Deep Generative Models

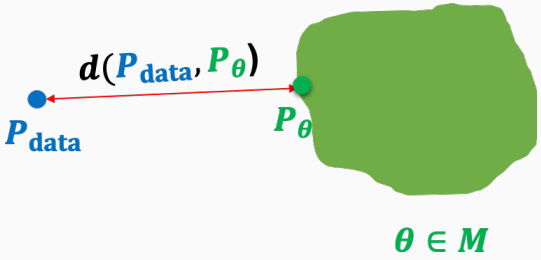
Lecture 12: Energy-Based Models

Aditya Grover

UCLA

Recap

$x^{(j)} \sim P_{\text{data}}$
$j = 1, 2, \ldots, m$

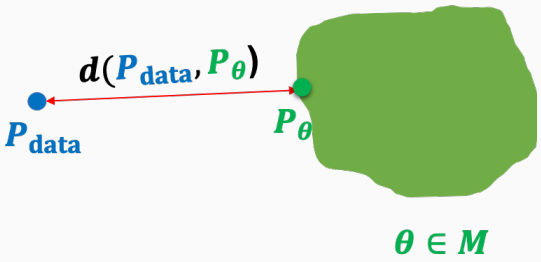


- Autoregressive models. $p_\theta(x_1, x_2, \cdots, x_n) = \prod_{i=1}^{n} p_\theta(x_i \mid x_{<i})$
- Normalizing flow models. $p_\theta(\mathbf{x}) = p(\mathbf{z}) |\det J_{f_\theta}(\mathbf{x})|$, where $\mathbf{z} = f_\theta(\mathbf{x})$
- Variational autoencoders: $p_\theta(\mathbf{x}) = \int p(\mathbf{z}) p_\theta(\mathbf{x} \mid \mathbf{z}) \mathrm{d}\mathbf{z}$

Cons: Model architectures are restricted

Recap

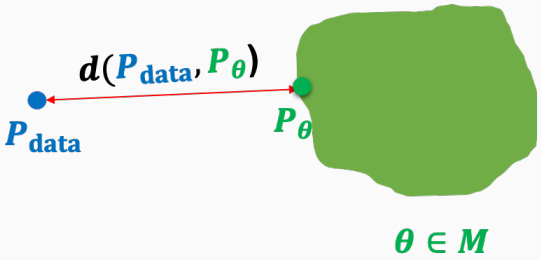
$x^{(j)} \sim P_{\text{data}}$
$j = 1, 2, \ldots, m$



- Generative Adversarial Networks (GANs)
  - $\min_\theta \max_\phi E_{\mathbf{x} \sim p_{\text{data}}}[\log D_\phi(\mathbf{x})] + E_{\mathbf{z} \sim p(\mathbf{z})}[\log(1 - D_\phi(G_\theta(\mathbf{z})))]$.
  - Two sample tests. Can optimize f-divergences and the Wasserstein distance
  - Very flexible model architectures. But likelihood is intractable, training is unstable, hard to evaluate, and has mode collapse issues

Today's lecture

$x^{(j)} \sim P_{\text{data}}$
$j = 1, 2, \ldots, m$



Energy-based models (EBMs).

- Very flexible model architectures
- Stable training
- Relatively high sample quality
- Flexible composition

Parameterizing probability distributions

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

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)

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

Given any function $f_\theta(\mathbf{x})$, we can choose

- $g_\theta(\mathbf{x}) = f_\theta(\mathbf{x})^2$
- $g_\theta(\mathbf{x}) = \exp(f_\theta(\mathbf{x}))$
- $g_\theta(\mathbf{x}) = |f_\theta(\mathbf{x})|$
- $g_\theta(\mathbf{x}) = \log(1 + \exp(f_\theta(\mathbf{x})))$

Parameterizing probability distributions

Probability distributions $p(\mathbf{x})$ satisfy two key 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)

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)

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:

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

Then by definition, $\int p_\theta(\mathbf{x}) \mathrm{d}\mathbf{x} = 1$.

Example: choose $g_\theta(\mathbf{x})$ such that the volume is an *analytical* function of θ.

1. **Gaussian**. $g_{(\mu,\sigma)}(x) = e^{-\frac{(x-\mu)^2}{2\sigma^2}}$. **Volume** is: $\int e^{-\frac{x-\mu}{2\sigma^2}} dx = \sqrt{2\pi\sigma^2}$
2. **Exponential**. $g_\lambda(x) = e^{-\lambda x}$. Volume is: $\int_0^{+\infty} e^{-\lambda x} dx = \frac{1}{\lambda}$
3. $g_\theta(x) = h(x) \exp\{\theta \cdot T(x)\}$. Volume is $\exp\{A(\theta)\}$, where $A(\theta) = \log \int h(x) \exp\{\theta \cdot T(x)\} \mathrm{d}\mathbf{x}$. $\rightarrow$ **Exponential family.** E.g.: Normal, Poisson, Bernoulli, beta, gamma etc.

Despite being restrictive, the above functional forms are very useful as building blocks for more complex distributions

Likelihood based learning

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

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

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

1. **Autoregressive:** Products of normalized objects $p_\theta(\mathbf{x}) p_{\theta'(\mathbf{x})}(\mathbf{y})$:
   $\int_\mathbf{x} \int_\mathbf{y} p_\theta(\mathbf{x}) p_{\theta'(\mathbf{x})}(\mathbf{y}) \mathrm{d}\mathbf{x} \mathrm{d}\mathbf{y} = \int_\mathbf{x} p_\theta(\mathbf{x}) \underbrace{\int_\mathbf{y} p_{\theta'(\mathbf{x})}(\mathbf{y}) \mathrm{d}\mathbf{y}}_{=1} \mathrm{d}\mathbf{x} = \int_\mathbf{x} p_\theta(\mathbf{x}) \mathrm{d}\mathbf{x} = 1$
2. **Latent variables:** Mixtures of normalized objects
   $\alpha p_\theta(\mathbf{x}) + (1-\alpha) p_{\theta'}(\mathbf{x})$:
   $\int_\mathbf{x} \alpha p_\theta(\mathbf{x}) + (1-\alpha) p_{\theta'}(\mathbf{x}) \mathrm{d}\mathbf{x} = \alpha + (1-\alpha) = 1$

How about using models where the "volume"/normalization constant of $g_\theta(\mathbf{x})$ is not easy to compute analytically?

Energy-based model

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

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. Hence, log-probs is a natural scale. 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

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

Pros:

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

Cons:

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 $\mathbf{x}$.

Applications of Energy-based models

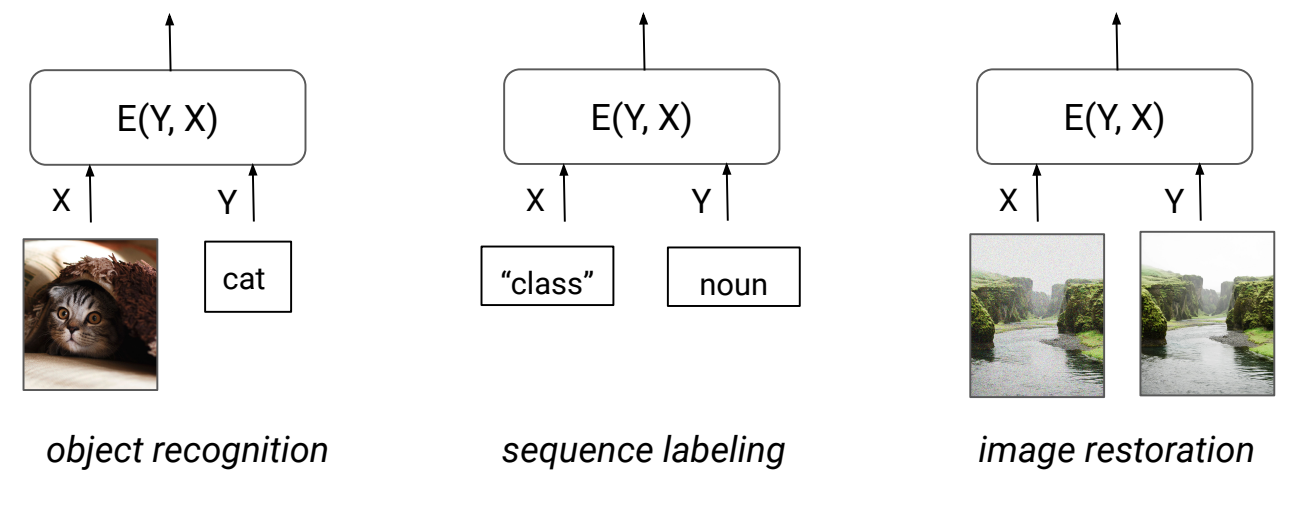
$$p_\theta(\mathbf{x}) = \frac{1}{\int \exp(f_\theta(\mathbf{x}))d\mathbf{x}} \exp(f_\theta(\mathbf{x})) = \frac{1}{Z(\theta)} \exp(f_\theta(\mathbf{x}))$$

- 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

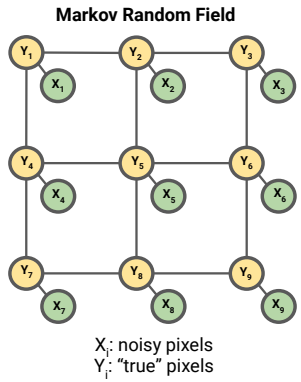
Applications of Energy-based models



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 $p(\mathbf{y}|\mathbf{x})$. Or equivalently, 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: Product of Experts

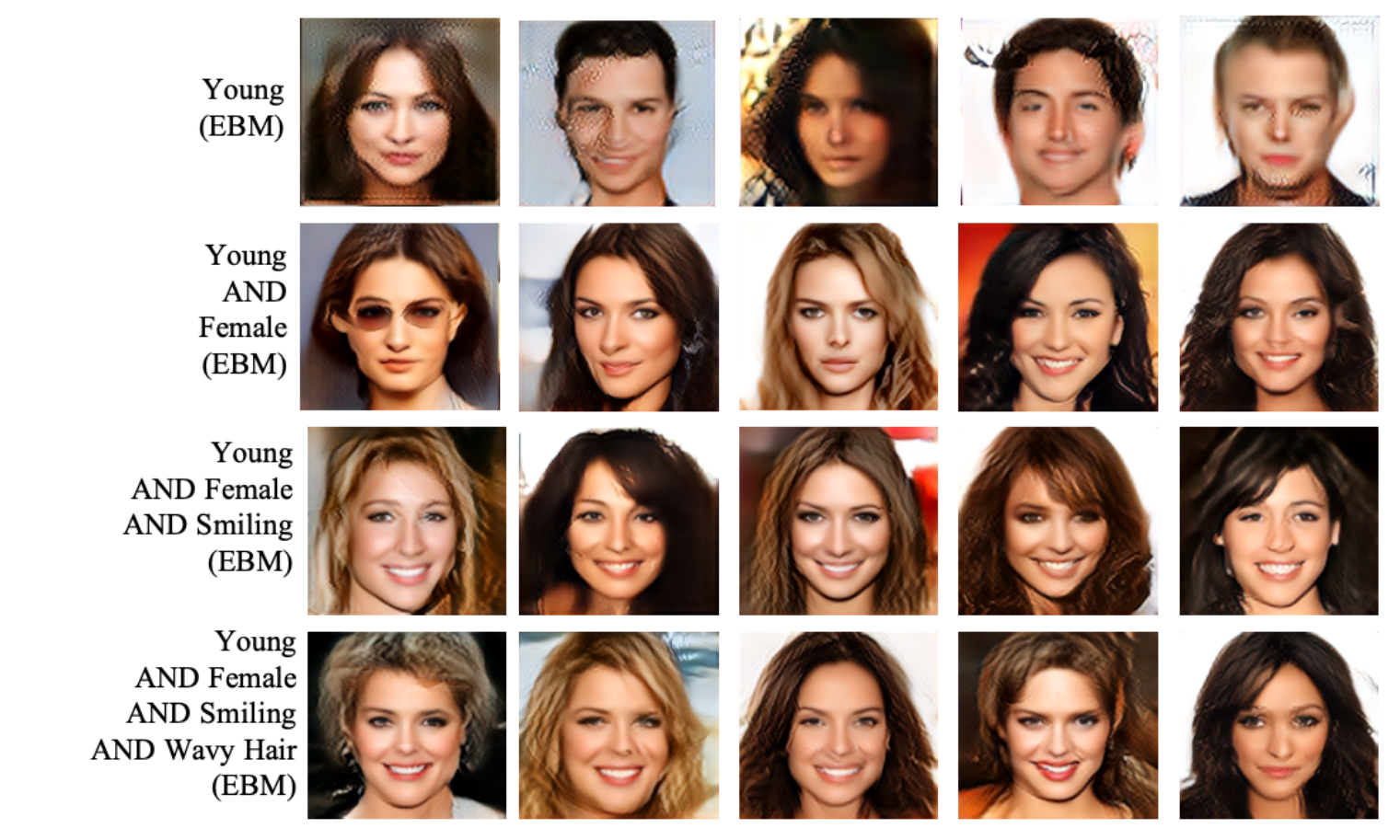
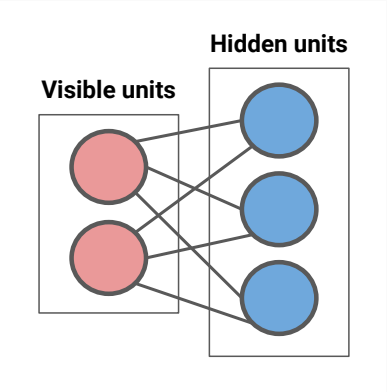


Image source: Du et al., 2020.

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

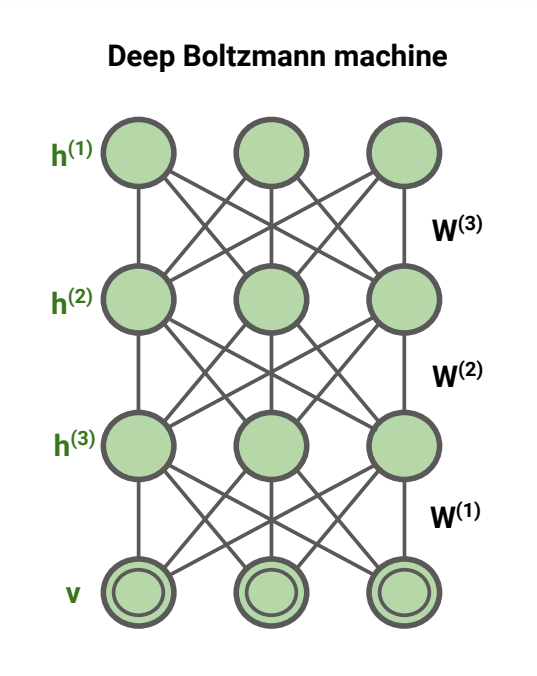
$$p_{W,b,c}(\mathbf{x},\mathbf{z}) = \frac{1}{Z}\exp\left(\mathbf{x}^T W\mathbf{z} + b\mathbf{x} + c\mathbf{z}\right) = \frac{1}{Z}\exp\left(\sum_{i=1}^{n}\sum_{j=1}^{m} x_i z_j w_{ij} + b\mathbf{x} + c\mathbf{z}\right)$$



- 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

Example: Deep Boltzmann Machines

Stacked RBMs are 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.

Deep Boltzmann Machines: samples

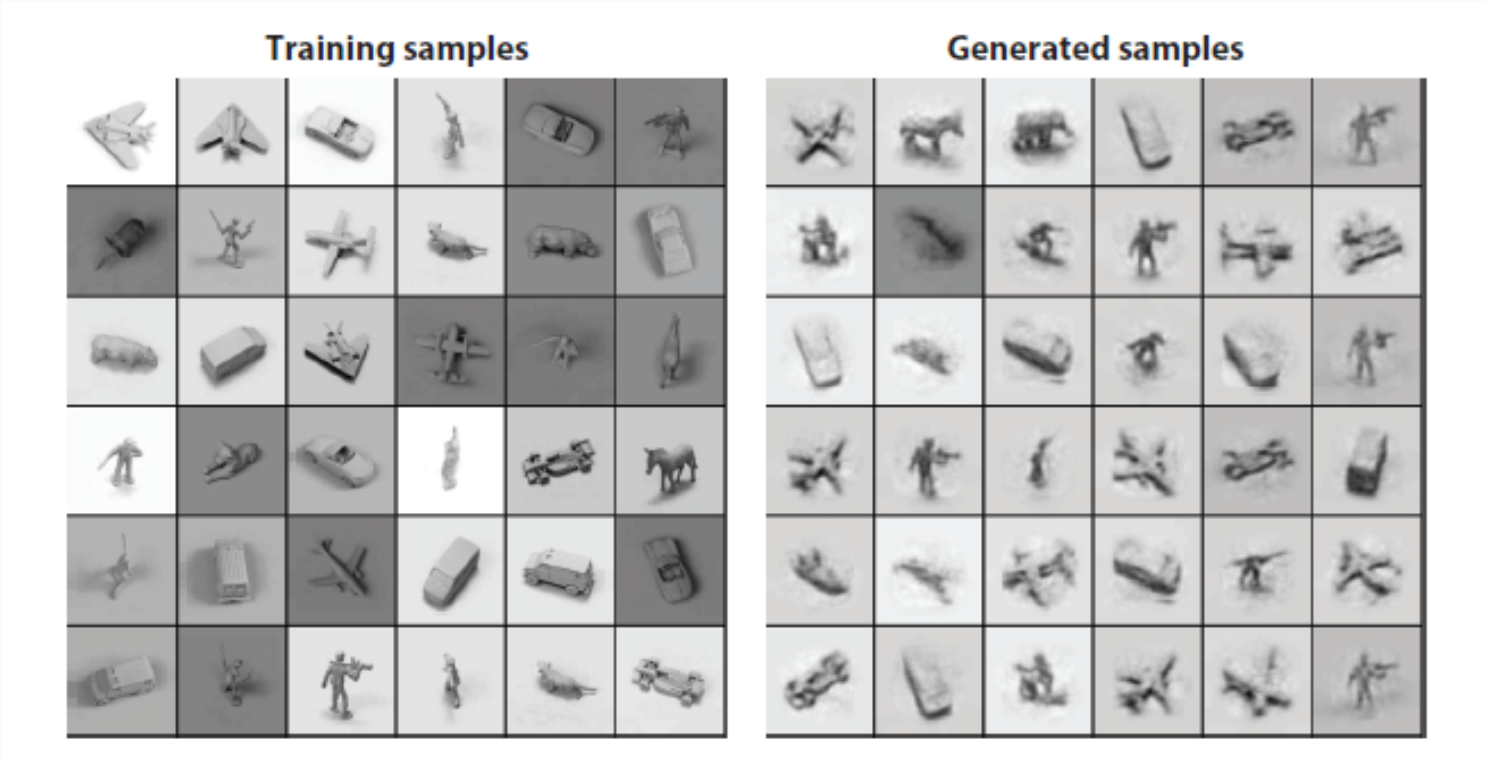


Image source: Salakhutdinov and Hinton, 2009.

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. No feature learning

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 $\mathbf{x}$.

Computing the normalization constant is hard

- As an example, the RBM joint distribution is

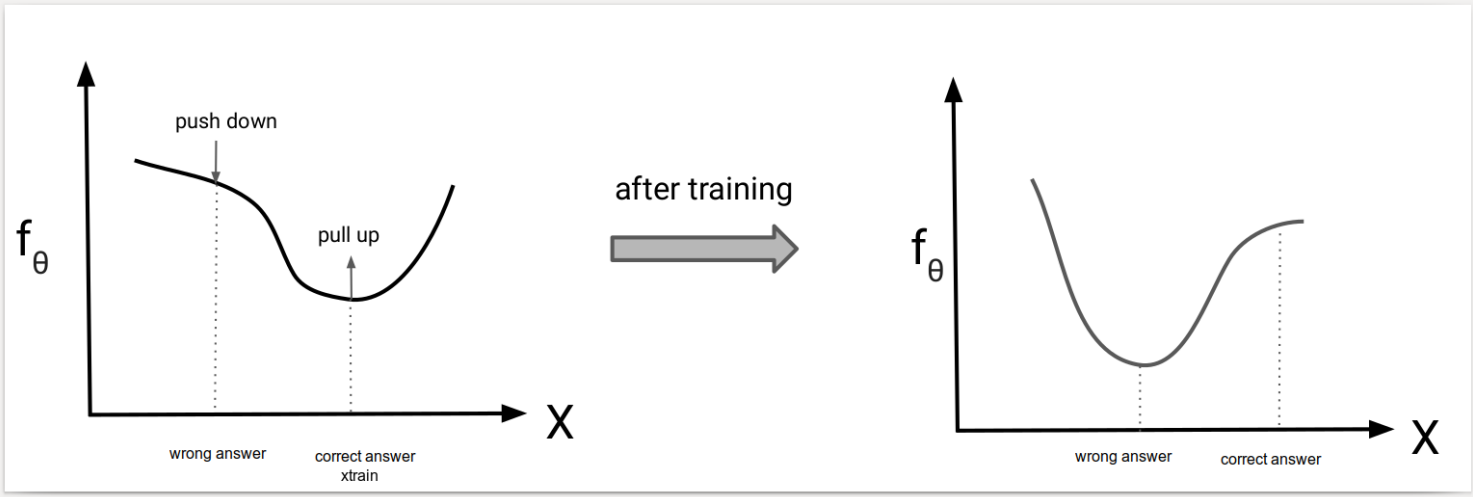
$$p_{W,b,c}(\mathbf{x}, \mathbf{z}) = \frac{1}{Z} \exp\left(\mathbf{x}^T W \mathbf{z} + b\mathbf{x} + c\mathbf{z}\right)$$

  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

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

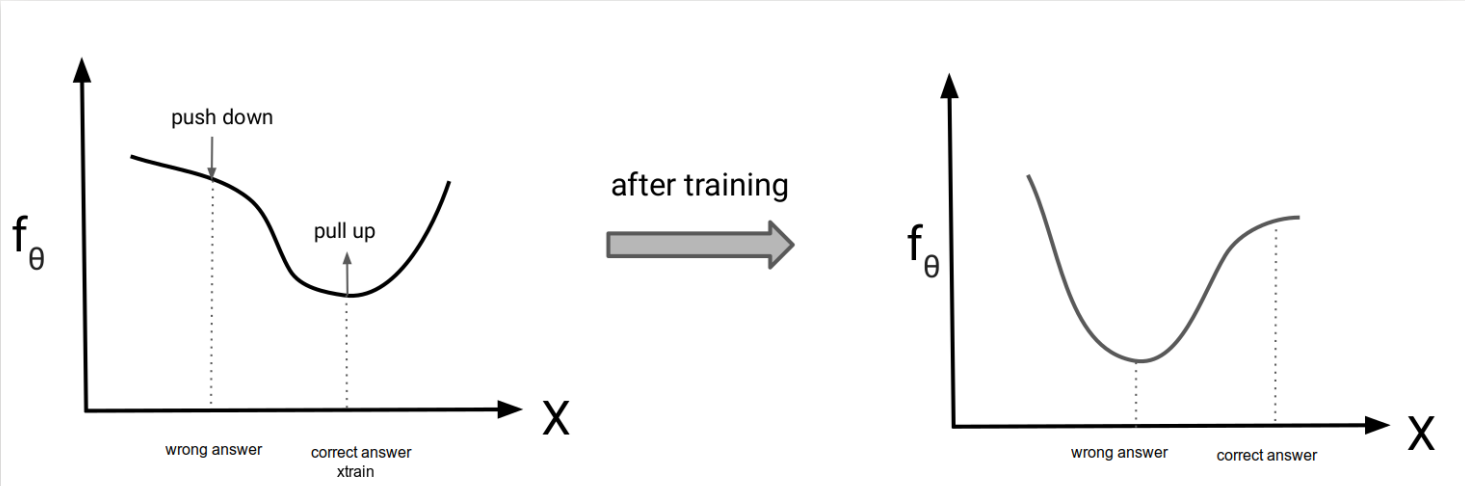
- **Note:** it is a well defined function of the parameters W, b, c, but no simple closed-form. Takes time exponential in n, m to compute. This means that *evaluating* the objective function $p_{W,b,c}(\mathbf{x}, \mathbf{z})$ for likelihood based learning is hard.
- **Observation:** Optimizing the likelihood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

Training intuition



- Goal: maximize $\frac{\exp\{f_\theta(\mathbf{x}_{train})\}}{Z(\theta)}$. Increase numerator, decrease denominator.
- **Intuition**: because the model is not normalized, increasing the un-normalized log-probability $f_\theta(\mathbf{x}_{train})$ by changing θ does **not** guarantee that $\mathbf{x}_{train}$ 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.

Contrastive Divergence



- Goal: maximize $\frac{\exp\{f_\theta(x_{train})\}}{Z(\theta)}$
- **Idea**: Instead of evaluating $Z(\theta)$ exactly, use a Monte Carlo estimate.
- **Contrastive divergence algorithm**: sample $x_{sample} \sim p_\theta$, 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.

Contrastive Divergence

- Maximize log-likelihood: $\max_\theta f_\theta(x_{train}) - \log Z(\theta)$.
- Gradient of log-likelihood:

$$\begin{aligned}
& \nabla_\theta f_\theta(x_{train}) - \nabla_\theta \log Z(\theta) \\
= \quad & \nabla_\theta f_\theta(x_{train}) - \frac{\nabla_\theta Z(\theta)}{Z(\theta)} \\
= \quad & \nabla_\theta f_\theta(x_{train}) - \frac{1}{Z(\theta)} \int \nabla_\theta \exp\{f_\theta(x)\} \mathrm{d}x \\
= \quad & \nabla_\theta f_\theta(x_{train}) - \frac{1}{Z(\theta)} \int \exp\{f_\theta(x)\} \nabla_\theta f_\theta(x) \mathrm{d}x \\
= \quad & \nabla_\theta f_\theta(x_{train}) - \int \frac{\exp\{f_\theta(x)\}}{Z(\theta)} \nabla_\theta f_\theta(x) \mathrm{d}x \\
= \quad & \nabla_\theta f_\theta(x_{train}) - E_{x_{sample}}[\nabla_\theta f_\theta(x_{sample})] \\
\approx \quad & \nabla_\theta f_\theta(x_{train}) - \nabla_\theta f_\theta(x_{sample}),
\end{aligned}$$

  where $x_{sample} \sim \exp\{f_\theta(x_{sample})\}/Z(\theta)$.
- How to sample?

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 an iterative approach called Markov Chain Monte Carlo:
  1. Initialize x^0 randomly, $t = 0$
  2. Let $x' = x^t +$ noise
     - 2.1 If $f_\theta(x') > f_\theta(x^t)$, let $x^{t+1} = x'$
     - 2.2 Else let $x^{t+1} = x'$ with probability $\exp(f_\theta(x') - f_\theta(x^t))$
  3. Go to step **??**
- Works in theory, but can take a very long time to converge

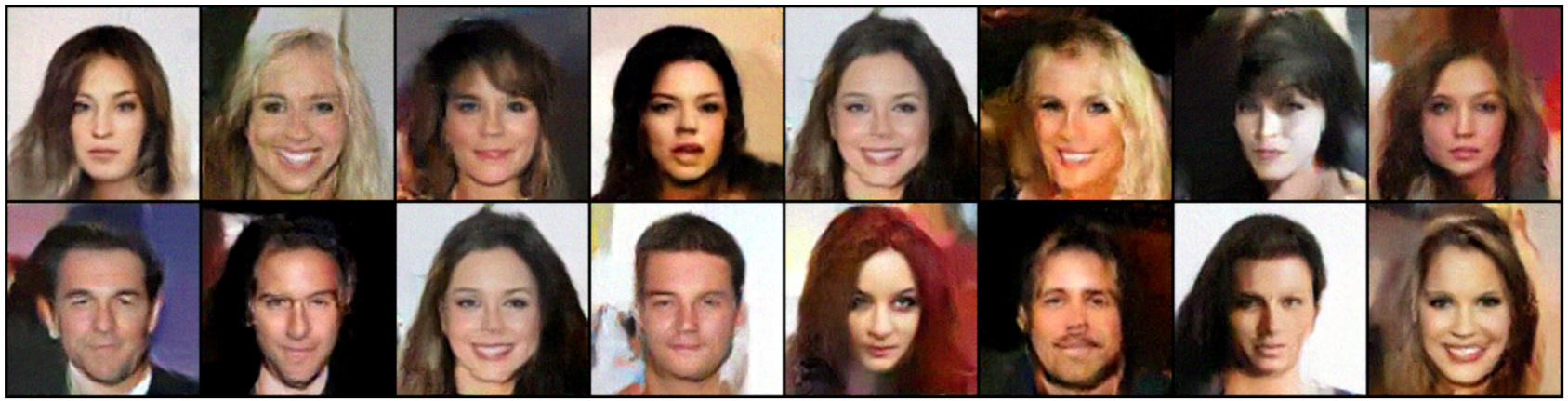
Sampling from energy-based models

- For any continuous distribution $p_\theta(\mathbf{x})$, suppose we can compute its gradient (the **score function**) $\nabla_{\mathbf{x}} \log p_\theta(\mathbf{x})$.
- Let $\pi(\mathbf{x})$ be a prior distribution that is easy to sample from.
- Langevin MCMC.
  - $\mathbf{x}^0 \sim \pi(\mathbf{x})$
  - Repeat $\mathbf{x}^{t+1} \sim \mathbf{x}^t + \epsilon \, \nabla_{\mathbf{x}} \log p_\theta(\mathbf{x}^t) + \sqrt{2\epsilon} \, \mathbf{z}^t$ for $t = 0, 1, 2, \cdots, T-1$, where $\mathbf{z}^t \sim \mathcal{N}(0, I)$.
  - If $\epsilon \to 0$ and $T \to \infty$, we have $\mathbf{x}^T \sim p_\theta(\mathbf{x})$.
- Note that for energy-based models

$$\begin{aligned} \nabla_{\mathbf{x}} \log p_\theta(\mathbf{x}) &= \nabla_{\mathbf{x}} f_\theta(\mathbf{x}) - \underbrace{\nabla_{\mathbf{x}} \log Z(\theta)}_{=0} \\ &= \nabla_{\mathbf{x}} f_\theta(\mathbf{x}) \end{aligned}$$

Modern energy-based models

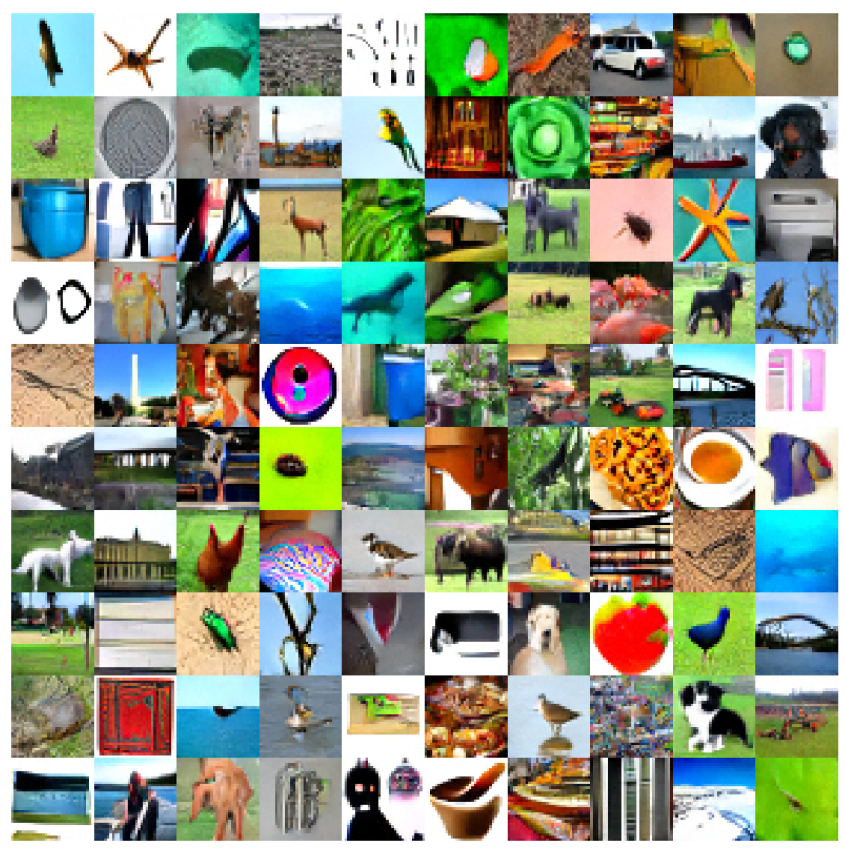
Langevin sampling



Face samples

Image source: Nijkamp et al. 2019

Modern energy-based models



ImageNet samples

Image source: Du et al., 2019