Variational Autoencoder

Architecture

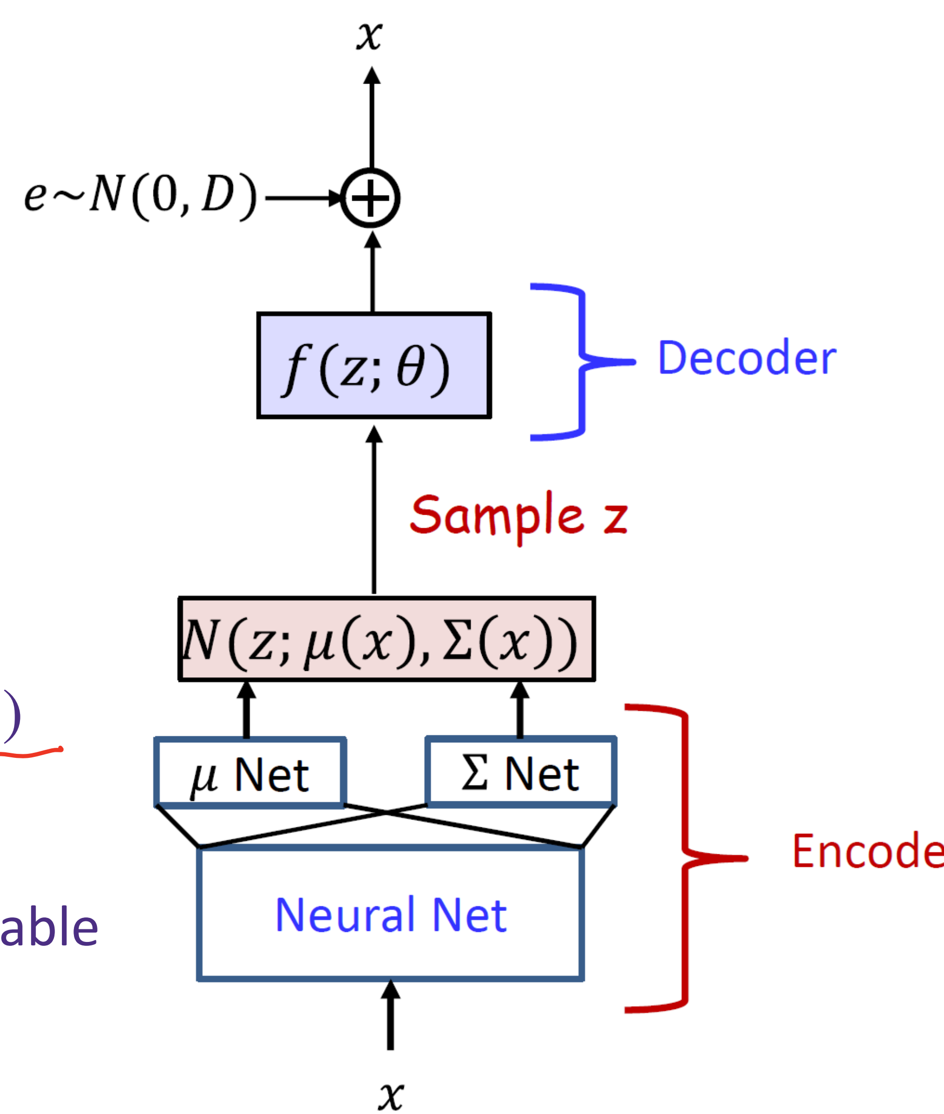
- Auto-encoder: $x \rightarrow z \rightarrow x$
- Encoder: $q(z \mid x; \phi) : x \rightarrow z$
- Decoder: $p(x \mid z; \theta) : z \rightarrow x$

- Isomorphic Gaussian:

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

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

- Probabilistic model interpretation: latent variable model.



VAE Training

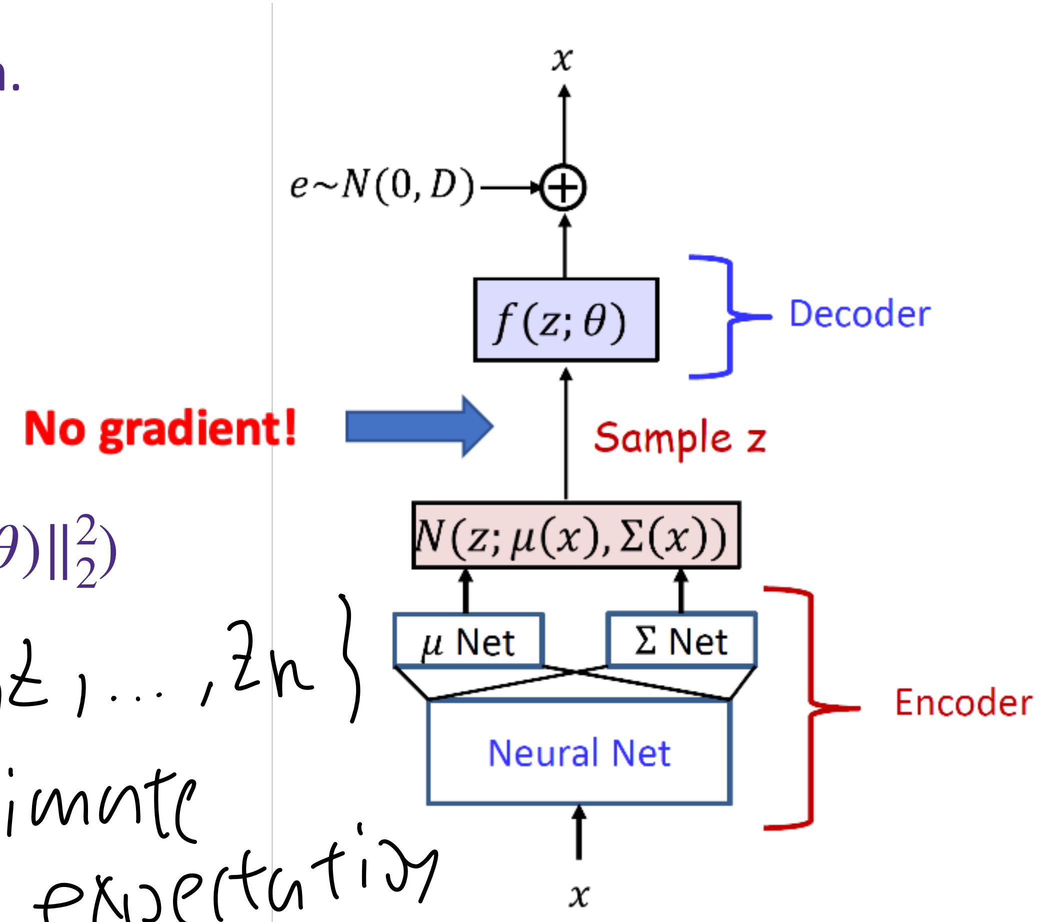
standard MLE: $\mathbb{E}_{x\sim p}\, l(f, y)$

$KL(p \| q) = \mathbb{E}_p \log \frac{p}{q}$

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

$x \mapsto$ sample $\{z_1, \ldots, z_n\}$

$\to$ approximate expectation



VAE Training

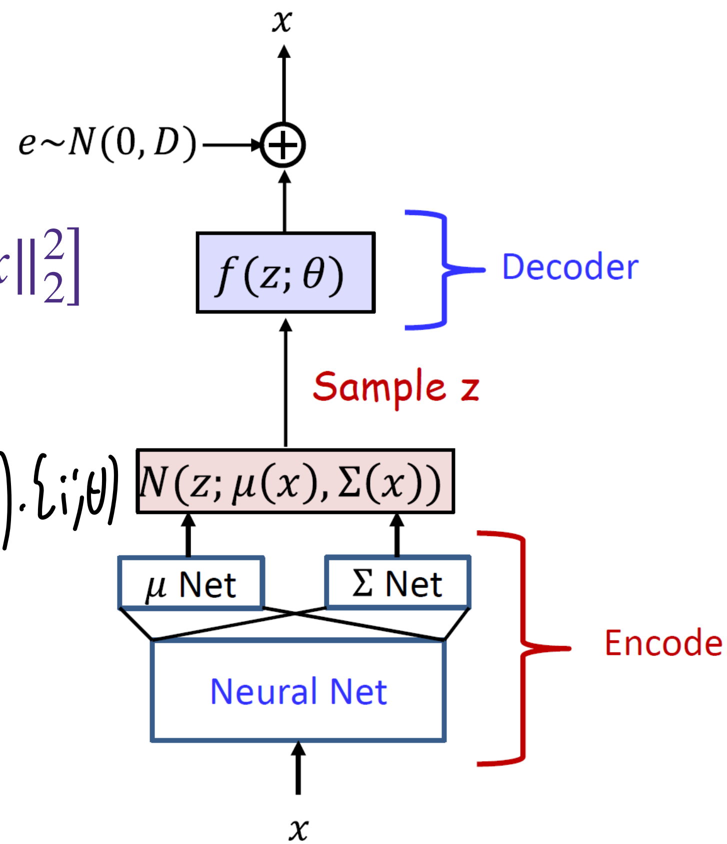
- Likelihood term (reconstruction loss):
  - Gradient for ϕ . Loss: $L(\phi) = \mathbb{E}_{z\sim q(z;\phi)}\left[\log p(x\,|\,z)\right]$
  - Reparameterization trick:
    - $z \sim N(\mu, \Sigma) \Leftrightarrow z = \mu + \epsilon, \epsilon \sim N(0,\Sigma)$
  - $L(\phi) \propto \mathbb{E}_{z\sim q(z|\phi)}\left[\|f(z;\theta)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

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

take $\epsilon_1, \ldots \epsilon_k \sim N(0, I)$

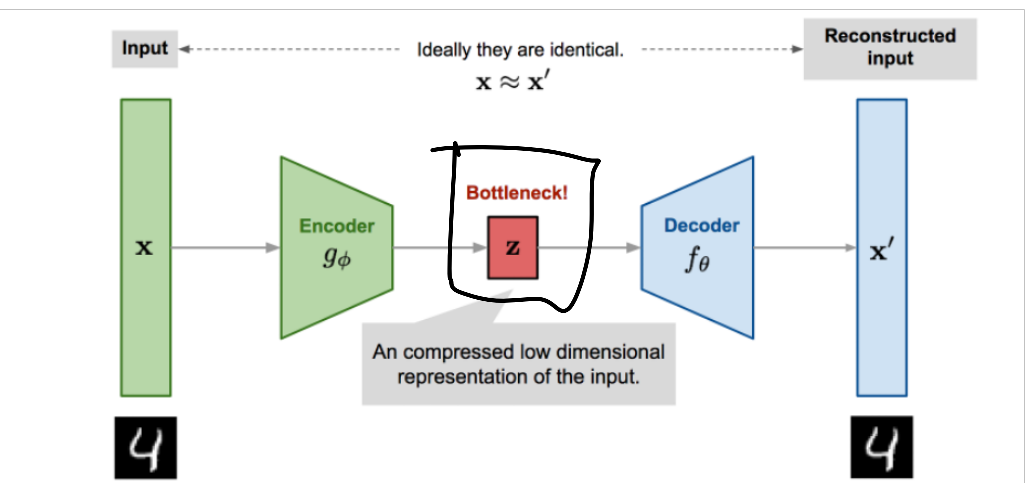
approx loss $\propto \sum_i \|f(\mu(x,\phi) + \sigma(x;\phi)\cdot\epsilon_i;\theta) - x\|_2^2$

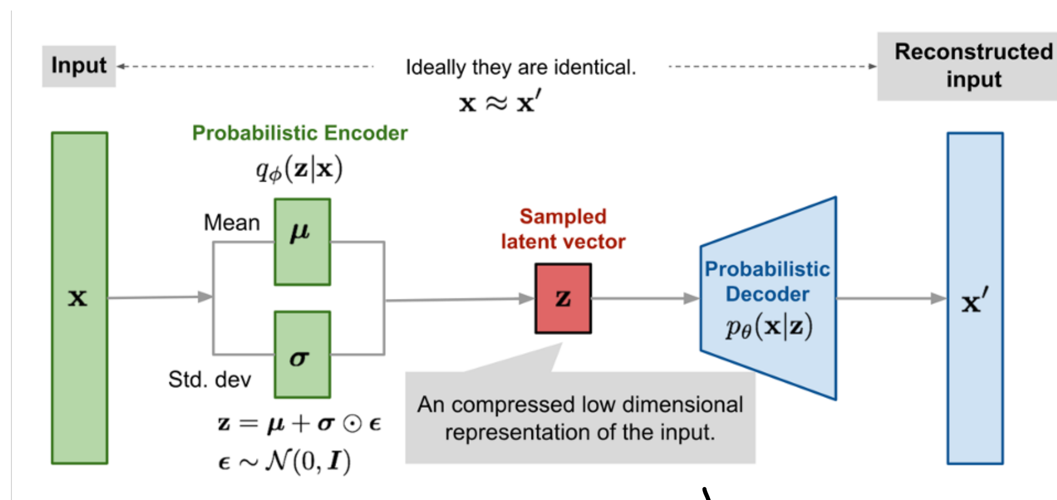
- End-to-end training



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





$z \sim N(0, I)$

apply $P(x \mid z; \theta)$

(ϵ) $\quad x \sim f(z;\theta) + \epsilon$

Conditioned VAE

- Semi-supervised learning: some labels are also available

decoder

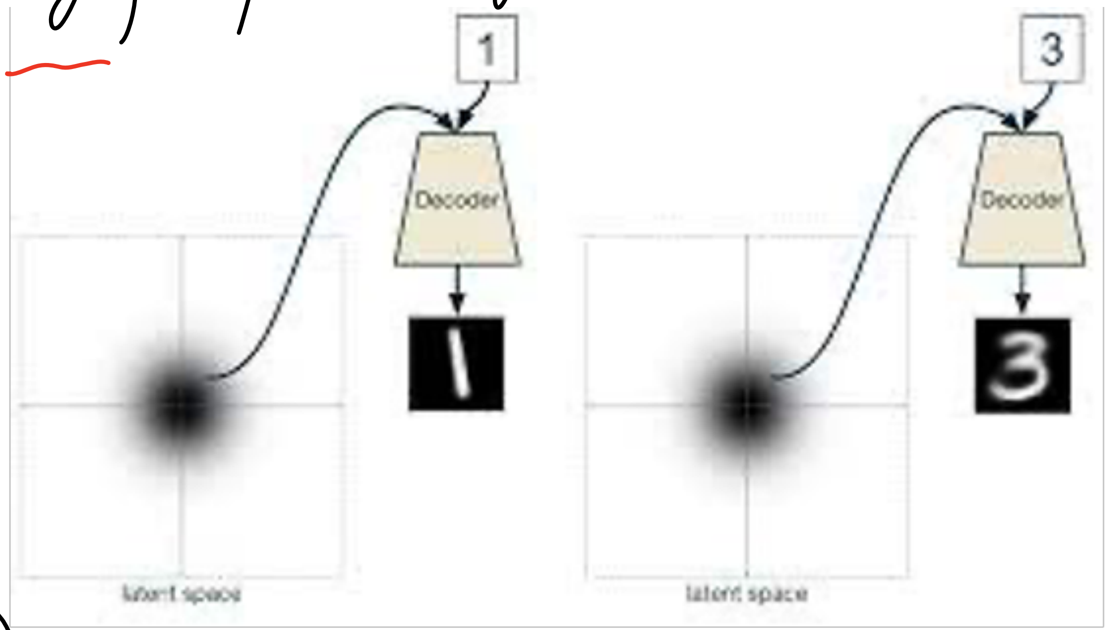
$p(x \mid z, \underset{\sim}{y}, \theta)$

$f(z, y; \theta)$

$z \sim N(0, I)$

$f(z, y, \theta)$

y



conditioned generation

Comments on VAE

- Pros:
  - Flexible architecture
  - Stable training

- Cons:
  - Inaccurate probability evaluation (approximate inference)

$$p(q_\phi(x))$$

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

$z = \sum_{S} E(S)$

- 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

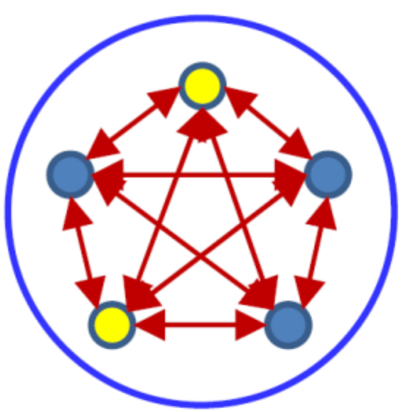
$T \to \infty$

2^d terms

$y_i \in \{0,1\}$

$\boxed{S = y}$

$S \in \mathbb{R}^d$,



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

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

Boltzmann Machine: Training

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

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

$y' \in \{0,1\}$

  - Maximum log-likelihood:

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

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

2^d : # of y'

Boltzmann Machine: Training

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

$W \in \mathbb{R}^{d \times d}$, focus on W_{ij}

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

$\Rightarrow$ Monte-Carlo sample $S = \{y^1, \ldots, y^K\}$

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

Boltzmann Machine: ~~Training~~ Sampling

Sampling: MCMC

Initialize $y(0) \in \Omega^d$

for $t = 1, \ldots, T$

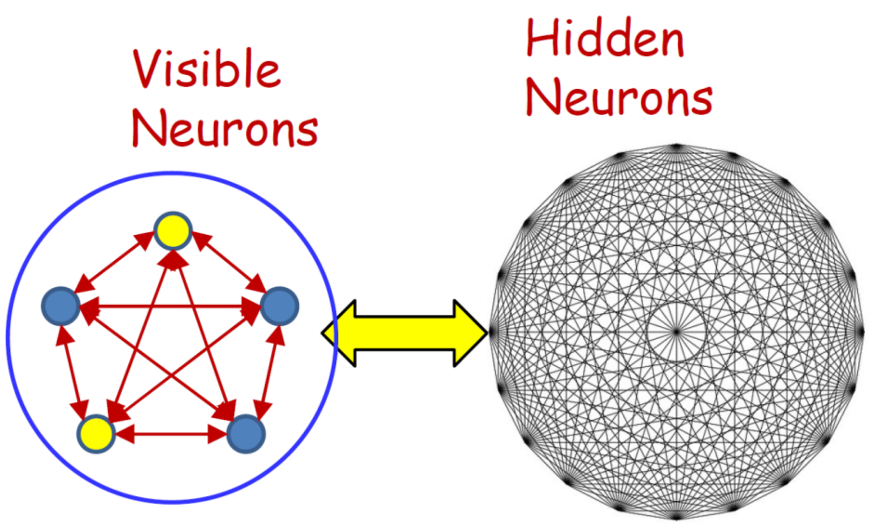
iterate $i \in \{1, \ldots, d\}$, conditioned sampling

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

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

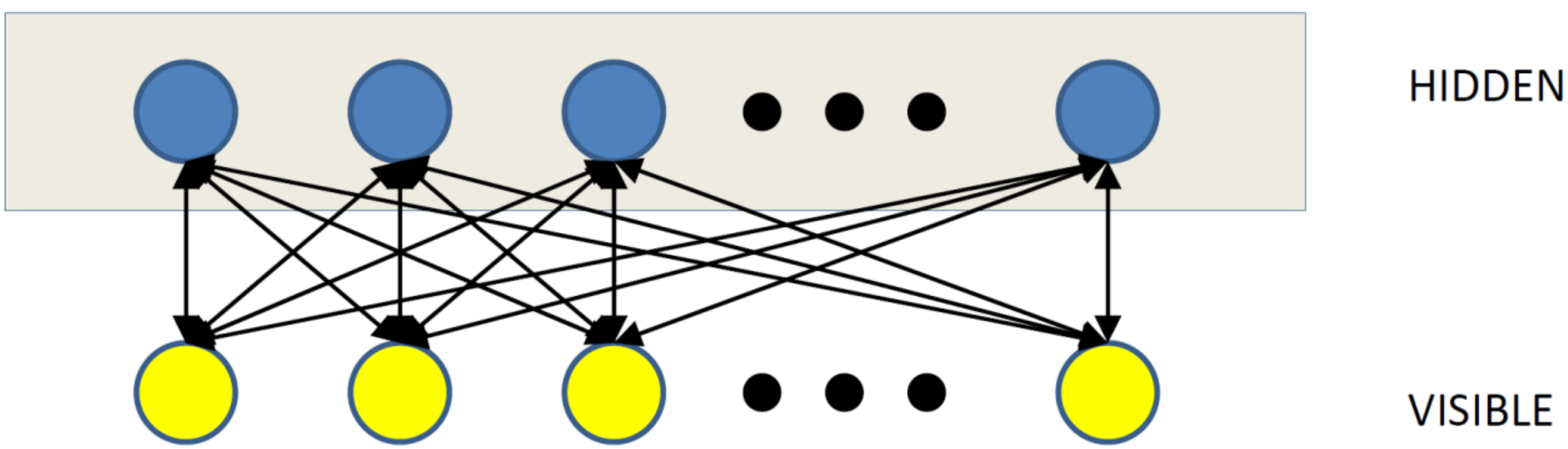
Boltzmann Machine with Hidden Neurons

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



Restricted Bolzmann Machine

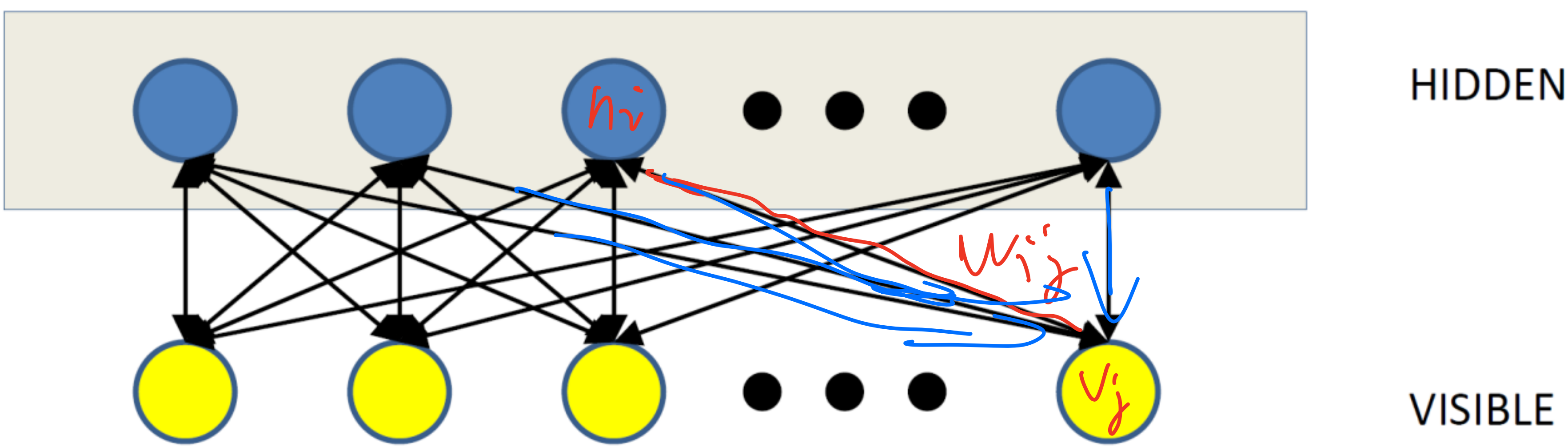
- A structured Boltzmann Machine
  - Hidden neurons are only connected to visible neurons
  - No intra-layer connections
  - Invented by Paul Smolensky in ’89
  - Became more practical after Hinton invested fast learning algorithms in mid 2000



Restricted Bolzmann Machine

init $h(0), v(0)$
generate $h(t)$ conditioning $v(t-1)$
$v(t)$ condition on $h(t)$

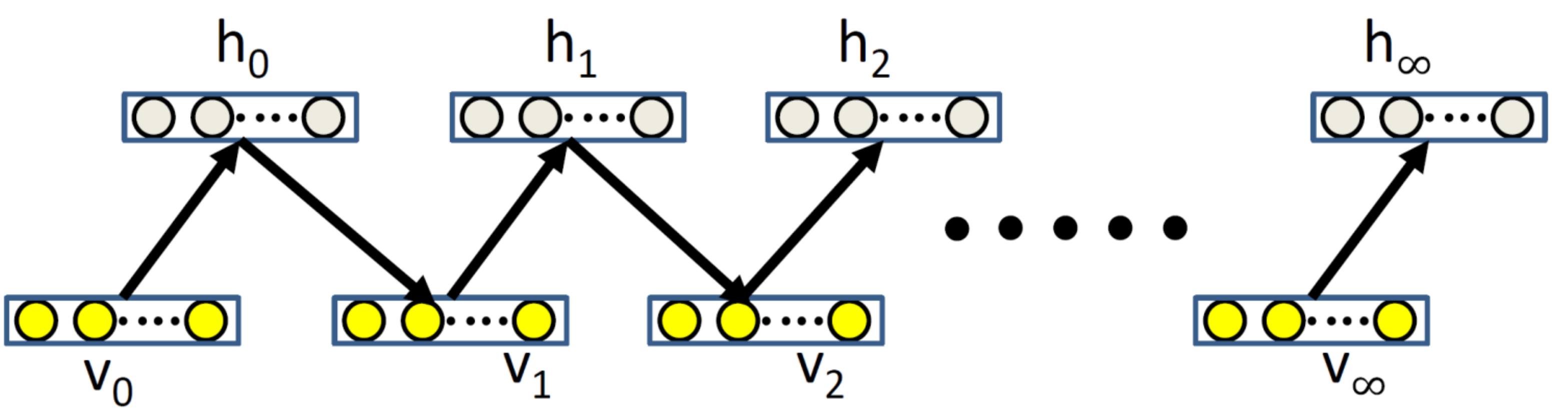
- Computation Rules
  - Iterative sampling MCMC
  - Hidden neurons h_i: $z_i = \sum_j w_{ij} v_j$, $P(h_i \mid v) = \dfrac{1}{1 + \exp(-z_i)}$
  - Visible neurons v_j: $z_j = \sum_i w_{ij} h_i$, $P(v_j \mid h) = \dfrac{1}{1 + \exp(-z_j)}$



Restricted Bolzmann Machine

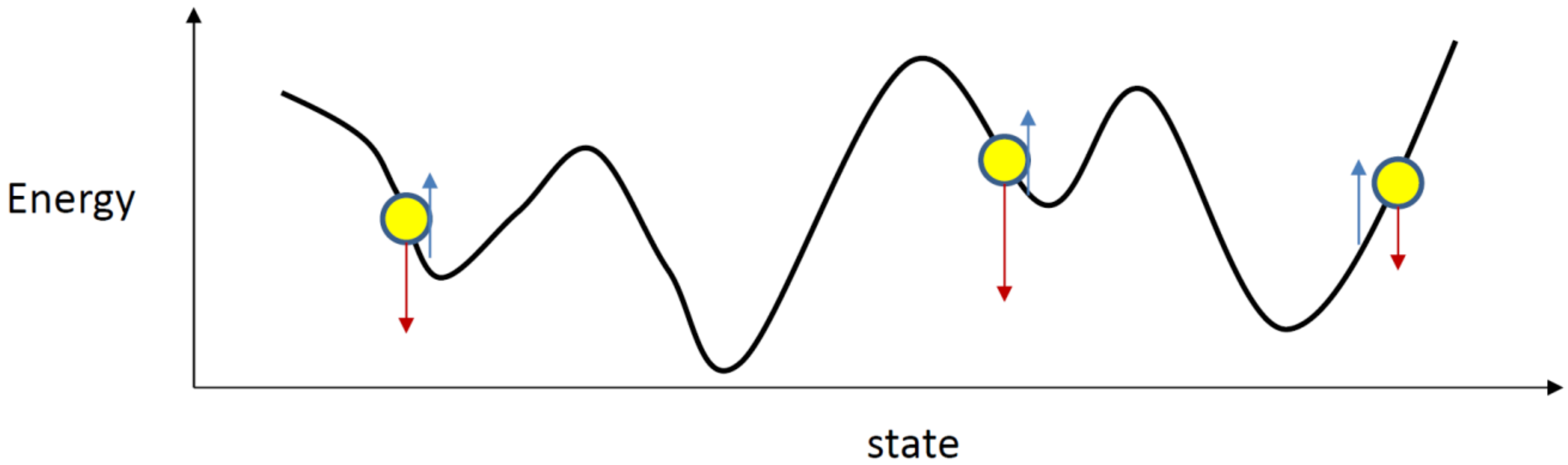
- Sampling:
  - Randomly initialize visible neurons v_0
  - Iterative sampling between hidden neurons and visible neurons
  - Get final sample (v_∞, h_∞)

- Training:
  - MLE
  - Sampling to approximate gradient



Restricted Bolzmann Machine

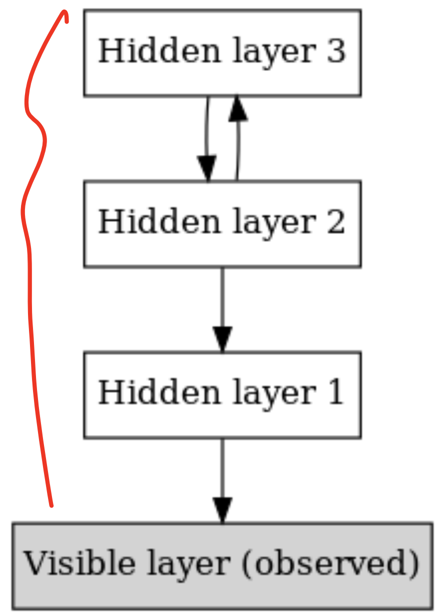
- Maximum likelihood estimated:
- $\nabla_{w_{ij}} L(W) = \frac{1}{N_P K} \sum_{v \in P} v_{0i} h_{0j} - \frac{1}{M} \sum v_{\infty i} h_{\infty j}$
- No need to lift up the entire energy landscape!
    - Raising the neighborhood of desired patterns is sufficient



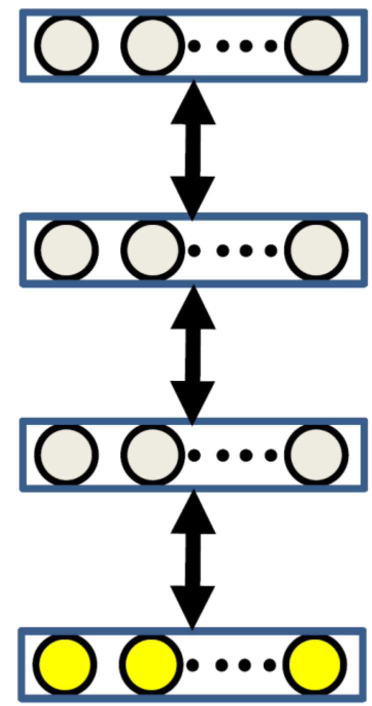
Deep Bolzmann Machine

fix h_1, h_2, h_3, V

- 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

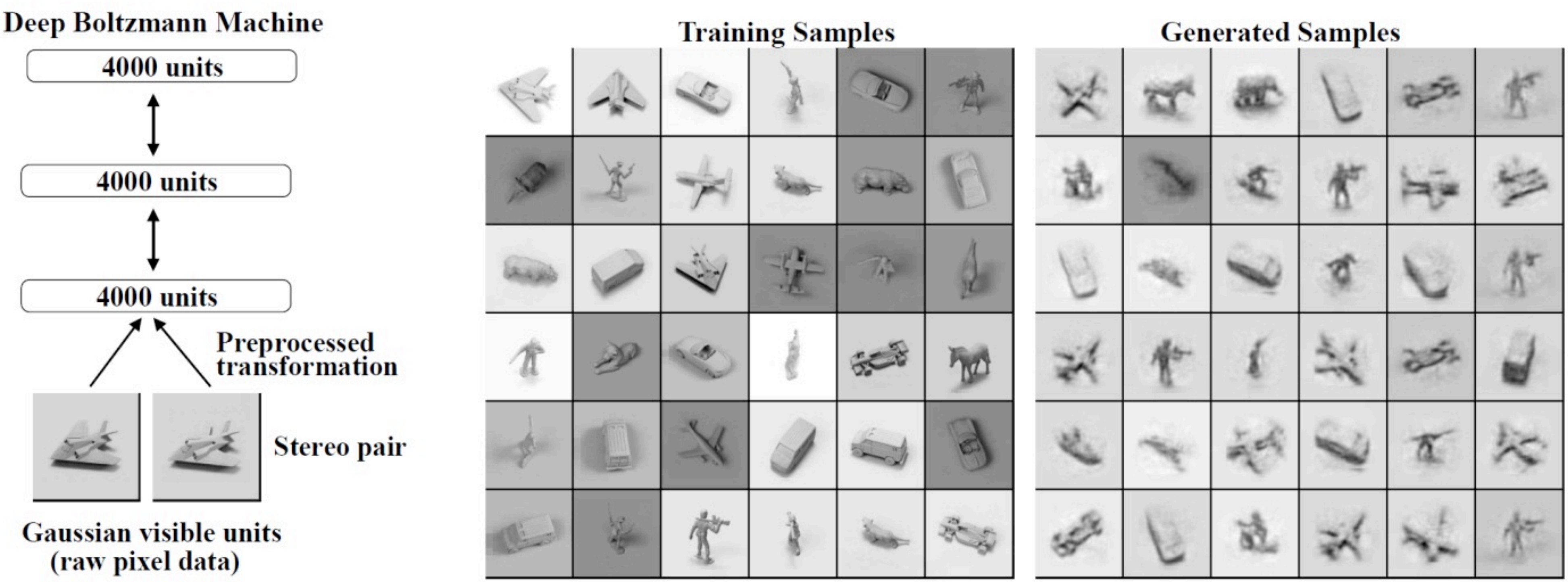


deep belief net



Deep Boltzmann Machine

Deep Bolzmann Machine



Summary

- Pros: powerful and flexible
  - An arbitrarily complex density function $p(x) = \frac{1}{z}\exp(-E(x))$

VAE or GAN

$z \sim N(0, I)$, $f(z)$

- Cons: hard to sample / train
  - Hard to sample:
    - MCMC sampling

$\Rightarrow \{X_1, X_2, \cdots X_T\} \approx p(X)$

  - Partition function
    - No closed-form calculation for likelihood
    - Cannot optimize MLE loss exactly
    - MCMC sampling