Course Materials for GEN-AI

Northeastern University

These materials have been prepared and sourced for the course **GEN-AI** at Northeastern University. Every effort has been made to provide proper citations and credit for all referenced works.

If you believe any material has been inadequately cited or requires correction, please contact me at:

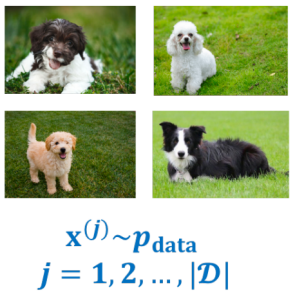
Instructor: Ramin Mohammadi
r.mohammadi@northeastern.edu

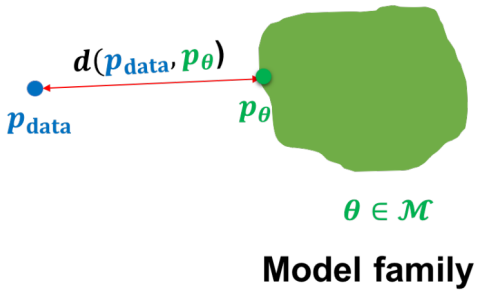
Thank you for your understanding and collaboration.

Content sourced from multiple references - see Reference section for details

Energy Based Models

1 Introduction





Model Families

- **Autoregressive Models:**
$$p_G(x) = \prod_{i=1}^{n} p_G(x_i \mid x_{<i})$$
- **Variational Autoencoders:**
$$p_G(x) = \int p_G(x, z)\, dz$$
- **Normalizing Flow Models:**
$$p_X(x; \theta) = p_Z\left(f_\theta^{-1}(x)\right) \left| \det \left(\frac{\partial f_\theta^{-1}(x)}{\partial x} \right) \right|$$

All the above families are trained by minimizing KL divergence $D_{KL}(p_{\text{data}} \| p_G)$ or equivalently maximizing likelihoods (or approximations).

Additionally we saw that we could learn this using some sort of likelihood free models like GANs:

- **GAN**
$$\min_G \max_D V(G, D) = \mathbb{E}_{x \sim p_{\text{data}}}[\log D(x)] + \mathbb{E}_{z \sim p_z}[\log(1 - D(G(z)))].$$
- **Two sample tests** Which can approximatly optmize f−divergence and Wasserstein distance.

Very flexible model architectures. But likelihood is intractable, training is unstable, hard to evaluate, and has mode collapse issues.

Next, we will define another model family with a different approach for parametrization of probabilistic models called Energy based models which:

- Very flexible model architectures. lifts all the restrictions on kind of NNs we can use to define a valid probability mass function.
- Stable training as we are using likelihood based model and not just sampling procedure.
- Relatively high sample quality and closely related to **Diffusion models**.
- Flexible composition which allows you combine different types of generative models.

1.1 Parametrizing probability distribution - definition

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

Properties:

1. Non-negative: $p(\theta) \geq 0$ - easy
2. Sum-to-one: $\sum_{\theta} p(v) = 1$ (or $\int p(\theta)\, d\theta = 1$ for continuous variables) - Normalized constraints (more difficult)

Coming up with a non-negative function $p_\theta(x)$ is not hard. Given any function $f_\theta(x)$, we can choose:

- $g_\theta(x) = f_\theta(x)^2$
- $g_\theta(x) = \exp(f_\theta(x))$
- $g_\theta(x) = |f_\theta(x)|$
- $g_\theta(x) = \log(1 + \exp(f_\theta(x)))$
- etc.

Sum-to-one is key: As using NNs as our function for $p(\theta)$ which is tricky to enforce the results to sum to 1.

Total "volume" is fixed: increasing $p(x_{\text{train}})$ guarantees that x_{train} becomes relatively more likely (compared to the rest).

the analogy is that there is a cake and you can divide it in any sort of ways but if you make one slice bigger the other slices becomes smaller. So we need to guarantee when we increasing the probability of one data, that automatically reducing he probability of other things smaller.

Problem:

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

For example a model like autoregressive irrespective of value of θ will have the total mass probability as 1.

1.2 Parameterizing Probability Distributions - solution

Problem: $g_\theta(x)$ might not be normalized.

Solution:

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

Then by definition, $\int p_\theta(x)dx = \int \frac{g_\theta(x)}{Z(\theta)}dx = \frac{Z(\theta)}{Z(\theta)} = 1$.

Example: Choose $g_\theta(x)$ so that we know the volume *analytically* as a function of θ, this means to select from functions that the integral has a closed form solution .

1. $g_{(\mu,\sigma)}(x) = e^{-\frac{(x-\mu)^2}{2\sigma^2}}$. **Volume:** $\int e^{-\frac{x-\mu}{2\sigma^2}}dx = \sqrt{2\pi\sigma^2}$. $\rightarrow \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$ *Gaussian dist.*
2. $g_\lambda(x) = e^{-\lambda x}$. **Volume:** $\int_0^{+\infty} e^{-\lambda x}dx = \frac{1}{\lambda}$. $\rightarrow \lambda e^{-\lambda x}$ *Exponential dist.*
3. $g_\theta(x) = h(x)\exp\{\theta \cdot T(x)\}$. **Volume:** $\exp\{A(\theta)\}$, where $A(\theta) = \log \int h(x)\exp\{\theta \cdot T(x)\}dx$. $\rightarrow$ *Exponential family.*

Examples of the Exponential Family:

- Normal, Poisson, Exponential, Bernoulli
- Beta, Gamma, Dirichlet, Wishart, etc.

Note: Function forms $g_\theta(x)$ need to allow *analytical integration.* Despite being restrictive, they are very useful as building blocks for more complex distributions.

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

For example:

1. **Autoregressive:** Products of normalized objects $p_\theta(x)p_{\theta'}(y|x)$ is also normalized:
   - In some sense an autoregressive model is a way of defining the joint probability distribution / joint density function that is normalized by design because its a product of conditionals that are normalized.
   - Here, first object is parametrized by θ and the second object is parametrized by θ' which is function over x. For every θ' the distribution which we get over y is normalized. the full object which we get by multiplying them together is guaranteed to be normalized.

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

2. **Mixture Models:** Mixtures of normalized objects $\alpha p_\theta(x) + (1-\alpha)p_{\theta'}(x)$:
   - We can look at latent variable models as a way for combining normalized objects and building a more complicated one that is guaranteed to be normalized.

$$\int_x \alpha p_\theta(x) + (1-\alpha)p_{\theta'}(x)dx = \alpha + (1-\alpha) = 1$$

In above examples, this is possible due to the fact the over functions/objects are normalized by default so their product or mixture is also normalized. How about using models where the "volume"/normalization constant of $g_\theta(x)$ is not easy to compute analytically?

Energy-Based Models

To address cases where the normalization constant $Z(\theta)$ is difficult or impossible to compute analytically, we turn to energy-based models, which define probabilities in terms of **unnormalized energy functions** $f_\theta(x)$. This framework allows for great flexibility while relying on a partition function to ensure proper normalization.

$$p_\theta(x) = \underbrace{\frac{1}{\int \exp(f_\theta(x))dx}}_{\text{normalized by exp}} \underbrace{\exp(\underbrace{f_\theta(x)}_{unnormalized})}_{\text{Object normalized}} = \frac{1}{Z(\theta)} \exp(f_\theta(x))$$

The volume/normalization constant $Z(\theta)$, also called the **partition function**, is defined as:

$$Z(\theta) = \int \exp(f_\theta(x))dx = \int \exp(-E_\theta(x))dx$$

Softmax is an example of energy-based model.

Why exponential (and not, for example, $f_\theta(x)^2$)?

1. **Capture large variations in probability:** Log-probability is the natural scale to work with when representing probability distributions. Using $\exp(f_\theta(x))$ enables the model to handle very large variations in probability, which would otherwise require a highly non-smooth $f_\theta(x)$.
2. **Connection to exponential families:** Many common probability distributions (e.g., Gaussian, Bernoulli, Poisson) belong to the exponential family, which can be expressed in this form. Energy-based models generalize this framework.
3. **Rooted in statistical physics:** These distributions arise naturally under principles such as *maximum entropy* and the *second law of thermodynamics*. The term $-f_\theta(x)$ is referred to as the **energy**, hence the name *energy-based models*. Intuitively, configurations x with lower energy (i.e., higher $f_\theta(x)$) are more likely, aligning with physical principles.

Pros and Cons of Energy-Based Models

Pros:

1. Can plug in pretty much any function $f_\theta(x)$ you want.

Cons (lots of them):

1. Sampling from $p_\theta(x)$ is hard.
   - **Complex energy landscape:** High-dimensional distributions with multiple modes make efficient sampling challenging.
   - **Intractable partition function:** $Z(\theta) = \int \exp(f_\theta(x))dx$ is often computationally infeasible, complicating direct sampling.
2. Evaluating and optimizing the likelihood $p_\theta(x)$ is hard (learning is hard).
3. No feature learning in vanilla form(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 with the number of dimensions of x. Nevertheless, some tasks do not require knowing $Z(\theta)$.

2 Applications of Energy Based models

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

There are certain tasks which do not require to know the partition function, for example:

Example 1: Given x, x', evaluating $p_\theta(x)$ or $p_\theta(x')$ requires $Z(\theta)$.

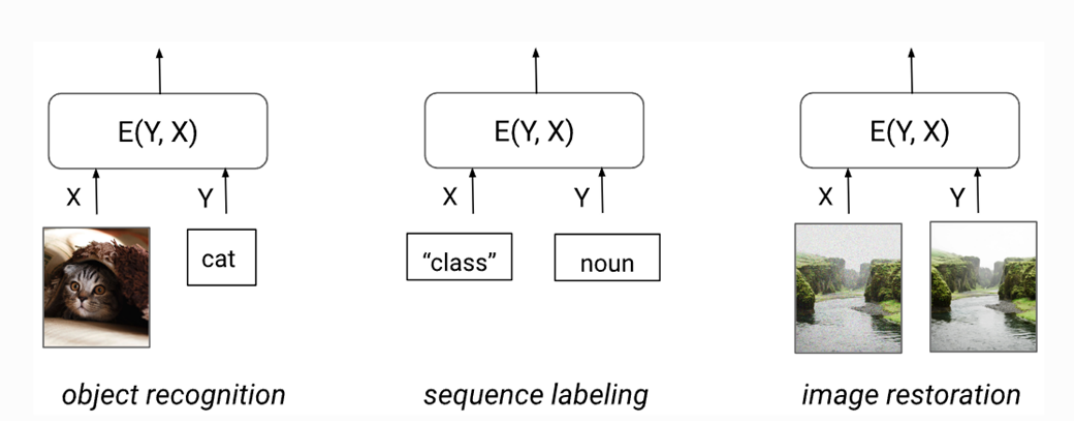
- However, their ratio does not involve $Z(\theta)$:

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

- Using the pie analogy, we can get their relative sizes of two slices without knowing the actual size the pie.
- This means we can easily check which one is more likely without knowing how likely it is.

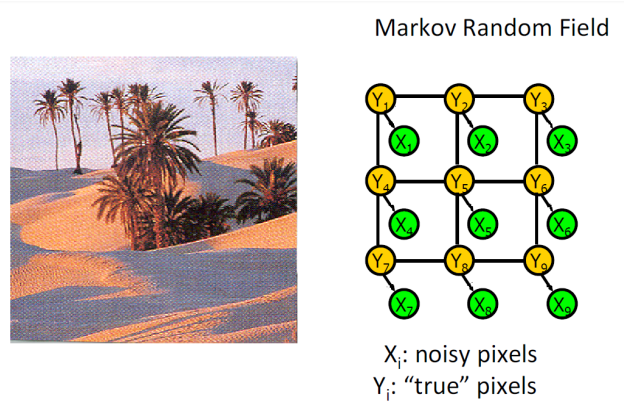
Which is useful for tasks like: **anomaly detection, denoising**

Example 2:



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

Example 3 - 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 $\mathbf{y}$.
- **Markov Random Field:**

- $\mathbf{x}_i$: noisy pixels
- $\mathbf{y}_i$: "true" pixels

- We model the joint probability distribution $p(\mathbf{y}, \mathbf{x})$ which is an energy based model defined as:

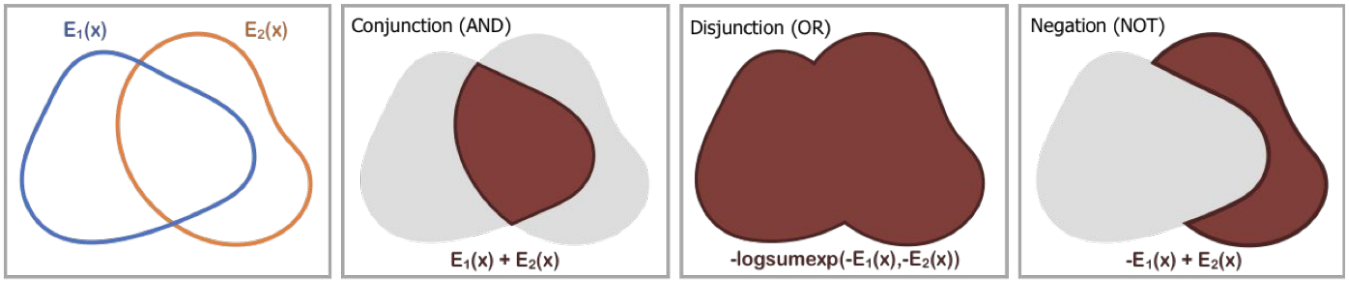
$$p(\mathbf{y}, \mathbf{x}) = \frac{1}{Z} \exp\left(\sum_i \psi_i(x_i, y_i) + \sum_{(i,j)\in E} \psi_{ij}(y_i, y_j)\right)$$

  where:

  - $\psi_i(x_i, y_i)$: Function to determine relationship between the i-th corrupted pixel and the i-th original pixel.
    - Here we expect the corrupted pixel to be fairly similar to the original one.
  - $\psi_{ij}(y_i, y_j)$: Function that defines some sort of prior that neighboring pixels tend to have the same value.
    - generates local interaction between all other pixels that are close to each other in the image.

- **Goal:** Recover the original image $\mathbf{y}$.
- **Solution:** Maximize $p(\mathbf{y}|\mathbf{x})$ by finding the best y. Although the $p(y, x)$ depends on the normalization constant Z, it basically doesn't matter for the optimization which is simply a comparison between x and any y as it is constant for all y.
- However, during the training we still need to know Z

Example 4 - Products 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 (e.g., 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 independently, it is tempting to ensemble them as:

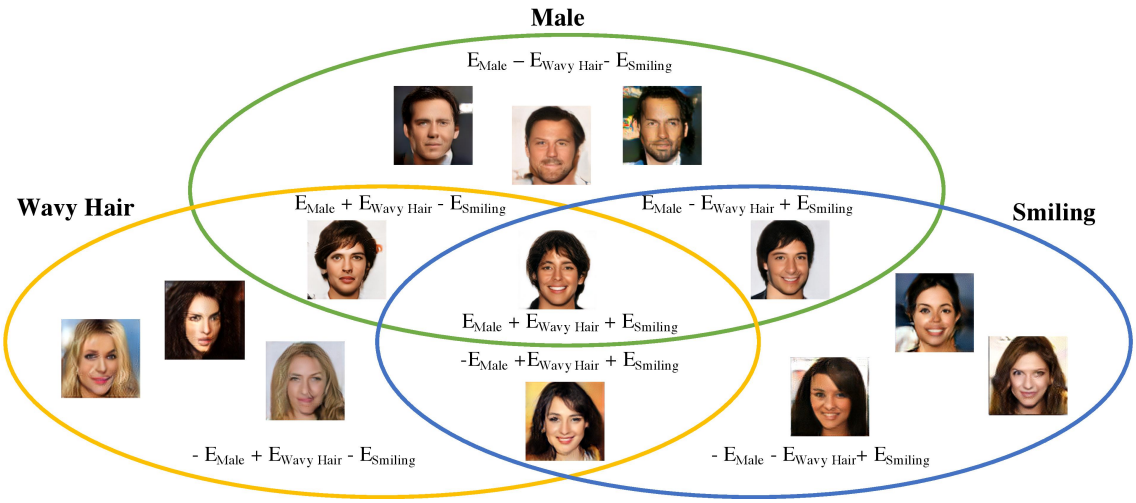
$$p_{\theta_1}(\mathbf{x}) q_{\theta_2}(\mathbf{x}) r_{\theta_3}(\mathbf{x}).$$

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

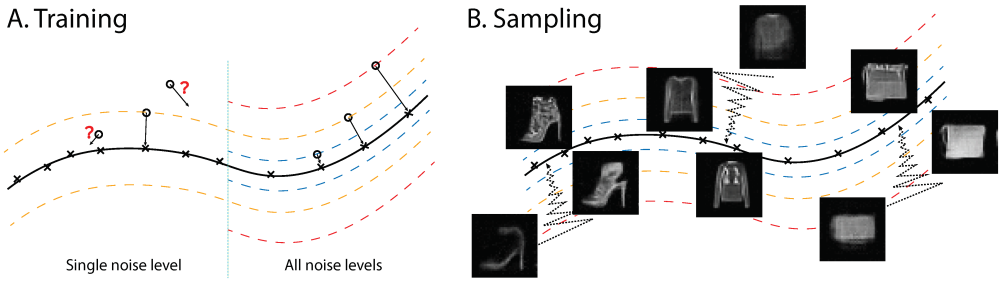
$$p_{\theta_1,\theta_2,\theta_3}(\mathbf{x}) = \frac{1}{Z(\theta_1, \theta_2, \theta_3)} q_{\theta_1}(\mathbf{x}) r_{\theta_2}(\mathbf{x}) t_{\theta_3}(\mathbf{x}),$$

where $Z(\theta_1, \theta_2, \theta_3)$ is the normalization constant.

Note: This behaves similarly to an AND operation (e.g., the probability is zero as long as one model gives zero probability), unlike mixture models $(\alpha q_{\theta_1}(\mathbf{x}) + (1-\alpha) r_{\theta_2}(\mathbf{x}))$ which behave more like an OR operation.



By combining the above operators, we can controllably generate images with complex relationships. For example, given EBMs trained on male, smiling, and black haired faces, through combinations of negation, disjunction and conjunction, we can selectively generate images in a Venn diagram as shown below:

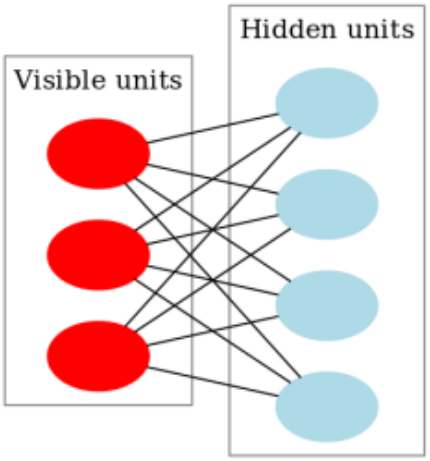


We can similarly train separate EBMs on the attributes of young, female, smiling, and wavy hair. Through conjunction on each model sequentially, we are able to generate successively more refined versions of human faces. Surprisingly, we find that generations of our model are able to become increasingly more refined by adding more models.

Example 5: Restricted Boltzman Machine:

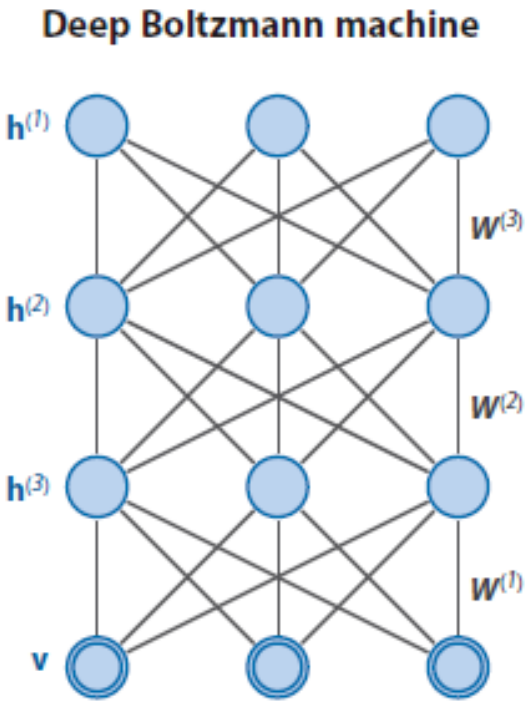
- **Two types of variables:**
  1. $\mathbf{x} \in \{0,1\}^n$ are visible variables (e.g., pixel values).
  2. $\mathbf{z} \in \{0,1\}^m$ are latent variables.
- **The joint distribution is:**

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



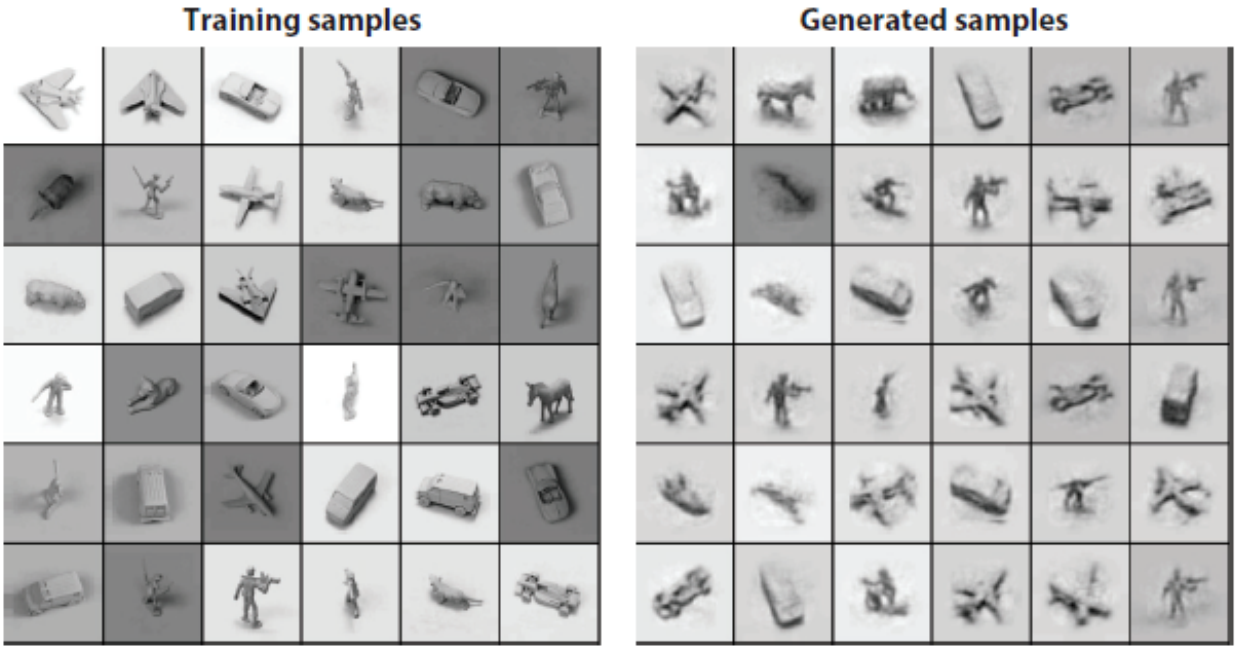
- **Restricted Boltzmann Machine (RBM):** Restricted because there are no visible-visible and hidden-hidden connections, i.e., no $x_i x_j$ or $z_i z_j$ terms in the objective.

Application of RBM:



Stacked RBMs are one of the first deep generative models:

- Bottom layer variables **v** represent pixel values.
- Layers above (**h**) represent *higher-level features* (e.g., corners, edges, etc.).
- Early deep neural networks for *supervised learning* required pre-training using stacked RBMs to make them work.



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:

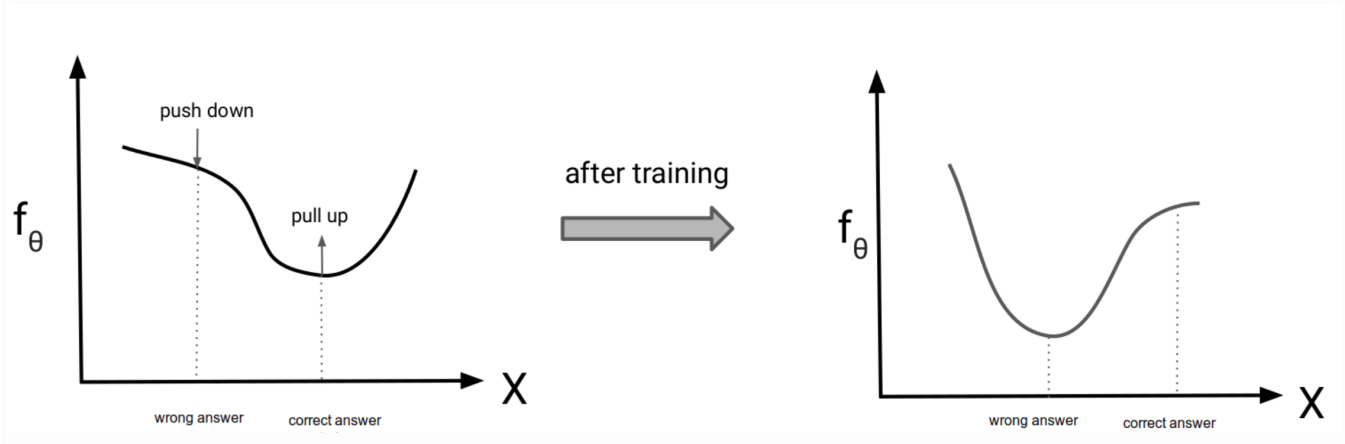
  1. $\mathbf{x} \in \{0,1\}^n$ are visible variables (e.g., pixel values).
  2. $\mathbf{z} \in \{0,1\}^m$ are latent variables.

- The normalization constant (the "volume") is:

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

- **Notes:**
  - $Z(W, b, c)$ is a well-defined function of the parameters W, b, c, but it is hard to compute.
  - Computing $Z(W, b, c)$ takes time exponential in n and m, making it infeasible for large-scale problems.
  - the summation over x requires 2^n and for z requires 2^m values.
  - This means that evaluating the objective function $p_{W,b,c}(\mathbf{x}, \mathbf{z})$ for likelihood-based learning is difficult.
- **Optimization Challenges:**
  - Optimizing the un-normalized probability $\exp(\mathbf{x}^T W \mathbf{z} + b\mathbf{x} + c\mathbf{z})$ is relatively easy with respect to the trainable parameters W, b, c.
  - However, optimizing the likelihood $p_{W,b,c}(\mathbf{x}, \mathbf{z})$, which involves $Z(W, b, c)$, is computationally difficult.

3 Training intuition



- **Goal:** Maximize the likelihood

$$p_\theta(\mathbf{x}_{\text{train}}) = \frac{\exp\{f_\theta(\mathbf{x}_{\text{train}})\}}{Z(\theta)}.$$

  where:

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

$$\max_\theta \log(x_{train}) = \max_\theta \log f_\theta(x_{train}) - \log(Z_\theta)$$

  In order to increase the $p_\theta(\mathbf{x}_{\text{train}})$ we need to increase the numerator and decrease the denominator.

- **Intuition:** Because the model is not normalized, increasing the un-normalized log-probability $f_\theta(\mathbf{x}_{\text{train}})$ by changing θ does **not** guarantee that $\mathbf{x}_{\text{train}}$ becomes relatively more likely (compared to the rest).
- We also need to take into account the effect of changing θ on other "wrong points" and try to *push them down* to also make $Z(\theta)$ small.

Idea: Instead of evaluating $Z(\theta)$ exactly, use a Monte Carlo estimate.

3.1 Contrastive Divergence Algorithm

Is an approximation to KL−Divergence between data and model.

- **Contrastive Divergence Algorithm:**
  - Sample $\mathbf{x}_{\text{sample}} \sim p_\theta$ (Negative samples).
  - Take a step on:

$$\nabla_\theta \left(f_\theta(\mathbf{x}_{\text{train}}) - f_\theta(\mathbf{x}_{\text{sample}})\right).$$

  - The goal is to make the training data more likely than a typical sample from the model. Recall, comparisons are easy in energy-based models!
  - **Maximize log-likelihood:**

$$\max_\theta f_\theta(\mathbf{x}_{\text{train}}) - \log Z(\theta).$$

    In the cake analogy, $f_\theta(\mathbf{x}_{\text{train}})$ is detecting the size of the slice while $\log Z(\theta)$ tells us how much the size of the whole cake changes.

– **Gradient of log-likelihood:**

$$\nabla_\theta f_\theta(\mathbf{x}_{\text{train}}) - \nabla_\theta \log Z(\theta)$$

here $\nabla_\theta \log Z(\theta)$ will tell us if we change θ by a some value how it will impact the partition function.

$$= \nabla_\theta f_\theta(\mathbf{x}_{\text{train}}) - \frac{\nabla_\theta Z(\theta)}{Z(\theta)}$$

as gradient is linear, we can push it inside the equation.

$$= \nabla_\theta f_\theta(\mathbf{x}_{\text{train}}) - \frac{1}{Z(\theta)} \int \nabla_\theta \exp\{f_\theta(\mathbf{x})\} d\mathbf{x}$$

$$= \nabla_\theta f_\theta(\mathbf{x}_{\text{train}}) - \frac{1}{Z(\theta)} \int \exp\{f_\theta(\mathbf{x})\} \nabla_\theta f_\theta(\mathbf{x}) d\mathbf{x}$$

$$= \nabla_\theta f_\theta(\mathbf{x}_{\text{train}}) - \int \underbrace{\frac{\exp\{f_\theta(\mathbf{x})\}}{Z(\theta)}}_{p(x)} \nabla_\theta f_\theta(\mathbf{x}) d\mathbf{x}$$

$$= \nabla_\theta f_\theta(\mathbf{x}_{\text{train}}) - \mathbb{E}_{\mathbf{x}_{\text{sample}}}[\nabla_\theta f_\theta(\mathbf{x}_{\text{sample}})]$$

$$\approx \nabla_\theta f_\theta(\mathbf{x}_{\text{train}}) - \nabla_\theta f_\theta(\mathbf{x}_{\text{sample}})$$

where:

* $\nabla_\theta f_\theta(\mathbf{x}_{\text{train}})$ is the gradient of the energy assigned at point x_{train}
* $\mathbb{E}_{\mathbf{x}_{\text{sample}}}[\nabla_\theta f_\theta(\mathbf{x}_{\text{sample}})]$ is the expected gradient for model distribution.

– Unbiased estimate of the true gradient using a single sample.

$$\approx \nabla_\theta f_\theta(\mathbf{x}_{\text{train}}) - \nabla_\theta f_\theta(\mathbf{x}_{\text{sample}}),$$

where: $\mathbf{x}_{\text{sample}} \sim p(\theta) = \exp\ f_\theta(\mathbf{x}_{\text{sample}})\}/Z(\theta)$

– Which shows that the Contrastive Divergence is a Monte Carlo estimation which can be determined using a single sample.

– as long as we follow this strategy we are increasing the relative probability of x_{train}

Challenge: While the contrastive divergence algorithm seems simple, the key issue is **sampling**, which remains difficult in practice. **How to sample?**

3.2 Sampling in Energy-Based Models:

- **Challenge:** There is no direct way to sample like in autoregressive or flow models.
  – The main issue: Cannot easily compute how likely each possible sample is.
- However, we can easily compare two samples $\mathbf{x}, \mathbf{x}'$.

3.2.1 Markov Chain Monte Carlo (MCMC):

1. Initialize $\mathbf{x}^0$ randomly, $t = 0$.
2. For t = 1, 2, ... , T-1
   (a) Let $\mathbf{x}' = \mathbf{x}^t + \text{noise}$.
   (b) If $f_\theta(\mathbf{x}') > f_\theta(\mathbf{x}^t)$, let $\mathbf{x}^{t+1} = \mathbf{x}'$.
   (c) Else, let $\mathbf{x}^{t+1} = \mathbf{x}'$ with probability (to escape locality):

$$min(1, \exp(\ \underbrace{f_\theta(\mathbf{x}')}_{\text{proposed sample}} - \underbrace{f_\theta(\mathbf{x}^t)}_{\text{current sample}}\ ).)$$

Drawback: While this works in theory, it can take a very long time to converge in practice ($T \to \infty$)).]

3.2.2 Unadjusted Langevin MCMC:

1. Initialize $\mathbf{x}^0 \sim \pi(\mathbf{x})$.
2. Repeat for $t = 0, 1, 2, \ldots, T-1$:
   - Sample $\mathbf{z}^t \sim \mathcal{N}(0, I)$.
   - Update:
   $$\mathbf{x}^{t+1} = \mathbf{x}^t + \epsilon \nabla_{\mathbf{x}} \log p_\theta(\mathbf{x})\big|_{\mathbf{x}=\mathbf{x}^t} + \sqrt{2\epsilon}\mathbf{z}^t.$$
   - Accept x^{t+1} with probability $min(1, \frac{P_\theta(x^{t+1})}{P_\theta(x^t)})$

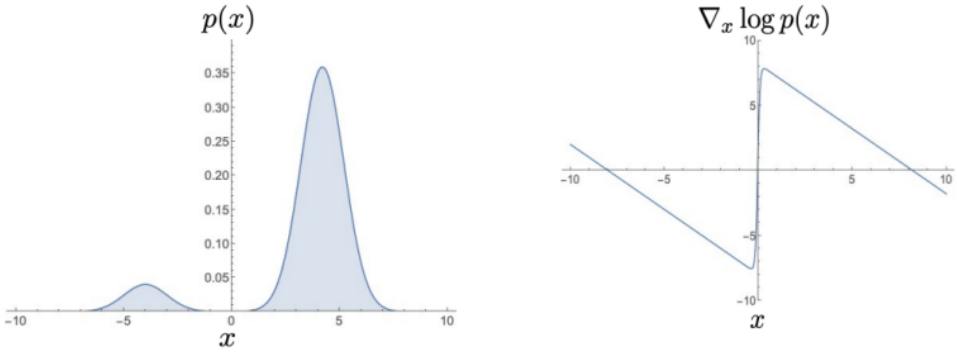
Properties:

- $\mathbf{x}^T$ converges to a sample from $p_\theta(\mathbf{x})$ when $T \to \infty$ and $\epsilon \to 0$.
- $\nabla_{\mathbf{x}} \log p_\theta(\mathbf{x}) = \nabla_{\mathbf{x}} f_\theta(\mathbf{x})$ for continuous energy-based models.
- Convergence slows down as dimensionality grows.

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

As we saw, sampling techniques tends to be expensive. Could we train without sampling? as an alternative to Contrastive Divergence algorithm? Yes, we could use methods like: **Score Matching, Noise Contrastive Estimation and Adversarial training** which do not depends on the constant function Z.

4 Score Function



Energy-based model:

$$p_\theta(\mathbf{x}) = \frac{\exp(f_\theta(\mathbf{x}))}{Z(\theta)}, \quad \log p_\theta(\mathbf{x}) = f_\theta(\mathbf{x}) - \log Z(\theta).$$

(Stein) Score Function:

$$s_\theta(\mathbf{x}) := \nabla_{\mathbf{x}} \log p_\theta(\mathbf{x}) = \nabla_{\mathbf{x}} f_\theta(\mathbf{x}) - \underbrace{\nabla_{\mathbf{x}} \log Z(\theta)}_{=0} = \nabla_{\mathbf{x}} f_\theta(\mathbf{x}).$$

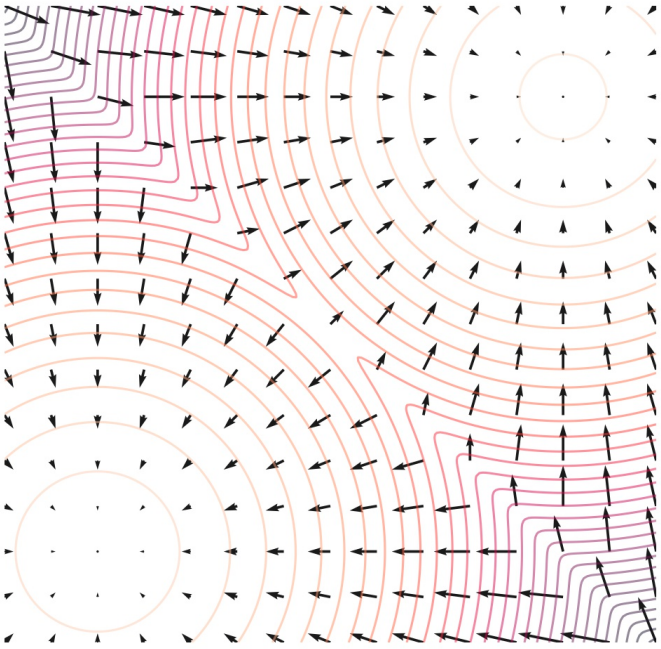
This is the gradient of log likelihood with respect to x which tells us how our likelihood will change if we make a small change to the data. This value will be function of x which means at every axis there will be a different gradient and function of θ because log−likelihood itself is parametrized by NNs with weights θ.

- **Gaussian distribution:**

$$p_\theta(\mathbf{x}) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \quad \longrightarrow \quad s_\theta(\mathbf{x}) = -\frac{x-\mu}{\sigma^2}.$$

- **Gamma distribution:**

$$p_\theta(\mathbf{x}) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x} \quad \longrightarrow \quad s_\theta(\mathbf{x}) = \frac{\alpha-1}{x} - \beta.$$



The score function of a mixture of two Gaussians.

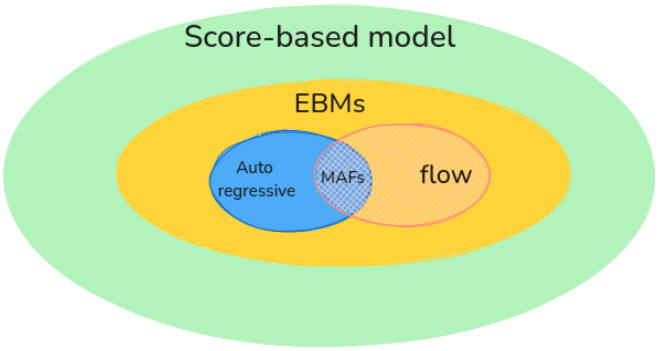
The image illustrates a vector field (or score field) overlaid on contour lines of a probability distribution. It represents the score function for a mixture of two Gaussian distributions. The arrows point towards the modes of the Gaussian components, and the contours highlight areas of equal probability density. It is commonly used in the context of energy-based models to depict gradients of the log-probability or energy function.

- **Contours:** The contour lines represent level sets of the probability density function $p_\theta(\mathbf{x})$ or the energy function $f_\theta(\mathbf{x})$. Regions with tighter contours (near the centers of the clusters of arrows) indicate steep gradients or higher density changes.
- **Arrows:** The arrows show the direction and magnitude of the gradient of the log-probability:

$$\nabla_{\mathbf{x}} \log p_\theta(\mathbf{x}) = \nabla_{\mathbf{x}} f_\theta(\mathbf{x}),$$

  known as the *score function*. These gradients point toward regions of higher probability or lower energy. These arrows tell us to increase the likelihood push the gradient towards the mean of Gaussian.
- **Energy-Based Models:** In energy-based models, the score function is used for sampling (e.g., via Langevin dynamics) or optimization. It provides the direction for moving toward regions of higher likelihood.

The broad idea is to apply Score-based to train other energy based models like auto-regressive and flow models by directly modeling the score function instead of the energy function. The score model will be not be a likelihood or energy based and instead it will be a vector-valued function(s).



Goal:

Our goal is to identify a function s_θ within our model family, parameterized by θ, that closely approximates the gradient vector field of the true data density.

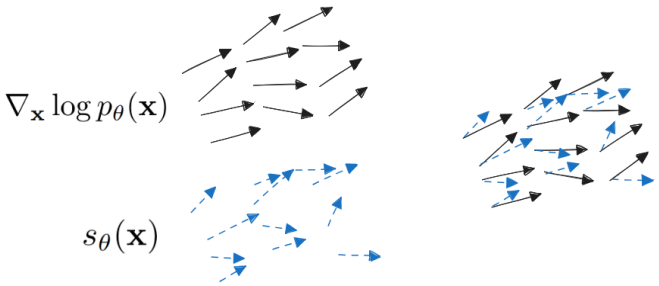
$$s_\theta(x) \approx \nabla_x \log p_\theta(x)$$

4.1 Fisher Divergence between $p(\mathbf{x})$ and $q(\mathbf{x})$:

Is a method to see the similarity between p and q by calculating the average L_2 difference between their score function.

$$D_F(p,q) := \frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p}\left[\|\nabla_\mathbf{x} \log p(\mathbf{x}) - \nabla_\mathbf{x} \log q(\mathbf{x})\|_2^2\right]$$

4.2 Score Matching:



Minimizing the Fisher divergence between $p_{\text{data}}(\mathbf{x})$ and the energy-based model $p_\theta(\mathbf{x}) \propto \exp\{f_\theta(\mathbf{x})\}$:

$$\frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}\left[\|\nabla_\mathbf{x} \log p_{\text{data}}(\mathbf{x}) - \nabla_\mathbf{x} \log p_\theta(\mathbf{x})\|_2^2\right]$$

$$= \frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}\left[\|\nabla_\mathbf{x} \log p_{\text{data}}(\mathbf{x}) - s_\theta(\mathbf{x})\|_2^2\right]$$

$$= \frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}\left[\|\nabla_\mathbf{x} \log p_{\text{data}}(\mathbf{x}) - \nabla_\mathbf{x} f_\theta(\mathbf{x})\|_2^2\right]$$

which is difference between the gradients of the true data distribution and the model. As we have seen before the $\nabla_\mathbf{x} \log p_{\text{data}}$ is unknown.

$$\frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}\left[(\nabla_\mathbf{x} \log p_{\text{data}}(\mathbf{x}) - \nabla_\mathbf{x} f_\theta(\mathbf{x}))^2\right]$$

$$= \frac{1}{2} \int p_{\text{data}}(\mathbf{x}) \left[(\nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x}) - \nabla_{\mathbf{x}} f_\theta(\mathbf{x}))^2 \right] d\mathbf{x}$$

$$= \frac{1}{2} \int p_{\text{data}}(\mathbf{x}) (\nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x}))^2 d\mathbf{x} + \frac{1}{2} \int p_{\text{data}}(\mathbf{x}) (\nabla_{\mathbf{x}} f_\theta(\mathbf{x}))^2 d\mathbf{x}$$

$$- \int p_{\text{data}}(\mathbf{x}) \nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x}) \nabla_{\mathbf{x}} f_\theta(\mathbf{x}) d\mathbf{x}$$

How to deal with $\nabla_{\mathbf{x}} \log p_{\mathbf{data}}(\mathbf{x})$ given only samples? Integration by parts!

Recall Integration by parts:

$$\int f'g = fg - \int fg'.$$

- $f = p_{\text{data}}(\mathbf{x})$: The raw data probability density function.
- $f' = \nabla_{\mathbf{x}} p_{\text{data}}(\mathbf{x})$: The gradient of the probability density function.
- $g = \nabla_{\mathbf{x}} f_\theta(\mathbf{x})$: The score function for the model distribution.
- $g' = \nabla^2_{\mathbf{x}} f_\theta(\mathbf{x})$: The Hessian of the log-probability for the model distribution.

Substitution:

The integrand

$$p_{\text{data}}(\mathbf{x}) \nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x}) \nabla_{\mathbf{x}} f_\theta(\mathbf{x})$$

can be rewritten using:

$$\nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x}) = \frac{\nabla_{\mathbf{x}} p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x})}.$$

Substituting this into the integral:

$$\int p_{\text{data}}(\mathbf{x}) \nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x}) \nabla_{\mathbf{x}} f_\theta(\mathbf{x}) \, d\mathbf{x} = \int \frac{\nabla_{\mathbf{x}} p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x})} \cdot p_{\text{data}}(\mathbf{x}) \cdot \nabla_{\mathbf{x}} f_\theta(\mathbf{x}) \, d\mathbf{x}.$$

Simplify:

$$- \int \nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x}) \nabla_{\mathbf{x}} f_\theta(\mathbf{x}) d\mathbf{x} = - \int f' \cdot g \, d\mathbf{x}.$$

Back to integral by parts:

$$- \int \nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x}) \nabla_{\mathbf{x}} f_\theta(\mathbf{x}) d\mathbf{x}$$

$$= \underbrace{\underbrace{-p_{\text{data}}(\mathbf{x}) \nabla_{\mathbf{x}} f_\theta(\mathbf{x}) \big|_{-\infty}^{\infty}}_{=0}}_{\text{f.g}} + \underbrace{\int p_{\text{data}}(\mathbf{x}) \nabla^2_{\mathbf{x}} f_\theta(\mathbf{x}) d\mathbf{x}}_{\int fg'}$$

Assumption: $p_{\text{data}}(\mathbf{x}) \to 0$ as $\mathbf{x} \to \pm\infty$, assuming the data distribution has finite support or tails that decay to zero. This is a typical property of well-behaved probability distributions, such as Gaussians, where the density becomes negligible at extreme values of $\mathbf{x}$.

$$= \int p_{\text{data}}(\mathbf{x}) \nabla^2_{\mathbf{x}} f_\theta(\mathbf{x}) d\mathbf{x}.$$

Univariate score matching

$$= \underbrace{\frac{1}{2}\int p_{\text{data}}(\mathbf{x}) \left(\nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})\right)^2 d\mathbf{x}}_{\text{const. w.r.t. } \theta}. + \frac{1}{2}\int p_{\text{data}}(\mathbf{x}) \left(\nabla_{\mathbf{x}} f_\theta(\mathbf{x})\right)^2 d\mathbf{x} + \int p_{\text{data}}(\mathbf{x}) \nabla_{\mathbf{x}}^2 f_\theta(\mathbf{x}) d\mathbf{x}$$

$$= \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}\left[\frac{1}{2}\left(\nabla_{\mathbf{x}} f_\theta(\mathbf{x})\right)^2 + \nabla_{\mathbf{x}}^2 f_\theta(\mathbf{x})\right] + \text{const.}$$

Multivariate score matching

$$= \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}\left[\frac{1}{2}\|\nabla_{\mathbf{x}} f_\theta(\mathbf{x})\|_2^2 + \text{tr}\left(\nabla_{\mathbf{x}}^2 f_\theta(\mathbf{x})\right)\right] + \text{const.}$$

Note: $\nabla_{\mathbf{x}}^2 f_\theta(\mathbf{x})$ represents the Hessian of $f_\theta(\mathbf{x})$. The trace represents the sum of diagonal entries of the Hessian, which corresponds to the divergence of the gradient field (x). This term encapsulates how the log-probability curvature behaves in all dimensions of the space.

1. Sample a mini-batch of datapoints $\{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n\} \sim p_{\text{data}}(\mathbf{x})$.
2. Estimate the score matching loss with the empirical mean:

$$\frac{1}{n}\sum_{i=1}^{n}\left[\frac{1}{2}\|\nabla_{\mathbf{x}} f_\theta(\mathbf{x}_i)\|_2^2 + \text{trace}(\nabla_{\mathbf{x}}^2 f_\theta(\mathbf{x}_i))\right]$$

$$\underbrace{= \frac{1}{n}\sum_{i=1}^{n}\left[\frac{1}{2}\|\nabla_{\mathbf{x}} f_\theta(\mathbf{x}_i)\|_2^2 + \text{trace}(\nabla_{\mathbf{x}}^2 f_\theta(\mathbf{x}_i))\right]}_{\text{Implicit Score Matching (Hyvärinen, 2005)}}$$

3. Perform stochastic gradient descent.
4. No need to sample from the EBM!

Caveat: Computing the trace of the Hessian $\text{tr}(\nabla_{\mathbf{x}}^2 f_\theta(\mathbf{x}))$ is, in general, very expensive for large models.

References

[1] Stefano Ermon. *CS236*.

[2] Yilun Du, Shuang Li, Igor Mordatch Compositional Visual Generation with Energy Based Models