

Deep Generative Models

Energy-Based Models

Hamid Beigy

Sharif University of Technology

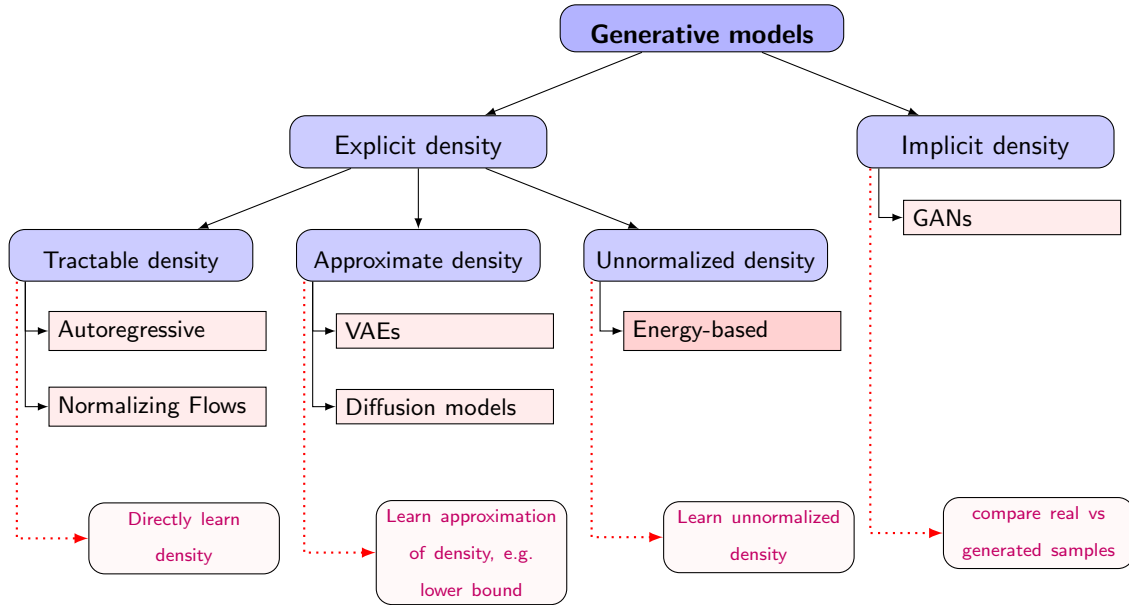
May 11, 2024



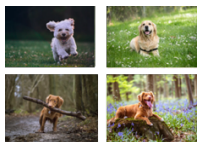


1. Introduction
2. Energy-based models
3. Likelihood Based Methods
4. Score-Based Generative Models
5. Noise Contrastive Estimation based Methods
6. Adversarial training
7. Hybrid Modeling
8. Summary
9. References

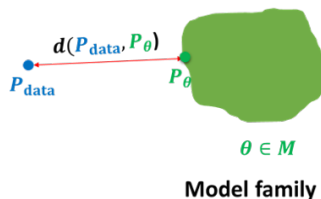
Introduction



1. Assume that the observed variable \mathbf{x} is a random sample from an underlying process, whose true distribution $p_{data}(\mathbf{x})$ is unknown.



$$\begin{aligned} \mathbf{x}_i &\sim P_{data} \\ i &= 1, 2, \dots, n \end{aligned}$$



2. We attempt to approximate this process with a chosen model, $p_{\theta}(\mathbf{x})$, with parameters θ such that $\mathbf{x} \sim p_{\theta}(\mathbf{x})$.
3. Learning is the process of searching for the parameter θ such that $p_{\theta}(\mathbf{x})$ well approximates $p_{data}(\mathbf{x})$ for any observed \mathbf{x} , i.e.

$$p_{\theta}(\mathbf{x}) \approx p_{data}(\mathbf{x})$$

4. We wish $p_{\theta}(\mathbf{x})$ to be sufficiently flexible to be able to adapt to the data for obtaining sufficiently accurate model and to be able to incorporate prior knowledge.



Autoregressive models

1. Tractable density
2. Density is estimated as

$$p(\mathbf{x}; \theta) = \prod_{j=1}^m p(\mathbf{x}_j \mid \mathbf{x}_{<j}; \theta)$$

3. Tractable likelihood
4. No inferred latent factors

Normalizing flow models

1. Exact density
2. Density is estimated as

$$p_{\theta}(\mathbf{x}) = p_{\mathbf{z}}(\mathbf{z})|\det(\mathbf{J}_f)|$$

where $\mathbf{z} = f(\mathbf{x})$

3. Tractable likelihood
4. Latent feature representation

Latent variable models

1. Approximated density
2. Density is estimated as

$$p(\mathbf{x}; \theta) = \int p(\mathbf{x}, \mathbf{z}; \theta) dz$$

3. Intractable likelihood
4. Latent feature representation

Generative adversarial networks

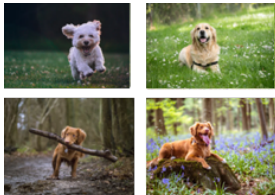
1. Implicit density
2. Can optimize f-divergences and Wasserstein distance

$$\min_G \max_D \mathbb{E}_{\mathbf{x} \sim p_{data}(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})} [\log(1 - D(G(\mathbf{z})))]$$

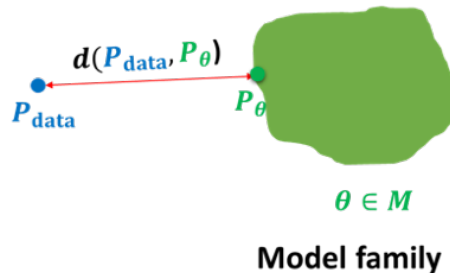
3. Intractable likelihood
4. Latent feature representation
5. Very flexible model architectures, unstable training, hard evaluation, mode collapse

Energy-based models

1. Very flexible model architectures
2. Stable training
3. Relatively high sample quality
4. Flexible composition



$$\begin{aligned}x_i &\sim P_{\text{data}} \\ i &= 1, 2, \dots, n\end{aligned}$$





1. Probability distributions $p_\theta(\mathbf{x})$ are a key building block in generative modeling.
2. They have the following properties
 - **Non-negative:** $p_\theta(\mathbf{x}) \geq 0$
 - **Sum to one:** $\sum_{\mathbf{x}} p_\theta(\mathbf{x}) = 1$ or $\int_{\mathbf{x}} p_\theta(\mathbf{x}) d\mathbf{x} = 1$
3. Making non-negativeness is easy and we can choose any of the following function:

$$g_\theta(\mathbf{x}) = f_\theta(\mathbf{x})^2$$

$$g_\theta(\mathbf{x}) = \exp(f_\theta(\mathbf{x}))$$

$$g_\theta(\mathbf{x}) = |f_\theta(\mathbf{x})|$$

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

4. In general $Z_\theta = \sum_{\mathbf{x}} g_\theta(\mathbf{x}) \neq 1$.
5. Hence, $g_\theta(\mathbf{x})$ is not a valid probability mass function or density.



1. The maintaining $g_{\theta}(\mathbf{x}) \geq 0$ is easy but making $\sum_{\mathbf{x}} p_{\theta}(\mathbf{x}) = 1$ is a hard problem.
2. A solution is to normalize $g_{\theta}(\mathbf{x})$ by its volume as

$$p_{\theta}(\mathbf{x}) = \frac{g_{\theta}(\mathbf{x})}{\text{Volume}(g_{\theta})} = \frac{g_{\theta}(\mathbf{x})}{\int_{\mathbf{x}} g_{\theta} d\mathbf{x}}$$

3. Then, by definition we have $\int_{\mathbf{x}} p_{\theta}(\mathbf{x}) d\mathbf{x} = 1$.
4. We can calculate the volume analytically if we choose some analytical functions such as

Density function

$$g_{(\mu, \sigma)}(x) = e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

$$g_{\lambda}(x) = e^{-\lambda x}$$

$$g_{\theta}(x) = h(x) \exp(\theta T(x))$$

Volume

$$\int_{\mathbf{x}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx = \sqrt{2\pi\sigma^2}$$

$$\int_0^{\infty} e^{-\lambda x} dx = \frac{1}{\lambda}$$

$$\exp(\log \int h(x) \exp(\theta T(x)) dx)$$

5. The above functional forms seem to be restrictive but they are very useful as building blocks for more complex distributions.



1. The density function given by an EBM is

$$p_{\theta}(\mathbf{x}) = \frac{\exp(-E_{\theta}(\mathbf{x}))}{Z_{\theta}}$$

2. $E_{\theta}(\mathbf{x})$ (the energy) is a nonlinear regression function with parameters θ .
3. Alternatively, happiness, $H(\mathbf{x}) = -E(\mathbf{x})$, is used to avoid multiple minus signs.
4. Z_{θ} denotes the normalizing constant (partition function):

$$Z_{\theta} = \int \exp(-E_{\theta}(\mathbf{x})) d\mathbf{x}$$

5. Z_{θ} is constant w.r.t \mathbf{x} and is a function of θ .
6. Evaluation and differentiation of $\log p_{\theta}(\mathbf{x})$ w.r.t. its parameters involves a typically intractable integral.
7. Here, we don't care about the exact density (which needs to compute the partition function Z_{θ}), but only interested in the relative order of densities.



1. We can fit $p_\theta(\mathbf{x})$ to $p_{data}(\mathbf{x})$ by maximizing the expected log-likelihood function over the data distribution, defined by

$$\mathbb{E}_{\mathbf{x} \sim p_{data}(\mathbf{x})}[\log p_\theta(\mathbf{x})]$$

2. Here the expectation can be easily estimated with samples from the dataset.
3. Maximizing likelihood is equivalent to minimizing the KL divergence between $p_{data}(\mathbf{x})$ and $p_\theta(\mathbf{x})$, because

$$\begin{aligned} -\mathbb{E}_{\mathbf{x} \sim p_{data}(\mathbf{x})}[\log p_\theta(\mathbf{x})] &= D_{KL}(p_{data}(\mathbf{x}) \parallel p_\theta(\mathbf{x})) - \mathbb{E}_{\mathbf{x} \sim p_{data}(\mathbf{x})}[\log p_{data}(\mathbf{x})] \\ &= D_{KL}(p_{data}(\mathbf{x}) \parallel p_\theta(\mathbf{x})) - \text{constant} \end{aligned}$$

4. We cannot directly compute the likelihood of an EBM as in the maximum likelihood approach due to the intractable normalizing constant Z_θ .
5. Nevertheless, we can still estimate the gradient of the log-likelihood with MCMC approaches, allowing for likelihood maximization with gradient ascent.

Likelihood Based Methods



1. The density function given by an EBM is

$$p_{\theta}(\mathbf{x}) = \frac{\exp(-E_{\theta}(\mathbf{x}))}{Z_{\theta}}$$

2. The gradient of negative log-likelihood (NLL) is decomposed to:

$$\nabla_{\theta} \log p_{\theta}(\mathbf{x}) = -\nabla_{\theta} E_{\theta}(\mathbf{x}) - \nabla_{\theta} \log Z_{\theta}$$

3. The first gradient term, $-\nabla_{\theta} E_{\theta}(\mathbf{x})$, is straightforward to evaluate with automatic differentiation.
4. The challenge is in approximating the second gradient term, $\nabla_{\theta} \log Z_{\theta}$, which is intractable to compute exactly.



$\nabla_{\theta} \log Z_{\theta}$ can be rewritten as follows:

$$\begin{aligned}
 \nabla_{\theta} \log Z_{\theta} &= \nabla_{\theta} \log \int \exp(-E_{\theta}(\mathbf{x})) d\mathbf{x} \\
 &= \left(\int \exp(-E_{\theta}(\mathbf{x})) d\mathbf{x} \right)^{-1} \nabla_{\theta} \int \exp(-E_{\theta}(\mathbf{x})) d\mathbf{x} \\
 &= \left(\int \exp(-E_{\theta}(\mathbf{x})) d\mathbf{x} \right)^{-1} \int \nabla_{\theta} \exp(-E_{\theta}(\mathbf{x})) d\mathbf{x} \\
 &= \left(\int \exp(-E_{\theta}(\mathbf{x})) d\mathbf{x} \right)^{-1} \int \exp(-E_{\theta}(\mathbf{x})) (-\nabla_{\theta} E_{\theta}(\mathbf{x})) d\mathbf{x} \\
 &= \int \left(\int \exp(-E_{\theta}(\mathbf{x})) d\mathbf{x} \right)^{-1} \exp(-E_{\theta}(\mathbf{x})) (-\nabla_{\theta} E_{\theta}(\mathbf{x})) d\mathbf{x} \\
 &= \int \frac{\exp(-E_{\theta}(\mathbf{x}))}{Z_{\theta}} (-\nabla_{\theta} E_{\theta}(\mathbf{x})) d\mathbf{x} \\
 &= \int p_{\theta}(\mathbf{x}) (-\nabla_{\theta} E_{\theta}(\mathbf{x})) d\mathbf{x} = \mathbb{E}_{\mathbf{x} \sim p_{\theta}(\mathbf{x})} [-\nabla_{\theta} E_{\theta}(\mathbf{x})]
 \end{aligned}$$

Thus, we can obtain an unbiased one-sample Monte Carlo estimate of the log-likelihood gradient by

$$\nabla_{\theta} \log Z_{\theta} \approx -\nabla_{\theta} E_{\theta}(\tilde{\mathbf{x}})$$

where $\tilde{\mathbf{x}} \sim p_{\theta}(\mathbf{x})$ is a random sample from the distribution over \mathbf{x} given by the EBM.



1. As long as we can draw random samples from the model, we have access to an unbiased Monte Carlo estimate of the log-likelihood gradient, allowing us to optimize the parameters with stochastic gradient ascent.
2. Since drawing random samples is far from being trivial, much of the literature has focused on methods for efficient MCMC sampling from EBMs.
3. Some efficient MCMC methods make use of the fact that the gradient of the log-probability w.r.t. \mathbf{x} (score) is equal to the (negative) gradient of the energy, therefore easy to calculate:

$$\nabla_{\mathbf{x}} \log p_{\theta}(\mathbf{x}) = -\nabla_{\mathbf{x}} E_{\theta}(\mathbf{x}) - \underbrace{\nabla_{\mathbf{x}} \log Z_{\theta}}_{=0} = -\nabla_{\mathbf{x}} E_{\theta}(\mathbf{x})$$

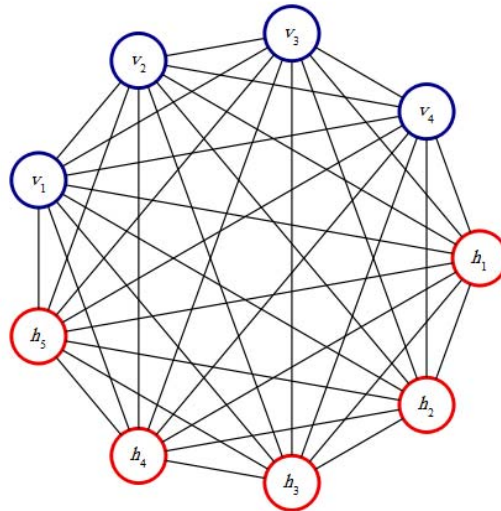
4. For example, Langevin MCMC initially draws a sample \mathbf{x}_0 from a simple prior distribution, and then uses a process for K steps with step size $\epsilon > 0$:

$$\mathbf{x}^{k+1} \leftarrow \mathbf{x}^k + \frac{\epsilon^2}{2} \nabla_{\mathbf{x}} \log p_{\theta}(\mathbf{x}) + \epsilon \mathbf{z}^k$$

where $\mathbf{z}^k \sim \mathcal{N}(0, \mathbf{I})$ is a Gaussian noise term.



1. BMs are fully connected networks of binary units.
2. BM is an undirected symmetric network of binary units that are divided into **visible** and **hidden** units.





1. BMs are theoretically capable of **learning any given distribution**.
2. The network **sets the strengths of the connections between the units** to capture the **correlations** between them to build a generative network **capable of producing new examples of the same distribution**.
3. Since all variables in a BM are not directly observed, it gives us a handle to control the sampling of new examples.
4. The model can take in an incomplete example and use it to output the complete version.



1. BM is a network with an **energy** defined for the overall network.
2. For a BM with only observed units, the energy is defined as

$$\begin{aligned} E_{\theta}(\mathbf{x}) &= - \sum_{ij} w_{ij} x_i x_j - \sum_{i=1} b_i x_i \\ &= -\mathbf{x}^{\top} \mathbf{W} \mathbf{x} - \mathbf{b}^{\top} \mathbf{x} \end{aligned}$$

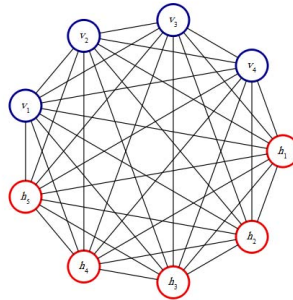
- $\mathbf{x} = (x_1, x_2, \dots, x_d) \in \{0, 1\}^d$ is the input vector.
 - $\mathbf{W} = (w_{ij})$ is the weight matrix
 - $\mathbf{b} = (b_1, b_2, \dots, b_d) \in \{0, 1\}^d$ is the bias vector.
3. The joint probability distribution defined as

$$p_{\theta}(\mathbf{x}) = \frac{\exp(-E_{\theta}(\mathbf{x}))}{Z_{\theta}}$$

Z_{θ} is Partition function that ensures $\sum_{\mathbf{x}} p_{\theta}(\mathbf{x}) = 1$.



1. BM becomes more powerful when not all the variables are observed.
2. The **latent variables** can act similarly to **hidden units** in a MLP.



3. By decomposing units into two subsets: **visible \mathbf{v}** and **hidden units \mathbf{h}** , we obtain.

$$E_{\theta}(\mathbf{v}, \mathbf{h}) = -\mathbf{v}^T \mathbf{R} \mathbf{v} - \mathbf{v}^T \mathbf{W} \mathbf{h} - \mathbf{h}^T \mathbf{S} \mathbf{h} - \mathbf{b}^T \mathbf{v} - \mathbf{c}^T \mathbf{h}$$

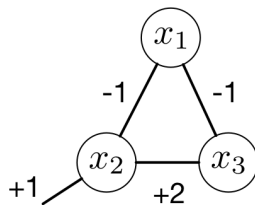
4. The joint probability distribution defined as

$$p_{\theta}(\mathbf{v}, \mathbf{h}) = \frac{\exp(-E(\mathbf{v}, \mathbf{h}))}{Z_{\theta}}$$

Z_{θ} is **Partition function** that ensures $\sum_{\mathbf{x}} p_{\theta}(\mathbf{x}) = 1$.



Example:



x_1	x_2	x_3	$w_{12}x_1x_2$	$w_{13}x_1x_3$	$w_{23}x_2x_3$	$b_{2}x_2$	$H(\mathbf{x})$	$\exp(H(\mathbf{x}))$	$p(\mathbf{x})$
-1	-1	-1	-1	-1	2	-1	-1	0.368	0.0021
-1	-1	1	-1	1	-2	-1	-3	0.050	0.0003
-1	1	-1	1	-1	-2	1	-3	0.368	0.0021
-1	1	1	1	1	2	1	5	148.413	0.8608
1	-1	-1	1	1	2	-1	3	20.086	0.1165
1	-1	1	1	-1	-2	-1	-3	0.050	0.0003
1	1	-1	-1	1	-2	1	-1	0.368	0.0021
1	1	1	-1	-1	2	1	1	2.718	0.0158

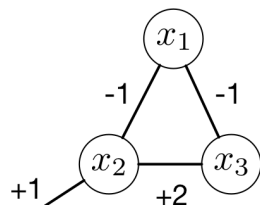
$$\mathcal{Z} = 172.420$$

Figure: Roger Grosse



Marginal probabilities:

$$\begin{aligned}
 p(x_1 = 1) &= \frac{1}{Z} \sum_{\mathbf{x}:x_1=1} \exp(H(\mathbf{x})) \\
 &= \frac{20.086 + 0.050 + 0.368 + 2.718}{172.420} \\
 &= 0.135
 \end{aligned}$$



x_1	x_2	x_3	$w_{12}x_1x_2$	$w_{13}x_1x_3$	$w_{23}x_2x_3$	b_2x_2	$H(\mathbf{x})$	$\exp(H(\mathbf{x}))$	$p(\mathbf{x})$
-1	-1	-1	-1	-1	2	-1	-1	0.368	0.0021
-1	-1	1	-1	1	-2	-1	-3	0.050	0.0003
-1	1	-1	1	-1	-2	1	-3	0.368	0.0021
-1	1	1	1	1	2	1	5	148.413	0.8608
1	-1	-1	1	1	2	-1	3	20.086	0.1165
1	-1	1	1	-1	-2	-1	-3	0.050	0.0003
1	1	-1	-1	1	-2	1	-1	0.368	0.0021
1	1	1	-1	-1	2	1	1	2.718	0.0158

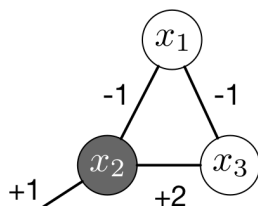
$$Z = 172.420$$

Figure: Roger Grosse



Conditional probabilities:

$$\begin{aligned}
 p(x_1 = 1 | x_2 = -1) &= \frac{\sum_{\mathbf{x}: x_1=1, x_2=-1} \exp(H(\mathbf{x}))}{\sum_{\mathbf{x}: x_2=-1} \exp(H(\mathbf{x}))} \\
 &= \frac{20.086 + 0.050}{0.368 + 0.050 + 20.086 + 0.050} \\
 &= 0.980
 \end{aligned}$$



x_1	x_2	x_3	$w_{12}x_1x_2$	$w_{13}x_1x_3$	$w_{23}x_2x_3$	b_2x_2	$H(\mathbf{x})$	$\exp(H(\mathbf{x}))$	$p(\mathbf{x})$
-1	-1	-1	-1	-1	2	-1	-1	0.368	0.0021
-1	-1	1	-1	1	-2	-1	-3	0.050	0.0003
-1	1	-1	1	-1	-2	1	-3	0.368	0.0021
-1	1	1	1	1	2	1	5	148.413	0.8608
1	-1	-1	1	1	2	-1	3	20.086	0.1165
1	-1	1	1	-1	-2	-1	-3	0.050	0.0003
1	1	-1	-1	1	-2	1	-1	0.368	0.0021
1	1	1	-1	-1	2	1	1	2.718	0.0158

Figure: Roger Grosse



1. Learning algorithms for BMs are usually based on **maximum likelihood**.
2. All BMs have an **intractable partition function**, so the **maximum likelihood gradient must be approximated**.
3. An interesting property of BMs is that the update for a particular w_{ij} depends only on the statistics of x_i and x_j .



1. A BM admits the following likelihood for points $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}$.

$$\mathcal{L}(\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}) = \prod_{i=1}^n p(\mathbf{x}^{(i)})$$

2. We will work with the log-likelihood instead of the true likelihood.

$$\begin{aligned} \log \mathcal{L}(\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}) &= \sum_{k=1}^n \log \frac{\exp(H(\mathbf{x}^{(k)}))}{Z_{\theta}} \\ &= \sum_{k=1}^n \log \left(\exp(H(\mathbf{x}^{(k)})) \right) - \log Z_{\theta} \\ &= \sum_{k=1}^n H(\mathbf{x}^{(k)}) - \log Z_{\theta} \end{aligned}$$

3. The aim is to maximize $\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})}[\mathcal{L}(\mathbf{x})]$

$$\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} [\mathcal{L}(\mathbf{x})] = \sum_{k=1}^n p_{\text{data}}(\mathbf{x} = \mathbf{x}^{(k)}) \mathcal{L}(\mathbf{x}^{(k)})$$



1. Now, deriving the gradient with respect to the weights ($\nabla_{w_{i,j}} \log \mathcal{L}$)

$$\begin{aligned} \nabla_{w_{i,j}} \left[\sum_{k=1}^n p_{\text{data}}(\mathbf{x} = \mathbf{x}^{(k)}) \left(H(\mathbf{x}^{(k)}) - \log Z \right) \right] &= \sum_{k=1}^n p_{\text{data}}(\mathbf{x} = \mathbf{x}^{(k)}) \nabla_{w_{i,j}} H(\mathbf{x}^{(k)}) \\ &\quad - \sum_{k=1}^n p_{\text{data}}(\mathbf{x} = \mathbf{x}^{(k)}) \nabla_{w_{i,j}} \log Z_{\theta} \end{aligned}$$

2. The first term equals to

$$\begin{aligned} \sum_{k=1}^n p_{\text{data}}(\mathbf{x} = \mathbf{x}^{(k)}) \nabla_{w_{i,j}} H(\mathbf{x}^{(k)}) &= \sum_{k=1}^n p_{\text{data}}(\mathbf{x} = \mathbf{x}^{(k)}) \nabla_{w_{i,j}} \left[\sum_{i \neq j} w_{i,j} x_i^{(k)} x_j^{(k)} \right. \\ &\quad \left. + \sum_i p_{\text{data}}(\mathbf{x} = \mathbf{x}^{(k)}) b_i x_i^{(k)} \right] \\ &= \sum_{k=1}^n p_{\text{data}}(\mathbf{x} = \mathbf{x}^{(k)}) x_i^{(k)} x_j^{(k)} \\ &= \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} [x_i x_j] \end{aligned}$$



1. The second term equals to

$$\begin{aligned}\nabla_{w_{i,j}} \log Z_\theta &= \nabla_{w_{i,j}} \log \sum_{\mathbf{x}} \exp(H(\mathbf{x})) \\ &= \frac{1}{\sum_{\mathbf{x}} \exp(H(\mathbf{x}))} \nabla_{w_{i,j}} \sum_{\mathbf{x}} \exp(H(\mathbf{x})) = \frac{1}{Z_\theta} \nabla_{w_{i,j}} \sum_{\mathbf{x}} \exp(H(\mathbf{x})) \\ &= \frac{1}{Z_\theta} \sum_{\mathbf{x}} \exp(H(\mathbf{x})) \nabla_{w_{i,j}} H(\mathbf{x}) = \sum_{\mathbf{x}} \frac{\exp(H(\mathbf{x}))}{Z_\theta} \nabla_{w_{i,j}} H(\mathbf{x}) \\ &= \sum_{\mathbf{x}} p_\theta(\mathbf{x}) \nabla_{w_{i,j}} H(\mathbf{x}) \\ &= \sum_{\mathbf{x}} p_\theta(\mathbf{x}) [x_i x_j] \\ &= \mathbb{E}_{\mathbf{x} \sim p_\theta} [x_i x_j]\end{aligned}$$

2. By combining the above equations, the gradient w.r.t weights becomes

$$\nabla_{w_{i,j}} \log \mathcal{L} = \mathbb{E}_{\mathbf{x} \sim p_\theta(\mathbf{x})} [x_i x_j] - \mathbb{E}_{\mathbf{x} \sim p_\theta(\mathbf{x})} [x_i x_j]$$

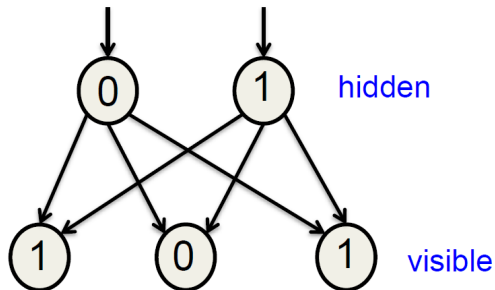
3. By combining the above equations, the gradient w.r.t biases becomes

$$\nabla_{b_i} \log \mathcal{L} = \mathbb{E}_{\mathbf{x} \sim p_\theta} [x_i] - \mathbb{E}_{\mathbf{x} \sim p_\theta} [x_i]$$



1. In BM, we generate in two steps:

- Pick the hidden states from $p(\mathbf{h})$.
- Pick the visible states from $p(\mathbf{v}|\mathbf{h})$.



2. The probability of generating a visible vector, \mathbf{v} , is computed by summing over all possible hidden states.

$$p(\mathbf{v}) = \sum_{\mathbf{h}} p(\mathbf{h})p(\mathbf{v}|\mathbf{h})$$



1. Given an ordered set of variable, x_1, \dots, x_d , and a starting configuration $x^0 = (x_1^0, \dots, x_d^0)$,

Gibbs sampling uses the following procedure

- Repeat until convergence for $t = 1, 2, \dots$,
 - Set $\mathbf{x} \leftarrow \mathbf{x}^{t-1}$.
 - For each variable x_i in the order we fixed:
 - 1) Sample $x'_i \sim p(x_i \mid \mathbf{x}_{-i})$.
 - 2) Update $\mathbf{x} \leftarrow (x_1, \dots, x'_i, \dots, x_d)$.
 - Set $\mathbf{x}^t \leftarrow \mathbf{x}$.

We use \mathbf{x}_{-i} to denote all variables in \mathbf{x} except x_i .

2. It is often very easy to performing each sampling step, since we only need to condition x_i on other variables.
3. Note that when we update x_i , we immediately use its new value for sampling other variables x_j .



1. We derive $p(x_i | \mathbf{x}_{-i})$ using probability of axioms and discarding bias terms

$$\begin{aligned} p(x_i = 1 | \mathbf{x}_{-i}) &= \frac{p(x_i = 1, \mathbf{x}_{-i})}{p(x_i = 1, \mathbf{x}_{-i}) + p(x_i = 0, \mathbf{x}_{-i})} \\ &= \frac{\exp\left(\left[\sum_{i \neq j} w_{ij} x_j\right]\right)}{1 + \exp\left(\left[\sum_{i \neq j} w_{ij} x_j\right]\right)} \\ &= \frac{1}{1 + \exp\left(\left[-\sum_{j \neq i} w_{ij} x_j\right]\right)} \\ &= \sigma\left(\sum_{j \neq i} w_{i,j} x_j\right) \end{aligned}$$



1. Let $d = 3$, we need to define

$$x'_0 \sim p(x_0 | x_1, x_2)$$

$$x'_1 \sim p(x_1 | x'_0, x_2)$$

$$x'_2 \sim p(x_2 | x'_0, x'_1)$$

2. Each dimension is binary, the above 3 models must necessarily return the probability of observing a 1.
3. Note that when we update x_i , we immediately use its new value for sampling other variables x_j .



1. We drive $p(x_0|x_1, x_2)$ using probability of axioms

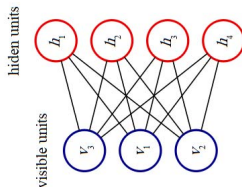
$$\begin{aligned}
 p(x_0 = 1|x_1, x_2) &= \frac{p(x_0 = 1, x_1, x_2)}{p(x_1, x_2)} = \frac{p(x_0 = 1, x_1, x_2)}{\sum_{x_0 \in \{0,1\}} p(x_0, x_1, x_2)} \\
 &= \frac{p(x_0 = 1, x_1, x_2)}{p(x_0 = 0, x_1, x_2) + p(x_0 = 1, x_1, x_2)} \\
 &= \frac{1}{1 + \frac{p(x_0=0, x_1, x_2)}{p(x_0=1, x_1, x_2)}} = \frac{1}{1 + \frac{\exp(H(x_0=0, x_1, x_2))}{\exp(H(x_0=1, x_1, x_2))}} \\
 &= \frac{1}{1 + \exp(H(x_0 = 0, x_1, x_2) - H(x_0 = 1, x_1, x_2)))} \\
 &= \frac{1}{1 + \exp(\sum_{i \neq j} w_{ij} x_i x_j + \sum_i b_i x_i - (\sum_{i \neq j} w_{ij} x_i x_j + \sum_i b_i x_i))} \\
 &= \frac{1}{1 + \exp(-\sum_{j \neq i=0} w_{ij} x_j - b_i)} \\
 &= \sigma \left(\sum_{j \neq i=0} w_{i,j} x_j + b_i \right)
 \end{aligned}$$

Likelihood Based Methods

Restricted Boltzmann Machine (RBM)



1. The **tractability** of the joint distribution is one of the **biggest drawbacks** of BMs.
2. RBMs are a special type of BMs with two layers: **One visible** and **one hidden** layer.



3. The connections in an RBM are **undirected** and the graph is a **bipartite graph**.
4. The probability density is calculated by

$$p(\mathbf{v}, \mathbf{h}) = \frac{1}{Z_\theta} \exp(-E_\theta(\mathbf{v}, \mathbf{h}))$$
$$E_\theta(\mathbf{v}, \mathbf{h}) = -\mathbf{v}^\top \mathbf{W} \mathbf{h} - \mathbf{b}^\top \mathbf{v} - \mathbf{c}^\top \mathbf{h}$$
$$Z_\theta = \sum_{\mathbf{v} \in \{0,1\}^D} \sum_{\mathbf{h} \in \{0,1\}^F} \exp(-E_\theta(\mathbf{v}, \mathbf{h}))$$



1. This bipartite architecture allows us to have more control over the joint distribution.
2. RBMs are a powerful replacement for fully connected BMs when building a deep architecture because of the independence of units within the same layer, which allows for more freedom and flexibility.
3. The [latent variables](#) can act similarly to [hidden units](#) in a MLP.
4. RBMs can be trained using the techniques of [maximum likelihood](#).
5. Sampling from an RBM can be done using [Gibbs sampling](#) method or any other [Markov Chain Monte Carlo \(MCMC\)](#) method.



1. Hidden units are conditionally **independent** given the **visible units** and **vice versa**.

$$p(v_i = 1 | \mathbf{h}) = \sigma \left(\sum_j w_{ij} h_j + b_i \right)$$

$$p(h_j = 1 | \mathbf{v}) = \sigma \left(\sum_i w_{ij} v_i + c_j \right)$$

2. Given **visible \mathbf{v}** , we can sample each **h** independently.
3. Given **hidden \mathbf{h}** , we can sample each **v_j** independently.



1. The model assigns the following probability to a visible vector \mathbf{v}

$$p(\mathbf{v}) = \sum_{\mathbf{h}} p(\mathbf{v}, \mathbf{h})$$

2. The hidden variables can be explicitly marginalized out

$$\begin{aligned} p(\mathbf{v}) &= \frac{1}{Z_{\theta}} \sum_{\mathbf{h}} \exp(-E(\mathbf{v}, \mathbf{h})) \\ &= \frac{1}{Z_{\theta}} \sum_{\mathbf{h}} \exp(\mathbf{v}^{\top} \mathbf{W} \mathbf{h} + \mathbf{b}^{\top} \mathbf{v} + \mathbf{a}^{\top} \mathbf{h}) \\ &= \frac{1}{Z_{\theta}} \exp(\mathbf{b}^{\top} \mathbf{v}) \prod_{j=1}^F \sum_{h_j \in \{0,1\}} \exp(a_j h_j + \sum_i W_{ij} v_i h_j) \\ &= \frac{1}{Z_{\theta}} \exp(\mathbf{b}^{\top} \mathbf{v}) \prod_{j=1}^F \left(1 + \exp(a_j + \sum_i W_{ij} v_i) \right) \end{aligned}$$



1. Bipartite graph structure of RBM has the following property.
2. Conditionals $p(\mathbf{h}|\mathbf{v})$ and $p(\mathbf{v}|\mathbf{h})$ are factorized and easy computed.

$$\begin{aligned}
 p(\mathbf{h}|\mathbf{v}) &= \frac{p(\mathbf{h}, \mathbf{v})}{p(\mathbf{v})} = \frac{1}{p(\mathbf{v})} \frac{1}{Z_\theta} \exp(\mathbf{b}^\top \mathbf{v} + \mathbf{c}^\top \mathbf{h} + \mathbf{v}^\top \mathbf{W} \mathbf{h}) \\
 &= \frac{1}{Z'} \exp(\mathbf{c}^\top \mathbf{h} + \mathbf{v}^\top \mathbf{W} \mathbf{h}) \\
 &= \frac{1}{Z'} \exp\left(\sum_j c_j h_j + \sum_j \mathbf{v}^\top \mathbf{W}_{:j} h_j\right) \\
 &= \frac{1}{Z'} \prod_j \exp(c_j h_j + \mathbf{v}^\top \mathbf{W}_{:j} h_j)
 \end{aligned}$$

3. Normalizing the distributions over individual binary h

$$\begin{aligned}
 p(h_j = 1|\mathbf{v}) &= \frac{\tilde{p}(h_j = 1|\mathbf{v})}{\tilde{p}(h_j = 0|\mathbf{v}) + \tilde{p}(h_j = 1|\mathbf{v})} \\
 &= \frac{\exp(c_j + \mathbf{v}^\top \mathbf{W}_{:j})}{\exp(0) + \exp(c_j + \mathbf{v}^\top \mathbf{W}_{:j})} = \sigma(c_j + \mathbf{v}^\top \mathbf{W}_{:j})
 \end{aligned}$$

4. Similarly

$$p(v_i = 1|\mathbf{h}) = \sigma(c_i + \mathbf{W}_{i:} \mathbf{h})$$



1. Given a set of observations $\{\mathbf{v}_1, \dots, \mathbf{v}_m\}$, the derivative of the log-likelihood with respect to the model parameters is

$$\frac{1}{m} \sum_{k=1}^m \frac{\partial \log p(\mathbf{v}_k)}{\partial w_{ij}} = \mathbb{E}_{p_{data}}[v_i h_j] - \mathbb{E}_{p_\theta}[v_i h_j]$$

$$\frac{1}{m} \sum_{k=1}^m \frac{\partial \log p(\mathbf{v}_k)}{\partial c_j} = \mathbb{E}_{p_{data}}[h_j] - \mathbb{E}_{p_\theta}[h_j]$$

$$\frac{1}{m} \sum_{k=1}^m \frac{\partial \log p(\mathbf{v}_k)}{\partial b_i} = \mathbb{E}_{p_{data}}[v_i] - \mathbb{E}_{p_\theta}[v_i]$$

2. Exact maximum likelihood learning in this model is intractable because exact computation of the expectation of p_θ takes time that is exponential in $\min\{D, F\}$.
3. In practice, learning is done by following an approximation to the gradient of a different objective function, called the **Contrastive Divergence (CD) algorithm**

$$\Delta W = \alpha (\mathbb{E}_{p_{data}}[\mathbf{v}\mathbf{h}^T] - \mathbb{E}_{p_T}[\mathbf{v}\mathbf{h}^T])$$

where α is the learning rate and p_T represents a distribution defined by running a Gibbs chain initialized at the data for T full steps.



- Step 1** Initialize model parameters
- Step 2** Take input vector to the visible nodes
- Step 3** Compute probabilities of hidden nodes
- Step 4** Sample hidden configuration
- Step 5** Reconstruction visible nodes
- Step 6** Update model parameters
- Step 7** Repeat steps 2-6 for multiple iterations



1. For modeling real-valued vectors, we extend RBMs to Gaussian–Bernoulli variant
2. Let $\mathbf{v} \in \mathbb{R}^D$, and $\mathbf{h} \in \{0, 1\}^F$. The energy of the joint state $\{\mathbf{v}, \mathbf{h}\}$ of the Gaussian RBM is defined as follows:

$$E_{\theta}(\mathbf{v}, \mathbf{h}) = \sum_{i=1}^D \frac{(v_i - b_i)^2}{2\sigma_i^2} - \sum_{i=1}^D \sum_{j=1}^F w_{ij} h_j \frac{v_i}{\sigma_i} - \sum_{j=1}^F c_j h_j$$

3. The marginal distribution over the visible vector \mathbf{v} takes the following form:

$$p_{\theta}(\mathbf{v}) = \sum_{\mathbf{h}} \frac{\exp(-E_{\theta}(\mathbf{v}, \mathbf{h}))}{Z_{\theta}}$$

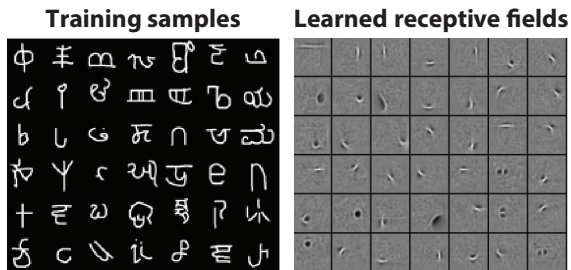
4. The conditional distributions are

$$p(v_i = x \mid \mathbf{h}) = \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left(-\frac{(x - b_i - \sigma_i \sum_j h_j w_{ij})^2}{2\sigma_i^2}\right)$$

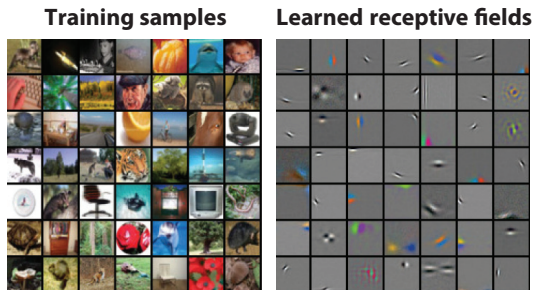
$$p(h_j = 1 \mid \mathbf{v}) = \sigma\left(1 + \sum_i w_{ij} \frac{v_i}{\sigma_i}\right)$$

5. Each visible unit is modeled by a Gaussian distribution, the mean of which is shifted by the weighted combination of the hidden unit activations.

1. The learned receptive fields of Bernoulli–Bernoulli RBM



2. The learned receptive fields of Gaussian–Bernoulli RBM

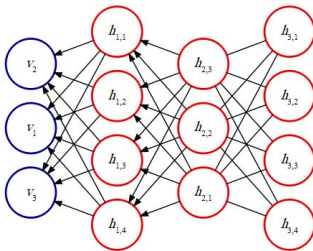


Likelihood Based Methods

Deep Belief Networks



1. A single layer of binary features is not the best way to capture the structure in high-dimensional input data.
2. DBN is a hybrid PGM involving both directed and undirected connections.
3. Deep belief networks consisting of many hidden layers.
 - Connections between top two layers are undirected
 - Connections between all other layers is directed, pointing towards data.



$$p(\mathbf{v}, \mathbf{h}^{(1)}, \mathbf{h}^{(2)}, \dots, \mathbf{h}^{(k)}) = p(\mathbf{v}|\mathbf{h}^{(1)})p(\mathbf{h}^{(1)}|\mathbf{h}^{(2)}) \dots p(\mathbf{h}^{(k-2)}|\mathbf{h}^{(k-1)})p(\mathbf{h}^{(k-1)}, \mathbf{h}^{(k)})$$

4. $p(\mathbf{h}^{(k-1)}, \mathbf{h}^{(k)})$ (the marginal distribution over the top two layers) is an RBM.



1. Now obtain the joint distribution over $\{\mathbf{v}, \mathbf{h}^{(1)}\}$ of the DBN (**drive the following**):

$$\begin{aligned}
 p_{\theta}(\mathbf{v}, \mathbf{h}^{(1)}) &= p_{\theta}(\mathbf{v} \mid \mathbf{h}^{(1)}) \sum_{\mathbf{h}^{(2)}} p_{\theta}(\mathbf{h}^{(1)}, \mathbf{h}^{(2)}) \\
 &= \prod_i p_{\theta}(v_i \mid \mathbf{h}^{(1)}) \times \frac{1}{Z_{\theta}^{(2)}} \prod_i \left(1 + \exp \left(\sum_j w_{ji}^{(2)} h_j^{(1)} \right) \right) \\
 &= \prod_i \frac{\exp \left(v_i \sum_j w_{ij}^{(1)} h_j^{(1)} \right)}{1 + \exp \left(v_i \sum_j w_{ij}^{(1)} h_j^{(1)} \right)} \times \frac{1}{Z_{\theta}^{(2)}} \prod_i \left(1 + \exp \left(\sum_j w_{ji}^{(2)} h_j^{(1)} \right) \right) \\
 &= \frac{1}{Z_{\theta}^{(1)}} \prod_i \exp \left(v_i \sum_j w_{ij}^{(1)} h_j^{(1)} \right) \quad \text{because } w_{ij}^{(1)} = w_{ji}^{(2)} \text{ and } Z_{\theta}^{(1)} = Z_{\theta}^{(2)} \\
 &= \frac{1}{Z_{\theta}^{(1)}} \exp \left(\sum_{ij} w_{ij}^{(1)} v_i h_j^{(1)} \right)
 \end{aligned}$$

which is identical to the joint distribution over $\{\mathbf{v}, \mathbf{h}^{(1)}\}$ defined by an RBM.

weight matrices $\mathbf{W}^{(1)}, \dots, \mathbf{W}^{(k)}$.

2. It contains $k + 1$ bias vectors $\mathbf{b}^{(0)}, \dots, \mathbf{b}^{(1)}$, where $\mathbf{b}^{(0)}$ is bias vector for visible layer.



1. Probability distribution represented by DBN is

$$p(\mathbf{h}^{(k-1)}, \mathbf{h}^{(k)}) \propto \exp\left(\left[\mathbf{b}^{(k)\top} \mathbf{h}^{(k-1)} + \mathbf{b}^{(k-1)\top} \mathbf{h}^{(k)} + \mathbf{h}^{(k-1)\top} \mathbf{W}^{(k)} \mathbf{h}^{(k)}\right]\right)$$
$$p(h_i^{(j)} = 1 | \mathbf{h}^{(j+1)}) = \sigma\left(b_i^{(j)} + \mathbf{W}_{:i}^{(j+1)} \mathbf{h}^{(j+1)}\right)$$
$$p(v_i = 1 | \mathbf{h}^{(1)}) = \sigma\left(b_i^{(0)} + \mathbf{W}_{:i}^{(1)} \mathbf{h}^{(1)}\right)$$

2. For generating a sample from a DBN, do

- Use several Gibbs sampling steps from top two hidden layers.
- Use a single pass of ancestral sampling through rest of model.

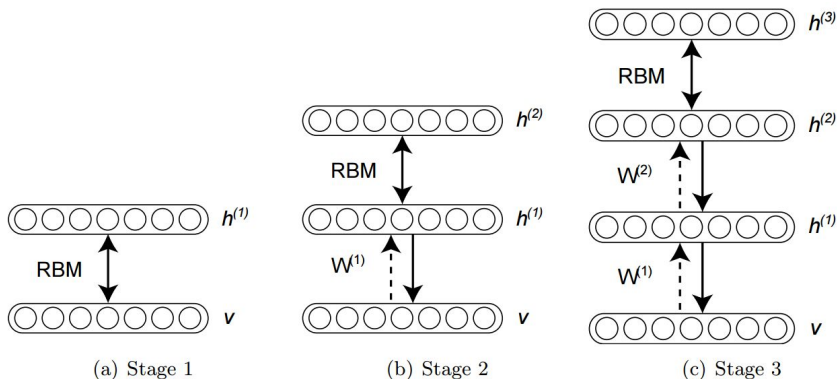


1. Deep belief networks training

1.1 We first train the bottom RBM with parameters $\mathbf{W}^{(1)}$.

1.2 We then initialize the second layer weights to $\mathbf{W}^{(2)} = \mathbf{W}^{(1)}$, ensuring that the two-hidden-layer DBN is at least as good as our original RBM.

1.3 Improve the fit of the DBN to the training data by untying and refitting parameters $\mathbf{W}^{(2)}$.



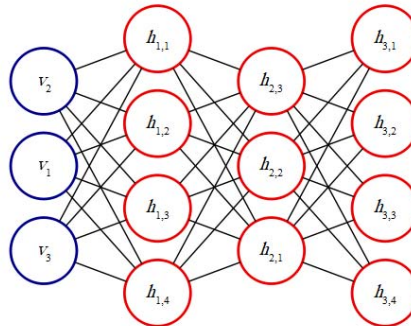
2. Find the variational lower bound of the log-likelihood of the two-hidden-layer DBN.

Likelihood Based Methods

Deep Boltzmann Machine

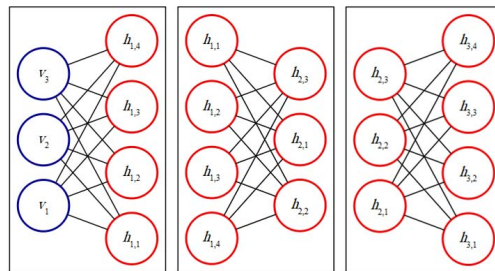
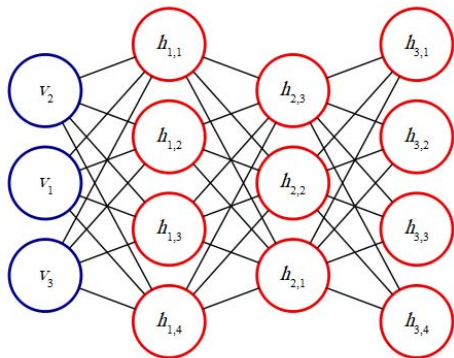


1. DBM is an undirected deep network of **several hidden layers** (Salakhutdinov and Larochelle 2010).
2. Every **unit is connected** to every unit from the **adjacent layers**.
3. There are **no connections** between **units of the same layer**.



4. **Derive the conditional probability of each layer given its above layer.**
5. **Derive derivative of the log-likelihood with respect to the model parameters.**

1. DBMs can also be viewed as a group of RBMs stacked together.

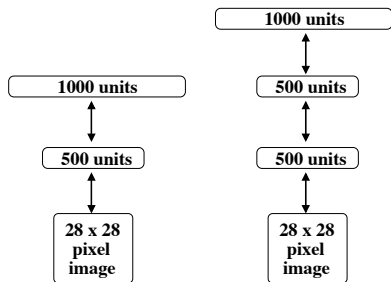


2. Training of DBMs is often done in two stages:

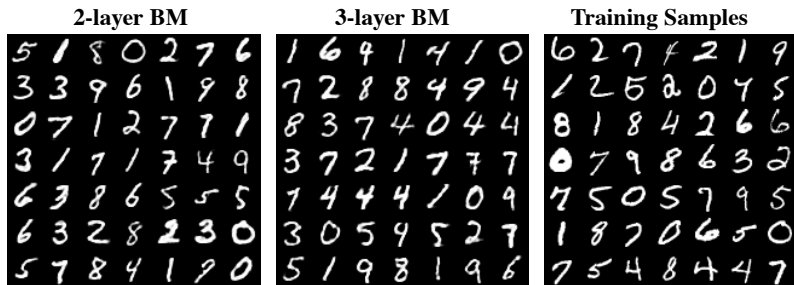
- A **pre-training stage** where every RBM is trained independently.
- a **fine tuning stage** where the network is trained at once using backpropagation.



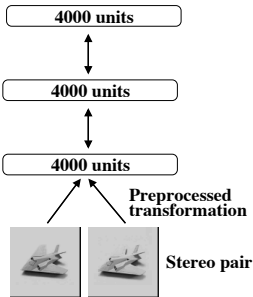
1. Considering two architectures for MNIST dataset.



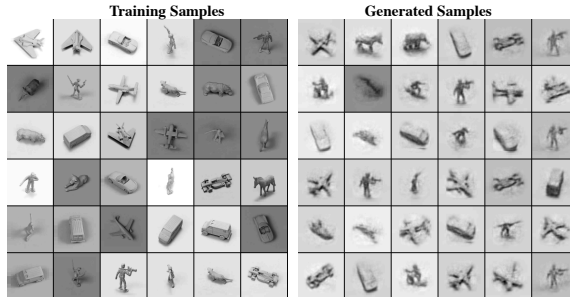
2. The results using Gibbs sampling.



1. Considering the following architecture for NORB dataset.



2. The results using Gibbs sampling.



Score-Based Generative Models



1. The density function given by an EBM is

$$p_{\theta}(\mathbf{x}) = \frac{\exp(-E_{\theta}(\mathbf{x}))}{Z_{\theta}}$$

2. The gradient of negative log-likelihood (NLL) is decomposed to:

$$\nabla_{\theta} \log p_{\theta}(\mathbf{x}) = -\nabla_{\theta} E_{\theta}(\mathbf{x}) - \nabla_{\theta} \log Z_{\theta}$$

3. The first gradient term, $-\nabla_{\theta} E_{\theta}(\mathbf{x})$, is straightforward to evaluate with automatic differentiation, but the exact computation of the second term is intractable.

$$\nabla_{\theta} \log Z_{\theta} = \mathbb{E}_{\mathbf{x} \sim p_{\theta}(\mathbf{x})}[-\nabla_{\theta} E_{\theta}(\mathbf{x})]$$

4. Sampling converges slowly in high dimensional spaces and is thus very expensive, yet we need sampling for each training iteration in contrastive divergence.

5. **The goal is training without sampling**

- Score Matching
- Noise Contrastive Estimation
- Adversarial training



1. Let $f(\mathbf{x})$ and $g(\mathbf{x})$ be two continuously differentiable real-valued functions.
2. If $f(\mathbf{x})$ and $g(\mathbf{x})$ have equal first derivatives everywhere, then $f(\mathbf{x}) = g(\mathbf{x}) + \text{Constant}$.
3. When $f(\mathbf{x})$ and $g(\mathbf{x})$ are log-pdfs with equal first derivatives, the normalization requirement implies that $\int \exp(f(\mathbf{x}))d\mathbf{x} = \int \exp(g(\mathbf{x}))d\mathbf{x} = 1$ and $f(\mathbf{x}) \equiv g(\mathbf{x})$.
4. We can approximately learn an EBM by matching the first derivatives of its log-pdf to the first derivatives of the log-pdf of the data distribution.
5. If they match, then the EBM captures the data distribution exactly.
6. The **first-order gradient function of a log-pdf** is also called **the score of that distribution**.
7. For training EBMs, it is useful to transform the equivalence of distributions to the equivalence of scores, because the score of an EBM can be easily obtained.



1. The score of an EBM can be easily obtained by $\nabla_{\mathbf{x}} \log p_{\theta}(\mathbf{x}) = -\nabla_{\mathbf{x}} E_{\theta}(\mathbf{x})$, which does not involve the typically intractable normalizing constant Z_{θ} .
2. The **basic score matching objective** minimizes a discrepancy between two distributions called the **Fisher divergence**:

$$D_F(p_{data}(\mathbf{x}) \parallel p_{\theta}(\mathbf{x})) = \mathbb{E}_{p_{data}(\mathbf{x})} \left[\frac{1}{2} \|\nabla_{\mathbf{x}} \log p_{data}(\mathbf{x}) - \nabla_{\mathbf{x}} \log p_{\theta}(\mathbf{x})\|^2 \right]$$

3. The first term admits a trivial unbiased Monte Carlo estimator using the empirical mean of samples $\mathbf{x} \sim p_{data}(\mathbf{x})$.
4. The second term is generally impractical to calculate since it requires knowing $\nabla_{\mathbf{x}} \log p_{\theta}(\mathbf{x})$.
5. Under certain regularity conditions, the Fisher divergence can be rewritten using integration by parts, with second derivatives of $E_{\theta}(\mathbf{x})$ replacing the unknown first derivatives of $p_{data}(\mathbf{x})$:

$$D_F(p_{data}(\mathbf{x}) \parallel p_{\theta}(\mathbf{x})) = \mathbb{E}_{p_{data}(\mathbf{x})} \left[\frac{1}{2} \sum_{i=1}^D \left(\frac{\partial E_{\theta}(\mathbf{x})}{\partial x_i} \right)^2 + \left(\frac{\partial^2 E_{\theta}(\mathbf{x})}{\partial x_i^2} \right)^2 \right] + Constant$$

6. Computation of full second derivatives is quadratic in the dimensionality D , thus does not scale to high dimensionality.



1. The **Fisher divergence** can be rewritten as:

$$D_F(p_{data}(\mathbf{x}) \parallel p_{\theta}(\mathbf{x})) = \mathbb{E}_{p_{data}(\mathbf{x})} \left[\frac{1}{2} \sum_{i=1}^D \left(\frac{\partial E_{\theta}(\mathbf{x})}{\partial x_i} \right)^2 + \left(\frac{\partial^2 E_{\theta}(\mathbf{x})}{\partial x_i^2} \right)^2 \right] + \text{Constant}$$

2. SM only requires the trace of the Hessian, but it is still expensive to compute even with modern hardware and automatic differentiation packages.
3. For this reason, the implicit SM formulation has only been applied to relatively simple energy functions where computation of the second derivatives is tractable.
4. Score Matching assumes a continuous data distribution with positive density over the space, but it can be generalized to discrete or bounded data distributions.



1. The Score Matching objective requires several regularity conditions for $\log p_{data}(\mathbf{x})$:
 - it should be continuously differentiable
 - it should be finite everywhere
2. These conditions may not always hold in practice, such as **distribution of gray level of pixels in images**.
3. The distribution of digital images is typically **discrete** and **bounded**.
4. Therefore, $\log p_{data}(\mathbf{x})$ is **discontinuous** and is **negative infinity outside the range**, and therefore **SM is not directly applicable**.
5. To alleviate this difficulty, one can add a bit of noise to each data point: $\tilde{\mathbf{x}} = \mathbf{x} + \epsilon$
6. As long as the noise distribution $p(\epsilon)$ is smooth, the resulting noisy data distribution $q(\tilde{\mathbf{x}}) = \int q(\tilde{\mathbf{x}} | \mathbf{x}) p_{data}(\mathbf{x}) d\mathbf{x}$ is also smooth.
7. Thus the Fisher divergence $D_F(q(\tilde{\mathbf{x}}) || p_{\theta}(\tilde{\mathbf{x}}))$ is a proper objective.



1. It has been shown that the objective with noisy data can be approximated by **the noiseless Score Matching objective plus a regularization term**.
2. This **regularization** makes Score Matching applicable to a wider range of data distributions, but **still requires expensive second-order derivatives**.
3. One elegant and scalable solution to the above difficulty, is to show

$$\begin{aligned} D_F(q(\tilde{\mathbf{x}}) \parallel p_\theta(\tilde{\mathbf{x}})) &= \mathbb{E}_{q(\tilde{\mathbf{x}})} \left[\frac{1}{2} \|\nabla_{\mathbf{x}} \log q(\tilde{\mathbf{x}}) - \nabla_{\mathbf{x}} \log p_\theta(\tilde{\mathbf{x}})\|_2^2 \right] \\ &= \mathbb{E}_{q(\tilde{\mathbf{x}}, \mathbf{x})} \left[\frac{1}{2} \|\nabla_{\mathbf{x}} \log q(\tilde{\mathbf{x}} | \mathbf{x}) - \nabla_{\mathbf{x}} \log p_\theta(\tilde{\mathbf{x}})\|_2^2 \right] + \text{Constant} \end{aligned}$$

4. The above expectation is again approximated by the empirical average of samples, thus completely avoiding both the unknown term $p_{data}(\mathbf{x})$ and **computationally expensive second-order derivatives**.
5. This estimation method is called **Denosing Score Matching (DSM)**.



1. The major drawback of adding noise to data arises when $p_{data}(\mathbf{x})$ is already a **well-behaved distribution** that **satisfies the regularity conditions** required by Score Matching.
2. In this case, $D_F(q(\tilde{\mathbf{x}}) \parallel p_\theta(\tilde{\mathbf{x}})) \neq D_F(p_{data}(\mathbf{x}) \parallel p_\theta(\mathbf{x}))$, and **DSM is not a consistent objective** because the optimal EBM matches **the noisy distribution $q(\tilde{\mathbf{x}})$** not $p_{data}(\mathbf{x})$.
3. This inconsistency becomes **non-negligible** when $q(\tilde{\mathbf{x}})$ significantly differs from $p_{data}(\mathbf{x})$.
4. One way to attenuate the inconsistency of DSM is to choose $q \approx p_{data}$.
5. This often significantly increases the variance of objective values and hinders optimization.
6. For example, suppose $q(\tilde{\mathbf{x}} | \mathbf{x}) = \mathcal{N}(\tilde{\mathbf{x}} | \mathbf{x}, \sigma^2 \mathbf{I})$, where $\sigma \approx 0$. The corresponding DSM objective is

$$D_F(q(\tilde{\mathbf{x}}) \parallel p_\theta(\tilde{\mathbf{x}})) = \mathbb{E}_{p_{data}} \left[\mathbb{E}_{\mathbf{z} \sim \mathcal{N}(0, \mathbf{I})} \left[\frac{1}{2} \left\| \frac{\mathbf{z}}{\sigma} + \nabla_{\mathbf{x}} \log p_\theta(\mathbf{x} + \sigma \mathbf{z}) \right\|_2^2 \right] \right]$$

$$\approx \frac{1}{2m} \sum_{i=1}^m \left\| \frac{\mathbf{z}^{(i)}}{\sigma} + \nabla_{\mathbf{x}} \log p_\theta(\mathbf{x}^{(i)} + \sigma \mathbf{z}^{(i)}) \right\|_2^2$$

where $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$ are some iid samples from p_{data} .



1. When $\sigma \rightarrow 0$, we can leverage Taylor series expansion to rewrite the Monte Carlo estimator to obtain

$$D_F(q(\tilde{\mathbf{x}}) \parallel p_\theta(\tilde{\mathbf{x}})) \approx \frac{1}{2m} \sum_{i=1}^m \left\{ \frac{2}{\sigma} \left(\mathbf{z}^{(i)} \right)^\top \nabla_{\mathbf{x}} \log p_\theta(\mathbf{x}^{(i)}) + \frac{\|\mathbf{z}^{(i)}\|_2^2}{\sigma^2} \right\} + \text{Constant}$$

2. When estimating the above expectation with samples, the variances of $(\mathbf{z}^{(i)})^\top \nabla_{\mathbf{x}} \log p_\theta(\mathbf{x}^{(i)})/\sigma$ and $\frac{\|\mathbf{z}^{(i)}\|_2^2}{\sigma^2}$ will both grow unbounded as $\sigma \rightarrow 0$ due to division by σ and σ^2 .
3. This enlarges the variance of DSM and makes optimization challenging.
4. Some methods were proposed to solve this issue.



1. By adding noise to data, DSM avoids the expensive computation of second-order derivatives.
2. However, DSM does not give a consistent estimator of the data distribution.
3. In order to use score matching for learning deep energy-based models, we have to compute $\|\nabla_{\mathbf{x}} \log p_{\theta}(\mathbf{x})\|_2^2$ and $\text{tr}(\nabla_{\mathbf{x}}^2 \log p_{\theta}(\mathbf{x}))$.
4. Term $\|\nabla_{\mathbf{x}} \log p_{\theta}(\mathbf{x})\|_2^2$ can be computed by one simple backpropagation of $E_{\theta}(\mathbf{x})$.
5. Term $\text{tr}(\nabla_{\mathbf{x}}^2 \log p_{\theta}(\mathbf{x}))$ requires much more number of backpropagations to compute.
6. Computing $\text{tr}(\nabla_{\mathbf{x}}^2 \log p_{\theta}(\mathbf{x}))$ requires a number of backpropagation that is proportional to the data dimension D (Martens, Sutskever, and Swersky 2012).
7. Therefore, score matching is not scalable when learning deep energy-based models on high-dimensional data.
8. **Sliced Score Matching** is an alternative to **Denoising Score Matching** that is **both consistent and computationally efficient** (Song, Garg, et al. 2019).



1. The idea is that **one dimensional data distribution is much easier to estimate** for score matching.
2. Song et. al. proposed to project **the scores onto random directions, such that the vector fields of scores of the data and model distribution become scalar fields** (Song, Garg, et al. 2019).
3. Then comparing the scalar fields to determine how far the model distribution is from the data distribution.
4. **Two vector fields are equivalent if and only if their scalar fields corresponding to projections onto all directions are the same.**
5. Let \mathbf{v} be a random projection direction and $p_{\mathbf{v}}$ as its distribution.
6. The random projected version of Fisher divergence is

$$D_{SF}(p_{data}(\mathbf{x}) \parallel p_{\theta}(\mathbf{x})) = \frac{1}{2} \mathbb{E}_{p_{data}} \left[(\mathbf{v}^T \nabla_{\mathbf{x}} \log p_{data}(\mathbf{x}) - \mathbf{v}^T \nabla_{\mathbf{x}} \log p_{\theta}(\mathbf{x}))^2 \right]$$

called **sliced Fisher divergence**.



1. Unfortunately, sliced Fisher divergence has the same problem as Fisher divergence, due to the unknown data score function $\nabla_{\mathbf{x}} \log p_{data}(\mathbf{x})$.
2. By using integration by parts, we obtain the following tractable alternative form

$$D_{SF}(p_{data}(\mathbf{x}) \parallel p_{\theta}(\mathbf{x})) = \mathbb{E}_{p_{data}} \left[\mathbf{v}^T \nabla_{\mathbf{x}}^2 \log p_{\theta}(\mathbf{x}) \mathbf{v} + \frac{1}{2} (\mathbf{v}^T \nabla_{\mathbf{x}} \log p_{\theta}(\mathbf{x}))^2 \right] + \text{Constant}$$

3. Term $\mathbf{v}^T \nabla_{\mathbf{x}} \log p_{\theta}(\mathbf{x})$ can be computed by one backpropagation for deep energy-based models.
4. Term $\mathbf{v}^T \nabla_{\mathbf{x}}^2 \log p_{\theta}(\mathbf{x}) \mathbf{v}$ involves Hessian, but it is in the form of Hessian-vector products, which can be computed within $O(1)$ backpropagations.
5. Therefore, the computation of sliced score matching does not depend on the dimension of data, and is much more scalable for training deep energy-based models on high dimensional datasets.



1. Instead of **minimizing the Fisher divergence between two vector-valued scores**, SSM randomly samples a projection vector \mathbf{v} , takes the inner product between \mathbf{v} and the two scores, and then compare the resulting two scalars.
2. Sliced Score Matching minimizes the following divergence called the **sliced Fisher divergence**

$$D_{SF}(p_{data}(\mathbf{x}) \parallel p_{\theta}(\mathbf{x})) = \mathbb{E}_{p_{data}} \left[\mathbb{E}_{p_{\mathbf{v}}} \left[\frac{1}{2} \sum_{i=1}^D \left(\frac{\partial E_{\theta}(\mathbf{x})}{\partial x_i} v_i \right)^2 + \sum_{i=1}^D \sum_{j=1}^D \frac{\partial^2 E_{\theta}(\mathbf{x})}{\partial x_i \partial x_j} v_i v_j \right] \right] + \text{Constant}$$

3. All expectations in the above objective can be estimated with empirical means.



1. Let $\{\mathbf{x}_1, \dots, \mathbf{x}_m\}$ be iid samples from the data distribution \mathbf{p}_{data} .
2. For each data point \mathbf{x}_i , randomly draw M random projection directions $\{\mathbf{v}_{i1}, \dots, \mathbf{v}_{iM}\} \sim \mathbf{P}_v$.
3. The sliced score matching objective can be estimated with empirical averages, giving rise to the following finite-sample estimator:

$$\frac{1}{mM} \sum_{i=1}^m \sum_{j=1}^M \left\{ \mathbf{v}_{ij}^T \nabla_{\mathbf{x}}^2 \log p_{\theta}(\mathbf{x}_i) \mathbf{v}_{ij} + \frac{1}{2} \left(\mathbf{v}_{ij}^T \nabla_{\mathbf{x}} \log p_{\theta}(\mathbf{x}_i) \right)^2 \right\}$$

4. Let $\hat{\theta}_{mM}$ be the minimizer of the above empirical estimator, and let θ^* be the true parameter corresponding to the data distribution such that $\mathbf{p}_{\theta^*} = \mathbf{p}_{data}$.
5. It has been shown that under some regularity conditions, $\hat{\theta}_{mM}$ is consistent and asymptotically normal.
6. Formally, for any $M \in \mathbb{N}^+$, when $m \rightarrow \infty$, we have

$$\hat{\theta}_{mM} \xrightarrow{P} \theta^*$$

$$\sqrt{m} \left(\hat{\theta}_{mM} - \theta^* \right) \xrightarrow{d} \mathcal{N}(0, \Sigma)$$

where Σ is some covariance matrix.

Noise Contrastive Estimation based Methods



1. The core idea behind NCE is to distinguish data samples from a dataset (signal) from artificially generated noise samples.
2. This is achieved by training a **binary classifier** that learns to classify whether a given sample comes from the actual data distribution or from a noise distribution.
3. The classifier implicitly learns the parameters of the data distribution.
4. NCE involves the following steps:
 - 4.1 A noise distribution is chosen, which should ideally be simple enough to sample from and calculate probabilities.
 - 4.2 Noise samples are generated from this noise distribution.
 - 4.3 A logistic regression model is trained to discriminate between samples from the true data distribution and the noise samples.
 - 4.4 The parameters learned by the logistic regression model are then used as estimates for the parameters of the true data distribution.



1. Advantages of NCE over MLE methods
 - 1.1 **Computational Efficiency:** NCE avoids the computation of the partition function, which can be intractable for large models.
 - 1.2 **Scalability:** NCE scales well with the size of the dataset and the complexity of the model.
 - 1.3 **Flexibility:** NCE can be applied to a wide range of models, including those where MLE is not feasible.
2. Challenges and considerations of using NCE
 - 2.1 **Choice of Noise Distribution:** The performance of NCE is sensitive to the choice of noise distribution. A poor choice can lead to suboptimal parameter estimation.
 - 2.2 **Hyperparameter Tuning:** NCE requires careful tuning of hyperparameters, including the number of noise samples and the learning rate for the classifier.
 - 2.3 **Convergence:** Ensuring convergence of the estimation process can be challenging, especially for complex models with many parameters.



1. NCE is based on the idea that we can learn an Energy-Based Model by contrasting it with another distribution with known density.
2. Let $p_{data}(\mathbf{x})$ be the data distribution, and let $p_n(\mathbf{x})$ be a chosen distribution with known density, called a noise distribution.
3. This noise distribution is usually simple and has a tractable pdf, like $\mathcal{N}(0, \mathbf{I})$, such that we can compute the pdf and generate samples from it efficiently.
4. Let y be a binary variable with Bernoulli distribution, which we use to define a mixture distribution of noise and data:

$$p_{n,data}(\mathbf{x}) = p(y=0) p_n(\mathbf{x}) + p(y=1) p_{data}(\mathbf{x})$$

5. Based on the Bayes' rule, given a sample \mathbf{x} from this mixture, the posterior probability of $y=0$ is

$$p_{n,data}(y=0 | \mathbf{x}) = \frac{p_{n,data}(\mathbf{x} | y=0) p(y)}{p_{n,data}(\mathbf{x})} = \frac{p_n(\mathbf{x})}{p_n(\mathbf{x}) + \alpha p_{data}(\mathbf{x})} \quad \text{Drive this equation.}$$

where $\alpha = \frac{p(y=1)}{p(y=0)}$.



1. Let our energy-based model has the following form:

$$p_{\theta}(\mathbf{x}) = \frac{\exp(-E_{\theta}(\mathbf{x}))}{Z_{\theta}}$$

2. Unlike other EBMs, Z_{θ} is treated as a learnable (scalar) parameter in NCE.
3. Given this model, we can define a mixture of noise and the model distribution:

$$p_{n,\theta}(\mathbf{x}) = p(y=0) p_n(\mathbf{x}) + p(y=1) p_{\theta}(\mathbf{x})$$

4. The posterior probability of $y=0$ given this noise/model mixture is

$$p_{n,\theta}(y=0 | \mathbf{x}) = \frac{p_n(\mathbf{x})}{p_n(\mathbf{x}) + \alpha p_{\theta}(\mathbf{x})}$$



1. In NCE, we indirectly fit $p_{\theta}(\mathbf{x})$ to $p_{data}(\mathbf{x})$ by fitting $p_{n,\theta}(y | \mathbf{x})$ to $p_{n,data}(y | \mathbf{x})$ through a standard conditional maximum likelihood objective:

$$\begin{aligned}\theta^* &= \arg \min_{\theta} \mathbb{E}_{p_{n,data}(\mathbf{x},y)} [D_{KL}(p_{n,data}(y | \mathbf{x}) || p_{n,\theta}(y | \mathbf{x}))] \\ &= \arg \max_{\theta} \mathbb{E}_{p_{n,data}} [\log p_{n,\theta}(y | \mathbf{x})]\end{aligned}$$

Derive this objective function

2. This optimization problem can be solved using stochastic gradient ascent.
3. Like any other deep classifier, when the model is sufficiently powerful, $p_{n,\theta^*}(y | \mathbf{x})$ will match $p_{n,data}(y | \mathbf{x})$ at the optimum.

$$\begin{aligned}p_{n,\theta^*}(y = 0 | \mathbf{x}) &= p_{n,data}(y = 0 | \mathbf{x}) \\ \iff \frac{p_n(\mathbf{x})}{p_n(\mathbf{x}) + \alpha p_{n,\theta^*}(\mathbf{x})} &= \frac{p_n(\mathbf{x})}{p_n(\mathbf{x}) + \alpha p_{n,data}(\mathbf{x})} \\ \iff p_{\theta^*}(\mathbf{x}) &= p_{data}(\mathbf{x})\end{aligned}$$

4. Consequently, $E_{\theta^*}(\mathbf{x})$ is an unnormalized energy function that matches the data distribution $p_{data}(\mathbf{x})$, and Z_{θ^*} is the corresponding normalizing constant.

Adversarial training



1. When training EBMs with MLE, we need to sample from the EBM per training iteration.
2. Sampling using multiple MCMC steps is expensive and requires careful tuning of the Markov chain.
3. One way to avoid this difficulty is to use non-MLE methods that do not need sampling, such as Score Matching and Noise Contrastive Estimation.
4. We can sidestep costly MCMC sampling by learning an auxiliary model through adversarial training, which allows fast sampling.
5. From the definition of EBMs, we can rewrite the maximum likelihood objective by introducing a variational distribution $q_\phi(\mathbf{x})$ parameterized by ϕ :

$$\begin{aligned}
 \mathbb{E}_{p_{data}(\mathbf{x})}[\log p_\theta(\mathbf{x})] &= \mathbb{E}_{p_{data}(\mathbf{x})}[-E_\theta(\mathbf{x})] - \log Z_\theta \\
 &= \mathbb{E}_{p_{data}(\mathbf{x})}[-E_\theta(\mathbf{x})] - \log \int \exp(-E_\theta(\mathbf{x})) \\
 &= \mathbb{E}_{p_{data}(\mathbf{x})}[-E_\theta(\mathbf{x})] - \log \int \exp(-E_\theta(\mathbf{x})) \frac{q_\phi(\mathbf{x})}{q_\phi(\mathbf{x})} \\
 &\leq \mathbb{E}_{p_{data}(\mathbf{x})}[-E_\theta(\mathbf{x})] - \int \exp(-E_\theta(\mathbf{x})) \frac{q_\phi(\mathbf{x})}{q_\phi(\mathbf{x})} \\
 &= \mathbb{E}_{p_{data}(\mathbf{x})}[-E_\theta(\mathbf{x})] - \mathbb{E}_{q_\phi(\mathbf{x})}[-E_\theta(\mathbf{x})] - H(q_\phi(\mathbf{x}))
 \end{aligned}$$

Using Jensen inequality



1. The upperbound of $\mathbb{E}_{p_{data}(\mathbf{x})}[\log p_{\theta}(\mathbf{x})]$ is

$$\mathbb{E}_{p_{data}(\mathbf{x})}[\log p_{\theta}(\mathbf{x})] \leq \mathbb{E}_{p_{data}(\mathbf{x})}[-E_{\theta}(\mathbf{x})] - \mathbb{E}_{q_{\phi}(\mathbf{x})}[-E_{\theta}(\mathbf{x})] - H(q_{\phi}(\mathbf{x}))$$

2. For EBM training,

- First minimize the upper bound with respect to $q_{\phi}(\mathbf{x})$ so that it is closer to the likelihood objective.
- Then maximize with respect to $E_{\theta}(\mathbf{x})$ as a surrogate for maximizing likelihood.

3. This amounts to using the following **maximin objective**

$$\max_{\theta} \min_{\phi} \mathbb{E}_{q_{\phi}(\mathbf{x})}[E_{\theta}(\mathbf{x})] - \mathbb{E}_{p_{data}(\mathbf{x})}[E_{\theta}(\mathbf{x})] - H(q_{\phi}(\mathbf{x}))$$

4. Optimizing the above objective is similar to training GANs and can be achieved by adversarial training.
5. The variational distribution $q_{\phi}(\mathbf{x})$ should allow both fast sampling and efficient entropy evaluation to **make the maximin objective function tractable**.

Hybrid Modeling



1. Consider using deep generative modeling in the context of finding the joint distribution over observables and decision variables that is factorized as

$$p(\mathbf{x}, y) = p(y | \mathbf{x}) p(\mathbf{x})$$

where $\mathbf{x} \in \mathbb{R}^D$ and $y \in \{0, 1, \dots, K - 1\}$.

2. By taking the logarithm of the joint we obtain two additive components:

$$\log p(\mathbf{x}, y) = \log p(y | \mathbf{x}) + \log p(\mathbf{x})$$

3. How can we model the above problem using EBMs?
4. Let $E_\theta(\mathbf{x}, y)$ be parameterized by a neural network $NN_\theta(\mathbf{x})$ where its input is \mathbf{x} and returns K values: $NN_\theta : \mathbb{R}^D \mapsto \mathbb{R}^K$.
5. This means that we can define energy function as

$$E_\theta(\mathbf{x}, y) = -NN_\theta(\mathbf{x})[y]$$

where $[y]$ denotes the specific output of the neural networks $NN_\theta(\mathbf{x})$.



1. Then, the joint probability distribution is defined as

$$\begin{aligned} p(\mathbf{x}, y) &= \frac{\exp(-E_{\theta}(\mathbf{x}, y))}{Z_{\theta}} \\ &= \frac{\exp(NN_{\theta}(\mathbf{x})[y])}{Z_{\theta}} \end{aligned}$$

2. The marginal distribution $p(\mathbf{x})$ is

$$\begin{aligned} p_{\theta}(\mathbf{x}) &= \sum_y p(\mathbf{x}, y) \\ &= \frac{\sum_y \exp(NN_{\theta}(\mathbf{x})[y])}{Z_{\theta}} \end{aligned}$$

3. We can re-write the numerator in the following manner:

$$\begin{aligned} \sum_y \exp * NN_{\theta}(\mathbf{x})[y] &= \exp\left(\log\left\{\sum_y \exp(NN_{\theta}(\mathbf{x})[y])\right\}\right) \\ &= \exp(\text{LogSumExp}_y(NN_{\theta}(\mathbf{x})[y])) \end{aligned}$$

4. We can say that the **energy function of the marginal distribution** is expressed as $-\text{LogSumExp}_y(NN_{\theta}(\mathbf{x})[y])$.



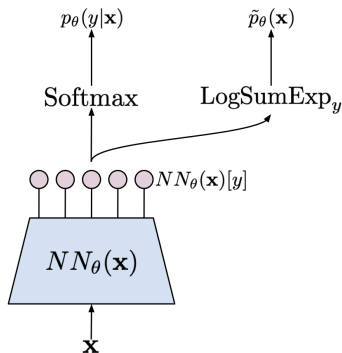
1. The conditional distribution $p_\theta(y | \mathbf{x})$ is

$$\begin{aligned}
 p_\theta(y | \mathbf{x}) &= \frac{p_\theta(\mathbf{x}, y)}{p_\theta(\mathbf{x})} \\
 &= \frac{\frac{\exp(NN_\theta(\mathbf{x})[y])}{Z_\theta}}{\frac{\sum_y \exp(NN_\theta(\mathbf{x})[y])}{Z_\theta}} \\
 &= \frac{\exp(NN_\theta(\mathbf{x})[y])}{\sum_y \exp(NN_\theta(\mathbf{x})[y])}.
 \end{aligned}$$

2. This means that the energy-based model could be used either as a classifier or a marginal distribution.
3. Any any classifier could be seen as an energy-based model (Grathwohl et al. 2020).
4. The logarithm of the joint distribution is

$$\begin{aligned}
 \log p_\theta(\mathbf{x}, y) &= \log \frac{\exp(f_\theta(\mathbf{x})[y])}{\sum_y \exp(NN_\theta(\mathbf{x})[y])} + \log \frac{\sum_y \exp(NN_\theta(\mathbf{x})[y])}{Z_\theta} \\
 &= \log \text{Softmax}(NN_\theta(\mathbf{x})[y]) + (\text{LogSumExp}_y(NN_\theta(\mathbf{x})[y]) - \log Z_\theta)
 \end{aligned}$$

1. The model requires a shared neural network that is used for calculating both distributions.



2. We have a single neural network to train and the **training objective is the logarithm of the joint distribution**.
3. The training objective is a sum of the logarithm of the conditional $p_{\theta}(y | \mathbf{x})$ and the logarithm of the marginal $p_{\theta}(\mathbf{x})$.
4. Calculating the gradient with respect to the parameters θ requires taking the gradient of each of the component separately (**Derive the weight update equations**).

Summary











1. Both Variational Autoencoders and EBM learn the parameters by maximizing the (marginal) log-likelihood, which can be interpreted also as the minimization of $D_{KL}(p_{data} || p_{\theta})$.
2. VAEs are intrinsically latent variable models imposing an information bottleneck and approximating the posterior on the latent variables $p_{data}(z | x)p(z \rightarrow x)$ through variational inference, whereas EBMs generally are not.
3. EBMs can easily extended to latent variable models (Xiao, Yan, and Amit 2020).
4. Che et. al. showed that GANs can be better understood through the lens of EBM (Che et al. 2020).
5. They showed that GAN generators and discriminators collaboratively learn an **implicit energy-based model**.

References






1. Paper [Learning Deep Generative Models](#) (Salakhutdinov 2015).
2. Chapter [A Tutorial on Energy-Based Learning](#) (Lecun et al. 2006).
3. Paper [How to Train Your Energy-Based Models](#) (Song and Kingma 2021).
4. Chapter [23 of Probabilistic Machine Learning: Advanced Topics](#) (Murphy 2023).
5. Chapter [3 of Deep Generative Modeling](#) (Tomczak 2022).



-  Che, Tong et al. (2020). “Your GAN is Secretly an Energy-based Model and You Should Use Discriminator Driven Latent Sampling”. In: *Advances in Neural Information Processing Systems*.
-  Grathwohl, Will et al. (2020). “Your classifier is secretly an energy based model and you should treat it like one”. In: *International Conference on Learning Representations*.
-  Lecun, Yann et al. (2006). “A tutorial on energy-based learning”. In: *Predicting structured data*. Ed. by G. Bakir et al. MIT Press.
-  Martens, James, Ilya Sutskever, and Kevin Swersky (2012). “Estimating the Hessian by Back-propagating Curvature”. In: *International Conference on Machine Learning*.
-  Murphy, Kevin P. (2023). *Probabilistic Machine Learning: Advanced Topics*. The MIT Press.
-  Salakhutdinov, Ruslan (2015). “Learning Deep Generative Models”. In: *Annual Review of Statistics and Its Application* 2, pp. 361–385.
-  Salakhutdinov, Ruslan and Hugo Larochelle (2010). “Efficient Learning of Deep Boltzmann Machines”. In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS)*. Vol. 9, pp. 693–700.
-  Song, Yang, Sahaj Garg, et al. (2019). “Sliced Score Matching: A Scalable Approach to Density and Score Estimation”. In: *Uncertainty in Artificial Intelligence*. Vol. 115, pp. 574–584.



-  Song, Yang and Diederik P. Kingma (2021). “How to Train Your Energy-Based Models”. In: *CoRR* abs/2101.03288.
-  Tomczak, Jakub M. (2022). *Deep Generative Modeling*. Springer.
-  Xiao, Zhisheng, Qing Yan, and Yali Amit (2020). “Exponential Tilting of Generative Models: Improving Sample Quality by Training and Sampling from Latent Energy”. In: *CoRR* abs/2006.08100.

Questions?