Introduction to Machine Learning: Lecture 3

Michael Kagan

SLAC

Hadron Collider Physics Summer School

August 25, 2021

Outline

- Lecture 1
  - Brief introduction to probability and statistics
  - Introduction to Machine Learning fundamentals
  - Linear Models

- Lecture 2
  - Neural Networks
  - Deep Neural Networks
  - Convolutional, Recurrent, and Graph Neural Networks

- Lecture 3
  - Unsupervised Learning
  - Autoencoders
  - Generative Adversarial Networks and Normalizing Flows

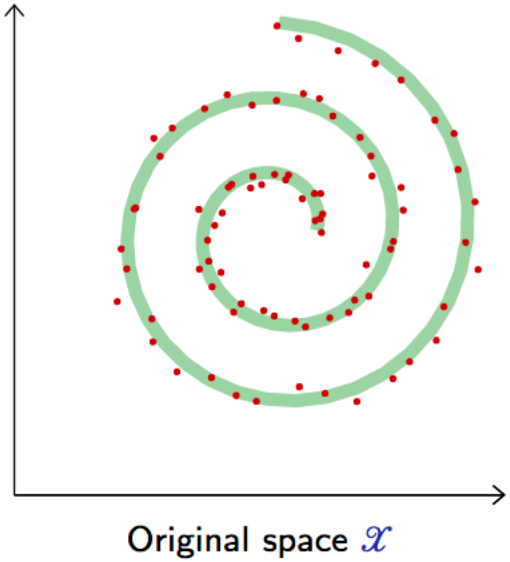
Beyond Regression and Classification

- Not all tasks are predicting a label from features
  - Data synthesis / simulation
  - Density estimation
  - Anomaly detection
  - Denoising, super resolution
  - Data compression
  - …

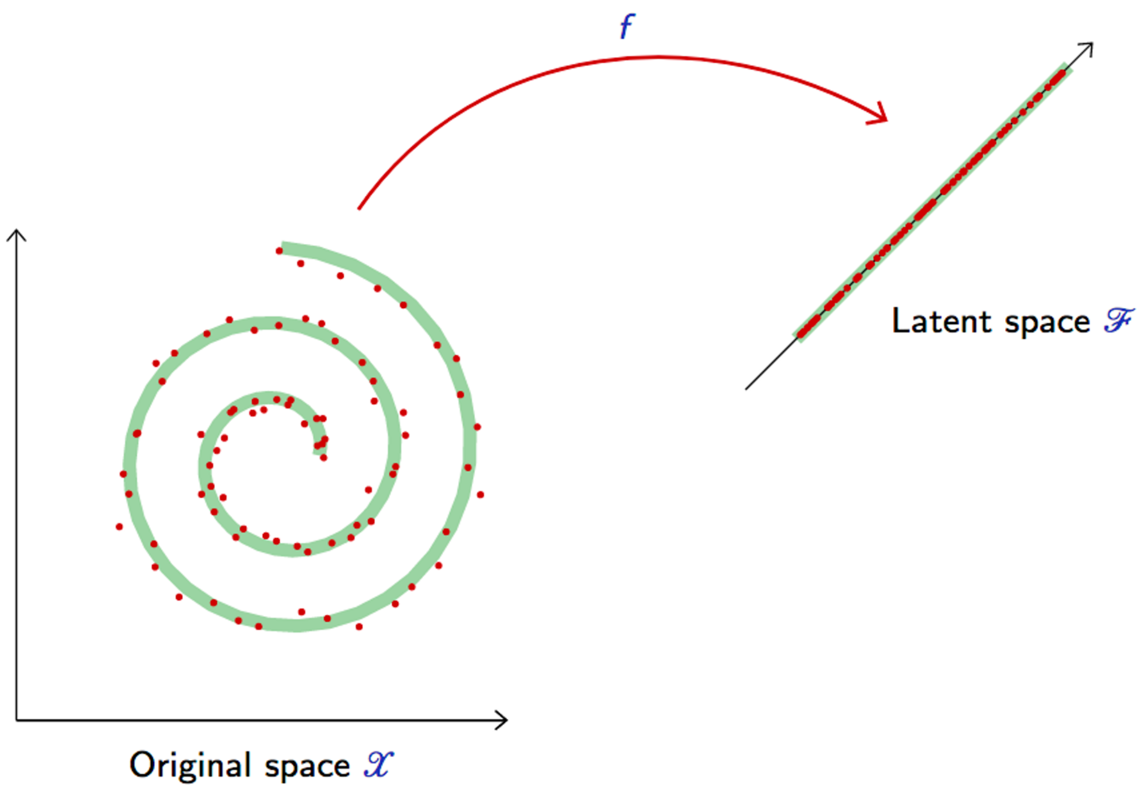
- Requires Unsupervised Learning

- Often framed as **modeling the lower dimensional "meaningful degrees of freedom"** that describe the data

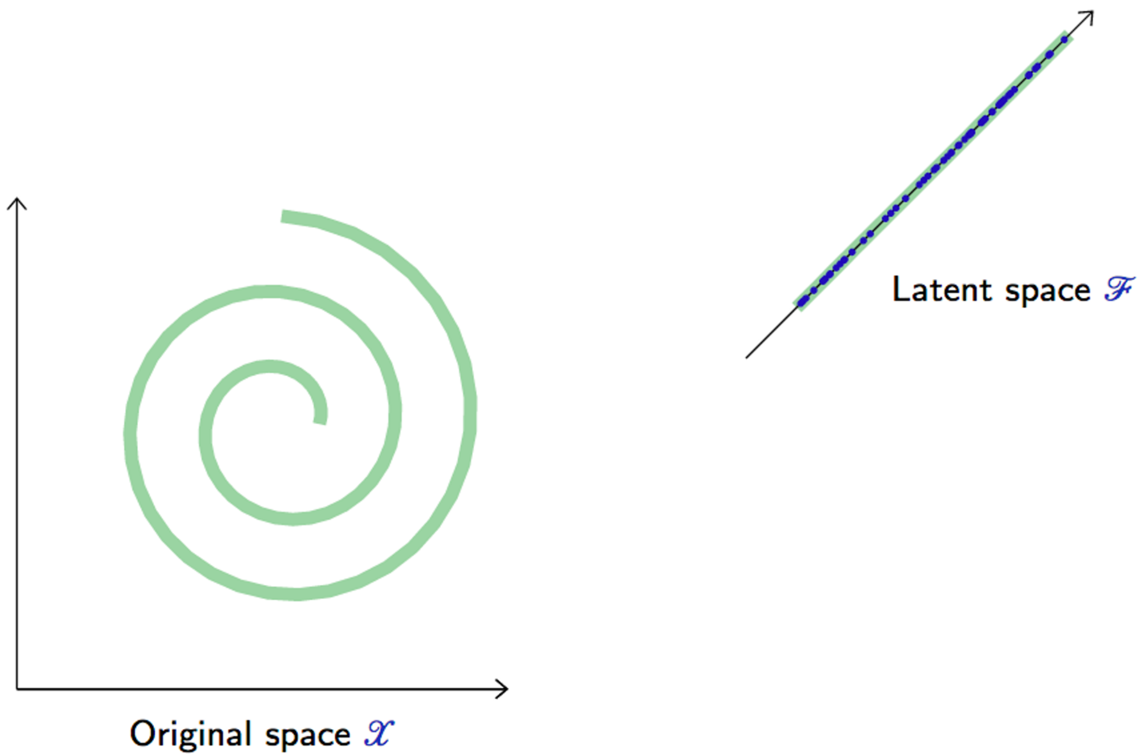
Modeling Data and Meaningful Degrees of Freedom



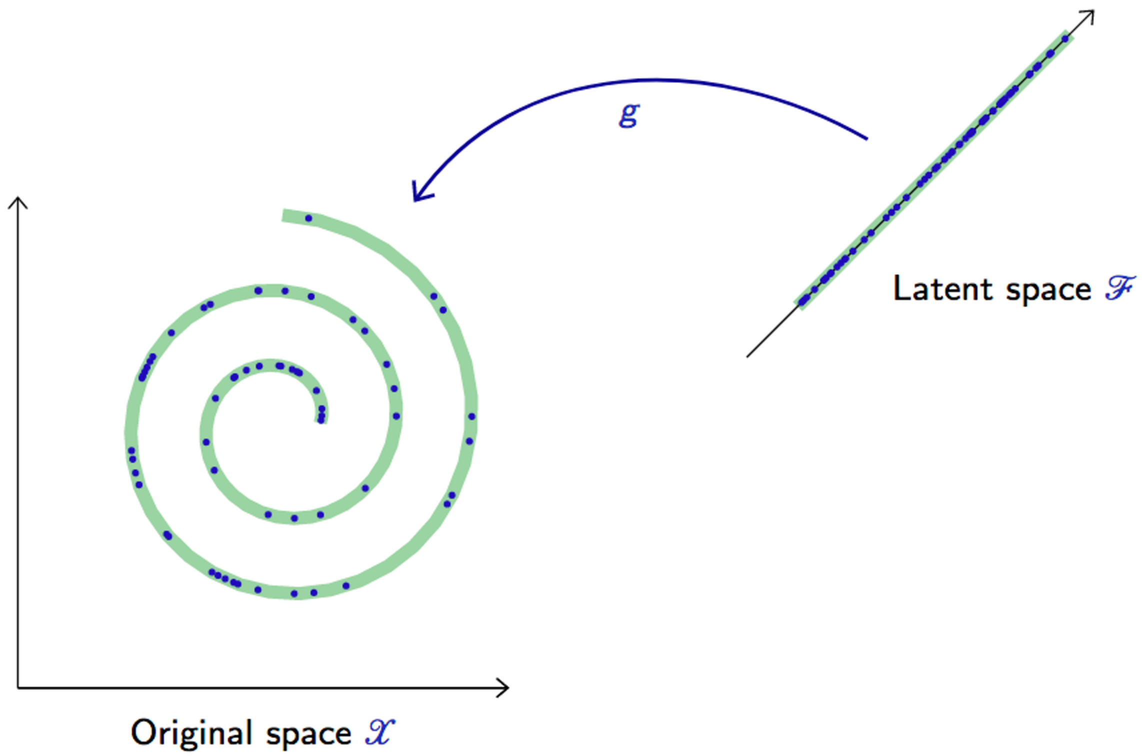
Modeling Data and Meaningful Degrees of Freedom



Modeling Data and Meaningful Degrees of Freedom



Modeling Data and Meaningful Degrees of Freedom



Autoencoders

How to Find Meaningful Representations

- Dimensionality Reduction / Compression
  - Compress the data to a *latent space* with smaller number of dimensions
  - Latent space must encode and retain the important information about the data
  - One way to frame "retaining important information": We can reconstruct original data from latent space
  - Can we learn this compression and latent space?

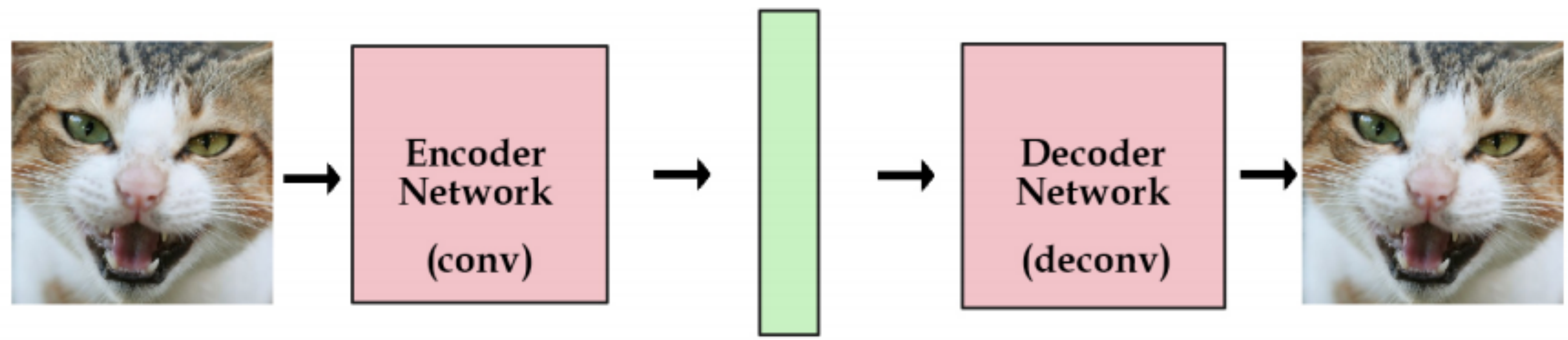
Autoencoders

- Autoencoders map a space to itself through a compression

x = Data z = Latent Space

$$x \to z \to \hat{x}$$

Full transformation should be close to the identity on the data



Autoencoders

- Autoencoders map a space to itself through a compression

x = Data z = Latent Space

$$x \to z \to \hat{x}$$

Full transformation should be close to the identity on the data

$$x \to f(x) = z$$

– **Encoder**: Map from to a lower dimensional latent space
  - Neural network $f_\theta(x)$ with parameters θ

Autoencoders

- Autoencoders map a space to itself through a compression

x = Data z = Latent Space

$$x \to z \to \hat{x}$$

Full transformation should be close to the identity on the data

$$x \to f(x) = z$$

$$z \to g(z) = \hat{x}$$

- **Encoder**: Map from to a lower dimensional latent space
  - Neural network $f_\theta(x)$ with parameters θ

- **Decoder**: Map from latent space back to data space
  - Neural network $g_\psi(z)$ with parameters ψ

Autoencoders

- Autoencoders map a space to itself through a compression

$x =$ Data $\qquad\qquad$ $\mathrm{z} =$ Latent Space

$$x \to z \to \hat{x}$$

Full transformation should be close to the identity on the data

$$x \to f(x) = z$$

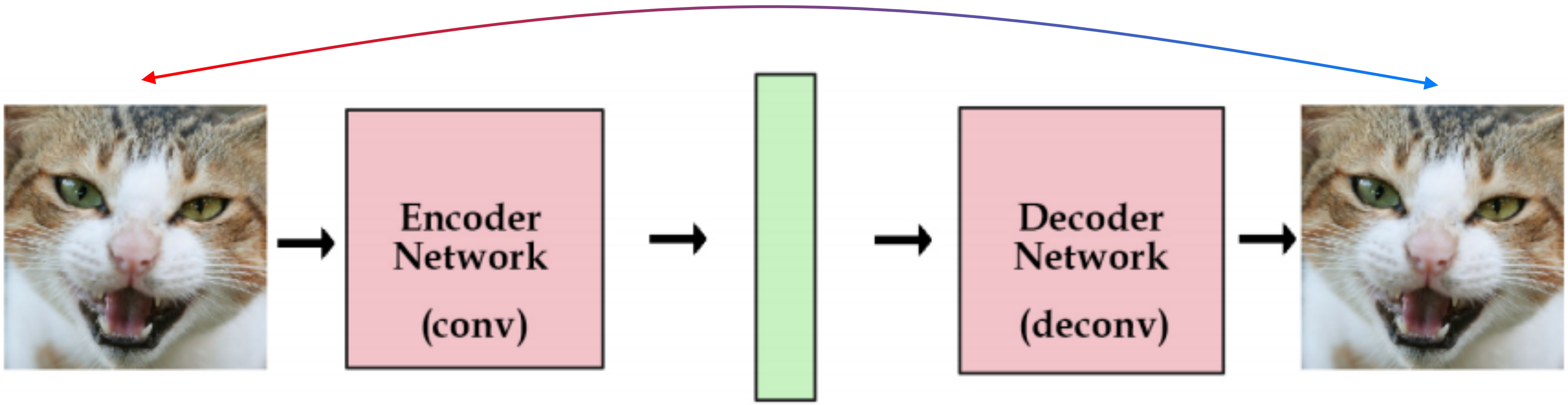
$$z \to g(z) = \hat{x}$$

We must:
- Choose latent dimension D
- Learn mapping $f(\cdot)$ and $g(\cdot)$

– **Encoder**: Map from to a lower dimensional latent space
  - Neural network $f_\theta(x)$ with parameters θ

– **Decoder**: Map from latent space back to data space
  - Neural network $g_\psi(z)$ with parameters ψ

Autoencoder Loss

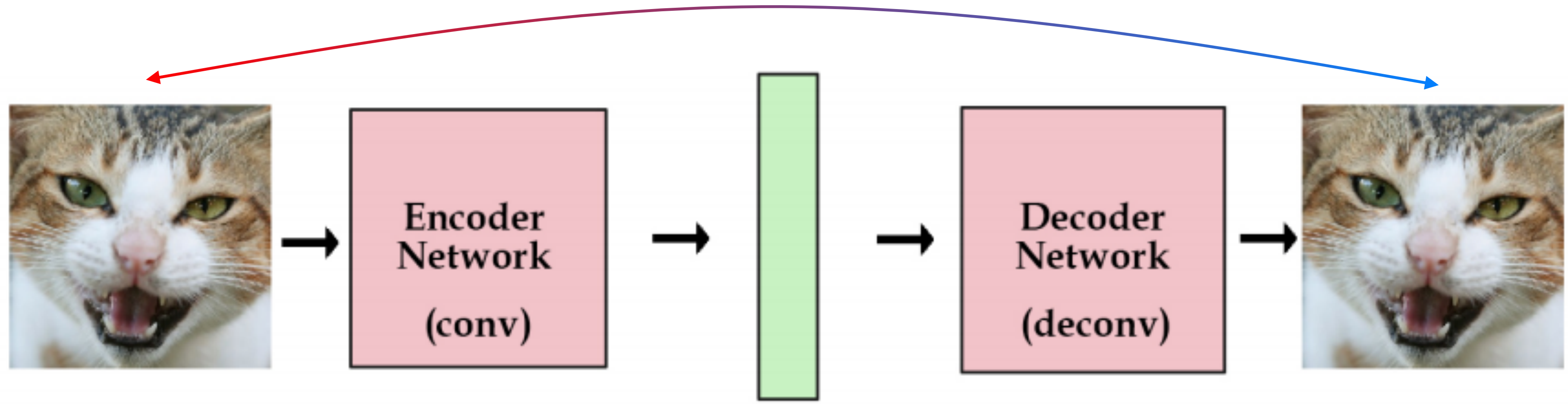


- Loss: mean *reconstruction loss* (MSE) between data and encoded-decoded data

$$L(\theta, \psi) = \frac{1}{N} \sum_n \left\| x_n - g_\psi\left(f_\theta(x_n)\right) \right\|^2$$

- Minimize this loss over parameters of encoder (θ) and decoder (ψ).

Autoencoder Loss



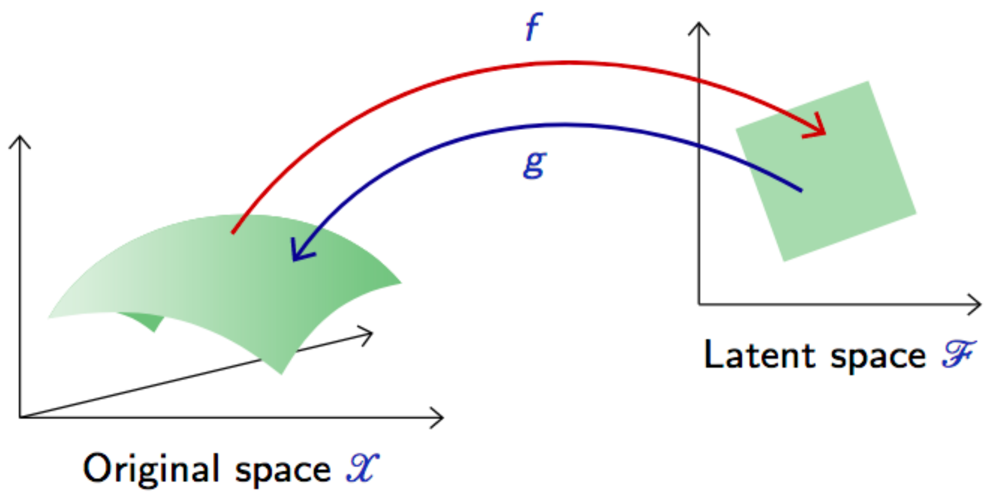
- Loss: mean *reconstruction loss* (MSE) between data and encoded-decoded data

$$L(\theta, \psi) = \frac{1}{N} \sum_n \left\| x_n - g_\psi\left(f_\theta(x_n)\right) \right\|^2$$

- Minimize this loss over parameters of encoder (θ) and decoder (ψ).

NOTE: if $f_\theta(x)$ and $g_\psi(z)$ are linear, optimal solution given by Principle Components Analysis

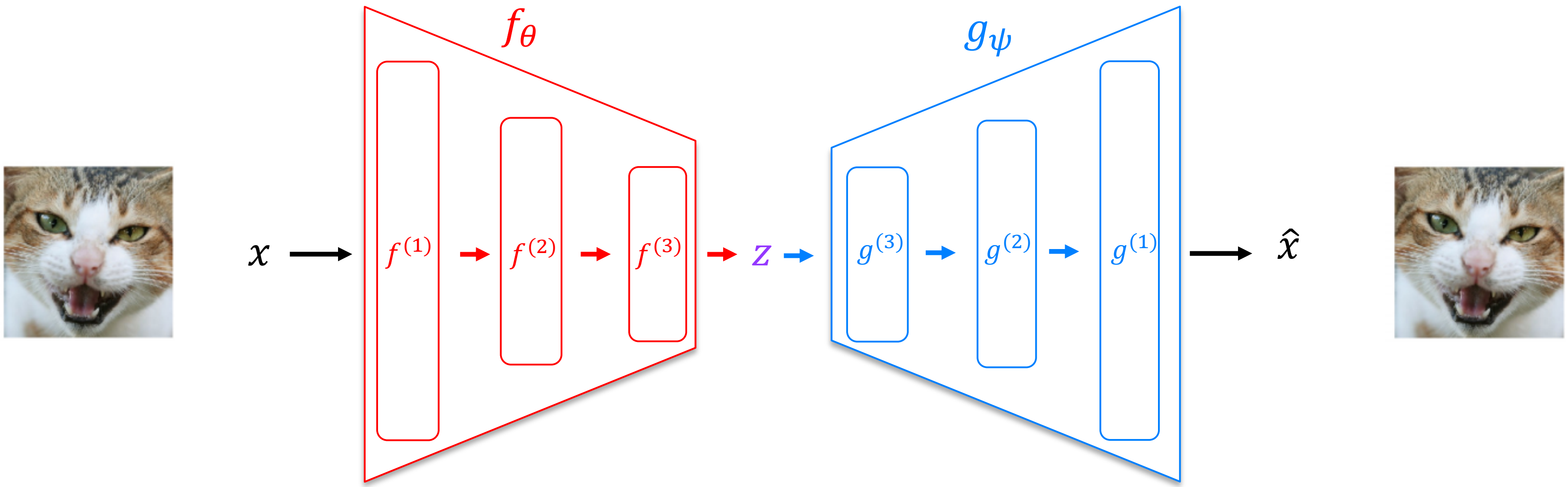
Autoencoder Mappings



If the latent space is of lower dimension:

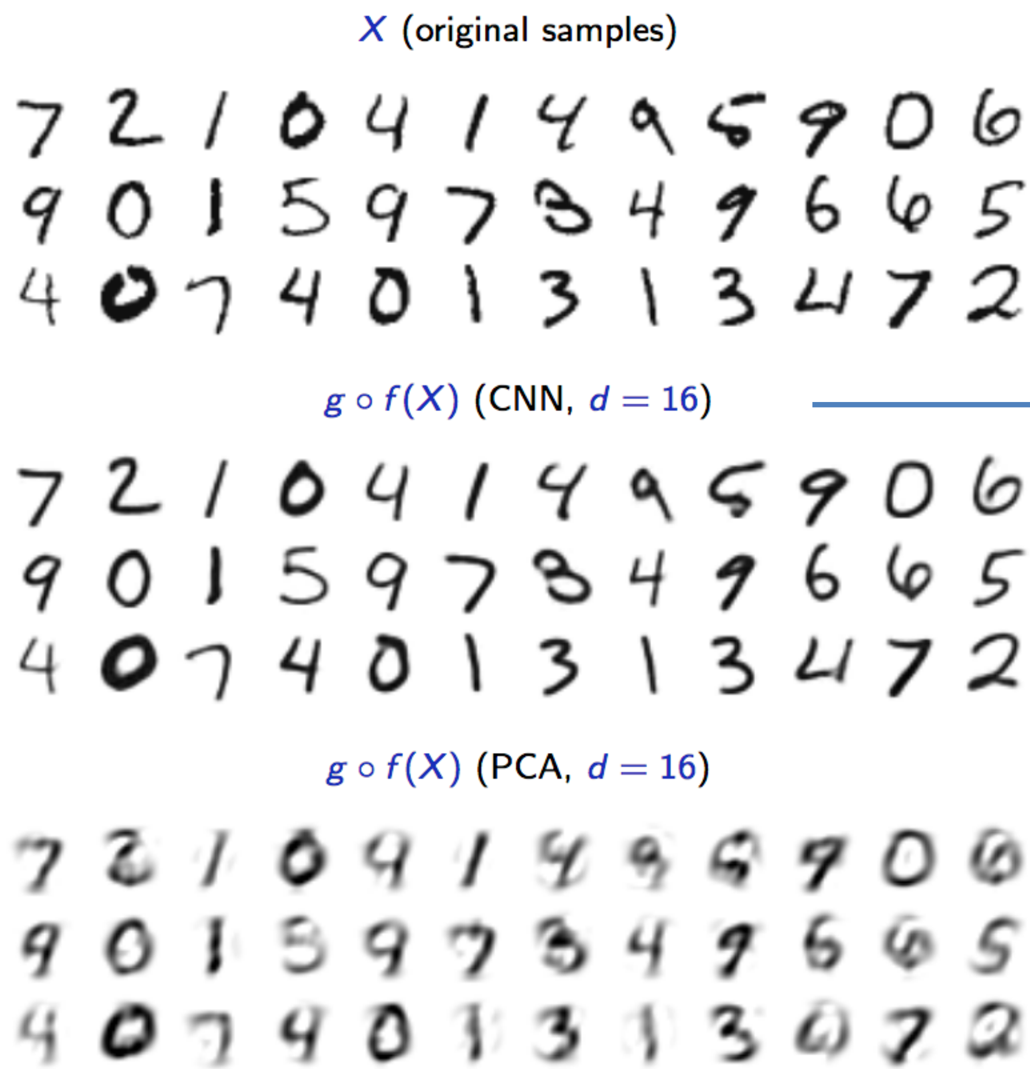
autoencoder must capture a “good” parametrization, and in particular dependencies between components

Deep Autoencoder



- When f_θ and g_ψ are multiple neural network layers, can learn complex mappings between x and z
  - f_θ and g_ψ can be Fully Connected, CNNs, RNNs, etc.
  - Choice of network structure will depend on data

Deep Convolutional Autoencoder



f_θ and g_ψ are each 5 convolutional layers

Fleuret, [Deep Learning Course]

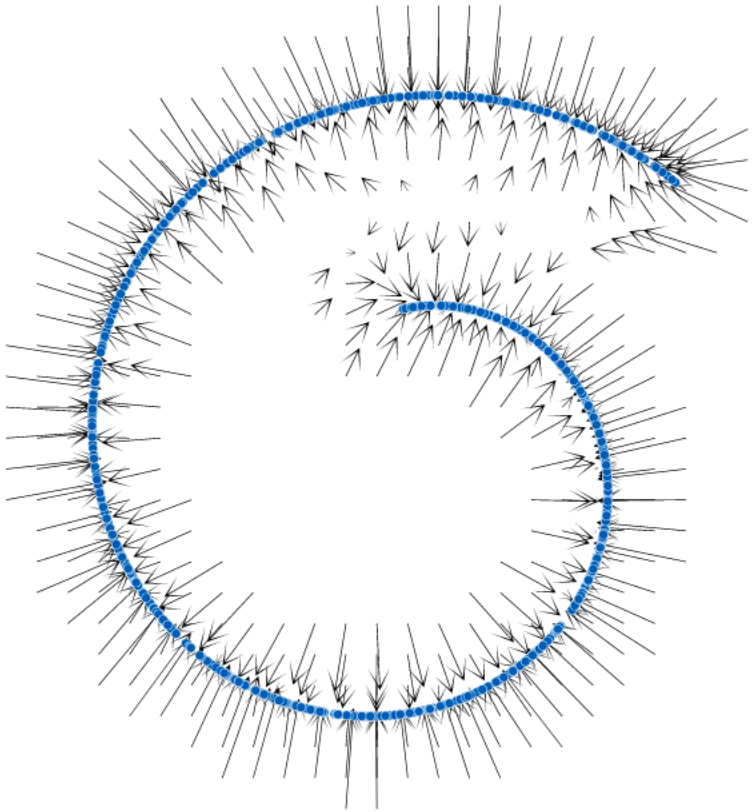
Denoising Autoencoders

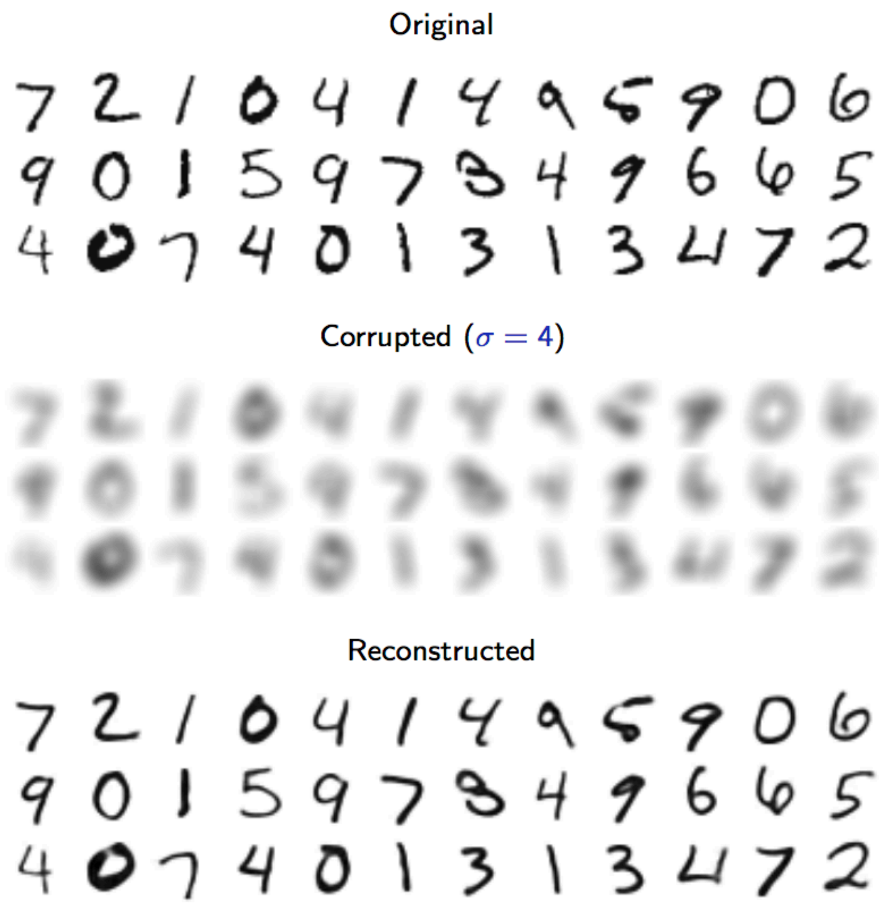
- Learn a mapping from corrupted data space $\tilde{\mathcal{X}}$ back to original data space
  - Mapping $\phi_w(\tilde{\mathcal{X}}) = \mathcal{X}$
  - ϕ_w will be a neural network with parameters w
- Loss:

$$\mathrm{L} = \frac{1}{N}\sum_{n} \| x_n - \phi_w(x_n + \epsilon_n) \|$$

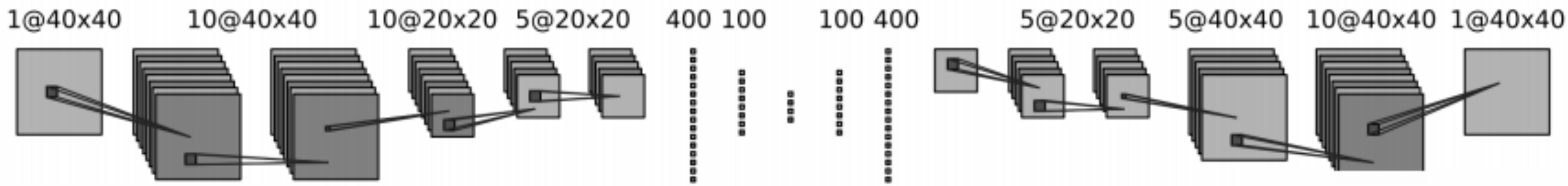
Perturbation, e.g. Gaussian noise

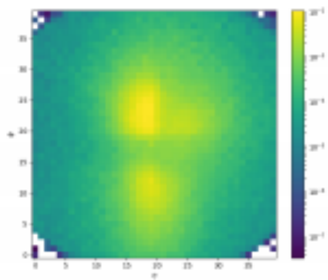
Denoising Autoencoders Examples



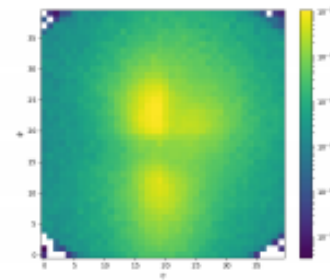


Autoencoders for Anomaly Detection in HEP

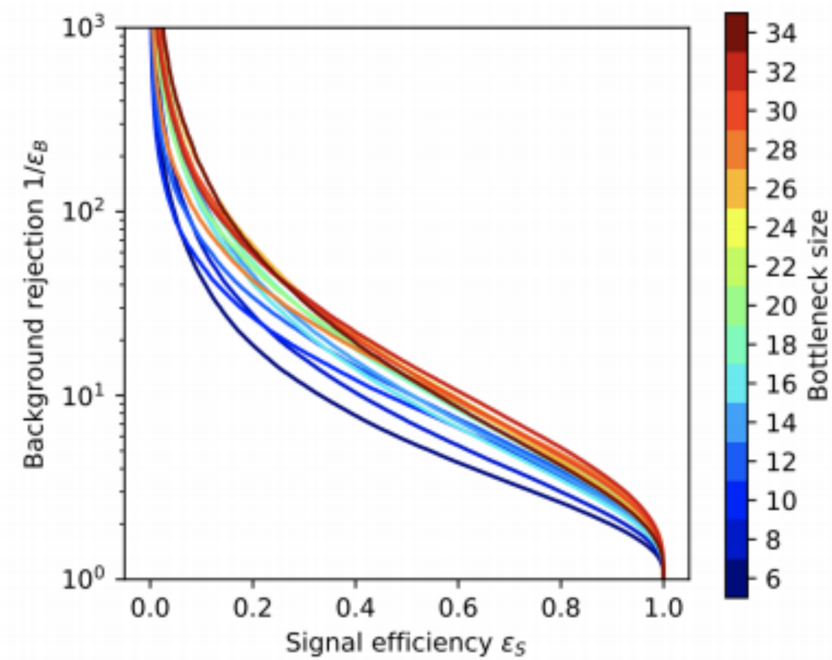




$$L_{\text{auto}} = \sum_{1600 \text{ pixels}} \left(k_T^{\text{norm,in}} - k_T^{\text{auto}} \right)^2$$



- Train on pure QCD light quark/ gluon jets and apply to top tagging
- Top quarks/ new physics identified as anomaly
  if poorly reconstructed by autoencoder

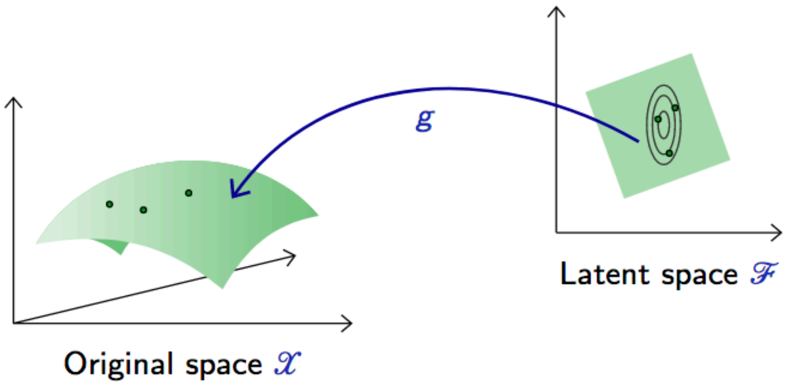


QCD or What?
T Heimel, GK, T Plehn, JM Thompson, 1808.08979
Searching for New Physics with Deep Autoencoders
M Farina, Y Nakai, D Shih, 1808.08992

Slide credit: G. Kasieczka, [Deep Learning in HEP]

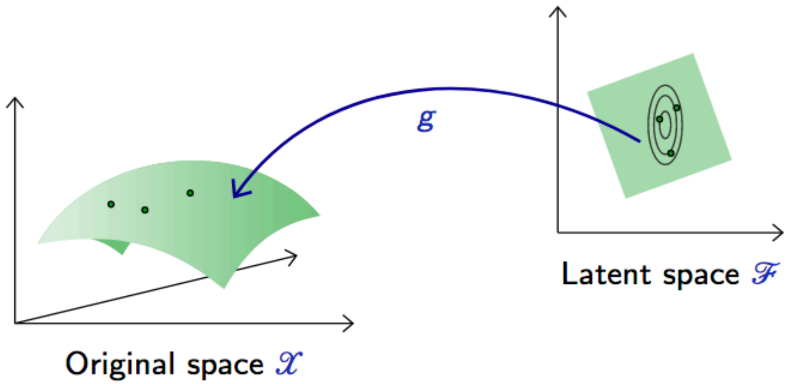
Can We Generate Data with Decoder?

- Can we sample in latent space and decode to generate data?

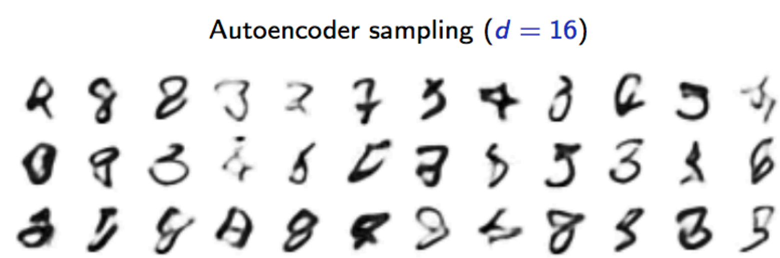


Can We Generate Data with Decoder?

- Can we sample in latent space and decode to generate data?

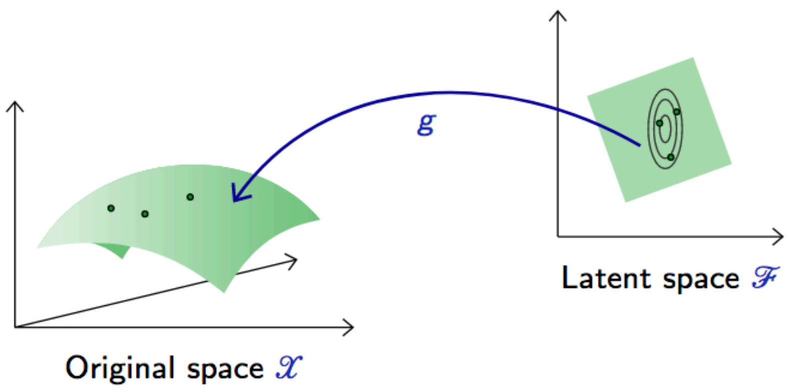


- What distribution to sample from in latent space?
  - Try Gaussian with mean and variance from data

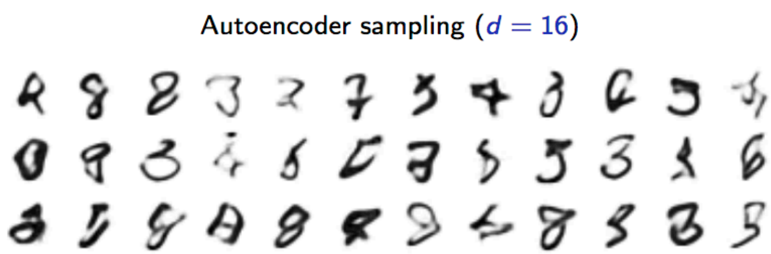


Can We Generate Data with Decoder?

- Can we sample in latent space and decode to generate data?



- What distribution to sample from in latent space?
  - Try Gaussian with mean and variance from data



- Doesn't work! Don't know the right latent space density
  - This can be done with a Variational Autoencoder (See Backup)

Fleuret, [Deep Learning Course]()

Generative Models

- Generative models aim to:
  - Learn distribution $p(x)$ that models PDF of the data
  - Draw samples of plausible data points

- Explicit Models
  - Can evaluate the density $p(x)$ of a data point x

- Implicit Models
  - Can only sample from $p(x)$, but not evaluate density

Generative Adversarial Networks

Generative Modeling as a Two Player Game

- Formulate generative modeling task as a two player game
- One player tries to output data that looks as real as possible
- Another player tries to compare real and fake data
- In this case we need:
  - A *generator* that can produce samples
  - A measure of *not too far from the real data*

Generative Adversarial Network (GAN)

Goodfellow et. al., 2014

- **Generator network $g_\theta(z)$** with parameters θ
  - Map sample from known $p(z)$ to sample in data space

$$x = g_\theta(z) \quad z \sim p(z)$$

  - We don't know what the learned distribution $p_\theta(x)$ is, but we can sample from it → *Implicit Model*

Generative Adversarial Network (GAN)

Goodfellow et. al., 2014

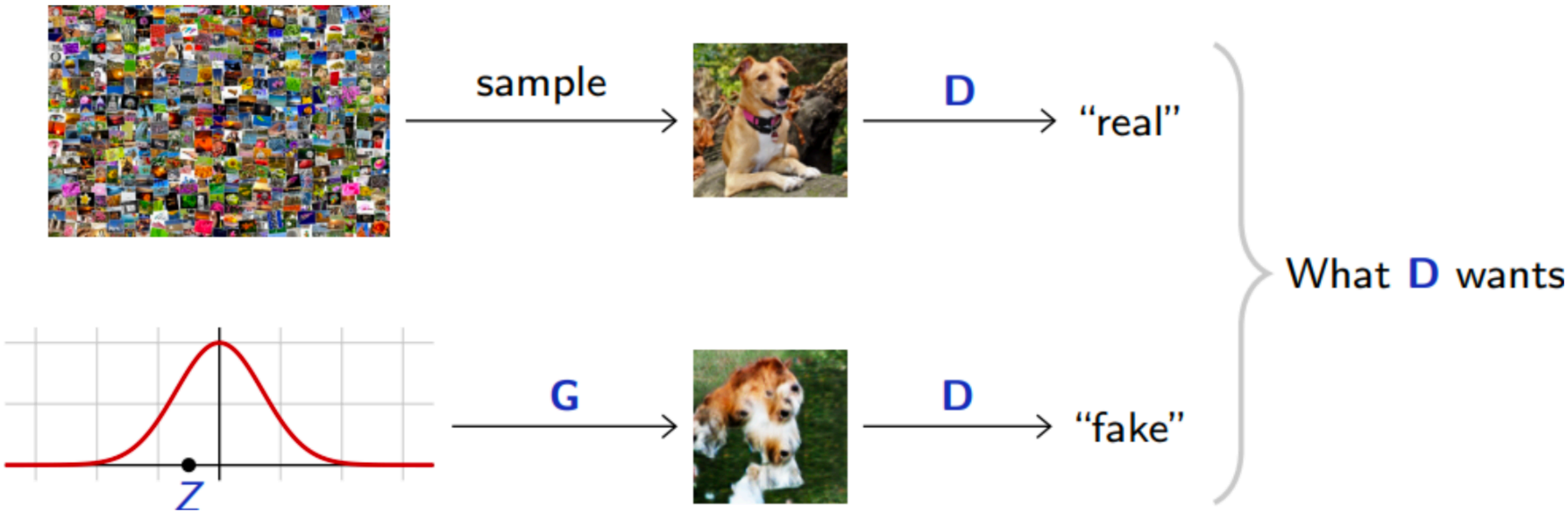
- **Generator network $g_{\theta}(z)$** with parameters θ
  - Map sample from known $p(z)$ to sample in data space

$$x = g_{\theta}(z) \quad z \sim p(z)$$

  - We don't know what the learned distribution $p_{\theta}(x)$ is, but we can sample from it → *Implicit Model*

- **Discriminator Network $d_{\phi}(x)$** with parameters ϕ
  - Classifier trained to distinguish between real and fake data
  - Classifier is learning to predict $p(input = real \mid x)$
  - Classifier is our measure of *not too far from the real data*

GAN Setup



- Generator goal:
  - Produce *fake* data to trick discriminator to classify as *real*
- Discriminator goal:
  - Minimizes miss-classification of data as real or fake
- *Adversarial* setup: two networks w/ opposing objectives

Fleuret, Deep Learning Course

GAN Objective

- Data
  - Real data samples: $\{x_i, y_i = 1\}$
  - Fake data samples: $\{\tilde{x}_i = g_\theta(z_i), \tilde{y}_i = 0\}$ with: $z_i \sim p(z)$

Usually Gaussian $\mathcal{N}(0,1)$

GAN Objective

- Data
  - Real data samples: $\{x_i, y_i = 1\}$
  - Fake data samples: $\{\tilde{x}_i = g_\theta(z_i), \tilde{y}_i = 0\}$ with: $z_i \sim p(z)$
- For a fixed generator, can train discriminator by minimizing the binary cross entropy

$$L(\phi) = -\frac{1}{2N}\sum_{i=1}^{N}\left[y_i \log d_\phi(x_i) + (1 - \tilde{y}_i)\log(1 - d_\phi(\tilde{x}_i))\right]$$

GAN Objective

- Data
  - Real data samples: $\{x_i, y_i = 1\}$
  - Fake data samples: $\{\tilde{x}_i = g_\theta(z_i), \tilde{y}_i = 0\}$ with: $z_i \sim p(z)$
- For a fixed generator, can train discriminator by minimizing the binary cross entropy

$$L(\phi) = -\frac{1}{2N}\sum_{i=1}^{N}\left[y_i \log d_\phi(x_i) + (1-\tilde{y}_i)\log(1-d_\phi(\tilde{x}_i))\right]$$

$$= -\mathbb{E}_{x\sim p_{data}(x)}\left[\log d_\phi(x_i)\right] - \mathbb{E}_{z\sim p(z)}\left[\log(1-d_\phi(g_\theta(z)))\right]$$

GAN Objective

- Data
  - Real data samples: $\{x_i, y_i = 1\}$
  - Fake data samples: $\{\tilde{x}_i = g_\theta(z_i), \tilde{y}_i = 0\}$ with: $z_i \sim p(z)$
- For a fixed generator, can train discriminator by minimizing the binary cross entropy

$$L(\phi) = -\frac{1}{2N}\sum_{i=1}^{N}\left[y_i \log d_\phi(x_i) + (1-\tilde{y}_i)\log(1-d_\phi(\tilde{x}_i))\right]$$

$$= -\mathbb{E}_{x\sim p_{data}(x)}\left[\log d_\phi(x_i)\right] - \mathbb{E}_{z\sim p(z)}\left[\log(1-d_\phi(g_\theta(z)))\right]$$

- Generator isn’t fixed → Must be trained

GAN Objective

- Consider objective as a *value function* of ϕ and θ

$$V(\phi, \theta) = \mathbb{E}_{x \sim p_{\text{data}}(x)} \Big[\log d_\phi(x) \Big] + \mathbb{E}_{z \sim p(z)} \Big[\log(1 - d_\phi(g_\theta(z)) \ ) \Big]$$

GAN Objective

- Consider objective as a *value function* of ϕ and θ

$$V(\phi, \theta) = \mathbb{E}_{x \sim p_{\text{data}}(x)}\Big[\log d_\phi(x)\Big] + \mathbb{E}_{z \sim p(z)}\Big[\log(1 - d_\phi(g_\theta(z)))\Big]$$

  – For fixed generator, $V(\phi, \theta)$ is high when discriminator is good, i.e. when generator is not producing good fakes

GAN Objective

- Consider objective as a *value function* of ϕ and θ

$$V(\phi,\theta) = \mathbb{E}_{x\sim p_{\text{data}}(x)}\Big[\log d_\phi(x)\Big] + \mathbb{E}_{z\sim p(z)}\Big[\log(1 - d_\phi(g_\theta(z))\ )\Big]$$

  - For fixed generator, $V(\phi,\theta)$ is high when discriminator is good, i.e. when generator is not producing good fakes
  - For perfect discriminator, $V(\phi,\theta)$ is low when generator is good, i.e. when generator confuses discriminator

GAN Objective

- Consider objective as a *value function* of ϕ and θ

$$V(\phi, \theta) = \mathbb{E}_{x \sim p_{\text{data}}(x)}\Big[\log d_\phi(x)\Big] + \mathbb{E}_{z \sim p(z)}\Big[\log(1 - d_\phi(g_\theta(z))\ )\Big]$$

  - For fixed generator, $V(\phi, \theta)$ is high when discriminator is good, i.e. when generator is not producing good fakes
  - For perfect discriminator, $V(\phi, \theta)$ is low when generator is good, i.e. when generator confuses discriminator

- So our optimization goal becomes:

$$\theta^* = \arg\min_\theta \max_\phi V(\phi, \theta)$$

GAN Objective

- Consider objective as a *value function* of ϕ and θ

$$V(\phi, \theta) = \mathbb{E}_{x \sim p_{\text{data}}(x)}\Big[\log d_\phi(x)\Big] + \mathbb{E}_{z \sim p(z)}\Big[\log(1 - d_\phi(g_\theta(z)))\Big]$$

  - For fixed generator, $V(\phi, \theta)$ is high when discriminator is good, i.e. when generator is not producing good fakes
  - For perfect discriminator, $V(\phi, \theta)$ is low when generator is good, i.e. when generator confuses discriminator

- So our optimization goal becomes:

$$\theta^* = \arg\min_\theta \max_\phi V(\phi, \theta)$$

NOTE: can prove that minimax solution corresponds to generator that perfectly reproduces data distribution

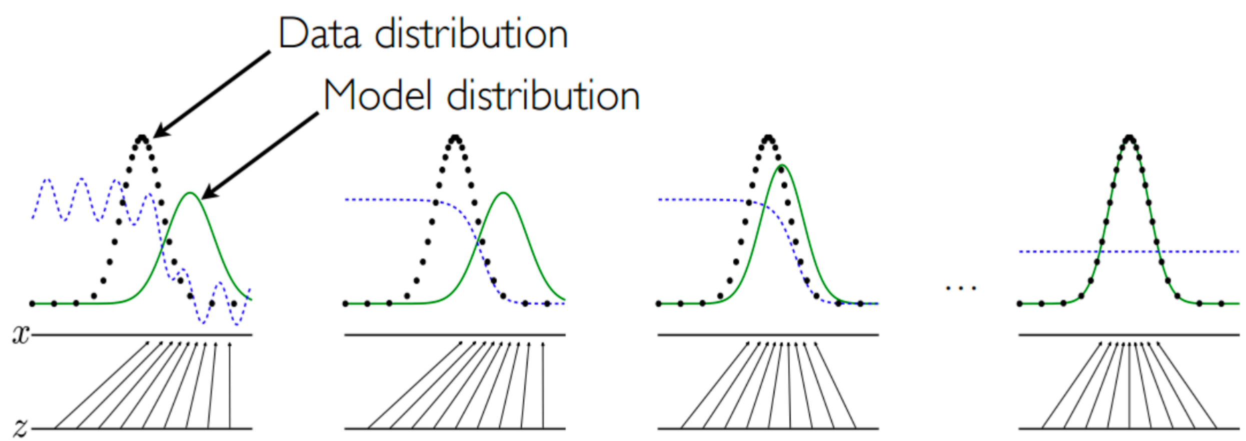
GAN Training

- Alternating Gradient descent to solve the min-max problem:

$$\theta \leftarrow \theta - \gamma \nabla_\theta V(\phi, \theta) = \theta - \gamma \frac{\partial V}{\partial d} \frac{\partial (d_\phi)}{\partial g} \frac{\partial g_\theta}{\partial \theta}$$

$$\phi \leftarrow \phi - \gamma \nabla_\phi V(\phi, \theta) = \phi - \gamma \frac{\partial V}{\partial d} \frac{d(d_\phi)}{d\phi}$$

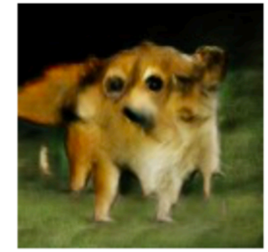
- For each θ step, take k steps in ϕ to keep discriminator near optimal

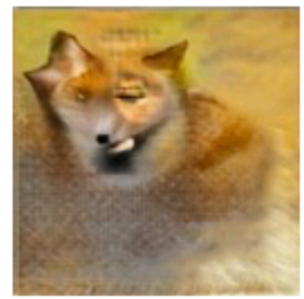


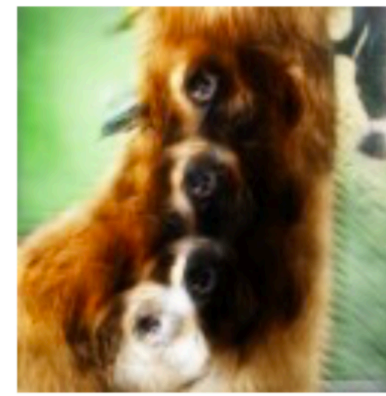
Poorly fit model After updating D After updating G Mixed strategy equilibrium

Examples

Goodfellow et. al., [2014]





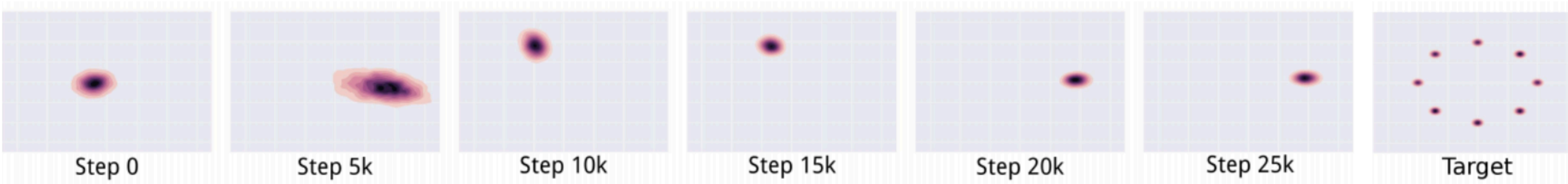


Not so good

Goodfellow 2016

Radford et al, 2015

Challenges



- **Oscillations without convergence**: unlike standard loss minimization, alternating stochastic gradient descent has no guarantee of convergence.

- **Vanishing gradients**: if classifier is too good, value function saturates → no gradient to update generator

- **Mode collapse**: generator models only a small sub-population, concentrating on a few data distribution modes.

- **Difficult to assess performance:** is generated data good enough?

- **Improvements** in training objective (WGAN) and model design have significantly helped with these challenges

Slide credit: G. Louppe

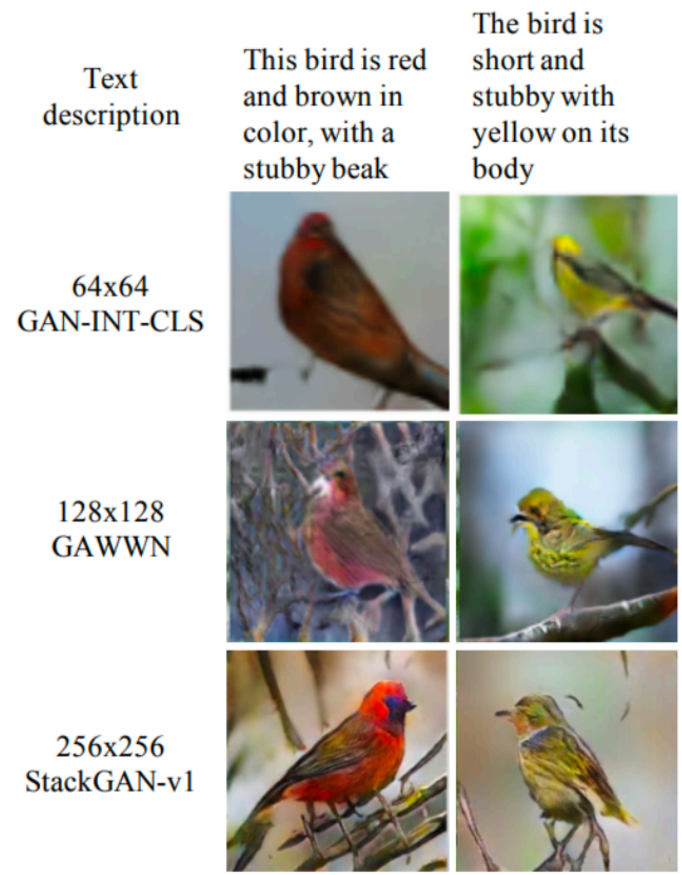
Mode collapse (Metz et al, 2016)

GAN Models

StyleGAN v2

(Karras et al, 2019)

Text-to-Image Synthesis with StackGAN

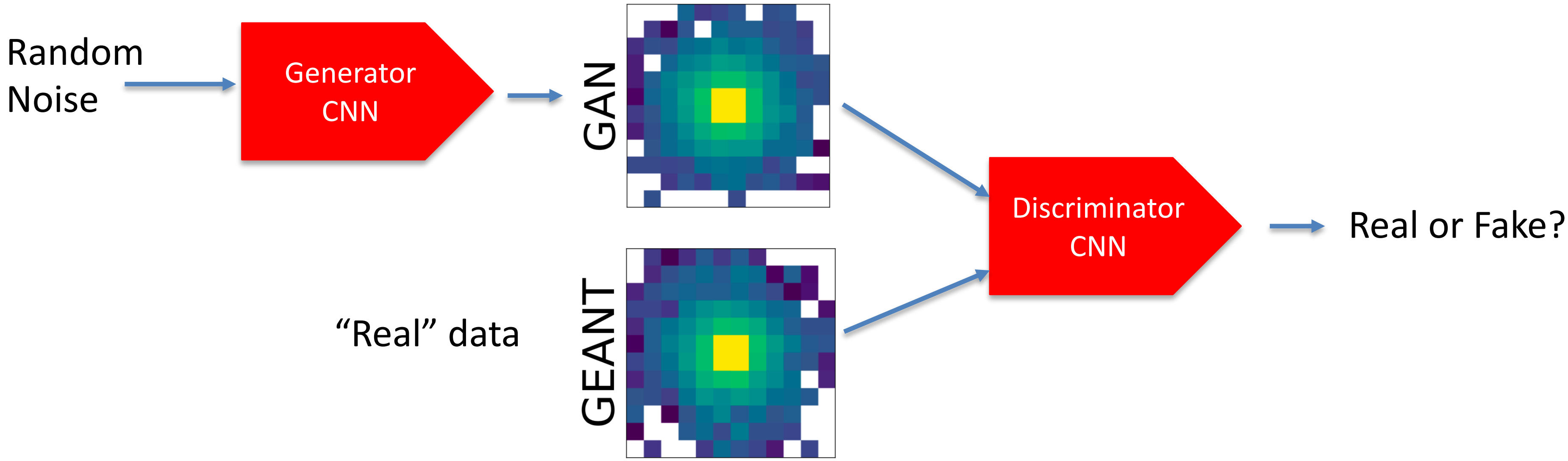


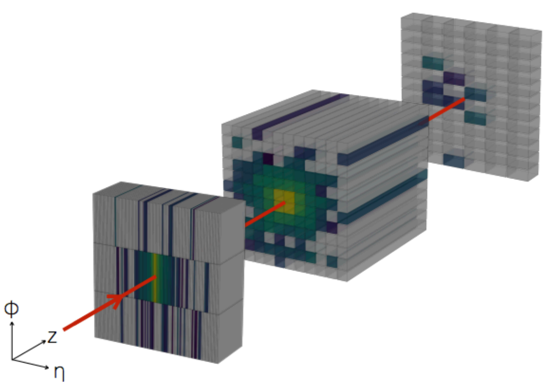
Zhang et. al. 2017

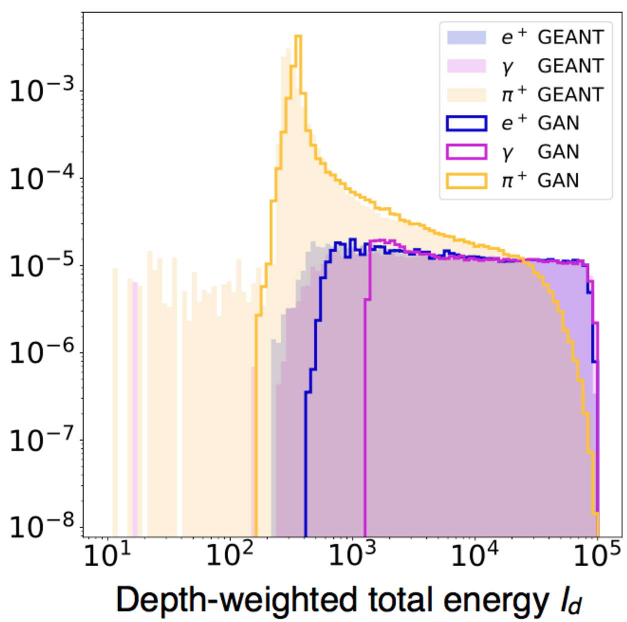
Image-to-Image Translation with CycleGAN

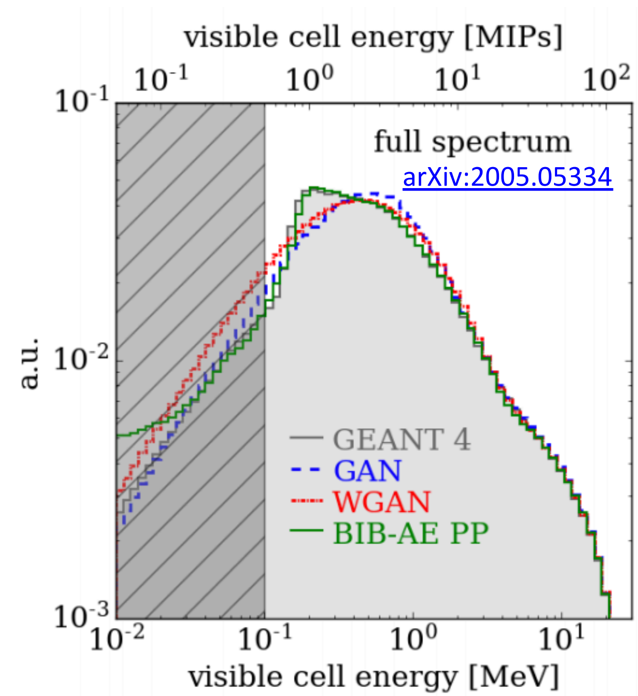
[Zhu et. al. 2017]

GANs for Calorimeter Energy Depositions







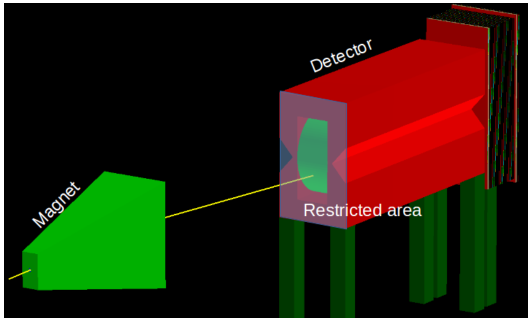


[PRD97, 014021 (2018)]
arXiv:1705.02355
arXiv:1701.05927

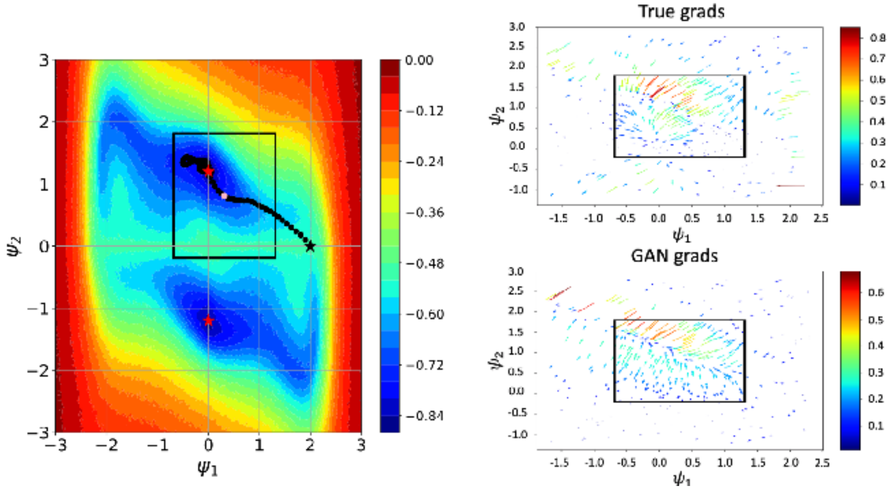
GANs for Detector Design

NeurIPS **33**, 14650-14662 (2020)

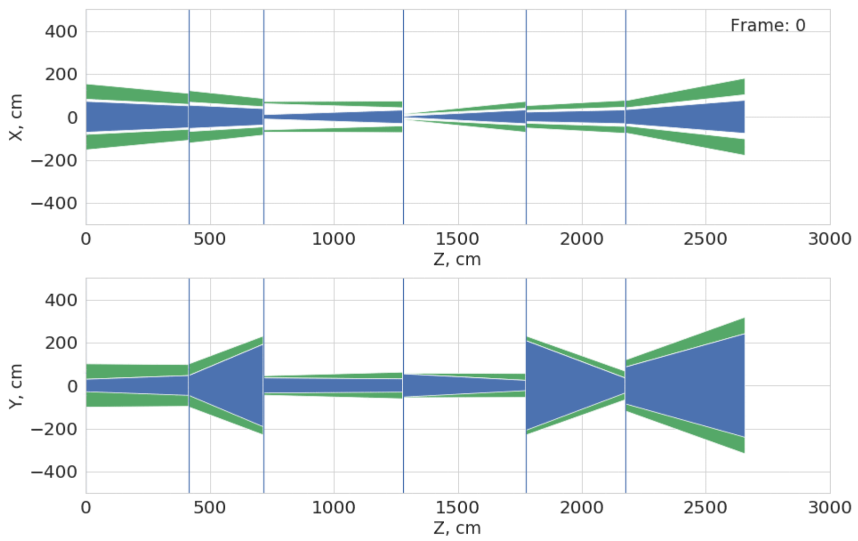
- Train GAN to emulate data from simulated detector, $\tilde{x} = g(z|\psi)$ conditioned on detector parameters ψ (e.g. magnet shape below)
- Define objective C to minimize: $\min_{\psi} \mathbb{E}_{\tilde{x}}[C(\tilde{x} = g(z|\psi))]$
- GAN is differentiable
  - Minimize with gradient descent



Toy Example

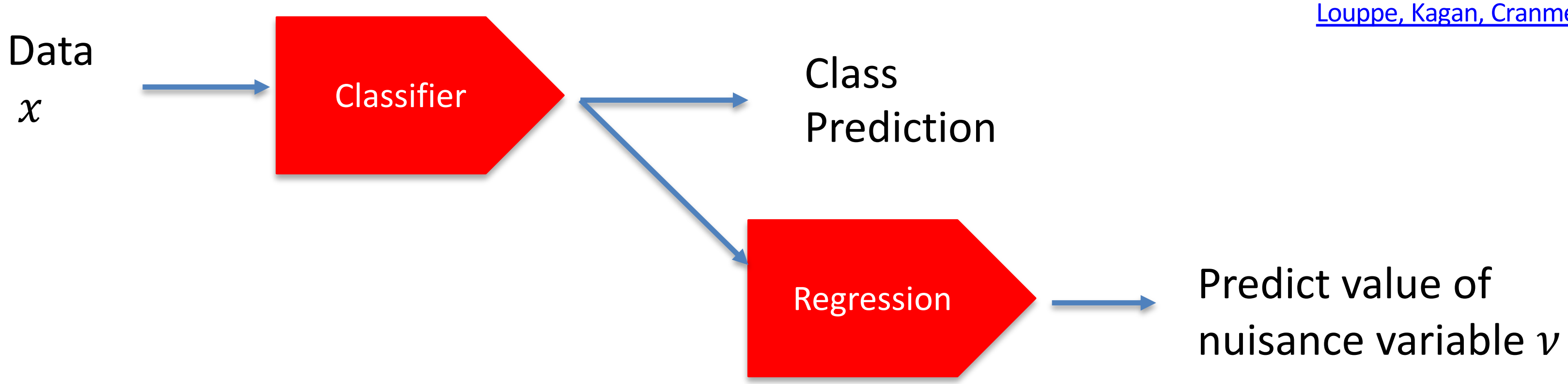


Magnet Optimization

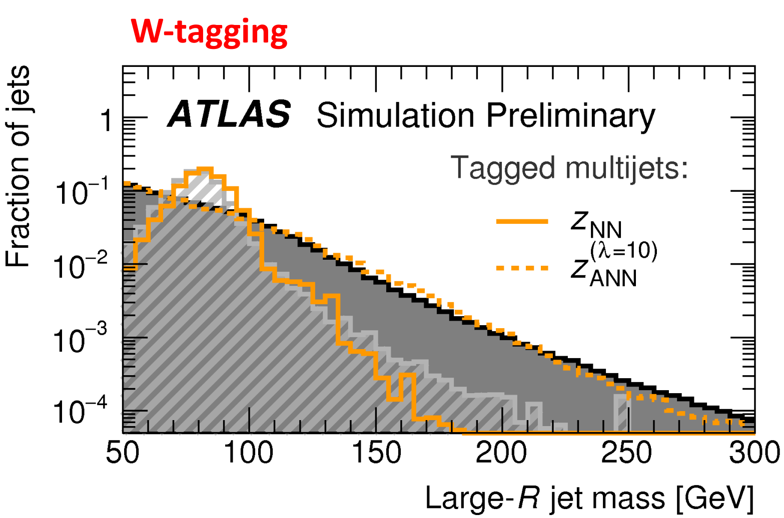


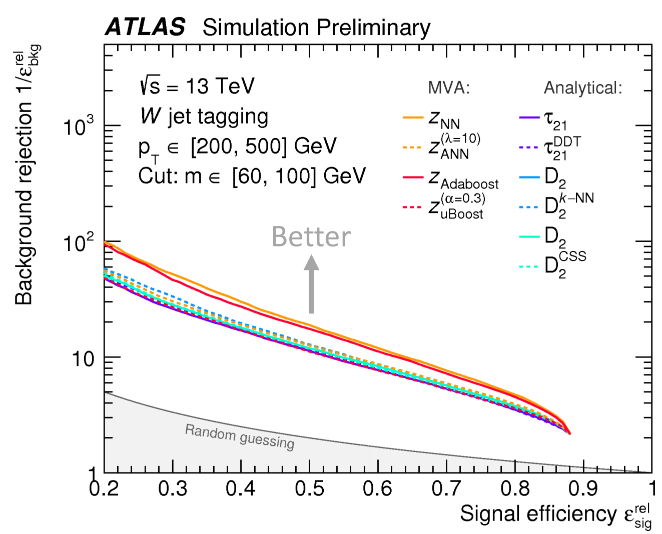
Adversarial Learning for Constraining Dependence

Louppe, Kagan, Cranmer, NeurIPS 30 (2017)



- Want to remove dependence of classifier on "nuisance" variable ν, e.g. a systematic, mass, etc.





ATL-PHYS-PUB-2018-014

Explicit Density Estimation with Normalizing Flows

- GAN can only learn to sample from a distribution

- Is there a way to learn the explicit density $p(x)$?

Reminder: Calculus Change of Variables

$$\int f\left(g(x)\right)\frac{\partial g(x)}{dx}dx = \int f(u)du \qquad \text{where } u = g(x)$$

Multivariate:

$$\int f\left(g(\boldsymbol{x})\right)\left|\det\frac{\partial g(\boldsymbol{x})}{d\boldsymbol{x}}\right|d\boldsymbol{x} = \int f(\boldsymbol{u})d\boldsymbol{u} \quad \text{where } \boldsymbol{u} = g(\boldsymbol{x})$$

Change of volume:
Determinant of Jacobian
of the transformation

Change of Variables in Probability

- If $f(x)$ is the pdf of x and $y(x)$ is a change of variables, since probability should not change:

$$P(x_a < x < x_b) = P(y(x_a) < y < y(x_b))$$

$$\int_{x_a}^{x_b} f(x)dx = \int_{y(x_a)}^{y(x_b)} g(y)dy$$

$$= \int_{x_a}^{x_b} g(y(x)) \left|\frac{dy}{dx}\right| dx$$

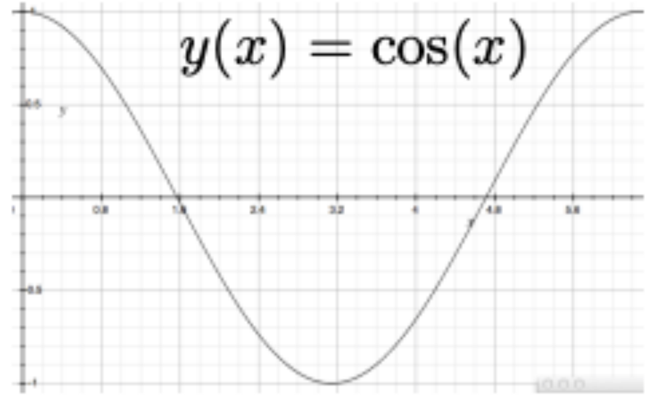
Rewrite of r.h.s.

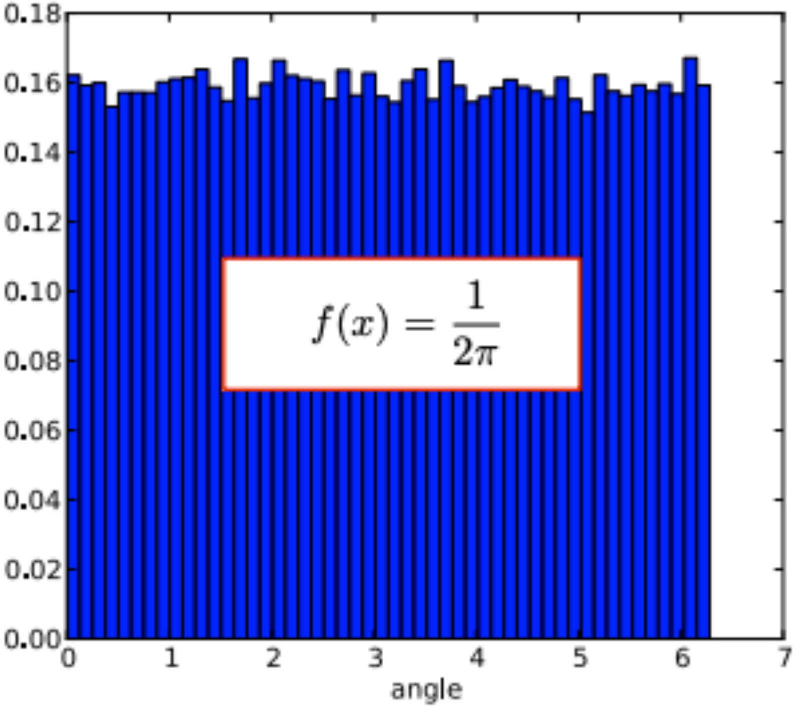
Thus: $$\boxed{f(x) = g(y) \left|\frac{dy}{dx}\right|}$$

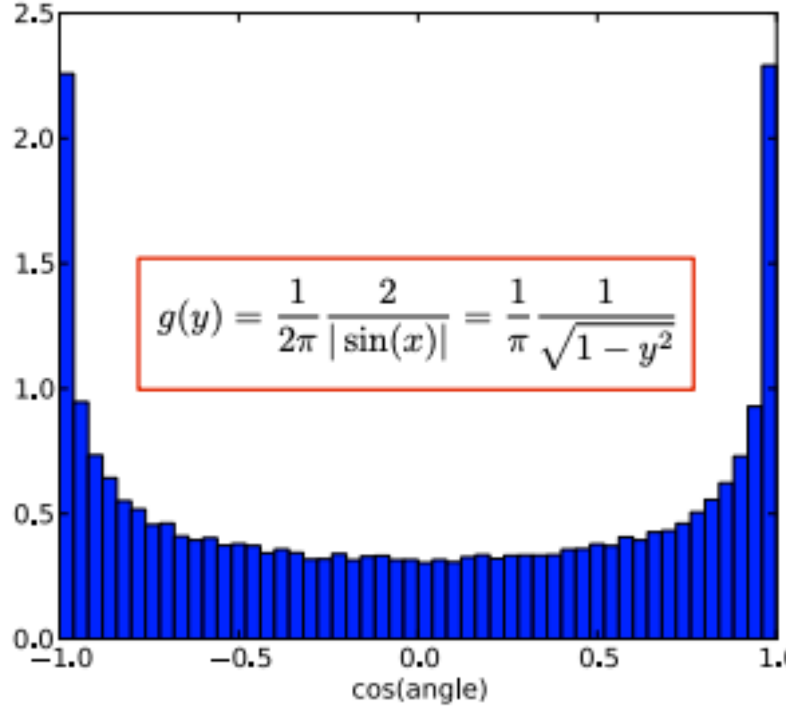
Distributions are related through the Jacobian

Example

$$f(x) = g(y)\left|\frac{dy}{dx}\right|$$







Slide Credit: K. Cranmer: [Intro to Stats.]

Change of Variables with Neural Networks

$$p_x(\boldsymbol{x}) = p_z(\boldsymbol{z}) \left| \det\left(\frac{\partial \phi(\boldsymbol{z})}{d\boldsymbol{z}}\right)^{-1} \right| \quad \text{where } \boldsymbol{x} = \phi(\boldsymbol{z})$$

- $x \equiv$ data we want to model
- $z \sim p(x)$ is a chosen noise distribution, usually Gaussian

- ϕ is continuous, invertible, differentiable, $\mathrm{z} = \phi^{-1}(x)$

Change of Variables with Neural Networks

$$p_x(\boldsymbol{x}) = p_z(\boldsymbol{z}) \left| \det \left(\frac{\partial \phi(\boldsymbol{z})}{d\boldsymbol{z}} \right)^{-1} \right| \quad \text{where } \boldsymbol{x} = \phi(\boldsymbol{z})$$

- $x \equiv$ data we want to model
- $z \sim p(x)$ is a chosen noise distribution, usually Gaussian

- ϕ is continuous, invertible, differentiable, $\mathrm{z} = \phi^{-1}(x)$

- Want to find $\phi(z)$ that transforms data $x \Leftrightarrow$ noise $z \sim p(z)$

Change of Variables with Neural Networks

$$p_x(\boldsymbol{x}) = p_z(\boldsymbol{z}) \left|\det\left(\frac{\partial\phi(\boldsymbol{z})}{d\boldsymbol{z}}\right)^{-1}\right| \quad \text{where } \boldsymbol{x} = \phi(\boldsymbol{z})$$

- $x \equiv$ data we want to model
- $z \sim p(x)$ is a chosen noise distribution, usually Gaussian

- ϕ is continuous, invertible, differentiable, $\mathrm{z} = \phi^{-1}(x)$

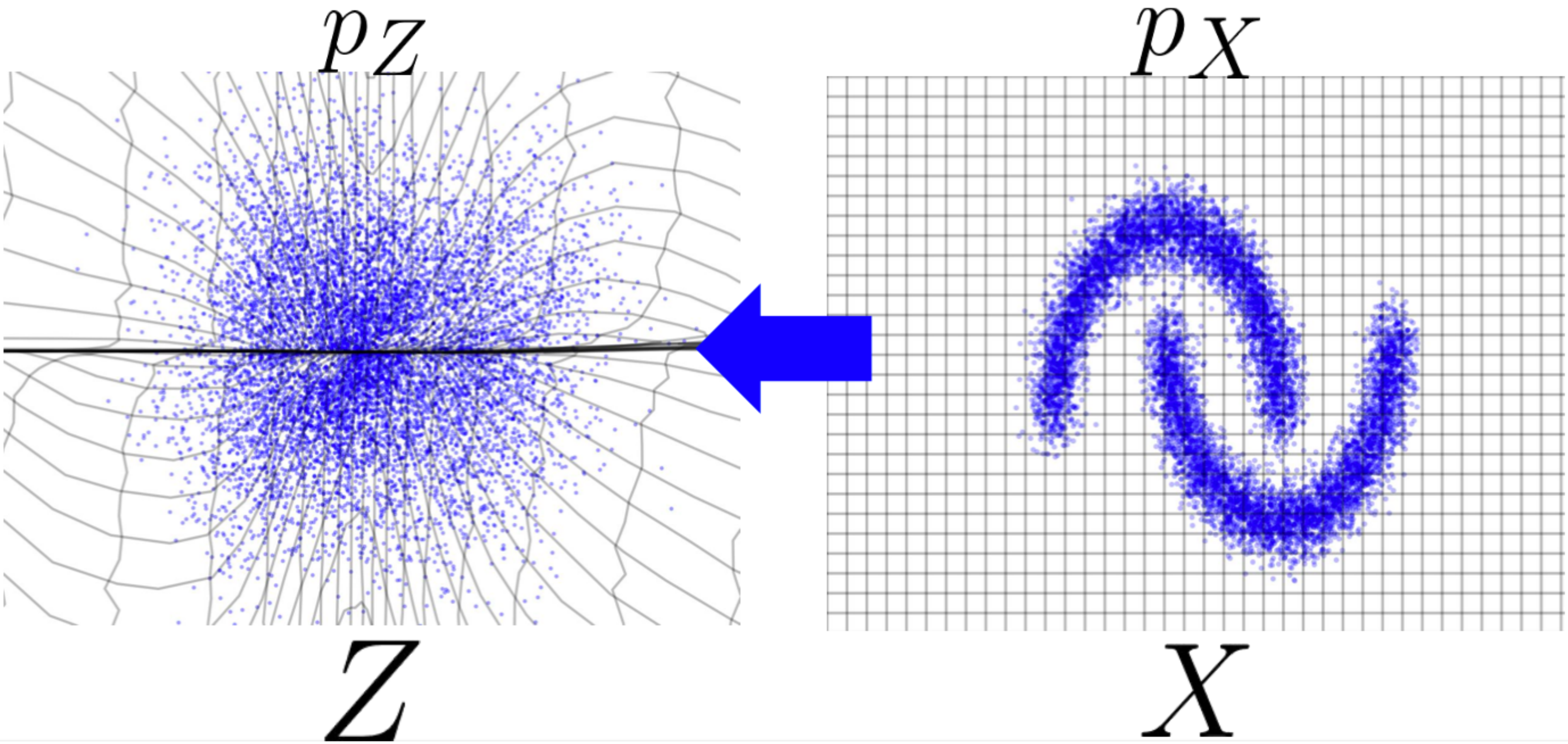
- Want to find $\phi(z)$ that transforms data $x \Leftrightarrow$ noise $z \sim p(z)$

$\phi(\boldsymbol{z})$ neural network
- Input = a sample of noise
- Output = a sample of X

$\Leftrightarrow$

$\phi^{-1}(\boldsymbol{x})$ inverse
- Input = a sample X
- Output = a sample of noise

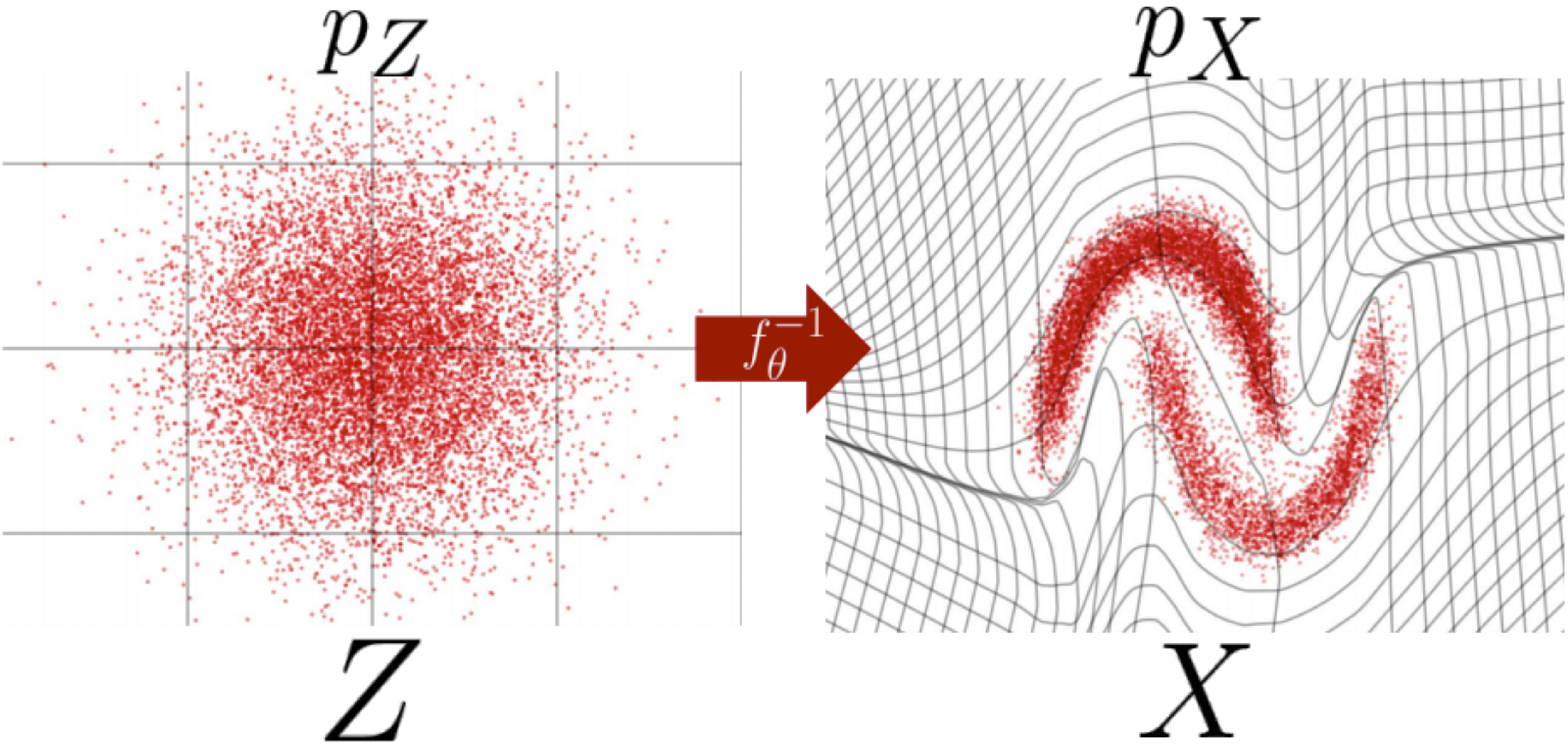
Density Estimation



Slide credit: [L. Dinh]

Generation

Generation through process reversion



Slide credit: [L. Dinh]

Example: Real NVP (Non-Volume Preserving) Flow

- Data vector $x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$
- **Transformation**: where $f(\cdot)$ and $g(\cdot)$ are neural networks

$$\phi(z): \quad \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} \phi_1(z) = z_1 \\ \phi_2(z) = z_2 f(z_1) + g(z_1) \end{pmatrix}$$

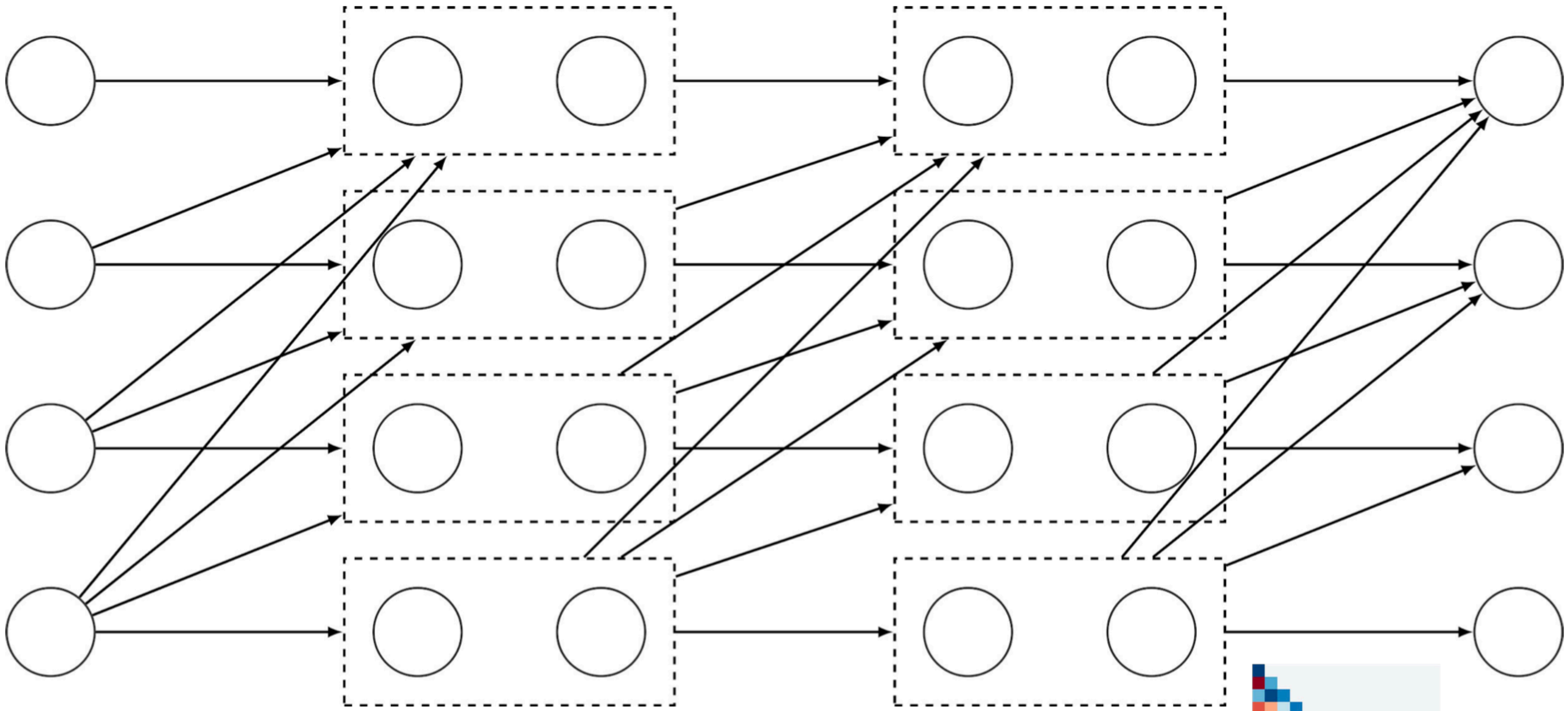
$$\phi^{-1}(x): \quad \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} \phi_1^{-1}(x) = x_1 \\ \phi_2^{-1}(x) = \frac{x_2 - g(x_1)}{f(x_1)} \end{pmatrix}$$

- **Determinant:** Use fact that Jacobian is lower triangular

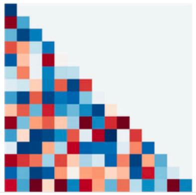
$$\det\left(\frac{\partial\phi(\boldsymbol{z})}{d\boldsymbol{z}}\right) = \det\begin{pmatrix} 1 & 0 \\ \left(\frac{\partial\phi_2(z)}{dz_1}\right) & f(z_1) \end{pmatrix} = \prod Diag\left(\frac{\partial\phi(\boldsymbol{z})}{d\boldsymbol{z}}\right) = f(z_2)$$

Neural Autoregressive Models

(Bengio[2], 1999; Larochelle & Murray, 2011; van den Oord et al., 2015; Uria et al., 2016)



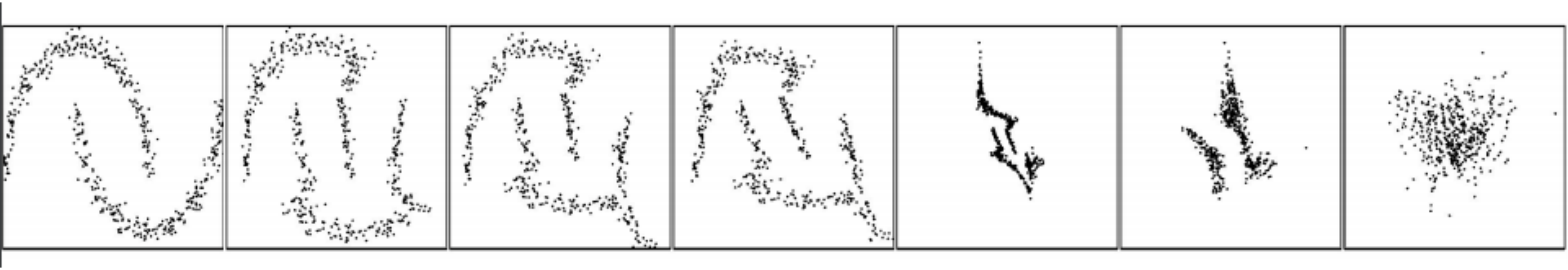
$$z_d = f_d(x_d; x_{<d})$$



$$x_d = f_d^{-1}(z_d; x_{<d})$$

Slide credit: [L. Dinh]()

Composing Flows



$$f_3 \circ f_2 \circ f_1$$

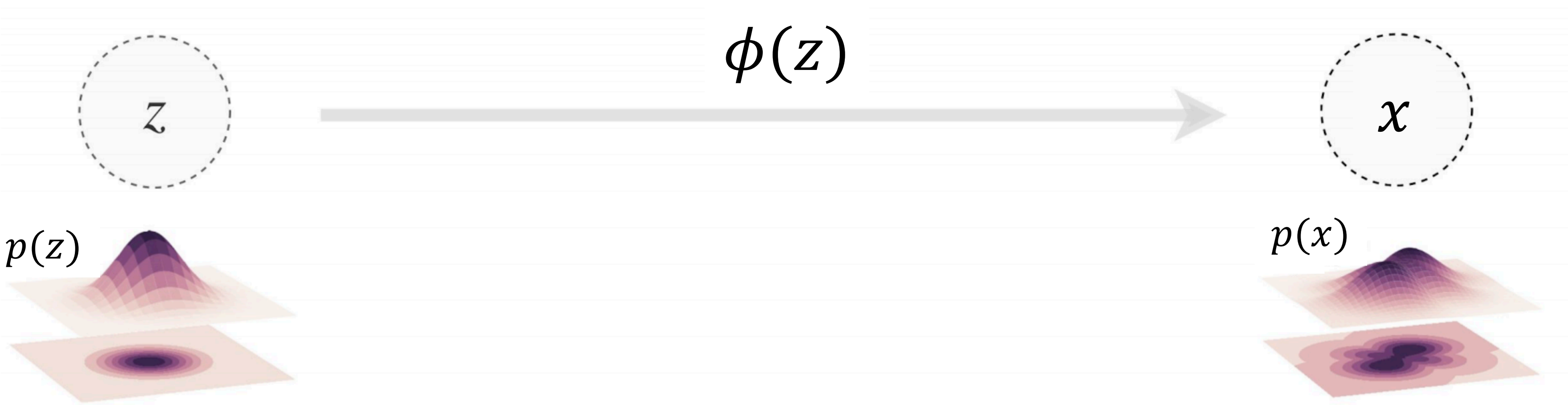
Inverse: $$(f_2 \circ f_1)^{-1} = f_1^{-1} \circ f_2^{-1}$$

Jacobian: $$\det(A \cdot B) = \det(A) \cdot \det(B)$$

Slide credit: L. Dinh

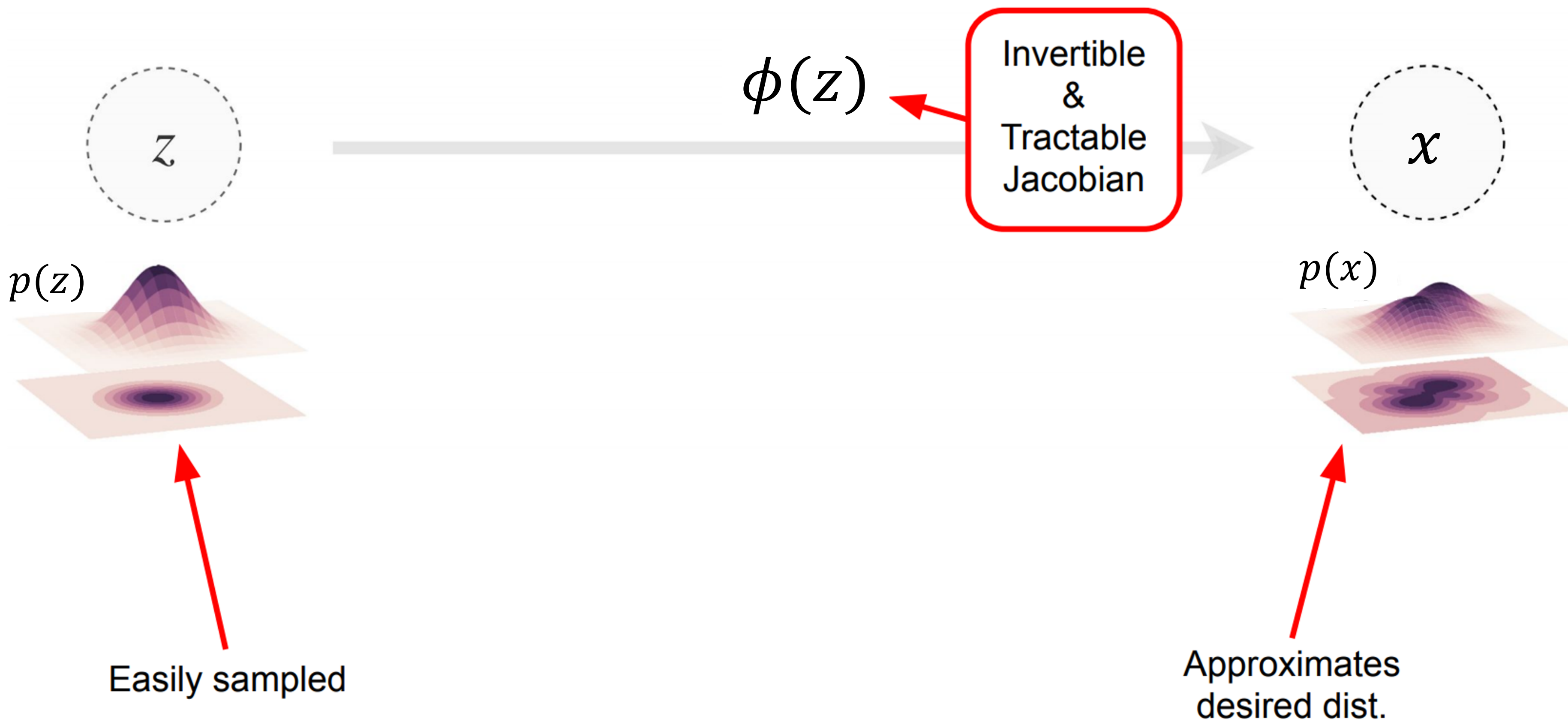
Normalizing Flows

$$p_x(\boldsymbol{x}) = p_z(\boldsymbol{z}) \left| \det \left(\frac{\partial \phi(\boldsymbol{z})}{d\boldsymbol{z}} \right)^{-1} \right|$$



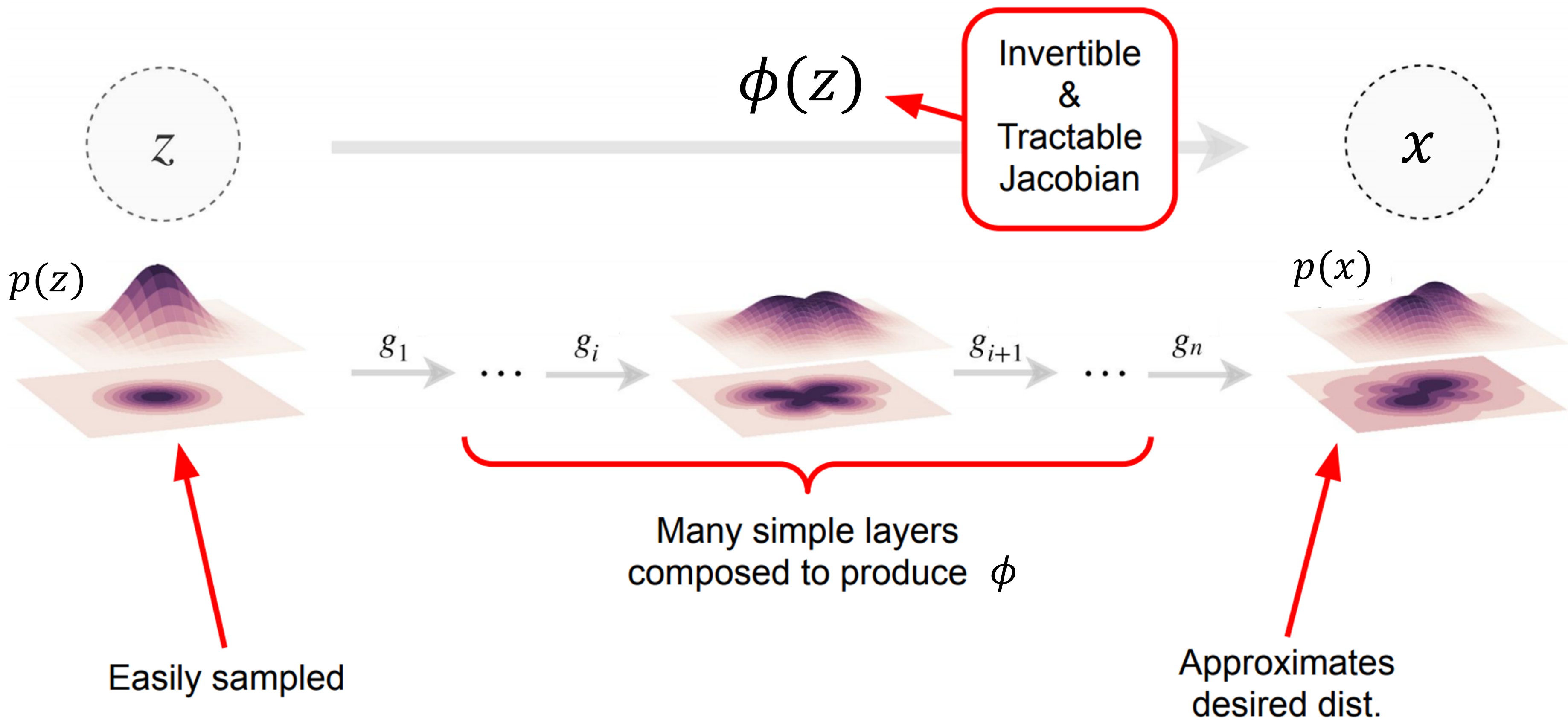
Slide credit: [G. Kanwar]()

Normalizing Flows



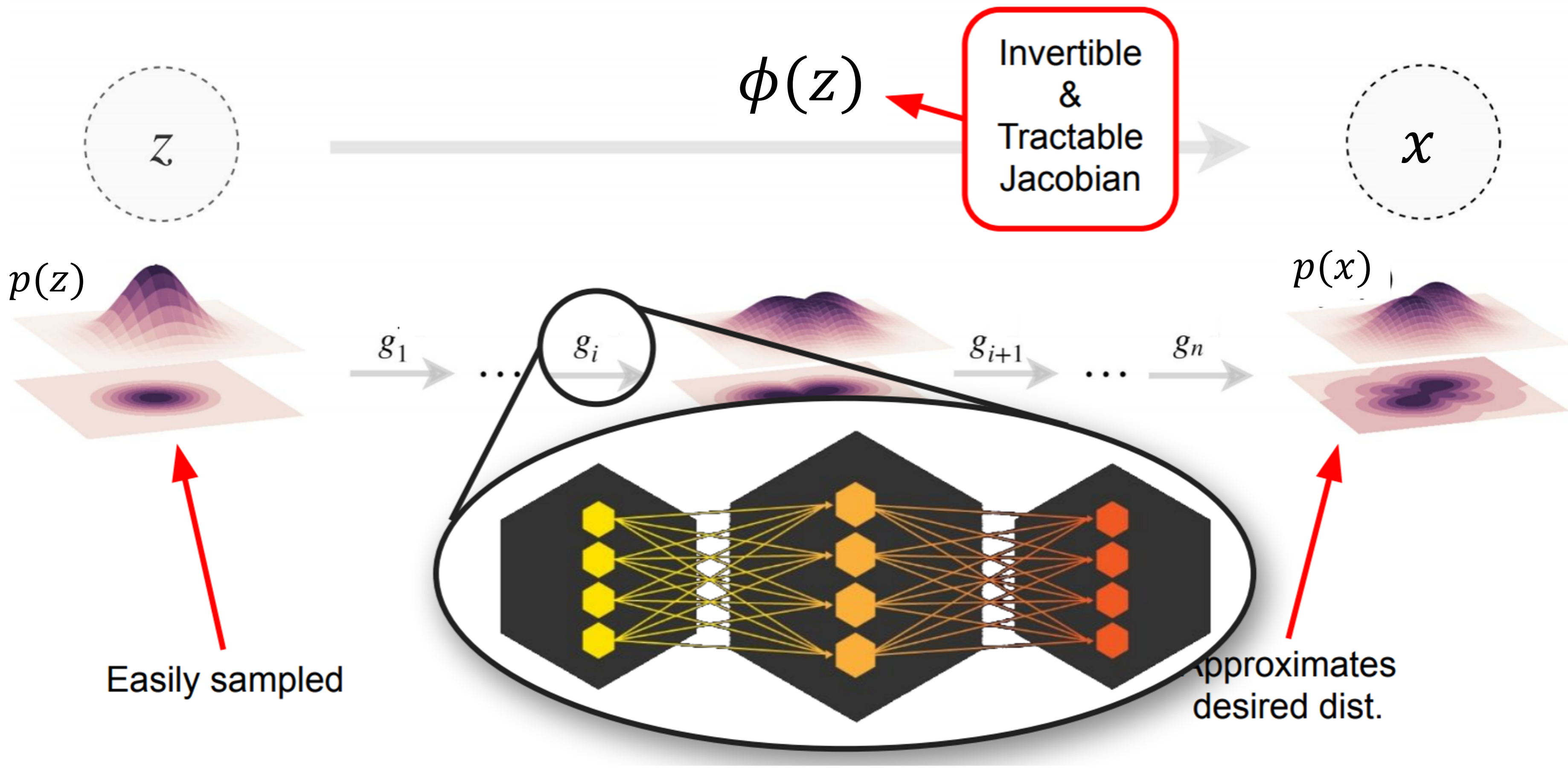
Slide credit: G. Kanwar

Normalizing Flows



Slide credit: [G. Kanwar]

Normalizing Flows



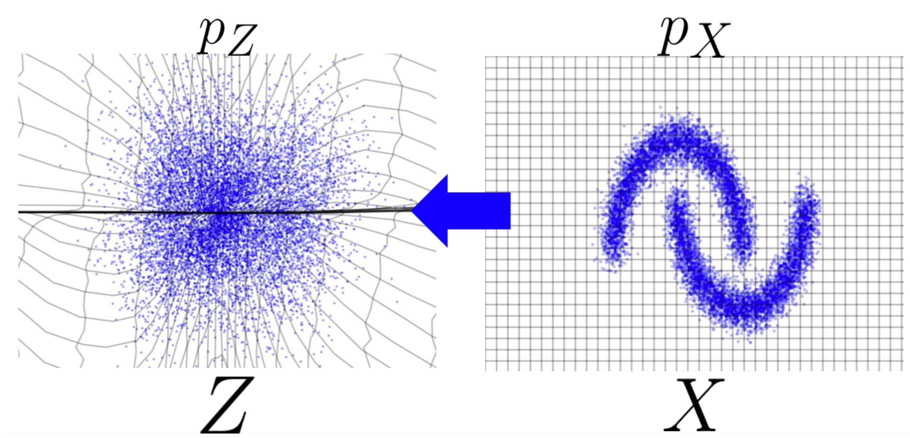
Slide credit: [G. Kanwar]()

Normalizing Flows Training

- **Learn** θ with maximum likelihood

$$\max_{\theta} p(x) = \max_{\theta} p_z\left(\phi_\theta^{-1}(x)\right) \left|\det\left(\frac{\partial \phi_\theta^{-1}(\boldsymbol{x})}{d\boldsymbol{x}}\right)\right|$$

Where $z = \phi^{-1}(x)$

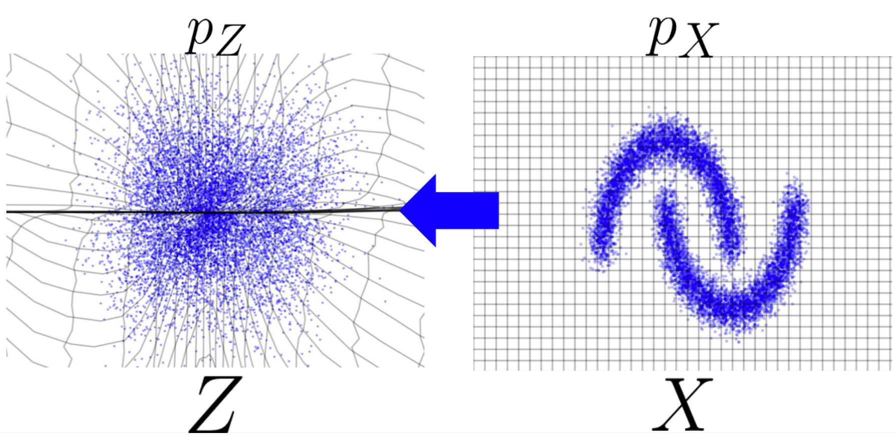


Normalizing Flows Training

- **Learn** θ with maximum likelihood

$$\max_{\theta} p(x) = \max_{\theta} p_z\left(\phi_\theta^{-1}(x)\right)\left|\det\left(\frac{\partial \phi_\theta^{-1}(\boldsymbol{x})}{d\boldsymbol{x}}\right)\right|$$

For each data point x
- Map back to point in z-space with $\phi^{-1}(x)$
- Evaluate probability in z-space with $p_z(\cdot)$



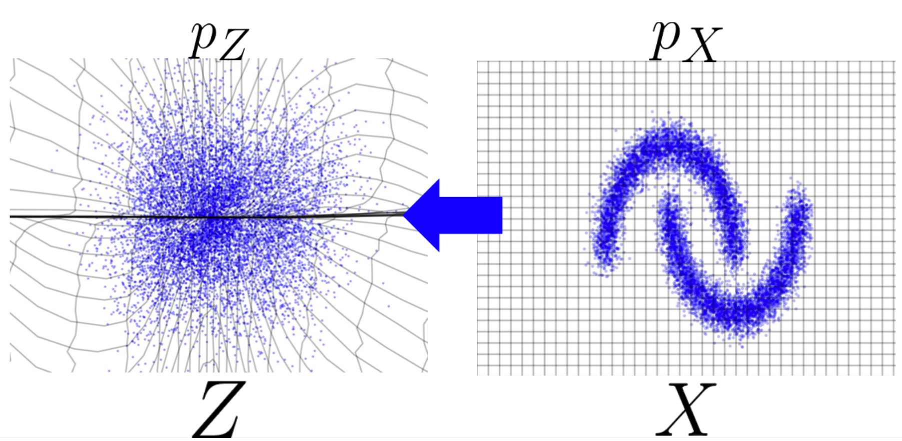
Normalizing Flows Training

- **Learn** θ with maximum likelihood

$$\max_\theta p(x) = \max_\theta p_z(\phi_\theta^{-1}(x)) \left|\det\left(\frac{\partial \phi_\theta^{-1}(\boldsymbol{x})}{d\boldsymbol{x}}\right)\right|$$

For each data point x
- Map back to point in z-space with $\phi^{-1}(x)$
- Evaluate probability in z-space with $p_z(\cdot)$

Account for volume change due to transformation $\phi^{-1}(x)$



Normalizing Flows Training

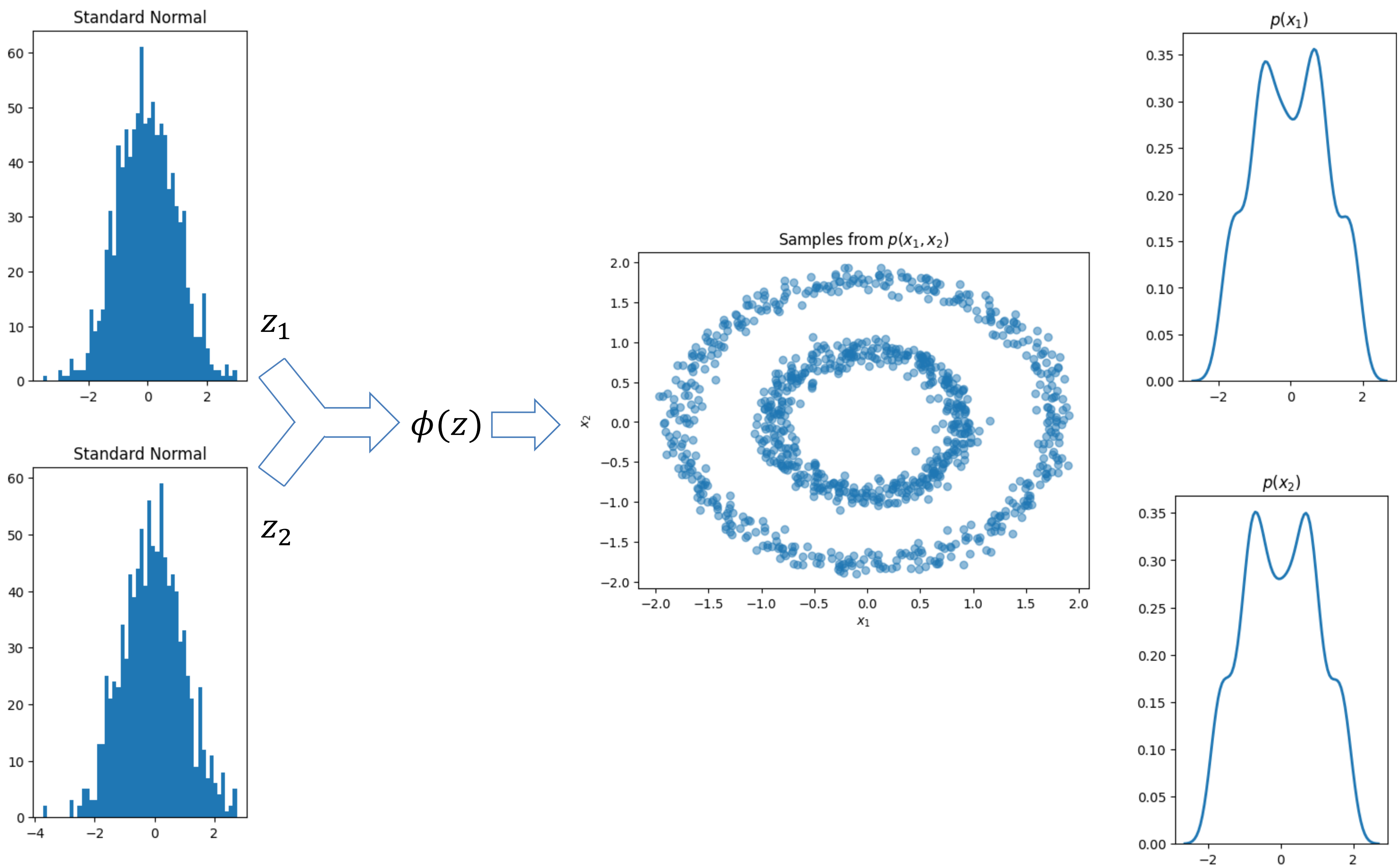
- **Learn** θ with maximum likelihood

$$\max_\theta p(x) = \max_\theta p_z\left(\phi_\theta^{-1}(x)\right)\left|\det\left(\frac{\partial \phi_\theta^{-1}(\boldsymbol{x})}{d\boldsymbol{x}}\right)\right|$$

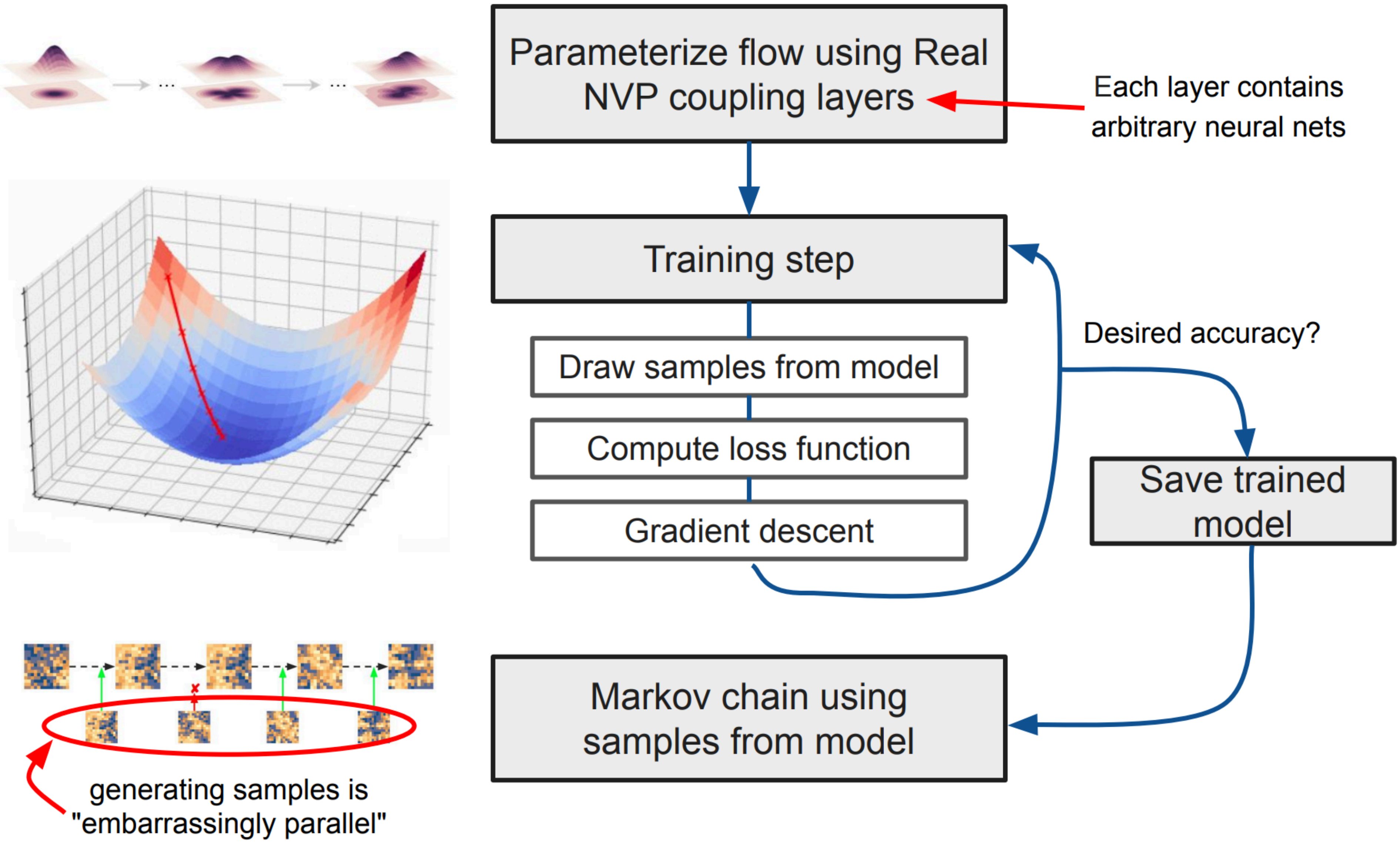
  - Gradient descent on θ
  - Find transformation s.t. data is most likely

- **Benefits** once trained
  - Can evaluate p(x) for any point X
  - Can generate "new" data points
    - Sample noise: $z \sim p(z)$
    - Transform: $\phi(z) = x$

Example Normalizing flow

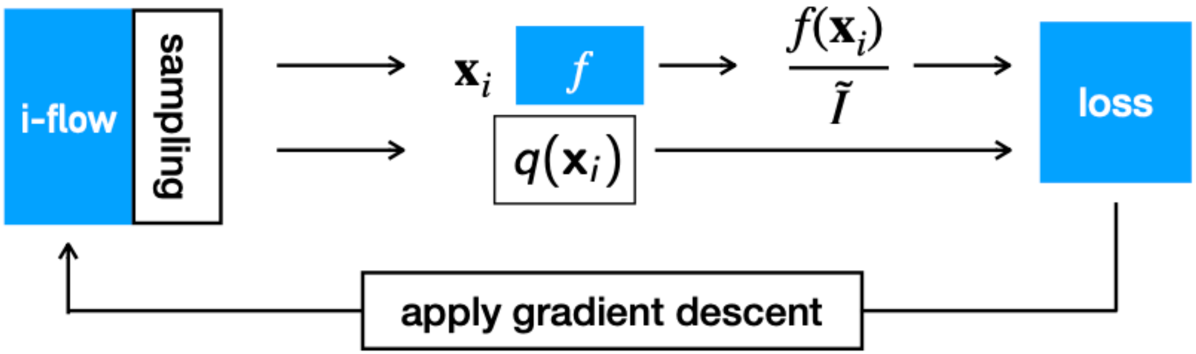


Applications: Sampling in Lattice QCD



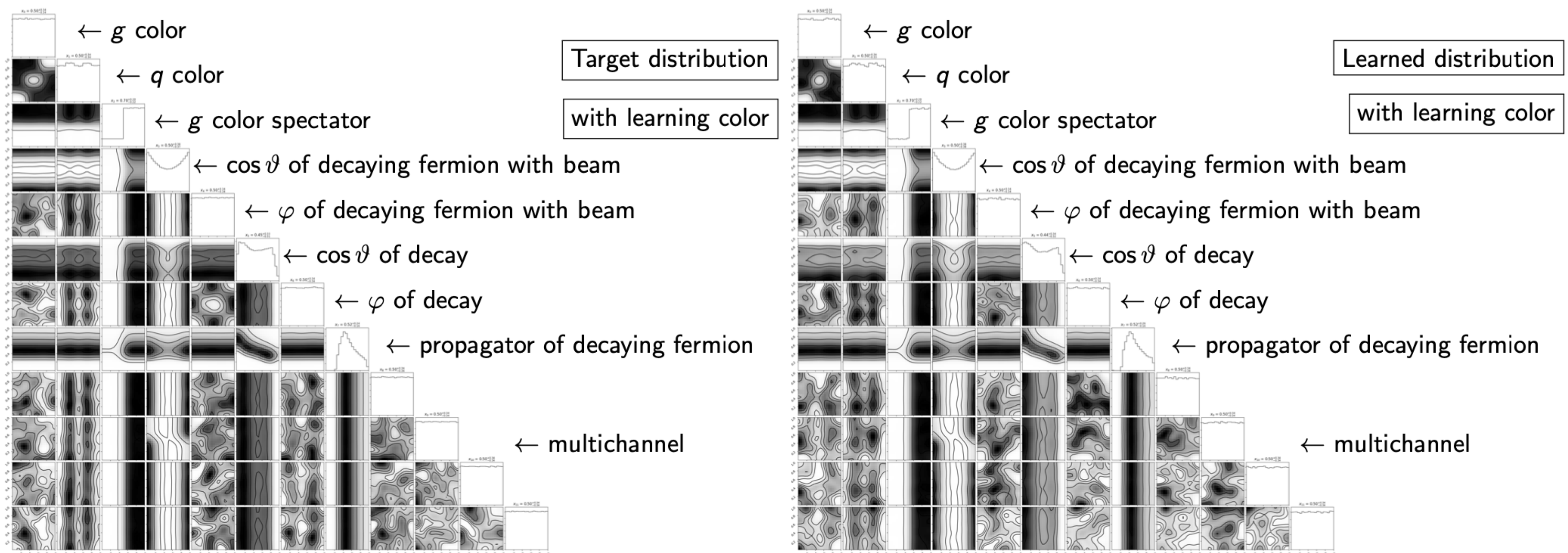
Slide credit: G. Kanwar

Event Generation with Normalizing Flows

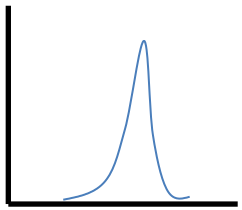


arXiv: 2001.05486, ML:ST
arXiv: 2001.10028, PRD
Slide credit: C. Krause

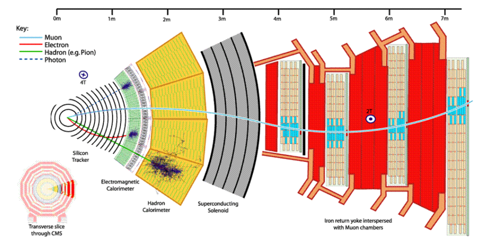
Example: Learning $e^+e^- \to 3j$



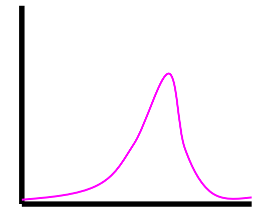
Unfolding with Normalizing Flows



$x \sim p(x)$ = input / true distribution



$p(y|x)$ = Detector smearing

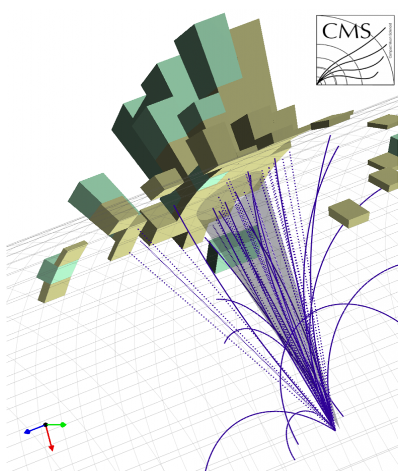


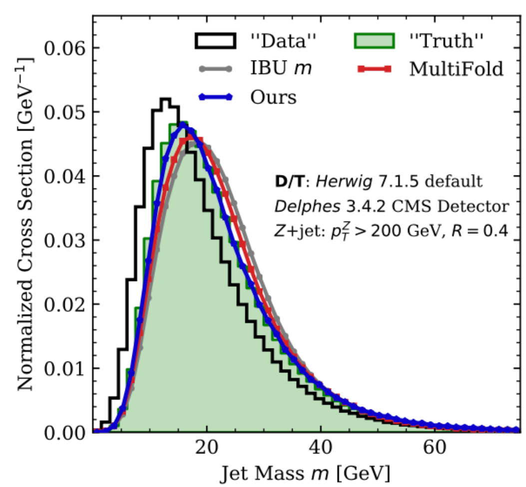
$y \sim p(y)$ = output / observed distribution

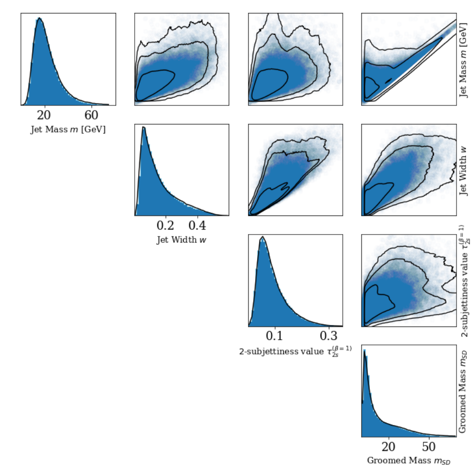
Observed distribution: $p(y) = \int p(y|x)p(x)dx \approx \sum_{x \sim p(x)} p(y|x)$

- Normalizing flows to model detector $p(y|x)$ trained with simulation
- Normalizing flow to model unknown truth $p_\theta(x)$
- Maximize data likelihood $p(y)$ → Gradient descent to learn parameters θ

Unfolding
Jet variables
in Z+jet events







PMLR 130:2107-2115, 2021

Conclusions

- Deep neural networks are an extremely powerful class of models
- We can express our inductive bias about a system in terms of model design, and can be adapted to a many types of data
- Even beyond classification and regression, deep neural networks allow for powerful model schemes such as Generative adversarial Networks and normalizing flows that open many new possible tasks where Machine Learning can be applied in HEP

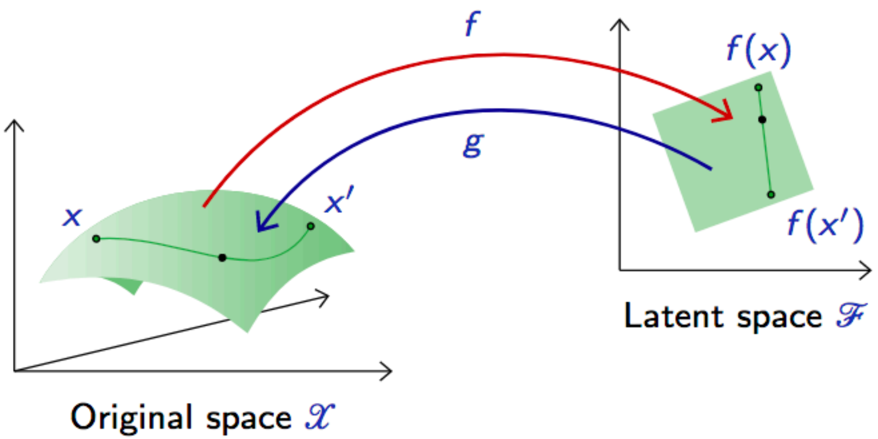
- Autoencoders learn the latent space, but we don't know what is the latent space distribution

- Autoencoder prescribes a deterministic relationship between data space and latent space

- One set of "meaningful degrees of freedom" can only describe one data space point

Interpolating in Latent Space

$$\alpha \in [0,1], \quad \xi(x, x', \alpha) = g((1-\alpha)f(x) + \alpha f(x')).$$



Autoencoder interpolation ($d = 8$)

Reparameterization trick

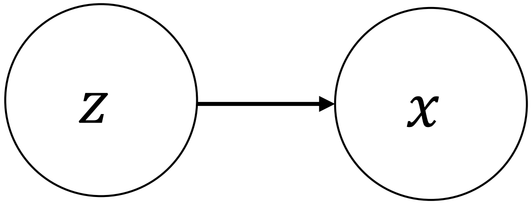
- For z$\sim p_\theta(z)$, rewrite z as a function of a random variable ϵ whose distributions $p(\epsilon)$ does not depend on θ
  - Gaussian Example:

$$z \sim \mathcal{N}(\mu, \sigma) \quad \rightarrow \quad z = \sigma * \epsilon + \mu \quad \textit{where} \;\; \epsilon \sim \mathcal{N}(0,1)$$

- VAE Loss

$$\max_{\theta,\psi} L(\theta,\psi) = \max_{\theta,\psi} \sum_{\epsilon \sim p(\epsilon)} \log p_\theta\left(x \middle| z_i = \epsilon * \sigma_\psi\,(x) + \mu_\psi(x)\right) - \log\left[\frac{q_\psi(z_i|x)}{p(z_i)}\right]$$

Latent Variable Models



- Observed random variable x depends on unobserved latent random variable z
  - Interpret z as the causal factors for x

- Joint probability: $p(x, z) = p(x|z)p(z)$

- $p(x|z)$ is a stochastic generation process from $z \rightarrow x$

- Inference from posterior: $$p(z|x) = \frac{p(x|z)p(z)}{p(x)}$$
  - Usually can't compute marginal $p(x) = \int p(x|z)p(z)dz$

Autoencoder: Deterministic to Probabilistic

- Consider probabilistic relationship between data and latent variables

$$x, z \sim p(x, z) = p(x|z)p(z)$$

Decoding data x from latent z

Prior over latent space

From Deterministic to Probabilistic Autoencoder

- Consider probabilistic relationship between data and latent variables

$$x, z \sim p(x, z) = p(x|z)p(z)$$

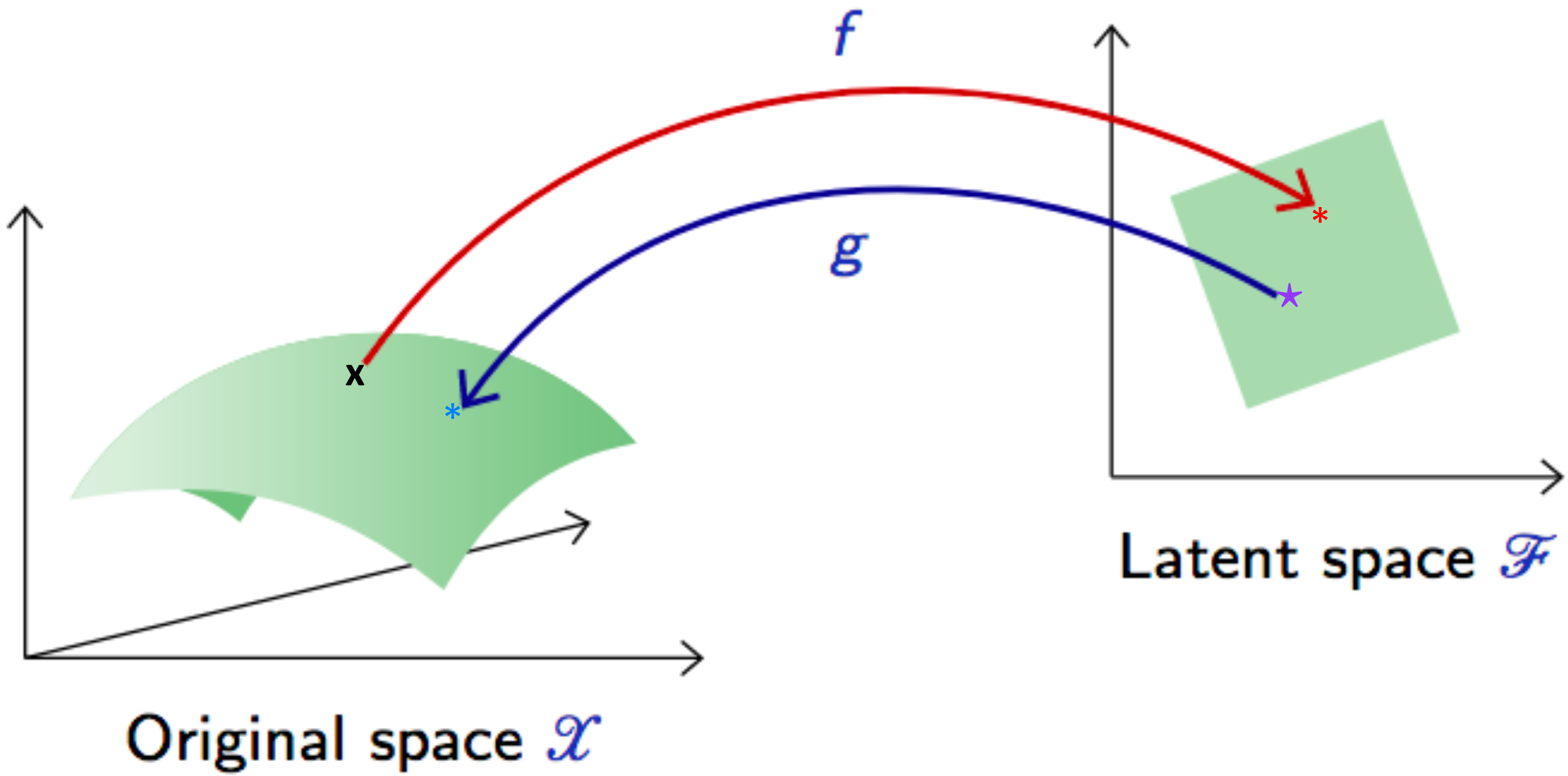
- Autoencoding

$$x \rightarrow q(z|x) \underset{sample}{\Longrightarrow} z \rightarrow p(x|z)$$

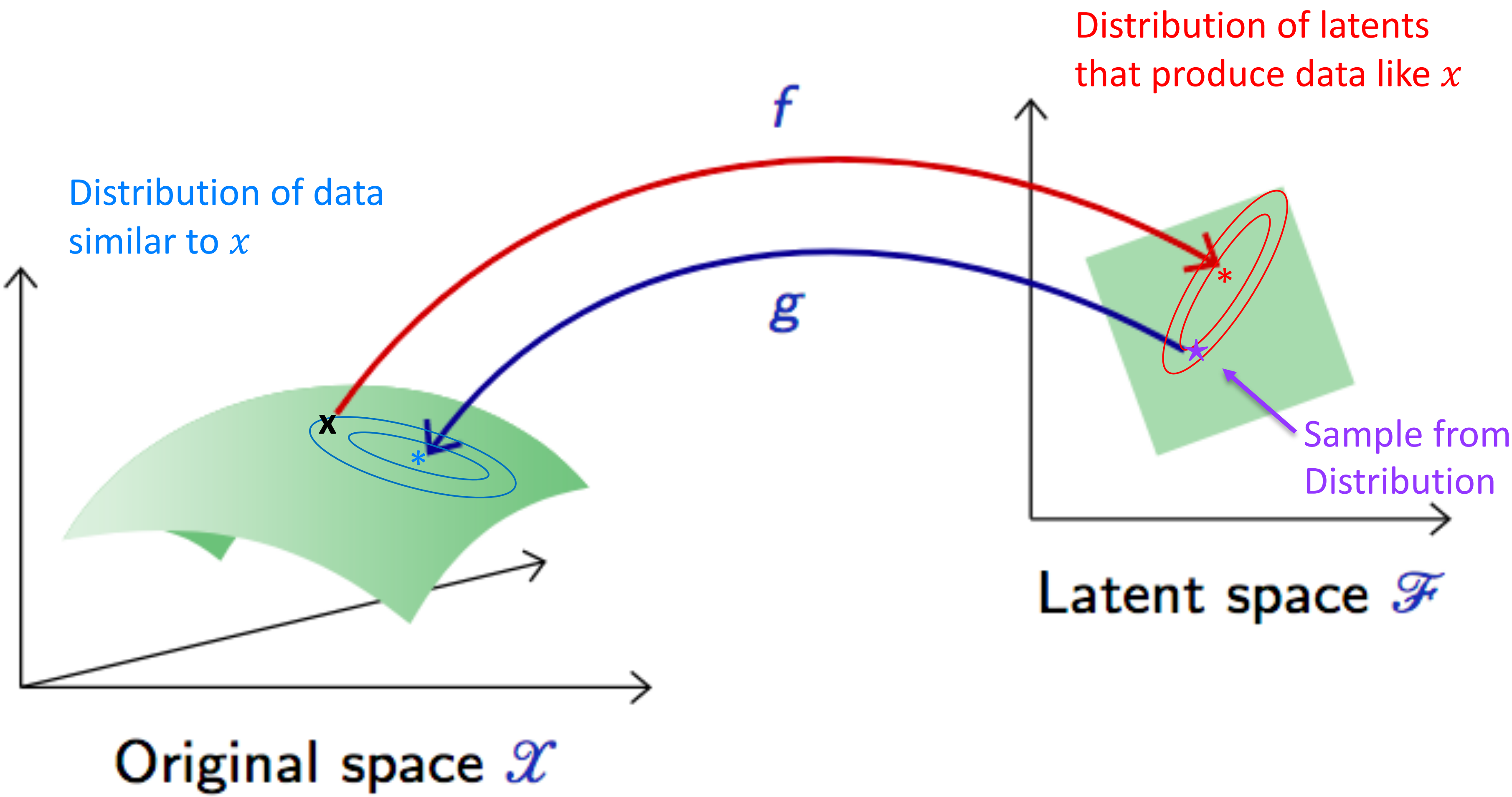
  – Choose simple prior distribution

  – **Encoder:** Learn what latents can produced data: $q(z|x)$
  – **Decoder:** Learn what data is produced by latent: $p(x|z)$

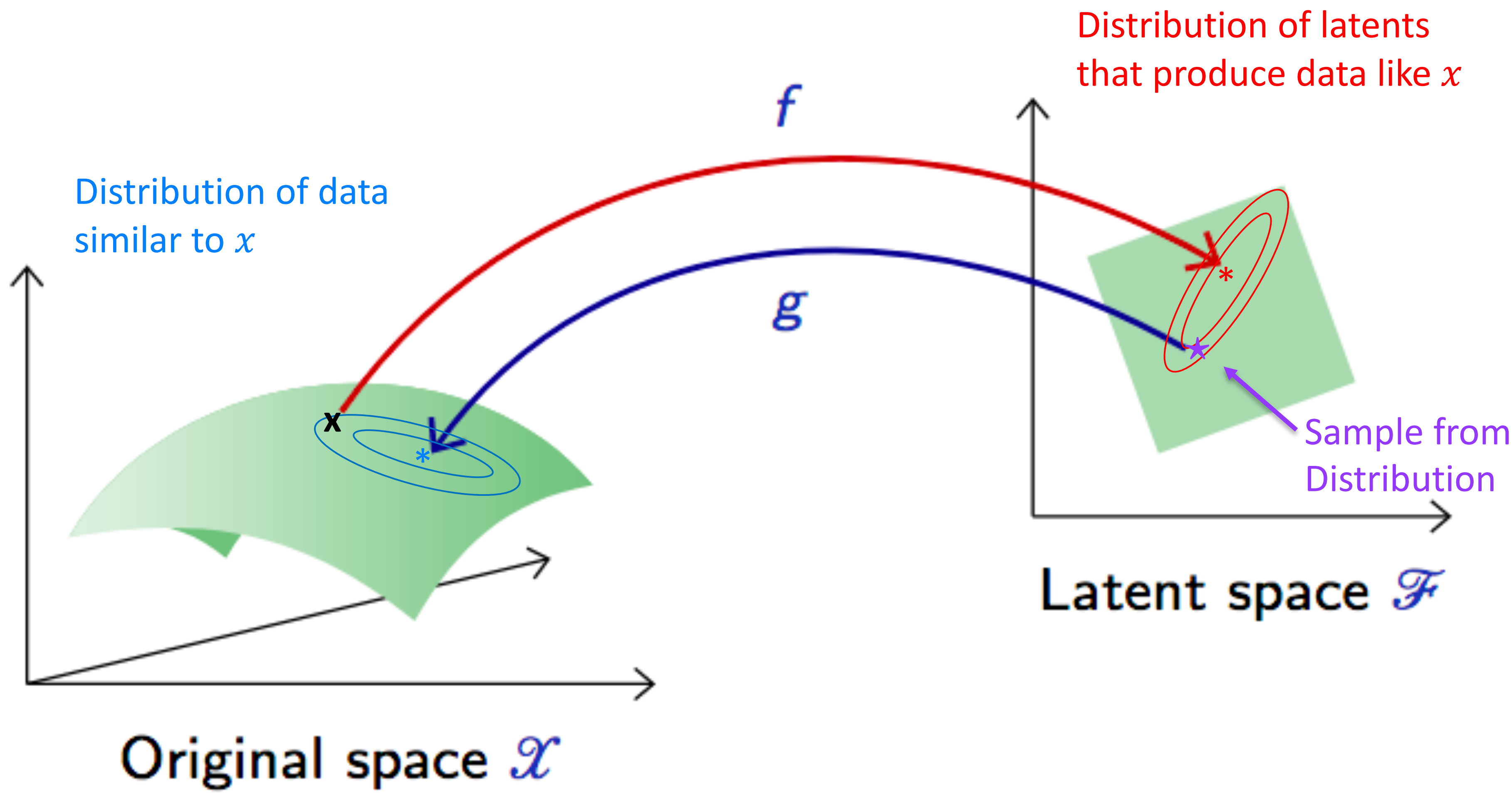
Autoencoder → Variational Autoencoder (VAE)



Autoencoder → Variational Autoencoder (VAE)



Autoencoder → Variational Autoencoder (VAE)



- Encode data x into distribution over latents $q(z|x)$
- For any sample z decode into distribution over data $p(x|z)$

Model Distributions with Parametrized Models

- PDF often depends on parameters θ we are interested in
  - Write the density as $f(x|\theta)$ or $f(x;\theta)$

- Choose a family we know: Gaussian

- Model the parameters θ as output of Neural Net

$$\mu(x) \equiv NN(x)$$
$$\sigma(x) \equiv NN(x)$$
$$p(z|x) = \mathcal{N}\big(z;\mu(x),\sigma(x)\big)$$

Encoding Distribution

$$\log p(z|x) = \frac{\big(z-\mu(x)\big)^2}{2\sigma^2(x)} - \frac{1}{2}\log\sigma(x) \cancel{-\log\sqrt{\pi}}$$

Model Distributions with Parametrized Models

- PDF often depends on parameters θ we are interested in
  - Write the density as $f(x|\theta)$ or $f(x;\theta)$

- Choose a family we know: Gaussian

- Model the parameters θ as output of Neural Net

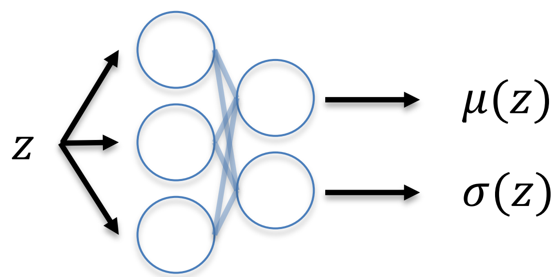
Decoding Distribution

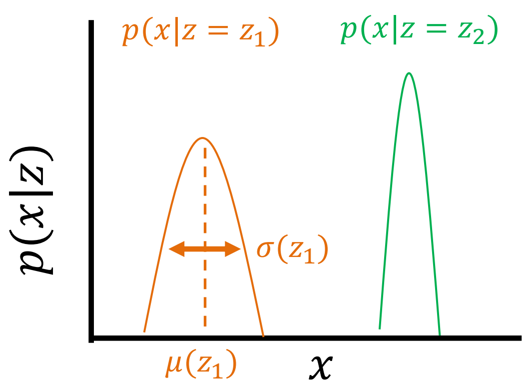
$$\mu(z) \equiv NN(z)$$
$$\sigma(z) \equiv NN(z)$$
$$p(x|z) = \mathcal{N}\big(x; \mu(z), \sigma(z)\big)$$

$$\log p(x|z) = \frac{\big(x - \mu(z)\big)^2}{2\sigma^2(z)} - \frac{1}{2}\log\sigma(z) \cancel{- \log\sqrt{\pi}}$$

Model Distributions with Parametrized Models

- PDF often depends on parameters θ we are interested in
  - Write the density as $f(x|\theta)$ or $f(x;\theta)$
- Choose a family we know: Gaussian
- Model the parameters θ as output of Neural Net





VAE Loss Function

$$\max_{\theta,\psi} L(\theta,\psi)$$

$$L(\theta,\psi) = \sum_{z\sim q_\psi(z|x)} \log p_\theta(x|z) - \sum_{z\sim q_\psi(z|x)} \log \frac{q_\psi(z|x)}{\mathcal{N}(z;0,1)}$$

- First Term
  - Check compatibility with original data after encoding and decoding
- Second Term
  - Check compatibility of encoded data with prior
  - Constraint on latent distribution

VAE Loss Function

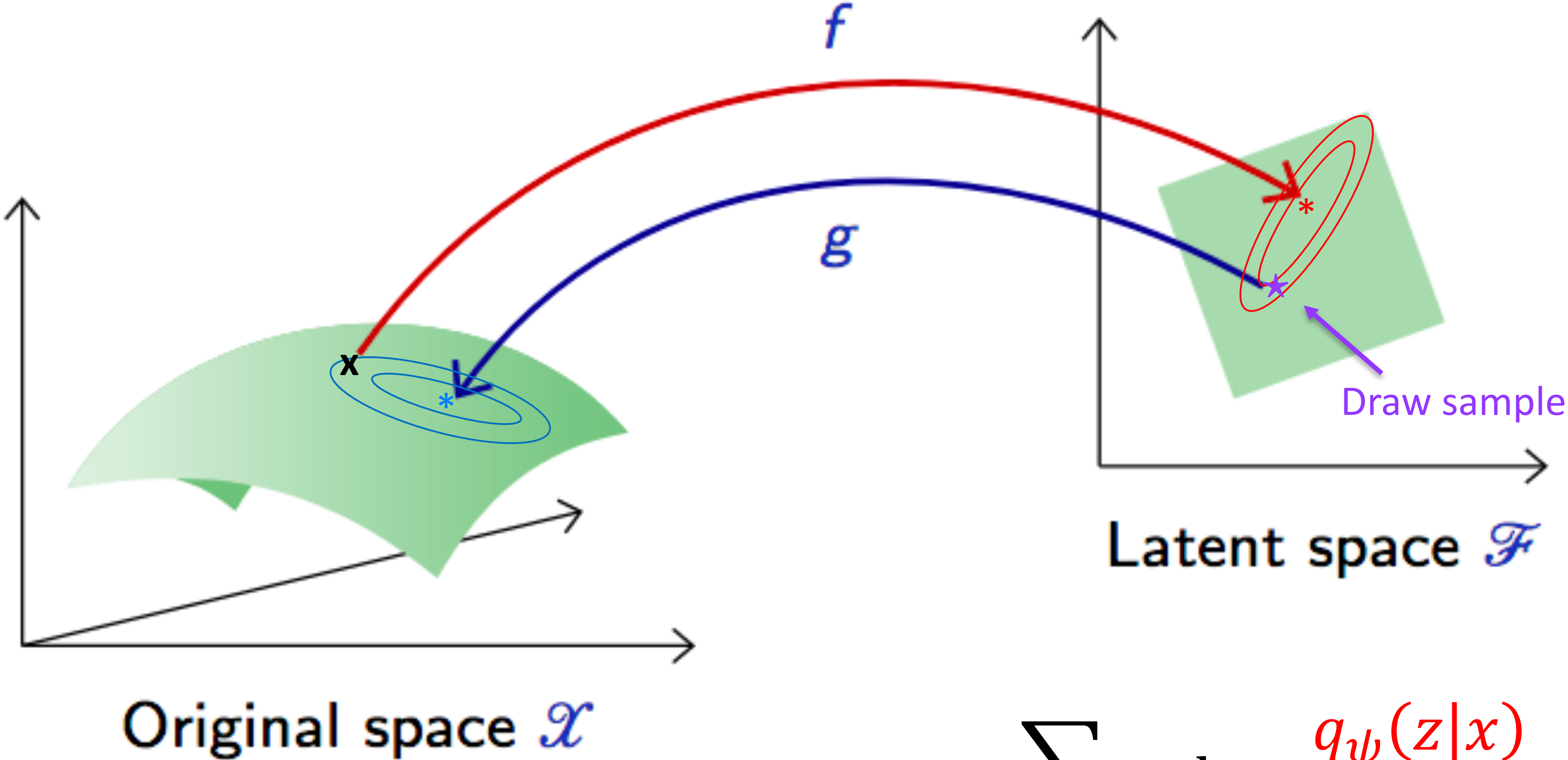
$$\max_{\theta,\psi} L(\theta,\psi)$$

$$L(\theta,\psi) = \sum_{z\sim q_\psi(z|x)} \log p_\theta(x|z) - \sum_{z\sim q_\psi(z|x)} \log \frac{q_\psi(z|x)}{\mathcal{N}(z;0,1)}$$

$$= \frac{1}{2} \sum_{z\sim q_\psi(z|x)} \frac{\left(x-\mu(z)\right)^2}{\sigma^2(z)} - \log \sigma(z)$$

$$- \frac{1}{2} \sum_{z\sim q_\psi(z|x)} \sigma(x) - \mu^2(x) - 1 - \log \sigma(x)$$

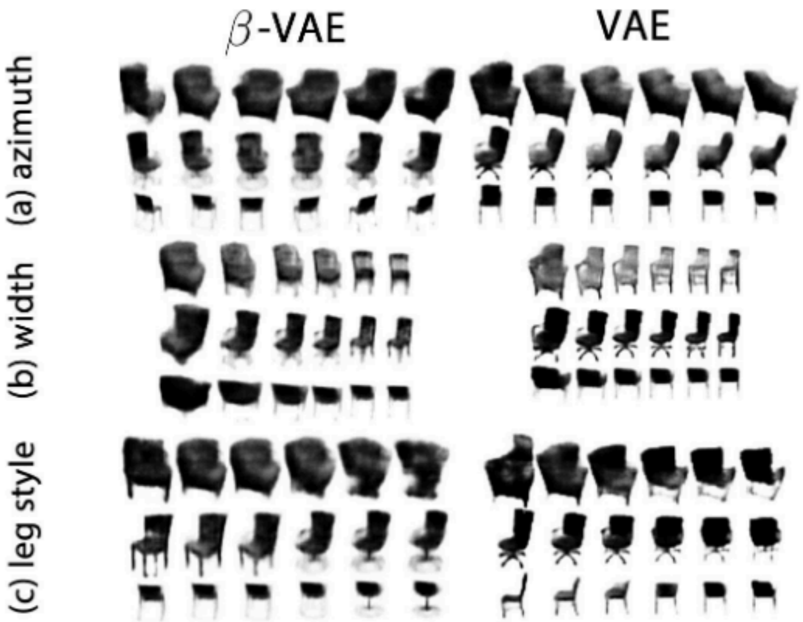
Probabilistic Picture



$$\sum_{z \sim q_\psi(z|x)} \log \frac{q_\psi(z|x)}{\mathcal{N}(z;0,1)}$$

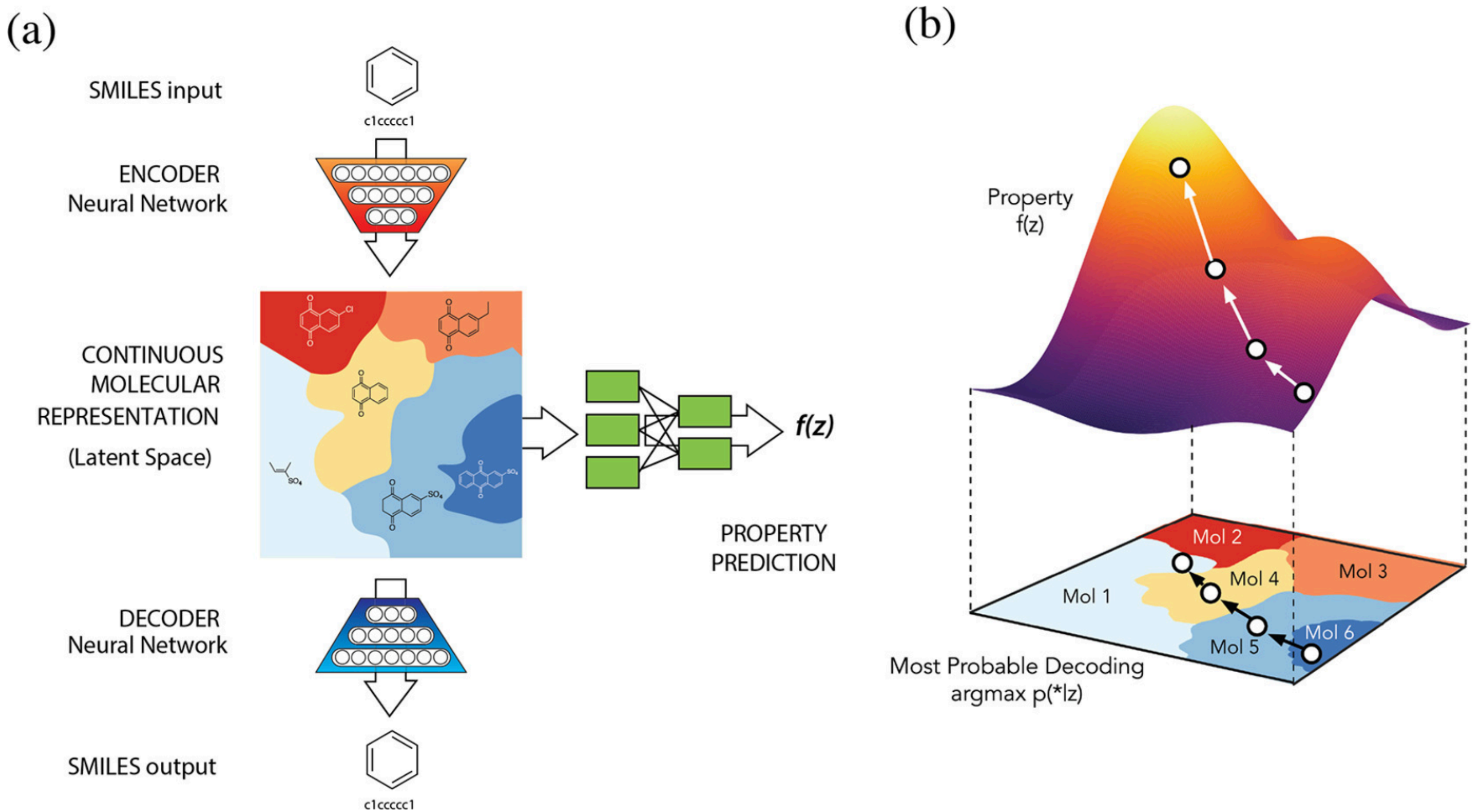
$$\sum_{z \sim q_\psi(z|x)} \log p_\theta(x|z)$$

Examples



Higgins et al., [2017]

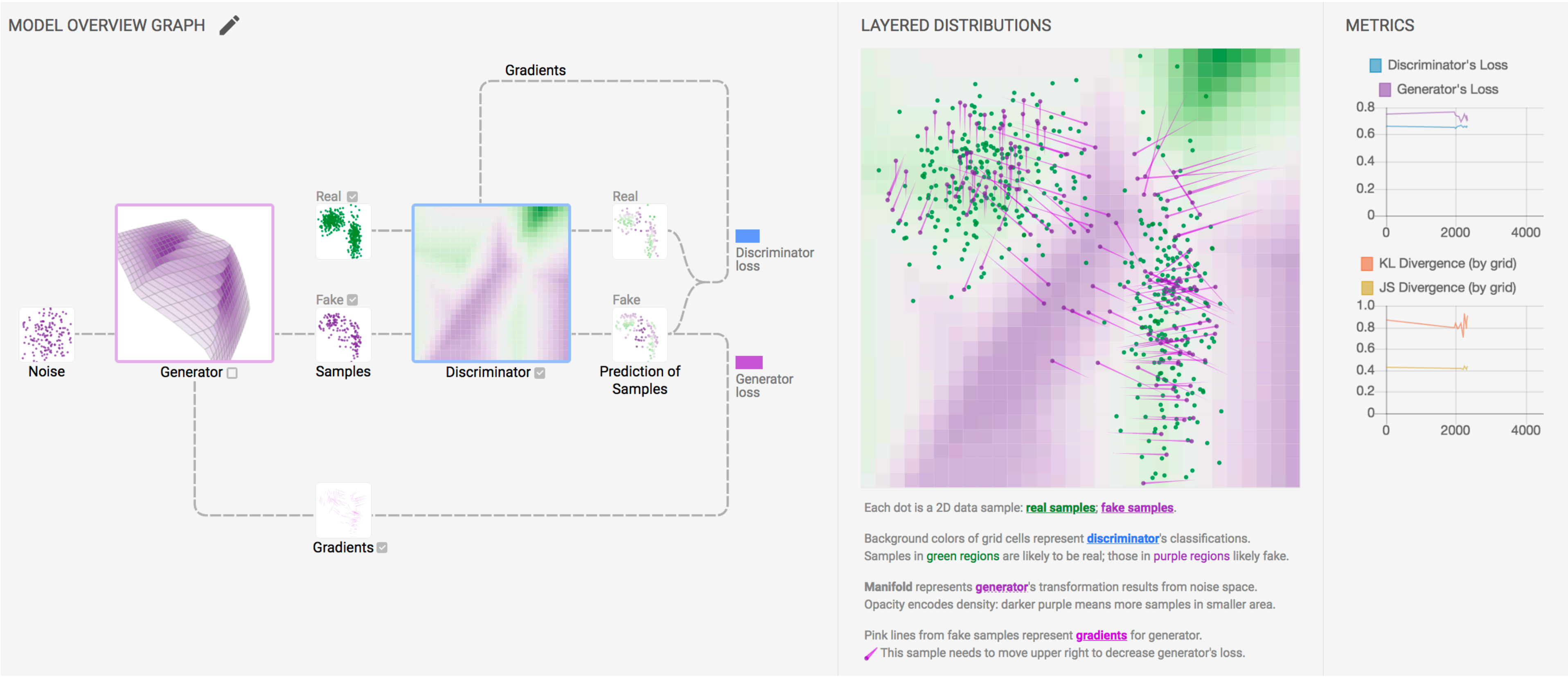
Examples



Design of new molecules with desired chemical properties.
(Gomez-Bombarelli et al, 2016)

Slide credit: [G. Louppe]

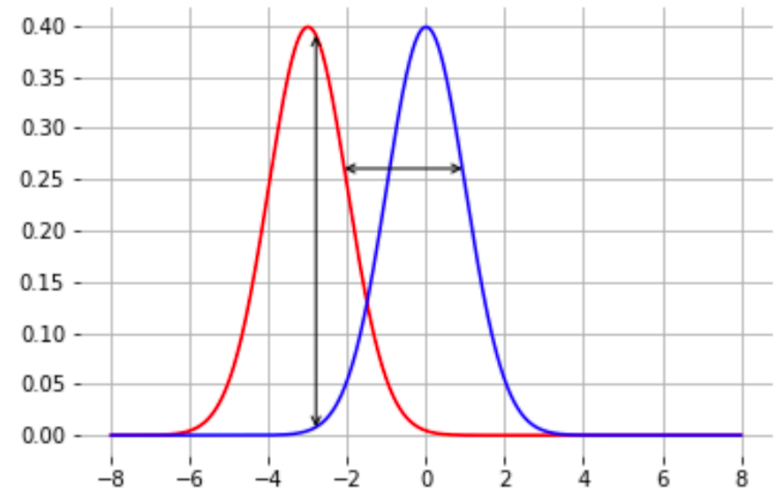
GAN Training Example



[GAN Lab Demo]()

Improving GANS

- Standard GANS compare real and fake distributions with Jensen-Shannon Divergence, "vertically"
- Wasserstein-GAN (Arjovsky et al, 2017) compares "horizontally" with Wasserstein-1 distance (a.k.a. Earth Movers distance)
- Substantially improves *vanishing gradient* and *mode collapse* problems!



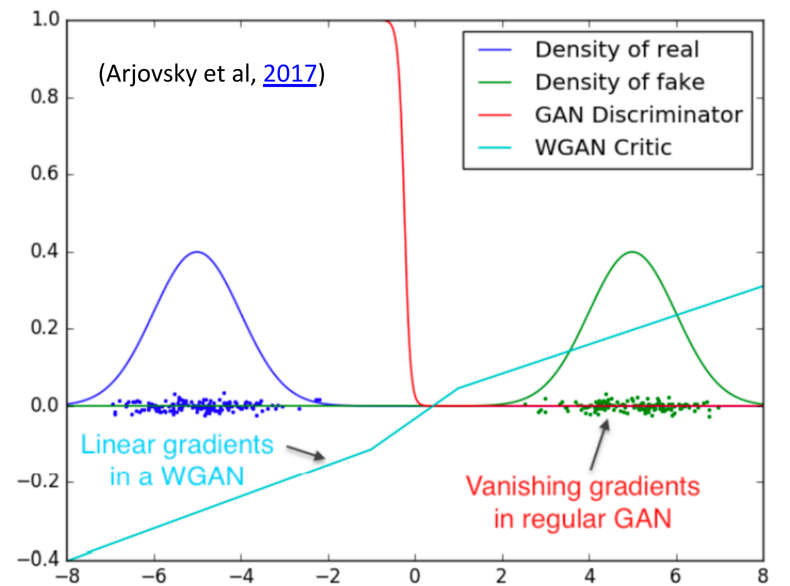


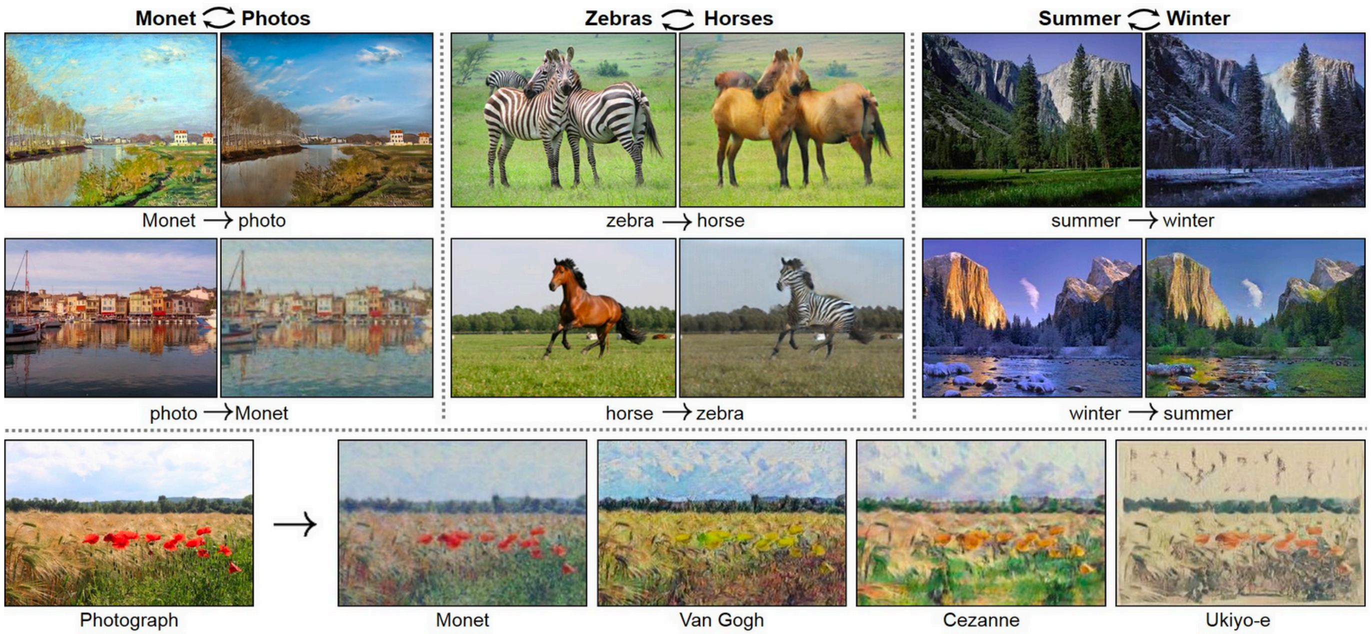
Figure 2: Optimal discriminator and critic when learning to differentiate two Gaussians. As we can see, the discriminator of a minimax GAN saturates and results in vanishing gradients. Our WGAN critic provides very clean gradients on all parts of the space.

BigGAN

(Brock et al, 2018)

Applications: Image-to-Image Translation with CycleGAN

- $p(z)$ doesn't have to be random noise
- CycleGAN uses *cycle-consistency loss* in addition to GAN loss
  - Translating from A→B→A should be consistent with original A



Applications: Text-to-Image Synthesis with StackGAN

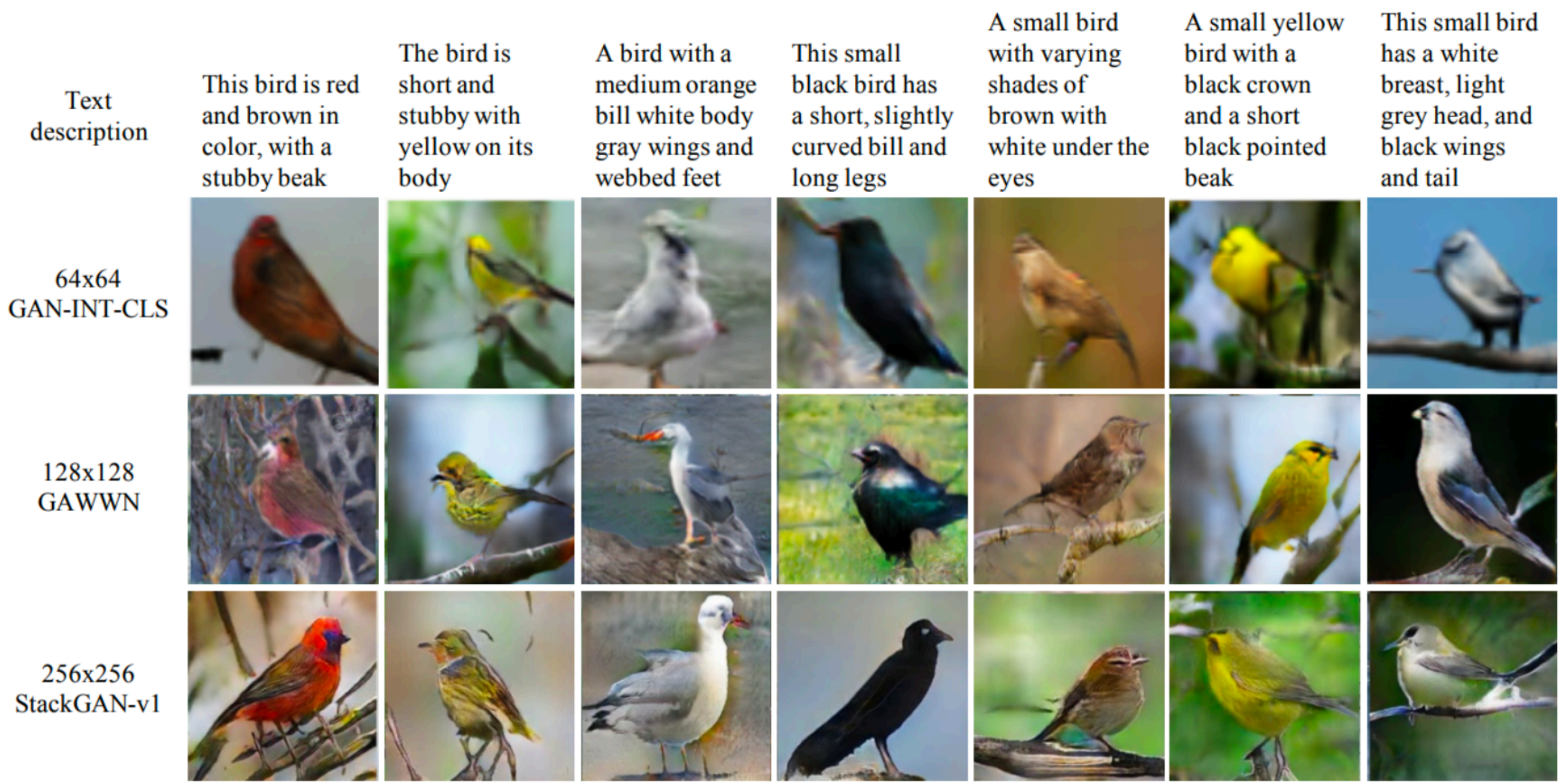
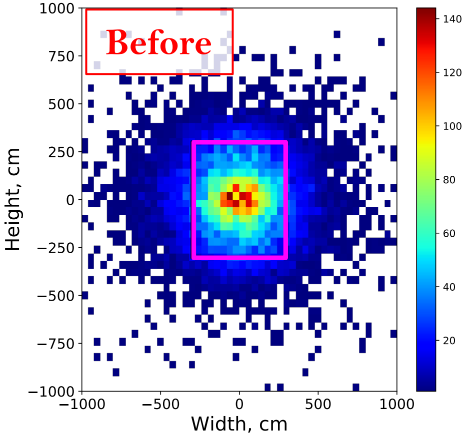
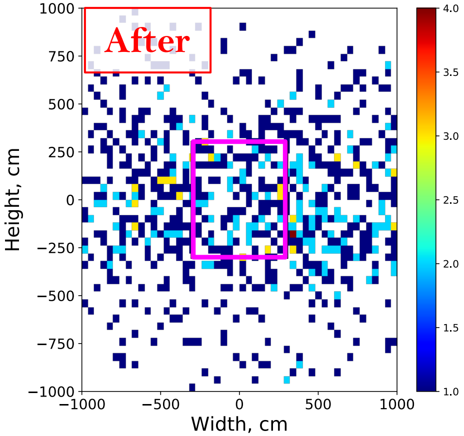


Fig. 3: Example results by our StackGAN-v1, GAWWN [29], and GAN-INT-CLS [31] conditioned on text descriptions from CUB test set.

(Zhang et al, 2017)

Design Optimization





Normalizing Flows

Normalizing Flows

$$\begin{tikzcd}[column sep=huge, row sep=huge]
x \arrow[r, shift left, "\phi^{-1}(x)"] \arrow[d, dashed, no head] & z \arrow[l, shift left, "\phi(z)"] \arrow[d, dashed, no head] \\
p_x(x) \arrow[r, shift left, "p_z(\phi^{-1}(x)) \left|\det\left(\frac{\partial \phi^{-1}(\boldsymbol{x})}{d\boldsymbol{x}}\right)\right|"] & p_z(z) \arrow[l, shift left, "p_z(z) \left|\det\left(\frac{\partial \phi(\boldsymbol{z})}{d\boldsymbol{z}}\right)^{-1}\right|"]
\end{tikzcd}$$