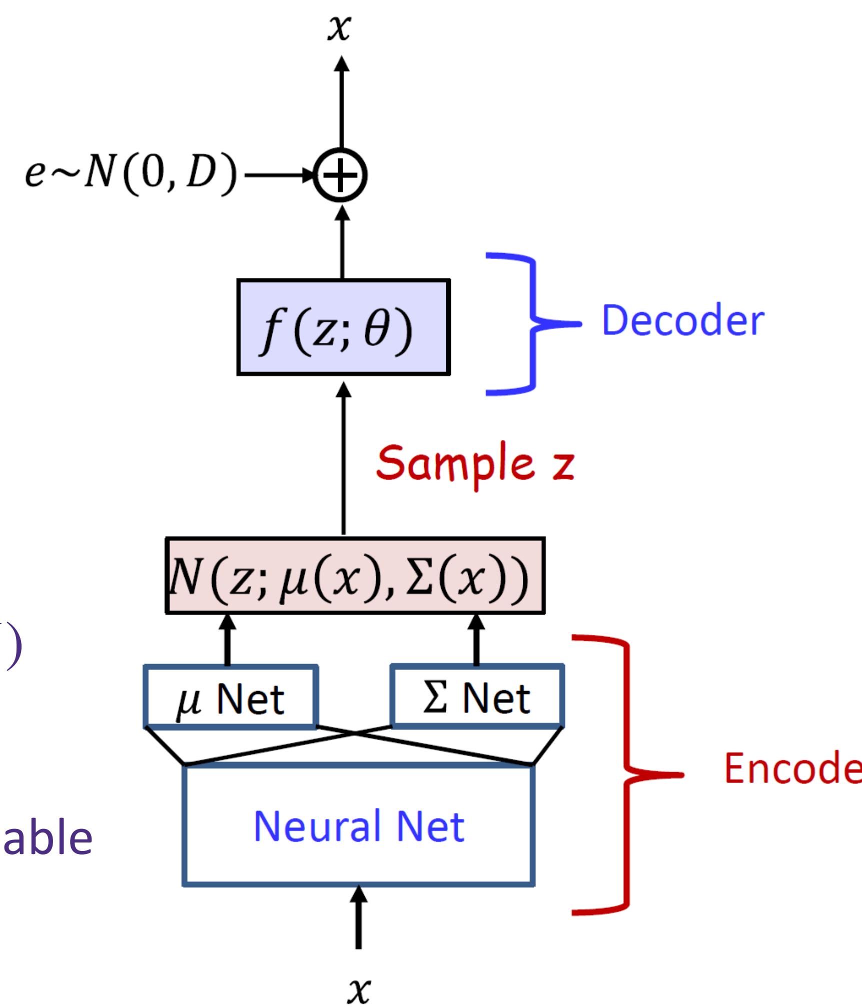
Variational Autoencoder

Architecture

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



- Isomorphic Gaussian:

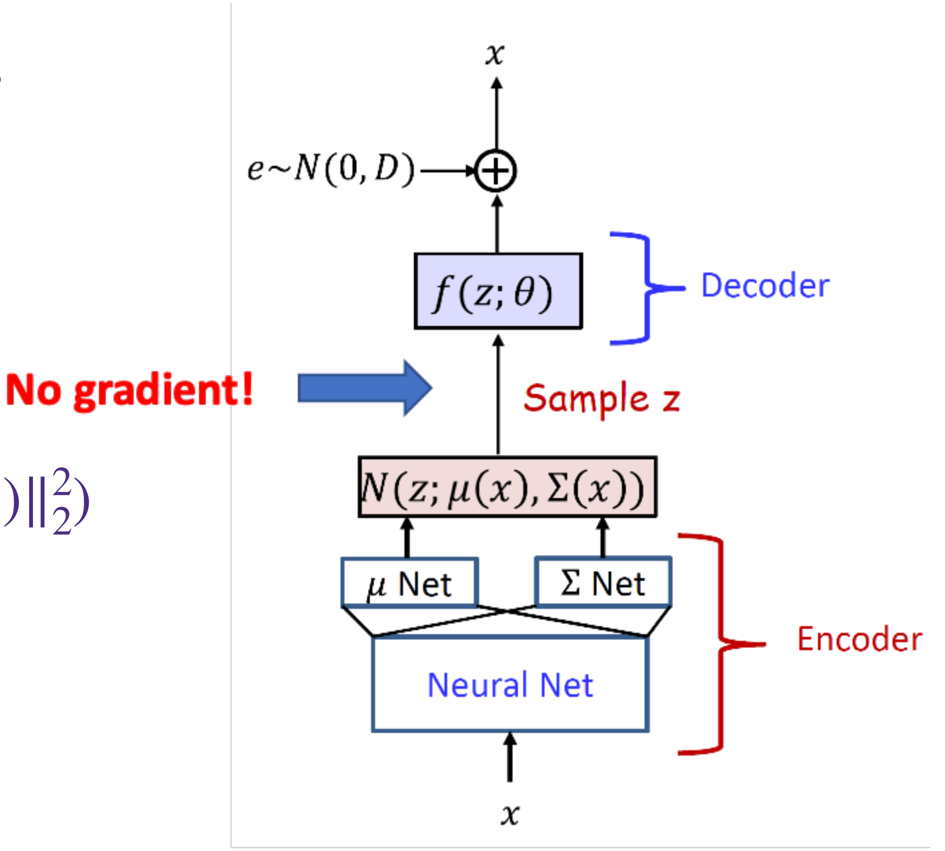
$q(z \mid x; \phi) = N(\mu(x; \phi), \text{diag}(\exp(\sigma(x; \phi))))$

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

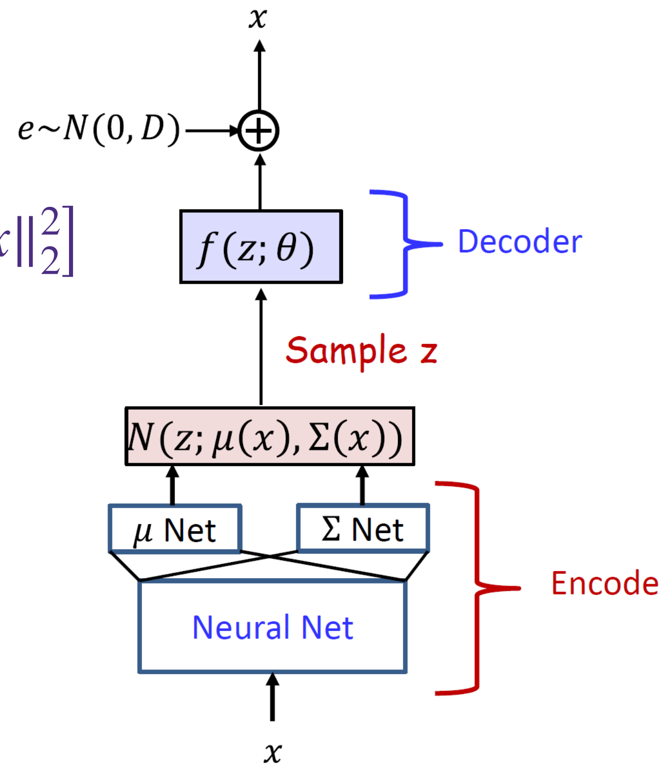


VAE Training

- 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:
    - $z \sim N(\mu, \Sigma) \Leftrightarrow z = \mu + \epsilon, \epsilon \sim N(0,\Sigma)$
  - $L(\phi) \propto \mathbb{E}_{z\sim q(z|\phi)}\left[\|f(z;\theta) - x\|_2^2\right]$

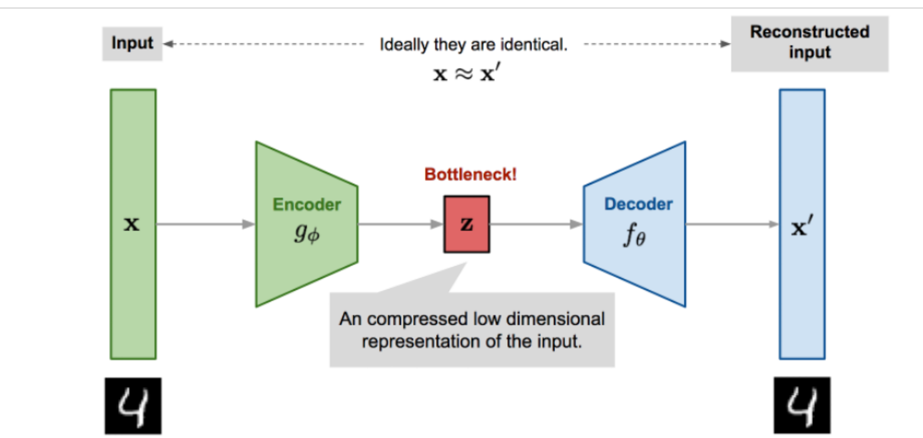
    $\propto \mathbb{E}_{\epsilon\sim N(0,I)}\left[\|f(\mu(x;\phi) + \sigma(x;\phi)\cdot\epsilon;\theta) - x\|_2^2\right]$
  - Monte-Carlo estimate for $\nabla L(\phi)$

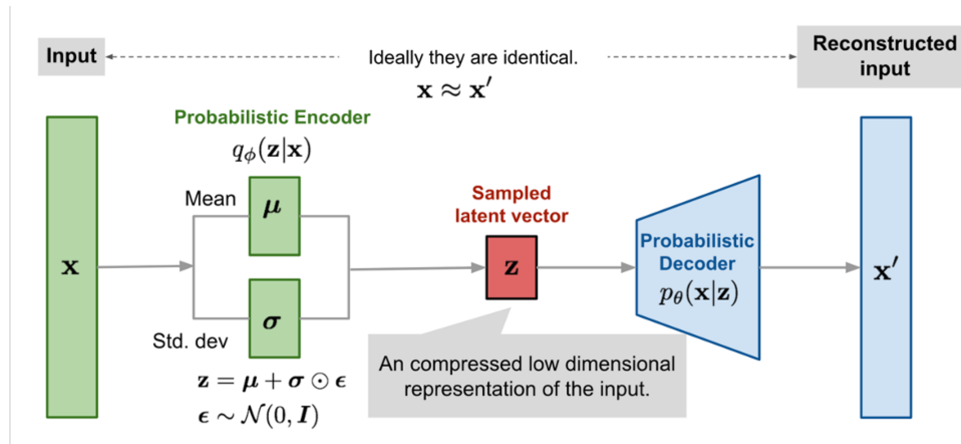
- End-to-end training



VAE vs. AE

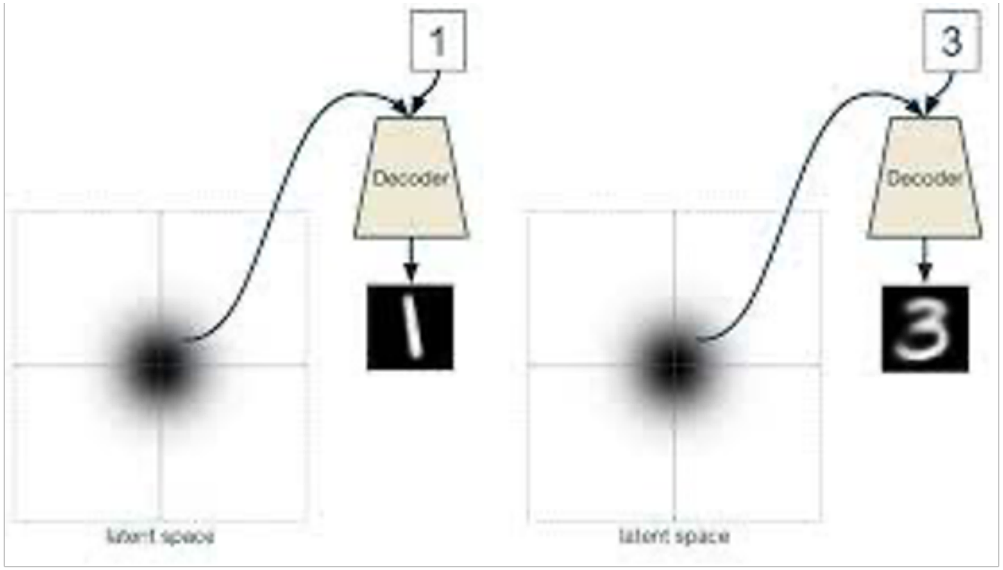
- AE: classical unsupervised representation learning method.
- VAE: a probabilistic model of AE
  - AE + Gaussian noise on z
  - KL penalty: L_2 constraint on the latent vector z





Conditioned VAE

- Semi-supervised learning: some labels are also available



conditioned generation

Comments on VAE

- Pros:
  - Flexible architecture
  - Stable training

- Cons:
  - Inaccurate probability evaluation (approximate inference)

Energy-Based Models

Energy-based Models

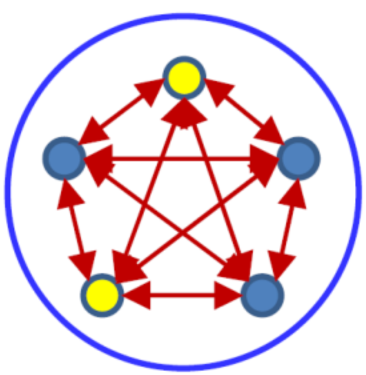
- Goal of generative models:
  - a probability distribution of data: $P(x)$

- 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:

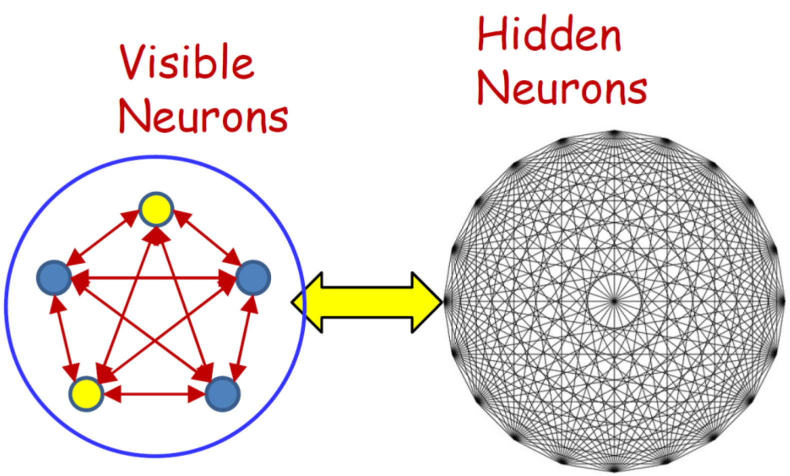
$$L(W) = \frac{1}{N} \sum_{y \in D} \frac{1}{2} y^\top W y - \log \sum_{y'} \exp(\frac{1}{2} y'^\top W y')$$

Boltzmann Machine: Training

Boltzmann Machine: Training

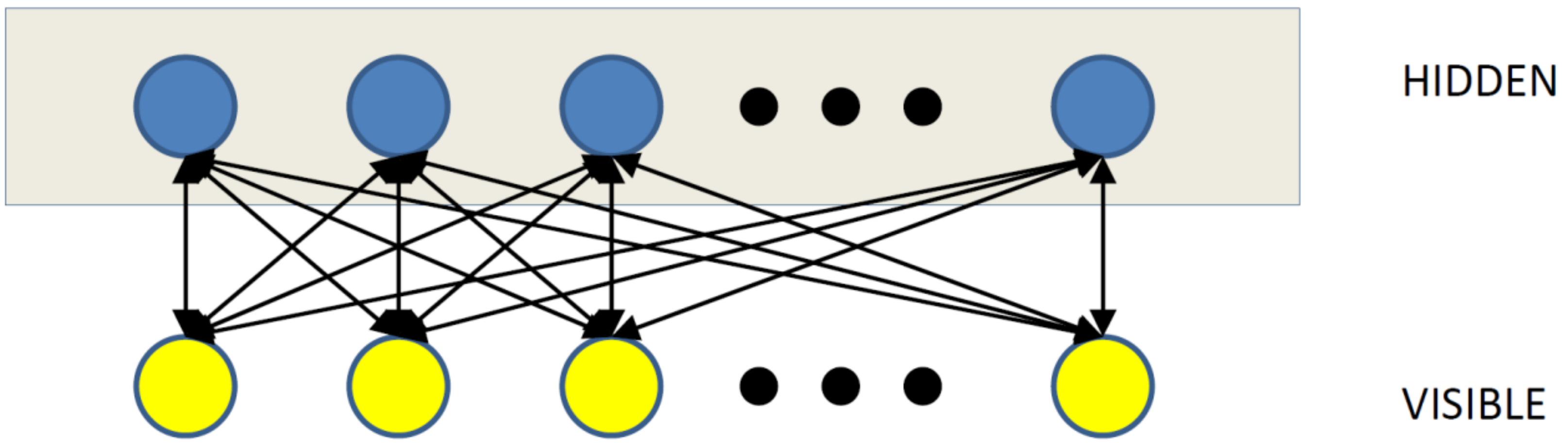
Boltzmann Machine with Hidden Neurons

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



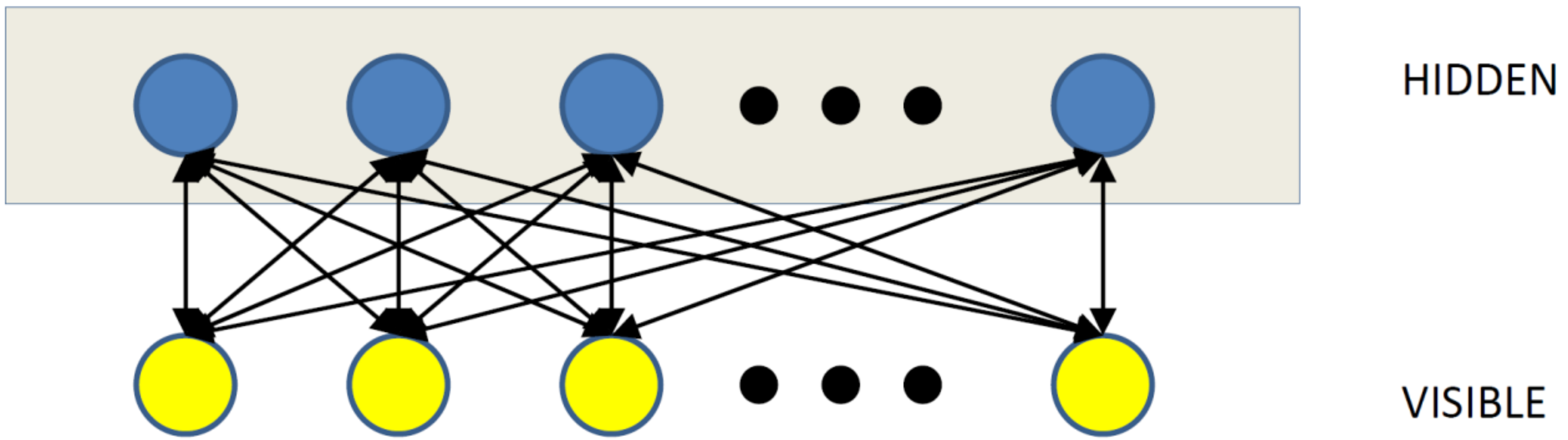
Restricted Bolzmann Machine

- 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

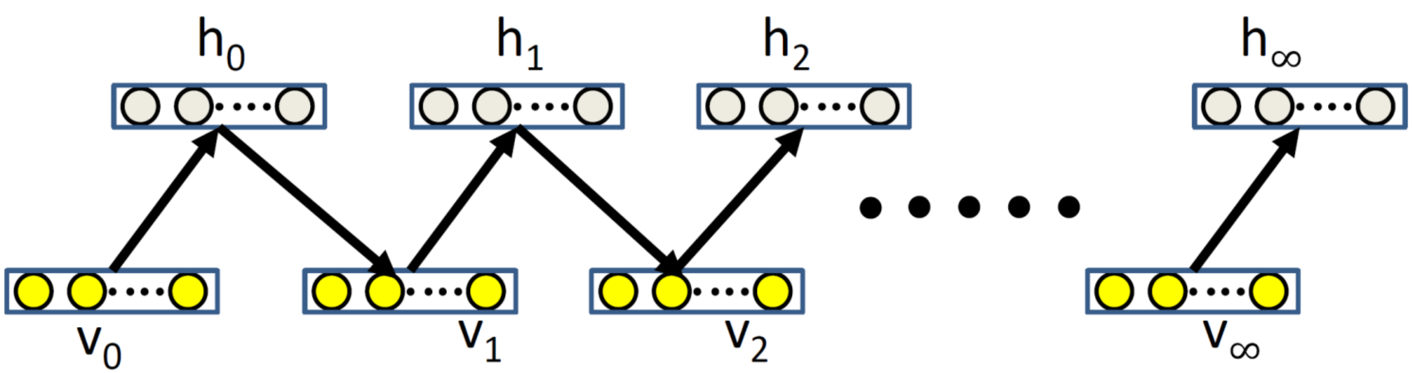
- Computation Rules
  - Iterative sampling
  - Hidden neurons h_i: $z_i = \sum_j w_{ij} v_j$, $P(h_i \mid v) = \dfrac{1}{1 + \exp(-z_i)}$
  - Visible neurons v_j: $z_j = \sum_i w_{ij} h_i$, $P(v_j \mid h) = \dfrac{1}{1 + \exp(-z_j)}$



Restricted Bolzmann Machine

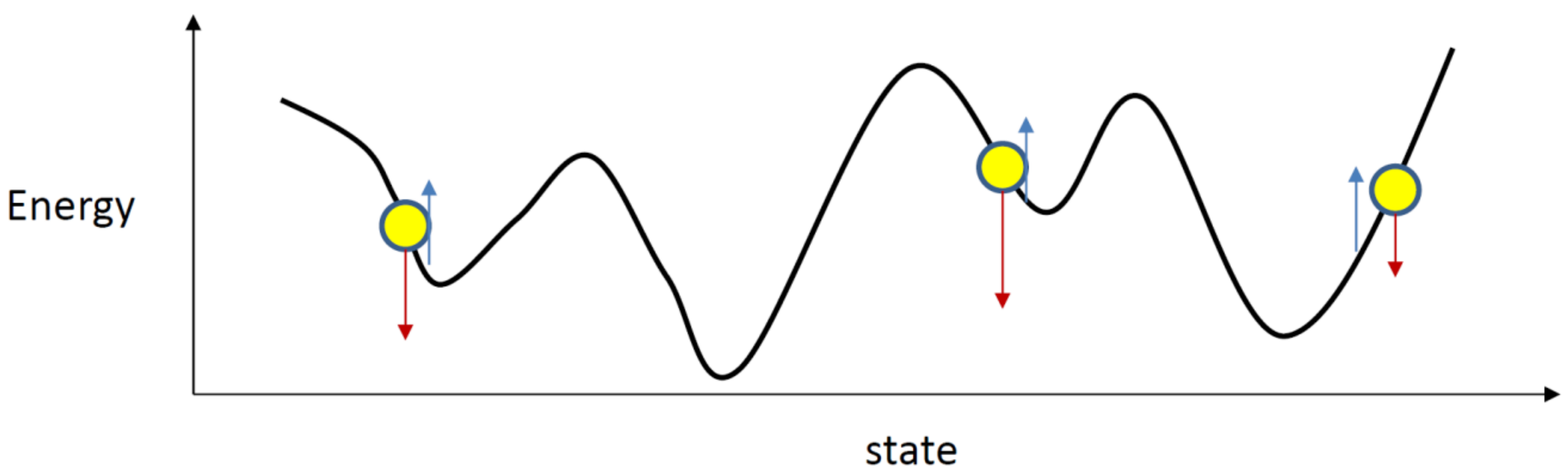
- Sampling:
  - Randomly initialize visible neurons v_0
  - Iterative sampling between hidden neurons and visible neurons
  - Get final sample (v_∞, h_∞)

- Training:
  - MLE
  - Sampling to approximate gradient



Restricted Bolzmann Machine

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

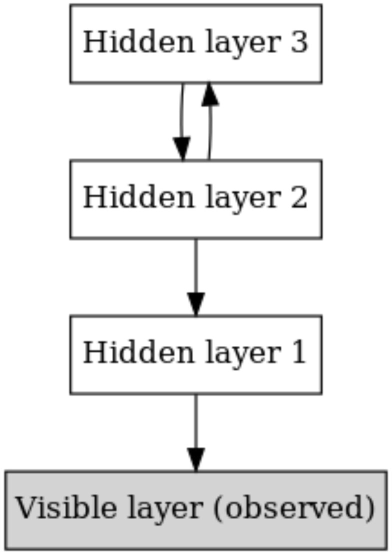


Deep Bolzmann Machine

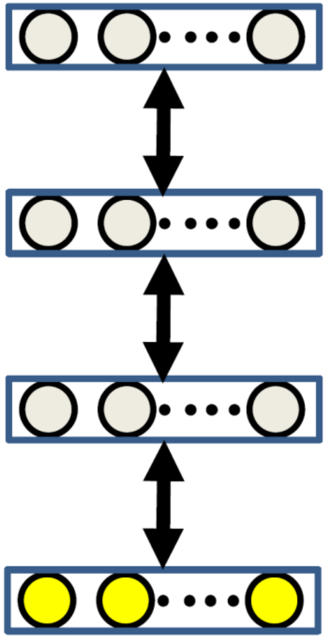
- Can we have a **deep** version of RBM?
  - Deep Belief Net ('06)
  - Deep Boltzmann Machine ('09)

- Sampling?
  - Forward pass: bottom-up
  - Backward pass: top-down

- Deep Bolzmann Machine
  - The very first deep generative model
  - Salakhudinov & Hinton

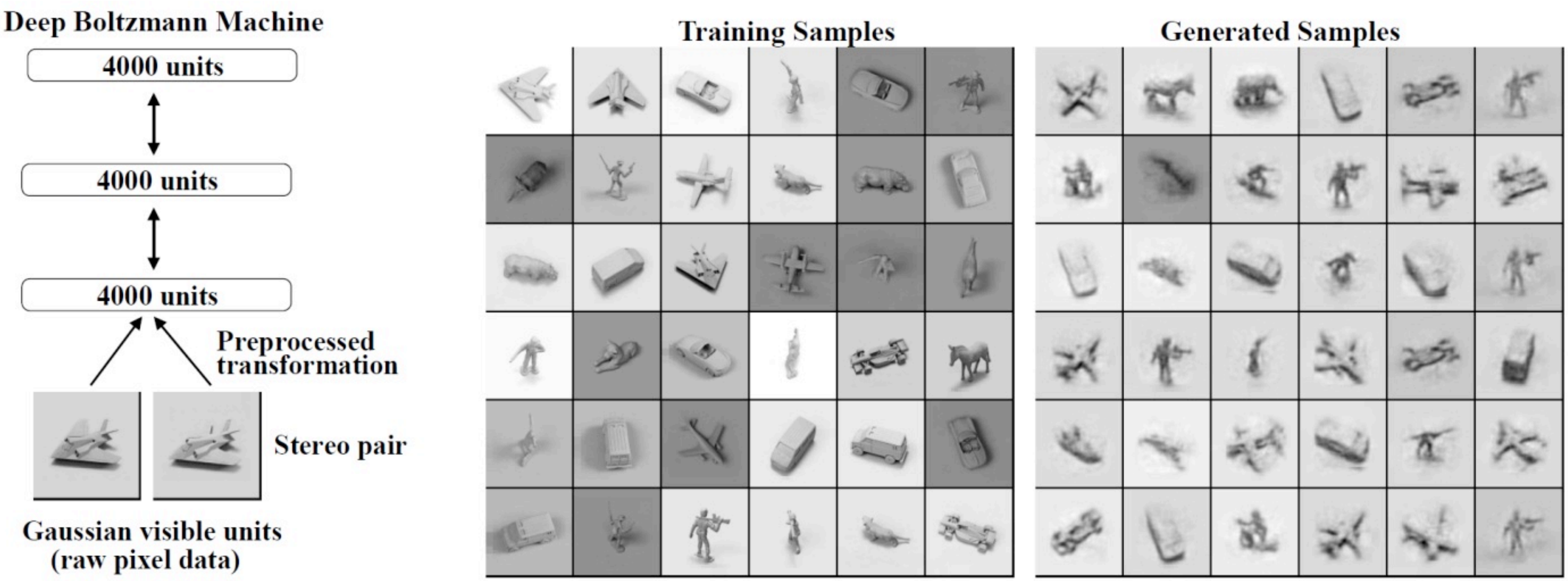


deep belief net



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

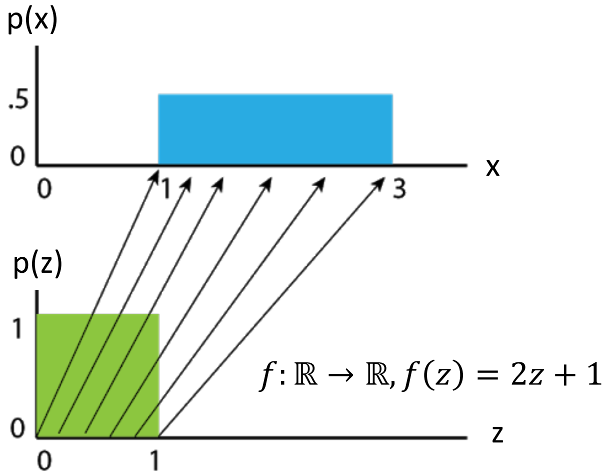
- Goal: design $p(x)$ such that
  - Easy to sample
  - Tractable likelihood (density function)

- 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

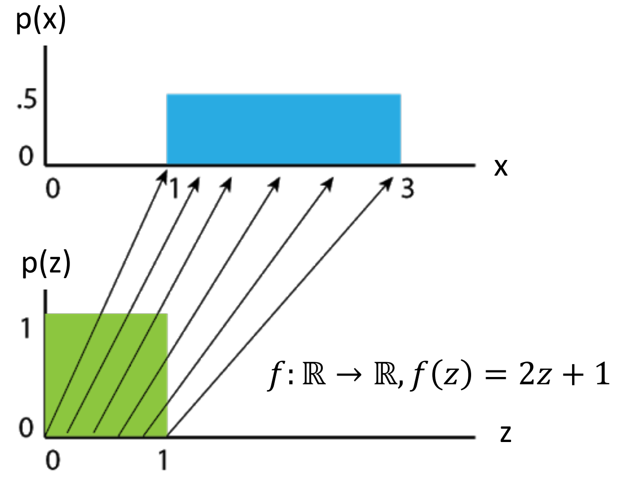
- Uniform: $z \sim \mathrm{Unif}[0,1]$
  - Density $p(z) = 1$
  - $x = 2z + 1$, then $p(x) = ?$



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 $a \neq 0$)
  - $x = f(z), p(x) = p(z)|\frac{dz}{dx}| = |f'(z)|^{-1}p(z)$
    - Assume $f(z)$ is a bijection

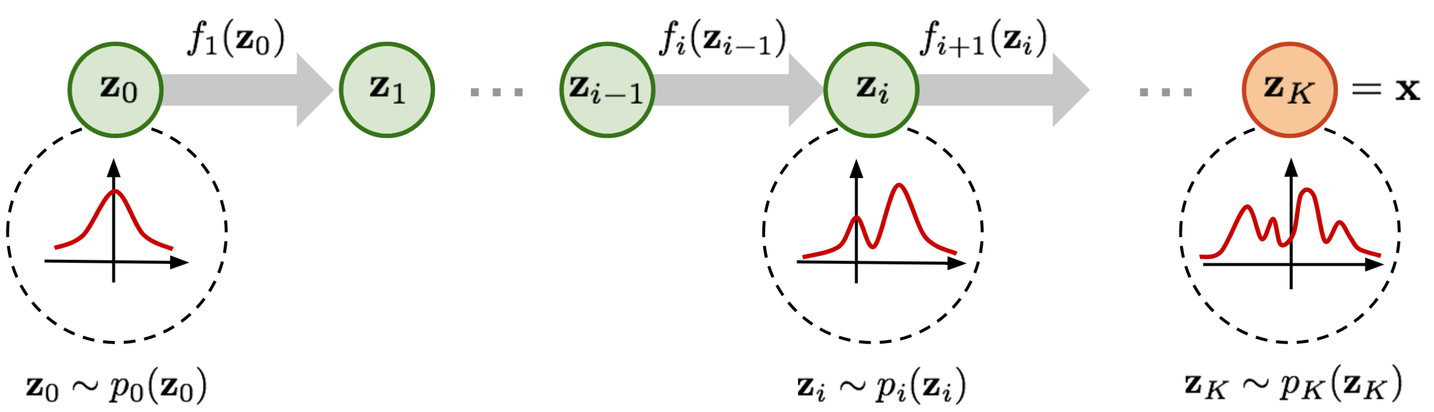


Change of variable

- Suppose $x = f(z)$ for some general non-linear $f(\cdot)$
  - The linearized change in volume is determined by the Jacobian of $f(\cdot)$:
  - $$\frac{\partial f(z)}{\partial z} = \begin{bmatrix} \frac{\partial f_1(z)}{\partial z_1} & \cdots & \frac{\partial f_1(z)}{\partial z_d} \\ \cdots & \cdots & \cdots \\ \frac{\partial f_d(z)}{\partial z_1} & \cdots & \frac{\partial f_d(z)}{\partial z_d} \end{bmatrix}$$
- 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

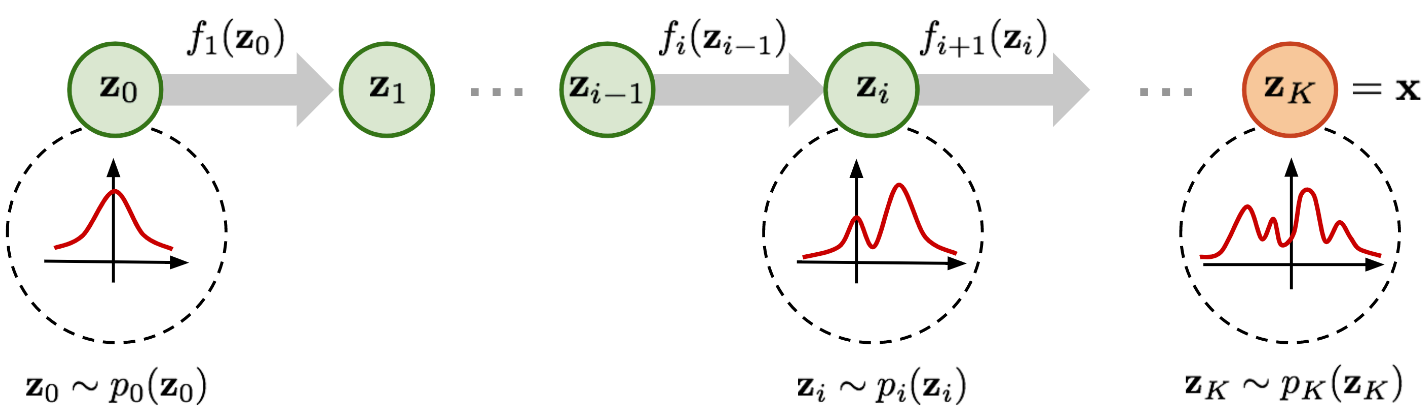
- Idea
  - Sample z_0 from an "easy" distribution, e.g., standard Gaussian
  - Apply K bijections $z_i = f_i(z_{i-1})$
  - The final sample $x = f_K(z_K)$ has tractable desnity
- Normalizing Flow
  - $z_0 \sim N(0,I), z_i = f_i(z_{i-1}), x = Z_K$ where $x, z_i \in \mathbb{R}^d$ and f_i is invertible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



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

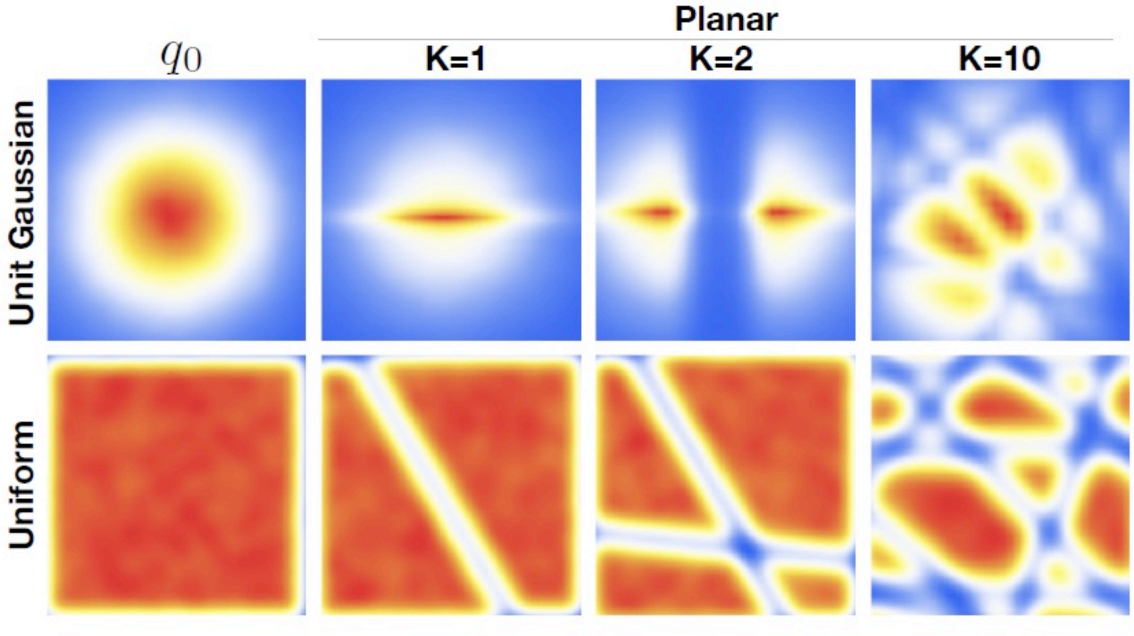
- Naive flow model requires extremely expensive computation
  - Computing determinant of $d \times d$ matrices
- Idea:
  - Design a good bijection $f_i(z)$ such that the determinant is easy to compute

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 (NICE, Dinh et al. ’14)

- Invertible 1x1 convolutions (Kingma et al. ’18)

- Auto-regressive flow:
  - WaveNet (Deepmind ’16)
  - PixelCNN (Deepmind ‘16)

Summary

- Pros:
  - Easy to sample by transforming from a simple distribution
  - Easy to evaluate the probability
  - Easy training (MLE)

- Con
  - Most restricted neural network structure
  - Trade expressiveness for tractability