Bayesian multimodeling: diffusion

2024

What is diffusion?

A process X_t is a diffusion if it has these properties:

- The time variable t is continuous;
- It is a Markov process;
- X_t is a continuous function of t.

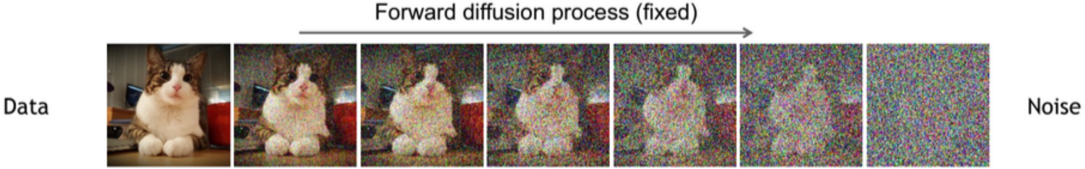
A diffusion process X_t can be expressed in Ito form:

$$dX_t = a(X_t)dt + b(X_t)dW_t,$$

where a is a drift, b is a noise coefficient, W_t is a Wiener process.

Diffusion forward process

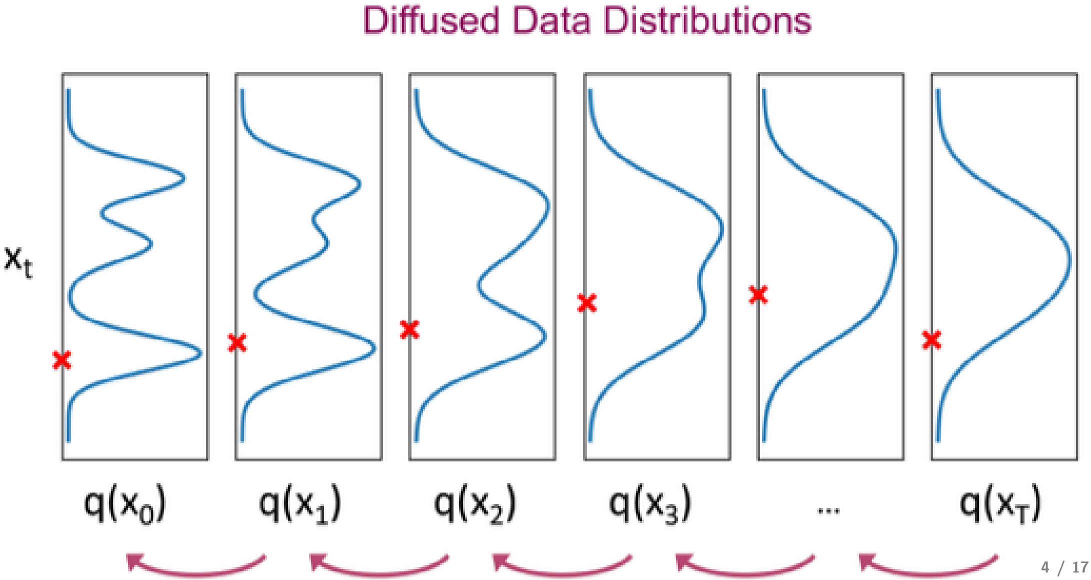
$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta\mathbf{I}).$$



$$\alpha_t = \prod \beta_t, \quad q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\sqrt{\alpha_t}, (1-\alpha_t)\mathbf{I}),$$

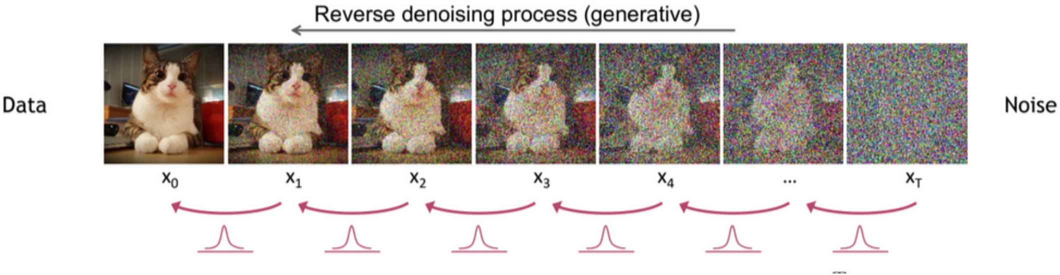
β_t is set for $\alpha_t \to 0$, and $q(\mathbf{x}_t|\mathbf{x}_0) \approx \mathcal{N}(\mathbf{0}, \mathbf{I})$.

Diffusion forward process



Diffusion backward process

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\boldsymbol{\mu}(\mathbf{x}_t), \sigma^2\mathbf{I}).$$



Diffusion as variational model

ELBO:

$$\mathsf{E}_q \log \frac{p_{\boldsymbol{\theta}}(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} =$$

$$= \mathsf{E}_q \log p_{\boldsymbol{\theta}}(\mathbf{x}_0|\mathbf{x}) - \sum_t KL(q(\mathbf{x}_{t-1}|\mathbf{x}_t)|p_{\boldsymbol{\theta}}(\mathbf{x}_{t-1}|\mathbf{x}_t)) + KL(q(\mathbf{x}_T|\mathbf{x}_0)|p_{\boldsymbol{\theta}}(\mathbf{x}_T).$$

Sum of KL can be computed by explicit formula:

$$KL(q(\mathbf{x}_{t-1}|\mathbf{x}_t)|p_{\boldsymbol{\theta}}(\mathbf{x}_{t-1}|\mathbf{x}_t)) = \mathrm{Const} + \mathsf{E}_q \frac{1}{2\sigma^2} \left\| \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}} \varepsilon \right) - \boldsymbol{\mu}(\mathbf{x}_t) \right\|^2.$$

Diffusion model training

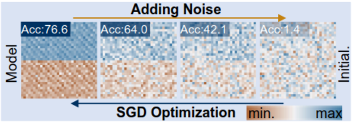
Algorithm 1 Training

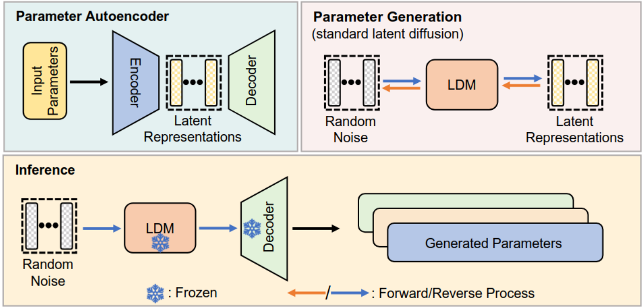
1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \mathrm{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0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

Algorithm 2 Sampling

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

Model parameters generation using diffusion models





Hypernetworks

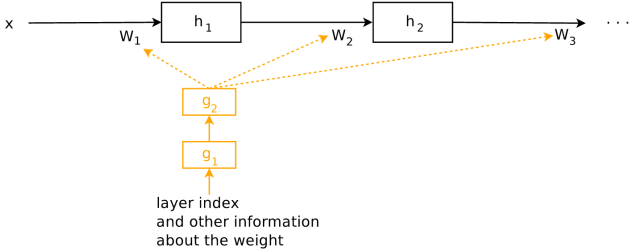
Definition

Given a set Λ.

Hypernetwork is a parametric mapping from Λ to set $\mathbb{R}^n$ of the model $\mathbf{f}$ parameters:

$$G : \Lambda \times \mathbb{R}^u \to \mathbb{R}^n,$$

where $\mathbb{R}^u$ is a set of hypernetwork parameters.



Ha et al., 2016

Diffusion and score matching

Score matching idea:

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = \frac{1}{Z(\boldsymbol{\theta})} exp\left(-E(\mathbf{x};\boldsymbol{\theta})\right).$$

To approximate unknown true distribution $p(\mathbf{x})$ we try to match scores of the distribution:

$$\mathsf{E}_{\mathbf{x}\in p(\mathbf{x})}\|\frac{\partial \log p_{\boldsymbol{\theta}}(\mathbf{x})}{\partial \mathbf{x}} - \frac{\partial \log p(\mathbf{x})}{\partial \mathbf{x}}\|^2 \to \min.$$

$$\approx \mathsf{E}_{\hat{\mathbf{x}},\mathbf{x}\in q(\hat{\mathbf{x}},\mathbf{x})}\|\frac{\partial \log p_{\boldsymbol{\theta}}(\mathbf{x})}{\partial \mathbf{x}}|_{\hat{\mathbf{x}}=\mathbf{x}} - \frac{\partial \log p(\hat{\mathbf{x}}|\mathbf{x})}{\partial \hat{\mathbf{x}}}\|^2 \to \min,$$

where $q(\hat{\mathbf{x}}) = q(\hat{\mathbf{x}}|\mathbf{x})p(\mathbf{x})$.

Autoencoder: generative model?

(Alain, Bengio 2012): consider regularized autoencoder:

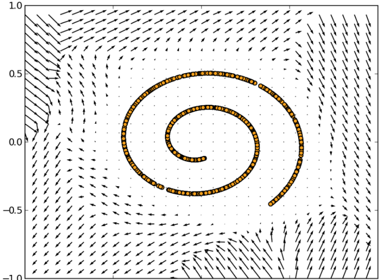
$$||\mathbf{f}(\mathbf{x}, \sigma) - \mathbf{x}||^2,$$

where σ is a noise level.

Then

$$\frac{\partial \log p(\mathbf{x})}{\partial \mathbf{x}} = \frac{||\mathbf{f}(\mathbf{x}, \sigma) - \mathbf{x}||^2}{\sigma^2} + o(1) \text{ when } \sigma \to 0.$$

Vector field induced by reconstruction error



Reference

- Lectures on Diffusion processes: https://math.nyu.edu/ goodman/teaching/StochCalc2018/notes/Lesson2.pdf
- Tutorial on diffusion models: https://cvpr2023-tutorial-diffusion-models.github.io/
- Wang, Kai, et al. "Neural network diffusion."arXiv preprint arXiv:2402.13144 (2024).
- Vincent, Pascal. "A connection between score matching and denoising autoencoders."Neural computation 23.7 (2011): 1661-1674.
- Alain G., Bengio Y. What regularized auto-encoders learn from the data-generating distribution //The Journal of Machine Learning Research. – 2014. – T. 15. – №. 1. – C. 3563-3593.

Organizational issues

- Technical meeting: next week, regular time
- Format: each team shows the basic code. You should have an idea how this will be transformed into demo.
- The draft version of blog-post and documentation will be checked in offline-mode.
- All the presented materials must be stored at the github
  - for blog-post, you can put read-only link for the overleaf if you write the post here.
  - Time-limit: 10 minutes.

Blog-post

Minimum to write:

- Abstract/Motivation text
- Plan with all the sections you are planning and a small explanation text for each section
- Add all the planned figures or their description (*"There will be a scatter plot with our benchmar of algorithms. The x axis corresponds to the accuracy, y axis corresponds to the model size"*).
- Think and add a figure/description of the figure for the headline.
- The text is in English.
- This is a minimum, the more you have the better it is.

Documentation

- The documentation must be avilable to deploy:
  - either it must be available online
  - Or there msut be instructions how to deploy it in stand-alone mode
  - At the end of the project the documentation must be deployed online.
- The structure must be established (have a look at your favorite libraries).
- The documentation must be able to run (at githubpages or standalone on your PC).
- For the demo show at least one auto-doc.
- Popular engines: mkdocs, sphinx.

Unit-tests

- Cover as much functions (and instructions in them) as possible. Minimal coverage is 75%.
- Synthetic tests are wellcome: generate dataset on the fly and check that at least your functions are working on them correctly.
- Segmented code is easier to test: try to segment you code into more or less elementary functions.
- Popular engines: unittest (built-in into python) + coverage, pytest + pytest-cov.