

# Energy-Based Models

Volodymyr Kuleshov

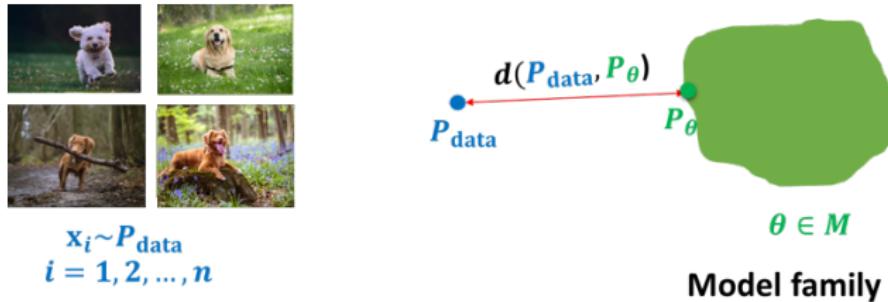
Cornell Tech

Lecture 11

# Announcements

- Assignment 2 is due today
- Assignment 3 will be out next Monday and will be due in two weeks
- Please don't forget to sign up for presentations

# Summary



Story so far:

- **Representation:** Latent variable vs. fully observed models
- **Learning Objectives & Algorithms:** Optimizing likelihood and its approximations (e.g., via variational inference) as well as other divergences

Plan for today: Normalized vs. Energy-Based Models

# Lecture Outline

## ① Energy-Based Models

- Motivation
- Definitions
- Exponential Families
- Motivating Applications

## ② Representation

- Ising Models
- Restricted Boltzmann Machines
- Deep Boltzmann Machines

## ③ Learning

- Likelihood-based learning
- Markov Chain Monte Carlo
- (Persistent) Contrastive Divergence

# Parameterizing Probability Distributions

Representing probability distributions  $p(x)$  is a key challenge in generative modeling. Probability distributions share two properties:

- ① non-negativity:  $p(x) \geq 0$
- ② sum-to-one:  $\sum_x p(x) = 1$  (or  $\int p(x)dx = 1$  for continuous variables)



Sum-to-one is key. Total “volume” is fixed: increasing  $p(x_{train})$  guarantees that  $x_{train}$  becomes relatively more likely (compared to the rest).

# Parameterizing Probability Distributions

Probability distributions  $p(\mathbf{x})$  are a key building block in generative modeling. Properties:

- ① non-negative:  $p(\mathbf{x}) \geq 0$
- ② sum-to-one:  $\sum_{\mathbf{x}} p(\mathbf{x}) = 1$  (or  $\int p(\mathbf{x}) d\mathbf{x} = 1$  for continuous variables)

Coming up with a non-negative function  $g_\theta(\mathbf{x})$  is not hard. For example:

- $g_\theta(\mathbf{x}) = f_\theta(\mathbf{x})^2$  where  $f_\theta$  is any neural network
- $g_\theta(\mathbf{x}) = \exp(f_\theta(\mathbf{x}))$  where  $f_\theta$  is any neural network
- ...

**Problem:**  $g_\theta(\mathbf{x}) \geq 0$  is easy, but  $g_\theta(\mathbf{x})$  might not sum-to-one.

$\sum_{\mathbf{x}} g_\theta(\mathbf{x}) \neq 1$  in general, so  $g_\theta(\mathbf{x})$  is not a valid probability or density.

# Parameterizing Probability Distributions

**Problem:**  $g_\theta(x) \geq 0$  is easy, but  $g_\theta(x)$  might not be normalized

**Solution:** Explicitly normalize  $g_\theta$ :

$$p_\theta(x) = \frac{1}{\text{Volume}(g_\theta)} g_\theta(x) = \frac{1}{\int g_\theta(x) dx} g_\theta(x)$$

Previously, we chose  $g_\theta(x)$  so that we know the volume *analytically*:

- ① **Autoregressive:** Products of normalized objects  $p_\theta(x)p_{\theta'(x)}(y)$ :

$$\int_x \int_y p_\theta(x)p_{\theta'(x)}(y) dx dy = \int_x p_\theta(x) \underbrace{\int_y p_{\theta'(x)}(y) dy}_{=1} dx = \int_x p_\theta(x) dx = 1$$

- ② **Latent variables:** Mixtures of normalized objects  $\alpha p_\theta(x) + (1 - \alpha)p_{\theta'}(x)$  :  
$$\int_x \alpha p_\theta(x) + (1 - \alpha)p_{\theta'}(x) dx = \alpha + (1 - \alpha) = 1$$
- ③ **Flows:** Construct  $p$  via bijection from a simple distribution and track volume change.

How about using models where the “volume” / normalization constant is not easy to compute analytically?

# Energy-Based Models

Energy-based models specify distributions of the form

$$p_\theta(x) = \frac{1}{\int \exp(f_\theta(\mathbf{x})) d\mathbf{x}} \exp(f_\theta(\mathbf{x})) = \frac{1}{Z(\theta)} \exp(f_\theta(\mathbf{x}))$$

The volume/normalization constant

$$Z(\theta) = \int \exp(f_\theta(\mathbf{x})) d\mathbf{x}$$

is also called the partition function. Why exponential (and not e.g.  $f_\theta(\mathbf{x})^2$ )?

- ① Want to capture very large variations in probability. log-probability is the natural scale we want to work with. Otherwise need highly non-smooth  $f_\theta$ .
- ② Many common distributions (“exponential families”) can be written this way.
- ③ These distributions arise under fairly general assumptions in statistical physics (maximum entropy, second law of thermodynamics).  $-f_\theta(\mathbf{x})$  is called the **energy**, hence the name.

# Energy-Based Models

Energy-based models specify distributions of the form

$$p_{\theta}(\mathbf{x}) = \frac{1}{\int \exp(f_{\theta}(\mathbf{x})) d\mathbf{x}} \exp(f_{\theta}(\mathbf{x})) = \frac{1}{Z(\theta)} \exp(f_{\theta}(\mathbf{x}))$$

Pros:

- ① extreme flexibility: can use pretty much any function  $f_{\theta}(\mathbf{x})$  you want

Cons: Computing  $Z(\theta)$  numerically is hard.

- ① Sampling from  $p_{\theta}(\mathbf{x})$  is in general hard
- ② Evaluating and optimizing likelihood  $p_{\theta}(\mathbf{x})$  is hard (learning is hard)
- ③ No feature learning (but can add latent variables)

Nevertheless, some tasks do not require knowing  $Z(\theta)$ . Some algorithms can train energy-based models without using  $Z(\theta)$  (next lectures).

# Applications of Energy-Based Models

Consider an energy-based model of the form:

$$p_{\theta}(\mathbf{x}) = \frac{1}{Z(\theta)} \exp(f_{\theta}(\mathbf{x}))$$

Given  $\mathbf{x}, \mathbf{x}'$  evaluating  $p_{\theta}(\mathbf{x})$  or  $p_{\theta}(\mathbf{x}')$  requires  $Z(\theta)$ . But evaluating  $\frac{p_{\theta}(\mathbf{x})}{p_{\theta}(\mathbf{x}')}$  doesn't:

$$\frac{p_{\theta}(\mathbf{x})}{p_{\theta}(\mathbf{x}')} = \exp(f_{\theta}(\mathbf{x}) - f_{\theta}(\mathbf{x}'))$$

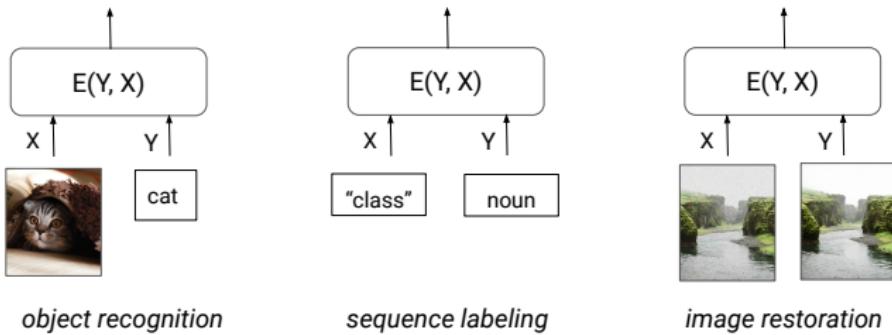
Given a trained energy-based model, many downstream applications are tractable:

- ① Anomaly detection: given new  $\mathbf{x}'$ , call anomaly if  $\frac{p_{\theta}(\mathbf{x})}{p_{\theta}(\mathbf{x}')}$  exceeds threshold
- ② Denoising & Imputation: using an optimization algorithm, find  $\mathbf{x}'$  close to  $\mathbf{x}$  with high  $\frac{p_{\theta}(\mathbf{x})}{p_{\theta}(\mathbf{x}')}$ . Also see Ising models example later.

# Applications of Conditional Energy-Based Models

A conditional energy model has an energy function  $E(X, Y)$  over inputs  $X$  and outputs  $Y$  (categories, sentences, pixel-level predictions).

- Given a trained  $E(X, Y)$ , we can solve many prediction tasks without needing  $Z$



- We can perform inference to find  $Y$  using optimization-based methods.
- There are also ways of training  $E(X, Y)$  without requiring  $Z$ .
- Energy-based learning is a general framework generalizing many types of models that you know: regression, SVMs, CRFs, ranking models, etc.

# Lecture Outline

## ① Energy-Based Models

- Motivation
- Definitions
- Exponential Families
- Motivating Applications

## ② Representation

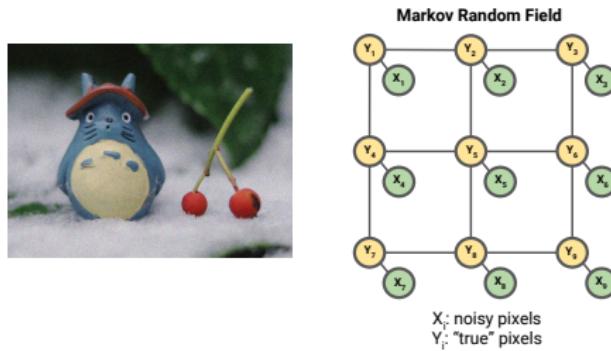
- Ising Models
- Restricted Boltzmann Machines
- Deep Boltzmann Machines

## ③ Learning

- Likelihood-based learning
- Markov Chain Monte Carlo
- (Persistent) Contrastive Divergence

# Ising Model

- There is a true image  $\mathbf{y} \in \{0, 1\}^{3 \times 3}$ , and a corrupted image  $\mathbf{x} \in \{0, 1\}^{3 \times 3}$ . We know  $\mathbf{x}$ , and want to somehow recover  $\mathbf{y}$ .



- We model the joint probability distribution  $p(\mathbf{y}, \mathbf{x})$  as

$$p(\mathbf{y}, \mathbf{x}) = \frac{1}{Z} \exp \left( \sum_i \psi_i(x_i, y_i) + \sum_{(i,j) \in E} \psi_{ij}(y_i, y_j) \right)$$

- $\psi_i(x_i, y_i)$ : the  $i$ -th corrupted pixel depends on the  $i$ -th original pixel
- $\psi_{ij}(y_i, y_j)$ : neighboring pixels tend to have the same value
- How did the original image  $\mathbf{y}$  look like? Solution: maximize  $p(\mathbf{y}|\mathbf{x})$ . Efficient algorithms (based on graph cuts) exist.

# Restricted Boltzmann Machine (RBM)

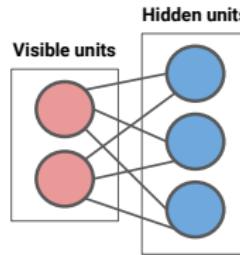
An RBM is an energy-based model with latent variables.

- Two types of variables:

- ①  $\mathbf{x} \in \{0, 1\}^n$  are visible variables (e.g., pixel values)
- ②  $\mathbf{z} \in \{0, 1\}^m$  are latent ones

- The joint distribution is

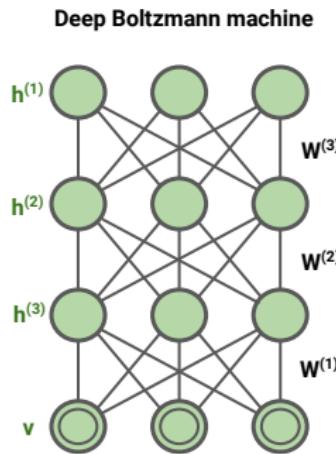
$$p_{W,b,c}(\mathbf{x}, \mathbf{z}) = \frac{1}{Z} \exp \left( \mathbf{x}^T W \mathbf{z} + b\mathbf{x} + c\mathbf{z} \right) = \frac{1}{Z} \exp \left( \sum_{i=1}^n \sum_{j=1}^m x_i z_j w_{ij} + b\mathbf{x} + c\mathbf{z} \right)$$



- Restricted because there are no visible-visible and hidden-hidden connections, i.e.,  $x_i x_j$  or  $z_i z_j$  terms in the objective

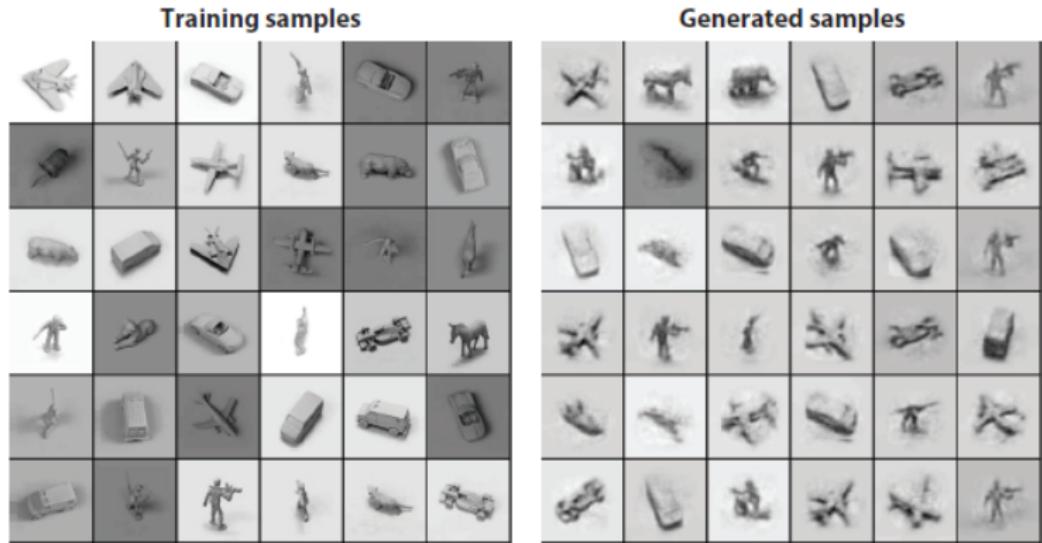
# Deep Boltzmann Machines

Stacked RBMs were one of the first deep generative models:



Bottom layer variables **v** are pixel values. Layers above (**h**) represent “higher-level” features (corners, edges, etc). Early deep neural networks for *supervised learning* had to be pre-trained like this to make them work.

# Boltzmann Machines: Samples



# Lecture Outline

## ① Energy-Based Models

- Motivation
- Definitions
- Exponential Families
- Motivating Applications

## ② Representation

- Ising Models
- Restricted Boltzmann Machines
- Deep Boltzmann Machines

## ③ Learning

- Likelihood-based learning
- Markov Chain Monte Carlo
- (Persistent) Contrastive Divergence

# Learning Energy-Based Models

In general, likelihood-based learning of energy-based models is intractable (it's hard to even compute the log likelihood!). We need approximations.

- ① Approximations to the maximum likelihood learning
  - Variational inference: optimization-based approximations
  - MCMC-based methods: sampling-based approximations (this lecture!)
- ② There also exist efficient methods that don't rely on likelihood!  
(e.g., score-based generative models, see the next lecture)

# Exponential Family Models

Energy-based models are closely related to *exponential family* models, such as:

$$p(x; \theta) = \frac{\exp(\theta^T f(x))}{Z(\theta)}.$$

- The vector  $f(x)$  is called the vector of sufficient statistics; these “featurize” or “summarize” the data: two  $x$ ’s with the same  $f(x)$  are equivalent.
- Examples: Gaussian:  $f(x) = (x, x^2)$ ,  $\theta = (\frac{\mu}{\sigma^2}, \frac{-1}{2\sigma^2})$ . Exponential, Binomial, Cauchy, Beta, Dirichlet, Ising, conditional RBMs, and many others.
- We will derive approximate learning methods for exponential family models.

# Exponential Families: Learning and Inference

Consider an exponential family model

$$p(x; \theta) = \frac{\exp(\theta^T f(x))}{Z(\theta)}.$$

Given a dataset  $D$ , we want to estimate  $\theta$  via maximum likelihood. We can show that the log-likelihood is concave and equals.

$$\frac{1}{|D|} \log p(D; \theta) = \frac{1}{|D|} \sum_{x \in D} \theta^T f(x) - \log Z(\theta).$$

The first term is linear in  $\theta$  and is easy to handle. The second term equals

$$\log Z(\theta) = \log \sum_x \exp(\theta^T f(x)).$$

It is not only hard to optimize, but it is hard to even evaluate that term.

# Exponential Families: Gradient-Based Learning

Can we train energy models using gradient-based maximum likelihood learning?

The log-likelihood equals  $\frac{1}{|D|} \log p(D; \theta) = \frac{1}{|D|} \sum_{x \in D} \theta^T f(x) - \log Z(\theta)$ .

The gradient of the linear part is easy to compute. The gradient of  $\log Z(\theta)$  is:

$$\begin{aligned}\nabla_\theta \log Z(\theta) &= \nabla_\theta \log \sum_x \exp(\theta^T f(x)) \\ &= \frac{1}{\sum_x \exp(\theta^T f(x))} \nabla_\theta \sum_x \exp(\theta^T f(x)) \\ &= \frac{1}{\sum_x \exp(\theta^T f(x))} \sum_x \exp(\theta^T f(x)) \cdot \nabla_\theta \theta^T f(x) \\ &= \frac{1}{\sum_x \exp(\theta^T f(x))} \sum_x \exp(\theta^T f(x)) \cdot f(x) \\ &= \mathbb{E}_{x \sim p}[f(x)].\end{aligned}$$

# Exponential Families: Moment Matching

Using the above derivation, we find the gradient of the log-likelihood to be:

$$\nabla_{\theta} \frac{1}{|D|} \log p(D; \theta) = \frac{1}{|D|} \sum_{x \in D} f(x) - \mathbb{E}_{x \sim p}[f(x)]$$

This is the difference between the expectations of the natural parameters under the empirical (i.e. data) and the model distribution.

# Approximate Learning Techniques in ExpFams

Computing the expectation requires sampling with respect to  $p$ . Sampling in general is intractable, and therefore so is computing the gradient.

$$\nabla_{\theta} \frac{1}{|D|} \log p(D; \theta) = \frac{1}{|D|} \sum_{x \in D} f(x) - \mathbb{E}_{x \sim p}[f(x)]$$

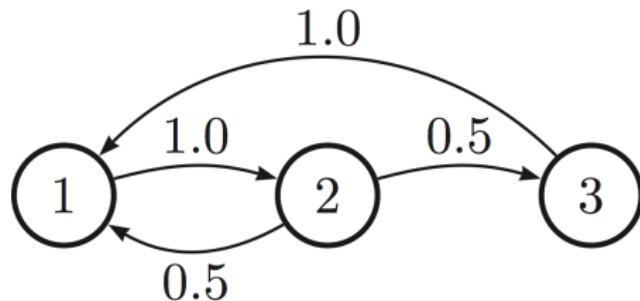
**Sampling-based approximations** learn  $p$  by approximating samples  $x \sim p$  and using the samples to approximate  $\mathbb{E}_{x \sim p}[f(x)]$  using Monte Carlo.

We will look at two approximate sampling methods:

- ① **MCMC sampling** draws  $x$  from  $p$  by simulating a Markov Chain.
- ② **(Persistent) contrastive divergence**, a variant of MCMC sampling which re-uses the same Markov Chain between iterations.

# Markov Chains: Definition

- A (discrete-time) Markov chain is a sequence of random variables  $S_0, S_1, S_2, \dots$  with  $S_i \in \{1, 2, \dots, d\}$  representing the state of a system.
- The initial state is distributed according to a probability  $P(S_0)$
- All subsequent states are generated from  $P(S_i | S_{i-1})$  that depends only on the previous random state.



- **Markov assumption:**  $P(S_i | S_{i-1})$  is the same at every step  $i$ . Transition probabilities depend only on the current state (not on how we got there).

# Markov Chains: Stationary Distribution

A Markov Chain **defines a probability distribution**.

- If the initial state  $S_0$  is drawn from a vector probabilities  $p_0$ , we may represent the probability  $p_t$  of ending up in each state after  $t$  steps as

$$p_t = T^t p_0 \quad T \in \mathbb{R}^{d \times d} \text{ and } T_{ij} = P(S_{\text{new}} = i \mid S_{\text{prev}} = j).$$

- The limit  $\pi = \lim_{t \rightarrow \infty} p_t$  (when it exists) is called a **stationary distribution** of the Markov chain. It's an eigenvector of  $T$ .

# Markov Chains: Detailed Balance

A sufficient condition for stationary distribution  $\pi$  to exist is called detailed balance:

$$\pi(x') T(x | x') = \pi(x) T(x' | x) \text{ for all } x, x'$$

It is easy to show that such a  $\pi$  must form a stationary distribution. Just sum both sides of the equation over  $x$  and simplify:

$$\pi(x') = \sum_x \pi(x) T(x' | x) \text{ for all } x \text{ means } \pi \text{ is eigenvector of } T$$

# Markov Chain Monte Carlo

The idea of Markov Chain Monte Carlo (MCMC) is to construct a Markov chain that generates samples from  $p$ .

- States are joint assignments  $x$  to the variables in the model
- The stationary distribution equals the model probability:

$$p(x; \theta) = \frac{\exp(\theta^T f(x))}{Z(\theta)}.$$

We may then use these samples for Monte Carlo integration (or in importance sampling).

# Markov Chain Monte Carlo Algorithms

An MCMC algorithm defines a transition operator  $T$  specifying a Markov chain, an initial variable assignment  $x_0$  and performs the following steps.

- ① Run the Markov chain from  $x_0$  for  $B$  burn-in steps.
- ② Run the Markov chain for  $N$  sampling steps and collect all the states that it visits.

Assuming  $B$  is sufficiently large, the latter collection of states will form samples from  $p$ .

# Constructing MCMC Chains with Metropolis-Hastings

The MH method constructs a transition operator  $T(x' | x)$  from two components:

- ① A transition kernel  $Q(x' | x)$ , specified by the user (something simple, like  $x + \text{noise}$ )
- ② An acceptance probability for moves proposed by  $Q$ , specified by:

$$A(x' | x) = \min \left( 1, \frac{P(x')Q(x | x')}{P(x)Q(x' | x)} \right).$$

- Note that the ratio  $P(x')/P(x)$  does not require knowing  $Z(\theta)$ .
- Encourages us to move towards more likely points in the distribution (e.g., consider formula for  $A$  when  $Q$  is uniform)
- When  $Q$  suggests a move to a low-probability region, we do that a certain fraction of the time.

At each step of the Markov chain, we choose a new point  $x'$  according to  $Q$ . Then, we either accept this proposed change (with probability  $\alpha$ ), or with probability  $1 - \alpha$  we remain at our current state.

# Proof of Metropolis-Hastings Method

Given any  $Q$  the MH algorithm will ensure that  $P$  will be a stationary distribution of the resulting Markov Chain. More precisely,  $P$  will satisfy the detailed balance condition with respect to the MH Markov chain.

Suppose first that  $A(x' | x) < 1$ ; then we can write:

$$A(x' | x) = \frac{P(x')Q(x | x')}{P(x)Q(x' | x)}$$

$$P(x')Q(x | x')A(x | x') = P(x)Q(x' | x)A(x' | x)$$

$$P(x')T(x | x') = P(x)T(x' | x),$$

which is simply the detailed balance condition.  $T(x | x')$  is full transition operator of MH obtained by applying both  $Q$  and  $A$ .

If  $A(x' | x) = 1$  and the above doesn't hold, then  $\frac{P(x')Q(x|x')}{P(x)Q(x'|x)} > 1$  and

$\frac{P(x)Q(x'|x)}{P(x')Q(x|x')} < 1$  and thus  $A(x | x') < 1$  and same argument holds.

# Gibbs Sampling

A special case of the Metropolis-Hastings methods is Gibbs sampling. We iterate through the variables one at a time; at each time step  $t$ , we:

- ① Sample  $x'_i \sim p(x_i | x_{-i}^t)$
- ② Set  $x^{t+1} = (x_1^t, \dots, x'_i, \dots, x_n^t)$ .
  - This is often easy, since we only need to condition  $x_i$  on small set of variables  $x_i$  directly depends on (its “Markov blanket”).

Gibbs sampling can be seen as a special case of MH with proposal  $Q(x'_i, x_{-i} | x_i, x_{-i}) = P(x'_i | x_{-i})$ . It is easy check that the acceptance probability simplifies to one.

# (Persistent) Contrastive Divergence

(Persistent) Contrastive Divergence maximizes the log-likelihood using gradient descent and uses MCMC to take samples to compute gradients.

Starting at some  $\theta_0$ , for  $t = 1, 2, \dots, T$ :

- ① Run MCMC chain to compute samples from  $p(x; \theta_t)$
- ② Use MCMC samples to compute gradient  $\nabla \log p(\text{data}; \theta_t)$ .
- ③ Take a gradient step  $\theta_{t+1} = \theta_t + \alpha \cdot \nabla \log p(\text{data}; \theta_t)$ .

Persistent CD does not re-start MCMC chain in Step 1.

- Since  $\theta_t \approx \theta_{t+1}$ , old chain for  $p(x; \theta_t)$  produces good samples for  $p(x; \theta_{t+1})$ .
- We initialize new chain at old samples and run it for a few steps with  $\theta_{t+1}$ .

# Energy-Based Models: Pros and Cons

$$p_{\theta}(\mathbf{x}) = \frac{1}{\int \exp(f_{\theta}(\mathbf{x}))} \exp(f_{\theta}(\mathbf{x})) = \frac{1}{Z(\theta)} \exp(f_{\theta}(\mathbf{x}))$$

Pros:

- ① Can plug in pretty much any function  $f_{\theta}(\mathbf{x})$  you want
- ② Can be combined with other model families
- ③ Generalize a lot of models into unifying framework

Cons:

- ① Sampling can in general be intractable
- ② Evaluating likelihood (learning) is hard in general
- ③ Latent variable inference is also challenges

However, there are efficient algorithms for these models (next lecture)!