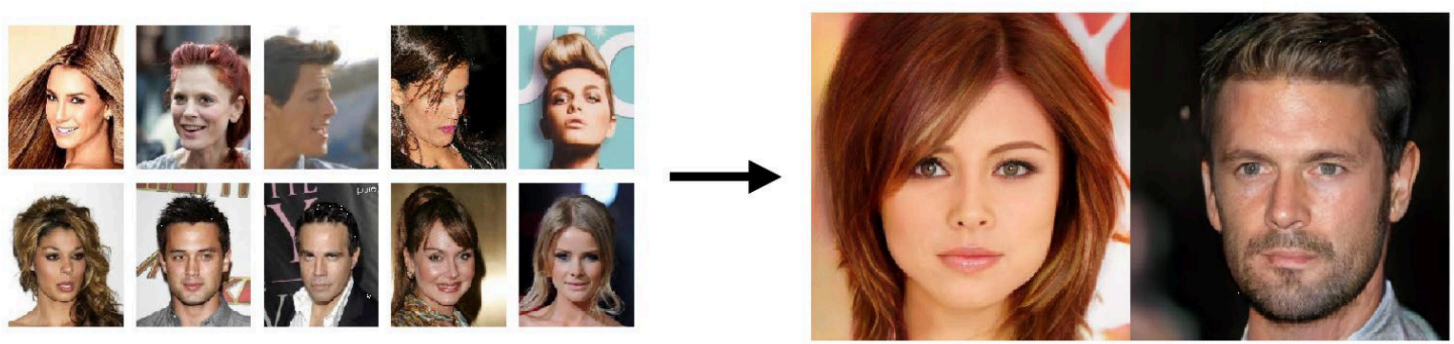
auto regressive language

Generative Models

AIGC: AI Generated Content

Distribution learning

- learn a distribution p_θ
- sample from p_θ



Training Data(CelebA)

Model Samples (Karras et.al., 2018)

4 years of progression on Faces

Brundage et al., 2017

Image credits to Andrej Risteski

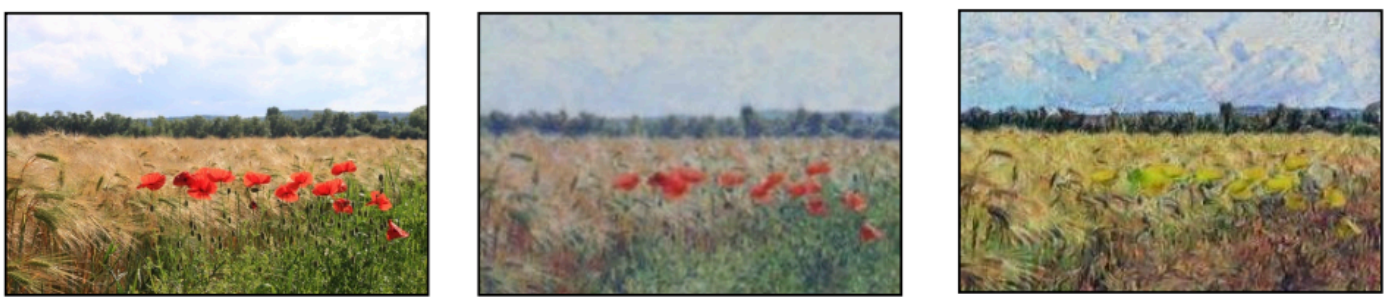
Distribution learning

BigGAN, Brock et al '18

Distribution learning

Conditional generative model P(zebra images | horse images)

Style Transfer



Input Image Monet Van Gogh

Image credits to Andrej Risteski

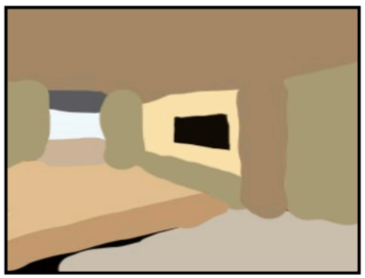
Distribution learning

Source actor

Target actor

Real-time reenactment

Generative model



Generative model
of realistic images

Generate

Stroke paintings to realistic images
[Meng, He, Song, et al., ICLR 2022]

"Ace of Pentacles"

Generative model
of paintings

Generate

Language-guided artwork creation
https://chainbreakers.kath.io @RiversHaveWings

Slides credit to Yang Song

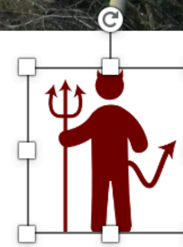
Generative model

given x, evaluate $p_\theta(x)$

High probability

Generative model of traffic signs

Low probability



Outlier detection
[Song et al., ICLR 2018]

Slide credit to Yang Song

Desiderata for generative models

p_θ

- **Probability evaluation**: given a sample, it is computationally efficient to evaluate the probability of this sample.

Given x, compute $p_\theta(x)$ efficiently

- **Flexible model family:** it is easy to incorporate any neural network models.

p_θ: CNN, transformer

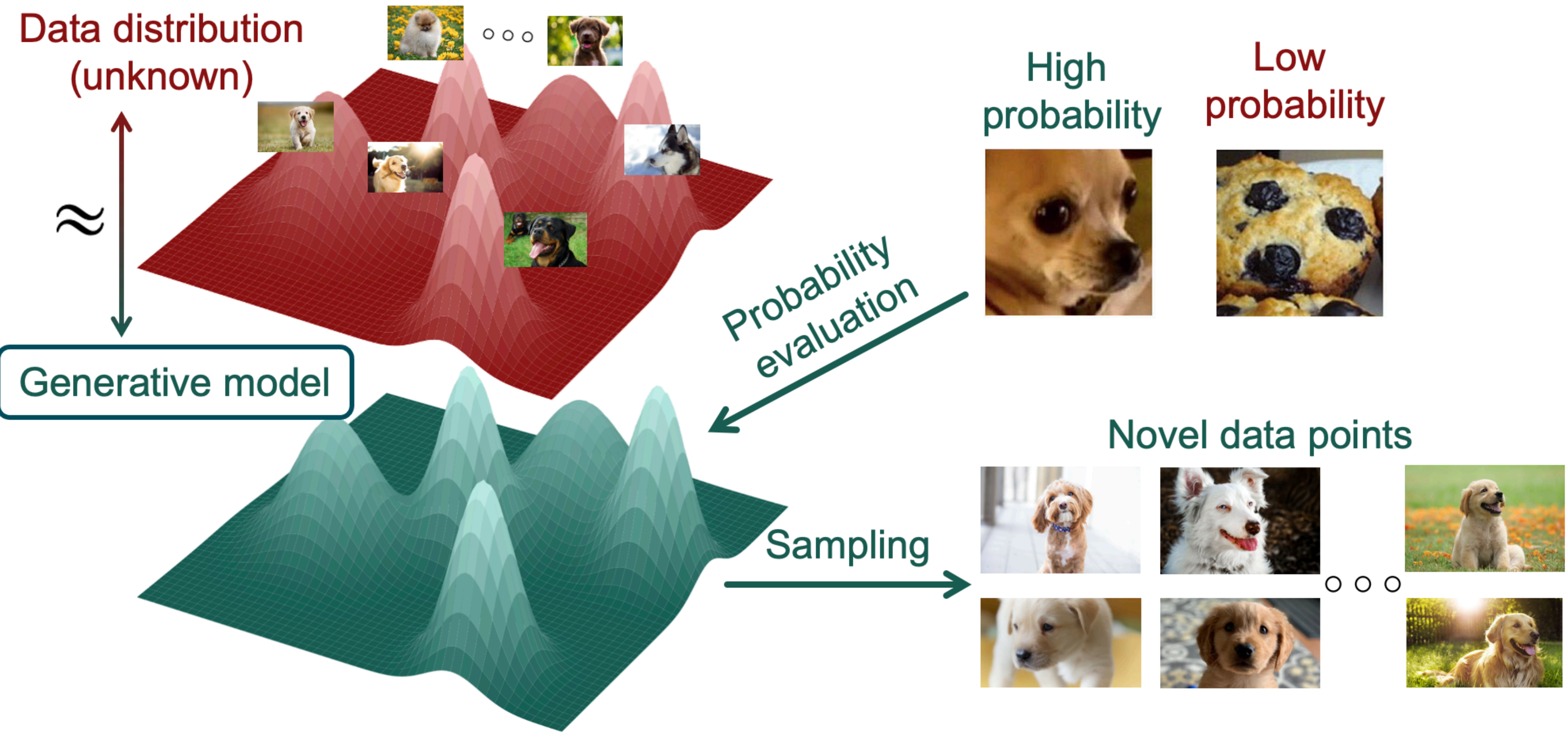
- **Easy sampling:** it is computationally efficient to sample a data from the probabilistic model.

Given p_θ

sample n data point $\Rightarrow \hat{p}_\theta$

$\hat{p}_\theta$ is close to p_θ

Desiderata for generative models

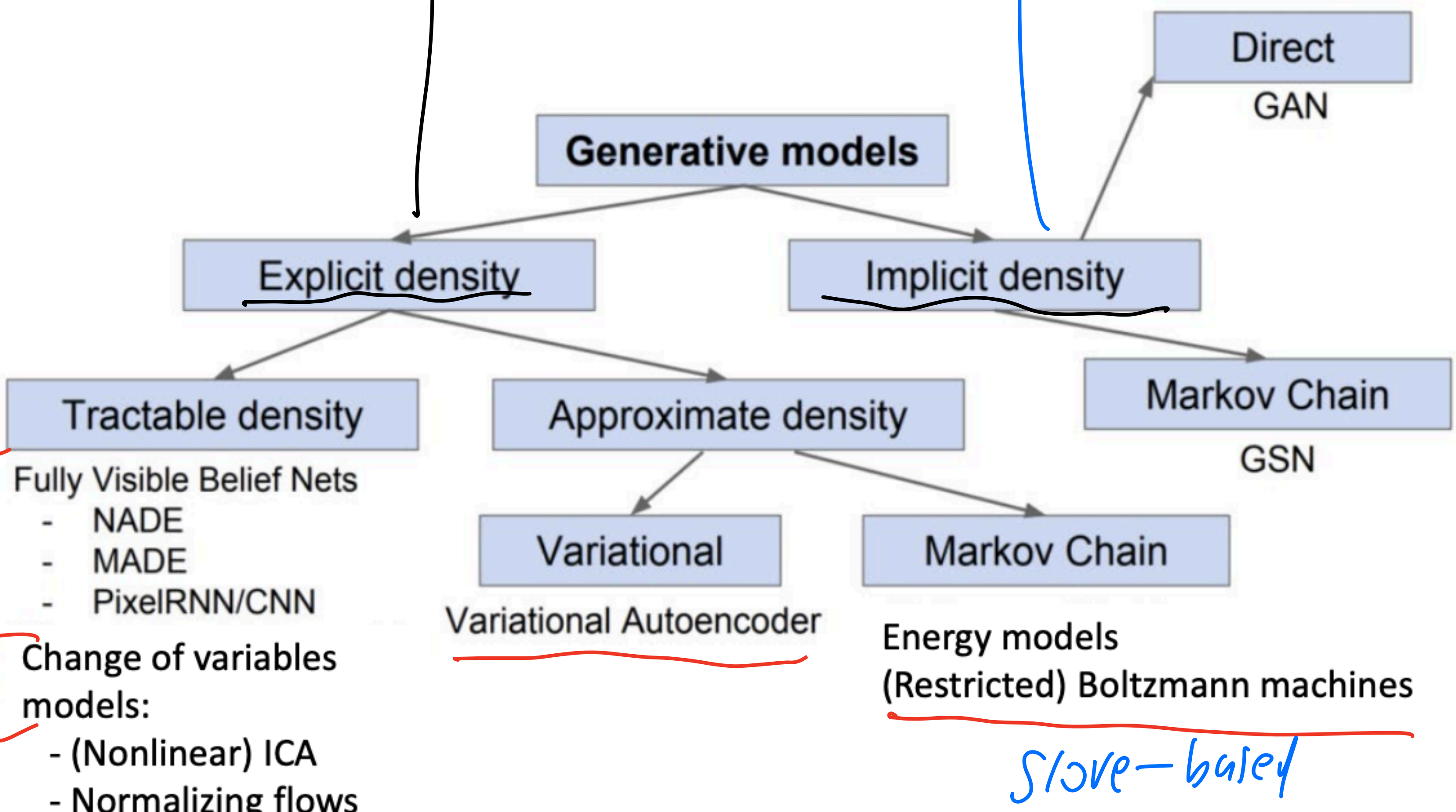


Slide credit to Yang Song

Taxonomy of generative models

given x, explicit expression $p_\theta(x)$

example
$z \sim N(0, I)$
$x = G_\theta(z)$



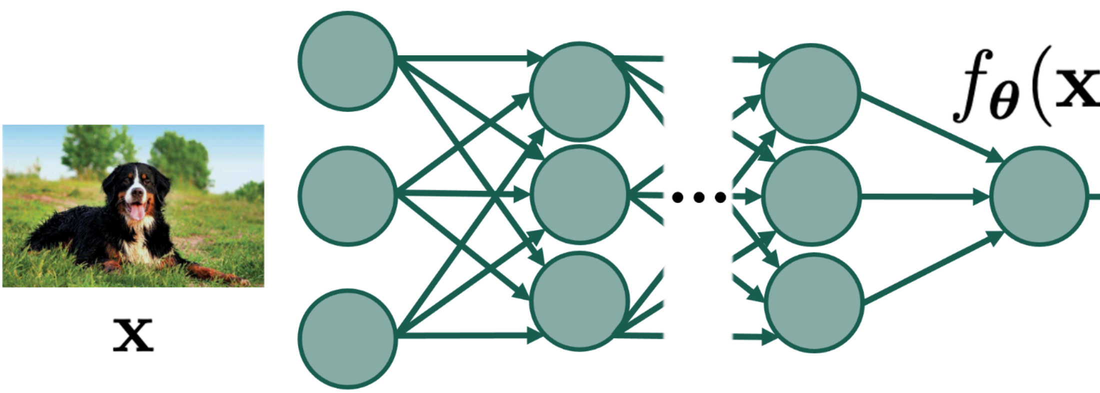
score-based
diffusion

Image credits to Andrej Risteski

Key challenge for building generative models

distribution

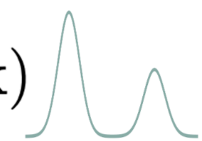
$$\begin{cases} p_\theta(x) \geq 0 \\ \int_x p_\theta(x)\,dx = 1 \end{cases}$$



$\mathbf{x}$

$f_{\boldsymbol{\theta}}(\mathbf{x})$ $e^{f_{\boldsymbol{\theta}}(\mathbf{x})}$

$$\frac{e^{f_{\boldsymbol{\theta}}(\mathbf{x})}}{\boxed{Z_{\boldsymbol{\theta}}}} = p_{\boldsymbol{\theta}}(\mathbf{x})$$



Normalizing constant

$$Z_\theta = \int_x e^{f_\theta(x)}\,dx$$

NP-hard

Key challenge for building generative models

Approximating the normalizing constant

- Variational auto-encoders [Kingma & Welling 2014, Rezende et al. 2014]
- Energy-based models [Ackley et al. 1985, LeCun et al. 2006]

Inaccurate probability evaluation

Using restricted neural network models

- Autoregressive models [Bengio & Bengio 2000, van den Oord et al. 2016]
- Normalizing flow models [Dinh et al. 2014, Rezende & Mohamed 2015]

Restricted model family

Generative adversarial networks (GANs)

- Model the generation process, not the probability distribution [Goodfellow et al. 2014]

Cannot evaluate probabilities

Slide credit to Yang Song

Training generative models

- **Likelihood-based:** maximize the likelihood of the data under the model (possibly using advanced techniques such as variational method or MCMC):

$$\max_{\theta} \sum_{i=1}^{n} \log p_{\theta}(x_i)$$

- Pros:
  - **Easy training**: can just maximize via SGD.
  - **Evaluation**: evaluating the fit of the model can be done by evaluating the likelihood (on test data).

- Cons:
  - **Large models needed**: likelihood objectve is hard, to fit well need very big model.
  - **Likelihood entourages averaging:** produced samples tend to be blurrier, as likelihood encourages "coverage" of training data.

Training generative models

- **Likelihood-free:** use a **surrogate loss** (e.g., GAN) to train a discriminator to differentiate real and generated samples.

- Pros:
  - **Better objective, smaller models needed:** objective itself is learned - can result in visually better images with smaller models.

- Cons:
  - **Unstable training:** typically min-max (saddle point) problems.
  - **Evaluation:** no way to evaluate the quality of fit.

GAN

Generative Adversarial Nets

Implicit Generative Model

- **Goal:** a sampler $g(\cdot)$ to generate images
- A simple generator $g(z;\theta)$:
  - $z \sim N(0,I)$
  - $x = g(z;\theta)$ deterministic transformation
- Likelihood-free training:
  - Given a dataset from some distribution p_{data}
  - Goal: $g(z;\theta)$ defines a distribution, we want this distribution $\approx p_{data}$
  - Training: minimize $D(g(z;\theta), p_{data})$
    - D is some distance metric (not likelihood)
  - Key idea: ***Learn* a differentiable** D

$\{x_i\}_{i=1}^{n}$

Metrics:

(1) Kullback-Leibler divergence

(2) Total variation:

$\int |P_1(x) - P_2(x)| dx$

(3) Wasserstein distance

(4) Jensen-Shannon Divergence

(5) Integral Probability Metric

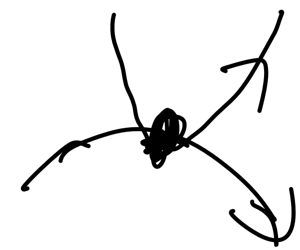
GAN (Goodfellow et al., '14)

- Parameterize the discriminator $D(\cdot\,;\phi)$ with parameter ϕ
- **Goal:** learn ϕ such that $D(x;\phi)$ measures how likely x is from p_{data}
  - $D(x,\phi) = 1$ if $x \sim p_{data}$
  - $D(x,\phi) = 0$ if $x! \sim p_{data}$
  - a.k.a., a binary classifier
- GAN: use a neural network for $D(\cdot\,;\phi)$
- **Training:** need both negative and positive samples
  - Positive samples: just the training data
  - Negative samples: use our sampler $g(\cdot\,;z)$ (can provide infinite samples).
- **Overall objectives:**
  - Generator: $\theta^* = \max_{\theta} D(g(z;\theta);\phi)$
  - Discriminator uses MLE Training:

$$\phi^* = \max_{\phi} \mathbb{E}_{x\sim p_{data}}[\log D(x;\phi)] + \mathbb{E}_{\hat{x}\sim g(\cdot)}[\log(1 - D(\hat{x};\phi))]$$

GAN (Goodfellow et al., '14)

- Generator $G(z;\theta)$ where $z \sim N(0,I)$
  - Generate realistic data
- Discriminator $D(x;\phi)$
  - Classify whether the data is real (from p_{data}) or fake (from G)



- Objective function: Saddle point optimization

$$L(\theta,\phi) = \underbrace{\min_{\theta} \max_{\phi}} \mathbb{E}_{x\sim p_{data}}\left[\log D(x;\phi)\right] + \mathbb{E}_{\hat{x}\sim G}\left[\log(1 - D(\hat{x};\phi))\right]$$

- Training procedure:
  - Collect dataset $\{(x,1)\,|\,x \sim p_{data}\} \cup \{(\hat{x},0) \sim g(z;\theta)\}$
  - Train discriminator
    $D: L(\phi) = \mathbb{E}_{x\sim p_{data}}\left[\log D(x;\phi)\right] + \mathbb{E}_{\hat{x}\sim G}\left[\log(1 - D(\hat{x};\phi))\right]$
  - Train generator $G: L(\theta) = \mathbb{E}_{z\sim N(0,I)}\left[\log D(G(z;\theta),\phi)\right]$
  - Repeat

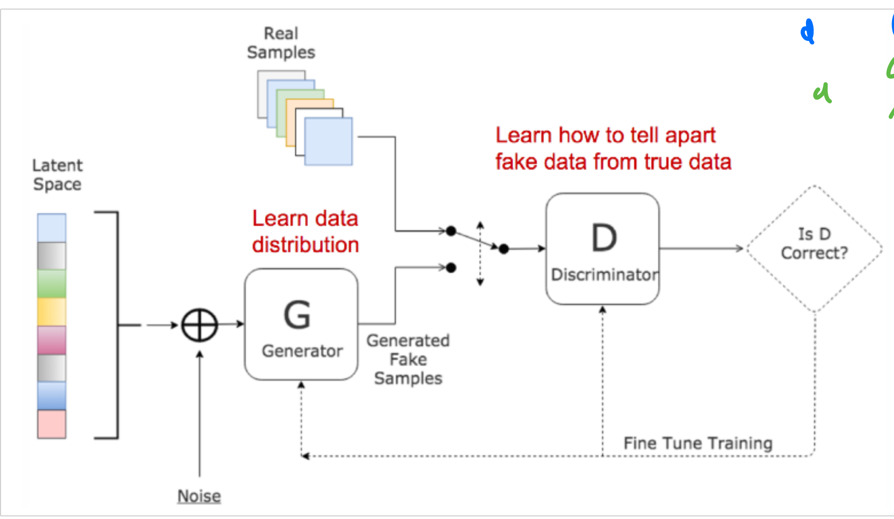
GAN (Goodfellow et al., '14)

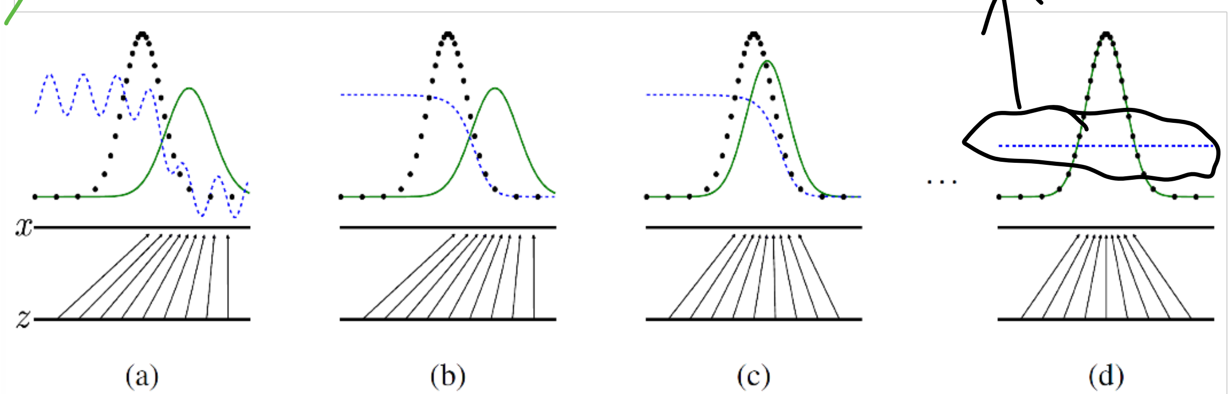
- Objective function:

$$L(\theta, \phi) = \min_{\theta} \max_{\phi} \mathbb{E}_{x \sim p_{data}} \left[\log D(x; \phi)\right] + \mathbb{E}_{\hat{x} \sim G} \left[\log(1 - D(\hat{x}; \phi))\right]$$

- Pdata
- discriminator
- generator

constant = 0.5





Math Behind GAN

$$L(\theta,\phi) = \min_{\theta} \max_{\phi} \mathbb{E}_{x\sim p_{data}}[\log D(x;\phi)] + \mathbb{E}_{\hat{x}\sim g(\cdot|\theta)}[\log(1-D(\hat{x};\phi))]$$

- Let D^*, g^* be the solution of
- Consider a given x

$$L_x(D) = p_{data}(x)\cdot\log D(x) + p_g(x)\cdot\log(1-D(x))$$

- D^* : $\frac{\partial L}{\partial D} = 0$ first-order condition

$$\Rightarrow \frac{p_{data}(x)}{D^*(x)} - \frac{p_g(x)}{1-D^*(x)} = 0$$

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

if $p_g = p_{data} \Rightarrow D^*(x) = 0.5$

Math Behind GAN

Consider optimal generator G^*, given optimal D^*

$$L(\theta, \phi) = \mathbb{E}_{x \sim P_{data}}\left[\log\left[\frac{P_{data}(x)}{P_{data}(x)+P_g(x)}\right]\right] + \mathbb{E}_{\hat{x} \sim g(\cdot)}\left[\log \frac{P_g(\hat{x})}{P_{data}(\hat{x})+P_g(\hat{x})}\right]$$

$$= \mathbb{E}_{x \sim P_{data}}\left[\log\left[\frac{P_{data}(x)}{\frac{P_{data}(x)+P_g(x)}{2}}\right]\right] + \mathbb{E}_{\hat{x} \sim g(\cdot)}\left[\log\left[\frac{P_g(\hat{x})}{\frac{P_{data}(x)+P_g(\hat{x})}{2}}\right]\right]$$

$$- \log 4$$

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

$$= KL\left(P_{data} \,\Big\|\, \frac{1}{2}(P_{data} + P_g)\right) + KL\left(P_g \,\Big\|\, \frac{1}{2}(P_{data} + P_g)\right)$$

$$= 2 \cdot \text{Jensen-Shannon Divergence}$$

KL-Divergence and JS-Divergence

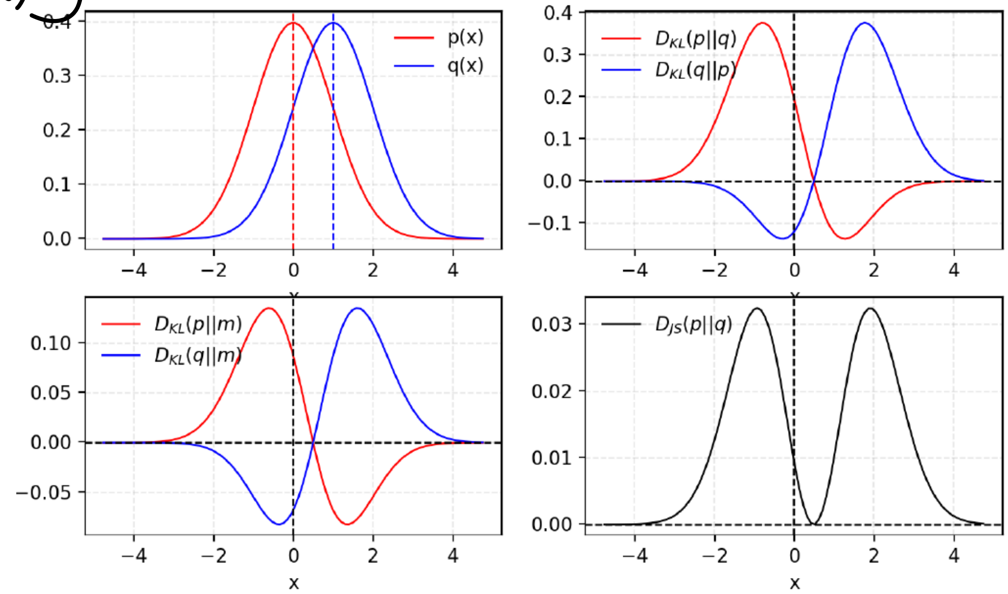
- $KL(p \| q) = \mathbb{E}_{x \sim p}\left[\log \frac{p(x)}{q(x)}\right]$

- $KL \geq 0$

  $KL = 0 \iff p = q$

$\Rightarrow JSD \geq 0$

$JSD = 0 \iff p = q$



Math Behind GAN

Given D^*

$$L(G) = 2\,JSD(P_g \| P_{data}) - \log 4$$

$\Rightarrow$ G^* must satisfy $P_{g^*} = P_{data}$

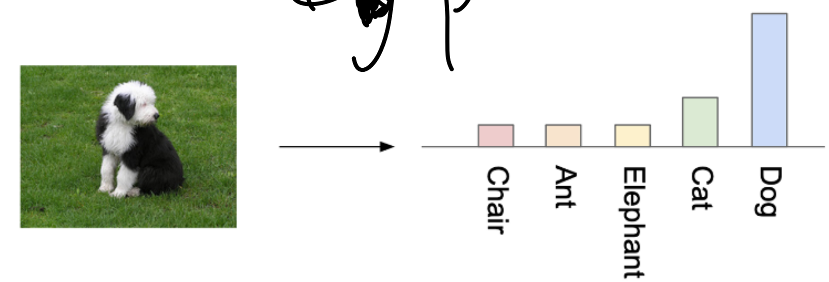
$\Rightarrow$ $L^* = -\log 4$

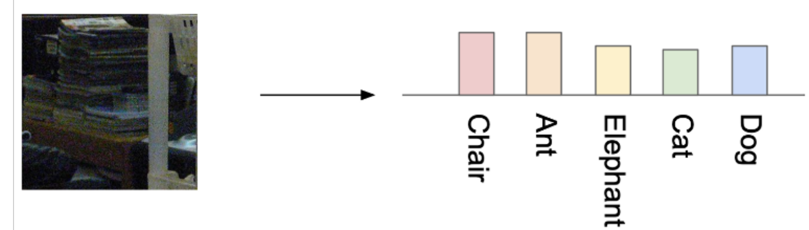
$\underline{D}$

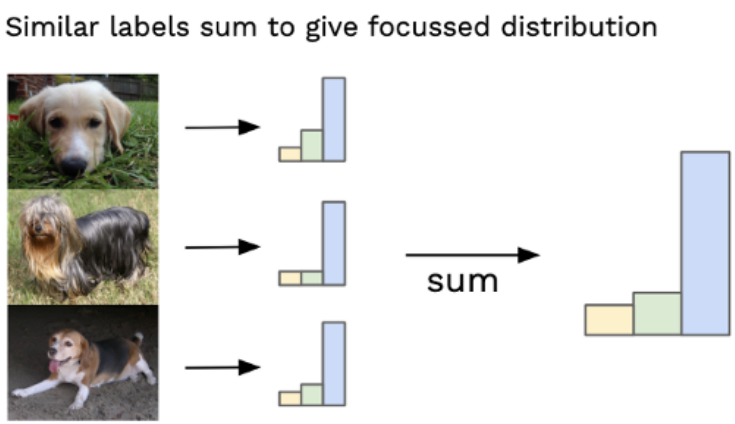
Evaluation of GAN

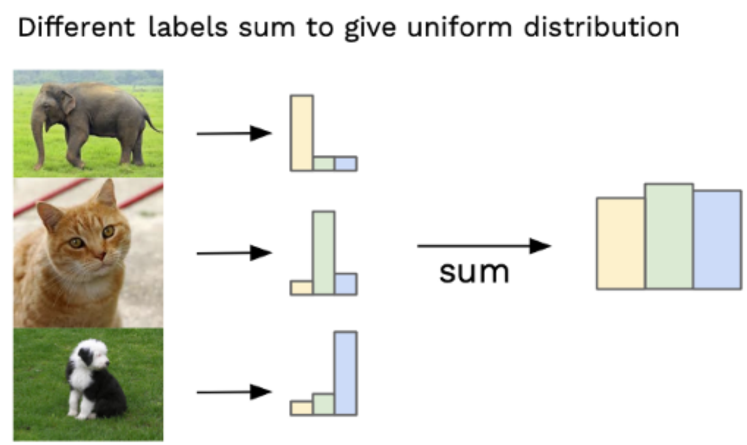
$\mathbb{E}_{y \sim p} \log(P(y))$

- No $p(x)$ in GAN.
- Idea: use a trained classifier $f(y \mid x)$:
- If $x \sim p_{data}$, $f(y|x)$ should have low entropy
  - Otherwise, $f(y \mid x)$ close to uniform.
- Samples from G should be diverse:
  - $p_f(y) = \mathbb{E}_{x \sim G}[f(y|x)]$ close to uniform.



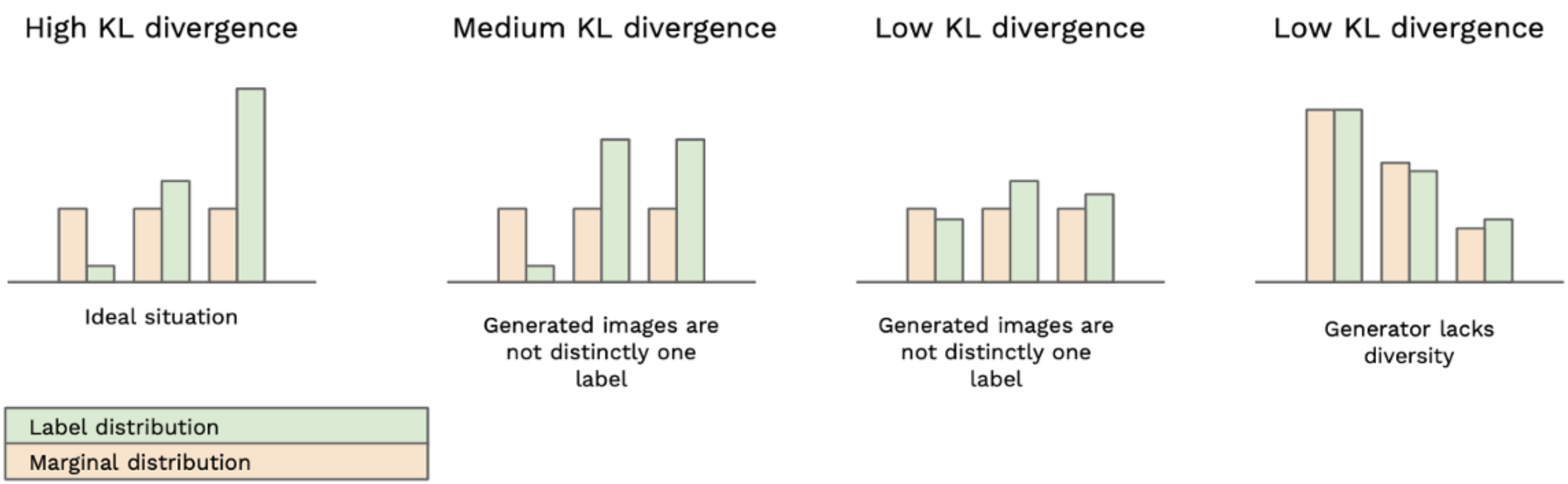






Evaluation of GAN

- **Inception Score** (IS, Salimans et al. '16)
  - Use Inception V3 trained on ImageNet as $f(y \mid x)$
  - $IS = \exp\left(\mathbb{E}_{x \sim G}\left[KL(\underbrace{f(y \mid x)}_{\text{dist for given } x} \,||\, \underbrace{p_f(y)}_{\text{marginal dist over classes}})\right]\right)$
  - Higher the better



Comments on GAN

- Other evaluation metrics:
  - Fréchet Inception Distance (FID): Wasserstein distance between Gaussians
- Mode collapse:
  - The generator only generate a few type of samples.
  - Or keep oscillating over a few modes.

$\min_x \max_y -xy$

- Training instability:
  - Discriminator and generator may keep oscillating
  - Example: $-xy$, generator x, discriminator y. NE: $x = y = 0$ but GD oscillates.
  - No stopping criteria.
  - Use Wsserstein GAN (Arjovsky et al. '17):

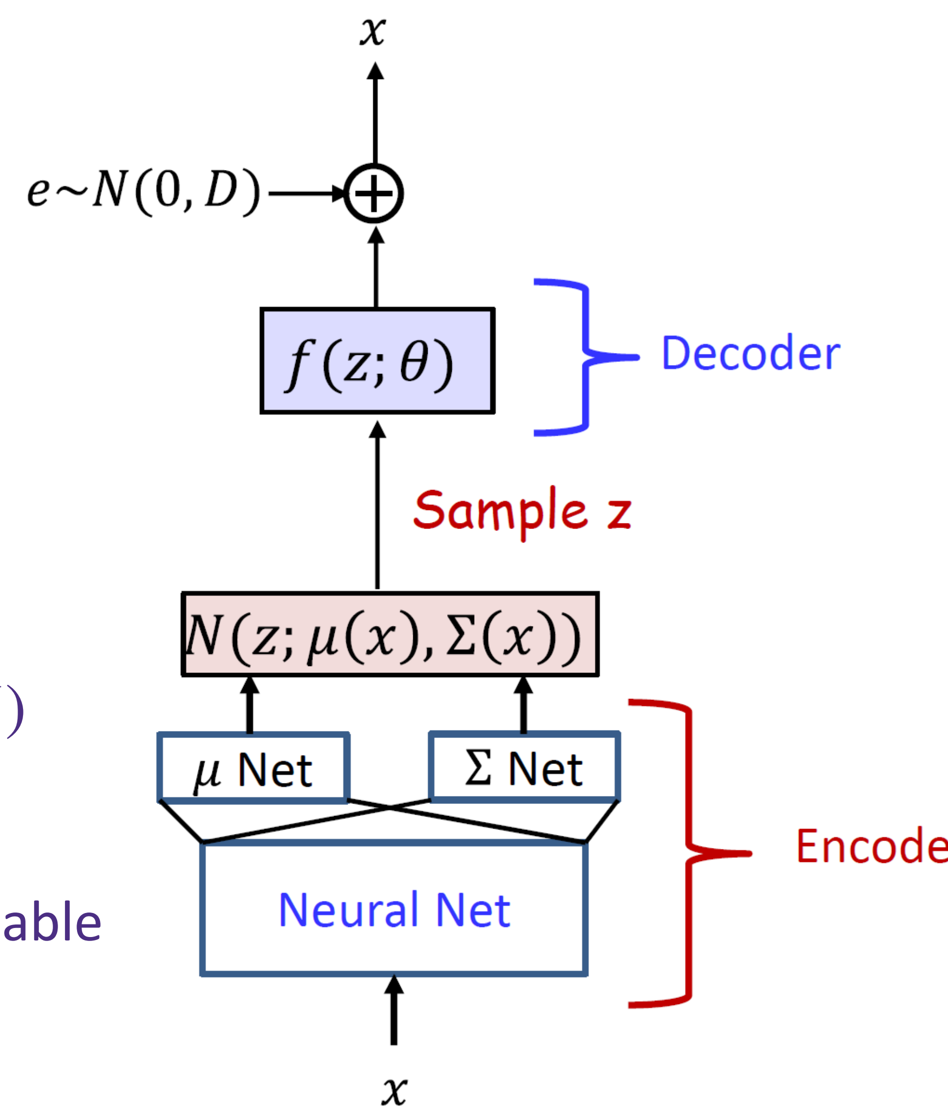
$$\min_G \max_{f:\text{Lip}(f)\le 1} \mathbb{E}_{x\sim p_{data}}\left[f(x)\right] - \mathbb{E}_{\hat{x}\sim p_G}[f(\hat{x})]$$

  - And need many other tricks…

Variational Autoencoder

Architecture

- Auto-encoder: $x \to z \to x$
- Encoder: $q(z \mid x; \phi) : x \to z$
- Decoder: $p(x \mid z; \theta) : z \to x$



- Isomorphic Gaussian:

$q(z \mid x; \phi) = N(\mu(x; \phi), \text{diag}(\exp(\sigma(x; \phi))))$

- Gaussian prior: $p(z) = N(0, I)$
- Gaussian likelihood: $p(x \mid z; \theta) \sim N(f(z; \theta), I)$

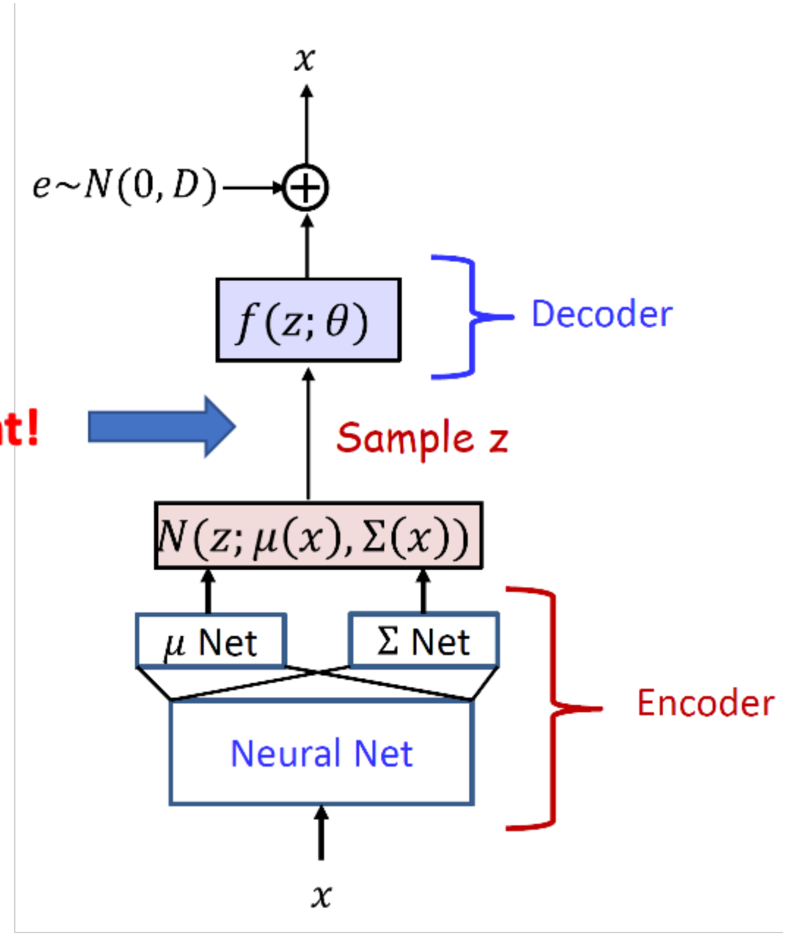
- Probabilistic model interpretation: latent variable model.

VAE Training

- Training via optimizing ELBO
  - $L(\phi, \theta; x) = \mathbb{E}_{z \sim q(z|x;\phi)}[\log p(z|x;\theta)] - KL\left(q(z|x;\phi)||p(z)\right)$
  - Likelihood term + KL penalty

- KL penalty for Gaussians has closed form.
- Likelihood term (reconstruction loss):
  - Monte-Carlo estimation
  - Draw samples from $q(z|x;\phi)$
  - Compute gradient of θ:
    - $x \sim N(f(z;\theta); I)$
    - $p(x) = \frac{1}{\sqrt{2\pi}}\exp(-\frac{1}{2}\|x - f(z;\theta)\|_2^2)$

No gradient!

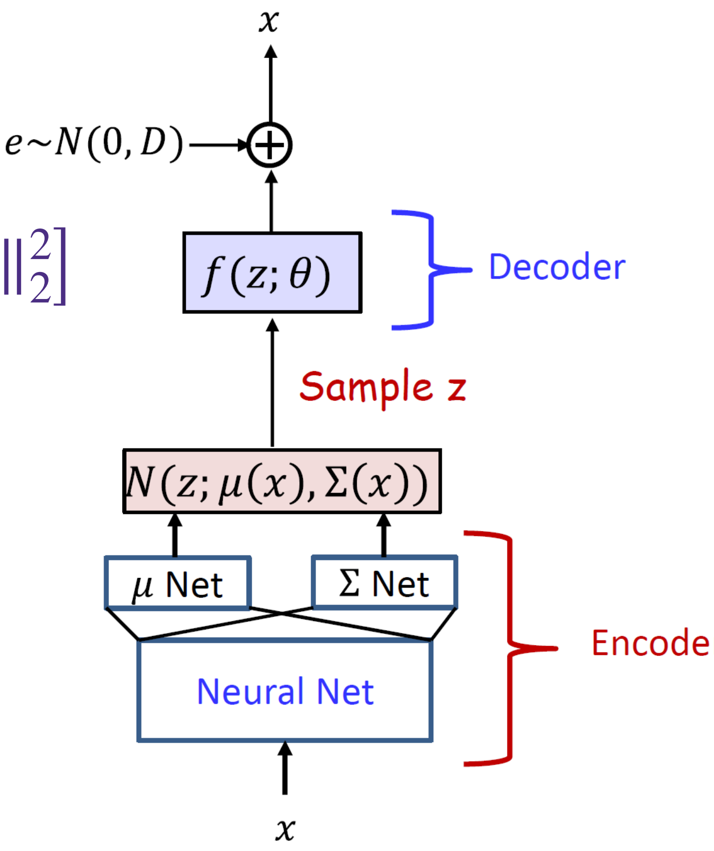


VAE Training

- Likelihood term (reconstruction loss):
  - Gradient for ϕ . Loss: $L(\phi) = \mathbb{E}_{z\sim q(z;\phi)}\left[\log p(x\,|\,z)\right]$
  - Reparameterization trick:
    - $z \sim N(\mu, \Sigma) \Leftrightarrow z = \mu + \epsilon, \epsilon \sim N(0,\Sigma)$
  - $L(\phi) \propto \mathbb{E}_{z\sim q(z|\phi)}\left[\|f(z;\theta) - x\|_2^2\right]$

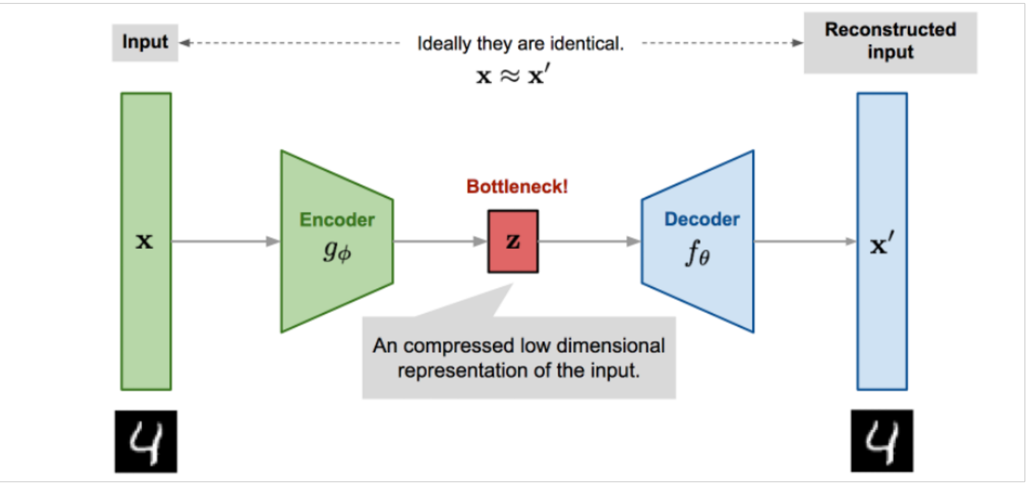
    $\propto \mathbb{E}_{\epsilon\sim N(0,I)}\left[\|f(\mu(x;\phi) + \sigma(x;\phi)\cdot\epsilon;\theta) - x\|_2^2\right]$
  - Monte-Carlo estimate for $\nabla L(\phi)$

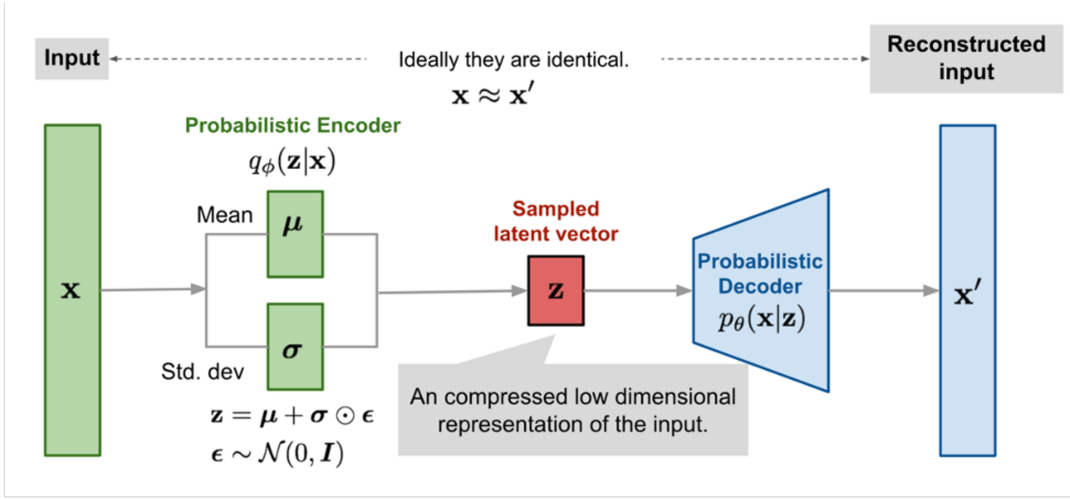
- End-to-end training



VAE vs. AE

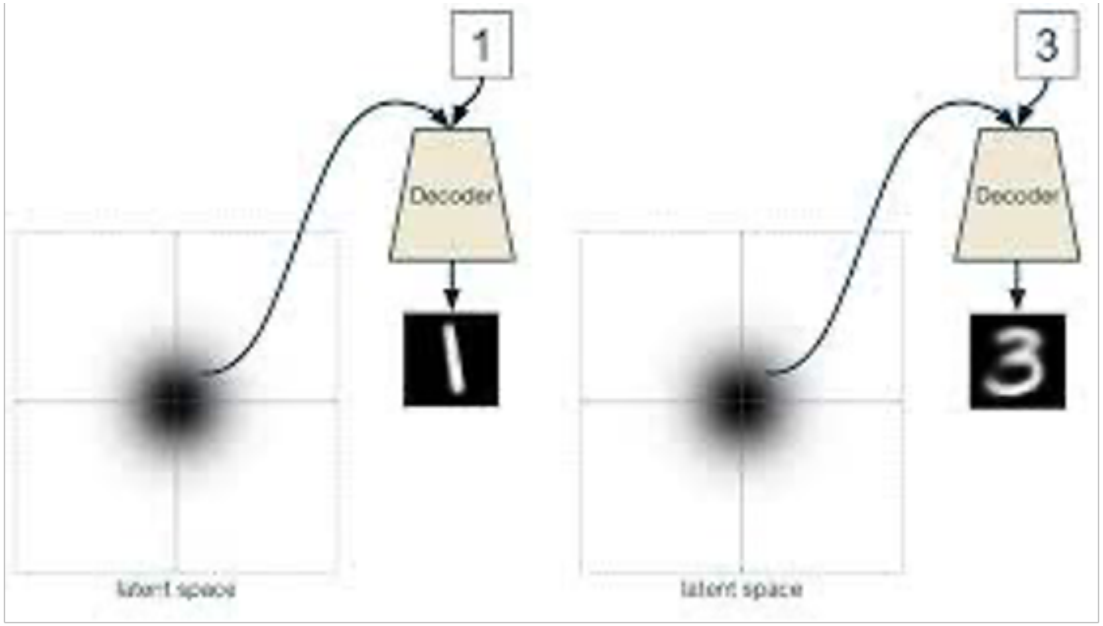
- AE: classical unsupervised representation learning method.
- VAR: a probabilistic model of AE
  - AE + Gaussian noise on z
  - KL penalty: L_2 constraint on the latent vector z





Conditioned VAE

- Semi-supervised learning: some labels are also available



conditioned generation

Comments on VAE

- Pros:
  - Flexible architecture
  - Stable training

- Cons:
  - Inaccurate probability evaluation (approximate inference)

Energy-Based Models

Energy-based Models

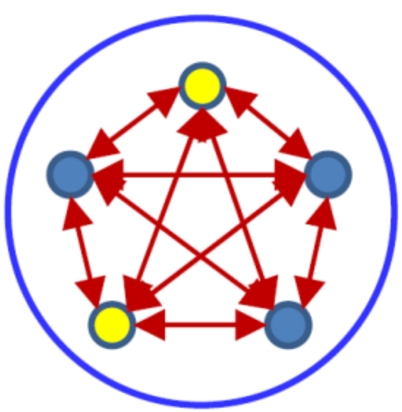
- Goal of generative models:
  - a probability distribution of data: $P(x)$

- Requirements
  - $P(x) \geq 0$ (non-negative)
  - $\int_x P(x)dx = 1$

- Energy-based model:
  - Energy function: $E(x;\theta)$, parameterized by θ
  - $P(x) = \frac{1}{z}\exp(-E(x;\theta))$ (why exp?)
  - $z = \int_z \exp(-E(x;\theta))dx$

Boltzmann Machine

- Generative model
  - $E(y) = -\frac{1}{2}y^\top W y$
  - $P(y) = \frac{1}{z}\exp(-\frac{E(y)}{T})$, T: temperature hyper-parameter
  - W: parameter to learn
- When y_i is binary, patterns are affecting each other through W



$$z_i = \frac{1}{T}\sum_j w_{ji} s_j$$

$$\boxed{P(s_i = 1 | s_{j \neq i}) = \frac{1}{1 + e^{-z_i}}}$$

Boltzmann Machine: Training

- Objective: maximum likelihood learning (assume T =1):
  - Probability of one sample:

$$P(y) = \frac{\exp(\frac{1}{2}y^\top y)}{\sum_{y'} \exp(y'^\top W y')}$$

  - Maximum log-likelihood:

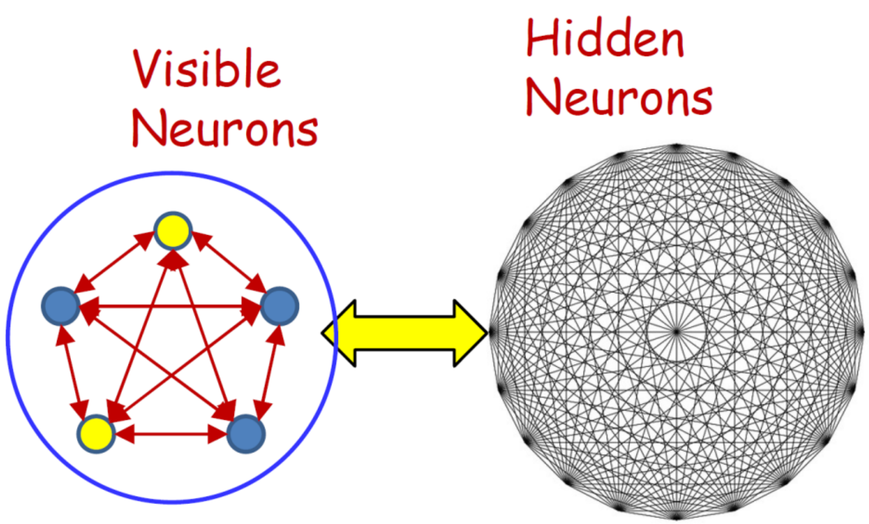
$$L(W) = \frac{1}{N}\sum_{y \in D} \frac{1}{2} y^\top W y - \log \sum_{y'} \exp(\frac{1}{2} y'^\top W y')$$

Boltzmann Machine: Training

Boltzmann Machine: Training

Boltzmann Machine with Hidden Neurons

- Visible and hidden neurons:
  - y: visible, h: hidden
- $P(y) = \sum_{h} P(y, v)$

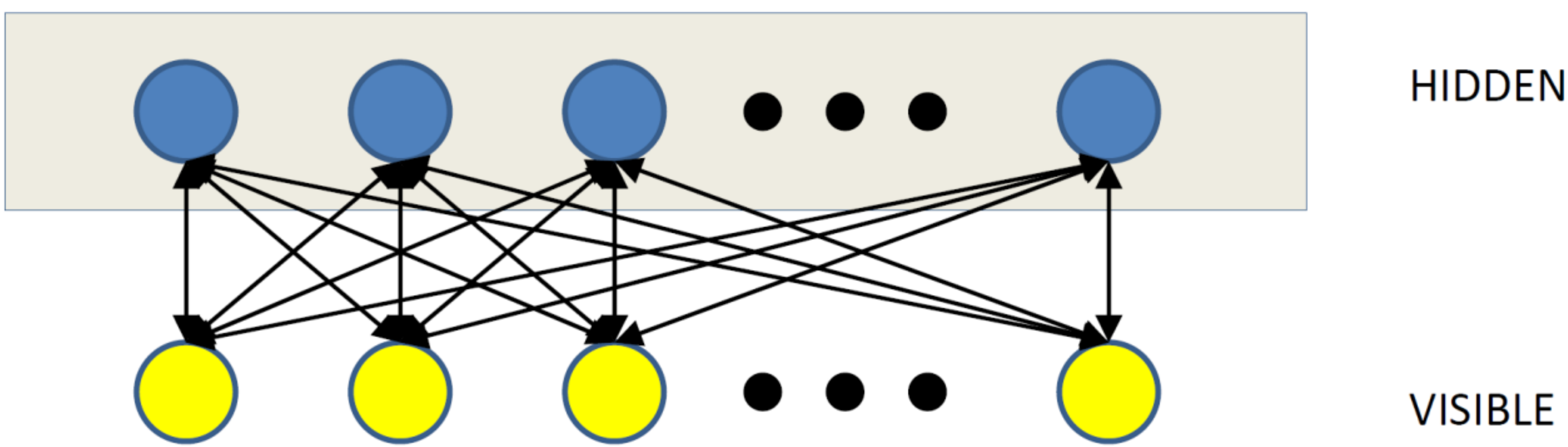


Boltzmann Machine with Hidden Neurons: Training

Boltzmann Machine with Hidden Neurons: Training

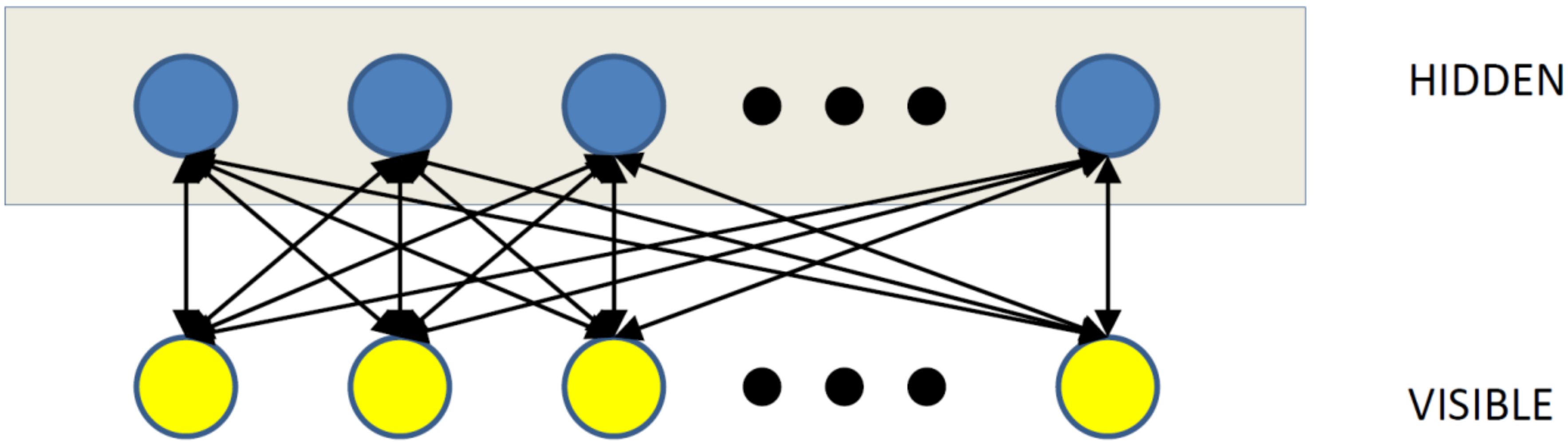
Restricted Bolzmann Machine

- A structured Boltzmann Machine
  - Hidden neurons are only connected to visible neurons
  - No intra-layer connections
  - Invented by Paul Smolensky in ’89
  - Became more practical after Hinton invested fast learning algorithms in mid 2000



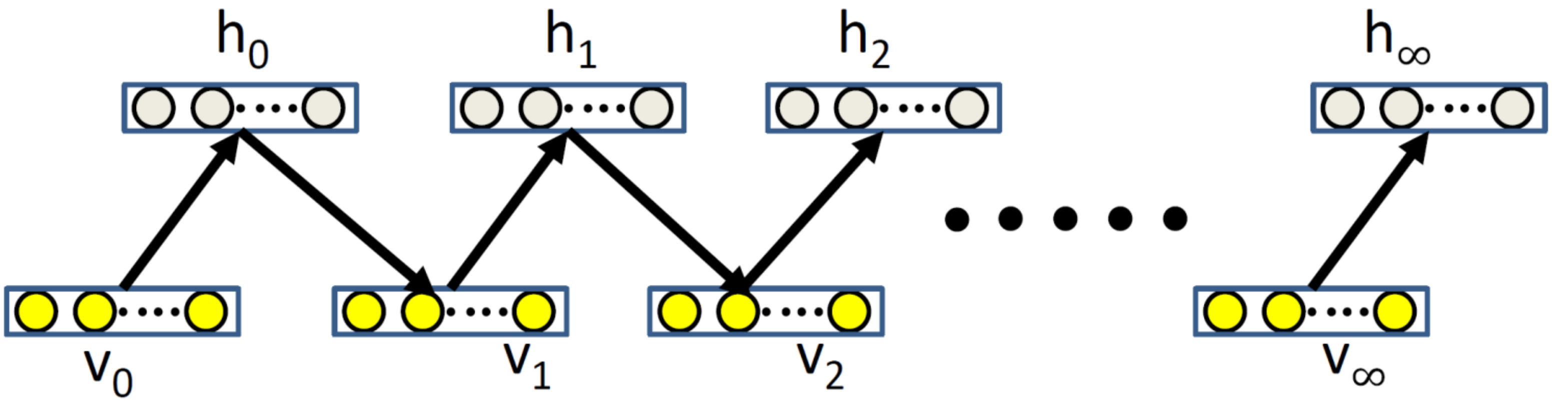
Restricted Bolzmann Machine

- Computation Rules
  - Iterative sampling
  - Hidden neurons h_i: $z_i = \sum_j w_{ij} v_j$, $P(h_i \mid v) = \dfrac{1}{1 + \exp(-z_i)}$
  - Visible neurons v_j: $z_j = \sum_i w_{ij} h_i$, $P(v_j \mid h) = \dfrac{1}{1 + \exp(-z_j)}$



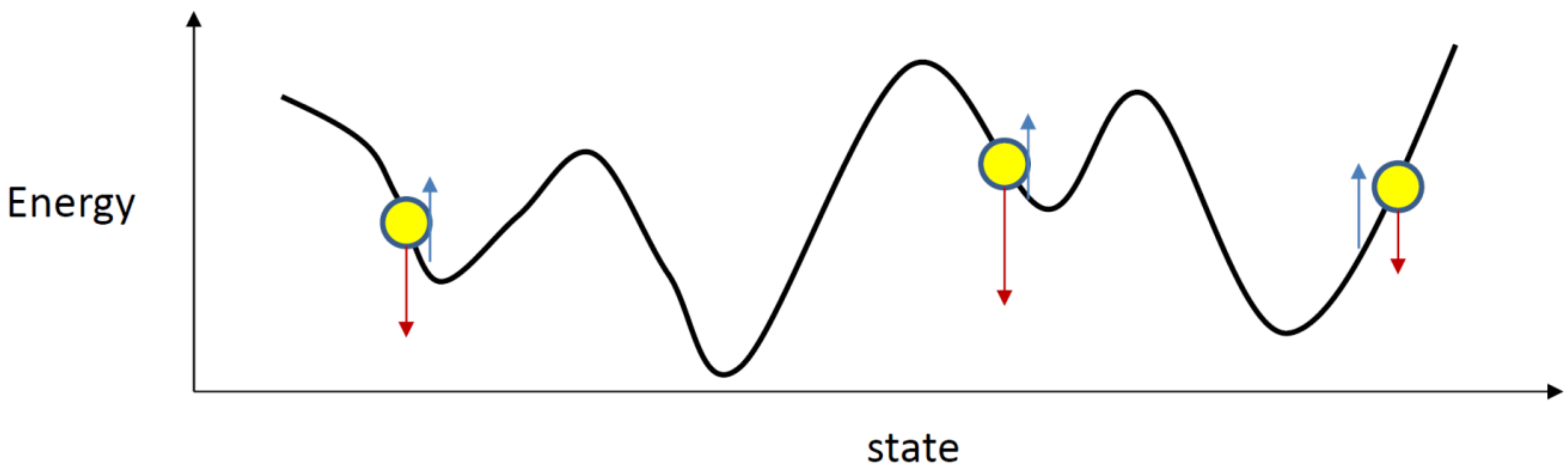
Restricted Bolzmann Machine

- Sampling:
  - Randomly initialize visible neurons v_0
  - Iterative sampling between hidden neurons and visible neurons
  - Get final sample (v_∞, h_∞)



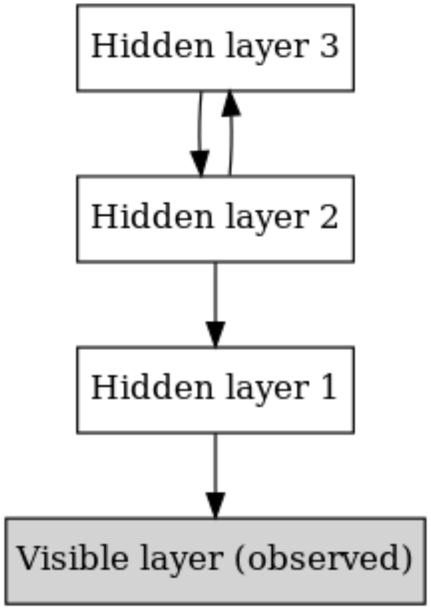
Restricted Bolzmann Machine

- Maximum likelihood estimated:
- $$\nabla_{w_{ij}} L(W) = \frac{1}{N_P K} \sum_{v \in P} v_{0i} h_{0j} - \frac{1}{M} \sum v_{\infty i} h_{\infty j}$$
- No need to lift up the entire energy landscape!
  - Raising the neighborhood of desired patterns is sufficient

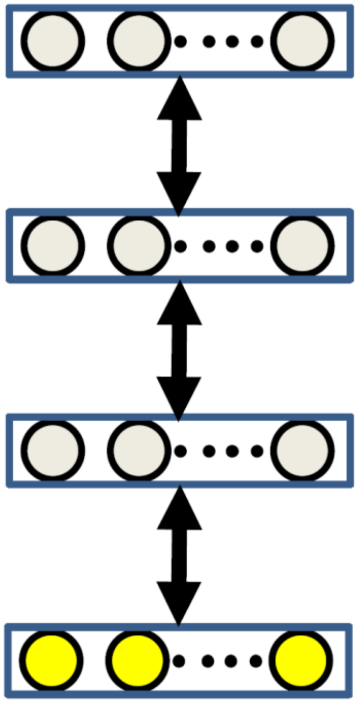


Deep Bolzmann Machine

- Can we have a **deep** version of RBM?
  - Deep Belief Net ('06)
  - Deep Boltzmann Machine ('09)
- Sampling?
  - Forward pass: bottom-up
  - Backward pass: top-down
- Deep Bolzmann Machine
  - The very first deep generative model
  - Salakhudinov & Hinton

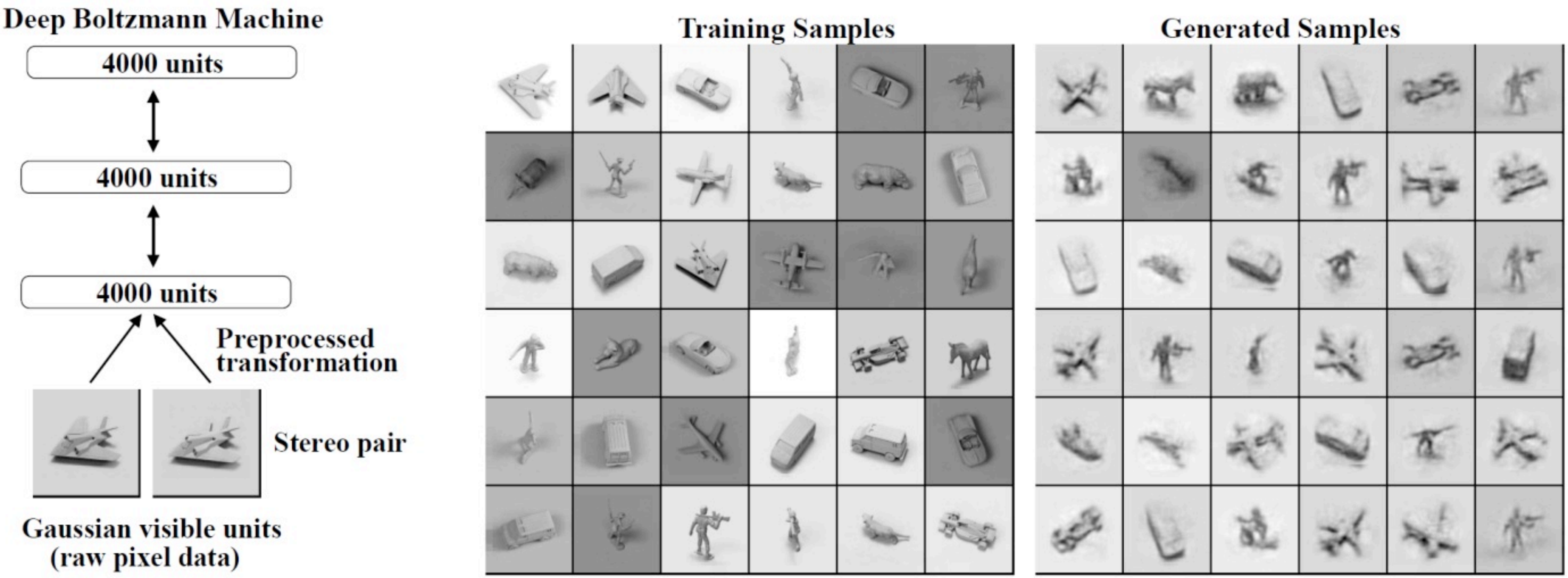


deep belief net



Deep Boltzmann Machine

Deep Bolzmann Machine



Summary

- Pros: powerful and flexible
  - An arbitrarily complex density function $p(x) = \frac{1}{z}\exp(-E(x))$
- Cons: hard to sample / train
  - Hard to sample:
    - MCMC sampling
  - Partition function
    - No closed-form calculation for likelihood
    - Cannot optimize MLE loss exactly
    - MCMC sampling

Normalizing Flows

Intuition about easy to sample

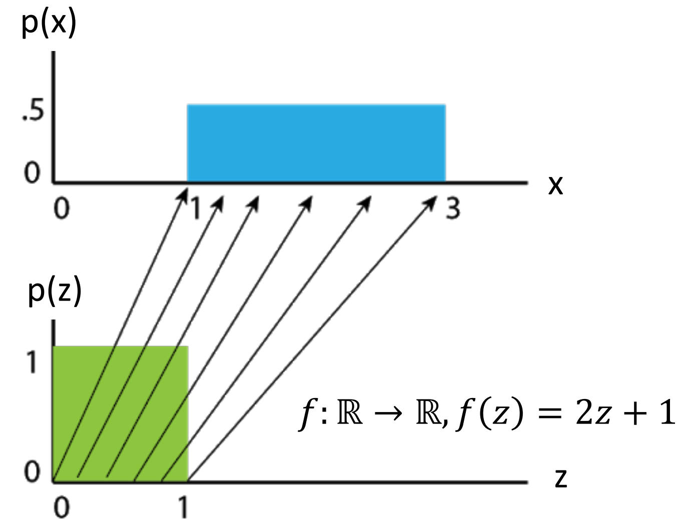
- Goal: design $p(x)$ such that
  - Easy to sample
  - Tractable likelihood (density function)

- Easy to sample
  - Assume a continuous variable z
  - e.g., Gaussian $z \sim N(0,1)$, or uniform $z \sim \text{Unif}[0,1]$
  - $x = f(z)$, x is also easy to sample

Intuition about tractable density

- Goal: design $f(z;\theta)$ such that
  - Assume z is from an "easy" distribution
  - $p(x) = p(f(z;\theta))$ has tractable likelihood

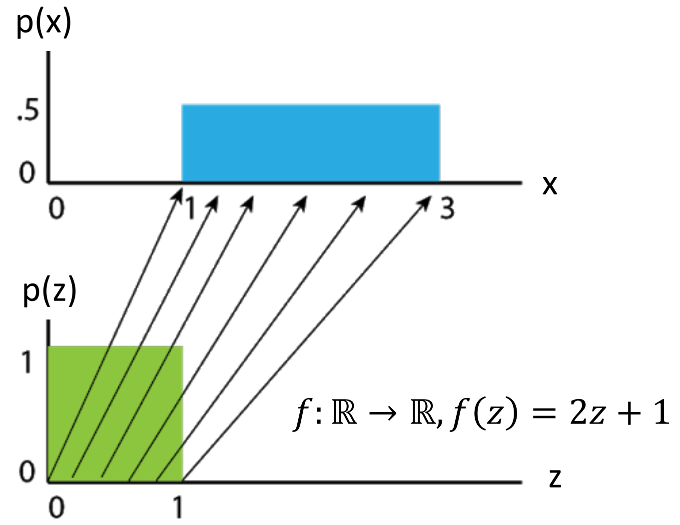
- Uniform: $z \sim \text{Unif}[0,1]$
  - Density $p(z) = 1$
  - $x = 2z + 1$, then $p(x) = ?$



Intuition about tractable density

- Goal: design $f(z;\theta)$ such that
  - Assume z is from an "easy" distribution
  - $p(x) = p(f(z;\theta))$ has tractable likelihood

- Uniform: $z \sim$ Unif$[0,1]$
  - Density $p(z) = 1$
  - $x = 2z + 1$, then $p(x) = 1/2$
    - $x = az + b$, then $p(x) = 1/|a|$ (for $a \neq 0$)
  - $x = f(z), p(z)\left|\dfrac{dz}{dx}\right| = |f'(z)|^{-1} p(z)$
    - Assume $f(z)$ is a bijection

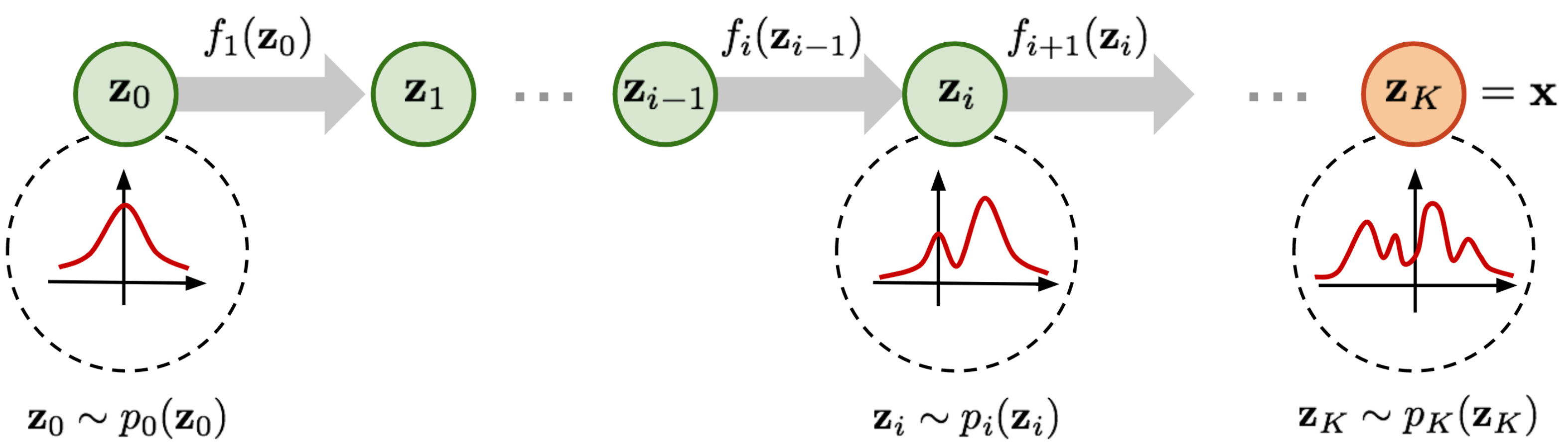


Change of variable

- Suppose $x = f(z)$ for some general non-linear $f(\cdot)$
    - The linearized change in volume is determined by the Jacobian of $f(\cdot)$:
    - $$\frac{\partial f(z)}{\partial z} = \begin{bmatrix} \frac{\partial f_z(x)}{\partial z_1} & \cdots & \frac{\partial f_1(z)}{\partial z_d} \\ \cdots & \cdots & \cdots \\ \frac{\partial f_d(z)}{\partial z_1} & \cdots & \frac{\partial f_d(z)}{\partial z_d} \end{bmatrix}$$
- Given a bijection $f(z) : \mathbb{R}^d \to \mathbb{R}^d$
    - $z = f^{-1}(x)$
    - $$p(x) = p(f^{-1}(x)) \left| \det \left(\frac{\partial f^{-1}(x)}{\partial x} \right) \right| = p(z) \left| \det \left(\frac{\partial f^{-1}(x)}{\partial x} \right) \right|$$
    - Since $\frac{\partial f^{-1}}{\partial x} = \left(\frac{\partial f}{\partial x} \right)^{-1}$ (Jacobian of invertible function)
    - $$p(x) = p(z) \left| \det \left(\frac{\partial f^{-1}(x)}{\partial x} \right) \right| = p(z) \left| \det \left(\frac{\partial f(z)}{\partial z} \right) \right|^{-1}$$

Normalizing Flow

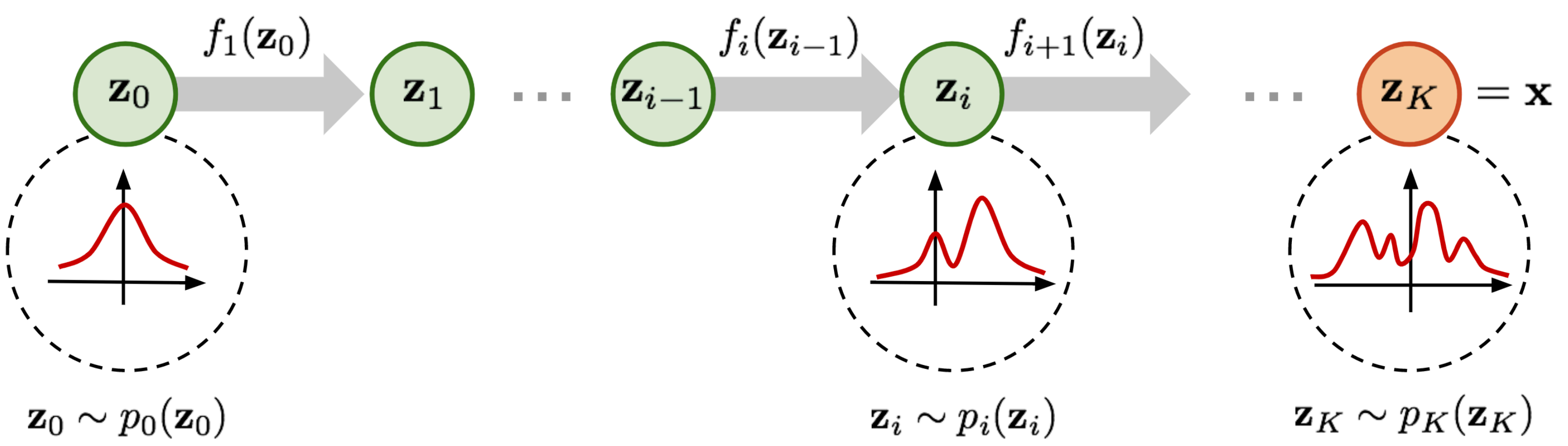
- Idea
  - Sample z_0 from an "easy" distribution, e.g., standard Gaussian
  - Apply K bijections $z_i = f_i(z_{i-1})$
  - The final sample $x = f_K(z_K)$ has tractable desnity
- Normalizing Flow
  - $z_0 \sim N(0,I), z_i = f_i(z_{i-1}), x = Z_K$ where $x, z_i \in \mathbb{R}^d$ and f_i is invertible
  - Every revertible function produces a normalized density function
  - $p(z_i) = p(z_{i-1}) \left| \det \left(\dfrac{\partial f_i}{\partial z_{i-1}} \right) \right|^{-1}$



Normalizing Flow

- Generation is trivial
  - Sample z_0 then apply the transformations
- Log-likelihood
  - $\log p(x) = \log p(Z_{k-1}) - \log \left| \det \left(\frac{\partial f_K}{\partial z_{K-1}} \right) \right|$
  - $\log p(x) = \log p(z_0) - \sum_i \log \left| \det \left(\frac{\partial f_i}{\partial z_{i-1}} \right) \right|$

$O(d^3)$!!!



Normalizing Flow

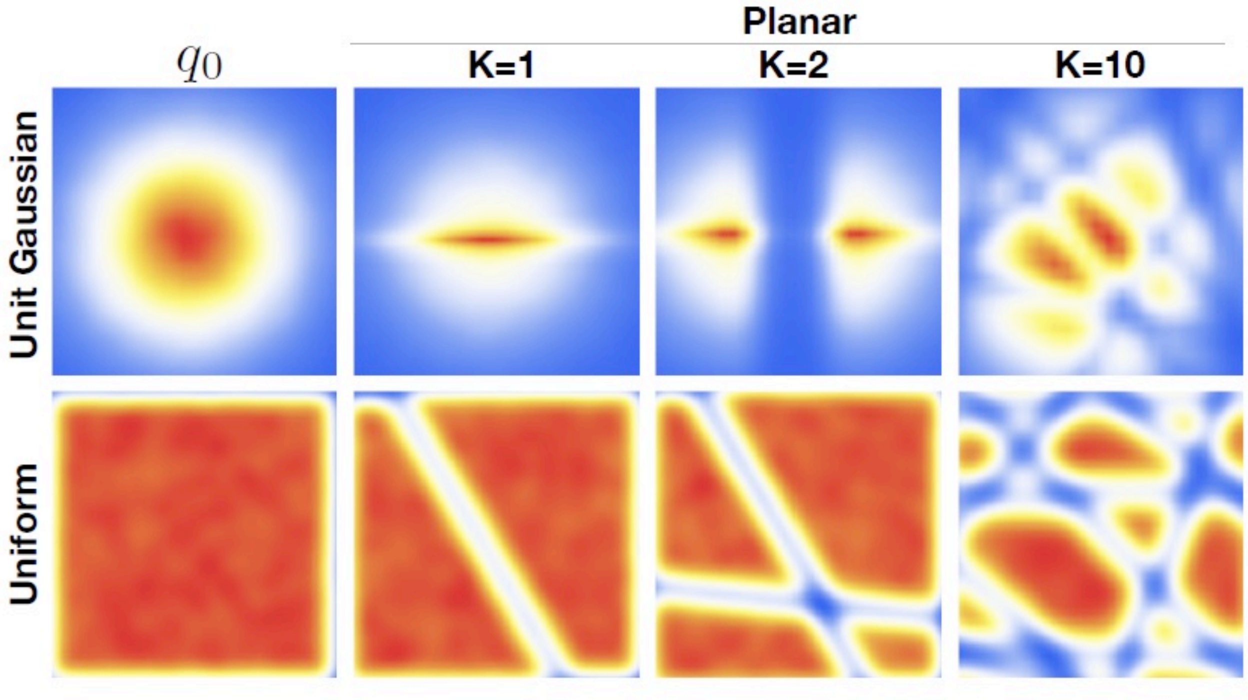
- Naive flow model requires extremely expensive computation
  - Computing determinant of $d \times d$ matrices
- Idea:
  - Design a good bijection $f_i(z)$ such that the determinant is easy to compute

Plannar Flow

- Technical tool: Matrix Determinant Lemma:
  - $\det(A + uv^\top) + (1 + v^\top A^{-1} u) \det A$
- Model:
  - $f_\theta(z) + z + u \odot h(w^\top z + b)$
  - $h(\cdot)$ chosen to be $\tanh(\cdot)(0 < h'(\cdot) < 1)$
  - $\theta = [u, w, b], \det\left(\frac{\partial f}{\partial z}\right) = \det(I + h'(w^\top z + b)uw^\top) = 1 + h'(w^\top z + b)u^\top w$
  - Computation in $O(d)$ time
  - Remarks:
    - $u^\top w > -1$ to ensure invertibility
    - Require normalization on u and w

Planar Flow (Rezende & Mohamed, ’16)

- $f_\theta(z) = z + uh\left(w^\top z + b\right)$
- 10 planar transformations can transform simple distributions into a more complex one



Extensions

- Other flow models uses triangular Jacobian
  - Suppose $x_i = f_i(z)$ only depends on $z_{\leq i}$