Introduction to Machine Learning: Generative Models

Farzad Kamalabadi

University of Illinois Urbana-Champaign

Heliophysics Summer School, 2025

Outline of this lecture

- Transition from supervised to unsupervised learning
- Variational Autoencoders (VAE)
- Generative Adversarial Networks (GAN)
- Diffusion-based Generative Models

Reading Material

-

Goals of the first part

- Getting to know Variational Auto-Encoders
- Understanding generative methods
- Differentiating between discriminative and generative methods

Reading material:

- D.P. Kingma and M. Welling; Auto-Encoding Variational Bayes; arxiv.org/abs/1312.6114
- C. Doersch; Tutorial on Variational Autoencoders; arxiv.org/abs/1606.05908

Recap: Supervised learning

Given a dataset $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}$ of data-label pairs, construct a mapping $F(w, x, y)$.

Examples:

- Linear regression
- Logistic regression
- Support vector machine
- Deep net

What if we don't have labels?

Without labels, we can still find structure in unlabeled data

$$\mathcal{D} = \{(x^{(i)})\}$$

Goal of unsupervised learning?
Less clear but generally:

- Recover hidden structure
- Data compression or dimensionality reduction
- Explore or explain data (generate data)
- Construct features for supervised learning

Why modeling a distribution $p(x)$?

- Synthesis of objects (images, text)
- Environment simulator (reinforcement learning, planning)
- Leveraging unlabeled data

Methods:

- Variational Auto-encoders
- Generative Adversarial Nets
- Score-based generative models (Diffusion models)
- PCA
- k-means
- Gaussian Mixture Models

Recap: Maximum likelihood so far?

Model:

$$p(\boldsymbol{y}|x) = \frac{\exp F(\boldsymbol{y}, x, \boldsymbol{w})/\epsilon}{\sum_{\hat{\boldsymbol{y}}} \exp F(\hat{\boldsymbol{y}}, x, \boldsymbol{w})/\epsilon}$$

- $\boldsymbol{y}$: discrete output space
- x: input data

Now:

How about modeling a distribution $p(x)$ for the data?

Given data points x, how can we model $p(x)$?

- Maximum likelihood approach:

$$\theta^* = \max_{\theta} \sum_i \log p(x^{(i)}; \theta)$$

  - Fit mean and variance (= parameters θ) of a distribution (e.g., Gaussian)
  - Fit parameters θ of a mixture distribution (e.g., mixture of Gaussian, k-means)
  - Use a variational auto-encoder

Recall: Linear regression (discriminative)

$$p(y^{(i)}|x^{(i)}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(y^{(i)} - \boldsymbol{w}^\top \phi(x^{(i)}))^2\right)$$

Now: (generative)

$$p(x^{(i)}|\underbrace{\mu, \sigma}_{\theta \text{ or } \boldsymbol{w}}) = \mathcal{N}(x^{(i)}|\mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(x^{(i)} - \mu)^2\right)$$

Important difference: we are now interested in modeling the distribution of the data $x^{(i)}$ and not the class labels $y^{(i)}$. Though it is sometimes ambiguous what you call data or labels.

More broadly:

Generative:

$$\ln p_\theta(x^{(i)}) = \ln \sum_{\boldsymbol{z}} p_\theta(x^{(i)}, \boldsymbol{z})$$

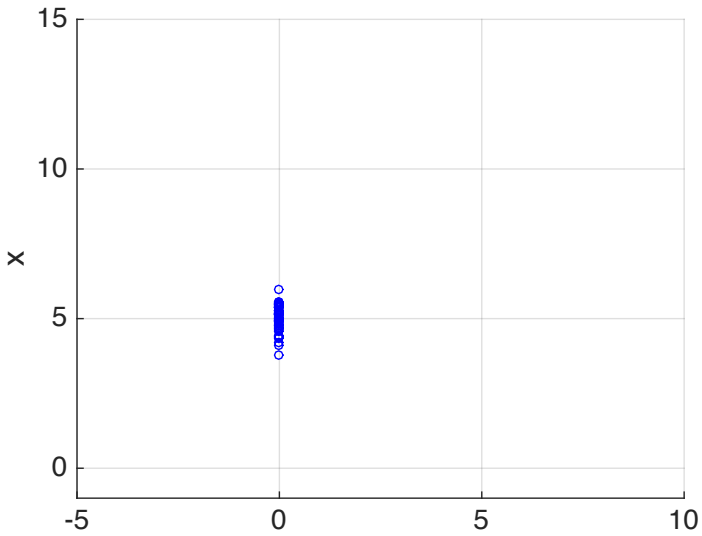
Discriminative:

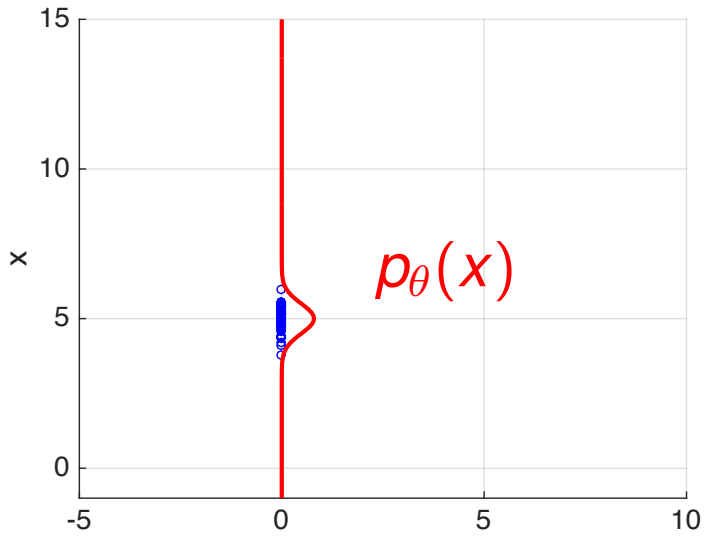
$$\ln p_{\boldsymbol{w}}(\boldsymbol{y}|x^{(i)})$$

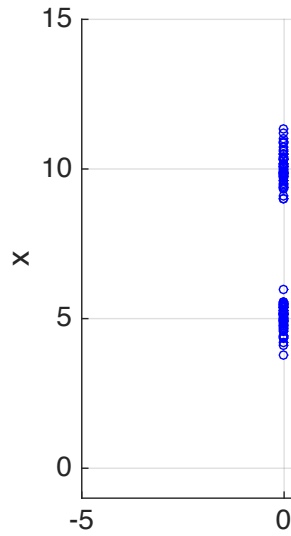
Observations:

- $\boldsymbol{z}$ latent variable; never observed
- $\boldsymbol{y}$ always completely observed

Example:

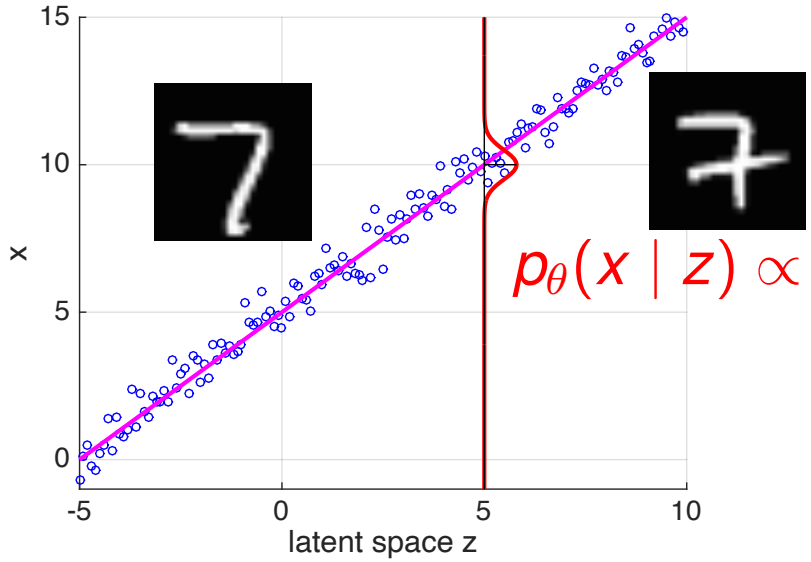






Increasing dimensions:

$x \in \mathbb{R}^{784}$



$$p_\theta(x \mid z) \propto \exp\left(-\frac{(x-\mu(z))^2}{2\sigma^2}\right)$$

Manifold hypothesis:

- x is a high dimensional vector
- data is concentrated around a low dimensional manifold

Examples of low-dimensional manifolds:

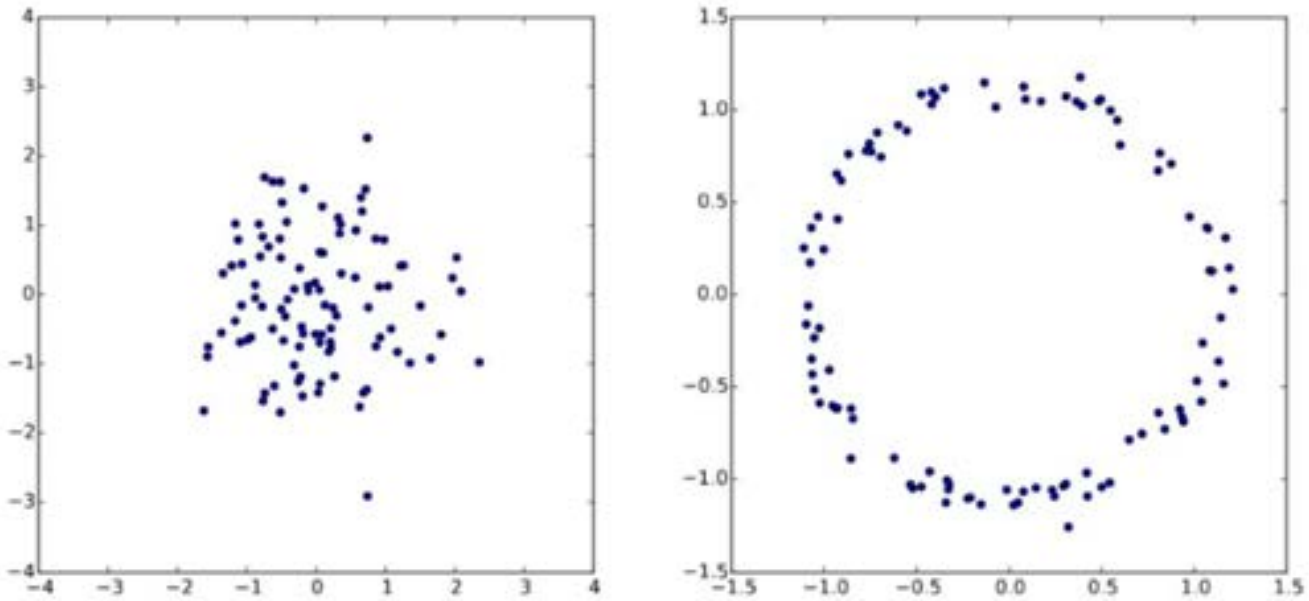
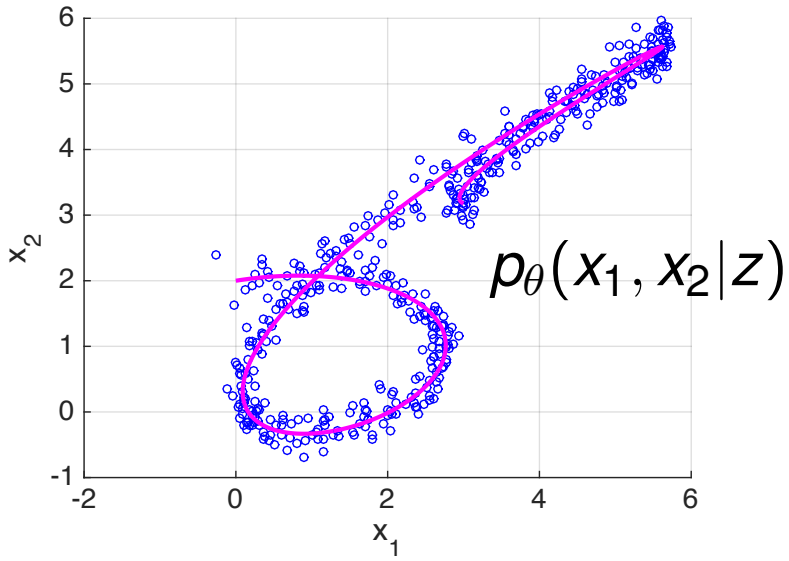


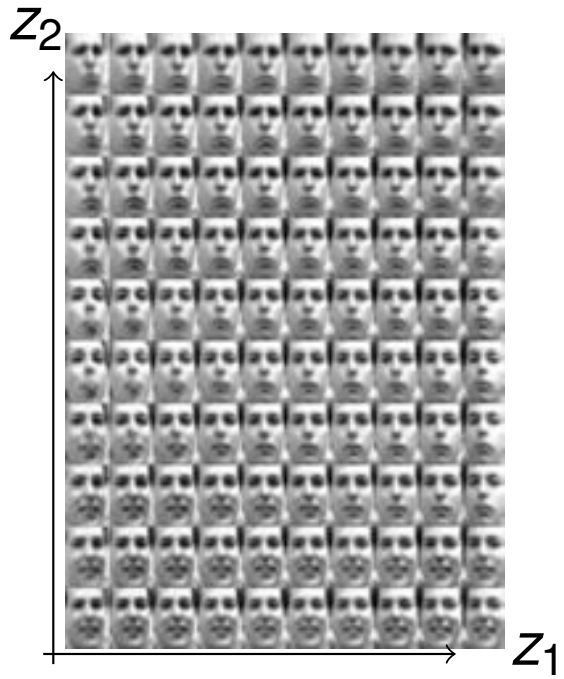
Figure 2: Given a random variable z with one distribution, we can create another random variable $X = g(z)$ with a completely different distribution. Left: samples from a gaussian distribution. Right: those same samples mapped through the function $g(z) = z/10 + z/||z||$ to form a ring. This is the strategy that VAEs use to create arbitrary distributions: the deterministic function g is learned from data.

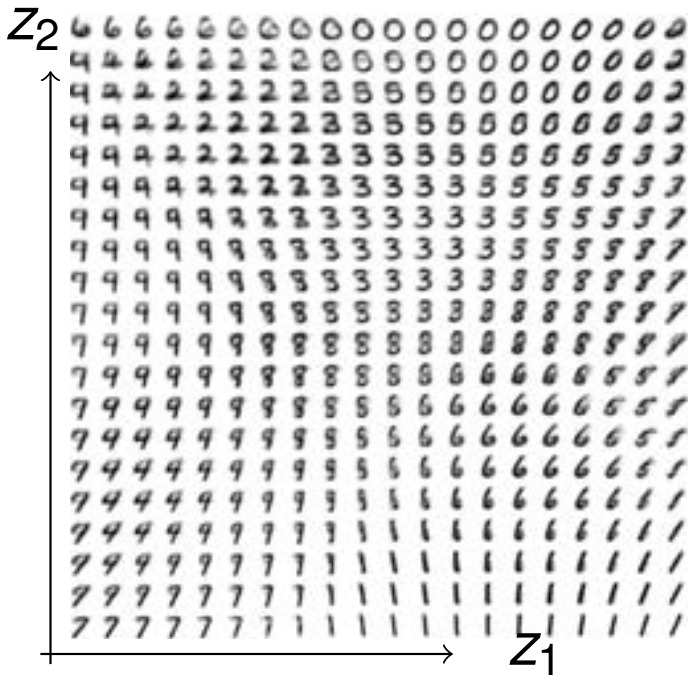
Examples of low-dimensional manifolds:

$z \in [0, 1] \quad \rightarrow$



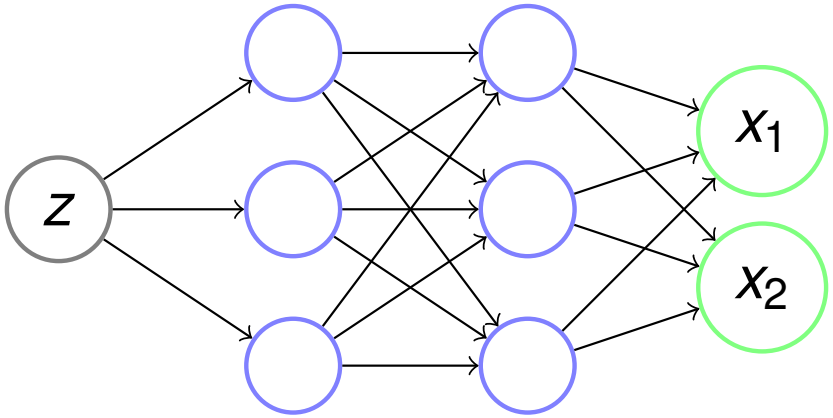
Examples of low-dimensional manifolds: ($z \in \mathbb{R}^2$)
Kingma and Welling: “Auto-Encoding Variational Bayes” (ICLR 2014)



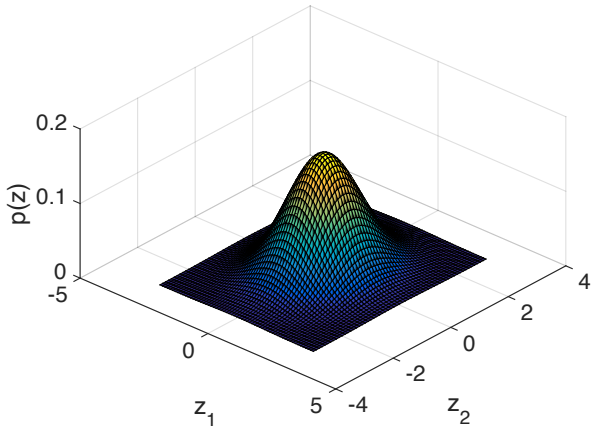


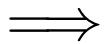
What is $p_\theta(x|z)$?

- Given z, we can generate, compute or sample data x
- Idea: z from simple Gaussian & transformation via deep net
- Decoder



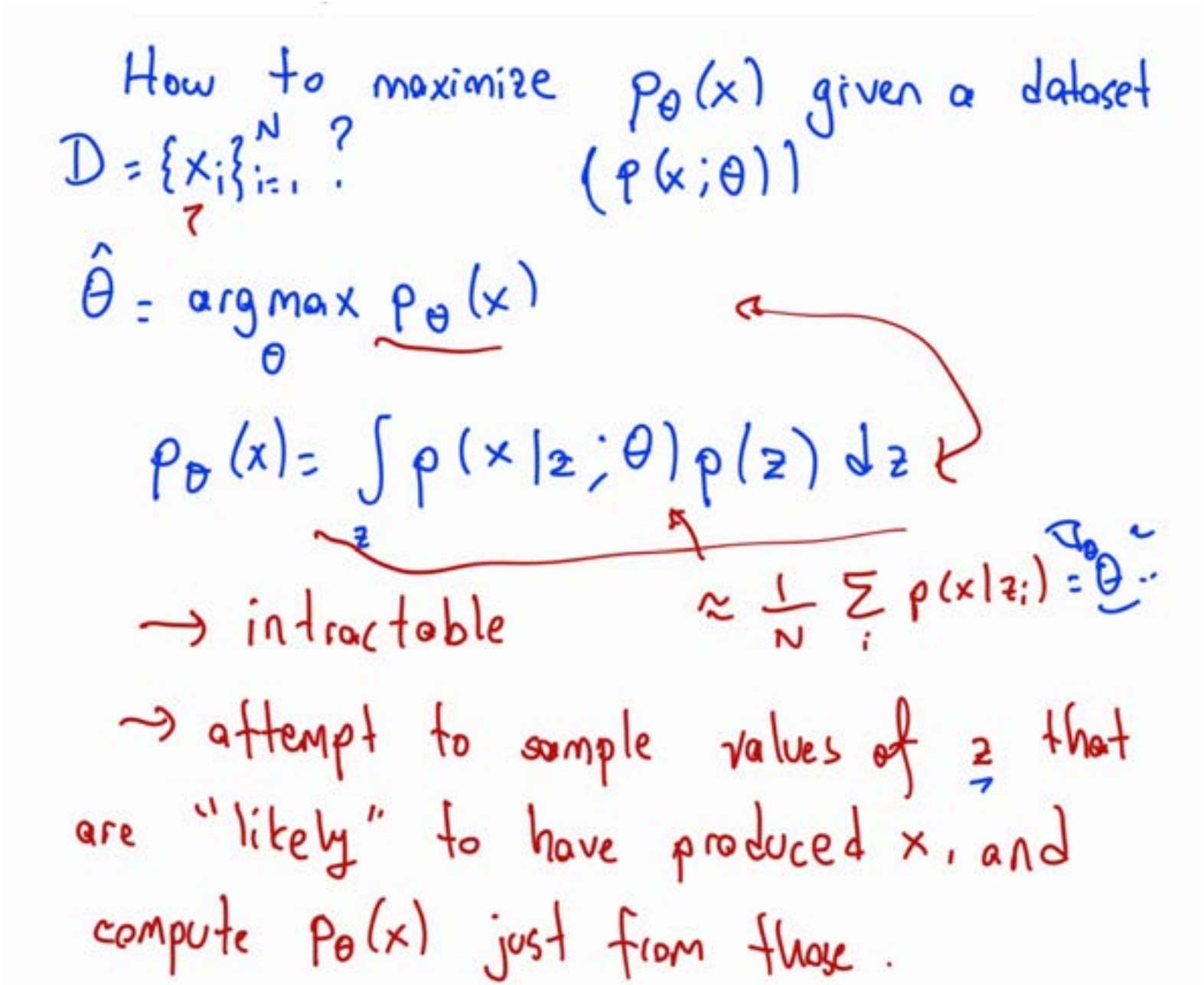
More generally:





But how to optimize for θ? How do we know which z maps to which x?

Examples of low-dimensional manifolds:



Training as an auto-encoder: How about

$$p_\theta(z|x) \qquad p_\theta(x|z)$$

$$x \qquad\qquad z \qquad\qquad x$$

What is $p_\theta(z|x)$?

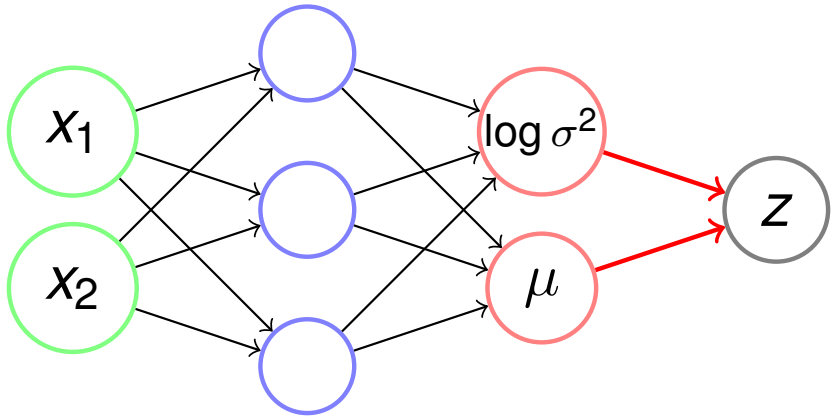
$$p_\theta(z|x) = \frac{p_\theta(x|z)p(z)}{p_\theta(x)} = \frac{p_\theta(x|z)p(z)}{\int_{\hat{z}} p_\theta(x|\hat{z})p(\hat{z})d\hat{z}}$$

How to compute normalization constant?

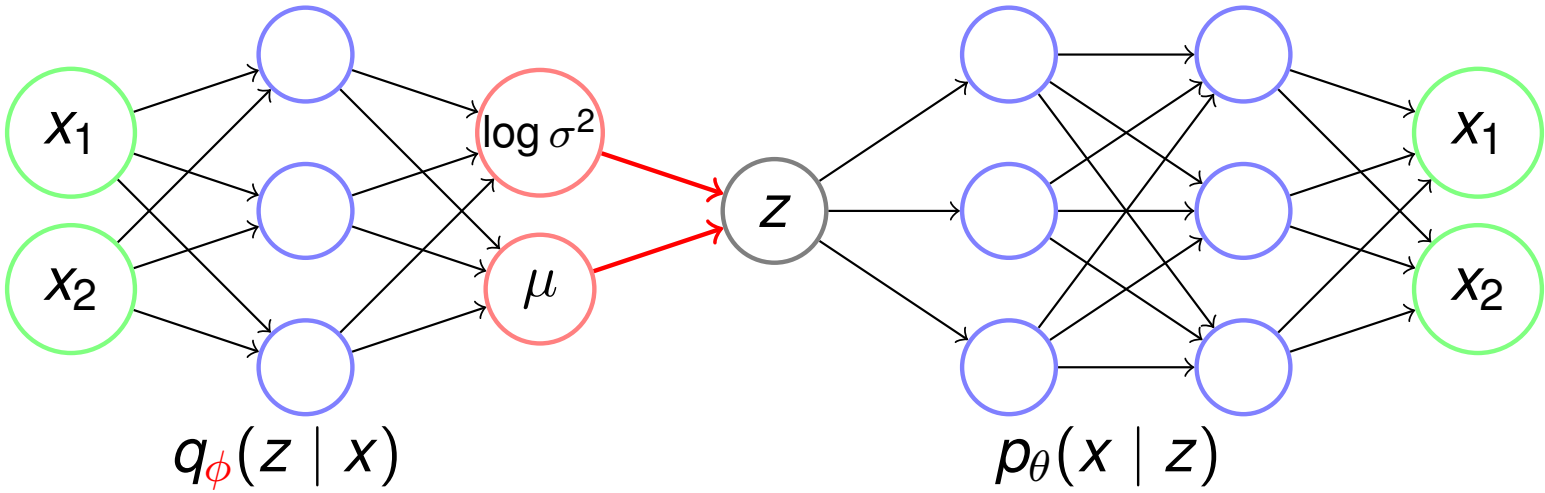
- Sampling techniques (generally very costly)
- Approximate $p_\theta(z|x)$ with $q_\phi(z|x)$ (encoder)

Encoder $q_\phi(z|x)$ as a deep net which computes mean and variance:

$$q_\phi(z|x) = \mathcal{N}(z; \mu_\phi(x), \sigma_\phi(x))$$



Variational auto-encoder architecture:



Learning parameters θ, ϕ via back-propagation

What do we want to optimize? What is the loss function?

$$
\begin{aligned}
\log p_\theta(x) &= \int_z q_\phi(z|x) \log p_\theta(x) \\
&= \int_z q_\phi(z|x) \log \frac{p_\theta(x,z)}{p_\theta(z|x)} \\
&= \int_z q_\phi(z|x) \log \left(\frac{p_\theta(x,z)}{q_\phi(z|x)} \cdot \frac{q_\phi(z|x)}{p_\theta(z|x)} \right) \\
&= \int_z q_\phi(z|x) \log \frac{p_\theta(x,z)}{q_\phi(z|x)} + \int_z q_\phi(z|x) \log \frac{q_\phi(z|x)}{p_\theta(z|x)} \\
&= \mathcal{L}(p_\theta, q_\phi) + D_{KL}(q_\phi, p_\theta) \\
&\geq \mathcal{L}(p_\theta, q_\phi)
\end{aligned}
$$

- Assume $q_\phi(z|x)$ can approximate $p_\theta(z|x)$ well
- $\mathcal{L}$ is often referred to as empirical lower bound (ELBO)

Approximate Inference:

$$
\begin{aligned}
\mathcal{L}(p_\theta, q_\phi) &= \int_z q_\phi(z|x) \log \frac{p_\theta(x,z)}{q_\phi(z|x)} \\
&= \int_z q_\phi(z|x) \log \frac{p_\theta(x|z)p(z)}{q_\phi(z|x)} \\
&= \int_z q_\phi(z|x) \log \frac{p(z)}{q_\phi(z|x)} + \int_z q_\phi(z|x) \log p_\theta(x|z) \\
&= -D_{KL}(q_\phi, p) + \mathbb{E}_{q_\phi}[\log p_\theta(x|z)]
\end{aligned}
$$

- Regularization $-D_{KL}(q_\phi, p)$ with prior $p(z)$ (Gaussian)
- Reconstruction $\mathbb{E}_{q_\phi}[\log p_\theta(x|z)]$

Regularization:

$$-D_{KL}(q_\phi, p) = \int_z q_\phi(z|x) \log \frac{p(z)}{q_\phi(z|x)}$$

- $p(z) = \mathcal{N}(z; 0, 1)$
- $q_\phi(z|x)$ is Gaussian with mean $\mu_\phi(x)$, variance $\sigma^2_\phi(x)$

Reconstruction:

$$\mathbb{E}_{q_\phi}[\log p_\theta(x|z)] = \int_z q_\phi(z|x) \log p_\theta(x|z)$$

Approximate $\mathbb{E}_{q_\phi}[\log p_\theta(x|z)]$ by sampling from $q_\phi(z|x)$

$$\mathbb{E}_{q_\phi}[\log p_\theta(x|z)] \approx \frac{1}{N} \sum_{i=1}^{N} \log p_\theta(x|z^i) \quad \text{where} \quad z^i \sim \mathcal{N}(z; \mu_\phi(x), \sigma_\phi(x))$$

Variational auto-encoder loss function:

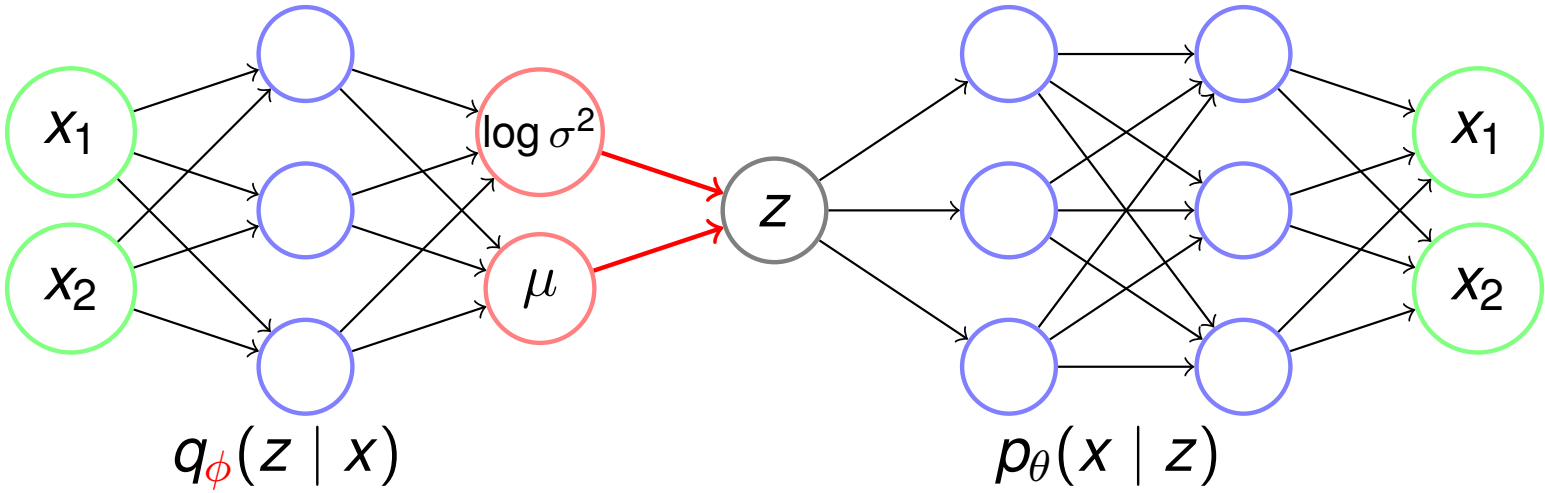
$$\mathcal{L}(p_\theta, q_\phi) \approx -D_{KL}(q_\phi, p) + \frac{1}{N}\sum_{i=1}^{N} \log p_\theta(x|z^i)$$

Intuitively:

$$\rightarrow \quad z \quad \rightarrow \quad \begin{matrix} \log p_\theta(x|z^1) \\ \vdots \\ \log p_\theta(x|z^N) \end{matrix}$$

Typically N small, e.g., N = 1

Where is an issue?



How to backpropagate through the sampling step?

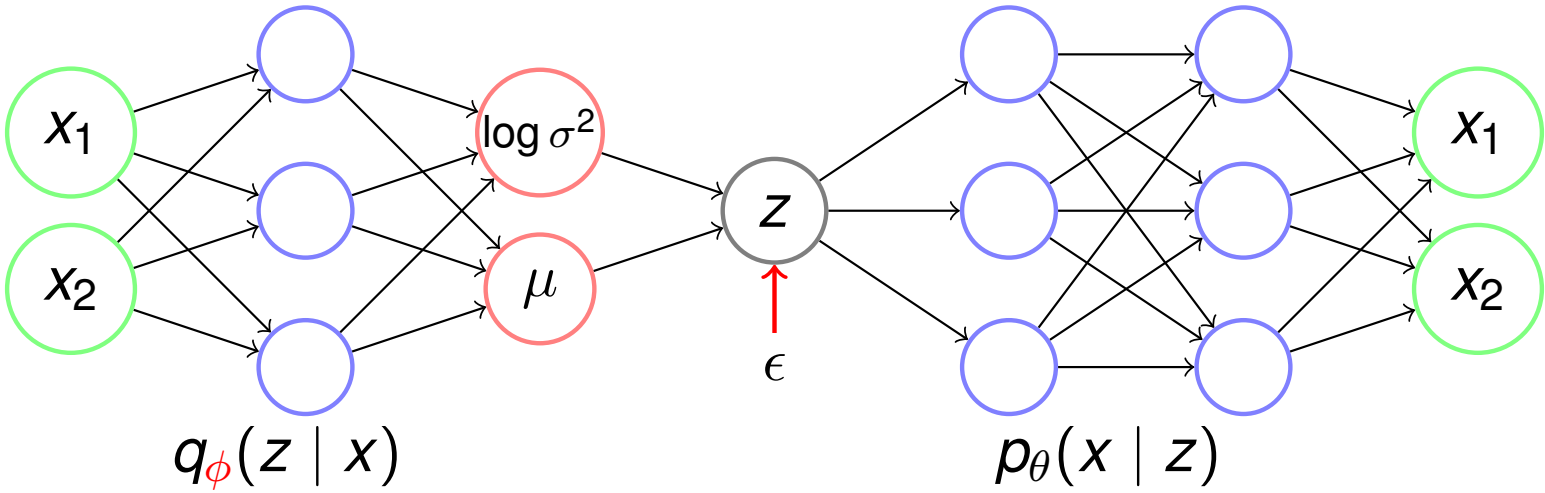
$$z \sim q_\phi(z|x) = \mathcal{N}(z; \mu_\phi(x), \sigma_\phi(x))$$

Reparameterization trick:

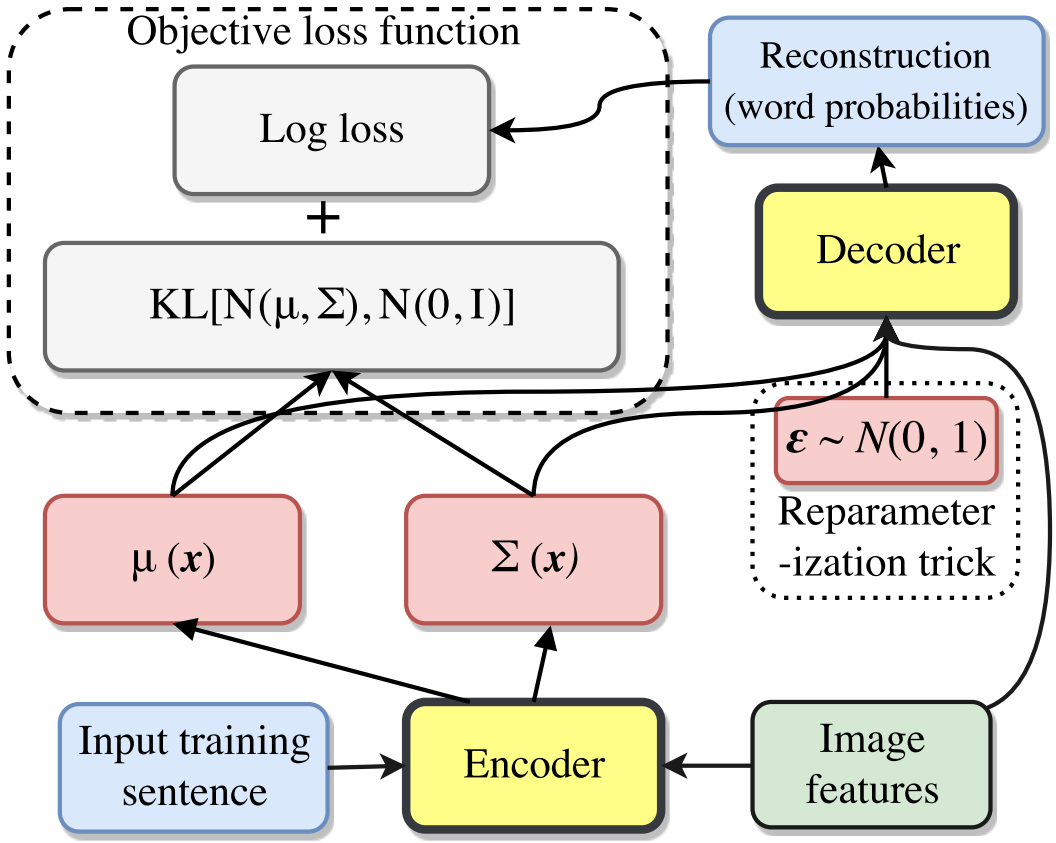
$$z \sim q_\phi(z|x) = \mathcal{N}(z; \mu_\phi(x), \sigma_\phi(x))$$

is equivalent to:

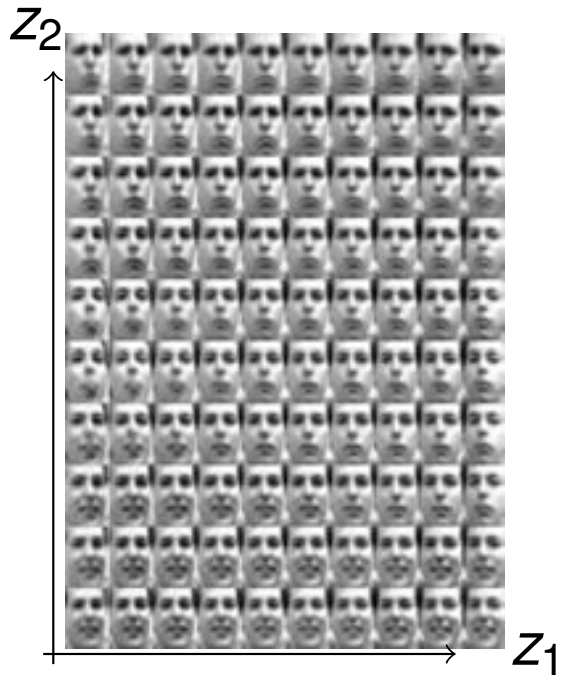
$$z = \mu_\phi(x) + \sigma_\phi(x) \cdot \epsilon \quad \text{where} \quad \epsilon \sim \mathcal{N}(0, 1)$$

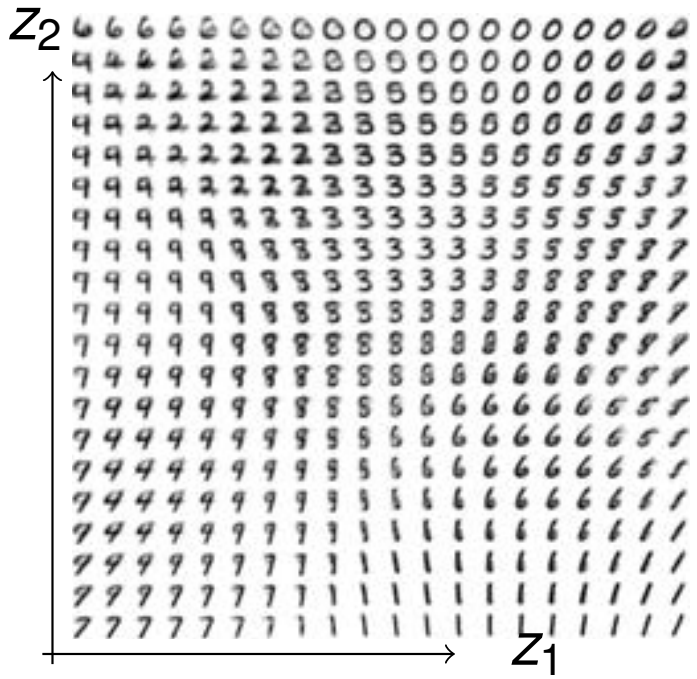


Variational auto-encoder flowchart:



Examples of low-dimensional manifolds: ($z \in \mathbb{R}^2$)
Kingma and Welling: “Auto-Encoding Variational Bayes” (ICLR 2014)





Generative Adversarial Nets

Goals of this section

- Getting to know Generative Adversarial Nets
- Understanding generative methods
- Differentiating between discriminative and generative methods

Reading Material:

- I. Goodfellow et al.; Generative Adversarial Networks; arxiv.org/abs/1406.2661
- M. Arjovsky et al.; Wasserstein GAN; arxiv.org/abs/1701.07875

Main idea:

Don't write a formula for $p(x)$, just learn to sample directly

Advantage:

No summation over large probability spaces

Formulate the problem as a game between two players:

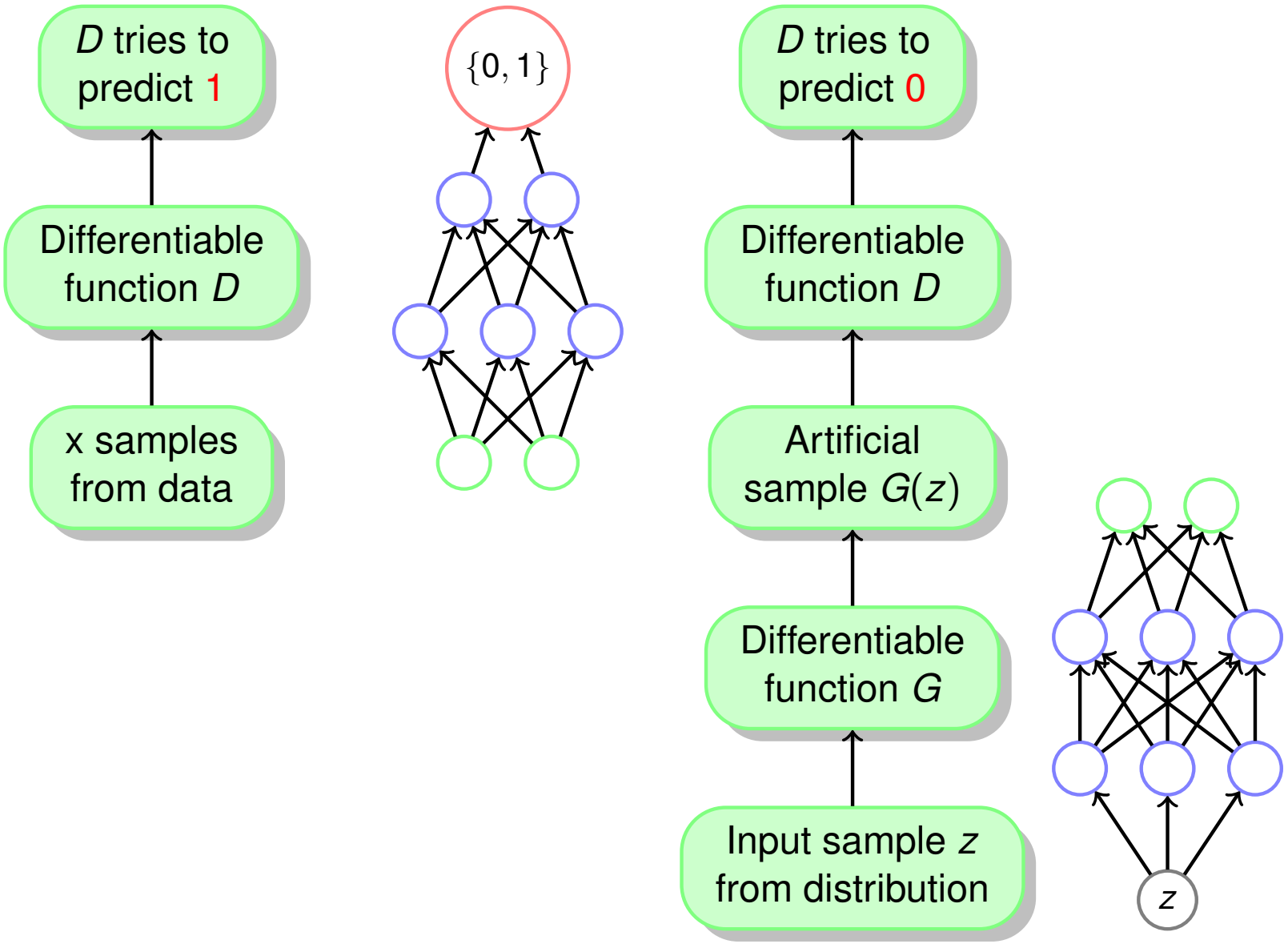
- Generator G
- Discriminator D

Task of the players

- G generates examples
- D predicts whether the example is artificial or real

G tries to "trick" D by generating samples that are hard to distinguish

Pictorially:



Mathematically:

- Generator $G_\theta(z)$
- Discriminator $D_w(x) = p(y = 1|x)$

How to choose w:

$$\min_w -\sum_x \log D_w(x) - \sum_z \log(1 - D_w(G_\theta(z)))$$

How to choose θ:

$$\max_\theta \min_w -\sum_x \log D_w(x) - \sum_z \log(1 - D_w(G_\theta(z)))$$

Generative adversarial nets:

$$\max_{\theta} \min_{w} -\sum_{x} \log D_w(x) - \sum_{z} \log(1 - D_w(G_\theta(z)))$$

How to optimize this theoretically?

Repeat until stopping criteria

1. Gradient step w.r.t. w
2. Gradient step w.r.t. θ

In practice:

Heuristics make this optimization more stable.

Analysis of generative adversarial nets:
What is the optimal discriminator assuming arbitrary capacity?

$$\min_D : \quad -\int_x p_{\text{data}}(x) \log D(x) dx - \int_z p_z(z) \log(1 - D(G_\theta(z))) dz$$
$$= -\int_x p_{\text{data}}(x) \log D(x) + p_G(x) \log(1 - D(x)) dx$$

Euler-Lagrange formalism:

$$S(D) = \int_x L(x, D, \dot{D}) dx$$

Stationary D from

$$\frac{\partial L(x, D, \dot{D})}{\partial D} - \cancel{\frac{d}{dx} \frac{\partial L(x, D, \dot{D})}{\partial \dot{D}}} = 0$$

What is the optimal discriminator assuming arbitrary capacity?

$$\frac{\partial L(x, D, \dot{D})}{\partial D} = -\frac{p_{\text{data}}}{D} + \frac{p_G}{1 - D} = 0$$

Consequently:

$$D^*(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_G(x)}$$

Given the optimal discriminator, what is the optimal generator?

$$
\begin{aligned}
& -\int_x p_{\text{data}}(x) \log D^*(x) + p_G(x) \log(1 - D^*(x)) dx \\
= \quad & -\int_x p_{\text{data}}(x) \log \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_G(x)} + p_G(x) \log \frac{p_G(x)}{p_{\text{data}}(x) + p_G(x)} dx \\
= \quad & -2\,\text{JSD}(p_{\text{data}}, p_G) + \log(4)
\end{aligned}
$$

$$
\text{JSD}(p_{\text{data}}, p_G) = \frac{1}{2}\,\text{KL}(p_{\text{data}}, M) + \frac{1}{2}\,\text{KL}(p_G, M) \quad \text{with} \quad M = \frac{1}{2}(p_{\text{data}} + p_G)
$$

Consequently:

$$
p_G(x) = p_{\text{data}}(x)
$$

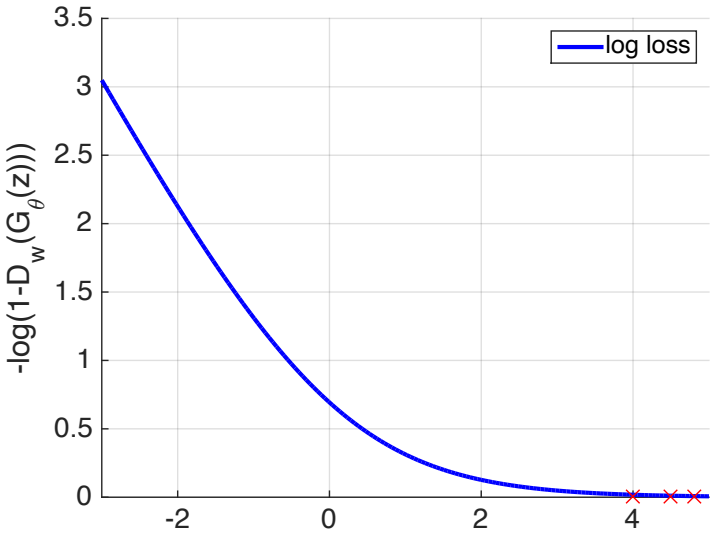
Some heuristics that improve optimization of:

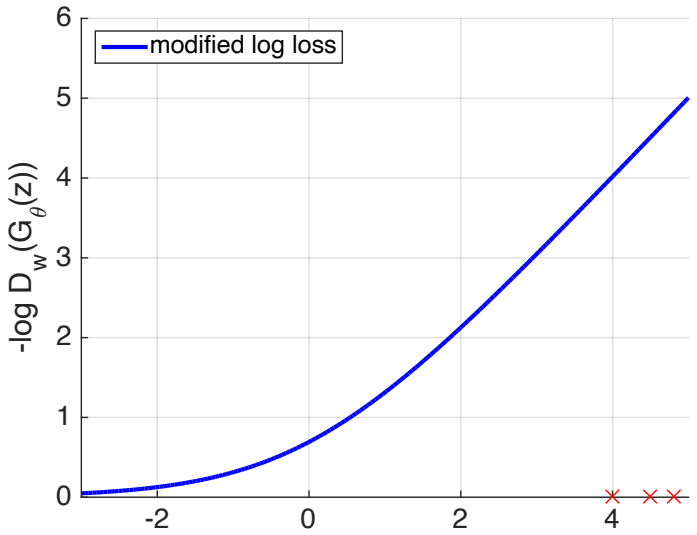
$$\max_{\theta} \min_{w} -\sum_{x} \log D_w(x) - \sum_{z} \log(1 - D_w(G_\theta(z)))$$

- If G is very bad and D is very good: almost no gradient
- Solve instead

$$\min_{\theta} -\sum_{z} \log D_w(G_\theta(z))$$

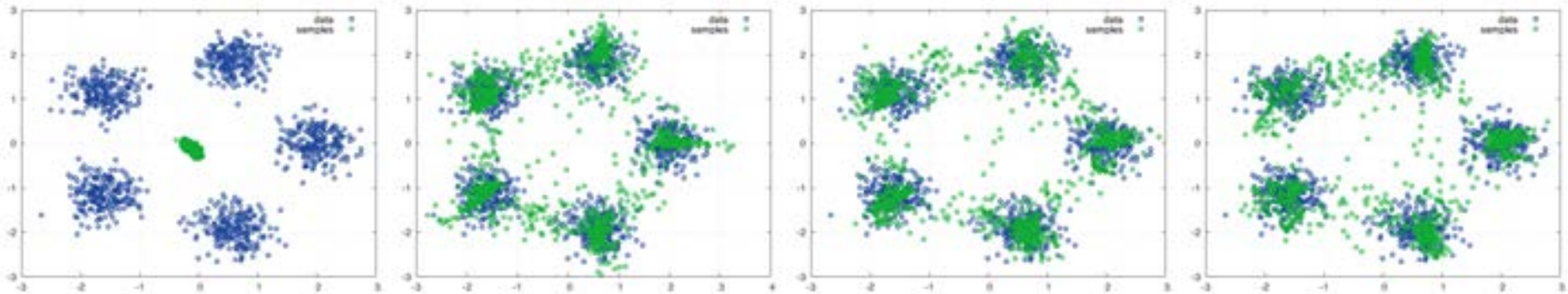
- Issue: no joint cost function for D and G





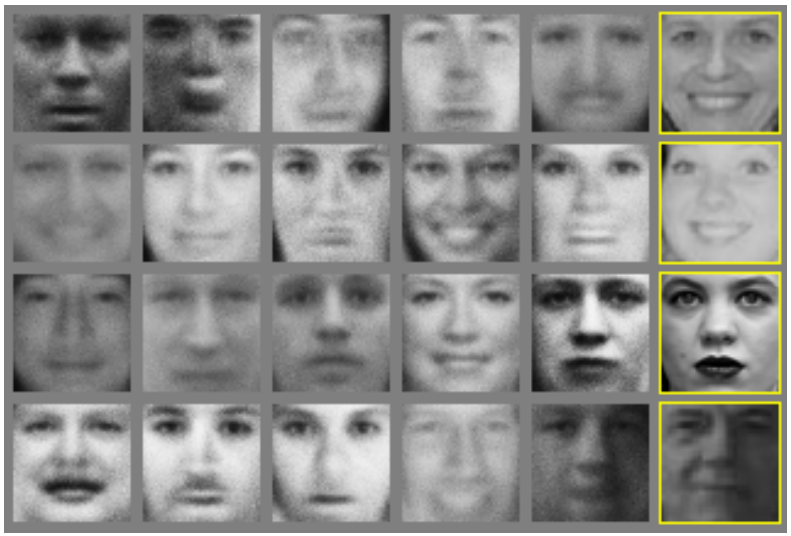
Plenty of additional impressive tricks.

Some results on toy data:



0 Iterations 2000 Iterations 4000 Iterations 5000 Iterations

Application: modeling of images



Radford et al. (2016); DCGAN

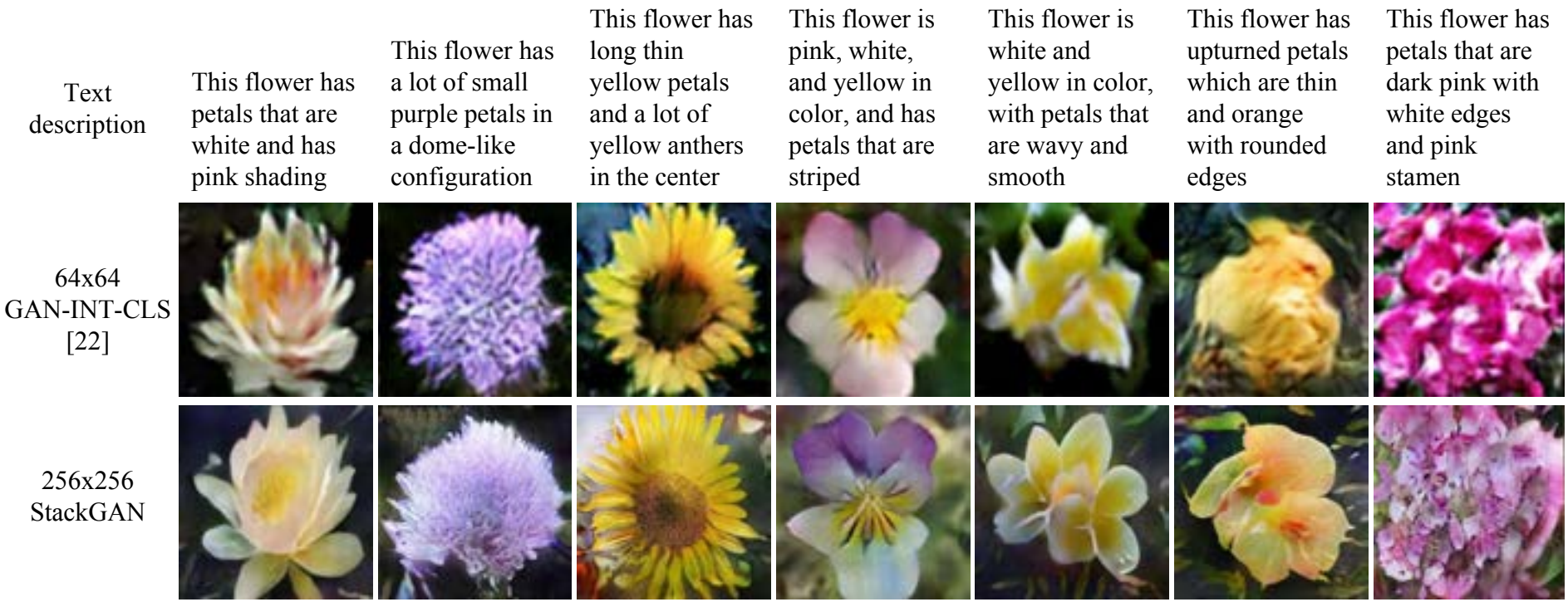


Figure 4. Example results by our proposed StackGAN and GAN-INT-CLS [22] conditioned on text descriptions from Oxford-102 test set.

(Score-based Generative Models)

Goals of this lecture

- Getting to know diffusion probabilistic generative models
- Getting to know score-based generative models
- Seeing how to apply them

Reading Material

- Ho, J., Jain, A., and Abbeel, P. Denoising diffusion probabilistic models. Advances in Neural Information Processing Systems 33 (2020): 6840-6851.
- Song, Y., Sohl-Dickstein, J., Kingma, D. P., Kumar, A., Ermon, S., and Poole, B. (2021). Score-based generative modeling through stochastic differential equations. in ICLR. arXiv preprint arXiv:2011.13456.
- Song, Y., Shen, L., Xing, L., and Ermon, S. (2022). Solving inverse problems in medical imaging with score-based generative models. in ICLR. arXiv preprint arXiv:2111.08005.

Likelihood based models

- Dataset $\{x_1, x_2, ..., x_N\}$, $x_i \sim p(x)$
- Modeled likelihood function

$$p_\theta(x) = \frac{e^{-f_\theta(x)}}{Z_\theta}$$

- $f_\theta : \mathbb{R}^d \to \mathbb{R}$
- Z_θ: Normalizing constant such that $\int p_\theta(x)dx = 1$.
- Maximum likelihood:

$$\max_\theta \sum_{i=1}^{N} \log p_\theta(x_i)$$

- Challenges:
  - Intractable normalizing constant computation
  - Restriction on model architecture

Score-based models

How can the aforementioned challenges be addressed?

- Use score based models: $\nabla_x \log p(x)$
- Learn $s_\theta(x) \approx \nabla_x \log p_\theta(x)$

How does learning score-based models help?

- No need for the computation of the normalizing factor

$$s_\theta(x) = -\nabla_x f_\theta(x) - \nabla_x \log Z_\theta = -\nabla_x f_\theta(x)$$

- Expands the family of models we can use

How to learn the score function?

$$\min_\theta \mathbb{E}_{p(x)}[\|\nabla_x \log p(x) - s_\theta(x)\|_2^2]$$

- Use score-matching techniques to solve it
- Estimate the parameters directly on the dataset, using SGD

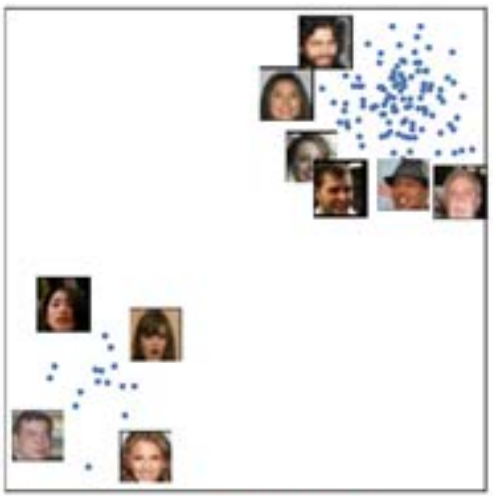
Langevin Dynamics

- Once $s_\theta(x) \approx \nabla_x \log p_\theta(x)$ is trained, iterative process called **Langevin dynamics** can be used to draw samples from $p(x)$.

$$x_{i+1} \leftarrow x_i + \epsilon \nabla_x \log p(x) + \sqrt{2\epsilon} z_i, \quad i = 0, 1, ..., K$$

  - $x_0 \sim \pi(x)$, an arbitrary prior distribution
  - $z_i \sim \mathcal{N}(0, I)$
  - When $\epsilon \to 0$ and $K \to \infty$, $x_K \sim p(x)$

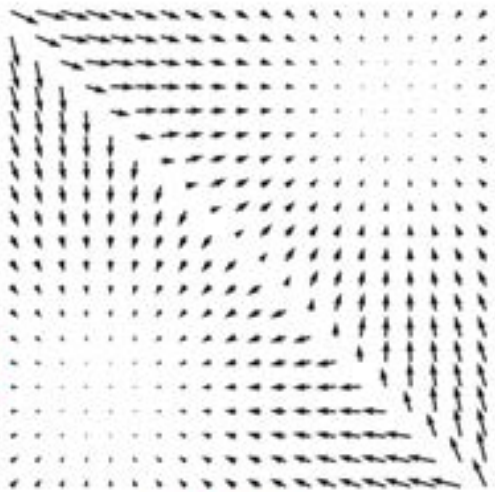
Score based generative modeling



Data samples

$\{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N\} \overset{\text{i.i.d.}}{\sim} p(\mathbf{x})$

score matching



Scores

$\mathbf{s}_\theta(\mathbf{x}) \approx \nabla_{\mathbf{x}} \log p(\mathbf{x})$

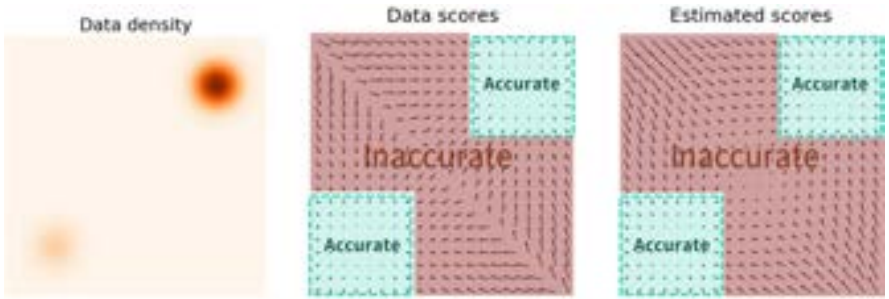
Langevin dynamics

New samples

Noise perturbations

Pitfalls:

- Less accurate score estimation for less densely populated areas



- Perturb data points with noise and train score-based models on the noisy data points

- Add noise: Cover low density regions, but large noise alters the original data distribution

Noise perturbations

Solution:

- Use multiple scales of noise perturbations, $\sigma_1 < \sigma_2 < ... < \sigma_L$

$$p_{\sigma_i}(x) = \int p(y)\mathcal{N}(x; y, {\sigma_i}^2 I)dy$$

- Train a noise conditional score-based model $s_\theta(x, i) \approx \nabla_x \log p_{\sigma_i}(x)$, $i = 1, 2, ..., L$.

- After training $s_\theta(x, i)$, we produce samples using annealed Langevin dynamics for $i = L, L-1, ..., 1$ in sequence.

Score-based generative modeling with SDEs

- # Noise scales $L \to \infty$
  - Unified framework
  - Higher quality samples
  - Exact log-likelihood computation
  - Controllable generation for inverse problem solving

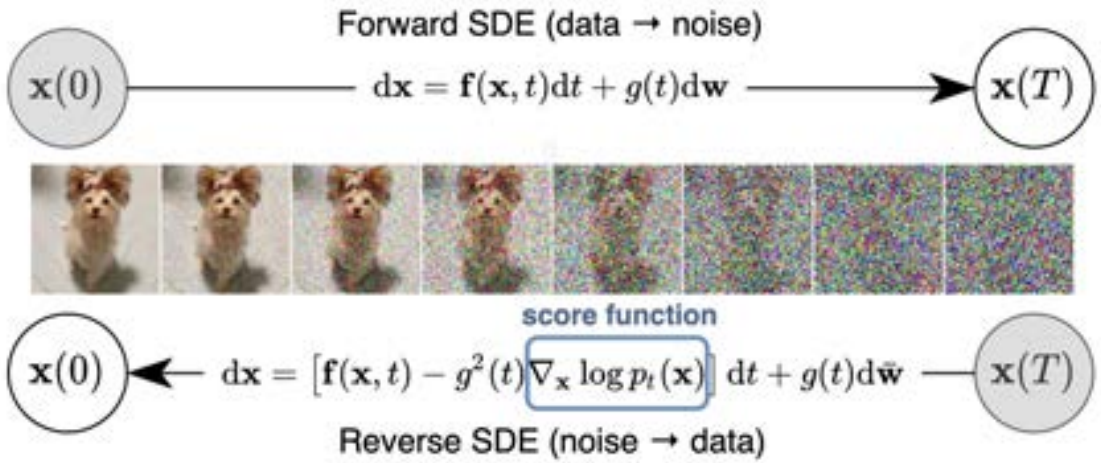
$$dx = f(x, t)dt + g(t)dw$$

- $f(., t) : \mathbb{R}^d \to \mathbb{R}^d$: The drift coefficient
- $g(t) \in \mathbb{R}$: Diffusion coefficient
- w: Standard Brownian motion
- dw: Small white noise

Reversing the SDE for sample generation

Reverse SDE

$$dx = [f(x,t) - g^2(t)\nabla_x \log p_t(x)]dt + g(t)dw$$



- Train time-dependent score based model $s_\theta(x,t) \approx \nabla_x \log p_{\sigma_i}(x)$

How to solve the reverse SDE?

Reverse SDE

$$dx = [f(x,t) - g^2(t)\nabla_x \log p_t(x)]dt + g(t)dw$$

- Numerical SDE solvers
- Reverse diffusion sampling

$$\Delta x \leftarrow [f(x,t) - g^2(t)s_\theta(x,t)]\Delta t + g(t)\sqrt{|\Delta t|}z_t$$

- $z_t \sim \mathcal{N}(0, I)$
- Use predictor-corrector process to generate higher quality samples at each t

Spacial cases: SMLD

SMLD(Score matching with Langevin dynamics)

$$x_i = x_{i-1} + \sqrt{\sigma_i^2 - \sigma_{i-1}^2} z_{i-1}, \quad i = 1, ..., N$$
$$z_{i-1} \sim \mathcal{N}(0, I)$$

Once $N \to \infty$:

$$dx = \sqrt{\frac{d[\sigma^2(t)]}{dt}} dw$$

Spacial cases: DDPM

DDPM(Denoising Diffusion Probabilistic Model)

$$x_i = \sqrt{1-\beta_i}x_{i-1} + \sqrt{\beta_i}z_{i-1}$$
$$0 < \beta_1, \beta_2, ..., \beta_N < 1$$

Once $N \to \infty$:

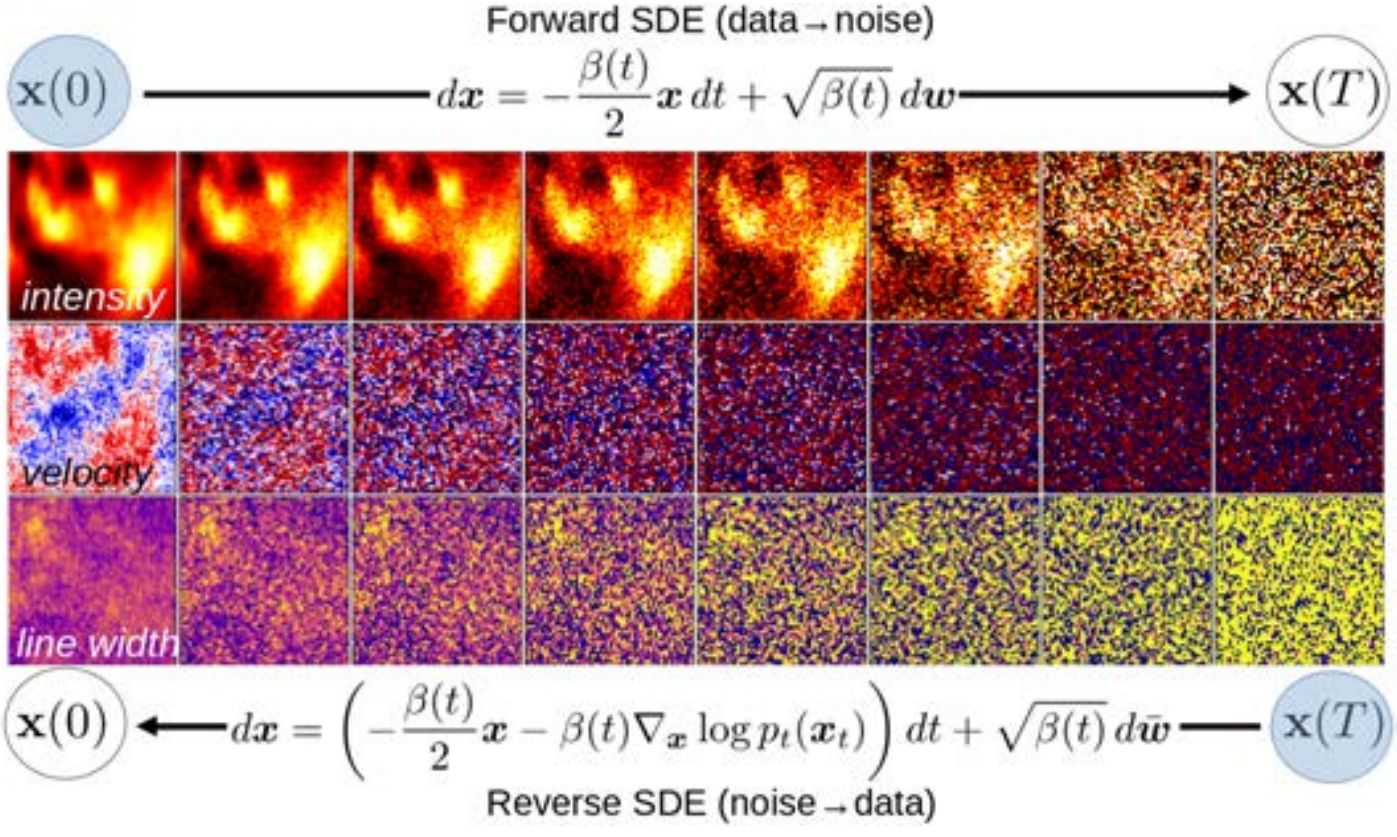
$$dx = -\frac{1}{2}\beta(t)xdt + \sqrt{\beta(t)}dw$$

In this paper:

$$dx = -\frac{1}{2}\beta(t)xdt + \sqrt{\beta(t)(1 - e^{-2\int_0^t \beta(s)ds})}dw$$

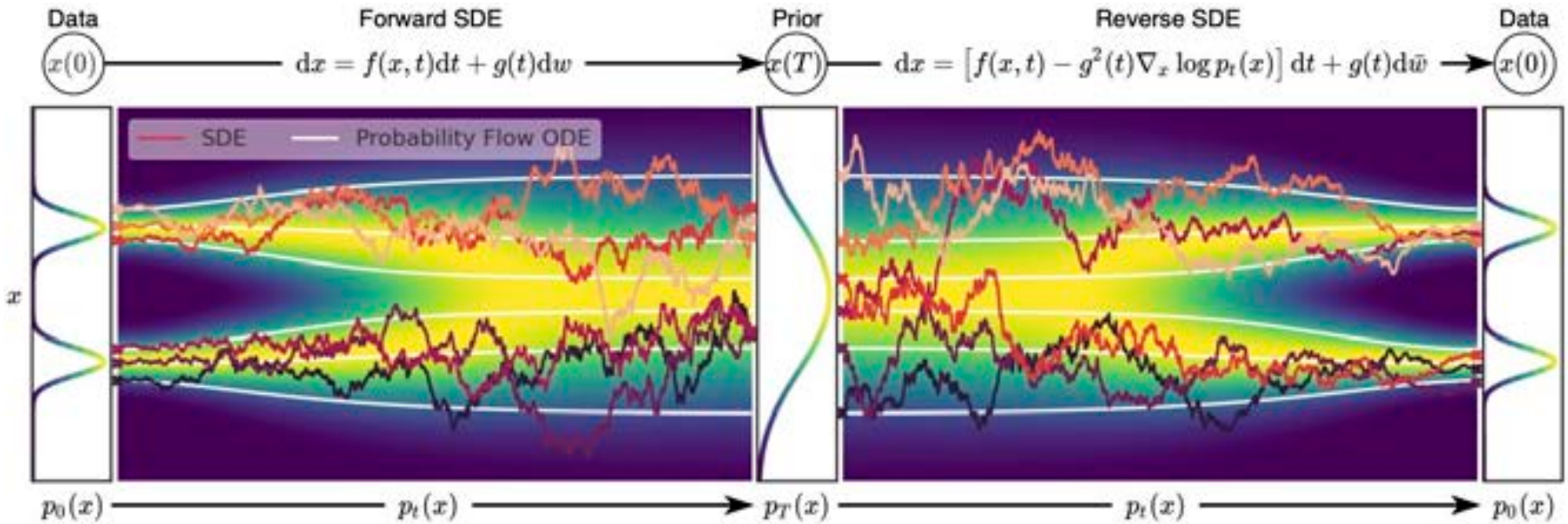
Unsupervised Inversion Methods

- Illustration of forward and reverse processes for the multi-channel parametrization of the data cube (intensity, velocity, line width)



Likelihood computation

- For a diffusion process, there exists a deterministic process whose trajectories share the same marginal probability densities $\{p_t\}_{t\in[0,T]}$ as the SDE.



- This deterministic process satisfies an ODE (probability flow ODE), given by

$$dx = [f(x,t) - \frac{1}{2}g^2(t)\nabla_x \log p_t(x)]dt$$

Score-Based Generative Models and Diffusion Models

- Score-based models learn the gradient of the log-density (score) of the data distribution.
- Diffusion models: gradually add noise to data (forward process), then learn to reverse this process.
- Powerful for generative modeling: can sample from complex, high-dimensional distributions.
- Recent advances enable their use as priors for inverse problems.

Diffusion Posterior Sampling (DPS): Concept

- DPS adapts diffusion models for solving inverse problems.
- Combines a learned prior (diffusion model) with the measurement likelihood.
- At each step, samples are nudged to agree with measurements while remaining plausible under the prior.
- Enables conditional sampling: $p(\mathbf{x}|\mathbf{y})$.

DPS for SSI: Problem Formulation

- Forward model: $\mathbf{y} = \mathcal{A}(\mathbf{x}) + \mathbf{n}$, where $\mathcal{A}$ is the nonlinear SSI operator.
- Goal: sample from the posterior $p(\mathbf{x}|\mathbf{y})$ given noisy SSI measurements.
- Prior: score-based diffusion model trained on solar parameter maps.
- Likelihood: measurement consistency via the forward model.

DPS Algorithm: Steps

- Start from Gaussian noise; iteratively denoise using the learned score network.
- At each step, update sample to increase likelihood (match measurements) and follow the prior.
- Key update (simplified):

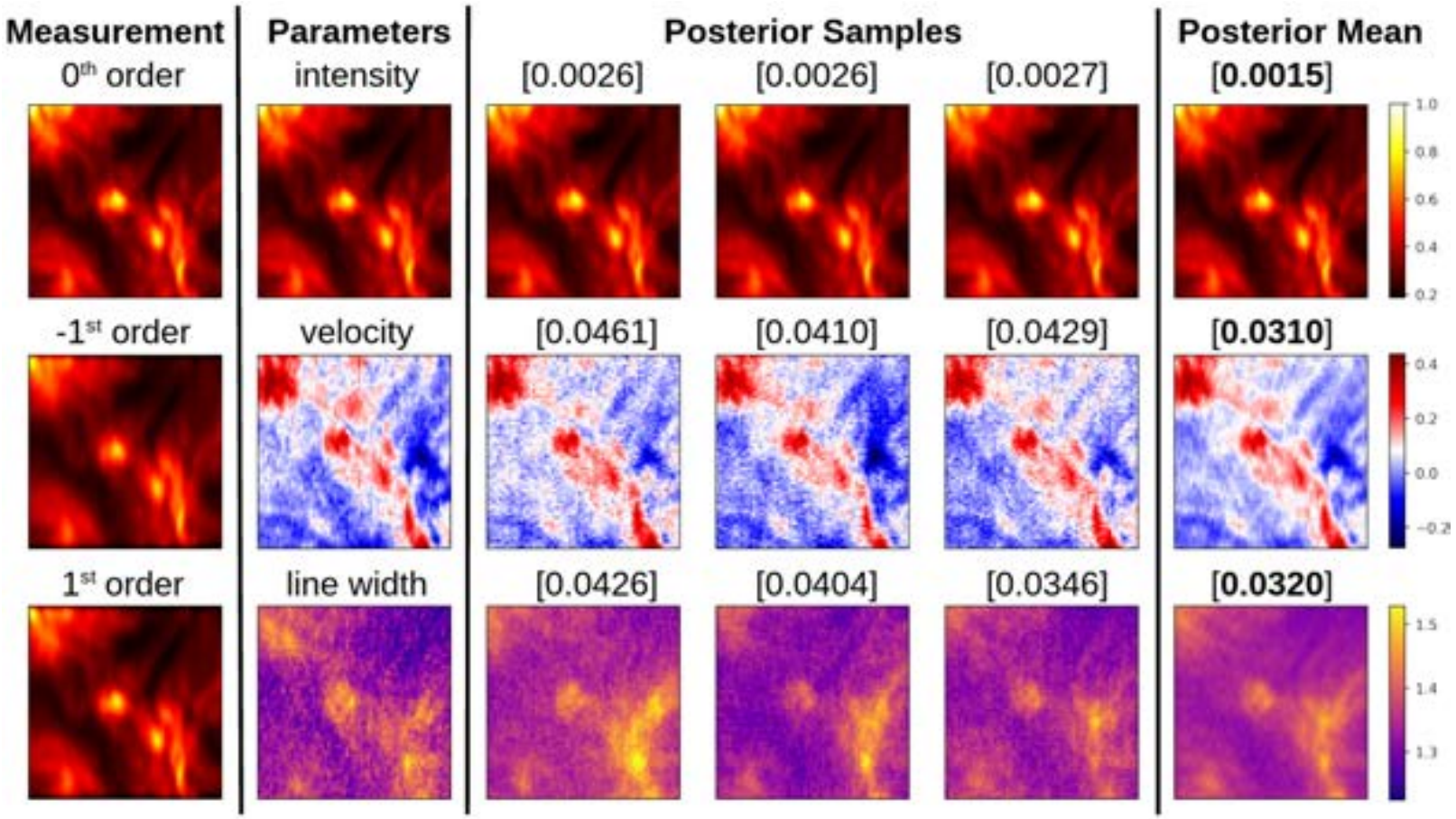
$$\mathbf{x}_{t-1} \leftarrow \mathbf{x}_t + \text{score prior} - \eta \nabla_{\mathbf{x}_t} \|\mathbf{y} - \mathcal{A}(\hat{\mathbf{x}}_0(\mathbf{x}_t))\|^2$$

- See Algorithm 1 in thesis for full details.

Training the Diffusion Model

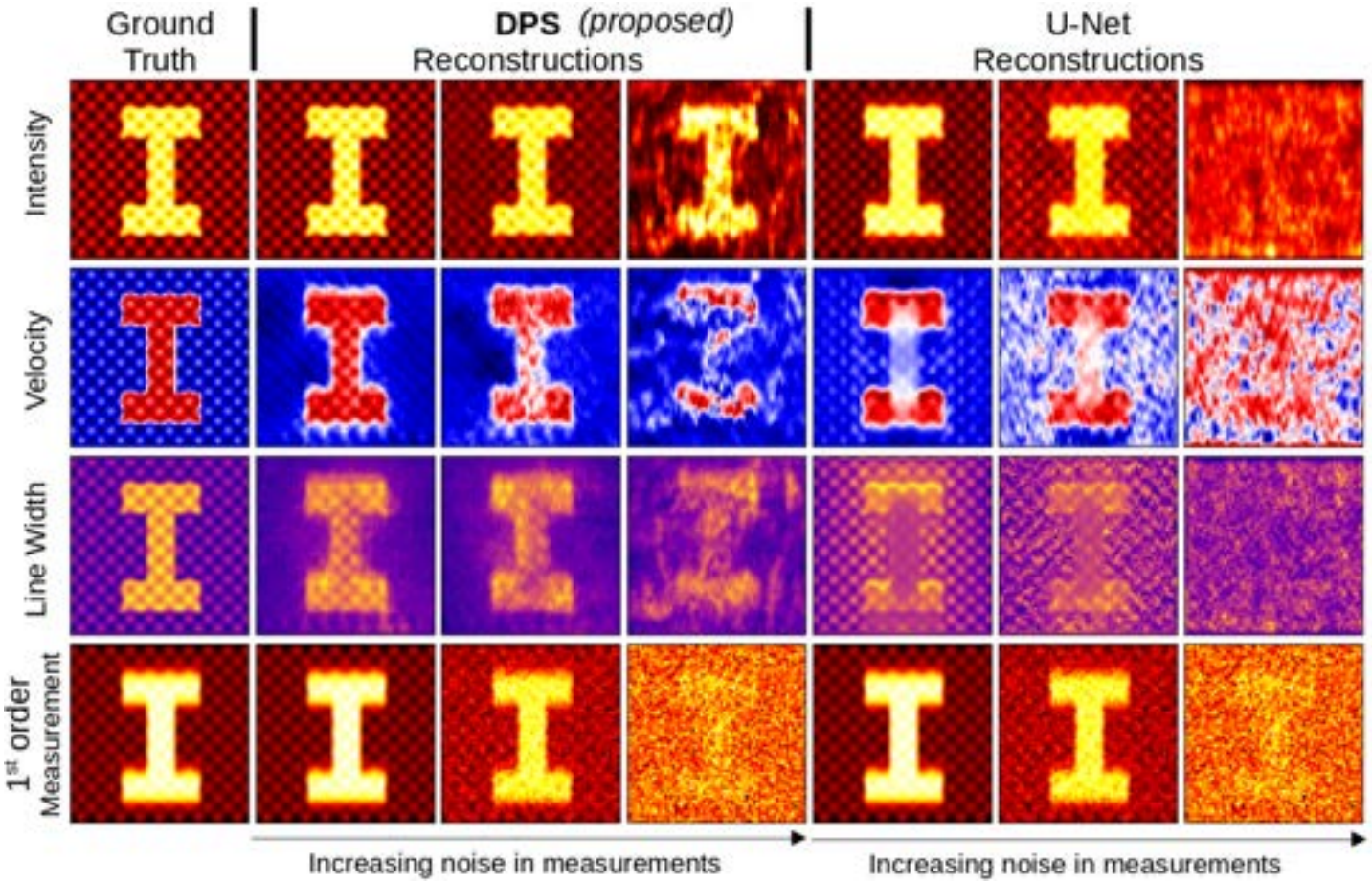
- Training data: parameter maps (intensity, velocity, line width) from Hinode/EIS solar observations.
- Network: NCSN++ architecture, trained to denoise at multiple noise levels.
- Training is unsupervised: only requires samples of $\mathbf{x}$, not paired measurements.
- Enables learning a powerful, data-driven prior for solar scenes.

Posterior Sampling Results: In-Distribution



- DPS generates multiple plausible reconstructions (posterior samples) for a given measurement.
- Posterior mean provides a point estimate; spread of samples quantifies uncertainty.

Posterior Sampling Results: Out-of-Distribution Robustness



- DPS is robust to out-of-distribution (OOD) test data, outperforming supervised U-Net.
- Critical for scientific imaging where ground truth is scarce and test data may differ from training.

Conclusions
Diffusion Posterior Sampling

- DPS enables principled uncertainty quantification and robust inference for SSI inversion.
- Combines the strengths of deep generative models and Bayesian inference.
- Outperforms supervised methods in challenging and OOD scenarios.
- Flexible framework: can be extended to more complex models, better priors, and joint optimization with instrument design.