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

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 $g(z;\theta)$ (can provide infinite samples).

- **Overall objectives:**
  - Generator: $\theta^* = \max_{\theta} D(g(z;\theta);\phi)$
  - Discriminator uses MLE Training:
    $\phi^* = \max_{\phi} \mathbb{E}_{x\sim p_{data}}[\log D(x;\phi)] + \mathbb{E}_{\hat{x}\sim g(\cdot)}[\log(1 - D(\hat{x};\phi))]$

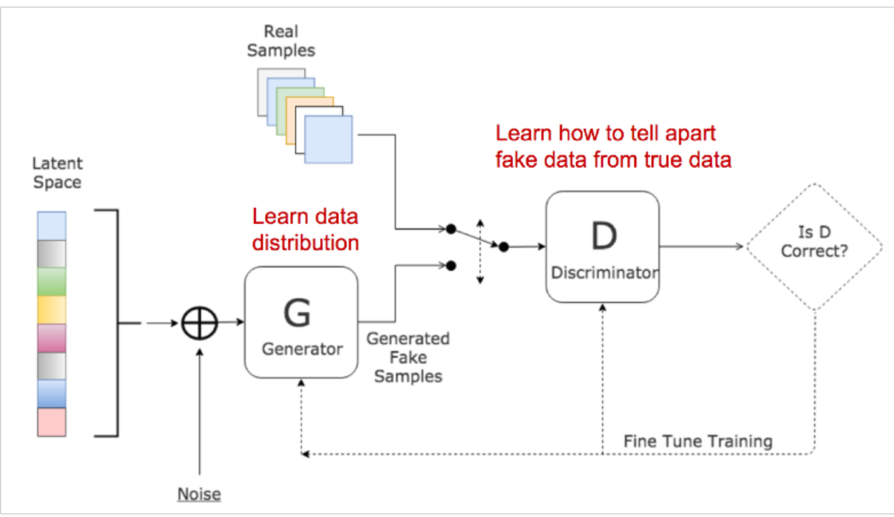
GAN (Goodfellow et al., '14)

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

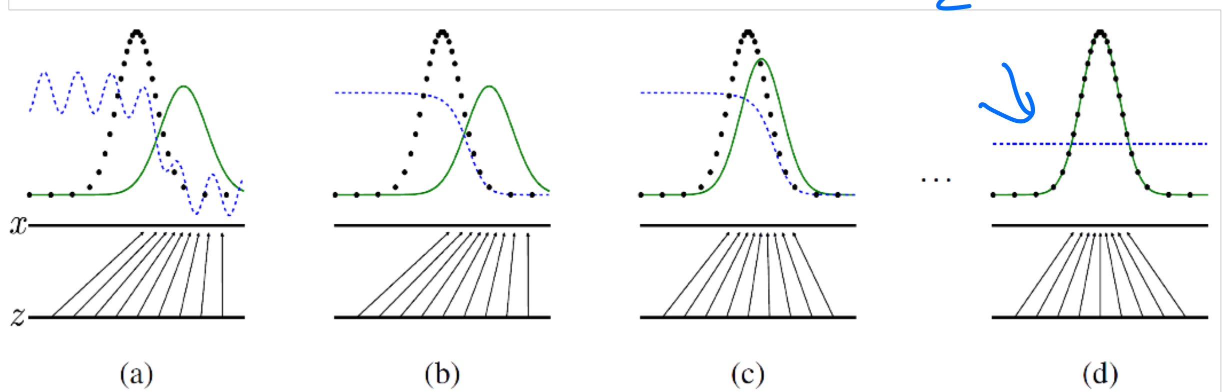
GAN (Goodfellow et al., ‘14)

- Objective function:

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



constant $\frac{1}{2}$



Math Behind GAN

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

- Let D^*, g^* be the solution to $\uparrow$
- For a given x, optimal $D^*(x)$

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

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

$$\Rightarrow \quad \frac{P_{data}(x)}{D^*(x)} - \frac{P_g(x)}{1 - D^*(x)} = 0$$

$$\Rightarrow \quad D^*(x) = \frac{P_{data}(x)}{P_{data}(x) + P_g(x)}$$

(If $P_g = P_{data}$, $\Rightarrow$ $D^*(x) = 0.5$)

Math Behind GAN

Consider optimal g^*, given optimal D^*

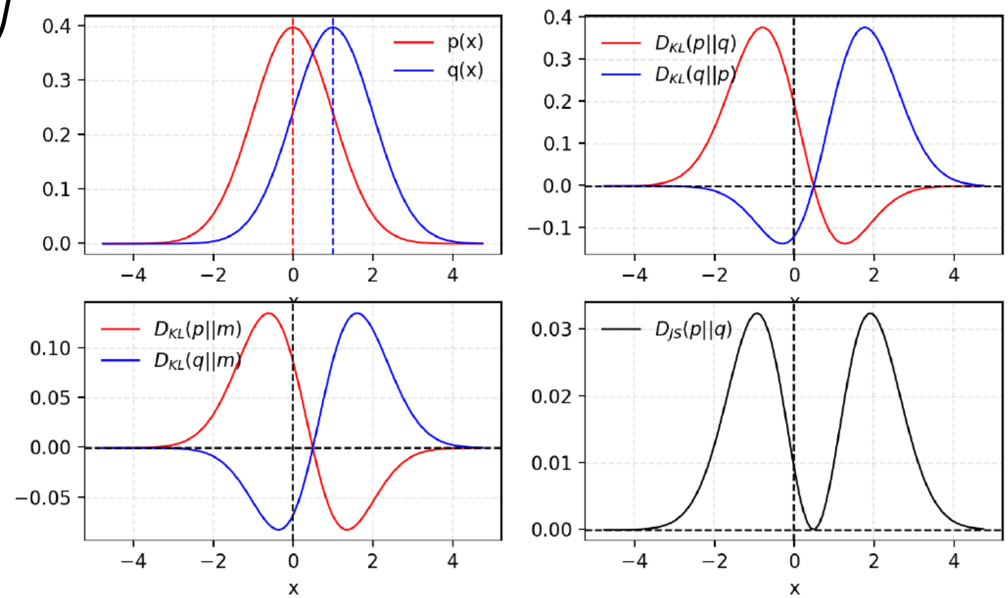
$$L(\theta,\phi) = \mathbb{E}_{x\sim p_{data}}\left[\log \frac{p_{data}(x)}{p_{data}(x)+p_g(x)}\right] + \mathbb{E}_{\hat{x}\sim g(\cdot)}\left[\log \frac{p_g(\hat{x})}{p_{data}(\hat{x})+p_g(\hat{x})}\right]$$

$$= \mathbb{E}_{x\sim p_{data}}\left[\log\left(\frac{p_{data}(x)}{\underbrace{\frac{p_{data}(x)+p_g(x)}{2}}_{\text{dist}}}\right)\right] + \mathbb{E}_{\hat{x}\sim g(\cdot)}\left[\log\left(\frac{p_g(\hat{x})}{\frac{p_{data}(\hat{x})+p_g(\hat{x})}{2}}\right)\right] - \log 4$$

$$= \underbrace{KL\left(p_{data} \,\Big\|\, \tfrac{1}{2}(p_{data}+p_g)\right) + KL\left(p_g \,\Big\|\, \tfrac{1}{2}(p_{data}+p_g)\right)}_{2\text{- Jensen-Shannon Divergence}} - \log 4$$

KL-Divergence and JS-Divergence

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



- $JSD(p||q)$

$$= \frac{1}{2}\left(KL(p||q) + KL(q||p)\right)$$

symmetric, $JSD \geq 0$, $JSD(p||q) = 0 \iff p = q$

Math Behind GAN

$\Rightarrow$ Given D^*

$$\min_{\theta} L(\theta) = 2 \cdot \underbrace{JSD(P_g \| P_{data})}_{\geq 0} - \log 4$$

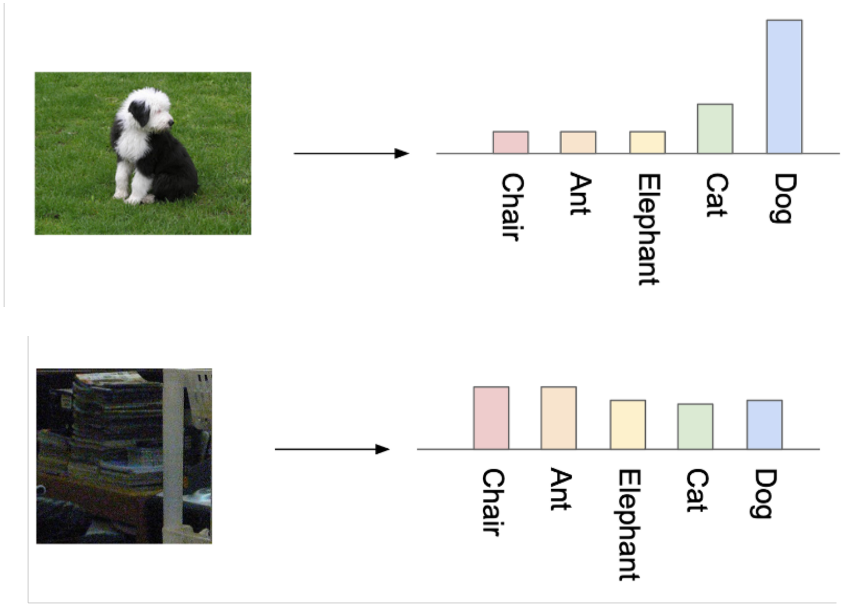
$\Rightarrow$ global minimizer g^*

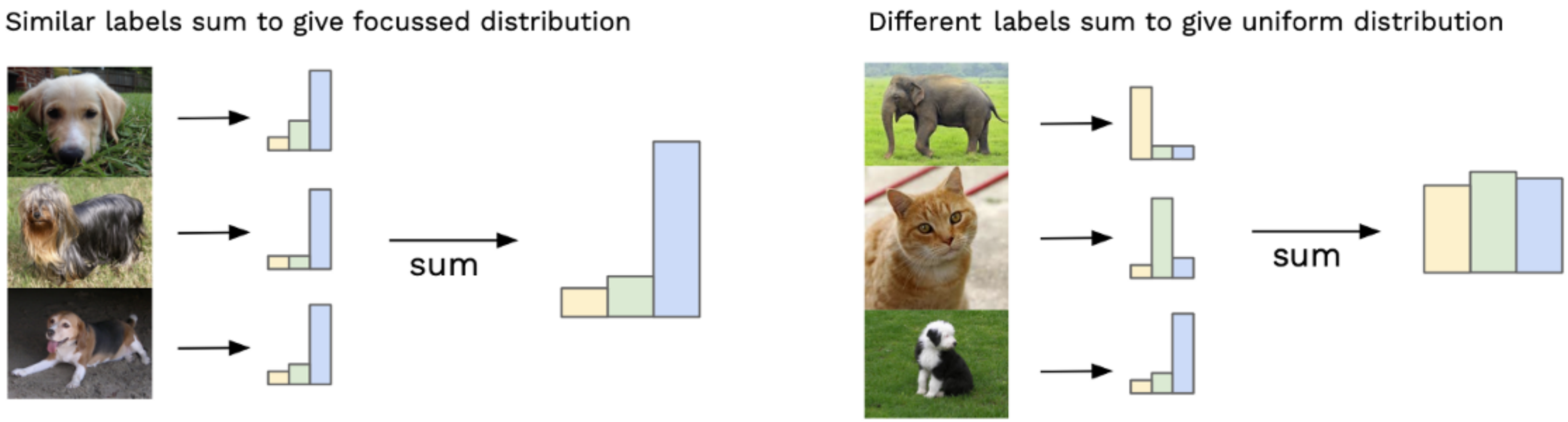
- $g^* = P_{data}$

- $L^* = -\log 4$

Evaluation of GAN

- 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.



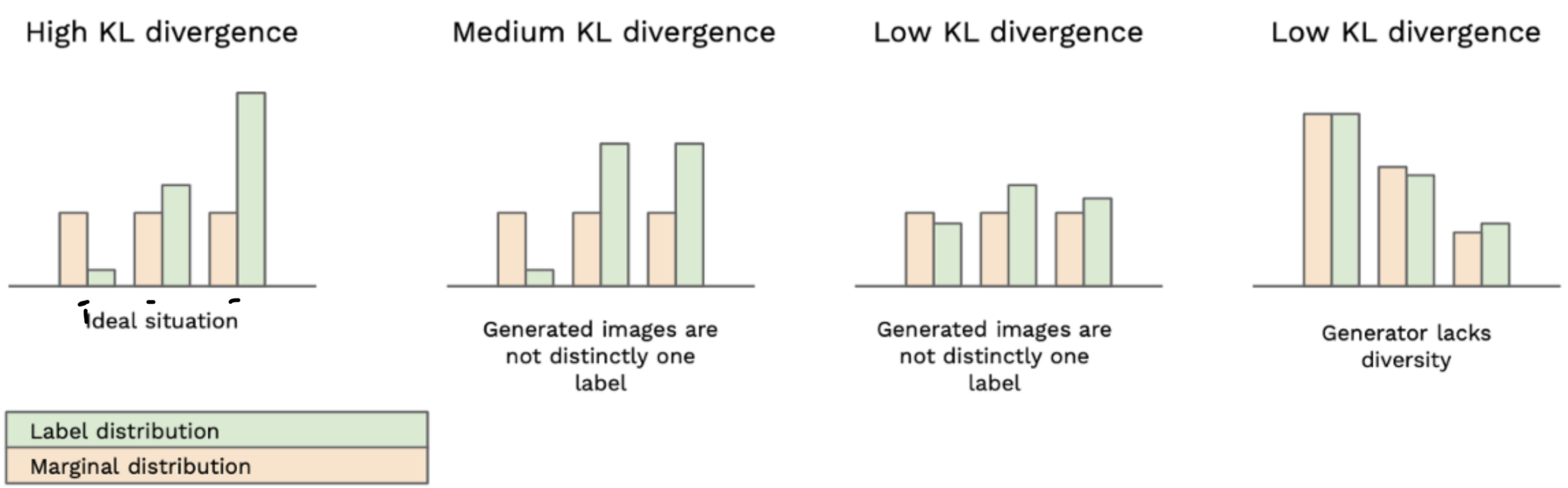


Evaluation of GAN

- **Inception Score** (IS, Salimans et al. ’16)
  - Use Inception V3 trained on ImageNet as $f(y|x)$
  - $IS = \exp\left(\mathbb{E}_{x\sim G}\left[KL(f(y|x)||p_f(y))\right]\right)$
  - Higher the better

low entropy

high entropy



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.

- 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):

    $$\min_G \max_{f:\text{Lip}(f)\leq 1} \mathbb{E}_{x\sim p_{data}}\left[f(x)\right] - \mathbb{E}_{\hat{x}\sim p_G}[f(\hat{x})]$$
  - And need many other tricks…

Variational Autoencoder

Architecture

$D(E(x)) \approx x$

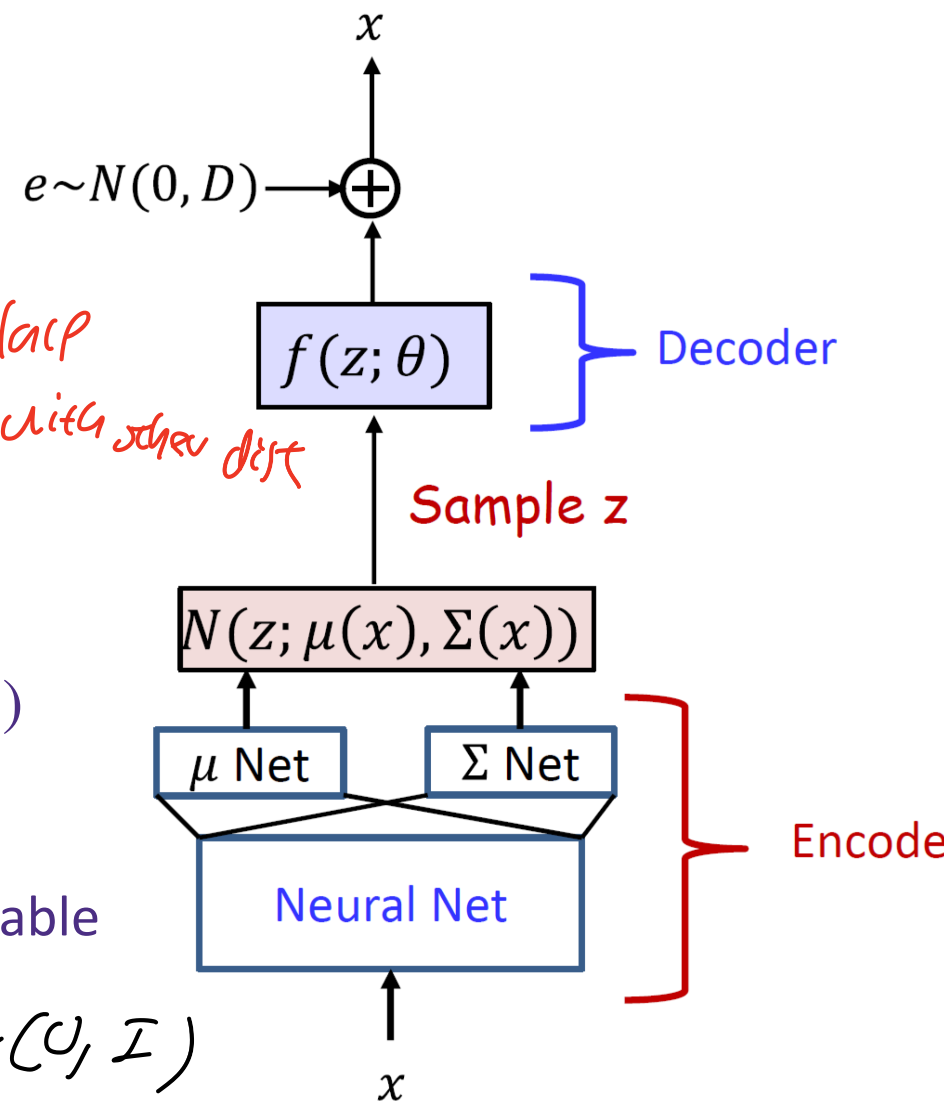
- Auto-encoder: $x \to z \to x$
- Encoder: $q(z \mid x; \phi) : x \to z$
- Decoder: $p(x \mid z; \theta) : z \to x$

- Isomorphic Gaussian:

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

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

can replace gaussian with other dist



- Probabilistic model interpretation: latent variable model.

· easy to generate: $z \sim N(0, I)$

draw $x \sim p(\cdot \mid z; \theta)$

from Bayesian variational inference

VAE Training

evidence lower bound

- 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

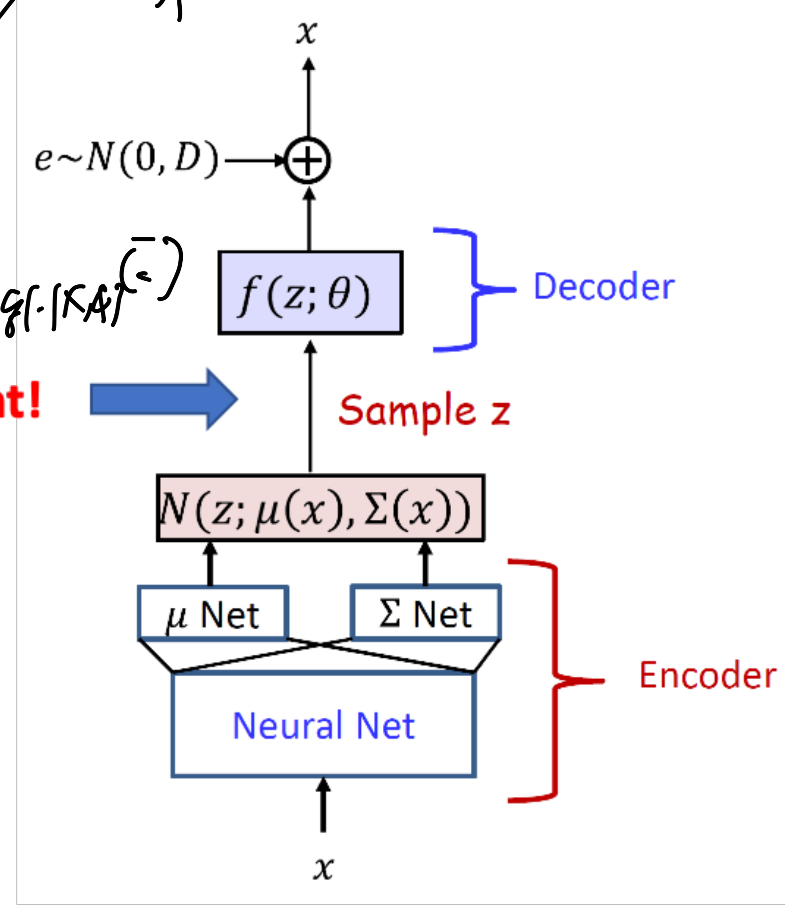
* including gradient

- 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:
    - $x \sim N(f(z;\theta); I)$
    - $p(x) = \frac{1}{\sqrt{2\pi}} \exp(-\frac{1}{2}\|x - f(z;\theta)\|_2^2)$

$\{z_1, \dots z_n\}$

approximate $\mathbb{E}_{z \sim q(\cdot|x)}[\cdot]$

No gradient!



VAE Training

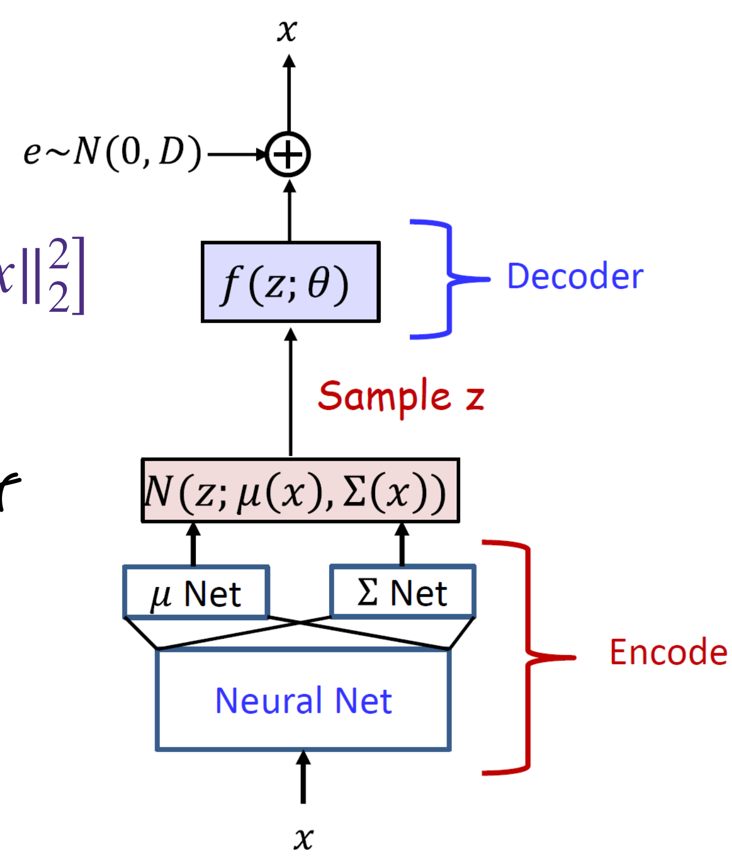
- Likelihood term (reconstruction loss):
  - Gradient for ϕ . Loss: $L(\phi) = \mathbb{E}_{z \sim q(z;\phi)} \left[\log p(x \mid z) \right]$
  - Reparameterization trick:
    - $z \sim N(\mu, \Sigma) \Leftrightarrow z = \mu + \epsilon, \epsilon \sim N(0,\Sigma)$
  - $L(\phi) \propto \mathbb{E}_{z \sim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

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

    sample $\{\varepsilon_1, \ldots, \varepsilon_k\}$

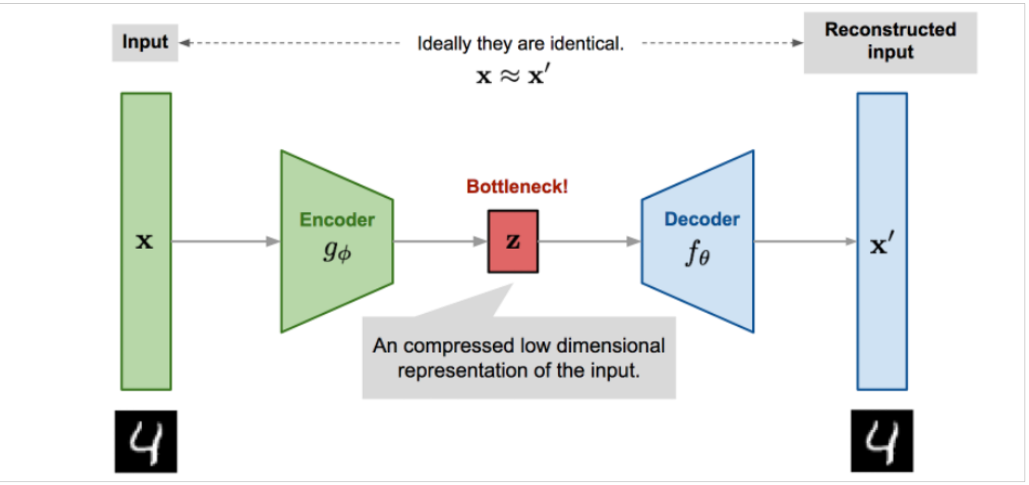
    in practice, 1 sample is sufficient
- End-to-end training

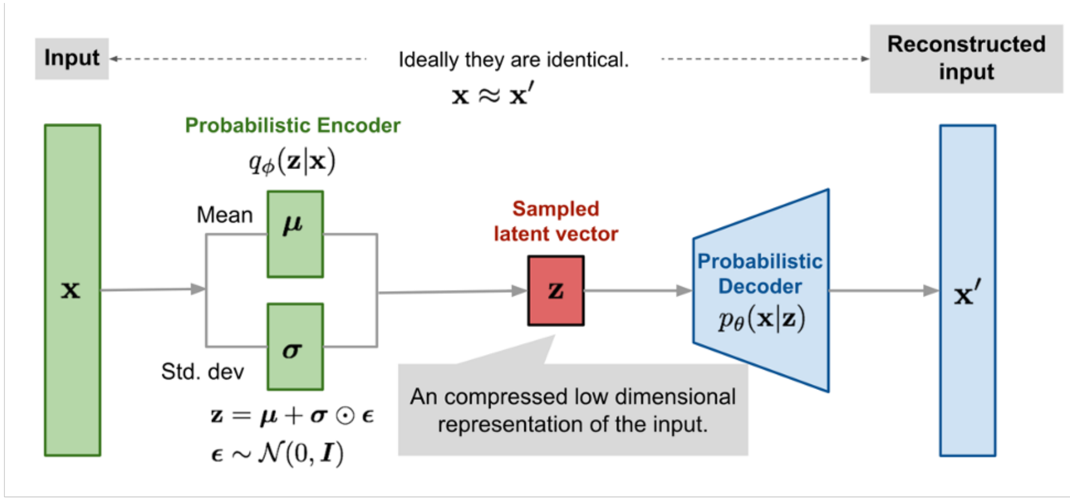
  θ, ϕ



VAE vs. AE

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

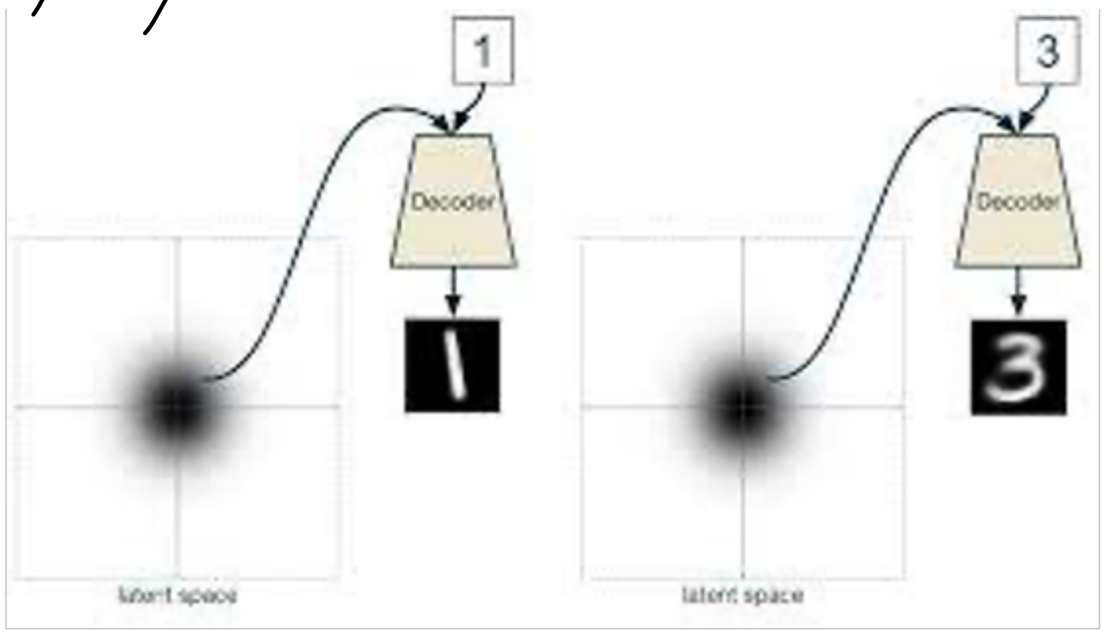




Conditioned VAE

- Semi-supervised learning: some labels are also available

$P(x \mid z, y, \theta)$



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: $P(x)$

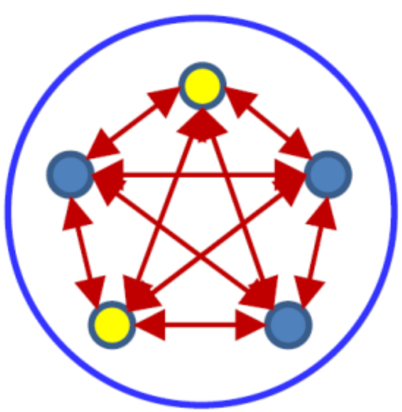
- Requirements
  - $P(x) \geq 0$ (non-negative)
  - $\int_x P(x)dx = 1$

- 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

$T \to \infty$



$s = y$

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

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

Boltzmann Machine: Training

- Objective: maximum likelihood learning (assume T =1):

  $y \in \mathbb{R}^d$

  - Probability of one sample:

    y: binary

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

  - Maximum log-likelihood:

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

$\{y_1, \ldots, y_N\} \in D$

$\#$ of y' : 2^d

Boltzmann Machine: Training

$$L(W) = \frac{1}{N}\sum_{y\in D} \frac{1}{2} y^T W y - \log \sum_{y'} \exp\left(\frac{1}{2} y'^T W y'\right)$$

$$W \in \mathbb{R}^{d\times d}$$

$$\nabla_{W_{ij}} L = \frac{1}{N}\sum_{y\in D} y_i \cdot y_j - \underbrace{\sum_{y'} \frac{\exp\left(\frac{1}{2} y'^T W y'\right)}{Z} y_i' \cdot y_j'}_{\mathbb{E}_{y'}\left[y_i' \cdot y_j'\right]}$$

$\Rightarrow$ monte-Carlo sample $S = \{y'_1, \dots, y'_k\}$

$$\Rightarrow \nabla_{W_{ij}} L \approx \frac{1}{N}\sum_{y\in D} y_i \cdot y_j - \frac{1}{N_2}\sum_{y'\in S} y_i' \cdot y_j'$$

Boltzmann Machine: Sampling

MCMC

- Initialize $y(0) \in \mathbb{R}^d$

for $t = 1, \ldots, N_2$

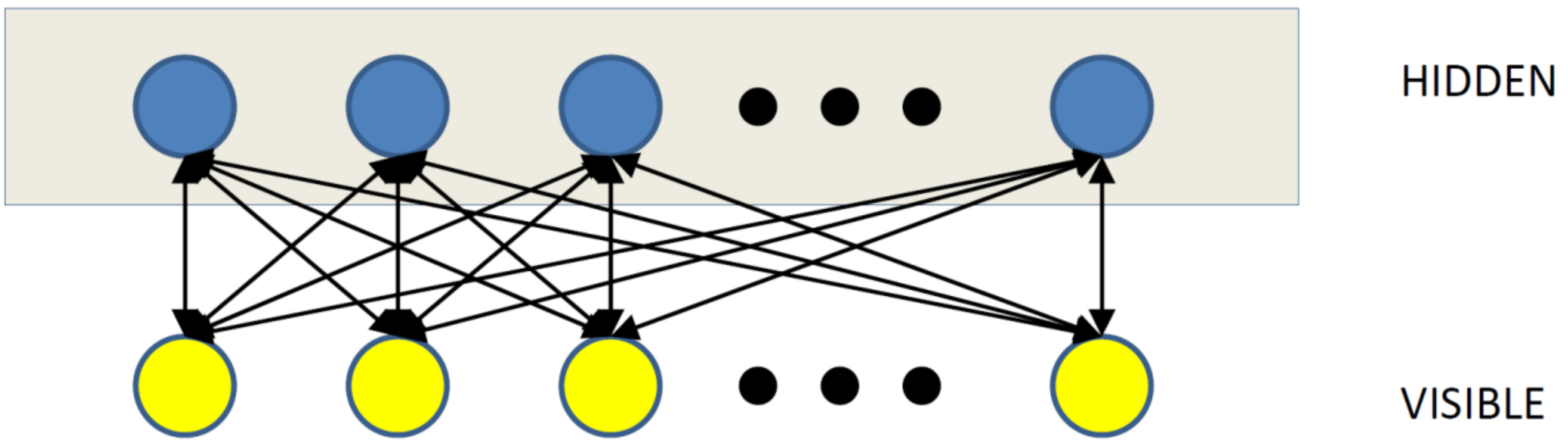
- iterate over $j = \{1, \ldots, d\}$, conditional sampling

$$y_j(t) \sim p(\cdot \mid y_{i \neq j}(t-1))$$

$\Rightarrow \{y(0), \ldots, y(t)\}$

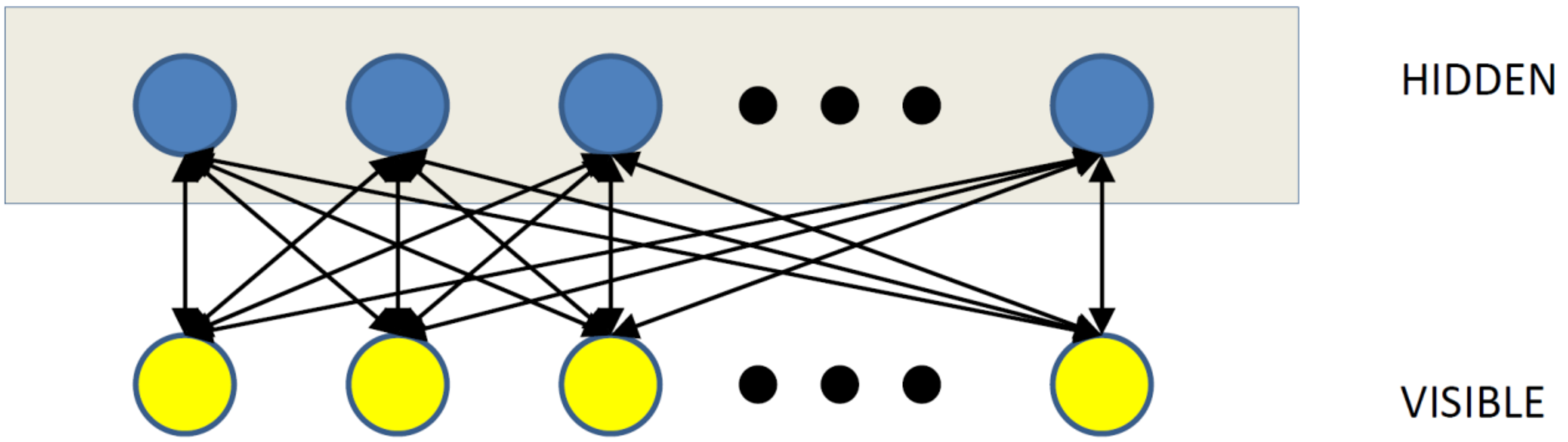
Restricted Bolzmann Machine

- 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

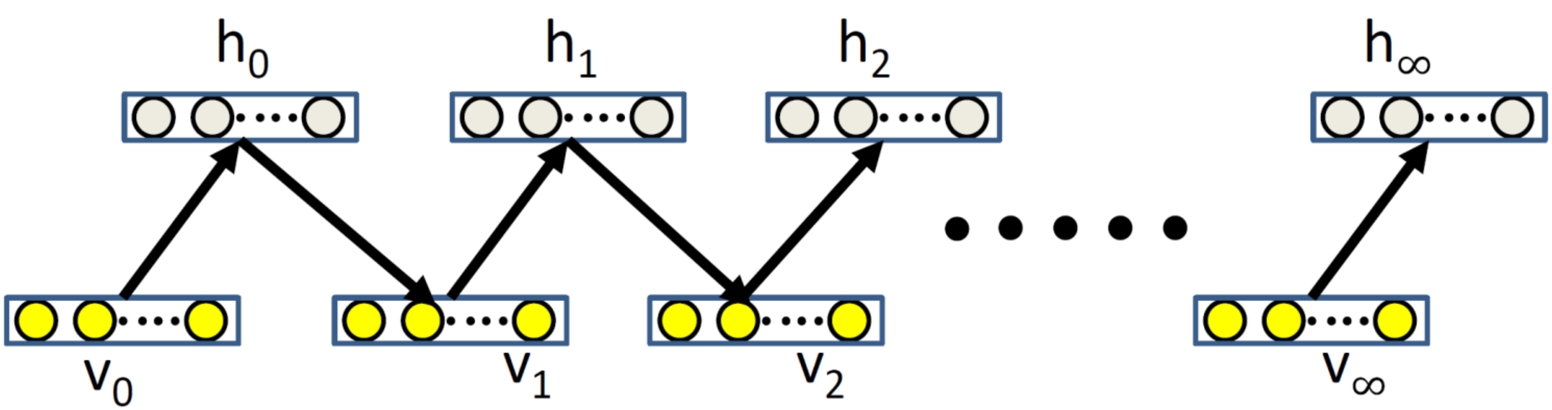
- Computation Rules
  - Iterative sampling
  - Hidden neurons h_i: $z_i = \sum_j w_{ij} v_j$, $P(h_i | v) = \dfrac{1}{1 + \exp(-z_i)}$
  - Visible neurons v_j: $z_j = \sum_i w_{ij} h_i$, $P(v_j | h) = \dfrac{1}{1 + \exp(-z_j)}$



Restricted Bolzmann Machine

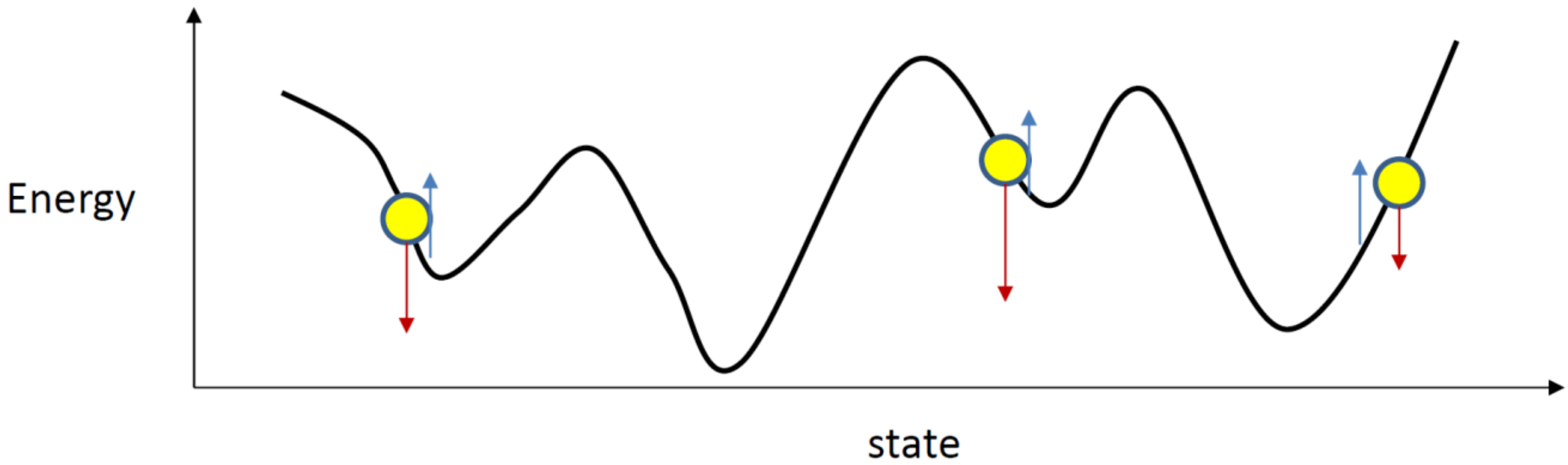
- Sampling:
  - Randomly initialize visible neurons v_0
  - Iterative sampling between hidden neurons and visible neurons
  - Get final sample (v_∞, h_∞)

- Training:
  - MLE
  - Sampling to approximate gradient



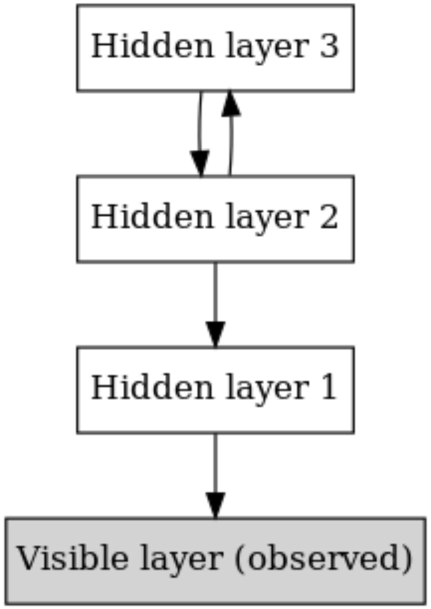
Restricted Bolzmann Machine

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

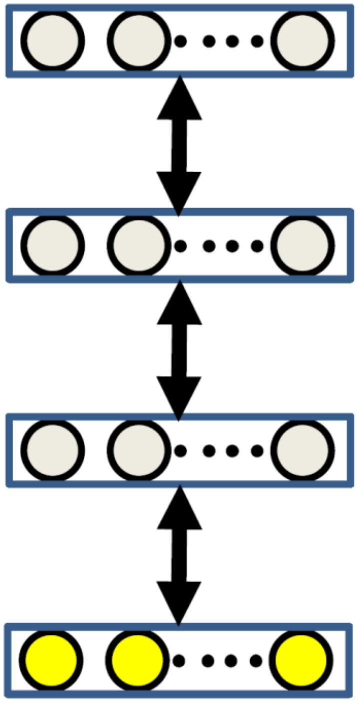


Deep Bolzmann Machine

- 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

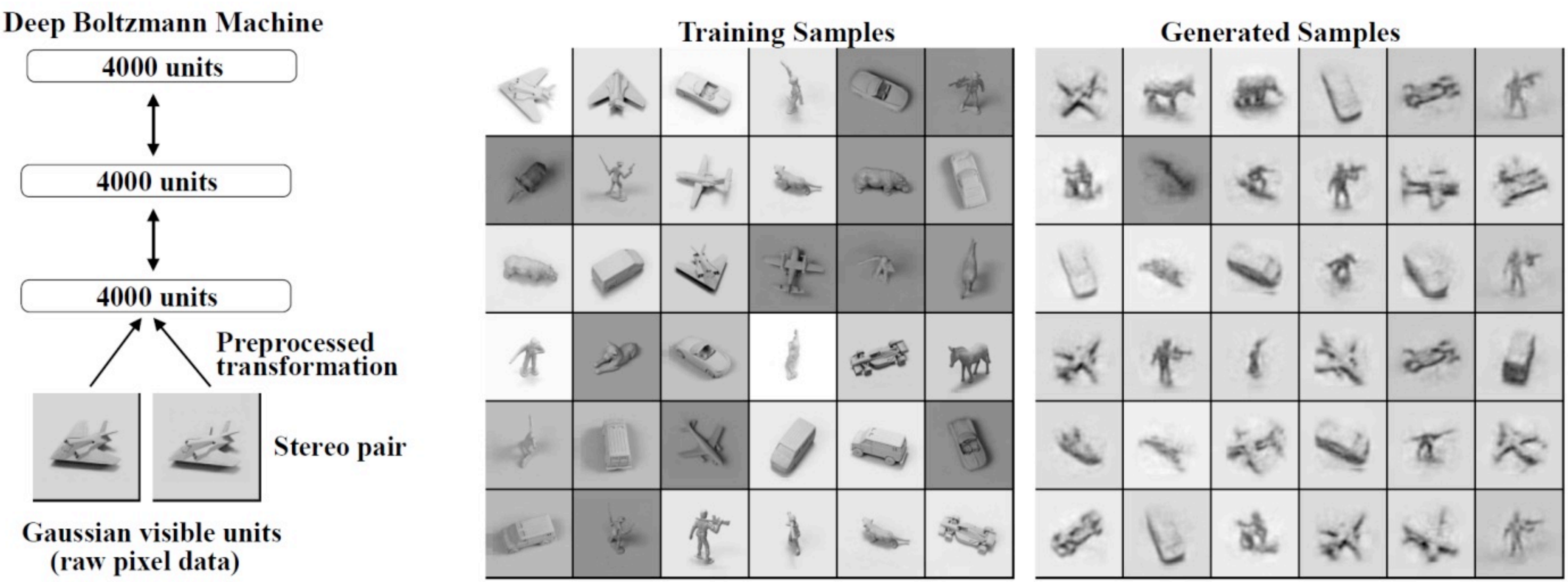


deep belief net



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