Homework 3: Part II
Variational Autoencoders, Diffusion Models, and Graph Convolutional Networks[1]

CMU 10-417/617: Intermediate Deep Learning (Fall 2023)

https://rsalakhucmu.github.io/10417-23/index.html
OUT: Wednesday, November 15th, 2023

DUE: Wednesday, November 29th, 2023, 11:59pm
TAs: Raghav Kapoor, Sachin Goyal, Vincent Tombari

START HERE: Instructions

Homework 3 (part II) covers topics on variational autoencoders, diffusion models, and graph neural networks. The homework includes math questions.

- **Collaboration policy:** Collaboration on solving the homework is allowed, after you have thought about the problems on your own. It is also OK to get clarification (but not solutions) from books or online resources, again after you have thought about the problems on your own. There are two requirements: first, cite your collaborators fully and completely (e.g., "Jane explained to me what is asked in Question 2.1"). Second, write your solution *independently*: close the book and all of your notes, and send collaborators out of the room, so that the solution comes from you only. See the Academic Integrity Section on the course site for more information: https://rsalakhucmu.github.io/10417-23/index.html
- **Late Submission Policy:** See the late submission policy here: https://rsalakhucmu.github.io/10417-23/index.html
- **Submitting your work:**
- **Gradescope:** For written problems such as short answer, multiple choice, derivations, proofs, or plots, we will be using Gradescope (https://gradescope.com/). Please write your solution in the LaTeX files provided in the assignment and submit in a PDF form. Put your answers in the question boxes (between `\begin{soln}` and `\end{soln}`) below each problem. Please make sure you complete your answers within the given size of the question boxes. **Handwritten solutions are not accepted and will receive zero credit.** Regrade requests can be made, however this gives the TA the opportunity to regrade your entire paper, meaning if additional mistakes are found then points will be deducted. For more information about how to submit your assignment, see the following tutorial (note that even though the assignment in the tutorial is handwritten, submissions must be typed): https://www.youtube.com/watch?v=KMPoby5g_nE&feature=youtu.be

Important Notes:

- Please check whether your solution boxes fit into the solution areas when you submit your pdf to Gradescope.
- If you have more text than the solution box, you can resize the box using height option in the latex command $\backslash begin\{soln\}\{height = 10cm\}$.
- You can also put $\backslash pagebreak$ before the solution environment (before $\backslash begin\{soln\}$) to make your final pdf look better-organized.

[1] Compiled on Friday 1st December, 2023 at 02:35

Problem 1 - Variational Autoencoders (12 points)

1 Background information on VAEs – optional reading

A variational autoencoder (VAE) is a probabilistic model whose generative process comprises a prior $p_\theta(\mathbf{h})$ over hidden variables and a decoder/generative model $p_\theta(\mathbf{x}|\mathbf{h})$. To sample a data point $\mathbf{x}$ from a VAE, you first sample a hidden vector $\mathbf{h} \sim p_\theta(\mathbf{h})$, then you sample a data vector $\mathbf{x} \sim p_\theta(\mathbf{x}|\mathbf{h})$. For example $p_\theta(\mathbf{h})$ might be a standard Gaussian distribution $\mathcal{N}(\mathbf{0}, \mathbf{I}_k)$, and $p_\theta(\mathbf{x}|\mathbf{h})$ might be a Gaussian distribution whose mean is computed by a neural network function of $\mathbf{h}$; in this case, θ would be the weights of the neural network.

With this generative process in mind, notice that computing the marginal probability $p_\theta(\mathbf{x})$ of a data point $\mathbf{x}$ requires you to integrate over all the possible hidden vectors $\mathbf{h}$ that could have generated $\mathbf{x}$. Formally,

$$p_\theta(\mathbf{x}) = \int_{\mathbf{h}} p_\theta(\mathbf{x}|\mathbf{h}) p_\theta(\mathbf{h}) d\mathbf{h}$$

Unfortunately, computing this integral over $\mathbf{h}$ is usually intractable. This is problematic because it means that we can't train the VAE in the straightforward way that most probabilistic models are trained — by choosing the parameters θ so as to maximize the log-likelihood of a training dataset $\mathbf{x}_1, \ldots \mathbf{x}_n$:

$$\operatorname*{argmax}_\theta \sum_{i=1}^{n} \log p_\theta(\mathbf{x}_i) \qquad \text{(we can't do this efficiently!)}$$

Therefore, rather than training a VAE by directly maximizing the log-probability $\log p_\theta(\mathbf{x})$ of the data, we instead train a VAE by maximizing a *lower bound* $\mathcal{L}(\mathbf{x}; \theta, \phi)$ on the log-probability of the data. (In general, maximizing a lower bound on some quantity is also guaranteed to indirectly maximize that quantity.)

This *variational lower bound* involves an additional model $q_\phi(\mathbf{h}|\mathbf{x})$. This q_ϕ is sometimes called the "encoder", and sometimes called a "recognition model." It is often a Gaussian distribution whose mean and standard deviation are both computed by a neural network — ϕ are the weights of this neural network.

How to use the recognition model q_ϕ to derive a lower bound $\mathcal{L}(\mathbf{x}; \theta, \phi)$ on the log-likelihood $\log p_\theta(\mathbf{x})$?

As you saw in class, for any data point $\mathbf{x}$, the log-probability $\log p_\theta(\mathbf{x})$ is lower bounded by:

$$\begin{aligned}
\log p_\theta(\mathbf{x}) &= \log \int_{\mathbf{h}} p_\theta(\mathbf{x}, \mathbf{h}) d\mathbf{h} && \text{(marginalization)} \\
&= \log \int_{\mathbf{h}} \frac{p_\theta(\mathbf{x}, \mathbf{h})}{q_\phi(\mathbf{h}|\mathbf{x})} q_\phi(\mathbf{h}|\mathbf{x}) d\mathbf{h} \\
&= \log \mathbb{E}_{\mathbf{h} \sim q_\phi(\mathbf{h}|\mathbf{x})} \left[\frac{p_\theta(\mathbf{x}, \mathbf{h})}{q_\phi(\mathbf{h}|\mathbf{x})} \right] && \text{(definition of expectation)} \\
&\geq \mathbb{E}_{\mathbf{h} \sim q_\phi(\mathbf{h}|\mathbf{x})} \left[\log \frac{p_\theta(\mathbf{x}, \mathbf{h})}{q_\phi(\mathbf{h}|\mathbf{x})} \right] && \text{(Jensen's inequality)} \\
&:= \mathcal{L}(\mathbf{x}; \theta, \phi)
\end{aligned}$$

2 The Actual Homework Problem

Consider a variational autoencoder (VAE) that consists of a recognition model $q_\phi(\mathbf{h}|\mathbf{x})$, a generative model $p_\theta(\mathbf{x}|\mathbf{h})$, and a prior $p_\theta(\mathbf{h})$ where $\mathbf{x}$ and $\mathbf{h}$ denote input and hidden variables, respectively.

As you saw in class (and as we derived above), for any data point $\mathbf{x}$, the log-probability $\log p_\theta(\mathbf{x})$ is lower bounded by:

$$\mathcal{L}(\mathbf{x}; \theta, \phi) := \mathbb{E}_{\mathbf{h} \sim q_\phi(\mathbf{h}|\mathbf{x})} \left[\log \frac{p_\theta(\mathbf{x}, \mathbf{h})}{q_\phi(\mathbf{h}|\mathbf{x})} \right]$$

Another common way of writing the variational lower bound $\mathcal{L}(\mathbf{x}; \theta, \phi)$ is:

$$\mathcal{L}(\mathbf{x}; \theta, \phi) = \mathbb{E}_{\mathbf{h} \sim q_\phi(\mathbf{h}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{h})] - D_{\mathrm{KL}}(q_\phi(\mathbf{h}|\mathbf{x}) || p_\theta(\mathbf{h})) \tag{1}$$

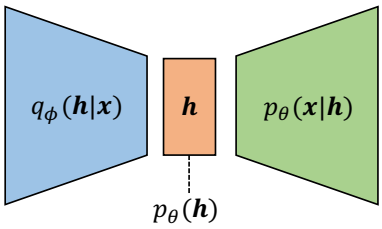


Figure 1: Variational autoencoder with a recognition model $q_\phi(\mathbf{h}|\mathbf{x})$, a generative model $p_\theta(\mathbf{x}|\mathbf{h})$, and a prior $p_\theta(\mathbf{h})$ where $\mathbf{x}$ and $\mathbf{h}$ denote input and hidden variables. $q_\phi(\mathbf{h}|\mathbf{x})$ approximates the true posterior $p_\theta(\mathbf{h}|\mathbf{x})$.

1. **(3 points)** Prove that these two ways of writing $\mathcal{L}(\mathbf{x};\theta,\phi)$ are equivalent. That is, prove that

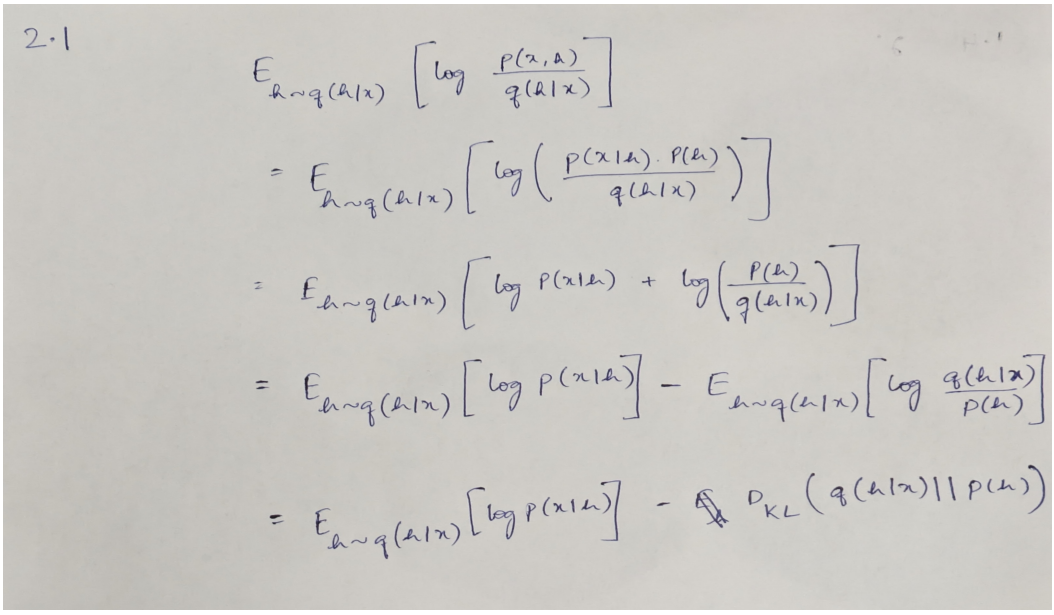
$$\mathbb{E}_{\mathbf{h}\sim q(\mathbf{h}|\mathbf{x})}\left[\log \frac{p(\mathbf{x},\mathbf{h})}{q(\mathbf{h}|\mathbf{x})}\right] = \mathbb{E}_{\mathbf{h}\sim q(\mathbf{h}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{h})] - D_{\mathrm{KL}}(q(\mathbf{h}|\mathbf{x})||p(\mathbf{h}))$$

(To remove notational clutter, we've suppressed the parameters θ and ϕ. You may do so too in your answer.)

Note that the KL divergence between two arbitrary distributions A and B with respective densities α and β is defined as:

$$D_{\mathrm{KL}}(A||B) = \mathbb{E}_{\mathbf{h}\sim A}\left[\log \frac{\alpha(\mathbf{h})}{\beta(\mathbf{h})}\right]$$

Solution



Now assume that $q_\phi(\mathbf{h}|\mathbf{x})$ is a diagonal Gaussian distribution whose mean vector and standard deviation vector are computed using neural networks:

$$q_\phi(\mathbf{h}|\mathbf{x}) = \mathcal{N}(\mathbf{h};\boldsymbol{\mu}(\mathbf{x}),\mathrm{diag}(\boldsymbol{\sigma}(\mathbf{x})^2))$$

where $\boldsymbol{\mu}:\mathbb{R}^d\to\mathbb{R}^k$ and $\boldsymbol{\sigma}:\mathbb{R}^d\to\mathbb{R}^k$ are neural networks with weights ϕ.

Further, assume that the prior $p(\mathbf{h})$ is an isotropic Gaussian distribution with mean zero and unit variance:

$$p(\mathbf{h}) = \mathcal{N}(\mathbf{h}; \mathbf{0}, \mathbf{I}_k)$$

For this special case, the KL term can be computed in closed form! Define $\text{GKL}(\boldsymbol{\mu}, \boldsymbol{\sigma})$ as the KL divergence between the distributions $\mathcal{N}(\boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma}^2))$ and $\mathcal{N}(\mathbf{0}, \mathbf{I}_k)$.[2] There is an analytical closed form for this KL divergence:

$$\text{GKL}(\boldsymbol{\mu}, \boldsymbol{\sigma}) = D_{KL}(\mathcal{N}(\boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma}^2)) \,||\, \mathcal{N}(\mathbf{0}, \mathbf{I}_k)) = \sum_{j=1}^{k} \left[-\log \sigma_j + \frac{\sigma_j^2 + \mu_j^2 - 1}{2} \right] \tag{2}$$

Therefore, the variational lower bound $\mathcal{L}(\mathbf{x}; \theta, \phi)$ from (1) can be written as:

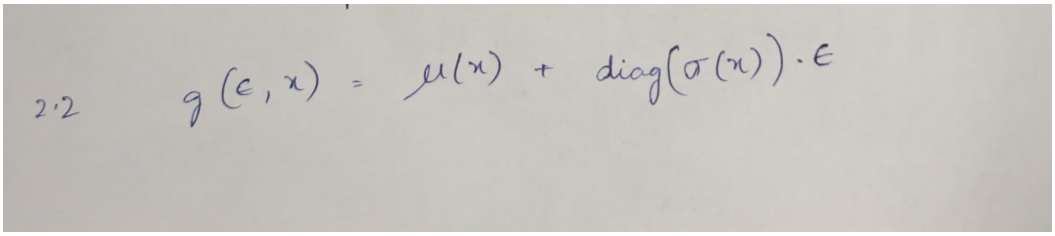
$$\mathcal{L}(\mathbf{x}; \theta, \phi) = \underbrace{\mathbb{E}_{\mathbf{h} \sim \mathcal{N}(\boldsymbol{\mu}(\mathbf{x}), \text{diag}(\boldsymbol{\sigma}(\mathbf{x})^2))}[\log p_\theta(\mathbf{x}|\mathbf{h})]}_{\mathcal{L}_{\text{recon}}(\mathbf{x}; \theta, \phi)} - \text{GKL}(\boldsymbol{\mu}(\mathbf{x}), \boldsymbol{\sigma}(\mathbf{x})) \tag{3}$$

where we denote the left term as $\mathcal{L}_{\text{recon}}$, as it resembles the reconstruction error term in an autoencoder.

2. **(3 points)** Give a function $g(\boldsymbol{\epsilon}, \mathbf{x})$ for which $\mathcal{L}_{\text{recon}}(\mathbf{x}; \theta, \phi)$ can be expressed as an expectation over $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ of the form:

$$\mathcal{L}_{\text{recon}}(\mathbf{x}; \theta, \phi) = \mathbb{E}_{\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})} \left[\log p_\theta(\mathbf{x}|g(\boldsymbol{\epsilon}, \mathbf{x})) \right] \tag{4}$$

Solution



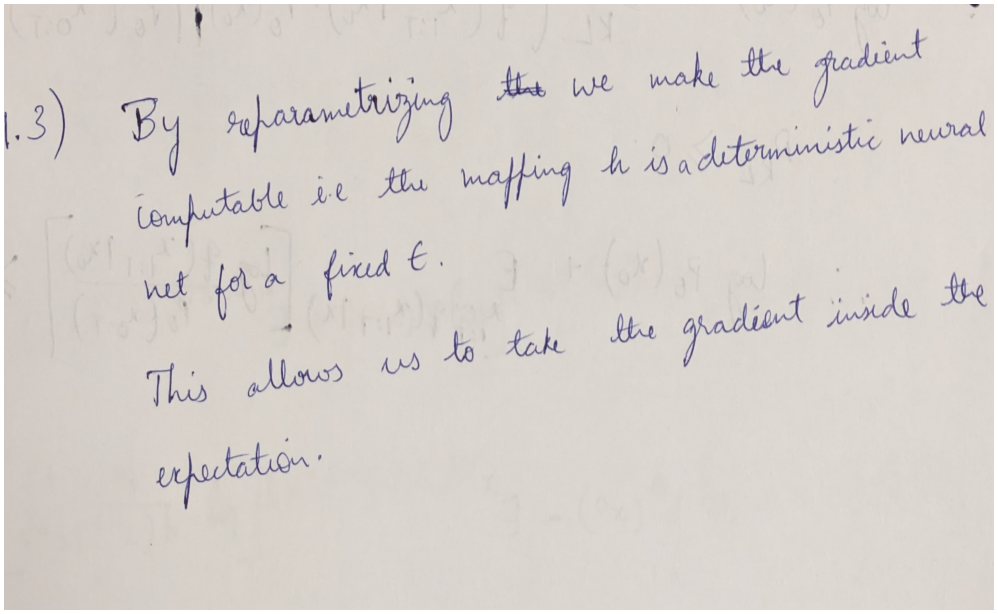
[2] We're overloading notation here by using $\boldsymbol{\mu}$ to denote an arbitrary mean vector, rather than the neural network that computes the mean vector. (Same for $\boldsymbol{\sigma}$)

3. **(3 points)** Explain why the following statement is true:

$$\nabla_{\theta,\phi}\mathcal{L}_{\text{recon}}(\mathbf{x};\theta,\phi) = \mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\mathbf{I})}\left[\nabla_{\theta,\phi}\ \log p_\theta(\mathbf{x}|g(\boldsymbol{\epsilon},\mathbf{x}))\right] \tag{5}$$

We're not looking for a rigorous proof; we are looking for you to cite a certain condition (mentioned in lecture 18!) satisfied by the reparameterization (4) but not the original parameterization (3), which allows us to move the gradient operator inside the expectation.

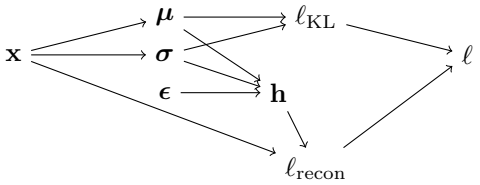
Solution



4. **(3 points)** Because of Equation (5), we can maximize the variational lower bound by sampling a single $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and applying backpropagation to minimize the following loss function:

$$\ell(\theta,\phi) = \underbrace{-\log p_\theta(\mathbf{x}|g(\boldsymbol{\epsilon},\mathbf{x}))}_{\ell_{\text{recon}}} + \underbrace{\text{GKL}(\boldsymbol{\mu}(\mathbf{x}),\boldsymbol{\sigma}(\mathbf{x}))}_{\ell_{\text{KL}}} \tag{6}$$

The computational graph looks like this:



That is, $\boldsymbol{\mu}$ is computed from $\mathbf{x}$ using the neural network $\boldsymbol{\mu}(\mathbf{x})$, $\boldsymbol{\sigma}$ is computed from $\mathbf{x}$ using the neural network $\boldsymbol{\sigma}(\mathbf{x})$, ℓ_{KL} is computed from $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ using (2), $\mathbf{h}$ is computed as $\mathbf{h} = g(\mathbf{x},\boldsymbol{\epsilon})$, ℓ_{recon} is computed from $\mathbf{h}$ and $\mathbf{x}$ as $-\log p_\theta(\mathbf{x}|\mathbf{h})$, and finally ℓ is computed using (6) as $\ell_{\text{recon}} + \ell_{\text{KL}}$.

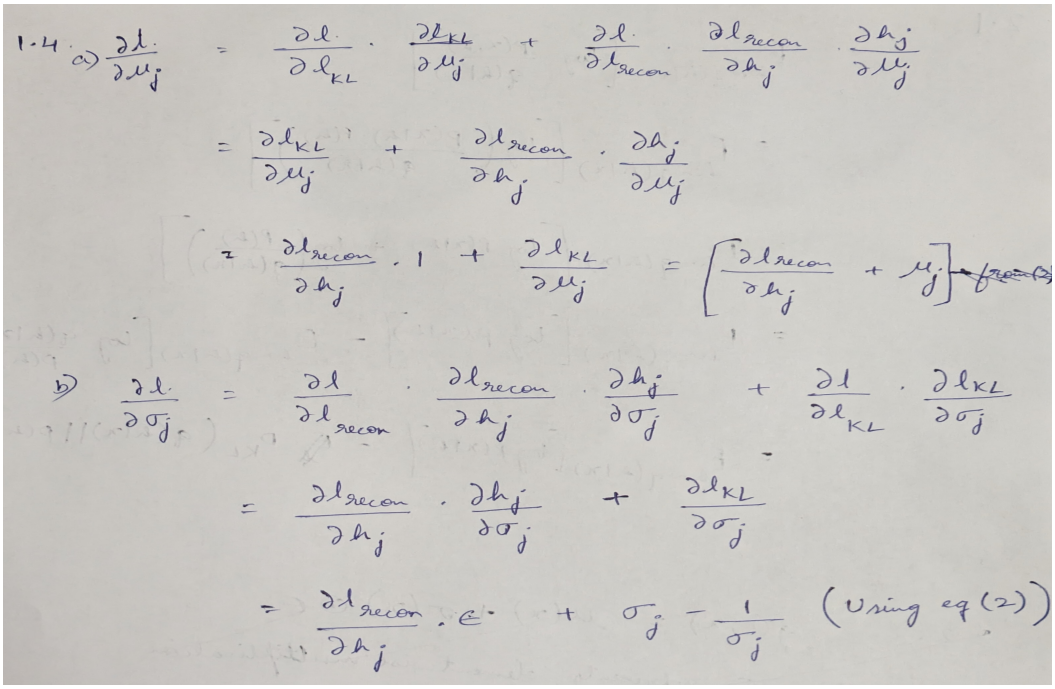
With this graph in mind, compute both of the following quantities:

(a) $\frac{\partial \ell}{\partial \mu_j}$, the derivative of ℓ w.r.t the j-th dimension of $\boldsymbol{\mu}$

(b) $\frac{\partial \ell}{\partial \sigma_j}$, the derivative of ℓ w.r.t the j-th dimension of $\boldsymbol{\sigma}$

Important!! You can take $\frac{\partial \ell_{\text{recon}}}{\partial h_j}$ as given — that is, your answer should include the quantity $\frac{\partial \ell_{\text{recon}}}{\partial h_j}$.[3]

Solution



Problem 2 - Graph Convolutional Networks (6 points)

The main challenge of adapting Graph Convolutional Networks (GCNs) to large-scale graphs is the scalability issue due to the uncontrollable neighborhood expansion in the aggregation stage. Several sampling algorithms have been proposed to limit the neighborhood expansion. In this problem, we train a performance-adaptive sampling strategy for GCNs that samples neighbors informative for a target task.

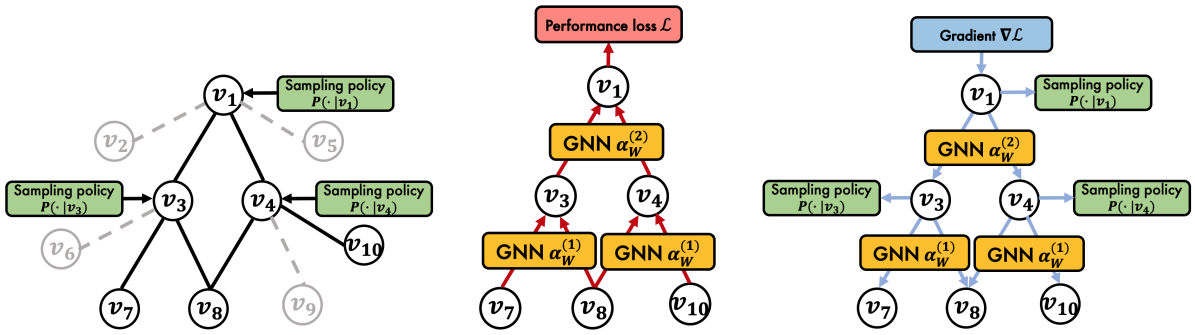
We first briefly review graph convolutional networks (GCNs) then describe how sampling operations operate and solve the scalability issue in GCNs. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ denote a graph with N nodes $v_i \in \mathcal{V}$ and edges $(v_i, v_j) \in \mathcal{E}$. Denote an adjacency matrix $A = (a(v_i, v_j)) \in \mathbb{R}^{N \times N}$ and a feature matrix $H^{(0)} \in \mathbb{R}^{N \times D^{(0)}}$ where $h_i^{(0)}$ denotes the $D^{(0)}$-dimensional feature vector of node v_i.

The GCN models stack layers of first-order spectral filters followed by a nonlinear activation functions to learn node embeddings. When $h_i^{(l)}$ denotes the hidden embeddings of node v_i in the l-th layer, the simple and general form of GCNs is as follows:

$$h_i^{(l+1)} = \alpha\left(\frac{1}{N(i)} \sum_{j=1}^{N} a(v_i, v_j) h_j^{(l)} W^{(l)}\right), \quad l = 0, \ldots, L-1 \tag{7}$$

where $a(v_i, v_j)$ is set to 1 when there is an edge from v_i to v_j, otherwise 0. $N(i) = \sum_{j=1}^{N} a(v_i, v_j)$ is the degree of node v_i; $\alpha(\cdot)$ is a nonlinear function (we commonly use ReLU); $W^{(l)} \in \mathbb{R}^{D^{(l)} \times D^{(l+1)}}$ is the learnable transformation matrix in the l-th layer with $D^{(l)}$ denoting the hidden dimension at the l-th layer.

[3] The previous version of this homework wrote "you can take $\frac{\partial \ell}{\partial h_j}$ as given." That version was indeed solveable, since $\frac{\partial \ell}{\partial h_j} = \frac{\partial \ell_{\text{recon}}}{\partial h_j}$, but it was pretty convoluted and confusing, for which we are sorry.



(a) Sampling process (b) Forward propagation (c) Backward propagation

Figure 2: GCN computation is composed of three steps: 1) sampling, 2) feedforward propagation, and 3) backpropagation. In the backpropagation process, the GCN and the sampling policy are optimized jointly to minimize the GCN performance loss.

GCNs require the full expansion of neighborhoods across layers, leading to high computation and memory costs. To circumvent this issue, sampling operations are added to GCNs to regulate the size of the neighborhood. We first recast Equation 7 as follows:

$$h_i^{(l+1)} = \alpha(\mathbb{E}_{j\sim p(j|i)}[h_j^{(l)}]W^{(l)}), \quad l = 0, \ldots, L-1 \tag{8}$$

where $p(j|i) = \frac{a(v_i,v_j)}{N(i)}$ **defines the probability of sampling** v_j **given** v_i**.** Then we approximate the expectation by Monte-Carlo sampling as follows:

$$h_i^{(l+1)} = \alpha(\frac{1}{k}\sum_{j\sim p(j|i)}^{k} h_j^{(l)}W^{(l)}), \quad l = 0, \ldots, L-1 \tag{9}$$

where k is the number of sampled neighbors for each node. Now, we regulate the size of neighborhood using k.

Now, instead of using the uniform sampler ($p(j|i) = \frac{a(v_i,v_j)}{N(i)}$), we train a new sampler $q_\theta(j|i)$ that is a trainable neural network where θ denotes parameters in q_θ. We want to learn $q_\theta(j|i)$ which maximizes the GCN performance. The key idea is that we learn a sampling policy by propagating gradients of the GCN performance loss through the non-differentiable sampling operation. Fig. 2 shows an overview. In the forward pass, GCNs sample neighbors with its sampling policy (Fig. 2(a)), then propagates their embeddings through the GCN (Fig. 2(b)). After a forward pass with sampling, the GCN computes the performance loss (e.g., cross-entropy for node classification) then back-propagates gradients of the loss (Fig. 2(c)). Now, we describe how the gradients of the loss pass through the non-differentiable sampling operation to update our sampling policy.

For concision of proofs, we omit the activation function $\alpha(x)$ **from now.** Except for the ReLU condition ($x > 0$), there is no difference in the final result. When θ denotes parameters in our sampling policy q_θ, we can write the sampling operation with $q_\theta(j|i)$ as follows:

$$h_i^{(l+1)} = \mathbb{E}_{j\sim q_\theta(j|i)}[h_j^{(l)}]W^{(l)}, \quad l = 0, \ldots, L-1 \tag{10}$$

Before being fed as input to the GCN transformation, $W^{(l)}$, the hidden embeddings go through a non-differentiable expectation under the sampling policy, which is non-differentiable.

1. (6 points) Given the loss $\mathcal{L}$ and the hidden embedding $h_i^{(l)}$ of node v_i at the l-th layer, prove the gradient of $\mathcal{L}$ w.r.t. the parameter θ of the sampling policy $q_\theta(j|i)$ is computed as follows:

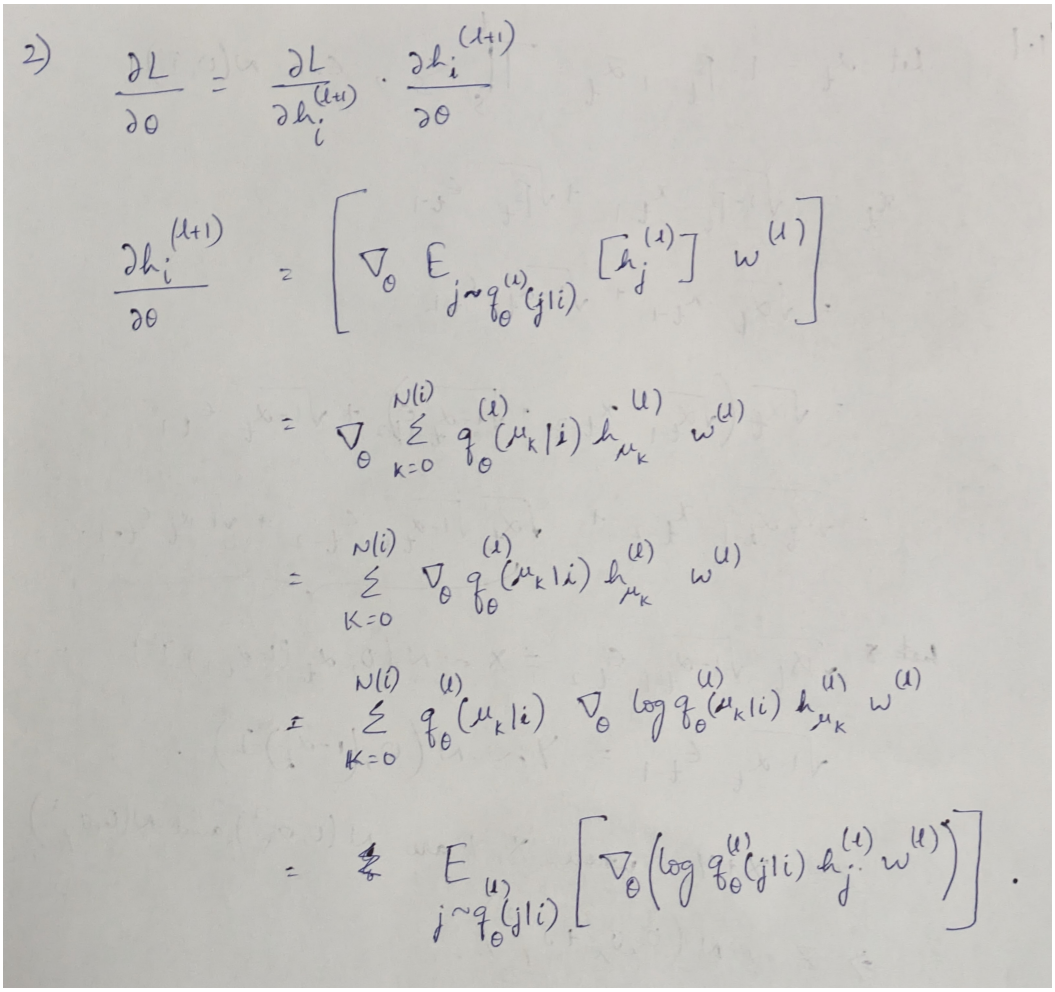
$$\nabla_\theta \mathcal{L} = \frac{d\mathcal{L}}{dh_i^{(l+1)}}\mathbb{E}_{j\sim q_\theta(j|i)}[\nabla_\theta \log q_\theta(j|i)h_j^{(l)}]W^{(l)} \tag{11}$$

Hint: to pass gradients of the loss through the expectation, we use the log derivative trick, widely used in reinforcement learning to compute gradients of stochastic policies.

$$\nabla_\theta \log q_\theta = \nabla_\theta q_\theta / q_\theta \tag{12}$$

You can assume the sampling policy $q_\theta^{(l)}(j|i)$ is defined per layer, thus $h_j^{(l)}$ is independent of θ (i.e., parameters of the sampling policy $q_\theta^{(l)}(j|i)$ at the l-th layer).

Solution



Diffusion Model (12 points)

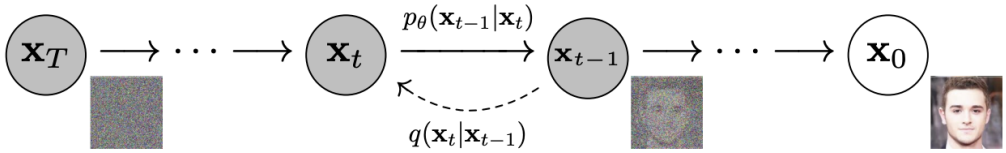


Figure 2: The directed graphical model considered in this work.

3 Diffusion Model Overview

As an overview of a diffusion model, we will be looking at the original paper on diffusion models. There have been further improvements on the model to reach the current state-of-the-art diffusion models used in stable diffusion and applications like DALL-E-2. Some of these improvements are discussed within the course lecture. However, for this problem, we are going to limit it to the basic diffusion model.

3.1 The TLDR of Diffusion Models

The spirit of diffusion models can be summarized as creating a Markov chain of steps that progressively takes an input and reduces it down to an isotropic Gaussian distribution. Then, we learn a parameterized model to learn the "de-noising" of these steps to learn the projection back to the original input. We can then sample from the isotropic Gaussian distribution and use the learned "de-noising" model to generate new samples from the input space.

A further simplification of this model is that beginning with a sample within the input space, we are taking progressive steps to leave this input space, then learning the projections that will help us return to the input space.

3.2 The Math of Diffusion Models

To begin, we have a forward stage of the algorithm. We begin with an observed input of x_0. In the forward steps, we want to progressively add noise until we achieve a unit Gaussian distribution. We will consider this noise to be parameterized by $\beta_t \in (0, 1)$.

$$q(\mathbf{x}_{1:T}|x_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}) \tag{13}$$

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}) \tag{14}$$

We also have a restriction that $\beta_0 < \beta_1 < \beta_{t-1} < \beta_t < 1$. We typically also set $\beta_t << 1$ because we want to learning small de-noising steps. Because the steps are small, the changes along the Markov Chain of latent states are easier to learn.

After the end of the forward stage, we have generated a set of latent states $x_1, x_2, ..., x_T$. In the backwards stage, we want to learn the model return from x_T to x_0.

$$p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) \tag{15}$$

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \Sigma_\theta(\mathbf{x}_t, t)) \tag{16}$$

Note that all parameters are within this backwards stage. Unlike a VAE, in which we parameterize both the encoder and decoder layers, a diffusion model holds the encoder layer as a fixed random process and only aims to learn the optimal backwards transformations.

Also similar to a VAE, we aim to learn the optimal parameters through variational inference. Recall from the VAE lectures that we can describe the varitional lower bound as:

$$\log p_\theta(x) \geq \mathbb{E}_{q(z|s)}[\log p_\theta(x|z)] - D_{KL}((q(z|x)||p_\theta(z))$$

Within the VAE, we had an observed variable of x and a latent variable of z. However, within the setting of diffusion models, we have an observed variable of x_0 and a sequence of generated latent variables of $x_{1:T}$. So, with some substitution and manipulation, we can define a cleaner variational lower bound:

$$\log p_\theta(x) \geq \mathbb{E}_{q(z|x)}[\log p_\theta(x|z)] - D_{KL}((q(z|x)||p_\theta(z)) \tag{17}$$
$$\geq \mathbb{E}_{q(x_{1:T}|x_0)}[\log p_\theta(x_0|x_{1:T})] - D_{KL}((q(x_{1:T}|x_0)||p_\theta(x_{1:T})) \tag{18}$$
$$\geq ... \tag{19}$$
$$\geq \mathbb{E}_q\left[\log \frac{p_\theta(x_{0:T})}{q(x_{1:T}|x_0)}\right] \tag{20}$$

4 Problems

1. Consider the forward diffusion process, where in each step, a small Gaussian noise is added to the current timestep's sample:

$$\mathbf{x_{t+1}}|\mathbf{x_t} \sim \mathcal{N}(\sqrt{1-\beta_t}\mathbf{x_t}, \beta_t\mathbf{I}),$$
$$\text{i.e.} q(\mathbf{x_{t+1}}|\mathbf{x_t}) \sim \mathcal{N}(\mathbf{x_{t+1}}; \sqrt{1-\beta_t}\mathbf{x_t}, \beta_t\mathbf{I}),$$

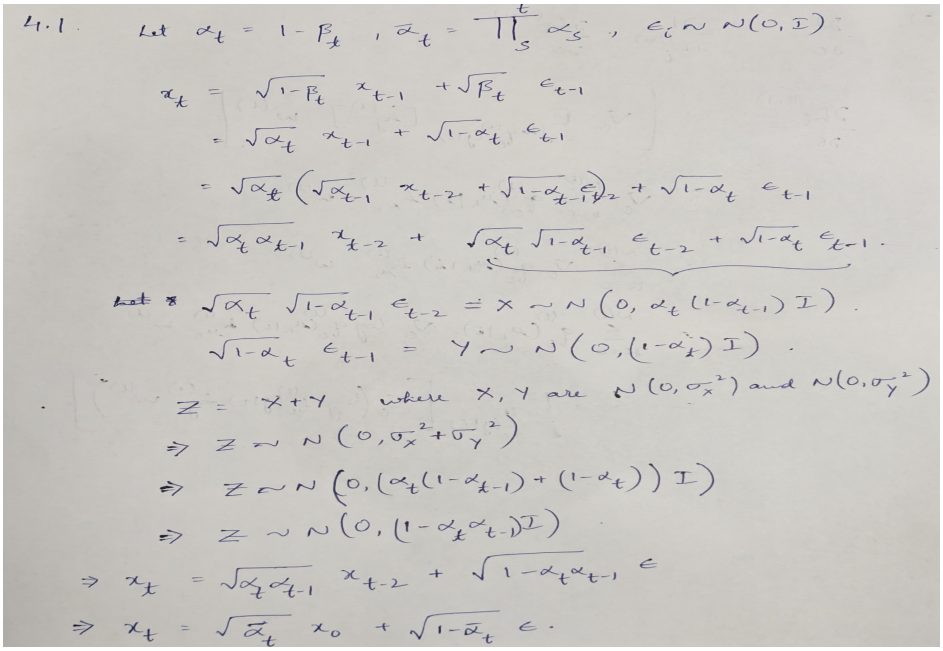
where $\mathbf{x}_0$ is sampled from the data distribution (e.g. an image), and $t = 0, 1, 2, \ldots T$.

Let β_t be a constant s.t. $\beta_t = \beta \in (0, 1)$, and assume a given, fixed initial data sample $\mathbf{x}_0$.

(a) (**4 points**) First, derive a closed form expression for the distribution of $\mathbf{x}_t|\mathbf{x}_0$. Write your answer in terms of $\beta, t, \mathbf{x}_0$, and $\mathbf{I}$.

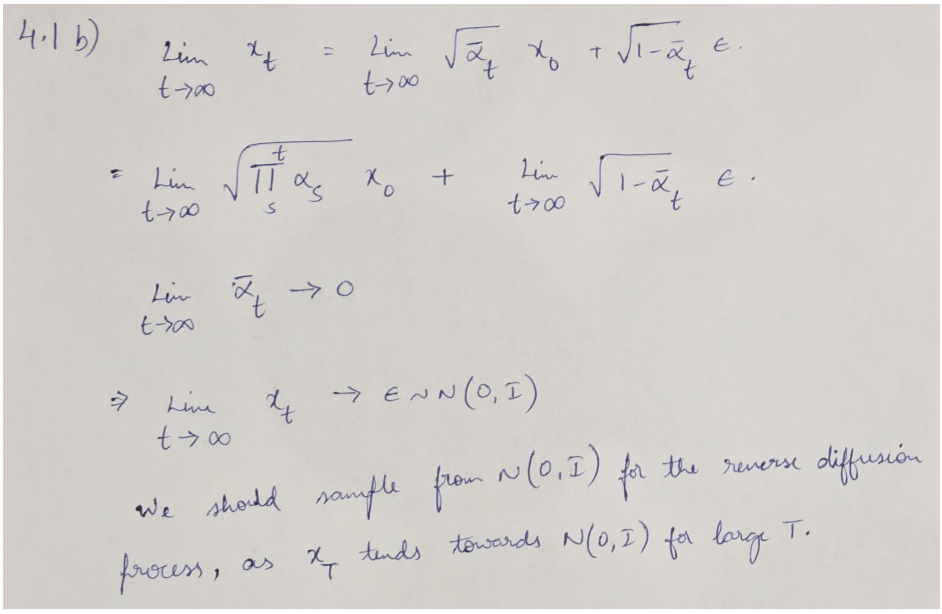
Hint: use the reparameterization trick to express $\mathbf{x_{t+1}}$ *as the summation of a random variable involving* $\mathbf{x_t}$ *and another random variable* $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.

Solution



(b) (**2 points**) Prove that as $T \to \infty$, $q(\mathbf{x}_T|\mathbf{x}_0)$ converges to an isotropic Gaussian distribution, and state what this distribution is (i.e. the parameters of this distribution). Afterwards, comment on how the reverse diffusion process should be initiated, i.e. what distribution should you sample from to begin the reverse diffusion process?

Solution



2. (**6 points**) As is the case with most generative models, diffusion models are trained by maximizing the likelihood of the data given the model parameters. The forward diffusion process simply adds noise and is a fixed process (with fixed parameters that we know ahead of time), whereas the reverse diffusion process is non-trivial, and this is the process that is learned with a model parameterized with θ.

   Define all forward steps with q, and all reverse steps with p. Since the reverse process is learned, we denote it with the learned parameters, p_θ.

   As stated above, our objective is

   $$\max_\theta p_\theta(\mathbf{x}_0),$$

   or equivalently,

   $$\max_\theta \log p_\theta(\mathbf{x}_0).$$

   Similar to VAE's, we can derive a variational lower bound to derive a tractable loss function:

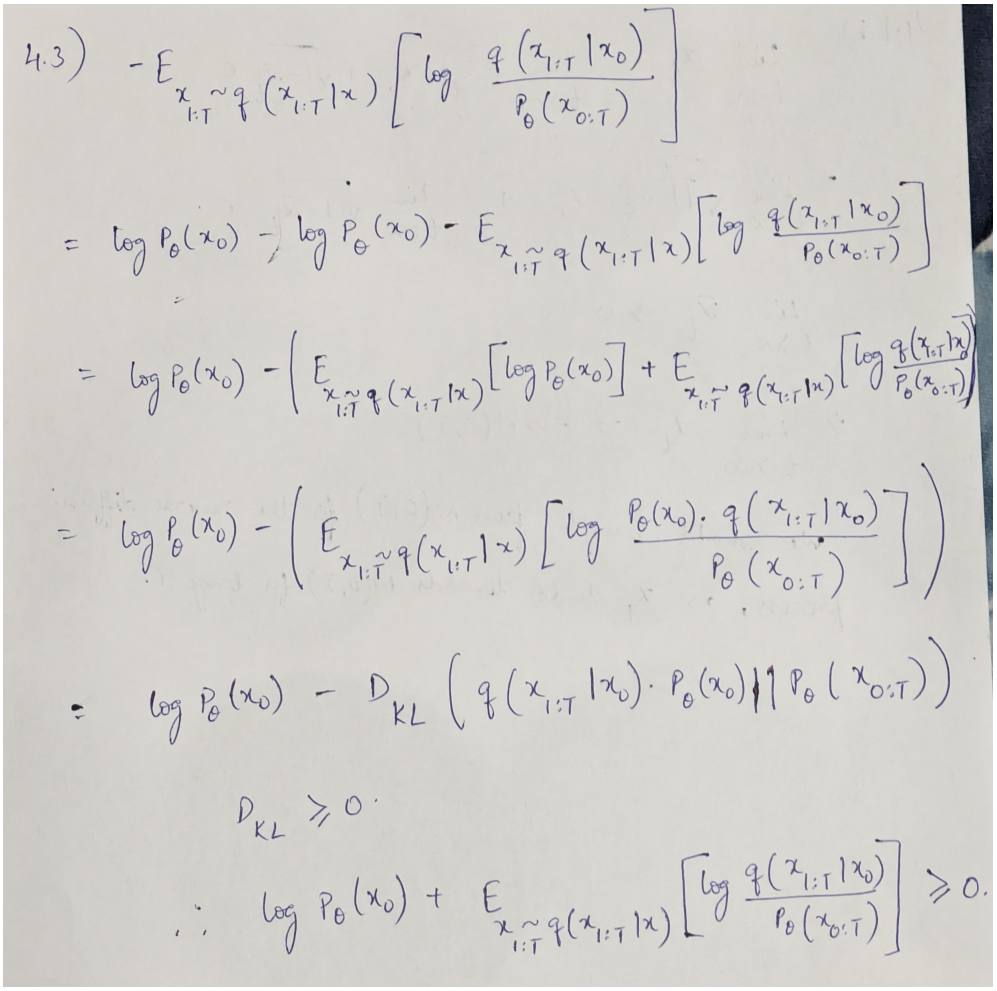
   $$-\mathbb{E}_{\mathbf{x}_{1:T}\sim q(\mathbf{x}_{1:T}|\mathbf{x}_)}\left[\log \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})}\right] \leq \log p_\theta(\mathbf{x}_0),$$

   where $q(\mathbf{x}_{1:T}|\mathbf{x}_0)$ denotes the distribution over the full random sequence $\mathbf{x}_{1:T}$ given $\mathbf{x}_0$.

   Prove that $-\mathbb{E}_{\mathbf{x}_{1:T}\sim q(\mathbf{x}_{1:T}|\mathbf{x}_)}\left[\log \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})}\right]$, in fact lower bounds $\log p_\theta(\mathbf{x}_0)$. Be explicit with all steps taken and do not skip any derivation steps. Explicitly state the random variable each expectation is taken over.

   Hint: Begin by adding and subtracting $\log p_\theta(\mathbf{x}_0)$ *from* $-\mathbb{E}_{\mathbf{x}_{1:T}\sim q(\mathbf{x}_{1:T}|\mathbf{x}_)}\left[\log \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})}\right]$, *i.e.* $-\mathbb{E}_{\mathbf{x}_{1:T}\sim q(\mathbf{x}_{1:T}|\mathbf{x}_)}\left[\log \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})}\right] = \log p_\theta(\mathbf{x}_0) - \log p_\theta(\mathbf{x}_0) - \mathbb{E}_{\mathbf{x}_{1:T}\sim q(\mathbf{x}_{1:T}|\mathbf{x}_)}\left[\log \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})}\right]$.

Solution


4.3) $-E_{x_{1:T}\sim q(x_{1:T}|x)}\left[\log \frac{q(x_{1:T}|x_0)}{P_\theta(x_{0:T})}\right]$
$= \log P_\theta(x_0) - \log P_\theta(x_0) - E_{x_{1:T}\sim q(x_{1:T}|x)}\left[\log \frac{q(x_{1:T}|x_0)}{P_\theta(x_{0:T})}\right]$
$= \log P_\theta(x_0) - \left(E_{x_{1:T}\sim q(x_{1:T}|x)}[\log P_\theta(x_0)] + E_{x_{1:T}\sim q(x_{1:T}|x)}\left[\log \frac{q(x_{1:T}|x_0)}{P_\theta(x_{0:T})}\right]\right)$
$= \log P_\theta(x_0) - \left(E_{x_{1:T}\sim q(x_{1:T}|x)}\left[\log \frac{P_\theta(x_0)\cdot q(x_{1:T}|x_0)}{P_\theta(x_{0:T})}\right]\right)$
$= \log P_\theta(x_0) - D_{KL}\left(q(x_{1:T}|x_0)\cdot P_\theta(x_0) \,||\, P_\theta(x_{0:T})\right)$
$D_{KL} \geq 0.$
$\therefore \log P_\theta(x_0) + E_{x_{1:T}\sim q(x_{1:T}|x)}\left[\log \frac{q(x_{1:T}|x_0)}{P_\theta(x_{0:T})}\right] \geq 0.$

Collaboration Questions Please answer the following:

After you have completed all other components of this assignment, report your answers to the collaboration policy questions detailed in the Academic Integrity Policies found here.

1. Did you receive any help whatsoever from anyone in solving this assignment? Is so, include full details.
2. Did you give any help whatsoever to anyone in solving this assignment? Is so, include full details.
3. Did you find or come across code that implements any part of this assignment ? If so, include full details even if you have not used that portion of the code.

Solution

Referred to online blogs for better understanding of Diffusion models slides.
Referred to the paper by Minji Yoon for GCN paper.
Referred to Article for VAE.