generated samples

Generative priors in inverse problems (2)

1) Learn a generative network $\mathcal{G}_\theta$

2) Use the generative network in the inverse problem

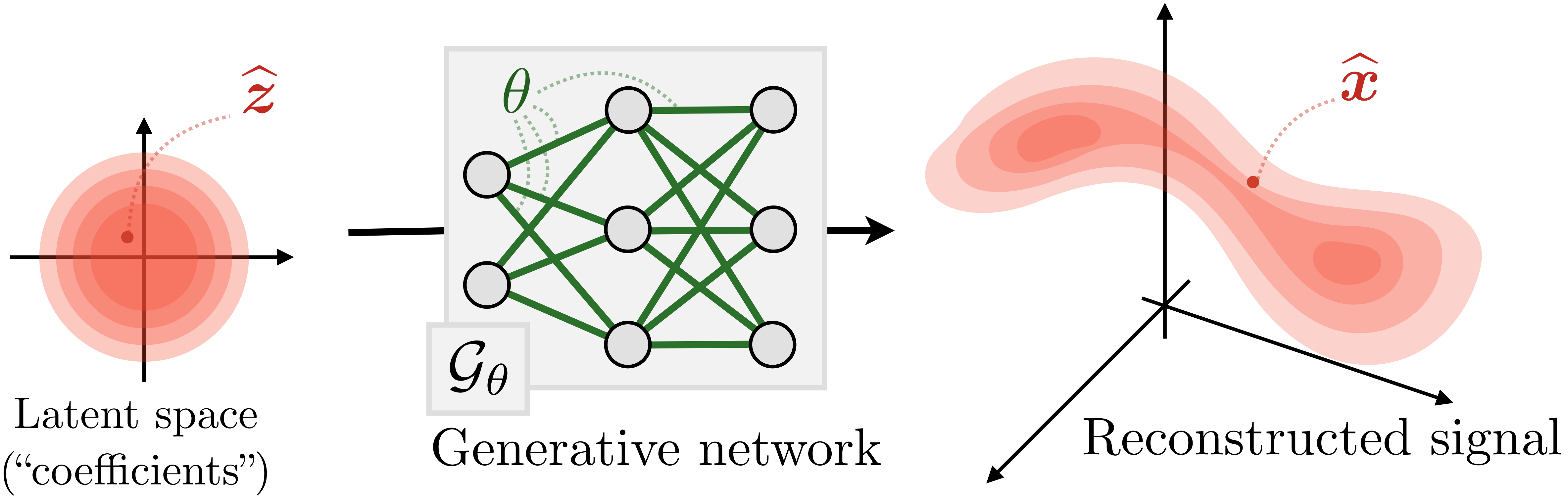
Generative priors in inverse problems (2)

1) Learn a generative network $\mathcal{G}_\theta$

2) Use the generative network in the inverse problem

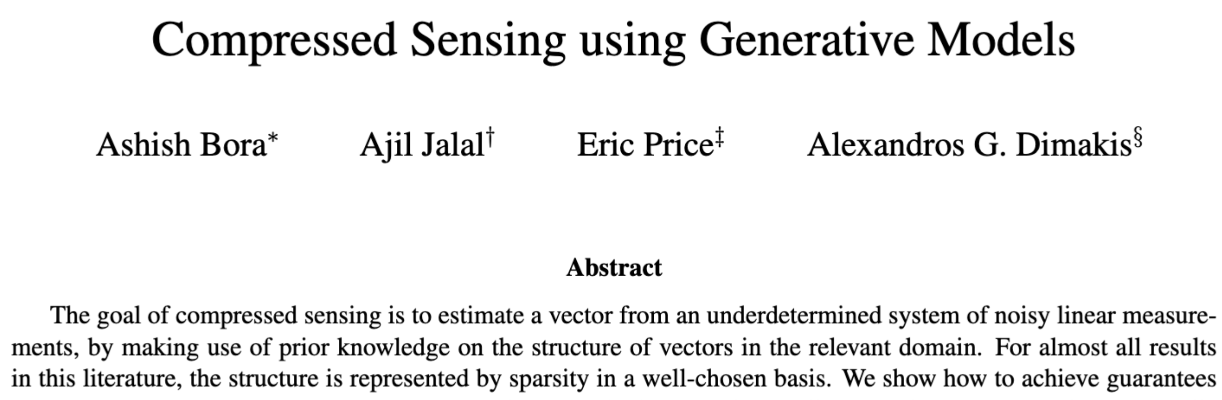
e.g. reconstructed signal in the range of the network

$$\widehat{\boldsymbol{x}} = \mathcal{G}_\theta(\widehat{\boldsymbol{z}}) \quad \text{with} \quad \widehat{\boldsymbol{z}} = \arg\min_{\boldsymbol{z}} \; \|\boldsymbol{y} - \Phi(\mathcal{G}_\theta(\boldsymbol{z}))\|_2^2$$



Generative priors in inverse problems

Some examples...



Compressed Sensing using Generative Models

Ashish Bora* Ajil Jalal† Eric Price‡ Alexandros G. Dimakis§

9 Mar 2017

Abstract

The goal of compressed sensing is to estimate a vector from an underdetermined system of noisy linear measurements, by making use of prior knowledge on the structure of vectors in the relevant domain. For almost all results in this literature, the structure is represented by sparsity in a well-chosen basis. We show how to achieve guarantees similar to

near the ra

$O(k \log L)$

using gene

can use 5-1

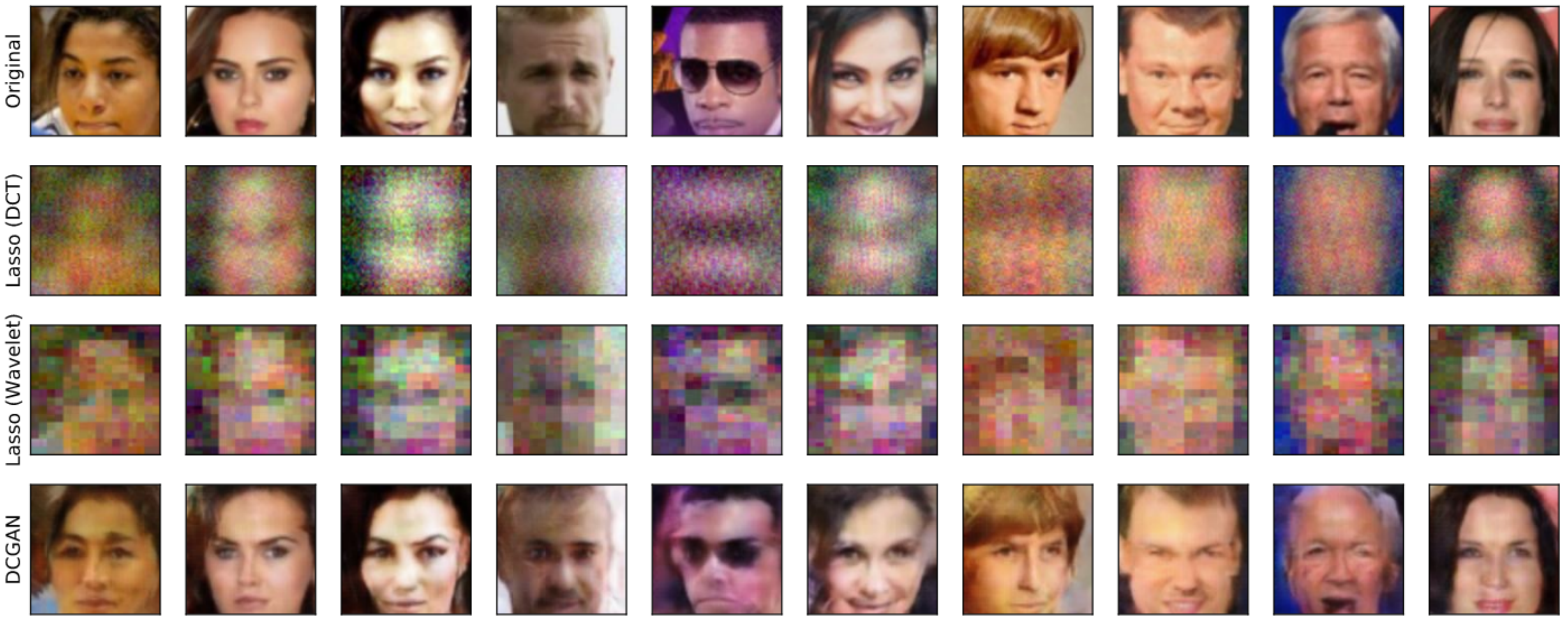
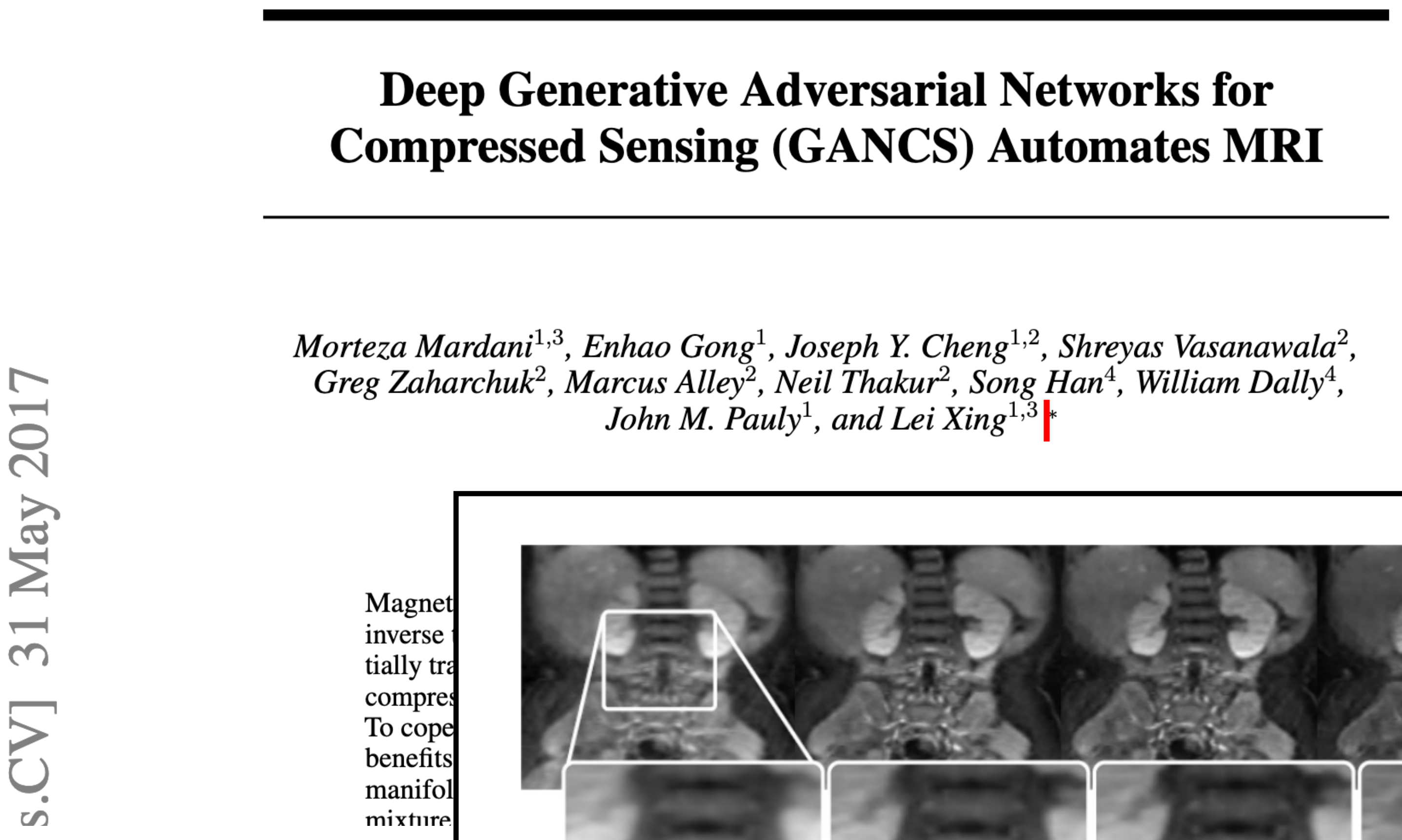


Figure 3: Reconstruction results on celebA with $m = 500$ measurements (of $n = 12288$ dimensional vector). We show original images (top row), and reconstructions by Lasso with DCT basis (second row), Lasso with wavelet basis (third row), and our algorithm (last row).

Generative priors in inverse problems

Some examples...



Deep Generative Adversarial Networks for Compressed Sensing (GANCS) Automates MRI

Morteza Mardani[1,3], Enhao Gong[1], Joseph Y. Cheng[1,2], Shreyas Vasanawala[2],
Greg Zaharchuk[2], Marcus Alley[2], Neil Thakur[2], Song Han[4], William Dally[4],
John M. Pauly[1], and Lei Xing[1,3] *

s.CV] 31 May 2017

Magnet
inverse
tially tra
compres
To cope
benefits
manifol
mixture

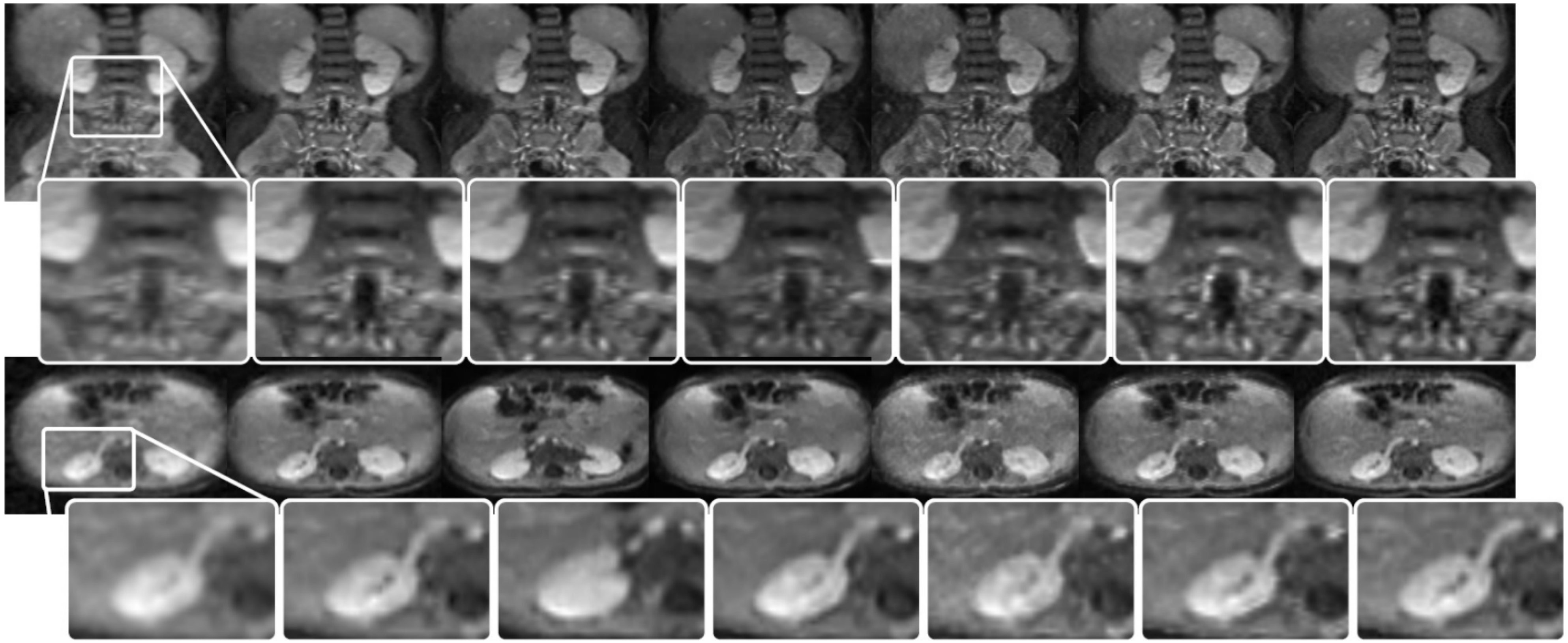
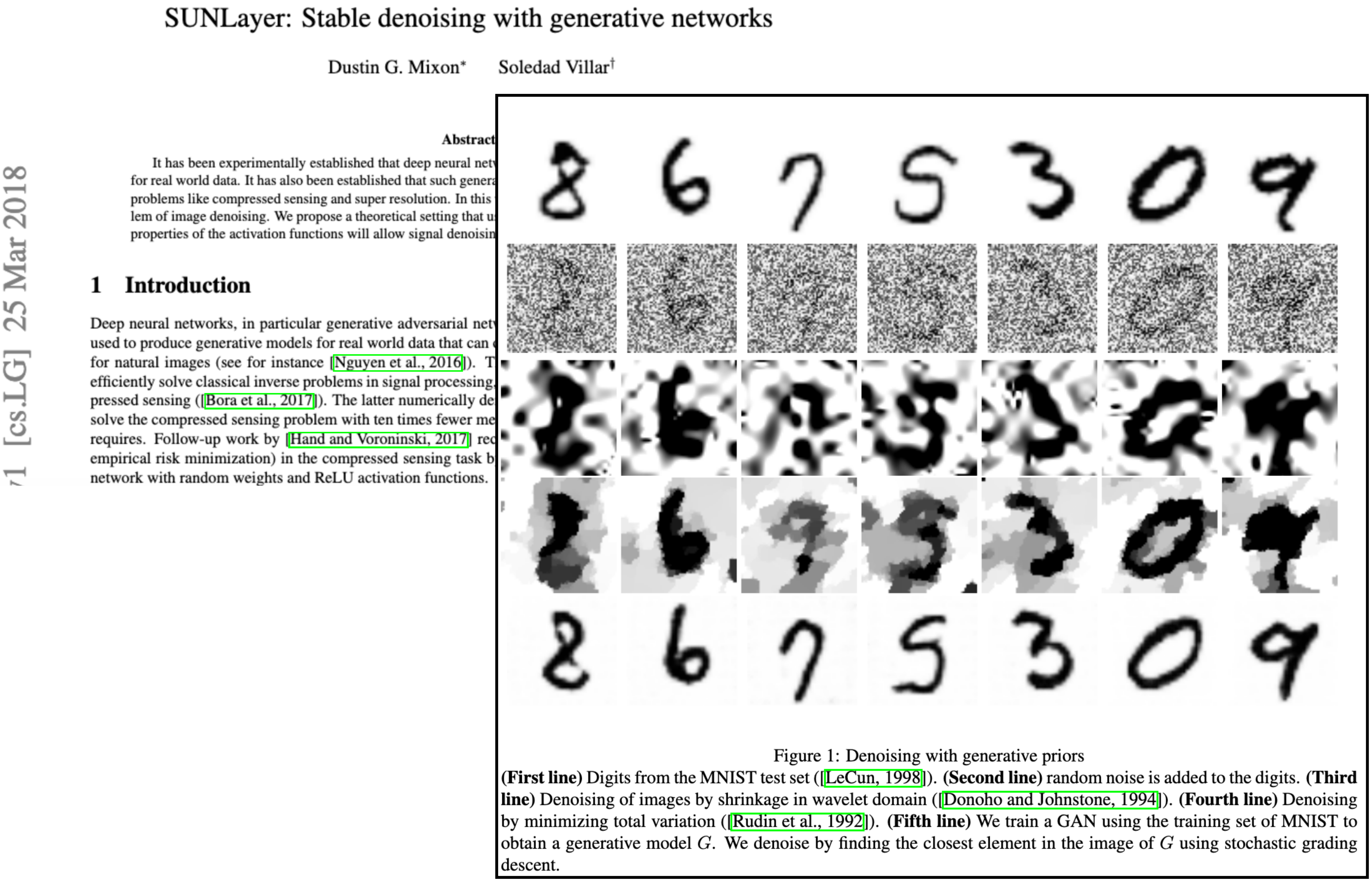


Fig. 2: Representative coronal (1st row) and axial (3rd row) images for a test patient retrieved by ZF (1st), CS-WV (2nd), ℓ_2-net (3th), ℓ_1-net (4th), GAN (5th), GANCS (6th), and gold-standard (7th).

Generative priors in inverse problems

Some examples...



SUNLayer: Stable denoising with generative networks

Dustin G. Mixon* Soledad Villar†

Abstract

It has been experimentally established that deep neural netw
for real world data. It has also been established that such genera
problems like compressed sensing and super resolution. In this
lem of image denoising. We propose a theoretical setting that u
properties of the activation functions will allow signal denoisin

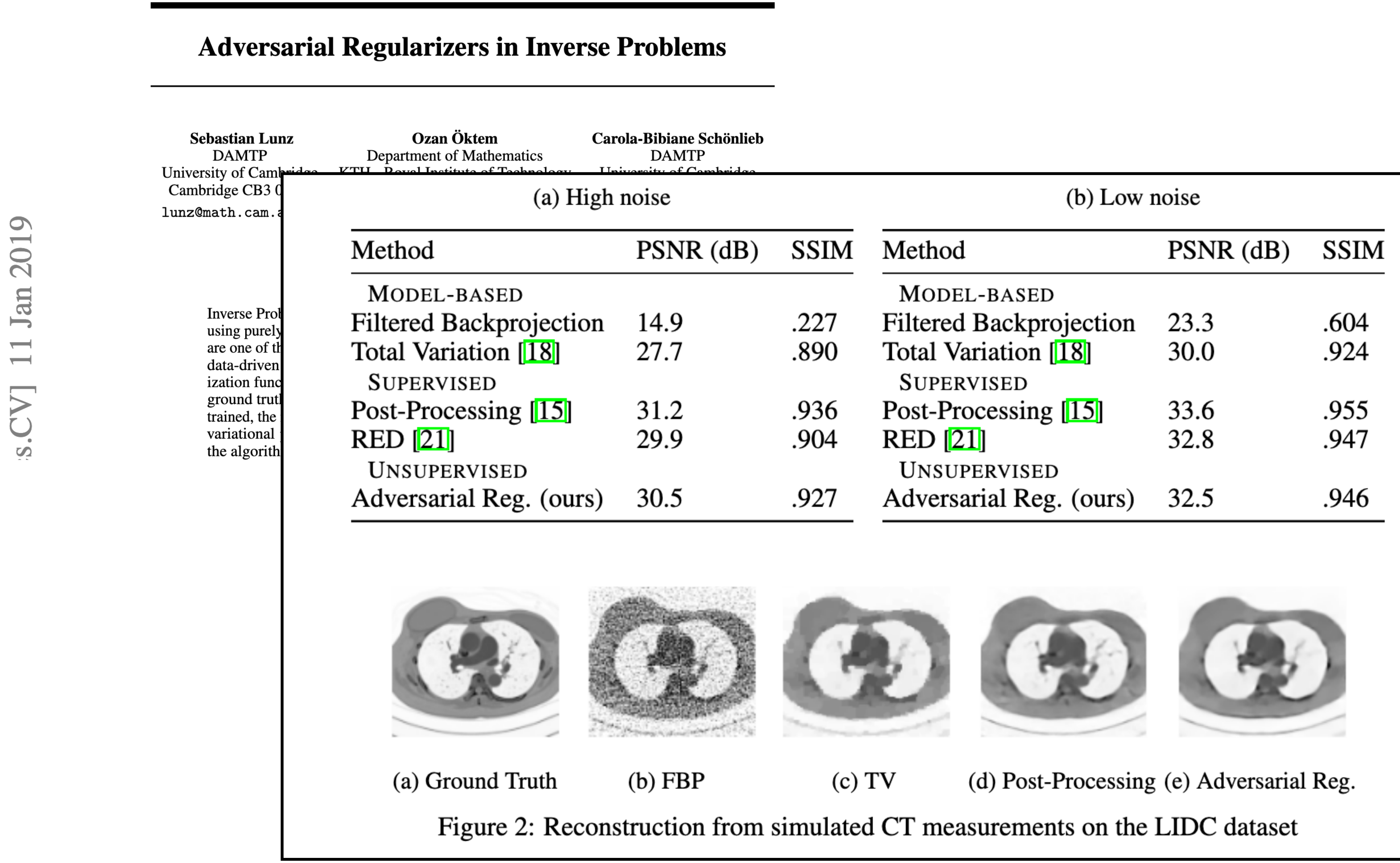
1 Introduction

Deep neural networks, in particular generative adversarial netw
used to produce generative models for real world data that can
for natural images (see for instance [Nguyen et al., 2016]). T
efficiently solve classical inverse problems in signal processing,
pressed sensing ([Bora et al., 2017]). The latter numerically de
solve the compressed sensing problem with ten times fewer me
requires. Follow-up work by [Hand and Voroninski, 2017] rec
empirical risk minimization) in the compressed sensing task b
network with random weights and ReLU activation functions.

Figure 1: Denoising with generative priors
(First line) Digits from the MNIST test set ([LeCun, 1998]). **(Second line)** random noise is added to the digits. **(Third line)** Denoising of images by shrinkage in wavelet domain ([Donoho and Johnstone, 1994]). **(Fourth line)** Denoising by minimizing total variation ([Rudin et al., 1992]). **(Fifth line)** We train a GAN using the training set of MNIST to obtain a generative model G. We denoise by finding the closest element in the image of G using stochastic grading descent.

Generative priors in inverse problems

Some examples...



s.CV] 11 Jan 2019

Adversarial Regularizers in Inverse Problems

Sebastian Lunz
DAMTP
University of Cam
Cambridge CB3 0
lunz@math.cam.a

Ozan Öktem
Department of Mathematics
KTH – Royal Institute of Technology

Carola-Bibiane Schönlieb
DAMTP
University of Cambridge

Inverse Prob
using purely
are one of th
data-driven
ization func
ground trut
trained, the
variational
the algorith

(a) High noise

Method	PSNR (dB)	SSIM
Model-based		
Filtered Backprojection	14.9	.227
Total Variation [18]	27.7	.890
Supervised		
Post-Processing [15]	31.2	.936
RED [21]	29.9	.904
Unsupervised		
Adversarial Reg. (ours)	30.5	.927

(b) Low noise

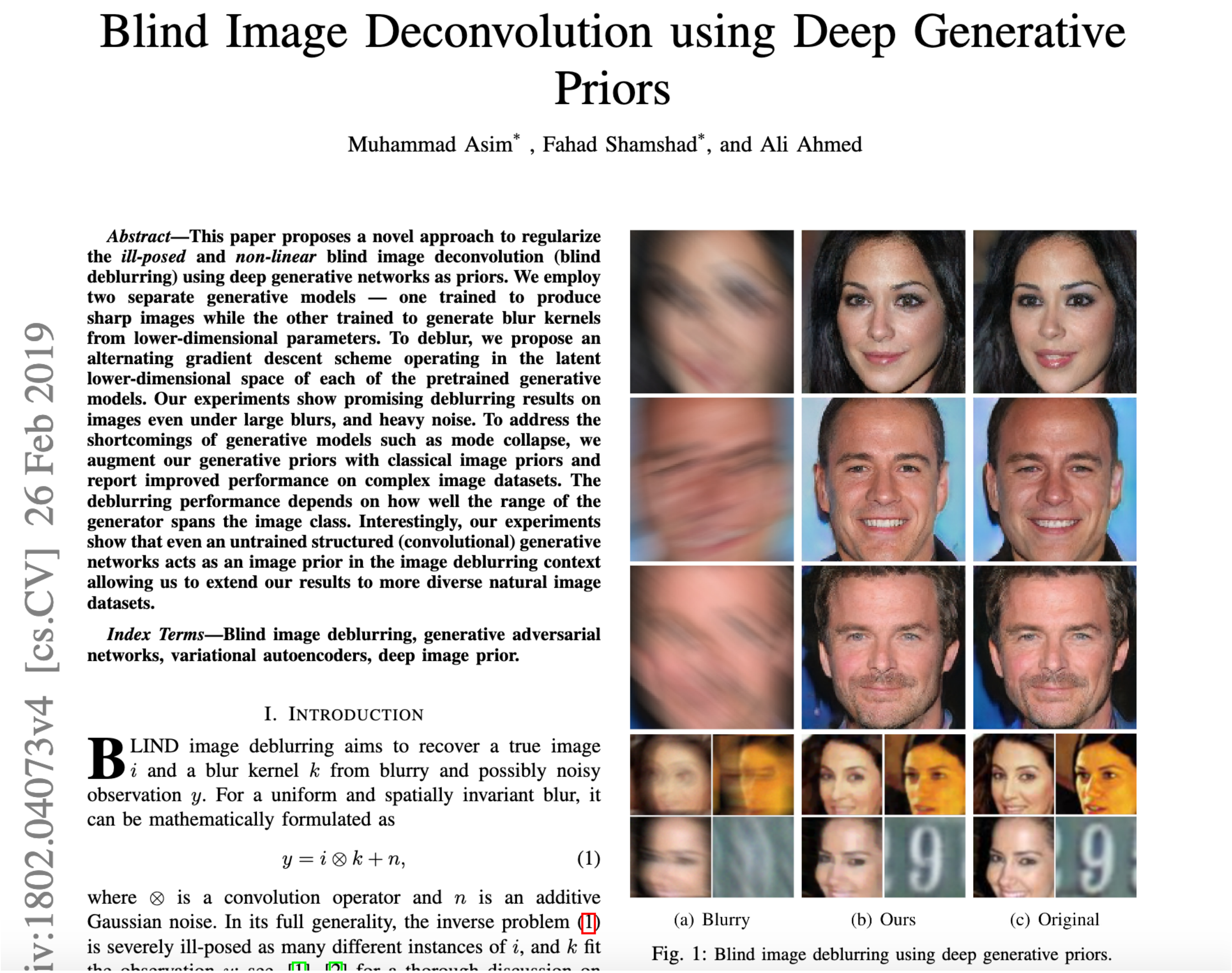
Method	PSNR (dB)	SSIM
Model-based		
Filtered Backprojection	23.3	.604
Total Variation [18]	30.0	.924
Supervised		
Post-Processing [15]	33.6	.955
RED [21]	32.8	.947
Unsupervised		
Adversarial Reg. (ours)	32.5	.946

(a) Ground Truth (b) FBP (c) TV (d) Post-Processing (e) Adversarial Reg.

Figure 2: Reconstruction from simulated CT measurements on the LIDC dataset

Generative priors in inverse problems

Some examples...



Blind Image Deconvolution using Deep Generative Priors

Muhammad Asim* , Fahad Shamshad*, and Ali Ahmed

***Abstract*—This paper proposes a novel approach to regularize the *ill-posed* and *non-linear* blind image deconvolution (blind deblurring) using deep generative networks as priors. We employ two separate generative models — one trained to produce sharp images while the other trained to generate blur kernels from lower-dimensional parameters. To deblur, we propose an alternating gradient descent scheme operating in the latent lower-dimensional space of each of the pretrained generative models. Our experiments show promising deblurring results on images even under large blurs, and heavy noise. To address the shortcomings of generative models such as mode collapse, we augment our generative priors with classical image priors and report improved performance on complex image datasets. The deblurring performance depends on how well the range of the generator spans the image class. Interestingly, our experiments show that even an untrained structured (convolutional) generative networks acts as an image prior in the image deblurring context allowing us to extend our results to more diverse natural image datasets.**

***Index Terms*—Blind image deblurring, generative adversarial networks, variational autoencoders, deep image prior.**

I. INTRODUCTION

BLIND image deblurring aims to recover a true image i and a blur kernel k from blurry and possibly noisy observation y. For a uniform and spatially invariant blur, it can be mathematically formulated as

$$y = i \otimes k + n, \tag{1}$$

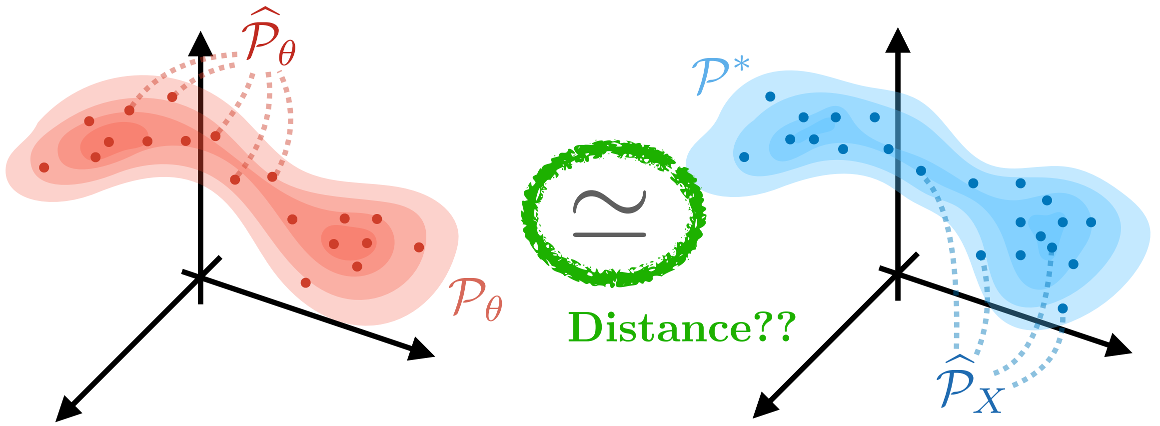
where $\otimes$ is a convolution operator and n is an additive Gaussian noise. In its full generality, the inverse problem (1) is severely ill-posed as many different instances of i, and k fit the observation y, see [1], [2] for a thorough discussion on

(a) Blurry (b) Ours (c) Original

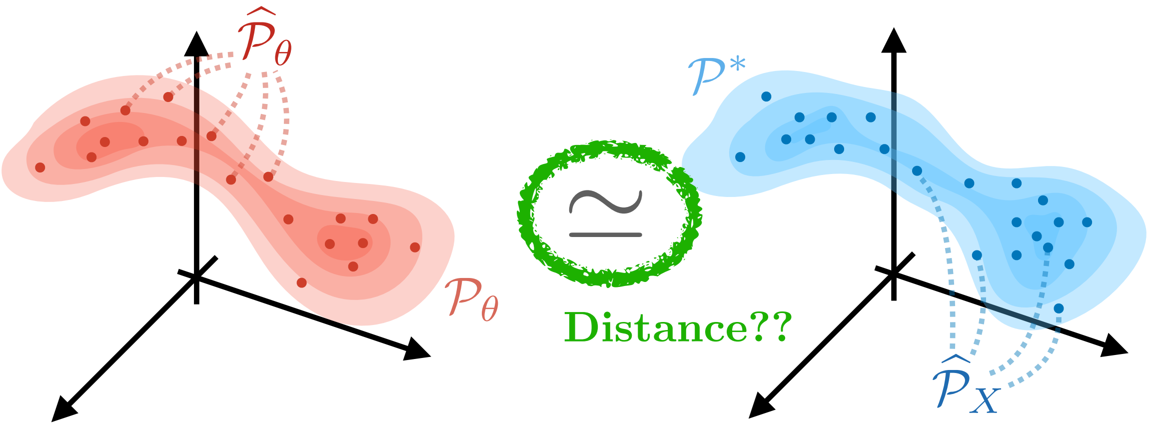
Fig. 1: Blind image deblurring using deep generative priors.

It's nice! ...where's the catch?

How to learn the generative network $\mathcal{G}_\theta$



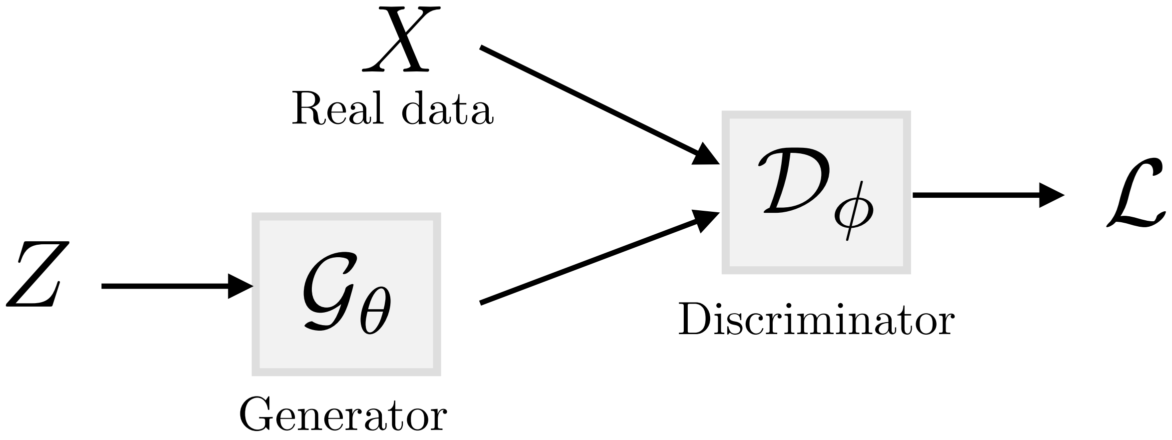
It’s nice! …where’s the catch?



<u>How</u> to learn the generative network $\mathcal{G}_\theta$

1) Golden standard: Generative Adversarial Networks

Learn a second “discriminator” network that classifies real/fake at the same time as the generator



X

Real data

Z

$\mathcal{G}_\theta$

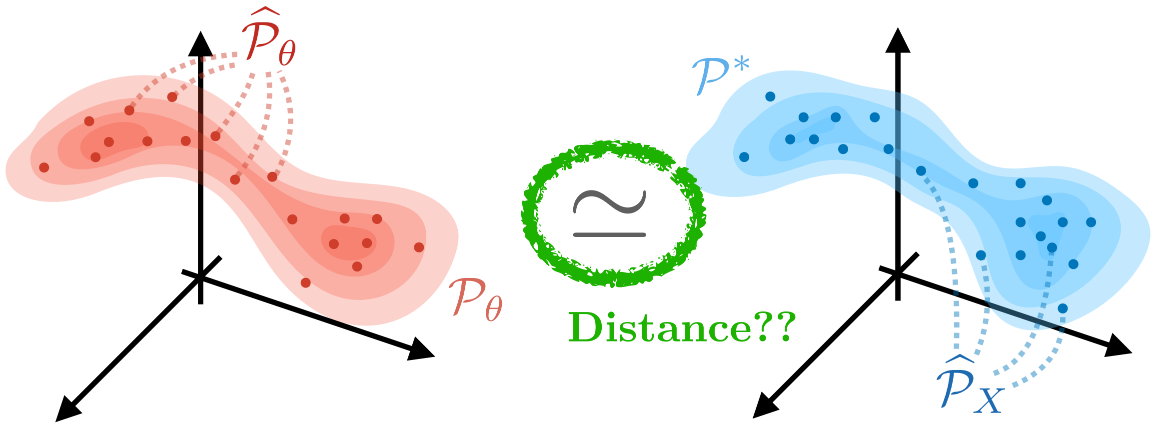
Generator

$\mathcal{D}_\phi$

Discriminator

$\mathcal{L}$

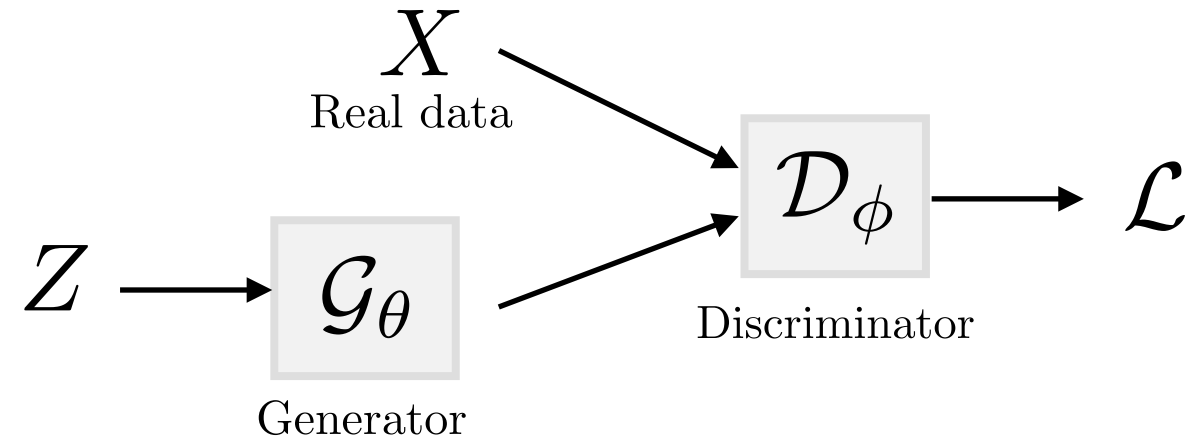
It's nice! ...where's the catch?



How to learn the generative network $\mathcal{G}_\theta$

1) Golden standard: Generative Adversarial Networks

Learn a second "discriminator" network that classifies real/fake at the same time as the generator



X

Real data

Z

$\mathcal{G}_\theta$

Generator

$\mathcal{D}_\phi$

Discriminator

$\mathcal{L}$

Very difficult to train (due to balancing of training discriminator/generator)

$$\min_\theta \max_\phi \mathcal{L}(\theta, \phi)$$

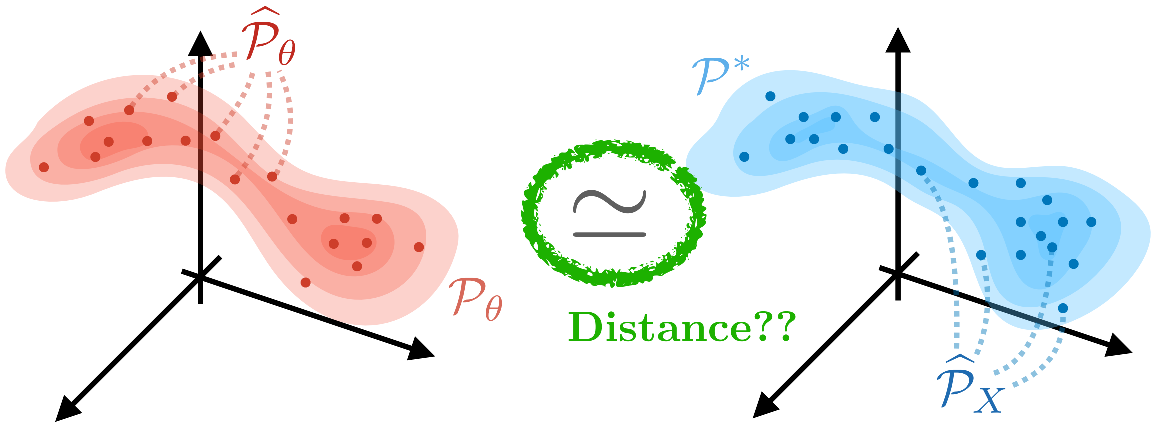
Ferenc Huszár

@fhuszar

GAN equilibrium

On twitter...

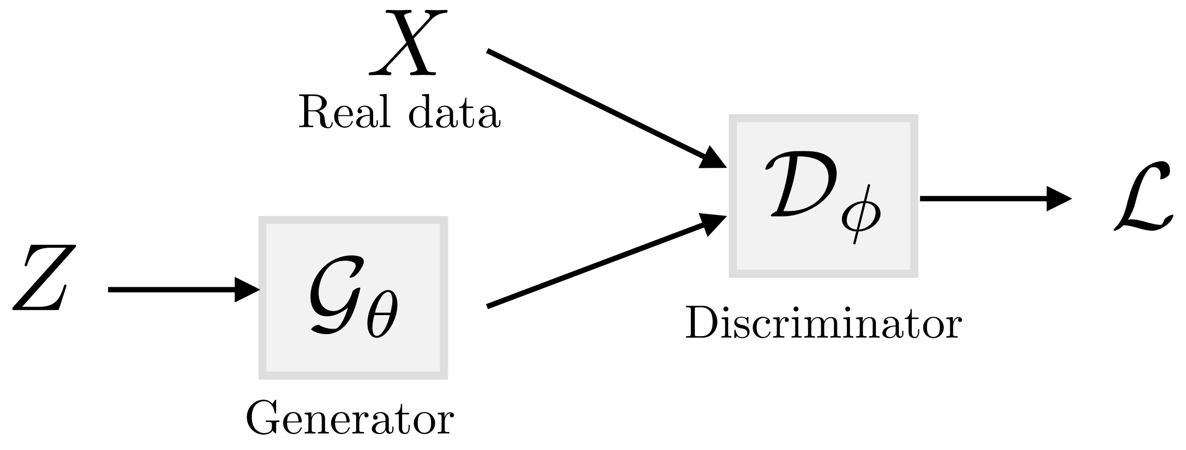
It’s nice! …where’s the catch?



How to learn the generative network $\mathcal{G}_\theta$

1) Golden standard: Generative Adversarial Networks

Learn a second “discriminator” network that classifies real/fake at the same time as the generator



Very difficult to train (due to balancing of training discriminator/generator)

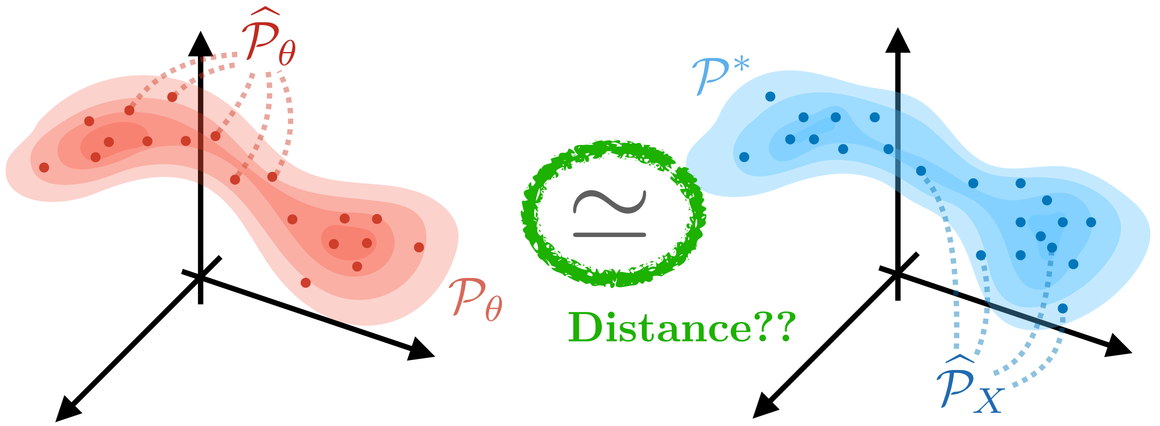
$$\min_\theta \max_\phi \mathcal{L}(\theta, \phi)$$

2) Maximum Mean Discrepancy

$$\min_\theta \ \mathrm{MMD}_\kappa(\widehat{\mathcal{P}}_X, \widehat{\mathcal{P}}_\theta)$$

Easier to train (no balancing)…

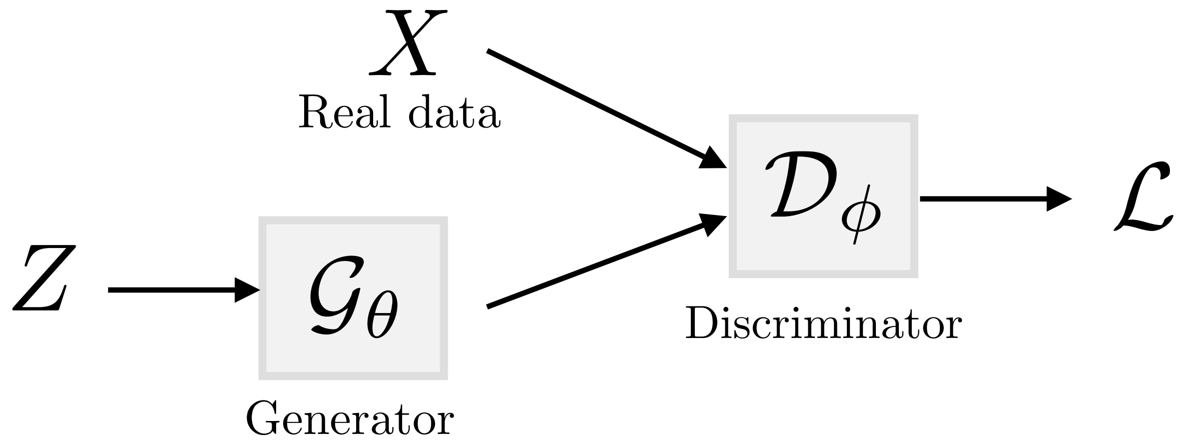
It’s nice! …where’s the catch?



How to learn the generative network $\mathcal{G}_\theta$

1) Golden standard: Generative Adversarial Networks

Learn a second “discriminator” network that classifies real/fake at the same time as the generator



X

Real data

Z

$\mathcal{G}_\theta$

Generator

$\mathcal{D}_\phi$

Discriminator

$\mathcal{L}$

Very difficult to train (due to balancing of training discriminator/generator)

$$\min_{\theta} \max_{\phi} \mathcal{L}(\theta, \phi)$$

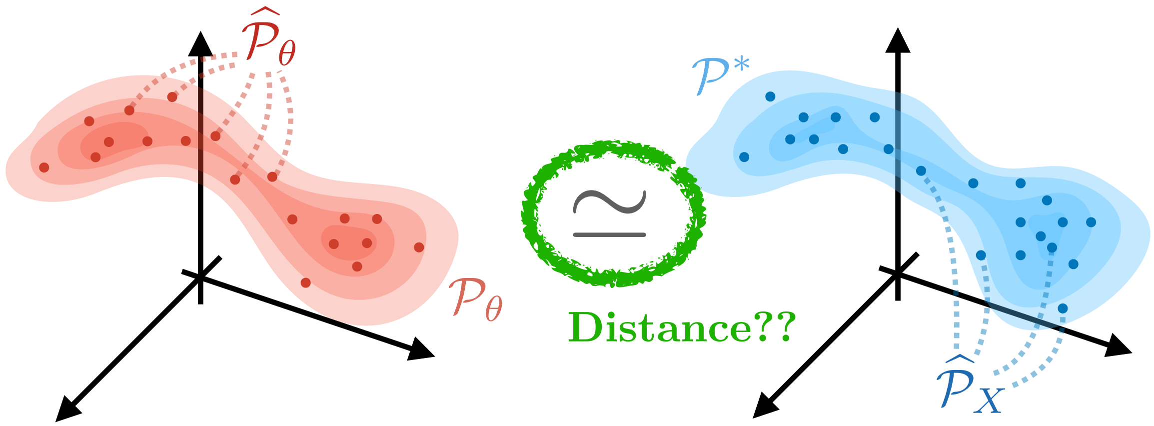
2) Maximum Mean Discrepancy

$$\min_{\theta} \ \mathrm{MMD}_\kappa(\widehat{\mathcal{P}}_X, \widehat{\mathcal{P}}_\theta)$$

$$\sum_{\substack{\boldsymbol{x}_i \in X \\ \boldsymbol{x}_j \in X}} \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) - 2 \sum_{\substack{\boldsymbol{x}_i \in X \\ \boldsymbol{z}_j \in Z}} \kappa(\boldsymbol{x}_i, \mathcal{G}_\theta(\boldsymbol{z}_j)) + \sum_{\substack{\boldsymbol{z}_i \in Z \\ \boldsymbol{z}_j \in Z}} \kappa(\mathcal{G}_\theta(\boldsymbol{z}_i), \mathcal{G}_\theta(\boldsymbol{z}_j))$$

Easier to train (no balancing)…

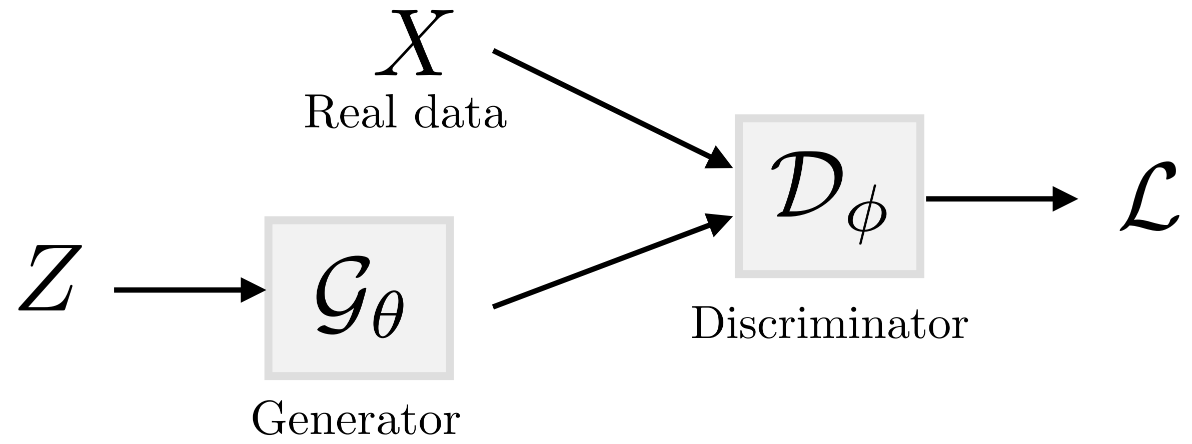
It's nice! ...where's the catch?



How to learn the generative network $\mathcal{G}_\theta$

1) Golden standard: Generative Adversarial Networks

Learn a second "discriminator" network that classifies real/fake at the same time as the generator



Very difficult to train (due to balancing of training discriminator/generator)

$$\min_\theta \max_\phi \mathcal{L}(\theta, \phi)$$

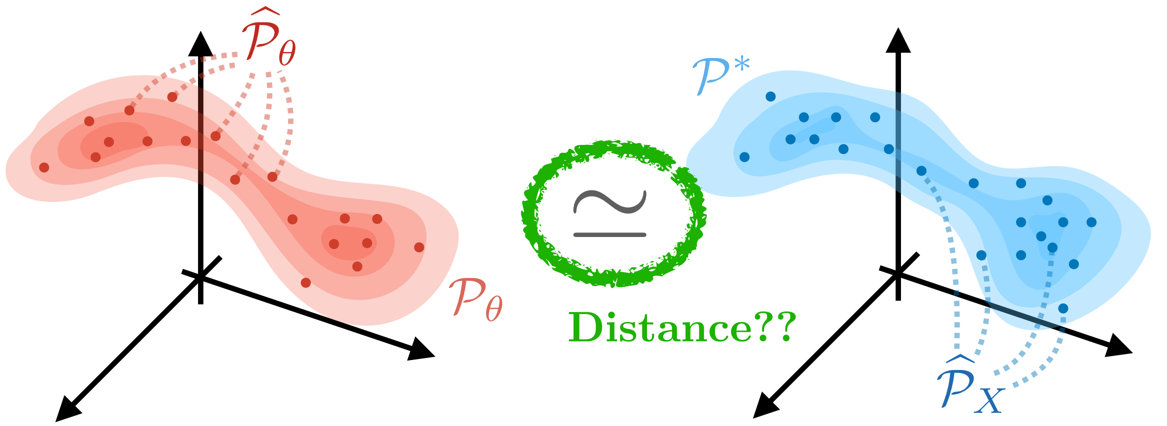
2) Maximum Mean Discrepancy

$$\min_\theta \; \mathrm{MMD}_\kappa(\widehat{\mathcal{P}}_X, \widehat{\mathcal{P}}_\theta)$$

$$\sum_{\substack{\boldsymbol{x}_i \in X \\ \boldsymbol{x}_j \in X}} \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) - 2 \sum_{\substack{\boldsymbol{x}_i \in X \\ \boldsymbol{z}_j \in Z}} \overbrace{\kappa(\boldsymbol{x}_i, \mathcal{G}_\theta(\boldsymbol{z}_j))}^{\text{Similarity between samples}} + \sum_{\substack{\boldsymbol{z}_i \in Z \\ \boldsymbol{z}_j \in Z}} \kappa(\mathcal{G}_\theta(\boldsymbol{z}_i), \mathcal{G}_\theta(\boldsymbol{z}_j))$$

Easier to train (no balancing)...

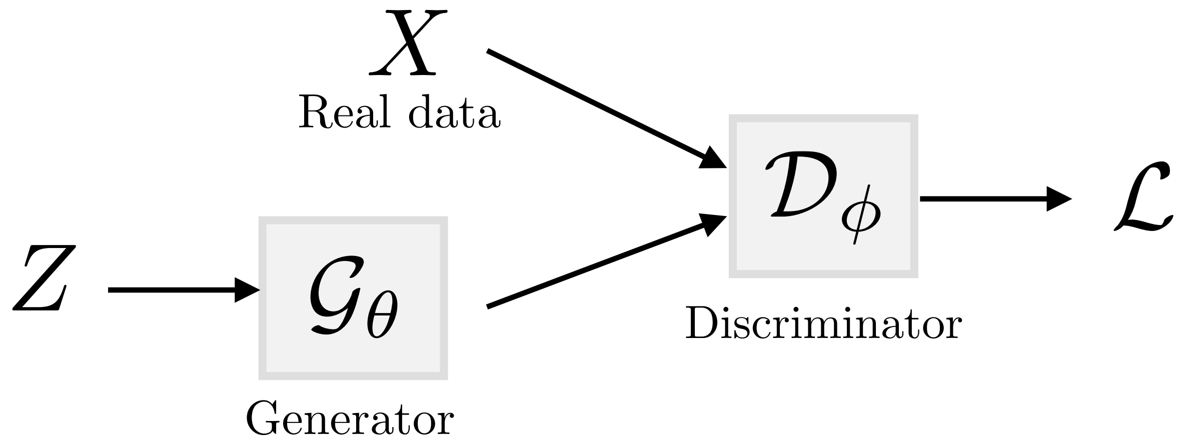
It's nice! ...where's the catch?



How to learn the generative network $\mathcal{G}_\theta$

1) Golden standard: Generative Adversarial Networks

Learn a second "discriminator" network that classifies real/fake at the same time as the generator



Very difficult to train (due to balancing of training discriminator/generator)

$$\min_\theta \max_\phi \mathcal{L}(\theta, \phi)$$

2) Maximum Mean Discrepancy

$$\min_\theta \ \mathrm{MMD}_\kappa(\widehat{\mathcal{P}}_X, \widehat{\mathcal{P}}_\theta)$$

Similarity between samples

$$\sum_{\substack{\boldsymbol{x}_i \in X \\ \boldsymbol{x}_j \in X}} \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) - 2 \sum_{\substack{\boldsymbol{x}_i \in X \\ \boldsymbol{z}_j \in Z}} \kappa(\boldsymbol{x}_i, \mathcal{G}_\theta(\boldsymbol{z}_j)) + \sum_{\substack{\boldsymbol{z}_i \in Z \\ \boldsymbol{z}_j \in Z}} \kappa(\mathcal{G}_\theta(\boldsymbol{z}_i), \mathcal{G}_\theta(\boldsymbol{z}_j))$$

equivalent to (for later)

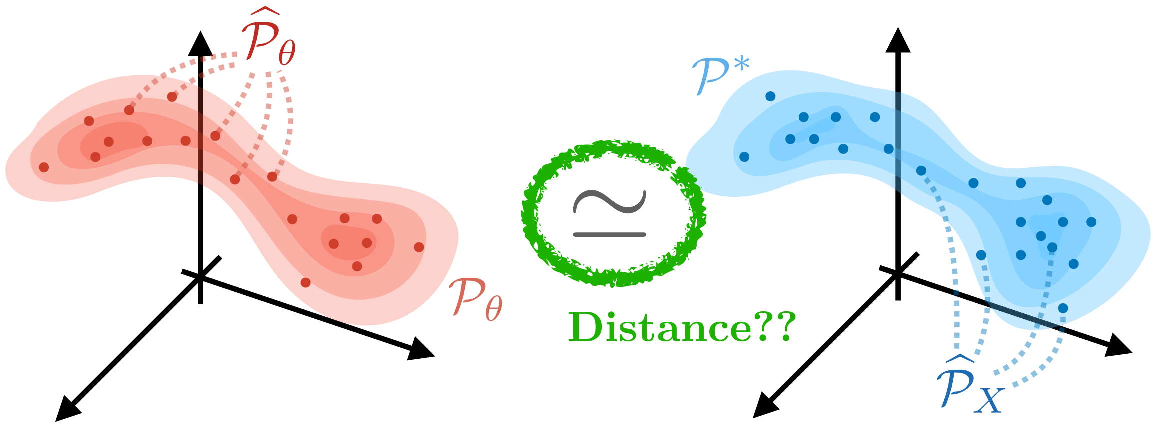
$$\mathbb{E}_{\boldsymbol{\omega} \sim \Lambda} \left| \sum_{\boldsymbol{x}_i \in X} e^{\mathrm{i}\boldsymbol{\omega}^T \boldsymbol{x}_i} - \sum_{\boldsymbol{z}_i \in Z} e^{\mathrm{i}\boldsymbol{\omega}^T \mathcal{G}_\theta(\boldsymbol{z}_i)} \right|^2$$

With $\Lambda = \mathcal{F}\kappa$

Easier to train (no balancing)...

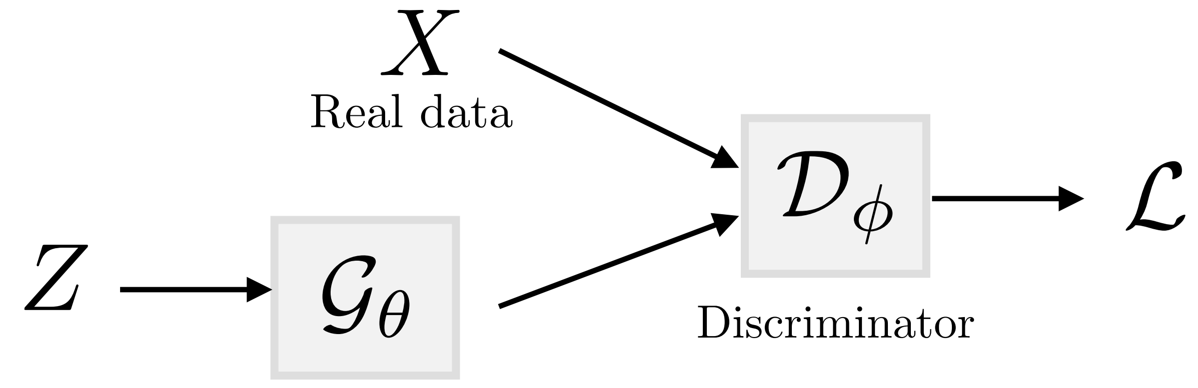
It's nice! ...where's the catch?

<u>How</u> to learn the generative network $\mathcal{G}_\theta$



1) Golden standard: Generative Adversarial Networks

Learn a second "discriminator" network that classifies real/fake at the same time as the generator



X
Real data

$Z \rightarrow \mathcal{G}_\theta$
Generator

$\mathcal{D}_\phi \rightarrow \mathcal{L}$
Discriminator

<u>Very difficult</u> to train (due to balancing of training discriminator/generator)

$$\min_\theta \max_\phi \mathcal{L}(\theta, \phi)$$

2) Maximum Mean Discrepancy

$$\min_\theta \ \mathrm{MMD}_\kappa(\widehat{\mathcal{P}}_X, \widehat{\mathcal{P}}_\theta)$$

Similarity between samples

$$\sum_{\substack{\boldsymbol{x}_i \in X \\ \boldsymbol{x}_j \in X}} \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) - 2 \sum_{\substack{\boldsymbol{x}_i \in X \\ \boldsymbol{z}_j \in Z}} \kappa(\boldsymbol{x}_i, \mathcal{G}_\theta(\boldsymbol{z}_j)) + \sum_{\substack{\boldsymbol{z}_i \in Z \\ \boldsymbol{z}_j \in Z}} \kappa(\mathcal{G}_\theta(\boldsymbol{z}_i), \mathcal{G}_\theta(\boldsymbol{z}_j))$$

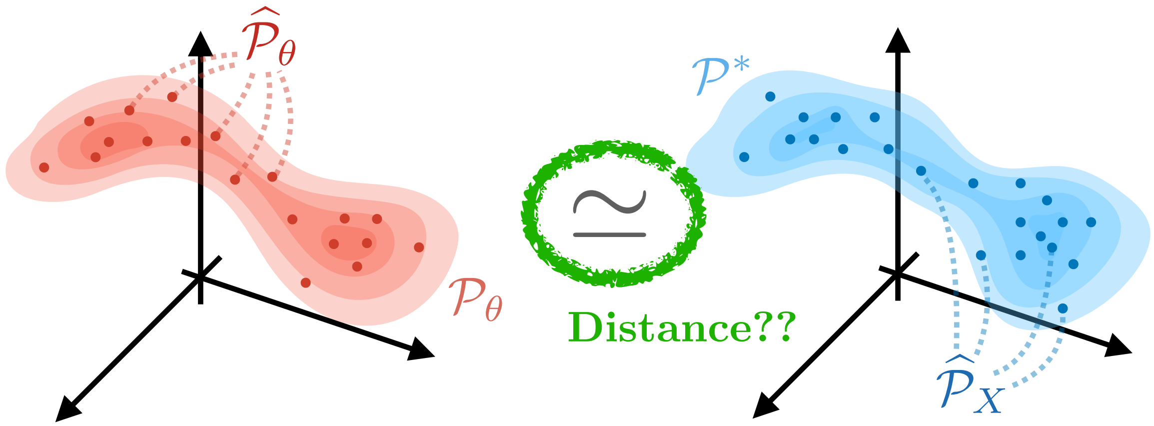
equivalent to (for later)

$$\mathbb{E}_{\boldsymbol{\omega} \sim \Lambda} \left| \sum_{\boldsymbol{x}_i \in X} e^{\mathrm{i}\boldsymbol{\omega}^T \boldsymbol{x}_i} - \sum_{\boldsymbol{z}_i \in Z} e^{\mathrm{i}\boldsymbol{\omega}^T \mathcal{G}_\theta(\boldsymbol{z}_i)} \right|^2$$

With $\Lambda = \mathcal{F}\kappa$

Easier to train (no balancing) but <u>quadratic complexity</u>...

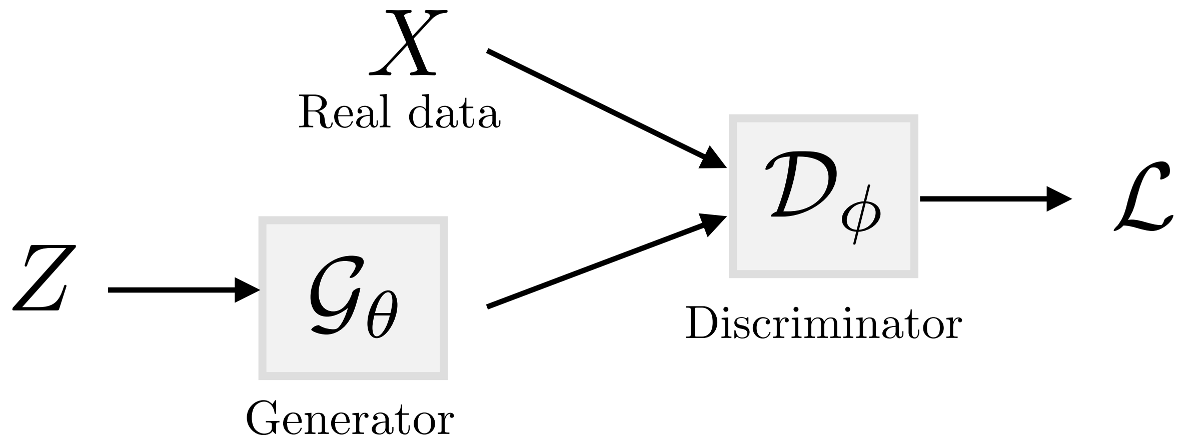
It’s nice! …where’s the catch?



How to learn the generative network $\mathcal{G}_\theta$

1) Golden standard: Generative Adversarial Networks

Learn a second “discriminator” network that classifies real/fake at the same time as the generator



Very difficult to train (due to balancing of training discriminator/generator)

$$\min_\theta \max_\phi \mathcal{L}(\theta, \phi)$$

2) Maximum Mean Discrepancy

$$\min_\theta \ \mathrm{MMD}_\kappa(\widehat{\mathcal{P}}_X, \widehat{\mathcal{P}}_\theta)$$

$$\sum_{\substack{\boldsymbol{x}_i \in X \\ \boldsymbol{x}_j \in X}} \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) - 2 \sum_{\substack{\boldsymbol{x}_i \in X \\ \boldsymbol{z}_j \in Z}} \kappa(\boldsymbol{x}_i, \mathcal{G}_\theta(\boldsymbol{z}_j)) + \sum_{\substack{\boldsymbol{z}_i \in Z \\ \boldsymbol{z}_j \in Z}} \kappa(\mathcal{G}_\theta(\boldsymbol{z}_i), \mathcal{G}_\theta(\boldsymbol{z}_j))$$

Similarity between samples

equivalent to (for later)

$$\sum_{\boldsymbol{z}_i \in Z} e^{\mathrm{i}\boldsymbol{\omega}^T \mathcal{G}_\theta(\boldsymbol{z}_i)} \Big|^2$$

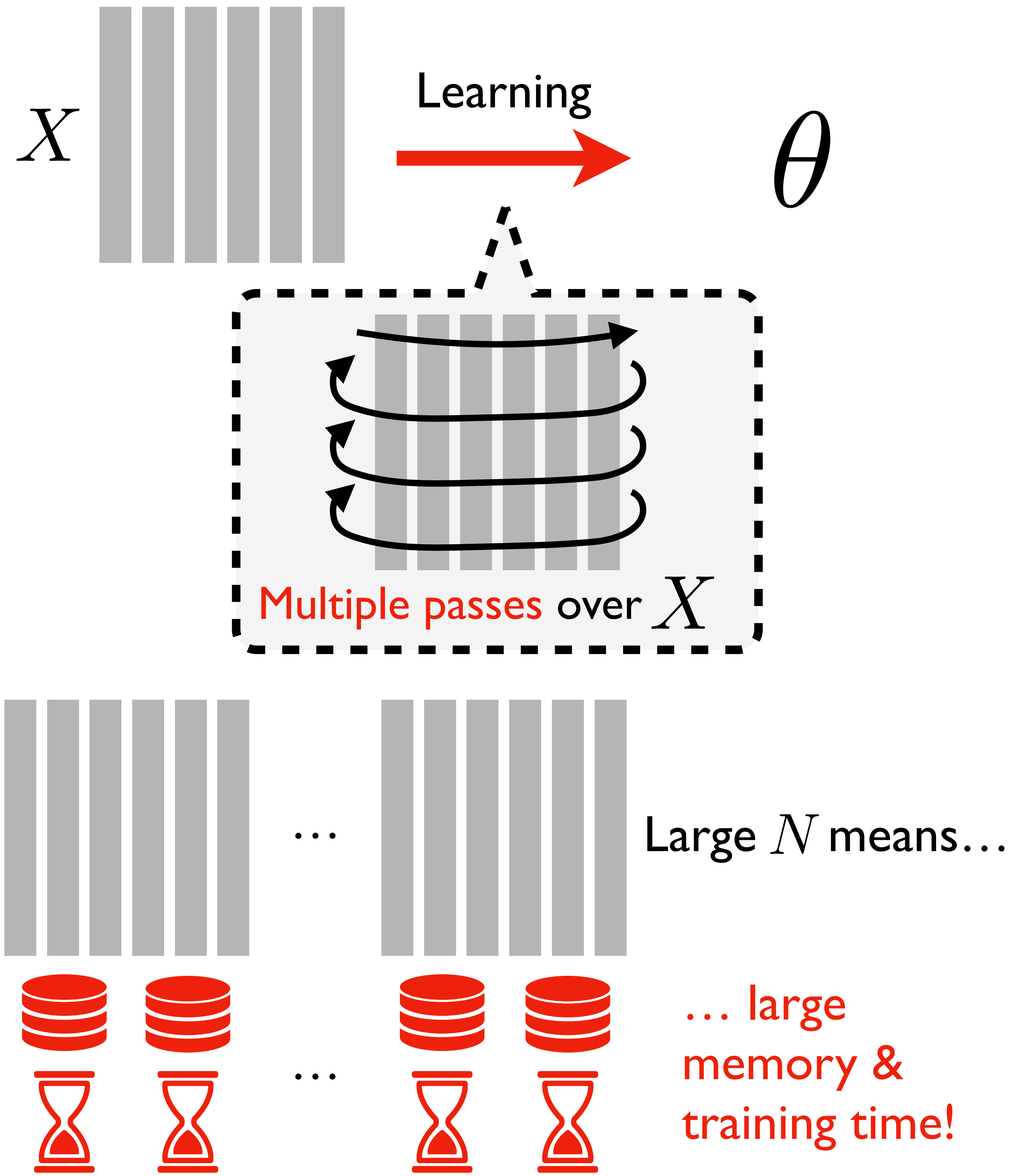
Training generative networks typically requires massive amounts of data!

Easier to train (no balancing) but quadratic complexity…

Compressive Learning to the rescue!

Compressive Learning: principle

Usual machine learning



X Learning θ

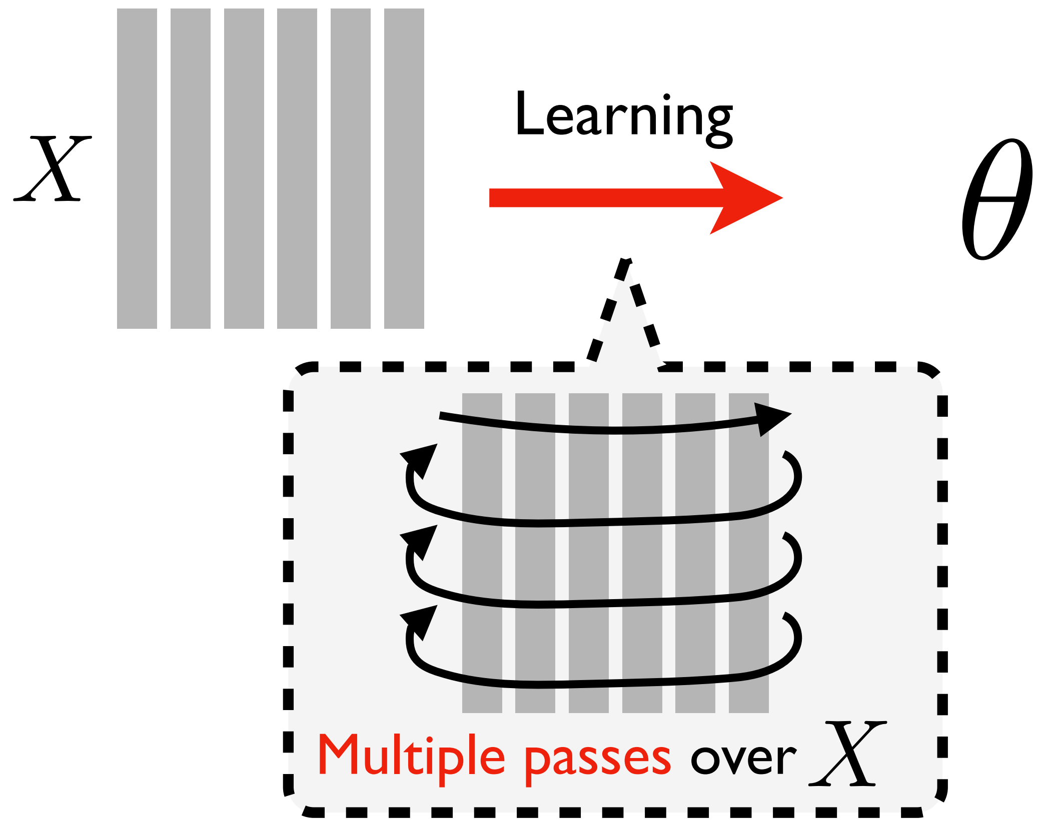
Multiple passes over X

Large N means…

… large memory & training time!

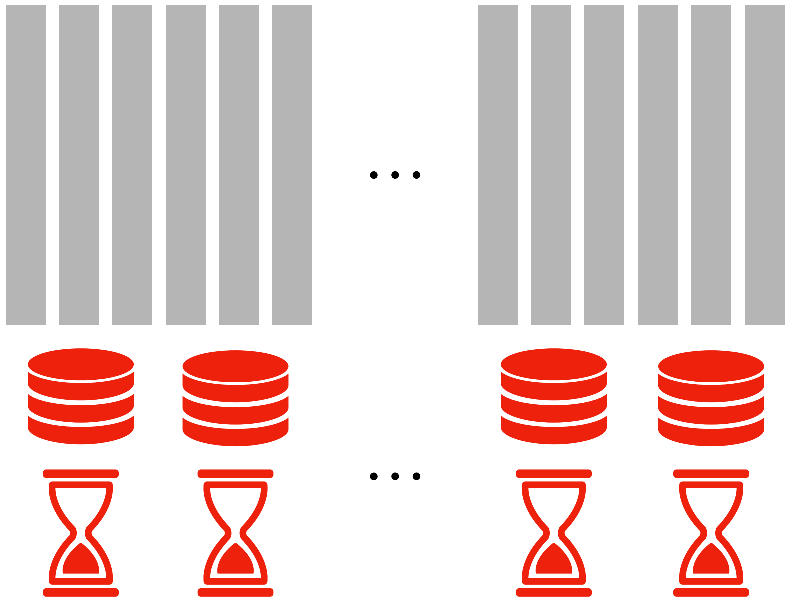
Compressive Learning: principle

Usual machine learning



X Learning θ

Multiple passes over X

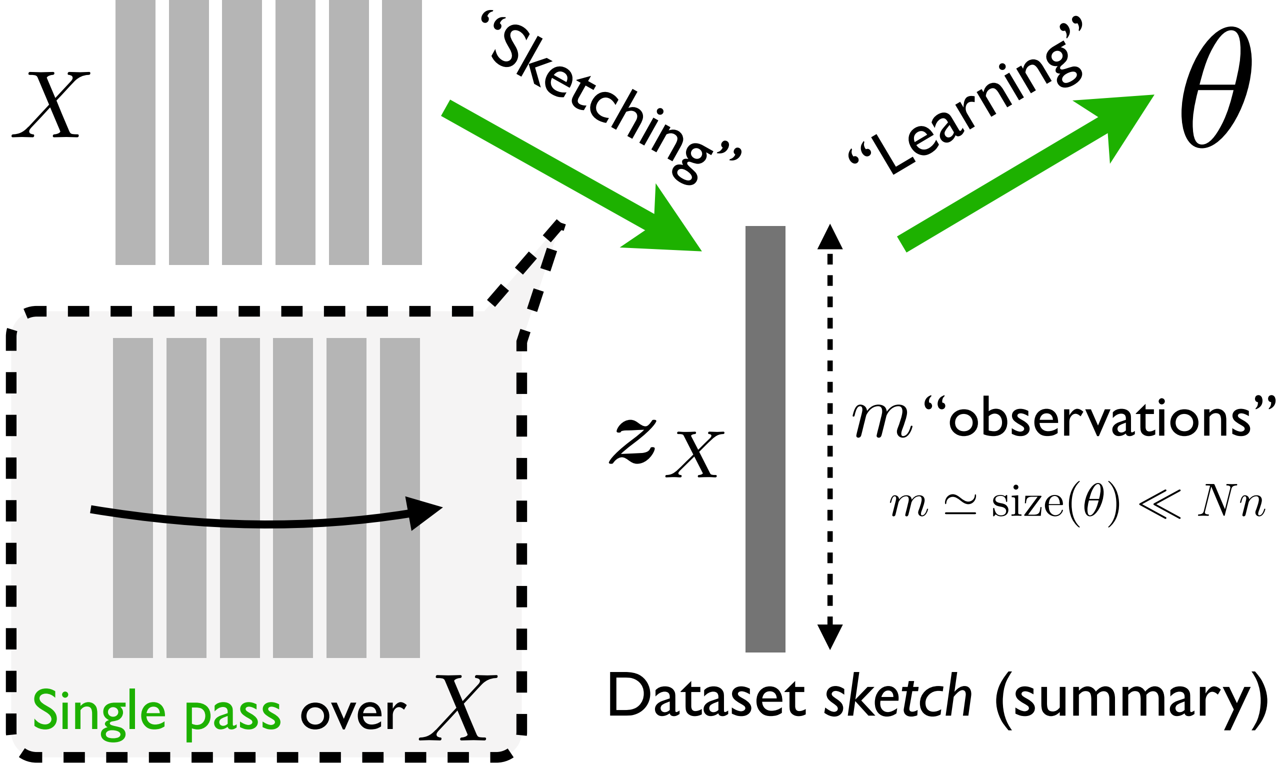


Large N means…

… large memory & training time!

Compressive Learning

R. Gribonval et al., "*Compressive Statistical Learning with Random Feature Moments,*" 2017



X "Sketching" "Learning" θ

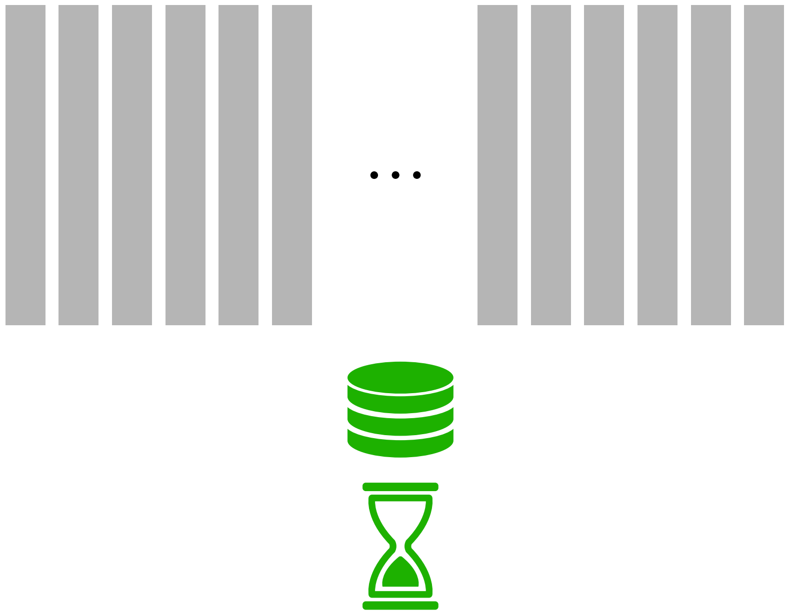
Single pass over X

z_X

m "observations"

$m \simeq \text{size}(\theta) \ll Nn$

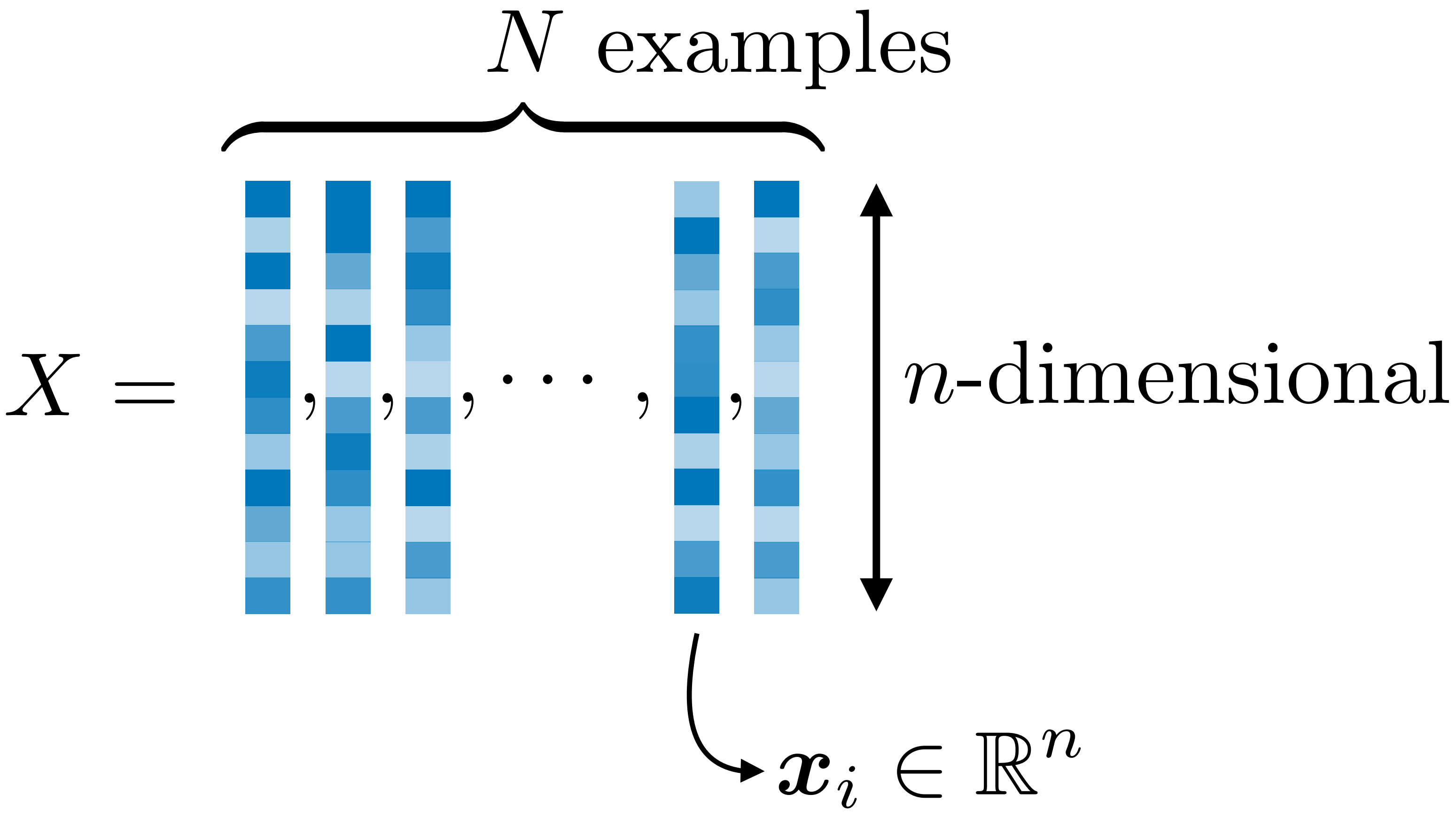
Dataset *sketch* (summary)



Large N means…

… *constant* memory & training time!

Sketching a dataset



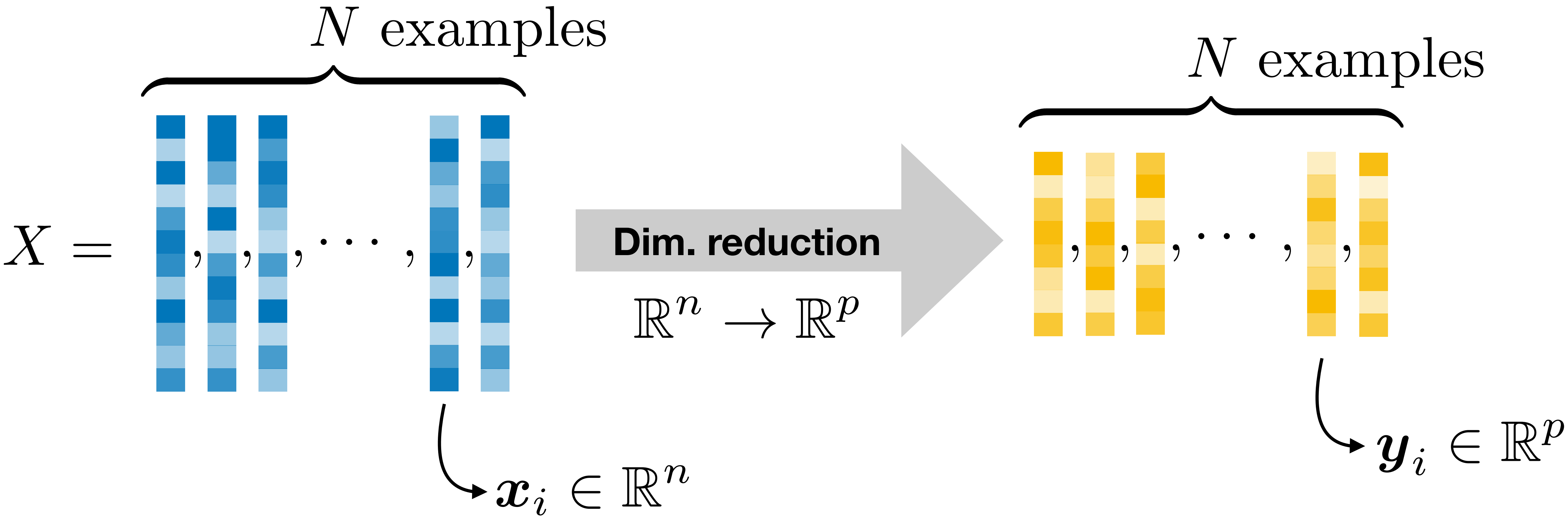
N examples

$X = \ , \ , \ , \cdots , \ , $

n-dimensional

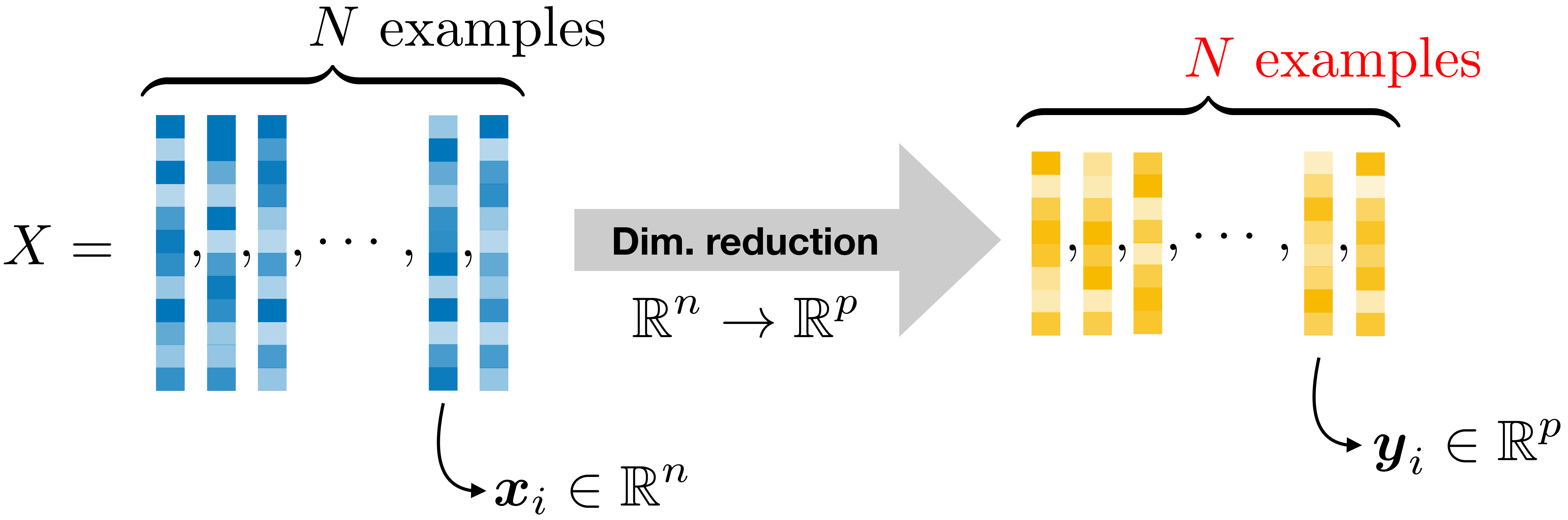
$\boldsymbol{x}_i \in \mathbb{R}^n$

Sketching a dataset



- **Compressed representation** ✓
- **Preserves relevant information** ✓

Sketching a dataset



N examples

$X = $ $, \dots ,$

Dim. reduction

$\mathbb{R}^n \to \mathbb{R}^p$

N examples

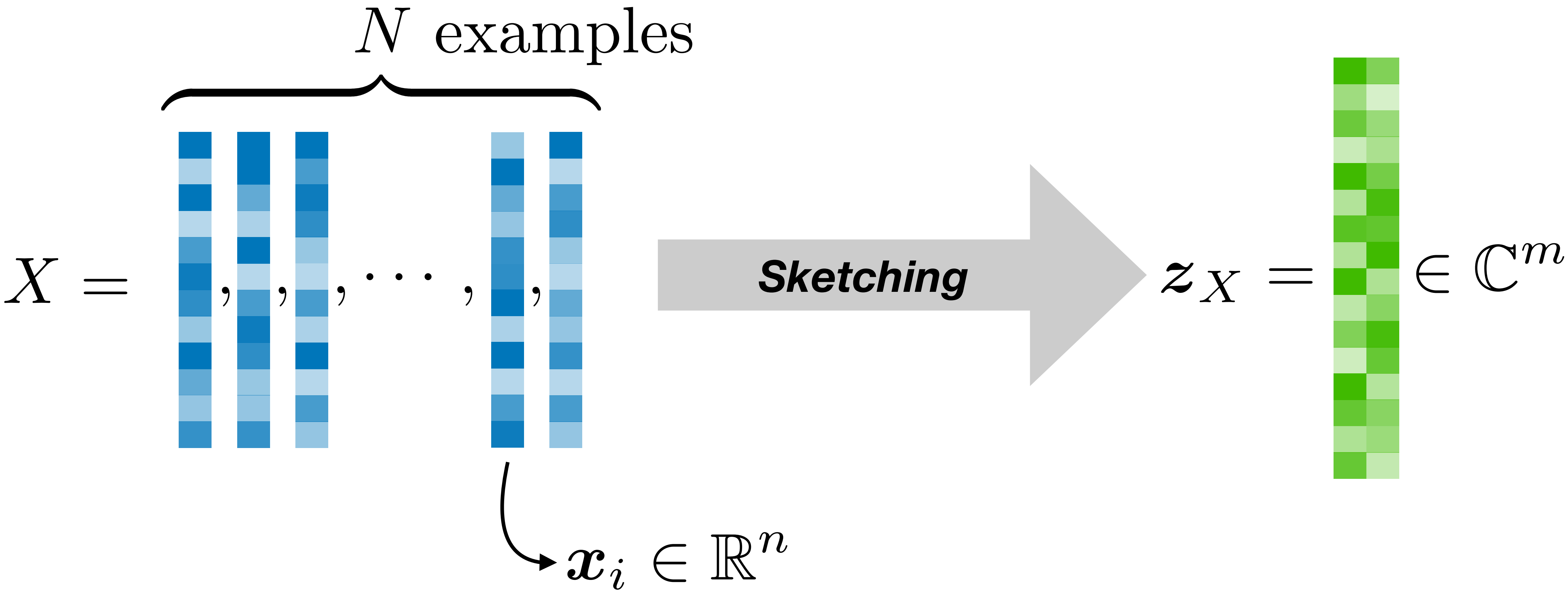
$\boldsymbol{x}_i \in \mathbb{R}^n$

$\boldsymbol{y}_i \in \mathbb{R}^p$

- **Compressed representation ✓**
- **Preserves relevant information ✓**
- **Constant number of examples ✗**

N **can be VERY large (“big data”)!**

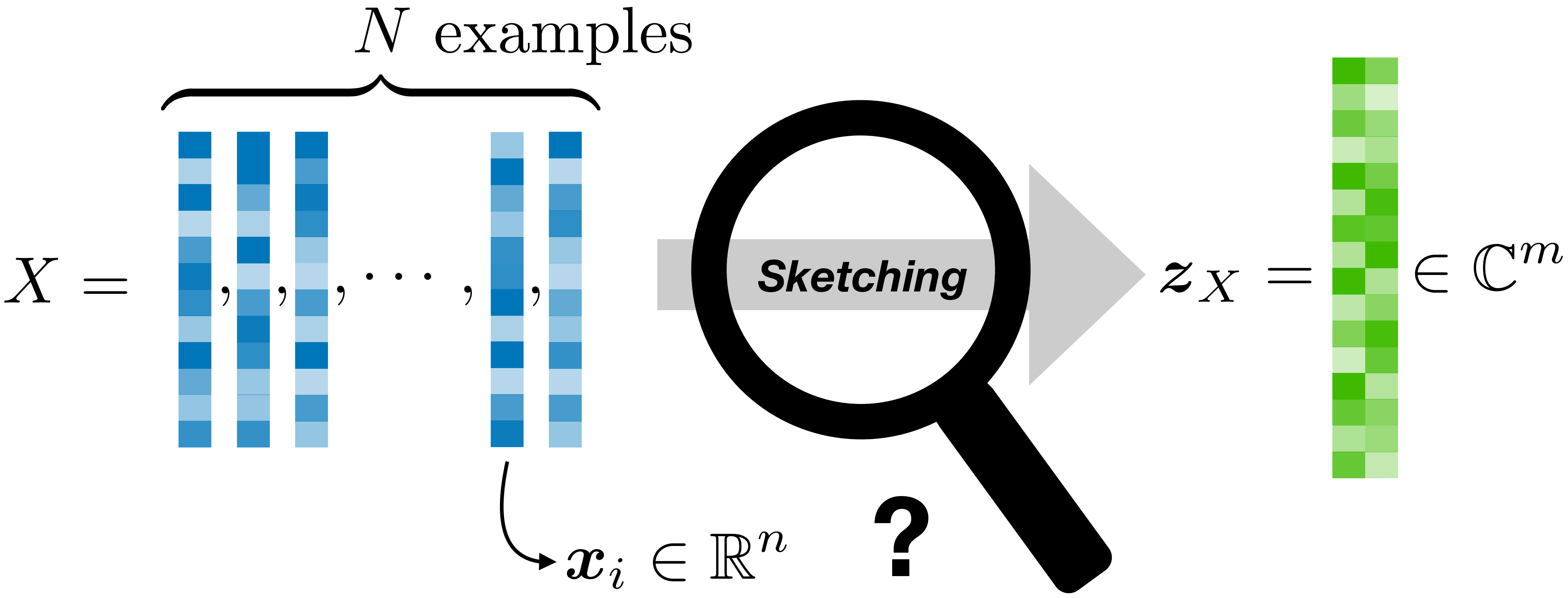
Sketching a dataset



- **Compressed representation** ✓
- **Preserves relevant information** ✓
- **Dataset summary = single vector** ✓

[Gribonval17]

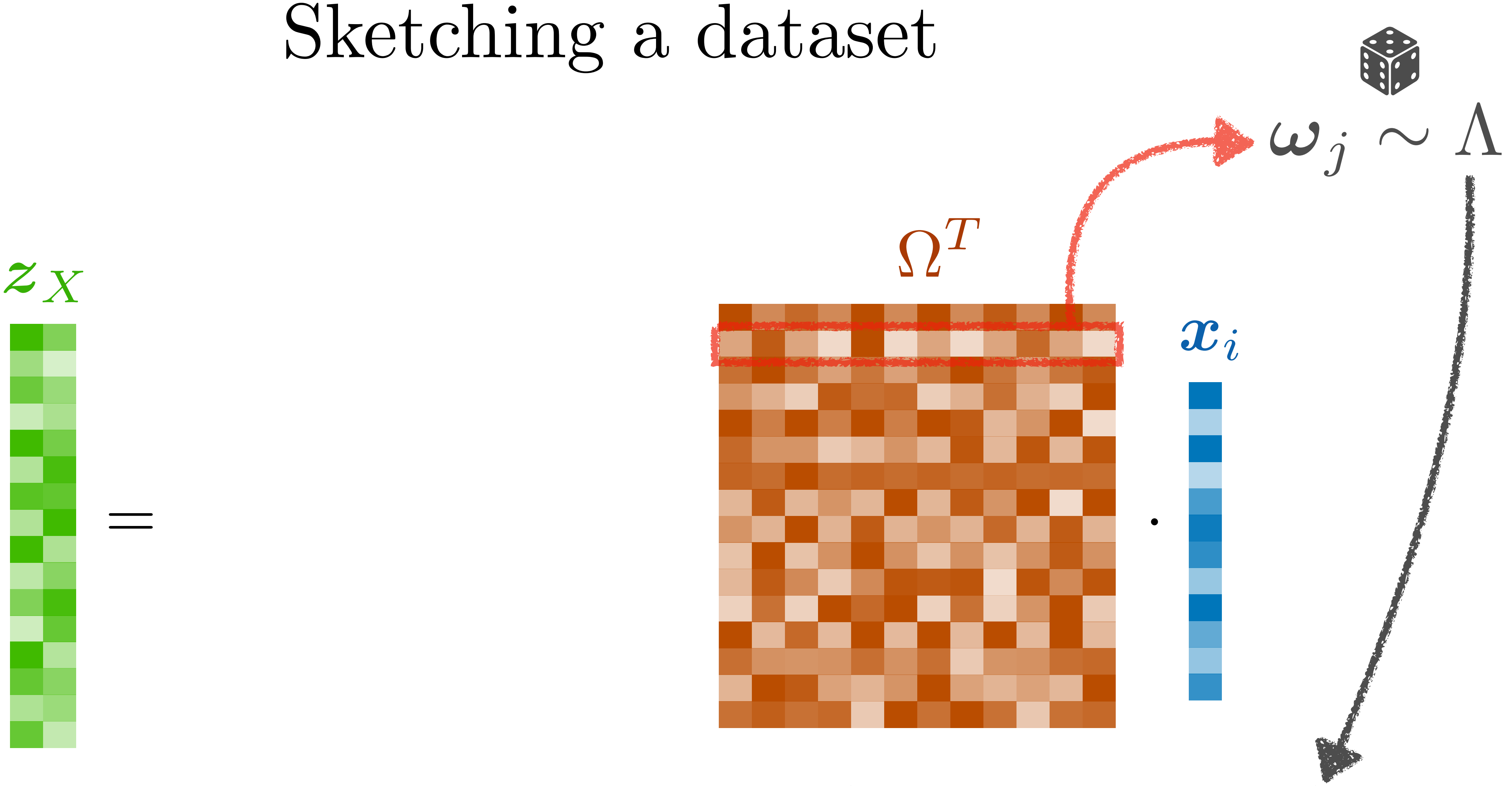
Sketching a dataset



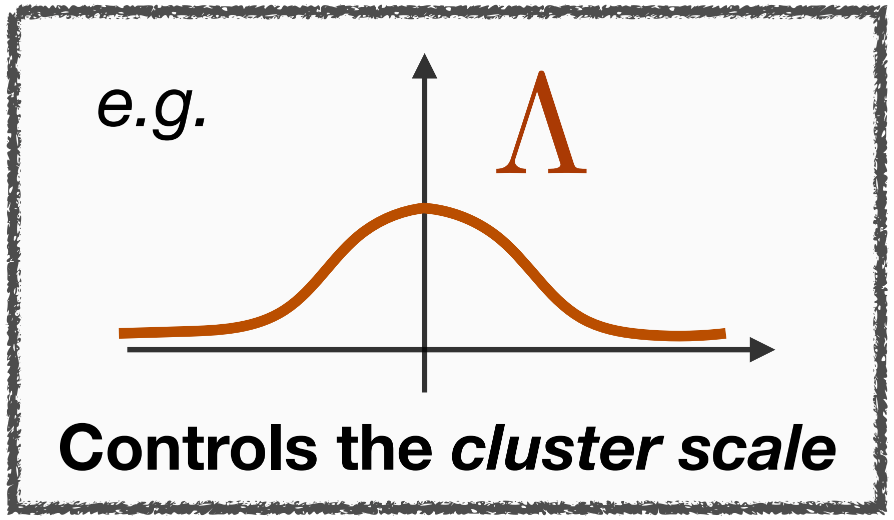
- **Compressed representation** ✓
- **Preserves relevant information** ✓
- **Dataset summary = single vector** ✓

[Gribonval17]

Sketching a dataset

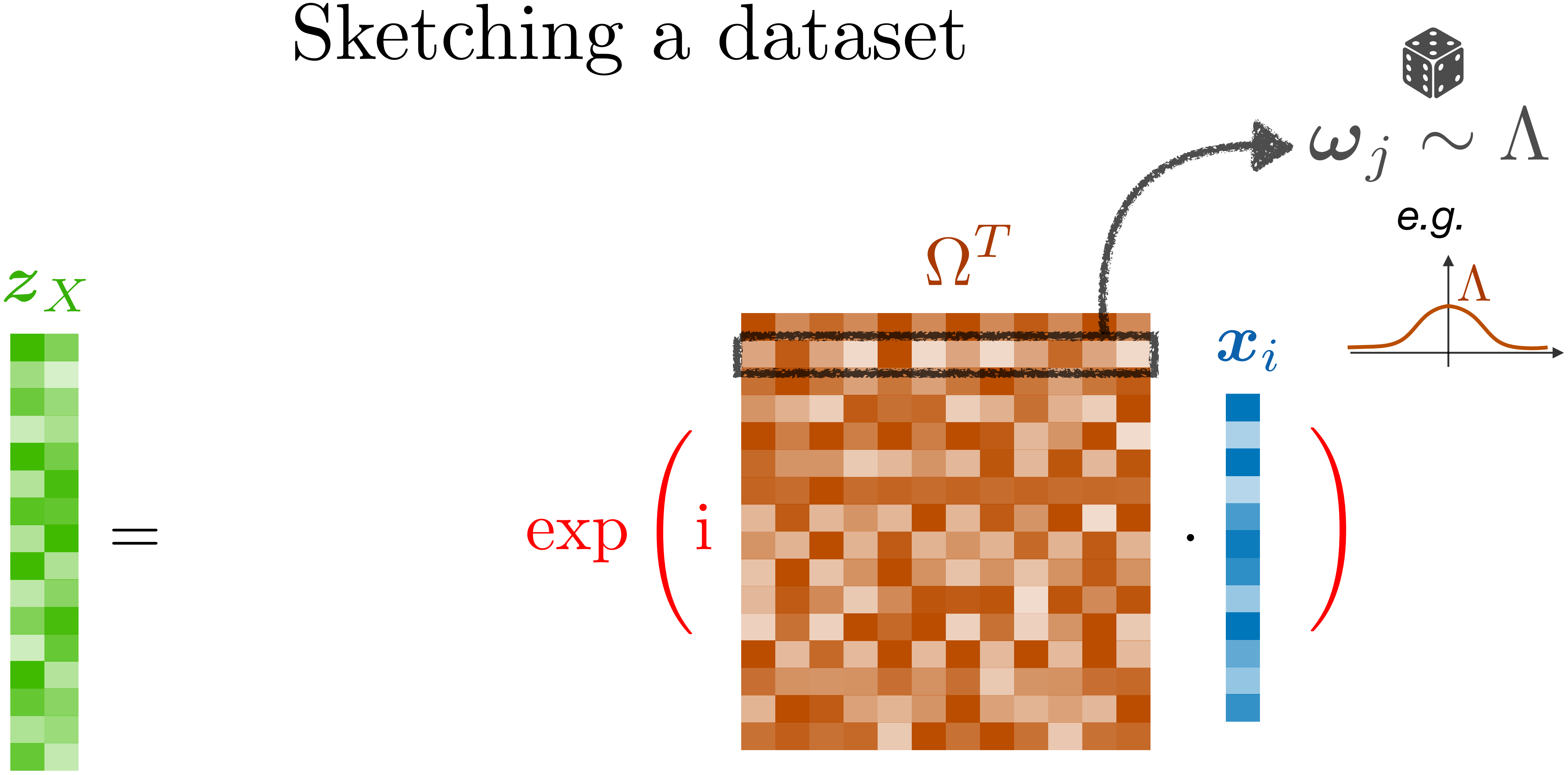


1. Project on m (random) vectors



Controls the *cluster scale*

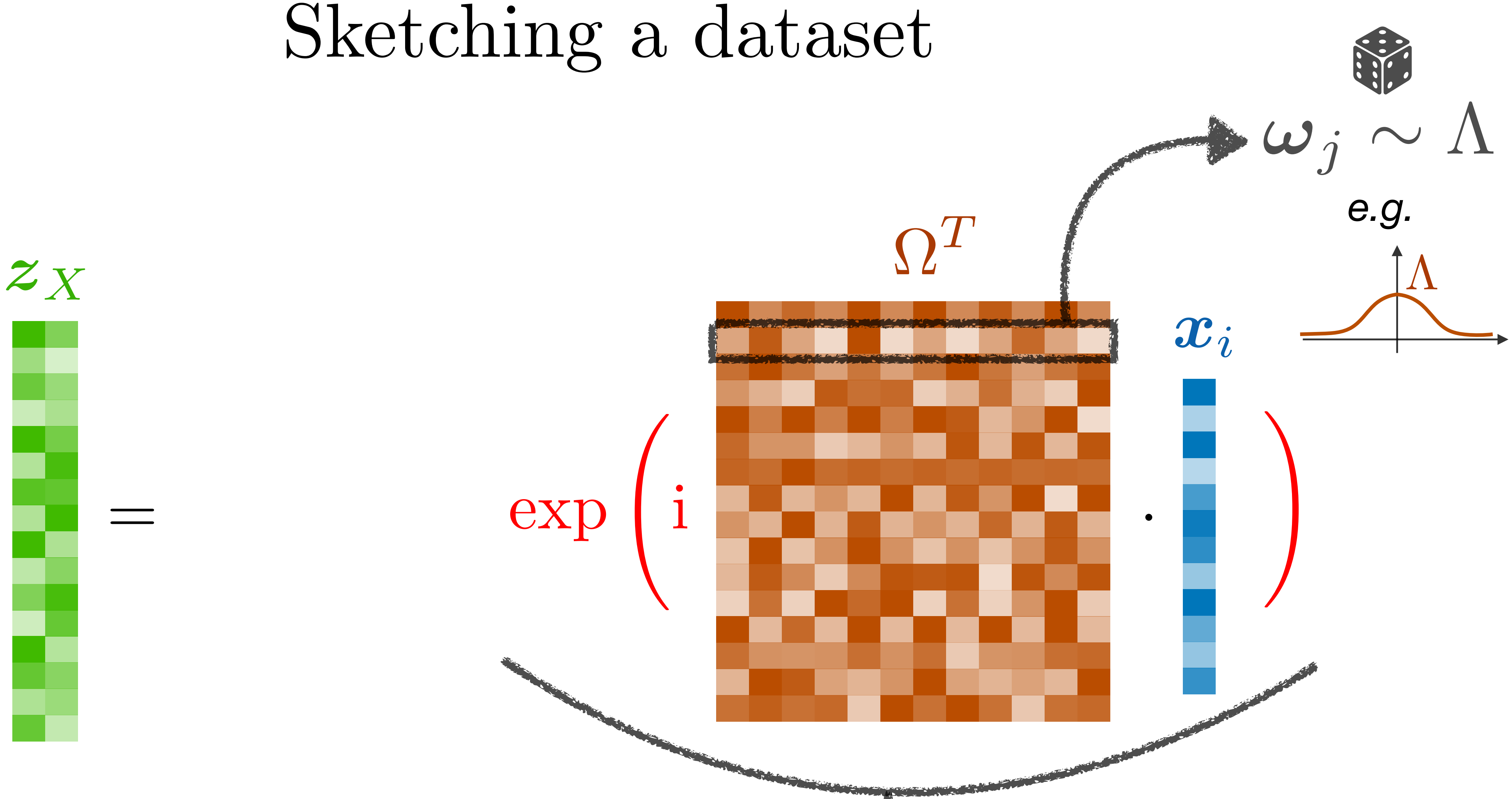
Sketching a dataset



1. Project on m (random) vectors

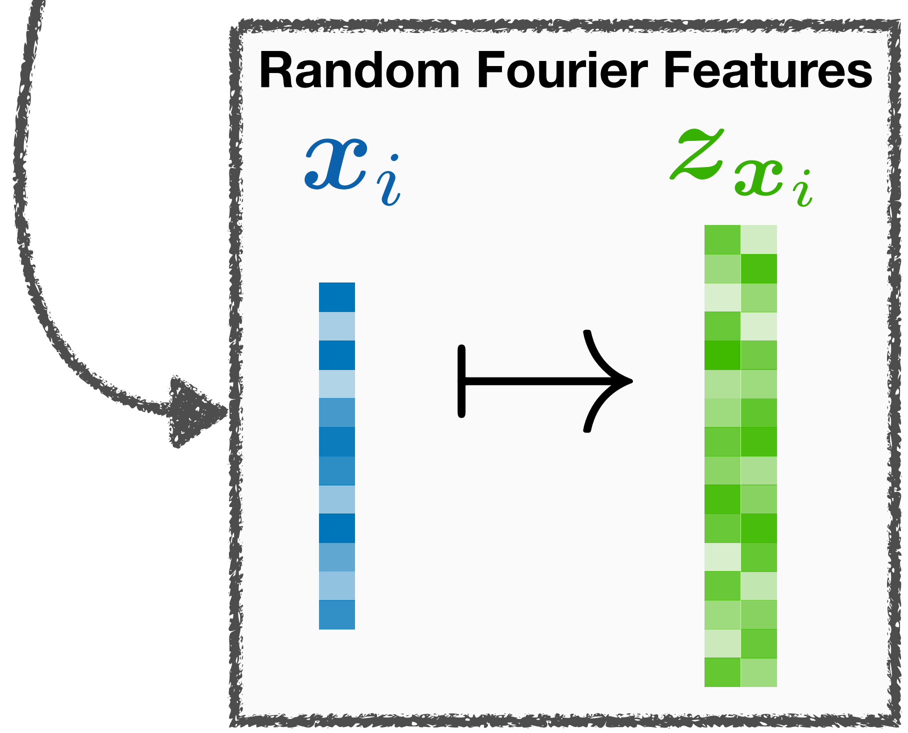
2. Nonlinear periodic signature function

Sketching a dataset



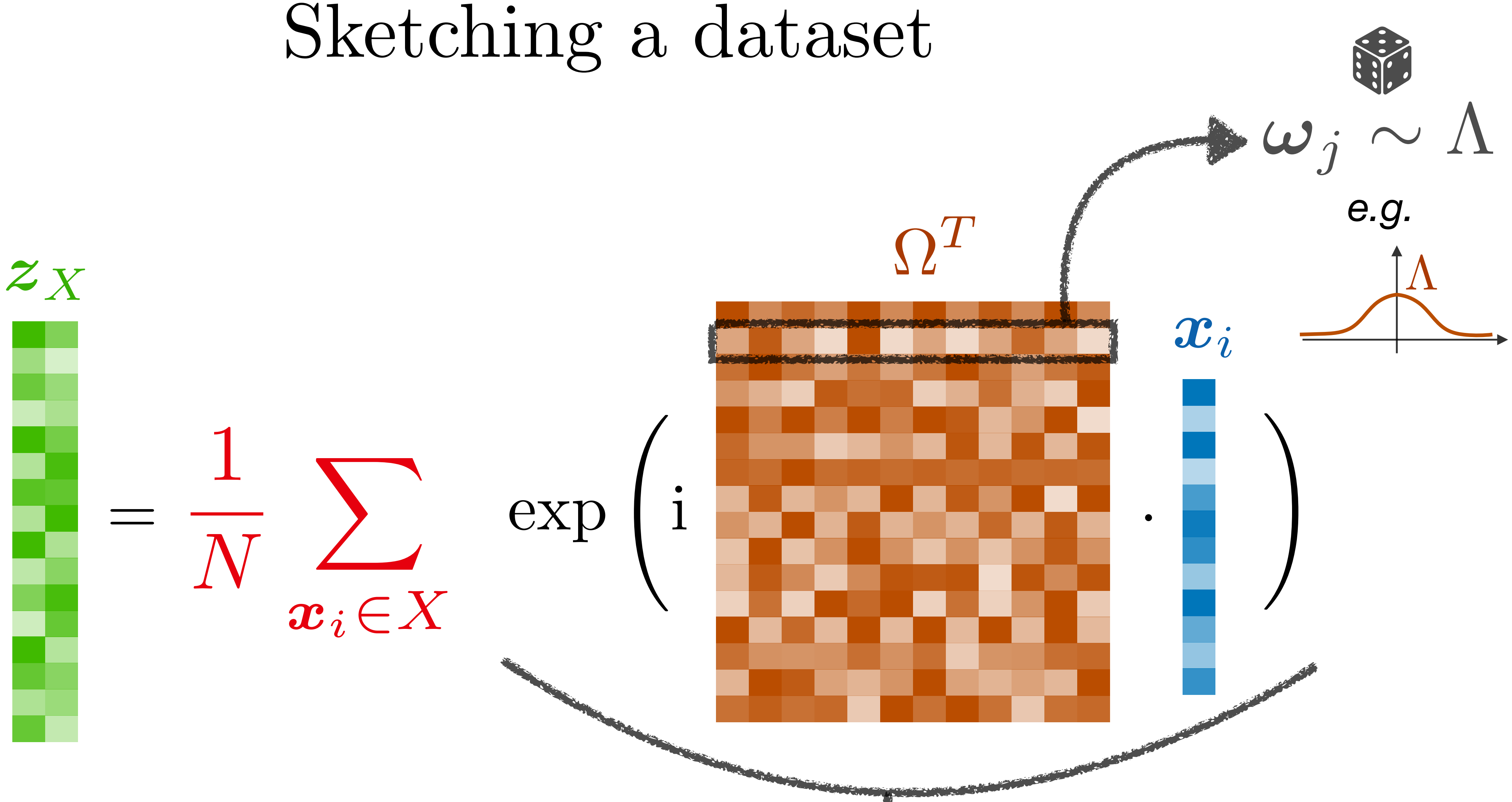
1. Project on m (random) vectors

2. Nonlinear periodic signature function



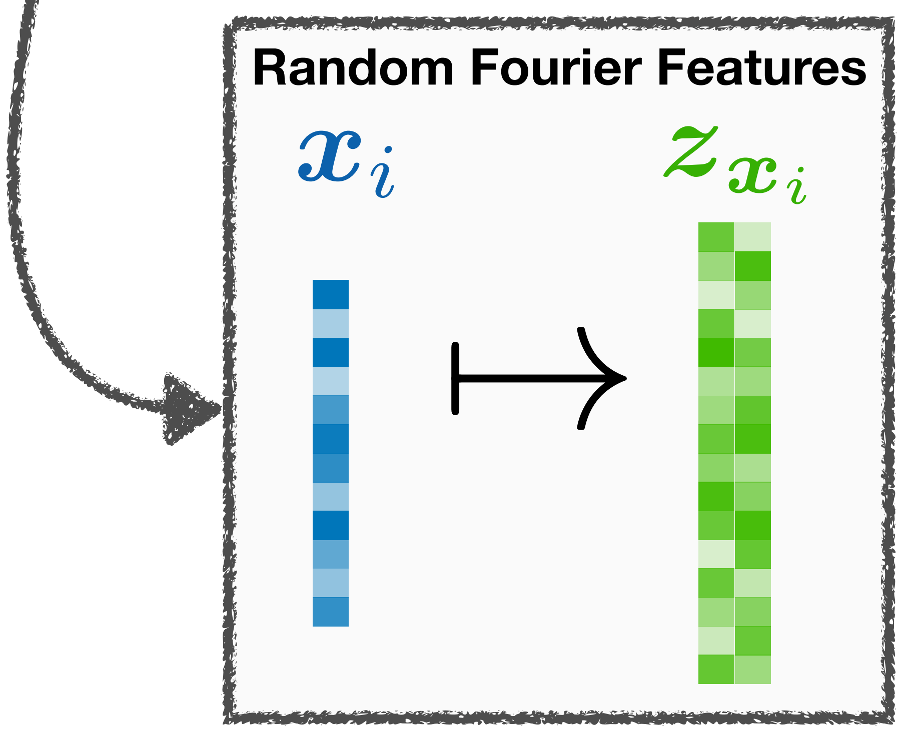
[A. Rahimi, B. Recht, "*Random Features for Large-scale Kernel Machines*",NIPS, 2007]

Sketching a dataset



$$\boldsymbol{z}_X = \frac{1}{N}\sum_{\boldsymbol{x}_i\in X}\exp\left(\mathrm{i}\;\Omega^T\cdot\boldsymbol{x}_i\right)$$

1. **Project on m (random) vectors**
2. **Nonlinear periodic signature function**
3. **Pooling (average)**

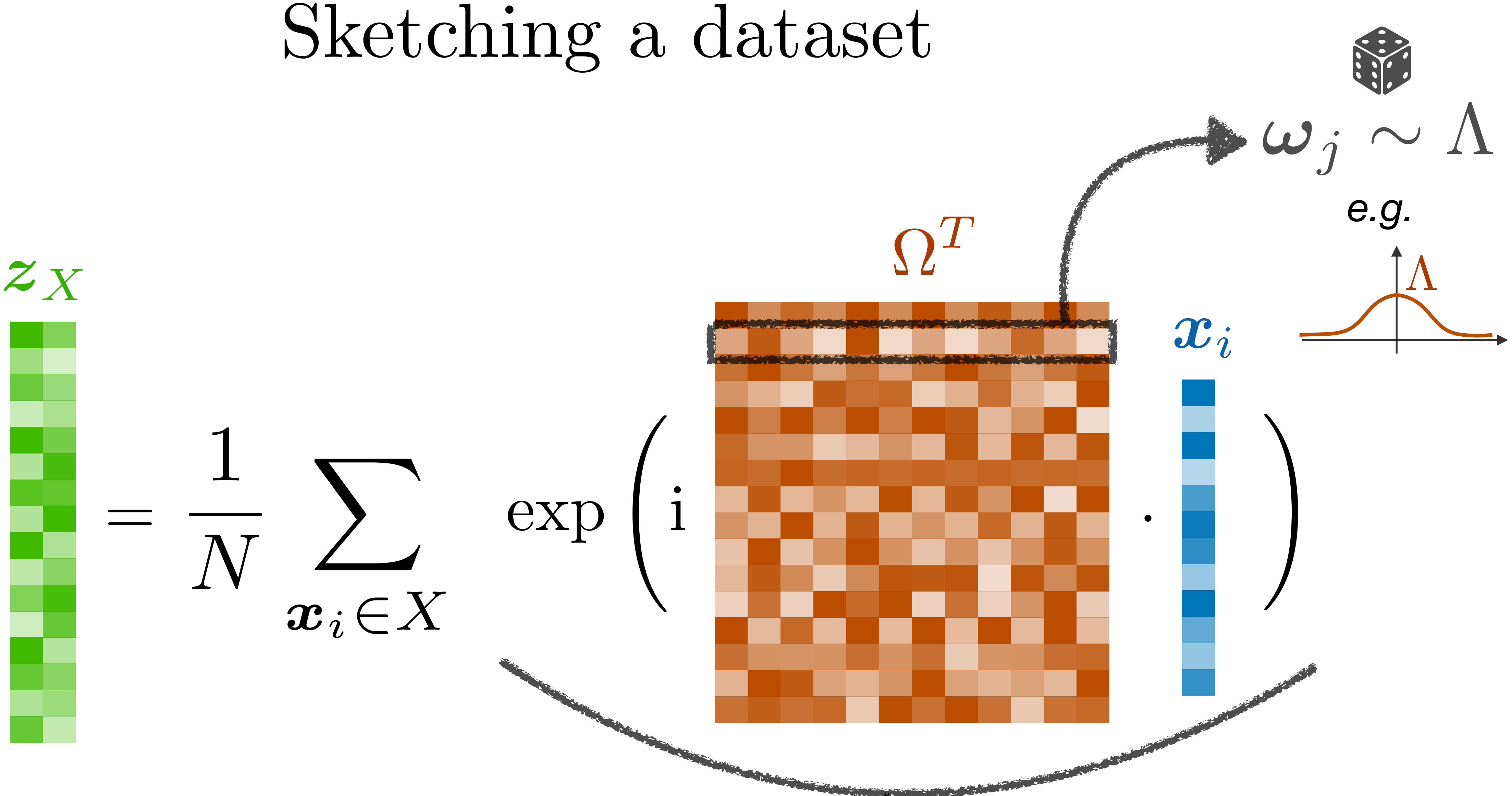


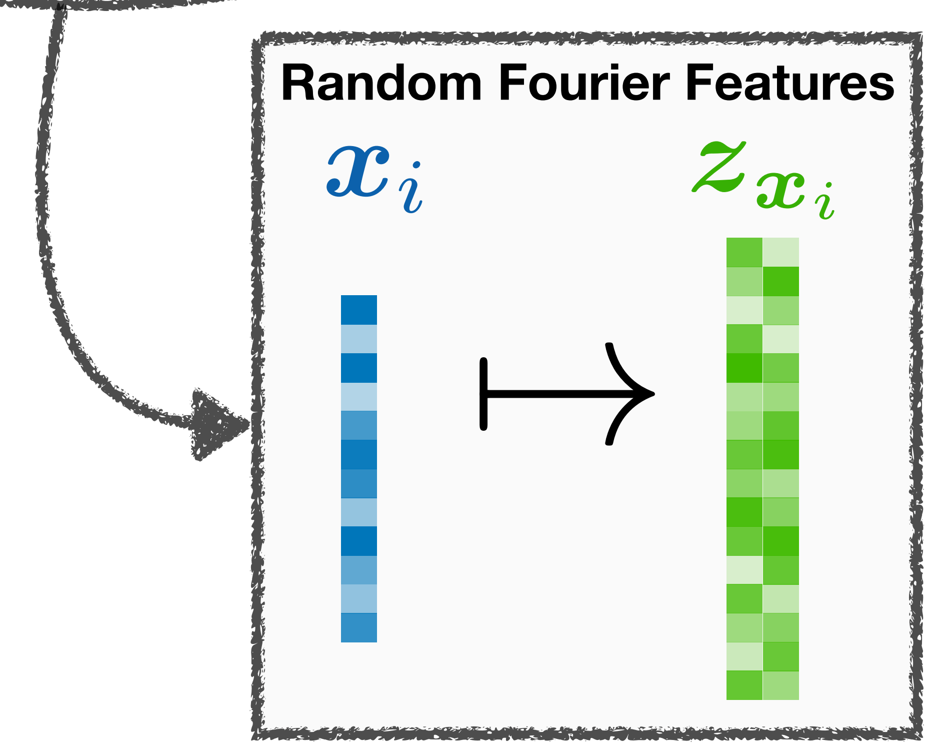
Random Fourier Features

$$\boldsymbol{x}_i \mapsto \boldsymbol{z}_{\boldsymbol{x}_i}$$

[A. Rahimi, B. Recht, "*Random Features for Large-scale Kernel Machines*",NIPS, 2007]

Sketching a dataset





1. Project on m (random) vectors

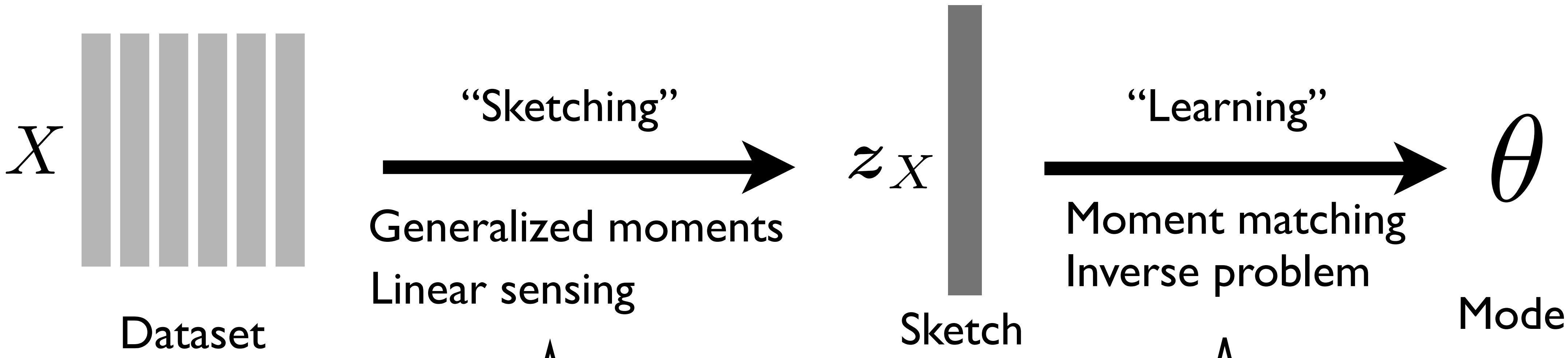
2. Nonlinear periodic signature function

3. Pooling (average)

$$\boldsymbol{z}_X = \left[\frac{1}{N} \sum_{\boldsymbol{x}_i \in X} e^{\mathrm{i}\boldsymbol{\omega}_j^T \boldsymbol{x}_i} \right]_{j=1}^{m} \in \mathbb{C}^m$$

[A. Rahimi, B. Recht, “*Random Features for Large-scale Kernel Machines*”,NIPS, 2007]

Compressive Learning: SoA in one slide



X Dataset

"Sketching"

Generalized moments

Linear sensing

z_X Sketch

"Learning"

Moment matching

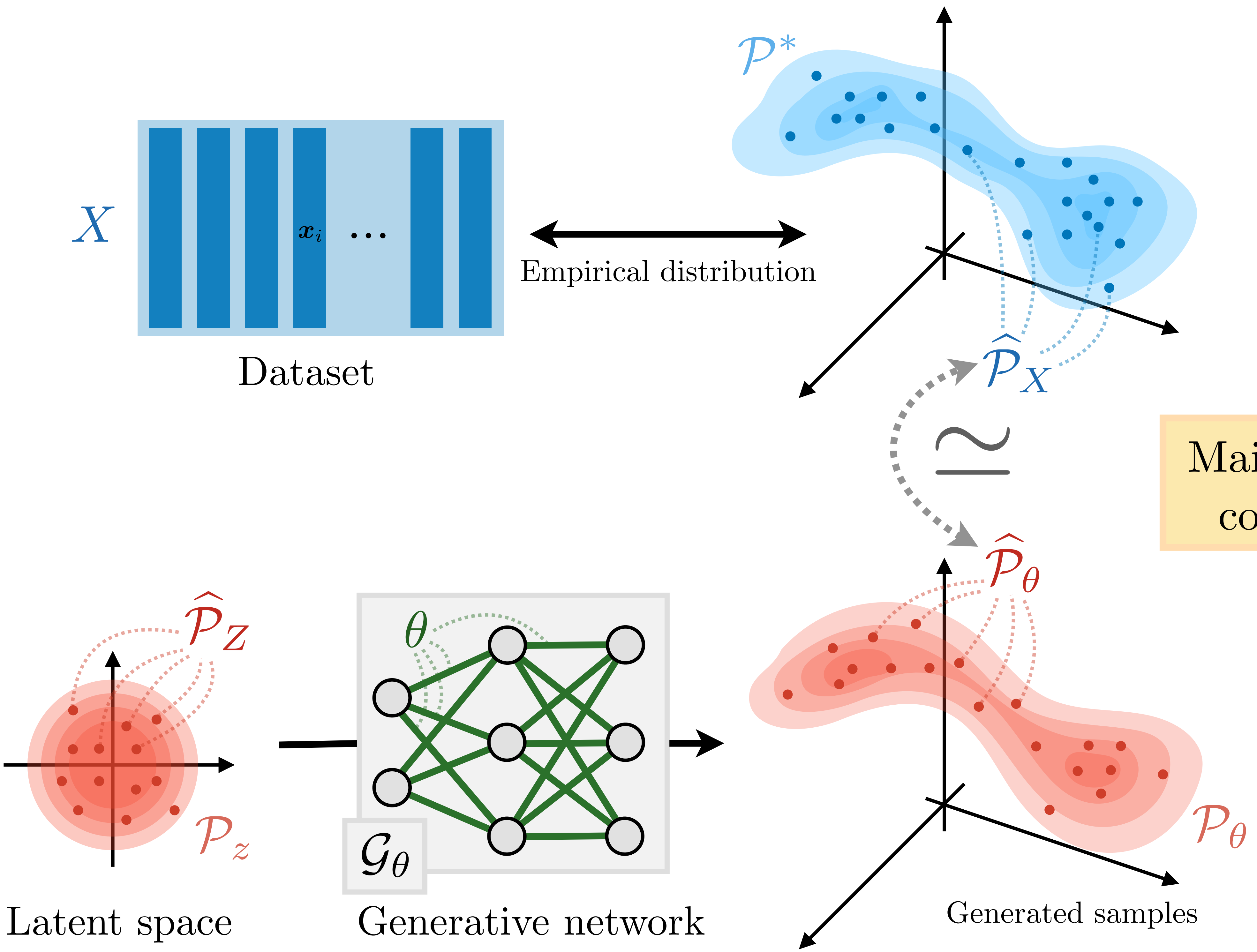
Inverse problem

θ Model

Advantages:
- One single pass on dataset (parallelizable)
- Harsh compression (ideal for large-scale datasets)
- Handy for privacy preservation

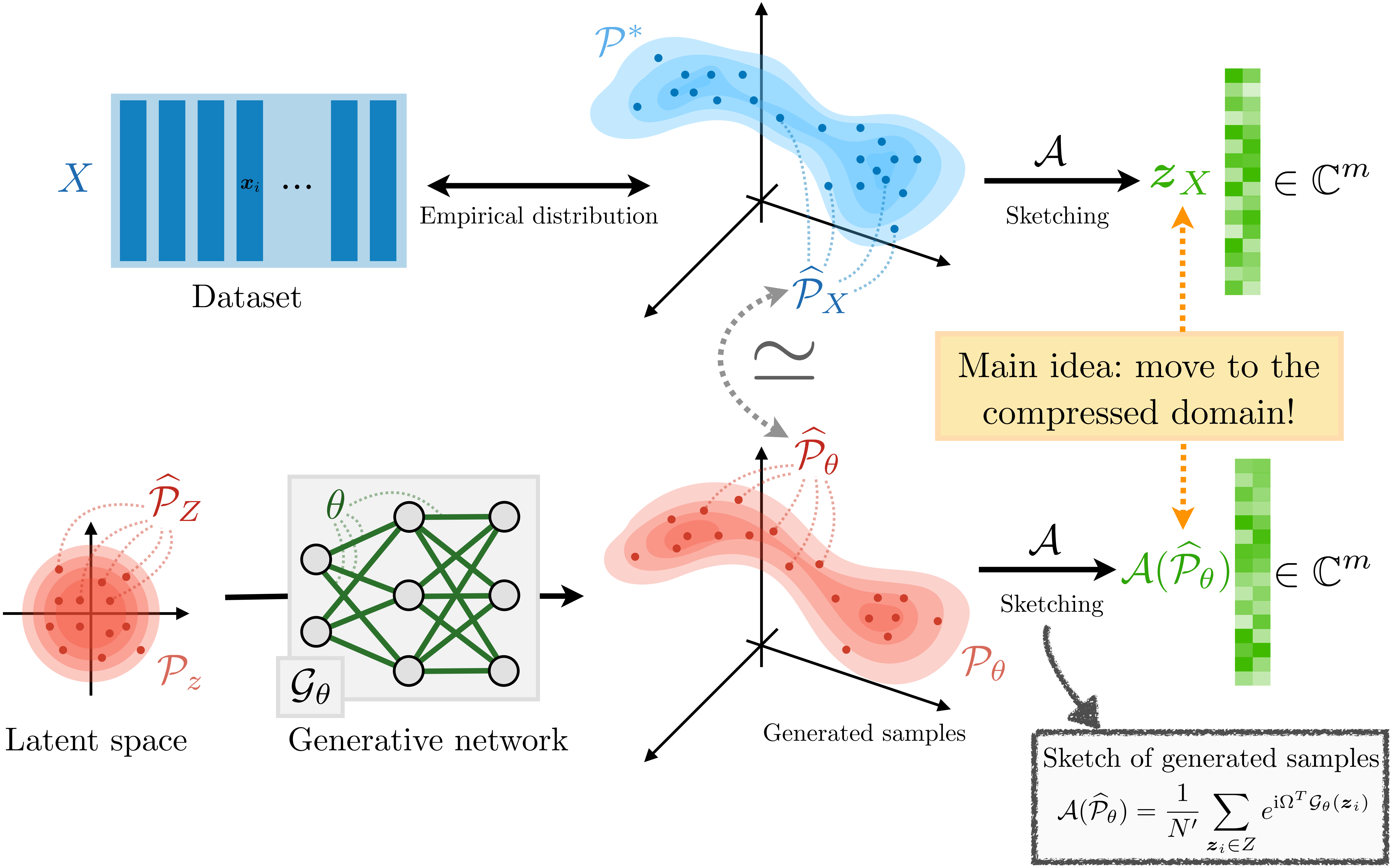
Existing (unsupervised) tasks:
- clustering: k-means, subspace clustering
- mixture model estimation: GMM, alpha-stable distributions
- Principal Component Analysis
- Independent Component Analysis
- Generative networks (this work)

Compressively learning generative networks



Main idea: move to the compressed domain!

Compressively learning generative networks

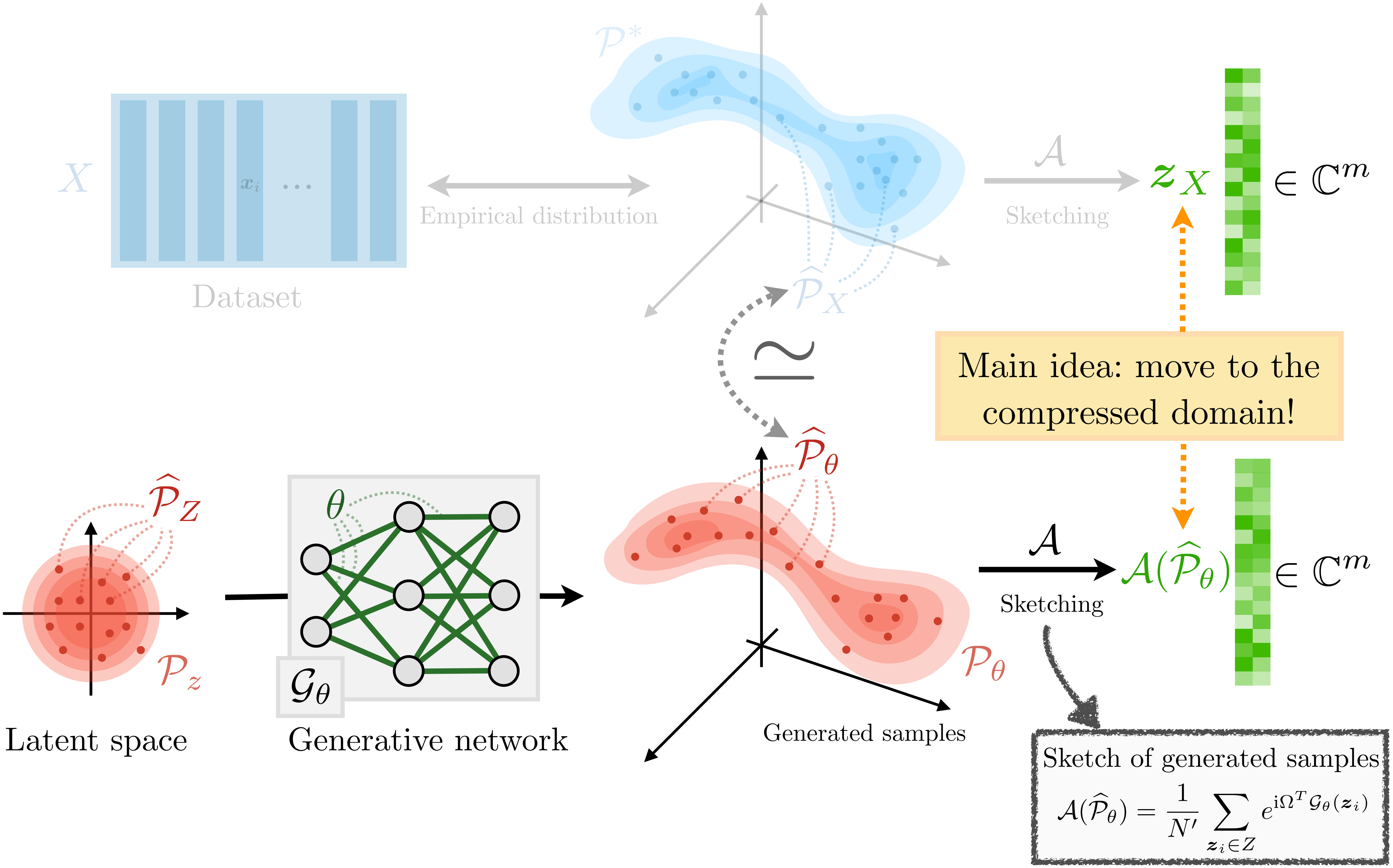


Main idea: move to the compressed domain!

Sketch of generated samples

$$\mathcal{A}(\widehat{\mathcal{P}}_\theta) = \frac{1}{N'} \sum_{\boldsymbol{z}_i \in Z} e^{\mathrm{i}\Omega^T \mathcal{G}_\theta(\boldsymbol{z}_i)}$$

Compressively learning generative networks



Main idea: move to the compressed domain!

Sketch of generated samples

$$\mathcal{A}(\widehat{\mathcal{P}}_\theta) = \frac{1}{N'} \sum_{\boldsymbol{z}_i \in Z} e^{\mathrm{i}\Omega^T \mathcal{G}_\theta(\boldsymbol{z}_i)}$$

Compressively learning generative networks

Proposed approach: match the sketches of real and generated data

$$\min_{\theta} \left\| \boldsymbol{z}_X - \frac{1}{N'} \sum_{\boldsymbol{z}_i \in Z} e^{\mathrm{i}\Omega^T \mathcal{G}_\theta(\boldsymbol{z}_i)} \right\|_2^2$$

Compressively learning generative networks

Proposed approach: match the sketches of real and generated data

$$\min_{\theta} \left\| \boldsymbol{z}_X - \frac{1}{N'} \sum_{\boldsymbol{z}_i \in Z} e^{\mathrm{i}\Omega^T \mathcal{G}_\theta(\boldsymbol{z}_i)} \right\|_2^2$$

$\simeq$

Sampled (Monte Carlo)
estimation to the MMD!

$\boldsymbol{\omega}_j \sim \Lambda$

$$\min_{\theta} \mathbb{E}_{\boldsymbol{\omega} \sim \Lambda} \left| \frac{1}{N} \sum_{\boldsymbol{x}_i \in X} e^{\mathrm{i}\boldsymbol{\omega}^T \boldsymbol{x}_i} - \frac{1}{N'} \sum_{\boldsymbol{z}_i \in Z} e^{\mathrm{i}\boldsymbol{\omega}^T \mathcal{G}_\theta(\boldsymbol{z}_i)} \right|^2$$

...but where dataset is accessed only once 👍

Compressively learning generative networks

Proposed approach: match the sketches of real and generated data

$$\min_{\theta} \left\| \boldsymbol{z}_X - \frac{1}{N'} \sum_{\boldsymbol{z}_i \in Z} e^{\mathrm{i}\Omega^T \mathcal{G}_\theta(\boldsymbol{z}_i)} \right\|_2^2$$

$\simeq$

Sampled (Monte Carlo) estimation to the MMD!

$\boldsymbol{\omega}_j \sim \Lambda$

$$\min_{\theta} \mathbb{E}_{\boldsymbol{\omega} \sim \Lambda} \left| \frac{1}{N} \sum_{\boldsymbol{x}_i \in X} e^{\mathrm{i}\boldsymbol{\omega}^T \boldsymbol{x}_i} - \frac{1}{N'} \sum_{\boldsymbol{z}_i \in Z} e^{\mathrm{i}\boldsymbol{\omega}^T \mathcal{G}_\theta(\boldsymbol{z}_i)} \right|^2$$

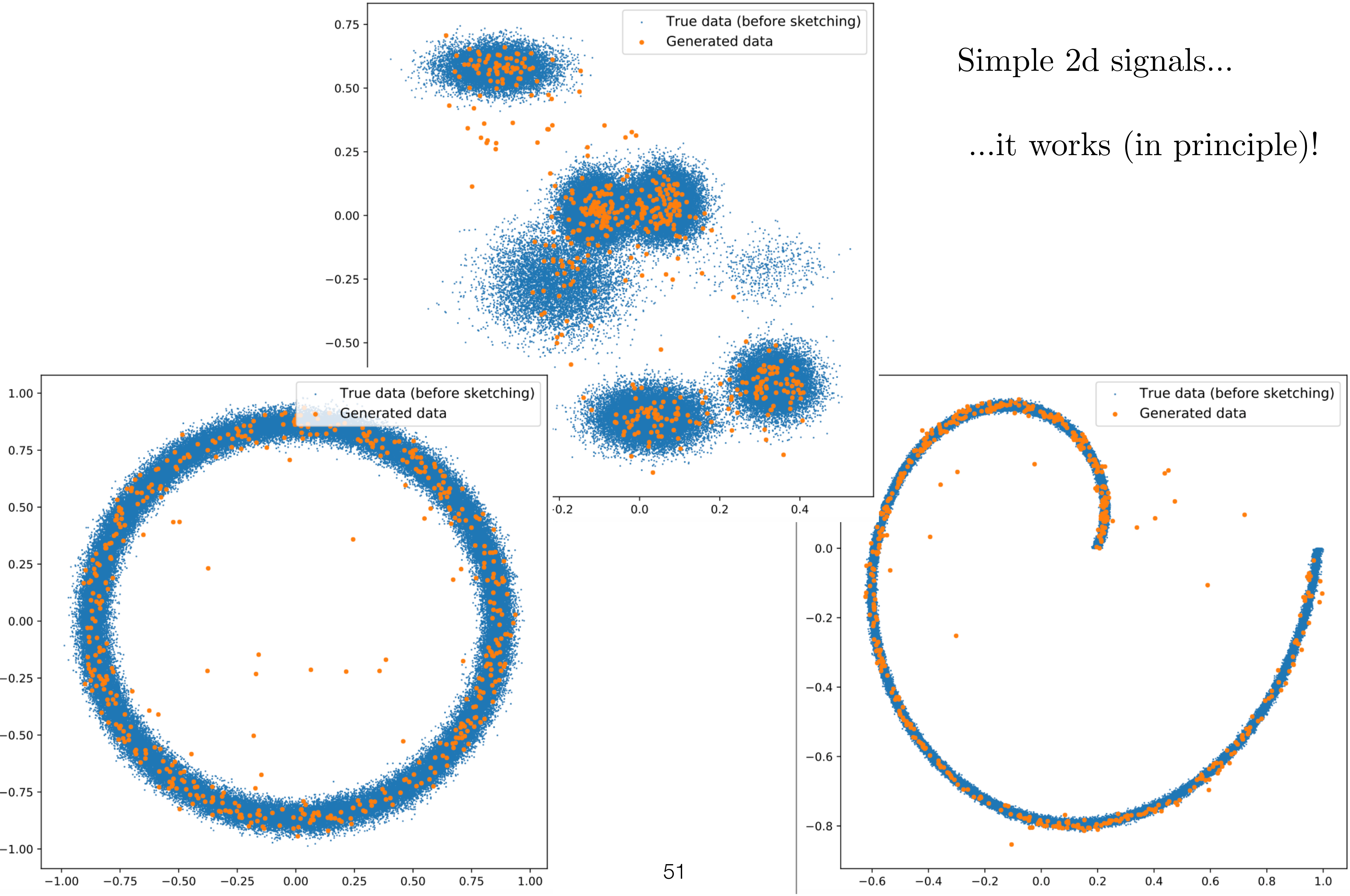
...but where dataset is accessed only once 👍

Practical learning algorithm?

Differentiable by chain rule (feat. backprop) 👍

$$\nabla_\theta \widehat{\mathcal{L}}(\theta; \boldsymbol{z}_X) = -2 \cdot \tfrac{1}{n'} \textstyle\sum_{i=1}^{n'} \Re \left[\boldsymbol{r}^H \left(\frac{\partial \Phi(\boldsymbol{u})}{\partial \boldsymbol{u}} \Big|_{\boldsymbol{u} = \mathcal{G}_\theta(\boldsymbol{z}_i)} \cdot \frac{\partial \mathcal{G}_\theta(\boldsymbol{z}_i)}{\partial \theta} \right) \right]$$

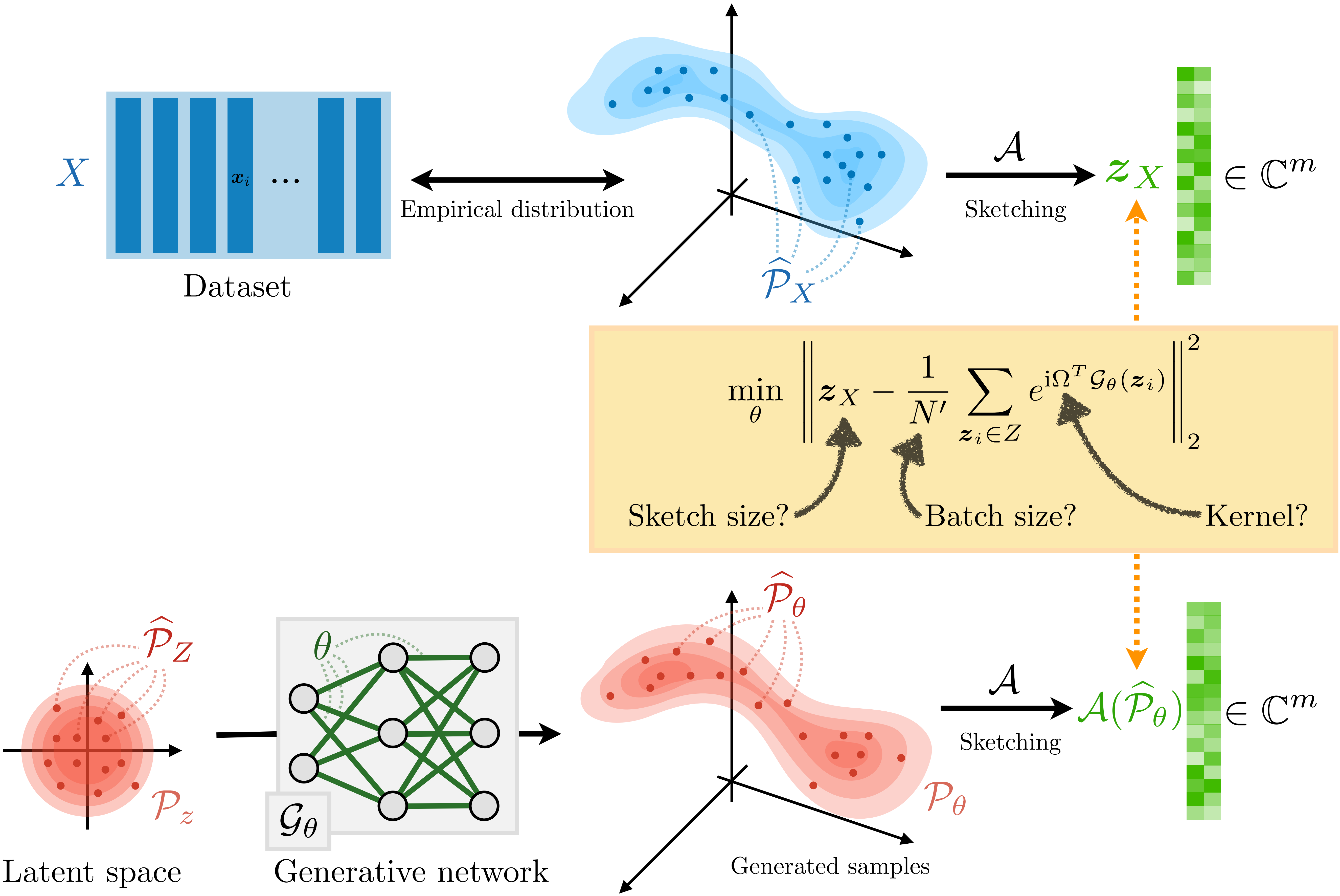
Preliminary results



Simple 2d signals...

...it works (in principle)!

To conclude: open challenges



$$\min_{\theta} \left\| \boldsymbol{z}_X - \frac{1}{N'} \sum_{\boldsymbol{z}_i \in Z} e^{\mathrm{i}\Omega^T \mathcal{G}_\theta(\boldsymbol{z}_i)} \right\|_2^2$$

Sketch size? Batch size? Kernel?