Deep Learning

Diffusion Models and Normalizing Flows

11-785 - Fall 2023

Abuzar Khan
Open Question

1. Say we have a data distribution $p$ that is a mixture of two 2D gaussians as shown below in red. We want to approximate this with one gaussian estimate $q$ using KL-divergence. Which of the following three will result from optimizing $D_{KL}(p \parallel q)$
2. and which from $D_{KL}(q \parallel p)$?
Open Question

1. Say we have a data distribution $p$ that is a mixture of two 2D gaussians as shown below in red. We want to approximate this with one gaussian estimate $q$ using KL-divergence. Which of the following three will result from optimizing $D_{KL}(p || q)$

2. and which from $D_{KL}(q || p)$?
Background

1. Generative Models and Discriminative models
2. Autoencoders
3. Variational Autoencoders
   1. Reparameterization trick
   2. ELBO
Sandcastles

How to create a sandcastle:

**Step 1:** Take a sandcastle

**Step 2:** Destroy the sandcastle

**Step 3:** Remember how you destroyed the sandcastle

**Step 4:** Reverse the process

---

**Key Idea**

Once you know how to reconstruct sandcastles, you can start with some different “sand”, apply this process, and end up with a different “sandcastle”
Part 1

Diffusion Models
ELBO Recap

Why use ELBO?
Directly maximizing $p(x)$ is very difficult:
• it involves either marginalizing over the entire latent space $Z$ (intractable for complex models) OR
• It involves having access to the ground truth latent encoder $p(z|x)$

ELBO:

$$\log(p(x)) \geq \mathbb{E}_{q_\phi(z \mid x)} \left[ \log \frac{p(x, z)}{q_\phi(z \mid x)} \right]$$

**Question:** Why does the $\geq$ show up here?  → With the derivation in the appendix, we see a $D_{KL}(q_\phi(z \mid x) \mid \mid p(z \mid x))$ term show up which is always $\geq 0$.

Applying chain-rule of probabilities:

$$ELBO = \mathbb{E}_{q_\phi(z \mid x)} \left[ \log p_\theta(x \mid z) \right] - D_{KL}(q_\phi(z \mid x) \mid \mid p(z))$$

Reconstruction  Prior matching
Variational Autoencoder Recap

Latent variable sampling: $z \sim \mathcal{N}(z; \mu_\phi(x), \sigma^2_\phi(x))$

Reparameterization trick: $z = \mu_\phi(x) + \sigma_\phi(x) \odot \epsilon, \ \epsilon \sim \mathcal{N}(0, I)$

Training:
- Jointly optimize $\theta$ and $\phi$
- Maximize $\text{ELBO}$

Empirically, we found that two things make VAEs work really well:
1. Increasing the depth of the networks
2. Introducing a hierarchy of latent variables (latent variables of latent variables)

$x \leftarrow z_1 \leftarrow z_2 \leftarrow \ldots \leftarrow z_T$, such that each latent is conditioned on all previous latents.

We are particularly interested in such HAVEs that where the process is a Markovian chain - MHVAE
Markovian Hierarchical Variational Autoencoder

Joint probability:
\[ p(x, z_{1:T}) = p(z_T)p_\theta(x \mid z_1) \prod_{t=2}^T p_\theta(z_{t-1} \mid z_t) \]

Posterior probability:
\[ q_\phi(z_{1:T} \mid x) = q_\phi(z_1 \mid x) \prod_{t=2}^T q_\phi(z_t \mid z_{t-1}) \]

Updated ELBO:
\[ \log(p(x)) \geq \mathbb{E}_{q_\phi(z_{1:T} \mid x)} \left[ \log \frac{p(x, z_{1:T})}{q_\phi(z_{1:T} \mid x)} \right] \]
Diffusion Models

Diffusion models are essentially **MHVAEs** with **3 restrictions:**

1. Latent dimension is the same as the data dimension
2. The encoder has no parameters to be learnt. It is defined to be a linear gaussian such that the $t^{th}$ gaussian is centered around the previous latent $z_{t-1}$
3. The parameters for the gaussians are scheduled such that the final latent is a standard gaussian.

$$z_T \sim \mathcal{N}(z_T; 0, I)$$

The first restriction allows for some mild abuse of notation:

$$q_\phi(x_{1:T} | x_0) = \prod_{t=1}^{T} q_\phi(x_t | x_{t-1})$$

$$p(x_{0:T}) = p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1} | x_t)$$
The first restriction allows for some mild abuse of notation:

\[
q_\phi(x_{1:T} \mid x_0) = \prod_{t=1}^{T} q_\phi(x_t \mid x_{t-1})
\]

\[
p(x_{0:T}) = p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1} \mid x_t)
\]
Diffusion Models – Diffusion Process

Following the second restriction, we now define the linear gaussian for the encoding (diffusion) process:

\[
q(x_t | x_{t-1}) = \mathcal{N}(x_t; \mu_t(x_{t-1}), \Sigma_t I)
\]

\[
\mu_t(x_{t-1}) = \sqrt{1 - \beta_t} x_{t-1}, \quad \Sigma_t = \beta_t
\]

We additionally define \( \alpha_t = 1 - \beta_t \).

\( \beta_t \) is defined to preserve variance across the diffusion steps.

We can now write

\[
q(x_t | x_{t-1}) = \mathcal{N}(x_t; \sqrt{\alpha_t} x_{(t-1)}(1 - \alpha_t)I)
\]

Using the reparameterization trick:

\[
x_t = \sqrt{\alpha_t} x_{(t-1)} + (\sqrt{1 - \alpha_t}) \epsilon, \quad \epsilon \sim \mathcal{N}(0, I)
\]

This takes us from time step 0 to \( t \) in one step!

From the third restriction, we get

\( \alpha_T \to 0 \)

Sum of two gaussians is another gaussian with mean as the sum of the two means and variance as the sum of the two variances.

\[
(1 - \alpha_t) \epsilon \to \mathcal{N}(\epsilon; 0, 1 - \alpha_t I)
\]

Define

\[
\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s
\]
Diffusion Models – Diffusion Process

\[ q(x_t|x_{t-1}) = \sqrt{\alpha_t}x_{(t-1)} + (\sqrt{1 - \alpha_t})\epsilon, \quad \epsilon \sim \mathcal{N}(0,I) \]

This formulation essentially paints a picture of this process to be incrementally adding noise till we reach \( x_T \) which is defined to be pure noise.
Diffusion Models – Generative Process

From the third assumption, we can write the exact prior on the final step $x_T$:

$$p(x_T) = \mathcal{N}(x_T; 0, I)$$

For all other steps, we can write a learned distribution:

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_t I)$$

Neural Network: U-Net
Denoising network

Exactly tractable variance

$$p_\theta(x_{0:T}) = p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)$$
Diffusion Models – Updated ELBO

\[
\log p(x) = \log \int p(x_{0:T}) dx_{0:T}
\]

\[
= \mathbb{E}_{q(x_1 | x_0)}[\log p_\theta(x_0 | x_1)] - \mathbb{E}_{q(x_{T-1} | x_0)}[D_{KL}(q(x_T | x_{T-1}) || p(x_T))]
\]

\[
- \sum_{t=1}^{T-1} \mathbb{E}_{q(x_{t-1}, x_{t+1} | x_0)}[D_{KL}(q(x_{t} | x_{t-1}) || p_\theta(x_{t} | x_{t+1}))]
\]

This has 2 random variables for each \( t \), this makes the computation slightly hard. We would prefer for there to be need for just 1!

We can arbitrarily modify the diffusion process distribution to

\[
q(x_t | x_{t-1}, x_0) = \frac{q(x_{t-1} | x_t, x_0)q(x_t | x_0)}{q(x_{t-1} | x_0)}
\]
Diffusion Models – Updated ELBO

\[ \log p(x) = \log \int p(x_{0:T}) dx_{0:T} \]

\[ \ldots * \]

\[ = \mathbb{E}_{q(x_1 \mid x_0)}[\log p_\theta(x_0 \mid x_1)] - D_{KL}(q(x_T \mid x_0) \mid \mid p(x_T)) - \sum_{t=2}^{T} \mathbb{E}_{q(x_t \mid x_0)}[D_{KL}(q(x_{t-1} \mid x_t, x_0) \mid \mid p_\theta(x_{t-1} \mid x_t))] \]

Reconstruction

Prior matching

Denoising

- **Reconstruction**: Reconstruction from least noisy version (hyperparameter choice can make this arbitrarily small)
- **Prior matching**: Moving the posterior closer to the true prior on the final noisy step (0 for diffusion models)
- **Denoising**: Divergence between approximate denoising ($p_\theta$) and true denoising ($q$) steps

$q(x_{t-1} \mid x_t, x_0)$ is **tractable** and can be calculated **exactly** without any approximation:

\[ q(x_{t-1} \mid x_t, x_0) = \mathcal{N}(x_{t-1} ; \bar{\mu}_t, \Sigma_t) \]

\[ \bar{\mu}_t = \frac{\sqrt{\alpha_t}(1 - \bar{\alpha}_{t-1})x_t + \sqrt{\bar{\alpha}_{t-1}}(1 - \alpha_t)x_0}{1 - \alpha_t}, \quad \Sigma_t = \frac{(1 - \alpha_t)(1 - \bar{\alpha}_{t-1})}{1 - \alpha_t} \]
Diffusion Models – Loss formulation

Loss can focus on the denoising term. Decomposing for each timestep, we can have the t\textsuperscript{th} loss term:

\[ L_t = D_{KL}(q(x_{t-1}|x_t, x_0) \ || p_\theta(x_{t-1}|x_t)) + C \]

Since both inputs of the divergence are gaussians, this further simplifies to:

\[ L_t = \mathbb{E}_q \left[ \frac{1}{2\Sigma_t} \|\bar{\mu}_t - \mu_\theta(x_t, t)\|^2 \right] + C \]
Diffusion Models – Loss formulation

Further, we have $x_t = \sqrt{\alpha_t}x_{(t-1)} + (\sqrt{1 - \alpha_t})\epsilon, \; \epsilon \sim \mathcal{N}(0, I)$ from definition.

This lets us rewrite the true mean of the denoising process as:

$$\bar{\mu}_t = \frac{1}{\sqrt{\alpha_t}} \left( x_t - \frac{(1 - \alpha_t)}{\sqrt{1 - \alpha_t}} \epsilon \right)$$

We can also write the predicted mean as:

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}} \left( x_t - \frac{(1 - \alpha_t)}{\sqrt{1 - \alpha_t}} \epsilon_\theta(x_t, t) \right)$$

This lets us reformulate the loss to present a noise prediction problem:

$$L_{t-1} = \mathbb{E}_{x_0, \epsilon} \left[ \frac{(1 - \alpha_t)^2}{2 \sum_t \alpha_t (1 - \bar{\alpha}_t)} ||\epsilon - \epsilon_\theta(x_t, t)||^2 \right] + C$$
How do we tell the model what timestep we are on?
Temporal encodings in the form of sinusoids (or anything, really)
Diffusion models - Summary

- Diffusion models are **Markovian Hierarchical VAEs** with extra restrictions
- The loss is the vanilla VAE ELBO loss with an added denoising term
- The encoder has **0 parameters**
- The true denoising posterior can be **exactly calculated**
- The problem can be reformulated as a noise prediction problem
- There’s a ton of math underlying a rather simple intuition
Part 2

Normalizing Flows
Sandcastles

How to create a sandcastle:

Step 1: Take a sandcastle
Step 2: Destroy the sandcastle
Step 3: Remember how you destroyed the sandcastle
Step 4: Reverse the process

Key Idea

Once you know how to reconstruct sandcastles, you can start with some different “sand”, apply this process, and end up with a different “sandcastle”
Normalizing Flows – Motivation

- In VAEs we are faced with an intractable likelihood calculation
- We use an ELBO instead as a surrogate objective to MLE
- What if we wanted to do MLE exactly?
- That would require us to go from sandcastle to sand, and back, without any approximation or estimation!

We would need for all the steps we do to be **invertible!**

It follows that $f^{-1} = g$
Normalizing Flows – Log likelihood

Bijection (and invertibility) allow us to directly compute the likelihood:

\[
\int p_x(x)dx = \int p_z(g(x))dz
\]

In multiple dimensions, we generalize to the determinant of the Jacobian

\[
p_x(x) = p_z(g(x)) \left| \frac{dg(x)}{dx} \right| \rightarrow p_z(g(x))|\text{det. } J(g(x))|
\]

\[
\log p_x(x) = \log p_z(g(x)) + \log |\text{det. } J(g(x))|
\]

Intuitively

\( z = g(x) \) determines where a point in x-space maps to z-space (where to move grains of sand)

\( |\text{det. } J(g(x))| \) describes how much probability mass (sand) gets moved in a local neighborhood.
Normalizing Flows – Closer look at the Jacobian

\[ z = g(a) = f^{-1}(a) \]

\[ J_{g(a)} = \begin{bmatrix}
\frac{dz_1}{da_1} & \cdots & \frac{dz_1}{da_k} \\
\vdots & \ddots & \vdots \\
\frac{dz_k}{da_1} & \cdots & \frac{dz_k}{da_k}
\end{bmatrix} \]
Normalizing Flows – Compositions

Bijections allow for composing several functions together!
This follows that we can now define:

\[ z \sim p(z) \]
\[ x = f_T \circ f_{T-1} \circ \cdots \circ f_1 (z) \]

\[ \log p_x(x) = \log p_z(z_0) + \sum_{t=1}^{T} \log |det J_{z_t}(g(z_{t-1}))|, \quad z_T = x, \; z_0 = z \]

**Inverse:** \[ z = g_1 \circ g_2 \circ \cdots \circ g_T(x) \]
Normalizing Flows – Characteristics

For a good (efficient) flow, we must have functions (steps) that are:

1. Expressive
2. Invertible
3. Offer cheap to compute Jacobian determinants

Computing a determinant is a cubic operation, but some special cases of matrices can make it very cheap. Especially, diagonal matrices:

For a diagonal matrix, the determinant is simply the product of its diagonal elements. Same applies for any triangular matrix!

In linear algebra, a diagonal matrix is a matrix (usually a square matrix) in which the entries outside the main diagonal (\(\_\_\_\_\) are all zero. The diagonal entries themselves may or may not be zero. Thus, the matrix \(D = (d_{ij})\) with \(n\) columns and \(n\) rows is diagonal if:

\[
d_{i,j} = 0 \text{ if } i \neq j \forall i, j \in \{1, 2, \ldots, n\}
\]

For example, the following matrix is diagonal:

\[
\begin{bmatrix}
1 & 0 & 0 \\
0 & 4 & 0 \\
0 & 0 & -2
\end{bmatrix}
\]
Normalizing Flows – Construction

Affine transform:

\[ z_2 = \alpha x_2 + \beta \]
Is there a problem here?
Normalizing Flows – Construction with a shuffle

Affine transform:
\[ z_2 = \alpha x_2 + \beta \]

This the most popular type of flow called as **Coupling Flow** – Used in implementations such as NICE and GLOW.
Normalizing Flows – In practice for images

- Multiscale architecture
- Split along channels
- Employ CNNs
- Perform permutations using 1x1 Conv layers

Figure 4: Random samples from the model, with temperature 0.7.

Figure 5: Linear interpolation in latent space between real images.
References

- Didrik Nielsen’s lecture, https://www.youtube.com/watch?v=2tVHbcUP9b8
- Hans van Gorp’s lecture, https://www.youtube.com/watch?v=yxVcnuRrKqQ&t=17s
- Tim Salimans’ lecture, https://www.youtube.com/watch?v=pea3sH6orMc
Appendix
Vanilla VAE ELBO optimization derivation

The KL divergence term that shows up tries to match the learned posterior $q$ to the true posterior $p$.

Since KL divergence is always positive, we can ignore that term and replace the equality with the inequality.

\[
\log p(x) = \log p(x) \int q_\phi(z|x)dz \\
= \int q_\phi(z|x)(\log p(x))dz \\
= \mathbb{E}_{q_\phi(z|x)}[\log p(x)] \\
= \mathbb{E}_{q_\phi(z|x)} \left[ \log \frac{p(x, z)}{q_\phi(z|x)} \right] \\
= \mathbb{E}_{q_\phi(z|x)} \left[ \log \frac{p(x, z)}{q_\phi(z|x)} \right] + \mathbb{E}_{q_\phi(z|x)} \left[ \log \frac{q_\phi(z|x)}{p(z|x)} \right] \\
= \mathbb{E}_{q_\phi(z|x)} \left[ \log \frac{p(x, z)}{q_\phi(z|x)} \right] + D_{KL}(q_\phi(z|x) \| p(z|x)) \\
\geq \mathbb{E}_{q_\phi(z|x)} \left[ \log \frac{p(x, z)}{q_\phi(z|x)} \right]
\]
Initial ELBO optimization derivation for diffusion models

$$\log p(x) = \log \int p(x_{0:T}) q(x_{1:T} | x_0) \, dx_{1:T}$$

$$= \log \mathbb{E}_{q(x_{1:T} | x_0)} \left[ \frac{p(x_{0:T})}{q(x_{1:T} | x_0)} \right]$$

$$\geq \mathbb{E}_{q(x_{1:T} | x_0)} \left[ \log \frac{p(x_{0:T})}{q(x_{1:T} | x_0)} \right]$$

$$= \mathbb{E}_{q(x_{1:T} | x_0)} \left[ \log \frac{p(x_T) \prod_{t=1}^{T-1} p(x_{t-1} | x_t)}{\prod_{t=1}^{T-1} q(x_t | x_{t-1})} \right]$$

$$= \mathbb{E}_{q(x_{1:T} | x_0)} \left[ \log \frac{p(x_T) \prod_{t=1}^{T-1} p(x_{t-1} | x_t)}{q(x_T | x_{T-1}) \prod_{t=1}^{T-1} q(x_t | x_{t-1})} \right]$$

$$= \mathbb{E}_{q(x_{1:T} | x_0)} \left[ \log \frac{p(x_T) \prod_{t=1}^{T-1} p(x_{t-1} | x_t)}{q(x_T | x_{T-1}) \prod_{t=1}^{T-1} q(x_t | x_{t-1})} \right]$$

$$= \mathbb{E}_{q(x_{1:T} | x_0)} \left[ \log \frac{p(x_T) p(x_0 | x_1)}{q(x_T | x_{T-1})} \right] + \mathbb{E}_{q(x_{1:T} | x_0)} \left[ \log \prod_{t=1}^{T-1} \frac{p(x_t | x_{t+1})}{q(x_t | x_{t-1})} \right]$$

$$= \mathbb{E}_{q(x_{1:T} | x_0)} \left[ \log p(x_0 | x_1) \right] + \mathbb{E}_{q(x_{1:T} | x_0)} \left[ \log \frac{p(x_T)}{q(x_T | x_{T-1})} \right] + \mathbb{E}_{q(x_{1:T} | x_0)} \left[ \sum_{t=1}^{T-1} \log \frac{p(x_t | x_{t+1})}{q(x_t | x_{t-1})} \right]$$

$$= \mathbb{E}_{q(x_{1:T} | x_0)} \left[ \log p(x_0 | x_1) \right] + \mathbb{E}_{q(x_{1:T} | x_0)} \left[ \log \frac{p(x_T)}{q(x_T | x_{T-1})} \right] + \sum_{t=1}^{T-1} \mathbb{E}_{q(x_{1:T} | x_0)} \left[ \log \frac{p(x_t | x_{t+1})}{q(x_t | x_{t-1})} \right]$$

$$= \mathbb{E}_{q(x_{1:T} | x_0)} \left[ \log p(x_0 | x_1) \right] - \mathbb{E}_{q(x_{T-1} | x_0)} \left[ D_{KL}(q(x_T | x_{T-1}) \ || \ p(x_T)) \right]$$

$$- \sum_{t=1}^{T-1} \mathbb{E}_{q(x_{t-1:T} | x_0)} \left[ D_{KL}(q(x_t | x_{t-1}) \ || \ p(x_t | x_{t+1})) \right]$$

reconstruction term

prior matching term

consistency term
Modified ELBO optimization derivation for diffusion models

Only involves one random variable per expectation term!