CSC 412/2506: Probabilistic Learning and Reasoning

Week 13 - 1/2: Diffusion Models

Michal Malyska

University of Toronto

Overview

- VAE Recap
- Intuition behind diffusions
- Diffusion modelling
- Simplifications
- Guided Diffusion
- Latent / Stable diffusion

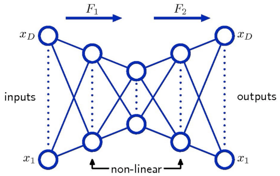
Recap: Autoencoders

Autoencoders reconstruct their input via an encoder and a decoder.

- **Encoder**: $g(x) = z \in F, \quad x \in X$
- **Decoder**: $f(z) = \tilde{x} \in X$
- where X is the data space, and F is the feature (latent) space.
- z is the code, compressed representation of the input, x. It is important that this code is a bottleneck, i.e. that

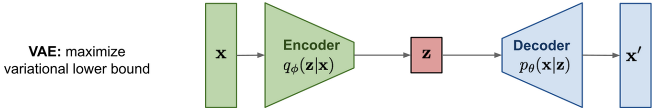
$$\dim F \ll \dim X$$

- Goal: $\tilde{x} = f(g(x)) \approx x$.



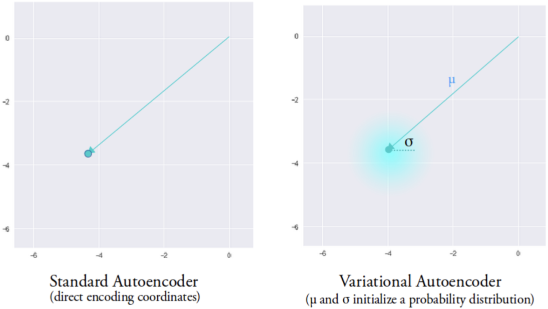
Variational Autoencoders

- Variational autoencoders (VAEs) encode inputs with uncertainty.
- Unlike standard autoencoders, the encoder of a VAE outputs a probability distribution, $q_\phi(z|x)$.
- Instead of the encoder learning an encoding vector, it learns two vectors: vector of means, μ, and another vector of standard deviations, σ.



Variational Autoencoders

- The mean μ controls where encoding of input is centered while the standard deviation controls how much can the encoding vary.



- Encodings are generated at random from the “circle”, the decoder learns that all nearby points refer to the same input.

Image credit: I. Shafkat

VAE vs Amortized VAE Pipeline

- For a given input (or minibatch) x_i,

  - **Standard VAE**
  - Sample $z_i \sim q_{\phi_i}(z|x_i) = \mathcal{N}(\mu_i, \sigma_i^2 I)$.

  - **Amortized VAE**
  - Sample $z_i \sim q_\phi(z|x_i) = \mathcal{N}(\mu_\phi(x_i), \Sigma_\phi(x_i))$

- Run the code through decoder and get likelihood: $p_\theta(x|z)$.
- Compute the loss function: $L(x;\theta,\phi) = -E_{z_\phi \sim q_\phi}\Big[\log p_\theta(x|z)\Big] + KL(q_\phi(z|x)||p(z))$
- Use gradient-based optimization to backpropogate $\nabla_\theta L$, $\nabla_\phi L$

Physical Intuition

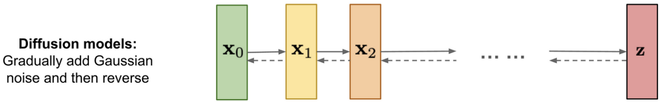
- Observation 1: Diffusion destroys structure.
- Think of a jar of water with a fresh drop of dye in it.
- Dye represents the probability density
- Goal: Learn structure of the probability density
- If we allow the diffusion to run long enough we end up with a uniform distribution of dye in the water.

What if we could reverse time?

- Recover data distribution by starting with a uniform distribution and running dynamics backwards

Adding Gaussian Noise

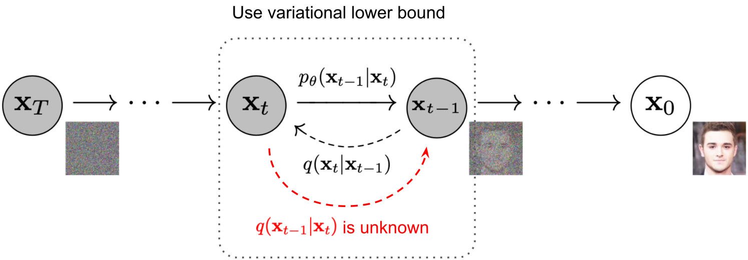
- In Brownian motion postion updates are small gaussians
- Both forward and backward in time!
- We can destroy our images with a large number of small gaussian updates.
- The reverse updates (from noise to data) should also be gaussian!
- We will try to learn a model that can estimate the mean and covariance of each step in the reverse process



Forward Diffusion

Given a data point sampled from a real data distribution $x_0 \sim q(x)$, let us define a **forward diffusion process** in which we add small amount of Gaussian noise to the sample in T steps, producing a sequence of noisy samples $x_1, \ldots, x_T$.

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t\mathbb{I})$$



Forward Diffusion

Can we do better than applying a gaussian 500 times in a row? Yes! Begin by defining: $\alpha_t = 1 - \beta_t$, $\bar{\alpha_t} = \prod_s^t \alpha_s$, $\epsilon_i \sim \mathcal{N}(0, \mathbb{I})$ Note that:

$$
\begin{aligned}
x_t &= \sqrt{1-\beta_t}x_{t-1} + \sqrt{\beta_t}\epsilon_{t-1} \\
&= \sqrt{\alpha_t}x_{t-1} + \sqrt{1-\alpha_t}\epsilon_{t-1} \\
&= \sqrt{\alpha_t\alpha_{t-1}}x_{t-2} + \sqrt{1-\alpha_t\alpha_{t-1}}\bar{\epsilon}_{t-2} \\
&= \dots \\
&= \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\bar{\epsilon}
\end{aligned}
$$

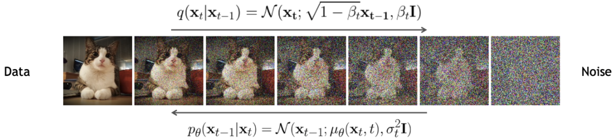
So if we want to have a diffusion at time T we can now get there in one single step

Reverse Diffusion

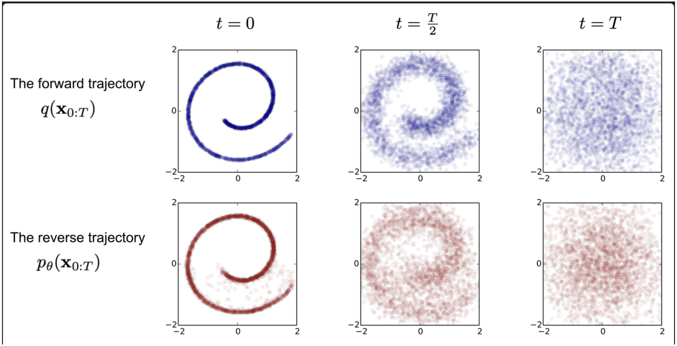
Now, if we can reverse the whole process, and sample from $q(x_{t-1}|x_t)$ we will be able to go from $\mathcal{N}$ to our data distribution. We can approach it similarily to what we did with VAEs - use a model p_θ to approximate these conditional probabilities.

$$p_\theta(x) = p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)$$

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



Reverse Diffusion



Model Fitting

How do we fit the model? ELBO.

$$logp_\theta(x_0) \geq \mathbb{E}_q \left[logp(x_T) + \sum_{t=1}^{T} log \frac{p_\theta(x_{t-1}|x_t)}{q(x_t|x_{t-1})} \right]$$

We can write the variational lower bound loss as:

$$L_{VLB} = L_T + L_{T-1} + \cdots + L_0$$

Where:

$$L_T = D_{KL}(q(x_T|x_0)||p_\theta(x_T))$$
$$L_t = D_{KL}(q(x_t|x_{t+1}, x_0)||p_\theta(x_{t-1}|x_t))$$
$$L_0 = -logp_\theta(x_0|x_1)$$

Model Fitting

$$L_T = D_{KL}(q(x_T|x_0)||p_\theta(x_T))$$
$$L_t = D_{KL}(q(x_t|x_{t+1}, x_0)||p_\theta(x_{t-1}|x_t))$$
$$L_0 = -log p_\theta(x_0|x_1)$$

Notice that all the KL divergences are comparing gaussian distributions. This means we have a closed form solution!

L_T is constant and can be ignored since q has no learnable parameters, and x_T is a Gaussian noise.

Parametrizing L_t

Recall that we need to learn a neural network to approximate the conditioned probability distributions in the reverse diffusion process:

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

We would like to train μ_θ to predict:

$$\tilde{\mu_t} = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon_t \right)$$

Since x_t is an input at training time, we reparametrize the gaussian noise term to make it predict ϵ_t from the input x_t at time step t:

$$\tilde{\mu_t} = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon_\theta(x_t, t) \right)$$

Simplification

The loss term L_t then becomes:

$$\begin{aligned} L_t &= \mathbb{E}_{x_0,\epsilon}\left[\frac{1}{2||\Sigma_\theta(x_t,t)||_2^2}||\tilde{\mu}_t(x_t,x_0)-\mu_\theta(x_t,t)||^2\right] \\ &= \mathbb{E}_{x_0,\epsilon}\left[\frac{(1-\alpha_t)^2}{2\alpha_t(1-\bar{\alpha}_t)||\Sigma_\theta||_2^2}||\epsilon_t-\epsilon_\theta(\sqrt{\bar{\alpha}_t}x_0+\sqrt{1-\bar{\alpha}_t}\epsilon_t,t)||^2\right] \end{aligned}$$

In practice it has been found that the unweighted loss term performs better:

$$\begin{aligned} L_t^{simple} &= \mathbb{E}_{t\sim[1,T],x_0,\epsilon_t}\left[||\epsilon_t-\epsilon_\theta(x_t,t)||^2\right] \\ &= \mathbb{E}_{t\sim[1,T],x_0,\epsilon_t}\left[||\epsilon_t-\epsilon_\theta(\sqrt{\bar{\alpha}_t}x_0+\sqrt{1-\bar{\alpha}_t}\epsilon_t,t)||^2\right] \end{aligned}$$

Algorithm 1 Training

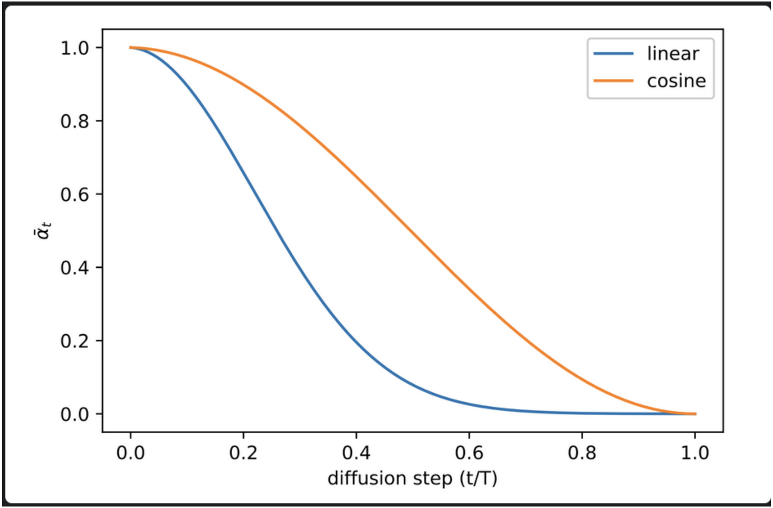
repeat
  $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
  $t \sim \text{Uniform}(\{1, \ldots, T\})$
  $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  Take gradient descent step on
    $\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$
until converged

Algorithm 2 Sampling

$\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
for $t = T, \ldots, 1$ **do**
  $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
  $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
end for
return $\mathbf{x}_0$

Choosing β_t

How do we choose the noise parameter β_t ?



Conditioned Generation

Generating novel images is cool, but generating novel images of specific things is even cooler. How can we do that? We turn our diffusion model into a conditional diffusion model:

$$p_\theta(x|y) = p(x_T|y)\prod_{t=1}^{T} p_\theta(x_{t-1}|x_t, y)$$

In general we aim to learn:

$$\begin{aligned}\nabla_{x_t} log p_\theta(x_t|y) &= \nabla_{x_t} log \frac{p_\theta(y|x_t)p_\theta(x_t)}{p_\theta(y)} \\ &= \nabla_{x_t} log p_\theta(x_t) + \nabla_{x_t} log p_\theta(y|x_t)\end{aligned}$$

which we usually modify by adding a scaling term s

$$\nabla_{x_t} log p_\theta(x_t) + s\nabla_{x_t} log p_\theta(y|x_t)$$

Conditioned Generation

It has been shown that instead, we can use an already trained classifier $f_\phi(y|x_t, t)$ to guide the diffusion:

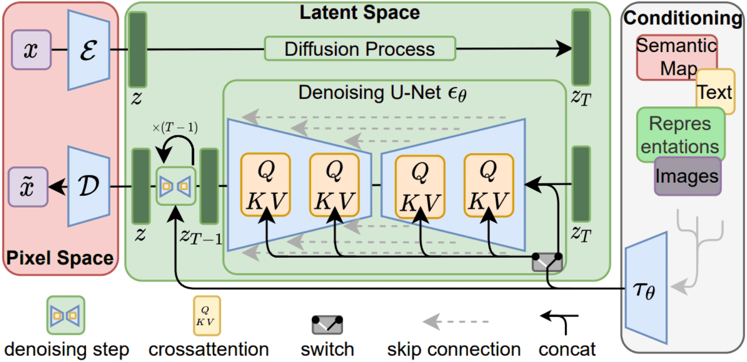
$$\mu_t(x_t|y) = \mu_\theta(x_t|y) + s\Sigma_\theta(x_t|y)\nabla_{x_t} log f_\phi(y|x_t, t)$$

Or given an image embedding $g(x)$ and text embedding $h(c)$ model like CLIP:

$$\mu_t(x_t|c) = \mu_\theta(x_t|c) + s\Sigma_\theta(x_t|c)\nabla_{x_t} g(x_t) \cdot h(c)$$

Latent Diffusion

Even for a 64x64 image, running a diffusion model for a large number of steps will be very expensive. To combat this problem, we can train an autoencoder, and do the diffusion in the latent space:



Summary

- Diffusion models work by gradually adding gaussian noise through a series of T steps into the original image, a process known as diffusion.
- To sample new data, we approximate the reverse diffusion process using a neural network.
- The training of the model is based on maximizing the ELBO
- Latent diffusion models (like stable diffusion) apply the diffusion process on a smaller latent space for computational efficiency using a variational autoencoder for the up and downsampling.