Intro to Diffusion Models

Part 0: Image Distributions

- Suppose I live in a world of 32 x 32 grayscale images of cats and that I have a dataset of 10k of them
- Each picture looks like this and is easily flattened to a 1 by 1024 vector
- Imagine if we plot them in $\mathbb{R}^{1024}$ and construct a density function which we'll visualize in $\mathbb{R}^{1025}$
- This distribution is what we're trying to learn
- The higher the density near an image X, the more images in our dataset resemble image X
- This distribution is complicated and high dimensional, but if we were able to learn the probability surface that "connects the dots" then we'd be able to sample from the distribution at will and generate new cat images
- A diffusion model is a model that approximates this "underlying image distribution" so that we can sample from it
    - We'll call this model p_θ where θ tells us that we are learning parameters that will help us approximate underlying image distribution
- For simplicity and because Gaussian mixtures are pretty versatile, we're going to work with multivariate gaussian distributions to try to approximate what's going on here

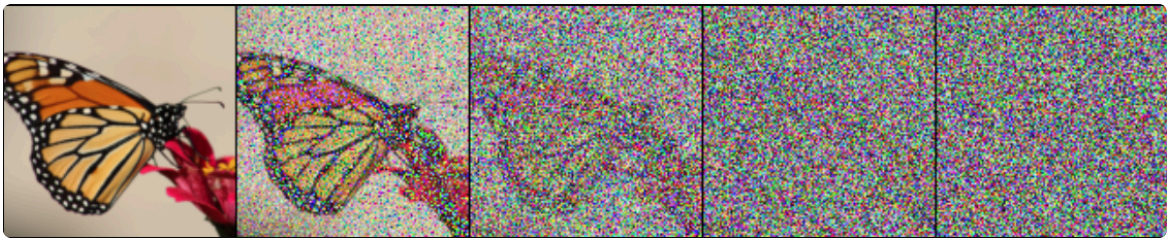
Part 1: Brief Gaussian Overview

- First, some notation
    - A lot of diffusion papers like to write things like $q(x_t|x_{t-1}) = \mathcal{N}(x_t|\sqrt{1-\beta_t}x_{t-1}, \beta_t I)$ which I find pretty confusing, so I'll instead try to write this
    - $q(x_t|x_{t-1})$ is a density function which tells you how much probability mass is under any given part of the distribution
    - Suppose $Z \sim \mathcal{N}(\mu, \sigma^2)$ then $p_Z(z) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\mu)^2/2\sigma^2}$
        - We can do location transformations by changing the mean and scale transformations by changing the variance
    - The same idea holds for higher dimensions. Suppose we have a vector $\vec{Y} = [Y_1, Y_2, \ldots, Y_n]$ where each of these components is marginally normal and any linear combination of them is normal

- This is a multivariate normal distribution, which we specify with a mean vector $\vec{\mu}$ and a covariance matrix, which is a $n \times n$ symmetric matrix where the $(i, j)th$ entry gives the covariance between the ith and jth component of $\vec{Y}$
- It also has an explicit density form that's pretty gross and not super illuminating

Part 2: The Two Big Ideas of Diffusion

- The two key insights that make it possible for diffusion models to approximate this underlying image distribution are "taking small steps" and "reversing time"
  - The first is that instead of trying to sample from the whole image distribution in one big sweep, instead we think of taking small steps towards our goal by starting from noise and getting closer and closer to a picture



  - It turns out, somewhat unsurprisingly, that starting from noise and getting signal is much harder than going the other direction, so let's start by adding noise and then trying to figure out how to get from noise to image

Part 3: What do Diffusion Models Actually Do?

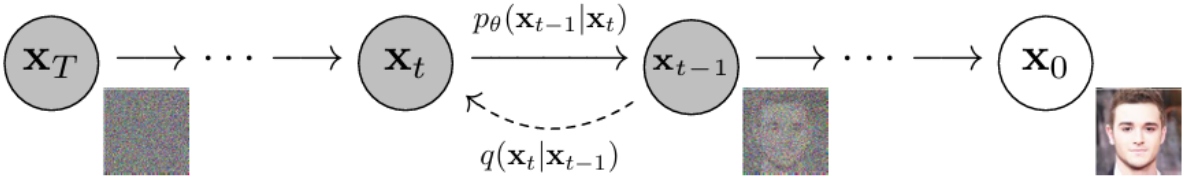


Figure 2: The Markov Chain showing the bi-directional diffusion process (Image source: Ho et al. 2020)

- This is the classic diffusion picture, so let's break it down a bit
- The idea is to design a noising process that we can understand really well analytically so that we can then approximate the reverse process with a neural network
- The goal of all this work is to get a good approximation p_θ of the reverse process during training so we can then put a random noise sample through p_θ and recover something that looks very much like a sample from our data distribution

- One thing to keep in mind is that we don't actually want to do "too good a job of this"
    - What I mean by this is that if we design a perfect reverse process, we'll always end up back at one of the initial training images we put in
        - There's a lot of literature out there on this now, but the big idea is that a lot of the remarkable creativity in diffusion models comes from small approximation errors in the reverse process that allow us to end up somewhere between two training pictures and generate something new

Part 4: The Forward Process: Adding Noise

- We want to make a couple of simplifying assumptions so we can solve this direction analytically
- First, we have our Gaussian assumption from above -- this is going to help simplify things a lot
- Second, we're going to introduce what's called the Markov Property
    - Intuitively, the Markov property says that given the present the future is independent of the past
    - Mathematically, we have $q(x_t|x_0, x_1, x_2, \ldots, x_{t-1}) = q(x_t|x_{t-1})$
    - This is really useful for lots of reasons, but especially because it makes this model a lot easier to train and use -- we don't have to store every previous state in order to get to the next one
        - Basically, it massively simplifies training and inference and allows nice, closed form solutions
    - Also, it works to produce nice results!

Part 4a: How does the Forward Process Actually Work?

- First, we pick a "variance schedule," which is just an increasing list of small numbers between 0 and 1 which tells us how much noise we add at each step
    - $\{\beta_{t\in(0,1)}\}_{t=1}^{T}$ where T is the time at which you've finally added enough noise that the image is indistinguishable from pure Gaussian Noise
    - We want $0 < \beta_0 < \beta_1 < \ldots < \beta_T < 1$
        - People often pick a linear schedule or other kinds
    - Side Note: It's also possible to learn an "optimal" variance schedule but this is harder -- this is usually called Variational Diffusion
- Now we'll actually define the forward process

    - We call this process $q(x_t|x_{t-1})$ and it follows a normal distribution $X_t|X_{t-1} \sim \mathcal{N}(\sqrt{1-\beta_t}x_{t-1}, \beta_t I)$
        - Why does this make sense? It means that at the next time step, we pick a picture pretty similar to the one at the previous time step but slightly shrunk towards 0, because we want to end up with pure gaussian noise at time T $X_T \sim \mathcal{N}(0, I)$
        - Our $\sqrt{1-\beta_t}$ is very close to 1 and our variance is very small for β_t small
- Now recall that we're interested in the joint distribution over time, so we actually care not about $q(x_t|x_{t-1})$ but about $q(x_0, x_1, x_2, \ldots, x_T)$
- Here we can use the Chain Rule of Probability and the Markov assumption to help us simplify this a lot
    - $q(x_0, x_1, x_2, \ldots, x_T) = q(x_0)q(x_1|x_0)q(x_2|x_1, x_0)q(x_3|x_2, x_1, x_0)\ldots q(x_T|x_{T-1}, x_{T-2}, \ldots, x_2, x_1, x_0)$
    - Then, applying the Markov Property, we have
        - $q(x_0, x_1, x_2, \ldots, x_T) = \prod_{t=1}^{T} q(x_t|x_{t-1})$
    - the TLDR is that we eventually get this lovely closed from solution that looks like $X_t|X_0 \sim \mathcal{N}(\sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)I)$
        - where $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{s=1}^{t}(1-\beta_t)$
    - This comes from a neat application of a re-parametrization trick (quite similar to the one used for VAEs)
        - Check out weng2021diffusion for a full derivation

Part 5: The Reverse Process!

- Great! So we have everything to do with adding noise in nice analytical closed form, but now we're on to the tricky bit
- First, we need to understand why this is a hard problem. After all, we've made a lot of simplifying assumptions
    - We need to learn the reverse diffusion process -- we can't easily solve this analytically, because it's really hard to find explicitly what $q(x_{t-1}|x_t)$ is -- it depends on the entire previous set of steps and is confusing.
    - Crucially, the Markov property is not necessarily true in reverse, so we might need all previous states
- This is where our neural network to parametrize p_θ comes in!
    - Often, this is a u-net or even more simply, just a convolutional neural network
    - We'll often include self-attention layers here
- We can get as far as saying that our joint distribution looks like
    - $p_\theta(x_0, x_1, x_2, \ldots, x_T)$ looks like $p(x_T)\prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)$

    - Where we know that $p_\theta(x_{t-1}|x_t)$ is given by $X_{t-1}|X_t \sim \mathcal{N}(\mu_\theta(x_t,t), \Sigma_t(x_t,t))$
    - We want to learn what those parameters are across all time steps
    - With a lot of analytical work, we find that $\mu_t = \frac{1}{\sqrt{\alpha_t}}(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_t)$ where $\epsilon_t \sim \mathcal{N}(0, I)$
        - This analytic form and especially the random vector ϵ_t is going to be really important later in training the model
- From here we have a few options for our trainable parameters
    - First we can simply fix $\Sigma_\theta(x_t, t) = \sigma_t^2 I$ and learn σ_t^2 (which is inadvisable, since the empirical results aren't that good), or fix $\sigma_t^2 = \beta_t$ or $\sigma_t^2 = f(\beta_t)$ for some fixed, predetermined, function f which you can pick from a variety of sources in the literature
- Now, we move on to what we train and what loss we use
    - We want to find the reverse Markov transitions that maximize the likelihood of the training data
    - This means that we want to maximize the log likelihood of p_θ
        - What that means intuitively is optimizing over the parameter space for values of the parameters that make the data we observe as likely as possible (this is just maximum likelihood estimation)
        - We take logs here because it turns multiplicative relationships into additive ones, which usually simplifies the optimization, and it's a monotonic transformation
    - Unfortunately, maximizing over $\log p_\theta$ itself too difficult of a problem to be solved practically.
        - Instead, we choose to minimize an upper bound on the NEGATIVE log likelihood which is equivalent in spirit to simply maximizing the likelihood
        - There are a few different ways to derive this upper bound, but the upshot is that it ends up being $L_{VLB} = \mathbb{E}_{q(x_0:T)}[\log(\frac{q(x_{1:T}|x_0)}{p_\theta(x_{0:T})})]$
    - Then, the training algorithm works as follows:
        - Repeat until the process converges
            - pick a random image x_0 from your training data
            - uniformly randomly sample a time step between 0 and T
            - pick a sample $\epsilon \sim \mathcal{N}(0, I)$
            - Take a gradient descent step on $\nabla_\theta ||\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t)||^2$
- Then, the sampling algorithm works exactly as you'd hope
    - First pick some random noise $x_T \sim \mathcal{N}(0, I)$
    - for t from 1 to T pick
        - $Z \sim \mathcal{N}(0, I)$
        - set $x_{t-1} = \frac{1}{\sqrt{\alpha_t}}(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t)) + \sigma_t Z$

- Return a picture of a cat x_0!!!!!

Part 6: Bibliography!

@misc{dieleman2023perspectives,
author = {Dieleman, Sander},
title = {Perspectives on diffusion},
url = {https://sander.ai/2023/07/20/perspectives.html},
year = {2023}
}
@book{pml2Book,
author = "Kevin P. Murphy",
title = "Probabilistic Machine Learning: Advanced Topics",
publisher = "MIT Press",
year = 2023,
url = "http://probml.github.io/book2"
}
@article{weng2021diffusion, title = "What are diffusion models?", author = "Weng, Lilian",
journal = "lilianweng.github.io", year = "2021", month = "Jul", url =
"https://lilianweng.github.io/posts/2021-07-11-diffusion-models/" }
@misc{ho2020denoisingdiffusionprobabilisticmodels,
title={Denoising Diffusion Probabilistic Models},
author={Jonathan Ho and Ajay Jain and Pieter Abbeel},
year={2020},
eprint={2006.11239},
archivePrefix={arXiv},
primaryClass={cs.LG},
url={https://arxiv.org/abs/2006.11239},
}

https://dzdata.medium.com/intro-to-diffusion-model-part-4-62bd94bd93fd