Introduction to Machine Learning: Lecture III

Michael Kagan

SLAC

INFN School of Statistics 2022

May 19, 2022

The Plan

- Lecture 1
  - 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

Beyond Regression and Classification

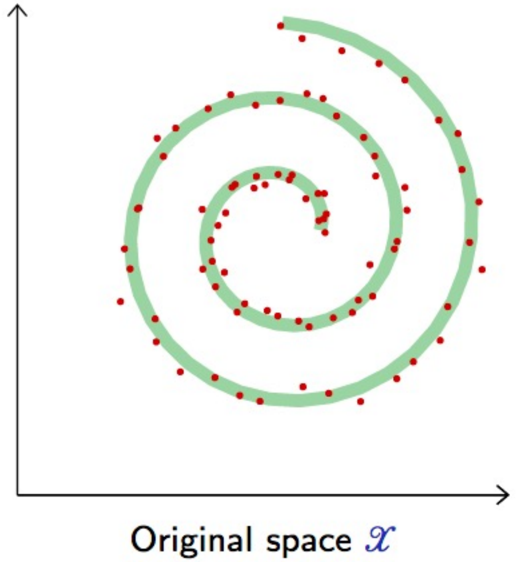
- Not all tasks are predicting a label from features, as in classification and regression

- May want / need to explicitly model a high-dim. signal
  - Data synthesis / simulation
  - Density estimation
  - Anomaly detection
  - Denoising, super resolution
  - Data compression
  - …

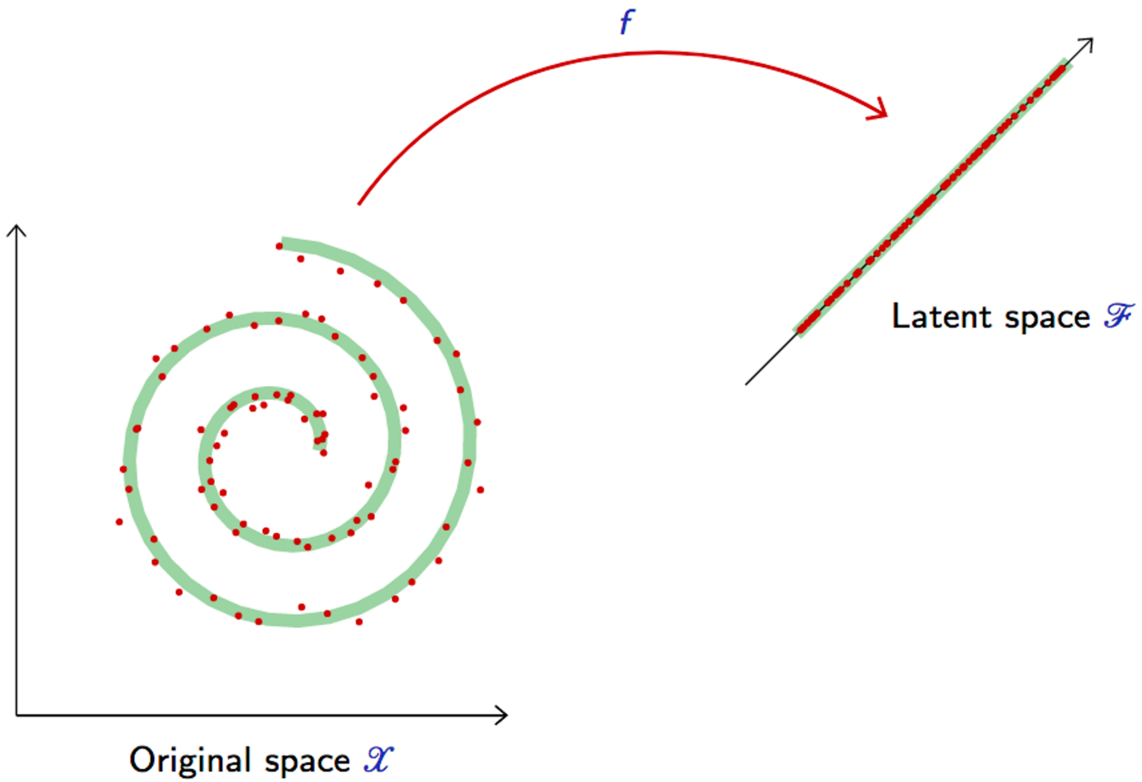
- Often don't have labels → Unsupervised Learning

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

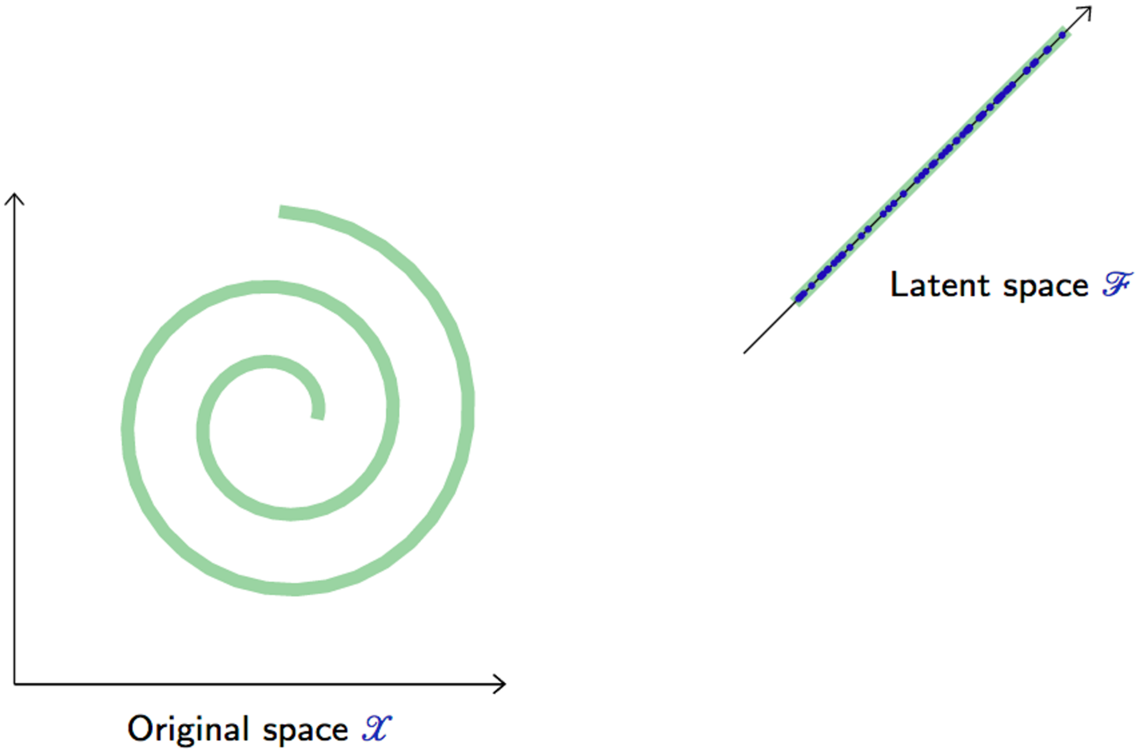
Modeling Data and Meaningful Degrees of Freedom



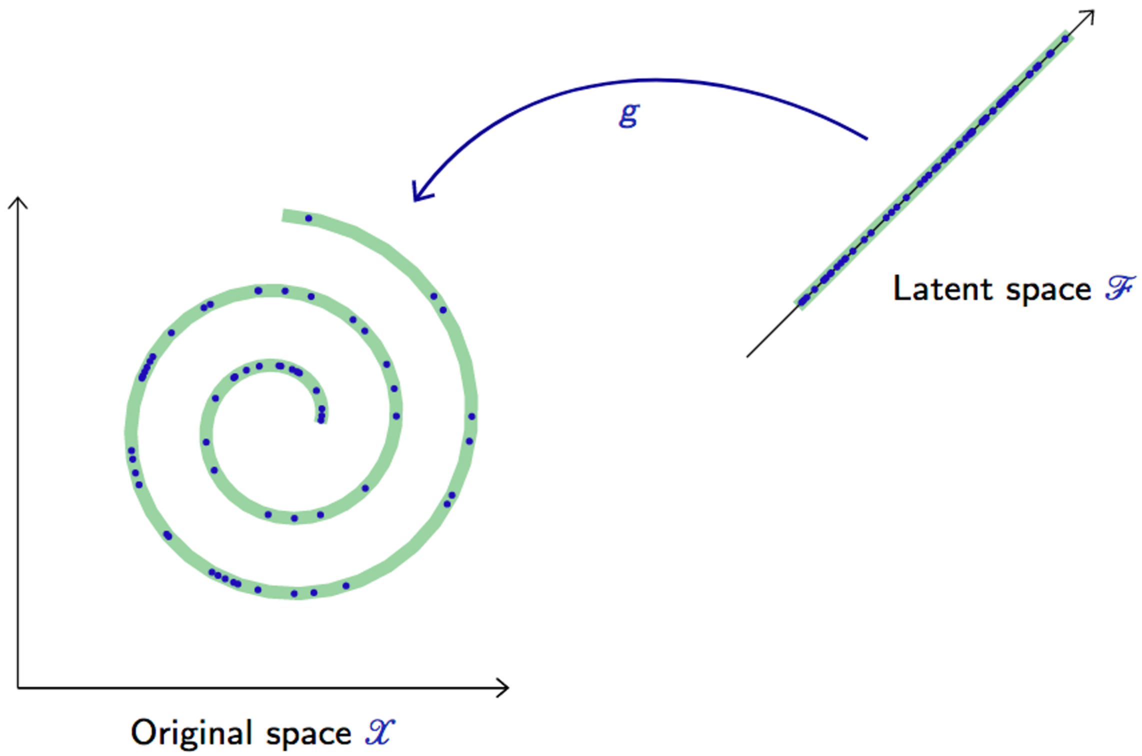
Modeling Data and Meaningful Degrees of Freedom



Modeling Data and Meaningful Degrees of Freedom



Modeling Data and Meaningful Degrees of Freedom



Autoencoders

Meaningful Representations

- How can we find the "meaningful degrees of freedom" in the data?

- Dimensionality Reduction / Compression
  - Can we compress the data to a *latent space* with smaller number of dimensions, and still recover the original data from this latent space representation?

  - Latent space must encode and retain the important information about the data

  - Can we learn this compression and latent space

Autoencoders

- Autoencoders map a space to itself through a compression, $x \rightarrow z \rightarrow \hat{x}$, and should be close to the identity on the data
  - Data: $x \in \mathcal{X}$ Latent space: $\mathrm{z} \in \mathcal{F}$
  - **Encoder**: Map from $\mathcal{X}$ to a lower dimensional latent space $\mathcal{F}$
    - Parameterize as neural network $f_\theta(x)$ with parameters θ
  - **Decoder**: Map from latent space $\mathcal{F}$ back to data space $\mathcal{X}$
    - Parameterize as neural network $g_\psi(z)$ with parameters ψ

Autoencoders

- Autoencoders map a space to itself through a compression, $x \rightarrow z \rightarrow \hat{x}$, and should be close to the identity on the data
  - Data: $x \in \mathcal{X}$ Latent space: $z \in \mathcal{F}$
  - **Encoder**: Map from $\mathcal{X}$ to a lower dimensional latent space $\mathcal{F}$
    - Parameterize as neural network $f_\theta(x)$ with parameters θ
  - **Decoder**: Map from latent space $\mathcal{F}$ back to data space $\mathcal{X}$
    - Parameterize as neural network $g_\psi(z)$ with parameters ψ
- What is the latent space? What are $f(x)$ and $g(z)$?
  - Choose a latent space dimension D
  - Learn mappings $f(x)$ to representation of size D, and back with $g(z)$

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

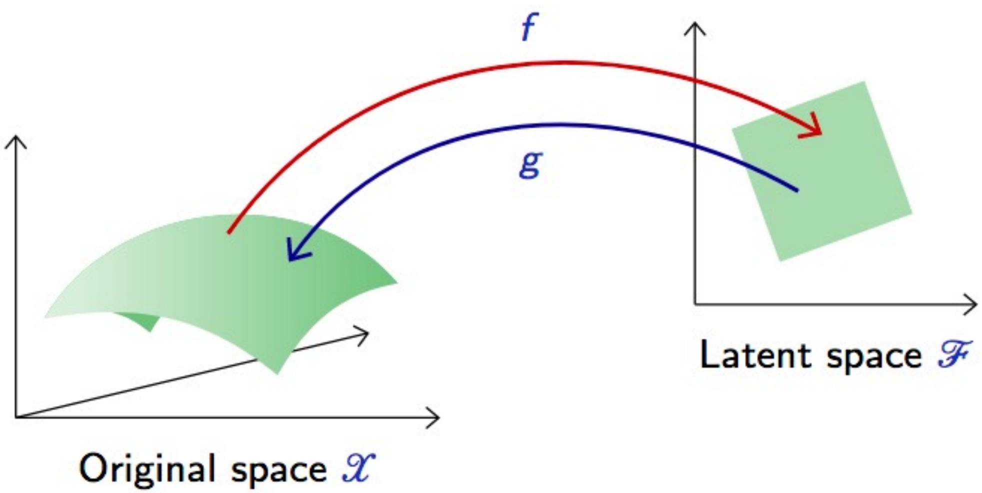
- Loss: mean *reconstruction loss* (MSE) between data and encoded-decoded data

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

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

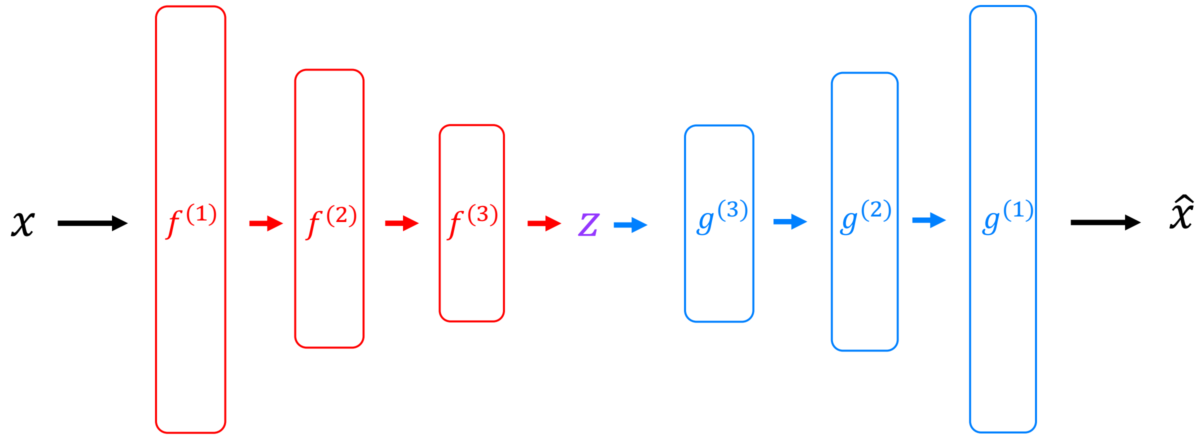
- NOTE: if $f_\theta(x)$ and $g_\psi(z)$ are linear, optimal solution given by Principle Components Analysis

Autoencoder Mappings



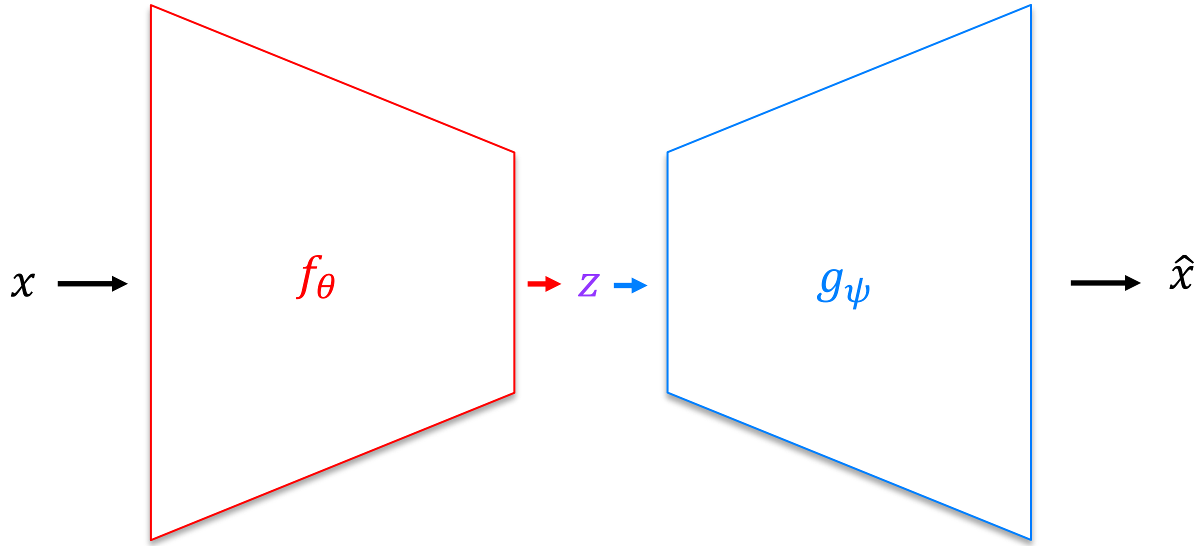
- If the latent space is of lower dimension, the autoencoder has to capture a “good” parametrization, and in particular dependencies between components

Deep Autoencoder



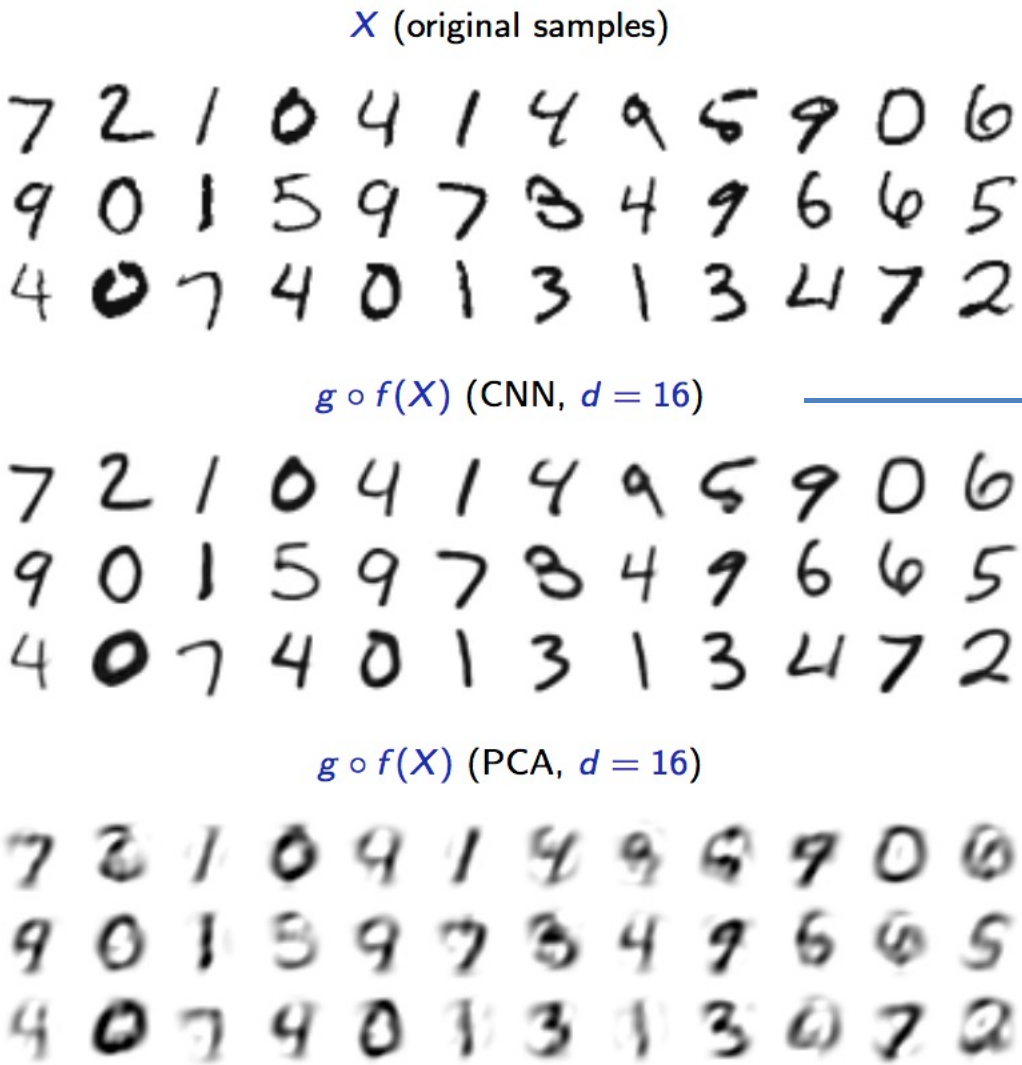
- When f_θ and g_ψ are multiple neural network layers, can learn complex mappings between $\mathcal{X}$ and $\mathcal{F}$
  - f_θ and g_ψ can be Fully Connected, CNNs, RNNs, etc.
  - Choice of network structure will depend on data

Deep Autoencoder



- When f_θ and g_ψ are multiple neural network layers, can learn complex mappings between $\mathcal{X}$ and $\mathcal{F}$
  - f_θ and g_ψ can be Fully Connected, CNNs, RNNs, etc.
  - Choice of network structure will depend on data

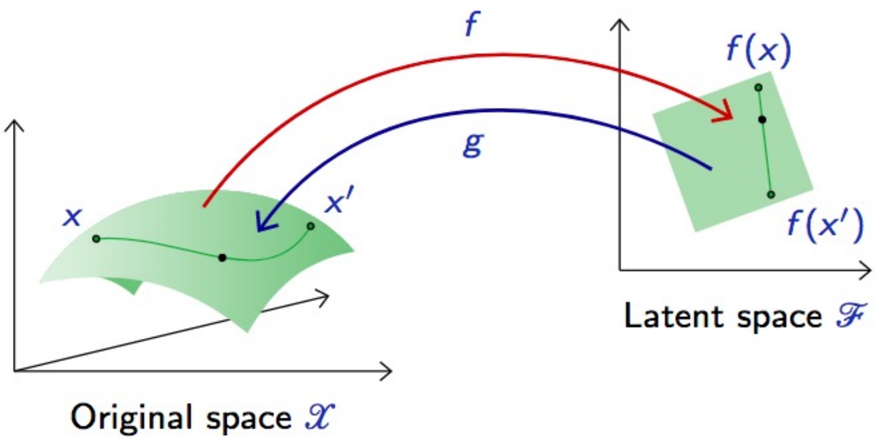
Deep Convolutional Autoencoder



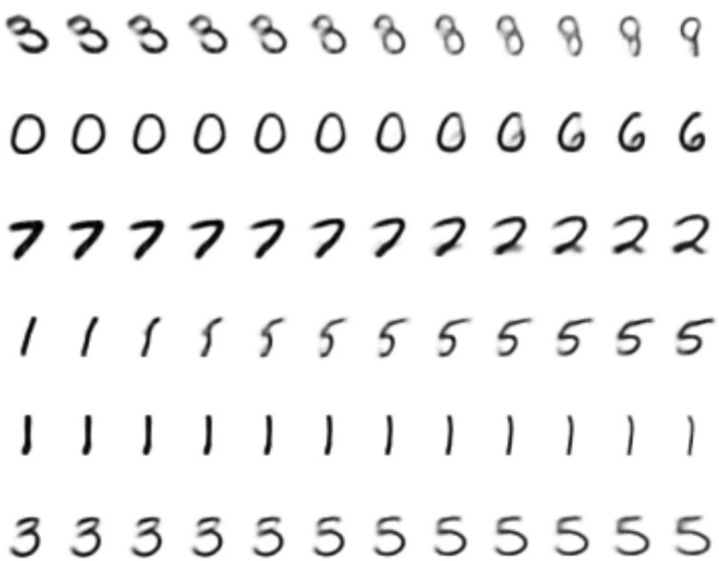
f_θ and g_ψ are each 5 convolutional layers

Interpolating in Latent Space

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

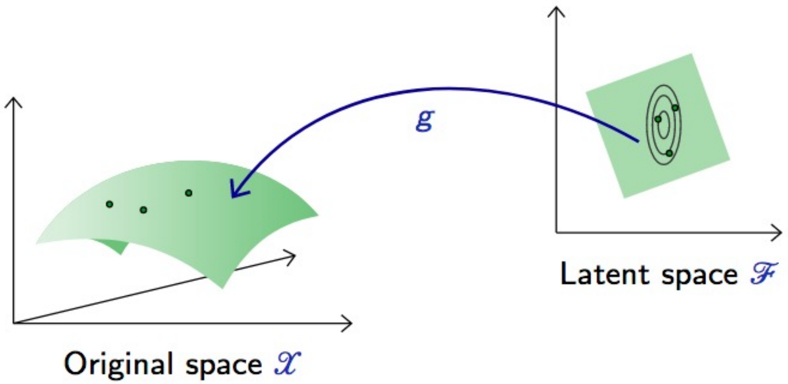


Autoencoder interpolation ($d = 8$)



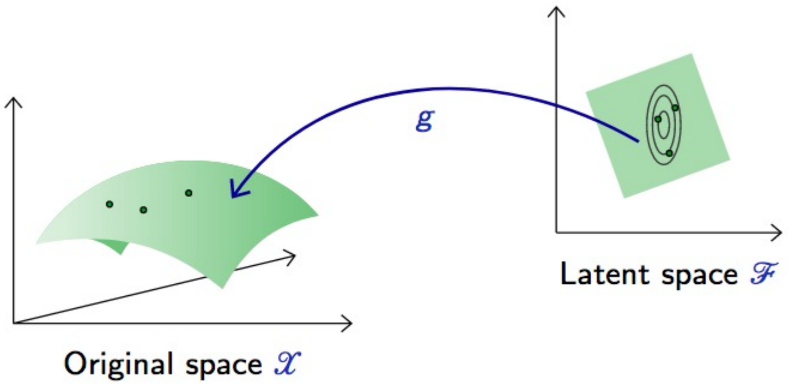
Can We Generate Data with Decoder?

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

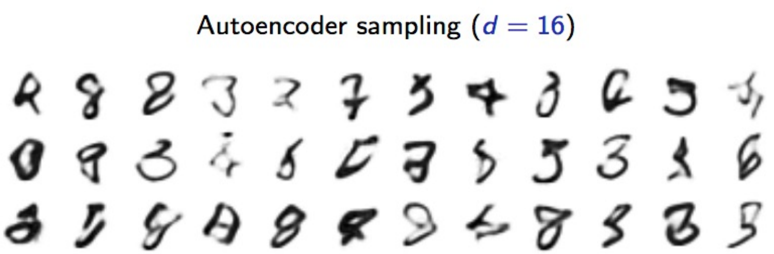


Can We Generate Data with Decoder?

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

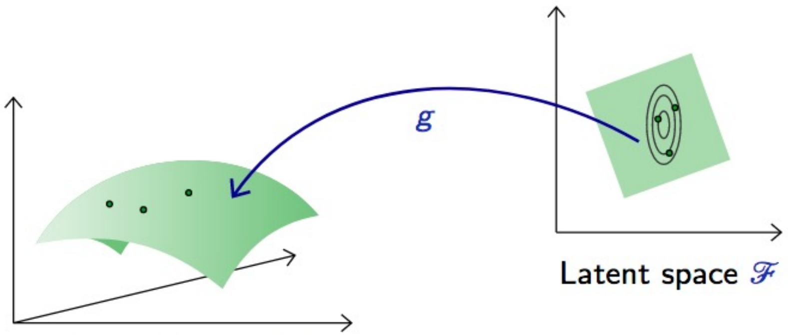


- What distribution to sample from in latent space?
  - Try Gaussian with mean and variance from data



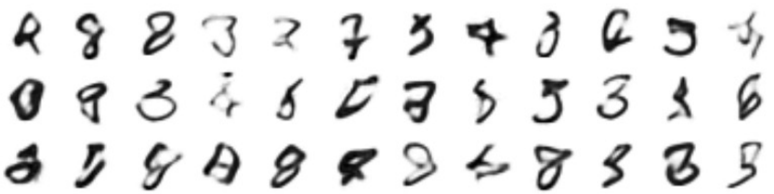
Can We Generate Data with Decoder?

- 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

Autoencoder sampling ($d = 16$)



- Don't know the right latent space density

Generative Models

Generative Models

- Generative models aim to:
  - Learn a distribution $p(x)$ that explains 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

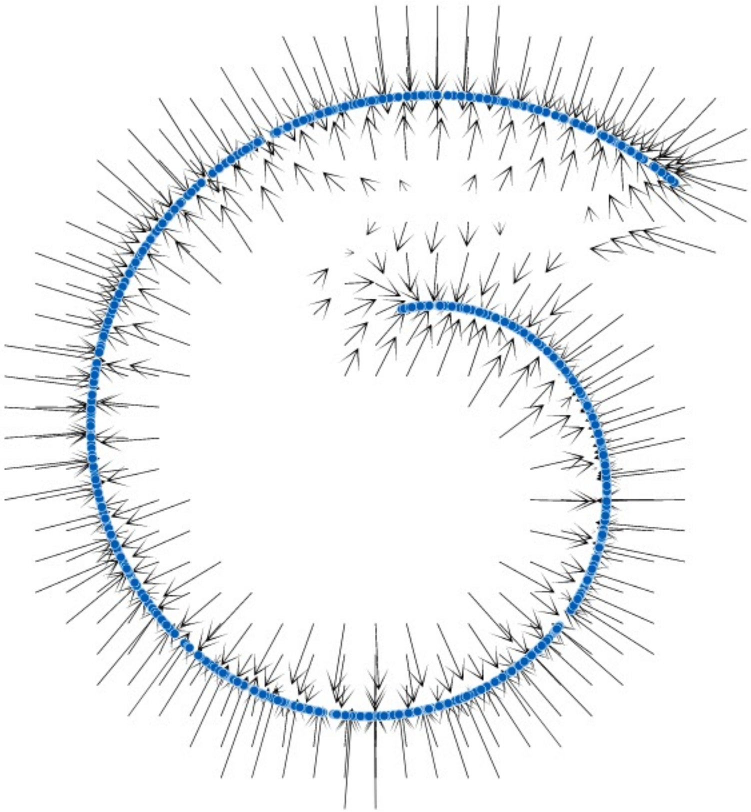
Denoising Autoencoders

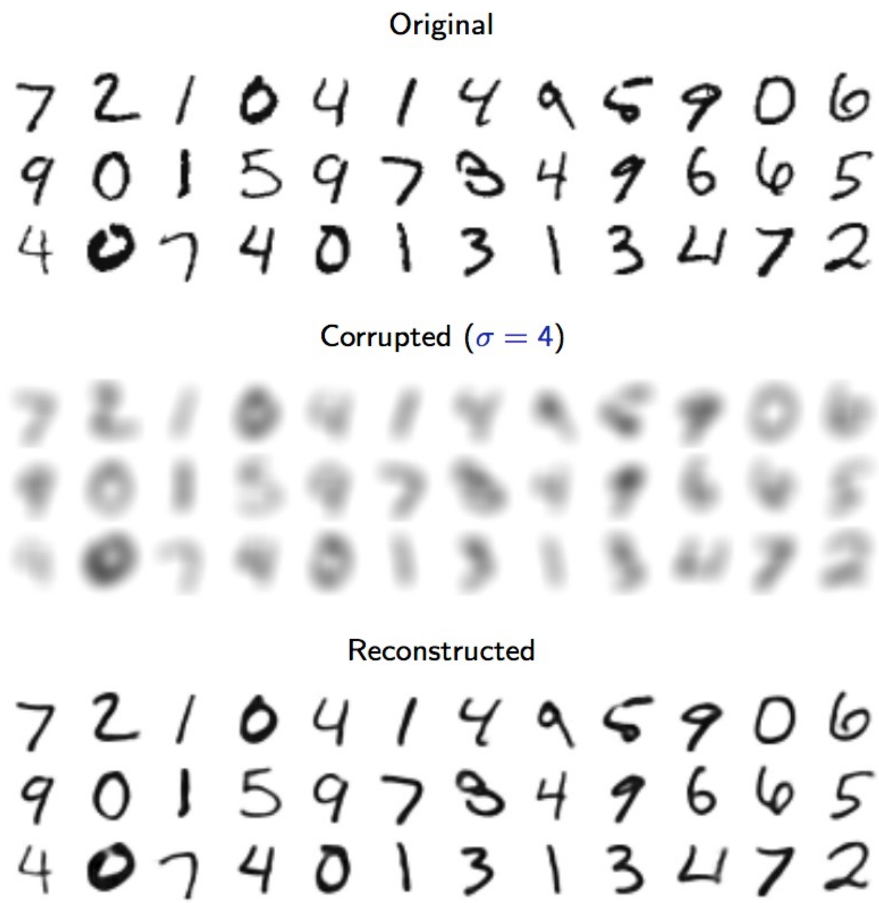
- Learn a mapping from corrupted data space $\widetilde{X}$ back to original data space
  - Mapping $\phi_w(\widetilde{X}) = X$
  - ϕ_w will be a neural network with parameters w
- Loss:

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

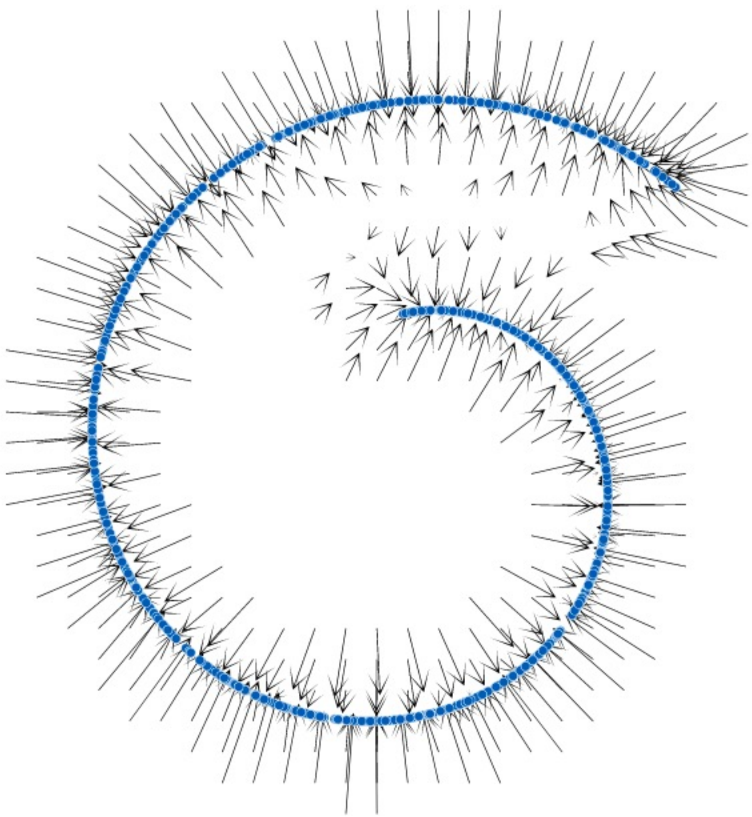
Perturbation, e.g. Gaussian noise

Denoising Autoencoders Examples





Denoising Autoencoders Examples



- Autoencoder learns the average behavior
- What if we care about these variations?
- Can we add a notion of variation in the autoencoder?

From Deterministic to Probabilistic Autoencoder

- Consider probabilistic relationship between data and latent variables

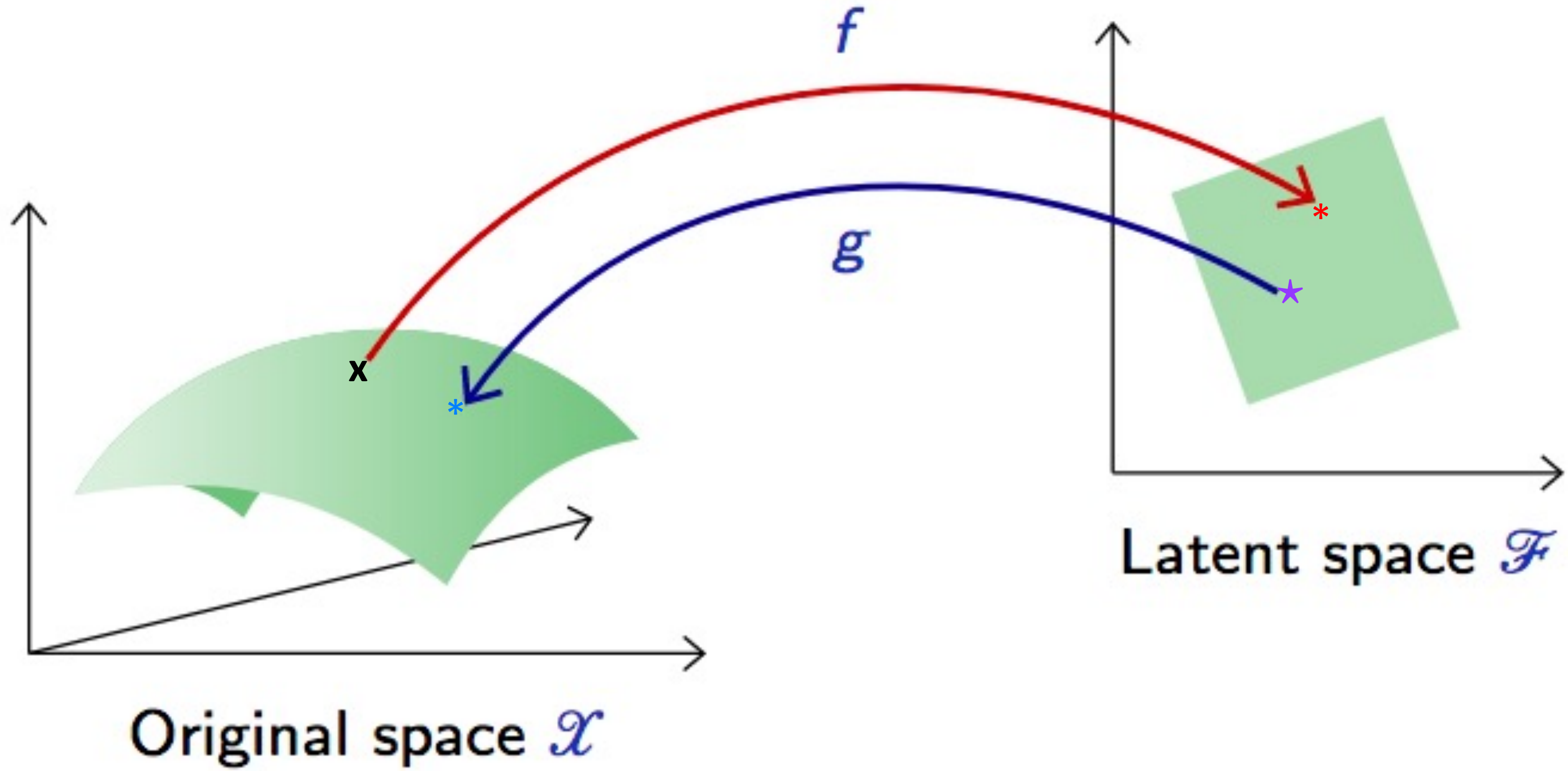
$$x, z \sim p(x, z) = p(x|z)p(z)$$

- Autoencoding

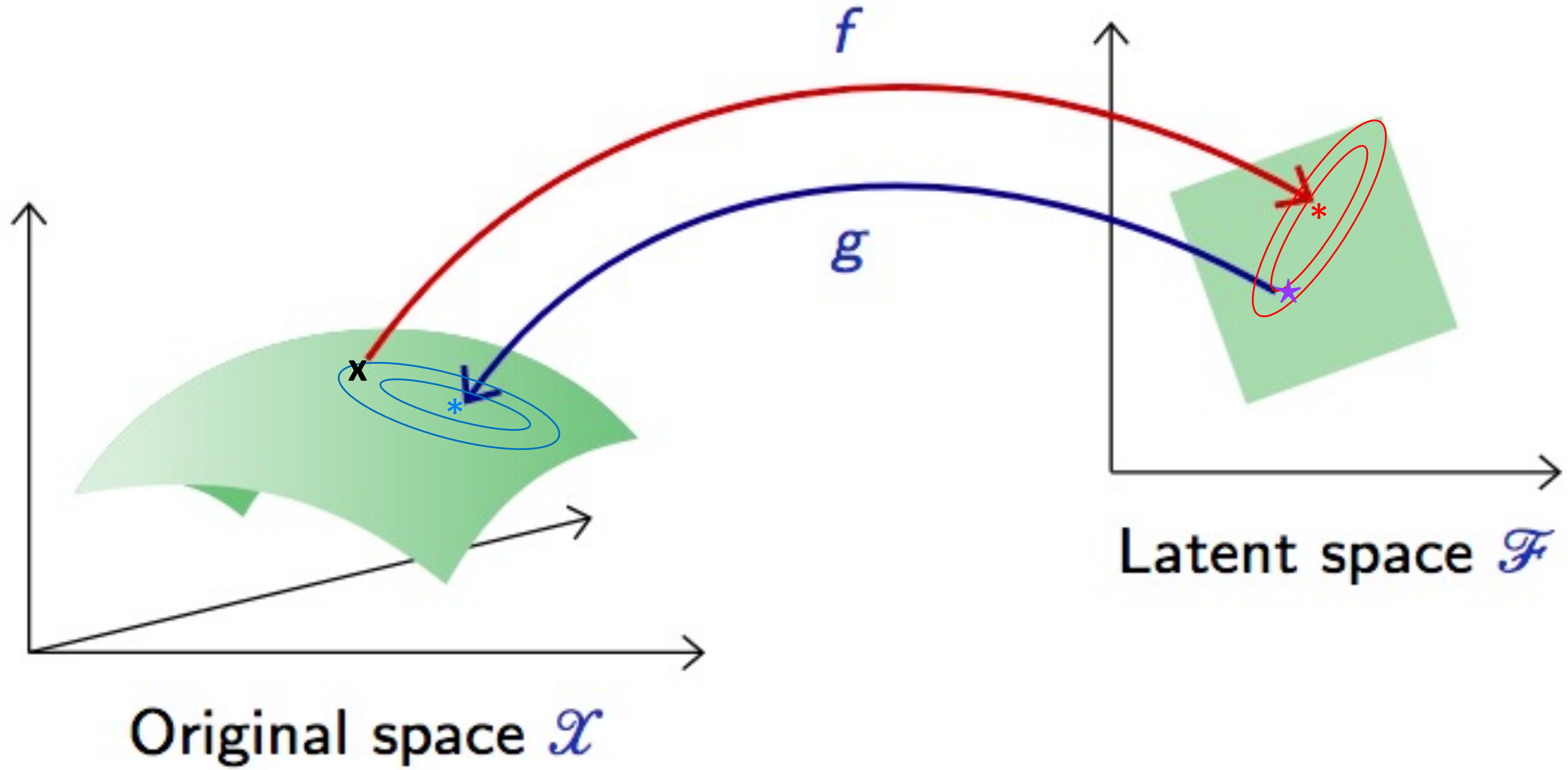
$$x \to q(z|x) \xrightarrow[sample]{} z \to p(x|z)$$

  - Choose simple prior distribution
  - **Encoder:** Learn what latents can produced data: $q(z|x)$
  - **Decoder:** Learn what data is produced by latent: $p(x|z)$

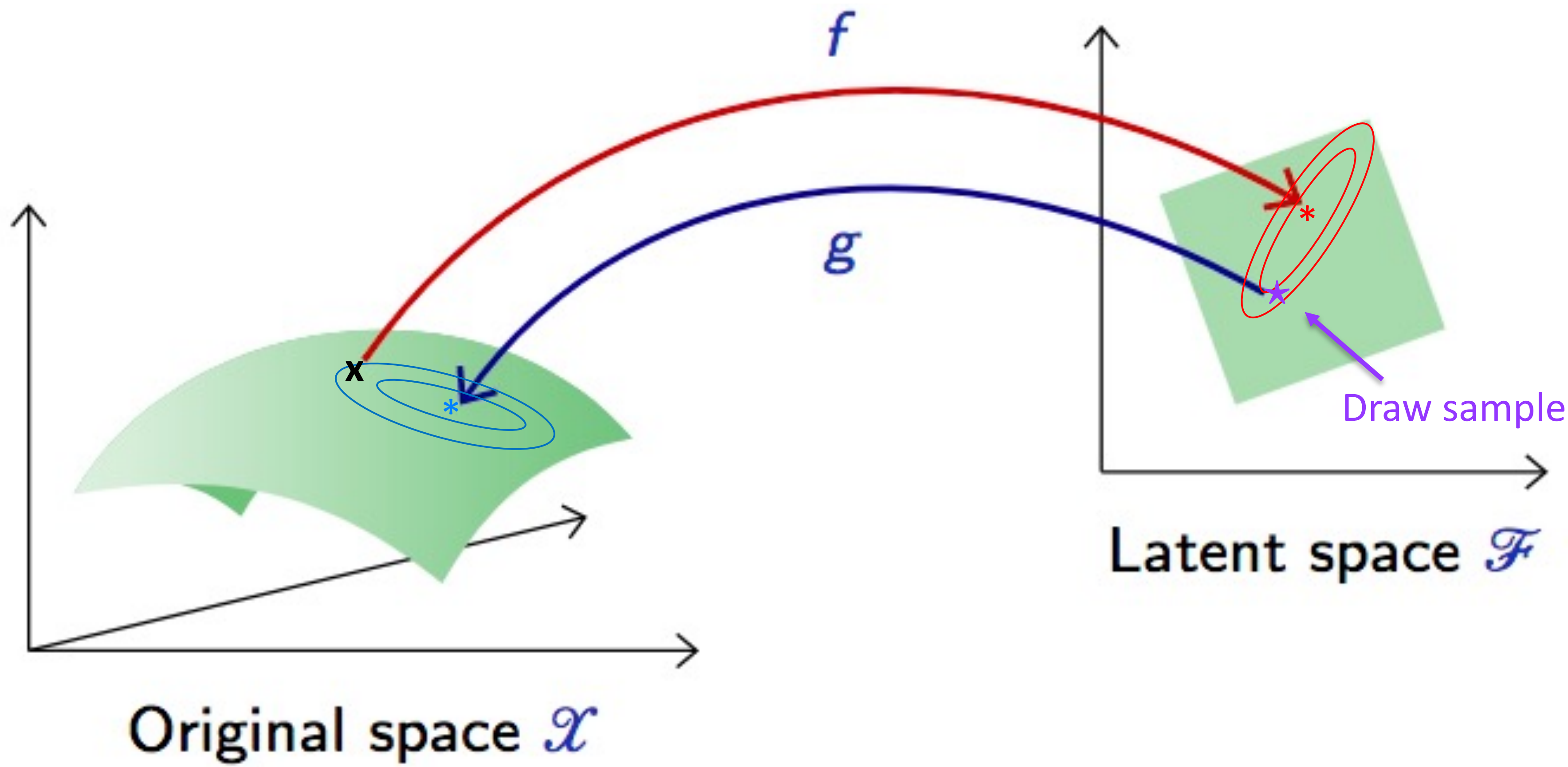
Autoencoder



Variational Autoencoder



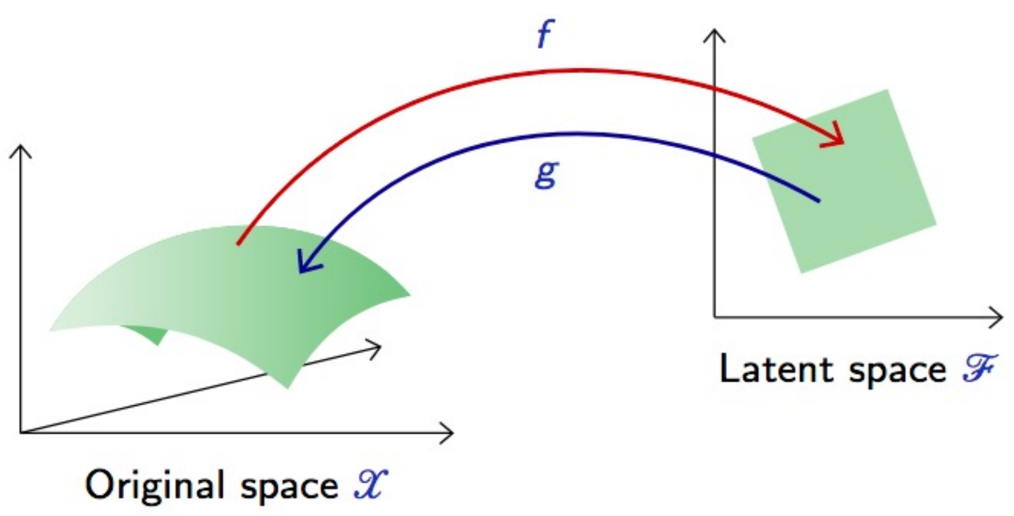
Variational Autoencoder



Encoding

- Typical encoder maps input x to "average" point in latent space

$$f(x) = \mu(x)$$

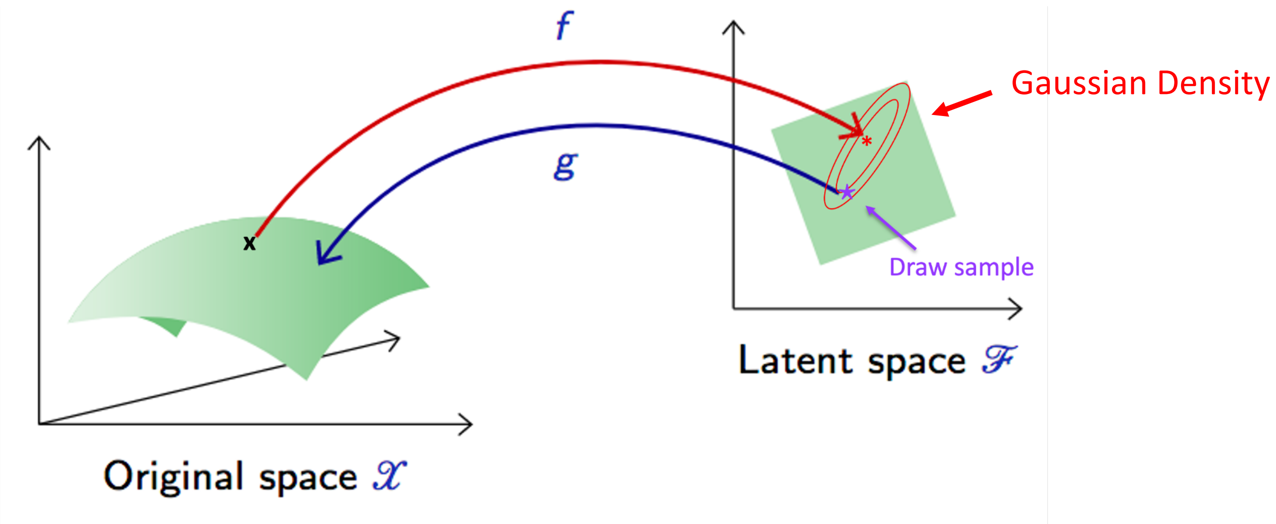


Encoding

- A VAE Encoder has two outputs: mean & variance function

$$f_\psi(x) = \{\mu_\psi(x), \sigma_\psi(x)\}$$

ψ are parameters of the NN



Encoding

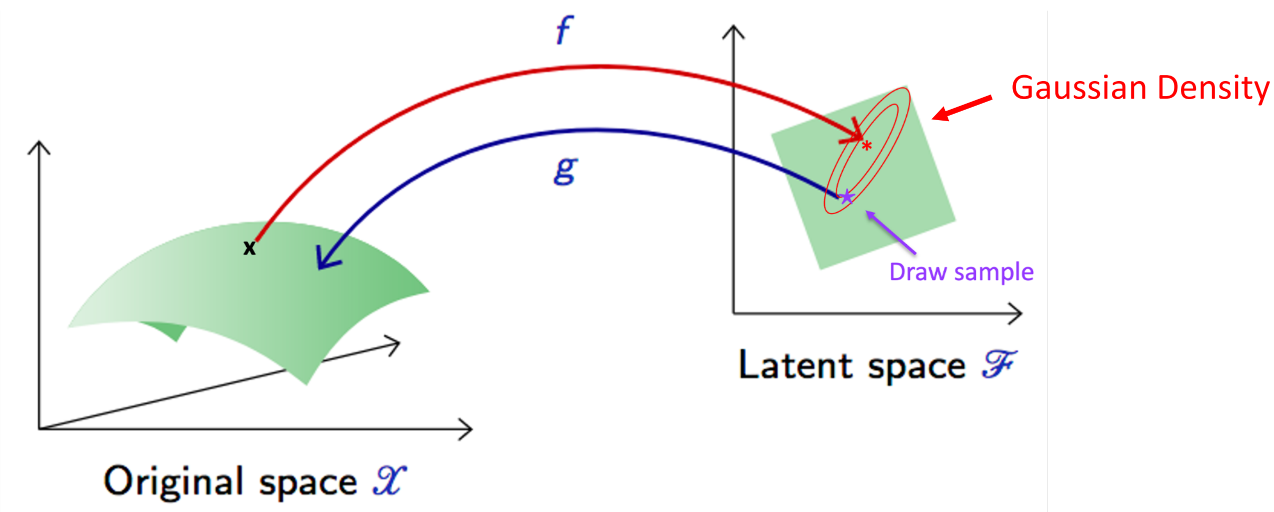
- A VAE Encoder has two outputs: mean & variance function

$$f_\psi(x) = \{\mu_\psi(x), \sigma_\psi(x)\}$$

ψ are parameters of the NN

- What is the probability of a point in latent space?

$$p_\psi(z|x) = N(z \mid \mu_\psi(x), \sigma_\psi(x))$$



Encoding

- A VAE Encoder has two outputs: mean & variance function

$$f_\psi(x) = \{\mu_\psi(x), \sigma_\psi(x)\}$$

ψ are parameters of the NN

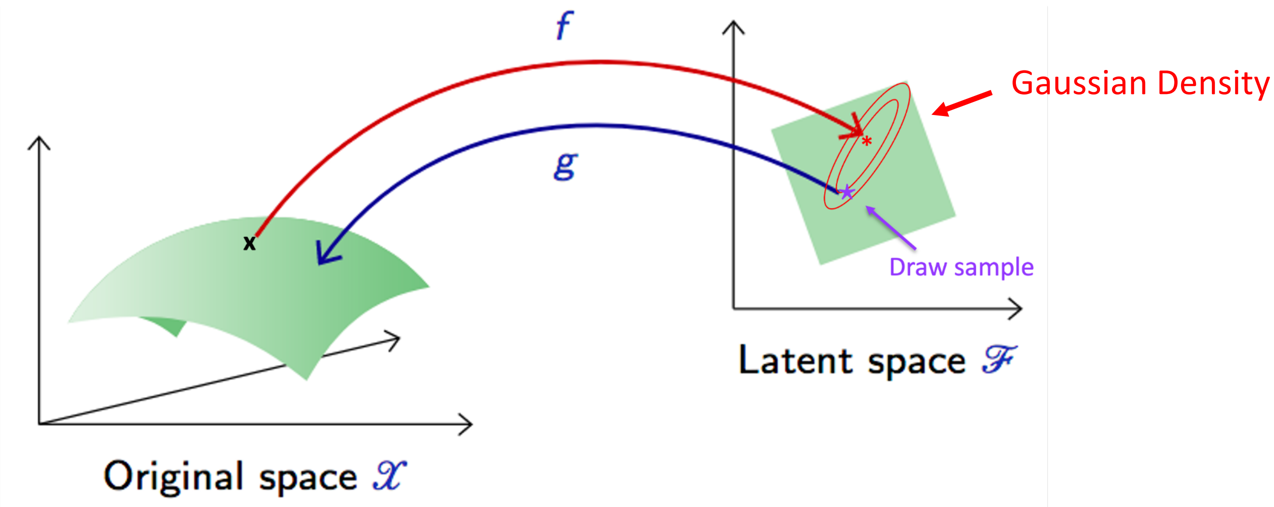
- What is the probability of a point in latent space?

$$p_\psi(z|x) = N(z \mid \mu_\psi(x), \sigma_\psi(x))$$

- How do we draw a sample in latent space?

$$z = \sigma_\psi(x) * \epsilon + \mu_\psi(x) \qquad \epsilon \sim N(0, I)$$

Re-parameterization trick



Encoding

- A VAE Encoder has two outputs: mean & variance function

$$f_\psi(x) = \{\mu_\psi(x), \sigma_\psi(x)\}$$

ψ are parameters of the NN

- What is the probability of a point in latent space?

$$p_\psi(z|x) = N(z \mid \mu_\psi(x), \sigma_\psi(x))$$

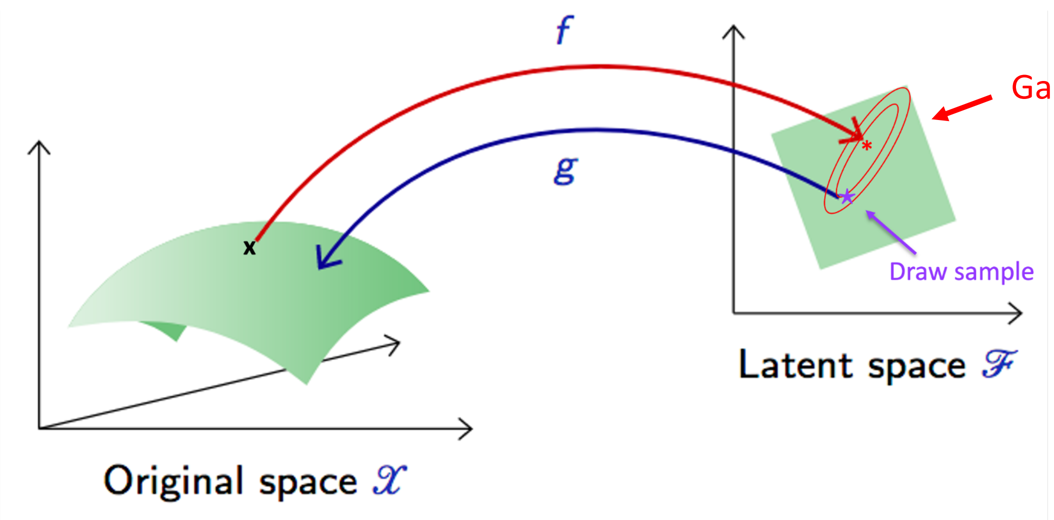
- How do we draw a sample in latent space?

$$z = \sigma_\psi(x) * \epsilon + \mu_\psi(x) \qquad \epsilon \sim N(0, I)$$

Re-parameterization trick

NOTE:
Could have chosen different density and use NN to predict params…

As long as we can sample using re-parameterization



Decoding

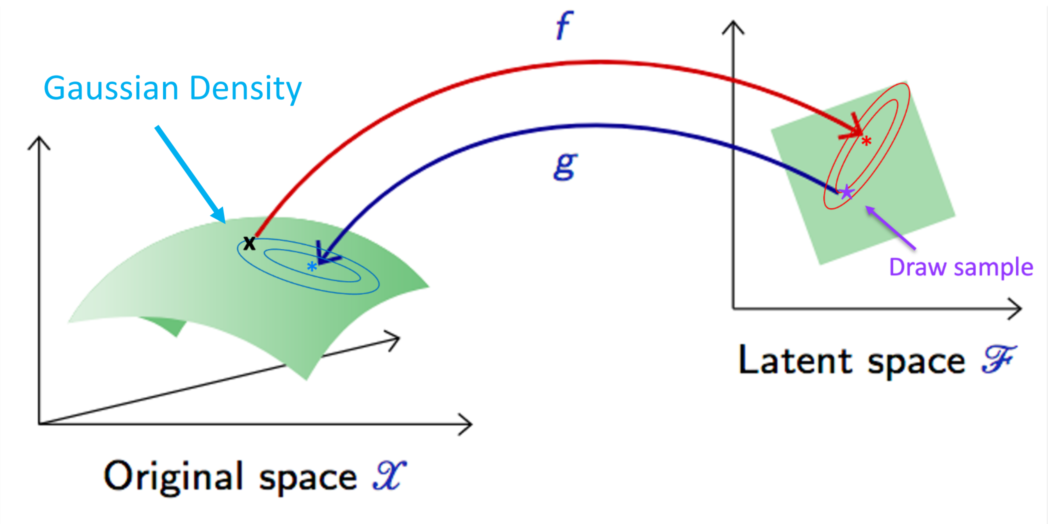
- Same approach, VAE decoder has two outputs

$$g_\theta(z) = \{\mu_\theta(z), \sigma_\theta(z)\}$$

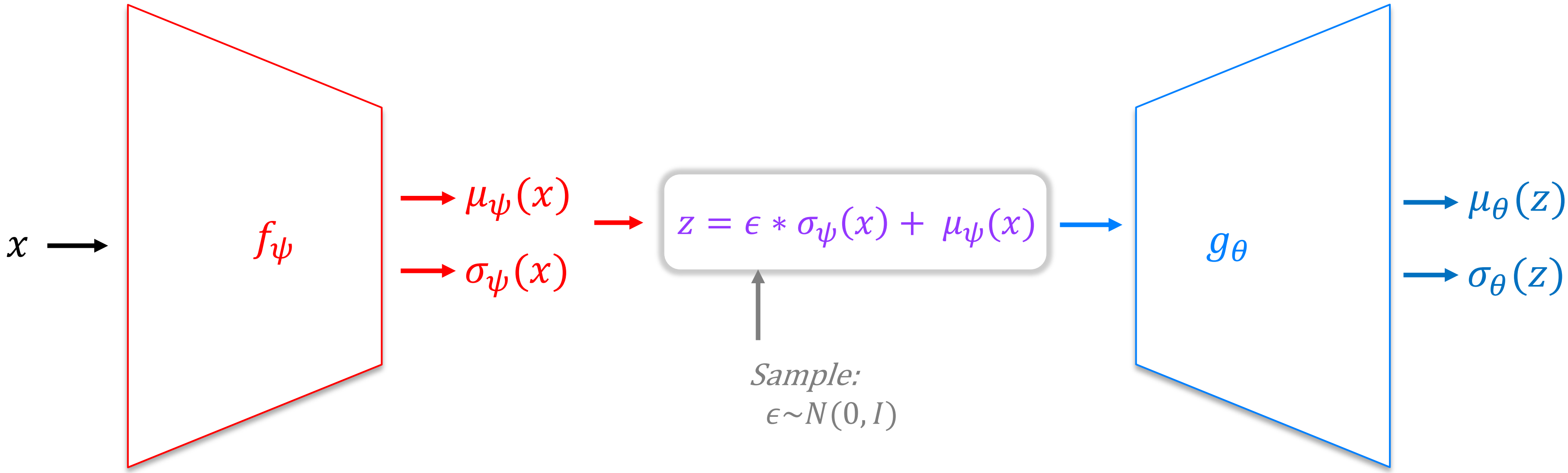
θ are parameters of the NN

- Likelihood of an observation x

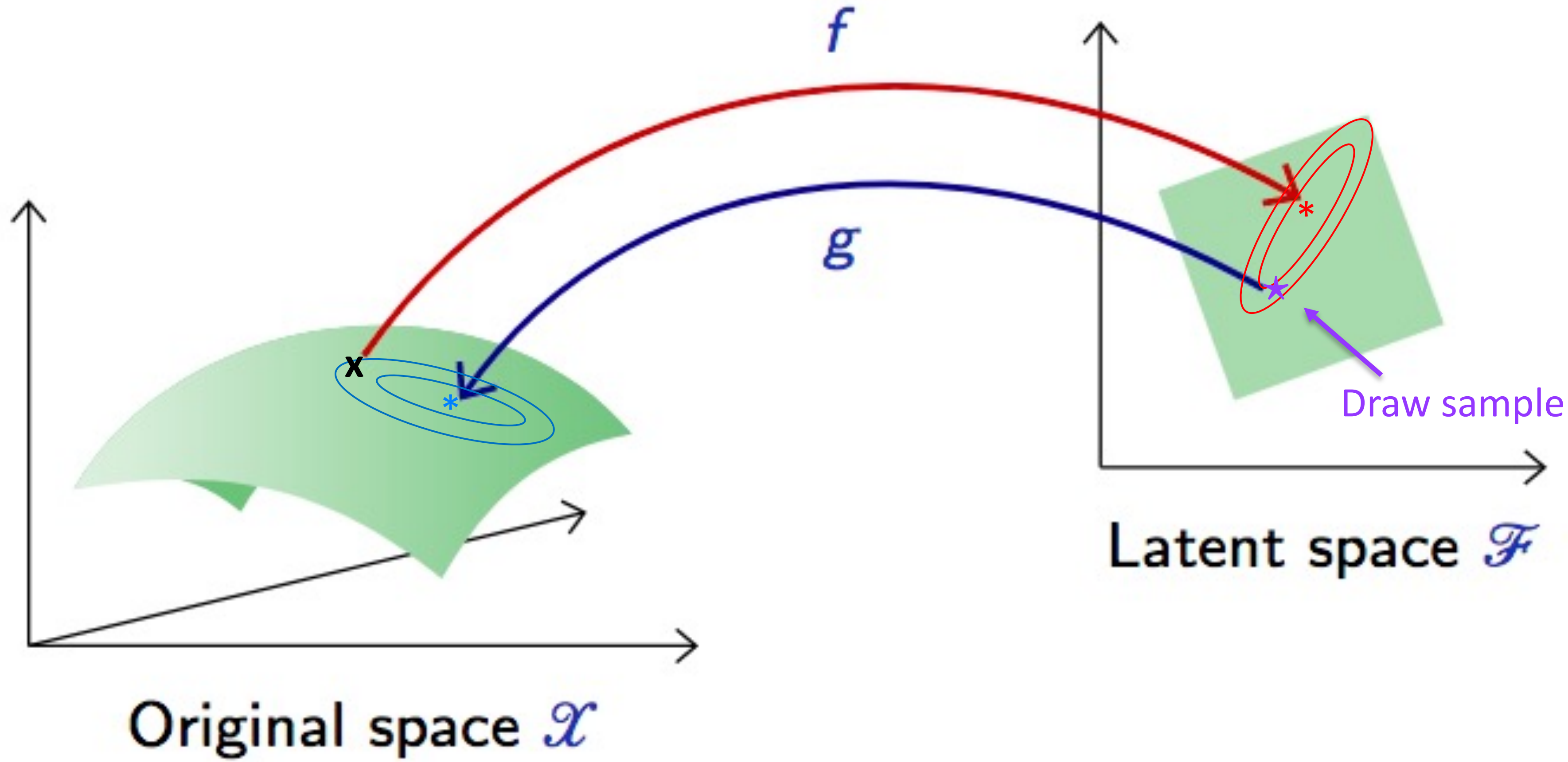
$$p_\theta(x|z) = N(x \mid \mu_\theta(z), \sigma_\theta(z))$$



Variational Autoencoder



What is the Loss for Training?



Reconstruction Loss: Maximize expected likelihood of decoding x from encodings of x

$$L_{reco} = \mathbb{E}_{z\sim q(z|x)}[\log p(x|z)] \approx \frac{1}{N} \sum_{z_i\sim q(z|x)} \log p(x|z_i)$$

Variational Autoencoder Training Loss

- $L_{reco} = \frac{1}{N} \sum_{z \sim q_\psi(z|x)} \log p_\theta(x|z_i)$

- Note that

$$\log p(x|z) = -\log \sigma_\theta(z) - \frac{\left(x - \mu_\theta(z)\right)^2}{\sigma_\theta(z)^2} + const$$

This looks almost exactly like the Autoencoder Loss

Which was a Mean Squared Error $\left(x - f\left(g(x)\right)\right)^2$

Here we have $z \equiv z_\psi(x)$

Variational Autoencoder Training Loss

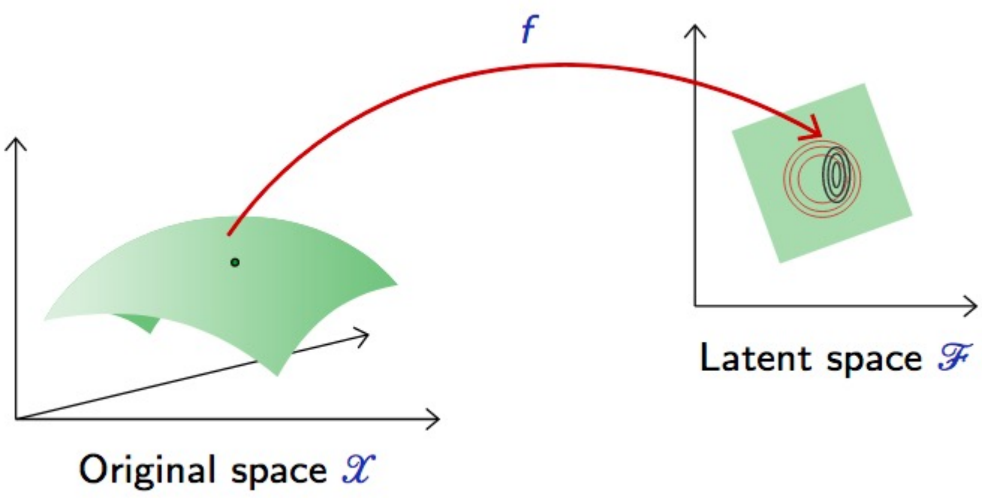
- $L_{reco} = \frac{1}{N} \sum_{z \sim q_\psi(z|x)} \log p_\theta(x|z_i)$

- What about encoder? How do we make sure it doesn't collapse around each point (i.e. only predict mean)

Variational Autoencoder Training Loss

- $L_{reco} = \frac{1}{N} \sum_{z \sim q_{\psi}(z|x)} \log p_{\theta}(x|z_i)$

- Use prior $p(z)$ for the latent space distribution,
  need to ensure the encoder is consistent with prior



Variational Autoencoder Training Loss

- $L_{reco} = \frac{1}{N} \sum_{z \sim q_\psi(z|x)} \log p_\theta(x|z_i)$

- Use prior $p(z)$ for the latent space distribution, **need to ensure the encoder is consistent with prior**

- Constrain difference between distributions with **Kullback–Leibler divergence**

$$D_{KL}[q(z|x)|p(z)] = \mathbb{E}_{q(z|x)}\left[\log \frac{q(z|x)}{p(z)}\right] = \int q(z|x) \log \frac{q(z|x)}{p(z)}\, dz$$

  - $D_{KL}[q|p] \geq 0$ and is only 0 when $q = p$

Variational Autoencoder Training Loss

- $L_{reco} = \frac{1}{N} \sum_{z \sim q_\psi(z|x)} \log p_\theta(x|z_i)$

- Use prior $p(z)$ for the latent space distribution,
need to ensure the encoder is consistent with prior

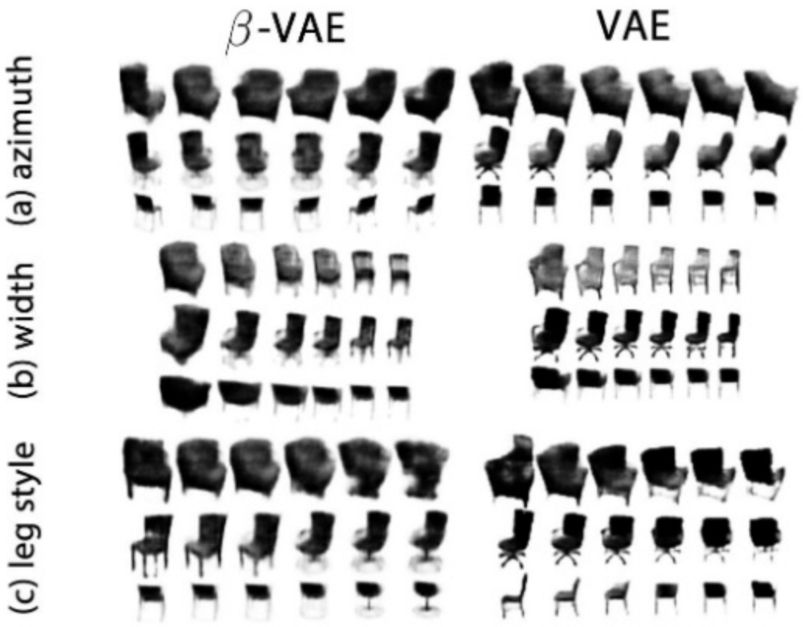
- VAE full objective

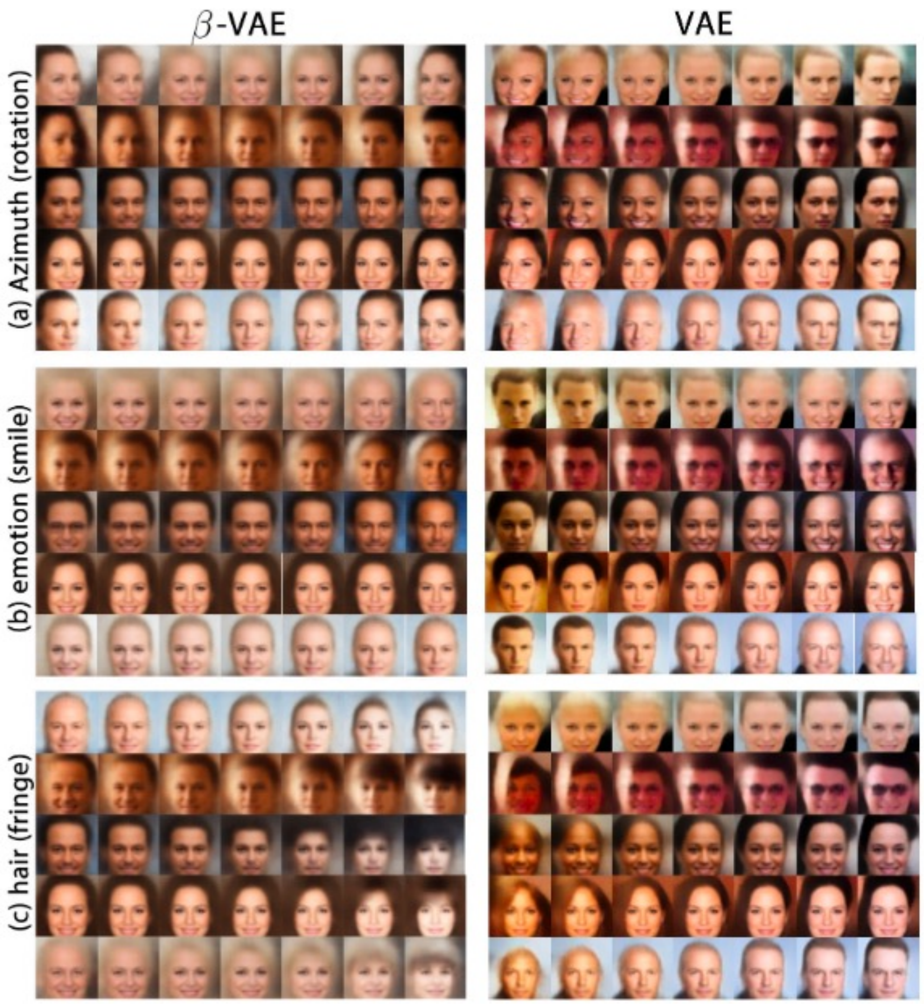
$$\max_{\theta,\psi} L(\theta, \psi) = \max_{\theta,\psi} \left[\mathbb{E}_{q_\psi(z|x)}[\log p_\theta(x|z)] - D_{KL}[q_\psi(z|x) | p(z)] \right]$$

Reconstruction Loss

Regularization of Encoder

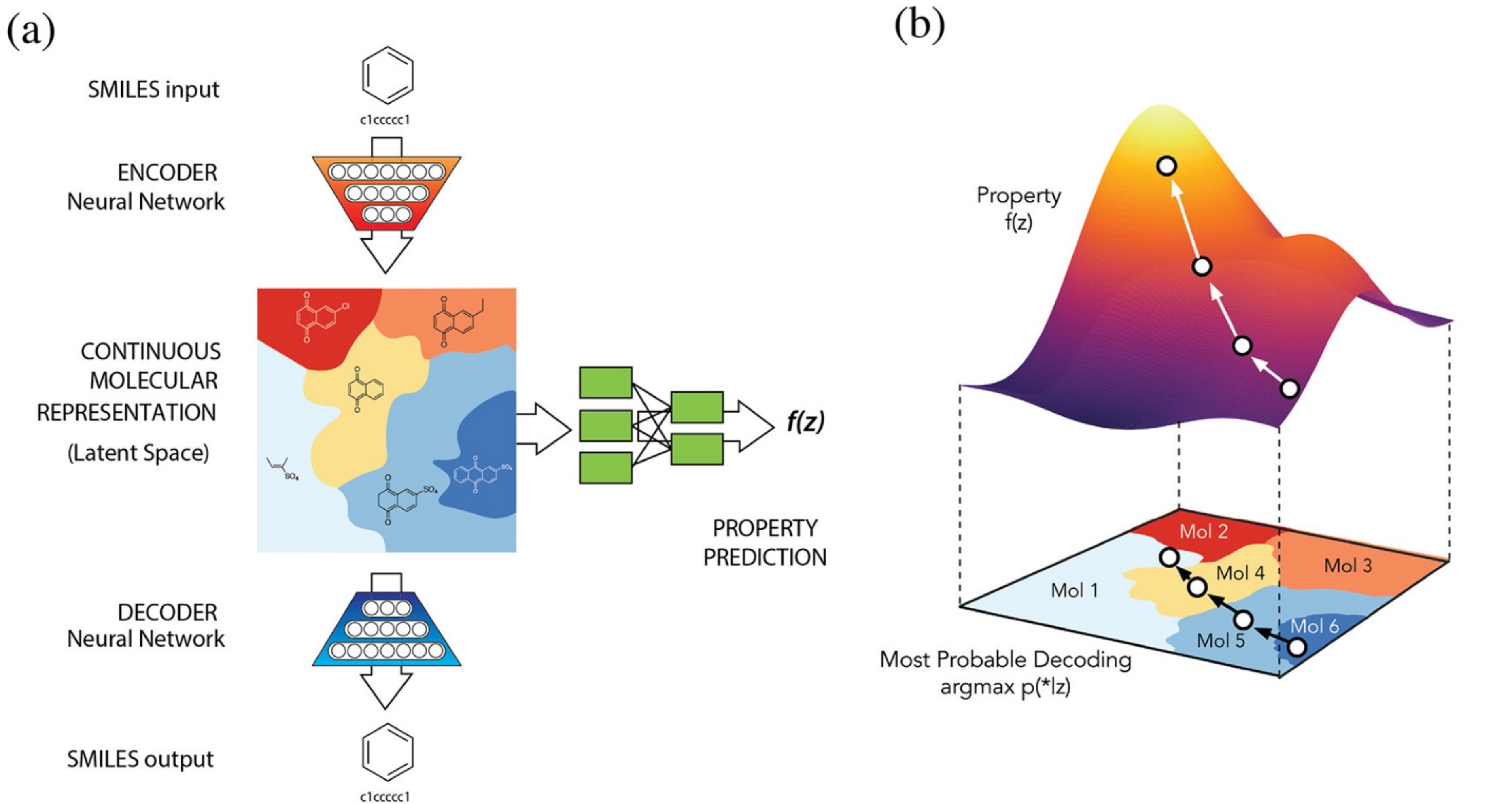
Examples





Higgins et al., [2017]

Examples



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

Slide credit: [G. Louppe]

Another Way To Do Generative Modeling…

Another Way To Do Generative Modeling…

- Formulate 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:
  1. A *generator* that can produce samples
  2. 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 generated distribution $p_\theta(x)$ is, but we can sample from it → *Implicit Model*

Generative Adversarial Network (GAN)

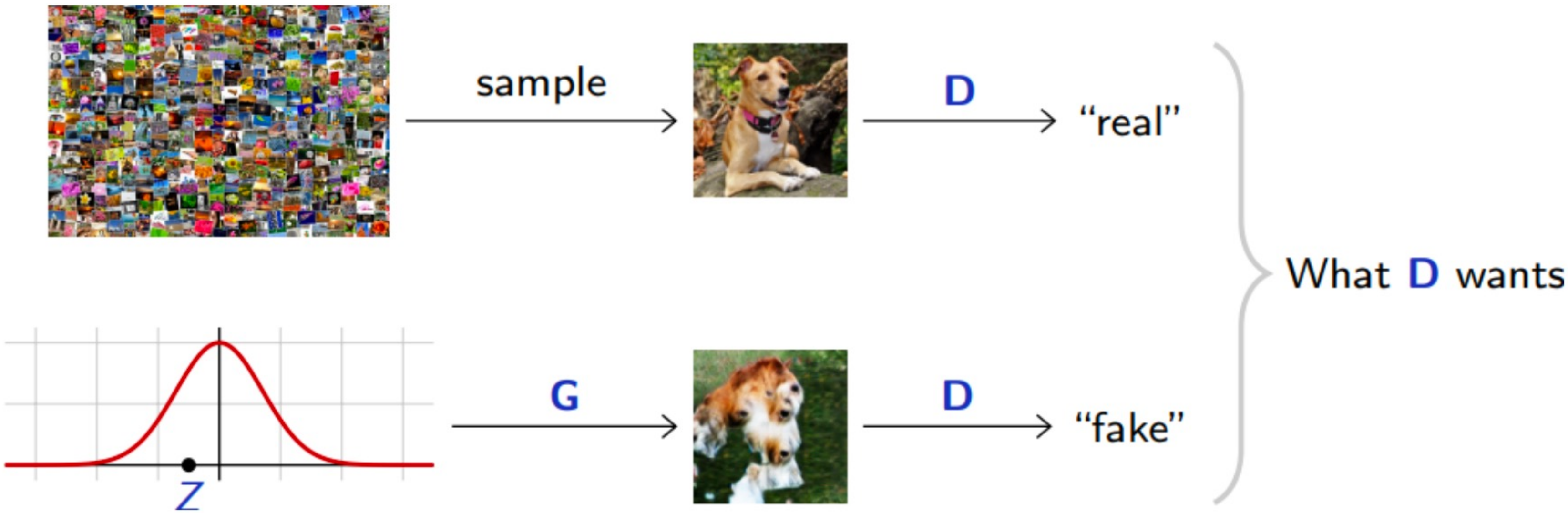
- **Generator network $g_\theta(z)$** with parameters θ
  - Map sample from known $p(z)$ to sample in data space

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

  - We don't know what the generated 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 $p(y = real \mid x)$
  - This classifier is our measure of *not too far from the real data*

GAN Setup



- Generator's goal is to produce *fake* data that tricks the discriminator to think it is *real* data

- Discriminator wants to miss-classify data as real or fake as little as possible

- The setup is *adversarial* because the two networks have 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: $z_i \sim p(z)$

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 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 cross entropy

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

$$= -\frac{1}{2N}\sum_{i=1}^{N}\Big[\log d_\phi(x_i) + \log(1-d_\phi(g_\theta(z_i))\ )\Big]$$

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 cross entropy

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

$$= -\frac{1}{2N}\sum_{i=1}^{N}\Big[\log d_\phi(x_i) + \log(1-d_\phi(g_\theta(z_i))\ )\Big]$$

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

GAN Objective

- However, generator isn't fixed… have to train it!

GAN Objective

- However, generator isn't fixed… have to train it!

- 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

- However, generator isn't fixed… have to train it!

- 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 a perfect discriminator, a good generator will confuse discriminator and $V(\phi, \theta)$ will be low

GAN Objective

- However, generator isn't fixed… have to train it!

- 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 a perfect discriminator, a good generator will confuse discriminator and $V(\phi,\theta)$ will be low

- So our optimization goal becomes:

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

GAN Objective

- However, generator isn't fixed… have to train it!

- 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 a perfect discriminator, a good generator will confuse discriminator and $V(\phi,\theta)$ will be low

- 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

$$q_{\theta^*}(x) = p_{data}(x)$$

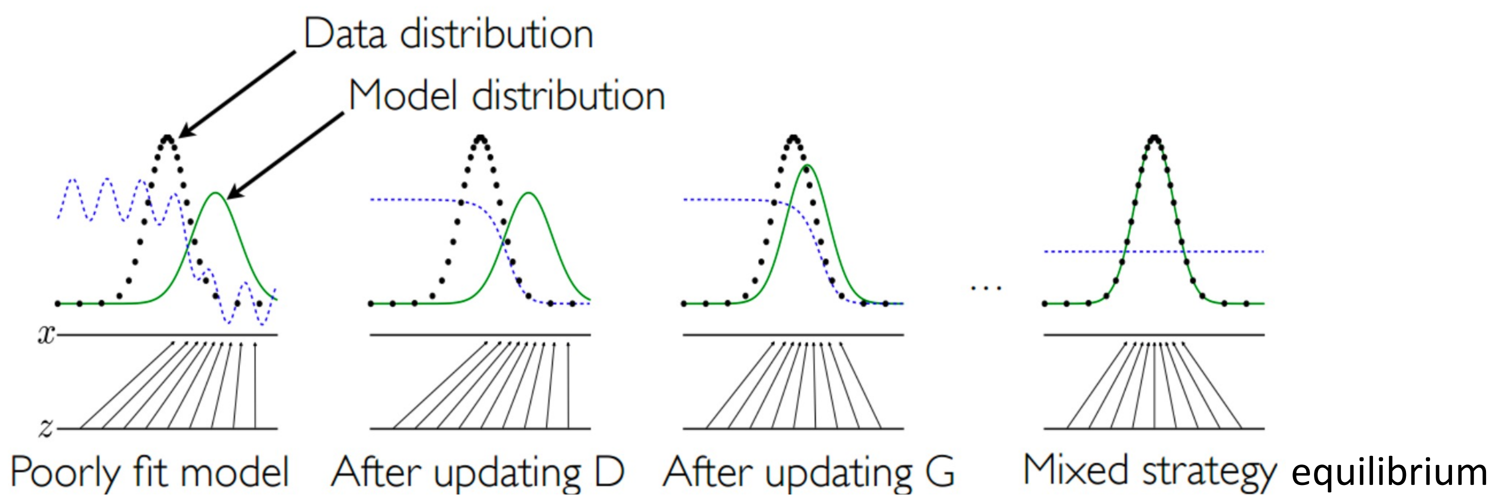
GAN Training

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

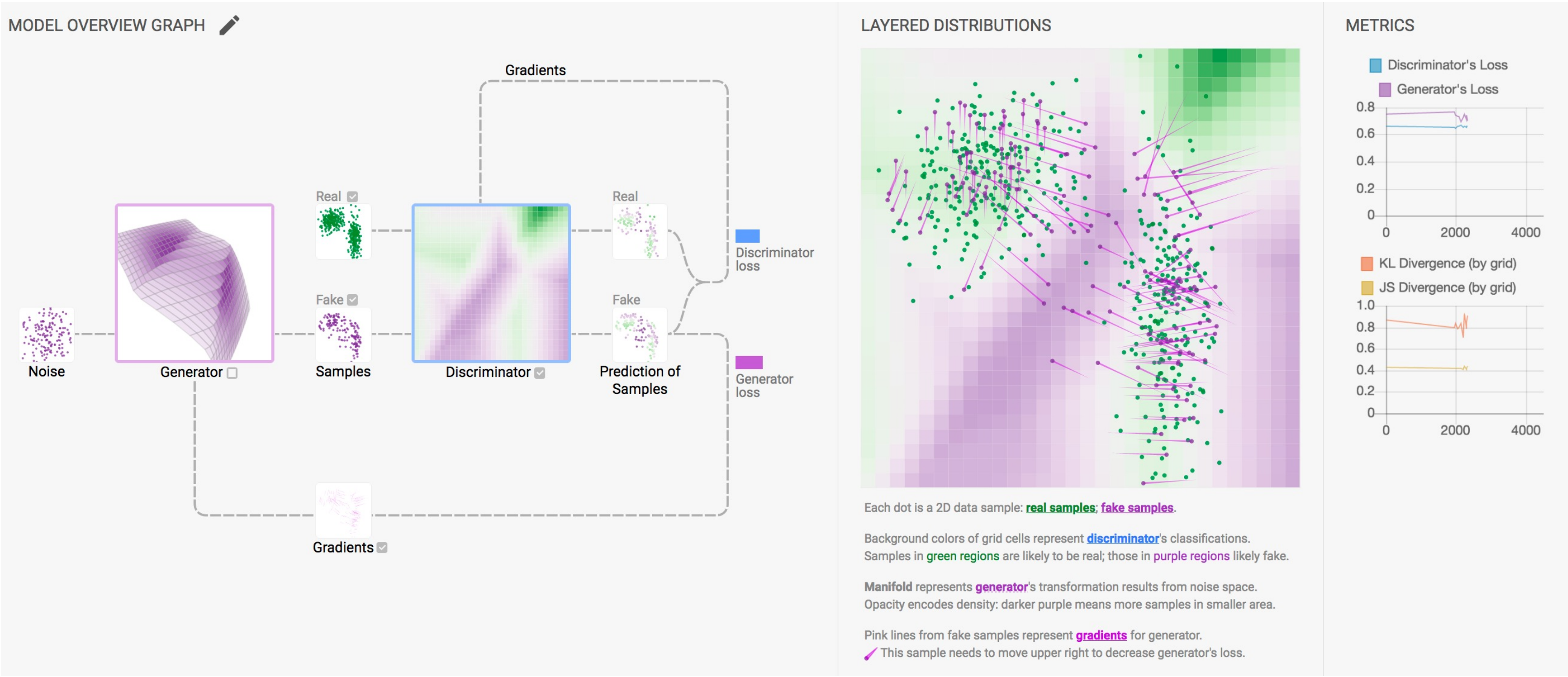
$$\theta \leftarrow \theta - \gamma \nabla_\theta V(\phi, \theta) = \theta - \gamma \frac{\partial V}{\partial d} \frac{\partial (d_\phi)}{\partial g} \frac{\partial g_\theta}{\partial \theta}$$

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

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



GAN Training Example



[GAN Lab Demo]

Examples

Goodfellow et. al., 2014

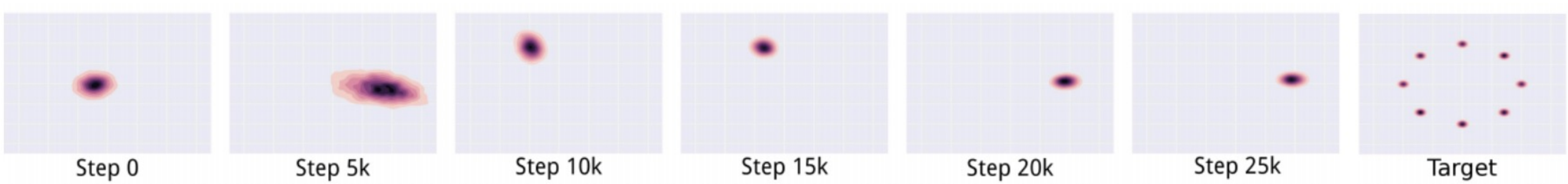
Not so good

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**, when are generated data good enough?



Slide credit: [G. Louppe]

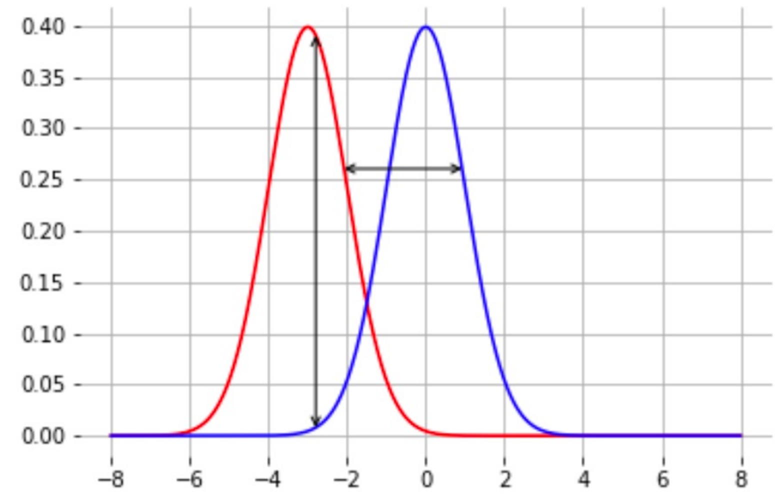
Mode collapse (Metz et al, 2016)

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!



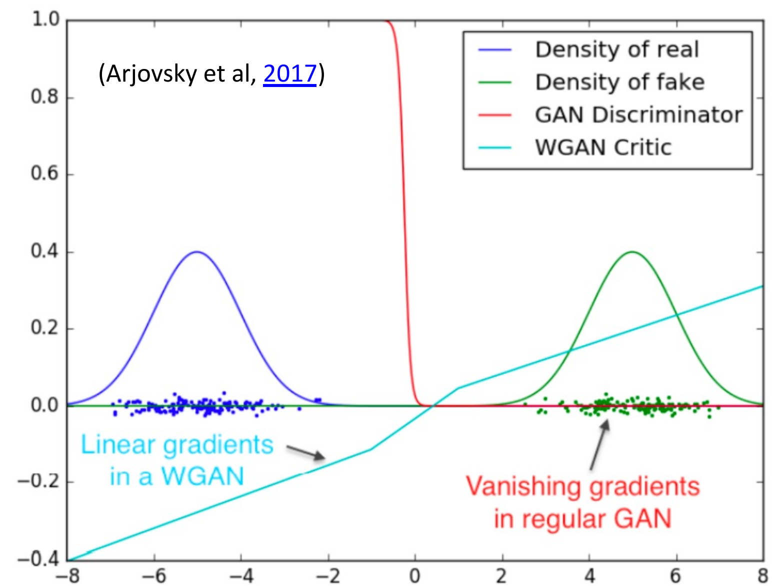
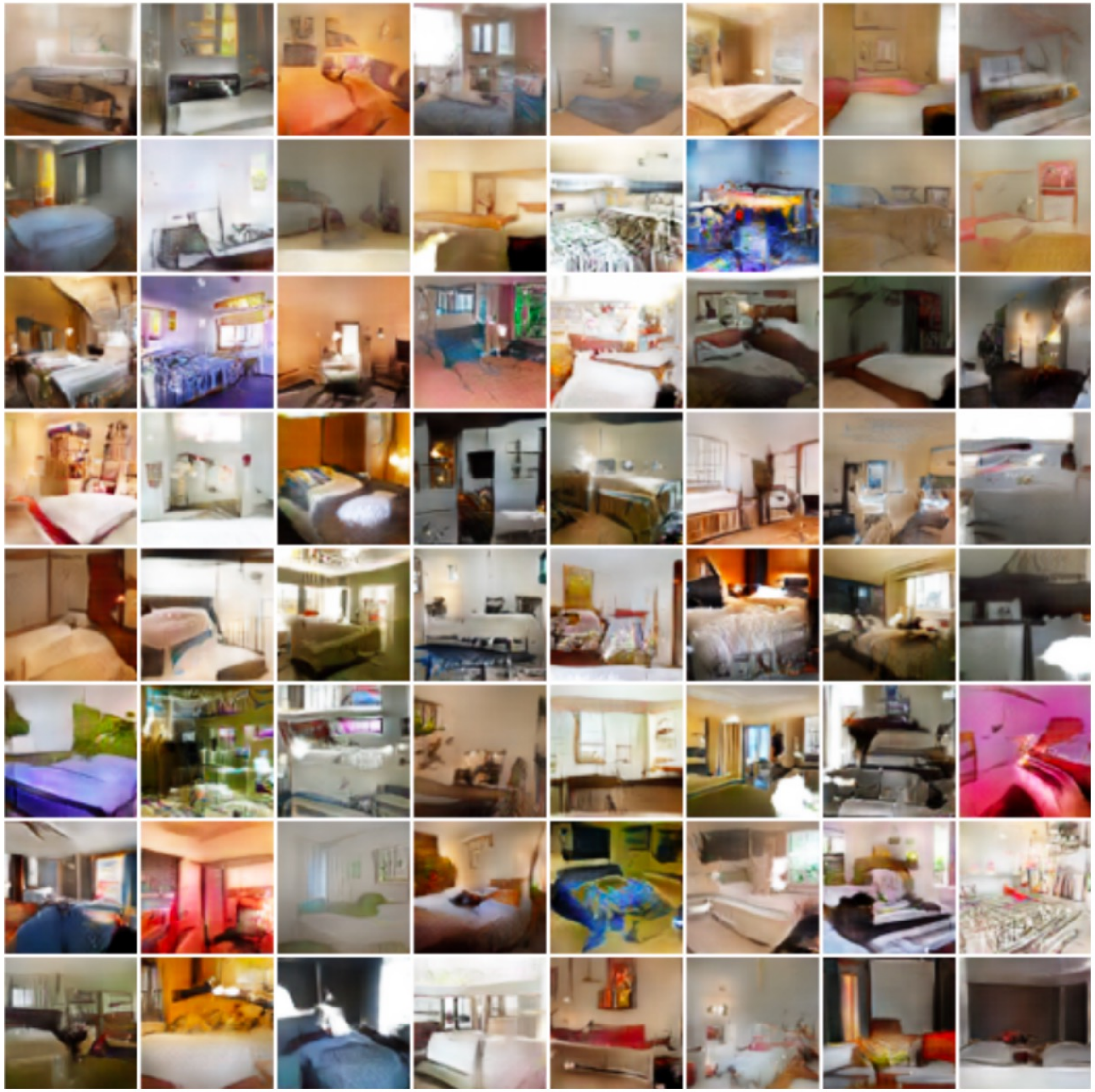


Figure 2: Optimal discriminator and critic when learning to differentiate two Gaussians. As we can see, the discriminator of a minimax GAN saturates and results in vanishing gradients. Our WGAN critic provides very clean gradients on all parts of the space.

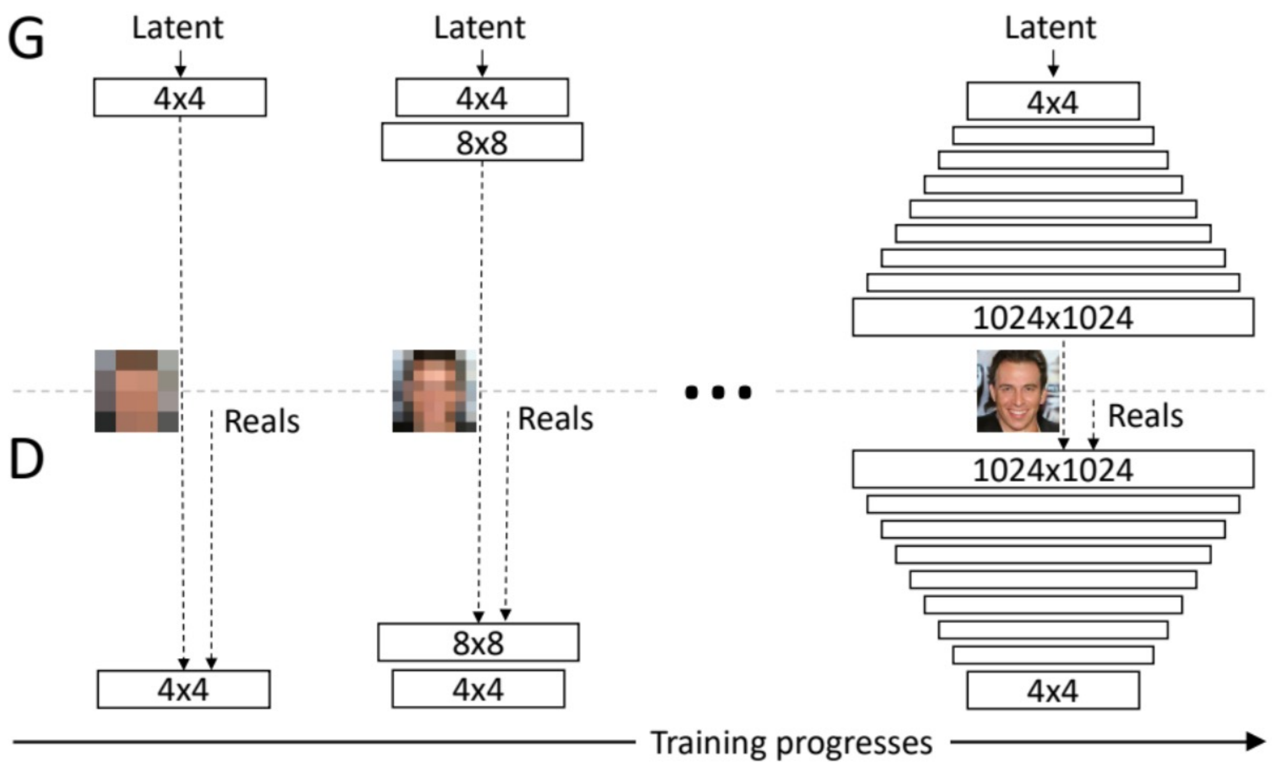
WGAN Examples



(Arjovsky et al, 2017)

Scaling Up

Progressive GAN



(Karras et al, 2017)

Scaling Up

StyleGAN v2

(Karras et al, 2019)

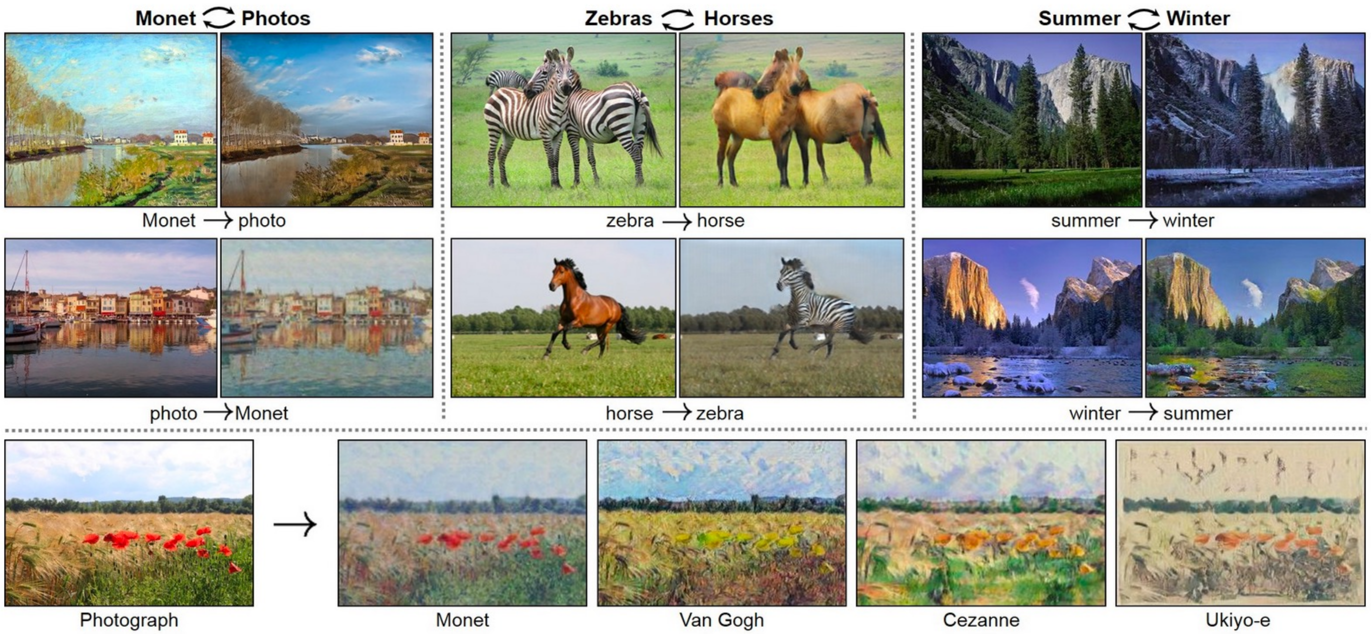
BigGAN

(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)

Summary

- 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 Variational Autoencoder and Generative adversarial Networks

Modeling High Dimensional Data

- Must first determine the question we want to ask, and formulate an appropriate loss function
  - Loss function encodes the quality of model prediction
  - Parameterize models with neural networks

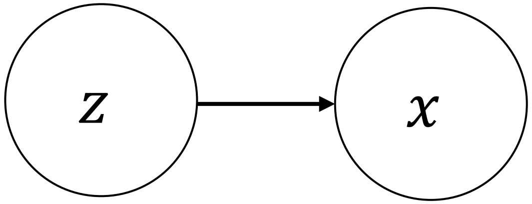
- Will have many of the same theoretical and practical issues as in classification and regression
  - What is the right class and structure of the model (CNN, RNN, graph, etc.) for the data?
  - How do we stably optimize the loss w.r.t. parameters?

- 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

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

How do we design Encoder and Decoder

- Classification / regression models make single predictions…

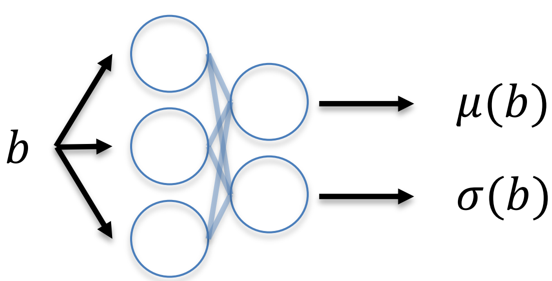
How to model a conditional density $p(a|b)$?

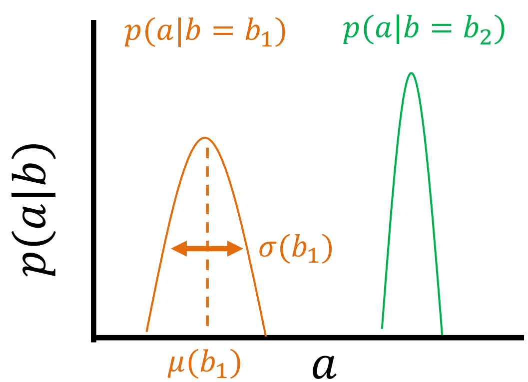
- Assume a known form of density, e.g. normal

$$p(a|b) = \mathcal{N}\big(a; \mu(b), \sigma(b)\big)$$

  – Parameters of density depend on conditioned variable

- Use neural network to model density parameters





The Decoder

- **Decoder**
  - Neural network with parameters θ
  - Input z → output estimate of Gaussian $\mu_\theta(z)$, $\sigma_\theta(z)$

- **Likelihood of a data point x**

$$\log p(x|z) = -\log \sigma_\theta(z) - \frac{\left(x - \mu_\theta(z)\right)^2}{\sigma_\theta(z)^2} + const$$

The Encoder

- **Encoder**
  - Neural network with parameters ψ
  - Input x → outputs estimate of Gaussian $\mu_\psi(x)$, $\sigma_\psi(x)$

- For reconstruction loss:
  - Need a value of z to evaluate decoder!
  - Need to gradient through z to encoder parameters

$$\max_{\theta,\psi} L(\theta,\psi) = \max_{\theta,\psi} \sum_{z_i \sim q_\psi(z|x)} \log p_\theta(x|z_i) - \log\left[\frac{q_\psi(z_i|x)}{p(z_i)}\right]$$

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

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

Explicit Density Estimation with Normalizing Flows

Explicit Density Estimation

- In VAE and GAN we can learn to sample from the distribution…

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

Reminder: Calculus Change of Variables

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

Multivariate:

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

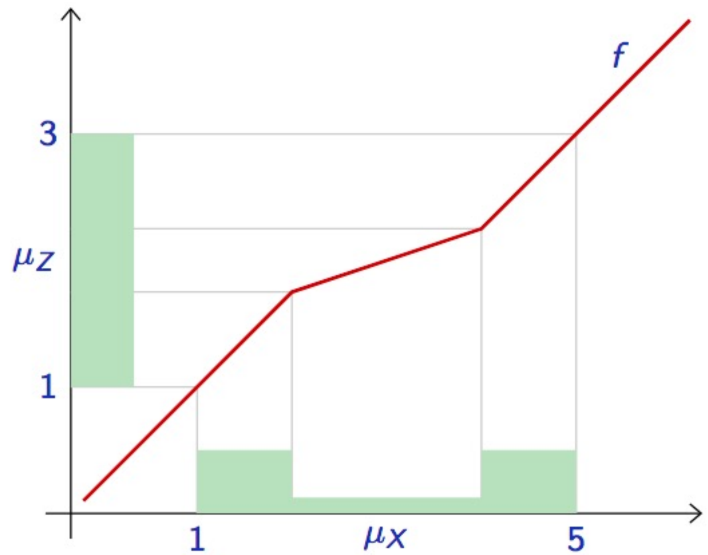
Determinant of Jacobian
of the transformation

→ Change of volume

Change of Variables in Probability

- If f is continuous, invertible, differentiable, and $x = f^{-1}(z) \equiv \phi(z)$ then

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



The term $\left| \det\left(\frac{\partial \phi(\boldsymbol{z})}{d\boldsymbol{z}}\right)^{-1} \right|$ accounts for the local stretching of space

Change of Variables with Neural Networks

- If f is continuous, invertible, differentiable, and $x = f^{-1}(z) \equiv \phi(z)$ then

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

- $x =$ data we want to model, $z =$ known noise
- $\phi_\theta(z)$ will be a neural network with parameters θ
  - Must be continuous, invertible, differentiable
- Output of ϕ is a potential sample x
  - **Learn the right ϕ**: adjust weights θ to maximize data probability (formula above)

Change of Variables with Neural Networks

- If f is continuous, invertible, differentiable, and $x = f^{-1}(z) \equiv \phi(z)$ then

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

- x = data we want to model, z = known noise

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

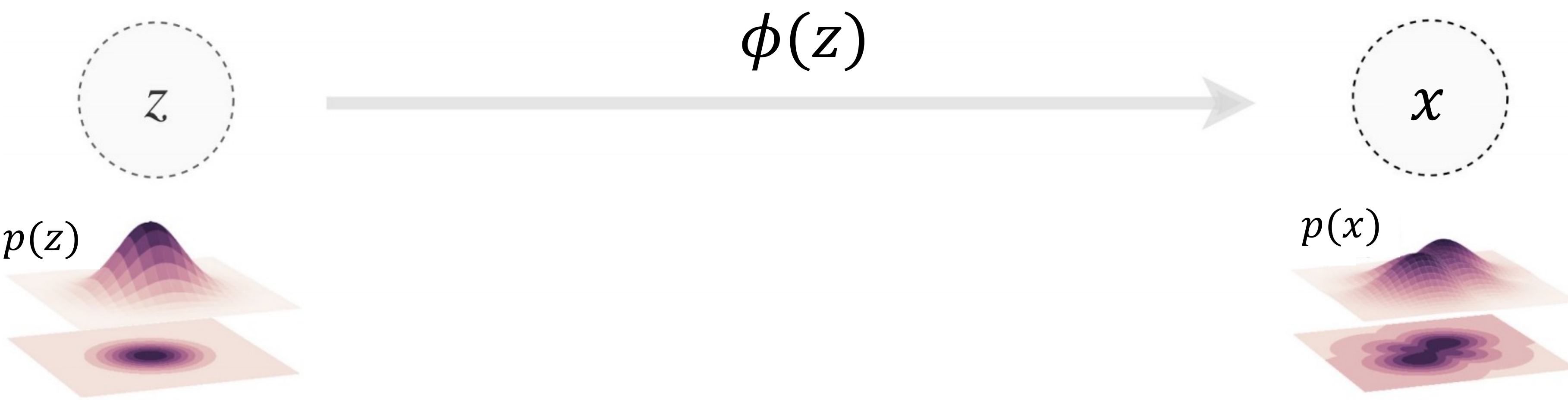
$\Longleftrightarrow$

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

- Calculate the probability of a sample using the formula above

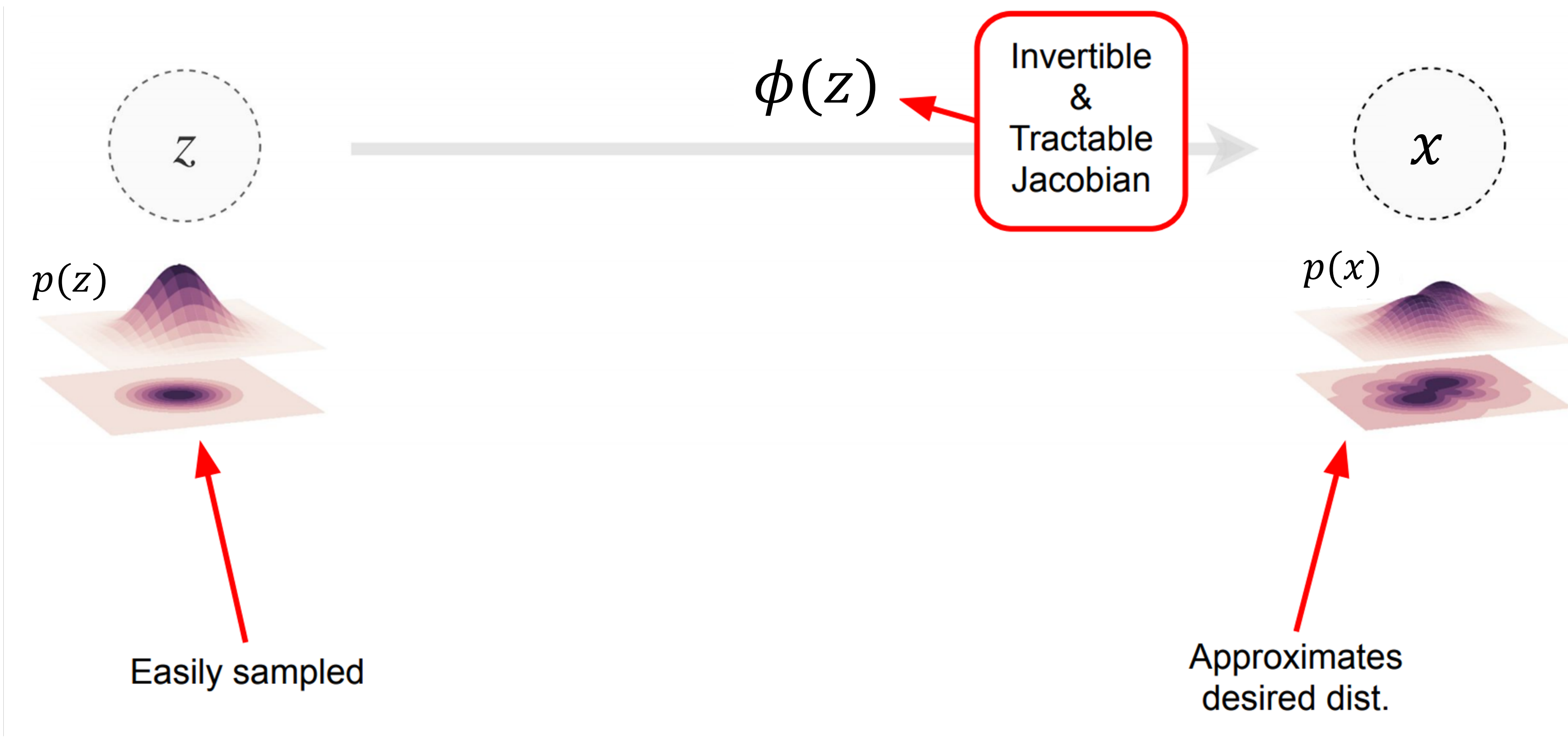
Normalizing Flows

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



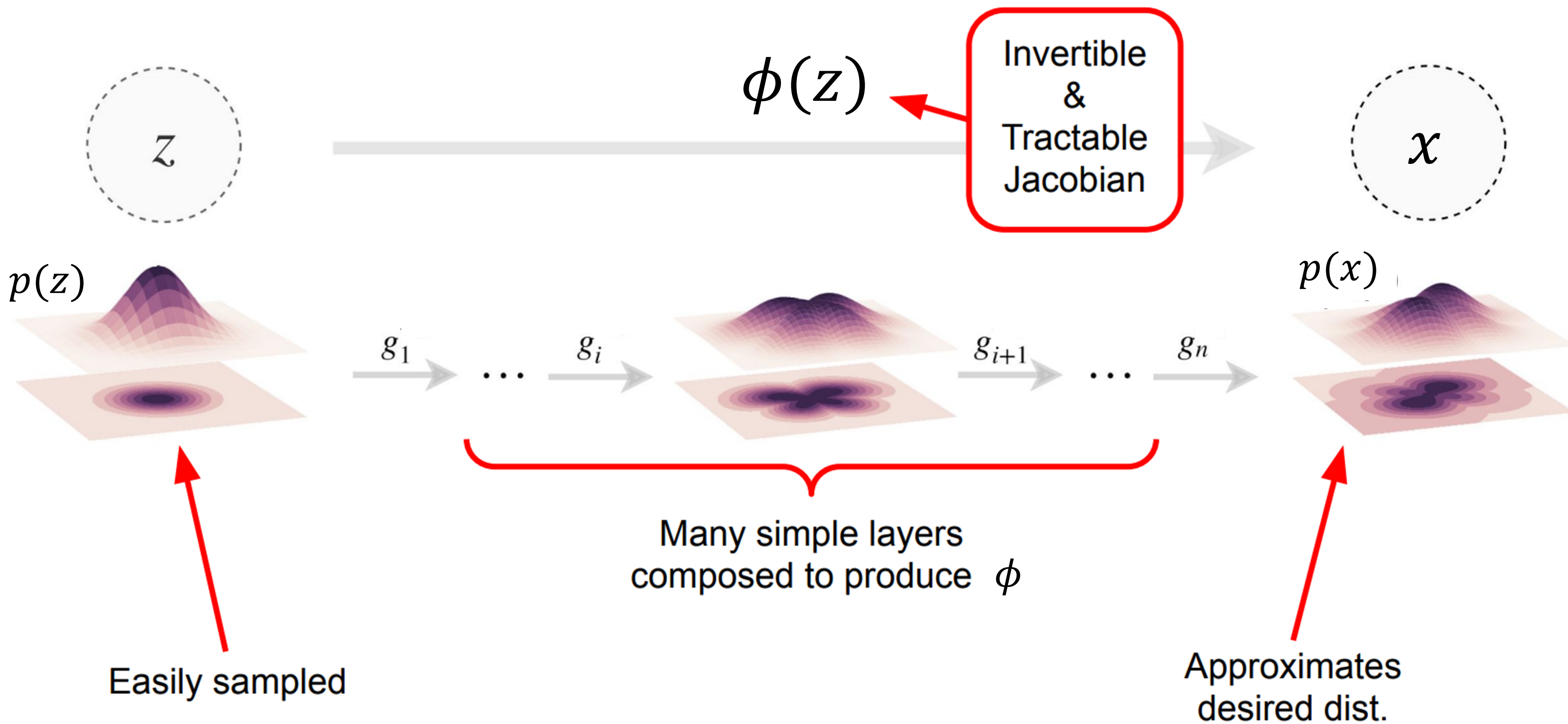
Slide credit: G. Kanwar

Normalizing Flows



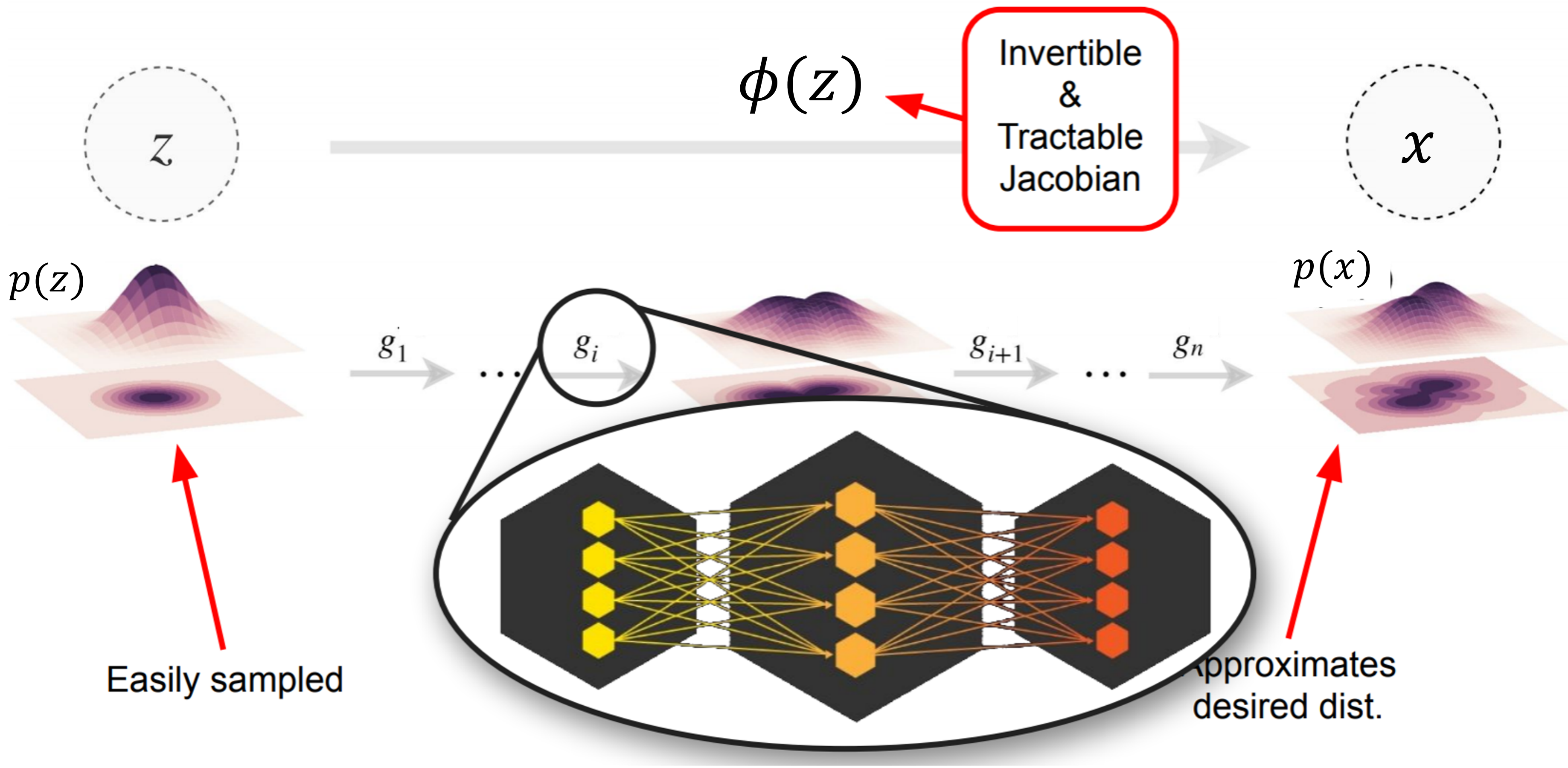
Slide credit: G. Kanwar

Normalizing Flows



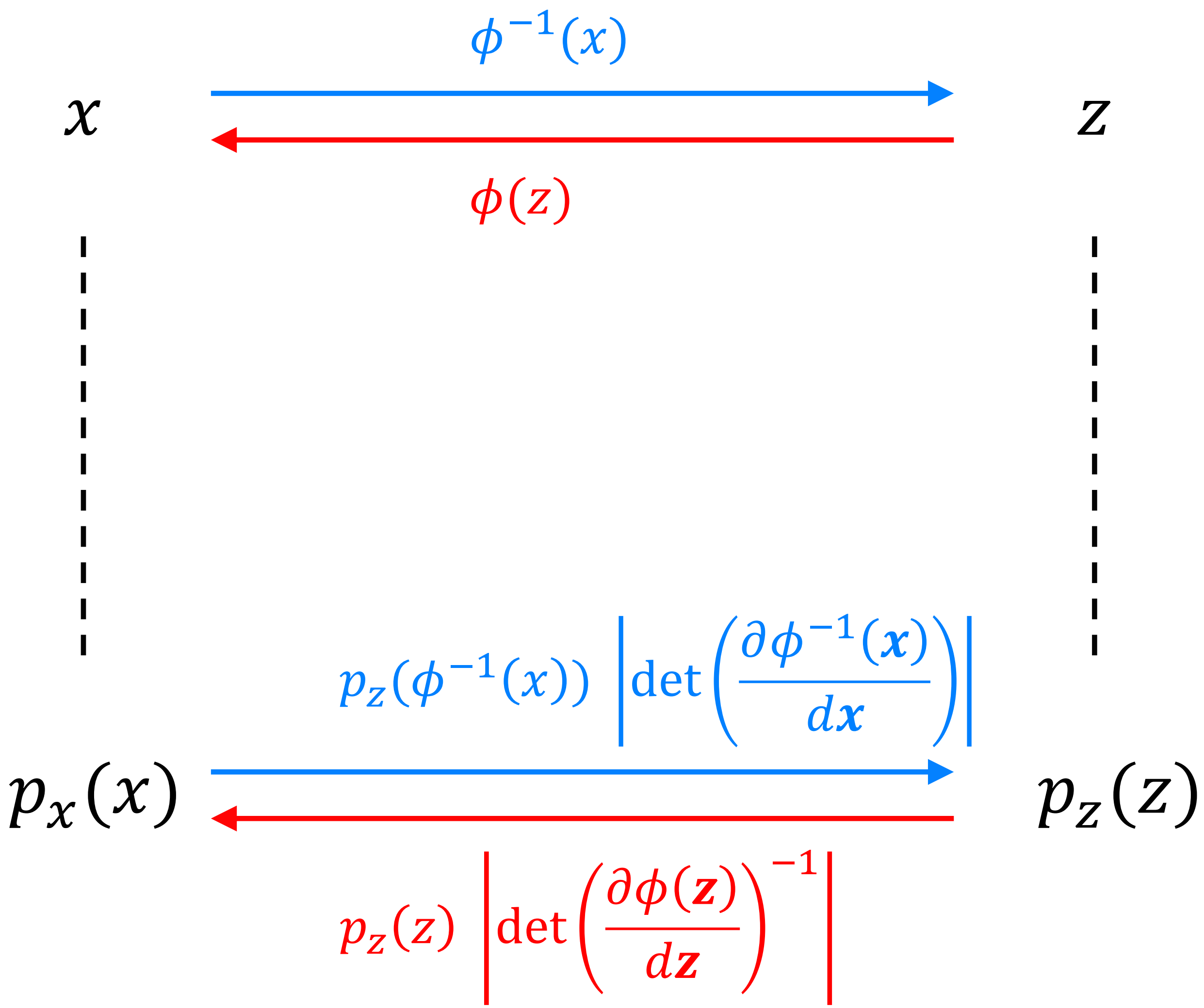
Slide credit: [G. Kanwar]

Normalizing Flows



Slide credit: G. Kanwar

Normalizing Flows



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: Real NVP

- Data vector $x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$

- Transformation

Functions f() and g() are neural networks

$\phi(z)$:

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

$\phi^{-1}(x)$:

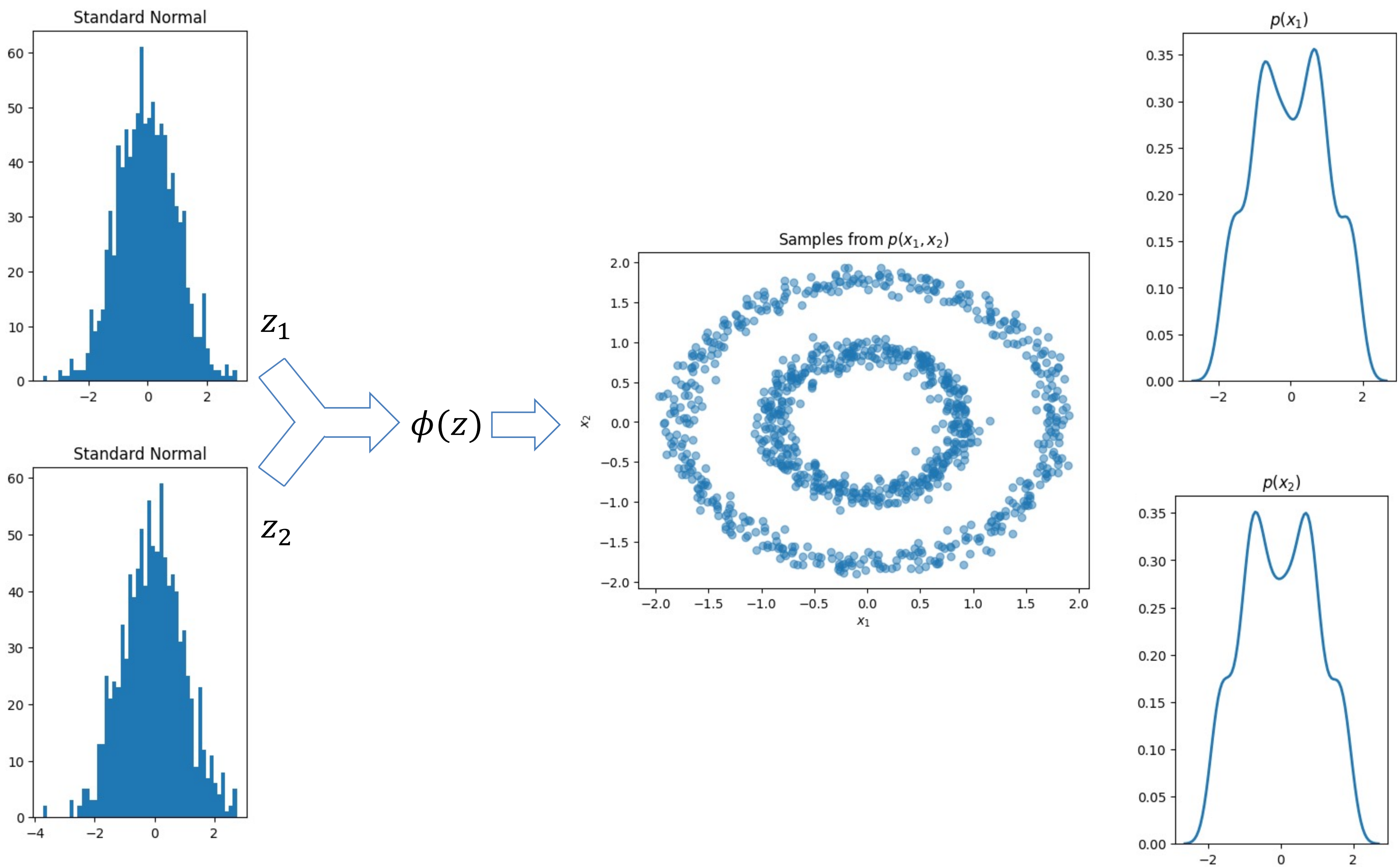
$$\begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} \phi_1^{-1}(x) \\ \phi_2^{-1}(x) \end{pmatrix} = \begin{pmatrix} x_1 \\ \left(x_2 - g(x_1)\right)/f(x_1) \end{pmatrix}$$

- Determinant:

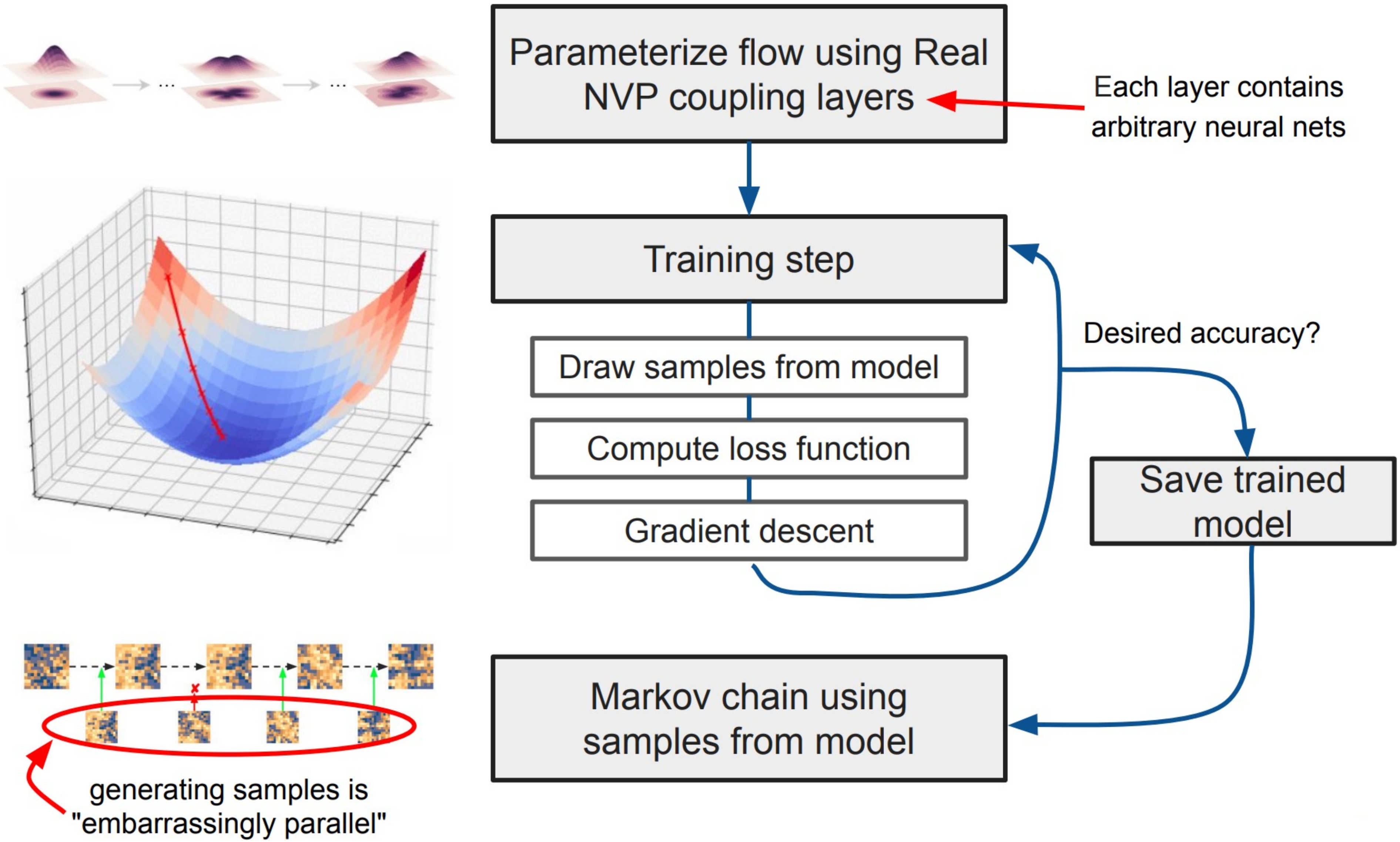
$$\det\left(\frac{\partial \phi(\boldsymbol{z})}{d\boldsymbol{z}}\right) = \det\begin{pmatrix} 1 & 0 \\ \left(\frac{\partial \phi_2(z)}{dz_1}\right) & f(z_1) \end{pmatrix} = f(z_2)$$

Jacobian is lower triangular

Example Normalizing flow



Applications: Sampling in Lattice QCD



Slide credit: G. Kanwar