Machine Learning for Graphs and Sequential Data

Deep Generative Models – Denoising Diffusion

Lecturer: Prof. Dr. Stephan Günnemann
www.cs.cit.tum.de/daml

Summer Term 2023

Data Analytics and
Machine Learning

Roadmap

- Deep Generative Models
  1. Introduction
  2. Normalizing Flows
  3. Variational Inference
  4. Generative Adversarial Networks
  5. **Denoising Diffusion**
     1. **Introduction**
     2. Probabilistic perspective
     3. Score matching perspective
     4. Example – Image synthesis (DALL-E 2)

Motivation

- Find a model with good generative capabilities that is easy to train
  - Diffusion currently outperforms GANs on many tasks including image generation

Prompt: „An astronaut riding a horse in a photorealistic style”

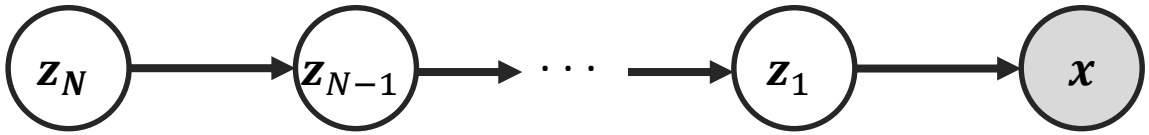
Image from [1]

Previously...

- Generation with normalizing flows / VAEs / GANs
  1) Sample noise $\boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
  2) Denoise to get data sample $\boldsymbol{x} = f_{\boldsymbol{\theta}}(\boldsymbol{z})$ resp. $\boldsymbol{x} \sim p(x; f_{\boldsymbol{\theta}}(\boldsymbol{z}))$
- Often limited by a *single step* and/or specific *parametric form*
  - Recall the difference in the loss and the constraints on the network

Diffusion Approach

1) Sample noise $\boldsymbol{z}_N \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
2) Probabilistic denoising to get slightly **less-noisy** sample $\boldsymbol{z}_{n-1} = f_{\boldsymbol{\theta}}(\boldsymbol{z}_n, n)$
3) Repeat 1) and 2) N times, to obtain $\boldsymbol{x} \mathrel{\widehat{=}} \boldsymbol{z}_0$



Goal

- Learn to sample from the true data distribution $p^*(\boldsymbol{x}), \boldsymbol{x} \in \mathbb{R}^d$
- **Idea:** gradually remove noise from an initial noisy sample until we get the sample from the data distribution: $\boldsymbol{x} \sim p^*(\boldsymbol{x})$

- But how to sample from $p^*(\boldsymbol{x})$ without knowing $p^*(\boldsymbol{x})$?
  - ➢ Again, we are interested in learning a model distribution $p_\theta(\boldsymbol{x})$ that approximates $p^*(\boldsymbol{x})$ and is easy to sample from

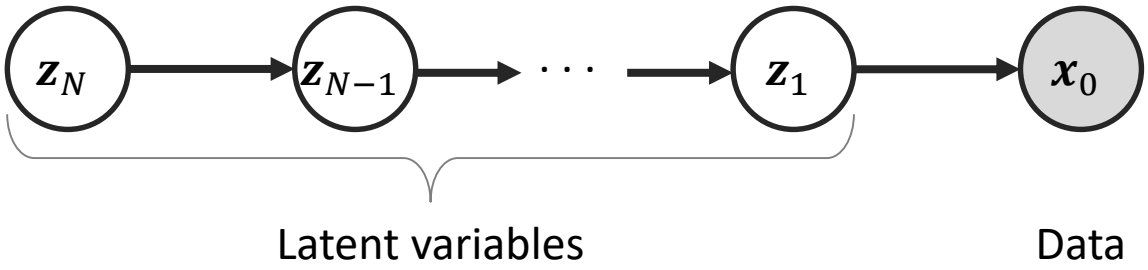
Image from [2]

Roadmap

- Deep Generative Models
  1. Introduction
  2. Normalizing Flows
  3. Variational Inference
  4. Generative Adversarial Networks
  5. **Denoising Diffusion**
     1. Introduction
     2. **Probabilistic perspective**
     3. Score matching perspective
     4. Example – Image synthesis (DALL-E 2)

Generation

- A diffusion model is essentially a latent variable model with a chain of latent variables
- Latent variable model: $p_\theta(\boldsymbol{x}_0) = \int p_\theta(\boldsymbol{x}_0, \boldsymbol{z}_{1:N}) \mathrm{d}\boldsymbol{z}_1 \dots \mathrm{d}\boldsymbol{z}_N$



- The „direction" from $\boldsymbol{z}_N$ to $\boldsymbol{x}_0$ (i.e. the generation) is usually called the **reverse process**:

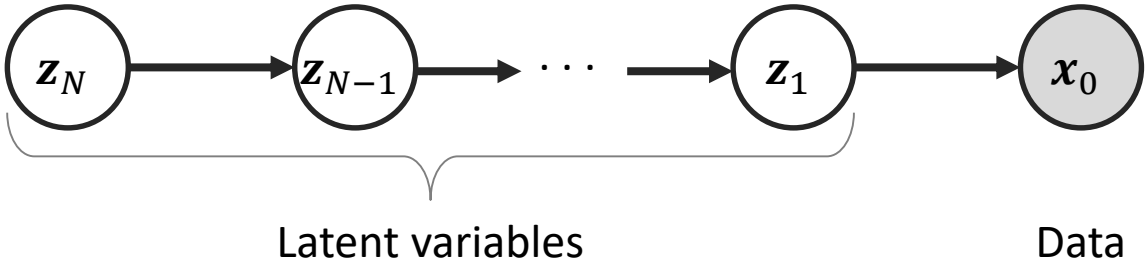
$$p_\theta(\boldsymbol{x}_0, \boldsymbol{z}_{1:N}) = p(\boldsymbol{z}_N) \prod_{n>1} p_\theta(\boldsymbol{z}_{n-1} | \boldsymbol{z}_n) \; p_\theta(\boldsymbol{x}_0 | \boldsymbol{z}_1)$$

  - $\boldsymbol{z}_{1:N} = \boldsymbol{z}_1, \dots, \boldsymbol{z}_N$ are latent variables in the same sample space as $\boldsymbol{x}_0$
  - Our model is a (learnable) **Markov process**!

Reverse Process Parametrization

- Reverse process – parametrization:

$$p_\theta(\boldsymbol{x}_0, \boldsymbol{z}_{1:N}) = p(\boldsymbol{z}_N) \prod_{n>1} p_\theta(\boldsymbol{z}_{n-1}|\boldsymbol{z}_n)\ p_\theta(\boldsymbol{x}_0|\boldsymbol{z}_1)$$



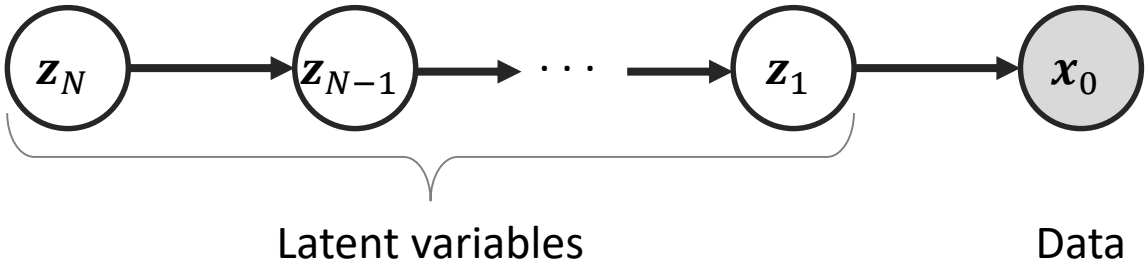
- What to pick as the individual distributions?
- A common choice:
  - $p_\theta(\boldsymbol{z}_{n-1}|\boldsymbol{z}_n) = \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{z}_n, n), \boldsymbol{\Sigma}_{\boldsymbol{\theta}}(\boldsymbol{z}_n, n))$
  - $p_\theta(\boldsymbol{x}_0|\boldsymbol{z}_1) = \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{z}_1, 1), \boldsymbol{\Sigma}_{\boldsymbol{\theta}}(\boldsymbol{z}_1, 1))$
  - $p(\boldsymbol{z}_N) = \mathcal{N}(\mathbf{0}, \boldsymbol{I})$

here $\boldsymbol{\mu}_{\boldsymbol{\theta}}$ and $\boldsymbol{\Sigma}_{\boldsymbol{\theta}}$ are neural networks with $\boldsymbol{\theta}$ being the model parameters (often $\boldsymbol{\Sigma}_{\boldsymbol{\theta}}$ is not learned but fixed)

How to Learn?

- Latent variable model:

$$p_\theta(\boldsymbol{x}_0, \boldsymbol{z}_{1:N}) = p(\boldsymbol{z}_N) \prod_{n>1} p_\theta(\boldsymbol{z}_{n-1}|\boldsymbol{z}_n) \;\; p_\theta(\boldsymbol{x}_0|\boldsymbol{z}_1)$$



- How to learn such a complex latent variable model?
  - i.e. how to find the parameters $\boldsymbol{\theta}$?

➢ Let's use Variational Inference!

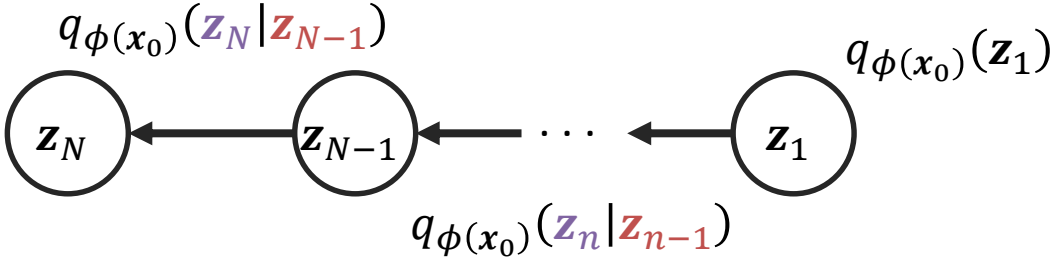
- Remember: besides θ, this also gives us a $q(\boldsymbol{z}_{1:N}) \approx p(\boldsymbol{z}_{1:N}|\boldsymbol{x}_0)$ and learning is done by optimizing the ELBO

Factorization of the Variational Distribution

- How is the distribution q picked in a diffusion model?

1. We assume a specific factorization of q → again a Markov process

$$q_{\phi(\boldsymbol{x}_0)}(\boldsymbol{z}_1, \dots, \boldsymbol{z}_N) = q_{\phi(\boldsymbol{x}_0)}(\boldsymbol{z}_1) \prod_{n>1} q_{\phi(\boldsymbol{x}_0)}(\boldsymbol{z}_n | \boldsymbol{z}_{n-1})$$



- Remember: In usual variational inference, every sample $\boldsymbol{x}_0$ gets its „own" variational distribution q. Thus, the parameters $\phi(\boldsymbol{x}_0)$ of the distribution depend on $\boldsymbol{x}_0$

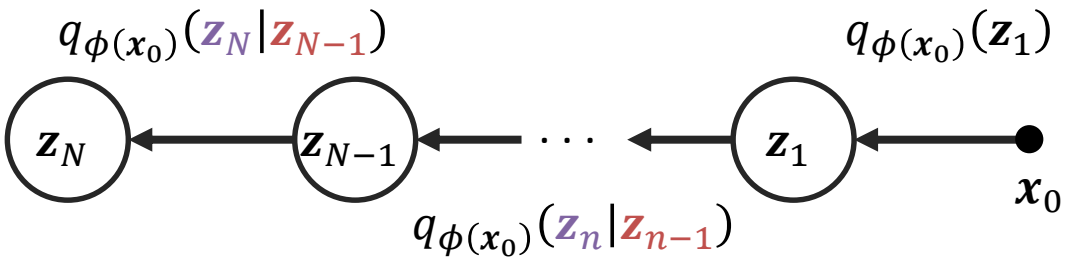
Forward Process Parametrization

- How is the distribution q picked in a diffusion model?

2. Unlike usual variational inference, the variational distribution q in a diffusion model is **not learned!**

- We pick the parametrization:
  - $q_{\phi(\boldsymbol{x}_0)}(\boldsymbol{z}_1) = \mathcal{N}\left(\sqrt{1-\beta_1}\boldsymbol{x}_0, \beta_1 \boldsymbol{I}\right)$
  - $q_{\phi(\boldsymbol{x}_0)}(\boldsymbol{z}_n|\boldsymbol{z}_{n-1}) = \mathcal{N}(\sqrt{1-\beta_n}\boldsymbol{z}_{n-1}, \beta_n \boldsymbol{I})$
  - $0 < \beta_1 < \beta_2 < \cdots < \beta_N < 1$ are noise scales

Note that q still depends on $\boldsymbol{x}_0$
→ we essentially have a "handcrafted" version of amortized inference



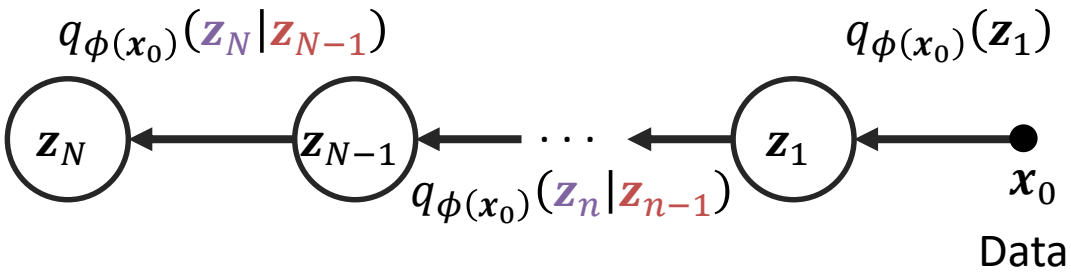
- This is the noising process, also called **forward process**

Discussion

- Why does it make sense to not learn q?
  - The actual goal is learning θ; q acts as a helper to approximate $\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_0)$
  - Since only p_θ is learned, a Diffusion Model is rather easy/stable to train
- Why is it ok?
  - Since $p_\theta(\boldsymbol{x}_0, \boldsymbol{z}_{1:N})$ comes from (multiple steps of) a neural network, it is quite powerful
    - It can likely very well reconstruct the data
    - With a fixed q, we intuitively regularize the model by requiring $p_\theta(\boldsymbol{z}_{1:N}|\boldsymbol{x}_0) \approx q_{\phi(\boldsymbol{x}_0)}(\boldsymbol{z}_{1:N})$
  - Since $p_\theta(\boldsymbol{z}_{n-1}|\boldsymbol{z}_n)$ is Gaussian, it is reasonable to assume $q_{\phi(\boldsymbol{x}_0)}(\boldsymbol{z}_n|\boldsymbol{z}_{n-1})$ to be Gaussian (in particular, since we have many diffusion steps)

→ And: The choice of q makes all subsequent calculations easy!

Forward process reparametrization



- Given the previous instantiation:
  - It is possible to sample z_n directly from x_0, which will be helpful in training:

$$q_{\phi(x_0)}(z_n) = \mathcal{N}\left(\sqrt{\bar{\alpha}_n}x_0, (1-\bar{\alpha}_n)\boldsymbol{I}\right) \text{ where } \bar{\alpha}_n = \prod_i^n \alpha_i \,, \alpha_i = 1-\beta_i$$

  - Using the reparametrization trick we get any z_n given x_0:

$$z_n = \sqrt{\bar{\alpha}_n}x_0 + \sqrt{(1-\bar{\alpha}_n)}\boldsymbol{\epsilon} \text{ where } \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$$

  - To ensure that sampling from our generative model is reasonable, β_i is generally choosen such that the distribution $q_{\phi(x_0)}(z_N) \approx p(z_N) = \mathcal{N}(\mathbf{0}, \boldsymbol{I})$

Learning via Optimizing the ELBO

- To repeat:
  - Latent variable model: $p_\theta(\boldsymbol{x}_0) = \int p_\theta(\boldsymbol{x}_0, \boldsymbol{z}_{1:N}) \mathrm{d}\boldsymbol{z}_1 \dots \mathrm{d}\boldsymbol{z}_N$
  - $\boldsymbol{x}_0$ is our data, $\boldsymbol{z}_{1:N} = \boldsymbol{z}_1, \dots, \boldsymbol{z}_N$ are latent variables
- We aim to learn θ via variational inference
  - Since maximizing the actual likelihood, i.e. $\max_\theta \sum_{\boldsymbol{x}_0} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}_0)$, is intractable
  - We are maximizing the ELBO

- Recall from the VI lecture, the ELBO is given by:

$$\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_0) \geq \mathbb{E}_{q(\boldsymbol{z}_{1:N})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_0, \boldsymbol{z}_{1:N}) - \log q_{\boldsymbol{\phi}(\boldsymbol{x}_0)}(\boldsymbol{z}_{1:N})\right]$$

$$= \mathbb{E}_{q(\boldsymbol{z}_{1:N})}\left[\log p(\boldsymbol{z}_N) + \sum_{n>1} \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{z}_{n-1}|\boldsymbol{z}_n)}{q_{\boldsymbol{\phi}(\boldsymbol{x}_0)}(\boldsymbol{z}_n|\boldsymbol{z}_{n-1})} + \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}_0|\boldsymbol{z}_1)}{q_{\boldsymbol{\phi}(\boldsymbol{x}_0)}(\boldsymbol{z}_1)}\right]$$

ELBO Simplifications

- Since $p(\boldsymbol{z}_N)$ is known and using Bayes rule, the ELBO *essentially* boils down to:

$$\mathbb{E}_{\boldsymbol{z}_1 \sim q_{\boldsymbol{\phi}(\boldsymbol{x}_0)}(\boldsymbol{z}_1)}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_0|\boldsymbol{z}_1)] - \sum_{n>1} \mathbb{KL}[q_{\boldsymbol{\phi}(\boldsymbol{x}_0)}(\boldsymbol{z}_{n-1}|\boldsymbol{z}_n)||p_{\boldsymbol{\theta}}(\boldsymbol{z}_{n-1}|\boldsymbol{z}_n)],$$

- where $q_{\boldsymbol{\phi}(\boldsymbol{x}_0)}(\boldsymbol{z}_{n-1}|\boldsymbol{z}_n) = \mathcal{N}\left(\tilde{\boldsymbol{\mu}}(\boldsymbol{x}_0, \boldsymbol{z}_n, n), \tilde{\beta}_n \boldsymbol{I}\right)$,

$$\tilde{\boldsymbol{\mu}}(\boldsymbol{x}_0, \boldsymbol{z}_n, n) = \frac{\sqrt{\alpha_n}(1-\bar{\alpha}_{n-1})}{1-\bar{\alpha}_n}\boldsymbol{z}_n + \frac{\sqrt{\bar{\alpha}_{n-1}}\beta_n}{1-\bar{\alpha}_n}\boldsymbol{x}_0 \text{ and } \tilde{\beta}_n = \frac{1-\bar{\alpha}_{n-1}}{1-\bar{\alpha}_n}\beta_n$$

- Since all distributions normal, the loss is available in closed-form

- For more details, check the full derivation in [6] and homework exercise

Model Parametrization

- Recall: $p_\theta(\boldsymbol{z}_{n-1}|\boldsymbol{z}_n) = \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{z}_n, n), \boldsymbol{\Sigma}_{\boldsymbol{\theta}}(\boldsymbol{z}_n, n))$

- How do we choose $\boldsymbol{\mu}_{\boldsymbol{\theta}}$ and $\boldsymbol{\Sigma}_{\boldsymbol{\theta}}$?
  - For simplicity set $\boldsymbol{\Sigma}_{\boldsymbol{\theta}}(\boldsymbol{z}_n, n) \coloneqq \tilde{\beta}_n \boldsymbol{I}$
  - Further, we observe that to minimize $\mathbb{KL}[q_{\phi(\boldsymbol{x}_0)}(\boldsymbol{z}_{n-1}|\boldsymbol{z}_n)||p_{\boldsymbol{\theta}}(\boldsymbol{z}_{n-1}|\boldsymbol{z}_n)]$, $\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{z}_n, n)$ and $\tilde{\boldsymbol{\mu}}(\boldsymbol{x}_0, \boldsymbol{z}_n, n)$ need to be similar.
  - Idea: Define $\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{z}_n, n) \coloneqq \tilde{\boldsymbol{\mu}}(\boldsymbol{x}_0, \boldsymbol{z}_n, n)$
- However, $\tilde{\boldsymbol{\mu}}$ requires $\boldsymbol{x}_0$, which we do not have in the reverse process
  - Recall the reparametrization trick: $\boldsymbol{z}_n = \sqrt{\bar{\alpha}_n}\boldsymbol{x}_0 + \sqrt{(1-\bar{\alpha}_n)}\boldsymbol{\epsilon}$
  - By inverting the equation, we can estimate $\boldsymbol{x}_0$ given $\boldsymbol{z}_n$ by predicting $\boldsymbol{\epsilon}$

$$\boldsymbol{x}_0 \approx f_\theta(\boldsymbol{z}_n, n) = \frac{\boldsymbol{z}_n - \sqrt{(1-\bar{\alpha}_n)}\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{z}_n, n)}{\sqrt{\bar{\alpha}_n}}$$

Full Model

- Our generative model can be summarized as follows:

$$p_{\boldsymbol{\theta}}(\boldsymbol{z}_{n-1}|\boldsymbol{z}_n) = \mathcal{N}(\tilde{\boldsymbol{\mu}}(f_\theta(\boldsymbol{z}_n, n), \boldsymbol{z}_n, n), \tilde{\beta}_n \boldsymbol{I})$$

Again, these are design choices which we discussed on the previous slide

- As we have seen, it is possible to reparameterize the model to predict the amount of noise $\boldsymbol{\epsilon}$ that was added to $\boldsymbol{x}_0$ to obtain $\boldsymbol{z}_n$: $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{z}_n, n)$
  - In practice diffusion models are often trained to minimize the simplified loss:

$$\mathcal{L} = \mathbb{E}_{n, \boldsymbol{x}_0, \boldsymbol{\epsilon}}\left[c_n \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_{\boldsymbol{\theta}}\left(\underbrace{\sqrt{\bar{\alpha}_n}\boldsymbol{x}_0 + \sqrt{(1-\bar{\alpha}_n)}\boldsymbol{\epsilon}}_{\boldsymbol{z}_n}, n\right)\right\|^2\right]$$

  - Where with $c_n = \frac{{\beta_n}^2}{2\tilde{\beta}_n \alpha_n (1-\bar{\alpha}_n)}$ $\mathcal{L}$ is the ELBO, but in practice $c_n = 1$ is predominantly used

Training

```
x_0 = random.choice(data) # Sample clean data

n = random.uniform(1, N) # Noise step
noise = random.normal(x_0.shape)


# Noisy data
z_n = sqrt(alpha_cumprod[n]) * x_0 + sqrt(1-alpha_cumprod[n]) * noise


predicted_noise = model(z_n, n)
loss = mean((predicted_noise - noise)**2)
```

Sampling

```
z_n = random.normal(d)  # Sample Gaussian noise
for n in reversed(range(1, N + 1)):
    predicted_noise = model(z_n, n)

    # Estimate x_0 based on z_n
    x_0 = (
      (z_n - sqrt(1 - alpha_bar[n]) * predicted_noise)
      / sqrt(alpha_bar[n])
    )

    # In the final iteration x_0 is already the final x_0 sample
    if n == 1: break

    # Sample z_{n-1} based on z_n and the current x_0 estimate
    z_n = random.normal(mu=... * z_n + ... * x_0, sigma=beta_tilde[n])
```

Alternative Notations

When reading literature on diffusion models, you will oftern see further notations:

- Often the latent variables $\boldsymbol{z}_1, \dots, \boldsymbol{z}_N$ are called $\boldsymbol{x}_1, \dots, \boldsymbol{x}_N$ and n is called t
- Often $q_{\phi(\boldsymbol{x}_0)}(\cdot)$ is denoted as $q(\cdot \,|\boldsymbol{x}_0)$
  - Note however, that the q distribution is over $\boldsymbol{z}$; not over $\boldsymbol{x}$
  - Technically $\boldsymbol{x}_0$ is used here for the purpose of amortized variational inference

Roadmap

- Deep Generative Models
  1. Introduction
  2. Normalizing Flows
  3. Variational Inference
  4. Variational Autoencoder
  5. Generative Adversarial Networks
  6. **Denoising Diffusion**
     1. Introduction
     2. Probabilistic perspective
     3. **Score matching perspective**
     4. Example – Image synthesis (DALL-E 2)

Score Matching Perspective

- Empirically multiple diffusion steps produce higher quality samples:

- But why do many diffusion steps help? To answer this question we will take a look at the score matching perspective of Diffusion Models.

- But also, remember our core question: how to sample from an (unknown) distribution?

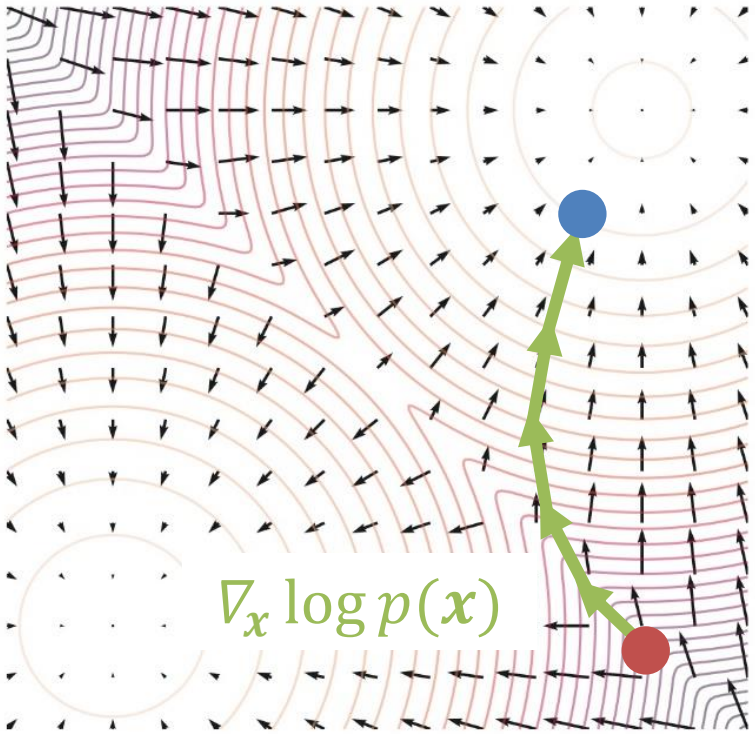
Image from [13]

Roadmap

- Deep Generative Models
  1. Introduction
  2. Normalizing Flows
  3. Variational Inference
  4. Generative Adversarial Networks
  5. **Denoising Diffusion**
     1. Introduction
     2. Probabilistic perspective
     3. **Score matching perspective**
        a. **Background: Sampling via Langevin dynamics**
        b. Score Matching and Noise Perturbations
        c. Full model
     4. Example – Image synthesis (DALL-E 2)

How to sample from a (known) distribution?

- **Idea:** Similar to gradient optimization, start with **random** $\boldsymbol{x}_N$ and perform gradient ascent using $\nabla_{\boldsymbol{x}} \log p(\boldsymbol{x})$ to find a **high probability** $\boldsymbol{x}_0$ after N steps
  - We „optimize" on the data probability to find high probability points
- Here we used the **score**, the gradient of the log-density: $\nabla_{\boldsymbol{x}} \log p(\boldsymbol{x})$

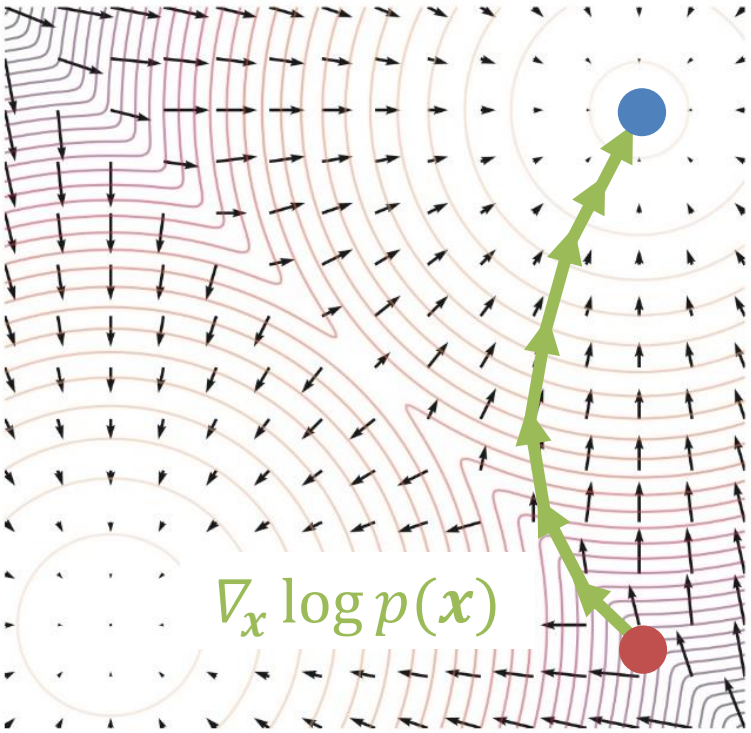


**This is just an illustration. We will see how to actually sample later.*

Image from [2]

Sampling using Langevin dynamic

- If we do gradient ascent on $p(\boldsymbol{x})$, we end up at single point (max. density)
  - Similar to maximum a posteriori estimate in Bayesian inference
- What we really want is to sample from $p(\boldsymbol{x})$
  - Similar to sampling from a posterior



$\boldsymbol{x}_0 \sim p(\boldsymbol{x})$

Wrong!

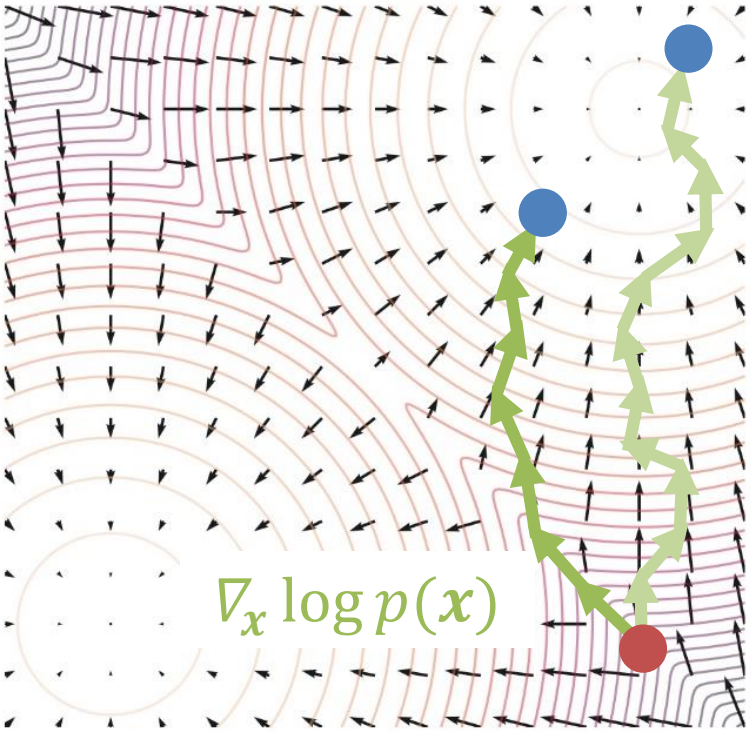
$\boldsymbol{x}_N \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$

Sampling using Langevin dynamic

- The solution is to add small Gaussian noise at each gradient ascent step
- We get $\boldsymbol{x}_0 \sim p(\boldsymbol{x})$ from random $\boldsymbol{x}_N$ with the following iterative procedure:

$$\boldsymbol{x}_{n-1} = \boldsymbol{x}_n + \delta_n \nabla_{\boldsymbol{x}_n} \log p(\boldsymbol{x}_n) + \sqrt{2\delta_n}\boldsymbol{\epsilon}$$

- where δ_n is step size and $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, then $\boldsymbol{x}_0 \to p(\boldsymbol{x})$ as $\delta_n \to 0$ and $N \to \infty$



$\boldsymbol{x}_0 \sim p(\boldsymbol{x})$

Correct!

$\boldsymbol{x}_N \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$

Sampling using Langevin dynamics

- The solution is to add small Gaussian noise at each gradient ascent step
- We get $\boldsymbol{x}_0 \sim p(\boldsymbol{x})$ from random $\boldsymbol{x}_N$ with the following iterative procedure:

$$\boldsymbol{x}_{n-1} = \boldsymbol{x}_n + \delta_n \nabla_{\boldsymbol{x}_n} \log p(\boldsymbol{x}_n) + \sqrt{2\delta_n}\boldsymbol{\epsilon}$$

- where δ_n is step size and $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, then $\boldsymbol{x}_0 \to p(\boldsymbol{x})$ as $\delta_n \to 0$ and $N \to \infty$

Why does this work?

The above equation is an Euler discretization of the Langevin SDE:

$$\mathrm{d}\boldsymbol{x}_t = \nabla_{\boldsymbol{x}} \log p(\boldsymbol{x}_t)\,\mathrm{d}t + \sqrt{2}\mathrm{d}\boldsymbol{W}_t$$

where $\boldsymbol{W}_t$ denotes standard Brownian motion (Wiener process).

- It can be shown that the probability of the solutions of the SDE $q(\boldsymbol{x}, t)$ will become stationary as $t \to \infty$ and equal to $p(\boldsymbol{x})$
- See Chapter 3 in [4] for 1-dim. proof and [5] for Bayesian application

Takeaway

- If we know the score $\nabla_{\boldsymbol{x}} \log p(\boldsymbol{x})$, we can sample!
- Unfortunately, we don't know the score of our distribution $p^*(\boldsymbol{x})$
- What to do?

Roadmap

- Deep Generative Models
  1. Introduction
  2. Normalizing Flows
  3. Variational Inference
  4. Generative Adversarial Networks
  5. **Denoising Diffusion**
     1. Introduction
     2. Probabilistic perspective
     3. **Score matching perspective**
        a. Background: Sampling via Langevin dynamics
        b. **Score Matching and Noise Perturbations**
        c. Full model
     4. Example – Image synthesis (DALL-E 2)

Learning the score with score matching

- **Goal:** Approximate true score $\nabla_{\boldsymbol{x}} \log p^*(\boldsymbol{x})$ with a neural network $s_{\boldsymbol{\theta}}(\boldsymbol{x})$
- We can use a simple squared loss [3]:

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{2} \int p^*(\boldsymbol{x}) \| \nabla_{\boldsymbol{x}} \log p^*(\boldsymbol{x}) - s_{\boldsymbol{\theta}}(\boldsymbol{x}) \|^2 \mathrm{d}\boldsymbol{x}$$

- **Problem:** No access to $\nabla_{\boldsymbol{x}} \log p^*(\boldsymbol{x})$
- **Solution:** It can be shown that the above loss is equal to [3]:

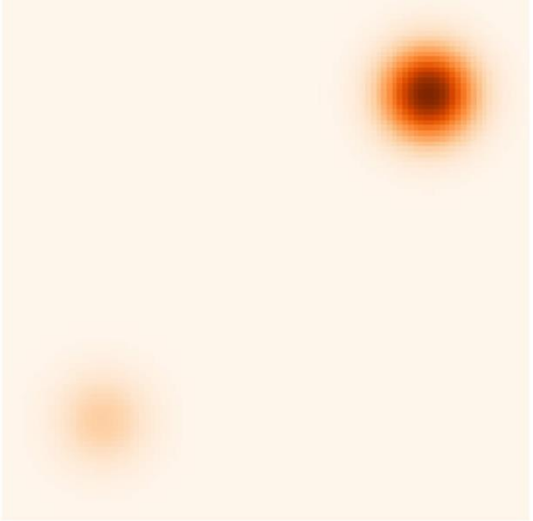
$$\mathcal{L}(\boldsymbol{\theta}) = \int p^*(\boldsymbol{x}) (\mathrm{Tr}\left(\nabla_{\boldsymbol{x}} s_{\boldsymbol{\theta}}(\boldsymbol{x})\right) + \frac{1}{2} \| s_{\boldsymbol{\theta}}(\boldsymbol{x}) \|^2) \mathrm{d}\boldsymbol{x}$$

- This is possible to compute!
  - It's an expectation over $\boldsymbol{x}$ and depends only on $s_{\boldsymbol{\theta}}$
  - We can learn the score using the data samples only, without having access to the true $p^*(\boldsymbol{x})$ or $\nabla_{\boldsymbol{x}} \log p^*(\boldsymbol{x})$

Issue with score matching

- We learn the score in areas where we have observed the points
  - Recall, loss is weighted by $p^*(\boldsymbol{x})$ meaning we ignore low density areas
- Score will be inaccurate outside high density areas
- When sampling, we start at a low density area
  - If we knew where high probability points were, we wouldn't be doing any of this

True data density



Learned scores

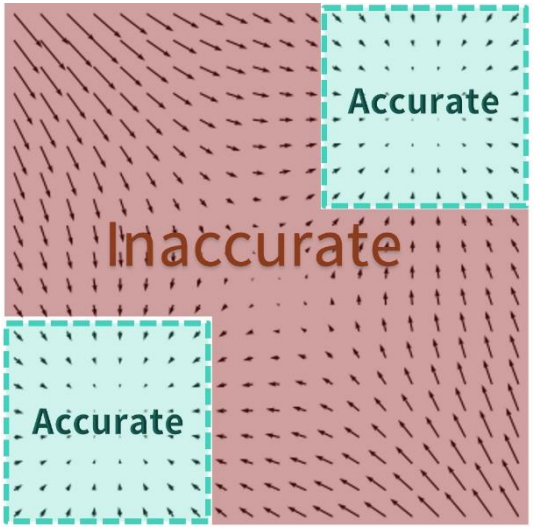
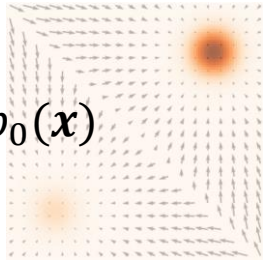


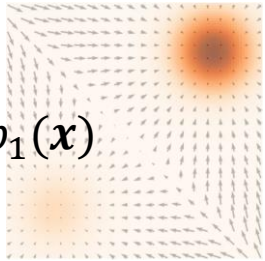
Image from [2]

Score matching with noise perturbations

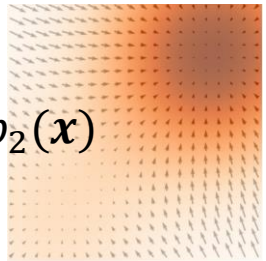
- **Solution:** Introduce multiple noise perturbations and learn the score for each of the perturbations



- Original data – we only learn scores around observed points
  - Elsewhere, the direction of the gradient is likely wrong



- Noisy data – we can learn the score on a greater area
  - If we can get the starting point close to the high density of p_1, resulting sample will be close to the high density of p_0 as well!



- Almost pure noise – contains almost no information about the original data but we learn the correct score everywhere
  - If the consecutive densities are similar, the final sample from p_n will be a good starting point for p_{n-1}, and so on...

Images from [2]

Score matching with noise perturbations – Example

- Let $0 < \sigma_1 < \sigma_2 < \cdots < \sigma_N$ be a set of increasing noise scales
- We denote the original (clean) data with $\boldsymbol{x}_0$
- Data corresponding to the noise scale n is obtained with:

$$\boldsymbol{x}_n = \boldsymbol{x}_0 + \sigma_n \boldsymbol{\epsilon}, \qquad \text{where } \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$$

  - That is, $p_n(\boldsymbol{x}|\boldsymbol{x}_0) = \mathcal{N}(\boldsymbol{x}_0, \sigma_n \boldsymbol{I})$
- We want to learn the **conditional** scores $\nabla_{\boldsymbol{x}} \log p_n(\boldsymbol{x}|\boldsymbol{x}_0)$
  - Use a neural network $s_{\boldsymbol{\theta}}(\boldsymbol{x}, n)$ conditioned on n
- Loss is similar to before, but summed over all noise levels:

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{n=1}^{N} \mathbb{E}_{p_n}[\|\nabla_{\boldsymbol{x}} \log p_n(\boldsymbol{x}|\boldsymbol{x}_0) - s_{\boldsymbol{\theta}}(\boldsymbol{x}, n)\|^2]$$

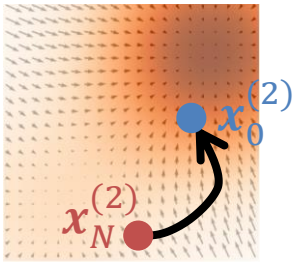
- Conditional score $\nabla_{\boldsymbol{x}} \log p_n(\boldsymbol{x}|\boldsymbol{x}_0)$ can be computed **in closed form** since $p_n(\boldsymbol{x}|\boldsymbol{x}_0)$ is a normal distribution with mean $\boldsymbol{x}_0$ and covariance $\sigma_n \boldsymbol{I}$

Score matching recap

- We can sample from $p(\boldsymbol{x})$ if we know the score $\nabla_{\boldsymbol{x}} \log p(\boldsymbol{x})$
- We can learn $\nabla_{\boldsymbol{x}} \log p(\boldsymbol{x})$ from data with a neural network $s_{\boldsymbol{\theta}}(\boldsymbol{x})$
  - ✕ The score will not be learned correctly in low density regions
- We can introduce noise perturbations and learn the score for N conditional distributions with a single neural network $s_{\boldsymbol{\theta}}(\boldsymbol{x}, n)$
  - First distribution $p_1(\boldsymbol{x}|\boldsymbol{x}_0)$ has to be close to data distribution
  - Final distribution $p_N(\boldsymbol{x}|\boldsymbol{x}_0)$ has to be close to pure noise
  - Consecutive distributions $p_{n-1}(\boldsymbol{x}|\boldsymbol{x}_0)$ and $p_n(\boldsymbol{x}|\boldsymbol{x}_0)$ should be similar

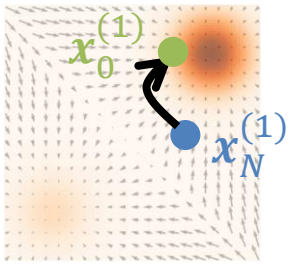
Sampling with multiple noise perturbations

$p_2(\boldsymbol{x})$



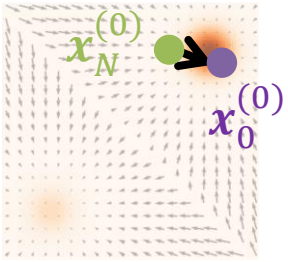
- Sample a starting point, e.g., $\boldsymbol{x}_N^{(2)} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
- Use Langevin dynamic to get $\boldsymbol{x}_{n-1} \leftarrow \boldsymbol{x}_n$, for $n = N, \dots, 1$
- Final sample $\boldsymbol{x}_0^{(2)}$ follows $p_2(\boldsymbol{x})$

$p_1(\boldsymbol{x})$



- Use $\boldsymbol{x}_0^{(2)}$ as a starting sample for $p_1(\boldsymbol{x})$: $\boldsymbol{x}_N^{(1)} = \boldsymbol{x}_0^{(2)}$
- Using Langevin dynamics get $\boldsymbol{x}_0^{(1)} \sim p_1(\boldsymbol{x})$
  - From initial $\boldsymbol{x}_N$, after N steps, using $s_{\boldsymbol{\theta}}(\boldsymbol{x}, n) \approx \nabla_{\boldsymbol{x}_n} \log p(\boldsymbol{x}_n)$

$p_0(\boldsymbol{x})$



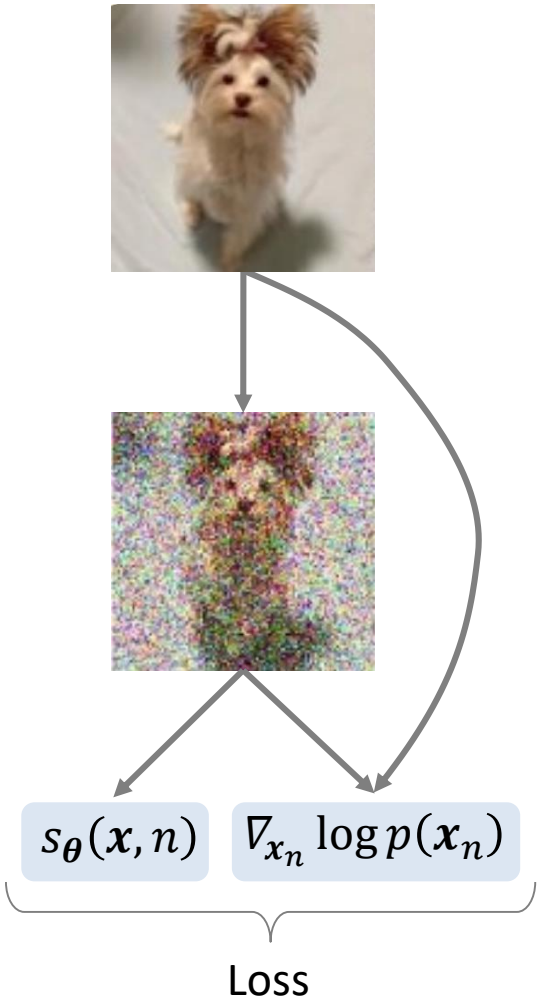
- Same as before: $\boldsymbol{x}_N^{(0)} = \boldsymbol{x}_0^{(1)}$
- Langevin dynamics: $\boldsymbol{x}_0^{(0)} \sim p_0(\boldsymbol{x})$ follows data distribution!

Roadmap

- Deep Generative Models
  1. Introduction
  2. Normalizing Flows
  3. Variational Inference
  4. Generative Adversarial Networks
  5. **Denoising Diffusion**
     1. Introduction
     2. Probabilistic perspective
     3. **Score matching perspective**
        a. Background: Sampling via Langevin dynamics
        b. Score Matching and Noise Perturbations
        c. **Full model**
     4. Example – Image synthesis (DALL-E 2)

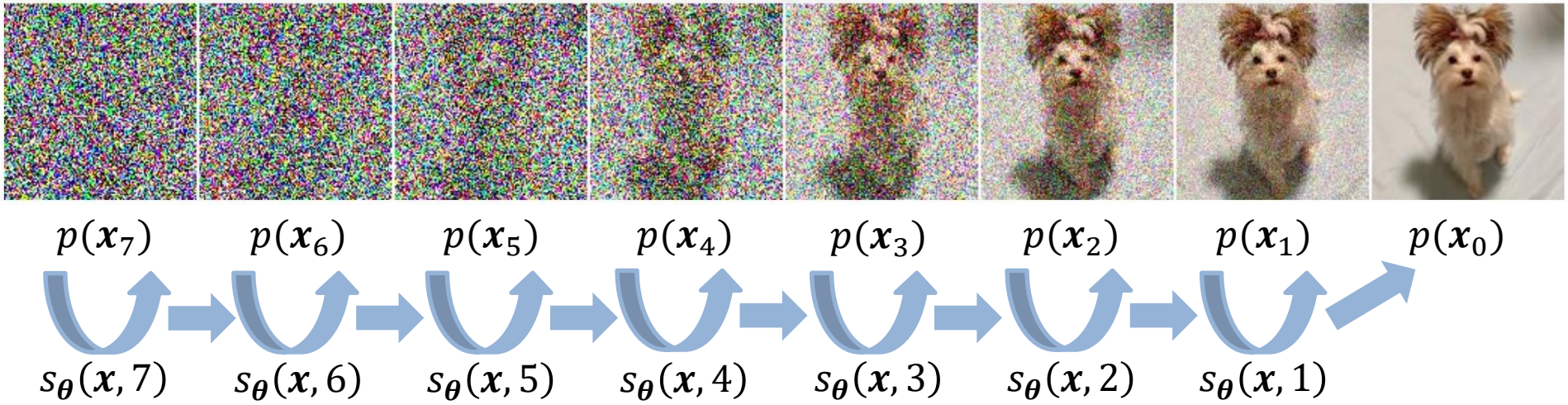
Full model – Training

```
x_0 = random.choice(data) # Clean data
n = random.uniform(1, N) # Noise step

noise = random.normal(x_0.shape)

x_n = x_0 + sigma[n] * noise # Noisy data

predicted_score = model(x_n, n)
true_score = -(x_n - x_0) / sigma[n]**2

loss = mean((predicted_score - true_score)**2)
```



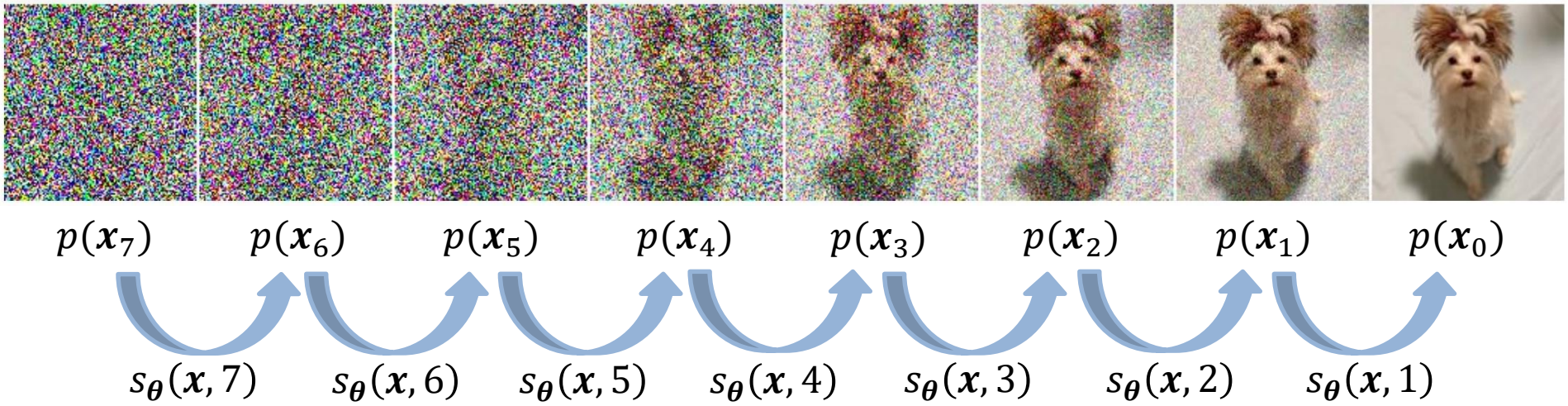
Full model – Sampling

```
x = random.normal(d) # Starting noise
for n in range(N, 0, -1): # Sample for all noise levels
    for i in range(L, 0, -1): # Langevin dynamics on one level
        e = random.normal(x.shape)
        x = x + delta[n] * model(x, n) + sqrt(2 * delta[n]) * e
x_0 = x
```



Full model – Sampling

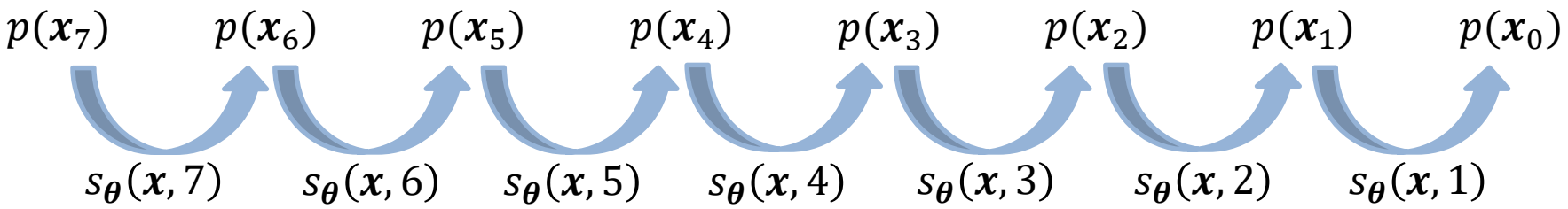
```
x = random.normal(d) # Starting noise
for n in range(N, 0, -1): # Sample for all noise levels
    # for i in range(L, 0, -1): # Langevin dynamics on one level
    e = random.normal(x.shape)
    x = x + delta[n] * model(x, n) + sqrt(2 * delta[n]) * e
x_0 = x
```



Full model – Sampling

```
x = random.normal(d) # Starting noise
for n in range(N, 0, -1): # Sample for all noise levels
    # for i in range(L, 0, -1): # Langevin dynamics on one level
    e = random.normal(x.shape)
    x = x + delta[n] * model(x, n) + sqrt(2 * delta[n]) * e
x_0 = x
```

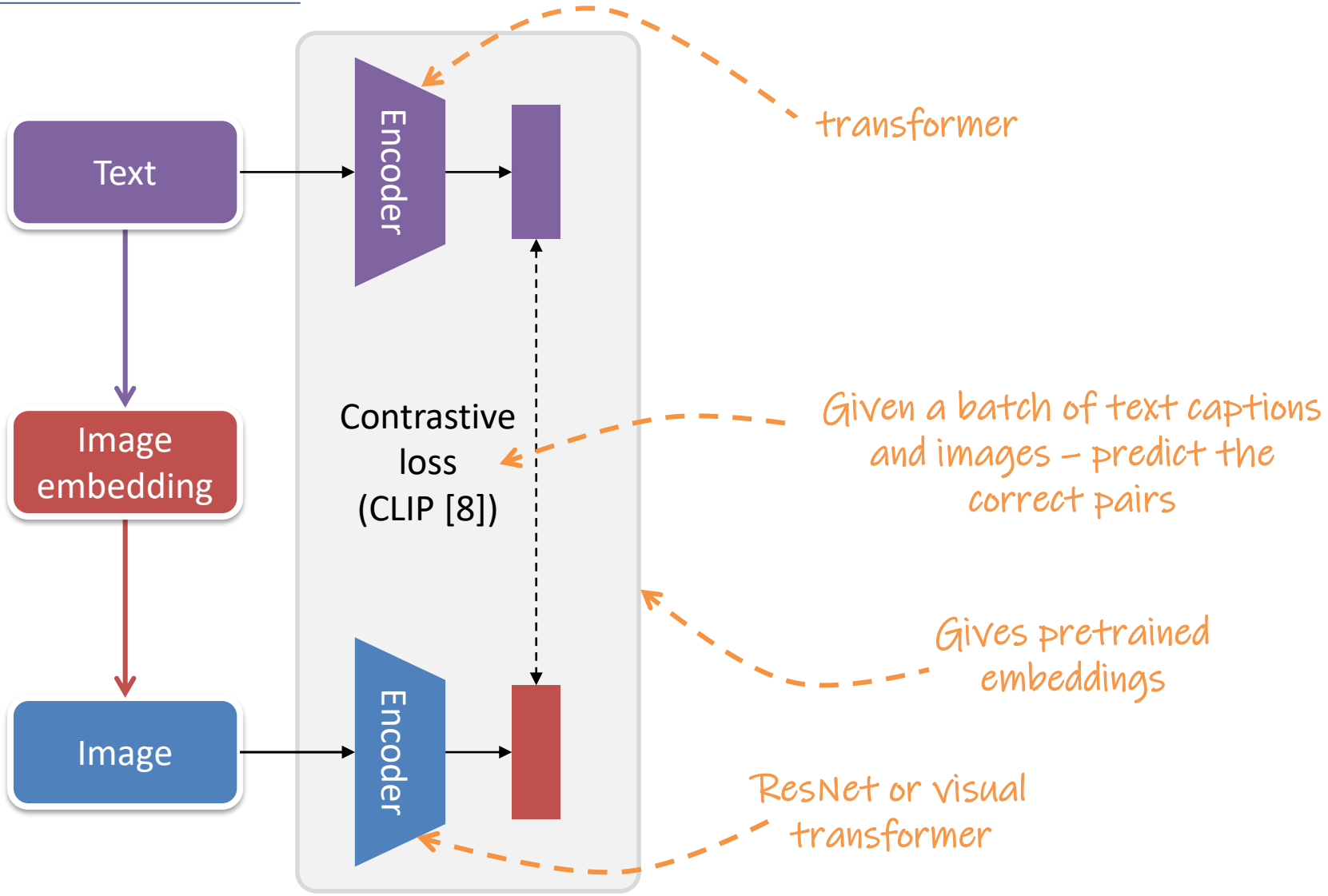
- Only one Langevin step per noise level
- Number of noise perturbations N should be large enough



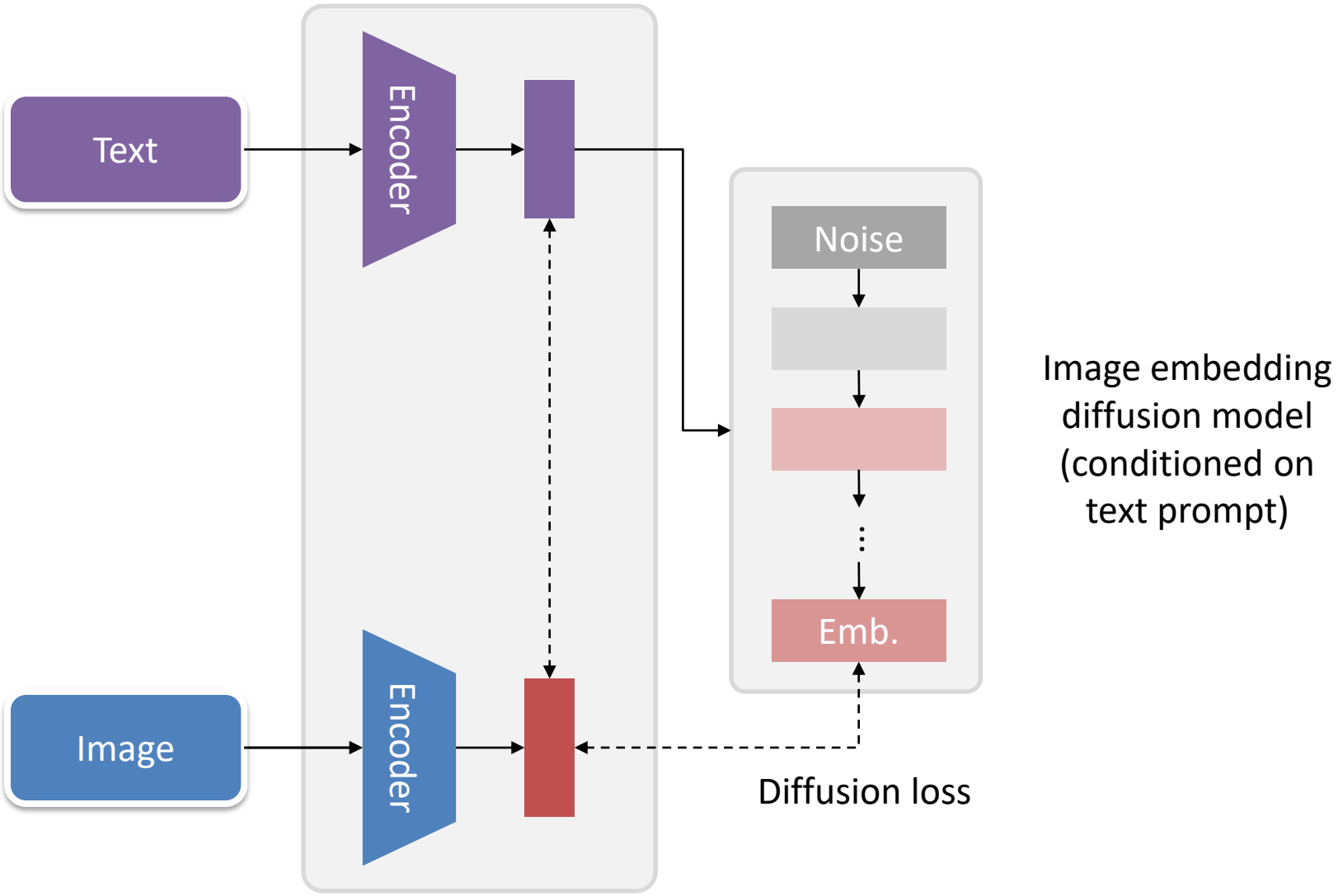
Roadmap

- Deep Generative Models
  1. Introduction
  2. Normalizing Flows
  3. Variational Inference
  4. Generative Adversarial Networks
  5. **Denoising Diffusion**
     1. Introduction
     2. Probabilistic perspective
     3. Score matching perspective
     4. **Example – Image synthesis (DALL-E 2)**

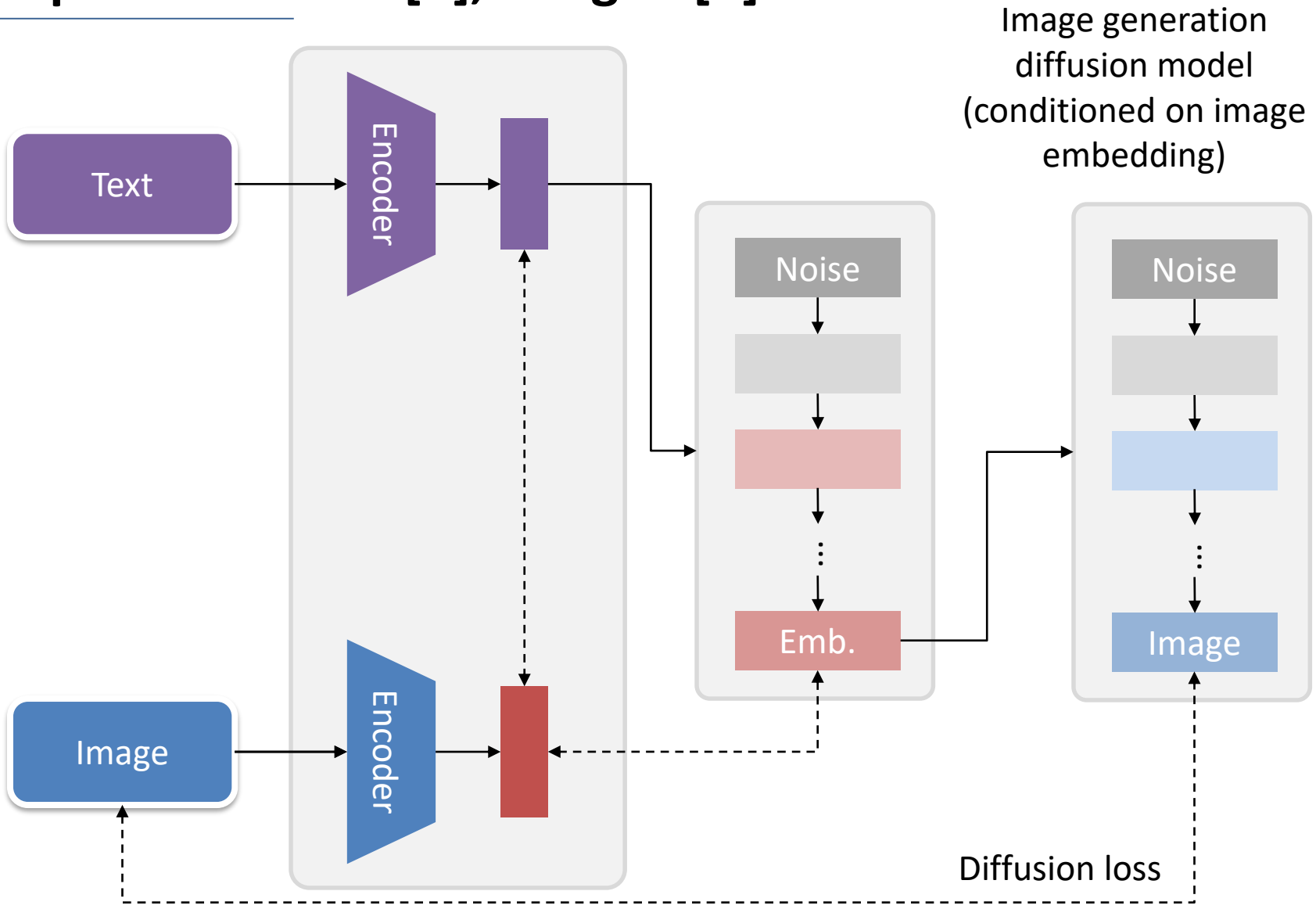
Example – DALL-E 2 [1], Imagen [7]...



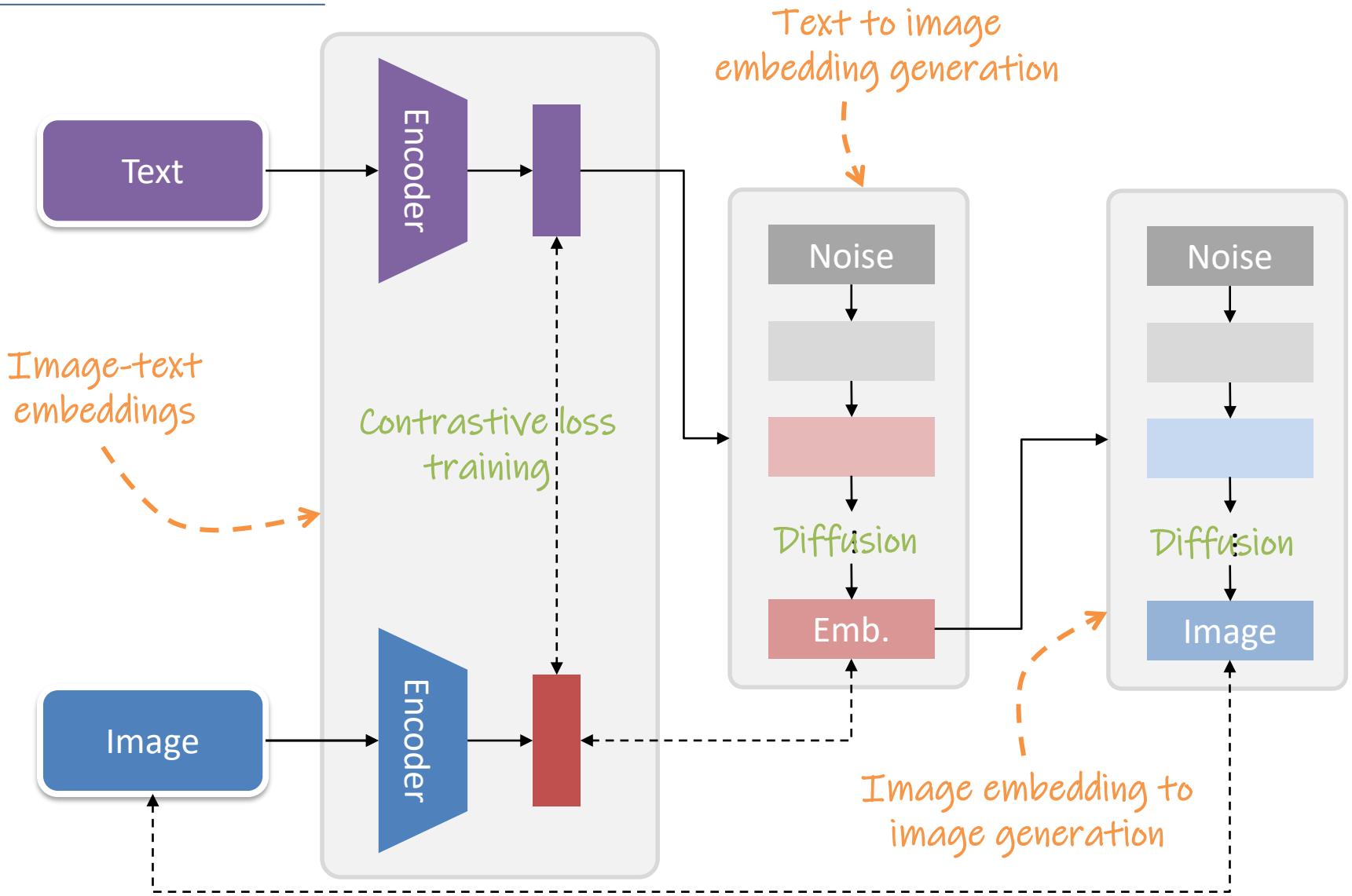
Example – DALL-E 2 [1], Imagen [7]...



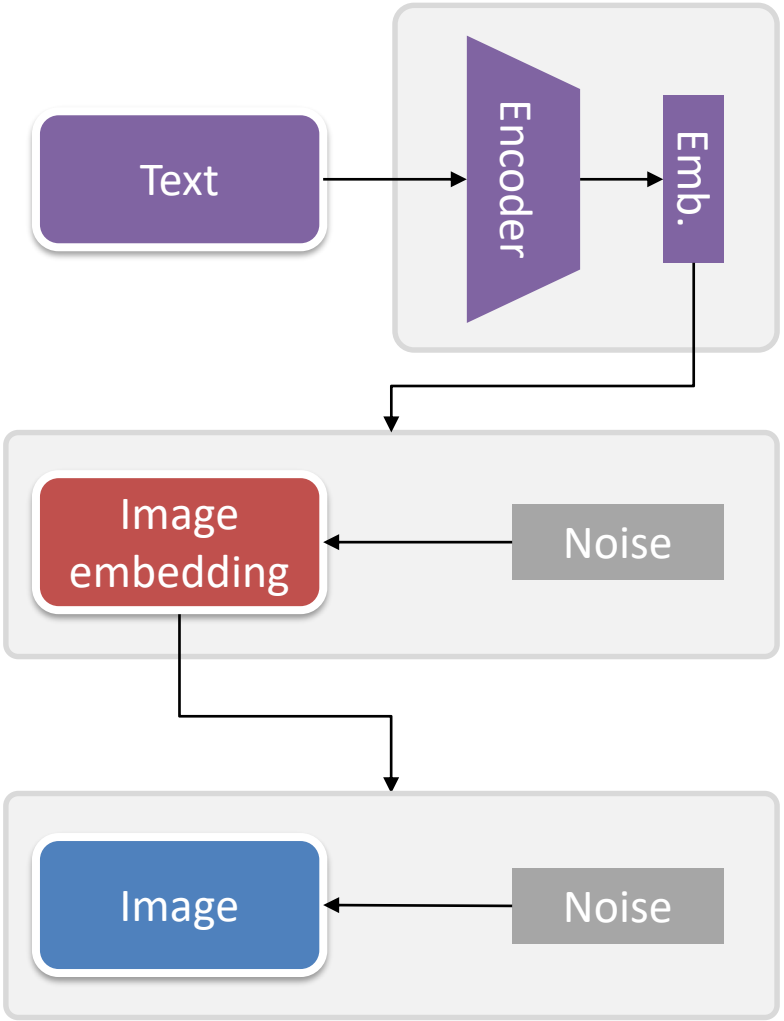
Example – DALL-E 2 [1], Imagen [7]...



Example – DALL-E 2 [1], Imagen [7]...



Example – DALL-E 2 [1], Imagen [7]...



Main ideas

- Transformer for text emb. [11]
- U-Net for diffusion model [12]
- Pretrained image-text embeddings [8]
- Generating from learned image embeddings
- Conditioning diffusion (guidance) on class, embedding... [9,10]
- Scaling data & models
- Efficient training

Problems and open questions

- Computational overhead of sampling (SOTA Diffusion Models have 1000-4000 steps)
- Limited meaningfulness of the latent variables (problem for representation learning, interpolation, latent optimization etc.)
- No exact likelihood evaluation
- How to handle complex data domains (e.g., see data considered in MLGS)

If you are interested in solving any of those problems or open questions, we offer

Master theses and Guided Research projects

References

[1] Ramesh et al., Hierarchical Text-Conditional Image Generation with CLIP Latents, 2022 (https://arxiv.org/abs/2204.06125, https://openai.com/dall-e-2/)

[2] Song et al., Score-Based Generative Modeling through Stochastic Differential Equations, 2021 (https://arxiv.org/abs/2011.13456, https://yang-song.net/blog/2021/score/)

[3] Hyvärinen, Estimation of non-normalized statistical models by score matching, 2005 (https://www.jmlr.org/papers/v6/hyvarinen05a.html)

[4] https://www.di.ens.fr/appstat/spring-2022/lecture_notes/SGLD.pdf

[5] Weilling and Teh, Bayesian learning via stochastic gradient langevin dynamics, 2011 (https://www.stats.ox.ac.uk/~teh/research/compstats/WelTeh2011a.pdf)

[6] Ho et al., Denoising Diffusion Probabilistic Models, 2020 (https://arxiv.org/abs/2006.11239)

[7] Saharia et al., Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding, 2022 (https://arxiv.org/abs/2205.11487)

[8] Radford et al., Learning Transferable Visual Models From Natural Language Supervision 2021 (https://arxiv.org/abs/2103.00020)

[9] Dhariwal & Nichol, Diffusion Models Beat GANs on Image Synthesis, 2021 (https://arxiv.org/abs/2105.05233)

[10] Ho & Salimans, Classifier-Free Diffusion Guidance, 2022 (https://arxiv.org/abs/2207.12598)

[11] Vaswani et al., Attention Is All You Need, 2017 (https://arxiv.org/abs/1706.03762)

[12] Ronneberger et al., U-Net: Convolutional Networks for Biomedical Image Segmentation, 2015 (https://arxiv.org/abs/1505.04597)

[13] Song et al., DENOISING DIFFUSION IMPLICIT MODELS, 2021 (https://arxiv.org/pdf/2010.02502.pdf)