Introduction to

Deep Generative Models

(Mô hình tạo sinh sâu)

Khoat Than

School of Information and Communication Technology

Hanoi University of Science and Technology

SoICT summer school, August 2024

Contents

Some successes: Text-to-image (2022)

- Draw pictures by descriptions

A bowl of soup

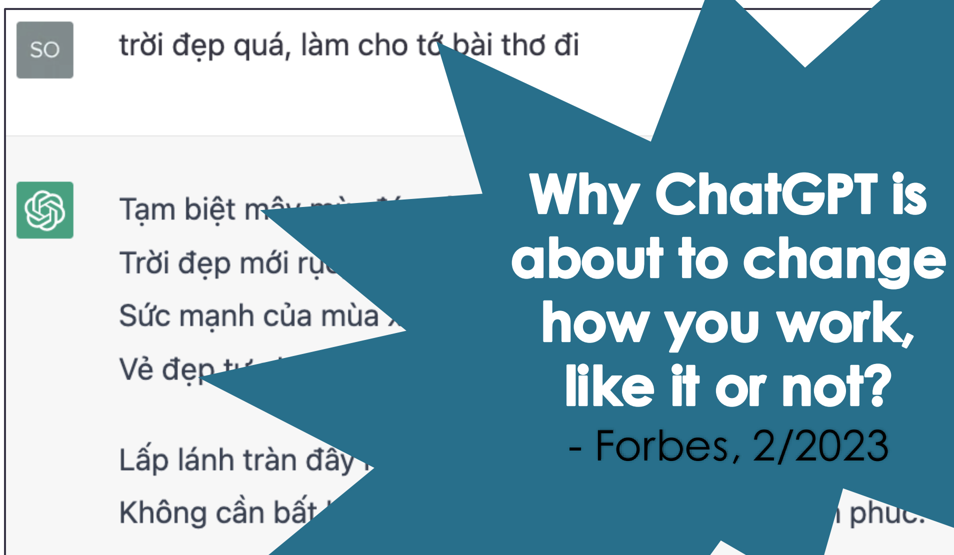
Google Imagen

An extremely angry bird.

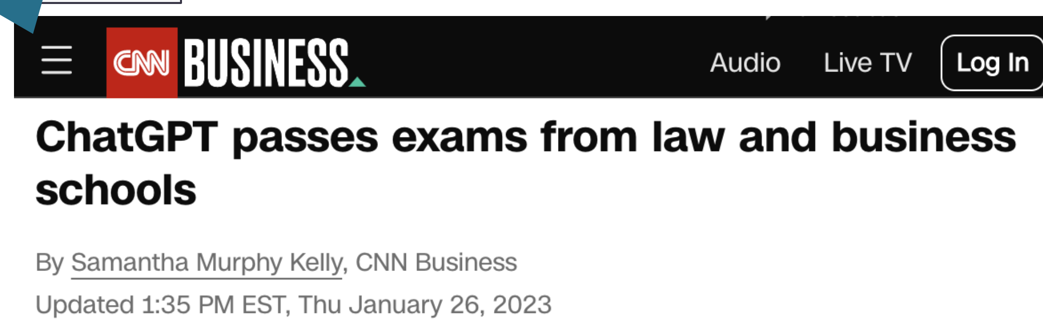
A cute corgi lives in a house made out of sushi.

Some successes: ChatGPT (2022)

- Human-level Chatting, Writing, QA,…



Why ChatGPT is about to change how you work, like it or not?
- Forbes, 2/2023


CNN BUSINESS Audio Live TV Log In

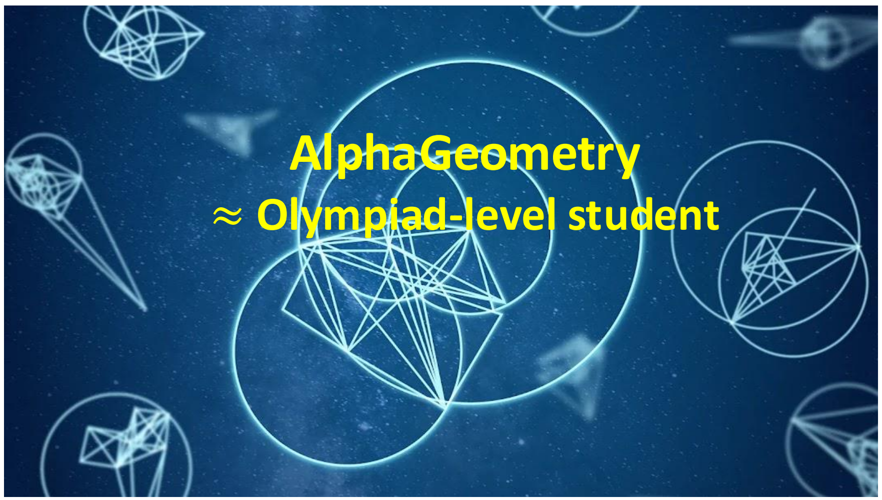
ChatGPT passes exams from law and business schools

By Samantha Murphy Kelly, CNN Business

Updated 1:35 PM EST, Thu January 26, 2023

OpenAI

Some successes: more



Gemini

Generative Models

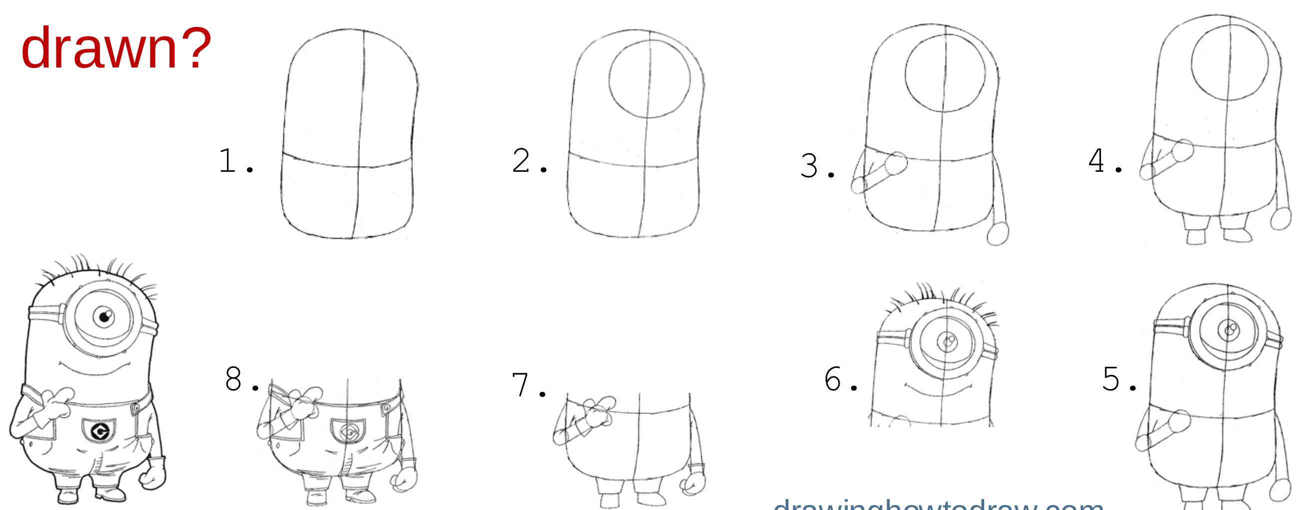
- Probabilistic models of data
- Sample: lấy mẫu dữ liệu (sinh/tạo ra dữ liệu)
- Evaluate likelihood: tính likelihood của dữ liệu cho trước
- Train: huấn luyện
- Representation: biểu diễn mới
- What if all we care about is sampling?
  - *Not in the training data, but the novel samples.*

Probabilistic models

Introduction

Probabilistic model

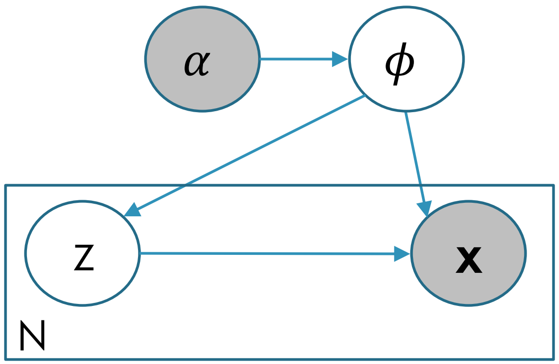
- Our assumption on how the data samples were generated
  (giả thuyết của chúng ta về quá trình mà các mẫu dữ liệu đã được sinh ra như thế nào)
- Example: how a sentence is generated?
  - We assume our brain does as follow:
  - *First choose the topic of the sentence*
  - *Generate the words one-by-one to form the sentence*
- How will TIM be drawn?



drawinghowtodraw.com

Probabilistic model

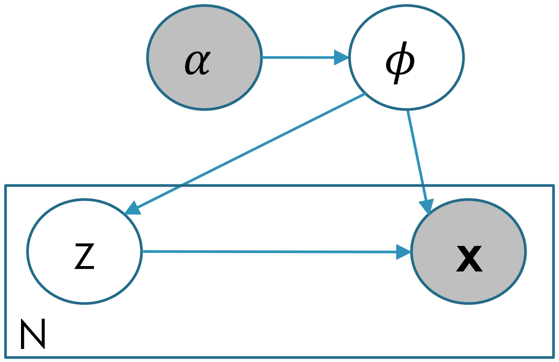
- A model sometimes consists of
  - **Observed variable** (e.g., $\boldsymbol{x}$) which models the observation (data instance)
    (biến quan sát được)
  - **Hidden variable** which describes the hidden things (e.g., z, ϕ)
    (biến ẩn)
  - **Relations** between the variables



- Each variable follows some probability distribution
  (mỗi biến tuân theo một phân bố xác suất nào đó)

Different types of models

- Probabilistic graphical model (PGM): Graph + Probability Theory (mô hình đồ thị xác suất)
  - Each vertex represents a random variable, grey circle means "observed", white circle means "latent"
  - Each edge represents the conditional dependence between two variables

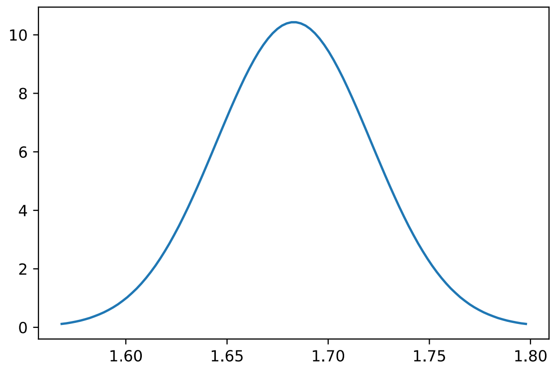


- Latent variable model: a PGM which has at least one latent variable
- **Generative model:** a model that enables us to generate data instances

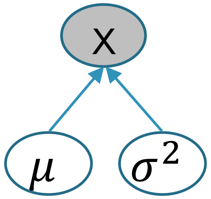
Univariate normal distribution

- We wish to know the average height of a person
  - We had collected a dataset from 10 people in Hanoi:
    D = {1.6, 1.7, 1.65, 1.63, 1.75, 1.71, 1.68, 1.72, 1.77, 1.62}
- Let x denote the random variable that represents the height of a person
- **Assumption:** x follows a Normal distribution (Gaussian) with the following *probability density function* (PDF)

$$\mathcal{N}(x|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(x-\mu)^2}$$



  - where $\{\mu, \sigma^2\}$ are the mean and variance
- Note:
  - $\mathcal{N}(x|\mu,\sigma^2)$ represents the class of normal distributions
  - This class is parameterized by $\boldsymbol{\theta} = (\mu, \sigma^2)$



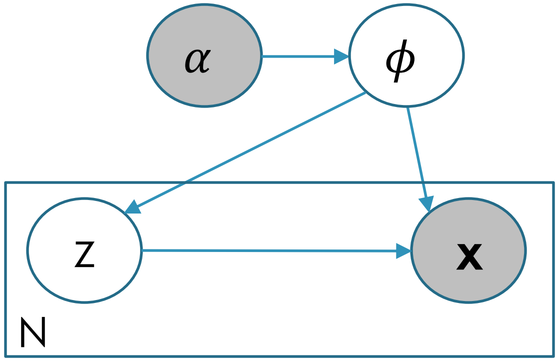
- **Learning:** we need to know specific values of $\{\mu, \sigma^2\}$

PGM: some well-known models

- Gaussian mixture model (GMM)
  - Modeling real-valued data
- Latent Dirichlet allocation (LDA)
  - Modeling the topics hidden in textual data
- Hidden Markov model (HMM)
  - Modeling time-series, i.e., data with time stamps or sequential nature
- Conditional Random Field (CRF)
  - for structured prediction
- Deep generative models
  - Modeling the hidden structures, generating artificial data

Probabilistic model: inference & learning

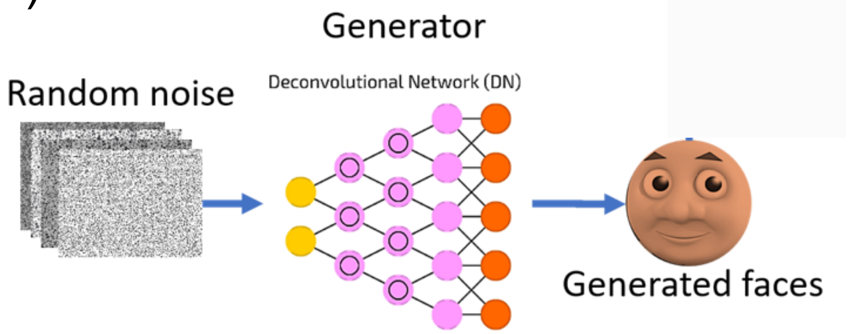
- **Inference** for a given instance $\boldsymbol{x}_n$
  (Suy diễn/phán đoán đối với một quan sát cho trước)
  - Recovery of the local variable (e.g., z_n), or
  - The distribution of the local variables (e.g., $P(z_n|\phi, \boldsymbol{x}_n)$)
  - Example: for GMM, we want to know z_n indicating which Gaussian did generate $\boldsymbol{x}_n$



- **Learning (estimation)**
  (Học/ước lượng mô hình)
  - Given a training dataset, estimate the joint distribution of the variables
    - E.g., estimate the density function $p(\phi, z_1, \dots, z_n, \boldsymbol{x}_1, \dots, \boldsymbol{x}_n|\alpha)$
    - E.g., estimate $P(\boldsymbol{x}_1, \dots, \boldsymbol{x}_n|\alpha)$
    - E.g., estimate α
    - Inference of local variables is often needed

Generative model: sampling

- **Sampling** data
  - Make novel data samples, given a trained model (tạo ra dữ liệu mới từ mô hình đã có)



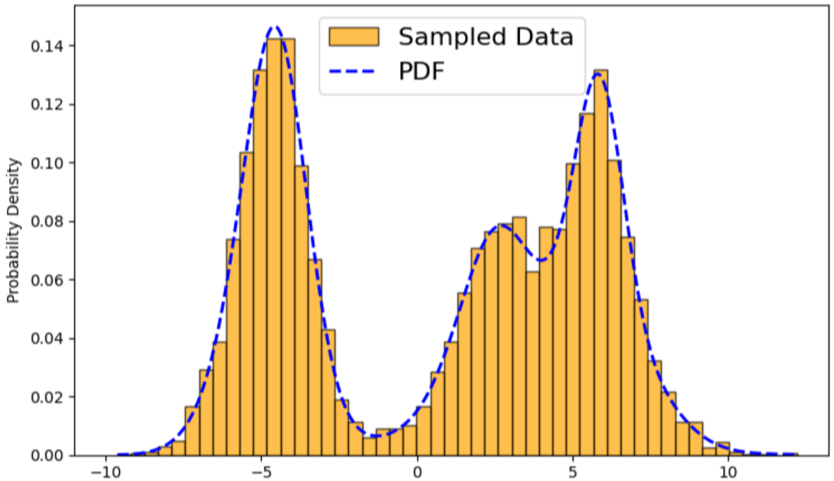
- Application:
  - Entertainment (ngành giải trí): videos, images, musics, …
  - Limited resources: khi khả năng thu thập được ít mẫu dữ liệu
  - Fashion: tạo mẫu quần/áo thời trang
  - Design: tạo mẫu trang thiết bị mới
  - Materials: tạo các vật liệu mới
  - …

Generative models

Learning

Learning a generative model

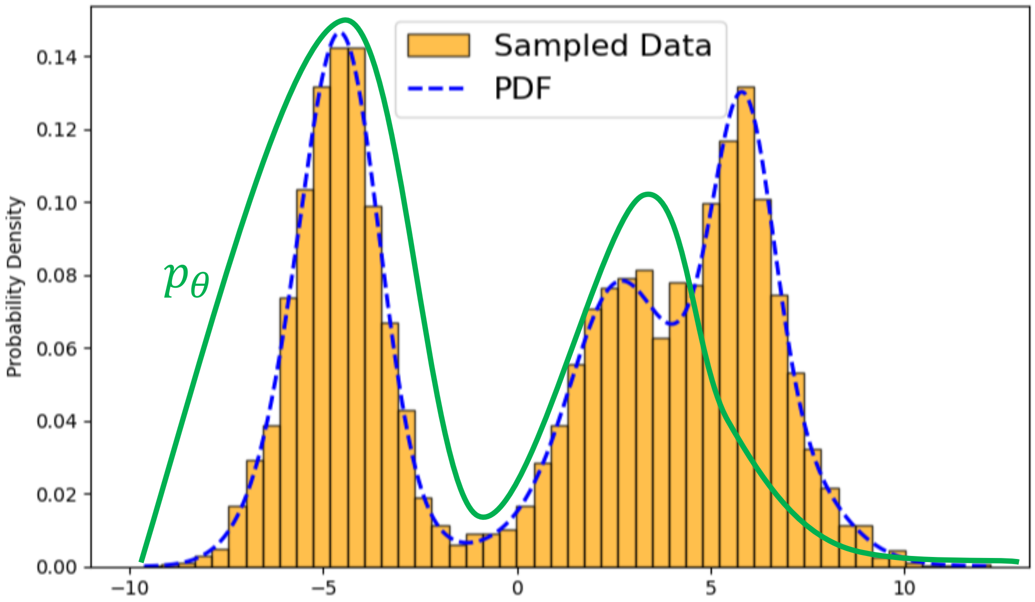
- Given a training set of examples, e.g., images of dogs
- We want to learn a probability distribution P(**x**) over images **x** such that
  - **Generation:** If we sample $\boldsymbol{x}_{new} \sim P(\boldsymbol{x})$, $\boldsymbol{x}_{new}$ should look like a dog (*sampling*)
  - **Density estimation:** P(**x**)
  - **Unsupervised representation learning:** We should be able to learn what these images have in common, e.g., ears, tail, etc. (*features*)



The sampling distribution

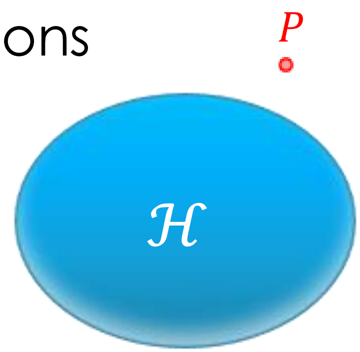
- Dataset **D** = {$\mathbf{x}_1$, $\mathbf{x}_2$, …, $\mathbf{x}_m$}
- **Hardness** of the learning problem:
  - P(**x**) in the space of all probability distributions

- In practice, we often find a $P_\theta(\boldsymbol{x})$ to **approximate** $P(\boldsymbol{x})$



Hypothesis space

- Usually, we can choose a restricted set $\mathcal{H}$ of distributions
  - Parameterized by $\theta \in \Theta$
  - A learner must find one $P_\theta \in \mathcal{H}$



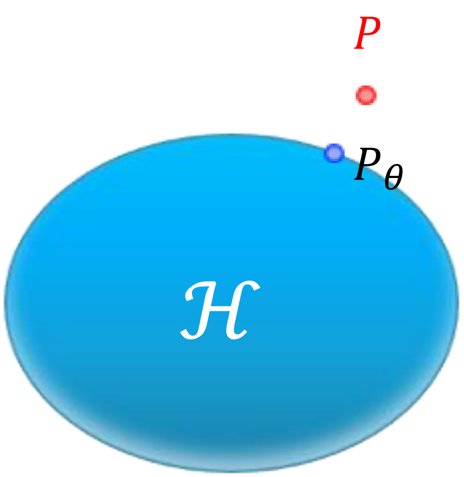
- Hypothesis space *(model family)*:

  a set $\mathcal{H}$ of distributions, providing candidates for a learner
  - Represents prior knowledge about a task
  - Represents our ***inductive bias*** or *preference*
- Each P_θ is often called a "**model**"
- Gaussian family:

$$\mathcal{H} = \{P_\theta : P_\theta \text{ is the normal distribution with } \theta = (\mu, \sigma), \mu \in \mathbb{R}, \sigma \in \mathbb{R}_+\}$$

Learning goal

- Find a model P_θ that precisely captures the distribution P from which our data was sampled
- Intractability:
  - P(**x**) is in the space of all probability distributions
  - The sampled data set is limited
  - Computational reasons
- We want to select P_θ to be the "best" approximation to the underlying distribution P
  - **What is "best"?**
  - Depends on specific task of interest



Learning as density estimation

- We want to learn the full distribution so that later we can answer any probabilistic inference query
- In this setting we can view the learning problem as **density estimation**
- We want to construct P_θ as "**close**" as possible to P
  (recall we assume we are given a dataset **D** of samples from P)

How do we evaluate "closeness"?

KL-divergence

- How should we measure distance between distributions?
- The **Kullback-Leibler divergence** (KL-divergence) between two distributions P and Q is defined as

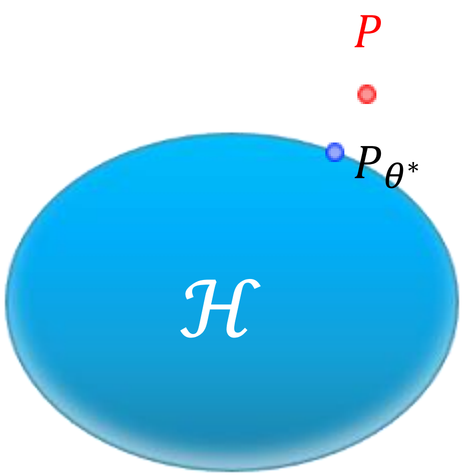
$$KL(P||Q) = \mathbb{E}_{\boldsymbol{x}\sim P(\boldsymbol{x})}\left(\log\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}\right)$$

  - where $\mathrm{p}(\boldsymbol{x})$ and $\mathrm{q}(\boldsymbol{x})$ represents the densities of P and Q, respectively
- Note that:
  - $KL(P||Q) \geq 0$ for any P and Q, and $KL(P||P) = 0$
  - $KL(P||Q) \neq KL(Q||P)$
- It measures the loss (in bits) when describing distribution P by Q.

Learning: a revisit

- We want to construct P_θ as **"close"** as possible to P (Given a dataset **D** of samples from P)
- Closeness by KL:

$$KL(P||P_\theta) = \mathbb{E}_{\boldsymbol{x}\sim P(\boldsymbol{x})}\left(\log \frac{p(\boldsymbol{x})}{p_\theta(\boldsymbol{x})}\right)$$



- Learning by minimizing $KL(P||P_\theta)$

$$\theta^* = \underset{\theta\in\Theta}{\operatorname{argmin}}\, KL(P||P_\theta)$$

  - Find the parameter θ^* that minimizes $KL(P||P_\theta)$
  - θ^* provides the minimal loss when compressing P by P_{θ^*}

Expected log-likelihood

- We can rewrite

$$KL(P||P_\theta) = \mathbb{E}_{\boldsymbol{x}\sim P(\boldsymbol{x})}\left(\log \frac{p(\boldsymbol{x})}{p_\theta(\boldsymbol{x})}\right) = \mathbb{E}_{\boldsymbol{x}\sim P(\boldsymbol{x})}(\log p(\boldsymbol{x})) - \mathbb{E}_{\boldsymbol{x}\sim P(\boldsymbol{x})}(\log p_\theta(\boldsymbol{x}))$$

  - The first term does not depend on θ
- Minimizing KL is equivalent to maximizing the *Expected log-likelihood* $\mathbb{E}_{\boldsymbol{x}\sim P(\boldsymbol{x})}(\log p_\theta(\boldsymbol{x}))$
- Learning can be done by **Maximum Likelihood Estimation (MLE)**

$$\theta^* = \operatorname*{argmax}_{\theta\in\Theta} \mathbb{E}_{\boldsymbol{x}\sim P(\boldsymbol{x})}(\log p_\theta(\boldsymbol{x}))$$

  - In general, we do not know P
  - So, we cannot access to the objective

Maximum likelihood

- We approximate the *expected log-likelihood* $\mathbb{E}_{\boldsymbol{x}\sim P(\boldsymbol{x})}(\log p_\theta(\boldsymbol{x}))$ by

$$\mathbb{E}_{\boldsymbol{x}\in \boldsymbol{D}}(\log p_\theta(\boldsymbol{x})) = \frac{1}{m}\sum_{\boldsymbol{x}\in \boldsymbol{D}} \log p_\theta(\boldsymbol{x})$$

  - Sometimes known as *Empirical log-likelihood* (note the similarity with empirical loss in ML)

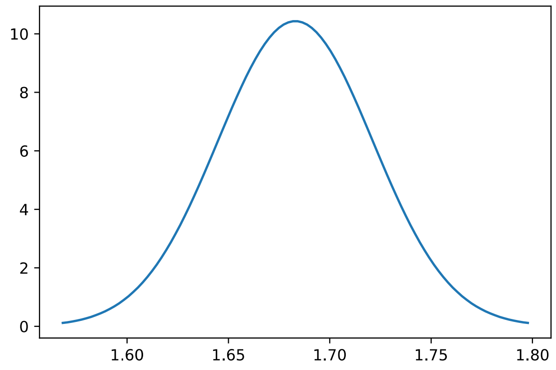
- MLE is the formulated as

$$\theta^* = \underset{\theta\in\Theta}{\operatorname{argmax}} \frac{1}{m}\sum_{\boldsymbol{x}\in \boldsymbol{D}} \log p_\theta(\boldsymbol{x})$$

  - This is equivalent to maximizing the likelihood $P(\boldsymbol{x}_1, \dots, \boldsymbol{x}_m) = \prod_{i=1}^{m} P(\boldsymbol{x}_i)$ for i.i.d. samples

MLE: Gaussian example (1)

- We wish to estimate the height of a person in the world.
- Use a dataset **D** = {1.6, 1.7, 1.65, 1.63, 1.75, 1.71, 1.68, 1.72, 1.77, 1.62}
  - Let x be the random variable representing the height of a person.
  - Model: assume that x follows a Gaussian distribution with ***unknown*** mean μ and variance σ^2
  - **Learning:** estimate (μ, σ) from the given data $\boldsymbol{D} = \{x_1, \dots, x_{10}\}$.
- Let $f(x|\mu, \sigma)$ be the density function of the Gaussian family, parameterized by (μ, σ).
  - $f(x_n|\mu, \sigma)$ is the likelihood of instance x_n.
  - $f(\boldsymbol{D}|\mu, \sigma)$ is the likelihood function of **D**.
- Using MLE, we will find



$$(\mu_*, \sigma_*) = \arg\max_{\mu, \sigma} f(\boldsymbol{D}|\mu, \sigma)$$

MLE: Gaussian example (2)

- **i.i.d assumption:** we assume that the data are independent and identically distributed (dữ liệu được sinh ra một cách độc lập)
  - As a result, we have $P(\boldsymbol{D}|\mu,\sigma) = P(x_1, \dots, x_{10}|\mu,\sigma) = \prod_{i=1}^{10} P(x_i|\mu,\sigma)$
- Using this assumption, MLE will be

$$(\mu_*, \sigma_*) = \arg\max_{\mu,\sigma} \prod_{i=1}^{10} f(x_i|\mu,\sigma) = \arg\max_{\mu,\sigma} \prod_{i=1}^{10} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(x_i-\mu)^2}$$

$$= \arg\max_{\mu,\sigma} \log \prod_{i=1}^{10} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(x_i-\mu)^2}$$

$$= \arg\max_{\mu,\sigma} \sum_{i=1}^{10} \left(-\frac{1}{2\sigma^2}(x_i-\mu)^2 - \log\sqrt{2\pi\sigma^2} \right)$$

Log trick, $\log \stackrel{\text{def}}{=} \ln$

- Using gradients (w.r.t μ, σ), we can find

$$\mu_* = \frac{1}{10}\sum_{i=1}^{10} x_i = 1.683, \qquad \sigma_*^2 = \frac{1}{10}\sum_{i=1}^{10} (x_i - \mu_*)^2 \approx 0.0015$$

Generative models

Approximation by mixture models

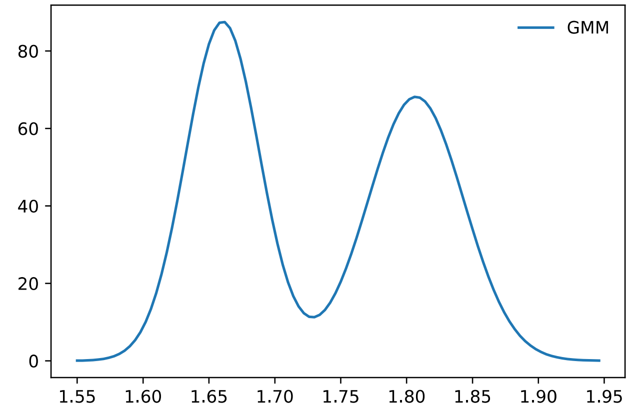
Learning the data distribution

- Dataset **D** = {$\mathbf{x}_1$, $\mathbf{x}_2$, …, $\mathbf{x}_m$}
  - Images about dogs
- **Hardness** of the learning problem:
  - P(**x**) is in the space of all probability distributions
- In practice, we often find a $P_\theta(\boldsymbol{x})$ to **approximate** $P(\boldsymbol{x})$
- How to choose a good model family?
  - Gaussian family?
    => too simple

Gaussian mixture model (GMM)

- GMM: we assume that the data are samples from K Gaussian distributions.
  - Each instance **x** is generated from one of those K Gaussians by the following ***generative process***:
    - *Take the component index* $z \sim Categorical(\boldsymbol{\phi})$
    - *Generate* $\boldsymbol{x} \sim Normal(\boldsymbol{\mu}_z, \boldsymbol{\Sigma}_z)$
- The density function is

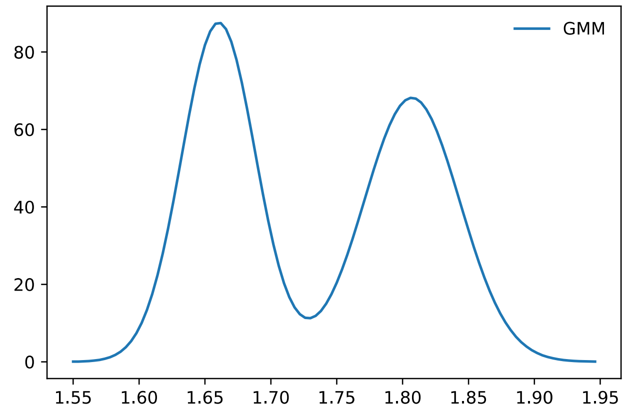
$$q(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\phi}) = \sum_{k=1}^{K} \phi_k \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$



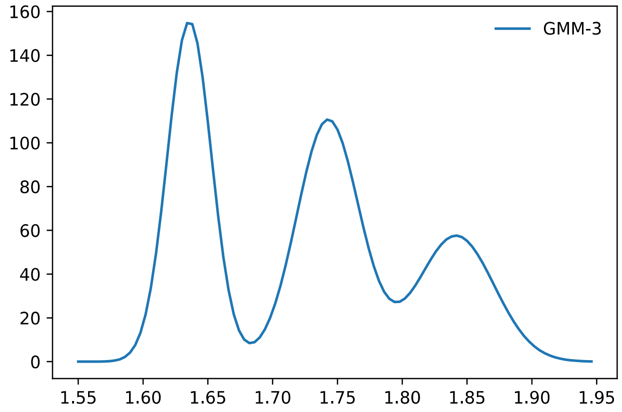
  - $\boldsymbol{\phi} = (\phi_1, \dots, \phi_K)$ represents the weights of the Gaussians: $\sum_{k=1}^{K} \phi_k = 1, \quad \phi_j \geq 0, \ \forall j$
  - Each Gaussian has density $\mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{\det(2\pi\boldsymbol{\Sigma})}} \exp\left[-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{x} - \boldsymbol{\mu})\right]$
- Note: z is an unobserved (latent) variable, x is observable

GMM: approximation ability

- The density $q(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\phi}) = \sum_{k=1}^{K} \phi_k \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$
  - Gaussian model: $K = 1$ component
- A larger K produces a more complex model Q



GMM with 2 components



GMM with 3 components

- **GMMs are universal approximators**
  - Any smooth density can be approximated arbitrarily well by a GMM with enough components

Dalal, S. R., and W. J. Hall. "Approximating Priors by Mixtures of Natural Conjugate Priors." J. of the Royal Statistical Society. Series B (Methodological), vol. 45, no. 2, 1983, pp. 278–286.

Infinite GMM

- *Mixture of an infinite number of Gaussians:* we assume that the data are samples from an infinite number of Gaussians
  - Each instance **x** is generated from one of those Gaussians by the following ***generative process***:
    - *Choose* $\boldsymbol{z} \sim Normal(0, \boldsymbol{I})$ P(**z**)
    - *Generate* $\boldsymbol{x} \sim Normal(\boldsymbol{\mu}_\theta(\boldsymbol{z}), \boldsymbol{\Sigma}_\theta(\boldsymbol{z}))$ P(**x** | **z**)
      - Where $\boldsymbol{\mu}_\theta, \boldsymbol{\Sigma}_\theta$ are neural networks, parameterized by θ
- **Universal approximator?**
- Each component is simple, but the marginal P(**x**) is very complex

Variational auto-encoder

Variational inference,
Amortized inference,
Sampling

Learning for GMM

- Learning by MLE:

$$\theta^* = \underset{\theta}{\operatorname{argmax}} \frac{1}{m} \sum_{\boldsymbol{x} \in \boldsymbol{D}} \log p_\theta(\boldsymbol{x})$$

  - where $p_\theta(\boldsymbol{x}) = \sum_{k=1}^{K} \phi_k \frac{1}{\sqrt{\det(2\pi\boldsymbol{\Sigma}_k)}} \exp\left[-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_k)\right]$, $\theta = (\boldsymbol{\phi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})$

- Evaluation of $\log p_\theta(\boldsymbol{x})$ is **hard** in general, since

$$\log p_\theta(\boldsymbol{x}) = \log \sum_{\text{All possible values of } \boldsymbol{z}} p_\theta(\boldsymbol{x}, \boldsymbol{z})$$

  - E.g., for $\boldsymbol{z} \in \{0,1\}^{100}$, the sum has 2^{100} terms

- It is even harder for more complex models

→ Approximation is needed

Evidence Lower Bound

- Note

$$\log p_\theta(\boldsymbol{x}) = \log \sum_{\boldsymbol{z}\in\mathcal{Z}} p_\theta(\boldsymbol{x},\boldsymbol{z}) = \log \sum_{\boldsymbol{z}\in\mathcal{Z}} \frac{q(\boldsymbol{z})}{q(\boldsymbol{z})} p_\theta(\boldsymbol{x},\boldsymbol{z}) = \log \mathbb{E}_{q(\boldsymbol{z})} \frac{p_\theta(\boldsymbol{x},\boldsymbol{z})}{q(\boldsymbol{z})}$$

- Since log is concave, Jensen Inequality suggests

$$\log \mathbb{E}_{q(\boldsymbol{z})} \frac{p_\theta(\boldsymbol{x},\boldsymbol{z})}{q(\boldsymbol{z})} \geq \mathbb{E}_{q(\boldsymbol{z})} \log \frac{p_\theta(\boldsymbol{x},\boldsymbol{z})}{q(\boldsymbol{z})} = \mathbb{E}_{q(\boldsymbol{z})} \log p_\theta(\boldsymbol{x},\boldsymbol{z}) - \mathbb{E}_{q(\boldsymbol{z})} \log q(\boldsymbol{z})$$

- This is called the Evidence Lower Bound (**ELBO**)
- For any $q(\boldsymbol{z})$

$$\log p_\theta(\boldsymbol{x}) \geq ELBO$$

  - For $\mathrm{ELBO} = \mathbb{E}_{q(\boldsymbol{z})} \log p_\theta(\boldsymbol{x},\boldsymbol{z}) - \mathbb{E}_{q(\boldsymbol{z})} \log q(\boldsymbol{z})$
- When $q(\boldsymbol{z}) = p_\theta(\boldsymbol{z}|\boldsymbol{x})$:

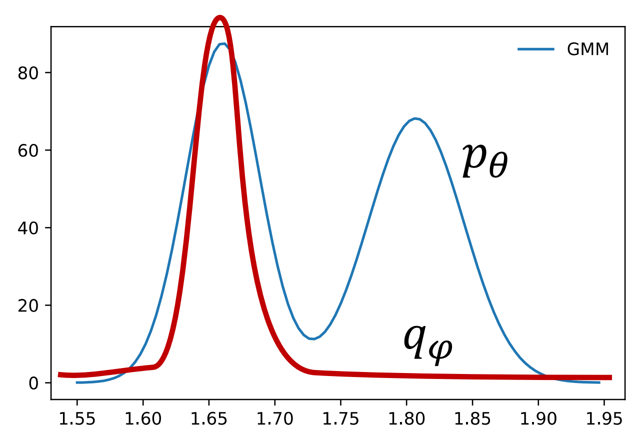
$$\log p_\theta(\boldsymbol{x}|\boldsymbol{\theta}) = \mathbb{E}_{p_\theta(\boldsymbol{z}|\boldsymbol{x})} \log p_\theta(\boldsymbol{x},\boldsymbol{z}|\boldsymbol{\theta}) - \mathbb{E}_{p_\theta(\boldsymbol{z}|\boldsymbol{x})} \log p_\theta(\boldsymbol{z}|\boldsymbol{x}) = \boldsymbol{ELBO}$$

Variational inference

- When the posterior $p_\theta(\boldsymbol{z}|\boldsymbol{x})$ is easy to compute, we can learn the model by maximizing

$$\frac{1}{m}\sum_{\boldsymbol{x}\in\boldsymbol{D}}\log p_\theta(\boldsymbol{x}) = \frac{1}{m}\sum_{\boldsymbol{x}\in\boldsymbol{D}}\left[\mathbb{E}_{p_\theta(\boldsymbol{z}|\boldsymbol{x})}\log p_\theta(\boldsymbol{x},\boldsymbol{z}|\boldsymbol{\theta}) - \mathbb{E}_{p_\theta(\boldsymbol{z}|\boldsymbol{x})}\log p_\theta(\boldsymbol{z}|\boldsymbol{x})\right]$$

  - E.g., for the case of GMM
- What if the posterior $p_\theta(\boldsymbol{z}|\boldsymbol{x})$ is intractable to compute?
- **Variational inference (VI):**
  - choose a family of *simple* distributions $q_\varphi(\boldsymbol{z})$, parameterized by φ (variational parameters)
  - then find φ^* so that $q_{\varphi^*}(\boldsymbol{z})$ is as close as possible to $p_\theta(\boldsymbol{z}|\boldsymbol{x})$



VI and KL

- Maximize the ELBO

$$\frac{1}{m}\sum_{i=1}^{m}\left[\mathbb{E}_{q_{\varphi_i}(\boldsymbol{z})}\log p_\theta(\boldsymbol{x}_i,\boldsymbol{z}|\boldsymbol{\theta}) - \mathbb{E}_{q_{\varphi_i}(\boldsymbol{z})}\log q_{\varphi_i}(\boldsymbol{z})\right]$$

  - given a training set **D** = {$\mathbf{x}_1$, $\mathbf{x}_2$, …, $\mathbf{x}_m$}
- Maximizing ELBO is equivalent to Minimizing KL, due to

$$\log p_\theta(\boldsymbol{x}) = ELBO + KL(q_\varphi(\boldsymbol{z})||p_\theta(\boldsymbol{z}|\boldsymbol{x}))$$

- Jointly optimize over
  - $\varphi_1, \ldots, \varphi_m$ (variational parameters)
  - θ (model parameters)

VI: some properties

- Pros:
  - Easy to be used in a large class of models
  - Efficient in practice
- Cons:
  - Hard to choose a good variational family
    - When we do not know the explicit form for the posterior $p_\theta(\boldsymbol{z}|\boldsymbol{x})$
  - For *inference*, given model param θ and instance $\boldsymbol{x}$, we estimate the posterior $p_\theta(\boldsymbol{z}|\boldsymbol{x})$ *by solving an optimization problem:*

$$\max_{\varphi} \mathbb{E}_{q_\varphi(\boldsymbol{z})} \log p_\theta(\boldsymbol{x}, \boldsymbol{z}|\boldsymbol{\theta}) - \mathbb{E}_{q_\varphi(\boldsymbol{z})} \log q_\varphi(\boldsymbol{z})$$

Expensive

  - Require too many variational parameters
    - Each instance $\mathbf{x}_i$ requires one specific φ_i ➔ O(m) parameters
    - GMM needs $O(mKn^2)$ params, where K is #components, n is #dims

- **Variational inference (VI):**
  - choose a family of *simple* distributions $q_\varphi(\boldsymbol{z})$, parameterized by φ
  - find φ^* so that $q_{\varphi^*}(\boldsymbol{z})$ is as close as possible to $p_\theta(\boldsymbol{z}|\boldsymbol{x})$

Amortized inference

$$\max_{\theta} \frac{1}{m} \sum_{\boldsymbol{x} \in \boldsymbol{D}} \log p_{\theta}(\boldsymbol{x}) \geq \max_{\theta, \varphi_1, \ldots, \varphi_m} \frac{1}{m} \sum_{\boldsymbol{x}_i \in \boldsymbol{D}} \mathrm{L}(\boldsymbol{x}_i; \theta, \varphi)$$

- Where $\mathrm{L}(\boldsymbol{x}_i; \theta, \varphi) = \mathbb{E}_{q_{\varphi_i}(\boldsymbol{z})} \log p_{\theta}(\boldsymbol{x}_i, \mathbf{z} | \boldsymbol{\theta}) - \mathbb{E}_{q_{\varphi_i}(\boldsymbol{z})} \log q_{\varphi_i}(\boldsymbol{z})$

- VI uses φ_i for each point $\mathbf{x}_i$.
  - May not scale well with large datasets; prone to overfitting
- **Amortization:** *we learn a **single** neural network $f_w: \boldsymbol{x} \mapsto \varphi$ that maps each input **x** to a set of (good) variational parameters*
  - f_w has a *trainable* parameter w
  - For a given input $\mathbf{x}_i$, f_w will produce the parameter $\varphi_i = f_w(\boldsymbol{x}_i)$ of the variational distribution $q_{\varphi_i}(\boldsymbol{z})$
- *Amortized inference:* feed instance **x** to the trained network to get the variational parameter $\varphi = f_w(\boldsymbol{x})$
  - No optimization ➔ cheap

Kingma, D. P. & Welling, M. (2014). Auto-Encoding Variational Bayes. *ICLR*.

Learning with amortized inference

- We can use using stochastic gradient descent to solve

$$\max_{\theta,\varphi_1,\ldots,\varphi_m} \sum_{\boldsymbol{x}_i \in \boldsymbol{D}} \mathrm{L}(\boldsymbol{x}_i; \theta, \varphi)$$

- *Initialize $\theta^{(0)}, \varphi^{(0)}$*
- *At iteration $j \geq 1$:*
  - *Randomly sample a data point $\mathbf{x}_i$ from **D***
  - *Compute $\nabla_\theta L\left(\boldsymbol{x}_i; \theta^{(j-1)}, \varphi^{(j-1)}\right)$ and $\nabla_\varphi L\left(\boldsymbol{x}_i; \theta^{(j-1)}, \varphi^{(j-1)}\right)$*
  - *Update $\theta^{(j)}, \varphi^{(j)}$ in the gradient direction*

- How to compute the gradients?
  - $\mathrm{L}(\boldsymbol{x}_i; \theta, \varphi) = \mathbb{E}_{q_{\varphi_i}(\boldsymbol{z})} \log p_\theta(\boldsymbol{x}_i, \boldsymbol{z} | \boldsymbol{\theta}) - \mathbb{E}_{q_{\varphi_i}(\boldsymbol{z})} \log q_{\varphi_i}(\boldsymbol{z})$
  - The expectation complicates gradient computation for φ

Reparameterization trick

- Consider **z** being **continuous**, and we want to compute a gradient with respect to φ of

$$\mathbb{E}_{q_\varphi(\boldsymbol{z})}[r(\boldsymbol{z})] = \int q_\varphi(\boldsymbol{z}) r(\boldsymbol{z}) d\boldsymbol{z}$$

- Suppose $q_\varphi(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{\mu}, \sigma^2 \boldsymbol{I})$ is Gaussian with parameters $\varphi = (\boldsymbol{\mu}, \sigma)$
  - Since $\boldsymbol{z} \sim q_\varphi(\boldsymbol{z})$, there exists representation $\boldsymbol{z} = \boldsymbol{\mu} + \sigma\boldsymbol{\epsilon}$ where $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \boldsymbol{I})$
- We can write

$$\mathbb{E}_{\boldsymbol{z} \sim q_\varphi(\boldsymbol{z})}[r(\boldsymbol{z})] = \mathbb{E}_{\boldsymbol{\epsilon} \sim \mathcal{N}(0, \boldsymbol{I})}[r(\boldsymbol{\mu} + \sigma\boldsymbol{\epsilon})]$$

$$\nabla_\varphi \mathbb{E}_{q_\varphi(\boldsymbol{z})}[r(\boldsymbol{z})] = \nabla_\varphi \mathbb{E}_{\boldsymbol{\epsilon}}[r(\boldsymbol{\mu} + \sigma\boldsymbol{\epsilon})] = \mathbb{E}_{\boldsymbol{\epsilon}}\left[\nabla_\varphi r(\boldsymbol{\mu} + \sigma\boldsymbol{\epsilon})\right]$$

- Easy to estimate via Monte Carlo if r is differentiable w.r.t. φ, since $\boldsymbol{\epsilon}$ is easy to sample
  - $\mathbb{E}_{\boldsymbol{\epsilon}}\left[\nabla_\varphi r(\boldsymbol{\mu} + \sigma\boldsymbol{\epsilon})\right] \approx \frac{1}{K}\sum_{j=1}^{K} \nabla_\varphi r\left(\boldsymbol{\mu} + \sigma\boldsymbol{\epsilon}_j\right),$ where $\boldsymbol{\epsilon}_1, \ldots, \boldsymbol{\epsilon}_K \sim \mathcal{N}(0, \boldsymbol{I})$

Variational auto-encoder (VAE)

- Since $q_\varphi(\boldsymbol{z})$ approximates the posterior $p_\theta(\boldsymbol{z}|\boldsymbol{x})$, we can write it as $q_\varphi(\boldsymbol{z}|\boldsymbol{x})$ and

$$
\begin{aligned}
\mathrm{L}(\boldsymbol{x};\theta,\varphi) &= \mathbb{E}_{q_\varphi(\boldsymbol{z}|\boldsymbol{x})} \log p_\theta(\boldsymbol{x},\boldsymbol{z}|\theta) - \mathbb{E}_{q_\varphi(\boldsymbol{z}|\boldsymbol{x})} \log q_\varphi(\boldsymbol{z}|\boldsymbol{x}) \\
&= \mathbb{E}_{q_\varphi(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x},\boldsymbol{z}|\theta) - \log p_\theta(\boldsymbol{z}) + \log p_\theta(\boldsymbol{z}) - \log q_\varphi(\boldsymbol{z}|\boldsymbol{x})\right] \\
&= \mathbb{E}_{q_\varphi(\boldsymbol{z}|\boldsymbol{x})}[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})] - KL(q_\varphi(\boldsymbol{z}|\boldsymbol{x}) \,||p_\theta(\boldsymbol{z}))
\end{aligned}
$$

- Maximize L: maximize $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ and push $q_\varphi(\boldsymbol{z}|\boldsymbol{x})$ close to $p_\theta(\boldsymbol{z})$
- **Encoder:**
  - Maps each data point $\boldsymbol{x}$ to a latent vector $\hat{\boldsymbol{z}}$, a sample from a Gaussian $(q_\varphi(\boldsymbol{z}|\boldsymbol{x}))$ with parameter $(\mu,\sigma) = Encoder_\varphi(\boldsymbol{x})$
- **Decoder:**
  - Reconstruct $\hat{\boldsymbol{x}}$ from a latent vector $\hat{\boldsymbol{z}}$, i.e., pick a sample from a Gaussian $(p_\theta(\boldsymbol{x}|\hat{\boldsymbol{z}}))$ with parameter $Decoder_\theta(\hat{\boldsymbol{z}})$

Kingma, D. P. & Welling, M. (2014). Auto-Encoding Variational Bayes. *ICLR*.

VAE

$$L(\boldsymbol{x};\theta,\varphi) = \mathbb{E}_{q_\varphi(\boldsymbol{z}|\boldsymbol{x})}[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})] - KL(q_\varphi(\boldsymbol{z}|\boldsymbol{x})\,||p_\theta(\boldsymbol{z}))$$

- Maximizing L:
  - The first term encourages accurate reconstruction $\hat{\boldsymbol{x}} \approx \boldsymbol{x}$
  - The KL term encourages $\hat{\boldsymbol{z}}$ to have a distribution similar to the prior $p_\theta(\boldsymbol{z})$
- Training: SGD + reparameterization trick

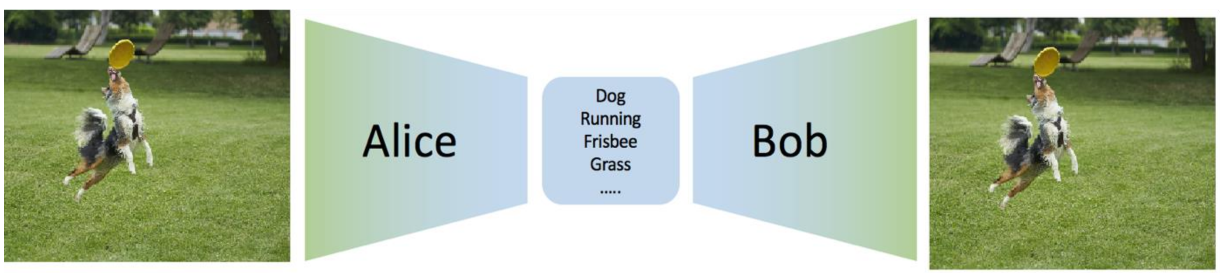


Image from Stefano Ermon

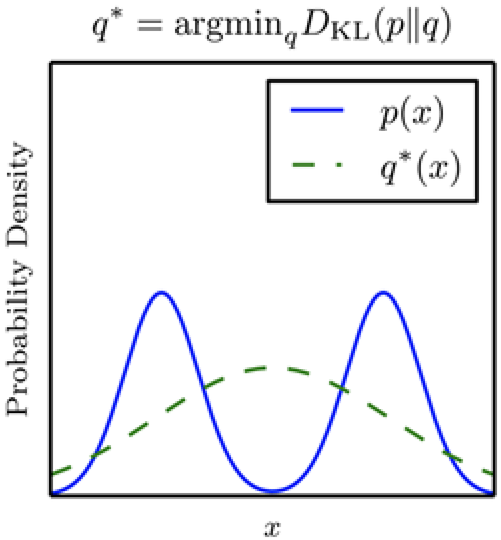
$$\boldsymbol{x} \xrightarrow{Encoder_\varphi} \hat{\boldsymbol{z}} \xrightarrow{Decoder_\theta} \hat{\boldsymbol{x}}$$

$p_\theta(\boldsymbol{x})$ $\quad$ $q_\varphi(\boldsymbol{z}|\boldsymbol{x})$ $\quad$ $p_\theta(\boldsymbol{x}|\boldsymbol{z})$

VAE: some properties

- Pros:
  - Efficient inference
  - Flexible and expressive (Universal approximator)
  - Good diversity of the synthetic samples
- Cons:
  - Blur images

VAE (2014)



Maximum likelihood

VQ-VAE (2017)

Generative Adversarial Networks

Introduction

Generative Adversarial Networks

$$\min_G \max_D \mathbb{E}_{\boldsymbol{x}\sim p_{data}}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}\left[\log\left(1 - D(G(\boldsymbol{z}))\right)\right]$$

- Two player minimax game between generator (G) and discriminator (D)
- D tries to maximize the log-likehood for the binary classification problem (D cố gắng cực đại hoá hàm log-likehood của bài toán phân loại nhị phân)
  - Data: real (1)
  - Generated: fake (0)
- G tries to minimize the log-probability of its samples being classified as “fake” by the discriminator D
  (G cố gắng cực tiểu hoá xác suất để D phân loại chính xác các mẫu dữ liệu do G tạo ra)

Generative Adversarial Networks

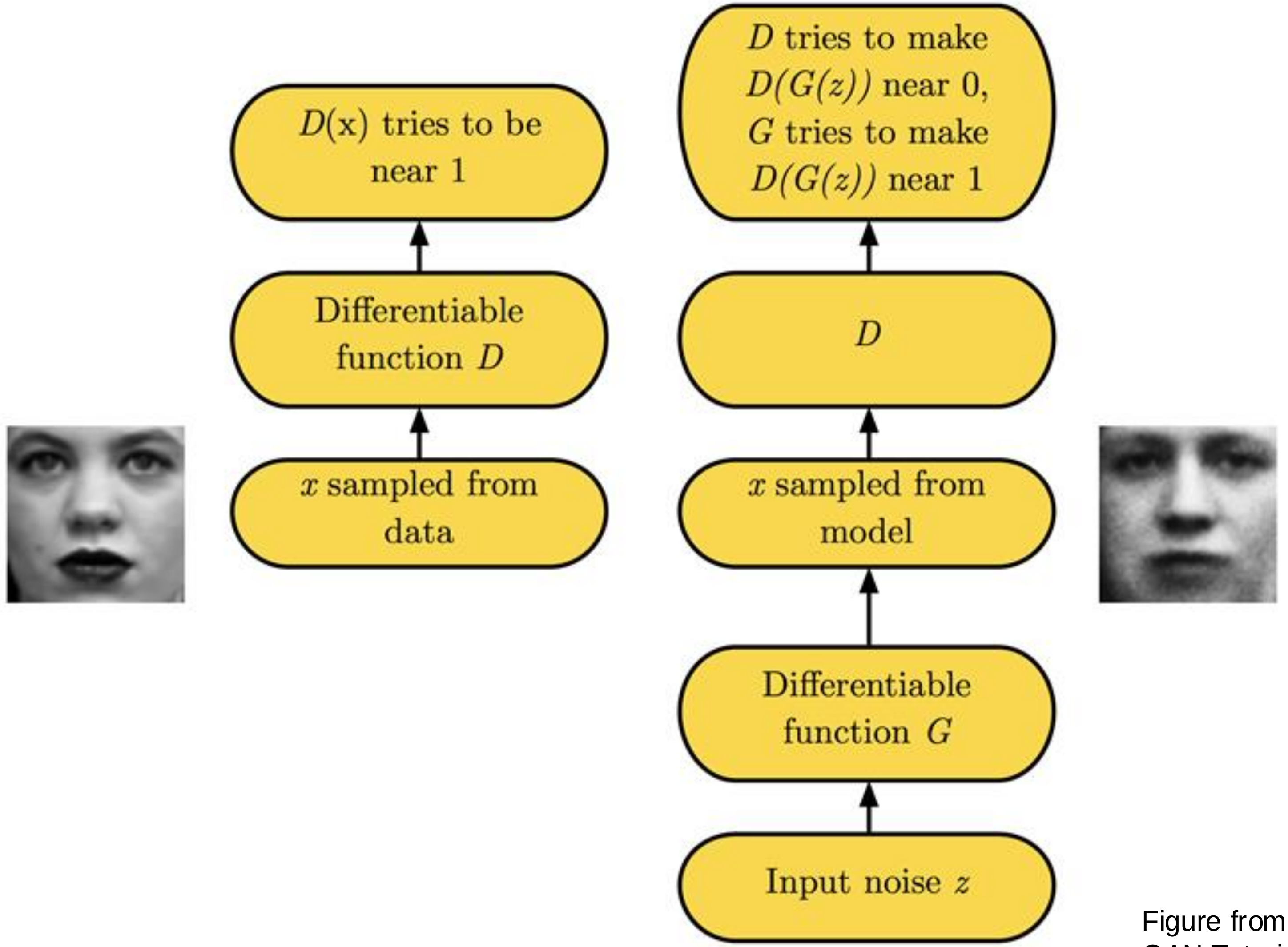


Figure from NeurIPS 2016 GAN Tutorial (Goodfellow)

Representation for the players

- D and G can be represented as two neural networks
- Discriminator:

$$D(\boldsymbol{x}) = NN(\boldsymbol{x}; \theta_d)$$

  - θ_d is the weight of the neural network which takes a sample ***x*** as input.
  - Output is a value in [0, 1].
    (biểu diễn D bằng một mạng nơron với trọng số θ_d, với đầu vào **x** thì trả về một giá trị thuộc [0, 1])
- Generator:

$$G(\boldsymbol{z}) = NN(\boldsymbol{z}; \theta_g)$$

  - θ_g is the weight of the neural network which takes a noise ***z*** as input.
  - ***z*** often follows a simple distribution, and is of low dimensionality.
  - Output is a fake sample $\boldsymbol{x} = G(\boldsymbol{z})$.
    (biểu diễn G bằng một mạng nơron với trọng số θ_g, với đầu vào **z** thì trả về một mẫu dữ liệu **x**)

GANs: pseudocode for training

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

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

D

  end for

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

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

G

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

[Goodfellow et al., NeurIPS 2014]

GAN

- See it in action: http://poloclub.github.io/ganlab/

GAN samples from 2014

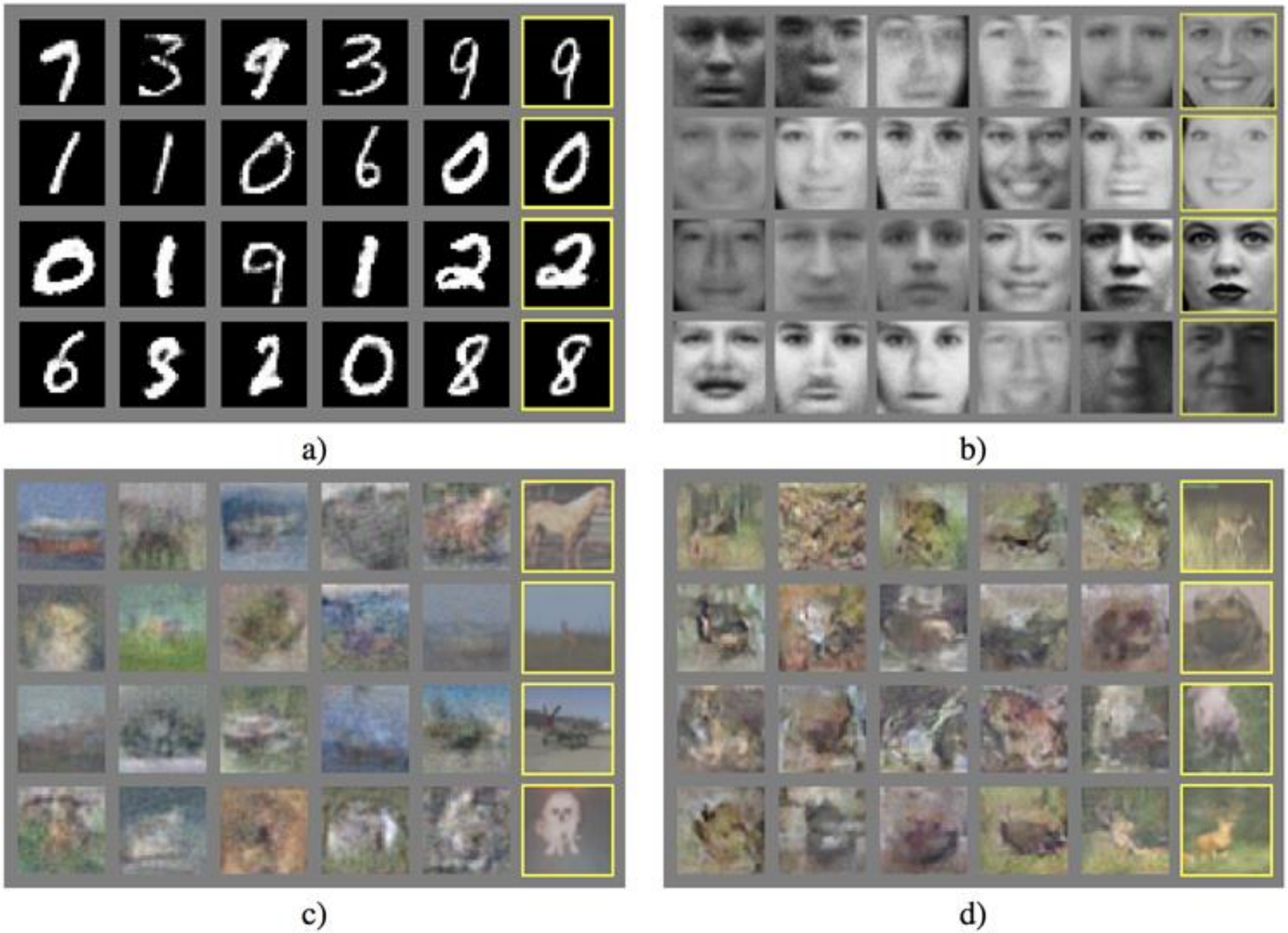


Figure from [Goodfellow et al., NeurIPS 2014]

Generative Adversarial Networks

- Key pieces of GAN
  - Fast sampling
  - No inference
  - Notion of optimizing directly for what you care about – perceptual samples

GAN: Bayes optimal discriminator

- What's the optimal discriminator given generated and true distributions?

$$V(G,D) = \mathbb{E}_{x \sim p_{data}}[\log D(x)] + \mathbb{E}_{z \sim p(z)}\left[\log\left(1 - D\big(G(z)\big)\right)\right]$$

$$= \int_x p_{data}(x) \log D(x)\, dx + \int_z p(z) \log\left(1 - D\big(G(z)\big)\right) dz$$

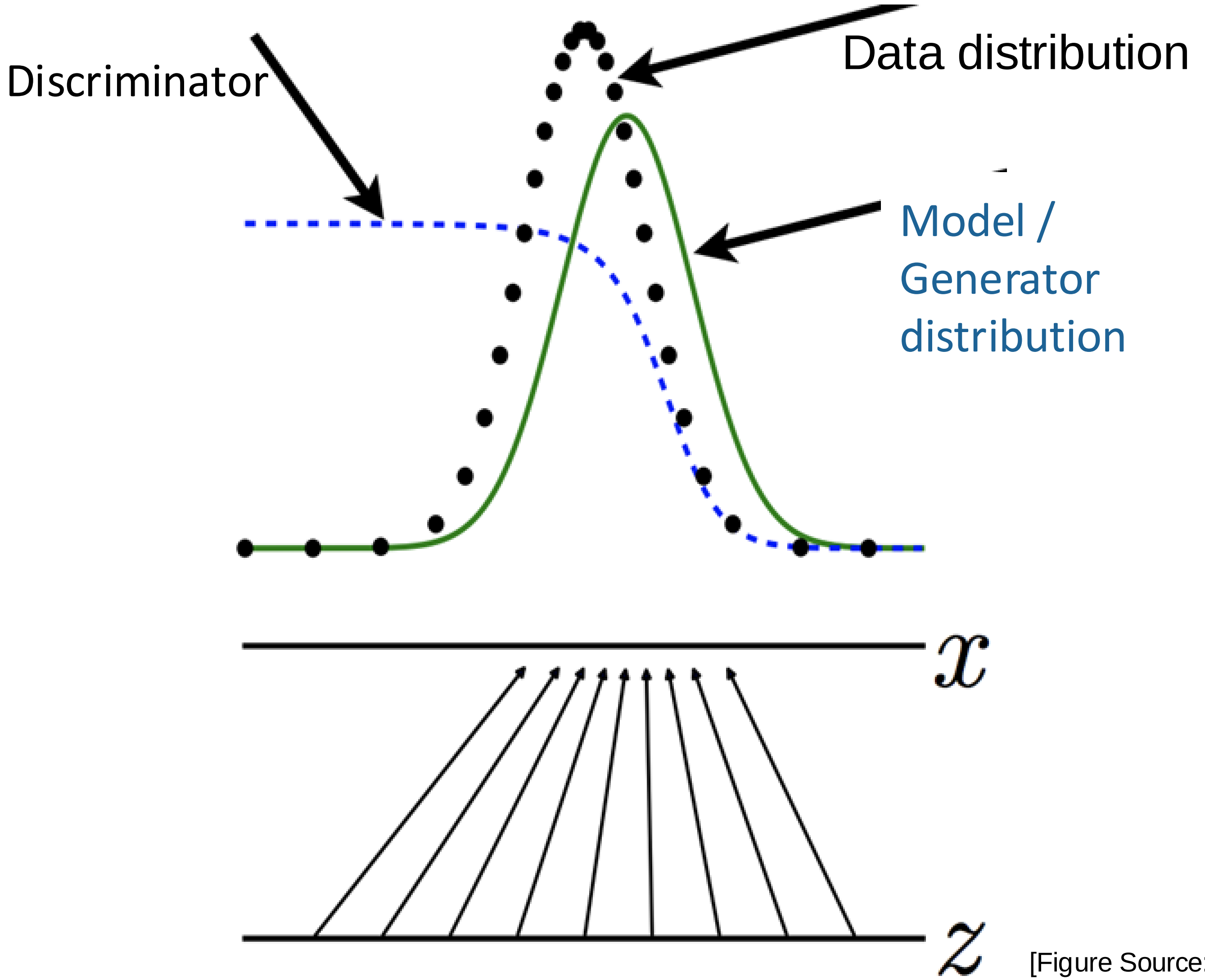
$$= \int_x p_{data}(x) \log D(x)\, dx + \int_x p_g(x) \log\big(1 - D(x)\big)\, dx$$

$$= \int_x \left[\, p_{data}(x) \log D(x) + p_g(x) \log\big(1 - D(x)\big)\right] dx$$

$$\nabla_y [a \log y + b \log(1-y)] = 0 \implies y^* = \frac{a}{a+b} \quad \forall\ (a,b) \in \mathbb{R}^2 \backslash (0,0)$$

$$\implies D^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}$$

GAN: Bayes optimal discriminator



[Figure Source: Goodfellow NeurIPS 2016 Tutorial on GANs]

Generator Objective under Optimal Discriminator

$$V(G, D^*) = \mathbb{E}_{x \sim p_{data}}[\log D^*(x)] + \mathbb{E}_{z \sim p_g}\left[\log\left(1 - D^*(x)\right)\right]$$

$$= \mathbb{E}_{x \sim p_{data}}\left[\log \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}\right] + \mathbb{E}_{z \sim p_g}\left[\log \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}\right]$$

$$= -\log(4) + \underbrace{KL\left(p_{data} \parallel \frac{p_{data} + p_g}{2}\right) + KL\left(p_g \parallel \frac{p_{data} + p_g}{2}\right)}_{\left(\text{Jensen–Shannon Divergence (JSD) of } p_{data} \text{ and } p_g\right) \geq 0}$$

(KL(p||q) is the Kullback-Leibler divergence between p and q)

$$V(G^*, D^*) = -\log(4) \;\; \text{when} \;\; p_g = p_{data}$$

- Given the Bayes-optimal D*, solving for G is equivalent to minimizing the JSD divergence between p_{data} and p_g

Behaviors across divergence measures

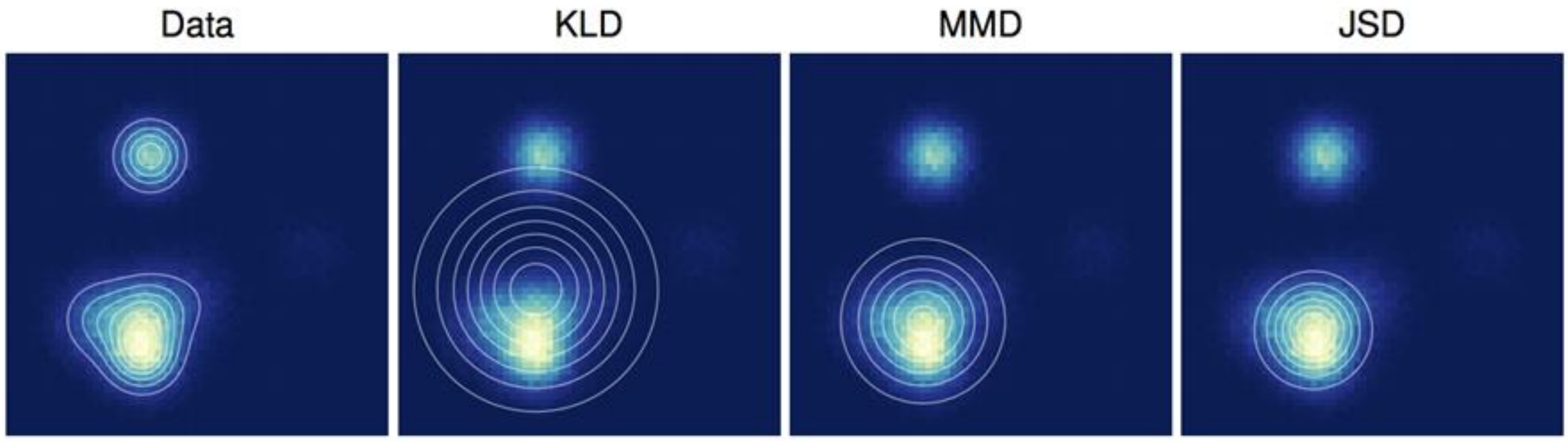
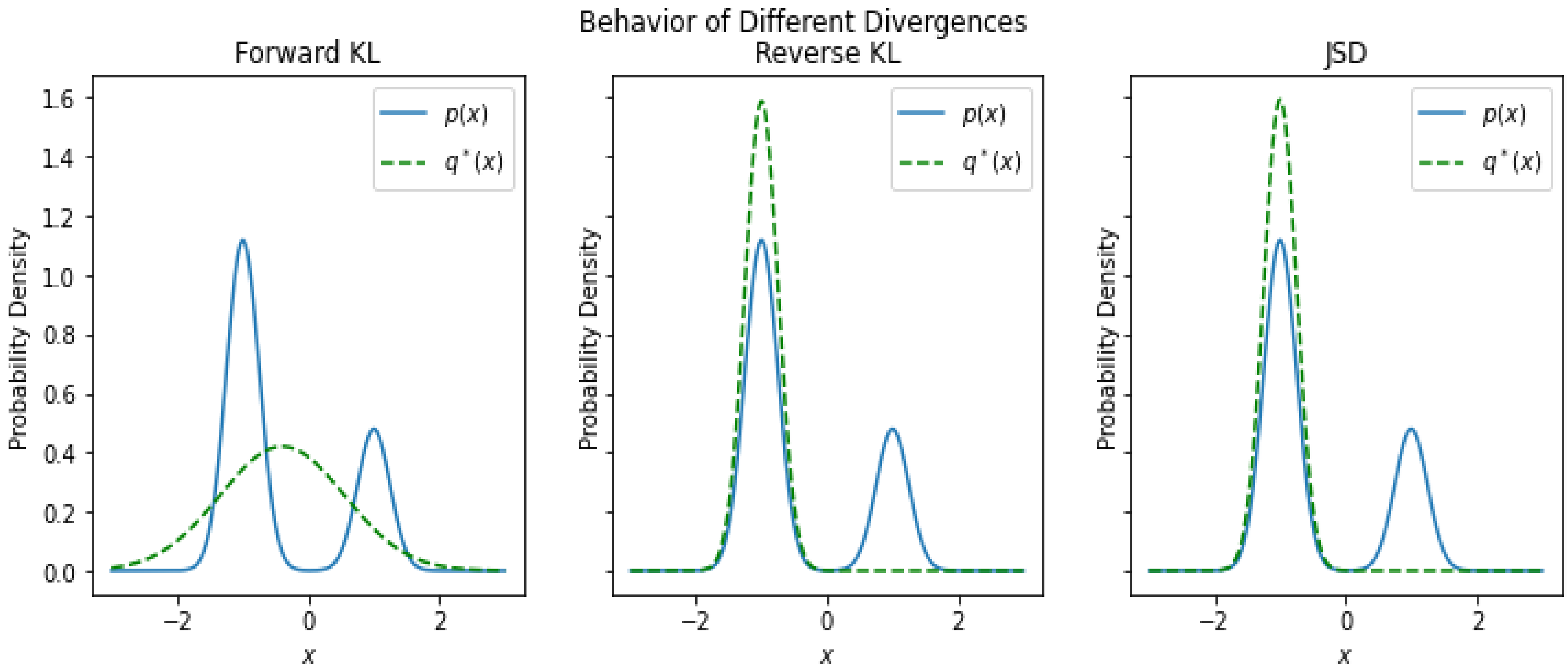


Figure 1: An isotropic Gaussian distribution was fit to data drawn from a mixture of Gaussians by either minimizing Kullback-Leibler divergence (KLD), maximum mean discrepancy (MMD), or Jensen-Shannon divergence (JSD). The different fits demonstrate different tradeoffs made by the three measures of distance between distributions.

[“A note on the evaluation of generative models” -- Theis, Van den Oord, Bethge 2015]

KL and JSD

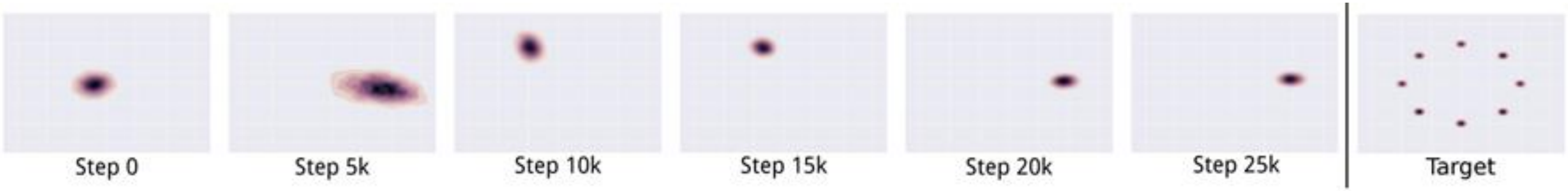


For given $p(x)$, find $q^*(x)$ that minimizes the divergence between them

Mode covering vs Mode seeking: Tradeoffs

- For compression, one would prefer to ensure all points in the data distribution are assigned probability mass.
- For generating good samples, blurring across modes spoils perceptual quality because regions outside the data manifold are assigned non-zero probability mass.
- Picking one mode without assigning probability mass on points outside can produce “better-looking” samples.
- Caveat: More expressive density models can place probability mass more accurately.

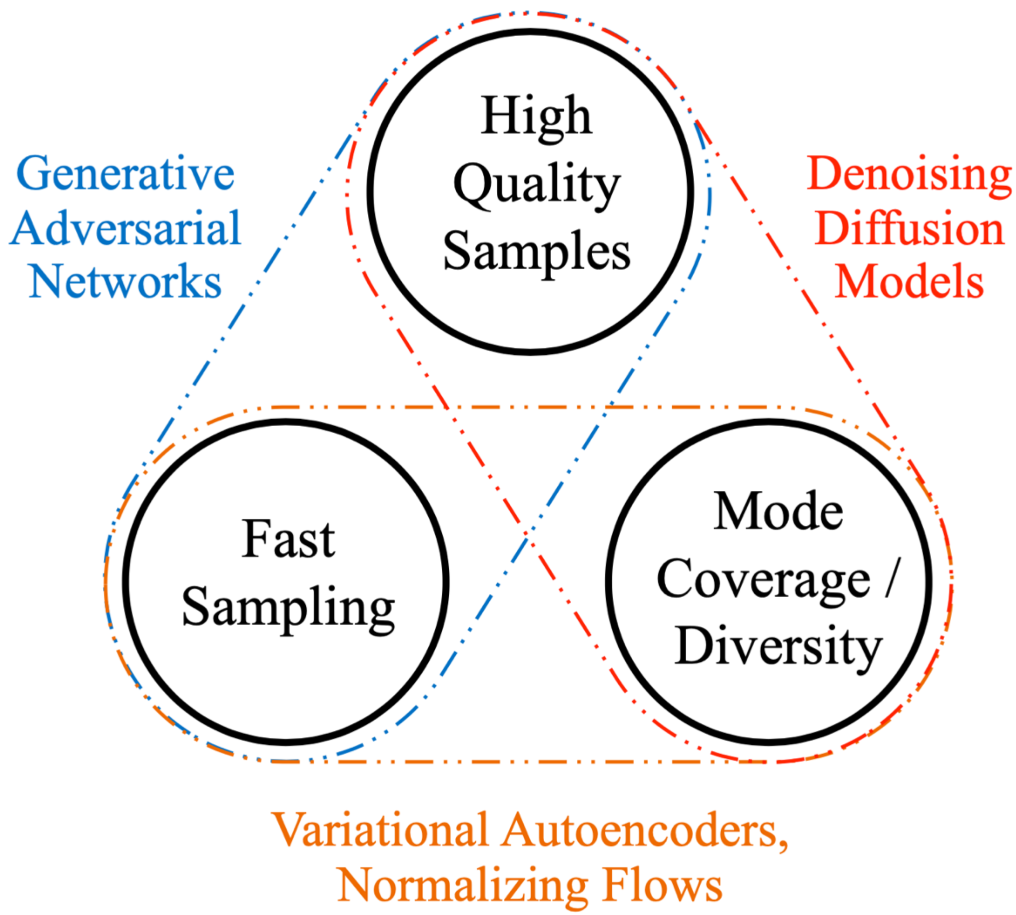
Mode Collapse



Standard GAN training collapses when the true distribution is a mixture of gaussians (Figure from Metz et al 2016)

More?

- Diffusion models
- …



Xiao, Z., Kreis, K., & Vahdat, A. Tackling the Generative Learning Trilemma with Denoising Diffusion GANs. In *ICLR, 2022*.

Thank you

Contact:
khoattq@soict.hust.edu.vn