Generative Models

Michal Drozdzal

mdrozdzal@fb.com

Generative models hold the potential to revolutionize ways we create digital content.

DALL·E 2

DALL·E 2 is an AI system that can create realistic images and art from a description in natural language.

Imagen

unprecedented photorealism × deep level of language understanding

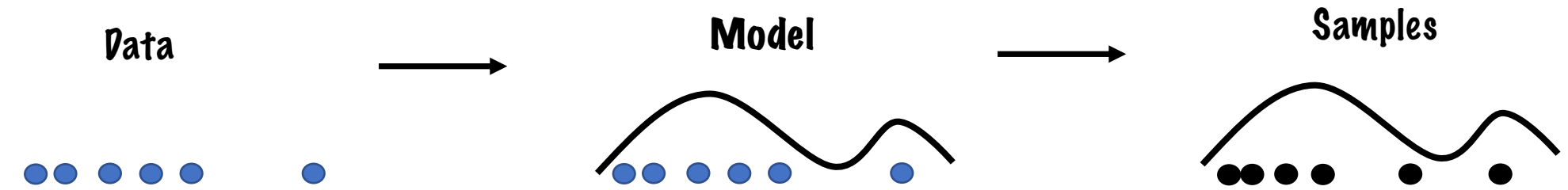
Imagen Video

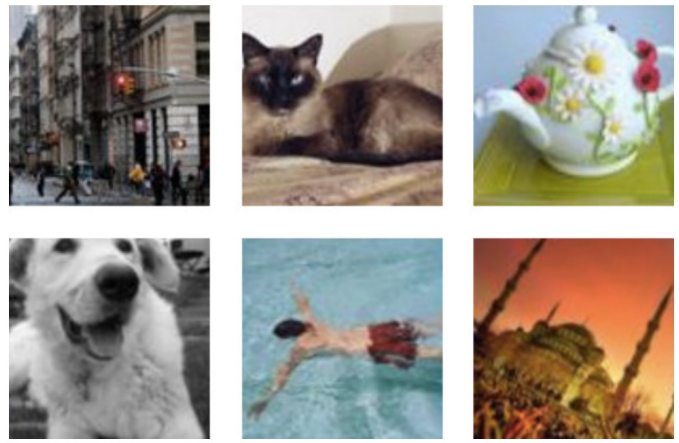
imagine · illustrate · inspire

Warning: the images on web pages are cherry picked.
Try available demos yourself to get a better sense of the capabilities of current generative models

Index

Image generation - density fitting





Conditional image generation

volcanos

monasteries

ants

Generator

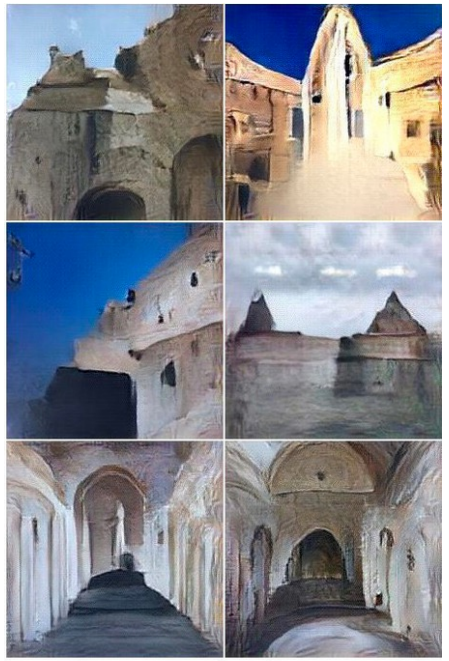


Image generation

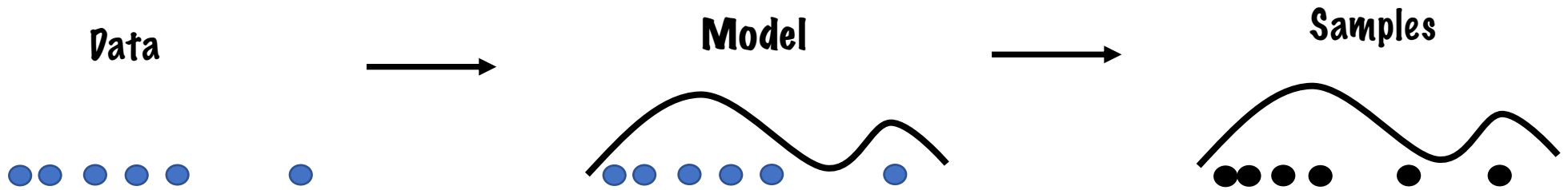
- Image generation can be both class conditional and unconditional.
- Image generation is generally seen as unsupervised/self-supervised.
- It can be seen as one-to-many (one class many images) or none-to-many (unconditional setup).
  - Comparing to other CV tasks:
    - Image classification - many-to-one mapping (many images map to a single class).
    - Image reconstruction - one-to-one mapping (one image maps to a single image).
- We will focus on the unconditional case.

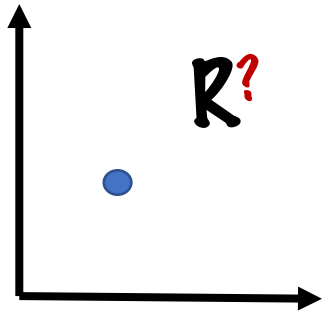
What are the applications for generative models?

Image generation has many applications:

Compression
Denoising
Inpainting
Texture synthesis
Semi-supervised learning
Unsupervised feature learning
Creativity
...

What makes image generation hard?
Data dimensionality



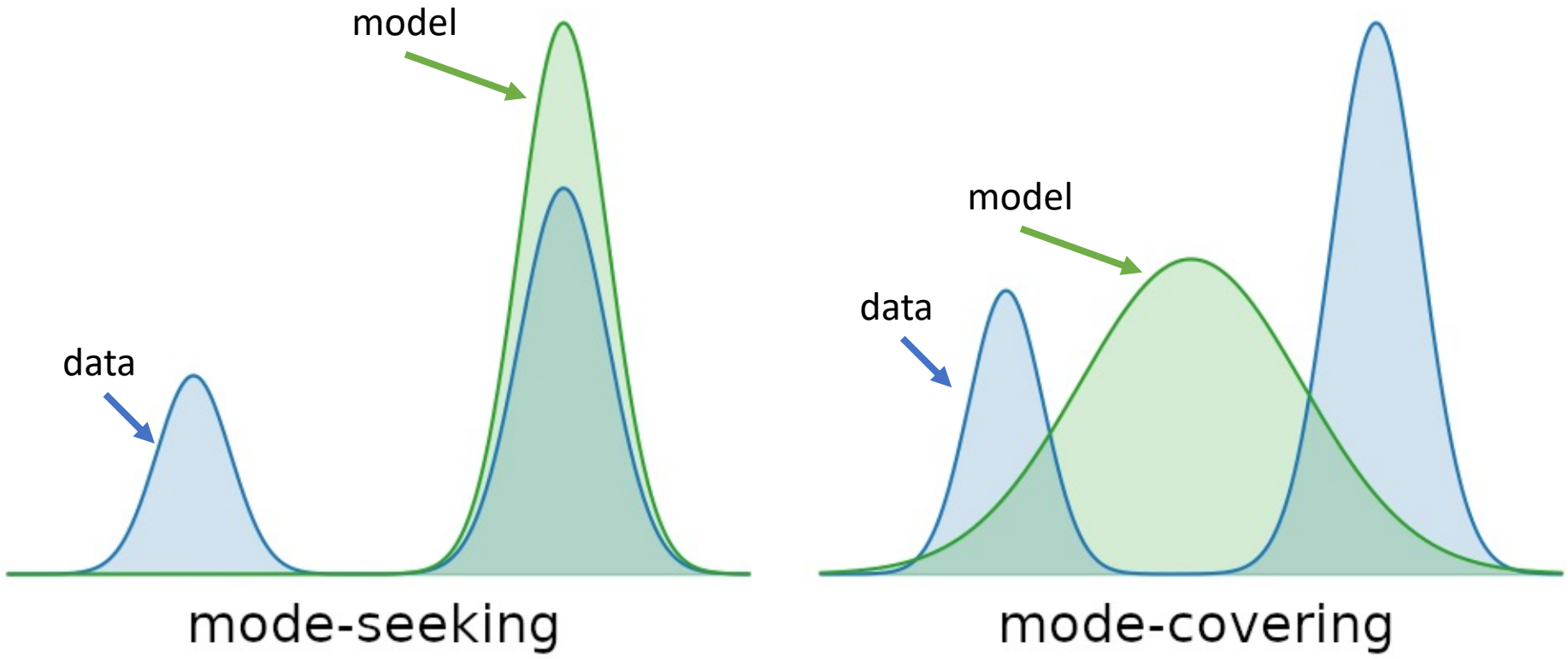


256 x 256 pixels RGB images

Each image is in R^{196608}

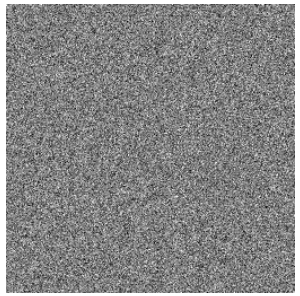
Density estimation in high dimensional spaces is hard....

A trade-off of image generation

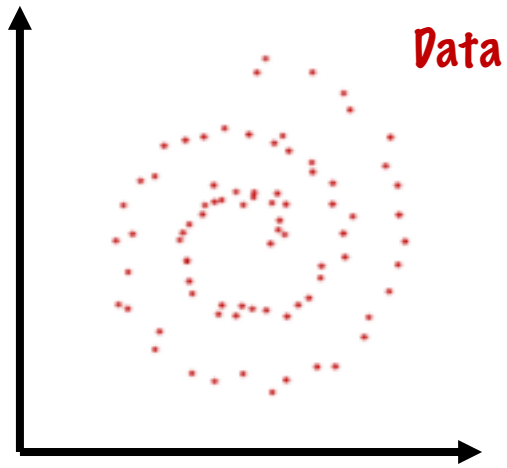


https://sander.ai/2020/03/24/audio-generation.html#generative-models

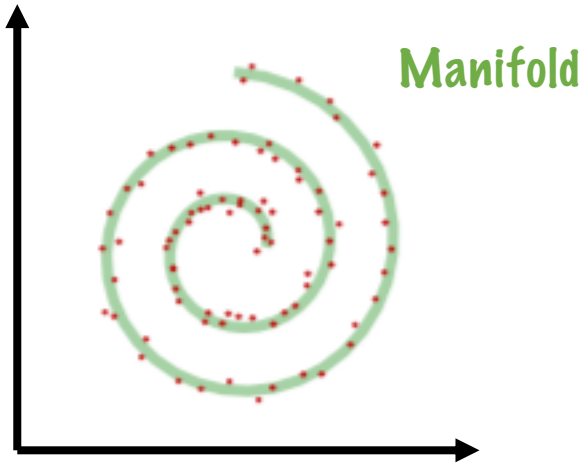
Manifold hypothesis



A sample from original space



Original space



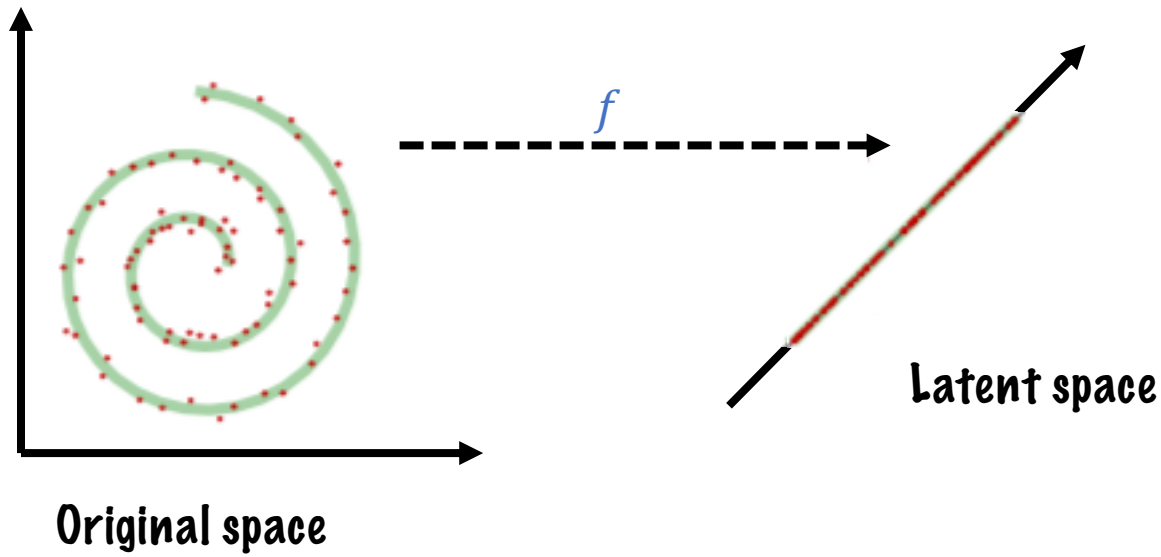
Original space

A sample from manifold

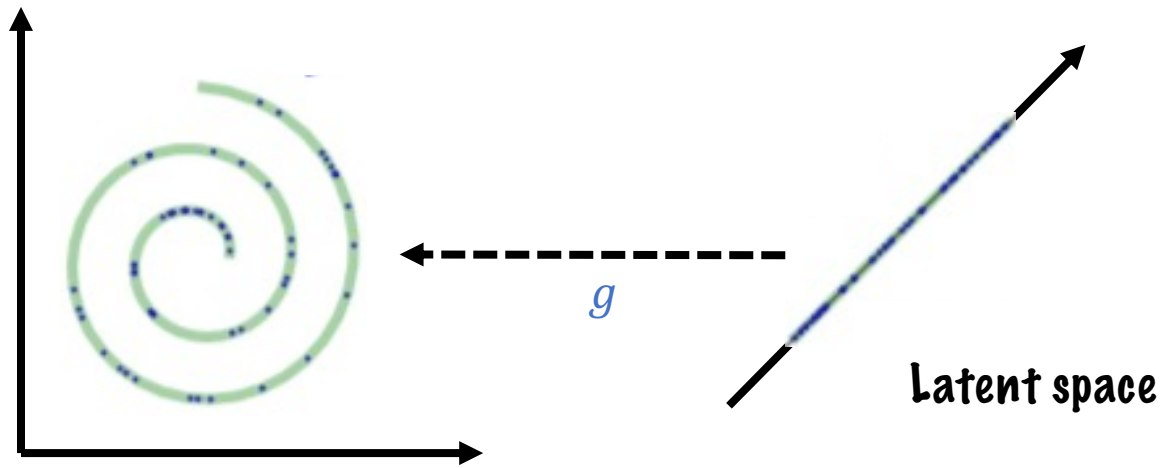
Latent space

f is a neural network

- encoder (network)
- inference (network)
- recognition (network)



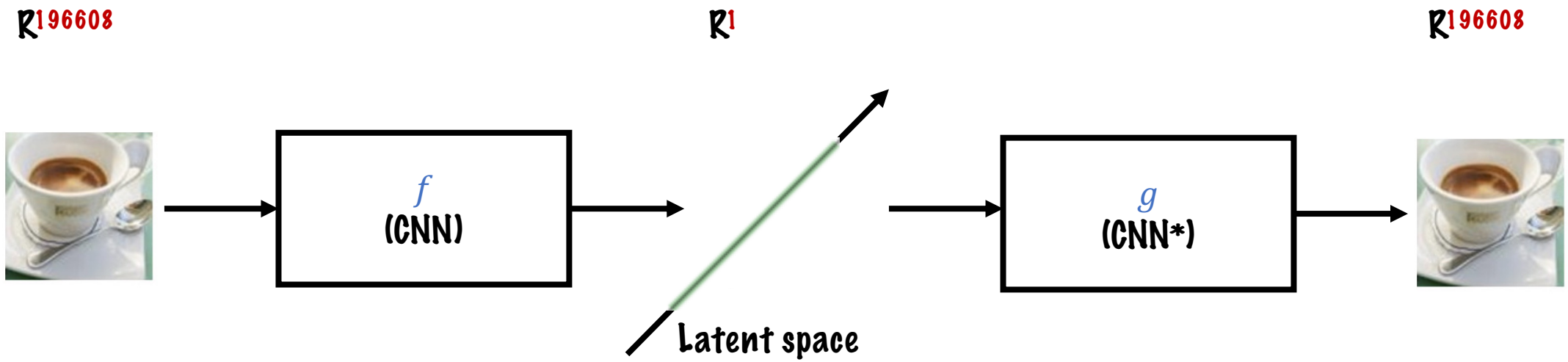
Latent space



g is a neural network

- decoder (network)
- generative (network)

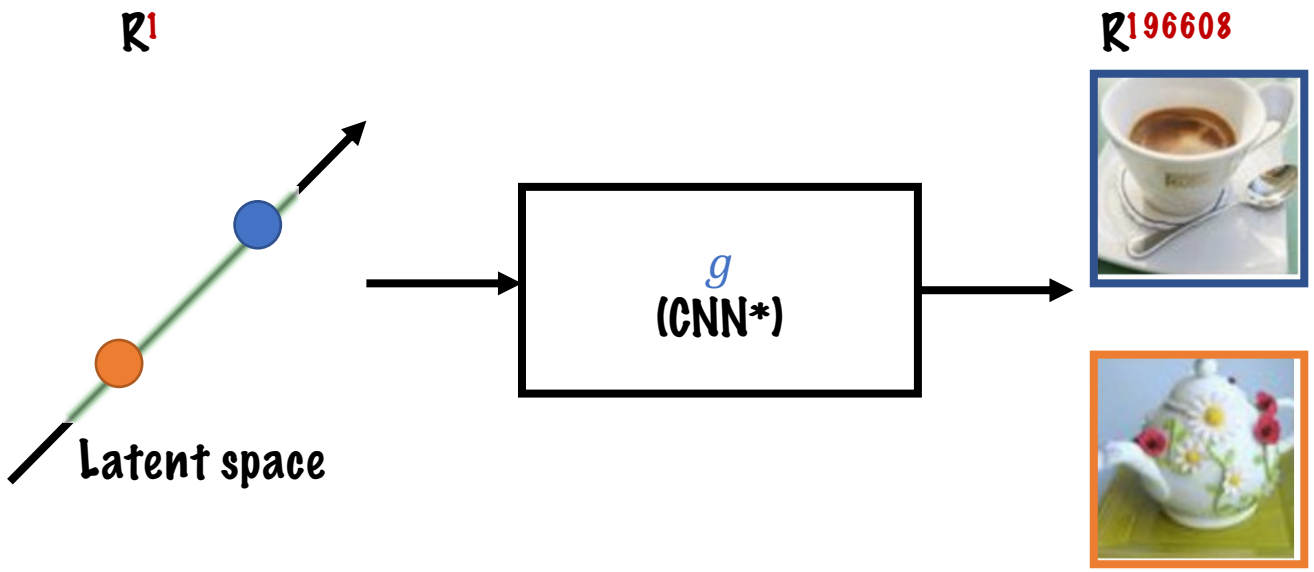
Encoder and Decoder networks (e.g. Autoencoder)



- Convolutional layers
- Poolings
- Non-linearities
- Fully connected layers
- Maybe some extras...

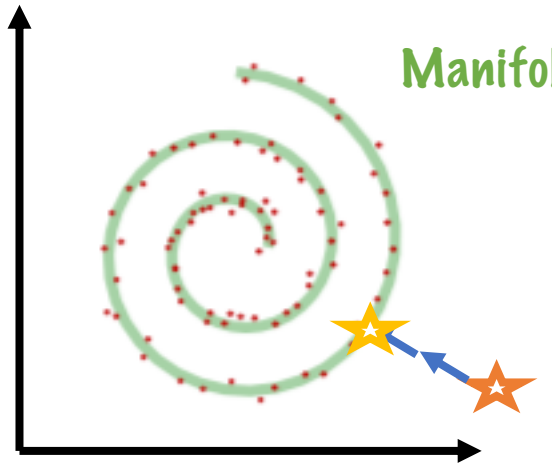
- Convolutional layers
- ~~Poolings~~
- Non-linearities
- Fully connected layers
- Maybe some extras...

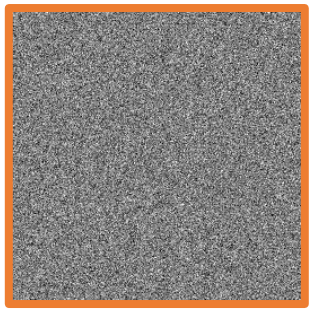
Generator (with latent space)



Variational autoencoders (VAEs)
Generative Adversarial Networks (GANs)

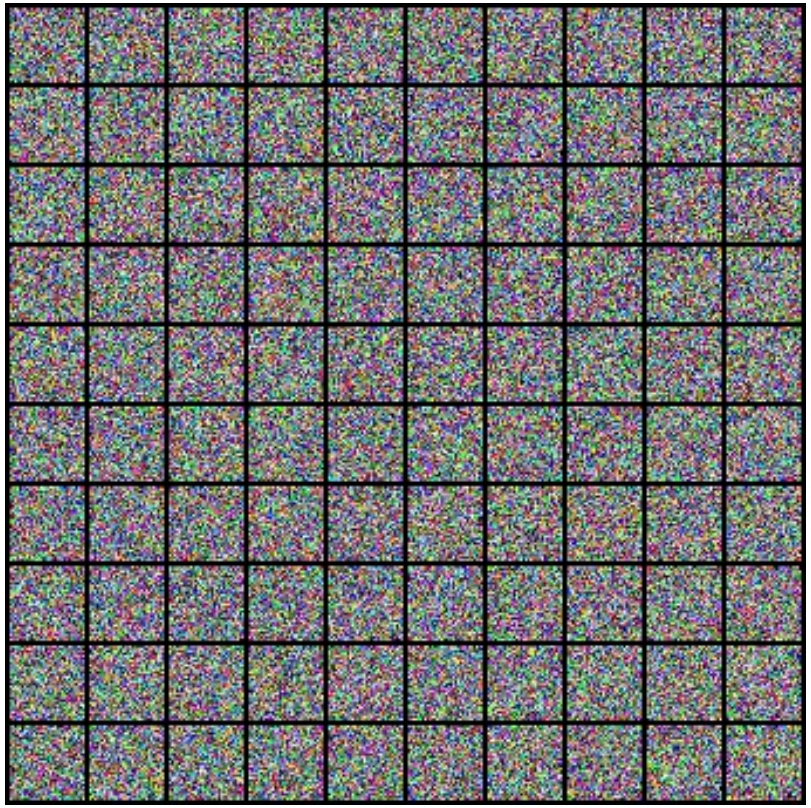
Denoising generator





Denoising Diffusion Probabilistic Models (DDPMs)

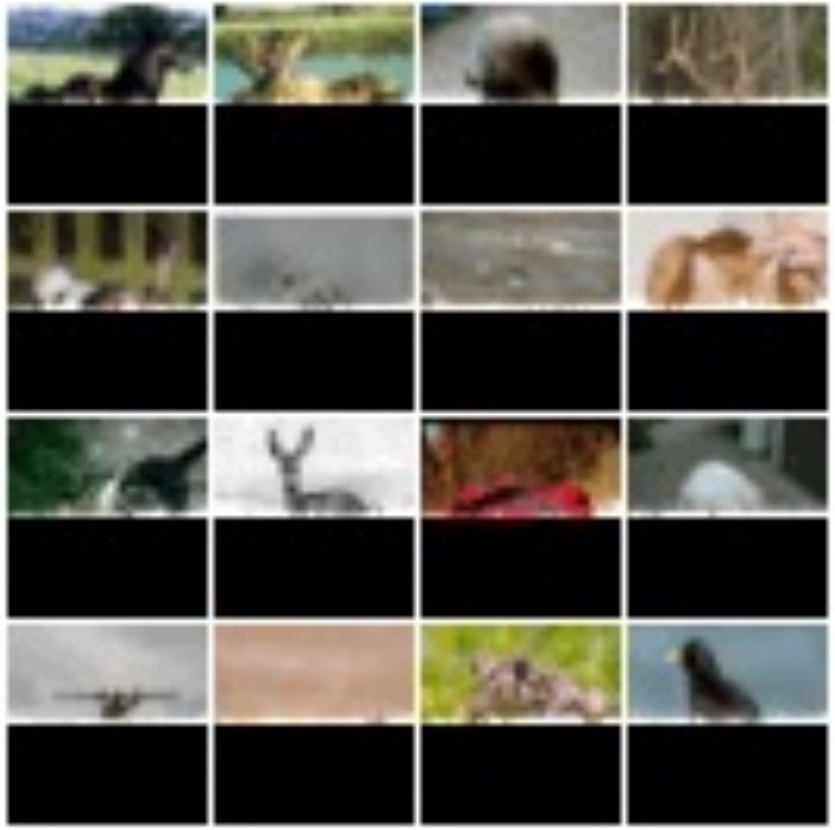
Denoising generator



Diffusion probabilistic models iteratively denoise the image.

This is a latent variable model where the latent variables have the same dimensions as images.

Generator (without latent space)



Autoregressive approaches use previously generated pixels to obtain new ones.

Why this would be a good idea?
Formulate image generation as multiple generations in R^256 (number of values in each channel in RGB images).

This model model is covered under bonus material, and we will likely not discuss it during this lecture.

Image generation vs retrieval

Generation:

- Incorporates learning
- Use a large dataset to learn a density model
- Can generate novel images (not in the dataset)

Retrieval:

- Might incorporate learning (e.g., embedding)
- Use a large dataset to retrieve images
- Cannot produce images that are not in the dataset

Intro: summary

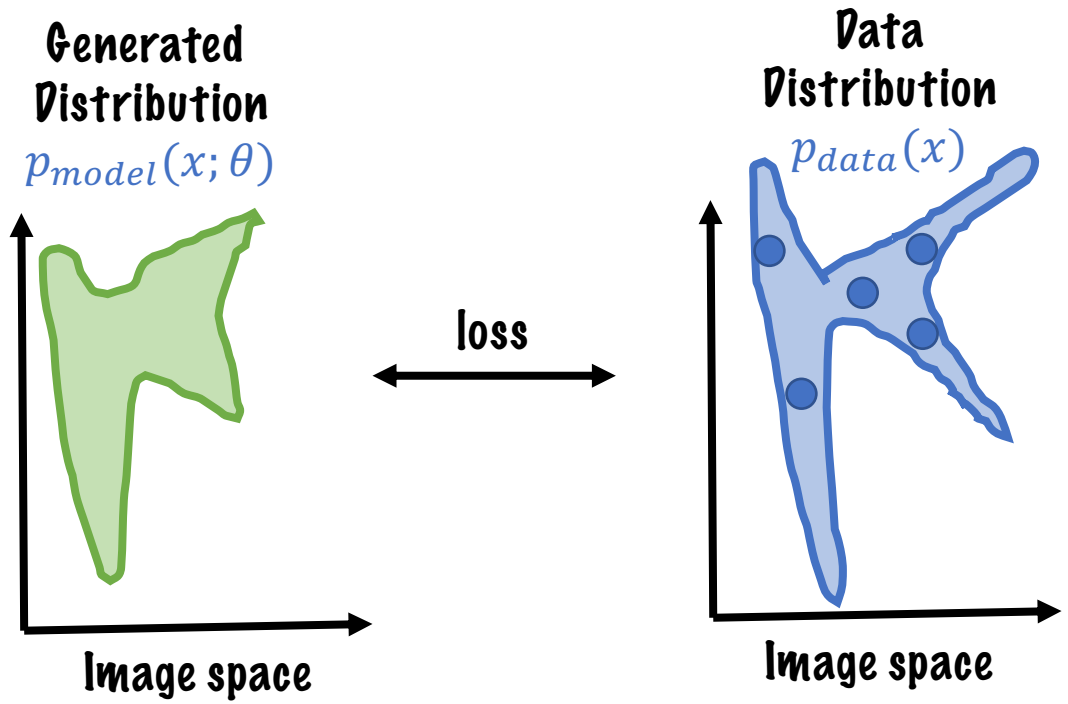
- Image generation is about models that produce images
- Image generation approach the problem by learning data distribution
- These models can create novel images
- The main challenge when building such systems is image dimensionality
- Different generative approaches include latent variable, denoising and autoregressive models

Index

Notation

$x^{(i)}$	Data point (e. g. single image)
$X = [x^{(1)}, x^{(2)}, \cdots, x^{(N)}]^T$	Dataset
$p(x)$	Probability distribution
$p(x\|y)$	Conditional probability distribution
$p(x, y)$	Joint probability distribution
$p(x; \theta)$ or $p_\theta(x)$	Probability distribution parametrized by θ
$\mathbb{E}[Y]$	Expected value

Generative models with maximum likelihood Intuition



Adjust model parameters (θ) to match the model distribution $p_{model}(x;\theta)$ with the data distribution $p_{data}(x)$.

Image adapted from OpenAI blog on generative models: https://blog.openai.com/generative-models/

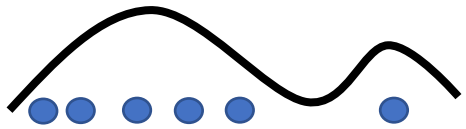
Maximum Likelihood

Data

$$X = [x^{(1)}, x^{(2)}, x^{(3)}, x^{(4)}, x^{(5)}, x^{(6)}]^T$$

$$x^{(i)} \sim p_{data}$$

Model



$$p_{model}(X;\theta)$$

Goal of learning (maximum likelihood):

$$\theta^* = \underset{\theta}{\operatorname{argmax}}\, p_{model}(X;\theta)$$

We can rewrite it as:

$$\theta^* = \underset{\theta}{\operatorname{argmax}}\, \mathbb{E}_X \log p_{model}(x^{(i)};\theta)$$

Derivation:

$$p_{model}(X;\theta) = \prod_{i=1}^{N} p_{model}\left(x^{(i)};\theta\right)$$

Log likelihood:

$$\log p_{model}(X;\theta) = \sum_{i=1}^{N} \log p_{model}\left(x^{(i)};\theta\right)$$

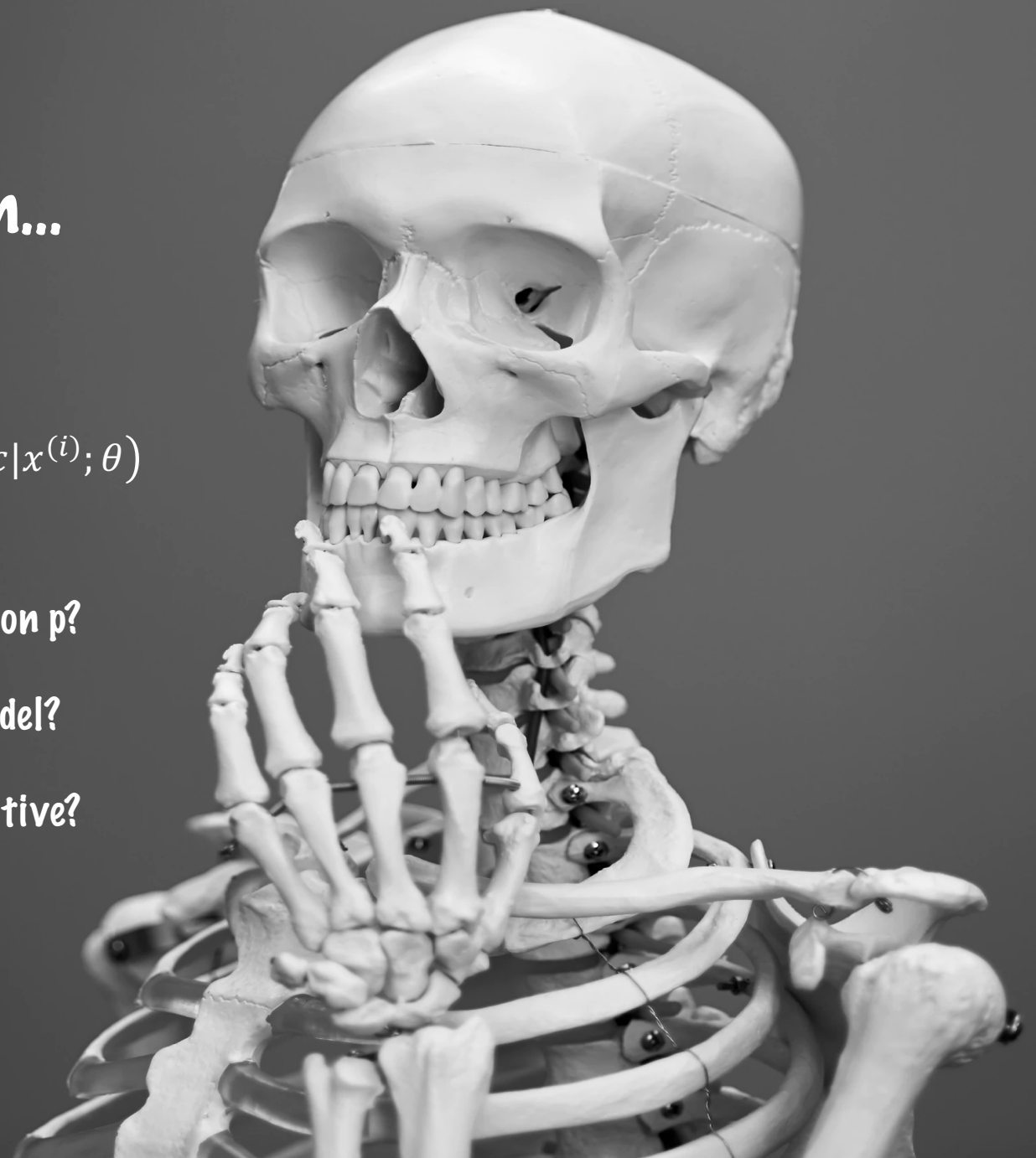
Small review
Image classification...
P(c | x)

$$\log p_{model}(C|X;\theta) = \sum_{i=1}^{N} \log p_{model}\left(c|x^{(i)};\theta\right)$$

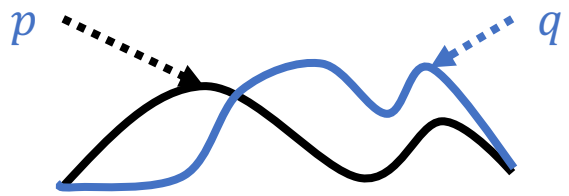
What is the name of the distribution p?

What is the structure of model?

What is the training principle/objective?



KL divergence



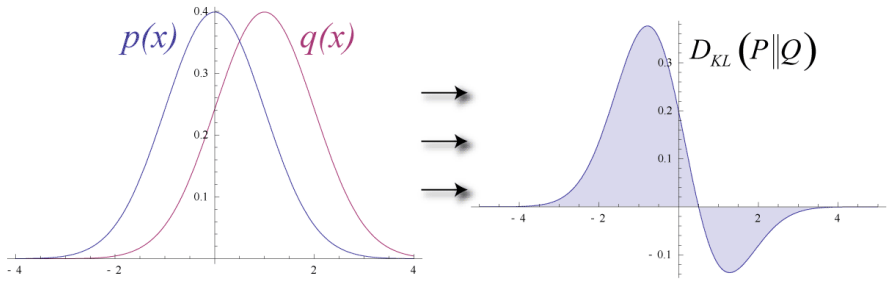
Kullback-Leibler (KL) divergence:

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

Kullback-Leibler (KL) divergence is asymmetric:

$$D_{KL}(p||q) \neq D_{KL}(q||p)$$

If $D_{KL}(p||q) = 0$ **then** $p = q$



Original Gaussian PDF's KL Area to be Integrated

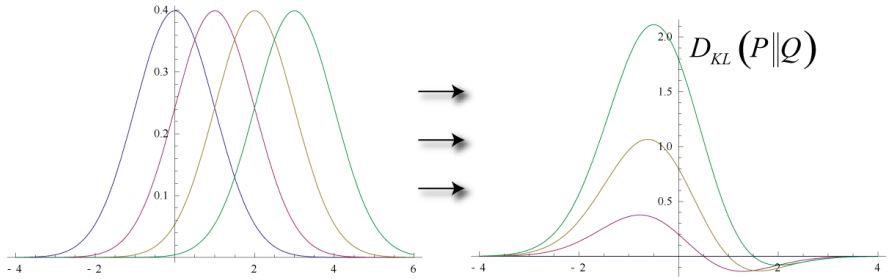


Image from Wikipedia

Maximizing likelihood is equivalent to minimization of the KL divergence between data generating distribution $p_{data}(x)$ **and the model** $p_{model}(x)$**.**

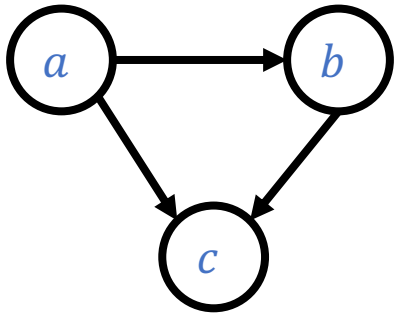
Index

Graphical model

Use a graph to express structure between random variables.

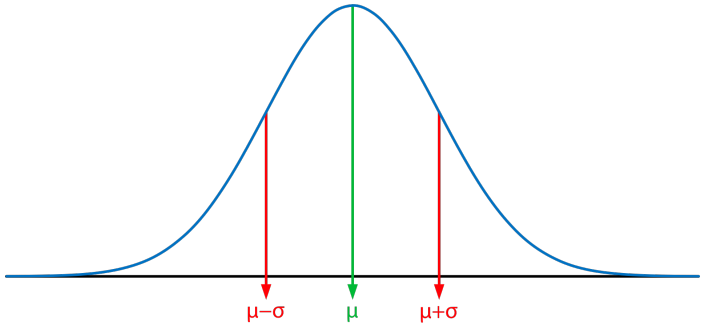
Example: Given 3 random variables a, b, c draw a graphical model of $p(a, b, c)$

Using product rule of probability, we can write $p(a, b, c) = p(c|a, b)p(a, b) = p(c|a, b)p(b|a)p(a)$

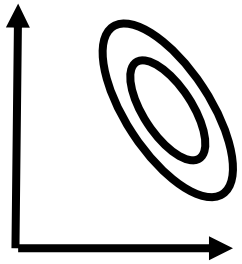


Gaussian distribution

$\mathcal{N}(\mu, \sigma)$



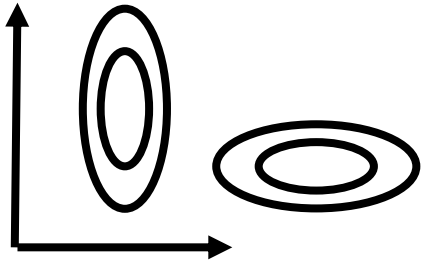
$\mathcal{N}(\boldsymbol{\mu}, \Sigma)$



$\boldsymbol{\mu} = [\mu_1, \mu_2]^T$

$\Sigma = \begin{bmatrix} \sigma_{11} & \sigma_{12} \\ \sigma_{21} & \sigma_{22} \end{bmatrix}$

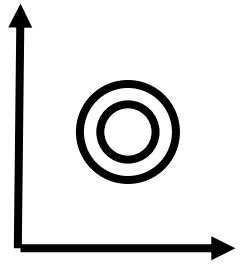
$\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma})$



$\boldsymbol{\mu} = [\mu_1, \mu_2]^T$

$\Sigma = \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix}$

$\mathcal{N}(\boldsymbol{\mu}, \sigma\mathbf{1})$

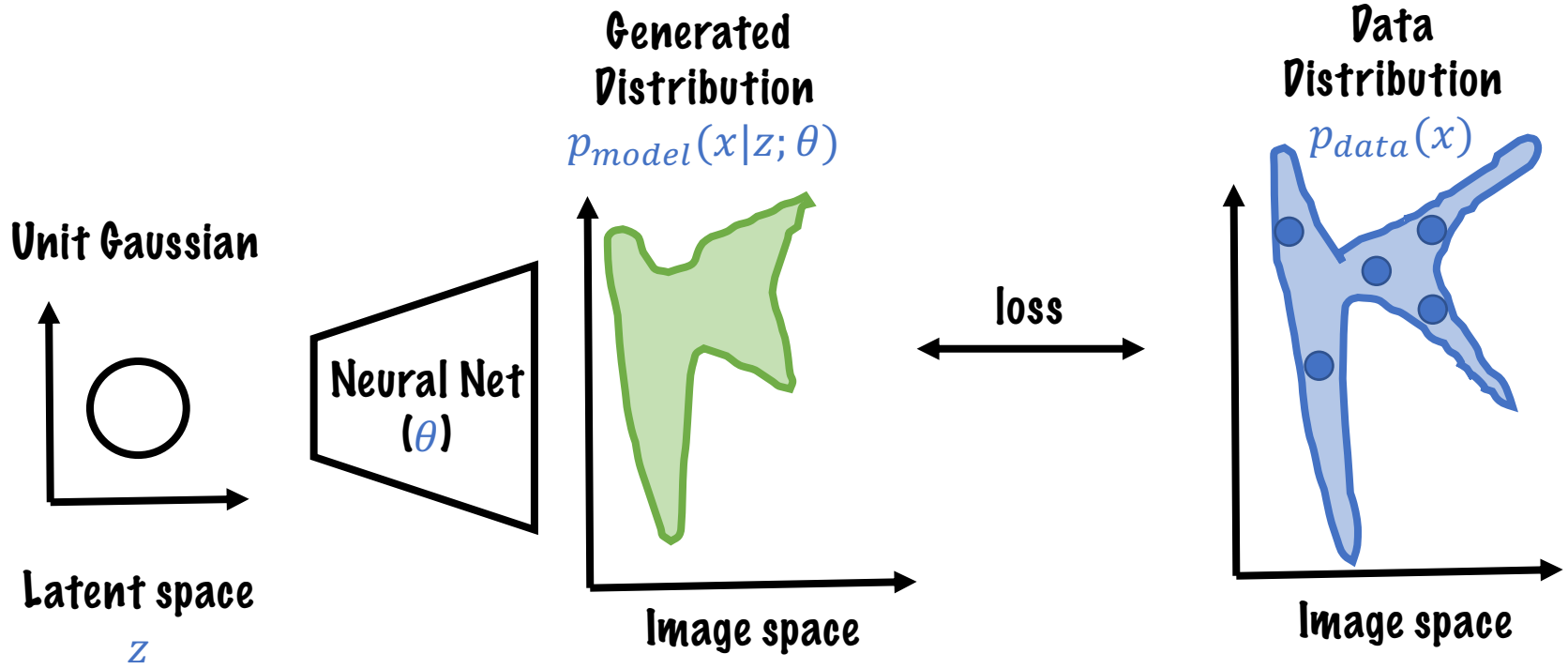


$\boldsymbol{\mu} = [\mu_1, \mu_2]^T$

$\Sigma = \begin{bmatrix} \sigma & 0 \\ 0 & \sigma \end{bmatrix}$

$\mathbf{1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

Generative models with latent variables Intuition

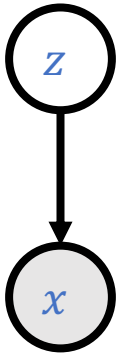


Adjust model parameters (θ) to match the model distribution $p_{model}(x|z;\theta)$ with the data distribution $p_{data}(x)$.

Image adapted from OpenAI blog on generative models: https://blog.openai.com/generative-models/

Variational inference

Latent Variable models



Hard to compute...

$$p_{model}(x^{(i)}) = \int p_{model}(x^{(i)}, z)dz$$

Variational inference: use an approximation to transform the integral into an expectation over a simple, known distribution.

Evidence Lower Bound (ELBO)

$$\log p(x^{(i)}) = \log \int p(x^{(i)}, z) dz = \log \int p(x^{(i)}, z) \frac{q(z)}{q(z)} dz =$$

$$= \log \mathbb{E}_q \frac{p(x^{(i)}, z)}{q(z)} \geq \mathbb{E}_q \left[\log \frac{p(x^{(i)}, z)}{q(z)} \right] = \mathcal{L}(x^{(i)})$$

Evidence Lower Bound (ELBO)

$\log p(x^{(i)})$ - **marginal log likelihood (evidence)**

$q(z)$ - **variational distribution**

$\log \mathbb{E}[Y] \geq \mathbb{E}[\log Y]$ - **Jensen's inequality**

$p(z|x^{(i)})$ - **posterior**

We can also write:

$$\log p(x^{(i)}) \geq \log p(x^{(i)}) - D_{KL}(q(z)||p(z|x^{(i)})) = \mathcal{L}(x^{(i)})$$

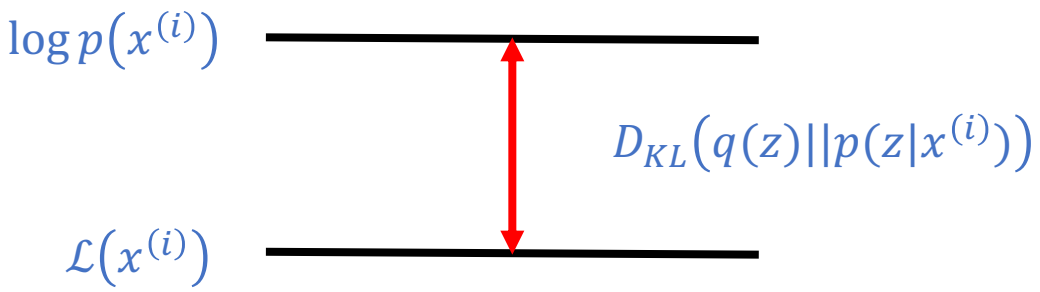
For an interested reader:

Note that: $p(z|x) = \frac{p(x,z)}{p(x)}$

$$D_{KL}(q(z)||p(z|x)) = \mathbb{E}_q \left[\log \frac{q(z)}{p(z|x)} \right] = \mathbb{E}_q \left[\log \frac{q(z)p(x)}{p(x,z)} \right] = \mathbb{E}_q \left[\log \frac{q(z)}{p(x,z)} \right] + \log p(x) = -\mathbb{E}_q \left[\log \frac{p(x,z)}{q(z)} \right] + \log p(x)$$

ELBO

$$\mathcal{L}\left(x^{(i)}\right) = \log p\left(x^{(i)}\right) - D_{KL}\left(q(z)||p(z|x^{(i)})\right)$$



The tightness of ELBO depends on how well the variational distribution approximates the true posterior.

How do we get q? (Amortized inference)

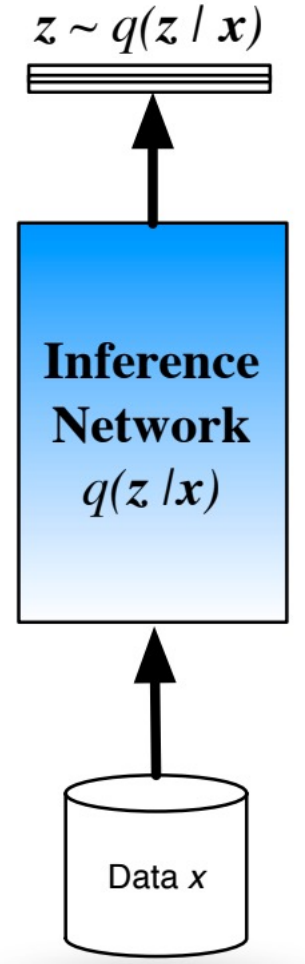
$$\mathbb{E}_q\left[\log \frac{p(x^{(i)}, z)}{q(z)}\right]$$

Vs.

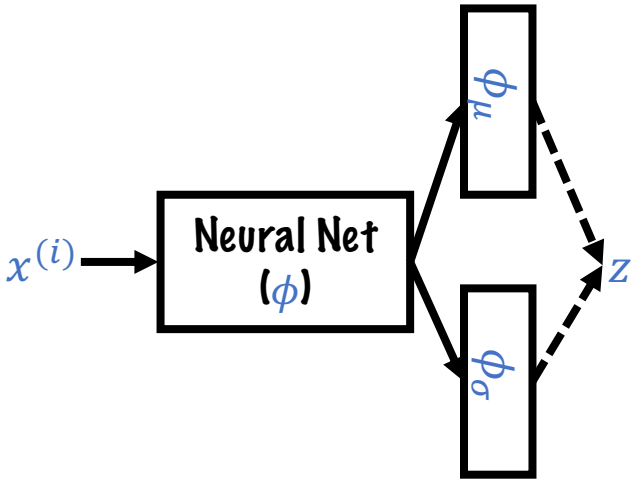
$$\mathbb{E}_{q(z|x)}\left[\log \frac{p(x, z)}{q(z|x)}\right]$$

Estimation of q is costly (e. g. EM)

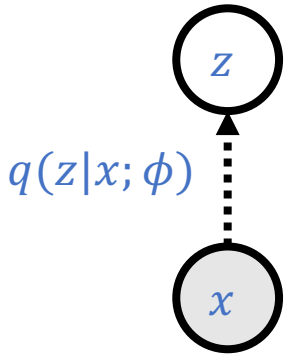
Amortize -> spread the inference cost over the whole dataset



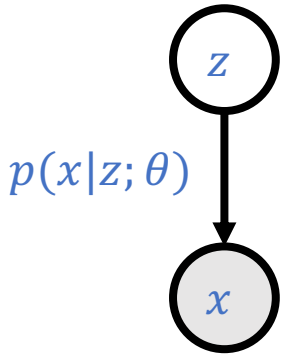
Variational Autoencoders (VAE)

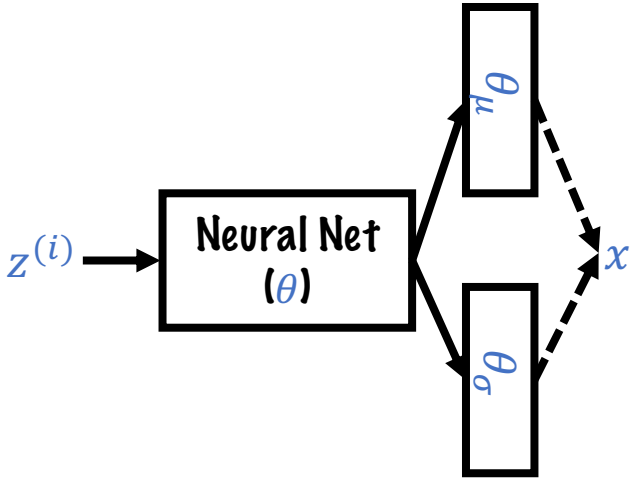


Inference model



Generative model





$$q(z|x^{(i)};\phi) = \mathcal{N}\left(z; \boldsymbol{f}_{\boldsymbol{\phi}}^{\boldsymbol{\mu}}(x^{(i)}), \boldsymbol{f}_{\boldsymbol{\phi}}^{\boldsymbol{\sigma}}(x^{(i)})\right)$$

$x^{(i)}$- our data points: $x^{(i)} \sim p_{data}$

$$p\left(x|z^{(i)};\theta\right) = \mathcal{N}\left(x; \boldsymbol{g}_{\boldsymbol{\theta}}^{\boldsymbol{\mu}}(z^{(i)}), \boldsymbol{g}_{\boldsymbol{\theta}}^{\boldsymbol{\sigma}}(z^{(i)})\right)$$

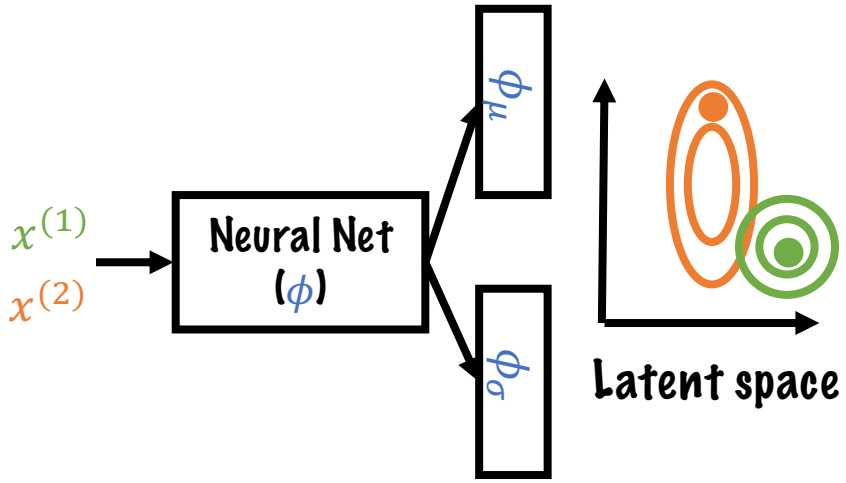
$z^{(i)}$- comes from model prior $z^{(i)} \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$ (generation)

and

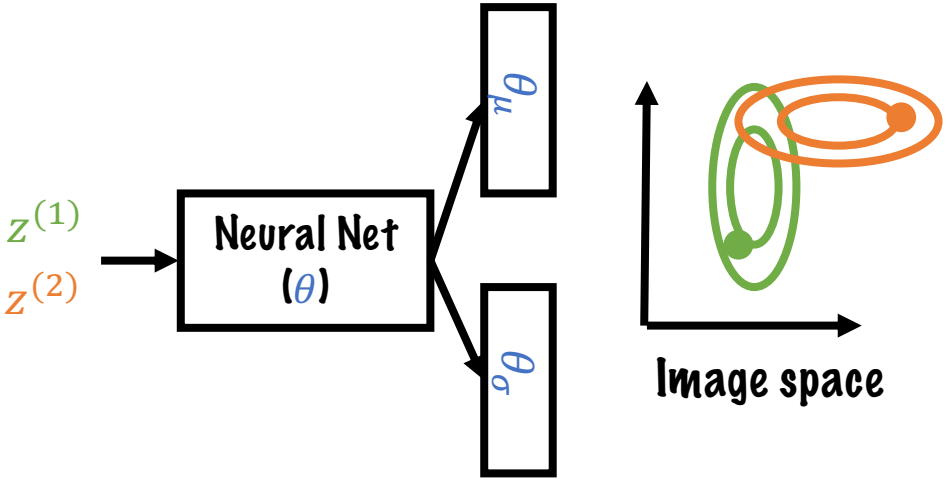
$z^{(i)}$- comes from model posterior $z^{(i)} \sim q(z|x^{(i)};\phi)$ (training)

Variational Autoencoders (VAE)

Inference model



Generative model



$$q(z|x^{(i)};\phi) = \mathcal{N}\left(z; \boldsymbol{f}^{\boldsymbol{\mu}}_{\boldsymbol{\phi}}(x^{(i)}), \boldsymbol{f}^{\boldsymbol{\sigma}}_{\boldsymbol{\phi}}(x^{(i)})\right)$$

$x^{(i)}$- our data points: $x^{(i)} \sim p_{data}$

$$p\left(x|z^{(i)};\theta\right) = \mathcal{N}\left(x; \boldsymbol{g}^{\boldsymbol{\mu}}_{\boldsymbol{\theta}}(z^{(i)}), \boldsymbol{g}^{\boldsymbol{\sigma}}_{\boldsymbol{\theta}}(z^{(i)})\right)$$

$z^{(i)}$- comes from model prior $z^{(i)} \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$ (generation)

and

$z^{(i)}$- comes from model posterior $z^{(i)} \sim q(z|x^{(i)};\phi)$ (training)

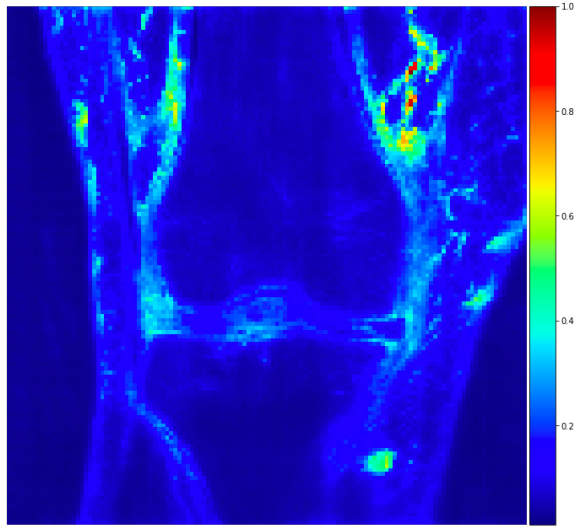
A visual example

θ_μ

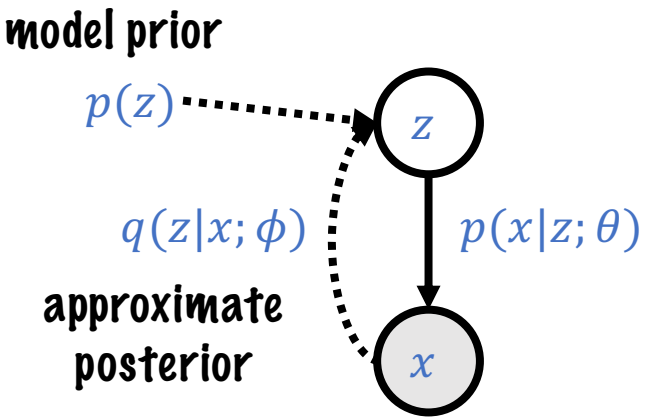
Mean

θ_σ

Std.



Variational Autoencoders (VAE)



ELBO for variational autoencoder:

$$\mathcal{L}(x,\theta,\phi) = \mathbb{E}_{q(z|x;\,\phi)}\left[\log \frac{p(x,z;\,\theta)}{q(z|x;\phi\,)}\right] =$$

$$\mathbb{E}_{q(z|x;\phi)}[\log p(x|z;\theta)] - D_{KL}(q(z|x;\phi)||p(z))$$

Reconstruction — Regularization

Reconstruction -> high quality of x from z

Regularization -> keeps the approximate posterior $q(z|x;\phi)$ close to the model prior $p(z)$

How to compute the expectation $\mathbb{E}_{q(z|x;\phi)}$?

Draw samples from the approximate posterior $z^{(i)} \sim q(z|x;\phi)$ to approximate the expectation:

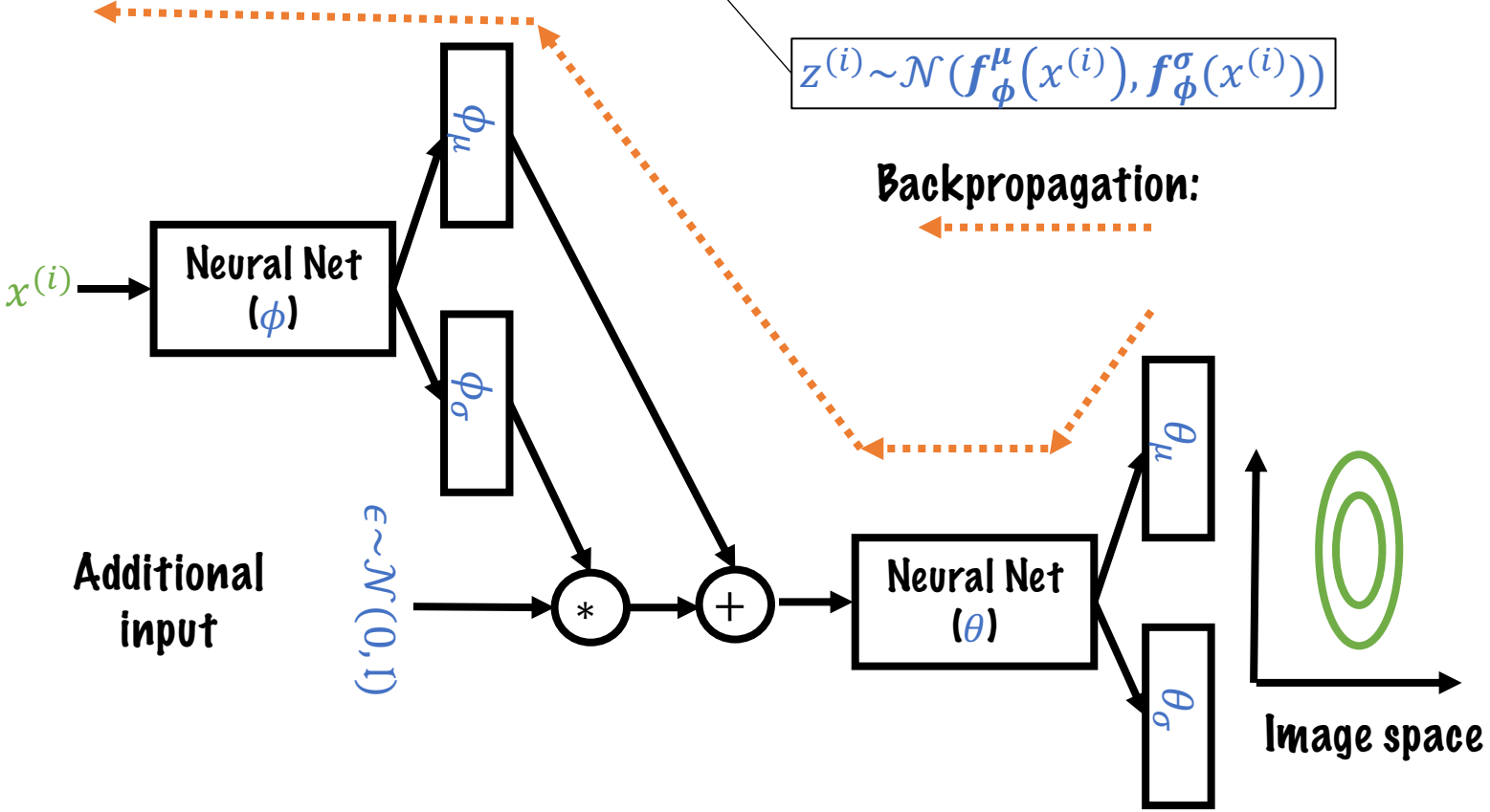
$$\mathbb{E}_{q(z|x;\phi)}[\log p(x|z;\theta)] \approx \frac{1}{S}\sum_{i=1}^{S} \log p(x|z^{(i)};\theta)$$

Variational Autoencoders (VAE)
Reparameterization trick

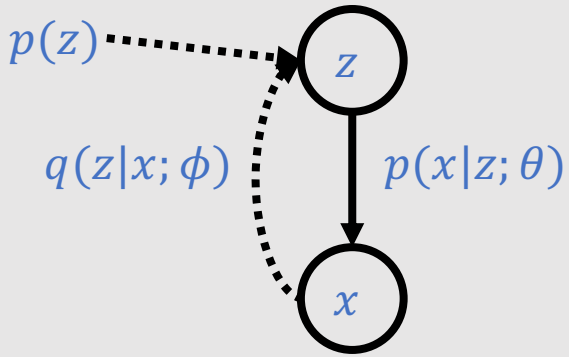
Reparameterization trick:

$$z^{(i)} = f_\phi^\mu(x^{(i)}) + f_\phi^\sigma(x^{(i)}) * \epsilon^{(i)}; \epsilon^{(i)} \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$$

$$z^{(i)} \sim \mathcal{N}(\boldsymbol{f_\phi^\mu}(x^{(i)}), \boldsymbol{f_\phi^\sigma}(x^{(i)}))$$



Reminder:



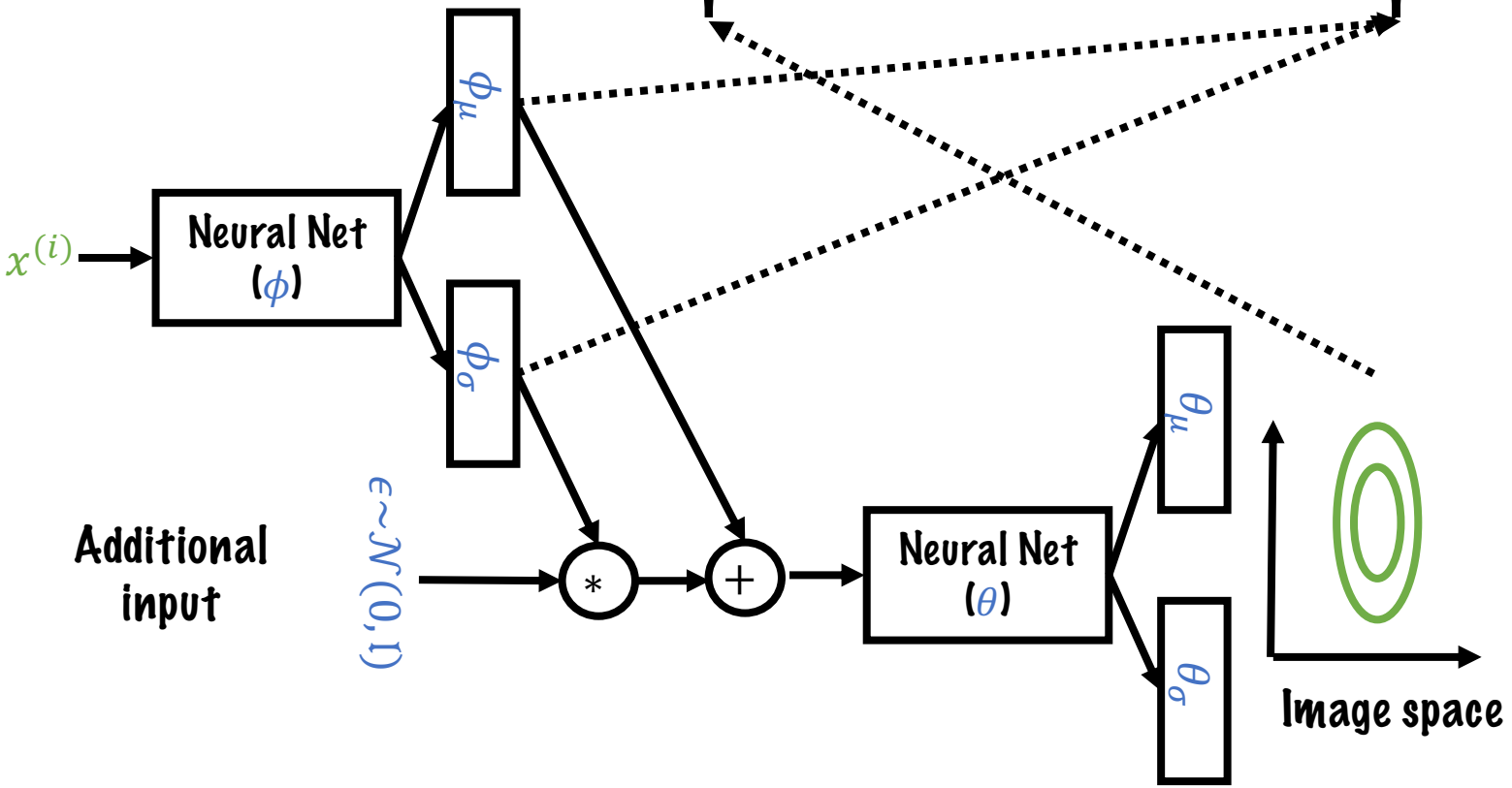
$$q(z|x^{(i)};\phi) = \mathcal{N}\left(z; \boldsymbol{f_\phi^\mu}(x^{(i)}), \boldsymbol{f_\phi^\sigma}(x^{(i)})\right)$$

$$p(x|z^{(i)};\theta) = \mathcal{N}(x; \boldsymbol{g_\theta^\mu}(z^{(i)}), \boldsymbol{g_\theta^\sigma}(z^{(i)}))$$

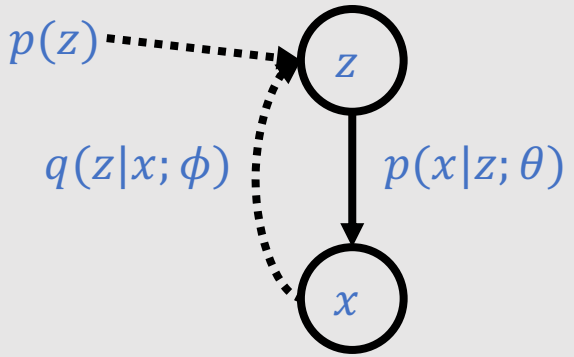
Variational Autoencoders (VAE)

Training objective:

$$\mathcal{L}(x^{(i)}, \theta, \phi) = \frac{1}{S}\sum_{j=1}^{S} \log p(x|f_\phi^\mu(x^{(i)}) + f_\phi^\sigma(x^{(i)}) * \epsilon^{(j)}; \theta) - D_{KL}(q(z|x^{(i)}; \phi)||p(z))$$



Reminder:



$$q(z|x^{(i)};\phi) = \mathcal{N}\left(z; \boldsymbol{f_\phi^\mu}(x^{(i)}), \boldsymbol{f_\phi^\sigma}(x^{(i)})\right)$$

$$p(x|z^{(i)};\theta) = \mathcal{N}(x; \boldsymbol{g_\theta^\mu}(z^{(i)}), \boldsymbol{g_\theta^\sigma}(z^{(i)}))$$

KL divergence can be computed analytically given that posterior and prior are normal distributions.

Gaussian log likelihood vs MSE

$$\log p\left(x^{(i)}|z\right) = -\frac{L}{2}\log 2\pi - \frac{1}{2}\sum_{l=1}^{L}\left(\log \boldsymbol{\sigma}_l - \frac{\left(x^{(i)}{}_l - \boldsymbol{\mu}_l\right)^2}{\boldsymbol{\sigma}_l}\right)$$

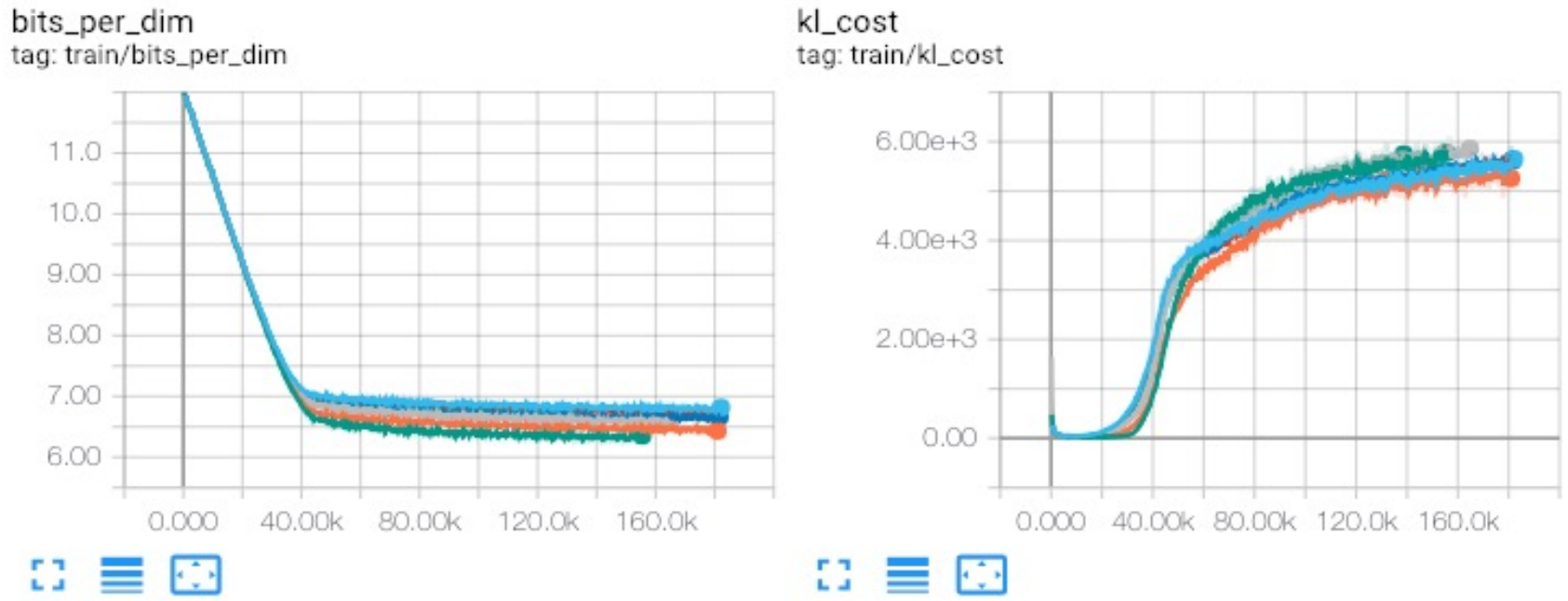
$$\text{MSE}(x^{(i)}) = \frac{1}{L}\sum_{l=1}^{L}\left(x^{(i)}{}_l - \boldsymbol{\mu}_l\right)^2$$

When minimizing MSE is equivalent to maximizing Gaussian LL?

Training of a VAE

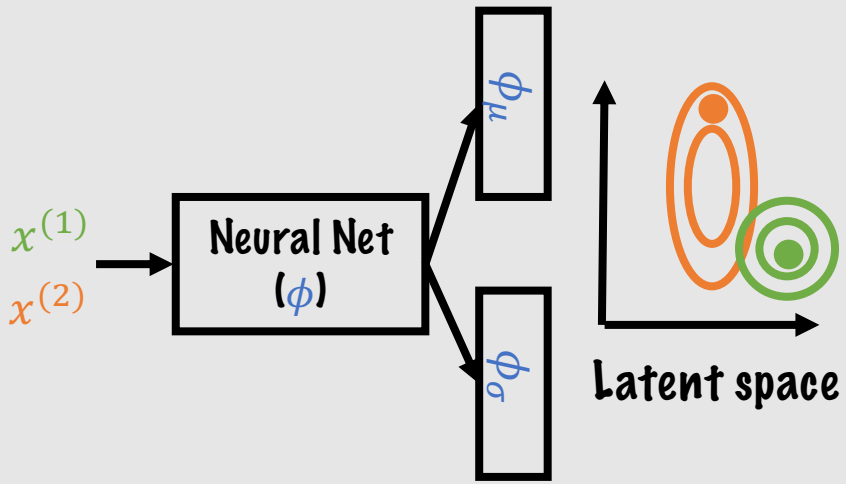
$$\mathcal{L}(x, \theta, \phi) = \mathbb{E}_{q(z|x;\,\phi)}\left[\log \frac{p(x, z;\,\theta)}{q(z|x;\phi\,)}\right] =$$

$$\underbrace{\mathbb{E}_{q(z|x;\phi)}[\log p(x|z;\theta)]}_{\text{Reconstruction}} \underbrace{-D_{KL}(q(z|x;\phi)||p(z))}_{\text{Regularization}}$$



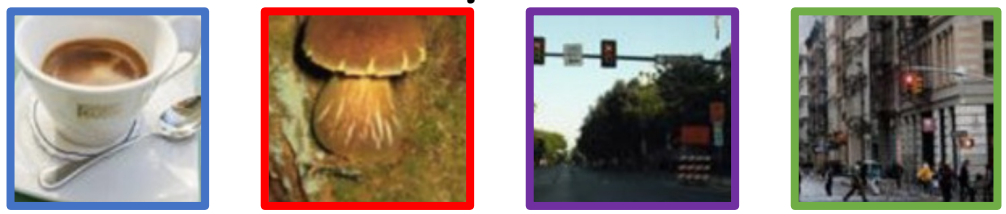
Latent variables in a trained VAE

Inference model

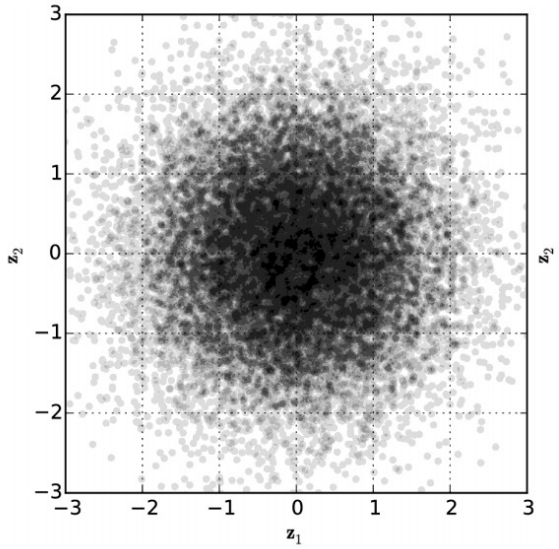


$D_{KL}\left(q(z|x^{(i)};\phi)||p(z)\right)$

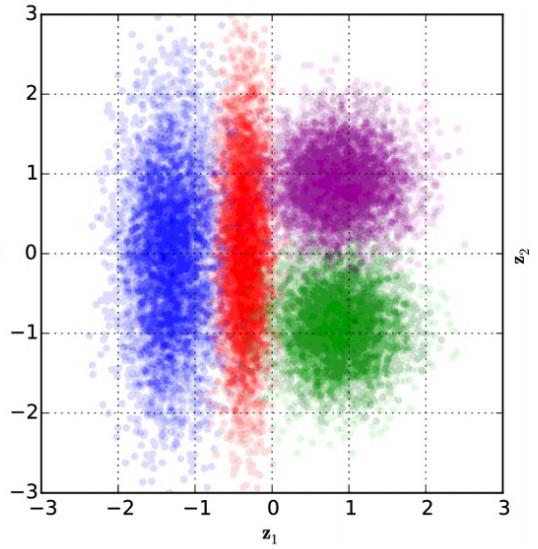
4 data points:



prior

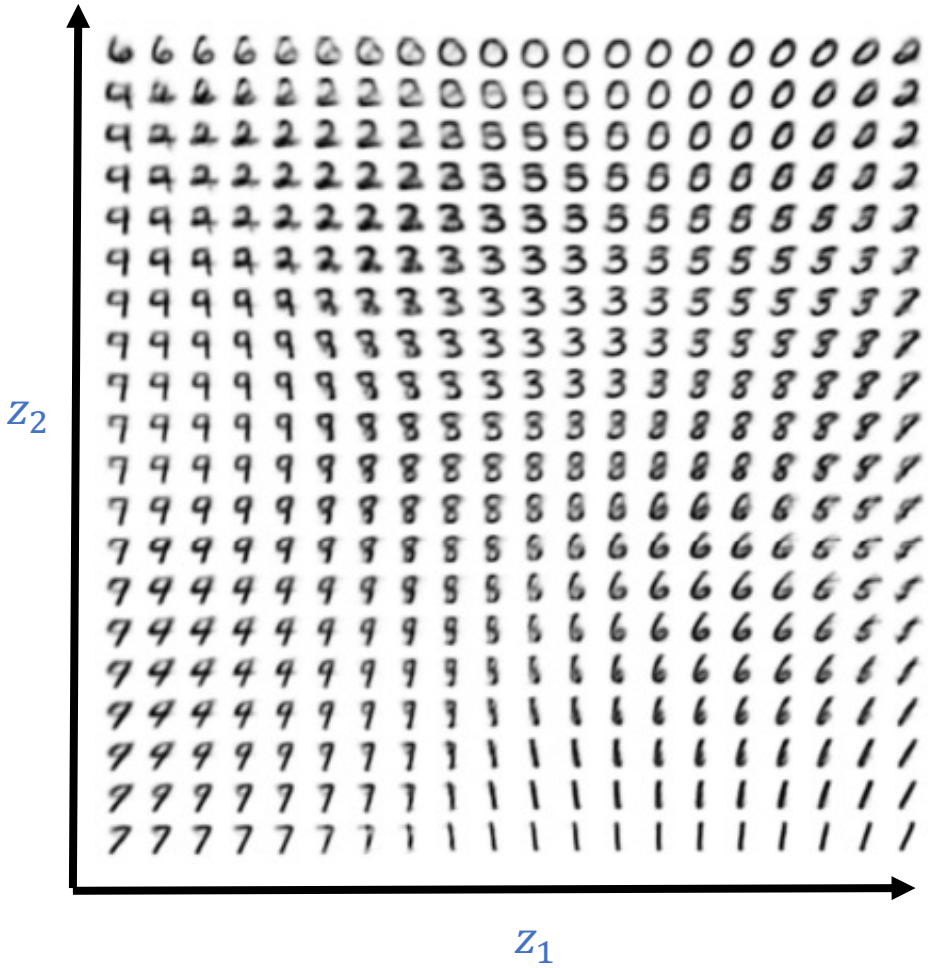


VAE posterior



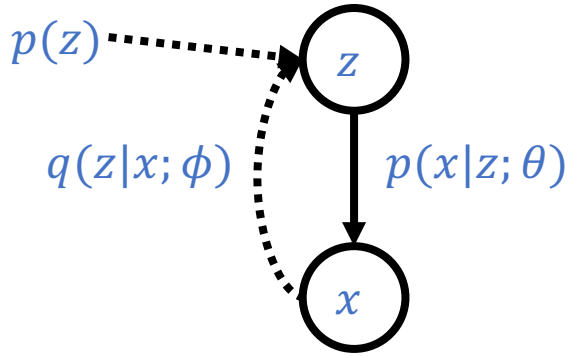
Latent space interpolation

Interpolation in the latent space



Current VAEs

VAE conclusions



\+

Principled approach to generative models
Allows the inference of posterior, useful in representation learning
Easy to sample

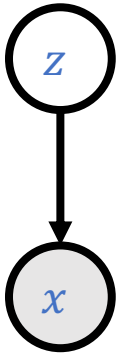
\-

Maximizes ELBO, hard to be SotA
Not easy to specify flexible posteriors

Index

Generative Diffusion Processes

Latent Variable models



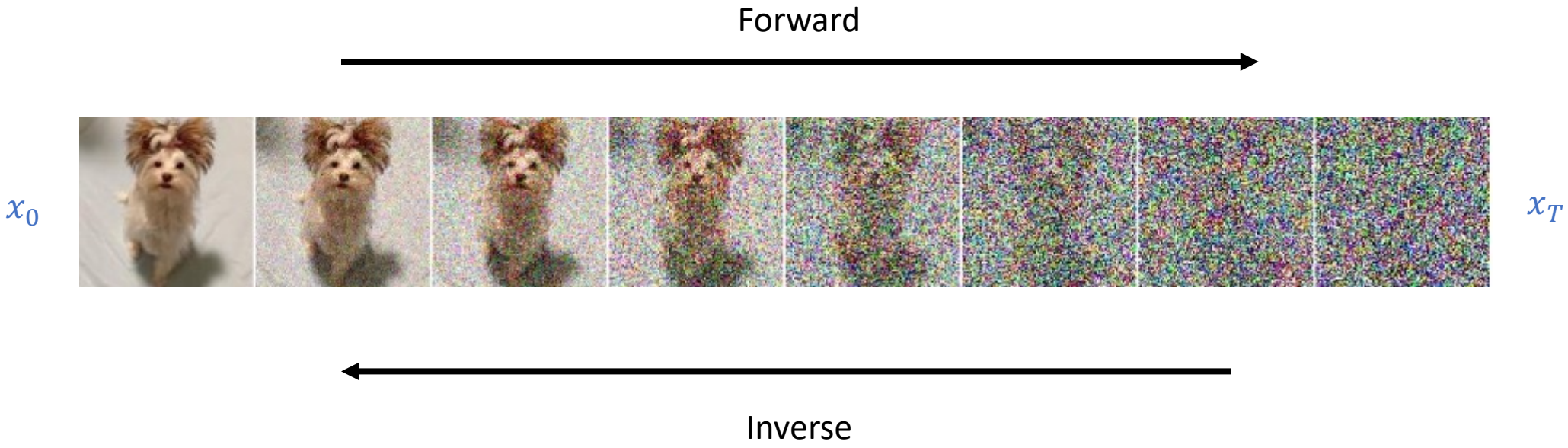
Hard to compute...

$$p_{model}(x^{(i)}) = \int p_{model}(x^{(i)}, z)dz$$

Variational inference: use an approximation to transform the integral into an expectation over a simple, known distribution.

Probabilistic Diffusion Models: gradually remove noise from the image.

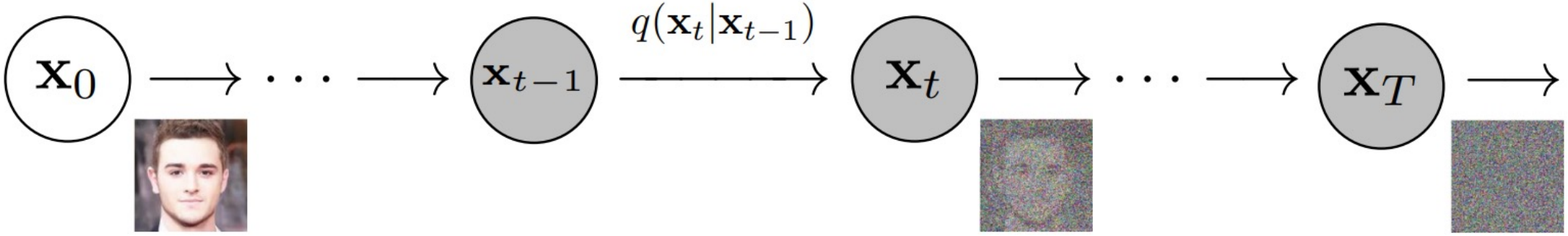
Forward and inverse diffusion processes



Forward process details

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) := \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I})$$

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) := \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}),$$



https://www.assemblyai.com/blog/diffusion-models-for-machine-learning-introduction/

Noise schedule in the forward process

Beta controls the noise schedule.

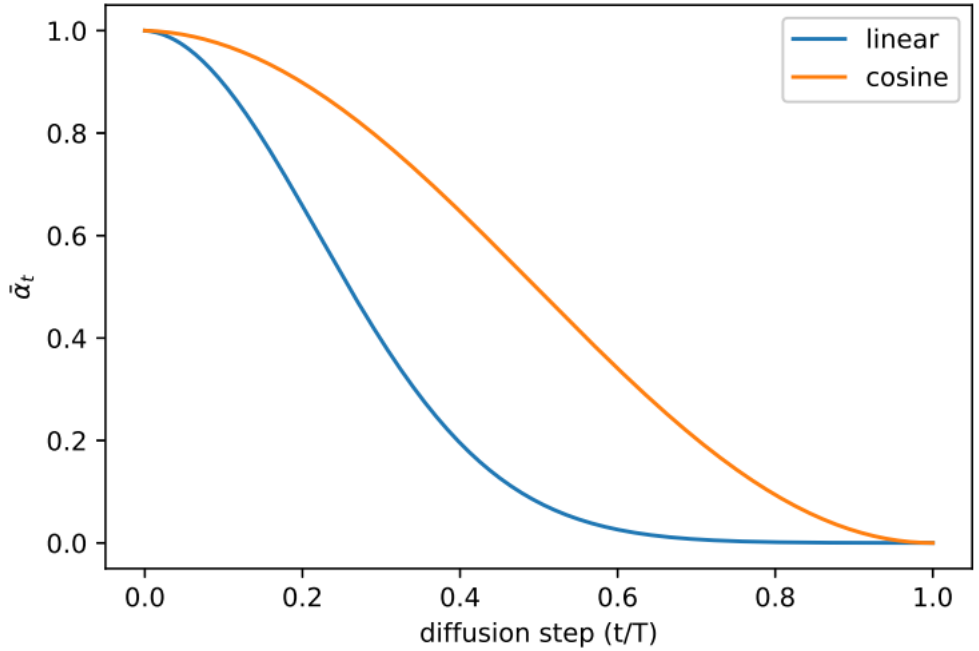
$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) := \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I})$$

We can rewrite the forward process to condition it on x_0.

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I})$$

$$\alpha_t := 1 - \beta_t \text{ and } \bar{\alpha}_t := \prod_{s=1}^{t} \alpha_s$$

Sample any x_t while having access only to x_0!



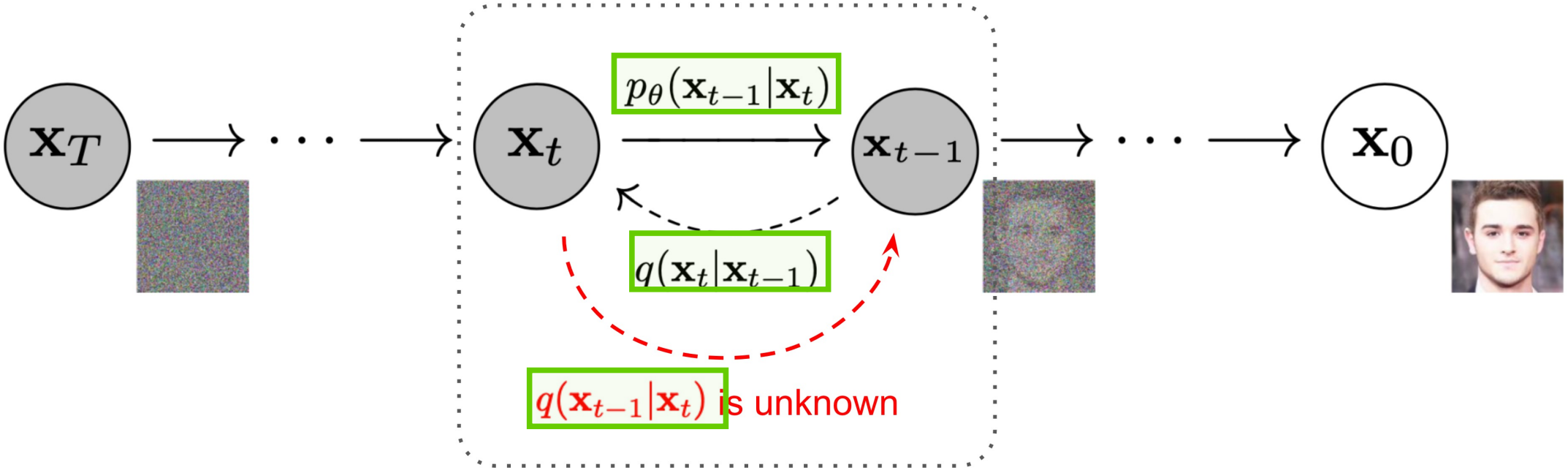
Linear beta schedule

Cosine beta schedule

Inverse process details

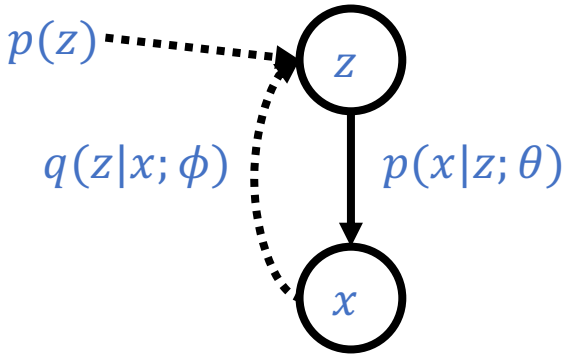
$$p_\theta(\mathbf{x}_{0:T}) := p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t),$$

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) := \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t))$$

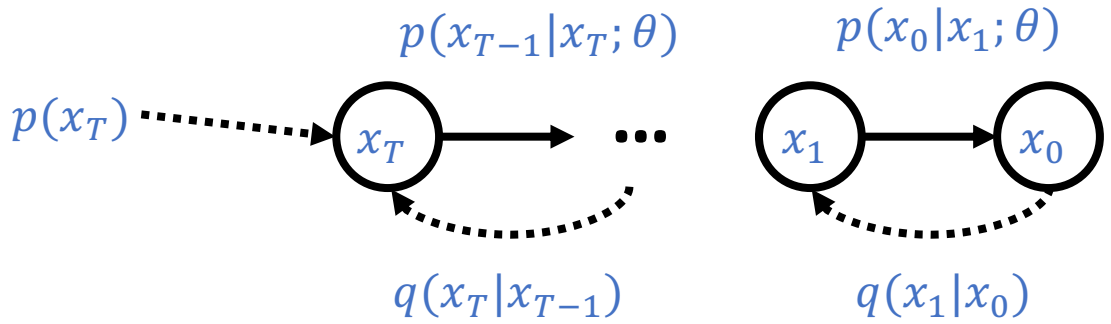


https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

Diffusion Probabilistic Model



VAE



Diffusion Probabilistic model

Training objective (ELBO)

$$\mathbb{E}\left[-\log p_\theta(\mathbf{x}_0)\right] \leq \mathbb{E}_q\left[-\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\right] :$$

$$\mathbb{E}_q\Big[\underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_T|\mathbf{x}_0) \parallel p(\mathbf{x}_T))}_{L_T} + \sum_{t>1} \underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) \parallel p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))}_{L_{t-1}} \underbrace{-\log p_\theta(\mathbf{x}_0|\mathbf{x}_1)}_{L_0}\Big]$$

This is constant and can be ignored

$$\mathbb{E}_{t,\mathbf{x}_0,\boldsymbol{\epsilon}}\left[\left\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t)\right\|^2\right]$$

Predict the noise injected at each timestep.

$$\prod_{i=1}^{D} \int_{\delta_-(x_0^i)}^{\delta_+(x_0^i)} \mathcal{N}(x; \mu_\theta^i(\mathbf{x}_1, 1), \sigma_1^2)\, dx$$

For details see: https://arxiv.org/pdf/2006.11239.pdf

Training and sampling

Algorithm 1 Training

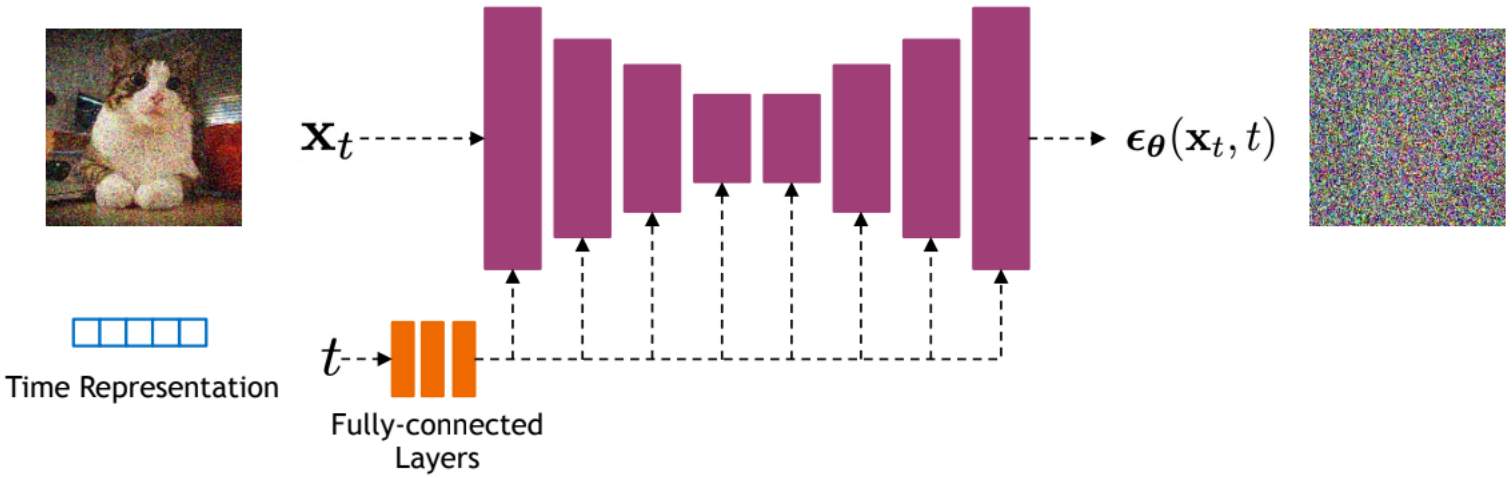
1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
6: **until** converged

Algorithm 2 Sampling

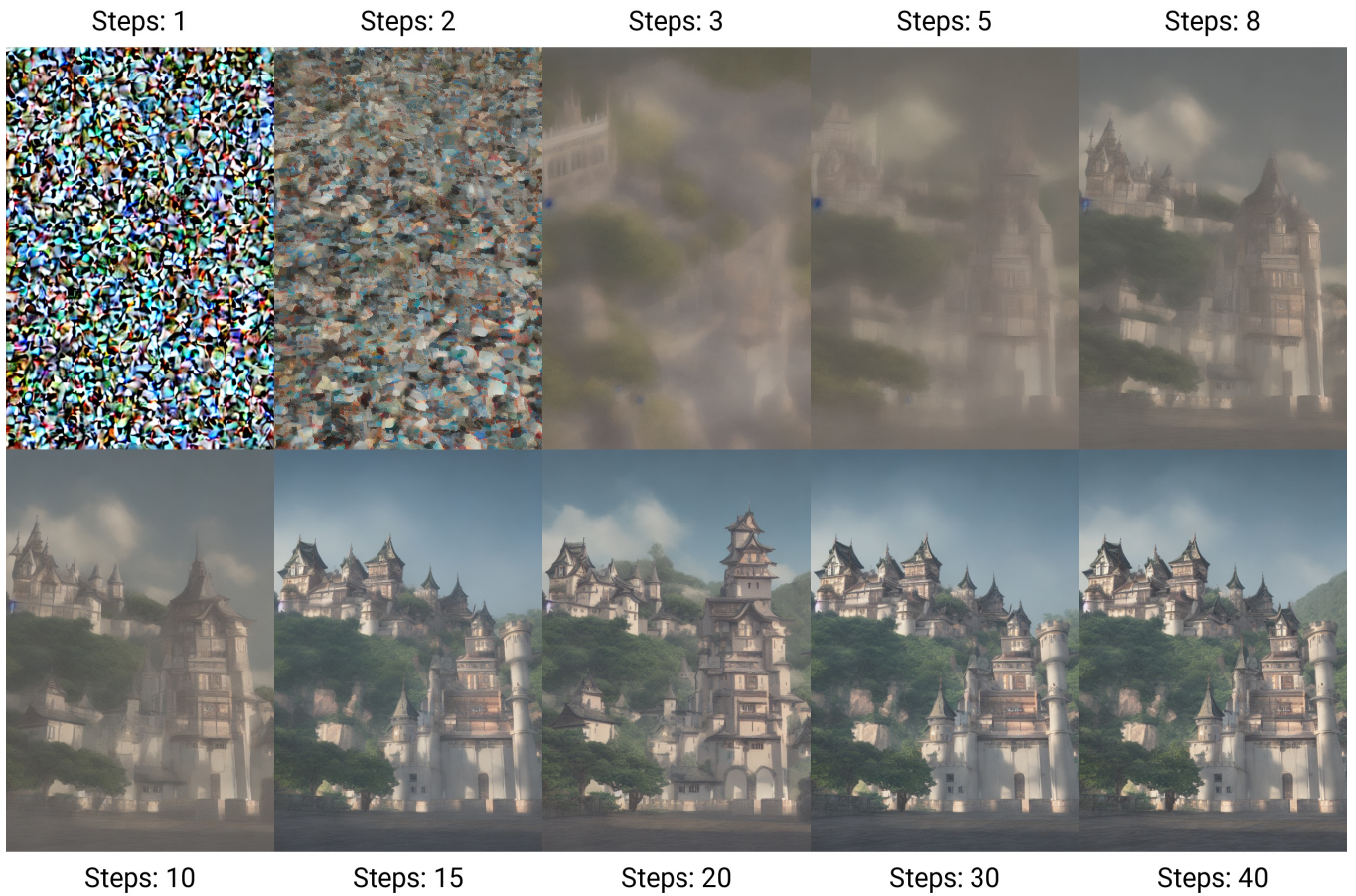
1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

For details see: https://arxiv.org/pdf/2006.11239.pdf

Architecture: Unet-like



Denoising process progress



Results

Score-based perspective on diffusion models

Give up modeling of the density: $\max_{\theta} p_{model}(X;\theta)$

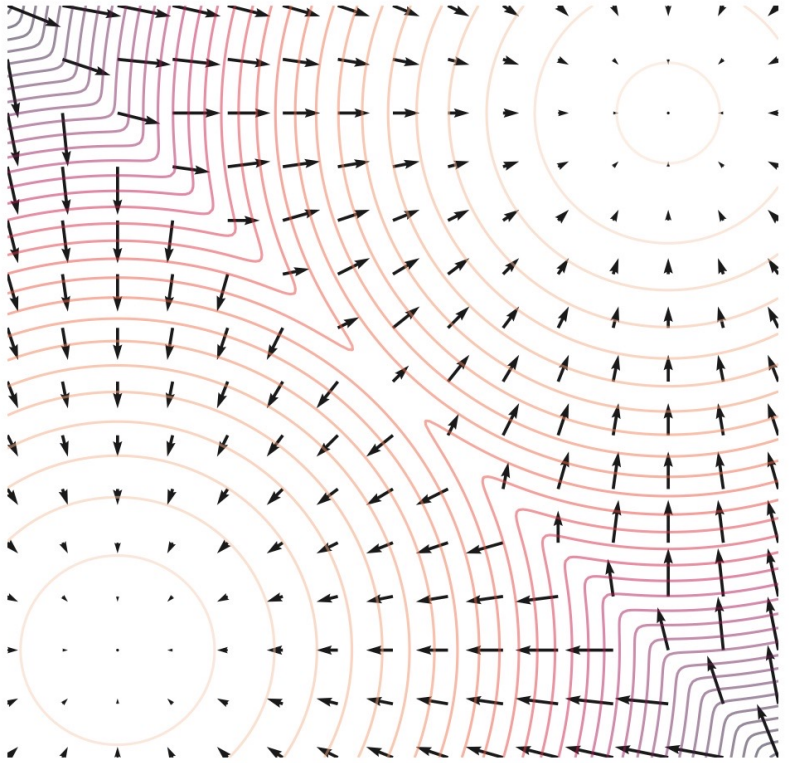
Focus on score instead: $\nabla_X p_{model}(X;\theta)$

Note that the score is not the same as: $\nabla_\theta p_{model}(X;\theta)$

Score function 2D intuition

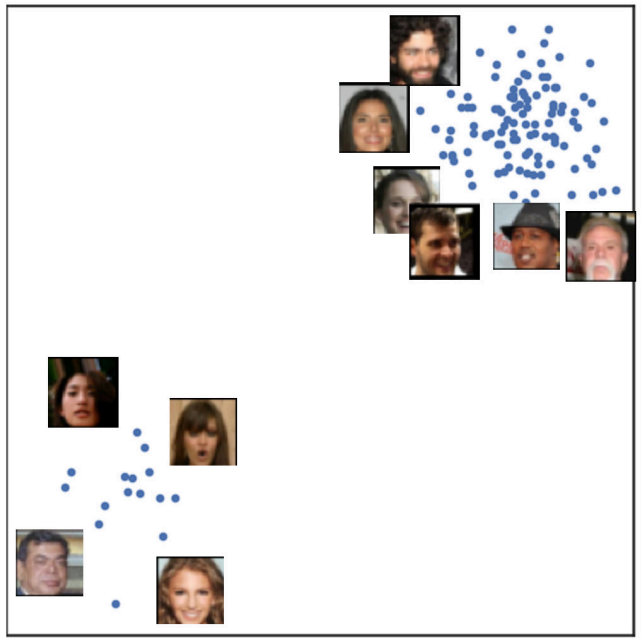
Score function: $\nabla_x \log p(x)$

Not to be confused with: $\nabla_\theta \log p(x;\theta)$



https://yang-song.net/blog/2021/score/

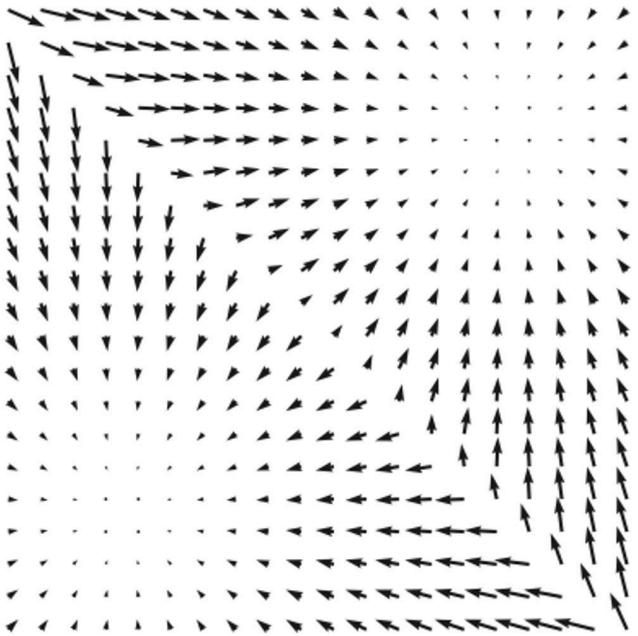
Score based generative models



Data samples

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

score matching



Scores

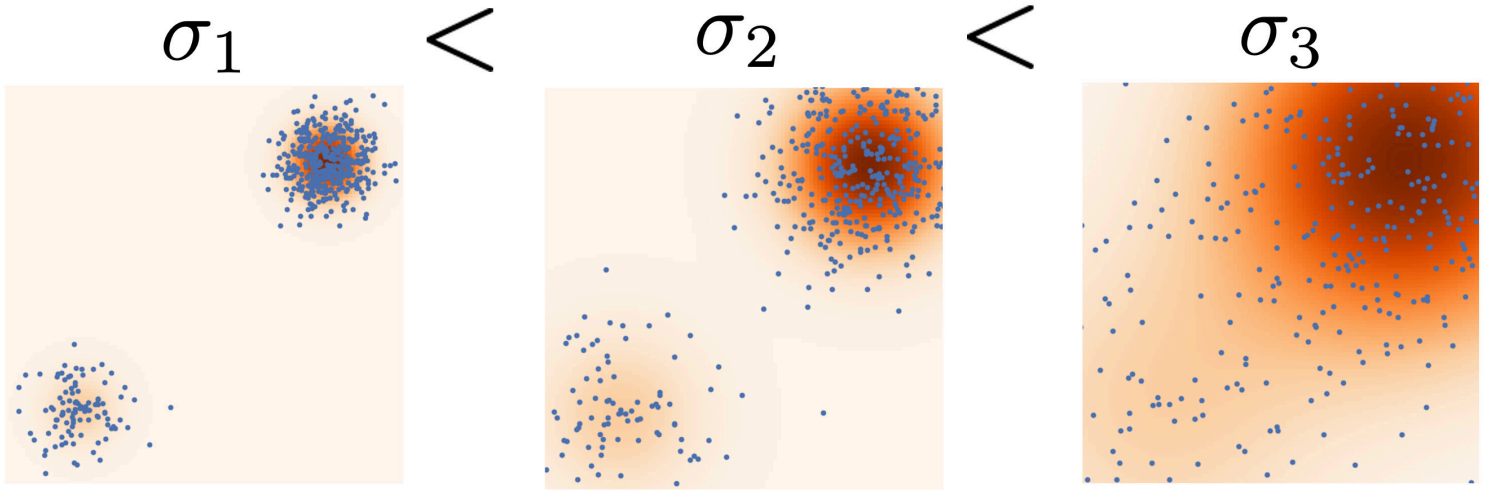
$$\mathbf{s}_\theta(\mathbf{x}) \approx \nabla_{\mathbf{x}} \log p(\mathbf{x})$$

Langevin dynamics

New samples

https://yang-song.net/blog/2021/score/

Score based models with multiple levels of perturbations



ELBO used for training diffusion probabilistic models is essentially equivalent to the weighted combination of score matching objectives used in score-based generative modeling.

https://yang-song.net/blog/2021/score/

Probabilistic Diffusion Models Summary

+

High quality samples with good distribution coverage
State-of-the-art results
Guidance easily extends the unconditional models to conditional ones

-

Slow to sample (iterative sampling)
Latent space lacks semantics

Index

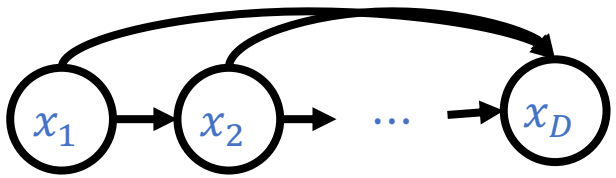
Let's play... real vs. fake

http://www.whichfaceisreal.com/index.php

Implicit model

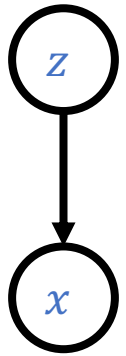
$$p_{model}(X;\theta) = \prod_{i=1}^{N} p_{model}\left(x^{(i)};\theta\right)$$

Autoregresive models



$$p_{model}\left(x_{1:D}^{(i)};\theta\right) = p_{model}\left(x_1^{(i)};\theta\right)\prod_{j=2}^{D} p_{model}\left(x_j^{(i)}\middle|x_{<j}^{(i)};\ \theta\right)$$

Latent Variable models



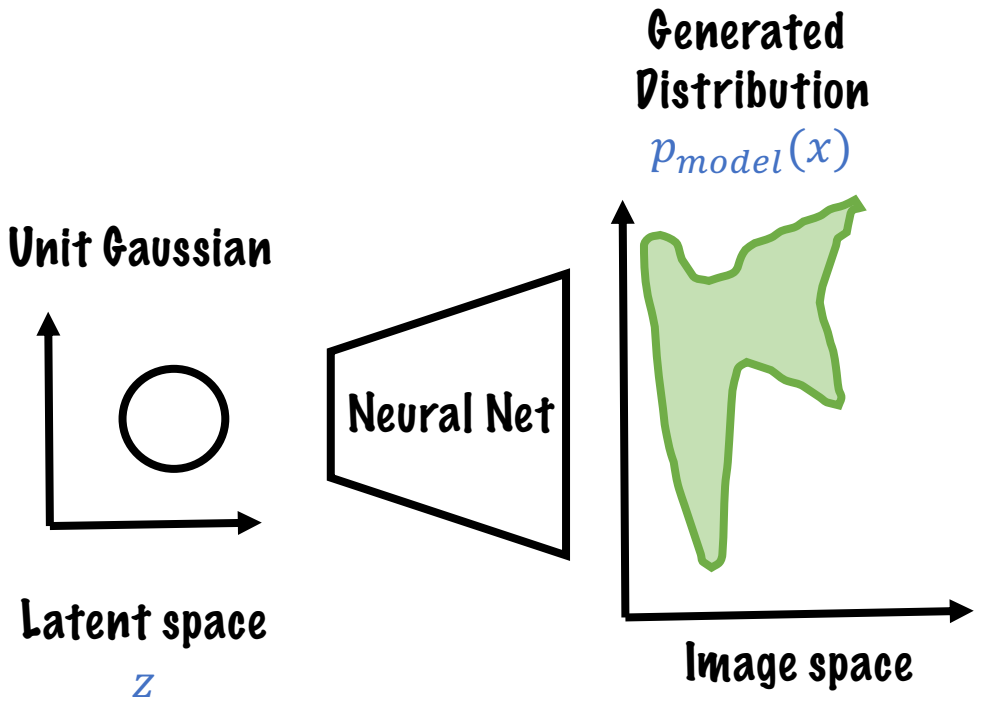
Hard to compute...

$$p_{model}\left(x^{(i)}\right) = \int p_{model}\left(x^{(i)},z\right)dz$$

Variational inference: use variational approximation

Implicit models: give up modeling the density $p_{model}\left(x^{(i)}\right)$

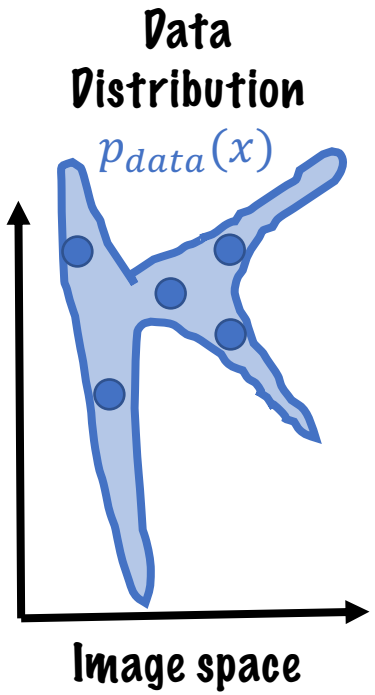
Implicit model



$$p_{model}(x) = p_{data}(x)$$

Two samples test:

$$\frac{p_{model}(x)}{p_{data}(x)} = 1$$



We do not care about the probabilities itself, we rather think how they are related.

Two sample test

Two sample test:

$$\frac{p_{model}(x)}{p_{data}(x)} = 1$$

$$p_{model}(x) = p_{data}(x)$$

We have two distributions -> 2 classes.

Let's write $p_{data}(x) = p(x|y=1)$ and $p_{model}(x) = p(x|y=-1)$.

$$\frac{p_{model}(x)}{p_{data}(x)} = \frac{p(x|y=-1)}{p(x|y=1)} = \frac{p(y=-1|x)p(x)}{p(y=-1)} \Big/ \frac{p(y=1|x)p(x)}{p(y=1)}$$

Bayes rule:

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)}$$

If $p(y=-1) = p(y=1)$ e.g. we have the same # of data from $p_{model}(x)$ and $p_{data}(x)$:

$$\frac{p_{model}(x)}{p_{data}(x)} = \frac{p(y=-1|x)}{p(y=1|x)}$$ ← Classification ratio!

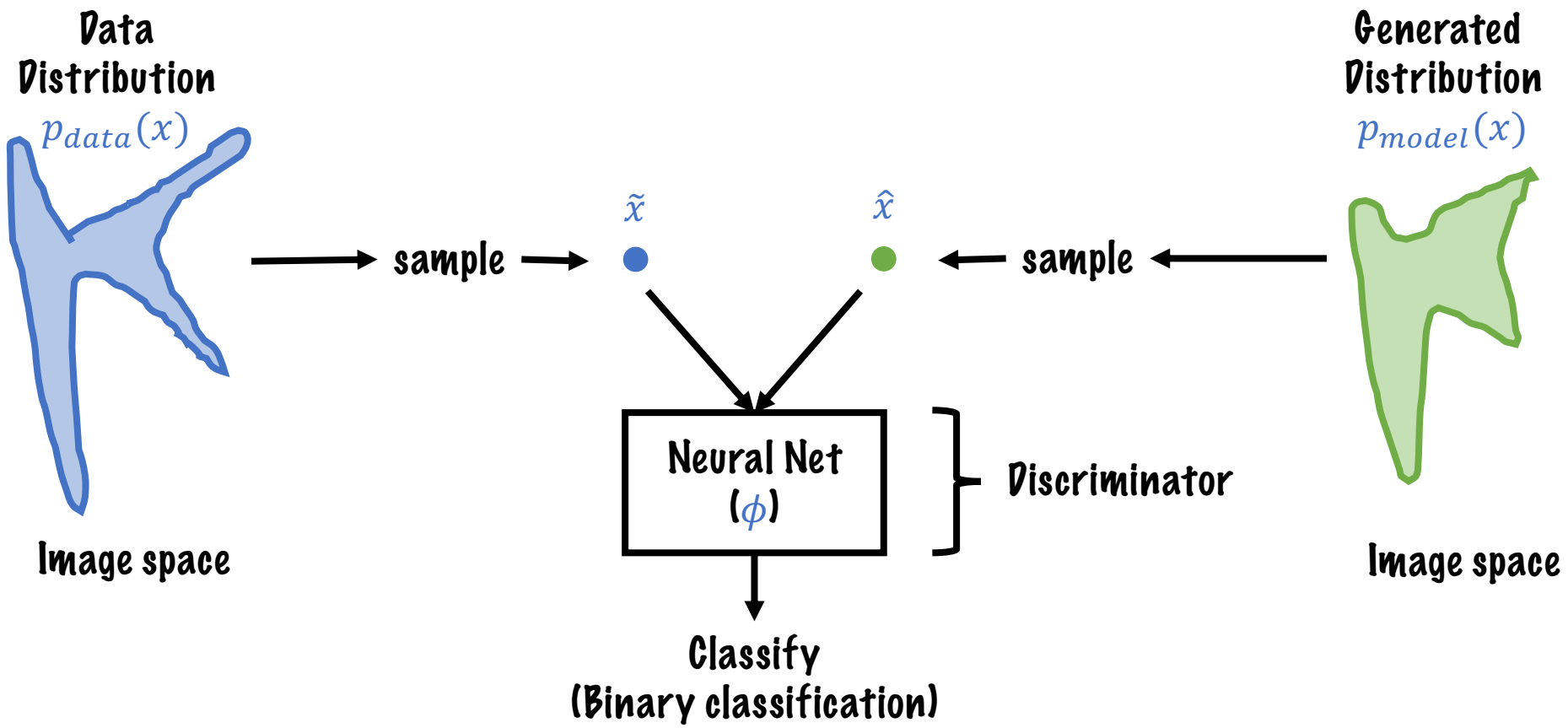
Supervised learning setup!

(x, y)

$x \in \{\hat{x}, \tilde{x}\}, y \in \{-1, 1\}$

Where $\hat{x}$ denotes samples form the model distribution $\hat{x} \sim p_{model}(x)$ and $\tilde{x}$ denotes samples form the data distribution $\tilde{x} \sim p_{data}(x)$.

Adversarial Loss



$$\log p(y|x;\phi) = \log p(y=1|\tilde{x};\phi) + \log p(y=-1|\hat{x};\phi) =$$
$$\log p(y=1|\tilde{x};\phi) + \log(1 - p(y=1|\hat{x};\phi))$$

Generative Adversarial Networks (GAN)

Generative Adversarial Networks

=

Adversarial Loss

+

Latent variable model

Generative Adversarial Networks (GAN) with perfect discriminator

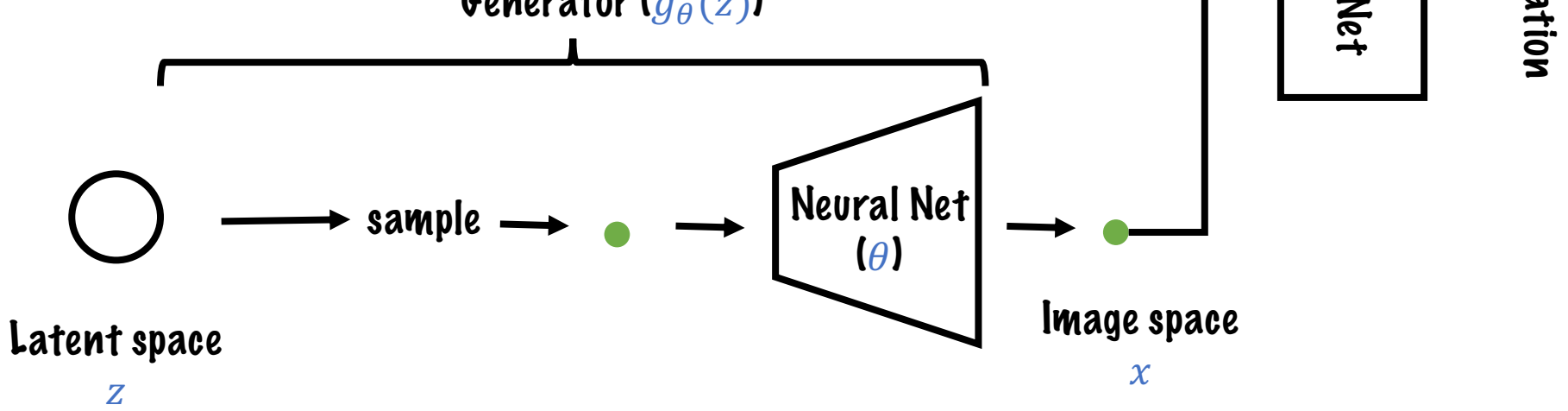
Let us assume to have access to a perfect discriminator.
We do not need to have access to the data distribution.
Goal: Learn the parameters of the generator.

Learning objective:

$$\underset{\theta}{\operatorname{argmin}}\, \mathcal{F}(x,\theta) = \underset{\theta}{\operatorname{argmin}}\, \mathbb{E}_{p_{gen}}[\log(1 - p(y=1|\hat{x}))]$$

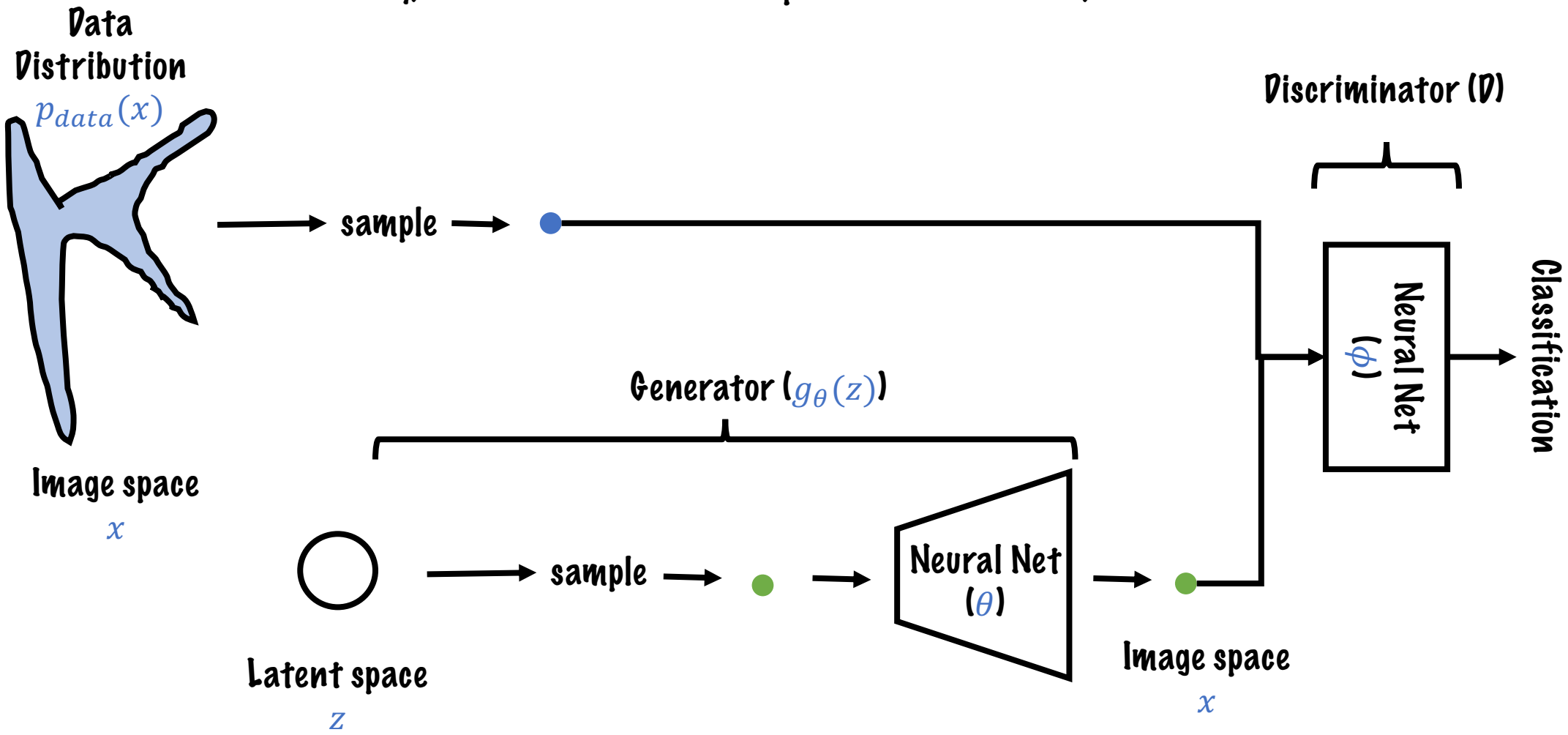
Fool the discriminator.
Easy to optimize.

$$\underset{\theta}{\operatorname{argmin}}\, \mathcal{F}(x,\theta) = \underset{\theta}{\operatorname{argmin}}\, \mathbb{E}_{p(z)}[\log(1 - p(y=1|g_\theta(z)))]$$



Generative Adversarial Networks (GAN)

Normally, we do not have access to perfect discriminator, we have to learn it.



Generative Adversarial Networks (GAN)

Adversarial loss have the following criterion:

$$\mathcal{F}(x, \theta, \phi) = \mathbb{E}_{p_{data}}[\log p(y = 1|\tilde{x}; \phi)] + \mathbb{E}_{p_{gen}}[\log(1 - p(y = 1|\hat{x}; \phi))]$$

We can add a generator to the criterion:

$$\mathcal{F}(x, \theta, \phi) = \mathbb{E}_{p_{data}}[\log p(y = 1|\tilde{x}; \phi)] + \mathbb{E}_{p(z)}[\log(1 - p(y = 1|g_\theta(z); \phi))]$$

The learning criteria of GAN is the following:

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

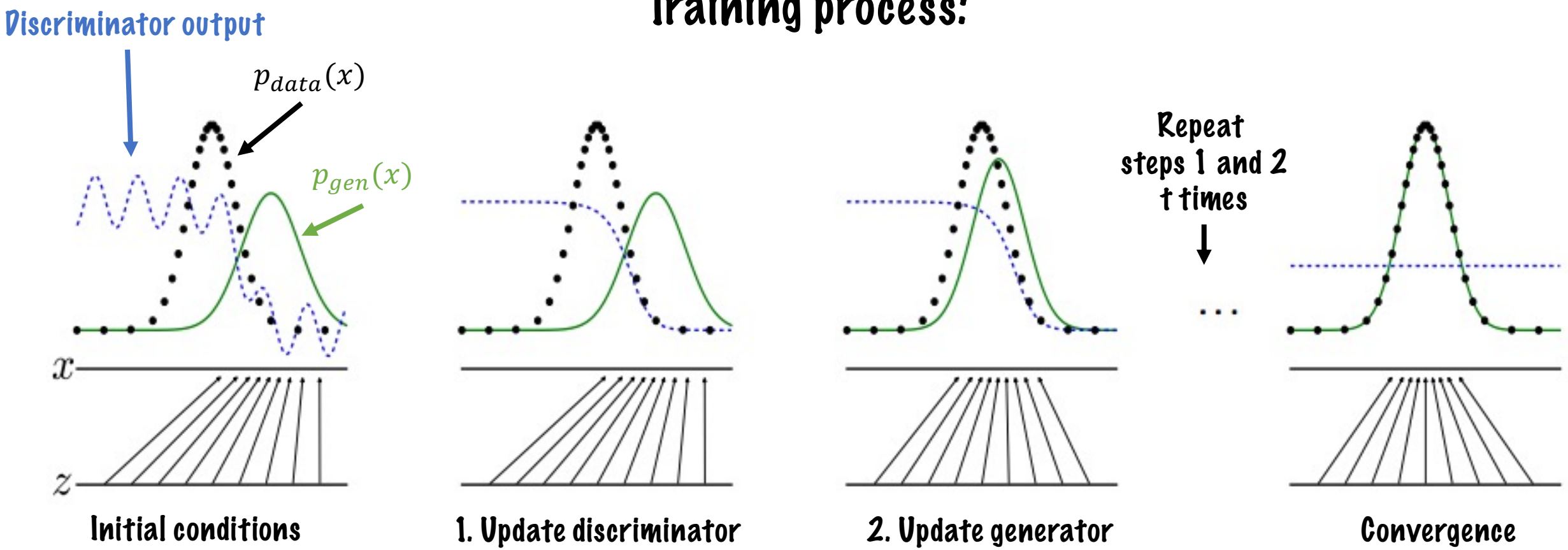
Learn a generator that fools discriminator.

Learn a good discriminator.

The learning objective of GAN is also referred to as min max game.

Generative Adversarial Networks (GAN)

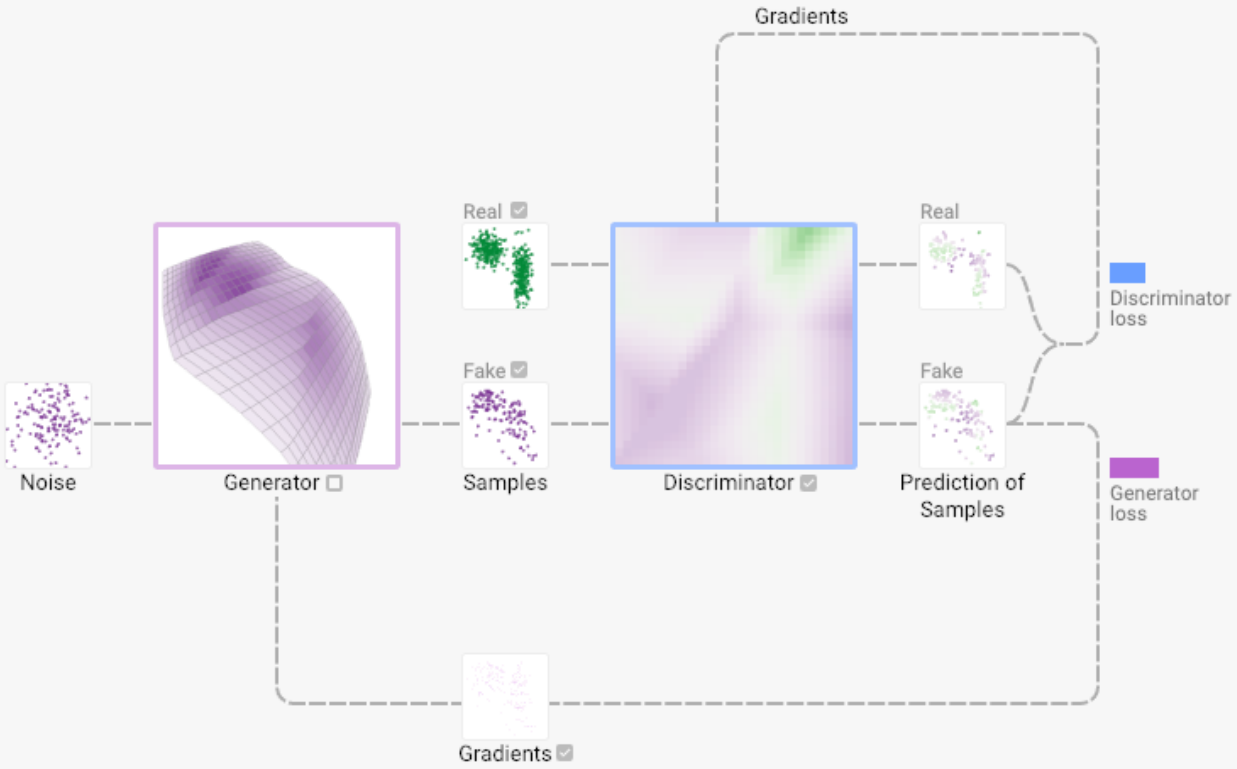
Training process:



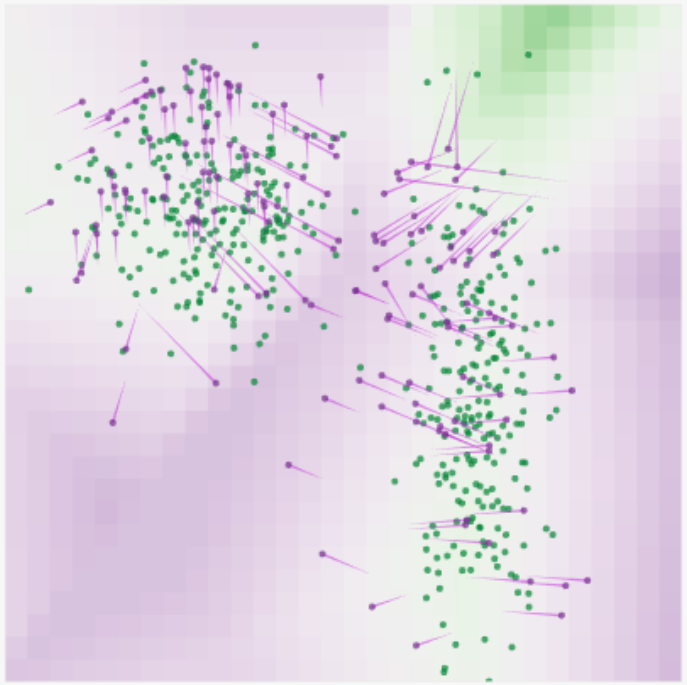
Initial conditions | 1. Update discriminator | 2. Update generator | Convergence

GAN lab

MODEL OVERVIEW GRAPH



LAYERED DISTRIBUTIONS



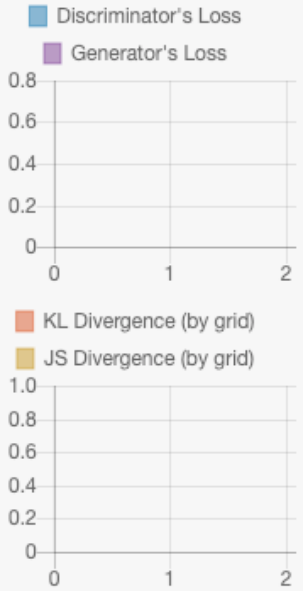
Each dot is a 2D data sample: **real samples**; **fake samples**.

Background colors of grid cells represent **discriminator**'s classifications.
Samples in **green regions** are likely to be real; those in **purple regions** likely fake.

Manifold represents **generator**'s transformation results from noise space.
Opacity encodes density: darker purple means more samples in smaller area.

Pink lines from fake samples represent **gradients** for generator.
This sample needs to move upper right to decrease generator's loss.

METRICS



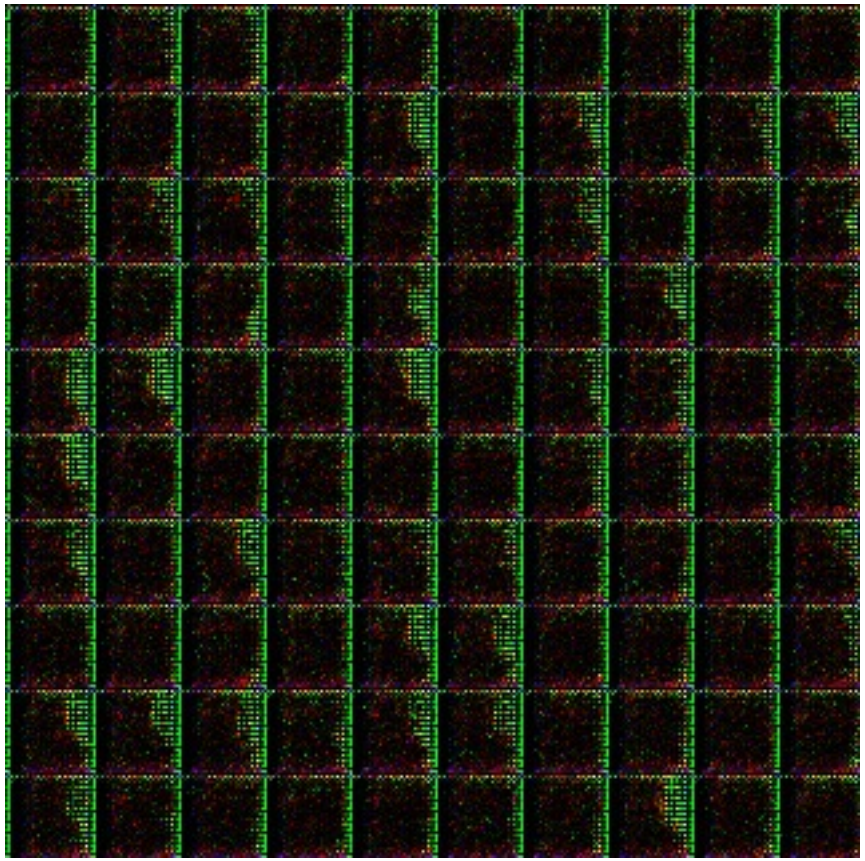
https://poloclub.github.io/ganlab/

Why is it hard to train a GAN?

- Mode collapse: generator models very well a small sub-space, concerning on a few modes.
- No convergence: there is no guarantee that min max game will actually converge.

VAE vs GAN training

VAE



GAN

BIGGAN
Samples

GigaGAN

A photo of a ramen taken from an angle, with some background.

A digital illustration of the Grand Canyon on the moon, depicted in shades of grey.

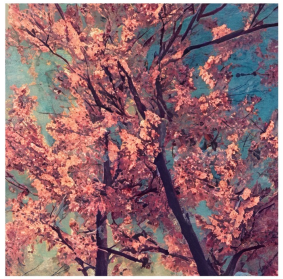
Smooth dining table with meats in an elegant restaurant with soft lighting.

The black hole in the space, observed by the Hubble Telescope.

Flowers in shape of a heart.

CG art of a majestic castle, evoking a sense of splendor.

Magritte painting of a clock on a beach.



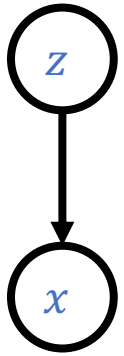
Cherry blossom trees, pastel watercolor style, 2K.

The New York skyline, 4k landscape photography.

Portrait of an old Asian man in Saharan Desert, dusk.

A futuristic city with a cyberpunk vibe, captured with long camera exposure

GAN conclusions



+

Easy to sample
Nice image generations

-

Hard to train (unstable)
Don't have inference model $p(z|x^{(i)})$
We cannot compute likelihood $p_{model}(X;\theta)$

Index

Evaluation

Evaluation depends on the exact task we solve with our generative model.

Evaluation of image generation is tricky.

- Image generation metrics:
  - We can evaluate data points (e. g. test set)
    - How likely is it to observe an image under the model? (Likelihood)
  - We can evaluate samples from the model
    - Are the samples plausible? (Visual evaluation or Inception score)

Test set likelihood

We want to compute the log likelihood over test set:

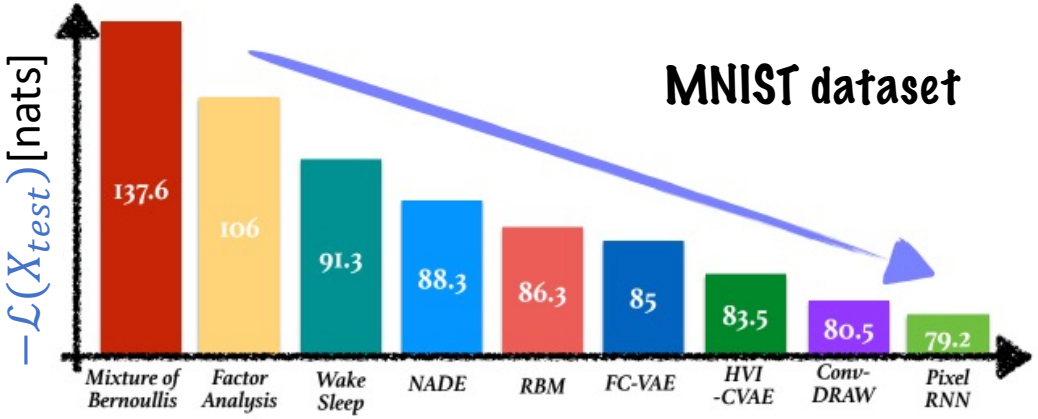
$$\mathcal{L}(X_{test}) = \log p_{model}(X_{test}\,;\theta)$$

That is expectation (mean log likelihood) over test set:

$$\mathcal{L}(X_{test}) = \mathbb{E}_{X_{test}} \log p_{model}(x^{(i)}_{test}\,;\theta)$$

In some papers people will report negative log likelihood: $-\mathcal{L}(X_{test})$

The base for the $\log$ determines units, can be either 2 (bits) or e (nats).



We cannot compute this metric for implicit models.

Figure from: https://drive.google.com/file/d/0B_wzP_JlVFcKMnFMNnAtYTVkV28/view

Visual quality of samples

Subjective metric!

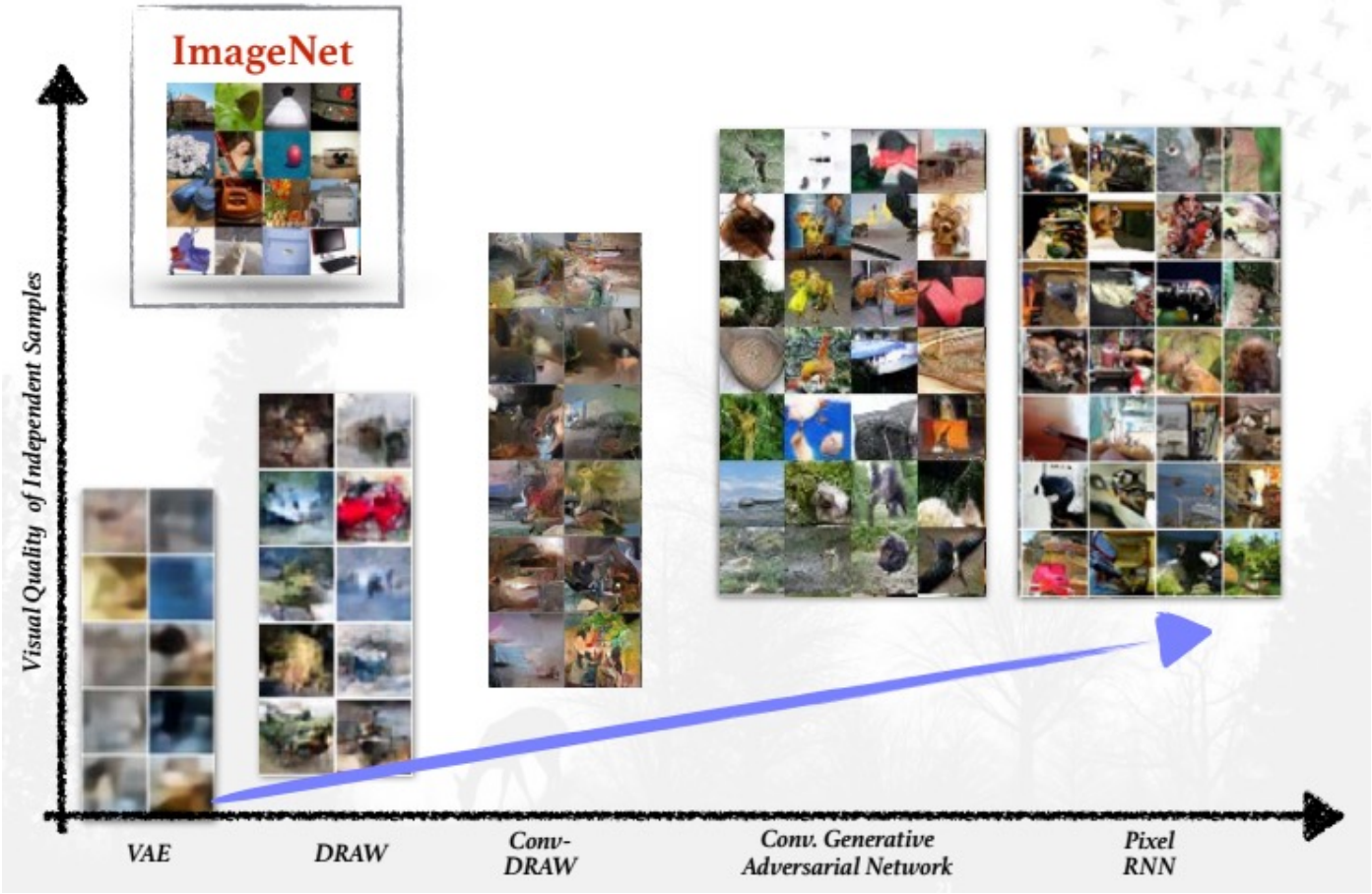
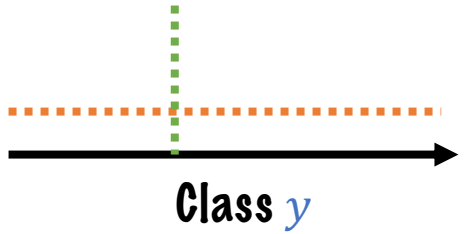


Figure from: https://drive.google.com/file/d/0B_wzP_JlVFcKMnFMNnAtYTVkV28/view

Inception score (IS)

Use inception v3 model trained on ImageNet to evaluate the samples.

The generative model should create samples that are:
Variable (all classes of imagenet are probable)
For a given data point, the inception model should be highly confident



$$IS = \exp(\mathbb{E}_{x \sim p_{model}} D_{KL}(p_{inception}(y|x)||p(y))$$

IS is not perfect.

Samples with high IS.

Fréchet Inception Distance (FID)

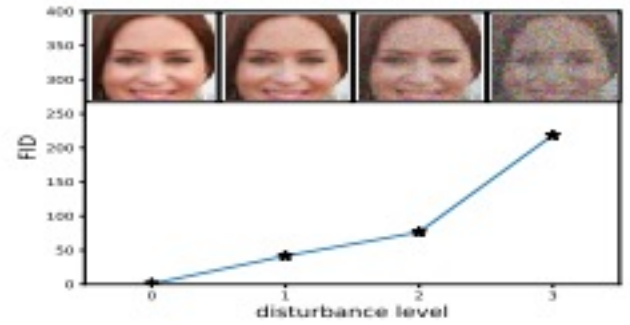
Use inception model trained on ImageNet to evaluate the samples.

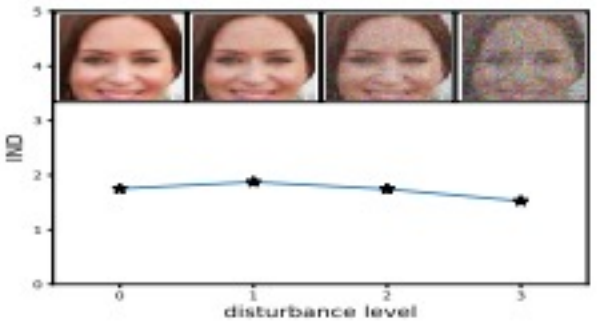
Compare the statistics of generated samples.

-> Fit a gaussian to an inception model activations at some level using real test set and generated images.

-> Compute the FID distance between the gaussains.

$$d^2((\boldsymbol{m}, \boldsymbol{C}), (\boldsymbol{m}_w, \boldsymbol{C}_w)) = \|\boldsymbol{m} - \boldsymbol{m}_w\|_2^2 + \mathrm{Tr}\left(\boldsymbol{C} + \boldsymbol{C}_w - 2(\boldsymbol{C}\boldsymbol{C}_w)^{1/2}\right).$$

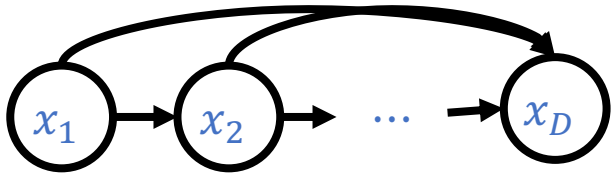




Index

Autoregressive models

$$p_{model}(X;\theta) = \prod_{i=1}^{N} p_{model}\left(x^{(i)};\theta\right)$$

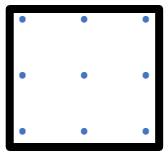


$$p_{model}\left(x_{1:D}^{(i)};\theta\right) = p_{model}\left(x_1^{(i)};\theta\right)\prod_{j=2}^{D} p_{model}\left(x_j^{(i)}\middle|x_{<j}^{(i)};\ \theta\right)$$

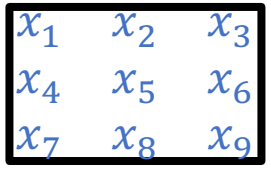
$$x_{<i} = x_1, \cdots, x_{i-1}$$

Notation:
x_i - dimension in x
D - dimension of x

How do we apply this to images? Impose some kind of ordering.



3 x 3 image

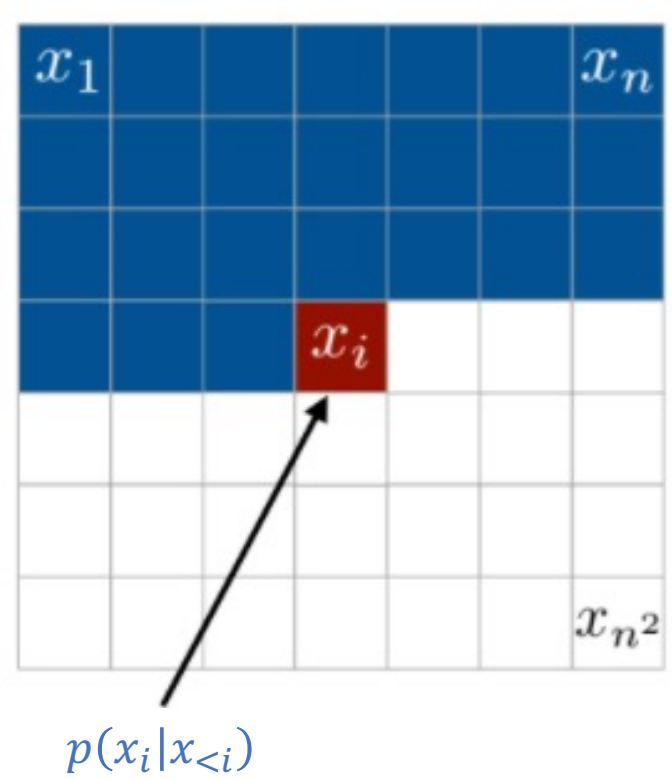


3 x 3 image with an ordering

Autoregressive models

Context:

$x_{<i} = x_1, \cdots, x_{i-1}$

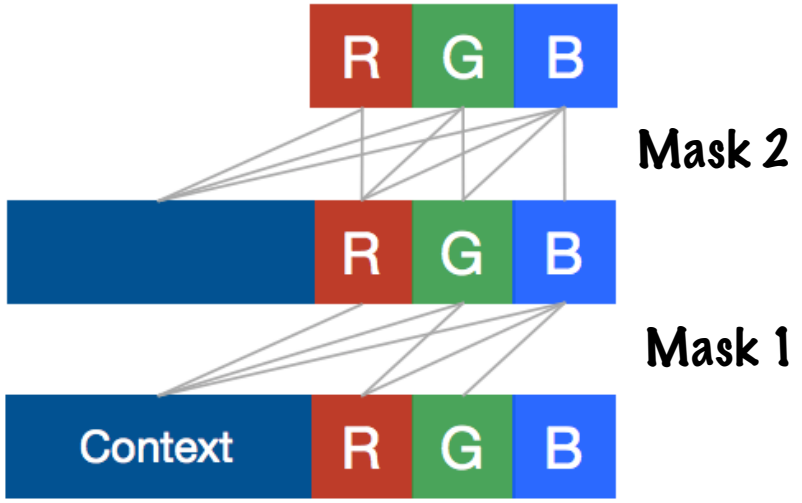


One more problem, images are RGB.

We need to order channels too:

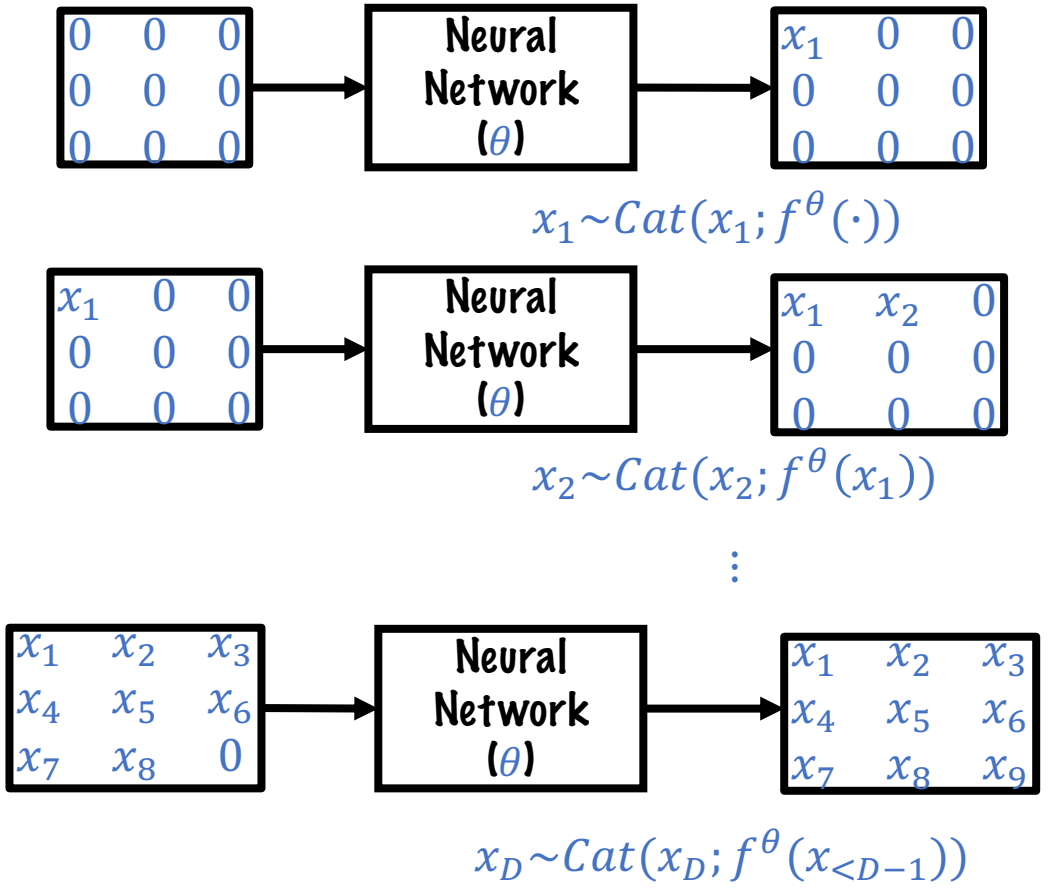
$$p(x_i|x_{<i}) = p(x_{i,R}|x_{<i})p(x_{i,G}|x_{i,R}, x_{<i})p(x_{i,B}|x_{i,G}x_{i,R}, x_{<i})$$

The order on channels is imposed with masks

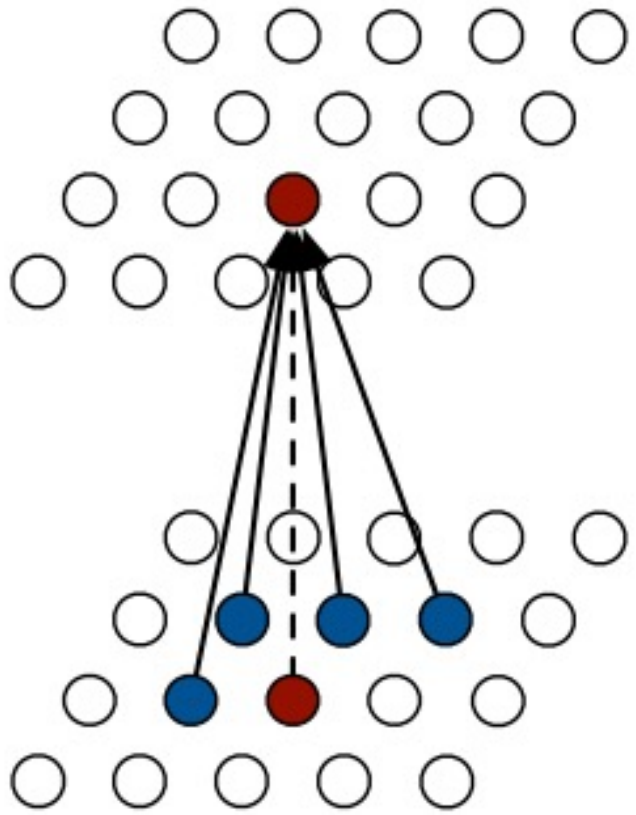


Autoregressive models

Image generation (sampling):



Autoregressive models Architectures



PixelCNN

LSTM

BiLSTM

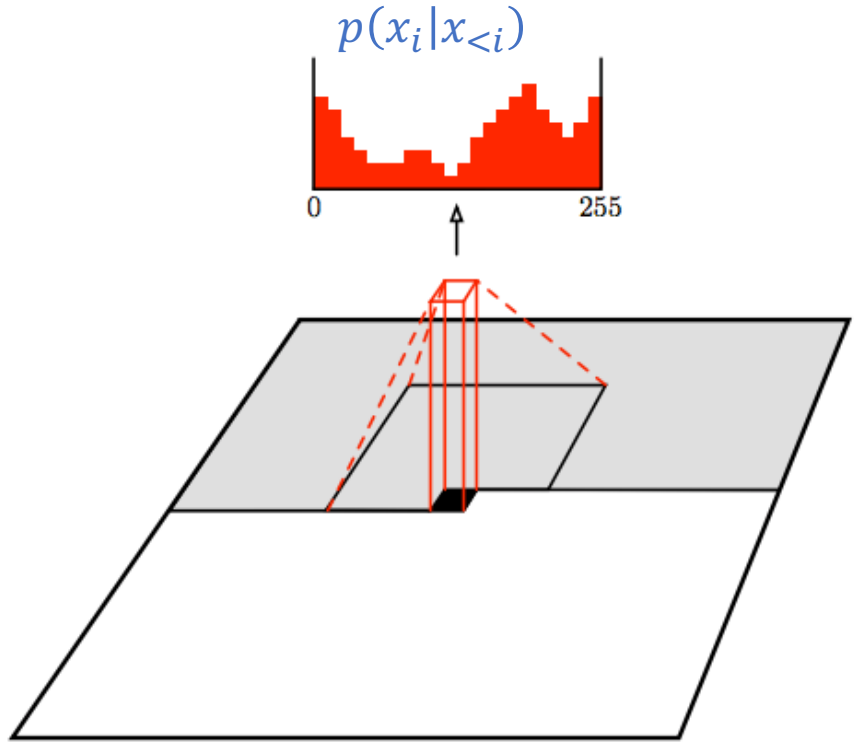
In general, RNNs

PixelCNN

Use masked receptive field as context -> Use masked convolution kernels

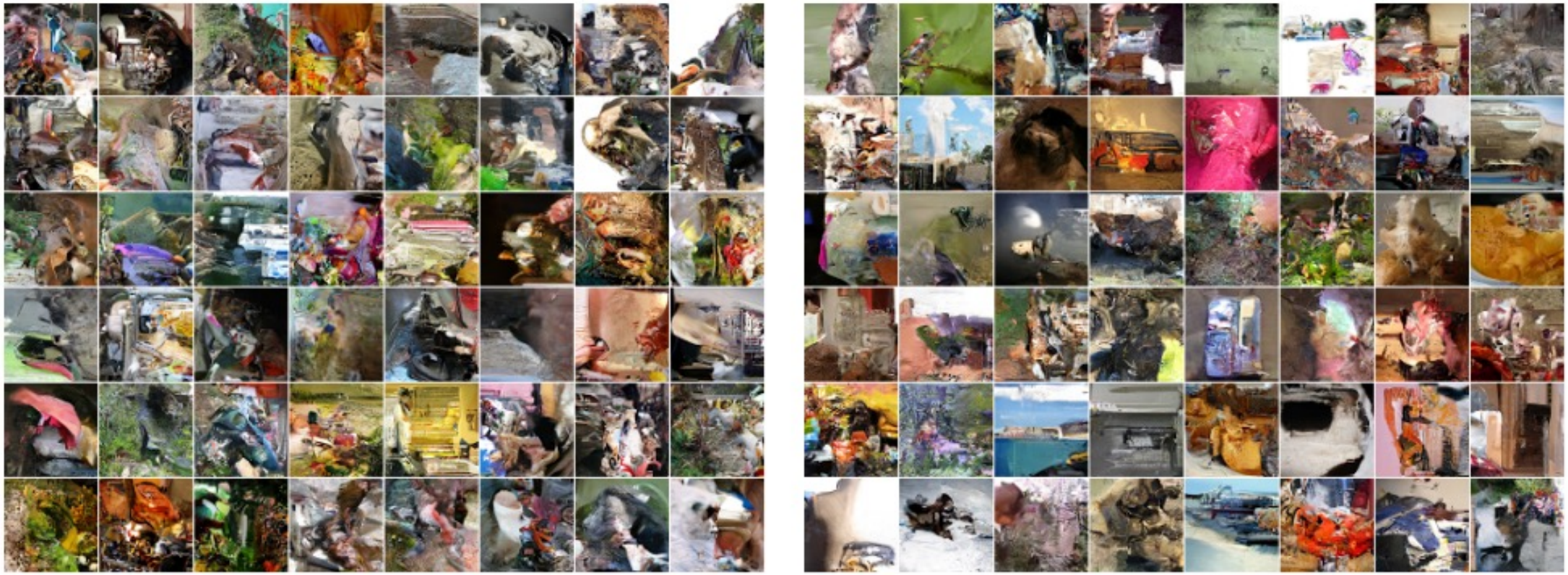
5 x 5 masked convolution kernel

1	1	1	1	1
1	1	1	1	1
1	1	0	0	0
0	0	0	0	0
0	0	0	0	0



PixelCNN Results

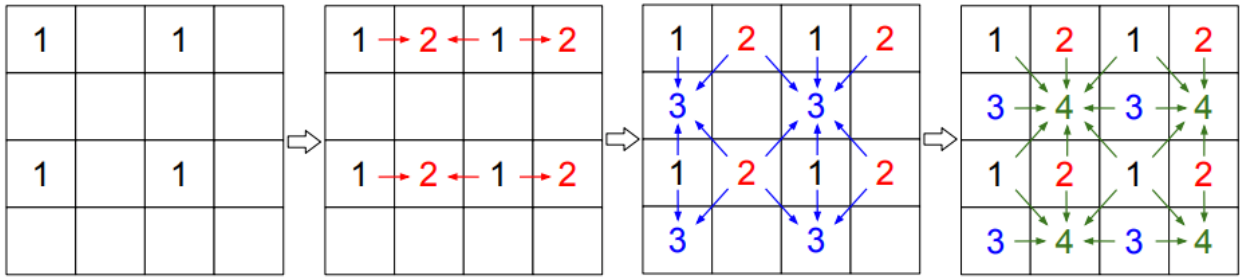
Samples from the model.

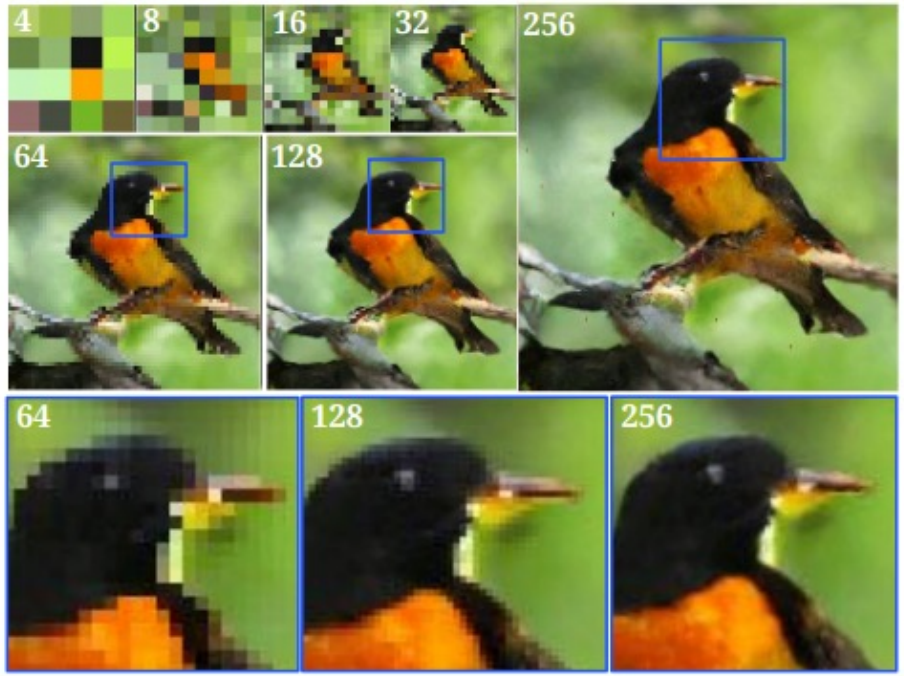


The model captures well local structure, not very well global structure.

Speed up the image generation

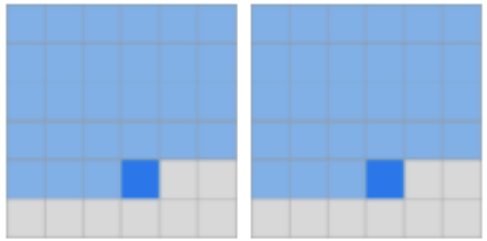
Multiscale generation!



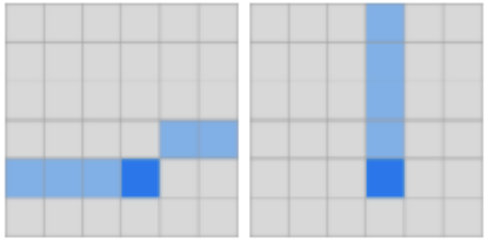


Sparse Transformers

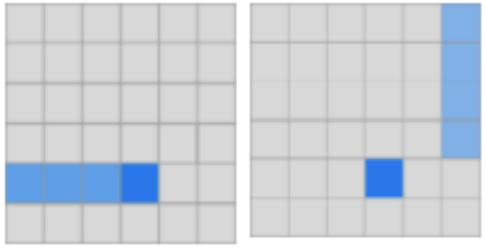
Masked Attention: a good way to implement neighborhood information



Normal transformer

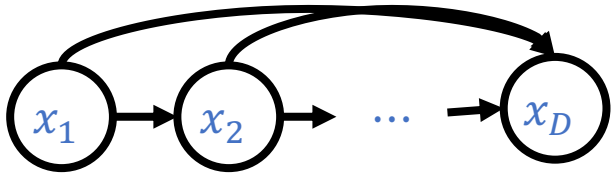


Strided attention



Fixed attention

PixelCNN conclusions



+

Can explicitly compute likelihood
Nice looking samples

-

Sequential generation (slow)
Order sensitive

?

What is the best way of imposing order...

Generative Models

Michal Drozdzal

mdrozdzal@fb.com