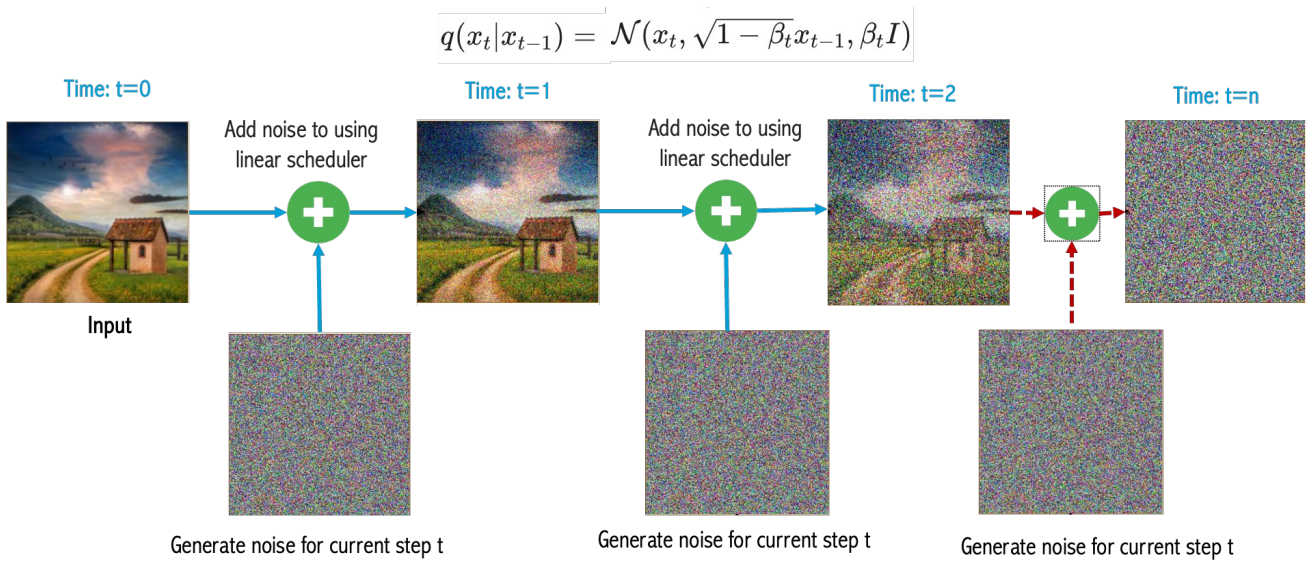
AI VIETNAM
All-in-One Course

Introduction to Diffusion Model



arXiv:1503.03585v8 [cs.LG] 18 Nov 2015

Deep Unsupervised Learning using Nonequilibrium Thermodynamics

Jascha Sohl-Dickstein JASCHA@STANFORD.EDU
Stanford University

Eric A. Weiss EAWEISS@BERKELEY.EDU
University of California, Berkeley

Niru Maheswaranathan NIRUM@STANFORD.EDU
Stanford University

Surya Ganguli SGANGULI@STANFORD.EDU
Stanford University

Abstract

A central problem in machine learning involves modeling complex data-sets using highly flexible families of probability distributions in which learning, sampling, inference, and evaluation are still analytically or computationally tractable. Here, we develop an approach that simultaneously achieves both flexibility and tractability. The essential idea, inspired by non-equilibrium statistical physics, is to systematically and slowly destroy structure in a data distribution through an iterative forward diffusion process. We then learn a reverse diffusion process that restores structure in data, yielding a highly flexible and tractable generative model of the data. This approach allows us to rapidly learn, sample from, and evaluate probabilities in deep generative models with thousands of layers or time steps, as well as to compute conditional and posterior probabilities under the learned model. We additionally release an open source reference implementation of the algorithm.

these models are unable to aptly describe structure in rich datasets. On the other hand, models that are *flexible* can be molded to fit structure in arbitrary data. For example, we can define models in terms of any (non-negative) function $\phi(\mathbf{x})$ yielding the flexible distribution $p(\mathbf{x}) = \frac{\phi(\mathbf{x})}{Z}$, where Z is a normalization constant. However, computing this normalization constant is generally intractable. Evaluating, training, or drawing samples from such flexible models typically requires a very expensive Monte Carlo process.

A variety of analytic approximations exist which ameliorate, but do not remove, this tradeoff–for instance mean field theory and its expansions (T, 1982; Tanaka, 1998), variational Bayes (Jordan et al., 1999), contrastive divergence (Welling & Hinton, 2002; Hinton, 2002), minimum probability flow (Sohl-Dickstein et al., 2011b;a), minimum KL contraction (Lyu, 2011), proper scoring rules (Gneiting & Raftery, 2007; Parry et al., 2012), score matching (Hyvärinen, 2005), pseudolikelihood (Besag, 1975), loopy belief propagation (Murphy et al., 1999), and many, many more. Non-parametric methods (Gershman & Blei, 2012) can also be very effective[1].

Vinh Dinh Nguyen
PhD in Computer Science

Outline

- Objective
- Application of Diffusion Models
- Why Do We Need Diffusion Model?
- Diffusion Model Detail Explanation and Implementation
- Summary

Outline

- Objective
- Application of Diffusion Models
- Why Do We Need Diffusion Model?
- Diffusion Model Detail Explanation and Implementation
- Summary

Objective

Deep Unsupervised Learning using Nonequilibrium Thermodynamics

Jascha Sohl-Dickstein JASCHA@STANFORD.EDU
Stanford University

Eric A. Weiss EAWEISS@BERKELEY.EDU
University of California, Berkeley

Niru Maheswaranathan NIRUM@STANFORD.EDU
Stanford University

Surya Ganguli SGANGULI@STANFORD.EDU
Stanford University

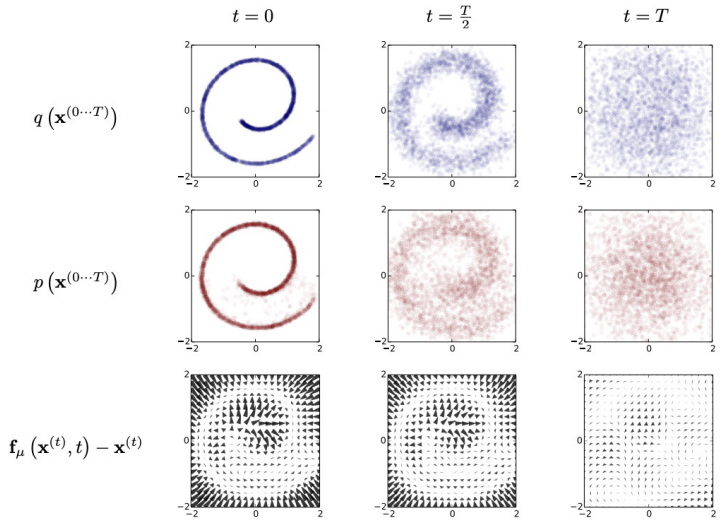
Abstract

A central problem in machine learning involves modeling complex data-sets using highly flexible families of probability distributions in which learning, sampling, inference, and evaluation are still analytically or computationally tractable. Here, we develop an approach that simultaneously achieves both flexibility and tractability. The essential idea, inspired by non-equilibrium statistical physics, is to systematically and slowly destroy structure in a data distribution through an iterative forward diffusion process. We then learn a reverse diffusion process that restores structure in data, yielding a highly flexible and tractable generative model of the data. This approach allows us to rapidly learn, sample from, and evaluate probabilities in deep generative models with thousands of layers or time steps, as well as to compute conditional and posterior probabilities under the learned model. We additionally release an open source reference implementation of the algorithm.

these models are unable to aptly describe structure in rich datasets. On the other hand, models that are *flexible* can be molded to fit structure in arbitrary data. For example, we can define models in terms of any (non-negative) function $\phi(\mathbf{x})$ yielding the flexible distribution $p(\mathbf{x}) = \frac{\phi(\mathbf{x})}{Z}$, where Z is a normalization constant. However, computing this normalization constant is generally intractable. Evaluating, training, or drawing samples from such flexible models typically requires a very expensive Monte Carlo process.

A variety of analytic approximations exist which ameliorate, but do not remove, this tradeoff–for instance mean field theory and its expansions (T, 1982; Tanaka, 1998), variational Bayes (Jordan et al., 1999), contrastive divergence (Welling & Hinton, 2002; Hinton, 2002), minimum probability flow (Sohl-Dickstein et al., 2011b;a), minimum KL contraction (Lyu, 2011), proper scoring rules (Gneiting & Raftery, 2007; Parry et al., 2012), score matching (Hyvärinen, 2005), pseudolikelihood (Besag, 1975), loopy belief propagation (Murphy et al., 1999), and many, many more. Non-parametric methods (Gershman & Blei, 2012) can also be very effective[1].

1. Understand a Diffusion Model
2. Understand a Forward Diffusion Process
3. Understand a Reverse Diffusion Process
4. Be able to Implement a Diffusion Model Using Pytorch



Outline

- Objective
- Application of Diffusion Models
- Why Do We Need Diffusion Model?
- Diffusion Model Detail Explanation and Implementation
- Summary

Applications of Diffusion Model

Textual Inversion

Text To Videos

Text To 3D

Text To Motion

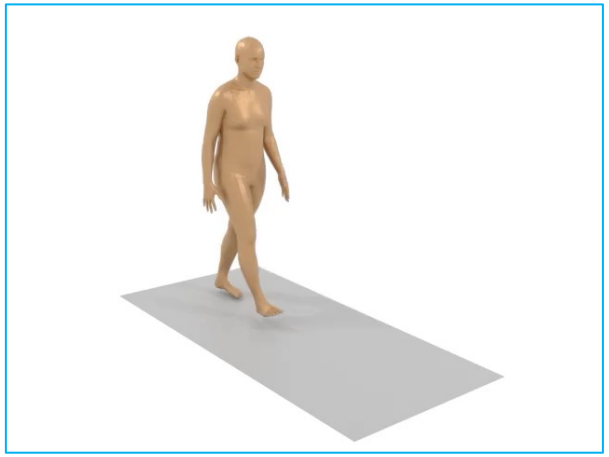
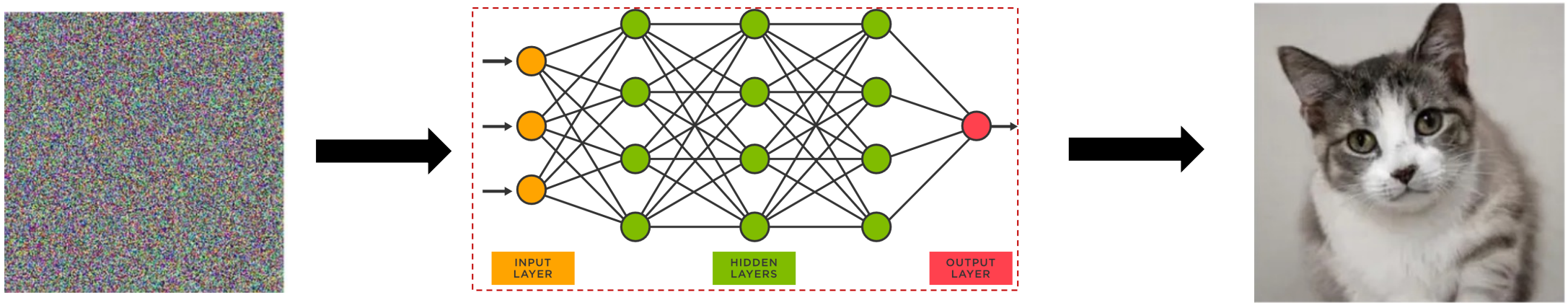


Image To Image

Image Inpainting

AutoEncoder, Variational AutoEncoder, and, Generative Adversarial Network: Limitations

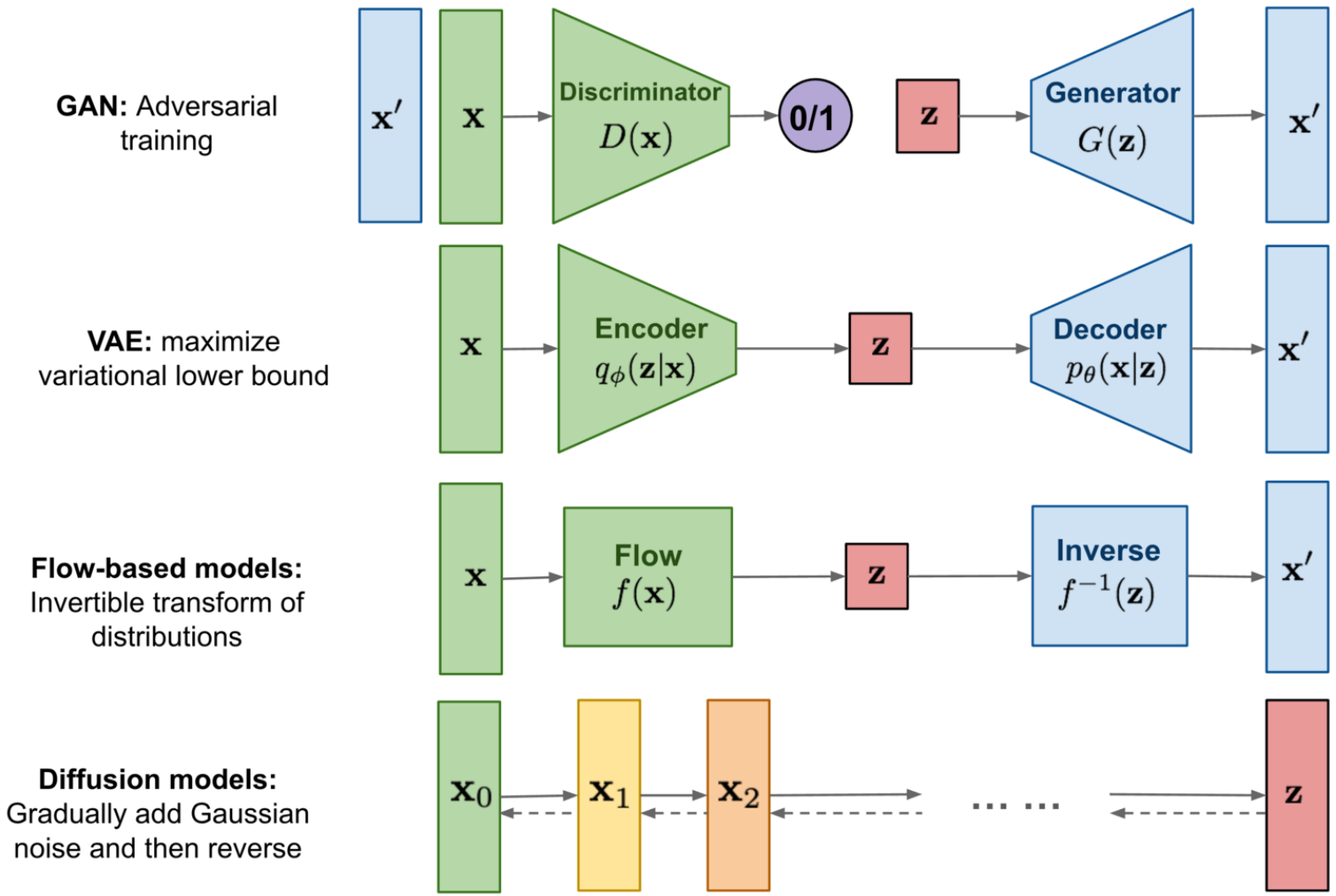
Illustration of a generative model based on noise input



Input Network Output

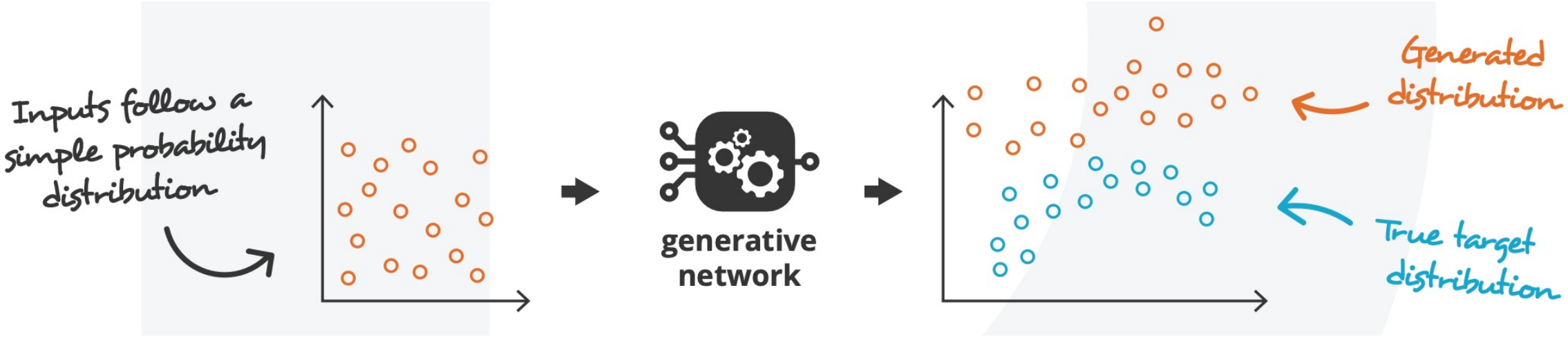
An example of artifacts in an image generated by GAN. You can see the artifacts in the cat's body and the distortion of the background at the top of the image.

AutoEncoder, Variational AutoEncoder, and, Generative Adversarial Network: Limitations



GAN's Main Idea: Review

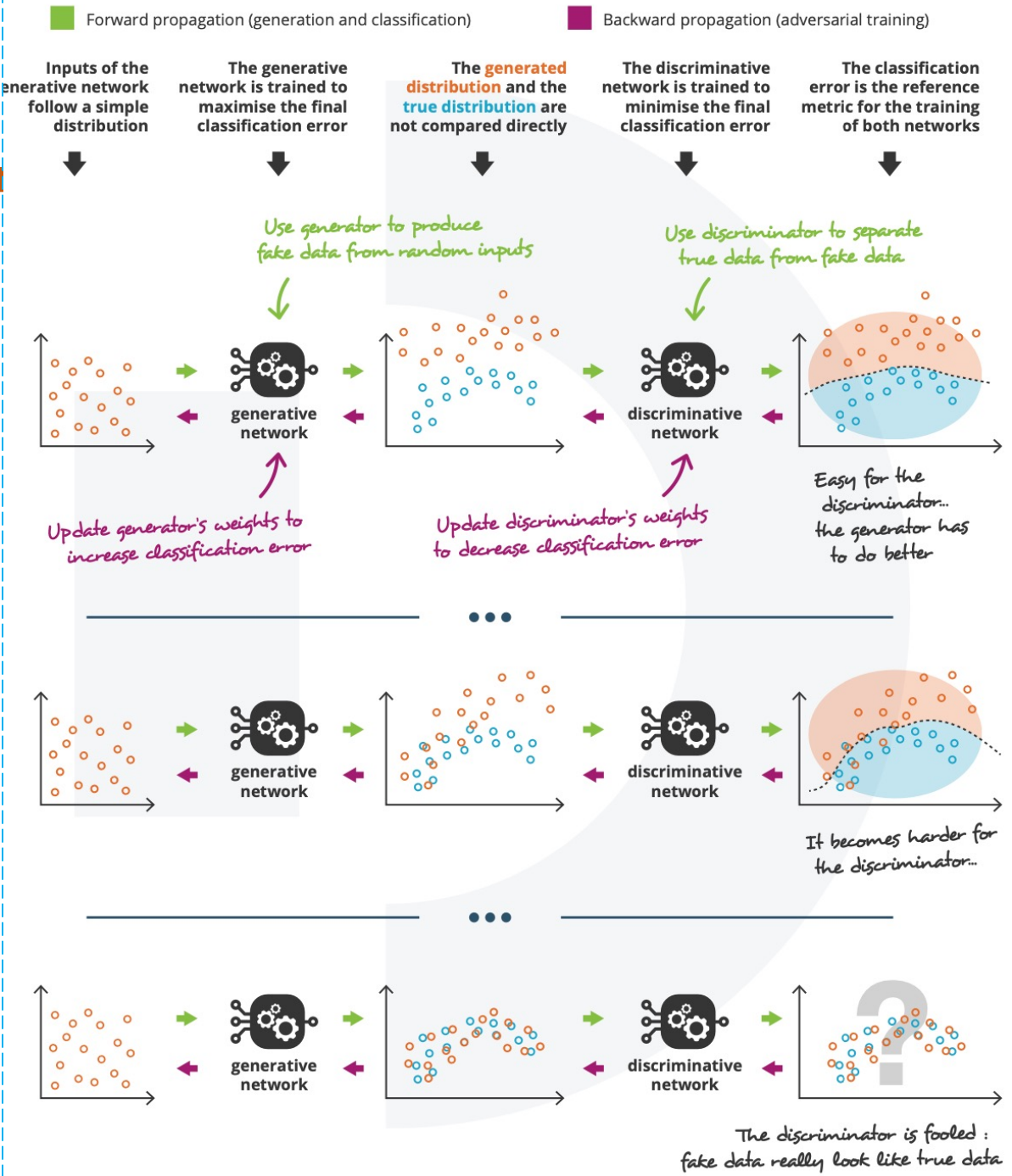
The first key point lies in the idea that **there exists a probability distribution describing the kind of data we try to generate**. We want to be able to sample new data from that distribution



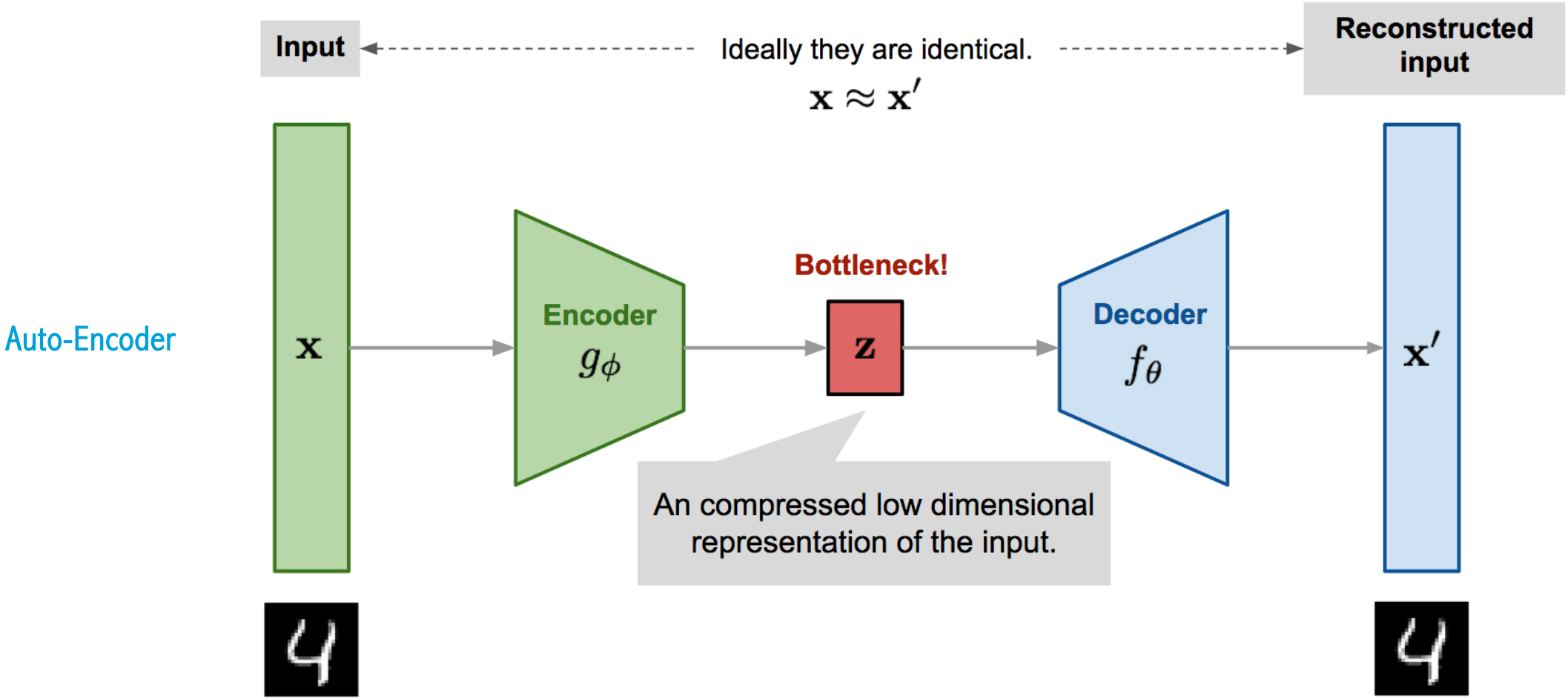
Source: https://towardsdatascience.com/understanding-generative-adversarial-networks-gans-cd6e4651a29

GAN Review

Source: https://towardsdatascience.com/understanding-generative-adversarial-networks-gans-cd6e4651a29

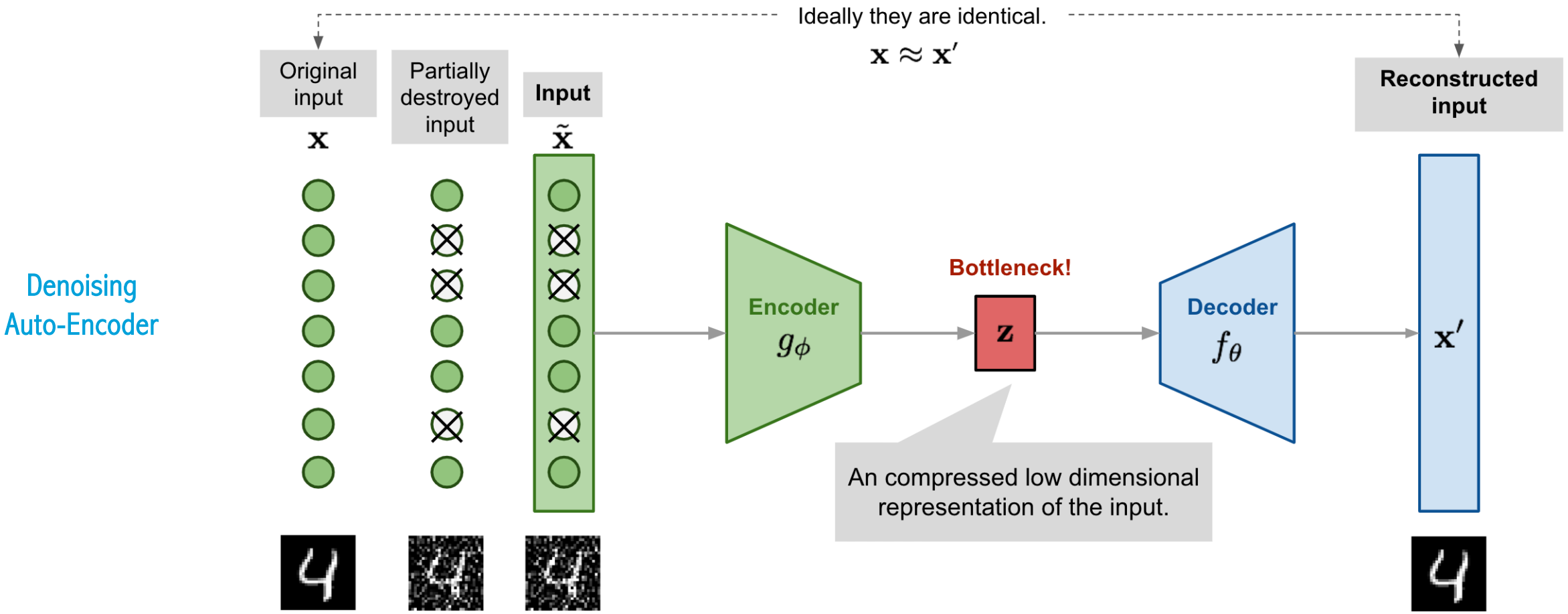


VAE's Main Idea: Review



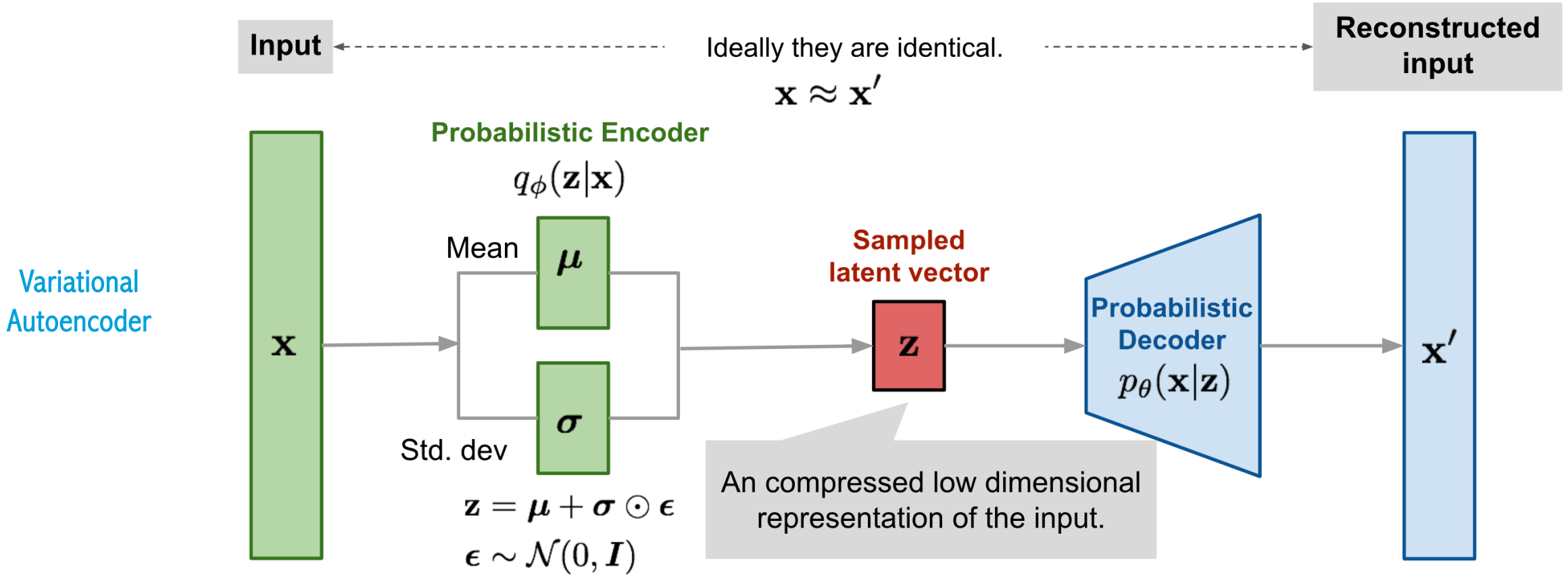
Source: https://towardsdatascience.com/understanding-generative-adversarial-networks-gans-cd6e4651a29

VAE's Main Idea: Review



Source: https://towardsdatascience.com/understanding-generative-adversarial-networks-gans-cd6e4651a29

VAE's Main Idea: Review



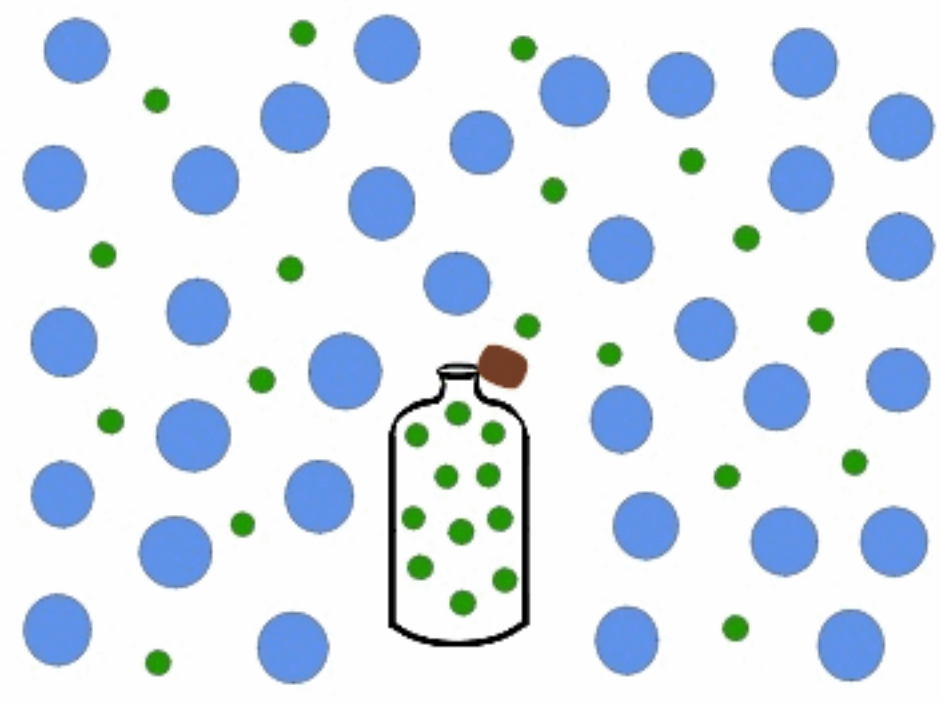
Source: https://towardsdatascience.com/understanding-generative-adversarial-networks-gans-cd6e4651a29

Outline

- Objective
- Application of Diffusion Models
- Why Do We Need Diffusion Models?
- Diffusion Model Detail Explanation and Implementation
- Summary

Why diffusion model?

In non-equilibrium statistical physics, the diffusion process refers to: "The movement of particles or molecules from an area of high concentration to an area of low concentration, driven by a gradient in concentration."

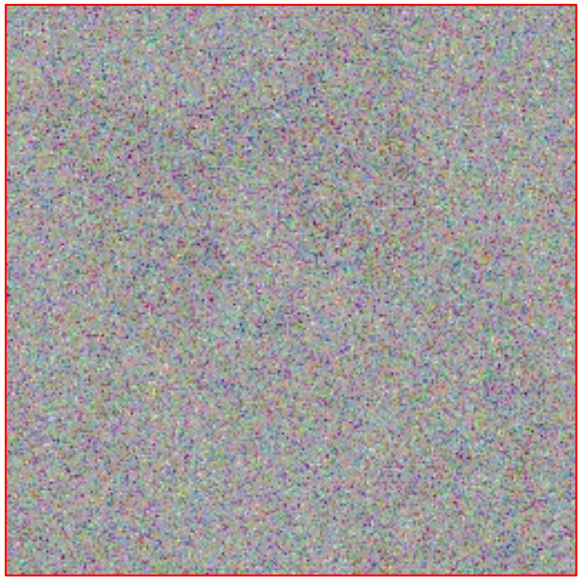


Source: https://chem.libretexts.org/Under_Construction/Purgatory/Kinetic_Theory_of_Gases

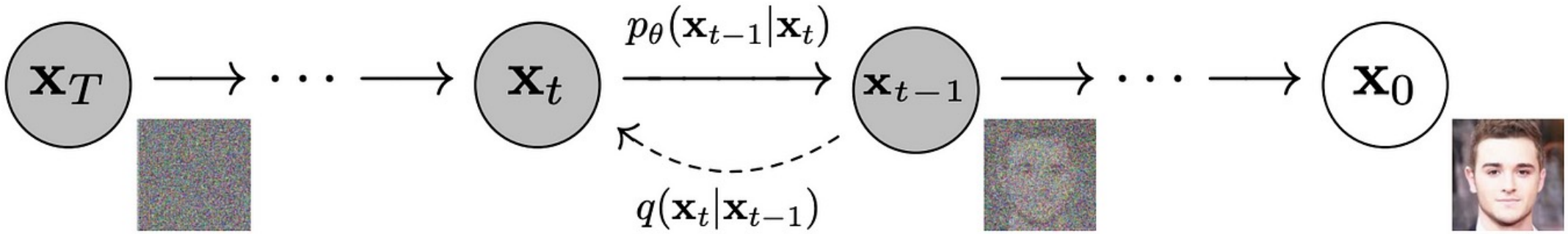
Why diffusion model?

it could be very beneficial if we wouldn't need to generate the image in one forward pass but iteratively improve it step by step. In this way, even if the model makes some mistakes, it has the opportunity to find and correct these mistakes in the later steps.

Diffusion models use many steps to generate the final image. In each step, the model makes a refinement of the result of the previous step. In this way, the model can gradually improve the result, get a better image after each step, and also fix errors that are made along the way.



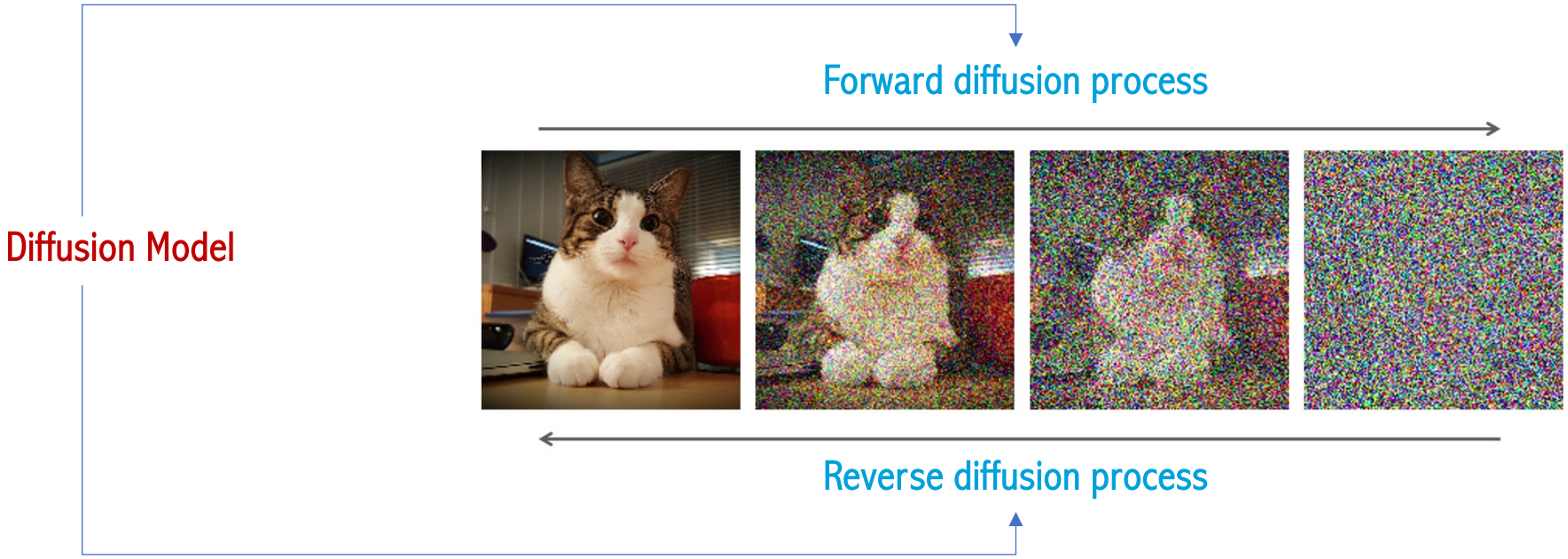
Diffusion Model



What is a diffusion model?

The idea of the diffusion model is not that old. In the 2015 paper called "Deep Unsupervised Learning using Nonequilibrium Thermodynamics", the Authors described it like this:

"The essential idea, inspired by non-equilibrium statistical physics, is to systematically and **slowly destroy structure in a data distribution** through an **iterative forward diffusion process**. We then learn a **reverse diffusion process** that **restores structure in data**, yielding a highly flexible and tractable generative model of the data".



Outline

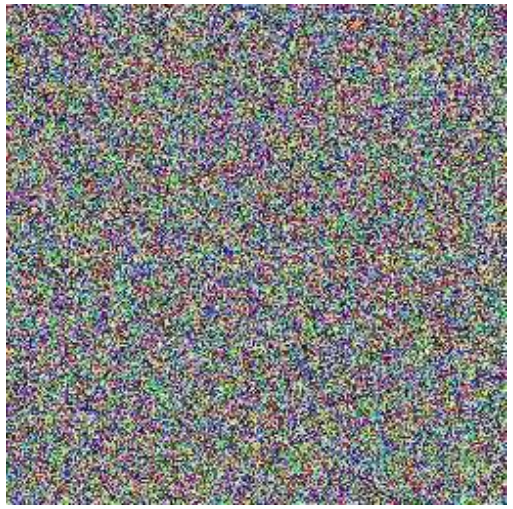
- Objective
- Application of Diffusion Models
- Why Do We Need Diffusion Model?
- Diffusion Model Detail Explanation and Implementation
- Summary

Forward Diffusion Process

Input

Some none-random distribution

Adding noise



Output

Purse noise

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t, \sqrt{1-\beta_t}x_{t-1}, \beta_t I) \qquad \{\beta_t \in (0,1)\}_{t=1}^{T}$$

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

β_t refers to something called **schedule** and values can range from 0 to 1. The values are usually kept low to prevent variance from exploding

The data sample $\mathbf{x}_0$ gradually loses its distinguishable features as the step t becomes larger. Eventually when $T \to \infty$, $\mathbf{x}_T$ is equivalent to an isotropic Gaussian distribution.

Gaussian Distribution: Recall

For a real-valued random variable x, the probability density function is defined as:

$$p(x;\mu,\sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} exp(\frac{-1}{2}(\frac{x-\mu}{\sigma})^2)$$

This is the probability density function for multivariate case:

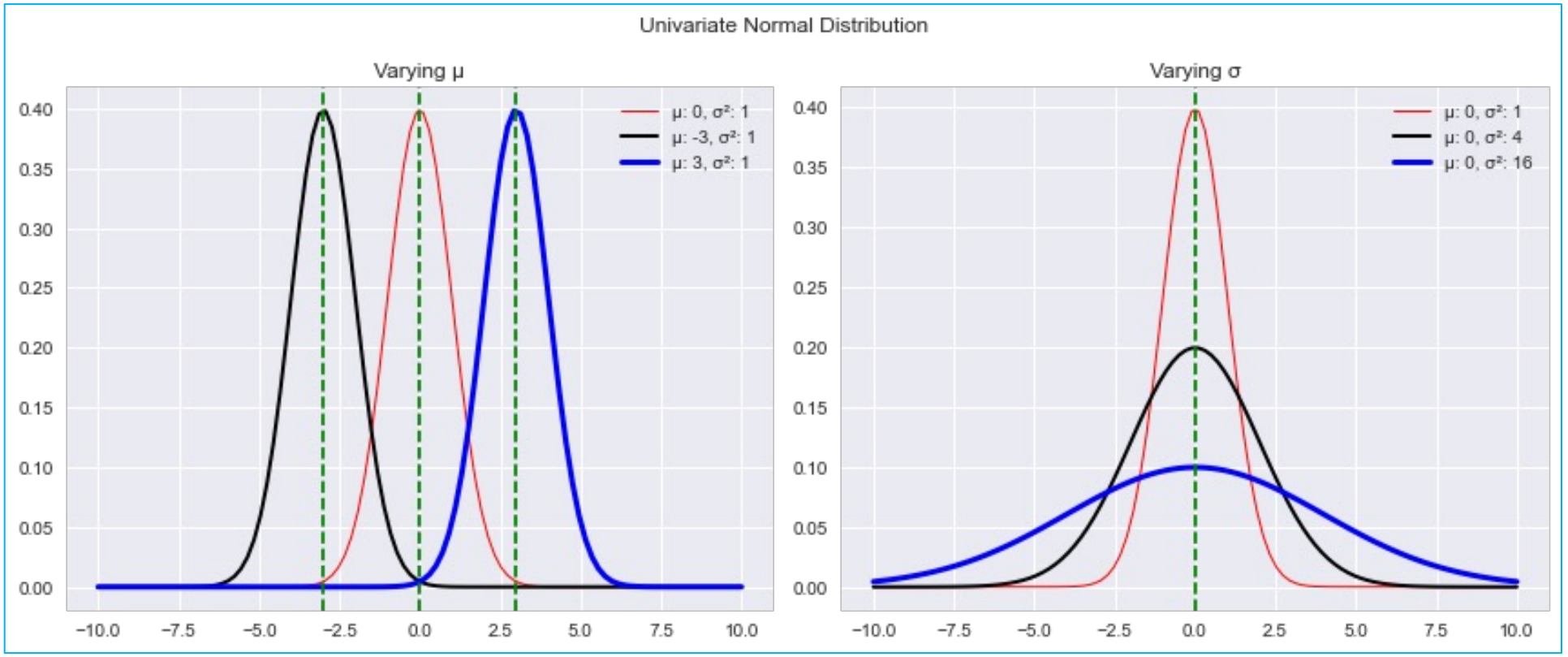
$$p(x;\mu,\Sigma) = \frac{1}{(2\pi)^{d/2}\,|\Sigma|^{1/2}}\; exp\,(\frac{-1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu))$$

An isotropic Gaussian is one where the covariance matrix Σ can be represented in this form:

$$\Sigma = \sigma^2 I$$

1. I is the Identity matrix
2. σ^2 is the scalar variance

Gaussian Distribution: Recall

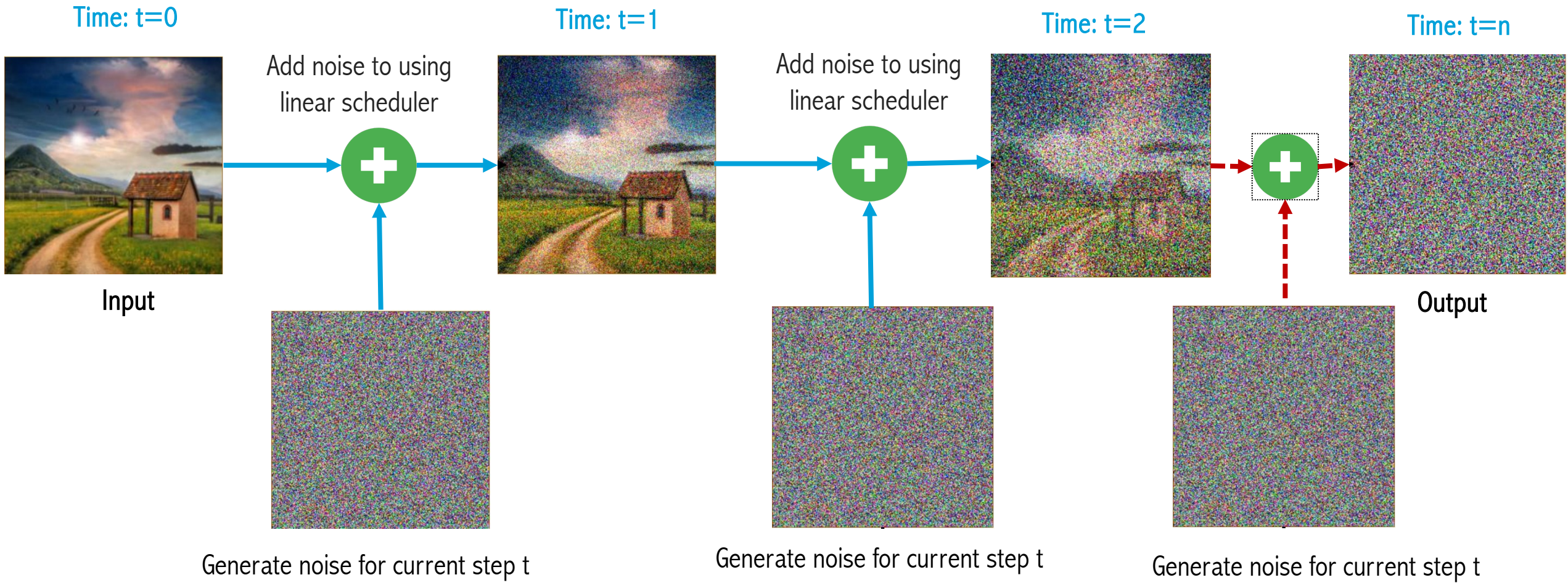


Moving the mean value shifts the distribution from right to left while changing sigma affects the shape of the distribution. As the value of sigma increases, the curve gets more and more flat. Let's see that in action.

AI VIETNAM
All-in-One Course

Forward Diffusion Process

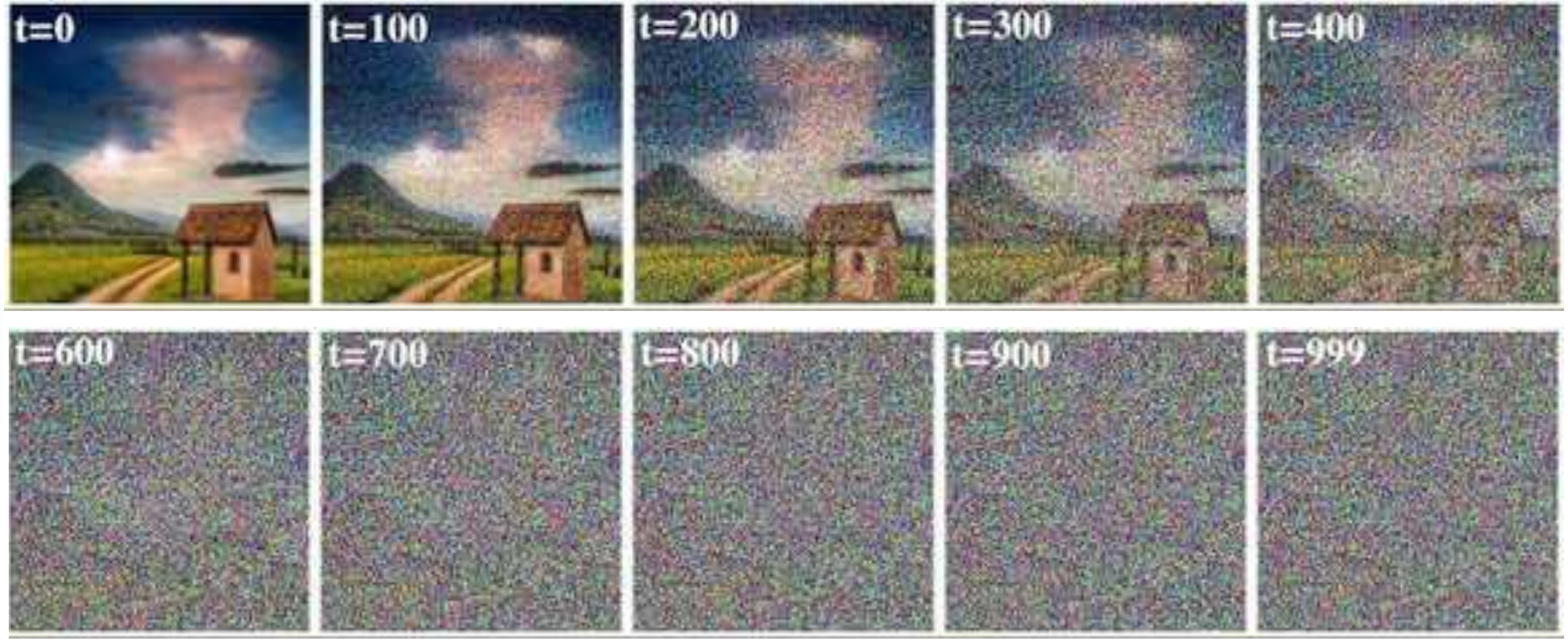
$$q(x_t|x_{t-1}) = \mathcal{N}(x_t, \sqrt{1-\beta_t}x_{t-1}, \beta_t I)$$



Why do we need the scaling factor $(1-\beta_t)$ to shift the mean?

Forward Diffusion Process

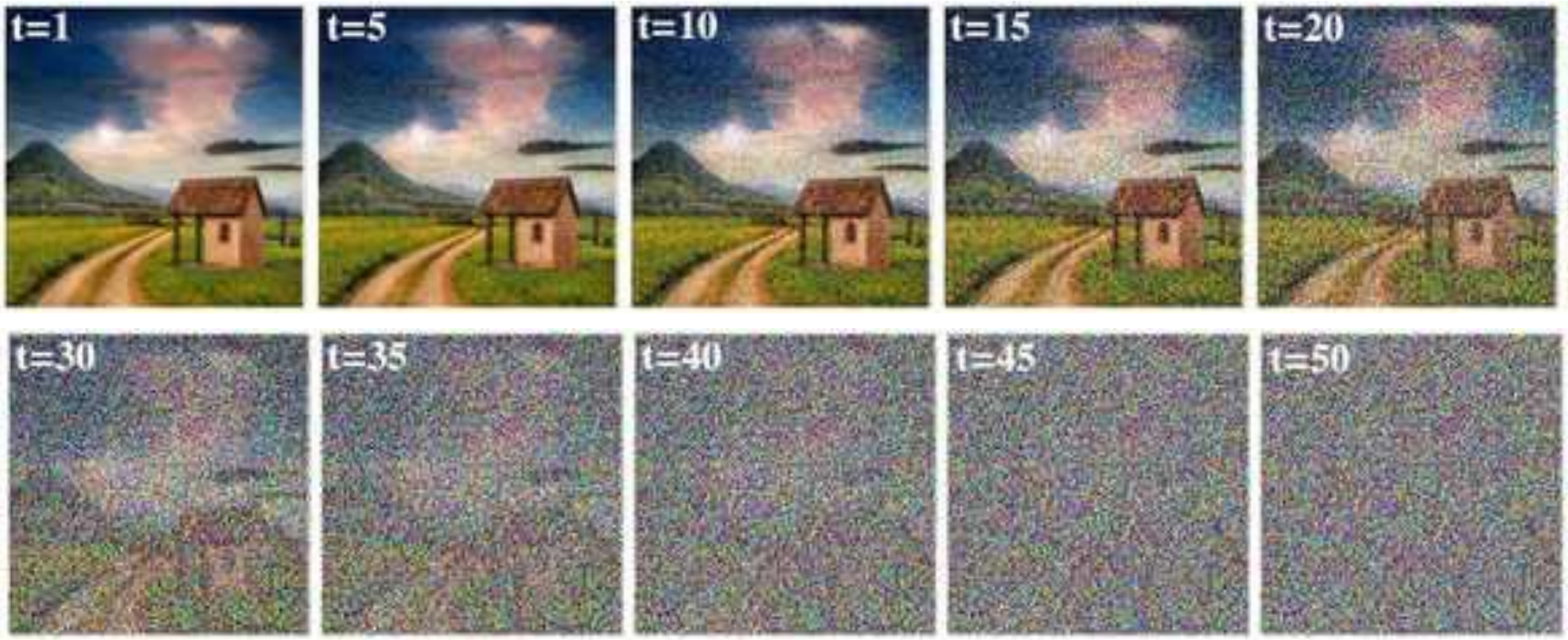
$$q(x_t|x_{t-1}) = \mathcal{N}(x_t, \sqrt{1-\beta_t}x_{t-1}, \beta_t I)$$



*Entire forward diffusion process using **linear schedule** with 1000 time steps*

Forward Diffusion Process

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

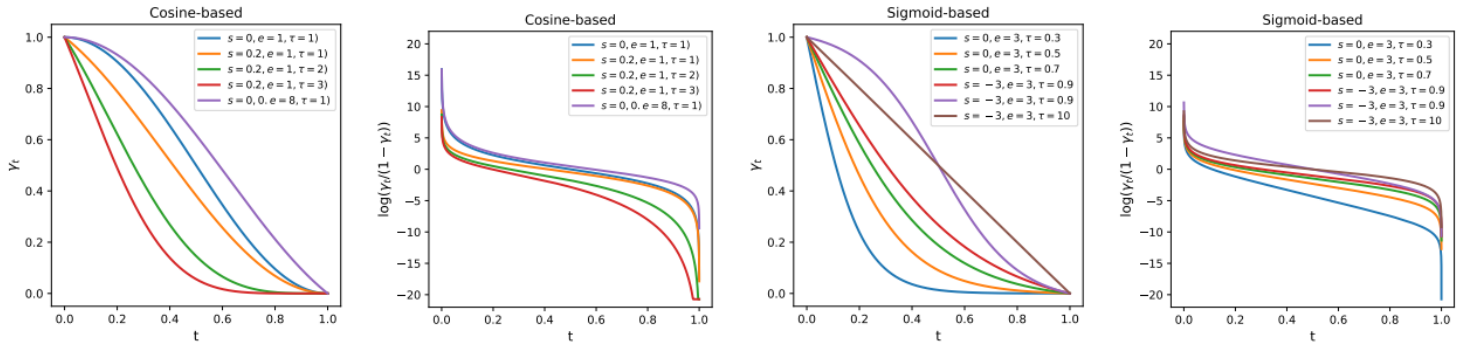


*Entire forward diffusion process using **Cosine schedule** with 50 time steps*

Importance of Noise Scheduling for Diffusion Models

Algorithm 1 Continuous time noise scheduling function $\gamma(t)$.

```
def simple_linear_schedule(t, clip_min=1e-9):
  # A gamma function that simply is 1-t.
  return np.clip(1 - t, clip_min, 1.)

def sigmoid_schedule(t, start=-3, end=3, tau=1.0, clip_min=1e-9):
  # A gamma function based on sigmoid function.
  v_start = sigmoid(start / tau)
  v_end = sigmoid(end / tau)
  output = sigmoid((t * (end - start) + start) / tau)
  output = (v_end - output) / (v_end - v_start)
  return np.clip(output, clip_min, 1.)

def cosine_schedule(t, start=0, end=1, tau=1, clip_min=1e-9):
  # A gamma function based on cosine function.
  v_start = math.cos(start * math.pi / 2) ** (2 * tau)
  v_end = math.cos(end * math.pi / 2) ** (2 * tau)
  output = math.cos((t * (end - start) + start) * math.pi / 2) ** (2 * tau)
  output = (v_end - output) / (v_end - v_start)
  return np.clip(output, clip_min, 1.)
```



(a) Cosine (b) Cosine (logSNR) (c) Sigmoid (d) Sigmoid (logSNR)

Forward Diffussion Process: Theory

The training process doesn't use examples in line with the forward process but rather it uses samples from arbitrary timestep t

Each training step we would need to iterate through t steps to generate 1 training sample

Entire noise to be added at *T*

$$q(x_{1:T}|x_0) := \prod_{t=1}^{T} q(x_t|x_{t-1})$$

Function composition

$$q_t(q_{t-1}(q_{t-2}(q_{t-3}(\cdots q_1(x_0)))))$$

Normal distribution formula for t=1

$$q(x_1|x_0) = \mathcal{N}(x_1, \sqrt{1-\beta_1}x_0, \beta_1 I)$$

The formula uses the result from the t=2 step

$$q(x_2|x_1) = \mathcal{N}(x_2, \sqrt{1-\beta_2}x_1, \beta_2 I)$$

Mean is dependent on the previous step

Forward Diffussion Process: Theory

Normal distribution formula for t=1

$$q(x_1|x_0) = \mathcal{N}(x_1, \sqrt{1-\beta_1}x_0, \beta_1 I)$$

The formula uses the result from the t=2 step

$$q(x_2|x_1) = \mathcal{N}(x_2, \sqrt{1-\beta_2}x_1, \beta_2 I)$$

Mean is dependent on the previous step

$$\alpha_t = 1 - \beta_t \Rightarrow q(x_1|x_0) = \mathcal{N}(x_1, \sqrt{\alpha_1}x_0, (1-\alpha_1)I)$$

Mean is changing through the steps

$$\begin{aligned}
\mu_t &= \sqrt{\alpha_t}x_{t-1} \\
&= \sqrt{\alpha_t} * \sqrt{\alpha_{t-1}}x_{t-2} = \sqrt{\alpha_t\alpha_{t-1}}x_{t-2} \\
&= \sqrt{\alpha_t\alpha_{t-1}} * \sqrt{\alpha_{t-2}}x_{t-3} = \sqrt{\alpha_t\alpha_{t-1}\alpha_{t-2}}x_{t-3} \\
&= \sqrt{\alpha_t\alpha_{t-1}\alpha_{t-2}\cdots} * \sqrt{\alpha_1}x_0 = \sqrt{\alpha_t\alpha_{t-1}\alpha_{t-2}\cdots\alpha_1}x_0 \\
&= \sqrt{\bar{\alpha}_t}x_0
\end{aligned}$$

$$\bar{\alpha}_t := \prod_{s=1}^{t} a_s$$

$$q(x_{1:T}|x_0) := \prod_{t=1}^{T} q(x_t|x_{t-1}) \Rightarrow q(x_t|x_0) = \mathcal{N}(x_t, \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)I)$$

Need to apply the reparameterization trick

Everything is great but that is not computable

Forward Diffussion Process: Theory

$$q(x_{1:T}|x_0) := \prod_{t=1}^{T} q(x_t|x_{t-1}) \quad \Rightarrow \quad q(x_t|x_0) = \mathcal{N}(x_t, \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)I)$$

Need to apply the reparameterization trick

Everything is great but that is not computable

$$(\mathcal{N}(\mu, \sigma^2) = \mu + \sigma * \epsilon)$$

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

ϵ is from $\mathcal{N}(0, 1)$

$$q(x_t|x_0) = \mathcal{N}(x_t, \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)I) = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon$$

We can calculate noise at any arbitrary step t

Forward Diffusion Process: Theory

Distribution of the noised images — Output — Mean μ_t — Variance Σ_t

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

Notations:

t : time step (from 0 to T)

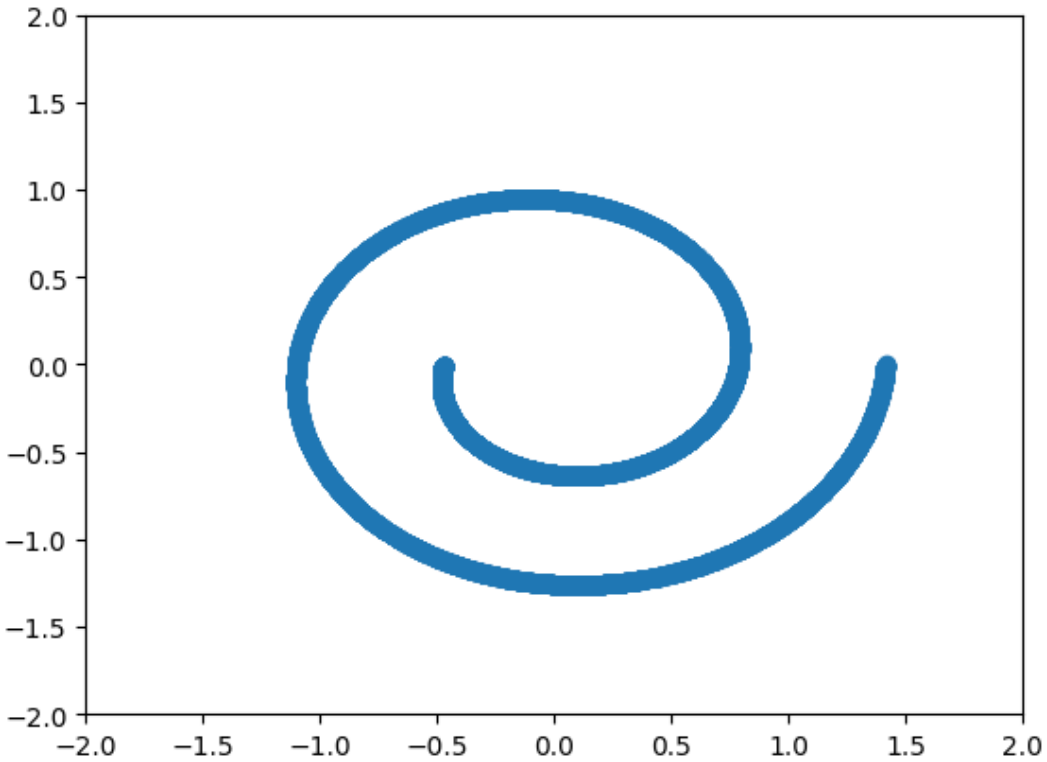
x_0 : a data sampled from the real data distrition $q(x)$ (i.e. $x_0 \sim q(x)$)

β_t : variance schedule ($0 \leq \beta_t \leq 1$, and β_0 = small number, β_T = large number)

I : identity matrix

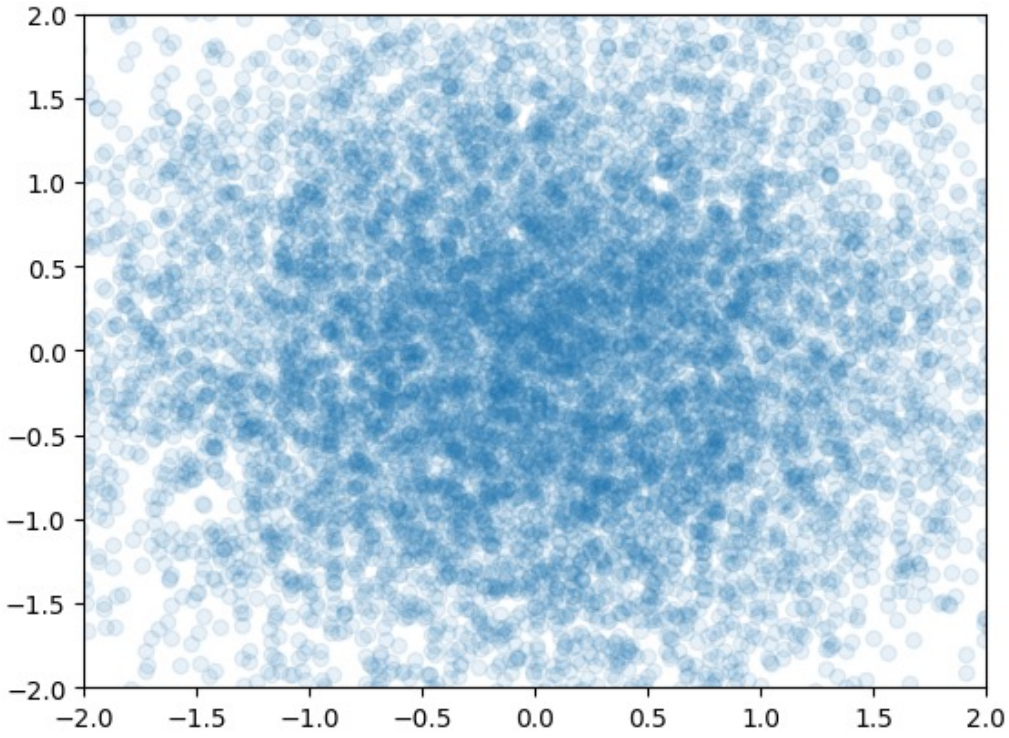
Forward Diffussion Process: Implementation

Swiss roll dataset



Input

Noise

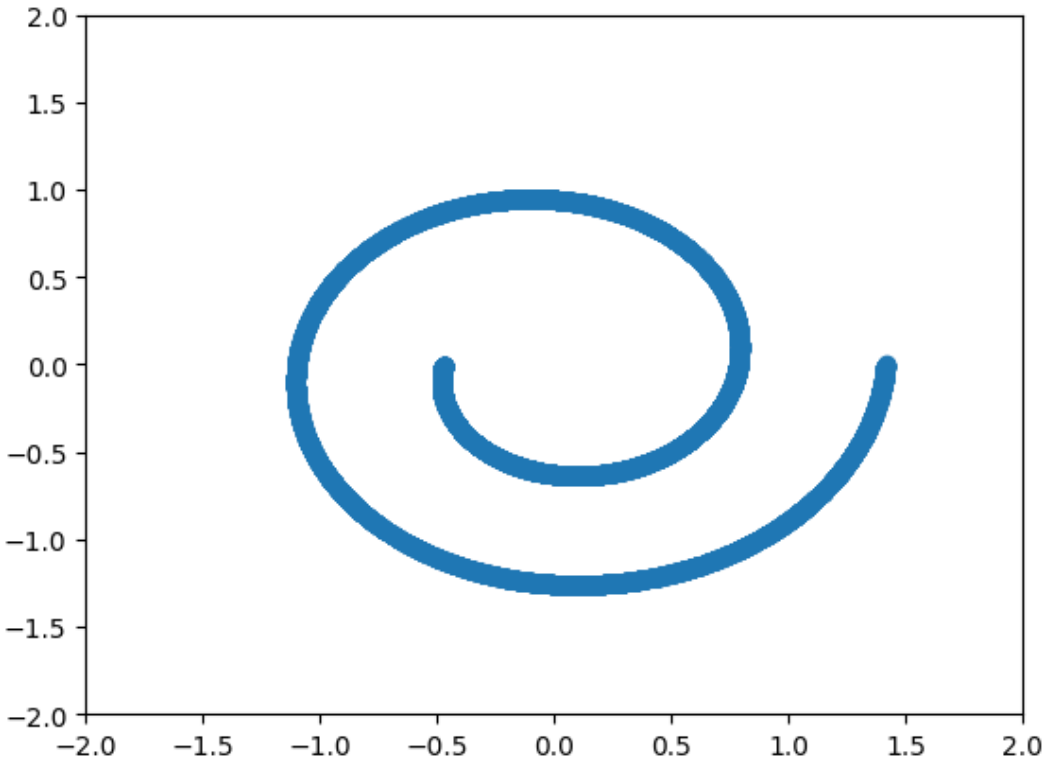


Output

$$q(x_{1:T}|x_0) := \prod_{t=1}^{T} q(x_t|x_{t-1})$$

Forward Diffussion Process: Implementation

Create swiss roll dataset

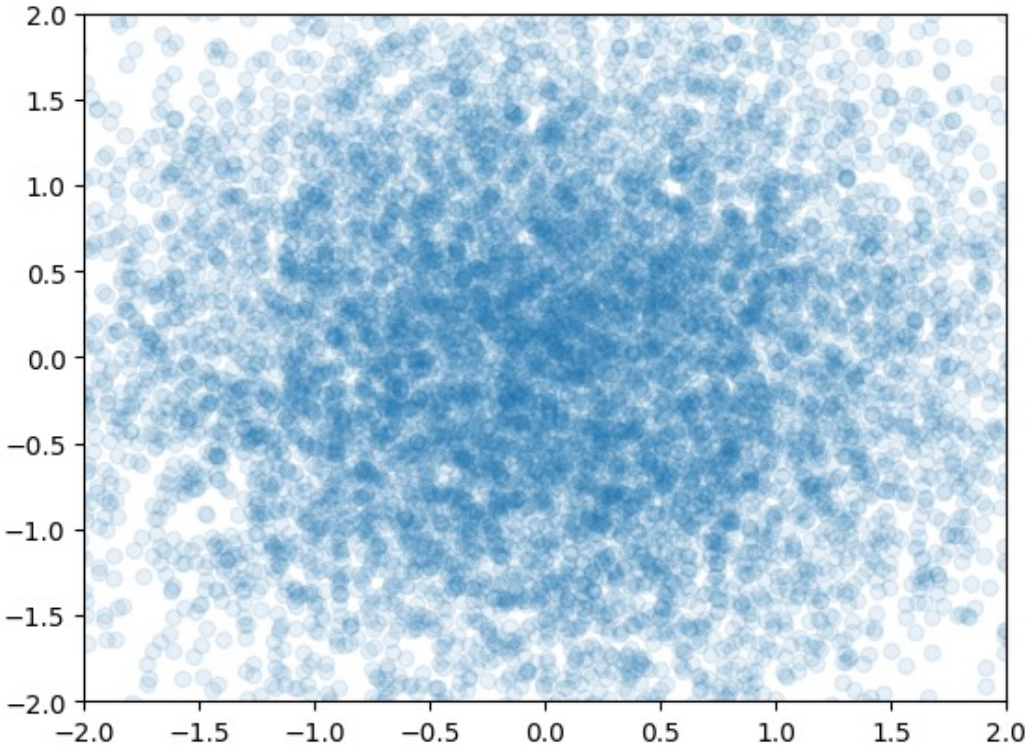



```
import numpy as np
import torch
import matplotlib.pyplot as plt
from sklearn.datasets import make_swiss_roll
```

```
n_samples = 10_000
data, _ = make_swiss_roll(n_samples)
data = data[:, [2, 0]] / 10
data = data * np.array([1, -1])
```

```
plt.scatter(data[:, 0], data[:, 1])
plt.xlim([-2, 2])
plt.ylim([-2, 2])
```

Forward Diffussion Process: Implementation

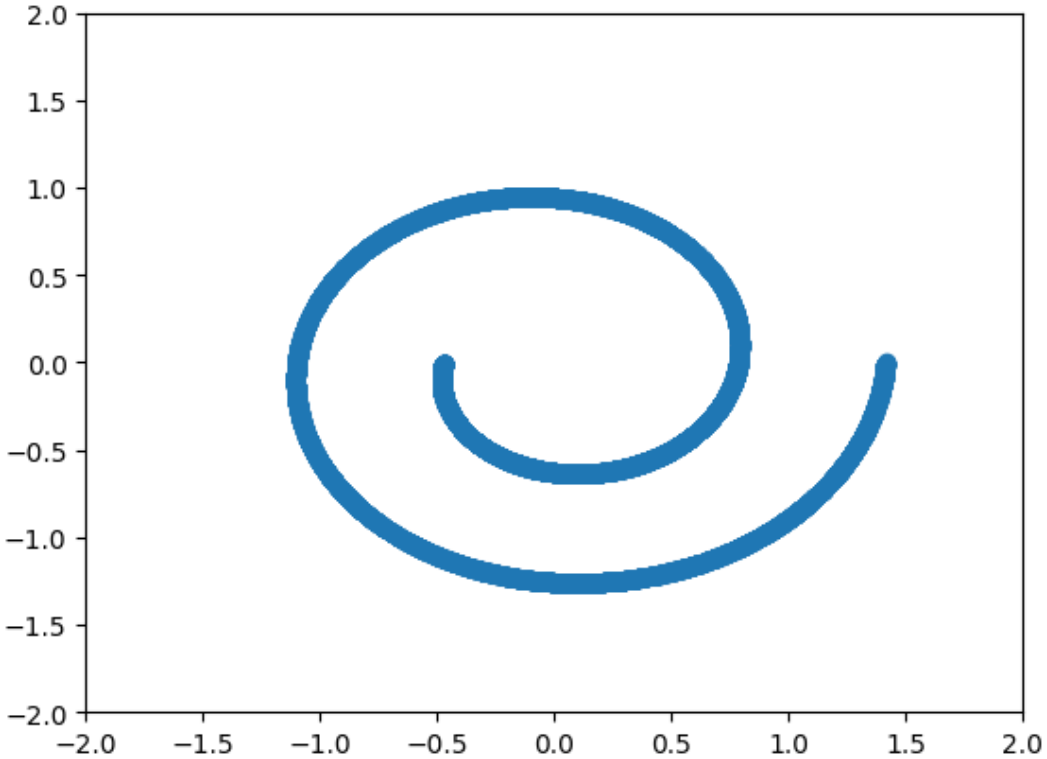



```python
def forward_process(data, T, betas):
    for t in range(T):
        beta_t = betas[t]
        mu = data * torch.sqrt(1 - beta_t)
        std = torch.sqrt(beta_t)
        # Sample from q(x_t | x_{t-1})
        data = mu + torch.randn_like(data) * std # data ~ N(mu, std)
    return data
```

$$q(x_{1:T}|x_0) := \prod_{t=1}^{T} q(x_t|x_{t-1})$$

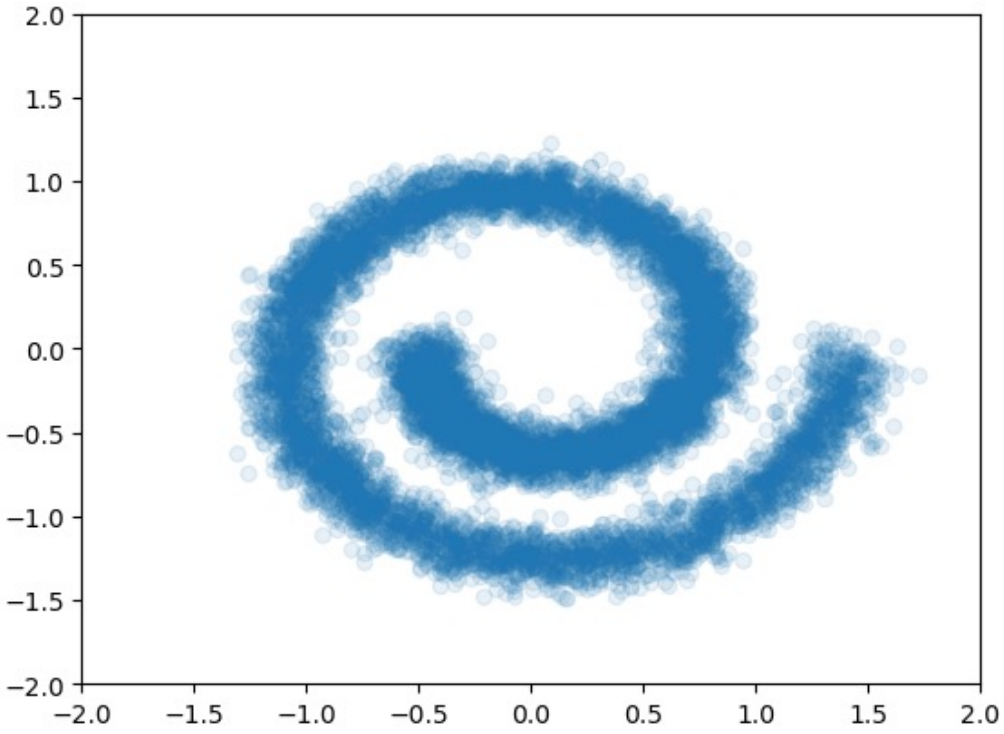
Forward Diffussion Process: Kernel Trick

Swiss roll dataset



Input

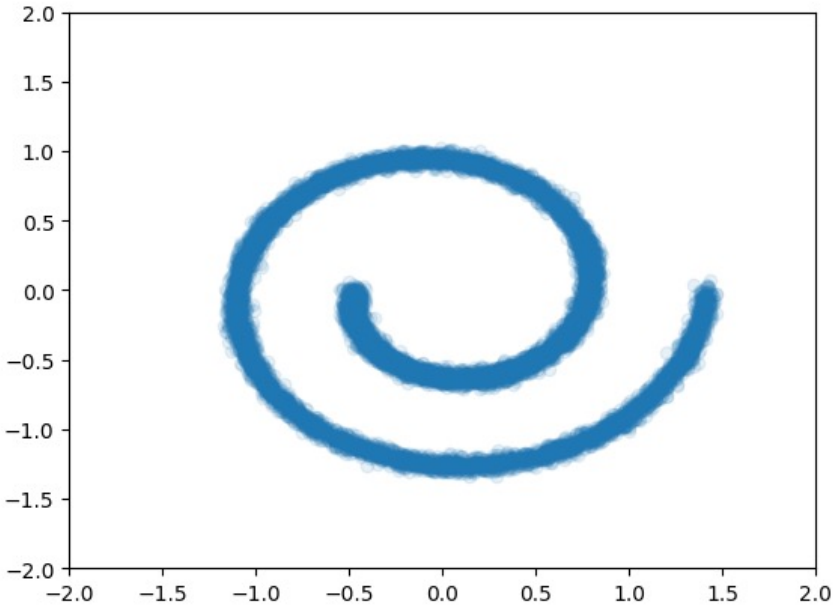
Noise



Output

$$q(x_t|x_0) = \mathcal{N}(x_t, \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)I) = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon$$

Forward Diffussion Process: Implementation

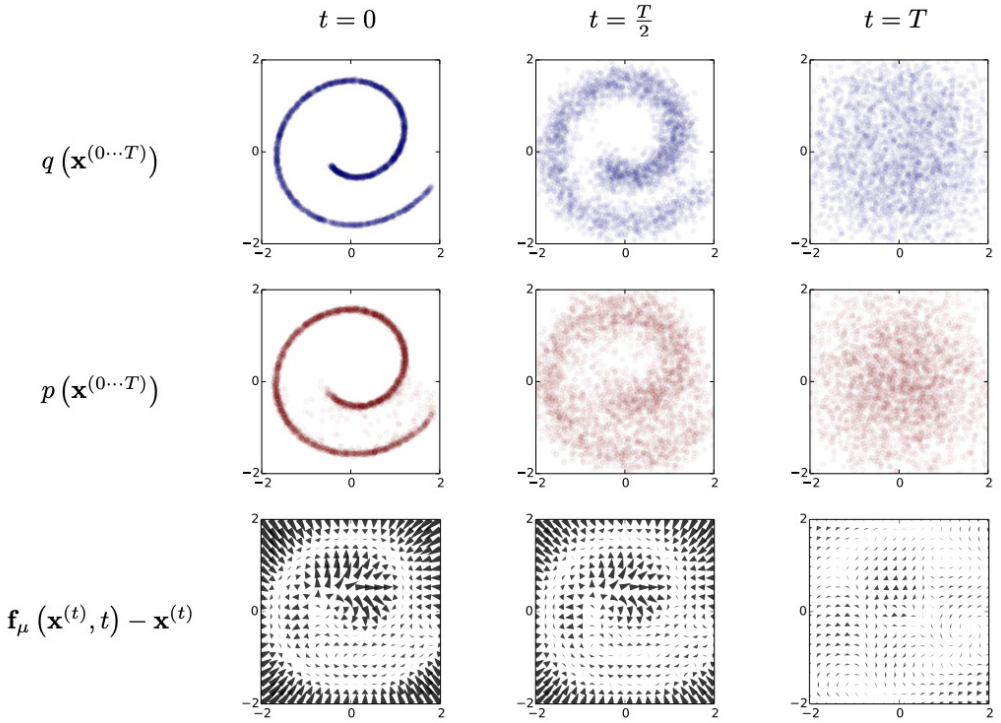



```python
class DiffusionModel():

    def __init__(self, T):

        self.betas = torch.sigmoid(torch.linspace(-18, 10, T)) * (3e-1 - 1e-5) + 1e-5
        self.alphas = 1 - self.betas
        self.alphas_bar = torch.cumprod(self.alphas, 0)

    def forward_process(self, x0, t):
        """
        :param t: Number of diffusion steps
        """

        assert t > 0, 't should be greater than 0'

        t = t -1 # Because we start indexing at 0

        mu = torch.sqrt(self.alphas_bar[t]) * x0
        std = torch.sqrt(1 - self.alphas_bar[t])
        epsilon = torch.randn_like(x0)

        return mu + epsilon * std # data ~ N(mu, std)
```

```python
model = DiffusionModel(50)
x0 = torch.from_numpy(data)
xT = model.forward_process(x0, 20)
```

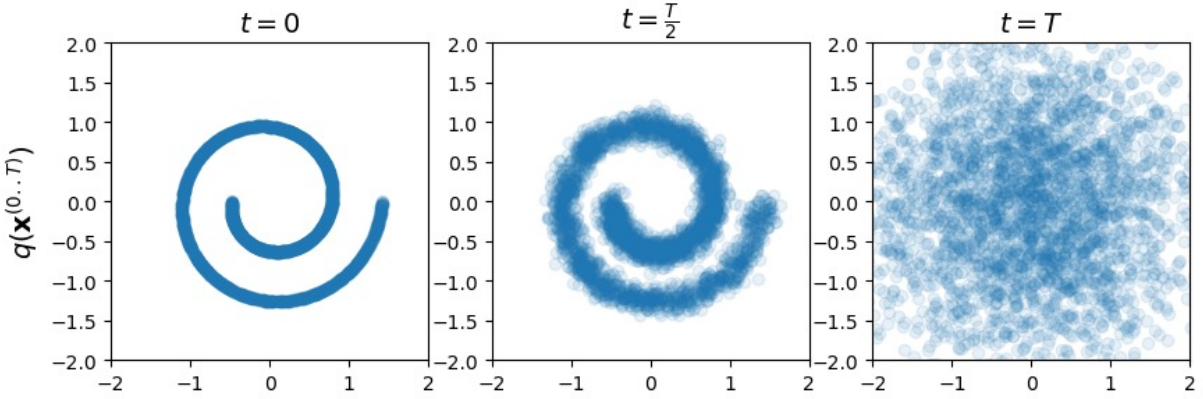
Forward Diffussion Process: Visualization

Results from the paper



https://arxiv.org/pdf/1503.03585.pdf

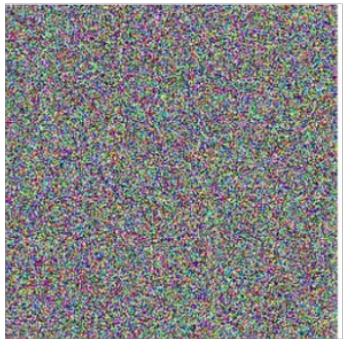
Our implementation result



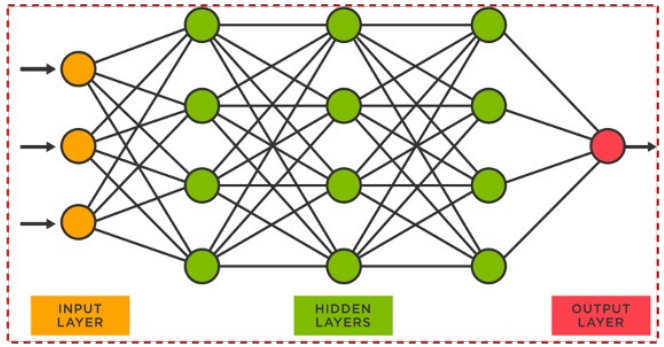

```
fontsize = 14
fig = plt.figure(figsize=(10, 3))

data = [x0, model.forward_process(x0, 20), model.forward_process(x0, 40)]
for i in range(3):

    plt.subplot(1, 3, 1+i)
    plt.scatter(data[i][:, 0].data.numpy(), data[i][:, 1].data.numpy(), alpha=0.1)
    plt.xlim([-2, 2])
    plt.ylim([-2, 2])
    plt.gca().set_aspect('equal')

    if i == 0: plt.ylabel(r'$q(\mathbf{x}^{(0..T)})$', fontsize=fontsize)
    if i == 0: plt.title(r'$t=0$', fontsize=fontsize)
    if i == 1: plt.title(r'$t=\frac{T}{2}$', fontsize=fontsize)
    if i == 2: plt.title(r'$t=T$', fontsize=fontsize)
plt.savefig('forward_process.png', bbox_inches='tight')
```

Reverse Diffussion Process

The goal of the reverse diffusion process is to convert pure noise into an image



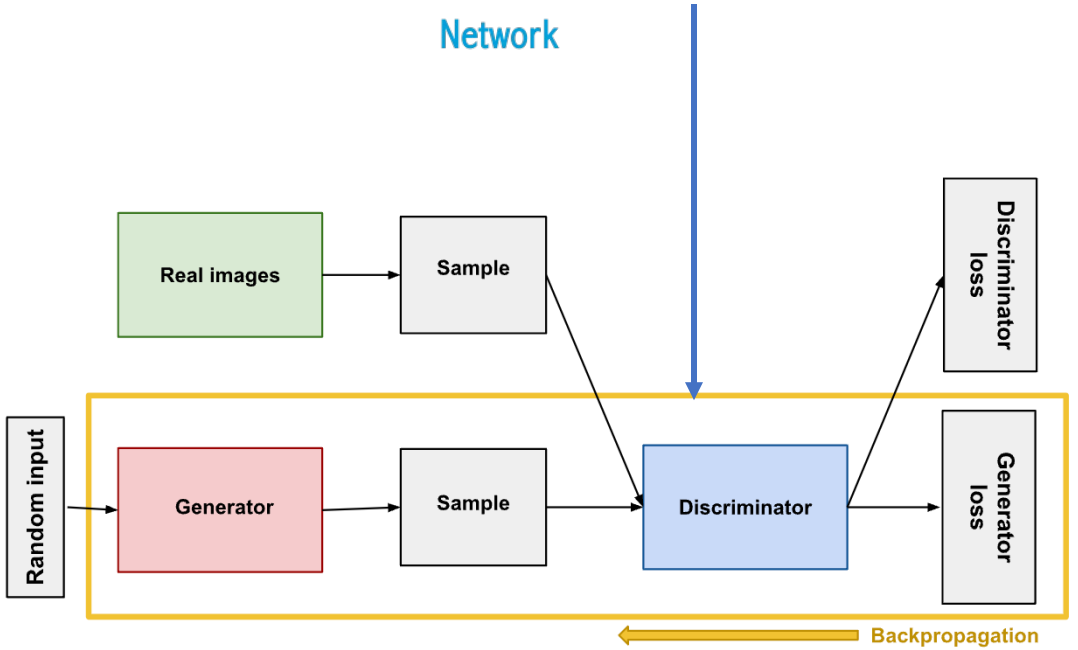
Input



Network

Output

GAN

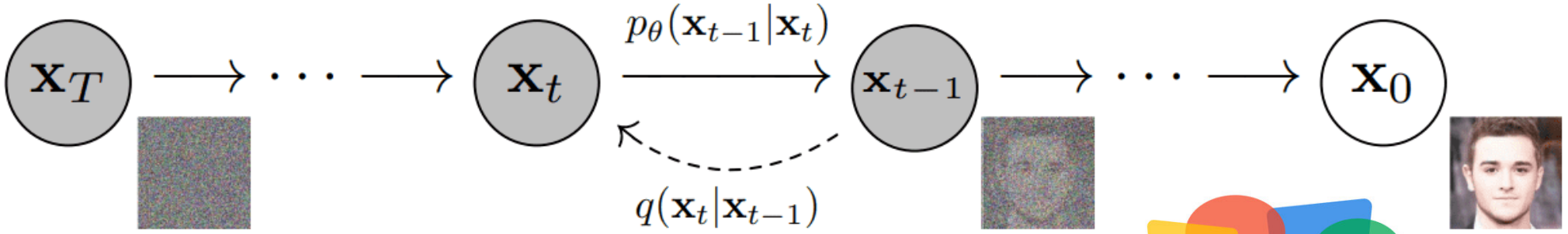


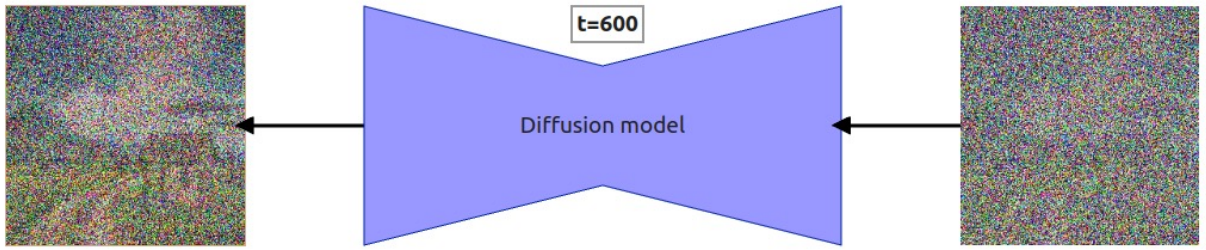
> *Learning in this framework involves estimating small perturbations to a diffusion process.* ***Estimating small perturbations is more tractable*** *than explicitly describing the full distribution with a single, non-analytically-normalizable, potential function. Furthermore, since a diffusion process exists for any smooth target distribution, this method can capture data distributions of arbitrary form.*

So why not just use GANs?

Something similar to the generator network

Reverse Diffusion Misconception



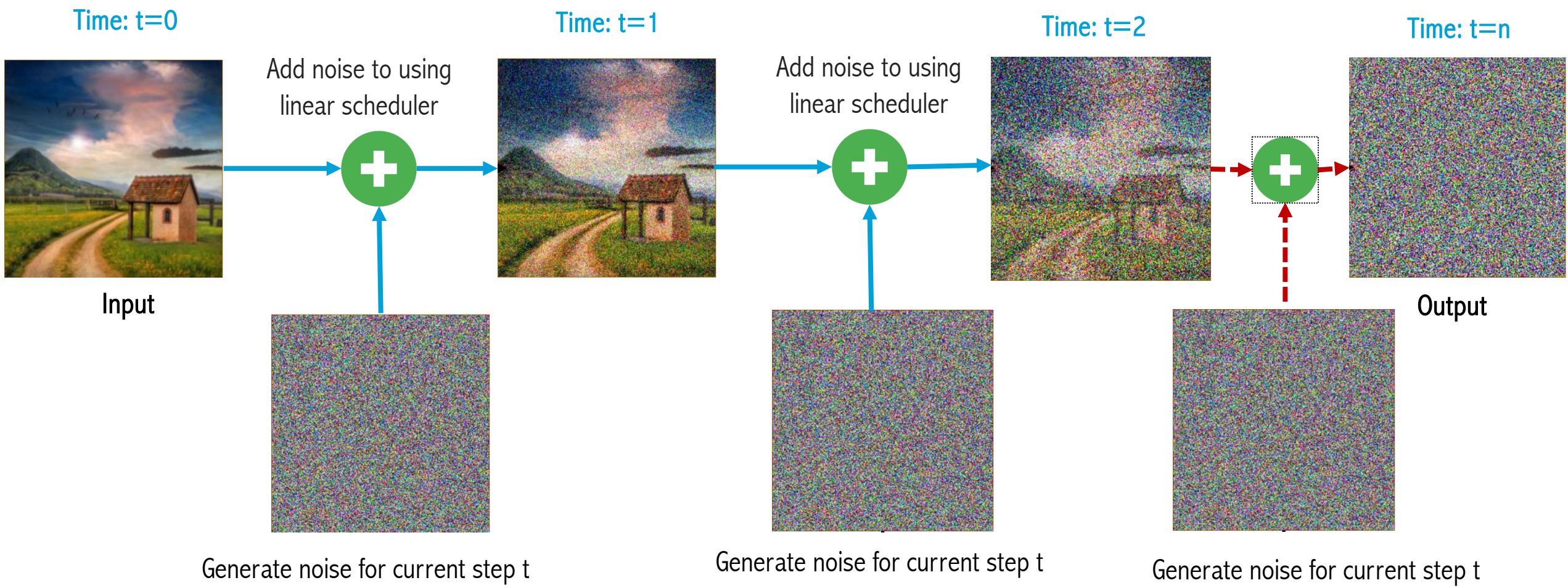


A neural network (called *diffusion model* for even more confusion) is removing noise from an input image or predicting the noise to be removed from an input (True or False?)

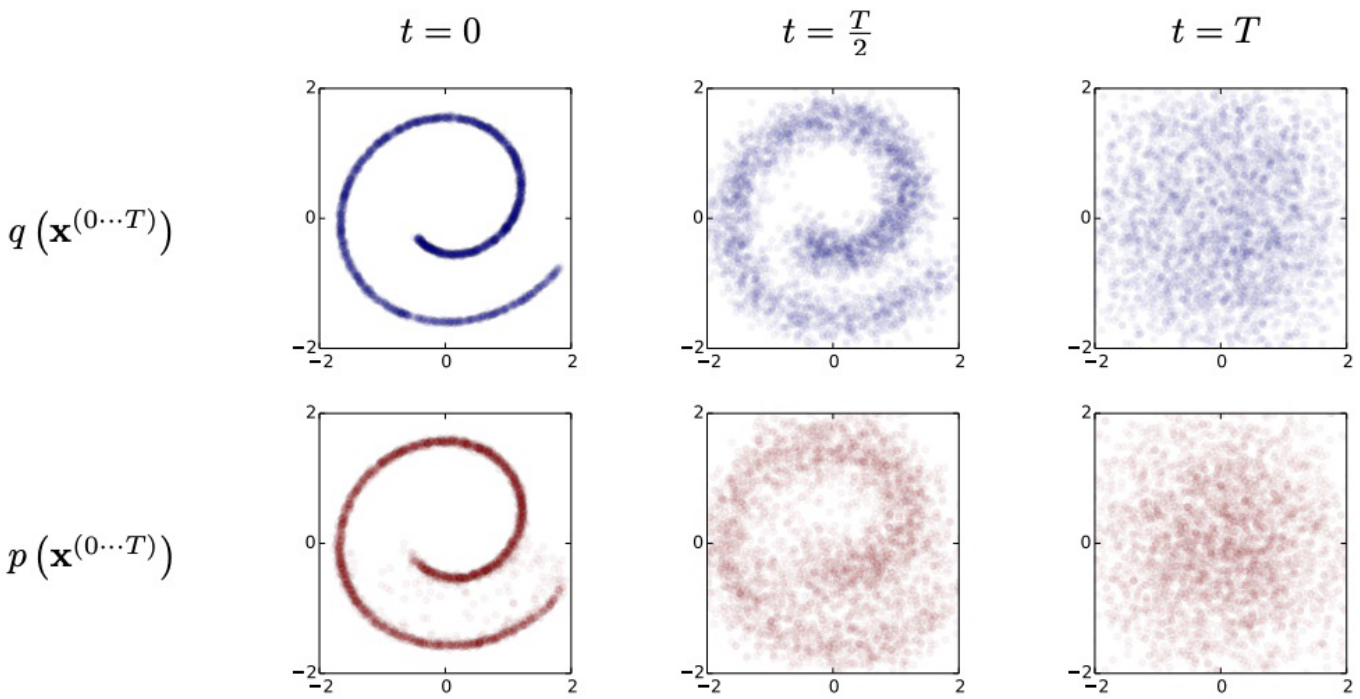
Diffusion model predicts the entire noise to be removed in a given timestep. This means that if we have timestep **t=600** then our Diffusion model tries to predict the entire noise on which removal we should get to **t=0**, not **t=599.**

Forward Diffusion Process: Review

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



Reverse Diffusion Process: Theory



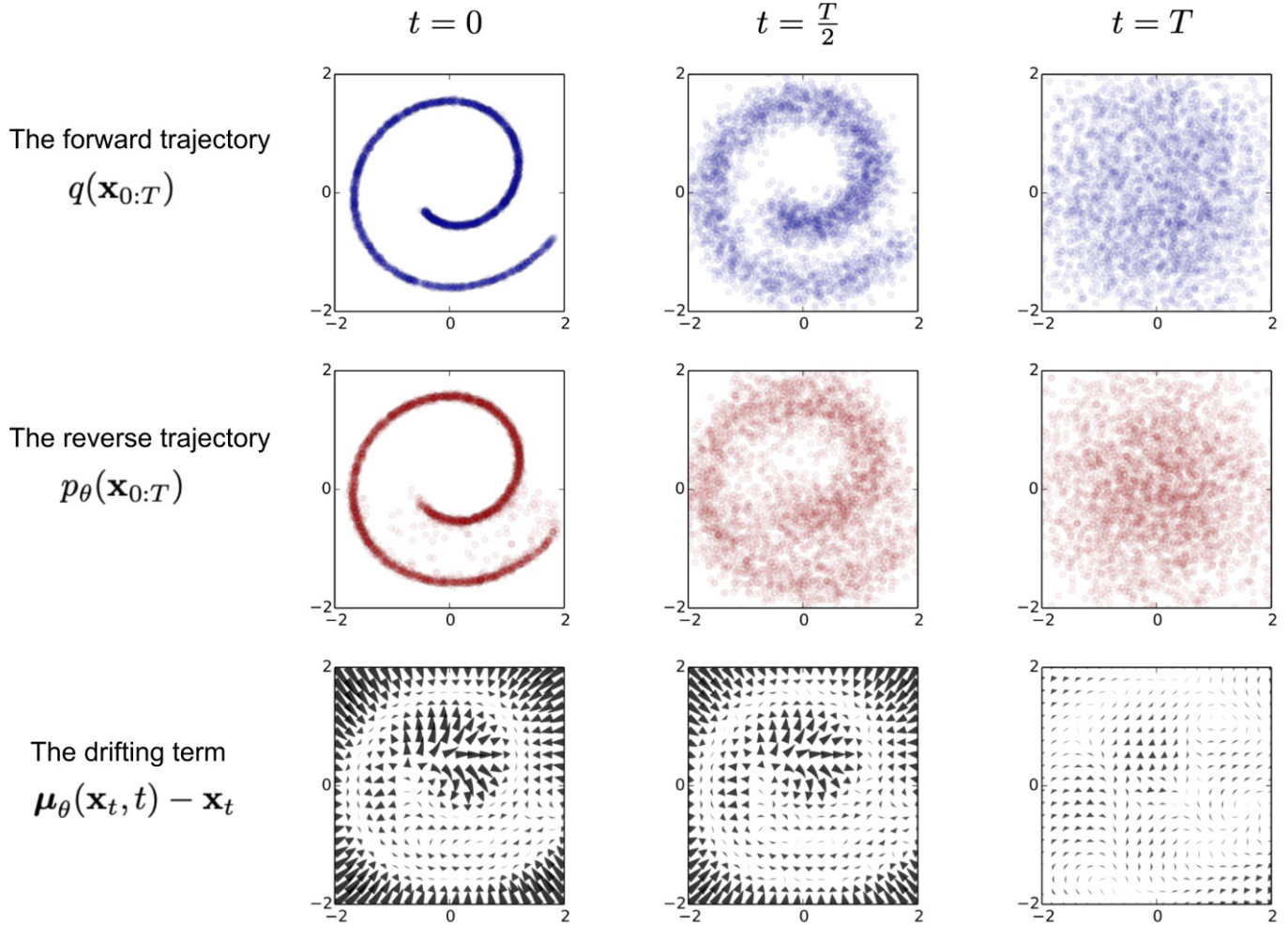
$$p\left(\mathbf{x}^{(T)}\right) = \pi\left(\mathbf{x}^{(T)}\right)$$

$$p\left(\mathbf{x}^{(0\cdots T)}\right) = p\left(\mathbf{x}^{(T)}\right) \prod_{t=1}^{T} p\left(\mathbf{x}^{(t-1)} | \mathbf{x}^{(t)}\right)$$

$$\mathbf{x}^{(T)} \sim \pi(\mathbf{x}^{(T)})$$
$$\mathbf{x}^{(T-1)} \sim p(\mathbf{x}^{(T-1)} \mid \mathbf{x}^{(T)})$$
$$\mathbf{x}^{(T-2)} \sim p(\mathbf{x}^{(T-2)} \mid \mathbf{x}^{(T-1)})$$
$$\dots$$
$$\mathbf{x}^{(0)} \sim p(\mathbf{x}^{(0)} \mid \mathbf{x}^{(1)})$$

Reverse Diffusion Process: Theory

$$p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T)\prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) \quad p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t))$$



It is noteworthy that the reverse conditional probability is tractable when conditioned on X0

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1}; \tilde{\boldsymbol{\mu}}(\mathbf{x}_t, \mathbf{x}_0), \tilde{\beta}_t \mathbf{I})$$

Reverse Diffusion Process: Theory

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

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

$$x_{t-1} = \mathcal{N}(x_{t-1}, \frac{1}{\sqrt{\alpha_t}}(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t)), \sqrt{\beta_t}\epsilon)$$

$$x_{t-1} = \frac{1}{\sqrt{a_t}}(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}}}\epsilon_\theta(x_t, t)) + \sqrt{\beta_t}\epsilon$$

ϵ is from $\mathcal{N}(0, 1)$

- $\epsilon_\theta(x_t, t)$ is our **model's output** (predicted noise)

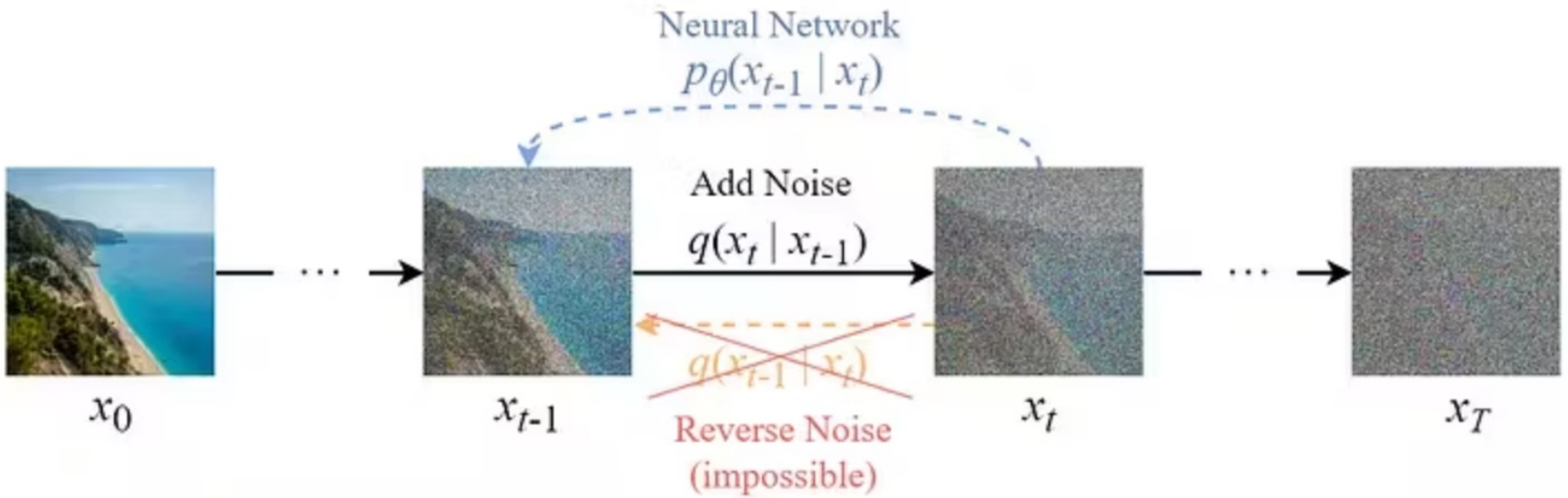
Reverse Diffusion Process: Theory

$$p\left(\mathbf{x}^{(T)}\right) = \pi\left(\mathbf{x}^{(T)}\right)$$

$$p\left(\mathbf{x}^{(0\cdots T)}\right) = p\left(\mathbf{x}^{(T)}\right)\prod_{t=1}^{T} p\left(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)}\right)$$

		Gaussian	*Binomial*
Well behaved (analytically tractable) distribution	$\pi\left(\mathbf{x}^{(T)}\right) =$	$\mathcal{N}\left(\mathbf{x}^{(T)}; \mathbf{0}, \mathbf{I}\right)$	$\mathcal{B}\left(\mathbf{x}^{(T)}; 0.5\right)$
Forward diffusion kernel	$q\left(\mathbf{x}^{(t)}\|\mathbf{x}^{(t-1)}\right) =$	$\mathcal{N}\left(\mathbf{x}^{(t)}; \mathbf{x}^{(t-1)}\sqrt{1-\beta_t}, \mathbf{I}\beta_t\right)$	$\mathcal{B}\left(\mathbf{x}^{(t)}; \mathbf{x}^{(t-1)}\left(1-\beta_t\right) + 0.5\beta_t\right)$
Reverse diffusion kernel	$p\left(\mathbf{x}^{(t-1)}\|\mathbf{x}^{(t)}\right) =$	$\mathcal{N}\left(\mathbf{x}^{(t-1)}; \mathbf{f}_\mu\left(\mathbf{x}^{(t)}, t\right), \mathbf{f}_\Sigma\left(\mathbf{x}^{(t)}, t\right)\right)$	$\mathcal{B}\left(\mathbf{x}^{(t-1)}; \mathbf{f}_b\left(\mathbf{x}^{(t)}, t\right)\right)$
Training targets		$\mathbf{f}_\mu\left(\mathbf{x}^{(t)}, t\right), \mathbf{f}_\Sigma\left(\mathbf{x}^{(t)}, t\right), \beta_{1\cdots T}$	$\mathbf{f}_b\left(\mathbf{x}^{(t)}, t\right)$
Forward distribution	$q\left(\mathbf{x}^{(0\cdots T)}\right) =$	$q\left(\mathbf{x}^{(0)}\right)\prod_{t=1}^{T} q\left(\mathbf{x}^{(t)}\|\mathbf{x}^{(t-1)}\right)$	
Reverse distribution	$p\left(\mathbf{x}^{(0\cdots T)}\right) =$	$\pi\left(\mathbf{x}^{(T)}\right)\prod_{t=1}^{T} p\left(\mathbf{x}^{(t-1)}\|\mathbf{x}^{(t)}\right)$	
Log likelihood	$L =$	$\int d\mathbf{x}^{(0)} q\left(\mathbf{x}^{(0)}\right)\log p\left(\mathbf{x}^{(0)}\right)$	
Lower bound on log likelihood	$K =$	$-\sum_{t=2}^{T}\mathbb{E}_{q\left(\mathbf{x}^{(0)},\mathbf{x}^{(t)}\right)}\left[D_{KL}\left(q\left(\mathbf{x}^{(t-1)}\|\mathbf{x}^{(t)},\mathbf{x}^{(0)}\right)\|p\left(\mathbf{x}^{(t-1)}\|\mathbf{x}^{(t)}\right)\right)\right] + H_q\left(\mathbf{X}^{(T)}\|\mathbf{X}^{(0)}\right) - H_q\left(\mathbf{X}^{(1)}\|\mathbf{X}^{(0)}\right) - H_p\left(\mathbf{X}^{(T)}\right)$	
Perturbed reverse diffusion kernel	$\tilde{p}\left(\mathbf{x}^{(t-1)}\|\mathbf{x}^{(t)}\right) =$	$\mathcal{N}\left(x^{(t-1)}; \mathbf{f}_\mu\left(\mathbf{x}^{(t)}, t\right) + \mathbf{f}_\Sigma\left(\mathbf{x}^{(t)}, t\right)\frac{\partial \log r\left(\mathbf{x}^{(t-1)'}\right)}{\partial \mathbf{x}^{(t-1)'}}\Big\|_{\mathbf{x}^{(t-1)'}=f_\mu\left(\mathbf{x}^{(t)},t\right)}, \mathbf{f}_\Sigma\left(\mathbf{x}^{(t)}, t\right)\right)$	$\mathcal{B}\left(x_i^{(t-1)}; \frac{c_i^{t-1}d_i^{t-1}}{x_i^{t-1}d_i^{t-1}+(1-c_i^{t-1})(1-d_i^{t-1})}\right)$

Reverse Diffusion Process: Theory



Target Distribution $q(x_{t-1}|x_t) = \mathcal{N}(x_{t-1};\ \tilde{\mu}_t(x_t, x_0),\ \tilde{\beta}_t I)$

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

Learnable parameters
(Neural Network)

Reverse Diffusion Process: Implementation

```python
def reverse_process(self, xt, t):
    """
    :param t: Number of diffusion steps
    """

    assert t > 0, 't should be greater than 0'
    assert self.T <= self.T, f't should be lower or equal than {self.T}'

    t = t - 1 # Because we start indexing at 0

    mu, std = self.model(xt, t)
    epsilon = torch.randn_like(xt)

    return mu + epsilon * std # data ~ N(mu, std)
```

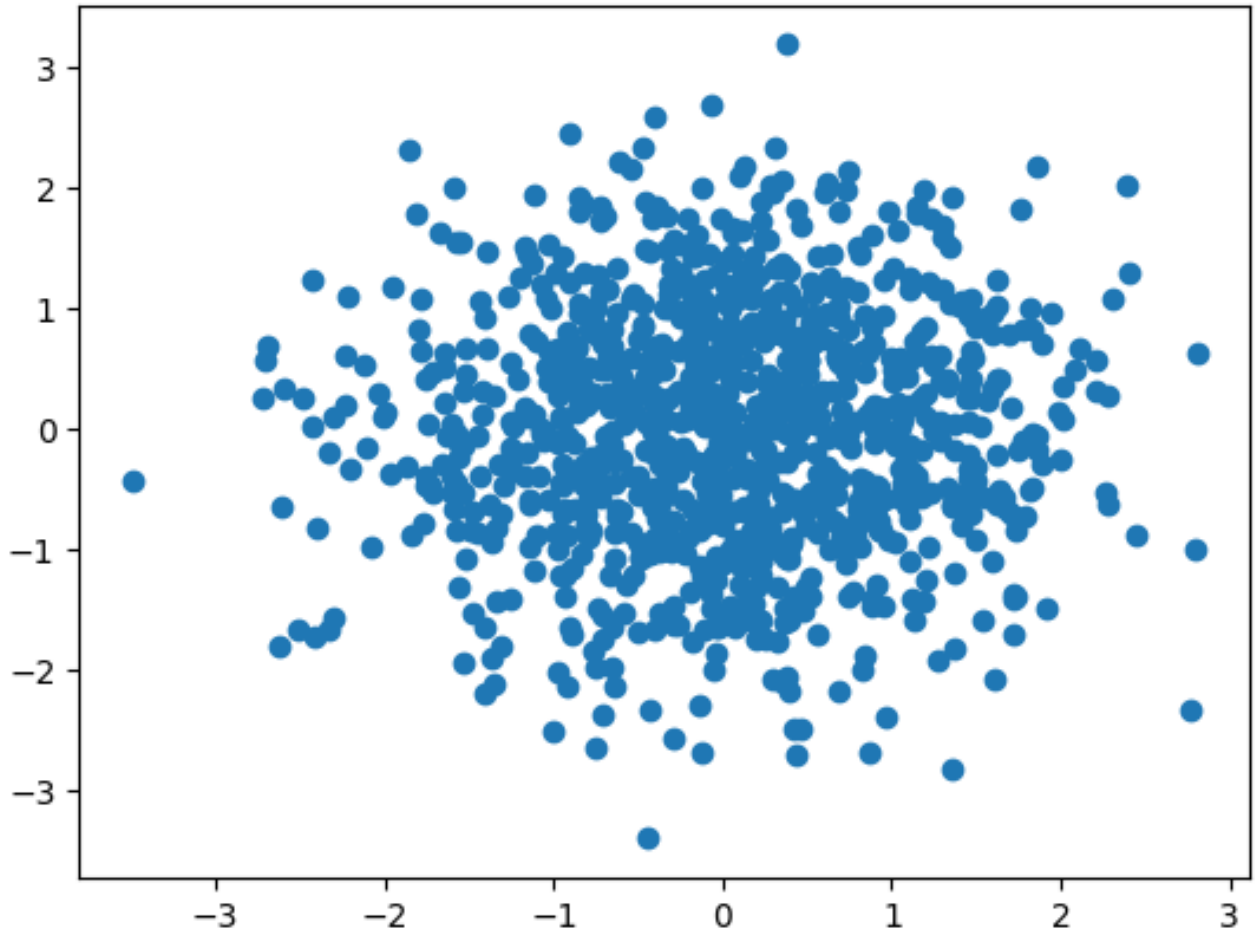
```python
def sample(self, batch_size):

    noise = torch.randn((batch_size, self.dim))
    x = noise

    samples = [x]
    for t in range(self.T, 0, -1):

        if not (t == 1):
            x = self.reverse_process(x, t)

        samples.append(x)

    return samples[::-1]
```

```python
class MLP(nn.Module):

    def __init__(self, N=40, data_dim=2, hidden_dim=64):
        super(MLP, self).__init__()

        self.network_head = nn.Sequential(nn.Linear(data_dim, hidden_dim),
                                          nn.ReLU(),
                                          nn.Linear(hidden_dim, hidden_dim),
                                          nn.ReLU(),)

        self.network_tail = nn.ModuleList([nn.Sequential(nn.Linear(hidden_dim, hidden_dim),
                                                         nn.ReLU(),
                                                         nn.Linear(hidden_dim, data_dim * 2),) for t in range(N)])

    def forward(self, x, t):

        h = self.network_head(x) # [batch_size, hidden_dim]
        tmp = self.network_tail[t](h) # [batch_size, data_dim * 2]
        mu, h = torch.chunk(tmp, 2, dim=1)
        var = torch.exp(h)
        std = torch.sqrt(var)

        return mu, std
```

Reverse Diffusion Process: Implementation

```
x0 = sample_batch(3_000)
mlp_model = torch.load('model_paper1')
model = DiffusionModel(40, mlp_model)
xT = model.forward_process(x0, 20)
```

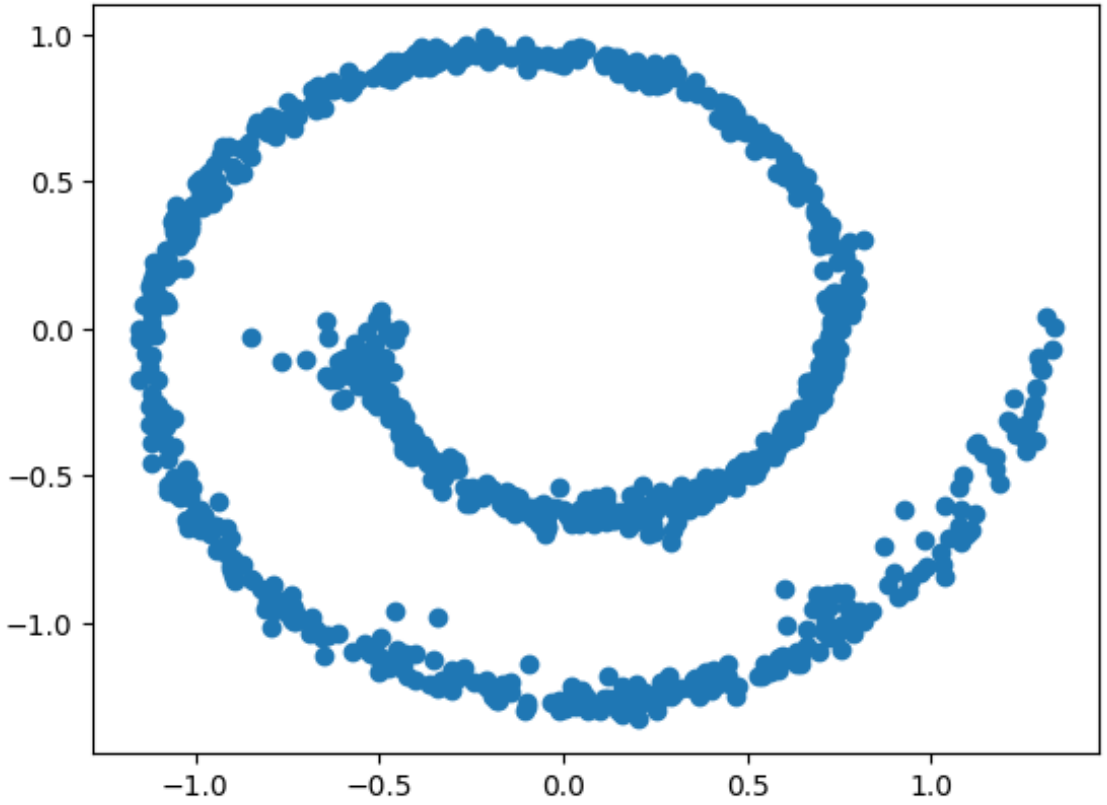
```
t = 40
plt.scatter(samples[t][:, 0].data.numpy(),
            samples[t][:, 1].data.numpy())
```



Reverse Diffusion Process: Implementation

```
x0 = sample_batch(3_000)
mlp_model = torch.load('model_paper1')
model = DiffusionModel(40, mlp_model)
xT = model.forward_process(x0, 20)
```

```
t = 5
plt.scatter(samples[t][:, 0].data.numpy()
, samples[t][:, 1].data.numpy())
```




QUIZ TIME

Diffusion Model: Training Theory

$$\mathbb{E}_{\mathbf{x}^{(0)}\sim\mathcal{D}_{train}} \log p(\mathbf{x}^{(0)})$$
$$= \mathbb{E}_{\mathbf{x}^{(0)}\sim q(\mathbf{x}^{(0)})} \log p(\mathbf{x}^{(0)})$$
$$= \int q(\mathbf{x}^{(0)}) \log p(\mathbf{x}^{(0)}) d\mathbf{x}^{(0)}$$

Training amounts to maximizing the model log

$$L = \int d\mathbf{x}^{(0)} q\left(\mathbf{x}^{(0)}\right) \log p\left(\mathbf{x}^{(0)}\right)$$

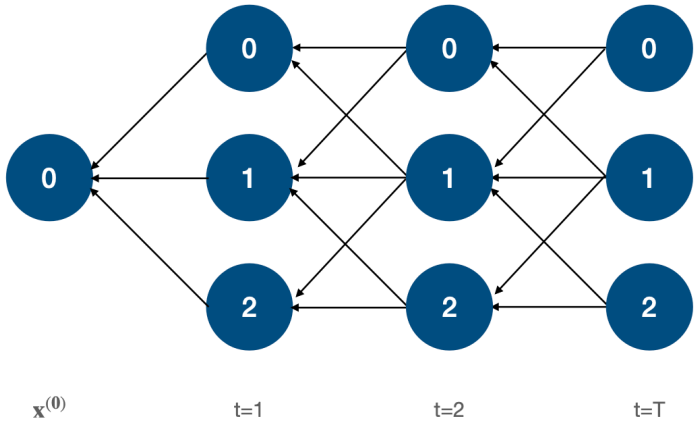
The probability the generative model assigns to the data:

$$p\left(\mathbf{x}^{(0)}\right) = \int d\mathbf{x}^{(1\cdots T)} p\left(\mathbf{x}^{(0\cdots T)}\right).$$

this integral is intractable

Taking a cue from annealed importance sampling and the Jarzynski equality

$$p\left(\mathbf{x}^{(0)}\right) = \int d\mathbf{x}^{(1\cdots T)} p\left(\mathbf{x}^{(0\cdots T)}\right) \frac{q\left(\mathbf{x}^{(1\cdots T)}|\mathbf{x}^{(0)}\right)}{q\left(\mathbf{x}^{(1\cdots T)}|\mathbf{x}^{(0)}\right)} \quad (7)$$
$$= \int d\mathbf{x}^{(1\cdots T)} q\left(\mathbf{x}^{(1\cdots T)}|\mathbf{x}^{(0)}\right) \frac{p\left(\mathbf{x}^{(0\cdots T)}\right)}{q\left(\mathbf{x}^{(1\cdots T)}|\mathbf{x}^{(0)}\right)} \quad (8)$$
$$= \int d\mathbf{x}^{(1\cdots T)} q\left(\mathbf{x}^{(1\cdots T)}|\mathbf{x}^{(0)}\right) \cdot p\left(\mathbf{x}^{(T)}\right) \prod_{t=1}^{T} \frac{p\left(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)}\right)}{q\left(\mathbf{x}^{(t)}|\mathbf{x}^{(t-1)}\right)}. \quad (9)$$



Diffusion Model: Training Theory

Training amounts to maximizing the model log

$$L = \int d\mathbf{x}^{(0)} q\left(\mathbf{x}^{(0)}\right) \log p\left(\mathbf{x}^{(0)}\right)$$

$$L = \int d\mathbf{x}^{(0)} q\left(\mathbf{x}^{(0)}\right) \log p\left(\mathbf{x}^{(0)}\right)$$
$$= \int d\mathbf{x}^{(0)} q\left(\mathbf{x}^{(0)}\right) \cdot$$
$$\log\left[\begin{array}{c}\int d\mathbf{x}^{(1\cdots T)} q\left(\mathbf{x}^{(1\cdots T)}|\mathbf{x}^{(0)}\right) \cdot \\ p\left(\mathbf{x}^{(T)}\right) \prod_{t=1}^{T} \frac{p\left(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)}\right)}{q\left(\mathbf{x}^{(t)}|\mathbf{x}^{(t-1)}\right)}\end{array}\right]$$

Jensen's inequality

$\log(\mathbb{E}[\mathbb{X}]) \geq (\mathbb{E}[\log \mathbb{X}])$

$$L \geq \int d\mathbf{x}^{(0\cdots T)} q\left(\mathbf{x}^{(0\cdots T)}\right) \cdot$$
$$\log\left[p\left(\mathbf{x}^{(T)}\right) \prod_{t=1}^{T} \frac{p\left(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)}\right)}{q\left(\mathbf{x}^{(t)}|\mathbf{x}^{(t-1)}\right)}\right]$$

$$L \geq K \qquad (13)$$
$$K = -\sum_{t=2}^{T} \int d\mathbf{x}^{(0)} d\mathbf{x}^{(t)} q\left(\mathbf{x}^{(0)}, \mathbf{x}^{(t)}\right) \cdot$$
$$D_{KL}\left(q\left(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)}, \mathbf{x}^{(0)}\right) \| p\left(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)}\right)\right)$$
$$+ H_q\left(\mathbf{X}^{(T)}|\mathbf{X}^{(0)}\right) - H_q\left(\mathbf{X}^{(1)}|\mathbf{X}^{(0)}\right) - H_p\left(\mathbf{X}^{(T)}\right) \cdot \qquad (14)$$

$$q(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)}, \mathbf{x}^{(0)}) = \frac{q(\mathbf{x}^{(t)}|\mathbf{x}^{(t-1)}, \mathbf{x}^{(0)}) q(\mathbf{x}^{(t-1)}|\mathbf{x}^{(0)})}{q(\mathbf{x}^{(t)}|\mathbf{x}^{(0)})}$$
$$= \frac{1}{\sqrt{\frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t}\sqrt{(2\pi)}} \exp\left[-\frac{1}{2}\left(\frac{\mathbf{x}^{(t-1)} - \left(\frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}^{(0)} + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}^{(t)}\right)}{\sqrt{\frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t}}\right)^2\right]$$

$$D_{KL}(p||q) = \log\frac{\sigma_q}{\sigma_p} + \frac{\sigma_p^2 + (\mu_p - \mu_q)^2}{2\sigma_q^2} - \frac{1}{2}$$

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)}$$

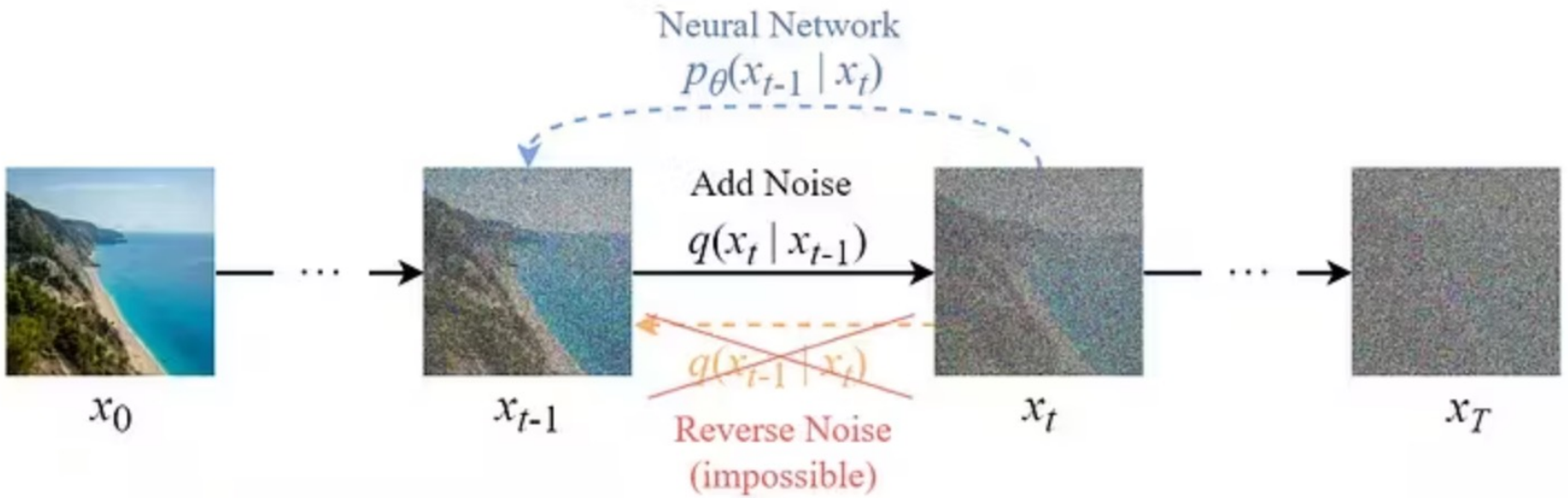
$$\frac{1}{\sigma\sqrt{(2\pi)}} \exp^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$

$$H(x) = \frac{1}{2} + \frac{1}{2}\log(2\pi\sigma^2)$$

$$H(\mathbf{x}^{(T)}) = \frac{1}{2} + \frac{1}{2}\log(2\pi)$$

$$H(\mathbf{x}^{(1)}|\mathbf{x}^{(0)}) = \frac{1}{2} + \frac{1}{2}\log(2\pi\beta_0)$$

Reverse Diffusion Process: Theory



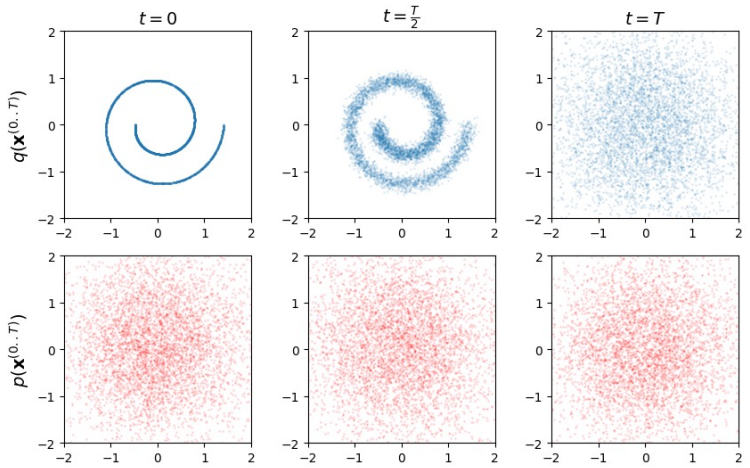
Target Distribution $q(x_{t-1}|x_t) = \mathcal{N}(x_{t-1};\ \tilde{\mu}_t(x_t, x_0),\ \tilde{\beta}_t I)$

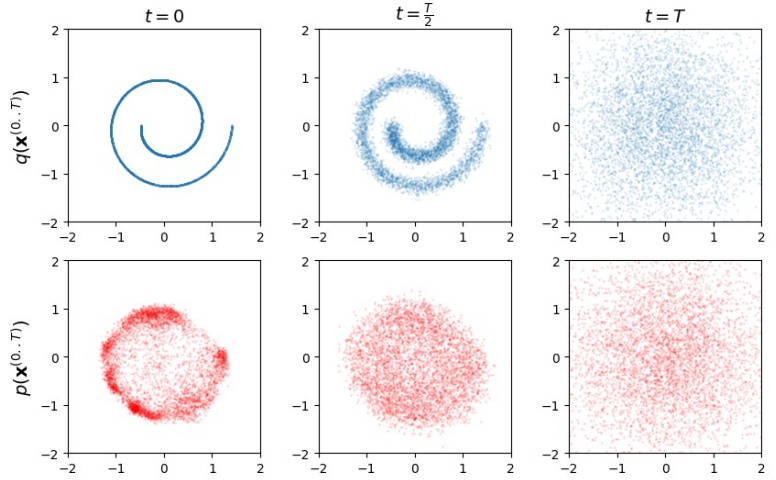
Approximated Distribution $p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1};\ \mu_\theta(x_t, t),\ \Sigma_\theta(x_t, t))$

Learnable parameters
(Neural Network)

Diffusion Model: Training Implementation

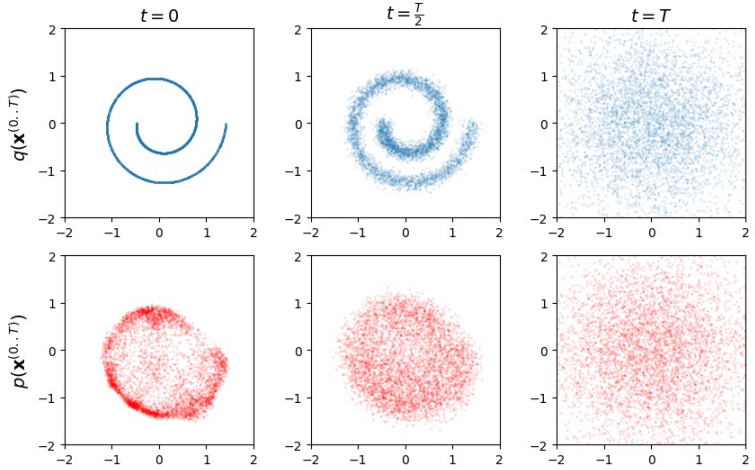
Epoch 1

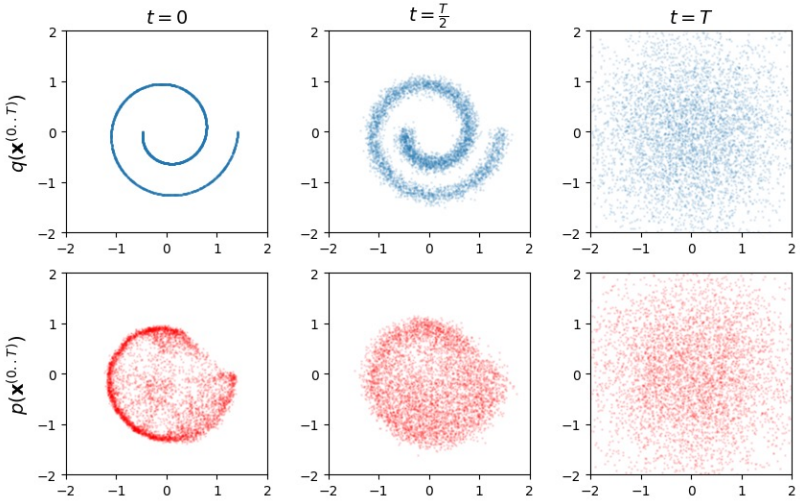




Epoch 5000

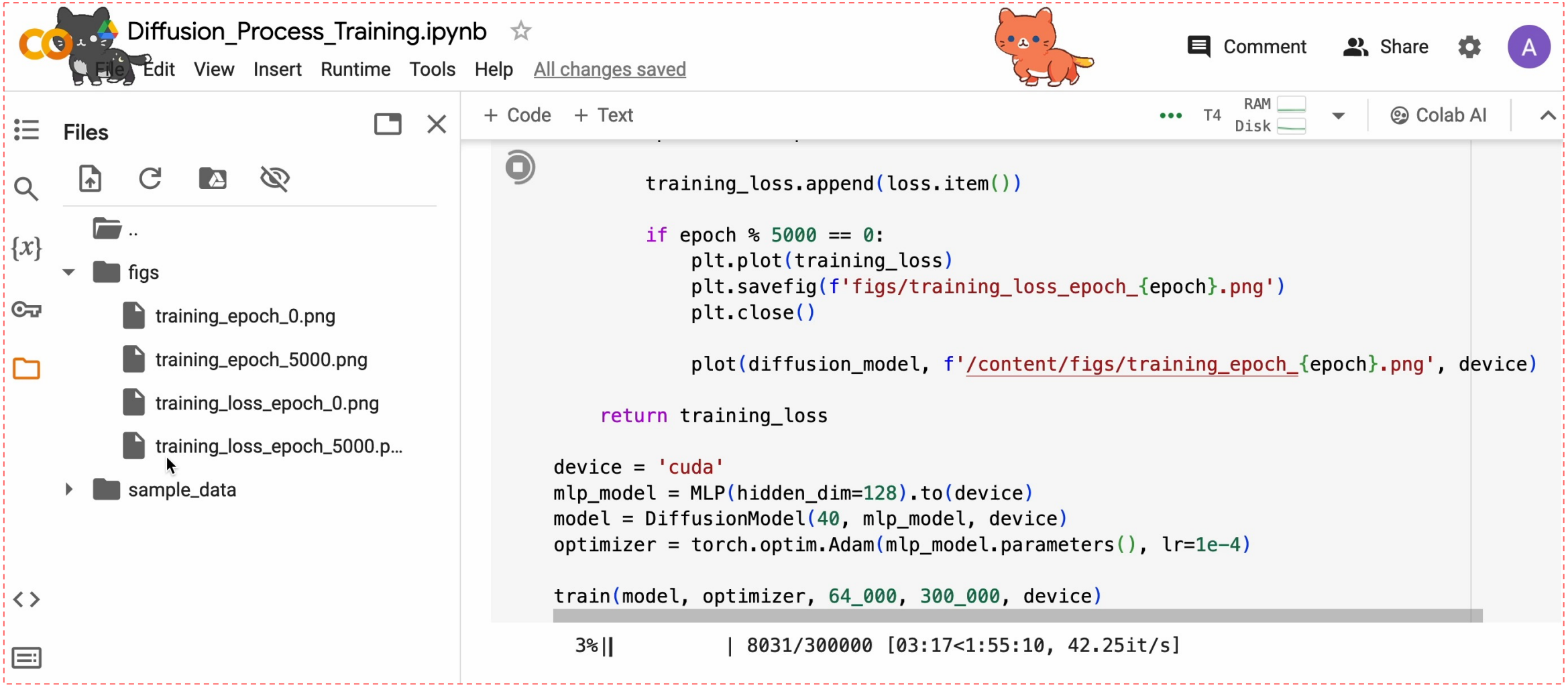
Epoch 15000





Epoch 30000

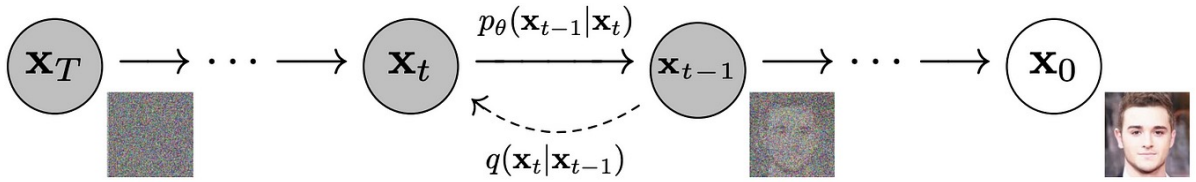
Diffusion Model: Training Implementation

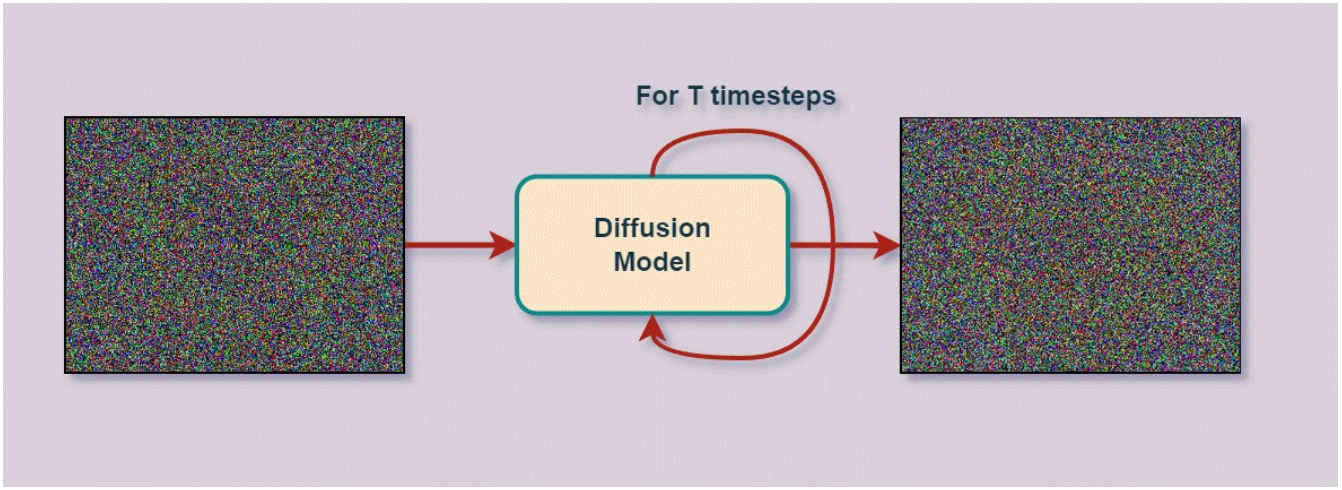


Outline

- Objective
- Application of Diffusion Models
- Why Do We Need Diffusion Model?
- Diffusion Model Detail Explanation and Implementation
- Summary

Summary





1. How Does Diffusion Model Work
2. How Does a Forward Diffusion Process Work
3. How Does a Reverse Diffusion Process
4. Be able to Implement a Diffusion Model Using Pytorch


Thank you