Energy Based Models

Volodymyr Kuleshov

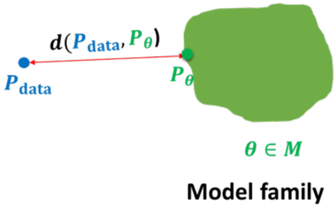
Cornell Tech

Lecture 11

Announcements

- Assignment 2 is due at midnight today!
    - If submitting late, please mark it as such.
- Submit Assignment 2 via Gradescope. The code is M45WYY.
    - Submit your pdf assignment as a photo/pdf
    - Submit your programming assignment as a zip file
- Sent out emails to resolve issues with presentation slots.

Summary



Story so far

- Representation: Latent variable vs. fully observed
- Objective function and optimization algorithm: Many divergences and distances optimized via likelihood-free (two sample test) or likelihood based methods

Plan for today: Normalized vs. Energy based models

Lecture Outline

1. Energy-Based Models
    - Motivation
    - Definitions
    - Exponential Families
2. Representation
    - Motivating Applications
    - Ising Models
    - Product of Experts
    - Restricted Boltzmann Machines
    - Deep Boltzmann Machines
3. Learning
    - Likelihood-based learning
    - Markov Chain Monte Carlo
    - (Persistent) Contrastive Divergence

Parameterizing probability distributions

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

1. non-negative: $p(x) \geq 0$
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 $p_\theta(\mathbf{x})$ is not hard. For example:

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

Problem: $g_\theta(\mathbf{x}) \geq 0$ is easy, but $g_\theta(\mathbf{x})$ might not sum-to-one.
$\sum_{\mathbf{x}} g_\theta(\mathbf{x}) = Z(\theta) \neq 1$ in general, so $g_\theta(\mathbf{x})$ is not a valid probability mass function or density

Parameterizing probability distributions

Problem: $g_\theta(\mathbf{x}) \geq 0$ is easy, but $g_\theta(\mathbf{x})$ might not be normalized
Solution:

$$p_\theta(\mathbf{x}) = \frac{1}{Volume(g_\theta)} g_\theta(\mathbf{x}) = \frac{1}{\int g_\theta(\mathbf{x}) d\mathbf{x}} g_\theta(\mathbf{x})$$

Then by definition, $\int p_\theta(\mathbf{x}) d\mathbf{x} = 1$. Typically, choose $g_\theta(\mathbf{x})$ so that we know the volume *analytically* as a function of θ. For example,

1. $g_{(\mu,\sigma)}(x) = e^{-\frac{(x-\mu)^2}{2\sigma^2}}$. Volume is: $\int e^{-\frac{x-\mu}{2\sigma^2}} dx = \sqrt{2\pi\sigma^2} \rightarrow$ **Gaussian**
2. $g_\lambda(x) = e^{-\lambda x}$. Volume is: $\int_0^{+\infty} e^{-\lambda x} dx = \frac{1}{\lambda}$. $\rightarrow$ **Exponential**
3. Etc.

We can only choose functional forms $g_\theta(\mathbf{x})$ that we can integrate *analytically*. This is very restrictive, but as we have seen, they are very useful as building blocks for more complex models (e.g., conditionals in autoregressive models)

Parameterizing probability distributions

Problem: $g_\theta(\mathbf{x}) \geq 0$ is easy, but $g_\theta(\mathbf{x})$ might not be normalized
Solution:

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

Typically, choose $g_\theta(x)$ so that we know the volume *analytically*. More complex models can be obtained by combining these building blocks. Main strategies:

1. **Autoregressive:** Products of normalized objects $p_\theta(x)p_{\theta'(x)}(y)$:
$$\int_x \int_y p_\theta(x)p_{\theta'(x)}(y)dxdy = \int_x p_\theta(x) \underbrace{\int_y p_{\theta'(x)}(y)dy}_{=1} dx = \int_x p_\theta(x)dx = 1$$
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 and track volume change.

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

Energy based model

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

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_θ.
2. Exponential families. Many common distributions can be written in this form.
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. Intuitively, configurations $\mathbf{x}$ with low energy (high $f_\theta(\mathbf{x})$) are more likely.

Energy based model

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

Pros:

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

Cons (lots of them):

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

Curse of dimensionality: The fundamental issue is that computing $Z(\theta)$ numerically (when no analytic solution is available) scales exponentially in the number of dimensions of $\mathbf{x}$. Nevertheless, some tasks do not require knowing $Z(\theta)$

Exponential family models

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

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

Exponential families are

- Log-concave in their natural parameters θ. The partition function $Z(\theta)$ is also log-convex in θ.
- The vector $f(x)$ is called the vector of sufficient statistics; these fully describe the distribution p; e.g. if p is Gaussian, θ contains (simple reparametrizations of) the mean and the variance of p.
- Maximizing the entropy $H(p)$ under the constraint $\mathbb{E}_p[f(x)] = \alpha$ (i.e. the sufficient statistics equal some value α) is an ExpFam.

Example: Gaussian: $f(x) = (x, x^2)$, $\theta = (\frac{\mu}{\sigma^2}, \frac{-1}{2\sigma^2})$.

Lecture Outline

1. Energy-Based Models
    - Motivation
    - Definitions
    - Exponential Families
2. **Representation**
    - Motivating Applications
    - Ising Models
    - Product of Experts
    - Restricted Boltzmann Machines
    - Deep Boltzmann Machines
3. Learning
    - Likelihood-based learning
    - Markov Chain Monte Carlo
    - (Persistent) Contrastive Divergence

Applications of Energy based models

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

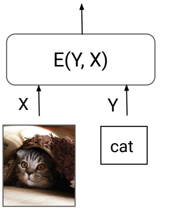
Given $\mathbf{x}$, $\mathbf{x}'$ evaluating $p_\theta(\mathbf{x})$ or $p_\theta(\mathbf{x}')$ requires $Z(\theta)$. However, their ratio

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

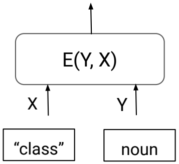
does not involve $Z(\theta)$. This means we can easily check which one is more likely. Applications:

1. anomaly detection
2. denoising

Applications of Energy based models



object recognition



sequence labeling

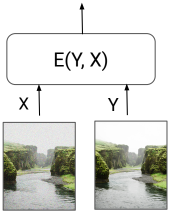
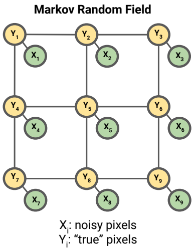


image restoration

Given a trained model, many applications require relative comparisons. Hence $Z(\theta)$ is not needed.

Example: 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 $p(\mathbf{y}|\mathbf{x})$

Example: Product of Experts

- Suppose you have trained several models $q_{\theta_1}(\mathbf{x})$, $r_{\theta_2}(\mathbf{x})$, $t_{\theta_3}(\mathbf{x})$. They can be different models (PixelCNN, Flow, etc.)
- Each one is like an *expert* that can be used to score how likely an input $\mathbf{x}$ is.
- Assuming the experts make their judgments indpendently, it is tempting to ensemble them as

$$p_{\theta_1}(\mathbf{x})q_{\theta_2}(\mathbf{x})r_{\theta_3}(\mathbf{x})$$

- To get a valid probability distribution, we need to normalize

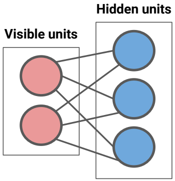
$$p_{\theta_1,\theta_2,\theta_3}(\mathbf{x}) = \frac{1}{Z(\theta_1, \theta_2, \theta_3)} q_{\theta_1}(\mathbf{x})r_{\theta_2}(\mathbf{x})t_{\theta_3}(\mathbf{x})$$

- Note: similar to an AND operation (e.g., probability is zero as long as one model gives zero probability), unlike mixture models which behave more like OR

Example: Restricted Boltzmann machine (RBM)

- RBM: 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

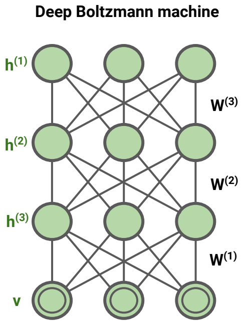
$$p_{W,b,c}(\mathbf{x},\mathbf{z}) = \frac{1}{Z}\exp\left(\mathbf{x}^T W\mathbf{z} + b\mathbf{x} + c\mathbf{z}\right) = \frac{1}{Z}\exp\left(\sum_{i=1}^{n}\sum_{j=1}^{m} x_i z_j w_{ij} + b\mathbf{x} + c\mathbf{z}\right)$$



- 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

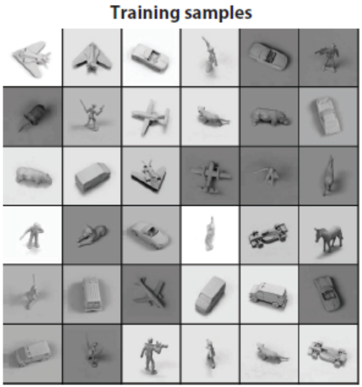
Deep Boltzmann Machines

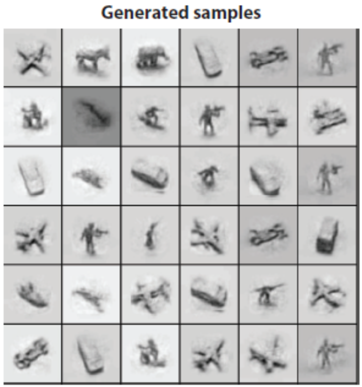
Stacked RBMs are one of the first deep generative models:



Bottom layer variables $\mathbf{v}$ are pixel values. Layers above ($\mathbf{h}$) 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
2. Representation
   - Motivating Applications
   - Ising Models
   - Product of Experts
   - Restricted Boltzmann Machines
   - Deep Boltzmann Machines
3. **Learning**
   - Likelihood-based learning
   - Markov Chain Monte Carlo
   - (Persistent) Contrastive Divergence

Energy based models: learning and inference

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

Pros:

1. can plug in pretty much any function $f_\theta(\mathbf{x})$ you want

Cons (lots of them):

1. Sampling is hard
2. Evaluating likelihood (learning) is hard
3. Feature learning is even harder

Curse of dimensionality: The fundamental issue is that computing $Z(\theta)$ numerically (when no analytic solution is available) scales exponentially in the number of dimensions of $\mathbf{x}$.
Can we still learn p? Yes! (but it will not be as fast)

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 θ via maximum likelihood. 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 θ and is easy to handle. The second term equals

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

Unlike the first term, this one does not decompose across x. It is not only hard optimize, but it is hard to even evaluate that term.

Computing the normalization constant is hard

- As an example, the RBM joint distribution is

$$p_{W,b,c}(\mathbf{x}, \mathbf{z}) = \frac{1}{Z} \exp\left(\mathbf{x}^T W \mathbf{z} + b\mathbf{x} + c\mathbf{z}\right)$$

  where

  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 normalization constant (the "volume") is

$$Z(W, b, c) = \sum_{\mathbf{x}\in\{0,1\}^n} \sum_{\mathbf{z}\in\{0,1\}^m} \exp\left(\mathbf{x}^T W \mathbf{z} + b\mathbf{x} + c\mathbf{z}\right)$$

- Note: it is a well defined function of the parameters W, b, c, but no simple closed-form. Takes time exponential in n, m to compute. This means that *evaluating* the objective function $p_{W,b,c}(\mathbf{x}, \mathbf{z})$ for likelihood based learning is hard.
- Optimizing the un-normalized probability $\exp\left(\mathbf{x}^T W \mathbf{z} + b\mathbf{x} + c\mathbf{z}\right)$ is easy (w.r.t. trainable parameters W, b, c), but *optimizing* the likelihood $p_{W,b,c}(\mathbf{x}, \mathbf{z})$ is also difficult..

Exponential families: gradient-based learning

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

Obtaining the gradient of the linear part is obviously easy. However,

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

Computing the expectation requires inference with respect to p. Inference in general is intractable, and therefore so is computing the gradient.

Exponential families: moment matching

The log-likelihood of an MRF is

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

Taking the gradient, and using our expression for the gradient of the partition function, we obtain the expression

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

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

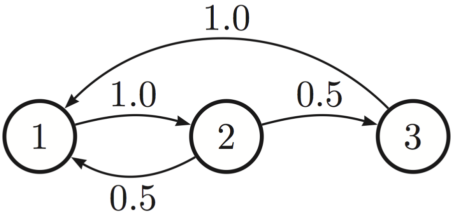
To compute gradients, we need to sample from the model. But this is hard!

We will look at two approximate methods:

1. **MCMC sampling** from the distribution at each step of gradient descent; we then approximate the gradient using Monte-Carlo.
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 $S_0, S_1, S_2, \ldots$ with $S_i \in \{1, 2, \ldots, d\}$, intuitively representing the state of a system.
- The initial state is distributed according to a probability $P(S_0)$
- All subsequent states are generated from $P(S_i \mid S_{i-1})$ that depends only on the previous random state.



Markov assumption: the probability $P(S_i \mid S_{i-1})$ is the same at every step i. The transition probabilities in the entire process depend only on the given state and not on how we got there.

Markov Chains: Stationary 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

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

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

A sufficent condition for a stationary distribution is called detailed balance:

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

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

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

Markov Chain Monte Carlo

The idea of MCMC will be to construct a Markov chain whose states will be joint assignments to the variables in the model and whose stationary distribution will equal the model probability

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

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

Constructing MCMC chains with Metropolis-Hastings

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

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

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

  - Encourages us to move towards more likely points in the distribution (imagine for example that 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 α), 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.

To see that, first observe that if $A(x' \mid x) < 1$, then $\frac{P(x)Q(x'|x)}{P(x')Q(x|x')} > 1$ and thus $A(x \mid x') = 1$. When $A(x' \mid x) < 1$, this lets us write:

$$\begin{aligned} A(x' \mid x) &= \frac{P(x')Q(x \mid x')}{P(x)Q(x' \mid x)} \\ P(x')Q(x \mid x')A(x \mid x') &= P(x)Q(x' \mid x)A(x' \mid x) \\ P(x')T(x \mid x') &= P(x)T(x' \mid x), \end{aligned}$$

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

Gibbs sampling

A widely-used 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 $x_i' \sim p(x_i \mid x_{-i}^t)$
2. Set $x^{t+1} = (x_1^t, ..., x_i', ..., x_n^t)$.
   - 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").

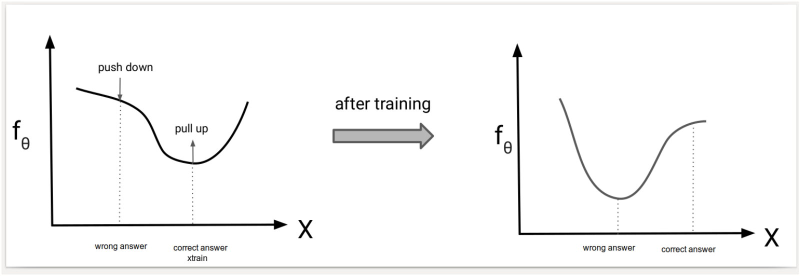
Gibbs sampling can be seen as a special case of MH with proposal $Q(x_i', x_{-i} \mid x_i, x_{-i}) = P(x_i' \mid x_{-i})$. It is easy check that the acceptance probability simplifies to one.

Sampling from Energy based models

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

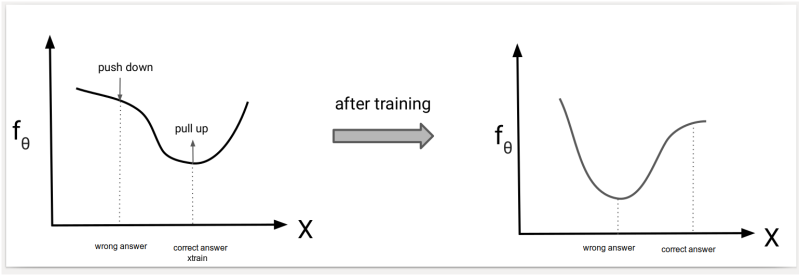
- No direct way to sample like in autoregressive or flow models. Main issue: cannot easily compute how likely each possible sample is
- However, we can easily compare two samples $\mathbf{x}, \mathbf{x}'$.
- Use iterative approach based on Metropolis-Hastings MCMC:
  1. Initialize x^0 randomly, $t = 0$
  2. Let $x' = x^t$ + noise
     1. If $f_\theta(x') > f_\theta(x^t)$, let $x^{t+1} = x'$
     2. Else let $x^{t+1} = x'$ with probability $\exp(f_\theta(x') - f_\theta(x^t))$
  3. Go to step 2
- Works in theory, but can take a very long time to converge

(Persistent) Contrastive Divergence



- Goal: maximize $\frac{f_\theta(x_{train})}{Z(\theta)}$
- **Idea**: Instead of evaluating $Z(\theta)$ exactly, use a Monte Carlo estimate.
- **Contrastive divergence algorithm**: sample $x_{sample} \sim p_\theta$ with MCMCM, take step on $\nabla_\theta \left(f_\theta(x_{train}) - f_\theta(x_{sample})\right)$. Make training data more likely than typical sample from the model. Recall comparisons are easy in energy based models!
- **Persistent** CD: reuse the Markov chain across SGD steps

Training intuition



- Goal: maximize $\frac{f_\theta(x_{train})}{Z(\theta)}$. Increase numerator, decrease denominator.
- **Intuition**: because the model is not normalized, increasing the un-normalized probability $f_\theta(x_{train})$ by changing θ does **not** guarantees that x_{train} becomes relatively more likely (compared to the rest).
- We also need to take into account the effect on other "wrong points" and try to "push them down" to *also* make $Z(\theta)$ small.

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. Can be combined with ideas from graphical models

Cons:

1. Sampling is hard
2. Evaluating likelihood (learning) is hard
3. Feature learning is even harder

Conclusion

- Energy-based models are another useful tool for modeling high-dimensional probability distributions.
- Very flexible class of models. Currently less popular because of computational issues.
- Energy based GANs: energy is represented by a discriminator. Contrastive samples (like in contrastive divergence) from a GAN-styke generator.
- Reference: LeCun et. al, *A Tutorial on Energy-Based Learning*