LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Deep Learning

Chapter 10: Generative Models

Bernd Bischl

Department of Statistics – LMU Munich

Winter term 2019

RECALL: UNSUPERVISED DEEP LEARNING

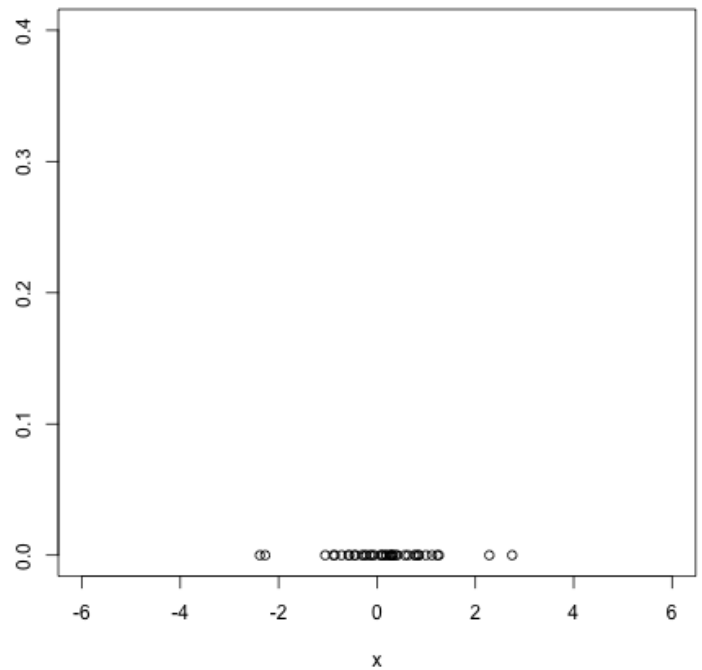
There are two main goals of **unsupervised deep learning learning**:

- **Representation Learning**
  - i.e. manifold learning, feature learning, etc.
  - Can be done by an autoencoder.
  - Applications examples:
    - dimensionality reduction / data compression
    - transfer learning / semi-supervised learning

- **Generative Models / Density Estimation**
  - Given a training set $S = (\mathbf{x}^{(1)}, \mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n)})$ where each $\mathbf{x}^{(i)} \sim p_{\text{data}}(\mathbf{x})$, the goal is to estimate $p_{\text{data}}(\mathbf{x})$.
  - Applications :
    - generating music, videos, volumetric models for 3D printing, synthetic data for learning algorithms, outlier identification, images denoising, inpainting etc.

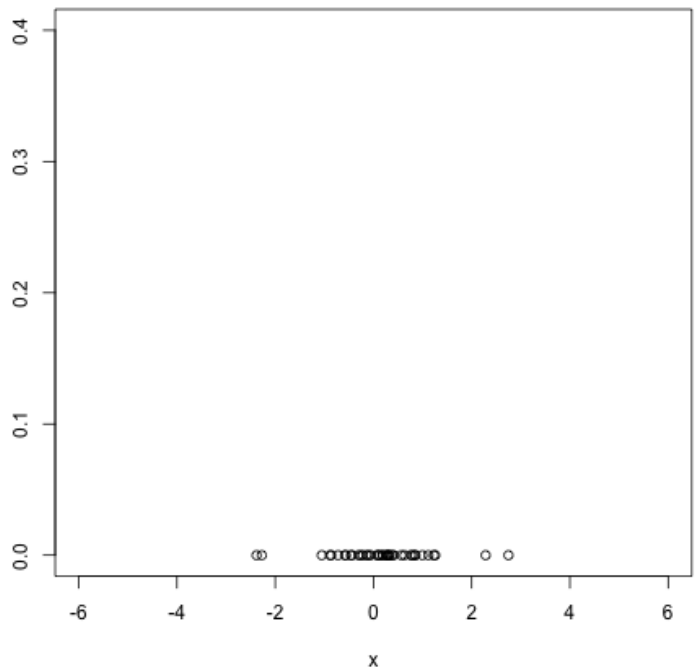
DENSITY FITTING / LEARNING A GENERATIVE MODEL

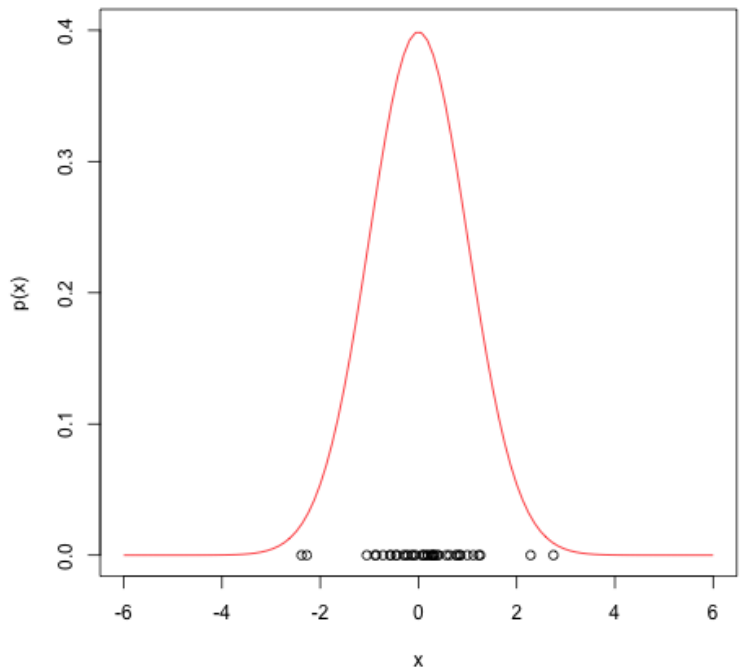
Given $x^{(1)}, x^{(2)}, \ldots, x^{(n)} \sim p_{\text{data}}$ learn a model p_θ of p_{data}.



DENSITY FITTING / LEARNING A GENERATIVE MODEL

Given $x^{(1)}, x^{(2)}, \ldots, x^{(n)} \sim p_{\text{data}}$ learn a model p_θ of p_{data}.



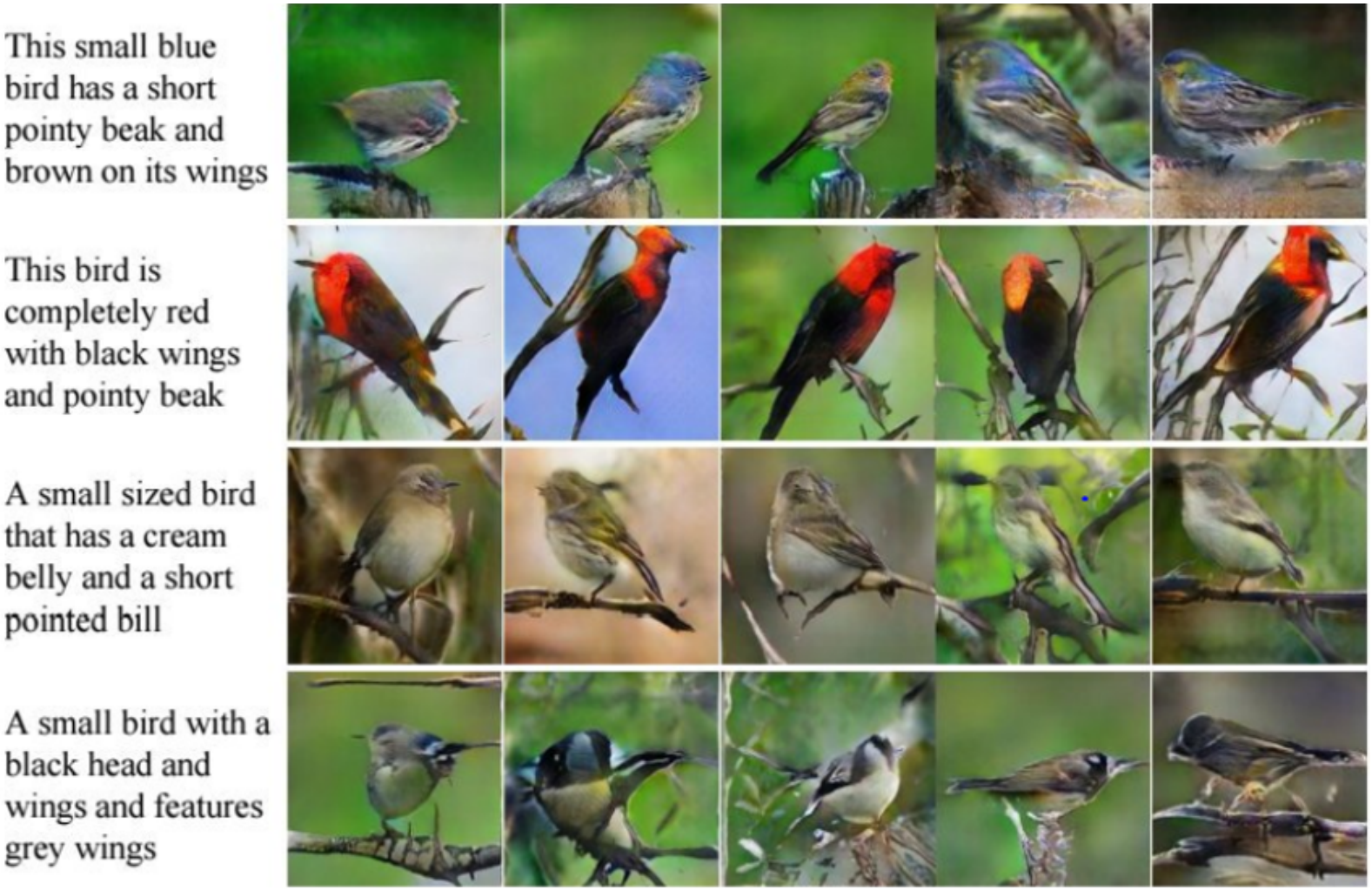


WHY GENERATIVE MODELS?

A generative model p_θ can be used for

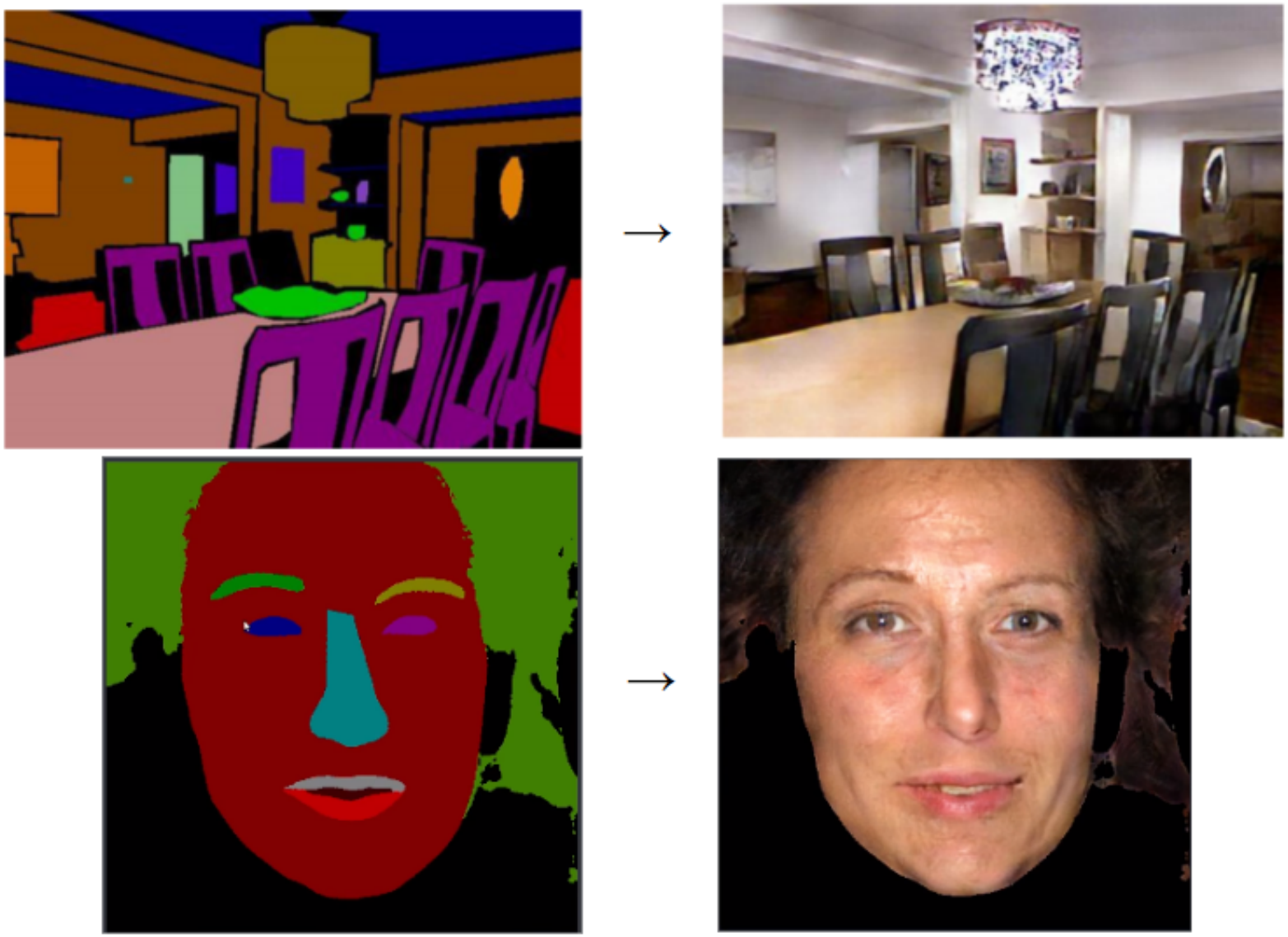
- sampling / data generation
- outlier detection
- missing feature extraction
- image denoising/ reconstruncion
- representation learning
- planing in reinforcement learning
- ...

APPLICATION EXAMPLE: GENERATING IMAGES FROM TEXT



Source : Zhang et al (2017)

APPLICATION EXAMPLE: SEMANTIC LABELS –> IMAGES



Source : Wang et al (2017)

APPLICATION EXAMPLE: IMAGE INPAINTING

Source : Demir et al (2018)

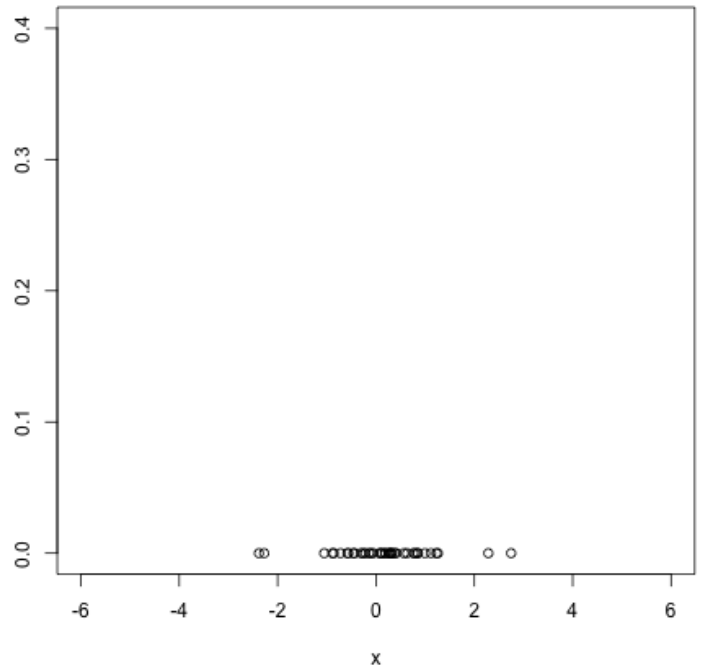
APPLICATION EXAMPLE: IMAGE GENERATION

None of these are real!

Source : Karras et al (2018)

Background

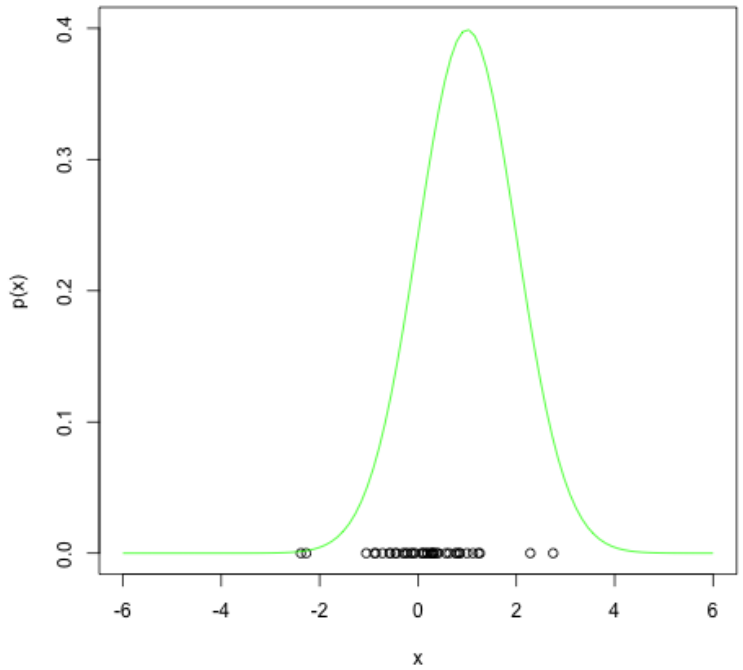
MAXIMUM LIKELIHOOD



We choose the model distribution p_θ to be Gaussian, that is

$$p_\theta(x^{(i)}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2}\frac{(x^{(i)}-\mu)^2}{\sigma^2}\right)$$

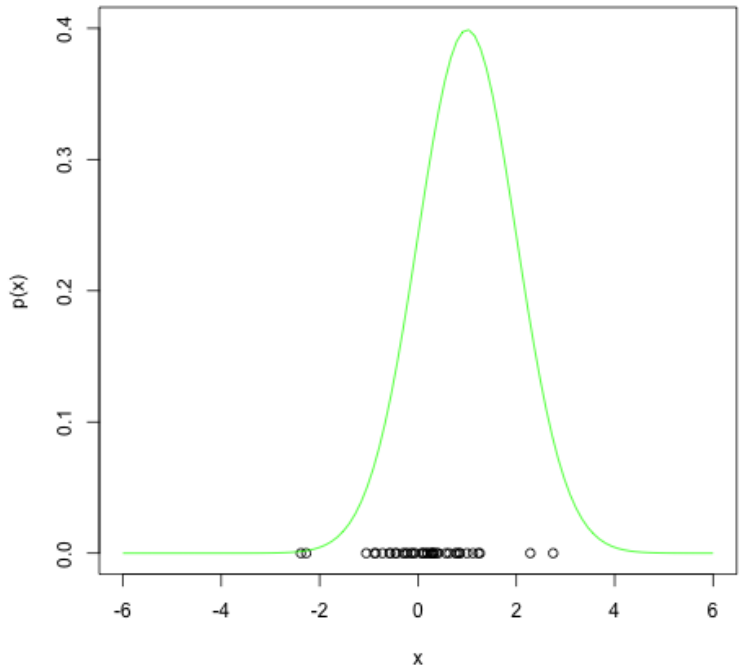
MAXIMUM LIKELIHOOD



We choose the model distribution p_θ to be Gaussian, that is

$$p_\theta(x^{(1)}, \ldots, x^{(n)}) = \prod_{i=1}^{n} p_\theta(x^{(i)}) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2}\frac{(x^{(i)} - \mu)^2}{\sigma^2}\right)$$

MAXIMUM LIKELIHOOD

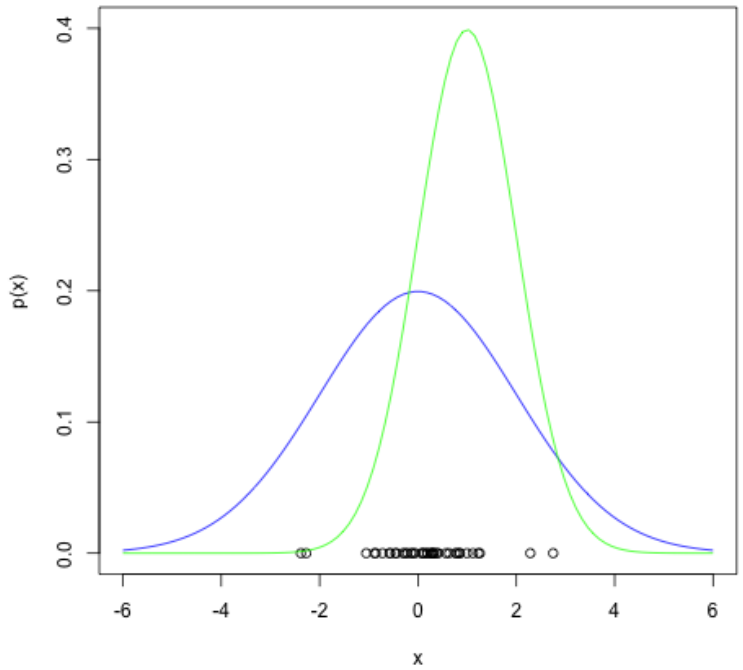


We choose the model distribution p_θ to be Gaussian, that is

$$p_\theta(x^{(1)}, \ldots, x^{(n)}) = \prod_{i=1}^{n} p_\theta(x^{(i)}) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2}\frac{(x^{(i)} - \mu)^2}{\sigma^2}\right)$$

Given the samples $x^{(1)}, \ldots, x^{(n)}$, how should we estimate $\theta = \{\mu, \sigma^2\}$?

MAXIMUM LIKELIHOOD

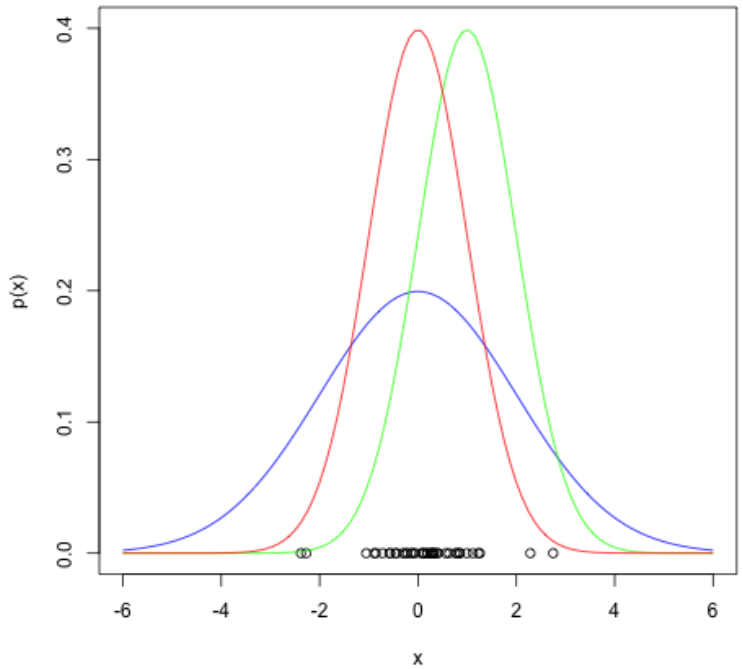


We choose the model distribution p_θ to be Gaussian, that is

$$p_\theta(x^{(1)}, \ldots, x^{(n)}) = \prod_{i=1}^{n} p_\theta(x^{(i)}) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2}\frac{(x^{(i)} - \mu)^2}{\sigma^2}\right)$$

Given the samples $x^{(1)}, \ldots, x^{(n)}$, how should we estimate $\theta = \{\mu, \sigma^2\}$?

MAXIMUM LIKELIHOOD



We choose the model distribution p_θ to be Gaussian, that is

$$p_\theta(x^{(1)}, \ldots, x^{(n)}) = \prod_{i=1}^{n} p_\theta(x^{(i)}) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2}\frac{(x^{(i)} - \mu)^2}{\sigma^2}\right)$$

Given the samples $x^{(1)}, \ldots, x^{(n)}$, how should we estimate $\theta = \{\mu, \sigma^2\}$?

RECALL: MAXIMUM LIKELIHOOD

The **likelihood function** is given by

$$L(\boldsymbol{\theta}|x^{(1)},\ldots,x^{(n)}) = \prod_{i=1}^{n} p_{\boldsymbol{\theta}}(x^{(i)}) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2}\frac{(x^{(i)}-\mu)^2}{\sigma^2}\right) \quad .$$

To maximize it, we often consider the **log-likelihood**

$$\begin{aligned}\log L(\boldsymbol{\theta}|x^{(1)},\ldots,x^{(n)}) &= \log \prod_{i=1}^{n} p_{\boldsymbol{\theta}}(x^{(i)}) = \sum_{i=1}^{n} \log p_{\boldsymbol{\theta}}(x^{(i)}) \\ &= \log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) - \frac{1}{2}\sum_{i=1}^{n}\frac{(x^{(i)}-\mu)^2}{\sigma^2} \quad .\end{aligned}$$

RECALL: MAXIMUM LIKELIHOOD

Setting derivatives equal to zero yields

$$\frac{\partial \log L(\boldsymbol{\theta}|x^{(1)},\ldots,x^{(n)})}{\partial \mu} = \frac{1}{\sigma^2}\Big(\sum_{i=1}^{n} x^{(i)} - n\mu\Big)$$

and

$$\frac{\partial \log L(\boldsymbol{\theta}|x^{(1)},\ldots,x^{(n)})}{\partial \sigma} = \frac{1}{2\sigma^2}\Big(\frac{1}{\sigma^2}\sum_{i=1}^{n}(x^{(i)} - \mu)^2 - n\Big) \quad .$$

Leading to

$$\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} x^{(i)} \quad \text{and} \quad \hat{\sigma} = \frac{1}{n}\sum_{i=1}^{n}(x^{(i)} - \hat{\mu})^2 \quad .$$

NOTES ON MAXIMUM LIKELIHOOD LEARNING

- For a model p with visible variables $\boldsymbol{x}$ and hidden variables $\boldsymbol{z}$, the likelihood computation involves

$$p(\boldsymbol{x}^{(i)} \mid \boldsymbol{\theta}) = \sum_{\boldsymbol{z}} p(\boldsymbol{x}^{(i)}, \boldsymbol{z} \mid \boldsymbol{\theta}) \quad .$$

  This is difficult, especially because of the sum which prevents the logarithm to act directly on the joint distribution.

NOTES ON MAXIMUM LIKELIHOOD LEARNING

- For a model p with visible variables $\boldsymbol{x}$ and hidden variables $\boldsymbol{z}$, the likelihood computation involves

$$p(\boldsymbol{x}^{(i)} \mid \boldsymbol{\theta}) = \sum_{\boldsymbol{z}} p(\boldsymbol{x}^{(i)}, \boldsymbol{z} \mid \boldsymbol{\theta}) \quad .$$

  This is difficult, especially because of the sum which prevents the logarithm to act directly on the joint distribution.

- If we can not find the maximum likelihood parameters analytically (i.e. by setting the derivative to zero) one can maximize the likelihood via SGD or related algorithms.

- If p_{data} is the true distribution underlying S, maximizing the logarithmic likelihood function corresponds to minimizing an empirical estimate of the Kullback-Leibler divergence $\text{KL}(p_{\text{data}} \,\|\, p)$.

KULLBACK-LEIBLER DIVERGENCE

Kullback-Leibler (KL) divergence between two distribution p and p_{data} over $\boldsymbol{x}$ is

- a (non-symmetric) measure of difference between p and p_{data},
- always positive, zero iff the distributions are the same,

KULLBACK-LEIBLER DIVERGENCE

Kullback-Leibler (KL) divergence between two distribution p and p_{data} over $\boldsymbol{x}$ is

- a (non-symmetric) measure of difference between p and p_{data},
- always positive, zero iff the distributions are the same,

and defined as

$$\begin{aligned}\text{KL}(p_{\text{data}} \,\|\, p) &= -\sum_{\boldsymbol{x}} p_{\text{data}}(\boldsymbol{x}) \ln \frac{p(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x})} \\ &= \underbrace{-\sum_{\boldsymbol{x}} p_{\text{data}}(\boldsymbol{x}) \ln p(\boldsymbol{x})}_{\substack{\text{can be approximated} \\ \text{by } \frac{1}{\ell}\sum_{n=1}^{\ell} \ln p(\boldsymbol{x}_n)}} + \underbrace{\sum_{\boldsymbol{x}} p_{\text{data}}(\boldsymbol{x}) \ln p_{\text{data}}(\boldsymbol{x})}_{\text{independent of } p}\end{aligned}$$

(sum turns to integral for continuous random variables).

Probabilistic graphical models

GRAPHICAL MODELS

Probabilistic graphical models describe probability distributions by mapping conditional dependence and independence properties between random variables on a graph structure.

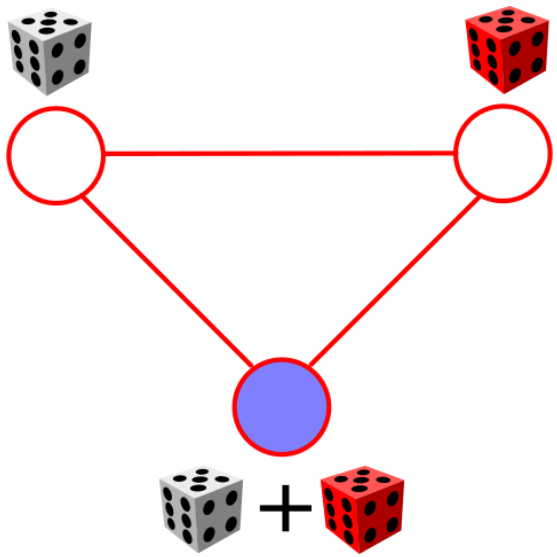
WHY AGAIN GRAPHICAL MODELS?

1. Graphical models visualize the structure of a probabilistic model; they help to develop, understand and motivate probabilistic models.

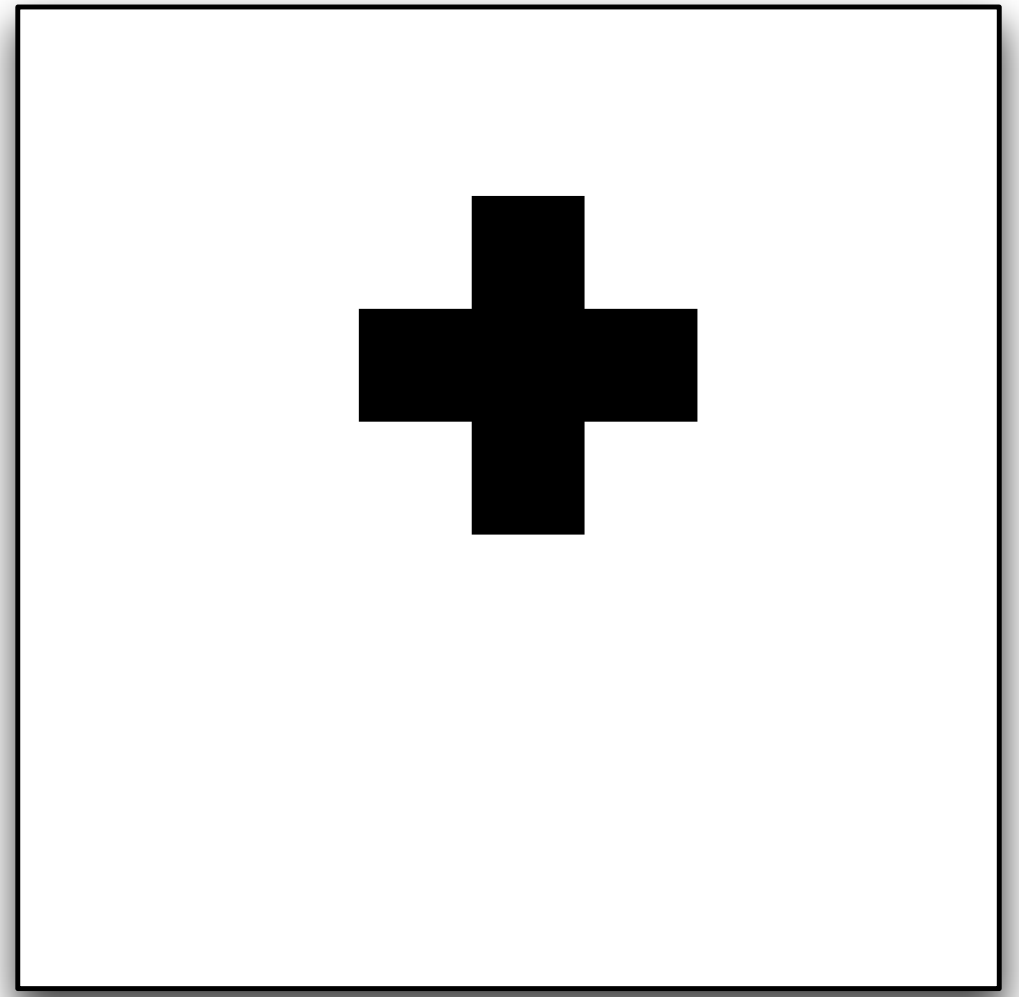
WHY AGAIN GRAPHICAL MODELS?

1. Graphical models visualize the structure of a probabilistic model; they help to develop, understand and motivate probabilistic models.

2. Complex computations (e.g., marginalization) can derived efficiently using algorithms exploiting the graph structure.

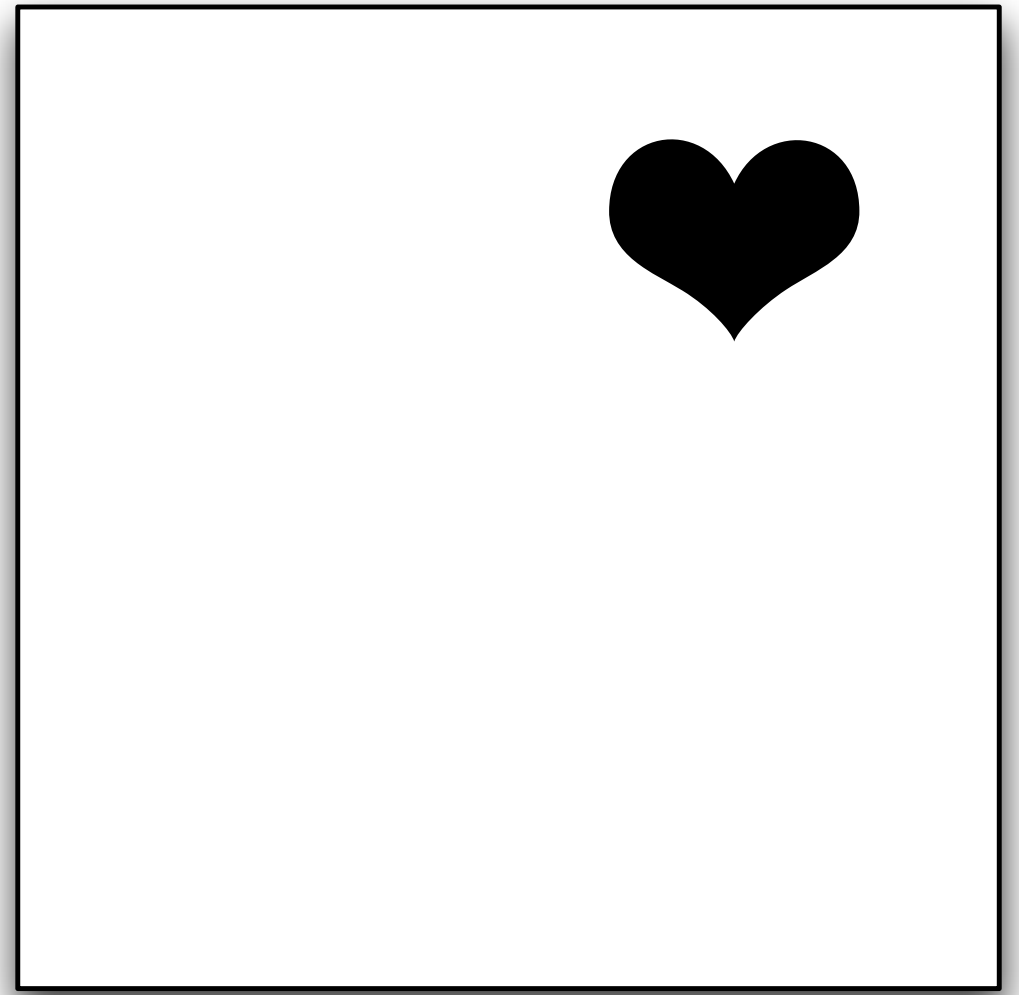


LATENT VARIABLES: MOTIVATION



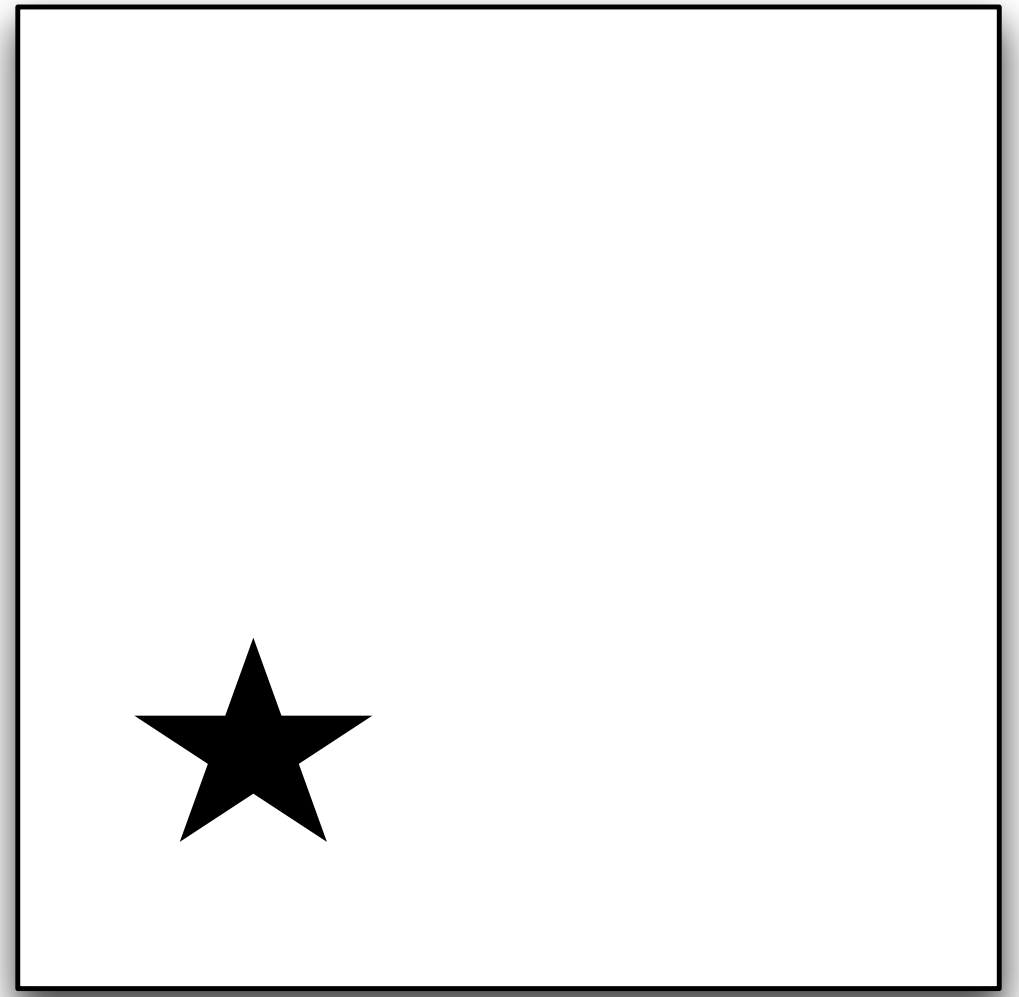
200×200 pixels $\quad \rightarrow \quad 2^{40000} - 1$ parameters?

LATENT VARIABLES: MOTIVATION



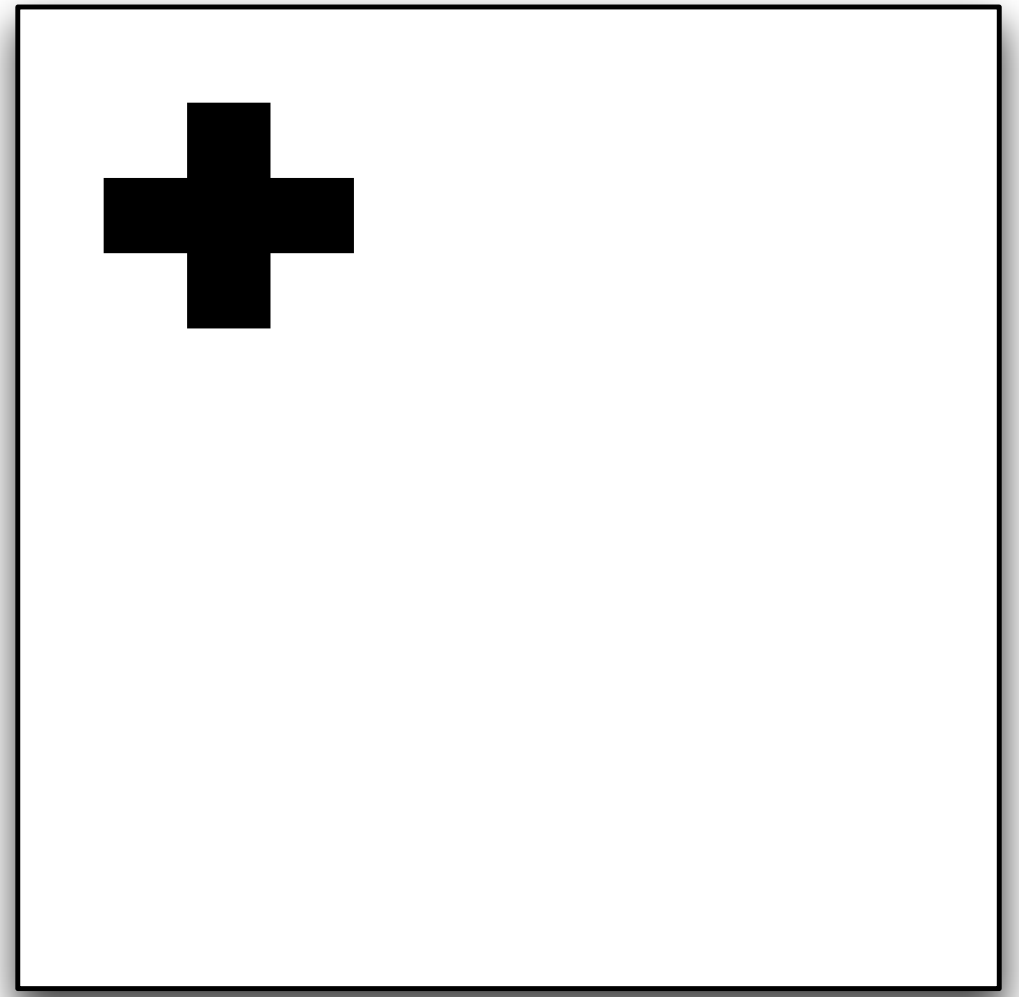
200×200 pixels $\quad \rightarrow \quad 2^{40000} - 1$ parameters?

LATENT VARIABLES: MOTIVATION



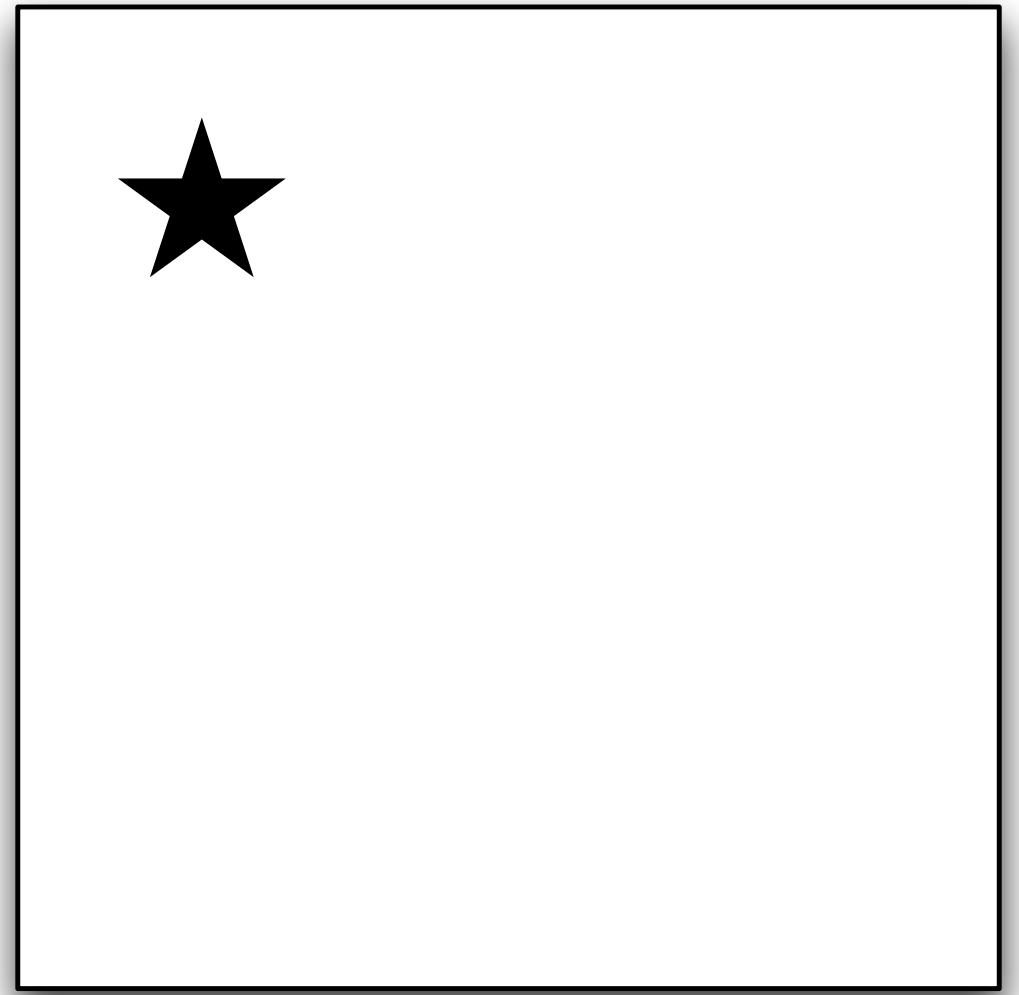
200×200 pixels $\quad \rightarrow \quad 2^{40000} - 1$ parameters?

LATENT VARIABLES: MOTIVATION



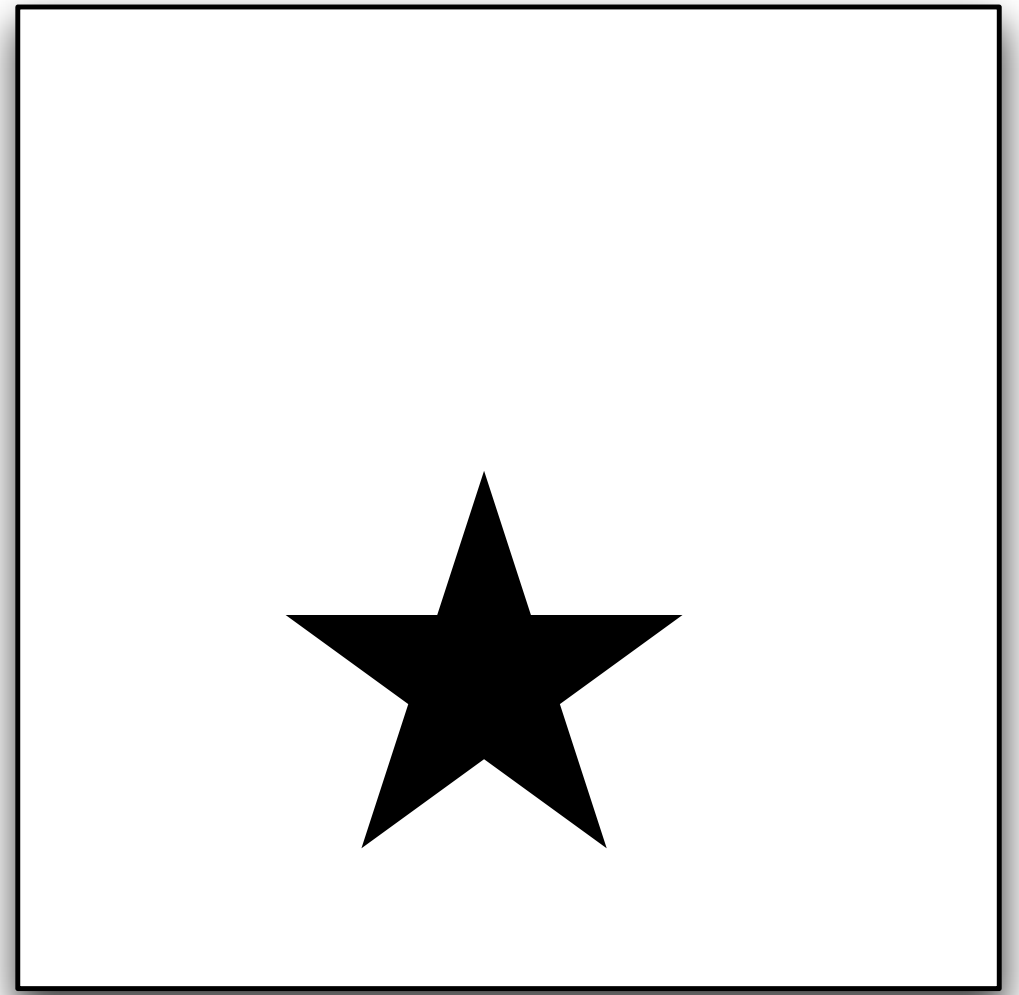
200×200 pixels $\quad \rightarrow \quad 2^{40000} - 1$ parameters?

LATENT VARIABLES: MOTIVATION



200×200 pixels $\quad \rightarrow \quad 2^{40000} - 1$ parameters?

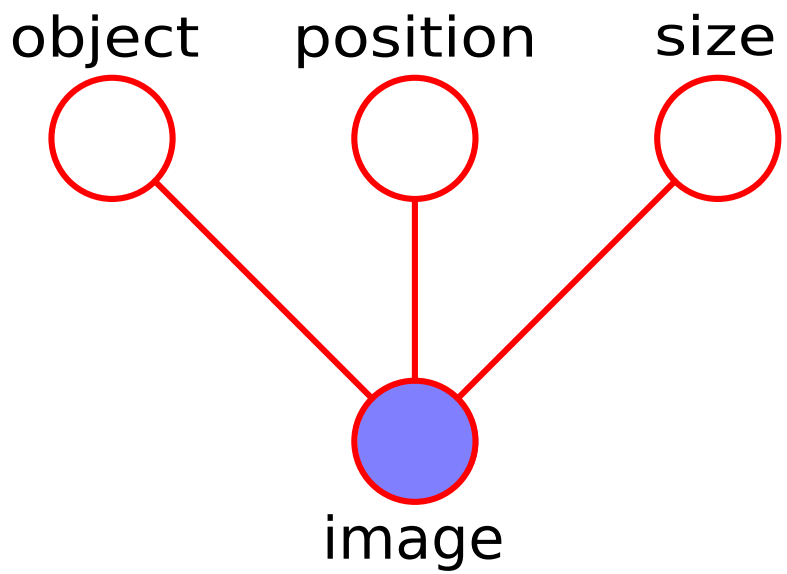
LATENT VARIABLES: MOTIVATION



200×200 pixels $\quad \rightarrow \quad 2^{40000} - 1$ parameters?

LATENT VARIABLES

- Additional nodes, which do not directly correspond to observations, allow to describe complex distributions over the visible variables by means of simple conditional distributions.
- The corresponding random variables are called *hidden* or *latent* variables.

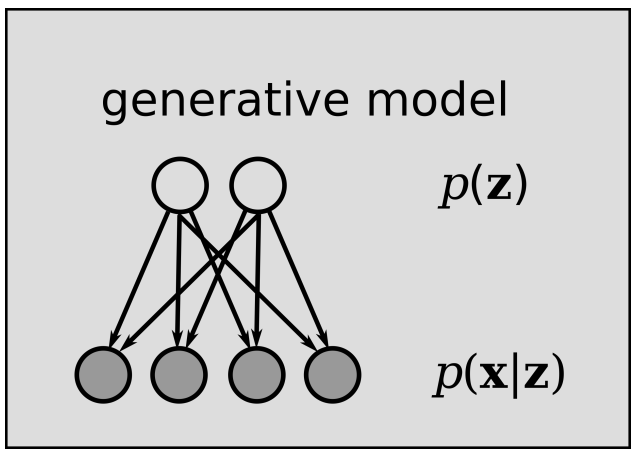


Directed generative models

DIRECTED GENERATIVE MODELS

Learn to generate $\boldsymbol{x}$ from some latent variables $\boldsymbol{z}$.

$$p_\theta(\boldsymbol{x}) = \int p_\theta(\boldsymbol{x}, \boldsymbol{z}) d\boldsymbol{z} = \int p_\theta(\boldsymbol{x}|\boldsymbol{z}) p_\theta(\boldsymbol{z}) d\boldsymbol{z}$$



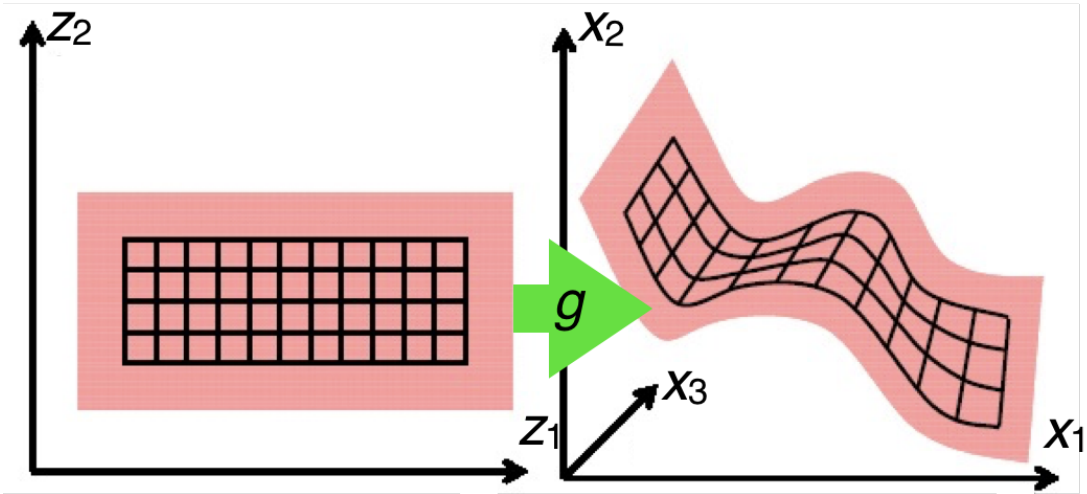


Image from: Ward, A. D., Hamarneh, G.: **3D Surface Parameterization Using Manifold Learning for Medial Shape Representation**, Conference on Image Processing, Proc. of SPIE Medical Imaging, 2007

DIRECTED GENERATIVE MODELS

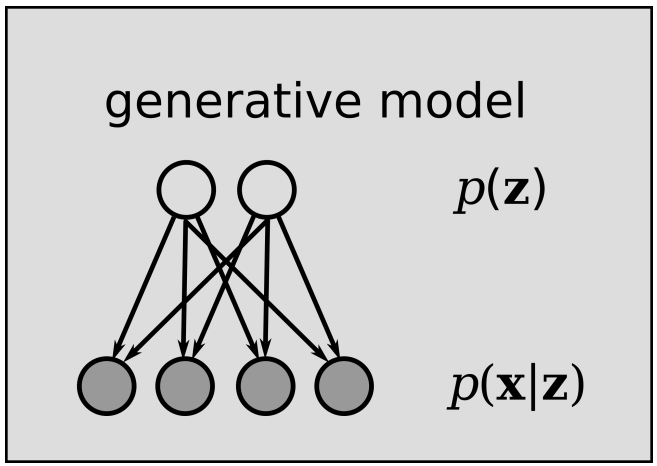
The classic DAG problem: Where does $\boldsymbol{z}$ come from?

- The posterior is given by $p_{\theta}(\boldsymbol{z}|\boldsymbol{x}) = \frac{p_{\theta}(\boldsymbol{x}|\boldsymbol{z})p_{\theta}(\boldsymbol{z})}{p_{\theta}(\boldsymbol{x})}$.

DIRECTED GENERATIVE MODELS

The classic DAG problem: Where does $\boldsymbol{z}$ come from?

- The posterior is given by $p_\theta(\boldsymbol{z}|\boldsymbol{x}) = \frac{p_\theta(\boldsymbol{x}|\boldsymbol{z})p_\theta(\boldsymbol{z})}{p_\theta(\boldsymbol{x})}$.
- But $p_\theta(\boldsymbol{x}) = \int p_\theta(\boldsymbol{x}|\boldsymbol{z})p_\theta(\boldsymbol{z})d\boldsymbol{z}$ is intractable.



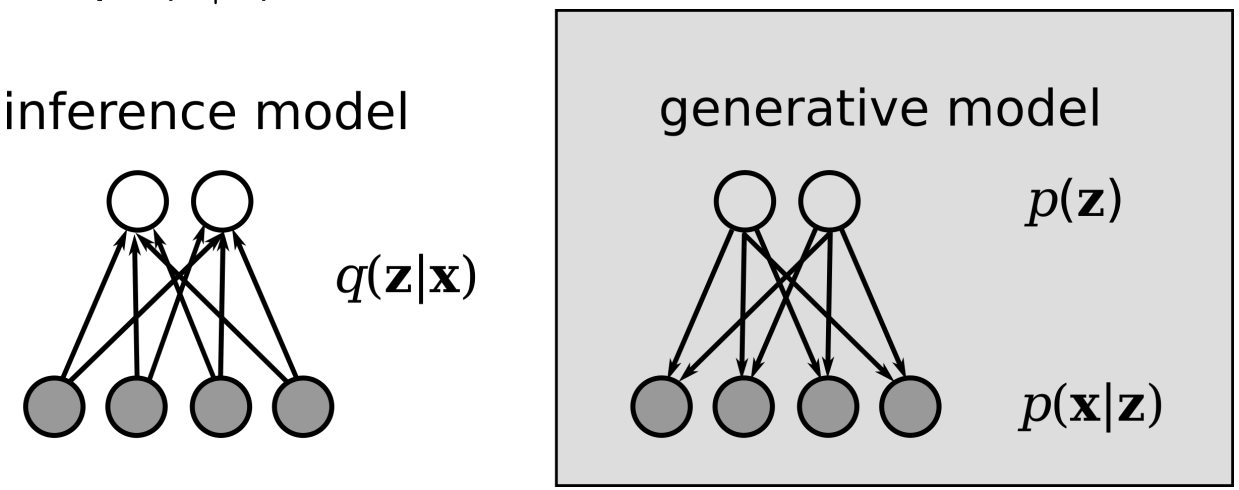
We will see two approaches to this problems:

- **Variational Autoencoders (VAEs)**
- **Generative Adversarial Networks (GANs)**

Variational Autoencoder (VAE)

VARIATIONAL AUTOENCODER (VAE)

The classic DAG problem: Where does $\boldsymbol{z}$ come from?
Idea: introduce an **inference model** $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ that learns to approximate the posterior $p_\theta(\boldsymbol{z}|\boldsymbol{x})$



Independently proposed by:

- Kingma and Welling, *Auto-Encoding Variational Bayes*, ICLR 2014
- Rezende, Mohamed and Wierstra, *Stochastic back-propagation and variational inference in deep latent Gaussian models.* ICML 2014

VAE-PARAMETER FITTING: VARIATIONAL LOWER BOUND

- $\log p(\boldsymbol{x})$ is intractable.
- But we can compute a **variational lower bound**

$$\log p_\theta(\boldsymbol{x}) = \log \int p_\theta(\boldsymbol{x}, \boldsymbol{z}) d\boldsymbol{z} = \log \int q_\phi(\boldsymbol{z}|\boldsymbol{x}) \frac{p_\theta(\boldsymbol{x}, \boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})} d\boldsymbol{z}$$

VAE-PARAMETER FITTING: VARIATIONAL LOWER BOUND

- $\log p(\boldsymbol{x})$ is intractable.
- But we can compute a **variational lower bound**

$$\log p_\theta(\boldsymbol{x}) = \log \int p_\theta(\boldsymbol{x}, \boldsymbol{z}) d\boldsymbol{z} = \log \int q_\phi(\boldsymbol{z}|\boldsymbol{x}) \frac{p_\theta(\boldsymbol{x}, \boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})} d\boldsymbol{z}$$

$$\geq \int q_\phi(\boldsymbol{z}|\boldsymbol{x}) \log \frac{p_\theta(\boldsymbol{x}, \boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})} d\boldsymbol{z}$$

Jensen's inequality

Let f be a concave function and $\mathbf{x}$ an integrable random variable. Then it holds: $f(\mathbb{E}[\mathbf{x}]) \geq \mathbb{E}[f(\mathbf{x})]$.

VAE-PARAMETER FITTING: VARIATIONAL LOWER BOUND

- $\log p(\boldsymbol{x})$ is intractable.
- But we can compute a **variational lower bound**

$$\begin{aligned}
\log p_\theta(\boldsymbol{x}) &= \log \int p_\theta(\boldsymbol{x}, \boldsymbol{z}) d\boldsymbol{z} = \log \int q_\phi(\boldsymbol{z}|\boldsymbol{x}) \frac{p_\theta(\boldsymbol{x}, \boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})} d\boldsymbol{z} \\
&\geq \int q_\phi(\boldsymbol{z}|\boldsymbol{x}) \log \frac{p_\theta(\boldsymbol{x}, \boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})} d\boldsymbol{z} \\
&= \int q_\phi(\boldsymbol{z}|\boldsymbol{x}) \big(\log p_\theta(\boldsymbol{x}|\boldsymbol{z}) + \log \frac{p_\theta(\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})} \big) d\boldsymbol{z}
\end{aligned}$$

VAE-PARAMETER FITTING: VARIATIONAL LOWER BOUND

- $\log p(\boldsymbol{x})$ is intractable.
- But we can compute a **variational lower bound**

$$
\begin{aligned}
\log p_\theta(\boldsymbol{x}) &= \log \int p_\theta(\boldsymbol{x}, \boldsymbol{z}) d\boldsymbol{z} = \log \int q_\phi(\boldsymbol{z}|\boldsymbol{x}) \frac{p_\theta(\boldsymbol{x}, \boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})} d\boldsymbol{z} \\
&\geq \int q_\phi(\boldsymbol{z}|\boldsymbol{x}) \log \frac{p_\theta(\boldsymbol{x}, \boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})} d\boldsymbol{z} \\
&= \int q_\phi(\boldsymbol{z}|\boldsymbol{x}) \big(\log p_\theta(\boldsymbol{x}|\boldsymbol{z}) + \log \frac{p_\theta(\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})} \big) d\boldsymbol{z} \\
&= \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})} \big[\log p_\theta((\boldsymbol{x}|\boldsymbol{z}) \big] - KL\big[q_\phi(\boldsymbol{z}|\boldsymbol{x}) || p_\theta(\boldsymbol{z}) \big]
\end{aligned}
$$

VAE-PARAMETER FITTING: VARIATIONAL LOWER BOUND

- $\log p(\boldsymbol{x})$ is intractable.
- But we can compute a **variational lower bound**

$$
\begin{aligned}
\log p_\theta(\boldsymbol{x}) &= \log \int p_\theta(\boldsymbol{x}, \boldsymbol{z}) d\boldsymbol{z} = \log \int q_\phi(\boldsymbol{z}|\boldsymbol{x}) \frac{p_\theta(\boldsymbol{x}, \boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})} d\boldsymbol{z} \\
&\geq \int q_\phi(\boldsymbol{z}|\boldsymbol{x}) \log \frac{p_\theta(\boldsymbol{x}, \boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x}}) d\boldsymbol{z} \\
&= \int q_\phi(\boldsymbol{z}|\boldsymbol{x}) \big(\log p_\theta(\boldsymbol{x}|\boldsymbol{z}) + \log \frac{p_\theta(\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})} \big) d\boldsymbol{z} \\
&= \underbrace{\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})} \big[\log p_\theta(\boldsymbol{x}|\boldsymbol{z}) \big] - KL\big[q_\phi((\boldsymbol{z}|\boldsymbol{x}) || p_\theta(\boldsymbol{z}) \big]}_{:= ELBO(\boldsymbol{\theta}, \phi, \boldsymbol{x})}
\end{aligned}
$$

VAE-PARAMETER FITTING: VARIATIONAL LOWER BOUND

$$ELBO(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\big[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\big] - KL\big[q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})||p_{\boldsymbol{\theta}}(\boldsymbol{z})\big]$$

VAE-PARAMETER FITTING: VARIATIONAL LOWER BOUND

$$ELBO(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\big[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\big] - KL\big[q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})||p_{\boldsymbol{\theta}}(\boldsymbol{z})\big]$$

- Also known as **Evidence Lower BOund (ELBO)**.

VAE-PARAMETER FITTING: VARIATIONAL LOWER BOUND

$$ELBO(\theta, \phi, \boldsymbol{x}) = \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})\right] - KL\left[q_\phi(\boldsymbol{z}|\boldsymbol{x})||p_\theta(\boldsymbol{z})\right]$$

- Also known as **Evidence Lower BOund (ELBO)**.
- First term resembles reconstruction loss.

VAE-PARAMETER FITTING: VARIATIONAL LOWER BOUND

$$ELBO(\theta, \phi, \boldsymbol{x}) = \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\big[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})\big] - KL\big[q_\phi(\boldsymbol{z}|\boldsymbol{x})||p_\theta(\boldsymbol{z})\big]$$

- Also known as **Evidence Lower BOund (ELBO)**.
- First term resembles reconstruction loss.
- Second term penalizes encoder for deviating from prior.

VAE-PARAMETER FITTING: VARIATIONAL LOWER BOUND

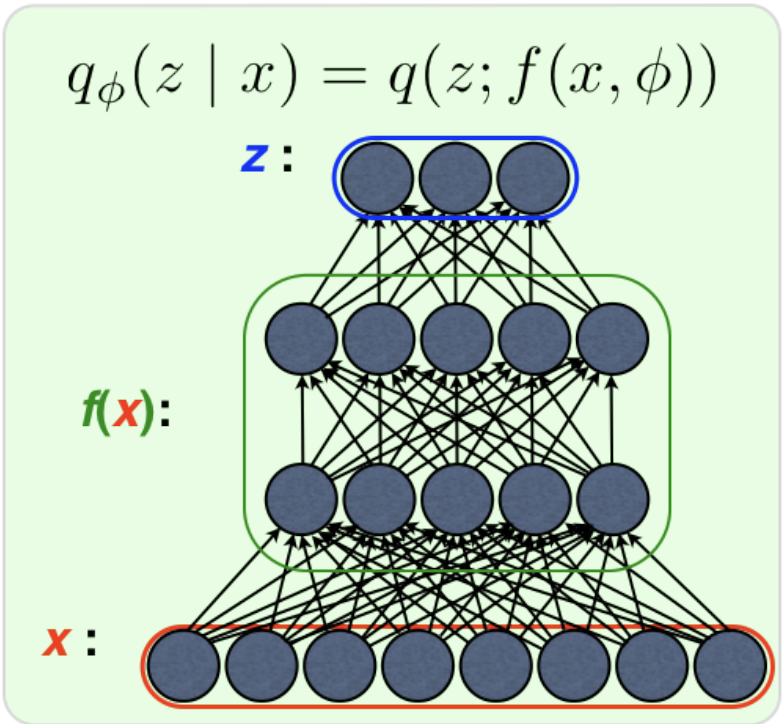
$$ELBO(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\big[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\big] - KL\big[q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})||p_{\boldsymbol{\theta}}(\boldsymbol{z})\big]$$

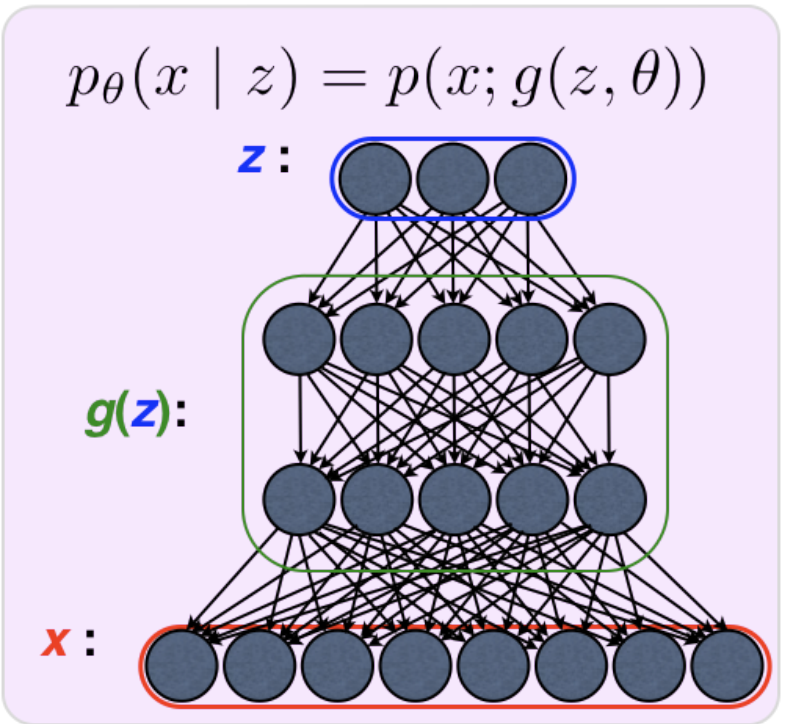
- Also known as **Evidence Lower BOund (ELBO)**.
- First term resembles reconstruction loss.
- Second term penalizes encoder for deviating from prior.
- It holds $ELBO(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{x}) = \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) - KL\big[q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})||p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})\big]$
  $\Rightarrow$ by maximizing the ELBO we maximize $p_{\boldsymbol{\theta}}(\boldsymbol{x})$ and minimize $KL\big[q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})||p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})\big]$.

VAE-MODEL DEFINITION

Idea:

- Set $p_\theta(\boldsymbol{z})$ to some simple distribution.
- Parametrize inference model and generative model with neural networks $f(\boldsymbol{x}, \phi)$ and $g(\boldsymbol{z}, \boldsymbol{\theta})$.

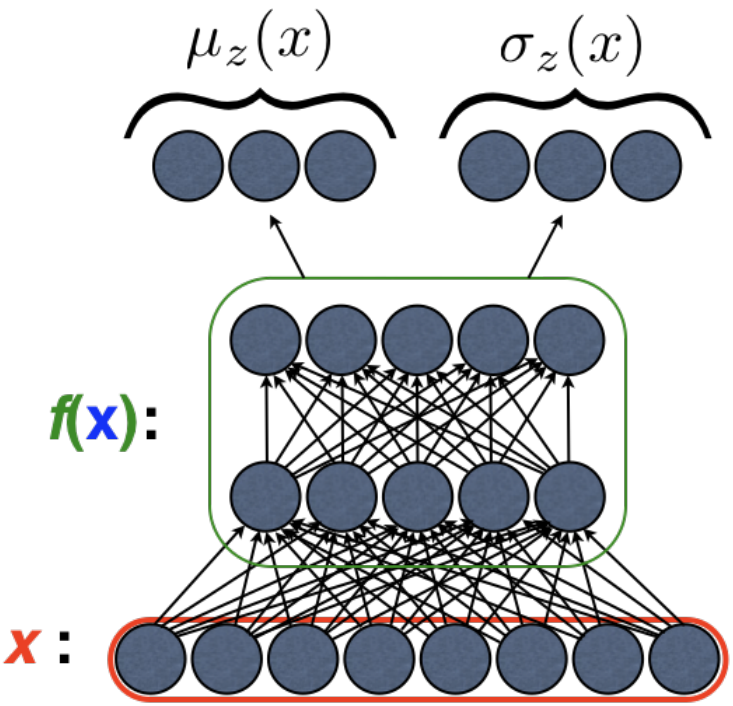




VAE-MODEL DEFINITION

Usually:

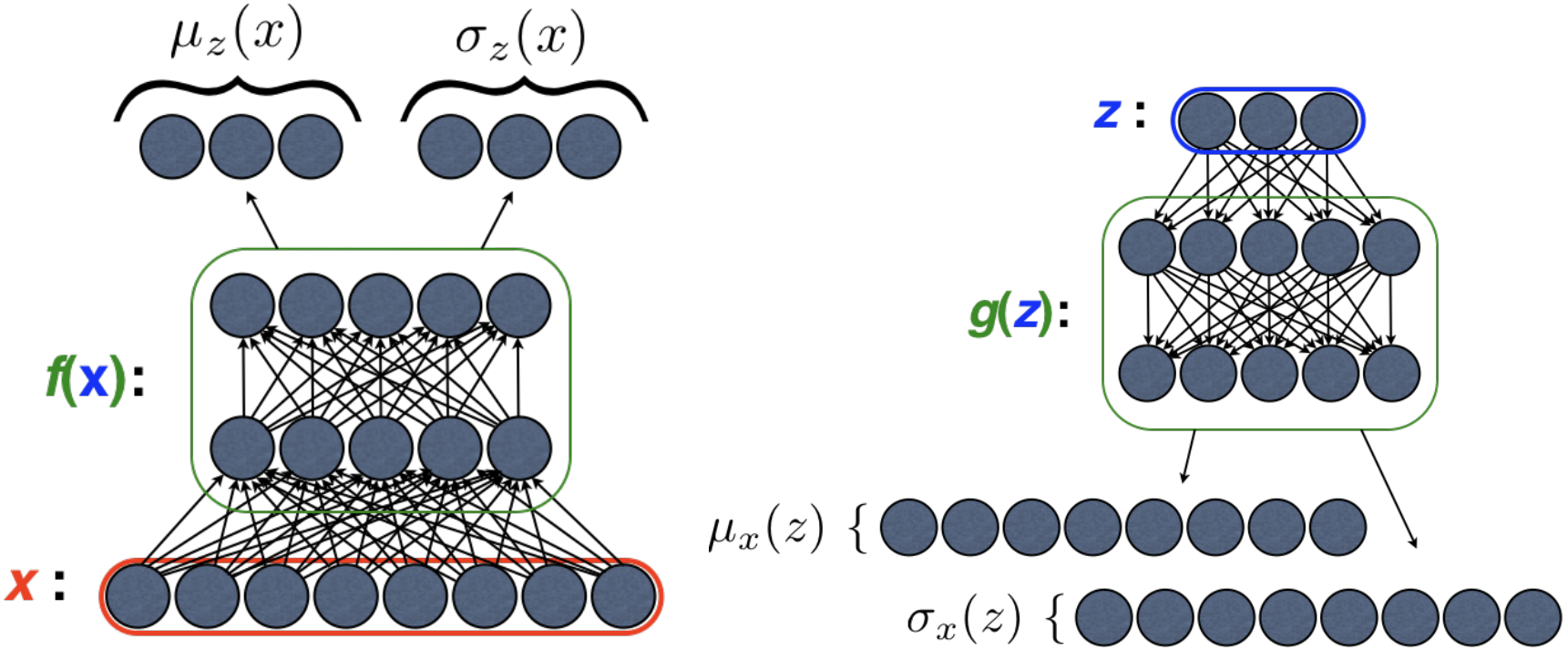
- $f(\boldsymbol{x}, \phi) = (\mu_{\boldsymbol{x}}(\boldsymbol{x}), \sigma_{\boldsymbol{z}}(\boldsymbol{x}))$ and $q_\phi(\boldsymbol{z}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}_{\boldsymbol{z}}(\boldsymbol{x}), \boldsymbol{\sigma}^2_{\boldsymbol{z}}(\boldsymbol{x}))$



VAE-MODEL DEFINITION

Usually:

- $f(\boldsymbol{x}, \phi) = (\mu_{\boldsymbol{z}}(\boldsymbol{x}), \sigma_{\boldsymbol{z}}(\boldsymbol{x}))$ and $q_\phi(\boldsymbol{z}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}_{\boldsymbol{z}}(\boldsymbol{x}), \boldsymbol{\sigma}^2_{\boldsymbol{z}}(\boldsymbol{x}))$
- $p_{\boldsymbol{\theta}}(\boldsymbol{z}, \boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{z}; 0, 1)$
- $g(\boldsymbol{z}) = (\mu_{\boldsymbol{x}}(\boldsymbol{z}), \sigma_{\boldsymbol{x}}(\boldsymbol{z}))$ and $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) = \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_{\boldsymbol{x}}(\boldsymbol{z}), \boldsymbol{\sigma}^2_{\boldsymbol{x}}(\boldsymbol{z}))$



VAE-PARAMETER FITTING: REPARAMETRIZATION TRICK

- Goal: Learn parameters ϕ and θ by maximizing

$$ELBO(\theta, \phi, \boldsymbol{x}) = \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\big[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})\big] - KL\big[q_\phi(\boldsymbol{z}|\boldsymbol{x})||p_\theta(\boldsymbol{h})\big]$$

  based on gradient ascent.

- Idea: Approximate first term

$$\begin{aligned}\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\big[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})\big] &= \mathbb{E}_{\boldsymbol{z}\sim\mathcal{N}(\boldsymbol{z};\boldsymbol{\mu}_{\boldsymbol{z}}(\boldsymbol{x}),\boldsymbol{\sigma}_{\boldsymbol{z}}(\boldsymbol{x}))}\big[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})\big] \\ &\approx \frac{1}{L}\sum_{l=1}^{L}\log p_\theta(\boldsymbol{x}|\boldsymbol{z}_l)\end{aligned}$$

- Problem: Given this average, how should one take derivatives w.r.t. ϕ?

VAE-PARAMETER FITTING: REPARAMETRIZATION TRICK

Recall: Linear transformation of a normal random variable

Let ϵ be standard normally distributed, i.e. $\epsilon \sim \mathcal{N}(\epsilon; \mathbf{0}, \mathbf{1})$. Then for $\boldsymbol{z} = \epsilon \cdot \boldsymbol{\sigma} + \boldsymbol{\mu}$ it holds: $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2)$

VAE-PARAMETER FITTING: REPARAMETRIZATION TRICK

Recall: Linear transformation of a normal random variable

Let ϵ be standard normally distributed, i.e. $\epsilon \sim \mathcal{N}(\epsilon; \mathbf{0}, \mathbf{1})$. Then for $\boldsymbol{z} = \epsilon \cdot \boldsymbol{\sigma} + \boldsymbol{\mu}$ it holds: $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2)$

Back to our problem:

$$\begin{aligned}\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})\right] &= \mathbb{E}_{\boldsymbol{z}\sim\mathcal{N}(\boldsymbol{z};\boldsymbol{\mu}_{\boldsymbol{z}}(\boldsymbol{x}),\boldsymbol{\sigma}_{\boldsymbol{z}}(\boldsymbol{x}))}\left[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})\right] \\ &\approx \frac{1}{L}\sum_{l=1}^{L} \log p_\theta(\boldsymbol{x}|\boldsymbol{z}_l)\end{aligned}$$

Solution: Define $\boldsymbol{z} = \epsilon \cdot \boldsymbol{\sigma}_{\boldsymbol{z}}(\boldsymbol{x}) + \boldsymbol{\mu}_{\boldsymbol{z}}(\boldsymbol{x})$ where $\epsilon \sim \mathcal{N}(\epsilon; \mathbf{0}, \mathbf{1})$.

VAE-PARAMETER FITTING: REPARAMETRIZATION TRICK

Recall: Linear transformation of a normal random variable

Let ϵ be standard normally distributed, i.e. $\epsilon \sim \mathcal{N}(\epsilon; \mathbf{0}, \mathbf{1})$. Then for $\boldsymbol{z} = \epsilon \cdot \boldsymbol{\sigma} + \boldsymbol{\mu}$ it holds: $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2)$

Back to our problem:

$$\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\big[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})\big] = \mathbb{E}_{\epsilon\sim\mathcal{N}(\epsilon;\mathbf{0},\mathbf{1})}\big[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})\big]$$

$$\approx \frac{1}{L}\sum_{l=1}^{L} \log p_\theta(\boldsymbol{x}|\boldsymbol{z}_l)$$

Solution: Define $\boldsymbol{z} = \epsilon \cdot \boldsymbol{\sigma}_{\boldsymbol{z}}(\boldsymbol{x}) + \boldsymbol{\mu}_{\boldsymbol{z}}(\boldsymbol{x})$ where $\epsilon \sim \mathcal{N}(\epsilon; \mathbf{0}, \mathbf{1})$.

VAE-PARAMETER FITTING: REPARAMETRIZATION TRICK

Recall: Linear transformation of a normal random variable

Let ϵ be standard normally distributed, i.e. $\epsilon \sim \mathcal{N}(\epsilon; \mathbf{0}, \mathbf{1})$. Then for $\boldsymbol{z} = \epsilon \cdot \boldsymbol{\sigma} + \boldsymbol{\mu}$ it holds: $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2)$

Back to our problem:

$$\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\big[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})\big] = \mathbb{E}_{\epsilon\sim\mathcal{N}(\epsilon;\mathbf{0},\mathbf{1})}\big[\log p_\theta(\boldsymbol{x}|\boldsymbol{z} = \epsilon \cdot \boldsymbol{\sigma}_{\boldsymbol{z}}(\boldsymbol{x}) + \boldsymbol{\mu}_{\boldsymbol{z}}(\boldsymbol{x}))\big]$$

$$\approx \frac{1}{L}\sum_{l=1}^{L} \log p_\theta(\boldsymbol{x}|\boldsymbol{z}_l)$$

Solution: Define $\boldsymbol{z} = \epsilon \cdot \boldsymbol{\sigma}_{\boldsymbol{z}}(\boldsymbol{x}) + \boldsymbol{\mu}_{\boldsymbol{z}}(\boldsymbol{x})$ where $\epsilon \sim \mathcal{N}(\epsilon; \mathbf{0}, \mathbf{1})$.

VAE-PARAMETER FITTING: REPARAMETRIZATION TRICK

Recall: Linear transformation of a normal random variable

Let ϵ be standard normally distributed, i.e. $\epsilon \sim \mathcal{N}(\epsilon; \mathbf{0}, \mathbf{1})$. Then for $\boldsymbol{z} = \epsilon \cdot \boldsymbol{\sigma} + \boldsymbol{\mu}$ it holds: $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2)$

Back to our problem:

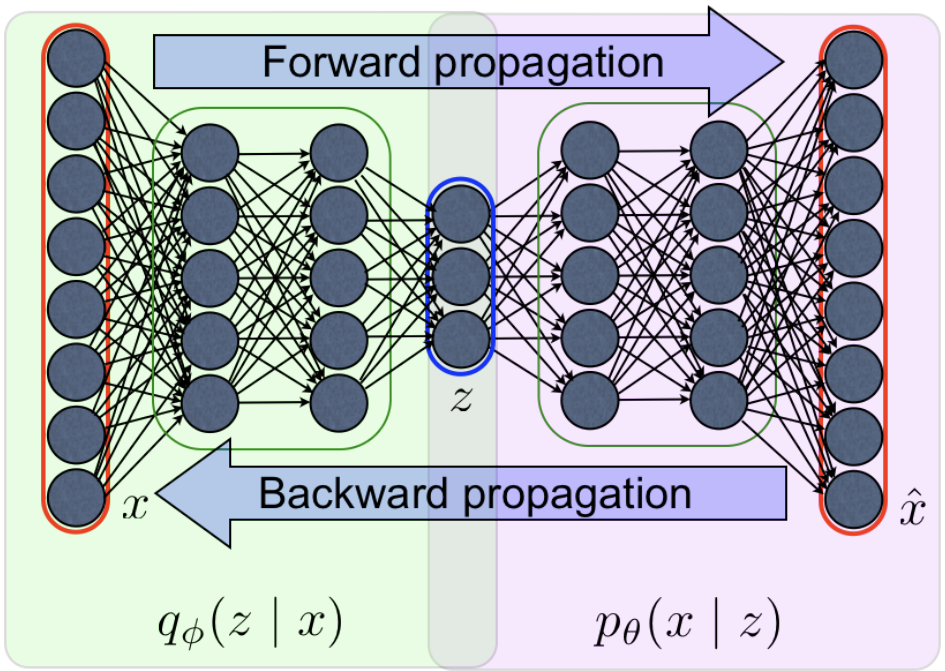
$$\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\big[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})\big] = \mathbb{E}_{\epsilon\sim\mathcal{N}(\epsilon;\mathbf{0},\mathbf{1})}\big[\log p_\theta(\boldsymbol{x}|\boldsymbol{z} = \epsilon \cdot \boldsymbol{\sigma}_{\boldsymbol{z}}(\boldsymbol{x}) + \boldsymbol{\mu}_{\boldsymbol{z}}(\boldsymbol{x}))\big]$$

$$\approx \frac{1}{L}\sum_{l=1}^{L} \log p_\theta(\boldsymbol{x}|\boldsymbol{z}_l = \epsilon_l \cdot \boldsymbol{\sigma}_{\boldsymbol{z}}(\boldsymbol{x}) + \boldsymbol{\mu}_{\boldsymbol{z}}(\boldsymbol{x}))$$

Solution: Define $\boldsymbol{z} = \epsilon \cdot \boldsymbol{\sigma}_{\boldsymbol{z}}(\boldsymbol{x}) + \boldsymbol{\mu}_{\boldsymbol{z}}(\boldsymbol{x})$ where $\epsilon \sim \mathcal{N}(\epsilon; \mathbf{0}, \mathbf{1})$.

VAE-TRAINING WITH BACKPROPAGATION

Due to the reparametrization trick, we can simultaneously train both the generative model $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ and the inference model $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ by maximizing the ELBO based on gradient asscent and backpropagation.



LATENT VARIABLES LEARNED BY A VAE

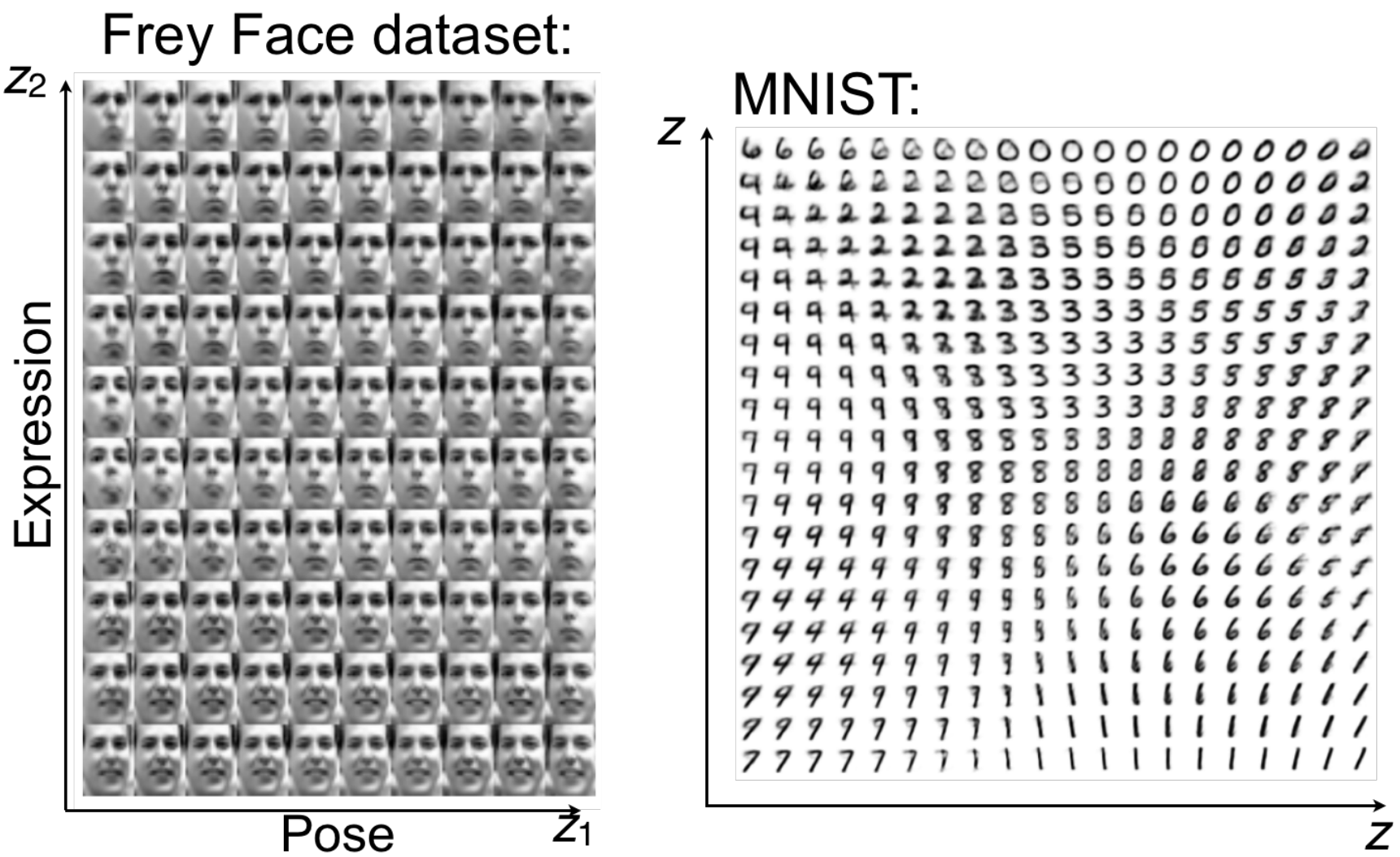
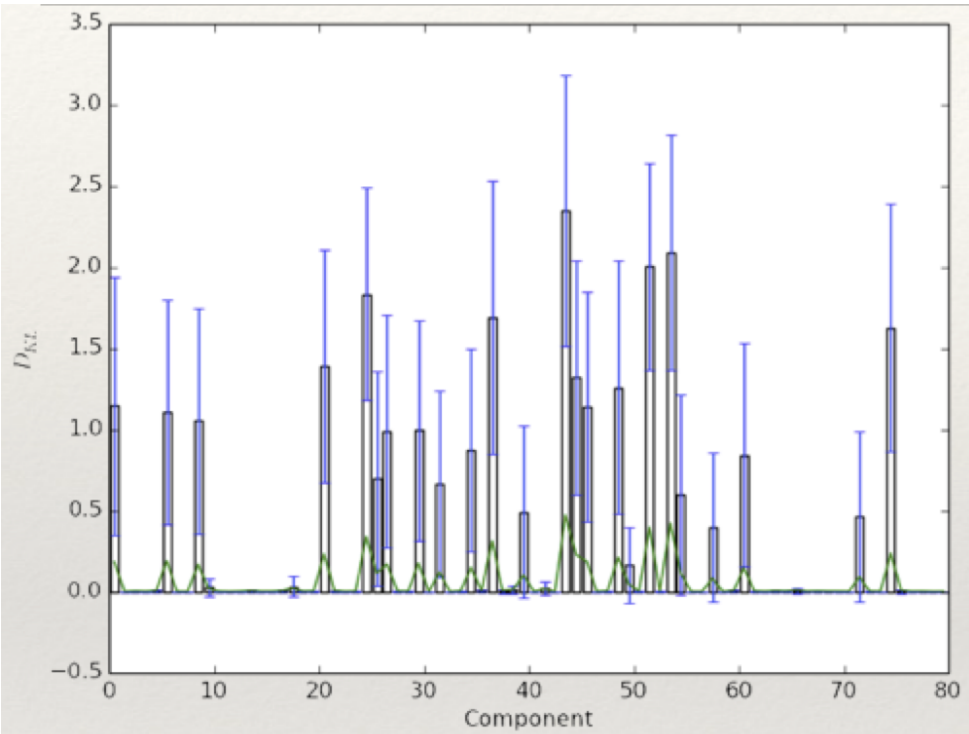


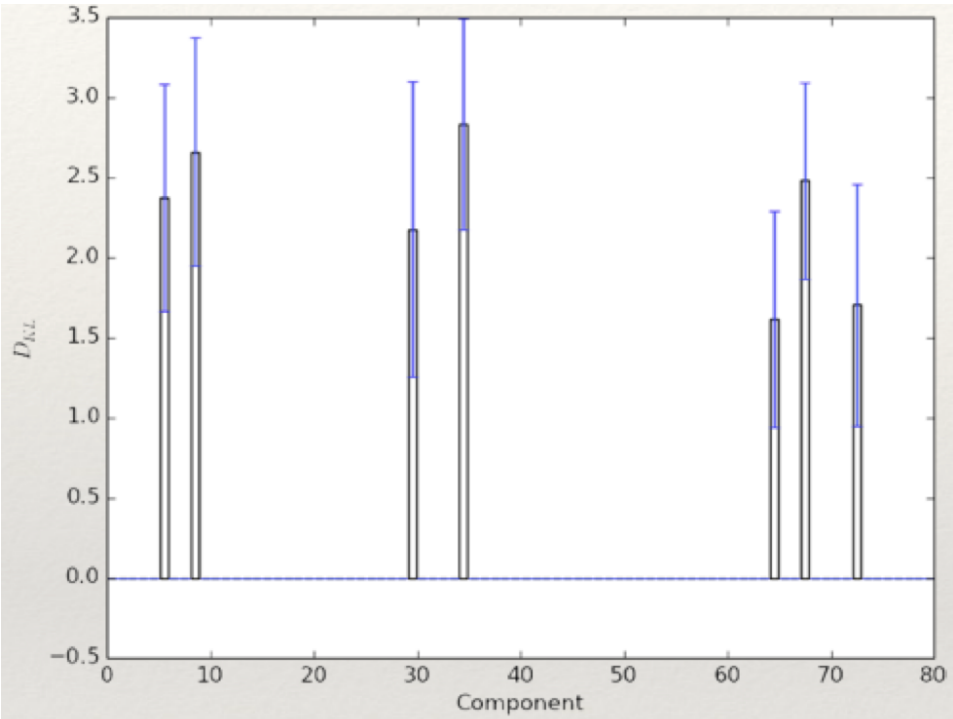
Figure: Goodfellow et al., 2016

VAE WITH DEEP ENCODER/DECODER: COMPONENTS COLLAPSE



Figures from Laurent Dinh & Vincent Dumoulin

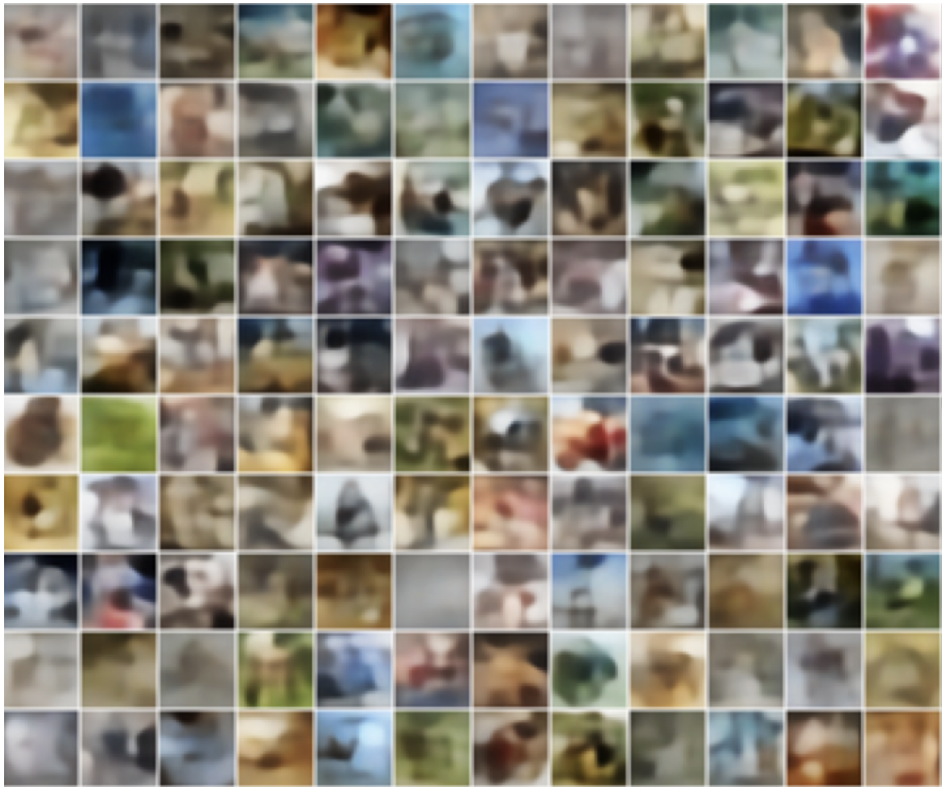
VAE WITH DEEPER ENCODER/DECODER: MORE COMPONENTS COLLAPSE



Figures from Laurent Dinh & Vincent Dumoulin

SAMPLES FROM A VANILLA VAE

Labeled Faces in the Wild (LFW)



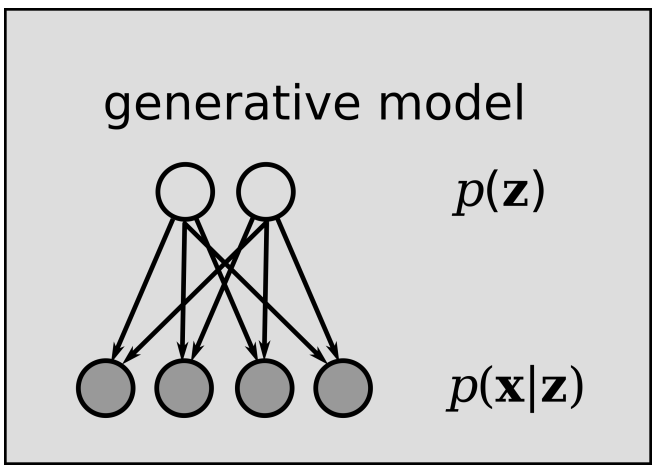
ImageNet (small)

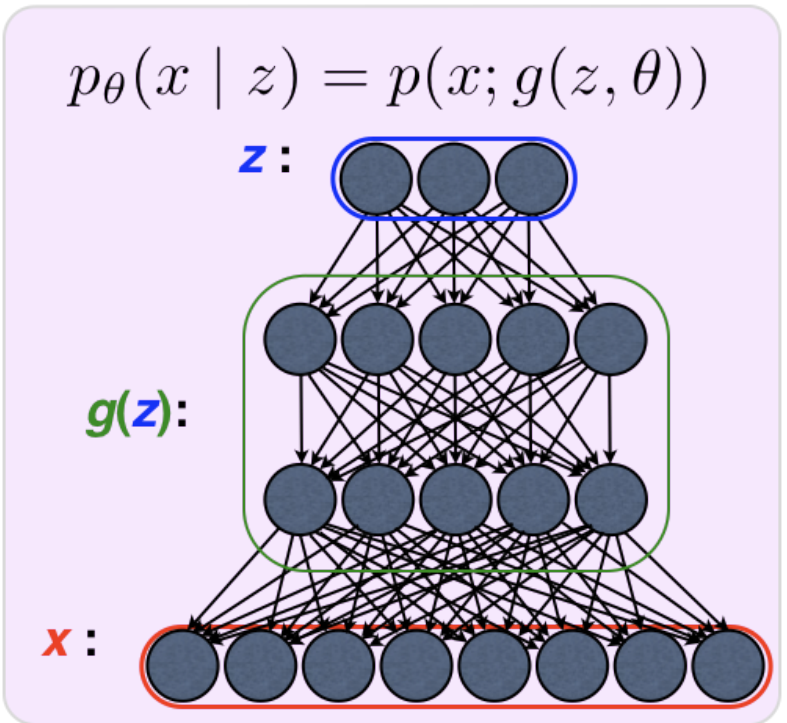
Generative Adversarial Networks (GANs)

GENERATIVE ADVERSARIAL NETWORKS (GANS)

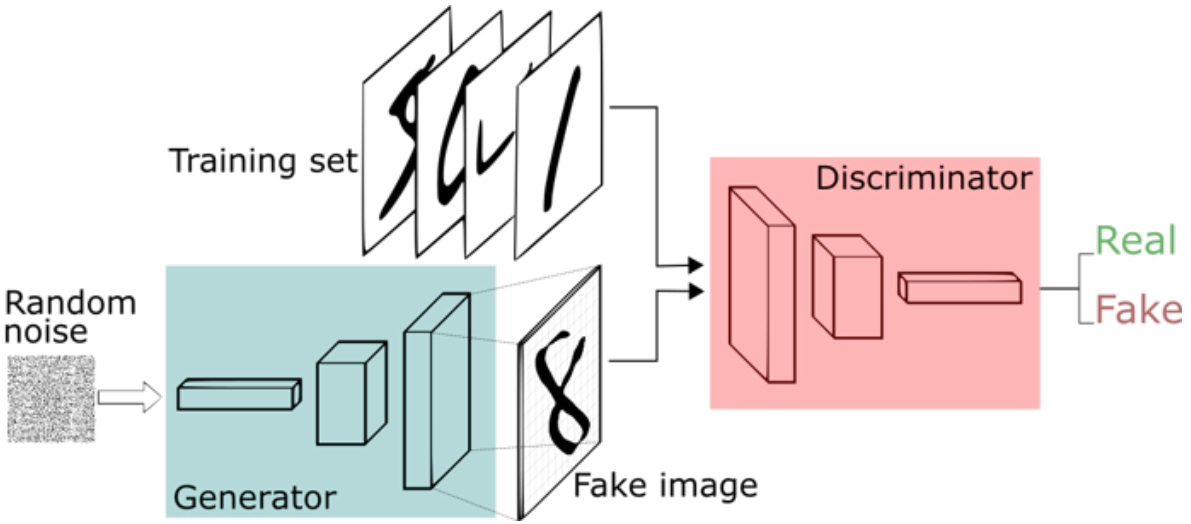
Generative adversarial networks

- define the generative model as in VAEs,
- but approach problem of learning a directed generative model $p(\mathbf{x}|\mathbf{z})$ from a totally different perspective.





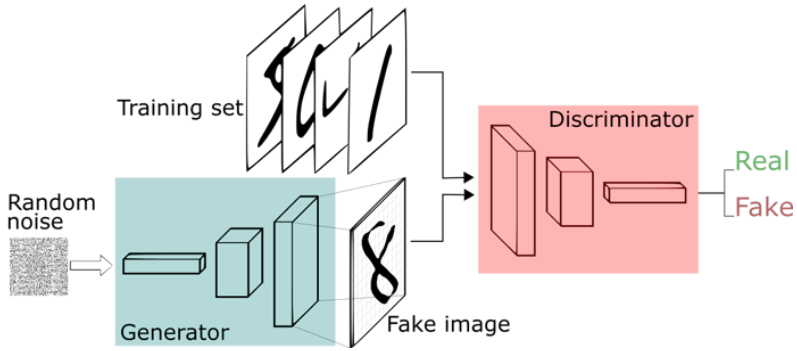
WHAT'S A GAN?



Source : dl4j

- In its simplest form, a GAN consists of two deep neural networks:
  - the generator
  - the discriminator
- The generator is fed a random noise vector which it transforms into a fake sample in a given domain.
- The discriminator is fed both real and fake samples and outputs a number between 0 and 1 indicating the probability of the input being real.

WHAT'S A GAN?



Source : dl4j

- The goal of the generator is to fool the discriminator into thinking that the synthesized samples are real
- The goal of the discriminator is to accurately recognize real samples and not be fooled by the generator.
- This sets off an arms race. As the generator gets better at producing realistic samples, the discriminator is forced to get better at detecting the fake samples which in turn forces the generator to get even better at producing realistic samples and so on.

GANS : FAKE CURRENCY ILLUSTRATION

The generative model can be thought of as analogous to a team of counterfeiters, trying to produce fake currency and use it without detection, while the discriminative model is analogous to the police, trying to detect the counterfeit currency. Competition in this game drives both teams to improve their methods until the counterfeits are indistinguishable from the genuine articles.

-Ian Goodfellow

MINIMAX LOSS FOR GANS

$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}(\mathbf{x})}}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))]$$

- $p_{\text{data}(\mathbf{x})}$ is our target, the data distribution.
- Recall, that we defined the generator to be neural network mapping a latend random vector $\mathbf{z}$ to generated samples $G(\mathbf{z})$. Thus even if the generator is a determinisic function, we have random outputs, i.e. variability.
- $p(\mathbf{z})$ is usually a uniform distribution or an isotropic Gaussian. It is typically fixed and not adapted during training.

MINIMAX LOSS FOR GANS

$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))]$$

- $G(\mathbf{z})$ is the output of the generator for a given state $\mathbf{z}$ of the latent variables.
- $D(\mathbf{x})$ is the output of the discriminator for a real sample $\mathbf{x}$.
- $D(G(\mathbf{z}))$ is the output of the discriminator for a fake sample $G(\mathbf{z})$ synthesized by the generator.

MINIMAX LOSS FOR GANS

$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}(\mathbf{x})}}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))]$$

- Roughly speaking, $\mathbb{E}_{\mathbf{x} \sim p_{\text{data}(\mathbf{x})}}[\log D(\mathbf{x})]$ is the log-probability of correctly classifying real data points as real.
- $\mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))]$ is the log-probability of correctly classifying fake samples as fake.
- Therefore, with each gradient update, the discriminator tries to push $D(\mathbf{x})$ toward 1 and $D(G(\mathbf{z}))$ toward 0. This is the same as maximizing V(D,G).
- The generator, on the other hand, only has control over $D(G(\mathbf{z}))$ and tries to push that toward 1 with each gradient update. This is the same as minimizing V(D,G).

GAN TRAINING : PSEUDOCODE

Algorithm Minibatch stochastic gradient descent training of GANs. The number of steps ,k to apply to the discriminatoris a hyperparameter

for number of training iterations **do**
  for k steps **do**
    Sample minibatch of m noise samples $\{\mathbf{z}^{(1)} \dots \mathbf{z}^{(m)}\}$ from the noise prior $p_g(\mathbf{z})$
    Sample minibatch of m examples $\{\mathbf{x}^{(1)} \dots \mathbf{x}^{(m)}\}$ from the data generating distribution $p_{\text{data}}(\mathbf{x})$.
    Update the discriminator by ascending its stochastic gradient:

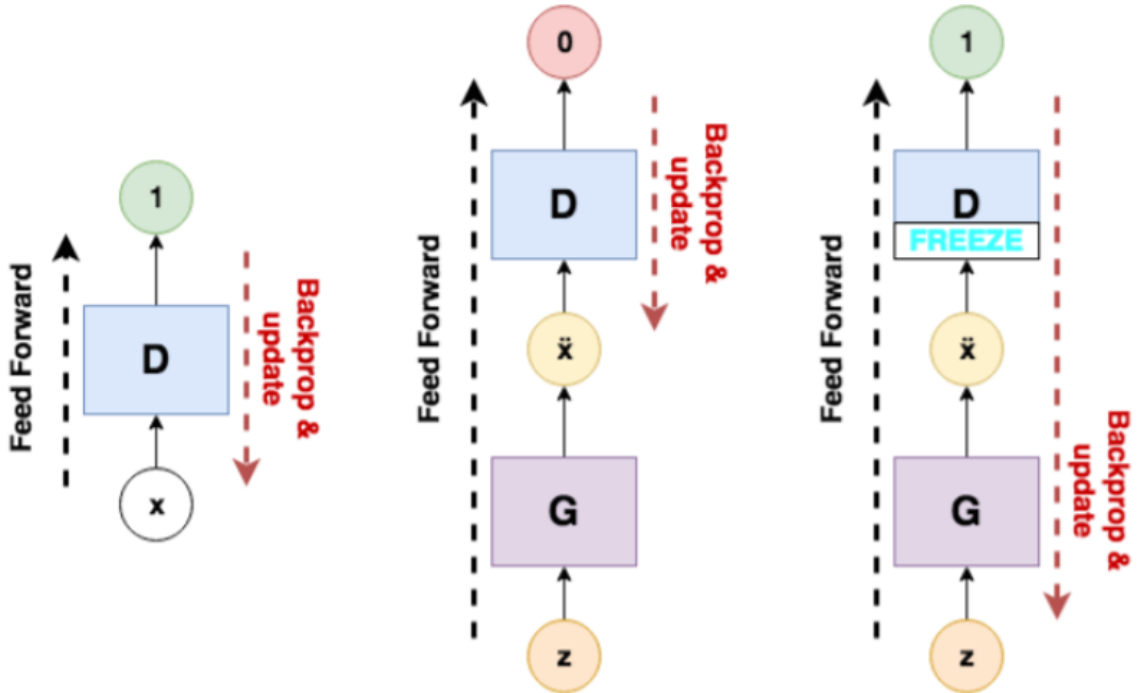
$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D(\mathbf{x}^{(i)}) + \log(1 - D(G(\mathbf{z}^{(i)}))) \right]$$

  end for
  Sample minibatch of m noise samples $\{\mathbf{z}^{(1)} \dots \mathbf{z}^{(m)}\}$ from the noise prior $p_g(\mathbf{z})$
  Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(1 - D(G(\mathbf{z}^{(i)})))$$

end for

GAN TRAINING : ILLUSTRATION



Source : Pascual et al (2017)

- For *k* steps, G's parameters are frozen and one performs **gradient ascent** on D to increase it's accuracy.
- Finally, D's parameters are frozen and one performs **gradient descent** on G to increase it's generation performance.
- Note, that G gets to peek at D's internals (from the back-propogated errors) but D doesn't get to peek at G.

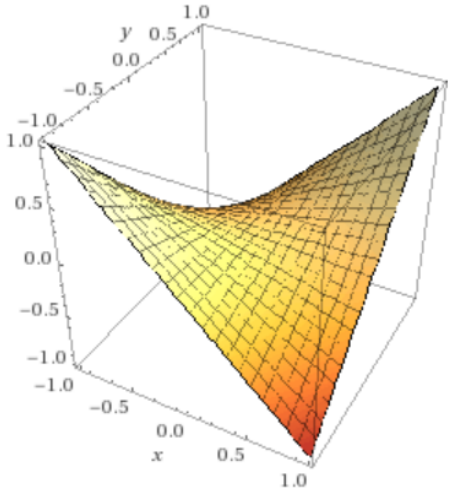
ADVERSARIAL TRAINING

- So, GANs have intuitive appeal and produce state-of-the-art results but the reason they're such an important breakthrough is that they introduce a whole new paradigm to training deep neural networks: Adversarial Training.

- Compared to plain-vanilla gradient descent, adversarial training is a whole different beast; one that we haven't figured out how to tame (yet).

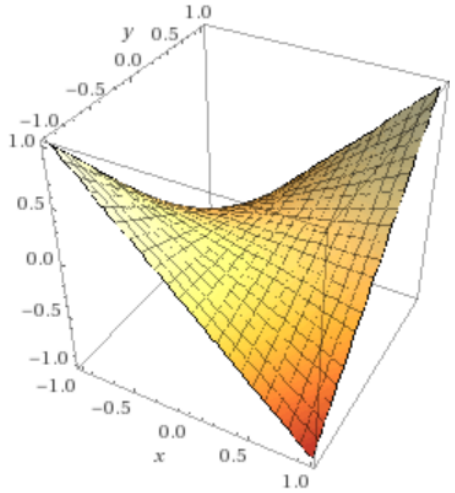
- To illustrate this, let's look at a simple example.

ADVERSARIAL TRAINING: A TOY EXAMPLE



- Consider the function $f(x, y) = xy$, where x and y are both scalars.
- Player A can control x and Player B can control y
- The loss:
  - Player A : $L_A(x, y) = xy$
  - Player B : $L_B(x, y) = -xy$
- This can be rewritten as $L(x, y) = \min_x \max_y xy$
- What we have here is a simple zero-sum game with its characteristic minimax loss

POSSIBLE BEHAVIOUR #1: CONVERGENCE

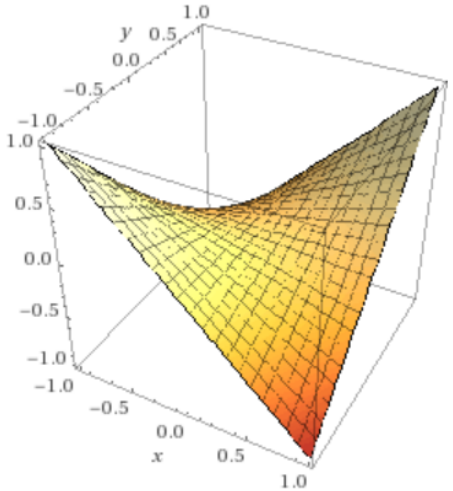


- The partial derivatives of the losses are:

$$\frac{\partial L_A}{\partial x} = y \ , \ \frac{\partial L_B}{\partial y} = -x$$

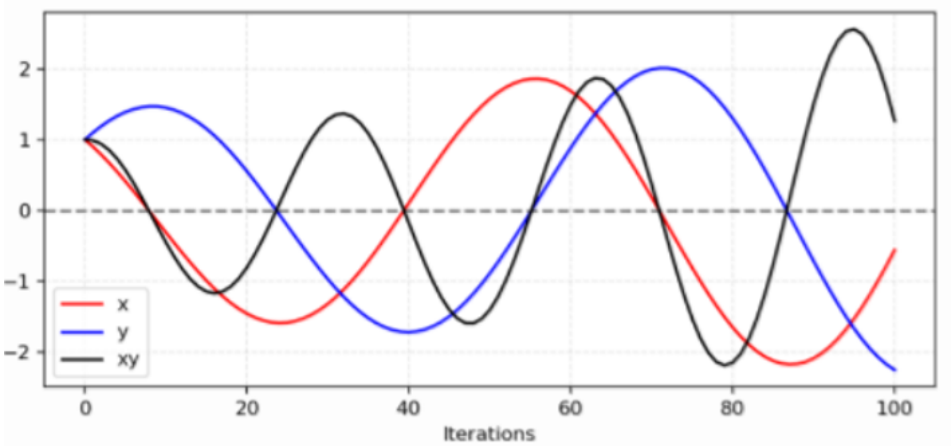
- In adversarial training, both players perform gradient descent on their respective losses.
- To perform simultaneous gradient descent, we update x with $x - \alpha \cdot y$ and y with $y + \alpha \cdot x$ simultaneously in one iteration, where α is the learning rate.

POSSIBLE BEHAVIOUR #1: CONVERGENCE



- In order for simultaneous gradient descent to converge to a fixed point, both gradients have to be simultaneously 0.
- They are both (simultaneously) zero only for the point (0,0).
- This is a saddle point of the function $f(x, y) = xy$.
  - The fixed point for a minimax game is typically a saddle point.
  - Such a fixed point is an example of a Nash equilibrium.
- In adversarial training, convergence to a fixed point is **NOT** guaranteed.

POSSIBLE BEHAVIOUR #2: CHAOTIC BEHAVIOUR



Credit : Lilian Weng

Figure: A simulation of our example for updating x to minimize xy and updating y to minimize -xy. The learning rate eta = 0.1. With more iterations, the oscillation grows more and more unstable.

- Once *x* and *y* have different signs, every following gradient update causes huge oscillation and the instability gets worse in time, as shown in the figure.

POSSIBLE BEHAVIOUR #3: CYCLES

Credit : Goodfellow

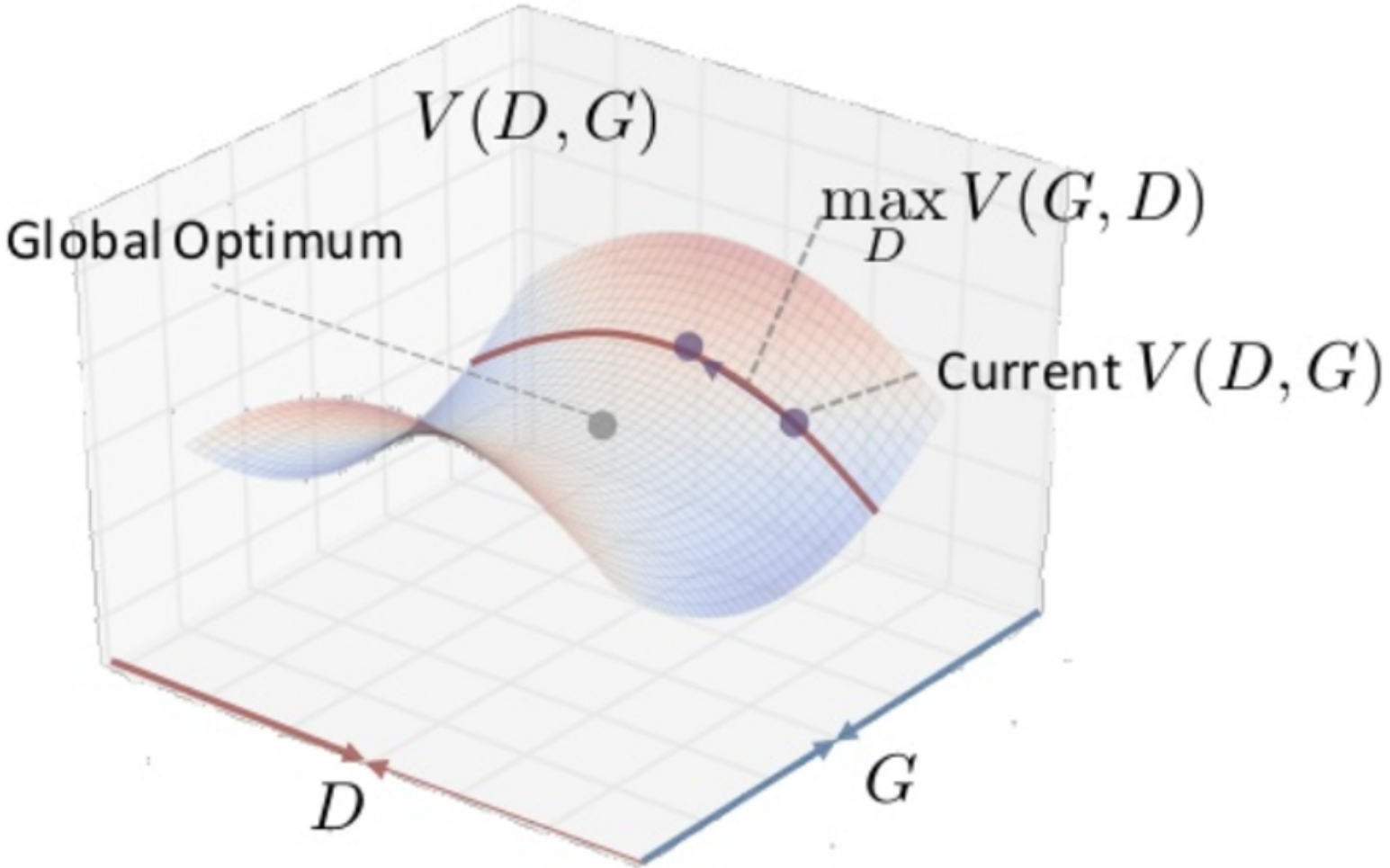
Figure: Simultaneous gradient descent with an infinitesimal step size can result in a circular orbit in the parameter space

- A Discrete Example: A never-ending game of Rock-Paper-Scissors where player A chooses 'Rock' $\rightarrow$ player B chooses 'Paper' $\rightarrow$ A chooses 'Scissors' $\rightarrow$ B chooses 'Rock' $\rightarrow$ ...
- **Moral:** Adversarial training is highly unpredictable. It can go in circles or become chaotic.

NON-STATIONARY LOSS SURFACE

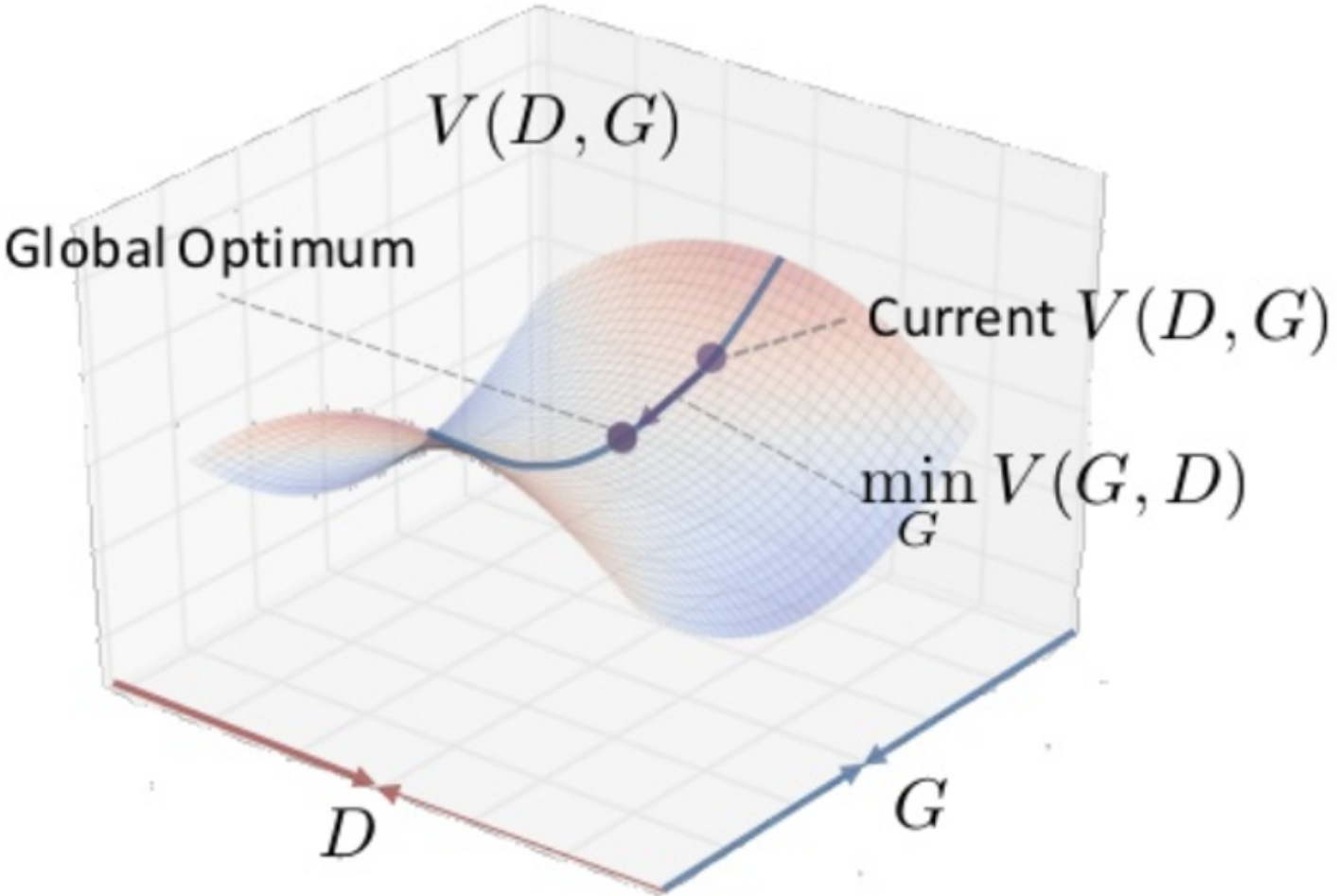
- Once again, it is extremely important to note that from the perspective of one of the players, the loss surface changes every time the other person makes a move.
- This is in stark contrast to the (full batch) gradient descent case where the loss surface is stationary no matter how many iterations of gradient descent are performed.

ILLUSTRATION OF CONVERGENCE



Credit : Mark Chang

ILLUSTRATION OF CONVERGENCE : FINAL STEP



Credit : Mark Chang

Such convergence is not guaranteed, however.

DIVERGENCE MEASURES

- Recall that the goal of generative modeling is to learn $p_{\text{data}}(\mathbf{x})$.
- In order to understand the differences between different generative models, it is sometimes helpful to consider them from the angle of **divergence measures**.
- A divergence measure quantifies the distance between two distributions. It is a measure of how different one distribution is from another.
- There are many different divergence measures that one can use here (such as the Kullback-Liebler divergence).
- One thing that all such measures have in common is that they are 0 if and only if the two distributions are equal to each other (otherwise, they are all positive).

DIVERGENCE MEASURES

- One approach to training generative models, then, is to explicitly minimize the distance between $p_{\text{data}}(\boldsymbol{x})$ and the model distribution $p(\boldsymbol{x})$ according to some divergence measure.
- If our generator has the capacity to model $p_{\text{data}}(\boldsymbol{x})$ perfectly, the choice of divergence doesn't matter much because they all achieve their minimum (that is,0) when $p(\boldsymbol{x}) = p_{\text{data}}(\boldsymbol{x})$.
- However, it is not likely that that the generator, which is parametrized by the weights of a neural network, is capable of perfectly modelling an arbitrary $p_{\text{data}}(\boldsymbol{x})$.
- In such a scenario, the choice of divergence measure matters, because the parameters that miniminize the various divergence measures differ.

IMPLICIT DIVERGENCE MEASURE OF GANS

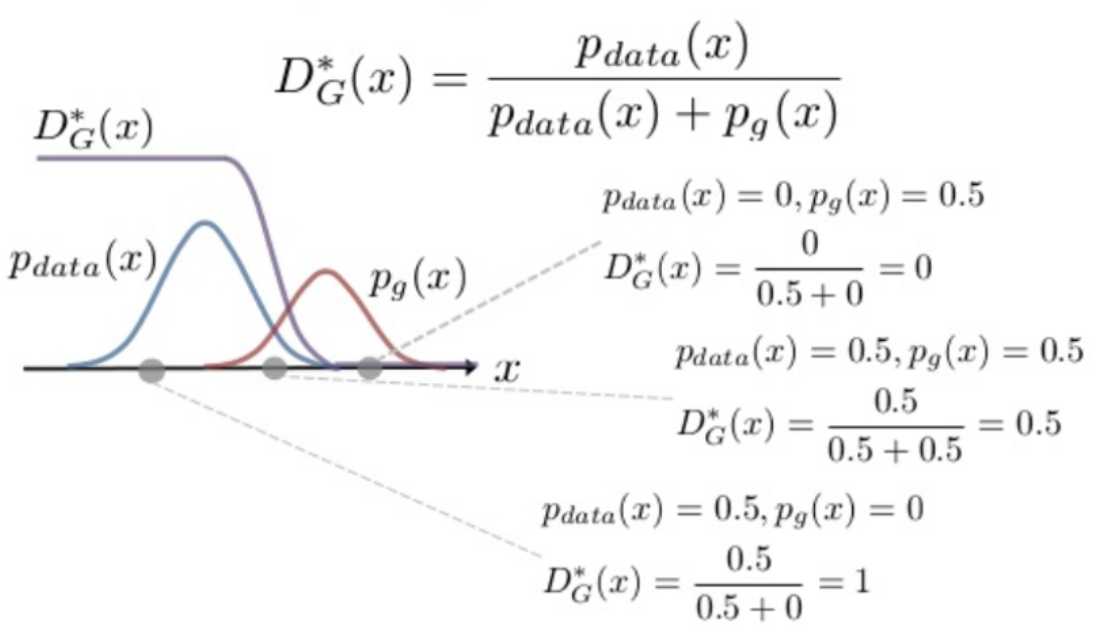
- GANs do not explicitly minimize any divergence measure.
- However, (under some assumptions!) optimizing the minimax loss is equivalent to implicitly minimizing a divergence measure.
- That is, if the optimal discriminator is found in every iteration, the generator minimizes the **Jensen-Shannon divergence (JSD)** (theorem and proof are given by the original GAN paper (Goodfellow et al, 2014)):

$$JS(p_{\text{data}}|p) = \frac{1}{2}KL(p_{\text{data}}||\frac{p_{\text{data}}+p}{2}) + \frac{1}{2}KL(p||\frac{p_{\text{data}}+p}{2})$$

$$KL(p_{\text{data}}|p) = E_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[log\frac{p_{\text{data}}(\boldsymbol{x})}{p(\boldsymbol{x})}]$$

OPTIMAL DISCRIMINATOR

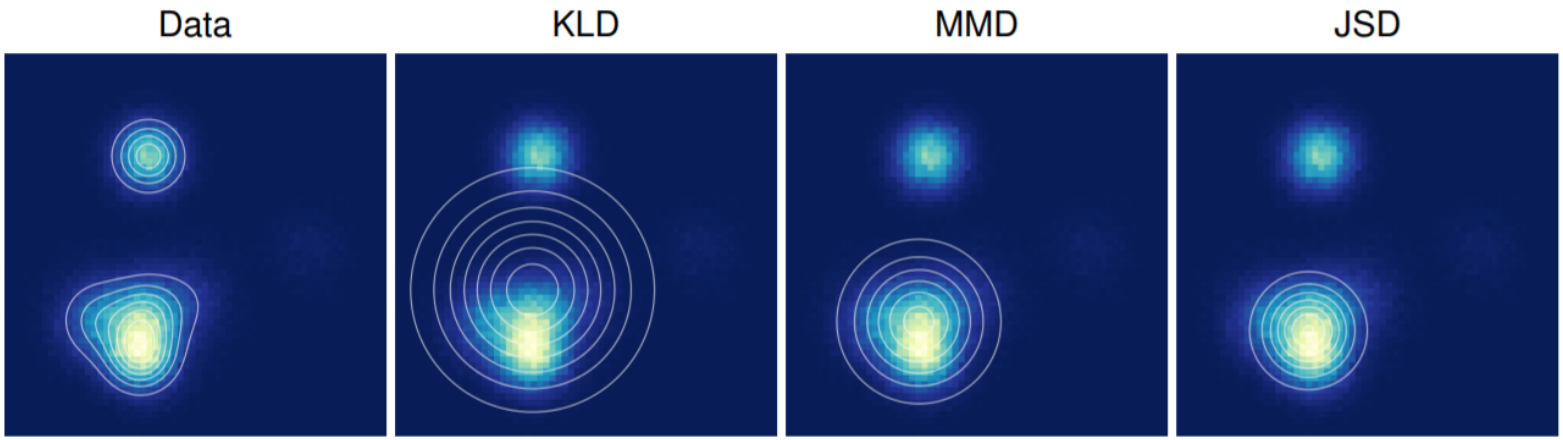
For G fixed, the optimal discriminator D is:



Credit : Mark Chang

- Note: The optimal solution is almost never found in practice, since the discriminator has a finite capacity and is trained on a finite amount of data.
- Therefore, the assumption needed to guarantee that the generator minimizes the JSD does usually not hold in practice.

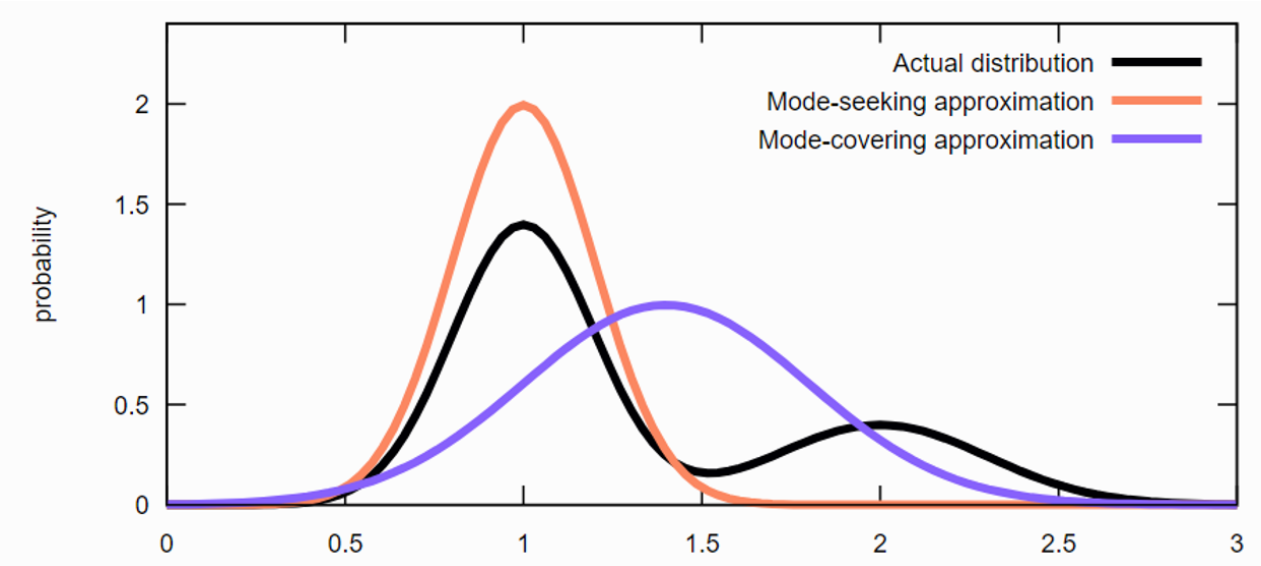
TRADE-OFFS MADE BY SOME COMMON DIVERGENCES



Source : Theis et al 2016

Figure: : An isotropic Gaussian distribution was fit to data drawn from a mixture of Gaussians by either minimizing Kullback-Leibler divergence (KLD), maximum mean discrepancy (MMD), or Jensen-Shannon divergence (JSD). The different fits demonstrate different tradeoffs made by the three measures of distance between distributions.

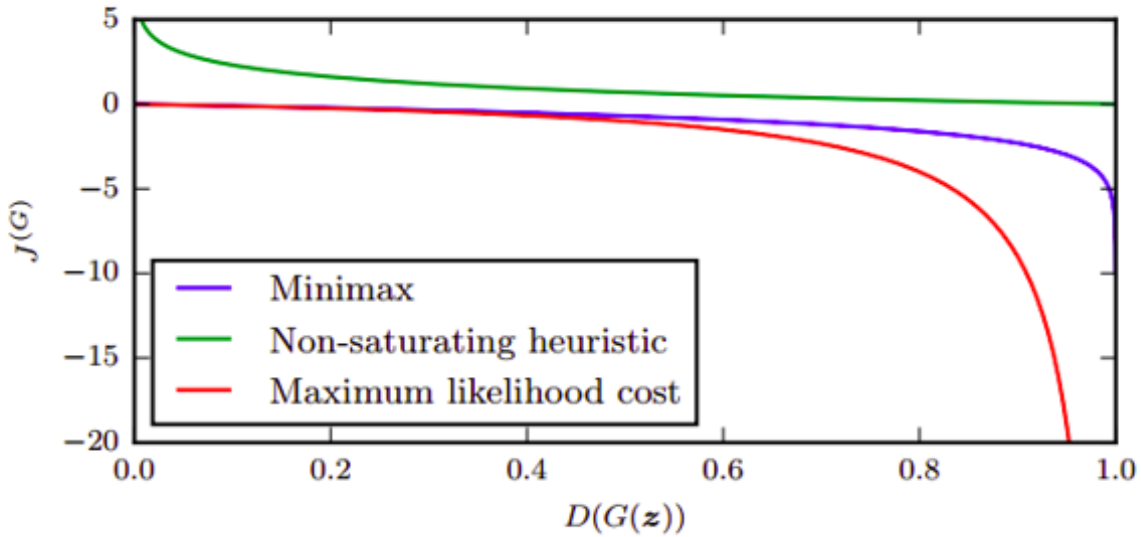
TRADE-OFFS MADE BY SOME COMMON DIVERGENCES



Credit : Aiden NIbali

- In this simplified 1-dimensional example, $p_{\text{data}}(x)$ is a bimodal distribution, but $p(x)$ only has the modelling capacity of a single Gaussian.
- Therefore, based on the divergence measure, $p(x)$ can either fit a single mode really well, i.e. be ‘mode-seeking´(e.g. JSD), or attempt to cover both modes, i.e. be mode-covering´(e.g. KLD).

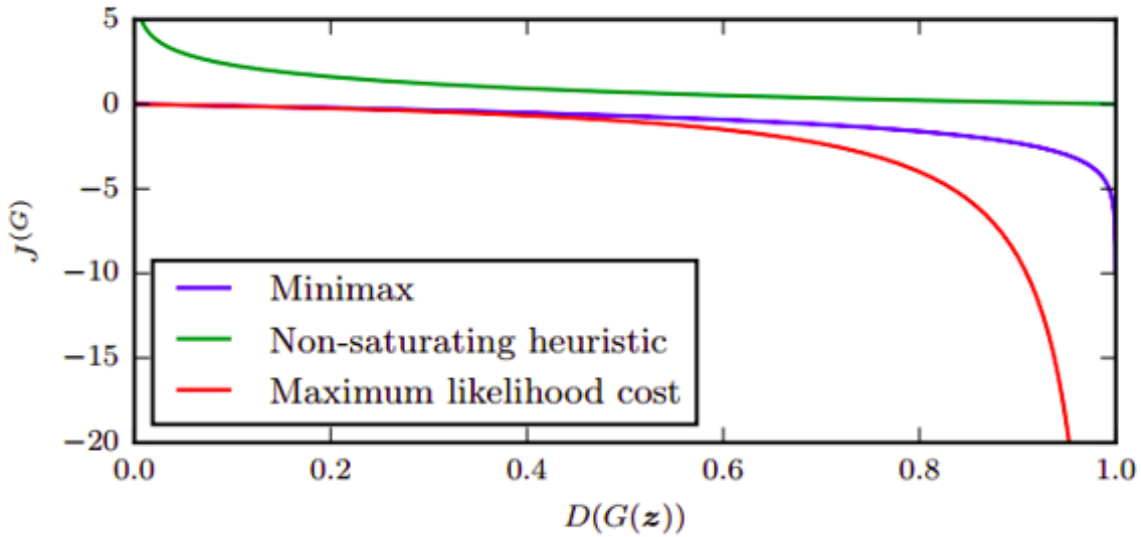
NON-SATURATING LOSS



Credit : Daniel Seita

- It was discovered that a relatively strong discriminator could completely dominate the generator.
- When optimizing the minimax loss, as the discriminator gets good at identifying fake images, i.e. as $D(G(\mathbf{z}))$ approaches 0, the gradient with respect to the generator parameters vanishes.

NON-SATURATING LOSS



Credit : Daniel Seita

- Solution: Use a non-saturating generator loss instead: $J^{(G)} = -\frac{1}{2}\mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[\log D(G(\mathbf{x}))]$
- In contrast to the minimax loss, when the discriminator gets good at identifying fake images, the magnitude of the gradient of $J^{(G)}$ increases and the generator is able to learn to produce better images in successive iterations.

OTHER LOSS FUNCTIONS

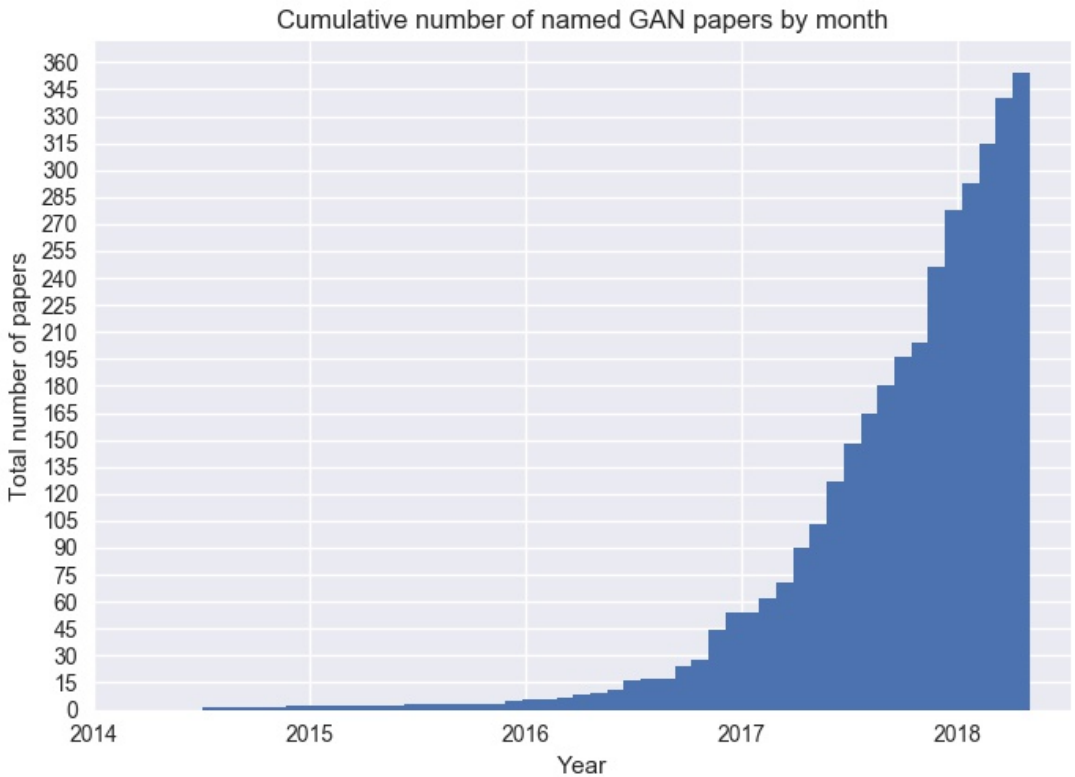
Various minimax losses for GAN training with different properties have been proposed:

GAN	Discriminator Loss	Generator Loss
MM GAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{GAN}} = -\mathbb{E}_{x\sim p_d}[\log(D(x))] - \mathbb{E}_{\hat{x}\sim p_g}[\log(1-D(\hat{x}))]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{GAN}} = \mathbb{E}_{\hat{x}\sim p_g}[\log(1-D(\hat{x}))]$
NS GAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{NSGAN}} = -\mathbb{E}_{x\sim p_d}[\log(D(x))] - \mathbb{E}_{\hat{x}\sim p_g}[\log(1-D(\hat{x}))]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{NSGAN}} = -\mathbb{E}_{\hat{x}\sim p_g}[\log(D(\hat{x}))]$
WGAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{WGAN}} = -\mathbb{E}_{x\sim p_d}[D(x)] + \mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{WGAN}} = -\mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})]$
WGAN GP	$\mathcal{L}_{\mathrm{D}}^{\mathrm{WGANGP}} = \mathcal{L}_{\mathrm{D}}^{\mathrm{WGAN}} + \lambda\mathbb{E}_{\hat{x}\sim p_g}[(\|\nabla D(\alpha x + (1-\alpha\hat{x})\|_2 - 1)^2]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{WGANGP}} = -\mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})]$
LS GAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{LSGAN}} = -\mathbb{E}_{x\sim p_d}[(D(x)-1)^2] + \mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})^2]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{LSGAN}} = -\mathbb{E}_{\hat{x}\sim p_g}[(D(\hat{x}-1)^2]$
DRAGAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{DRAGAN}} = \mathcal{L}_{\mathrm{D}}^{\mathrm{GAN}} + \lambda\mathbb{E}_{\hat{x}\sim p_d+\mathcal{N}(0,c)}[(\|\nabla D(\hat{x})\|_2 - 1)^2]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{DRAGAN}} = \mathbb{E}_{\hat{x}\sim p_g}[\log(1-D(\hat{x}))]$
BEGAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{BEGAN}} = \mathbb{E}_{x\sim p_d}[\|x - \mathrm{AE}(x)\|_1] - k_t\mathbb{E}_{\hat{x}\sim p_g}[\|\hat{x} - \mathrm{AE}(\hat{x})\|_1]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{BEGAN}} = \mathbb{E}_{\hat{x}\sim p_g}[\|\hat{x} - \mathrm{AE}(\hat{x})\|_1]$

Source : Lucic et al 2016

EXPLOSIVE GROWTH IN THE NUMBER OF (NAMED) GAN PAPERS

Understanding and improving GAN training is a very active area of research.

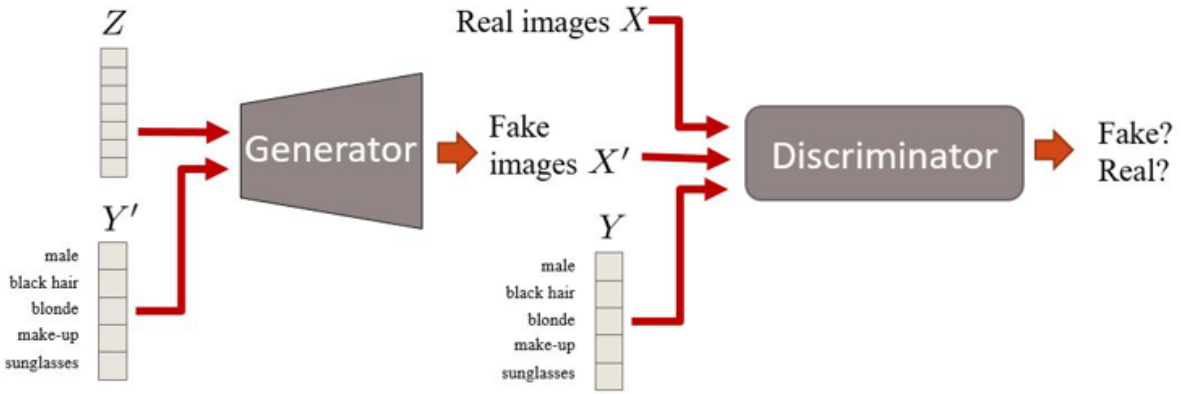


Credit : hindupuravinash

CONDITIONAL GANS: MOTIVATION

- In an ordinary GAN, the only thing that is fed to the generator are the latent variables **z**.
- A conditional GAN allows you to condition the generative model on additional variables.
- E.g. a generator conditioned on text input (in addition to **z**) can be trained to generate the image described by the text.

CONDITIONAL GANS : ARCHITECTURE



Credit : Guim Perarnau

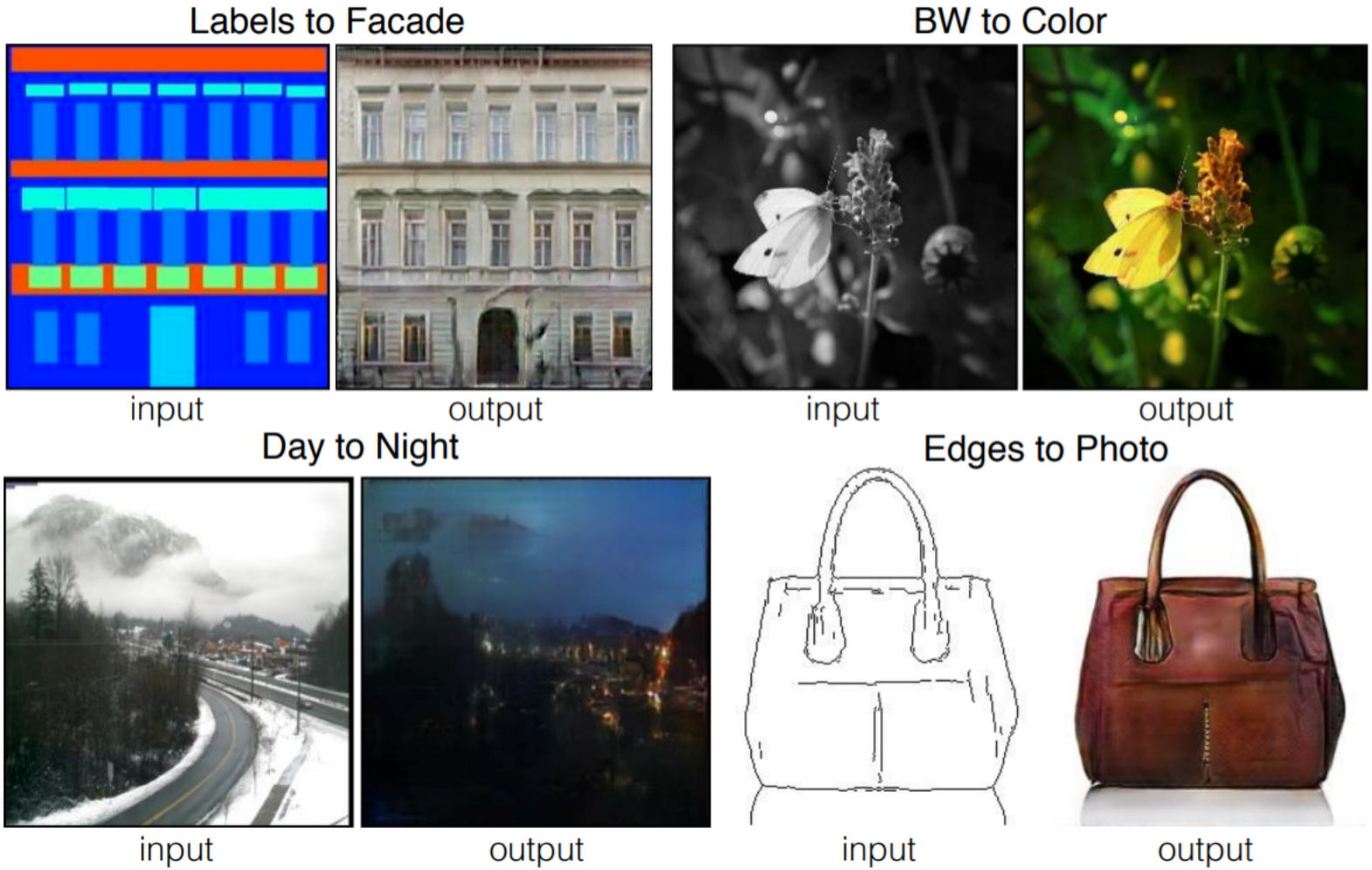
- In a conditional GAN, additional information in the form of vector ***y*** is fed to both the generator and the discriminator.
- ***z*** can then encode all variations in ***z*** that are not encoded by ***y***.
- E.g. ***y*** could encode the class of a hand-written number (from 0 to 9). Then, ***z*** could encode the style of the number (size, weight, rotation, etc).

CONDITIONAL GANS: EXAMPLE

MNIST digits generated conditioned on their class label.

Source : Mirza et al 2014

CONDITIONAL GANS: MORE EXAMPLES



Source : Isola et al 2016

MORE GENERATIVE MODELS

- Today, we learned about to kinds of (directed) generative models:
  - Variational Autoencoders (VAEs)
  - Generative Adversarial Networks (GANs).
- There are more interesting generative models, e.g.:
  - autoregressive models
  - restricted Boltzmann machines.
- Note:
  - It's important to bear in mind that generative models are not a solved problem.
  - There are many interesting hybrid models that combine two or more of these approaches.

ACKNOWLEDGEMENTS

Big thanks to

- **Christian Igel** (University of Copenhagen) for providing his material on graphical models
- **Aaron Courville** (Univerite de Montreal) for providing his material on VAEs

which helped creating this lecture.

REFERENCES

Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, Dimitris Metaxas (2017)

StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks

https://arxiv.org/abs/1612.03242

Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, Bryan Catanzaro (2017)

High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs

https://arxiv.org/abs/1711.11585

Ugur Demir, Gozde Unal (2018)

Patch-Based Image Inpainting with Generative Adversarial Networks

https://arxiv.org/abs/1803.07422

Tero Karras, Timo Aila, Samuli Laine, Jaakko Lehtinen (2018)

Progressive Growing of GANs for Improved Quality, Stability, and Variation

https://arxiv.org/abs/1710.10196

REFERENCES

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio (2014)

Generative Adversarial Networks

https://arxiv.org/abs/1406.2661

Santiago Pascual, Antonio Bonafonte, Joan Serra (2017)

SEGAN: Speech Enhancement Generative Adversarial Network

https://arxiv.org/abs/1703.09452

Ian Goodfellow (2016)

NIPS 2016 Tutorial: Generative Adversarial Networks

https://arxiv.org/abs/1701.00160

Lilian Weng (2017)

From GAN to WGAN

https://lilianweng.github.io/lil-log/2017/08/20/from-GAN-to-WGAN.html

REFERENCES

Mark Chang (2016)

Generative Adversarial Networks

https://www.slideshare.net/ckmarkohchang/generative-adversarial-networks

Lucas Theis, Aaron van den Oord, Matthias Bethge (2016)

A note on the evaluation of generative models

https://arxiv.org/abs/1511.01844

Aiden Nibali (2016)

The GAN objective, from practice to theory and back again

https://aiden.nibali.org/blog/2016-12-21-gan-objective/

Mehdi Mirza, Simon Osindero (2014)

Conditional Generative Adversarial Nets

https://arxiv.org/abs/1411.1784

REFERENCES

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, Alexei A. Efros (2016)

Image-to-Image Translation with Conditional Adversarial Networks

https://arxiv.org/abs/1611.07004

Guim Perarnau (2017)

Fantastic GANs and where to find them

https://guimperarnau.com/blog/2017/03/Fantastic-GANs-and-where-to-find-them