Energy-Based Models

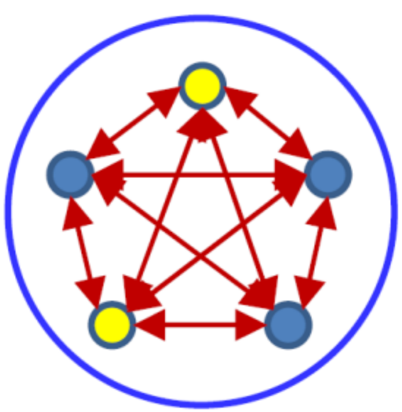
Energy-based Models

- 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

$y \in R^d$, $y_i \in \{1, -1\}$



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

$s_j = y_j$

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

Boltzmann Machine: Training

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

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

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

  - Maximum log-likelihood:

max W

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

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

of y' : 2^d

Boltzmann Machine: Training

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

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

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

$$= \mathbb{E}_{y'}[y_i' \cdot y_j']$$

sample $S = \{ y'_{(1)}, y'_{(2)}, \ldots, y'_{(n)} \}$

$$\Rightarrow \nabla_{W_{ij}} L \approx \frac{1}{N} \sum_{y \in D} y_i \cdot y_j - \frac{1}{n} \sum_{y'_{(k)} \in S} y'_{(k),i} \cdot y'_{(k),j}$$

Boltzmann Machine: Training

Sampling: MCMC

- initialize $y(0) \in R^d$

for $t = 1, \dots, T$

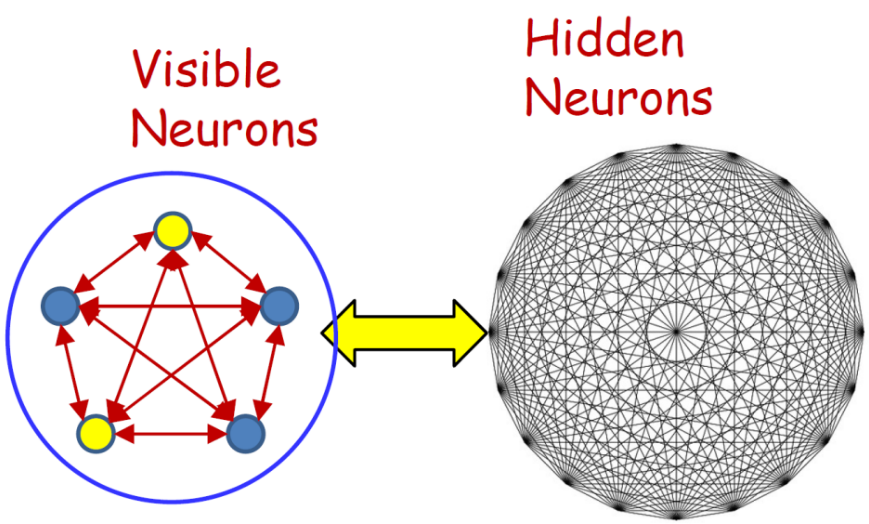
iterate over $i \in \{1, \dots, d\}$, conditional sampling

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

$T \to \infty$, $\{y(1), \dots, y(t)\} \to P$

Boltzmann Machine with Hidden Neurons

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

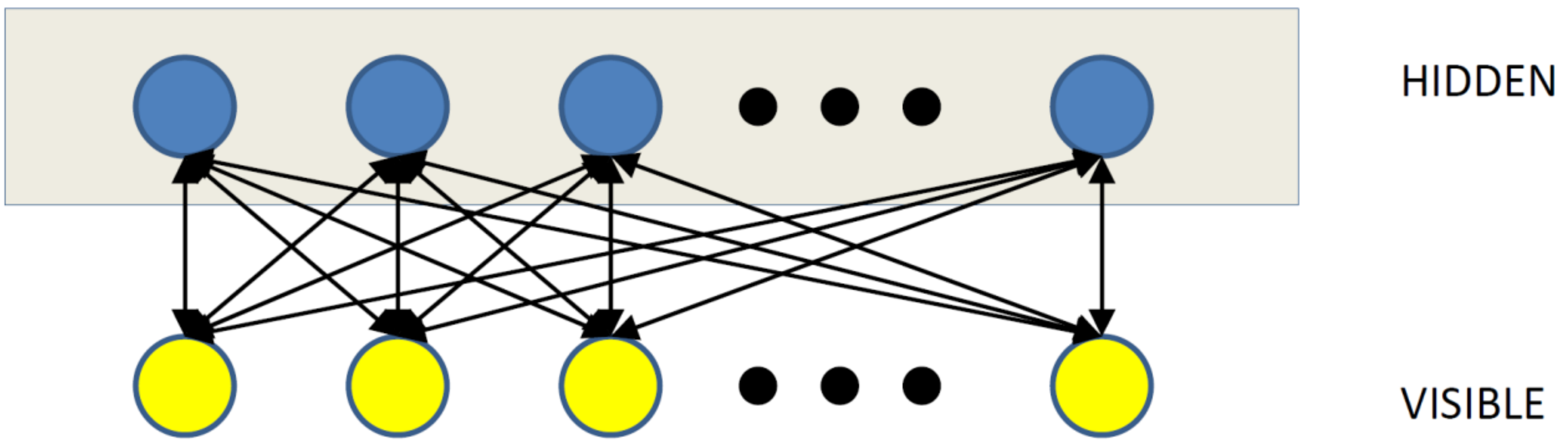


Boltzmann Machine with Hidden Neurons: Training

Boltzmann Machine with Hidden Neurons: Training

Restricted Bolzmann Machine

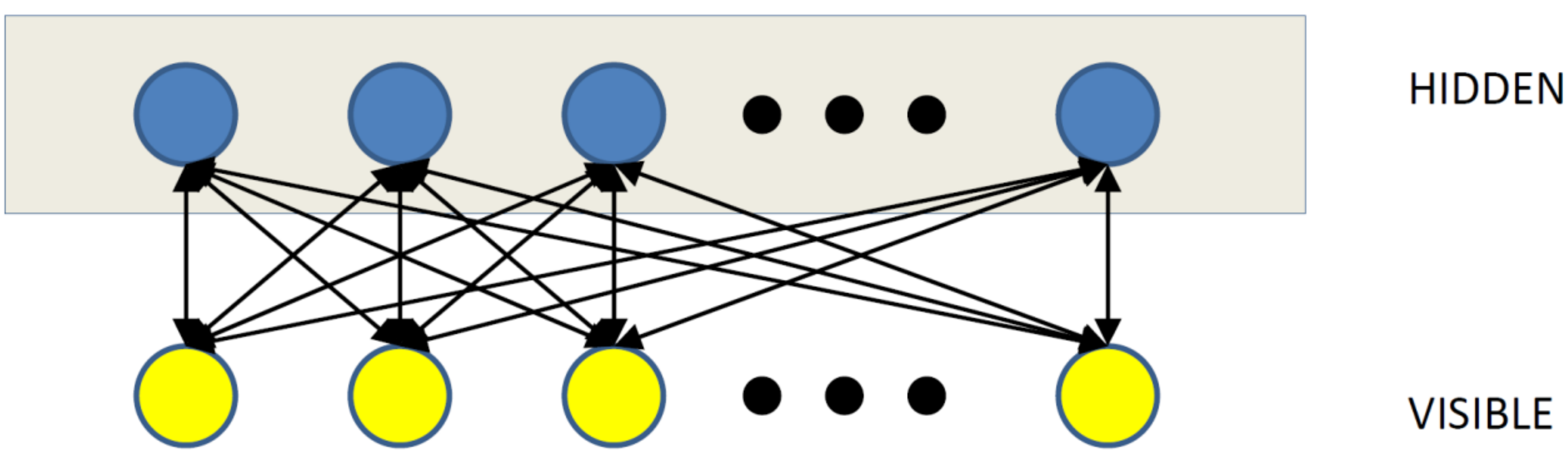
- A structured Boltzmann Machine
  - Hidden neurons are only connected to visible neurons
  - No intra-layer connections
  - Invented by Paul Smolensky in ’89
  - Became more practical after Hinton invested fast learning algorithms in mid 2000



Restricted Bolzmann Machine

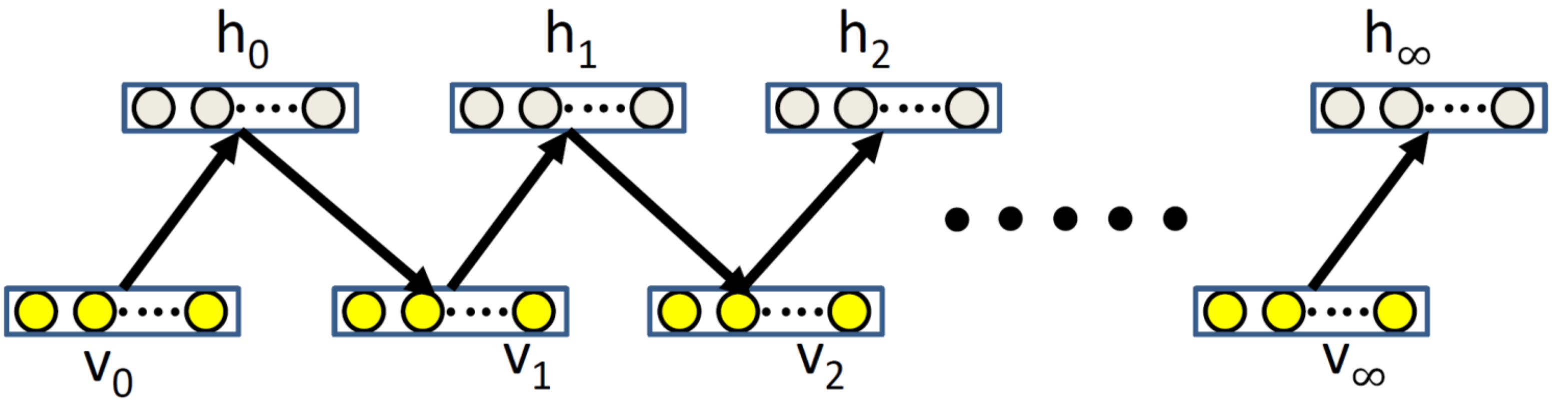
u(0), v(0)   W

- 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: $z_j = \sum_i w_{ij} h_i$, $P(v_j \mid h) = \dfrac{1}{1 + \exp(-z_j)}$



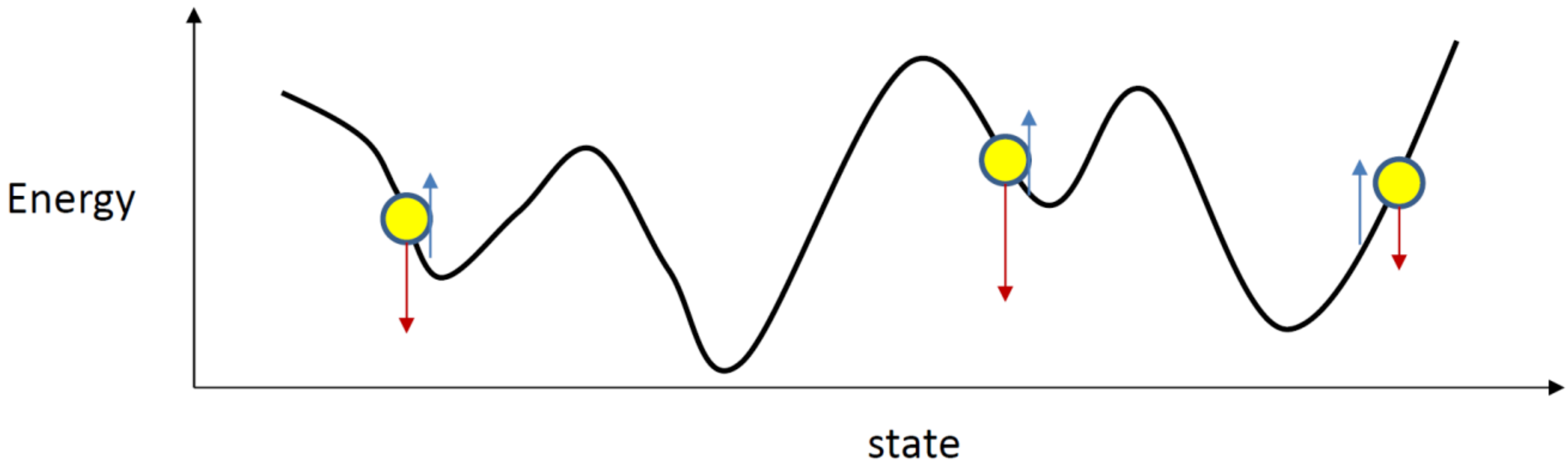
Restricted Bolzmann Machine

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



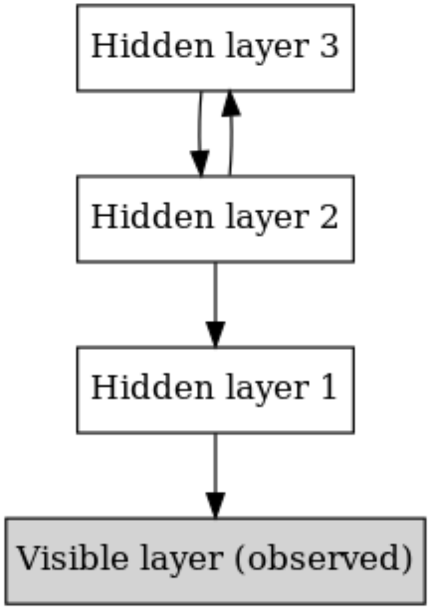
Restricted Bolzmann Machine

- 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

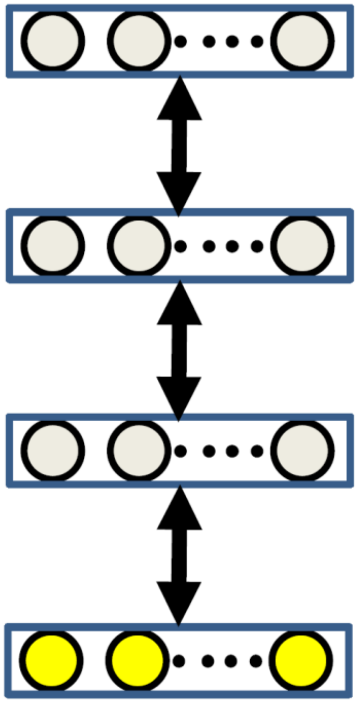


Deep Bolzmann Machine

- 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

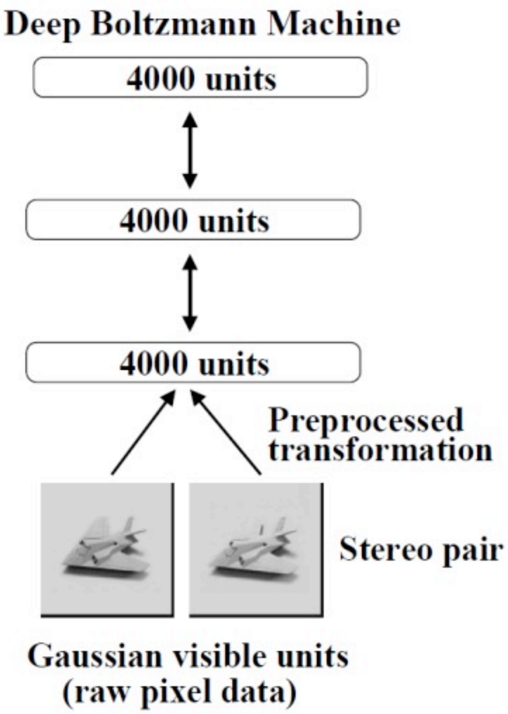


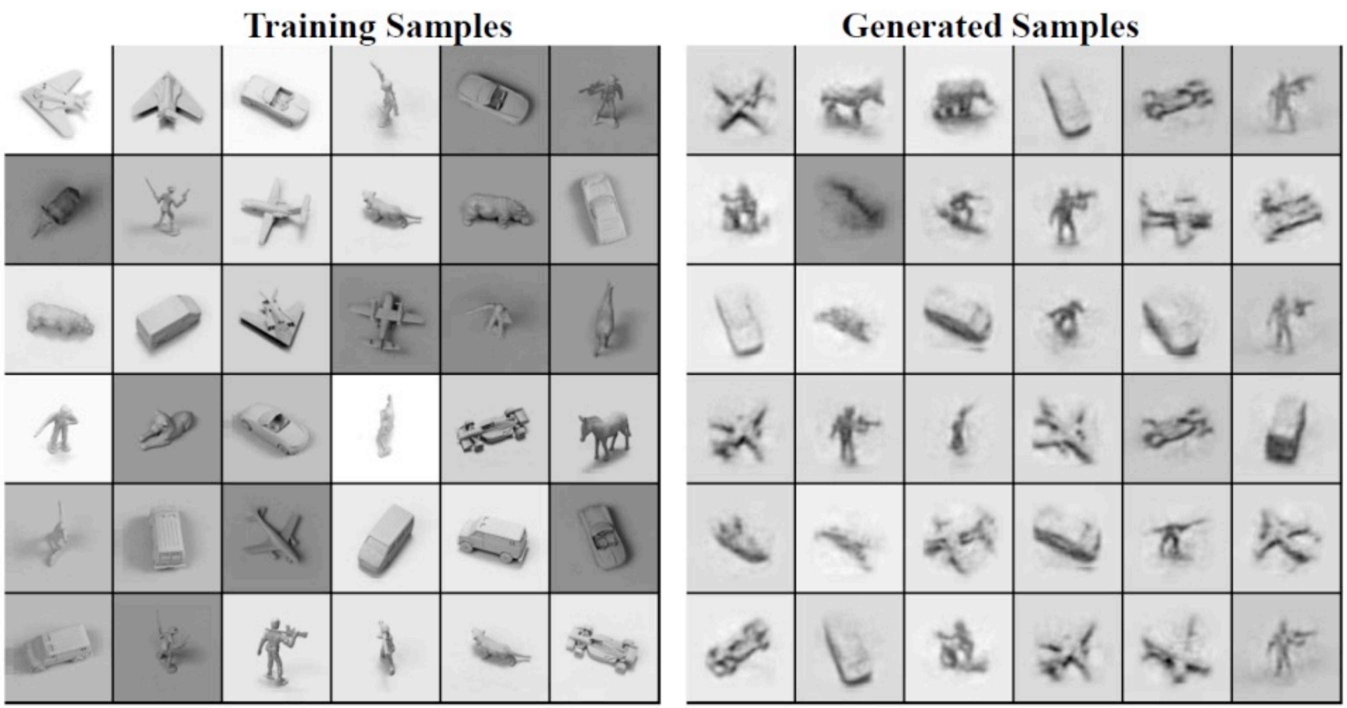
deep belief net



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

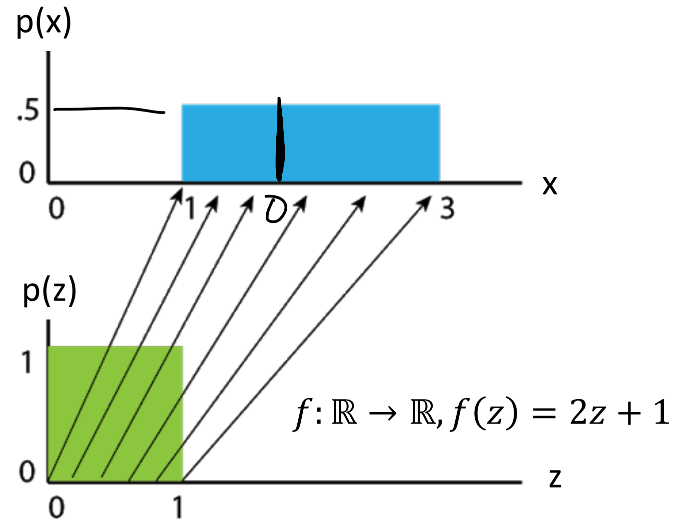
- Goal: design $p(x)$ such that
  - Easy to sample
  - Tractable likelihood (density function)

- 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

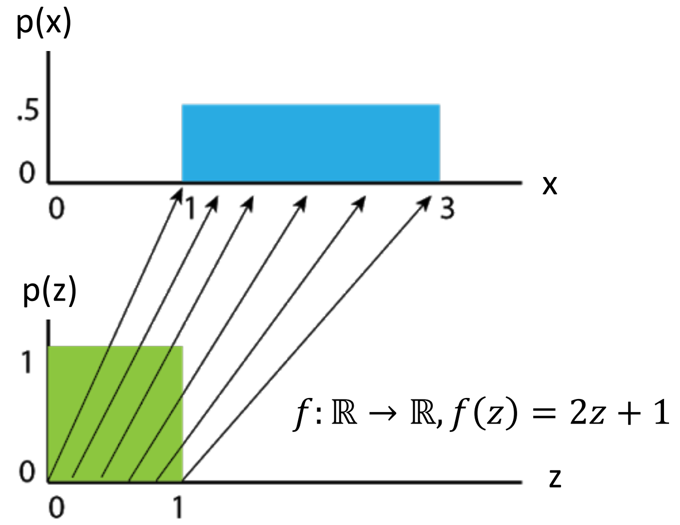
- Goal: design $f(z;\theta)$ such that
  - Assume z is from an "easy" distribution
  - $p(x) = p(f(z;\theta))$ has tractable likelihood

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

x ⟷ z

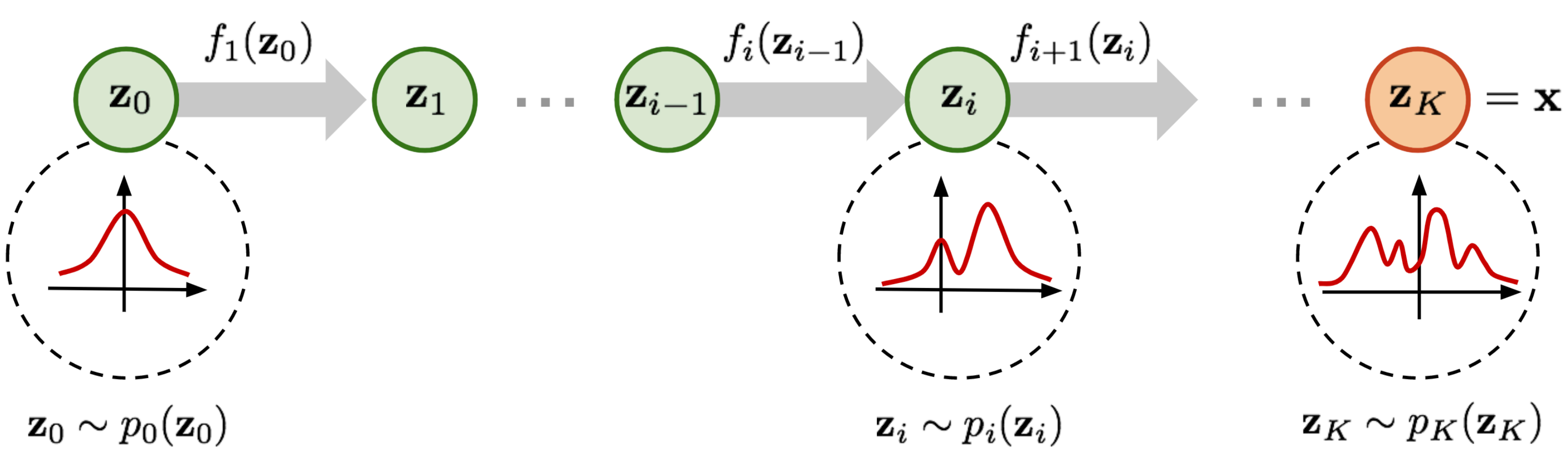
p(x)

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 $f(z) : \mathbb{R}^d \rightarrow \mathbb{R}^d$
  - $z = f^{-1}(x)$
  - $$p(x) = p(f^{-1}(x)) \left| \det \left(\frac{\partial f^{-1}(x)}{\partial x} \right) \right| = p(z) \left| \det \left(\frac{\partial f^{-1}(x)}{\partial x} \right) \right|$$
  - 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

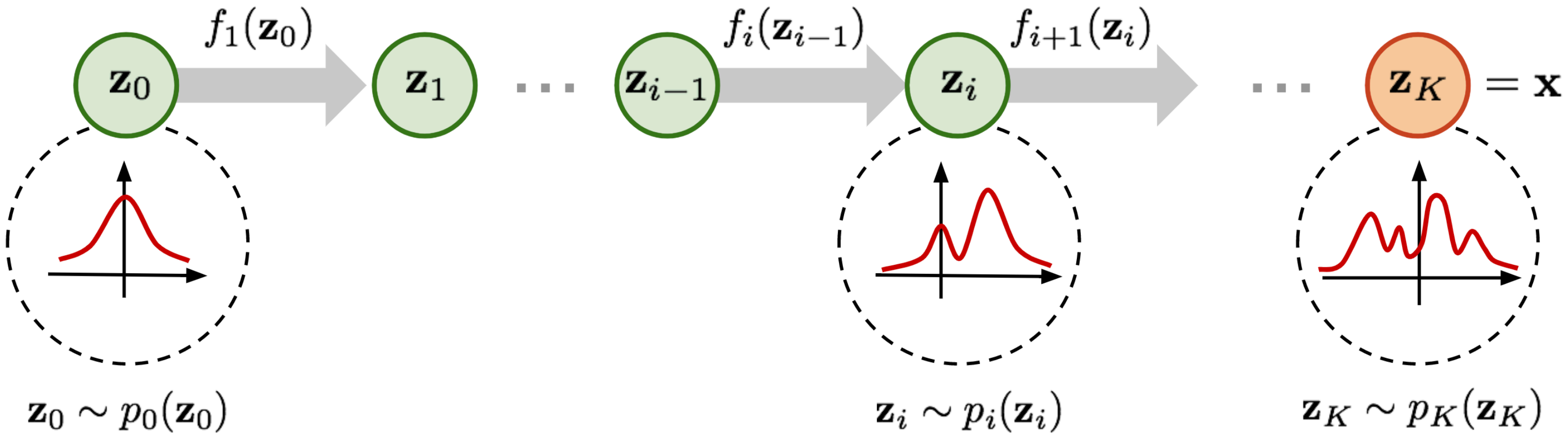
- Idea
  - Sample z_0 from an "easy" distribution, e.g., standard Gaussian
  - Apply K bijections $z_i = f_i(z_{i-1})$
  - The final sample $x = f_K(z_K)$ has tractable desnity
- Normalizing Flow
  - $z_0 \sim N(0,I), z_i = f_i(z_{i-1}), x = Z_K$ where $x, z_i \in \mathbb{R}^d$ and f_i is invertible
  - 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

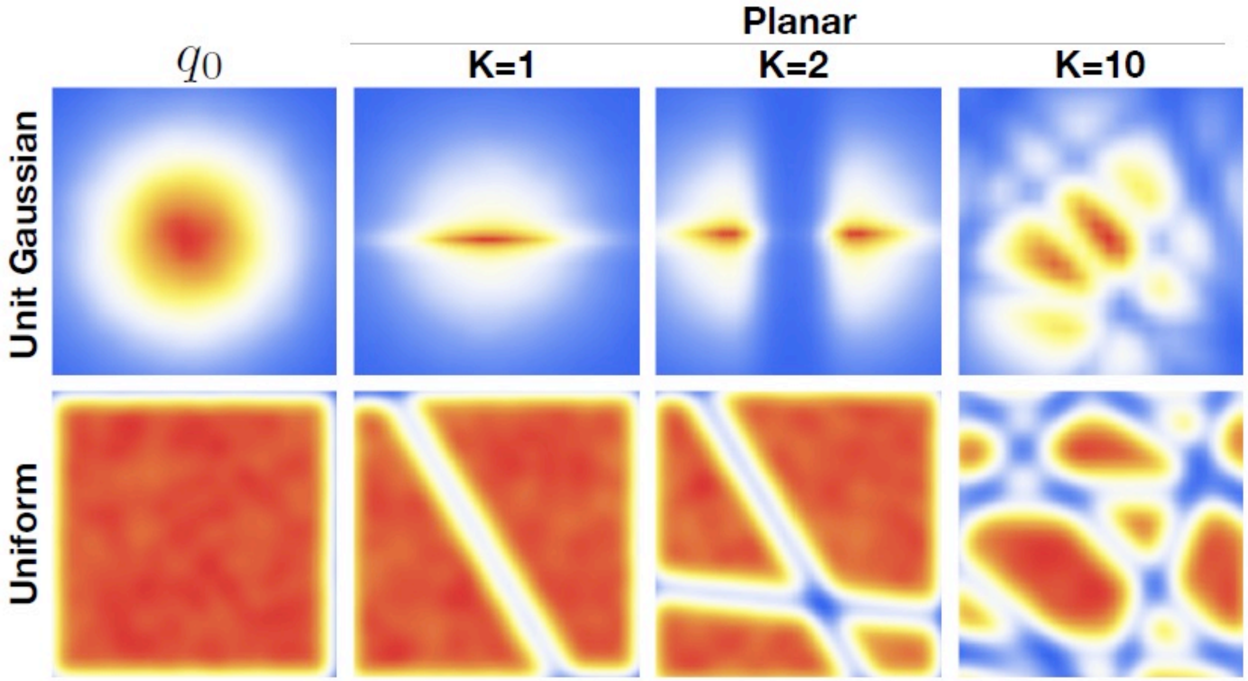
- Naive flow model requires extremely expensive computation
  - Computing determinant of $d \times d$ matrices
- Idea:
  - Design a good bijection $f_i(z)$ such that the determinant is easy to compute

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 $\tanh(\cdot) (0 < h'(\cdot) < 1)$
  - $\theta = [u, w, b]$, $\det \left(\frac{\partial f}{\partial z} \right) = \det(I + h'(w^\top z + b) u w^\top) = 1 + h'(w^\top z + b) u^\top w$
  - 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 $z_{\leq i}$

Score-Based Models and Diffusion Models

Recap: Boltzmann Machine Training

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

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

  - Maximum log-likelihood:

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

Can we avoid calculating the gradient of normalizing constant ($\nabla_x Z_\theta$)?

Score Matching

$$P(X) = \frac{e^{f_\theta(x)}}{\sum_{x'} e^{f_\theta(x')}}$$

$$\nabla_x \log P(x) = \nabla f_\theta(x) - \cancel{\nabla_x \log \sum_{x'} e^{f_\theta(x')}}$$

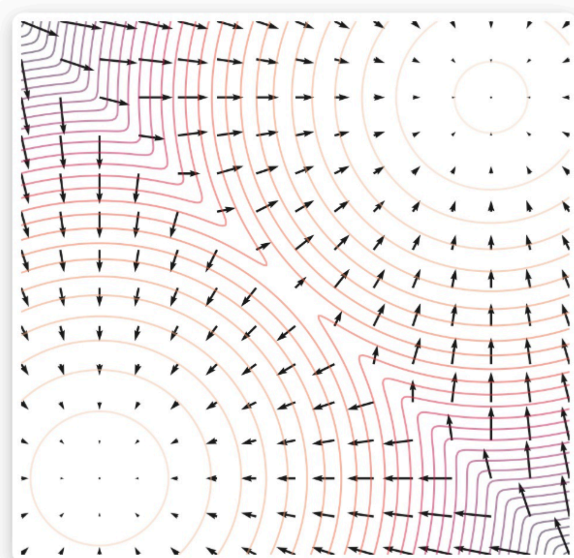
- Score Function
  - Definition:

$$\nabla_x \log p_{data}(x) : \mathbb{R}^d \to \mathbb{R}^d$$

- Idea: directly fitting the score function:
  - $\min_\theta \mathbb{E}_{p_{data}} \| \nabla_x \log p_\theta(x) - \nabla_x \log p_{data}(x) \|^2$
  - No need to compute $\nabla_x Z_\theta$!

- Problem:
  - How to compute $\nabla_x \log p_{data}(x)$?

$\{x_1, \ldots, x_N\}$



Score function (the vector field) and density function (contours) of a mixture of two Gaussians.

Score Matching

$$\mathbb{E}_{p_{data}} \| \nabla_x \log p_\theta(x) - \nabla_x \log p_{data}(x) \|^2$$

$$= \mathbb{E}_{p_{data}} \| \nabla_x \log p_\theta(x) \|^2 + \mathbb{E}_{p_{data}} \| \nabla_x \log p_{data}(x) \|^2$$

$$- 2 \mathbb{E}_{p_{data}} \langle \nabla_x \log p_\theta(x), \nabla_x \log p_{data}(x) \rangle$$

Integration by parts

$$\mathbb{E}_p \langle f(x), \nabla_x \log p(x) \rangle = - \mathbb{E}_p [\operatorname{div} f(x)]$$

where $\operatorname{div} f(x) = \sum_i \frac{\partial f_i(x)}{\partial x_i}$

$$\Rightarrow = +2 \mathbb{E}_{p_{data}} [\operatorname{Tr}(\nabla_x^2 \log p_\theta(x))]$$

$$\text{loss} \Leftarrow \mathbb{E}_{p_{data}} \| \nabla_x \log p_\theta(x) \|^2 + 2 \mathbb{E}_{p_{data}} [\operatorname{Tr} \nabla_x^2 \log p_\theta(x)]$$

Score Matching

Use NN to parametrize $\nabla_x \log p_\theta(x) : \mathbb{R}^d \to \mathbb{R}^d$

$s_\theta(x)$

$\{x_1, \ldots, x_N\}$,

$$\text{loss}: \frac{1}{N} \sum_{i=1}^{N} \underbrace{\| s_\theta(x_i) \|_2^2}_{O(d)} + 2 \underbrace{\left[\mathrm{Tr}\left(\nabla_x s_\theta(x_i) \right) \right]}_{O(d^2)}$$

Sliced Score Matching

$v^T D s_\theta(x) v$

$$L(\theta) = \frac{1}{N} \sum_{x \in D} \|s_\theta(x)\|^2 - 2\left[Tr(Ds_\theta(x))\right]$$

Random Projection

Let M : $d \times d$, v random, $v \in \mathbb{R}^d$, $\mathbb{E}[vv^T] = I$

$$\mathbb{E}[v^T M v] = \mathbb{E}(Tr(Mvv^T))$$
$$= Tr(M\,\mathbb{E}(vv^T))$$
$$= Tr(M)$$

$K \ll d$

Sample $v_1, \ldots, v_K$,

$$\frac{1}{K}\sum_{i=1}^{K} v_i^T M v_i \longrightarrow Tr(M)$$

$K \to \infty$

$O(dK)$

Score Matching: Langevin Dynamics

MCMC

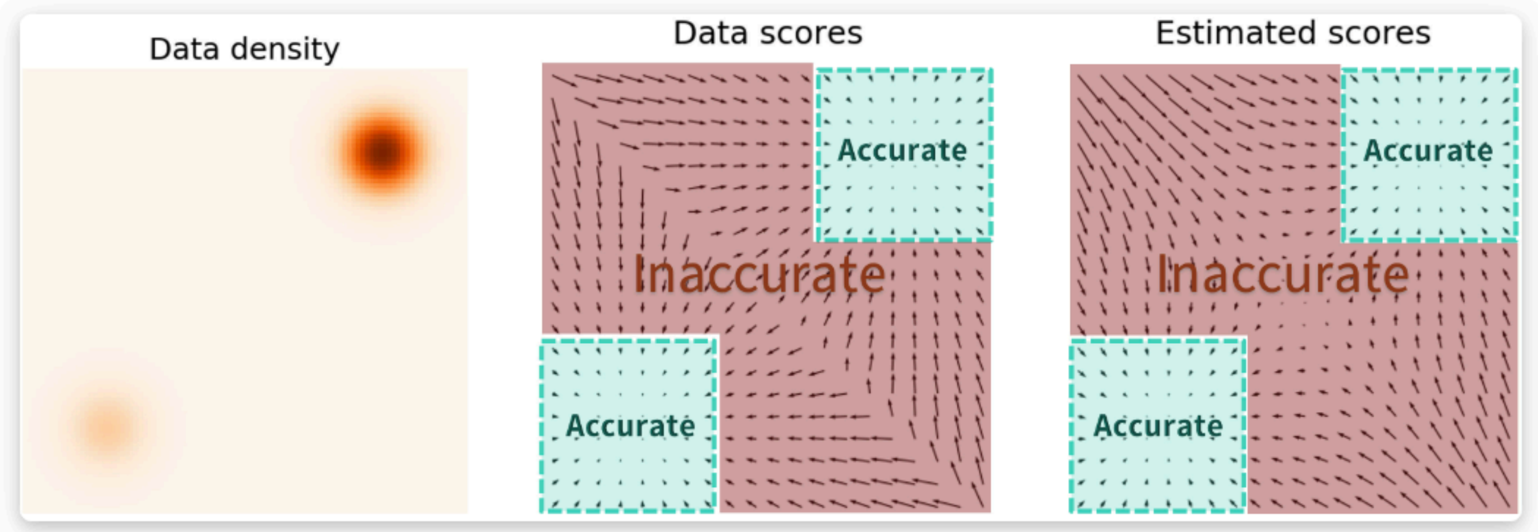
$$x_{t+1} \leftarrow x_t + \epsilon \nabla_x \log p(x) + \sqrt{2\epsilon} z_t, z_t \sim N(0, I)$$

Stationary (equilibrium distribution): p(x)

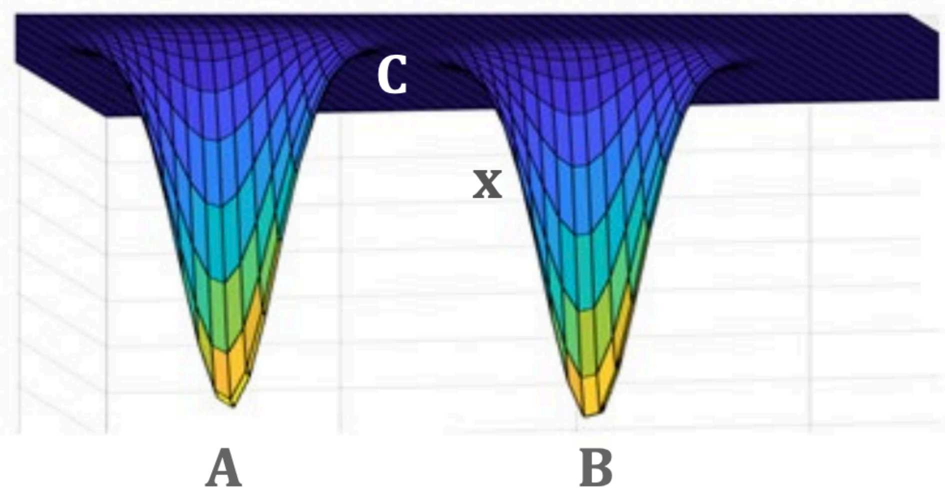
$\{x_1, \dots, x_T\} \xrightarrow{T \to \infty} p(x)$

Practical Issues

- Score function estimation is inaccurate in low density regions (few data available).



- Sampling is Slow



Annealing: Denoising Score Matching

- Fit several "smoothed" versions of p_{data}:
  - Choose temperatures: $\sigma_1, \sigma_2, \ldots, \sigma_T$
  - $p_{\sigma_i,data}(x) = p_{data}(x) * N(0,\sigma_i) = \int_\delta p_{data}(x - \delta)N(x; \delta, \sigma_i)d\delta$
  - Implementation:
    - Take a sample x, draw a sample $z \sim N(0,\sigma_i)$, output $x' = x + z$.

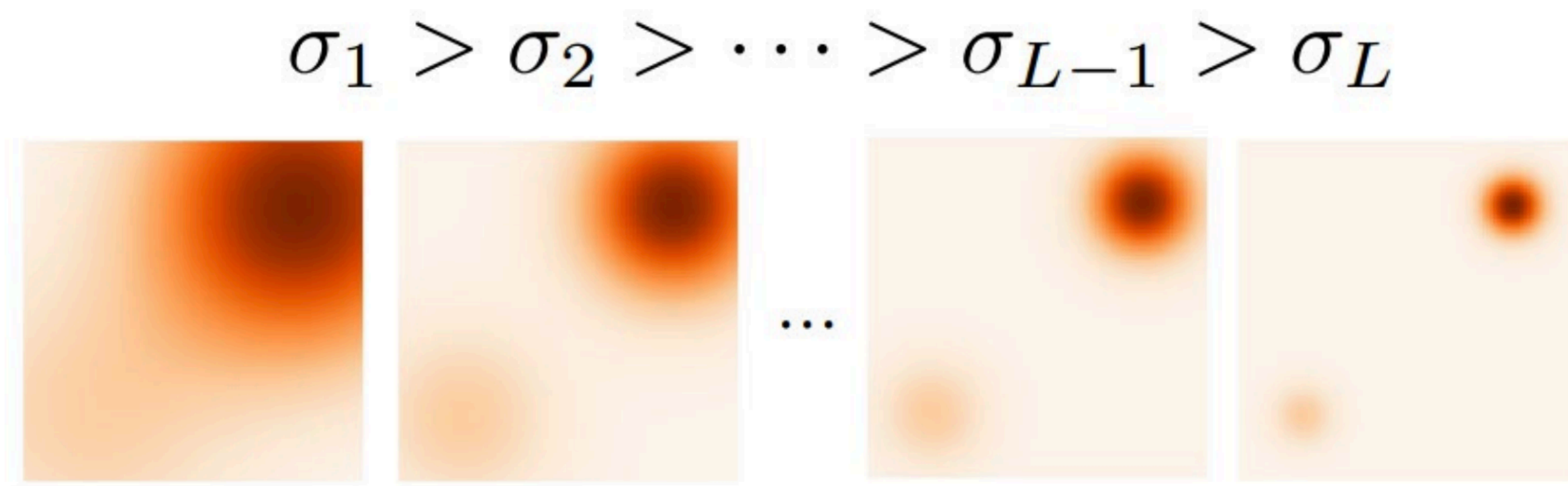


Figure by Stefano Ermon.

Annealing: Denoising Score Matching

$$\arg\min_{\theta} \sum_{i} \lambda(\sigma_i) \mathbb{E}_{x \sim p_{\sigma_i,data}} \|s_\theta(x, i) - \nabla_x \log p_{\sigma_i,data}(x)\|^2$$

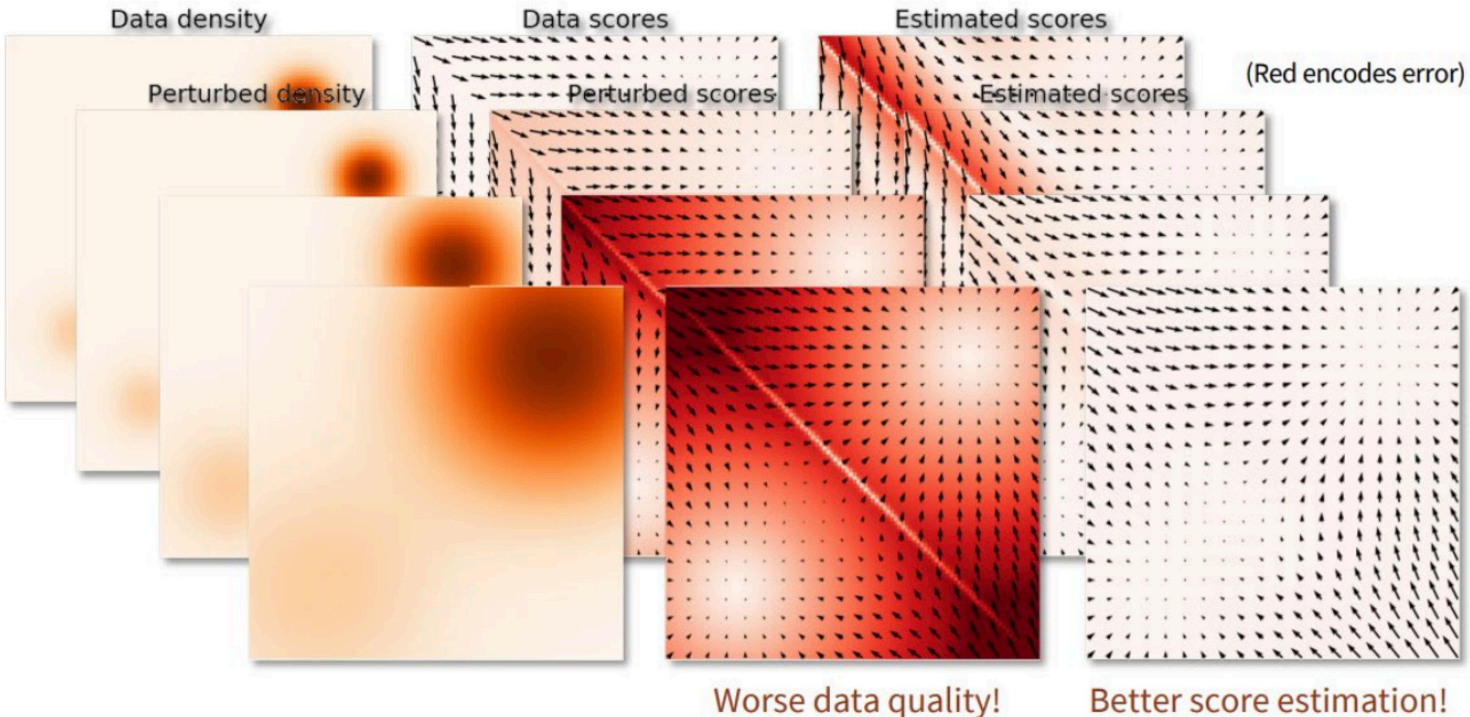


Figure by Stefano Ermon.

Annealed Langevin Dynamics

Algorithm 1 Annealed Langevin dynamics.

Require: $\{\sigma_i\}_{i=1}^{L}, \epsilon, T.$

Initialize $\tilde{\mathbf{x}}_0$

for $i \leftarrow 1$ to L **do**

  $\alpha_i \leftarrow \epsilon \cdot \sigma_i^2 / \sigma_L^2$ ▷ α_i is the step size.

  for $t \leftarrow 1$ to T **do**

    Draw $\mathbf{z}_t \sim \mathcal{N}(0, I)$

    $\tilde{\mathbf{x}}_t \leftarrow \tilde{\mathbf{x}}_{t-1} + \frac{\alpha_i}{2} \mathbf{s}_{\boldsymbol{\theta}}(\tilde{\mathbf{x}}_{t-1}, \sigma_i) + \sqrt{\alpha_i}\, \mathbf{z}_t$

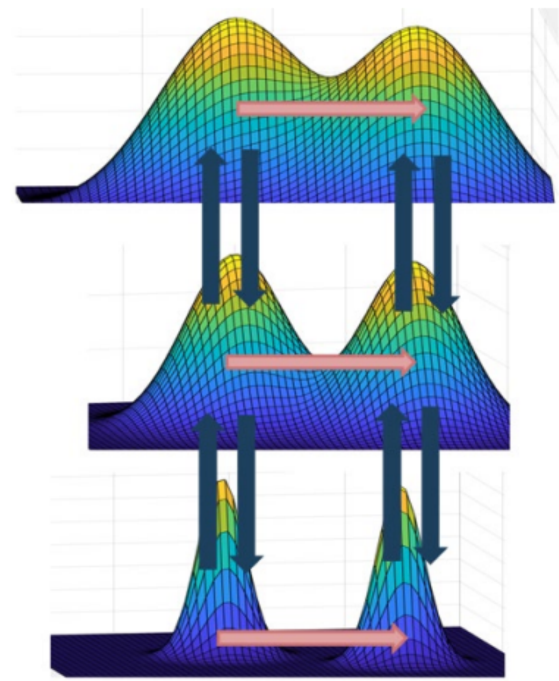
  end for

  $\tilde{\mathbf{x}}_0 \leftarrow \tilde{\mathbf{x}}_T$

end for

return $\tilde{\mathbf{x}}_T$

Figure from Song-Ermon ‘19

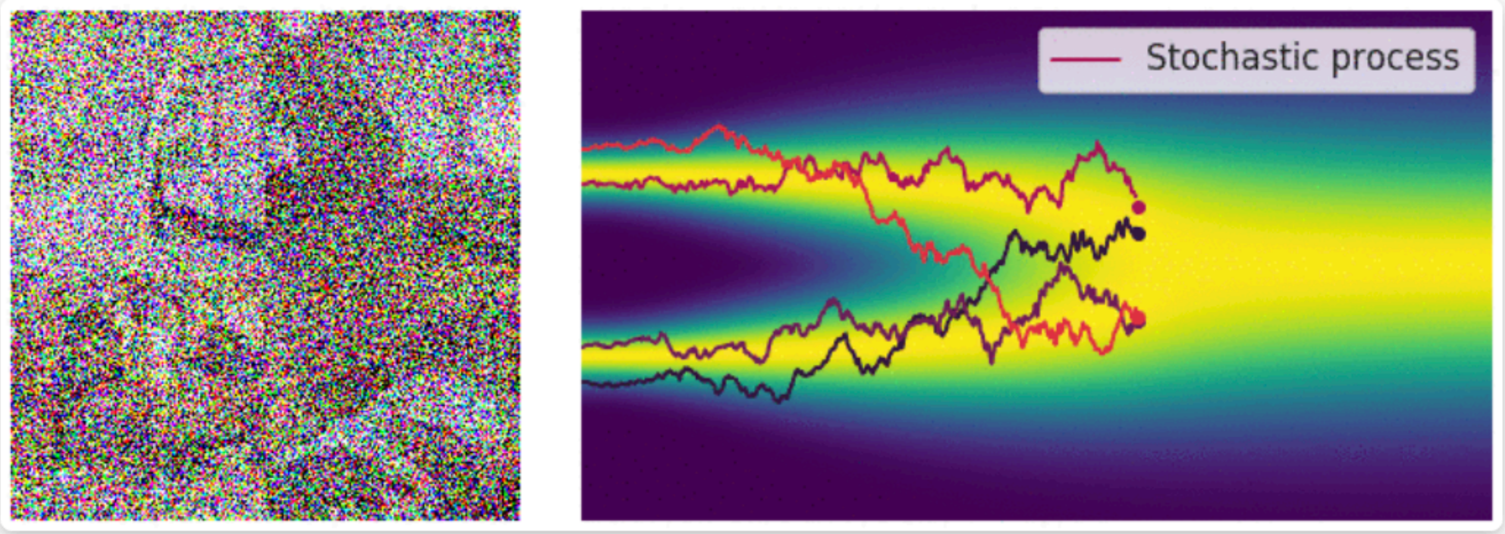


Diffusion Models

An image generated by Stable Diffusion based on the text prompt "a photograph of an astronaut riding a horse"

Perturbing Data with an SDE

- Let the number of noise scales approaches infinity!

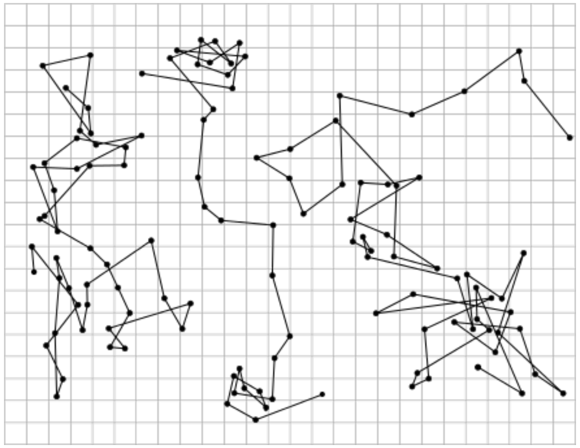


Perturbing data to noise with a continuous-time stochastic process.

Stochastic Differential Equations

$$dx = f(x,t)dt + g(t)dw$$

- x(0): real image, x(T): Gaussian noise.
- f(x,t): drift terms. g(t): diffusion coefficient.
- dw: Brownian motion
  - $w(t+u) - w(t) \sim N(0,u)$
- f(x,t) and g(t) are parts of the model.
  - Variance Exploding SDE: $dx = \sqrt{\frac{d[\sigma^2(t)]}{dt}}dw.$
  - Variance Preserving SDE: $dx = -\frac{1}{2}\beta(t)xdt + \sqrt{\beta(t)}dw.$
  - $\sigma(t), \beta(t)$ are hyper-parameters.



Reversing the SDE

- Reversing the SDE: finding some stochastic process that goes from noise to data.
  - Use to generate data!

- Theorem (Anderson ’82): there exists a reversing SDE, and it has a nice form:

$$dx = [f(x,t) - g^2(t)\nabla_x \log p_t(x)]dt + g(t)dw$$

- Strategy: learn the score function, then solve this reverse SDE.

Reversing the SDE

- Learning the score function: use score matching!

$$\arg\min_\theta \sum_i \lambda(\sigma_i)\mathbb{E}_{x\sim p_{\sigma_i,data}}\|s_\theta(x,i) - \nabla_x \log p_{\sigma_i,data}(x)\|^2$$

$$\Rightarrow \arg\min_\theta \mathbb{E}_{t\sim unif[0,T]}\mathbb{E}_{p_t(x)}\left[\lambda(t)\|s_\theta(x,t) - \nabla_x \log p_t(x)\|^2\right]$$

- Use existing techniques: sliced score matching
- No need to tune temperature schedule
    - Still need to choose a forward SDE, $\lambda(\sigma_i)$, etc
    - Typically choose $\lambda(t) \propto 1/\mathbb{E}\left[\|\nabla_{x(t)} \log p(x(t) \mid x(0))\|^2\right]$

Sampling by Solving the Reverse SDE

$$dx = [f(x,t) - g^2(t)\nabla_x \log p_t(x)]dt + g(t)dw$$

- Euler-Maruyama discretization:
  - $\Delta x \leftarrow [f(x,t) - g^2(t)s_\theta(x,t)]\Delta t + g(t)\sqrt{\Delta t}z_t$
  - $x \leftarrow x + \Delta x$
  - $t \leftarrow t + \Delta t$
- Other solvers:
  - Runge-Kutta
  - Predictor-corrector (Song et al. ’21)

Evaluating Probability by Converting to ODE

- De-randomizing SDE

$$dx = [f(x,t) - g^2(t)\nabla_x \log p_t(x)]dt + g(t)dw$$

$$dx = [f(x,t) - g^2(t)\nabla_x \log p_t(x)]dt, x(T) \sim p_T$$

- Given an initial distribution and an ODE, we can evaluate probability at any time
  - Say given $x(T) \sim p_T$ and $dx = f(x,t)dt$

$$\log p_0(x(0)) = \log p_T(X(T)) + \int_0^T Tr(Df_\theta(x,t))dt$$

  - Solve via ODE.