Deep Generative Models

Chapter 3: Generation by Explicit Distribution Learning

Ali Bereyhi

ali.bereyhi@utoronto.ca

Department of Electrical and Computer Engineering

University of Toronto

Summer 2025

Distribution Model: *Requirements*

We need to *a model that computes distribution at any sample*

- \+ *Can't you just use a Softmax?!*
- – Well! Not if I think about the whole x
  - ↳ *Recall how large the data space gets with large d*
  - ↳ *What if we have a continuous data space?*

Properties of Distributions

A distribution model $P_{\mathbf{w}}(x)$ should

1. *be non-negative at any point, i.e., $P_{\mathbf{w}}(x) \geqslant 0$*
2. *add up to one, i.e.,*

$$\sum_{x \in \mathbb{X}} P_{\mathbf{w}}(x) = 1 \qquad \int_{\mathbb{X}} P_{\mathbf{w}}(x)\,\mathrm{d}x = 1$$

Boltzmann Distribution

Say we have a model $f_{\mathbf{w}}(x) : \mathbb{X} \mapsto \mathbb{R}$: *we can build a distribution out of it by*

1. *computing an exponential function of it: for any* θ

$$\exp\left\{-\frac{f_{\mathbf{w}}(x)}{\theta}\right\}$$

2. *normalizing it to its marginalization*

$$P_{\mathbf{w}}(x) = \frac{1}{\mathcal{Z}(\mathbf{w}, \theta)} \exp\left\{-\frac{f_{\mathbf{w}}(x)}{\theta}\right\}$$

where $\mathcal{Z}(\mathbf{w}, \theta)$ *is defined as*

$$\mathcal{Z}(\mathbf{w}, \theta) = \sum_{x} \exp\left\{-\frac{f_{\mathbf{w}}(x)}{\theta}\right\}$$

Boltzmann's Recipe for *Building Distributions*

+ *Can we confirm that this is a distribution?*
– Sure! Just check the requirements

It is easy to see that both requirements are satisfied by this expression

1. *The exponential function is always non-negative*

$$P_{\mathbf{w}}(x) = \frac{1}{\mathcal{Z}(\mathbf{w}, \theta)} \exp\left\{-\frac{f_{\mathbf{w}}(x)}{\theta}\right\} > 0$$

2. *The distribution adds up to one*

$$\sum_x P_{\mathbf{w}}(x) = \sum_x \frac{1}{\mathcal{Z}(\mathbf{w}, \theta)} \exp\left\{-\frac{f_{\mathbf{w}}(x)}{\theta}\right\}$$

$$= \frac{1}{\mathcal{Z}(\mathbf{w}, \theta)} \underbrace{\left(\sum_x \exp\left\{-\frac{f_{\mathbf{w}}(x)}{\theta}\right\}\right)}_{\mathcal{Z}(\mathbf{w}, \theta)} = 1$$

Boltzmann Distribution

Boltzmann Distribution

The following distribution is called Boltzmann distribution at temperature θ

$$P_{\mathbf{w}}(x) = \frac{1}{\mathcal{Z}(\mathbf{w}, \theta)} \exp\left\{-\frac{f_{\mathbf{w}}(x)}{\theta}\right\}$$

Good to Know

Boltzmann distribution is the solution to the second law of thermodynamics: it describes the distribution of micro-state at its thermal equilibrium when

1. *The energy of the system is described by Hamiltonian $f_{\mathbf{w}}(x)$*
2. *The system is isolated from the outside world*

Boltzmann Distribution: *Components*

Despite its origins in physics: *for us*

> *Boltzmann builds distribution from an arbitrary function* $f_{\mathbf{w}}$

We focus on the case with $\theta = 1$,[1] *and refer to*

1. *the model* $f_{\mathbf{w}}(x)$ *as the energy model*
2. *the normalization term* $\mathcal{Z}(\mathbf{w})$ *as the partition function*

$$\mathcal{Z}(\mathbf{w}) = \sum_{x} \exp\{-f_{\mathbf{w}}(x)\}$$

Attention

Although the partition function does not depend on the sample point x*, it still depends on the model parameter* $\mathbf{w}$

[1] *We don't really need to keep it general, as our model has enough parameters in* $\mathbf{w}$

EBMs: *Energy-based Models*

Using Boltzmann distribution: *we can convert any learning model $f_{\mathbf{w}}$ into a model for data distribution*

such models are called energy-based models (EBMs)

Energy-based Models

An EBM describes a Boltzmann distribution whose energy model is given by a learnable model, e.g., a deep NN

+ *Is there any reason we are interested on these models?!*
– *Well! They are quite generic*

Universality of EBMs: *Gaussian Example*

Universality Argument *(informal)*

For almost any well-defined distribution Q, there exists a function f such that Q is expressed as a Boltzmann distribution with energy function f

Example: *Say $x \in \mathbb{R}$ is distributed with $\mathcal{N}(0,1)$, i.e.,*

$$Q(x) = \frac{1}{\sqrt{2\pi}} \exp\left\{-\frac{x^2}{2}\right\}$$

Setting the energy function to $f(x) = x^2/2$, it is easy to see that

$$\mathcal{Z} = \int \exp\left\{-\frac{x^2}{2}\right\} \mathrm{d}x = \sqrt{2\pi} \rightsquigarrow P(x) = \frac{1}{\sqrt{2\pi}} \exp\left\{-\frac{x^2}{2}\right\} = Q(x)$$

Universality of EBMs: *Bernoulli Example*

Example: *Say* $x \in \{0, 1\}$ *is distributed with* $\mathrm{Ber}(p)$, *i.e.*,

$$Q(x) = \begin{cases} p & x = 0 \\ 1 - p & x = 1 \end{cases}$$

Let σ^{-1} *denote inverse of Sigmoid function, i.e.,* $\beta = \sigma^{-1}(p)$ *implies* $p = \sigma(\beta)$. *Setting energy function to* $f(x) = \sigma^{-1}(p)\, x$, *the partition function reads*

$$\mathcal{Z} = \sum_x \exp\left\{-\sigma^{-1}(p)\, x\right\} = 1 + \exp\left\{-\sigma^{-1}(p)\right\}$$

and thus, the Boltzmann distribution reads

$$P(0) = \frac{1}{1 + \exp\left\{-\sigma^{-1}(p)\right\}} = \sigma\left(\sigma^{-1}(p)\right) = p = Q(0)$$

$$\rightsquigarrow P(x) = Q(x)$$

Computational EBMs

We can use Boltzmann's formula to convert

any computational model, e.g., deep NN, to a computational EBM

the computational EBMs can be roughly divided into two groups

- *Boltzmann Machines (BMs)*
  - ↳ *which consider a simple quadratic energy function with learnable weights*
  - ↳ *BMs are the basic forms of computational EBMs*
    - ↳ *Somehow like linear models for regression and classification*
  - ↳ *They are inspired by the Ising or SK model in statistical physics*
- *Neural EBMs*
  - ↳ *which consider a deep NN as the energy function*
  - ↳ *They are essentially the nonlinear version of BMs*

Computational EBMs: *Boltzmann Machine*

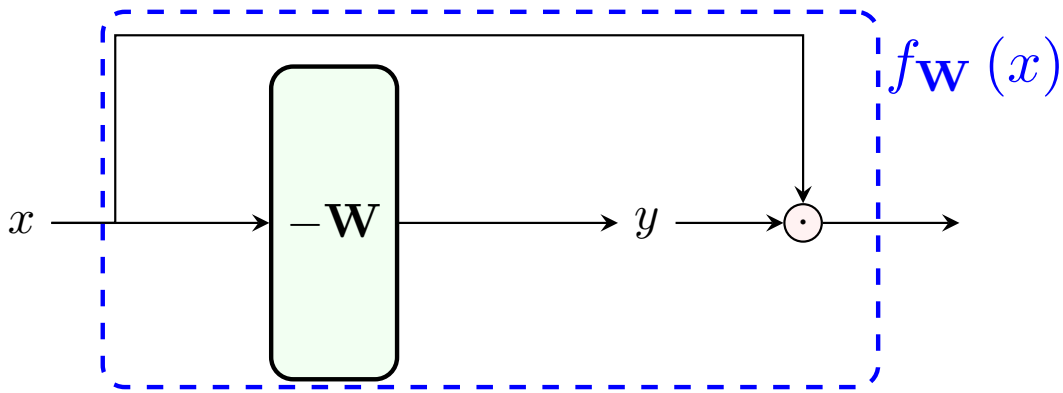
Boltzmann Machine (BM)

Consider data $x \in \{0,1\}^d$: BM models the distribution of x as a Boltzmann distribution with energy model

$$f_{\mathbf{W}}(x) = -\langle \mathbf{W}x; x \rangle$$

for a learnable $\mathbf{W} \in \mathbb{R}^{d \times d}$ which is symmetric with diagonal zero

We can think of energy model in this case as a *single linear layer*



Few Notes on *Boltzmann Machine*

This form of BM is often called

Fully Visible BM (FVBM)

as the energy model contain only the visible data elements, i.e., x_i's

+ *Can we have anything hidden?*
- *Yes! Latent representation, but we are not considering them yet*
  - ↳ *We get to know them in the next section*

BMs can be extended to other data spaces

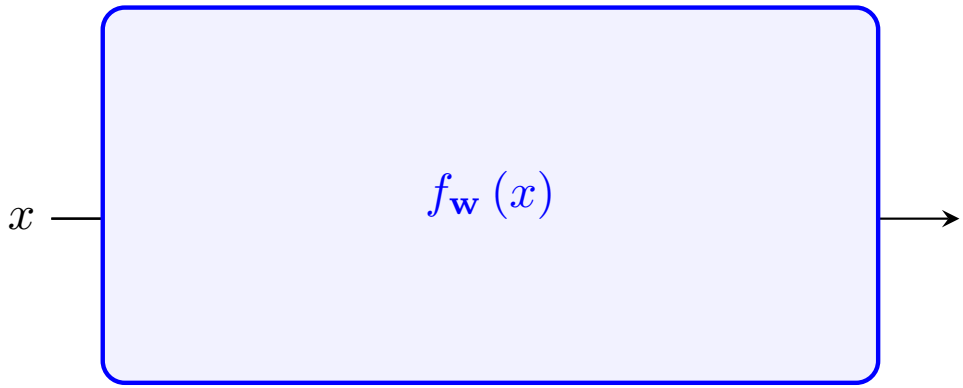
- *Continuous data space, i.e., $x_i \in \mathbb{R}$*
  - ↳ *BM describes a Gaussian distribution with learnable covariance*
  - ↳ *Some people call this model the Gaussian BM*
- *Discrete but not binary data space, e.g., $x_i \in \{0, \dots, 255\}$*
  - ↳ *Not too different from the basic BM*

Computational EBMs: *Neural EBMs*

Neural EBMs are the general form of an EBM

- *Data is not restricted to have binary entries*
- *Energy model does not be the simple quadratic one*

We can think of neural EBMs as *nonlinear BMs*, *i.e.*,



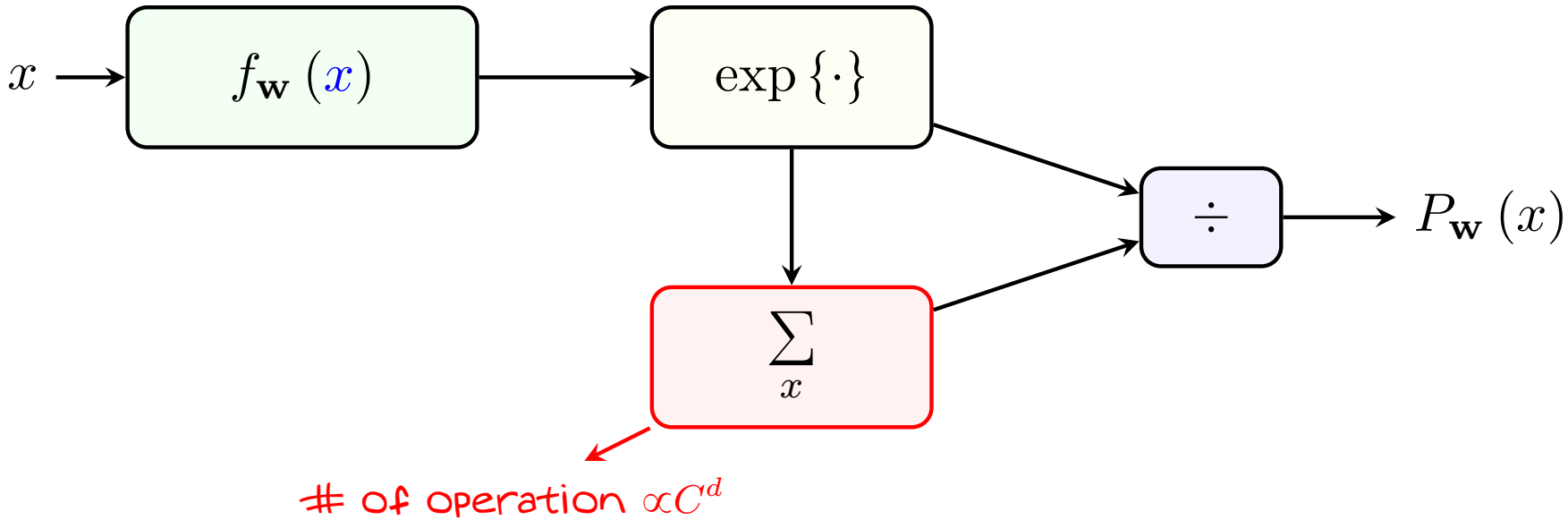
where $f_{\mathbf{w}}(x) : \mathbb{X} \mapsto \mathbb{R}$ *can be a very deep network*

Sampling EBMs

Say we trained an EBM: *we need to sample from it!*

+ *But, didn't you say sampling from a joint distribution is hard?*
– *Indeed! And, we can see it more clearly here!*

Say we are given trained energy model $\rightsquigarrow$ *we should compute distribution*



It's exponentially hard to compute EBMs from their energy models!

Training EBMs

+ *Oh! What about training EBMs then? Sounds to be the same!*
– Right! It's the same!

Say we are to train an EBM with energy model $f_{\mathbf{w}}(x)$ on dataset

$$\mathbb{D} = \left\{x^j : j = 1, \ldots, n\right\}$$

We use MLE $\rightsquigarrow$ let's look at the log-likelihood

$$\log \mathcal{L}(\mathbf{w}) = \frac{1}{n}\sum_j \log P_{\mathbf{w}}\left(x^j\right) = \frac{1}{n}\sum_j \log \frac{\exp\left\{-f_{\mathbf{w}}\left(x^j\right)\right\}}{\mathcal{Z}(\mathbf{w})}$$

$$= \frac{1}{n}\sum_j \left[-f_{\mathbf{w}}\left(x^j\right) - \log \mathcal{Z}(\mathbf{w})\right]$$

$$= -\log \mathcal{Z}(\mathbf{w}) - \frac{1}{n}\sum_j f_{\mathbf{w}}\left(x^j\right)$$

Training EBMs

We can define the empirical expectation of the energy model as

$$\hat{\mathbb{E}}_{x\sim P}\left\{f_{\mathbf{w}}\left(x\right)\right\} = \frac{1}{n}\sum_{j} f_{\mathbf{w}}\left(x^{j}\right)$$

which based on LLN goes to expectation on P as $n \to \infty$, since $x^j \sim P$

The log-likelihood can then be written as

$$\log \mathcal{L}\left(\mathbf{w}\right) = -\left[\hat{\mathbb{E}}_{x\sim P}\left\{f_{\mathbf{w}}\left(x\right)\right\} + \log \mathcal{Z}\left(\mathbf{w}\right)\right]$$

So, we can apply MLE training by minimizing negative log-likelihood

$$\hat{R}\left(\mathbf{w}\right) = \boxed{\hat{\mathbb{E}}_{x\sim P}\left\{f_{\mathbf{w}}\left(x\right)\right\}} + \boxed{\log \mathcal{Z}\left(\mathbf{w}\right)}$$

computed on dataset

# of operation $\propto C^d$

EBMs: *Complexity Challenge*

Complexity of EBM: *Training and Generation*

It is computationally complex to sample and train EBMs, since both

- *evaluation of $P_{\mathbf{w}}$ for sampling, and*
- *computation of MLE objective $\hat{R}(\mathbf{w})$ for training*

need computation of partition function which is exponentially hard

+ *We saw before that sampling is expoentially hard! So, this means that we are facing two exponentially hard tasks in EBMs! Right?!*
- Well! Indeed, these two hard tasks are the same thing
  ↳ *If we could sample $P_{\mathbf{w}}$, we could compute MLE objective as well!*
+ *How does it come?*
- Let's take a look at training again!

Training EBMs by *Sampling Them*

Assumption: *Genie-aided Sampling*

Assume that we have access to a genie which can sample $P_{\mathbf{w}}(x)$ by only knowing its energy function $f_{\mathbf{w}}(x)$

For training: *we start with some $\mathbf{w}$ and iteratively update using $\nabla \hat{R}(\mathbf{w})$*

↳ *We need $\nabla \hat{R}(\mathbf{w})$ to compute the gradient*

Let's compute the gradient

$$\begin{aligned}\nabla \hat{R}(\mathbf{w}) &= \nabla \hat{\mathbb{E}}_{x\sim P}\{f_{\mathbf{w}}(x)\} + \nabla \log \mathcal{Z}(\mathbf{w}) \\ &= \hat{\mathbb{E}}_{x\sim P}\{\nabla f_{\mathbf{w}}(x)\} + \nabla \log \mathcal{Z}(\mathbf{w})\end{aligned}$$

Backpropagation on energy model

?!

So, the main challenge is to compute $\nabla \log \mathcal{Z}(\mathbf{w})$!

Training EBMs by *Sampling Them*

Let's focus on the challenging gradient

$$\nabla \log \mathcal{Z}(\mathbf{w}) = \frac{\nabla \mathcal{Z}(\mathbf{w})}{\mathcal{Z}(\mathbf{w})}$$

From the definition of partition function, we have

$$\begin{aligned}\nabla \mathcal{Z}(\mathbf{w}) &= \sum_x \nabla \exp\{-f_{\mathbf{w}}(x)\} \\ &= -\sum_x \nabla f_{\mathbf{w}}(x) \exp\{-f_{\mathbf{w}}(x)\}\end{aligned}$$

So, we could say

$$\nabla \log \mathcal{Z}(\mathbf{w}) = -\sum_x \nabla f_{\mathbf{w}}(x) \boxed{\frac{\exp\{-f_{\mathbf{w}}(x)\}}{\mathcal{Z}(\mathbf{w})}}$$

$P_{\mathbf{w}}(x)$

Training EBMs by *Sampling Them*

This means that

$$\nabla \log \mathcal{Z}(\mathbf{w}) = -\sum_x \nabla f_{\mathbf{w}}(x)\, P_{\mathbf{w}}(x) = -\mathbb{E}_{x\sim P_{\mathbf{w}}}\{\nabla f_{\mathbf{w}}(x)\}$$

$$= -\nabla \mathbb{E}_{x\sim P_{\mathbf{w}}}\{f_{\mathbf{w}}(x)\}$$

Since the genie can give us samples from $P_{\mathbf{w}}$, we can estimate this expectation using samples drawn from the EBM, i.e.,

$$\mathbb{E}_{x\sim P_{\mathbf{w}}}\{f_{\mathbf{w}}(x)\} \approx \hat{\mathbb{E}}_{x\sim P_{\mathbf{w}}}\{f_{\mathbf{w}}(x)\}$$

and hence estimate $\nabla \log \mathcal{Z}(\mathbf{w})$ as

$$\nabla \log \mathcal{Z}(\mathbf{w}) = -\nabla \hat{\mathbb{E}}_{x\sim P_{\mathbf{w}}}\{f_{\mathbf{w}}(x)\}$$

Training Metric in EBMs

So, the gradient of the MLE objective can be estimated as

$$\nabla \hat{R}(\mathbf{w}) = \nabla \hat{\mathbb{E}}_{x \sim P}\{f_{\mathbf{w}}(x)\} - \nabla \hat{\mathbb{E}}_{x \sim P_{\mathbf{w}}}\{f_{\mathbf{w}}(x)\}$$

MLE Learning for EBMs

We can perform MLE learning using the gradients of the divergence metric

$$\hat{D}(\mathbf{w}) = \hat{\mathbb{E}}_{x \sim P}\{f_{\mathbf{w}}(x)\} - \hat{\mathbb{E}}_{x \sim P_{\mathbf{w}}}\{f_{\mathbf{w}}(x)\}$$

sampling from data distribution

sampling from model

This is an interesting expression: in MLE we try to simultaneously

- *reduce the energy at data samples, i.e.,* $f_{\mathbf{w}}(x^j)$ *for* $x^j \sim P$
- *increase the energy at model samples, i.e.,* $f_{\mathbf{w}}(\tilde{x}^j)$ *for* $\tilde{x}^j \sim P_{\mathbf{w}}$

Sampling and Training EBMs: *Summary*

To train and sample with EBMs, we only need a sampling genie

Sampling Genie

It can sample $P_{\mathbf{w}}(x)$ only knowing its energy function $f_{\mathbf{w}}(x)$

`Train_EBM(`$\mathbb{D}$`:dataset):`

Initiate the energy model $f_{\mathbf{w}}$ with some $\mathbf{w}$
for *multiple epochs* **do**
  Sample a batch of data samples $\{x^j : j = 1, \ldots, n\}$ from $\mathbb{D}$
  Sample a batch of model samples $\{\tilde{x}^j : j = 1, \ldots, n\} \leftarrow$ `SamplingGenie(`$f_{\mathbf{w}}$`)`
  for $j = 1, \ldots, n$ **do**
    Compute $\nabla f_{\mathbf{w}}(x^j)$ and $\nabla f_{\mathbf{w}}(\tilde{x}^j)$ by backpropagation over energy model
  end for
  Update $\mathbf{w}$ using `Opt_avg` $\{\nabla f_{\mathbf{w}}(x^j) - \nabla f_{\mathbf{w}}(\tilde{x}^j)\}$
end for
return *trained energy model $f_{\mathbf{w}}$*

Sampling and Training EBMs: *Summary*

To train and sample with EBMs, we only need a sampling genie

Sampling Genie

It can sample $P_{\mathbf{w}}(x)$ only knowing its energy function $f_{\mathbf{w}}(x)$

`Sample_EBM():`

1: *Generate a sample as* $\tilde{x} \leftarrow$ `SamplingGenie(`$f_{\mathbf{w}}$`)`
2: **return** $\tilde{x}$

Key Challenge: *Sampling EBMs*

We need to figure out a way to sample EBMs

+ *But, you said in the first section that this is exponentially hard!*
- Well! We talked about direct sampling
  - ↳ *In direct sampling, we generate an exact sample in one shot*

Since direct sampling is not tractable

we need to come up with an approximate way of sampling

A well-established way is to use Markov Chain Monte Carlo (MCMC) algorithms

Sampling via *Markov Chain Monte Carlo*

Markov Chain Monte Carlo (MCMC)

In MCMC sampling, a Markov chain

$$x^{(0)} \to x^{(1)} \to \cdots \to x^{(t)} \to \cdots$$

is defined which starting from an easy-to-sample $x^{(0)} \sim P^0$, it progresses in time such that the chain converges in distribution to the target $P_{\mathbf{w}}$

Example: *Say $x^{(0)} \sim P^0$ for an arbitrary P^0, we can define MCMC*

$$x^{(t)} = \sqrt{1-\varepsilon}x^{(t-1)} + \sqrt{\varepsilon}z_t$$

for some small ε and z_t sampled i.i.d. from $\mathcal{N}(0,1)$ at each t

↳ *We can show that as $t \to \infty$, sample $x^{(t)}$ converges to $\tilde{x} \sim \mathcal{N}(0,1)$*

MCMC: *Zero and First Order Methods*

MCMC methods can be roughly classified as

- *Zero-order MCMC methods, which use target distribution $P_{\mathbf{w}}(x)$ directly to build the Markov chain*
  - ↳ *Simplest MCMCs in terms in terms of computation*
  - ↳ *Take often long time to mix, i.e., start to converge at large t*
  - ↳ *Most famous over is Gibbs sampling which we learn next*
- *First-order methods, which use $\nabla_x \log P_{\mathbf{w}}(x)$ to build Markov chain*
  - ↳ *More computation is needed*
  - ↳ *Mix faster than zero-order methods, i.e., start to converge at smaller t*
  - ↳ *Most famous over is Langevin dynamics which we learn*
- *Second-order methods that use Hessian*
  - ↳ *Computationally expensive*
  - ↳ *We do not consider them here*

MCMC_0: *Gibbs Sampling*

Gibbs sampling is a famous zero-order MCMC algorithm

`Gibbs_Sampling():`

1: *Initiate sample* $x^{(0)}$
2: **for** $t = 1, \ldots, T$ **do**
3: **for** $i = 1, \ldots, d$ **do**
4: Sample $x_i^{(t)} \sim P_{\mathbf{w}}\left(x_i | x_{<i}^{(t)}, x_{>i}^{(t-1)}\right)$
5: **end for**
6: **end for**
7: **return** $x^{(T)}$ *for* T *larger than burn-in period*

+ *What is burn-in period?*
– This is time needed for $x^{(t)}$ to start converging $P_{\mathbf{w}}$ in some sense
  ↳ *The amount of time needed to forget bad initialization*
  ↳ T *is often set by heuristics, e.g.,* $T = 1000$

Note that at each t *we loop over dimensions* $\rightsquigarrow$ d *samplings*

MCMC_0: *Gibbs Sampling*

Gibbs Sampling: *Theoretical Guarantee*

Under some mild conditions, sample $x^{(T)}$ *converges to sample* $\tilde{x} \sim P_{\mathbf{w}}(x)$ *in distribution as* $T \to \infty$

+ *Sounds nice! But, we still need to work with target distribution* $P_{\mathbf{w}}$*!*
– Yes! But we need to only work with 1D conditionals
  ↳ *Similar to what we saw in AR modeling, they are much easier to sample*
  ↳ *With EBMs* $\rightsquigarrow$ *We don't need to compute partition function*

Good to Know

Gibbs sampling is efficient for simple EBMs like basic BM

Boltzmann Machine: *Gibbs Sampling*

Let's consider the example of Boltzmann Machine: *in this model, the data space is* $\mathbb{X} \in \{0,1\}^d$ *and the energy model is set to*

$$f_{\mathbf{W}}(x) = -\langle \mathbf{W}x; x \rangle = -\sum_i \sum_{\ell \neq i} W_{i,\ell} x_i x_\ell$$

where $W_{i,\ell}$ *is the entry at row* i *and column* ℓ *of* $\mathbf{W}$

- $\mathbf{W}$ *is symmetric, i.e.,* $W_{i,\ell} = W_{\ell,i}$
- $\mathbf{W}$ *has zero diagonal, i.e.,* $W_{i,i} = 0$
- $x_i \in \{0,1\}$ *for any* $i = 1, \ldots, d$

The corresponding EBM is then written as

$$P_{\mathbf{W}}(x) = \frac{\exp\{-f_{\mathbf{W}}(x)\}}{\mathcal{Z}(\mathbf{W})}$$

Boltzmann Machine: *Gibbs Sampling*

For Gibbs sampling, we need conditional distributions: *we use Bayes rule*

$$P_{\mathbf{W}}\left(x_i | x_{<i}, x_{>i}\right) = \frac{P_{\mathbf{W}}\left(x_{<i}, x_i, x_{>i}\right)}{P_{\mathbf{W}}\left(x_{<i}, x_{>i}\right)} = \frac{P_{\mathbf{W}}\left(x\right)}{P_{\mathbf{W}}\left(x_{<i}, x_{>i}\right)}$$

By marginalization, we can write

$$\begin{aligned} P_{\mathbf{W}}\left(x_{<i}, x_{>i}\right) &= \sum_{x_i} P_{\mathbf{W}}\left(x_{<i}, x_i, x_{>i}\right) \\ &= P_{\mathbf{W}}\left(x_{<i}, 0, x_{>i}\right) + P_{\mathbf{W}}\left(x_{<i}, 1, x_{>i}\right) \end{aligned}$$

So, we have

$$P_{\mathbf{W}}\left(x_i | x_{<i}, x_{>i}\right) = \frac{P_{\mathbf{W}}\left(x\right)}{P_{\mathbf{W}}\left(x_{<i}, 0, x_{>i}\right) + P_{\mathbf{W}}\left(x_{<i}, 1, x_{>i}\right)}$$

Boltzmann Machine: *Gibbs Sampling*

We can further simplify as

$$P_{\mathbf{W}}\left(x_i \middle| x_{<i}, x_{>i}\right) = \frac{\frac{\exp\left\{-f_{\mathbf{W}}\left(x\right)\right\}}{\mathcal{Z}\left(\mathbf{W}\right)}}{\frac{\exp\left\{-f_{\mathbf{W}}\left(x_{<i}, 0, x_{>i}\right)\right\}}{\mathcal{Z}\left(\mathbf{W}\right)} + \frac{\exp\left\{-f_{\mathbf{W}}\left(x_{<i}, 1, x_{>i}\right)\right\}}{\mathcal{Z}\left(\mathbf{W}\right)}}$$

$$= \frac{\exp\left\{-f_{\mathbf{W}}\left(x\right)\right\}}{\exp\left\{-f_{\mathbf{W}}\left(x_{<i}, 0, x_{>i}\right)\right\} + \exp\left\{-f_{\mathbf{W}}\left(x_{<i}, 1, x_{>i}\right)\right\}}$$

which does not depend on the partition function!

Boltzmann Machine: *Gibbs Sampling*

This is indeed a simple Bernoulli distribution

$$
\begin{aligned}
P_{\mathbf{W}}\left(0 \middle| x_{<i}, x_{>i}\right) &= \frac{\exp\left\{-f_{\mathbf{W}}\left(x_{<i}, 0, x_{>i}\right)\right\}}{\exp\left\{-f_{\mathbf{W}}\left(x_{<i}, 0, x_{>i}\right)\right\} + \exp\left\{-f_{\mathbf{W}}\left(x_{<i}, 1, x_{>i}\right)\right\}} \\
&= \frac{1}{1 + \exp\left\{f_{\mathbf{W}}\left(x_{<i}, 0, x_{>i}\right) - f_{\mathbf{W}}\left(x_{<i}, 1, x_{>i}\right)\right\}} \\
&= \sigma\left(f_{\mathbf{W}}\left(x_{<i}, 1, x_{>i}\right) - f_{\mathbf{W}}\left(x_{<i}, 0, x_{>i}\right)\right) \\
&= \sigma\left(\Delta f_{\mathbf{W}}\left(x_{<i}, x_{>i}\right)\right)
\end{aligned}
$$

So we could write

$$
P_{\mathbf{W}}\left(x_i \middle| x_{<i}, x_{>i}\right) = \begin{cases} \sigma\left(\Delta f_{\mathbf{W}}\left(x_{<i}, x_{>i}\right)\right) & x_i = 0 \\ 1 - \sigma\left(\Delta f_{\mathbf{W}}\left(x_{<i}, x_{>i}\right)\right) & x_i = 1 \end{cases}
$$

Sampling BMs by *Gibbs Sampling*

In Boltzmann machine: *the energy difference is rather easy*

$$\Delta f_{\mathbf{W}}\left(x_{<i}, x_{>i}\right) = -\sum_{\ell \neq i} W_{i,\ell} x_\ell = -\left\langle \mathbf{w}_{i,<i}; x_{<i} \right\rangle - \left\langle \mathbf{w}_{i,>i}; x_{>i} \right\rangle$$

which is a linear transform on all x_ℓ *except* x_i

`Gibbs_Sampling`$(T, \mathbf{W})$:

1: *Initiate sample* $x^{(0)}$
2: **for** $t = 1, \ldots, T$ **do**
3: **for** $i = 1, \ldots, d$ **do**
4: Sample $x_i^{(t)} \sim \mathrm{Ber}\left(-\left\langle \mathbf{w}_{i,<i}; x_{<i}^{(t)} \right\rangle - \left\langle \mathbf{w}_{i,>i}; x_{>i}^{(t-1)} \right\rangle\right)$
5: **end for**
6: **end for**
7: **return** $x^{(T)}$ *for* T *larger than burn-in period*

Training BMs by *Gibbs Sampling*

`Train_BM(`$\mathbb{D}$`:dataset):`

Initiate $\mathbf{W}$
for *multiple epochs* **do**
  Sample $\left\{x^j : j = 1, \ldots, n\right\}$ *from* $\mathbb{D}$
  Sample $\left\{\tilde{x}^j : j = 1, \ldots, n\right\} \leftarrow$ `Gibbs_Sampling(`$T, \mathbf{W}$`)`
  for $j = 1, \ldots, n$ **do**
    Compute $-x^j x^{j\mathsf{T}}$ *and* $-\tilde{x}^j \tilde{x}^{j\mathsf{T}}$ $\quad \#\nabla f_{\mathbf{W}}(x) = -xx^{\mathsf{T}}$
  end for
  Update as $\mathbf{W} \leftarrow \mathbf{W} - \eta$`Opt_avg` $\left\{\tilde{x}^j \tilde{x}^{j\mathsf{T}} - x^j x^{j\mathsf{T}}\right\}$
end for
return *trained* $\mathbf{W}$

`Sample_BM(`$T, \mathbf{W}$`):`

Generate a sample as $\tilde{x} \leftarrow$ `Gibbs_Sampling(`$T, \mathbf{W}$`)`
return $\tilde{x}$

MCMC_1: *Langevin Dynamics*

For neural EBMs, it's better to deploy first order MCMC algorithms

zero-order MCMC algorithms often take to long to converge

A well-known first-order algorithm is Langevin dynamics

`Langevin_Sampling():`

1: *Initiate sample* $x^{(0)}$ *and choose a converging series* $\{\epsilon_t\}$
2: **for** $t = 1, \ldots, T$ **do**
3: Sample $\eta_t \sim \mathcal{N}(0, 1)$
4: Update $x^{(t)} \leftarrow x^{(t-1)} + \frac{\epsilon_t}{2} \nabla_x \log P_{\mathbf{w}} \left(x^{(t-1)} \right) + \sqrt{\epsilon_t} \eta_t$
5: **end for**
6: **return** $x^{(T)}$ *for* T *larger than burn-in period*

Langevin Dynamics with *EBMs*

In EBMs, Langevin dynamics find an favorable form: *we have*

$$P_{\mathbf{w}}(x) = \frac{\exp\{-f_{\mathbf{w}}(x)\}}{\mathcal{Z}(\mathbf{w})}$$

Thus, we can write

$$\log P_{\mathbf{w}}(x) = -f_{\mathbf{w}}(x) - \log \mathcal{Z}(\mathbf{w})$$

Noting that $\mathcal{Z}(\mathbf{w})$ *does not depend on* x*, we can write*

$$\nabla_x \log P_{\mathbf{w}}(x) = -\nabla_x f_{\mathbf{w}}(x)$$

MCMC$_1$: *Langevin Dynamics in EBMs*

`Langevin_Sampling():`

1: *Initiate sample* $x^{(0)}$ *and choose a converging series* $\{\epsilon_t\}$
2: **for** $t = 1, \ldots, T$ **do**
3: Sample $\eta_t \sim \mathcal{N}(0, 1)$
4: Update $x^{(t)} \leftarrow x^{(t-1)} - \frac{\epsilon_t}{2} \nabla_x f_{\mathbf{w}} \left(x^{(t-1)} \right) + \sqrt{\epsilon_t} \eta_t$
5: **end for**
6: **return** $x^{(T)}$ *for* T *larger than burn-in period*

This algorithm can sample from $P_{\mathbf{w}}$ *by knowing only energy function*

↳ *This is indeed the sampling genie*

↳ *The algorithm however still needs large* T *to converge*

Conservative Sampling via *Langevin Dynamics*

Requiring large T to converge makes it impractical for training

↳ *A simple remedy is to use conservative sampling for training*

Conservative Sampling

In conservative sampling, we start from data sample as $x^{(0)}$ and apply a few steps of Langevin dynamics

$\texttt{CS}(k, x^{(0)})$:

1: *Choose a converging* $\{\epsilon_t\}$ `#sample` $x^{(0)} \sim P(x)$ `is given`
2: **for** $t = 1, \ldots, k$ **do**
3: Sample $\eta_t \sim \mathcal{N}(0, 1)$
4: Update $x^{(t)} \leftarrow x^{(t-1)} - \frac{\epsilon_t}{2} \nabla_x f_{\mathbf{w}} \left(x^{(t-1)} \right) + \sqrt{\epsilon_t} \eta_t$
5: **end for**
6: **return** $x^{(k)}$

Train EBMs by *Conservative Divergence*

With conservative sampling we can build efficient training loop for neural EBMs

`CDTrain_EBM(`k, $\mathbb{D}$`:dataset):`

1: *Initiate* $\mathbf{w}$
2: **for** *multiple epochs* **do**
3: *Sample a batch of data samples* $\left\{x^j : j = 1, \ldots, n\right\}$ *from* $\mathbb{D}$
4: **for** $j = 1, \ldots, n$ **do**
5: *Sample model as* $\tilde{x}^j \leftarrow$ `CS(`k, x^j`)`
6: *Compute conservative divergence* $\mathrm{CD}_k^j = \nabla f_{\mathbf{w}}\left(x^j\right) - \nabla f_{\mathbf{w}}\left(\tilde{x}^j\right)$
7: **end for**
8: *Update* $\mathbf{w}$ *using* `Opt_avg` $\left\{\mathrm{CD}_k^j\right\}$
9: **end for**
10: **return** *trained* $\mathbf{W}$

Sample EBMs after *Training Conservative Divergence*

Attention

We cannot use conservative sampling for generation, since we do not have access to data samples anymore!

`Sampling_EBM(`$f_{\mathbf{w}}$`):`

1: *Initiate sample* $x^{(0)} \sim \mathcal{N}(0, 1)$ *and choose converging* $\{\epsilon_t\}$
2: **for** $t = 1, \ldots, T$ **do**
3: Sample $\eta_t \sim \mathcal{N}(0, 1)$
4: Update $x^{(t)} \leftarrow x^{(t-1)} - \frac{\epsilon_t}{2} \nabla_x f_{\mathbf{w}} \left(x^{(t-1)} \right) + \sqrt{\epsilon_t} \eta_t$
5: **end for**
6: **return** $x^{(T)}$ *for* T *larger than burn-in period*

Wrap Up

In summary, *we could say*

- *EBMs convert any model to a distribution using Boltzmann formula*
- *They are not easy to sample* $\rightsquigarrow$ *this impacts both sampling and training*
  - ↳ *We cannot sample them directly, as it is exponentially complex*
  - ↳ *We can use MCMC approaches to sample them*
    - ↳ *These approaches are still slow, as they need time to converge*
    - ↳ *We can use tricks like conservative sampling to improve speed in training*

Good to Know

EBMs can also be extended to include latent variables as well

- *We get to know latent space in the next section*
- *Since EBMs are not very practical, we leave latent-space EBMs*
  - ↳ *If interested, take a look at restricted Boltzmann machine*

Grandfather of *Diffusion Models*

+ *If not practical, why did we spend time learning them?!*
- What we learned here serves us well in understanding *diffusion models*

Indeed, EBMs are the grandfathers of diffusion models: Langevin dynamics was inspired by Brownian motion which is

fundamental model for describing the diffusion process

Looking at the process of sampling with Langevin dynamics, Hyvärinen came up with an interesting conclusion in 2005

> *We would be fine if we try to learn so-called score function* $\nabla_x \log P(x)$

Score Matching: *A Short Overview*

Hyvärinen observed that *in Langevin dynamics, we update*

$$x^{(t)} \leftarrow x^{(t-1)} + \frac{\epsilon_t}{2} \nabla_x \log P_{\mathbf{w}} \left(x^{(t-1)} \right) + \sqrt{\epsilon_t} \eta_t$$

which means that we mainly make use of the score function

$$s_{\mathbf{w}} (x) = \nabla_x \log P_{\mathbf{w}} (x)$$

Hyvärinen showed that it is feasible to match this learnable function to the true score function, i.e., to find a way to feasibly minimize

$$\mathcal{J} (\mathbf{w}) = \mathbb{E}_{x \sim P} \left\{ \left| s_{\mathbf{w}} (x) - s (x) \right|^2 \right\}$$

This is called score-matching which is efficient alternative way to train EBMs

↳ *We learn them in Chapter 5, as diffusion models also rely on learning score!*