Lecture 12: Deep Generative Models I: Overview, Latent Variable Models & EM

Lan Xu
SIST, ShanghaiTech
Fall, 2023

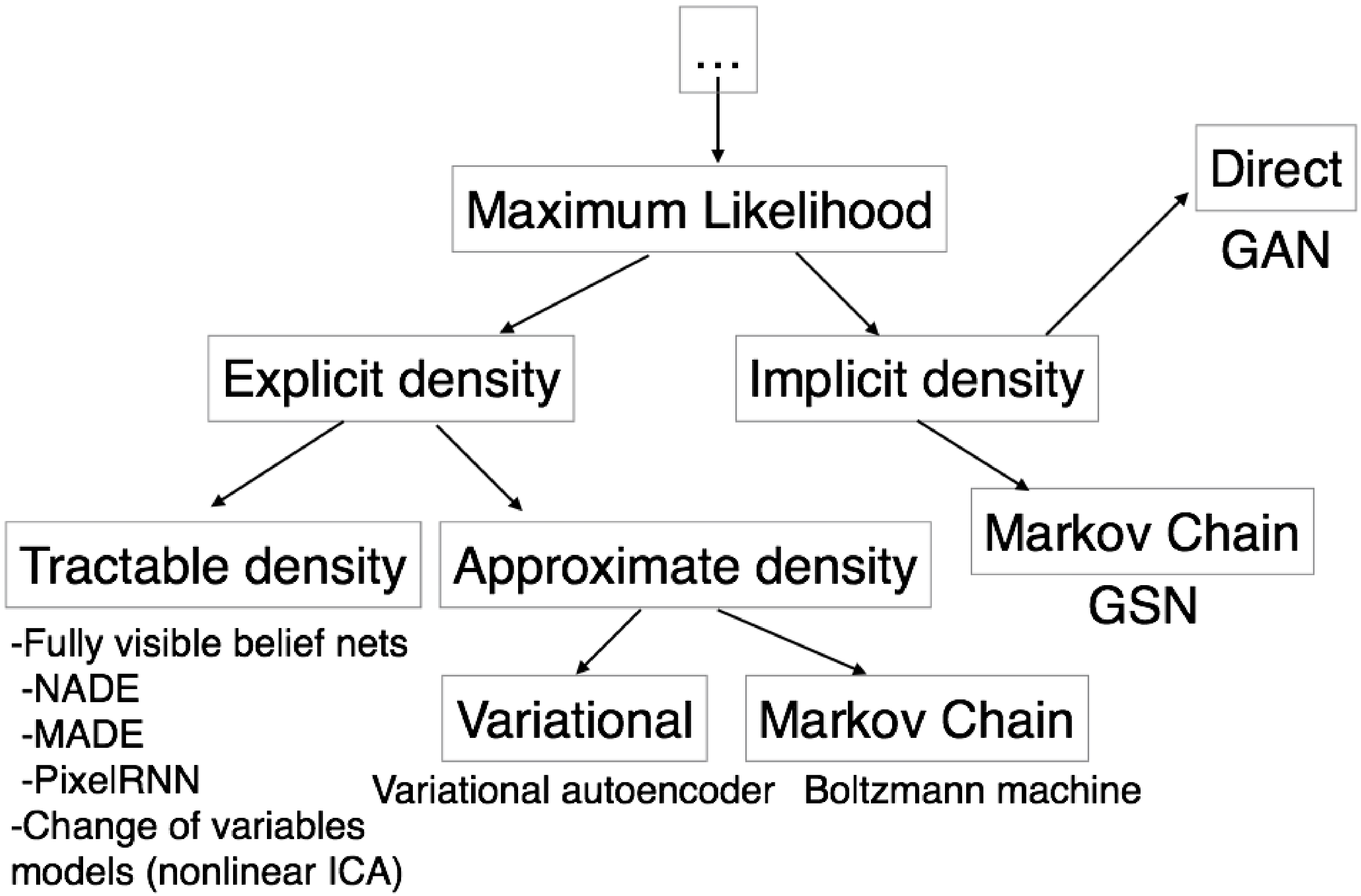
Outline

- Deep Generative Model Overview
- Unsupervised learning
  - Problem setup
- Latent variable model
  - EM algorithm and GMM
- Representation learning
  - AutoEncoder

Acknowledgement: Yingyu Liang@Princeton's & Feifei Li's cs231n notes

Taxonomy of Generative Models

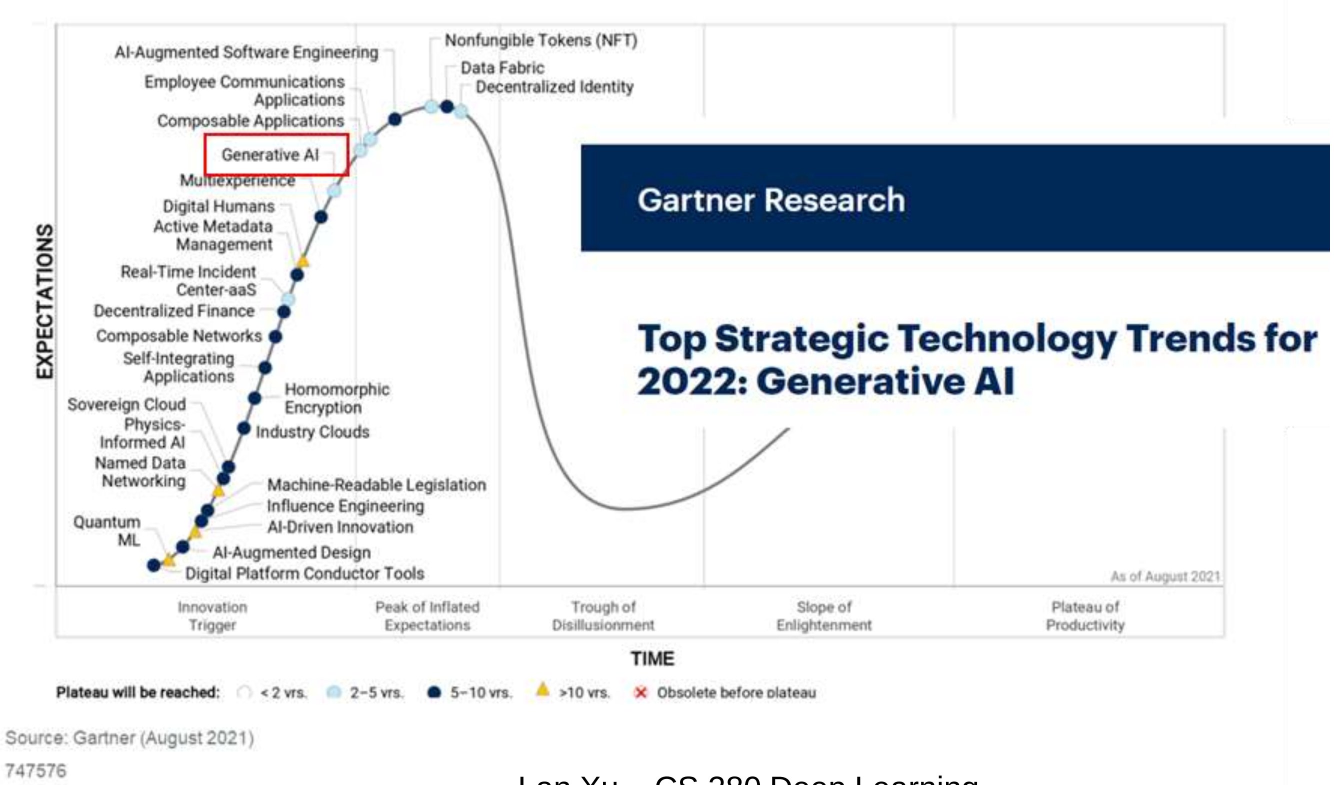
- Still happening ……



Which Face is Real?

Why Generative Model

- The new trends: big data/model/computing
- Inherent representation from data
- Enrich the data, accelerate content generation
- May help further AGI



Why Generative Model

- Debiasing: capable of uncovering underlying features in a dataset

Homogeneous skin color, pose

Diverse skin color, pose, illumination

Why Generative Model

- Multi-modalities, deeply influence our life

Why Generative Model

- Huge success for text-2-image
- Midjourney, Stable Diffusion, ……

a teddy bear on a skateboard in times square

hyperdetailed photography, photorealistic. close look, on a Canon EOS 5D Mark IV camera

Why Generative Model

- Huge success for text-2-image
- A lot of variants

鹤立鸡群

熊熊烈火

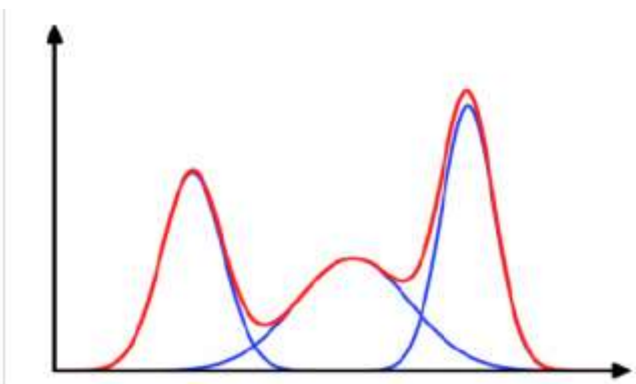
驴肉火烧

唐伯虎点秋香

Generative Models Overview

- Goal: Take as input training samples from some distribution → learn a model to represent distribution

Density Estimation



Sample Generation

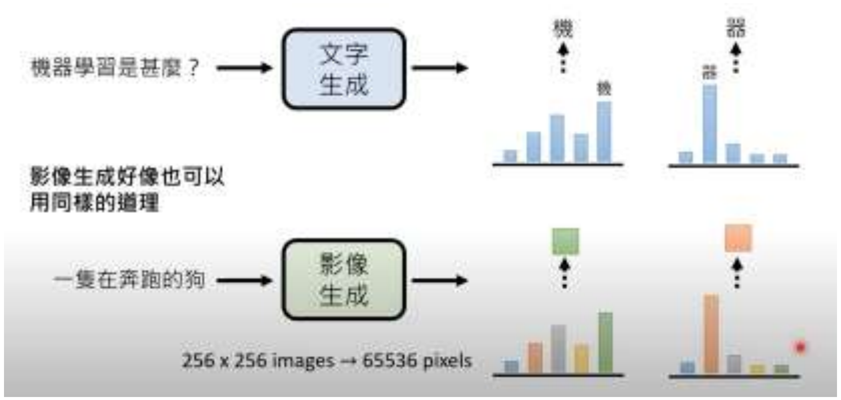
Input Samples:
P_data(x)

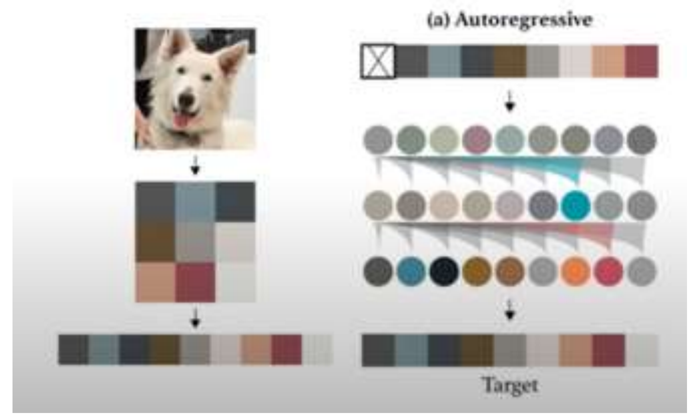
Generated Samples:
P_model(x)

Generative Models Overview

- Diffusion model
- Autoregressive: from text-gen to image-gen

A dog is running on the street.



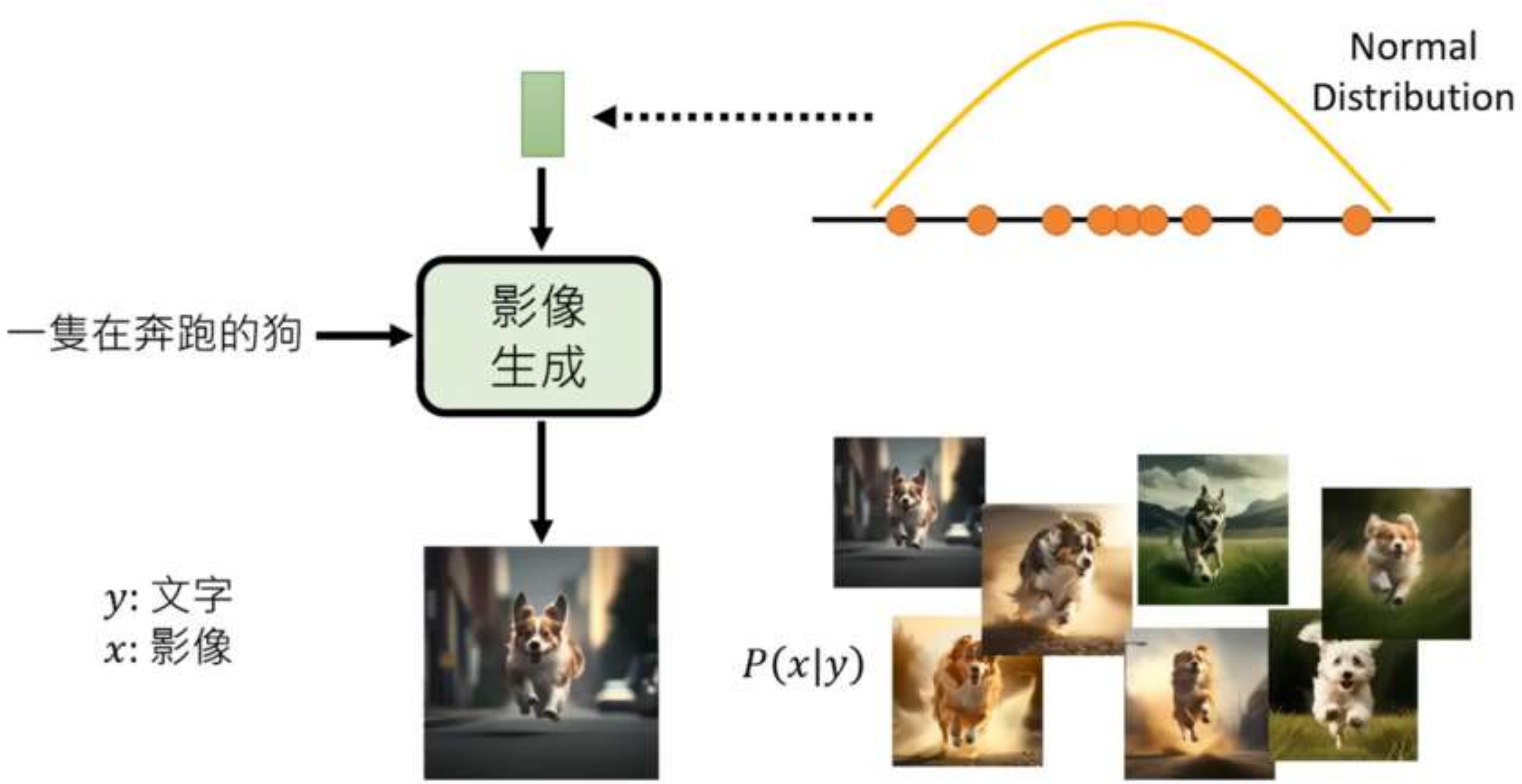


https://openai.com/research/image-gpt

From Prof. Hung-yi Lee, Generative AI

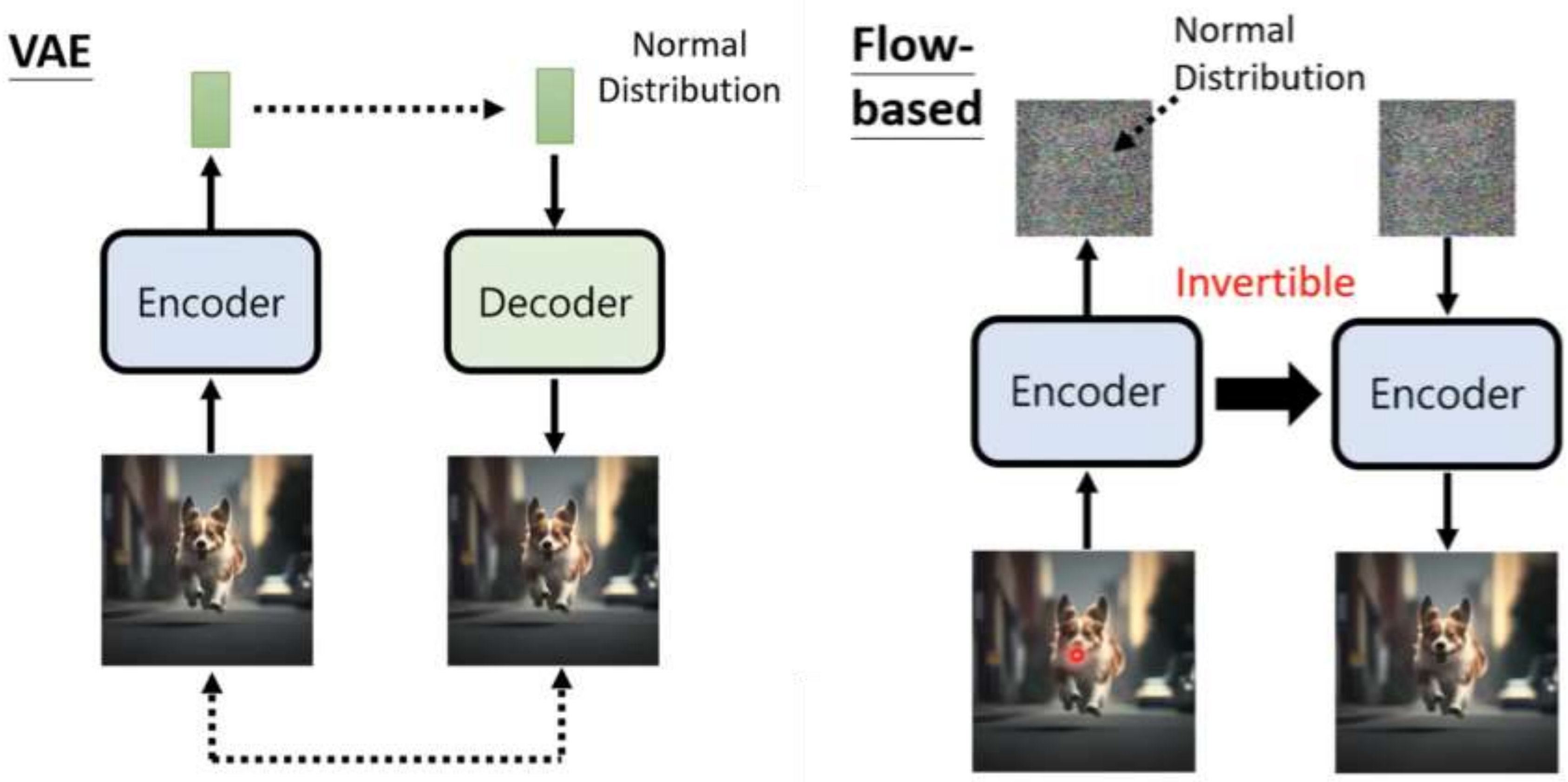
Generative Models Overview

- From normal distribution to specific image distribution



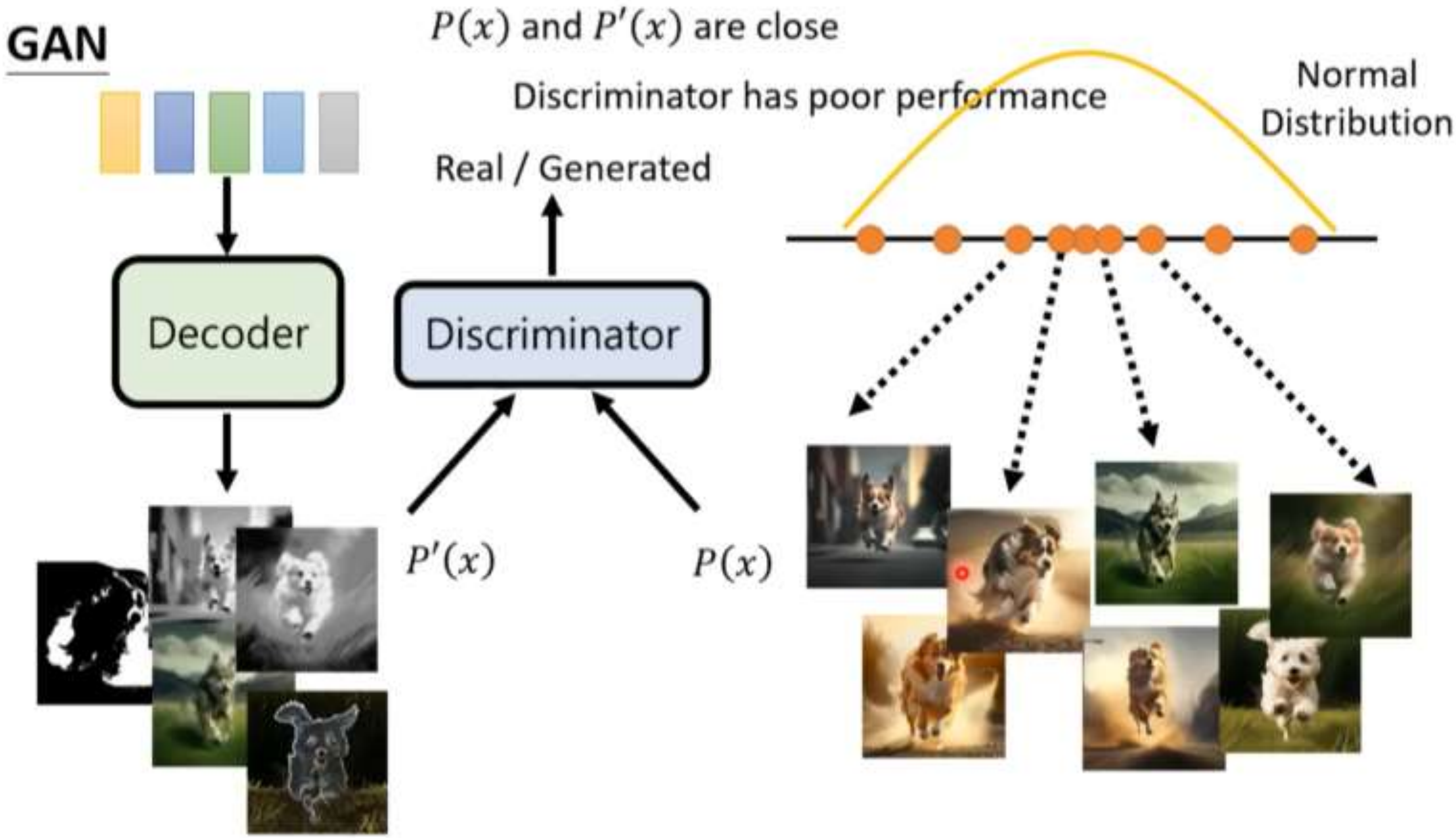
Generative Models Overview

- Modeling the distribution: VAE, GAN, normalizing flow (NF), **Diffusion**



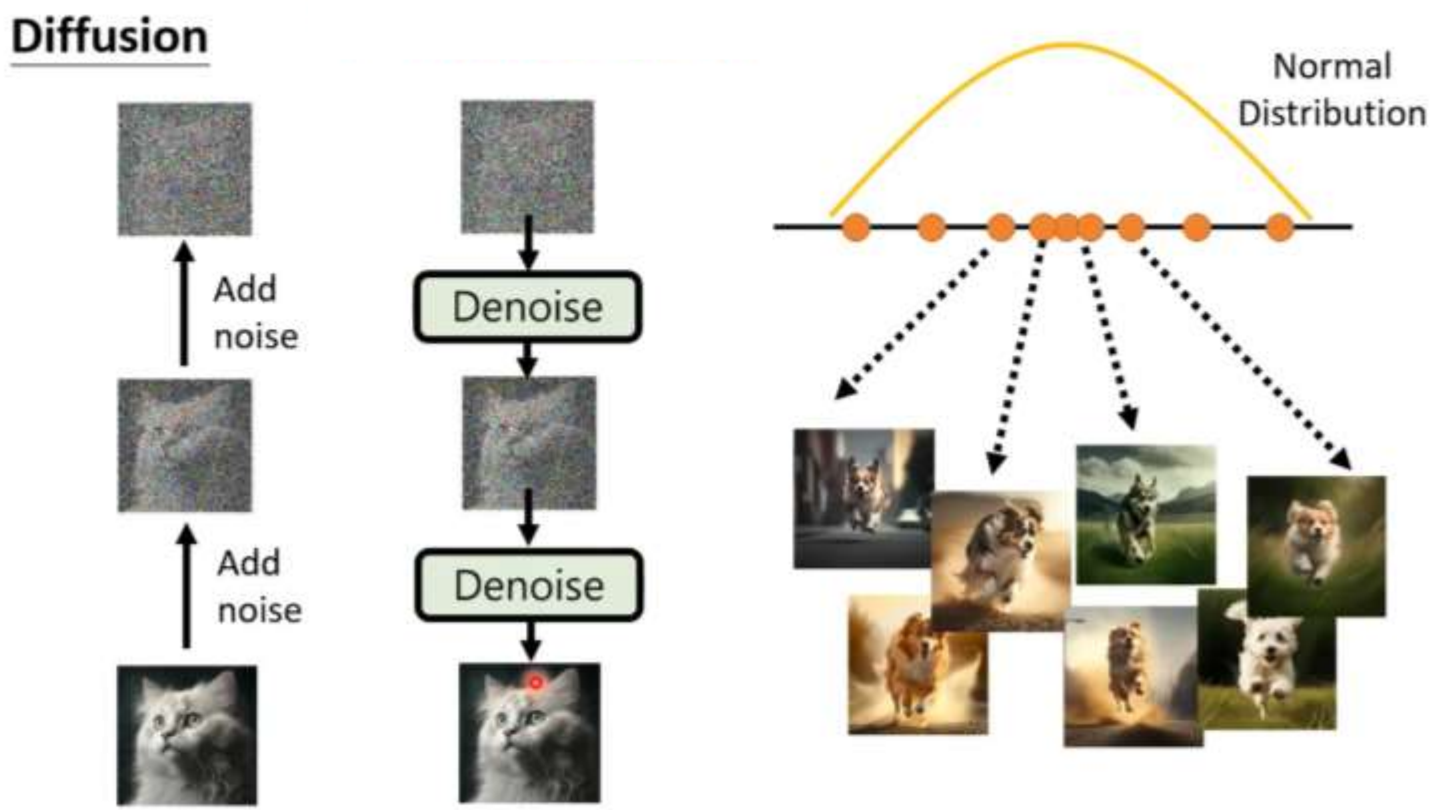
Generative Models Overview

- Modeling the distribution: VAE, GAN, normalizing flow (NF), **Diffusion**



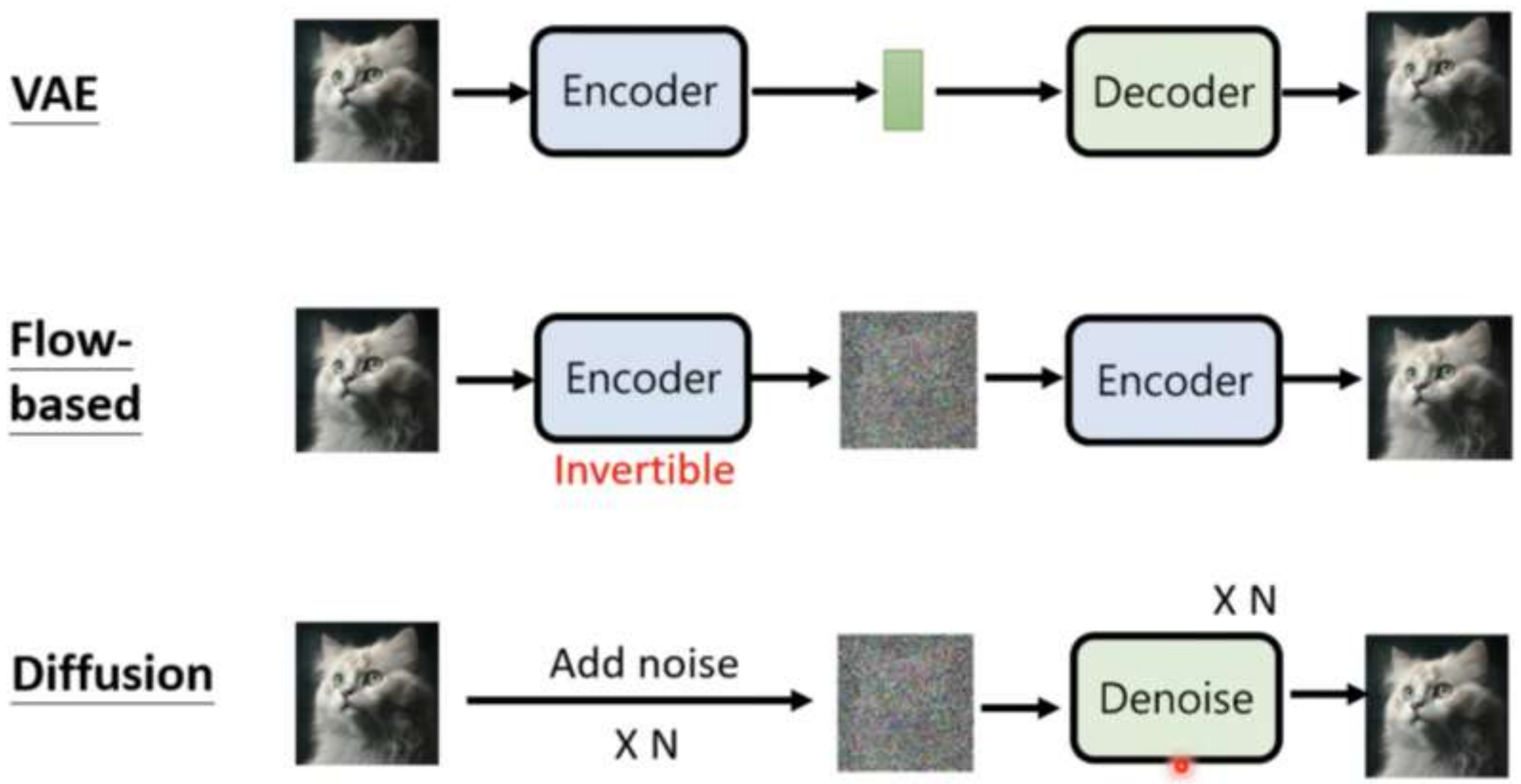
Generative Models Overview

- Modeling the distribution: VAE, GAN, normalizing flow (NF), **Diffusion**



Generative Models Overview

- Modeling the distribution: VAE, GAN, normalizing flow (NF), **Diffusion**



Huge success of text-2-img!

Sprouts in the shape of text 'Imagen' coming out of a fairytale book.

Google: Imagen

a teddy bear on a skateboard in times square

OpenAI: DALL·E 2

Stability AI: Stable-Diffusion

Midjourney

Huge success of text-2-img!

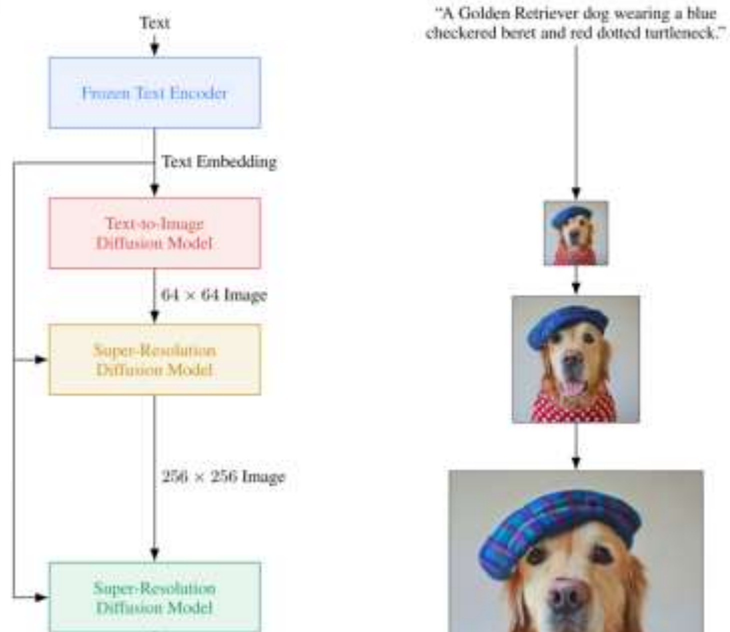
- Imagen (NeurIPS 2022)

Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding

Chitwan Saharia*, William Chan*, Saurabh Saxena†, Lala Li†, Jay Whang†, Emily Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, S. Sara Mahdavi, Rapha Gontijo Lopes, Tim Salimans, Jonathan Ho†, David J Fleet†, Mohammad Norouzi*

{sahariac,williamchan,mnorouzi}@google.com
{srbs,lala,jwhang,jonathanho,davidfleet}@google.com

Google Research, Brain Team
Toronto, Ontario, Canada



4.1 Training details

Unless specified, we train a 2B parameter model for the 64 × 64 text-to-image synthesis, and 600M and 400M parameter models for 64 × 64 → 256 × 256 and 256 × 256 → 1024 × 1024 for super-resolution respectively. We use a batch size of 2048 and 2.5M training steps for all models. We use 256 TPU-v4 chips for our base 64 × 64 model, and 128 TPU-v4 chips for both super-resolution

e input text
× 64 image.
the image,

Huge success of text-2-img!

- DALL·E2 (OpenAI)

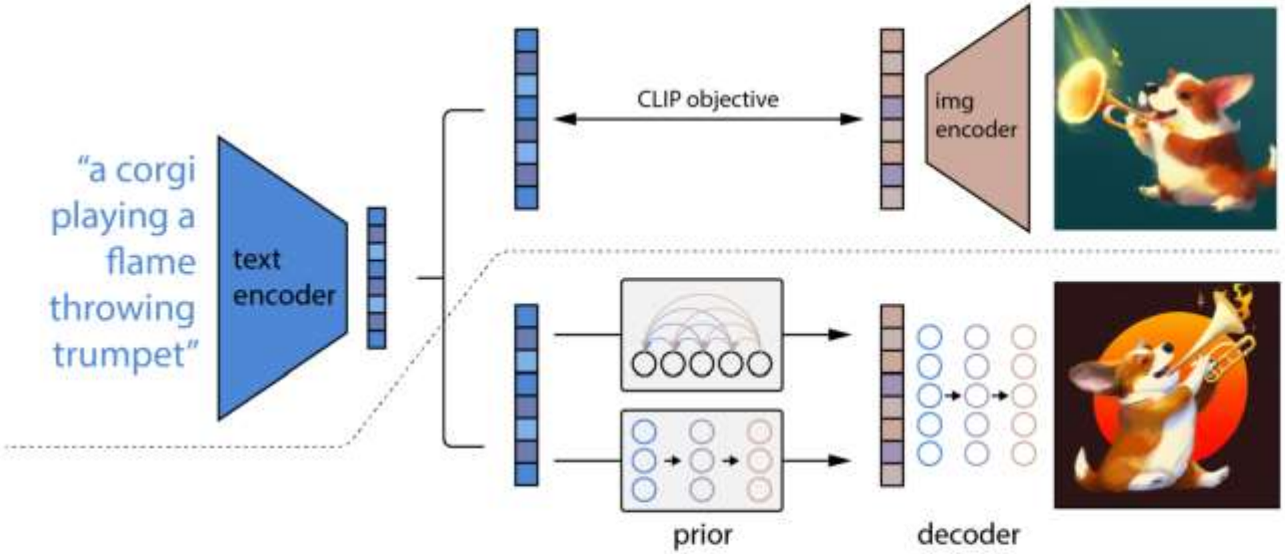
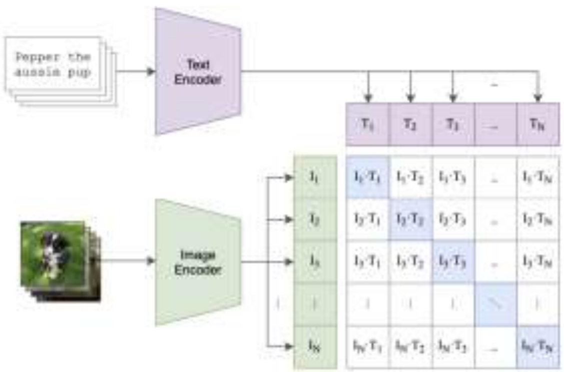


Figure 2: A high-level overview of unCLIP. Above the dotted line, we depict the CLIP training process, through which we learn a joint representation space for text and images. Below the dotted line, we depict our text-to-image generation process: a CLIP text embedding is first fed to an autoregressive or diffusion prior to produce an image embedding, and then this embedding is used to condition a diffusion decoder which produces a final image. Note that the CLIP model is frozen during training of the prior and decoder.

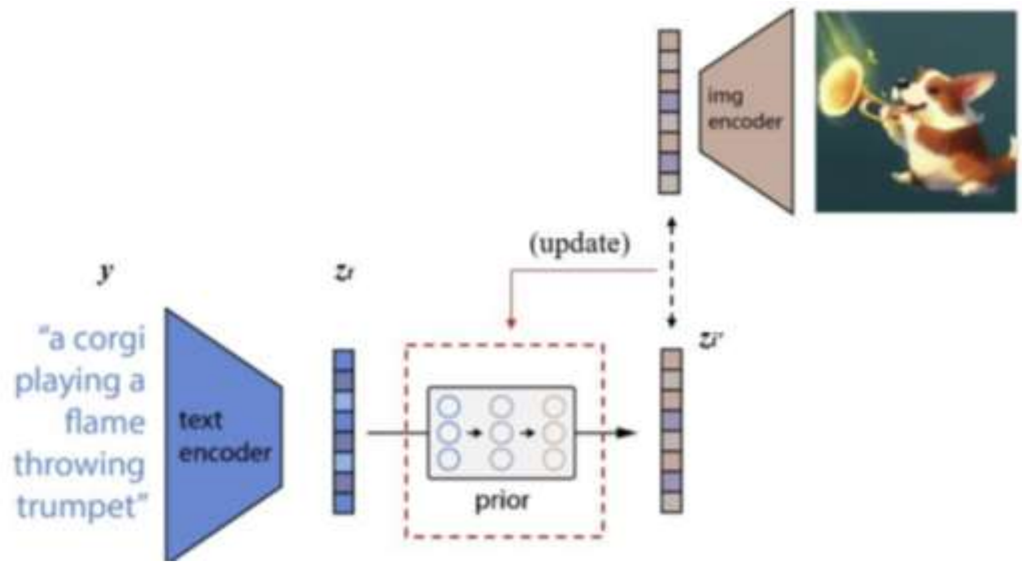
Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, Mark Chen, "Hierarchical Text-Conditional Image Generation with CLIP Latents", Arxiv 2022

Huge success of text-2-img!

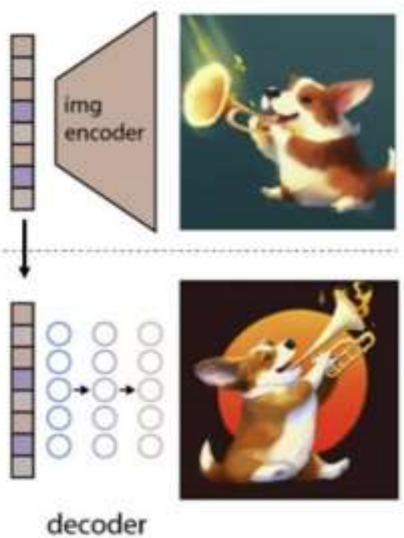
- DALL·E2 training



Step-1: CLIP Training



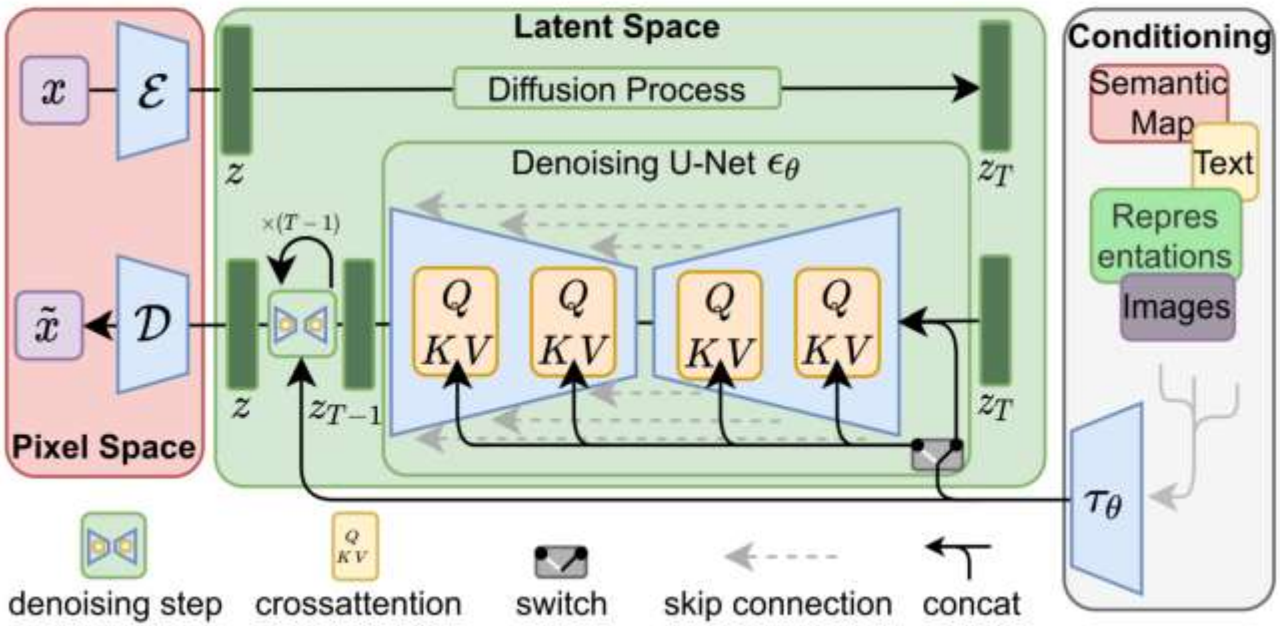
Step-2: Prior Training



Step-3: Decoder Training

Huge success of text-2-img!

- Stable Diffusion model (CVPR2022)



High-Resolution Image Synthesis with Latent Diffusion Models

Robin Rombach[1] * Andreas Blattmann[1] * Dominik Lorenz[1] Patrick Esser[R] Björn Ommer[1]

[1]Ludwig Maximilian University of Munich & IWR, Heidelberg University, Germany [R]Runway ML

https://github.com/CompVis/latent-diffusion

Huge success of text-2-img!

- Stable Diffusion model (CVPR2022)
- Long story between Stability AI, Runway ML and LAION-5B
- AI paradigm: data + algorithm + computing resource

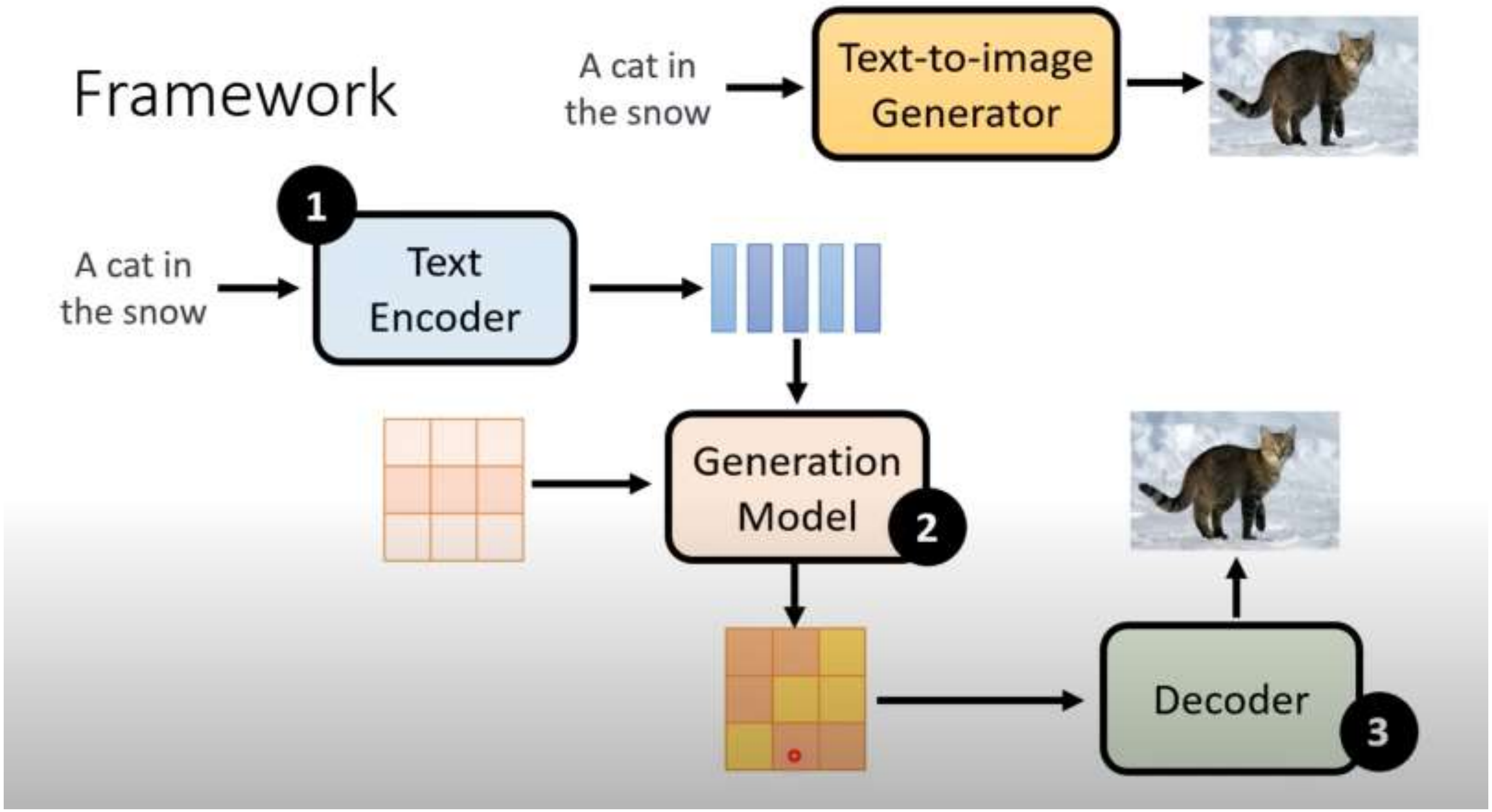
Computer Vision & Learning Group
Ludwig Maximilian University of Munich
(LMU)

~4000 A100 from
Stability AI

Huge text-image
dataset from
LAION

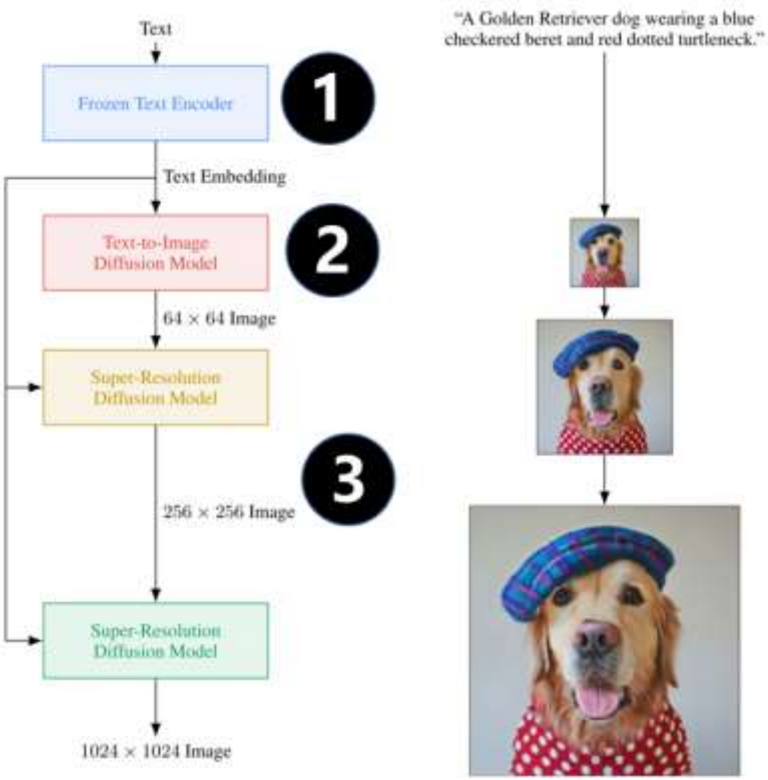
Huge success of text-2-img!

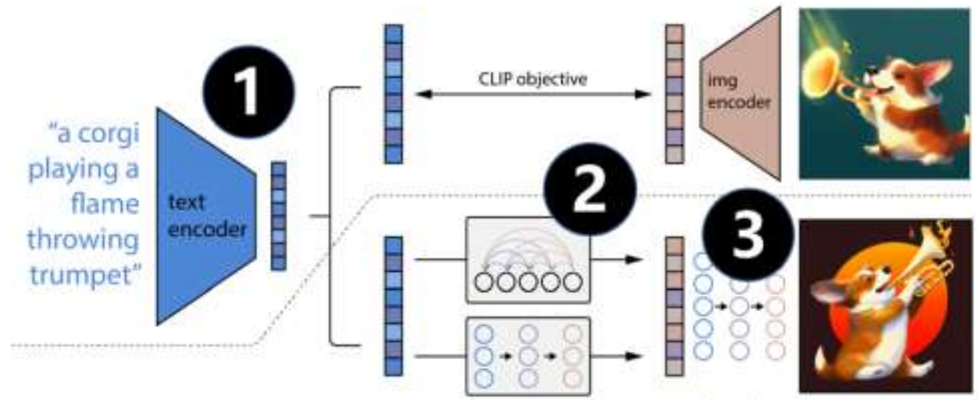
- A quick summary

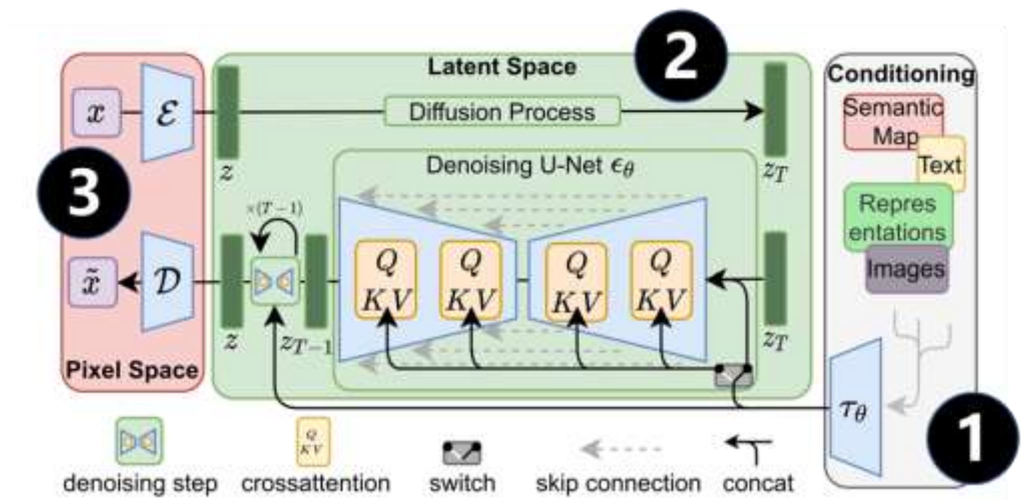


Huge success of text-2-img!

- A quick summary

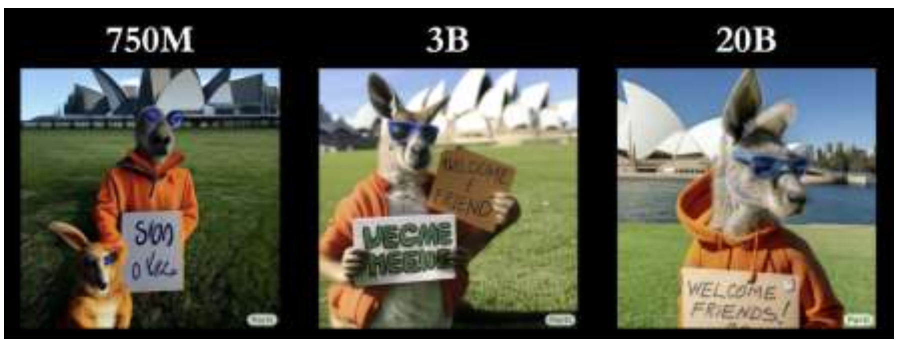




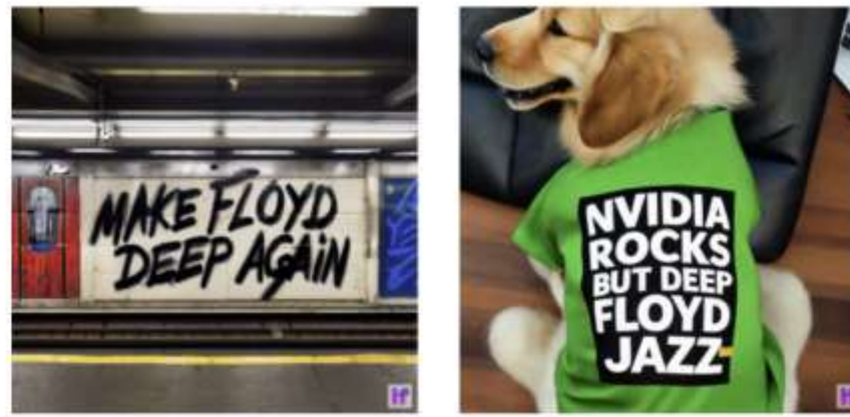


Huge success of text-2-img!

- The multi-modality framework is important
- The trend continues: big data, big modal ……
- Enjoy better text-encoder and suitable generator



Google: Parti Model, "Scaling Autoregressive Models for Content-Rich Text-to-Image Generation"



Stablity AI: DeepFloyd IF Model

T5-XXL as Text-encoder; pixel-level Diffusion

Huge success of text-2-img!

- Enjoy cross-modality abilities
- Enjoy downstream conditioning abilities

Conditioning using **ControlNet**

Subject-Driven Generation using **DreamBooth**

Editing Instructions using **InstructPix2Pix** (based on GPT-3)

Huge success of text-2-img-3D!

- Use NeRF as inherent representation to bridge 2D-DM with 3D scene
- More explicit disentanglement towards geometry, color, lighting …

3D Editing Instructions
using **InstructNeRF2NeRF**

NVDIA **Magic3D** 18 Nov 2022

OpenAI **Point-E** 21 Dec 2022

Outline

- Deep Generative Model Overview
- Unsupervised learning
  - Problem setup
- Latent variable model
  - EM algorithm and GMM
- Representation learning
  - AutoEncoder

Acknowledgement: Yingyu Liang@Princeton's & Feifei Li's cs231n notes

Unsupervised learning

- Task formulation

Unsupervised Learning

Data: x
Just data, no labels!

Goal: Learn some underlying hidden *structure* of the data

Examples: Clustering, dimensionality reduction, feature learning, density estimation, etc.

Unsupervised learning

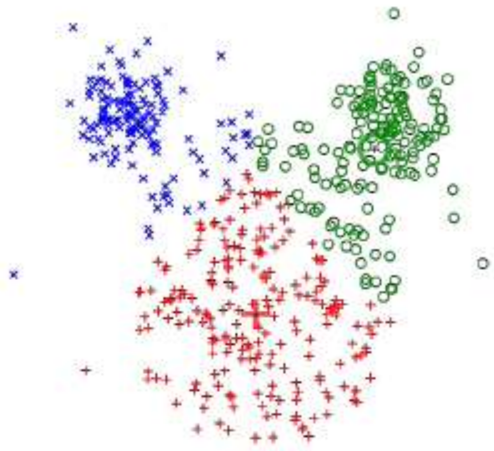
- Task formulation

Unsupervised Learning

Data: x
Just data, no labels!

Goal: Learn some underlying hidden *structure* of the data

Examples: Clustering, dimensionality reduction, feature learning, density estimation, etc.



K-means clustering

Unsupervised learning

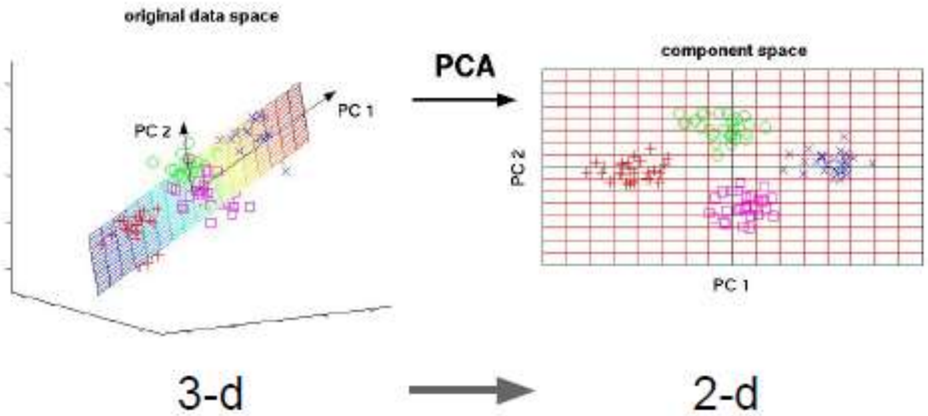
- Task formulation

Unsupervised Learning

Data: x
Just data, no labels!

Goal: Learn some underlying hidden *structure* of the data

Examples: Clustering, dimensionality reduction, feature learning, density estimation, etc.



Principal Component Analysis
(Dimensionality reduction)

Unsupervised learning

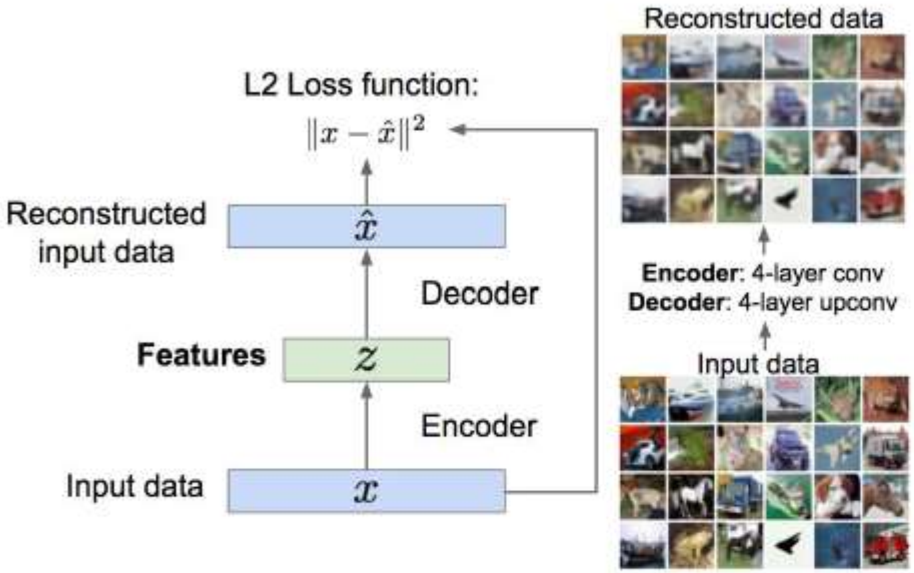
- Task formulation

Unsupervised Learning

Data: x
Just data, no labels!

Goal: Learn some underlying hidden *structure* of the data

Examples: Clustering, dimensionality reduction, feature learning, density estimation, etc.



Autoencoders
(Feature learning)

Unsupervised learning

- Task formulation

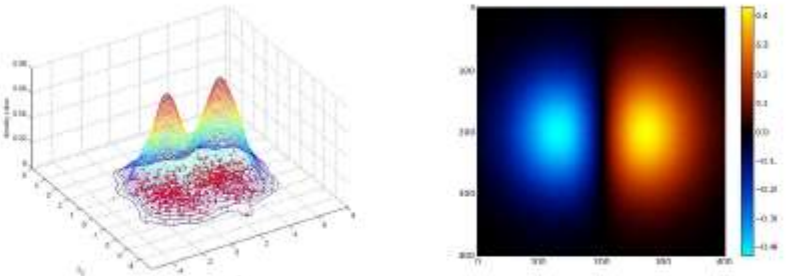
Unsupervised Learning

Data: x
Just data, no labels!

Goal: Learn some underlying hidden *structure* of the data

Examples: Clustering, dimensionality reduction, feature learning, density estimation, etc.

1-d density estimation



2-d density estimation

Outline

- Unsupervised learning
  - Problem setup
- Latent variable model
  - EM algorithm and GMM
- Representation learning
  - AutoEncoder

Acknowledgement: Yingyu Liang@Princeton's & Feifei Li's cs231n notes

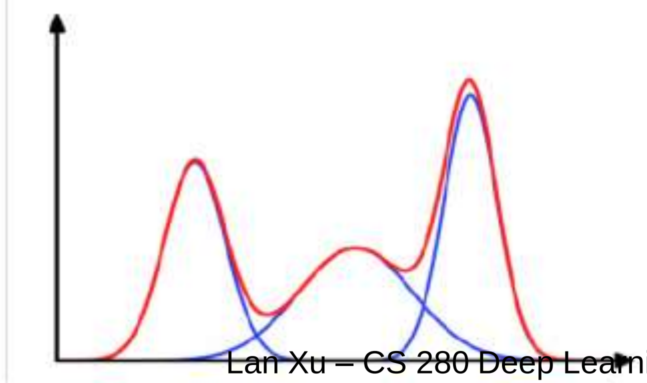
Latent variable model

- Data generation process
  - Latent variable $\boldsymbol{z}$ $\quad p(\boldsymbol{z}) = \text{something simple}$
  - A mapping from the latent space to observation $\boldsymbol{x}$

$$p(x) = \int p(x, z)\, dz \quad \text{where} \quad p(\boldsymbol{x}, \boldsymbol{z}) = p(\boldsymbol{x} \mid \boldsymbol{z}) p(\boldsymbol{z})$$

For example, a Gaussian mixture model

$$p_\theta(x) = \sum_{k=1}^{K} p_\theta(z = k) p_\theta(x | z = k)$$



Latent variable model

- Myth of the Cave
- Learn the true explanatory factors

Review: Gaussian Mixture Model

- Definition

  - A Gaussian mixture model represents a **distribution** as

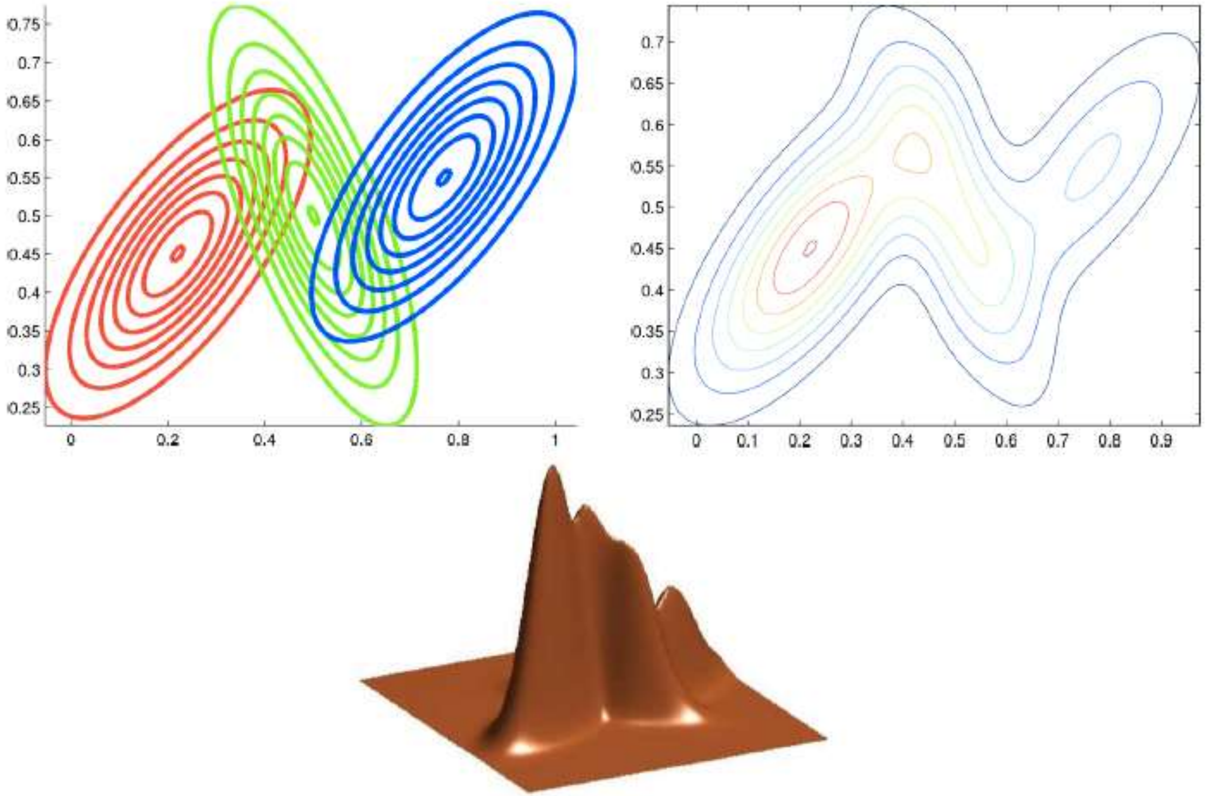
$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\mu_k, \Sigma_k)$$

with π_k the mixing coefficients, where:

$$\sum_{k=1}^{K} \pi_k = 1 \quad \text{and} \quad \pi_k \geq 0 \quad \forall k$$

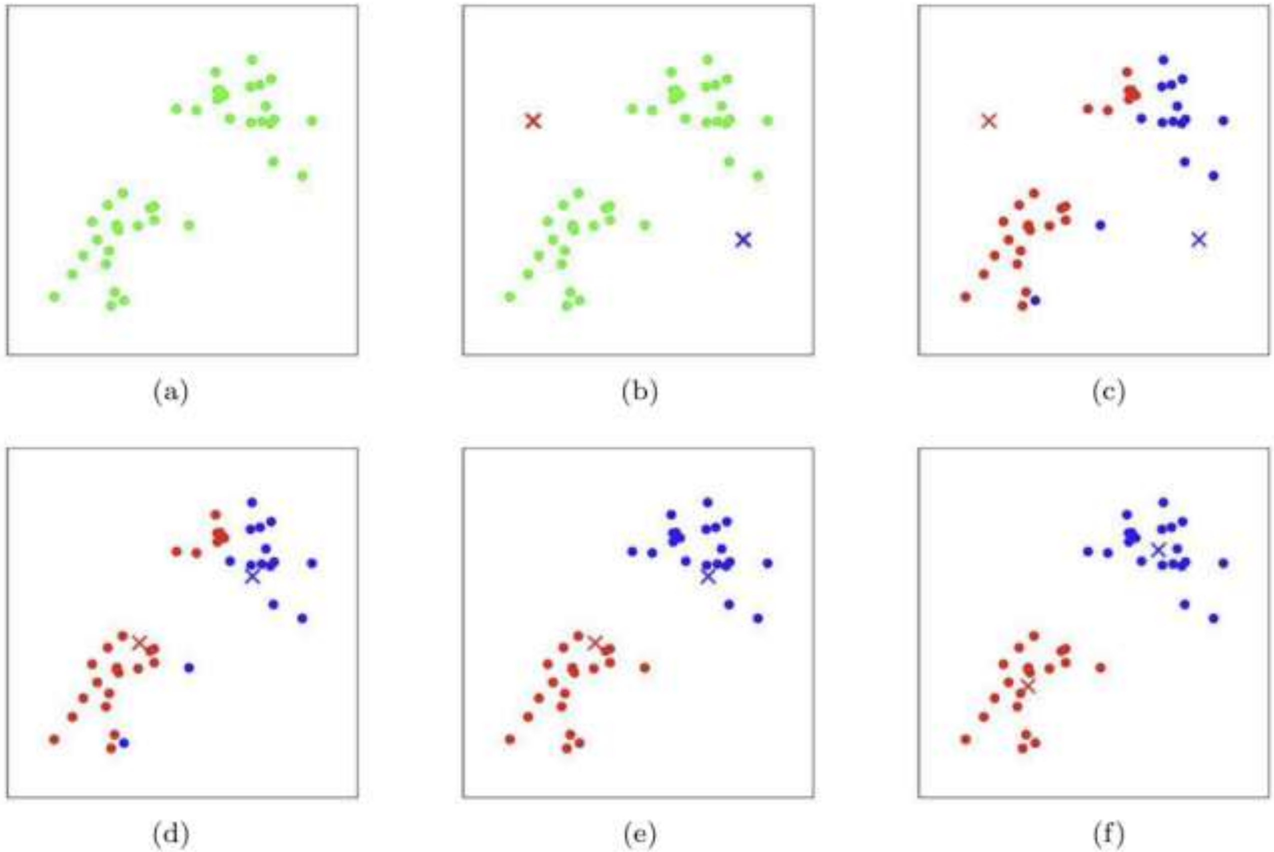
  - GMM is a density estimator
  - GMMs are universal approximators of densities (if you have enough Gaussians).

Review: 2D GMM example



Review: K-means

- Hard-thresholding



Fitting GMMs: Maximum Likelihood

- Maximum likelihood maximizes

$$\ln p(\mathbf{X}|\pi, \mu, \Sigma) = \sum_{n=1}^{N} \ln \left(\sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}^{(n)}|\mu_k, \Sigma_k) \right)$$

  w.r.t $\Theta = \{\pi_k, \mu_k, \Sigma_k\}$

- Problems:
  - Singularities: Arbitrarily large likelihood when a Gaussian explains a single point
  - Identifiability: Solution is up to permutations
- How would you optimize this?
- Can we have a closed form update?
- Don't forget to satisfy the constraints on π_k

Fitting GMMs as an LVM

- A latent variable model formulation
  - Introduce a hidden variable such that its knowledge would simplify the maximization
  - We could introduce a hidden (latent) variable z which would represent which Gaussian generated our observation $\mathbf{x}$, with some probability
  - Let $z \sim \text{Categorical}(\boldsymbol{\pi})$ $\quad$ (where $\pi_k \geq 0, \quad \sum_k \pi_k = 1$)
  - Then:

$$
\begin{aligned}
p(\mathbf{x}) &= \sum_{k=1}^{K} p(\mathbf{x}, z = k) \\
&= \sum_{k=1}^{K} \underbrace{p(z = k)}_{\pi_k} \underbrace{p(\mathbf{x}|z = k)}_{\mathcal{N}(\mathbf{x}|\mu_k, \Sigma_k)}
\end{aligned}
$$

Fitting GMMs

- A Gaussian mixture distribution:

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\mu_k, \Sigma_k)$$

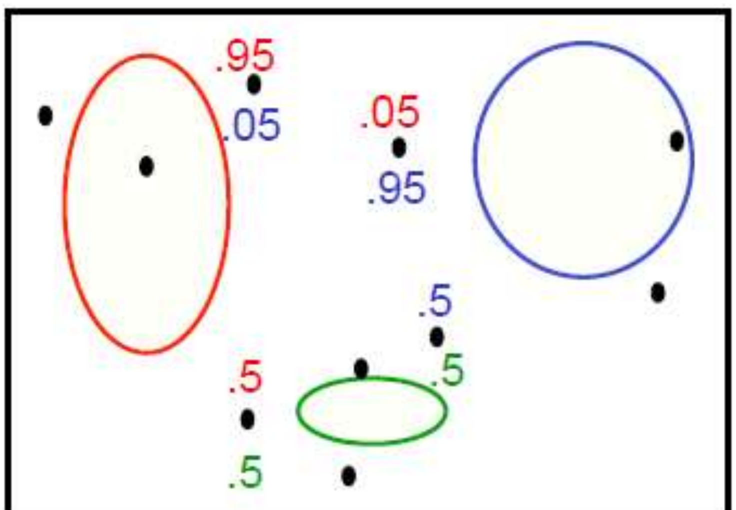
- We had: $z \sim \text{Categorical}(\pi)$ (where $\pi_k \geq 0$, $\sum_k \pi_k = 1$)
- Joint distribution: $p(\mathbf{x}, \mathbf{z}) = p(\mathbf{z})p(\mathbf{x}|\mathbf{z})$
- Log-likelihood:

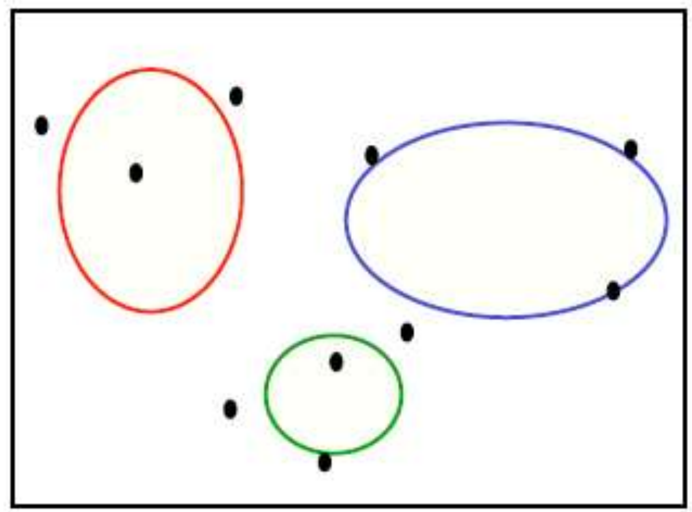
$$\begin{aligned}\ell(\pi, \mu, \Sigma) = \ln p(\mathbf{X}|\pi, \mu, \Sigma) &= \sum_{n=1}^{N} \ln p(\mathbf{x}^{(n)}|\pi, \mu, \Sigma) \\ &= \sum_{n=1}^{N} \ln \sum_{z^{(n)}=1}^{K} p(\mathbf{x}^{(n)}|\, z^{(n)}; \mu, \Sigma) p(z^{(n)}|\, \pi)\end{aligned}$$

- Note: We have a hidden variable $z^{(n)}$ for every observation
- General problem: sum inside the log
- How can we optimize this?

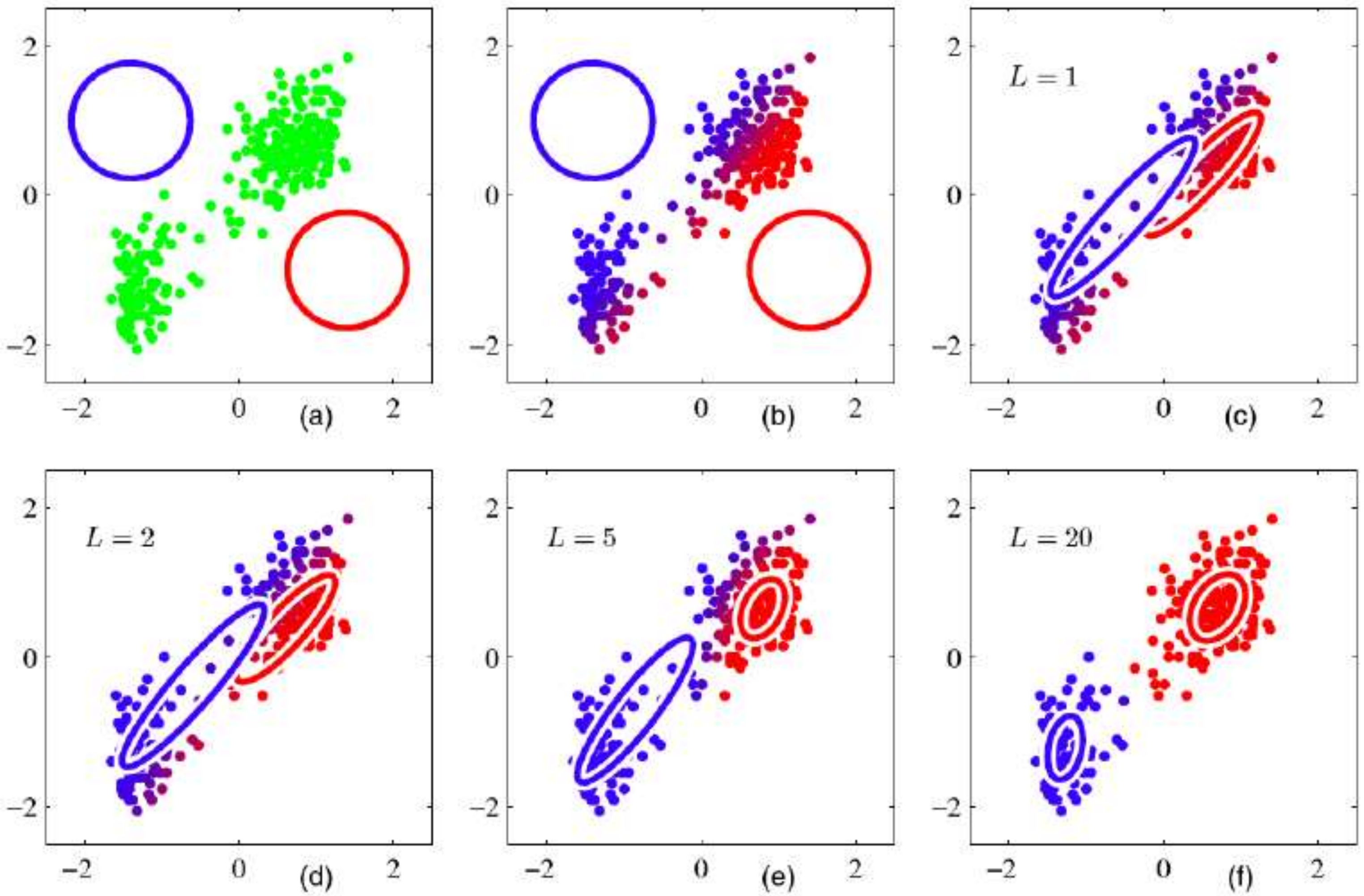
Expectation Maximization

- Optimization uses the Expectation Maximization algorithm, which alternates between two steps:
  1. E-step: Compute the posterior probability that each Gaussian generates each datapoint (as this is unknown to us)
  2. M-step: Assuming that the data really was generated this way, change the parameters of each Gaussian to maximize the probability that it would generate the data it is currently responsible for.





Expectation Maximization



Expectation Maximization

- Elegant and powerful method for finding maximum likelihood solutions for models with latent variables

  1. E-step:
     - In order to adjust the parameters, we must first solve the inference problem: Which Gaussian generated each datapoint?
     - We cannot be sure, so it's a distribution over all possibilities.

$$\gamma_k^{(n)} = p(z^{(n)} = k | \mathbf{x}^{(n)}; \pi, \mu, \Sigma)$$

  2. M-step:
     - Each Gaussian gets a certain amount of posterior probability for each datapoint.
     - At the optimum we shall satisfy

$$\frac{\partial \ln p(\mathbf{X}|\pi, \mu, \Sigma)}{\partial \Theta} = 0$$

     - We can derive closed form updates for all parameters

GMM: E-Step

- Conditional probability (using Bayes rule) of $\mathbf{z}$ given $\mathbf{x}$

$$\begin{aligned}
\gamma_k = p(z = k|\mathbf{x}) &= \frac{p(z = k)p(\mathbf{x}|z = k)}{p(\mathbf{x})} \\
&= \frac{p(z = k)p(\mathbf{x}|z = k)}{\sum_{j=1}^{K} p(z = j)p(\mathbf{x}|z = j)} \\
&= \frac{\pi_k \mathcal{N}(\mathbf{x}|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x}|\mu_j, \Sigma_j)}
\end{aligned}$$

- γ_k can be viewed as the responsibility

GMM: M-Step

- Log-likelihood:

$$\ln p(\mathbf{X}|\pi,\mu,\Sigma) = \sum_{n=1}^{N} \ln \left(\sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}^{(n)}|\mu_k,\Sigma_k) \right)$$

- Set derivatives to 0:

$$\frac{\partial \ln p(\mathbf{X}|\pi,\mu,\Sigma)}{\partial \mu_k} = 0 = \sum_{n=1}^{N} \frac{\pi_k \mathcal{N}(\mathbf{x}^{(n)}|\mu_k,\Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x}|\mu_j,\Sigma_j)} \Sigma_k(\mathbf{x}^{(n)} - \mu_k)$$

- We used:

$$\mathcal{N}(\mathbf{x}|\mu,\Sigma) = \frac{1}{\sqrt{(2\pi)^d|\Sigma|}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)\right)$$

and:

$$\frac{\partial(\mathbf{x}^T A\mathbf{x})}{\partial \mathbf{x}} = \mathbf{x}^T(A + A^T)$$

GMM: M-Step

$$\frac{\partial \ln p(\mathbf{X}|\pi,\mu,\Sigma)}{\partial \mu_k} = 0 = \sum_{n=1}^{N} \underbrace{\frac{\pi_k \mathcal{N}(\mathbf{x}^{(n)}|\mu_k,\Sigma_k)}{\sum_{j=1}^{K}\pi_j \mathcal{N}(\mathbf{x}|\mu_j,\Sigma_j)}}_{\gamma_k^{(n)}} \Sigma_k(\mathbf{x}^{(n)} - \mu_k)$$

- This gives

$$\mu_k = \frac{1}{N_k}\sum_{n=1}^{N}\gamma_k^{(n)}\mathbf{x}^{(n)}$$

with N_k the effective number of points in cluster k

$$N_k = \sum_{n=1}^{N}\gamma_k^{(n)}$$

- We just take the center-of gravity of the data that the Gaussian is responsible for
- Just like in K-means, except the data is weighted by the posterior probability of the Gaussian.
- Guaranteed to lie in the convex hull of the data (Could be big initial jump)

GMM: M-Step

- We can get similarly expression for the variance

$$\Sigma_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma_k^{(n)} (\mathbf{x}^{(n)} - \mu_k)(\mathbf{x}^{(n)} - \mu_k)^T$$

- We can also minimize w.r.t the mixing coefficients

$$\pi_k = \frac{N_k}{N}, \quad \text{with} \quad N_k = \sum_{n=1}^{N} \gamma_k^{(n)}$$

- The optimal mixing proportion to use (given these posterior probabilities) is just the fraction of the data that the Gaussian gets responsibility for.
- Note that this is not a closed form solution of the parameters, as they depend on the responsibilities $\gamma_k^{(n)}$, which are complex functions of the parameters
- But we have a simple iterative scheme to optimize

Summary: EM Algorithm for GMM

- Initialize the means μ_k, covariances Σ_k and mixing coefficients π_k
- Iterate until convergence:
  - E-step: Evaluate the responsibilities given current parameters

$$\gamma_k^{(n)} = p(z^{(n)}|\mathbf{x}) = \frac{\pi_k \mathcal{N}(\mathbf{x}^{(n)}|\mu_k, \Sigma_k)}{\sum_{j=1}^K \pi_j \mathcal{N}(\mathbf{x}^{(n)}|\mu_j, \Sigma_j)}$$

  - M-step: Re-estimate the parameters given current responsibilities

$$\begin{aligned} \mu_k &= \frac{1}{N_k} \sum_{n=1}^N \gamma_k^{(n)} \mathbf{x}^{(n)} \\ \Sigma_k &= \frac{1}{N_k} \sum_{n=1}^N \gamma_k^{(n)} (\mathbf{x}^{(n)} - \mu_k)(\mathbf{x}^{(n)} - \mu_k)^T \\ \pi_k &= \frac{N_k}{N} \quad \text{with} \quad N_k = \sum_{n=1}^N \gamma_k^{(n)} \end{aligned}$$

  - Evaluate log likelihood and check for convergence

$$\ln p(\mathbf{X}|\pi, \mu, \Sigma) = \sum_{n=1}^N \ln \left(\sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}^{(n)}|\mu_k, \Sigma_k) \right)$$

An Alternative View of EM

- Hard to maximize (log-)likelihood of data directly
- General problem: sum inside the log

$$\ln p(\mathbf{x}|\Theta) = \ln \sum_{z} p(\mathbf{x}, \mathbf{z}|\Theta)$$

- Complete data $\{\mathbf{x}, \mathbf{z}\}$, and $\mathbf{x}$ is the incomplete data
- If we knew z, then easy to maximize (replace sum over k with just the k where $z = k$)
- Unfortunately we are not given the complete data, but only the incomplete.

An Alternative View of EM

- Our knowledge about the latent variables is $p(\mathbf{Z}|\mathbf{X}, \Theta^{old})$
- In the E-step we compute $p(\mathbf{Z}|\mathbf{X}, \Theta^{old})$
- In the M-step we maximize w.r.t Θ

$$Q(\Theta, \Theta^{old}) = \sum_{z} p(\mathbf{Z}|\mathbf{X}, \Theta^{old}) \ln p(\mathbf{X}, \mathbf{Z}|\Theta)$$

  - E-step: Evaluate the responsibilities given current parameters

$$\gamma_k^{(n)} = p(z^{(n)}|\mathbf{x}) = \frac{\pi_k \mathcal{N}(\mathbf{x}^{(n)}|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x}^{(n)}|\mu_j, \Sigma_j)}$$

  - M-step: Re-estimate the parameters given current responsibilities

$$\max \sum_{n=1}^{N} \sum_{z^{(n)}=1}^{K} p(z^{(n)}|x^{(n)}) \ln[p(x^{(n)}|z^{(n)}; \mu_k, \Sigma_k) p(z^{(n)}|\pi_k)]$$

$$\Leftrightarrow \quad \max \sum_{n=1}^{N} \sum_{z^{(n)}=1}^{K} \gamma_k^{(n)} \ln[\pi_k \mathcal{N}(x^{(n)}|\mu_k, \Sigma_k)]$$

$$\Leftrightarrow \quad \max_{\pi_k, \mu_k, \Sigma_k} \sum_{n=1}^{N} \gamma_k^{(n)} \ln[\pi_k \mathcal{N}(x^{(n)}|\mu_k, \Sigma_k)] \quad \Rightarrow$$

$$\begin{aligned} \mu_k &= \frac{1}{N_k} \sum_{n=1}^{N} \gamma_k^{(n)} \mathbf{x}^{(n)} \\ \Sigma_k &= \frac{1}{N_k} \sum_{n=1}^{N} \gamma_k^{(n)} (\mathbf{x}^{(n)} - \mu_k)(\mathbf{x}^{(n)} - \mu_k)^T \\ \pi_k &= \frac{N_k}{N} \quad \text{with} \quad N_k = \sum_{n=1}^{N} \gamma_k^{(n)} \end{aligned}$$

General EM Algorithm

1. Initialize Θ^{old}
2. E-step: Evaluate $p(\mathbf{Z}|\mathbf{X}, \Theta^{old})$
3. M-step:

$$\Theta^{new} = arg\max_{\Theta} Q(\Theta, \Theta^{old})$$

   where

$$Q(\Theta, \Theta^{old}) = \sum_{z} p(\mathbf{Z}|\mathbf{X}, \Theta^{old}) \ln p(\mathbf{X}, \mathbf{Z}|\Theta)$$

4. Evaluate log likelihood and check for convergence (or the parameters). If not converged, $\Theta^{old} = \Theta$, Go to step 2

Why it works?

- Updating each Gaussian definitely improves the probability of generating the data if we generate it from the same Gaussians after the parameter updates.
  - But we know that the posterior will change after updating the parameters.
- A good way to show that this is OK is to show that there is a single function that is improved by both the E-step and the M-step.
  - The function we need is called Free Energy.

Review: Jensen's Inequality

- For concave function f, we have

$$f(\mathbb{E}[X]) \geq \mathbb{E}[f(X)]$$

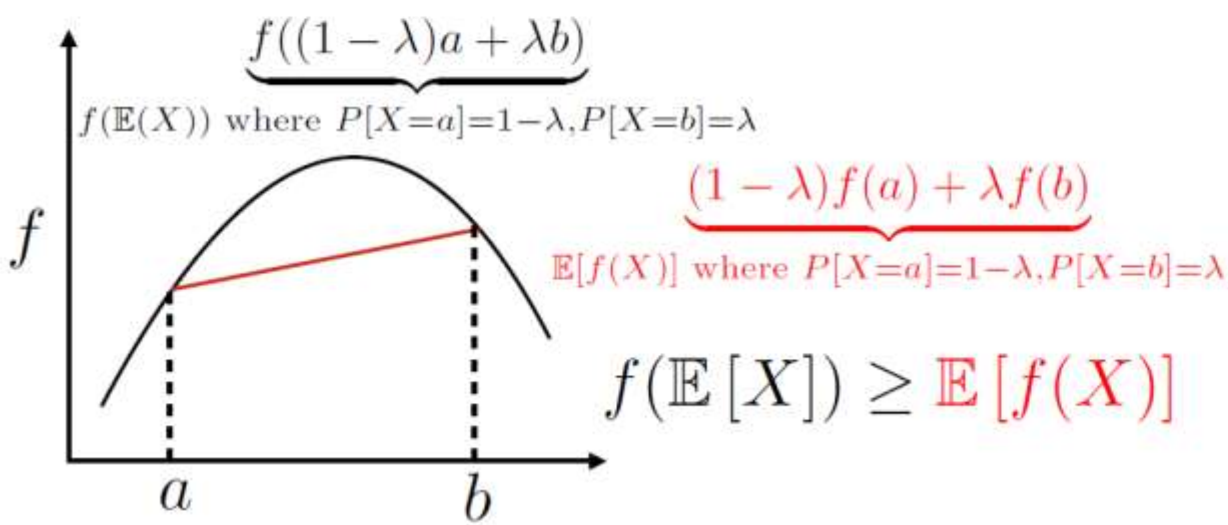


Figure: Jensen's Inequality

A simple latent variable model

- We assume that the data is generated i.i.d as

$$z \sim p(z) \qquad x \sim p(x|z)$$

  - z is latent/hidden and x is observed
- Bounding the marginal likelihood

$$\log p(x) = \log \int_z p(x,z) \text{ (Multiply and divide by } q(z))$$

$$= \log \int_z \frac{q(z)p(x,z)}{q(z)} = \log \mathbb{E}_{z\sim q(z)}\left[\frac{p(x,z)}{q(z)}\right] \text{ (By Jensen's Inequality)}$$

$$\geq \int_z q(z) \log \frac{p(x,z)}{q(z)} = \mathcal{L}(x;\theta,\phi)$$

$$= \underbrace{\mathbb{E}_{q(z)}[\log p(x,z)]}_{\text{Expectation of Joint distribution}} + \underbrace{\mathrm{H}(q(z))}_{\text{Entropy}}$$

Evidence Lower Bound (ELBO)

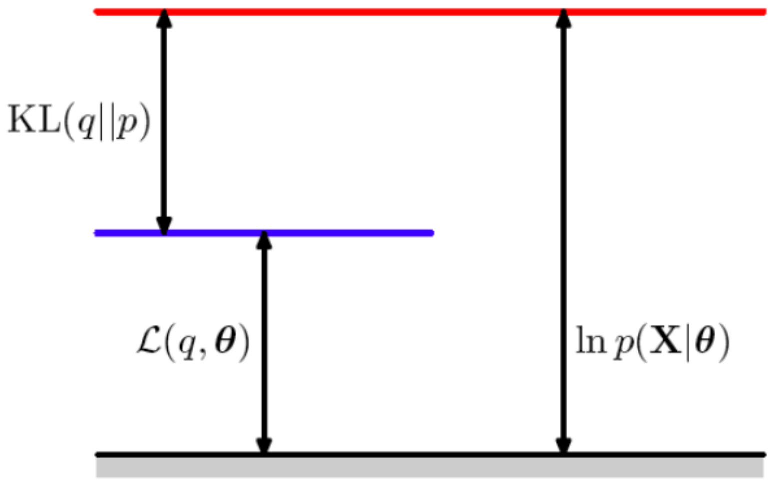
- When is the lower bound tight?
- Let's look at: objective function – lower bound

$$\log p(x;\theta) - \mathcal{L}(x;\theta,\phi)$$

$$\begin{aligned}
&\log p(x) - \int_z q(z) \log \frac{p(x,z)}{q(z)} \\
&= \int_z q(z) \log p(x) - \int_z q(z) \log \frac{p(x,z)}{q(z)} \\
&= \int_z q(z) \log \frac{q(z)p(x)}{p(x,z)} \\
&= \mathrm{KL}(q(z;\phi)||p(z|x))
\end{aligned}$$

Visualization of ELBO

$$\mathcal{L}(q, \Theta) \leq \ln p(\mathbf{X}|\Theta)$$



Key Point

The optimal $q(z;\phi)$ corresponds to the one that realizes
$\mathrm{KL}(q(z;\phi)||p(z|x)) = 0 \iff q(z;\phi) = p(z|x)$

E-step and M-step

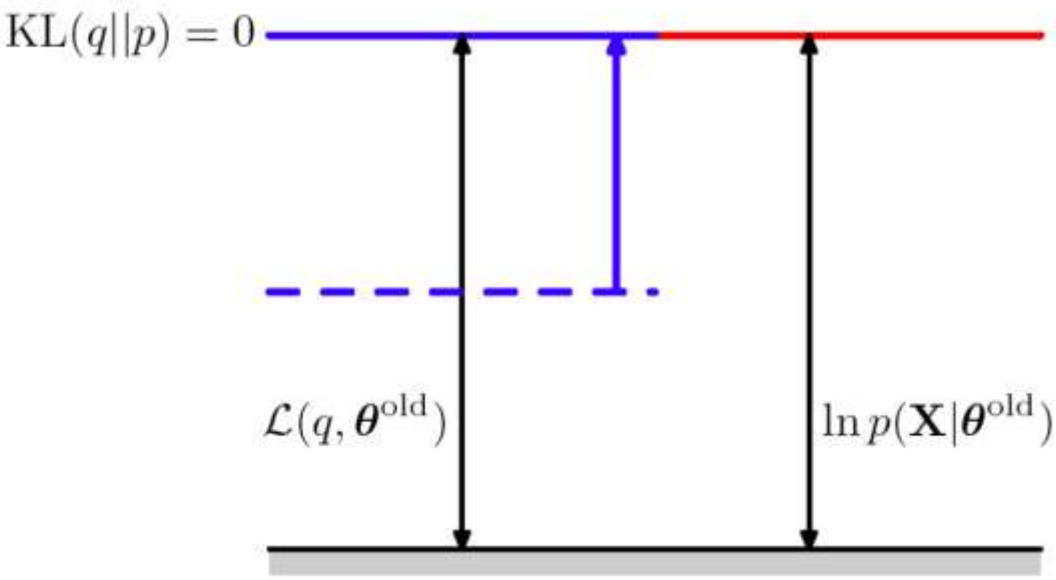
$$\ln p(\mathbf{X}|\Theta) = \mathcal{L}(q, \Theta) + KL(q||p(\mathbf{Z}|\mathbf{X}, \Theta))$$

- In the E-step we maximize w.r.t $q(\mathbf{Z})$ the lower bound $\mathcal{L}(q, \Theta)$
- Since $\ln p(\mathbf{X}|\theta)$ does not depend on $q(\mathbf{Z})$, the maximum $\mathcal{L}$ is obtained when the KL is 0
- This is achieved when $q(\mathbf{Z}) = p(\mathbf{Z}|\mathbf{X}, \Theta)$
- The lower bound $\mathcal{L}$ is then

$$\begin{aligned}\mathcal{L}(q, \Theta) &= \sum_{\mathbf{Z}} p(\mathbf{Z}|\mathbf{X}, \Theta^{old}) \ln p(\mathbf{X}, \mathbf{Z}|\Theta) - \sum_{\mathbf{Z}} p(\mathbf{Z}|\mathbf{X}, \Theta^{old}) \ln p(\mathbf{Z}|\mathbf{X}, \Theta^{old}) \\ &= Q(\Theta, \Theta^{old}) + \text{const}\end{aligned}$$

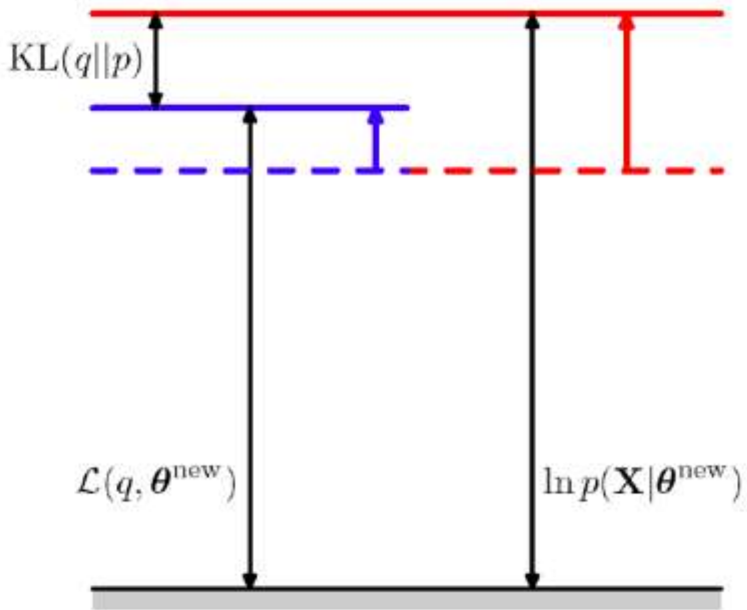
  with the content the entropy of the q distribution, which is independent of Θ
- In the M-step the quantity to be maximized is the expectation of the complete data log-likelihood
- Note that Θ is only inside the logarithm and optimizing the complete data likelihood is easier

Visualization of E-step



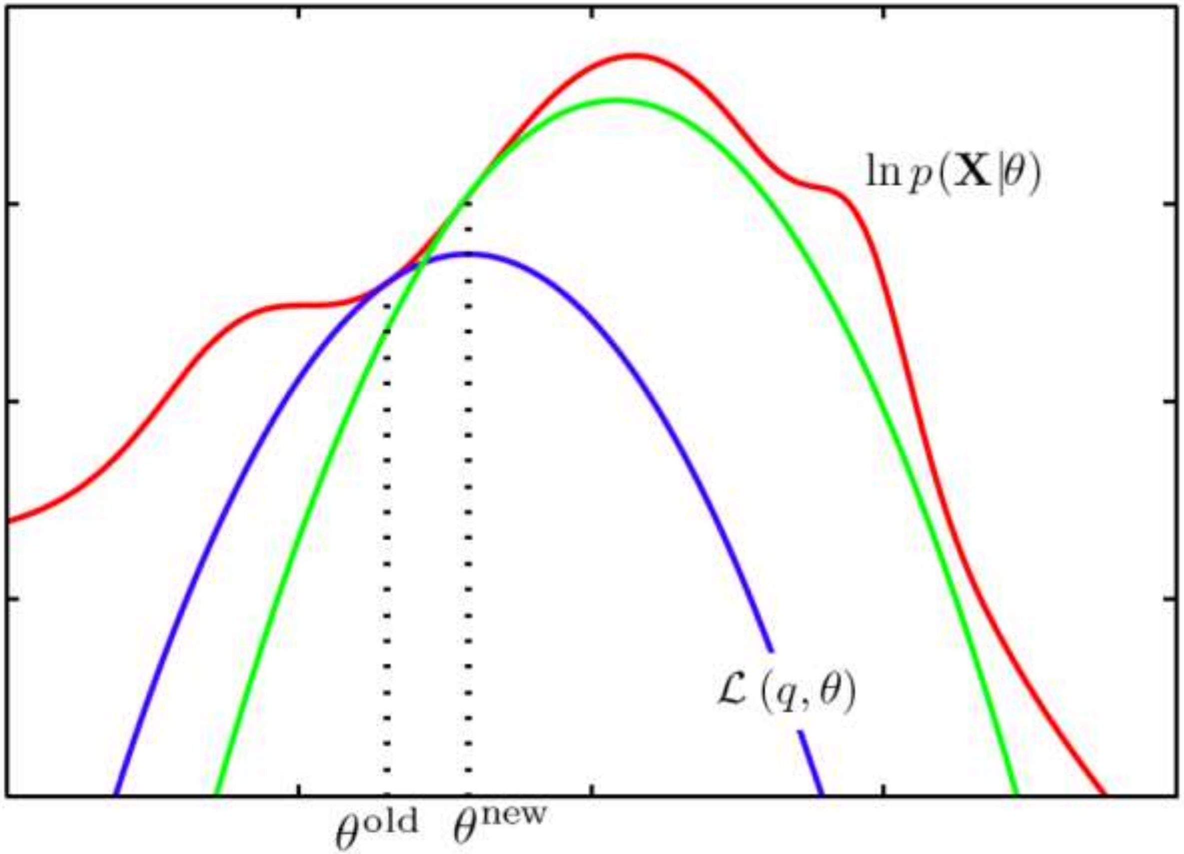
- The q distribution equal to the posterior distribution for the current parameter values Θ^{old}, causing the lower bound to move up to the same value as the log likelihood function, with the KL divergence vanishing.

Visualization of M-step



- The distribution $q(\mathbf{Z})$ is held fixed and the lower bound $\mathcal{L}(q, \Theta)$ is maximized with respect to the parameter vector Θ to give a revised value Θ^{new}. Because the KL divergence is nonnegative, this causes the log likelihood $\ln p(\mathbf{X}|\Theta)$ to increase by at least as much as the lower bound does.

Visualization of the EM Algorithm



- The EM algorithm involves alternately computing a lower bound on the log likelihood for the current parameter values and then maximizing this bound to obtain the new parameter values.

Summary of EM algorithms

- EM is coordinate ascent in ELBO

$$\int_z q(z) \log \frac{p(x,z)}{q(z)} = \mathcal{L}(x;\theta,\phi)$$

$$= \underbrace{\mathbb{E}_{q(z)}[\log p(x,z)]}_{\text{Expectation of Joint distribution}} + \underbrace{\mathrm{H}(q(z))}_{\text{Entropy}}$$

- Or coordinate descent in Free Energy

$$\mathcal{F} = -\mathcal{L}(x;\theta,q) = \underbrace{E_q[-\log p(x,z)]}_{\text{Expected energy}} - \underbrace{H(q(z))}_{\text{Entropy}}$$

- The E-step minimizes F by finding the best distribution over hidden configurations for each data point.
- The M-step holds the distribution fixed and minimizes F by changing the parameters that determine the energy of a configuration.

Recall EM GMM

- MLE: maximizing the log-likelihood

$$\ell(\theta) = \sum_{i=1}^{n} \log p(x^{(i)}; \theta)$$

- ELBO: Evidence Lower Bound

$$\begin{aligned}
\log p(\mathbf{x}) &= \log \int_z p(\mathbf{x}, z) \\
&= \log \int_z p(\mathbf{x}, z) \frac{q(\mathbf{z})}{q(\mathbf{z})} \\
&= \log(E_q[\frac{p(\mathbf{x}, \mathbf{z})}{q(\mathbf{z})}]) \\
&\geq E_q[\log p(\mathbf{x}, \mathbf{z})] - E_q[\log q(\mathbf{z})] \\
&= ELBO
\end{aligned}$$

$$\begin{aligned}
KL(q(\mathbf{z})||p(\mathbf{z}|\mathbf{x})) &= E_q[\log q(\mathbf{z})] - E_q[\log p(\mathbf{z}|\mathbf{x})] \\
&= E_q[\log q(\mathbf{z})] - E_q[\log p(\mathbf{z}, \mathbf{x})] + \log p(\mathbf{x}) \\
&= \log p(\mathbf{x}) - (E_q[\log p(\mathbf{z}, \mathbf{x})] - E_q[\log q(\mathbf{z})]) \\
&= \log p(\mathbf{x}) - ELBO
\end{aligned}$$

$$\begin{aligned}
EBLO &= E_q[\log p(\mathbf{x}, \mathbf{z})] - E_q[\log q(\mathbf{z})] \\
&= E_q[\log \frac{p(\mathbf{x}, \mathbf{z})}{p(\mathbf{z})}] - E_q[\log \frac{q(\mathbf{z})}{p(\mathbf{z})}] \\
&= E_q[\log p(\mathbf{x}|\mathbf{z})] - KL(q(\mathbf{z})||p(\mathbf{z}))
\end{aligned}$$

Outline

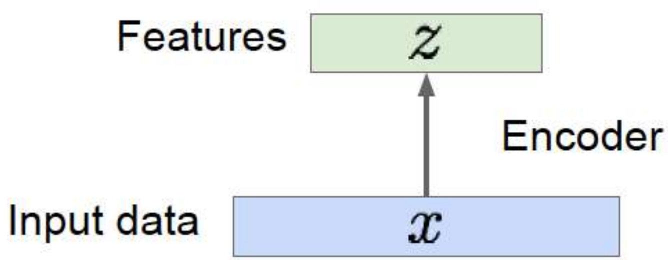
- Unsupervised learning
  - Problem setup
- Latent variable model
  - EM algorithm and GMM
- Representation learning
  - AutoEncoder

Acknowledgement: Yingyu Liang@Princeton's & Feifei Li's cs231n notes

Autoencoder

- Feature representation learning

Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data

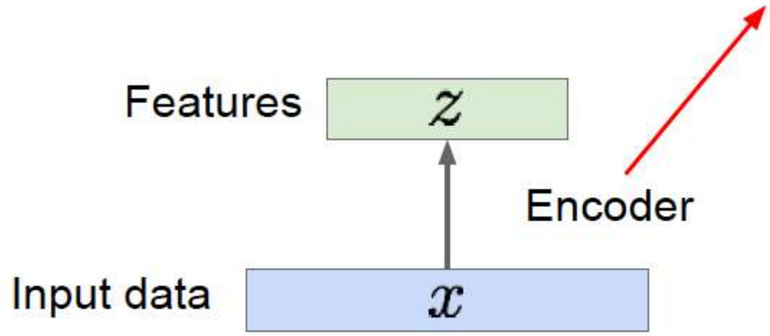


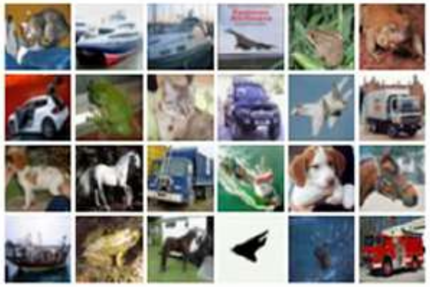
Autoencoder

- Feature representation learning

Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data

Originally: Linear + nonlinearity (sigmoid)
Later: Deep, fully-connected
Later: ReLU CNN

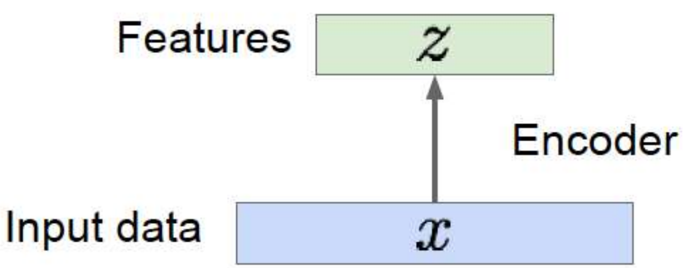




Autoencoder

- Feature representation learning

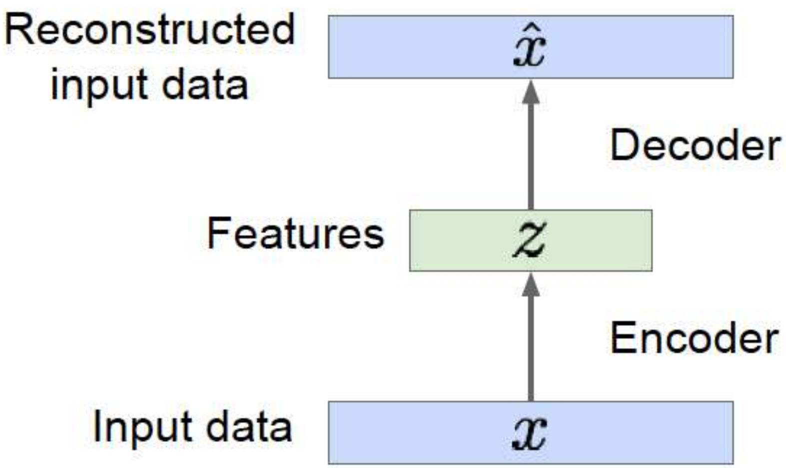
How to learn this feature representation?

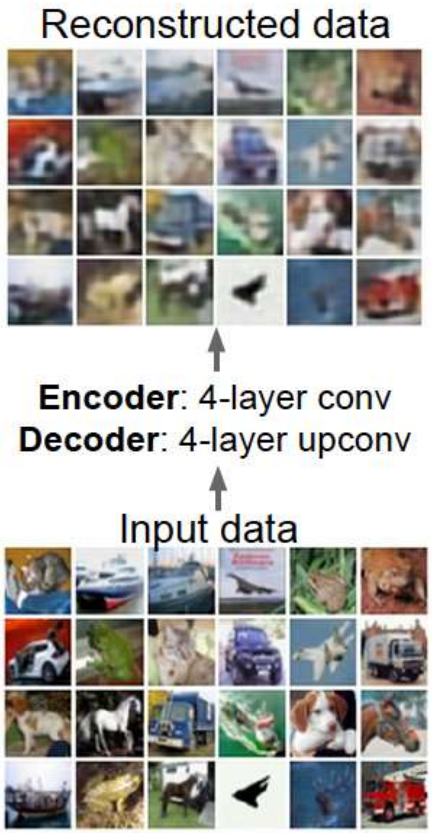


Autoencoder

- Feature representation learning

How to learn this feature representation?
Train such that features can be used to reconstruct original data
"Autoencoding" - encoding itself

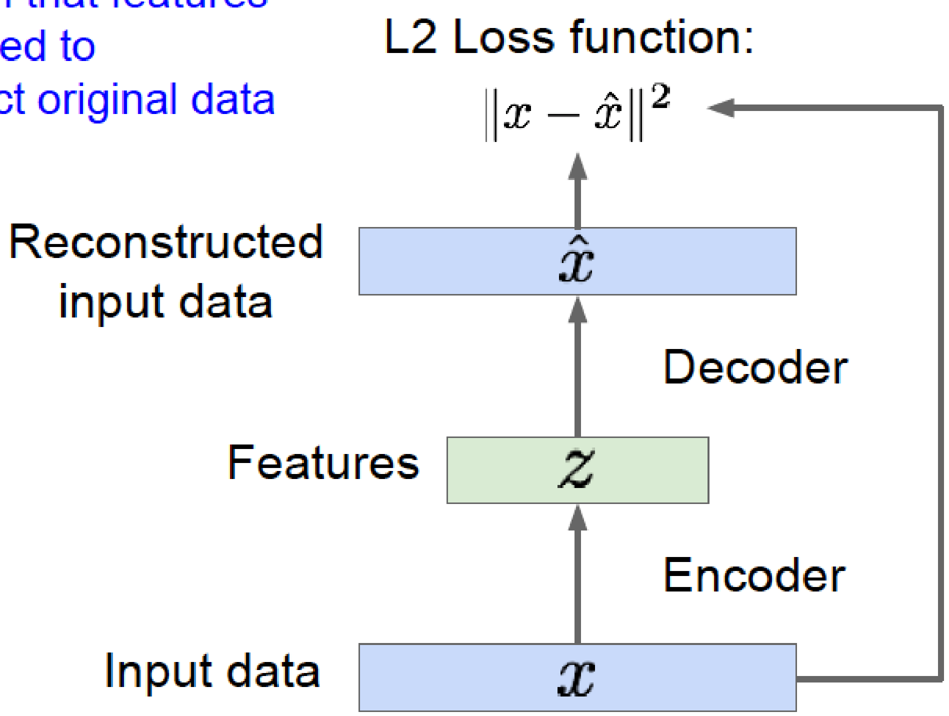




Autoencoder

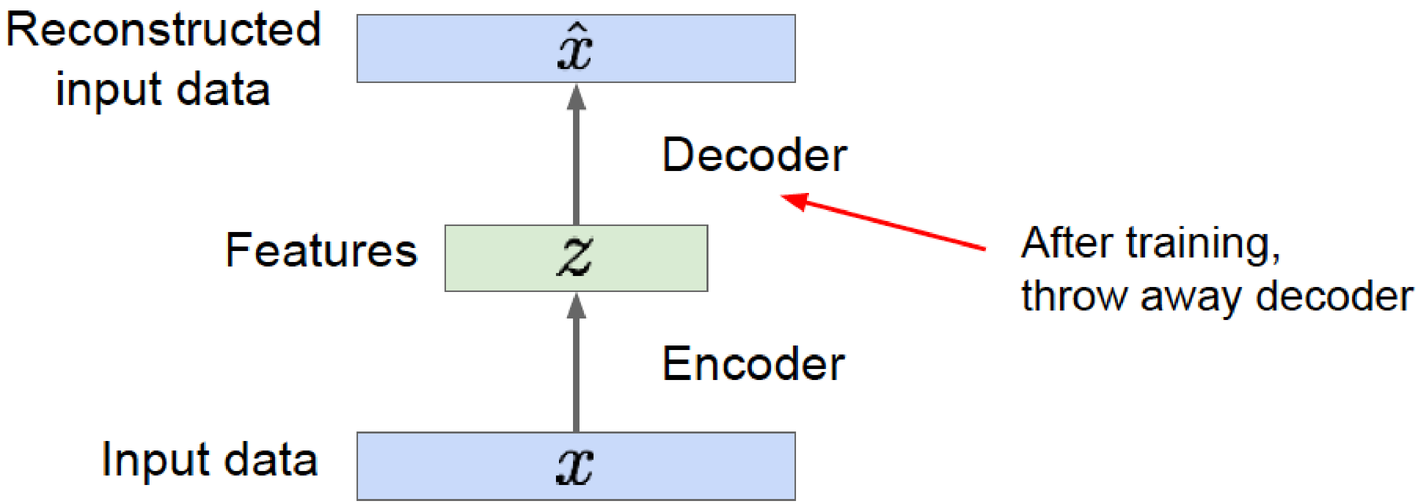
- Feature representation learning

Train such that features can be used to reconstruct original data



Autoencoder

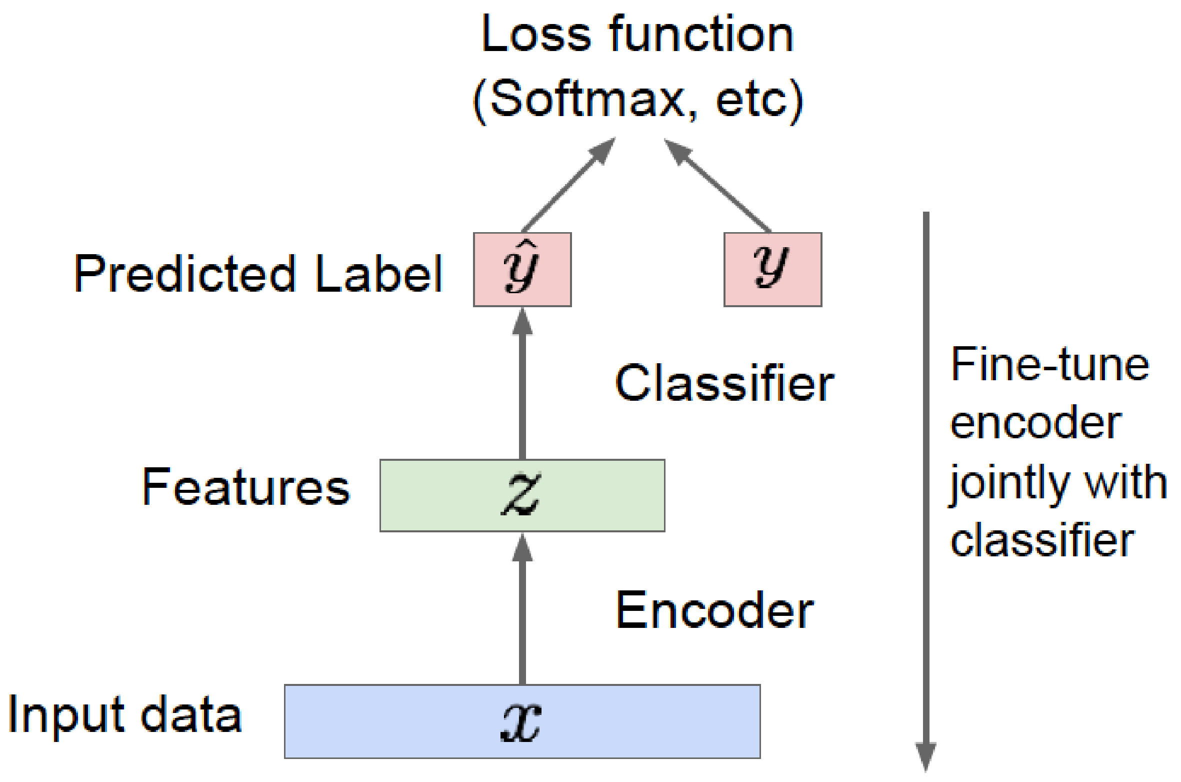
- Feature representation learning



Autoencoder

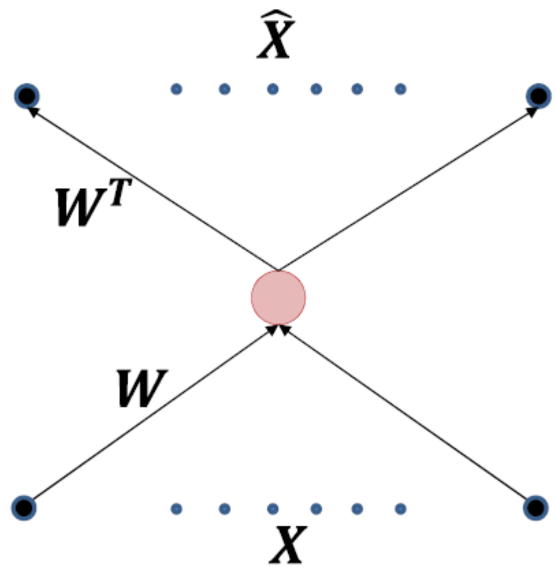
- Feature representation learning

Encoder can be used to initialize a **supervised** model



Autoencoder

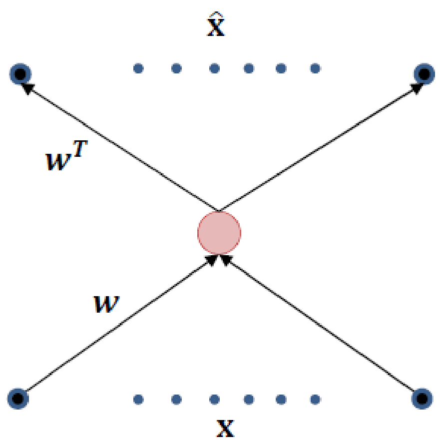
- Linear hidden layer example



- A single hidden unit
- Hidden unit has *linear* activation
- What will this learn?

Autoencoder

- Linear hidden layer example



Training: Learning W by minimizing L2 divergence

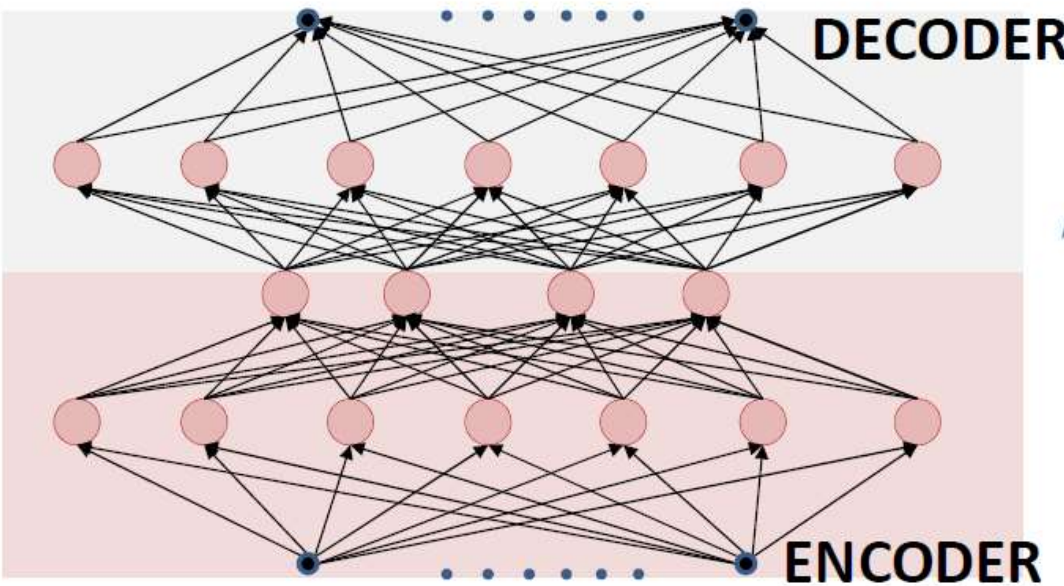
$$\hat{x} = w^T w x$$
$$div(\hat{x}, x) = \|x - \hat{x}\|^2 = \|x - w^T w x\|^2$$

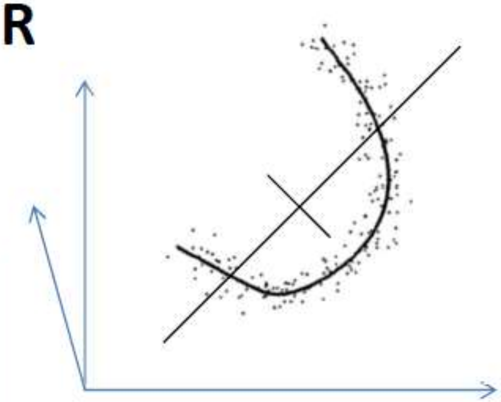
$$\hat{W} = \underset{W}{\operatorname{argmin}} E[div(\hat{x}, x)]$$
$$\hat{W} = \underset{W}{\operatorname{argmin}} E[\|x - w^T w x\|^2]$$

- This is just PCA!

Autoencoder

- Nonlinear hidden layer

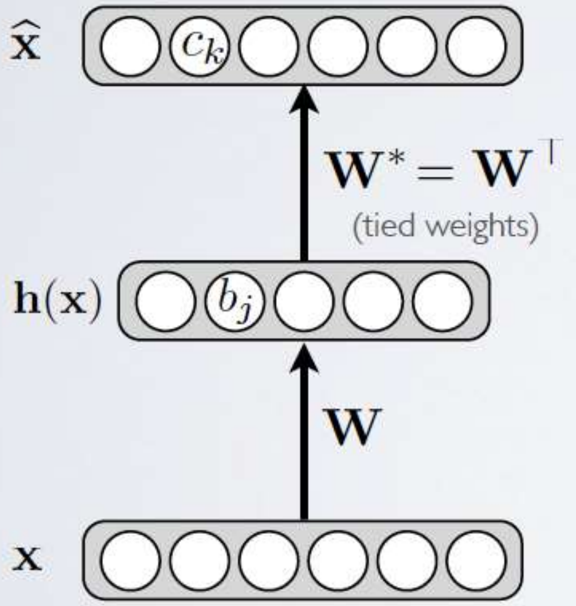




- With non-linearity
  - "Non linear" PCA
  - Deeper networks can capture more complicated manifolds
    - "Deep" autoencoders

Autoencoder

- Binary input example



Decoder

$$\begin{aligned}\widehat{\mathbf{x}} &= o(\widehat{\mathbf{a}}(\mathbf{x})) \\ &= \underbrace{\mathrm{sigm}}_{\text{for binary inputs}}(\mathbf{c} + \mathbf{W}^*\mathbf{h}(\mathbf{x}))\end{aligned}$$

Encoder

$$\begin{aligned}\mathbf{h}(\mathbf{x}) &= g(\mathbf{a}(\mathbf{x})) \\ &= \mathrm{sigm}(\mathbf{b} + \mathbf{W}\mathbf{x})\end{aligned}$$

$$l(f(\mathbf{x})) = -\sum_k \left(x_k \log(\widehat{x}_k) + (1 - x_k)\log(1 - \widehat{x}_k)\right)$$

- cross-entropy (more precisely: sum of Bernoulli cross-entropies)

Regularization

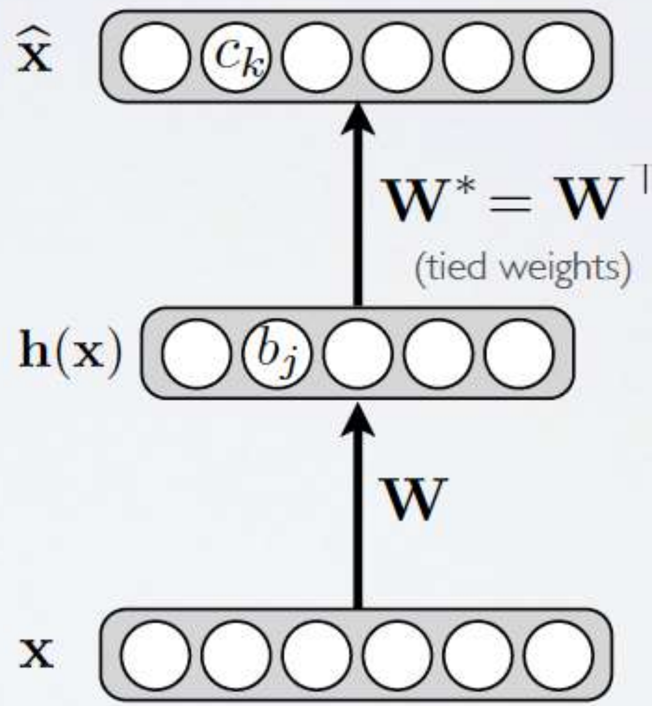
- Regularized autoencoders: add regularization term that encourages the model to have other properties
  - Sparsity of the representation (sparse autoencoder)
  - Robustness to noise or to the missing inputs (denoising autoencoder)
  - Smallness of the derivative of the representation (contracitve autoencoder)

Regularization

- Undercomplete representation

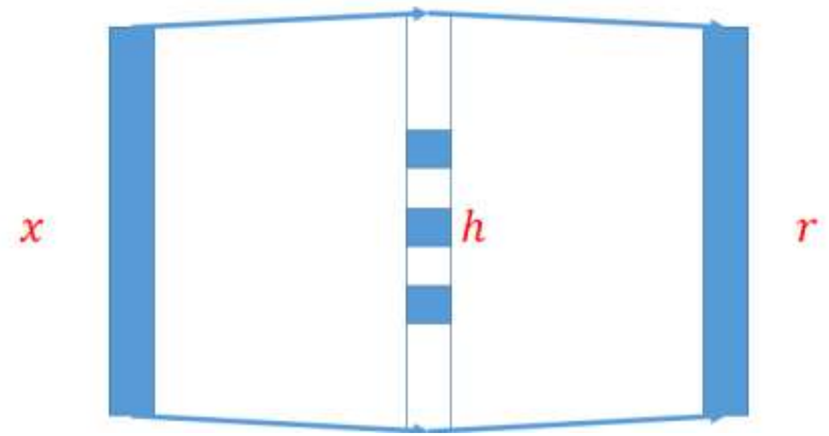
  - Hidden layer is undercomplete if smaller than the input layer
    - hidden layer "compresses" the input
    - will compress well only for the training distribution
  - Hidden units will be
    - good features for the training distribution

    - but bad for other types of input



Regularization

$$L_R = L(x, g(f(x))) + R(h)$$



- Sparse autoencoder
  - Constrain the code to have sparsity
  - Training: minimize a loss function

$$L_R = L(x, g(f(x))) + \lambda|h|_1$$

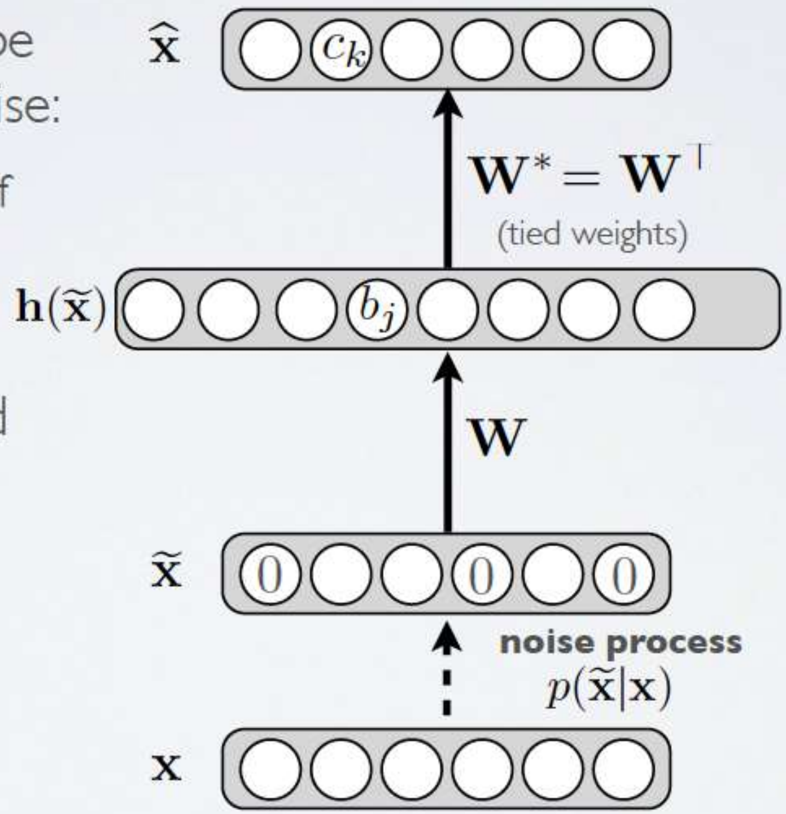
Regularization

- Denoising autoencoder

• Idea: representation should be robust to introduction of noise:
  - random assignment of subset of inputs to 0, with probability ν
  - Gaussian additive noise

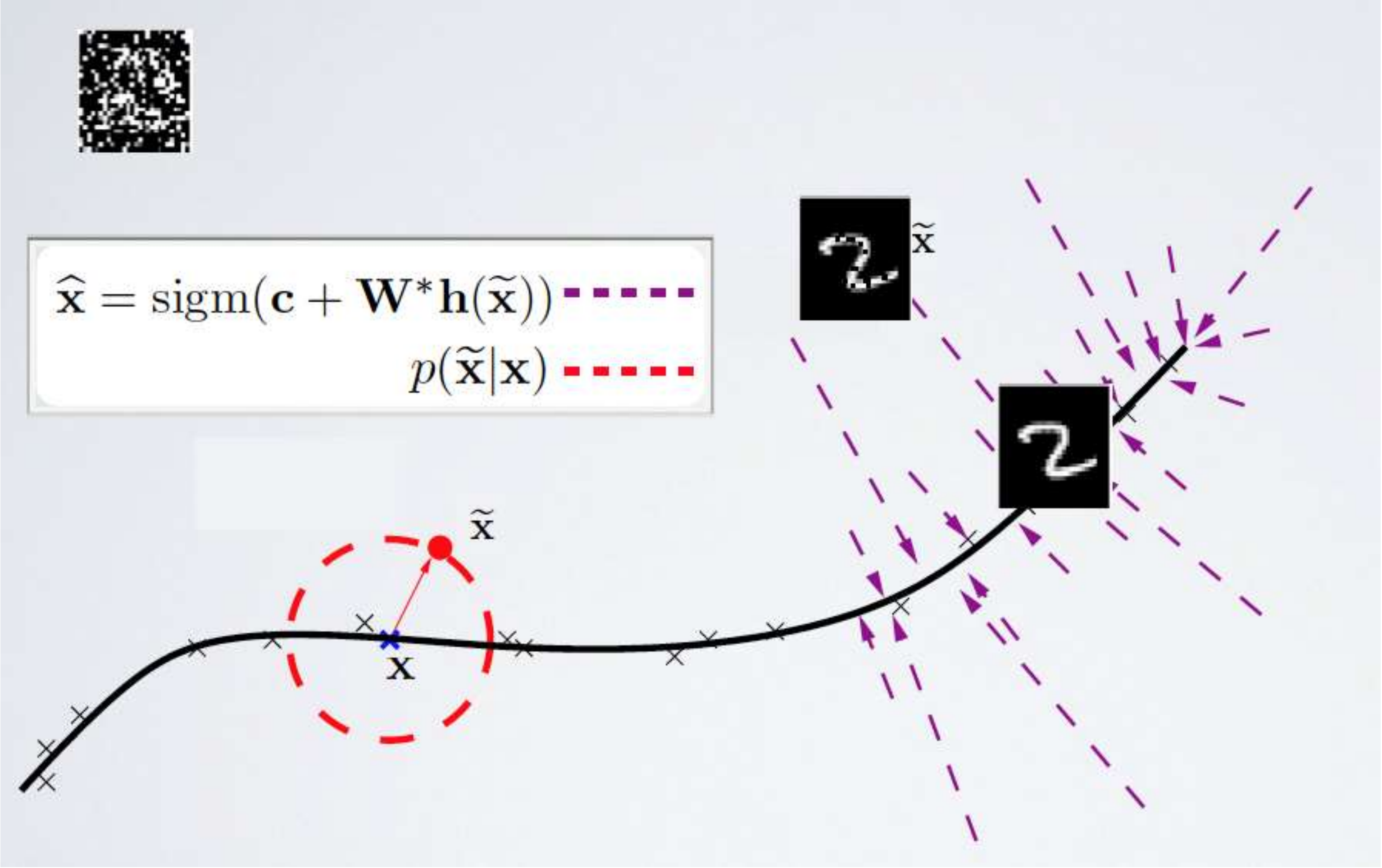
• Reconstruction $\widehat{\mathbf{x}}$ computed from the corrupted input $\widetilde{\mathbf{x}}$

• Loss function compares $\widehat{\mathbf{x}}$ reconstruction with the **noiseless input** $\mathbf{x}$



Regularization

- Denoising autoencoder



Summary

- Four typical problems in unsupervised learning:
  - Clustering, dimensionality reduction, representation learning, density estimation
- Latent variable model and EM algorithm:
  - ELBO and free energy
- Representation learning with AutoEncoder:
  - Reconstructing data with constraints
- Reading material
  - http://cs229.stanford.edu/notes2020spring/cs229-notes7a.pdf
  - http://cs229.stanford.edu/notes2020spring/cs229-notes7b.pdf
  - http://cs229.stanford.edu/notes2020spring/cs229-notes8.pdf

Acknowledgement: Yingyu Liang@Princeton's & Feifei Li's cs231n notes