Implementation of Diffusion Model

Hiroaki Kubo · Follow

4 min read · Dec 14, 2024

I wrote this article because I implemented diffusion model from scratch using the loss derived in **previous article**.

In this case, I used the swiss roll dataset and the algorithm basically followed **Denoising Diffusion Probabilistic Models**. The code which I implemented is **here**.

· · ·

Overview

When we implement the Diffusion model, the following parts need to be implemented. I will explain each of these steps.

- **Forward process**
- **Neural network for training**
- **Training**
- **Sampling**

. . .

Forward process

The forward process of adding Gaussian noise to the input data is expressed by the following equation. Diffusion model gradually adds Gaussian noise to the data according to a variance schedule $\beta 1,\ldots,\beta T$.

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

A notable property of the forward process is that it admits sampling xt, at an arbitrary timestep t in closed form: using the notation αt:=1−βt and αt¯ =∏{s=1}{t}αs, we have

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

Thus, we can get xt by using a **reparameterization trick**.

$$x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon \quad (\epsilon \sim \mathcal{N}(0, I))$$

The code to get $\bar{\alpha t}$ is here.

```
def calculate_parameters(diffusion_steps, min_beta, max_beta):
    step = (max_beta - min_beta) / diffusion_steps
    beta_ts = torch.arange(min_beta, max_beta + step, step)

    alpha_ts = 1 - beta_ts
    bar_alpha_ts = torch.cumprod(alpha_ts, dim=0)

    return beta_ts, alpha_ts, bar_alpha_ts
```

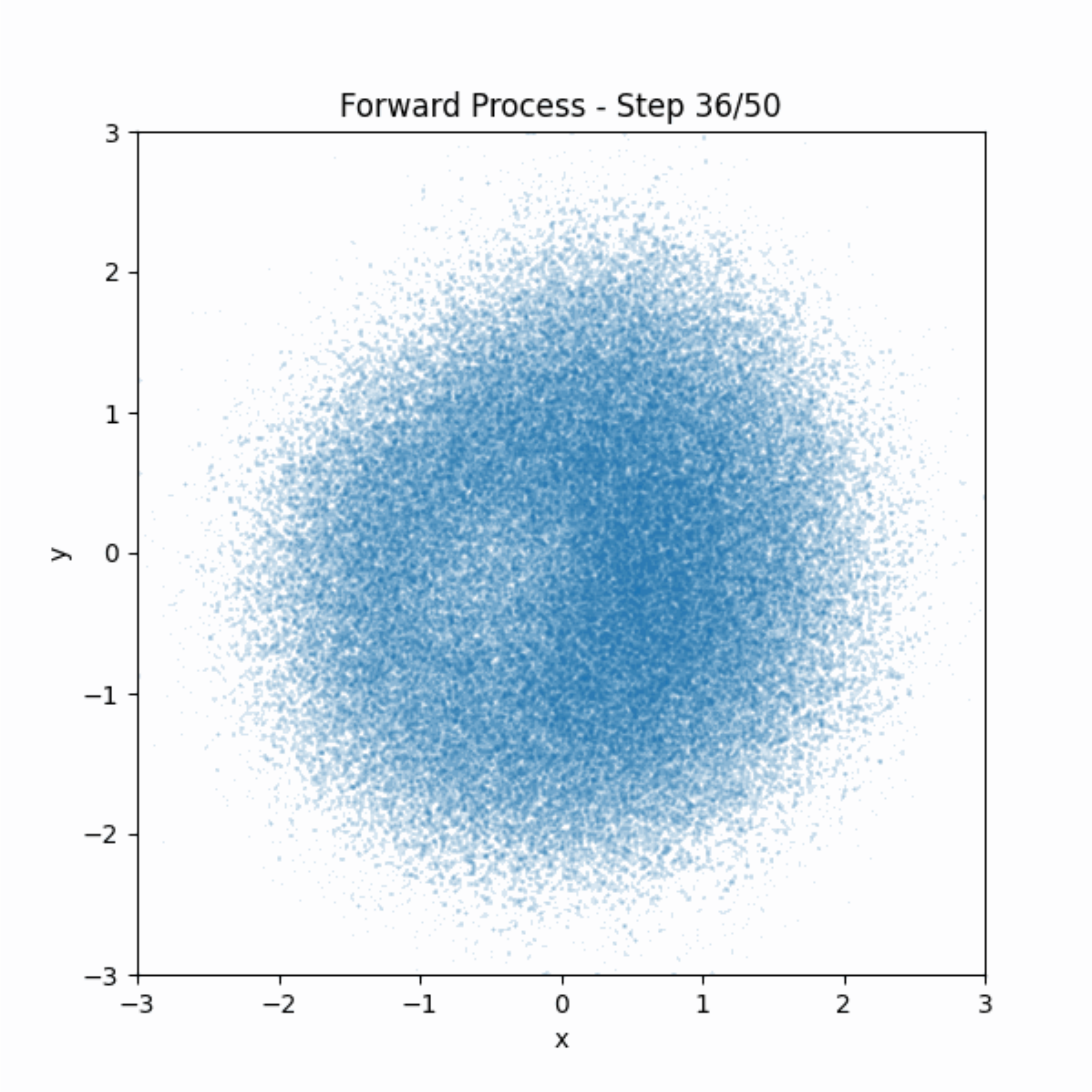
The code to retrieve data and ϵ at any given time in the forward process is as follows.

```
def calculate_data_at_certain_time(x_0, bar_alpha_ts, t):
    eps = torch.randn(size=x_0.shape)
    noised_x_t = (
        torch.sqrt(bar_alpha_ts[t]) * x_0 + torch.sqrt(1 - bar_alpha_ts[t]) 
    )

    return noised_x_t, eps
```

You can try a forward process with swiss roll by running **this code**. We set the forward process variance constants increasing linearly from $\beta 1$

$=0.0004$ to $\beta T=0.02$.



. . .

Neural network for training

Original paper uses **U-Net** backbone, but I used simple neural network for training this time because it is enough in this data. It has 4 hidden layers and use ReLU as an activation function.
If you want to check the architecture of model, you can run **this code**.

. . .

Training

In reverse process, we calculate x_{t-1} from x_t and timestep t by normal distribution.

$$x_{t-1} \sim \mathcal{N}(\mu_\theta(x_t, t), \Sigma_t)$$

The variance is fixed, so we need to predict only $\mu_\theta(x_t,t)$. $\mu_\theta(x_t,t)$ can be rewritten as follows by simplifying equations. If you want to know the details of that, please check **my previous article**.

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

Therefore, we train ϵ during training, loss function is as follows. If you want to know the details of that, please check **my previous article**.

$$L_{simple} := E_{t,x_0,\epsilon}\left[|\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t)|^2\right]$$

I followed below training algorithms of original paper.

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \mathrm{Uniform}(\{1, \ldots, T\})$
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

Parameters which I used during training is as follows.

```
- Optimizer -> Adam
- Batch size -> 128
- Epochs -> 30
- Diffusion timesteps -> 50
- Minimum beta -> 1e-4
- Maximum beta -> 0.02
```

You can train the diffusion model by running **this code**.

. . .

Sampling

To sample $x_{t-1} \sim p_\theta(x_{t-1}|x_t)$ is to compute below eqaution. It is used reparameterization trick.

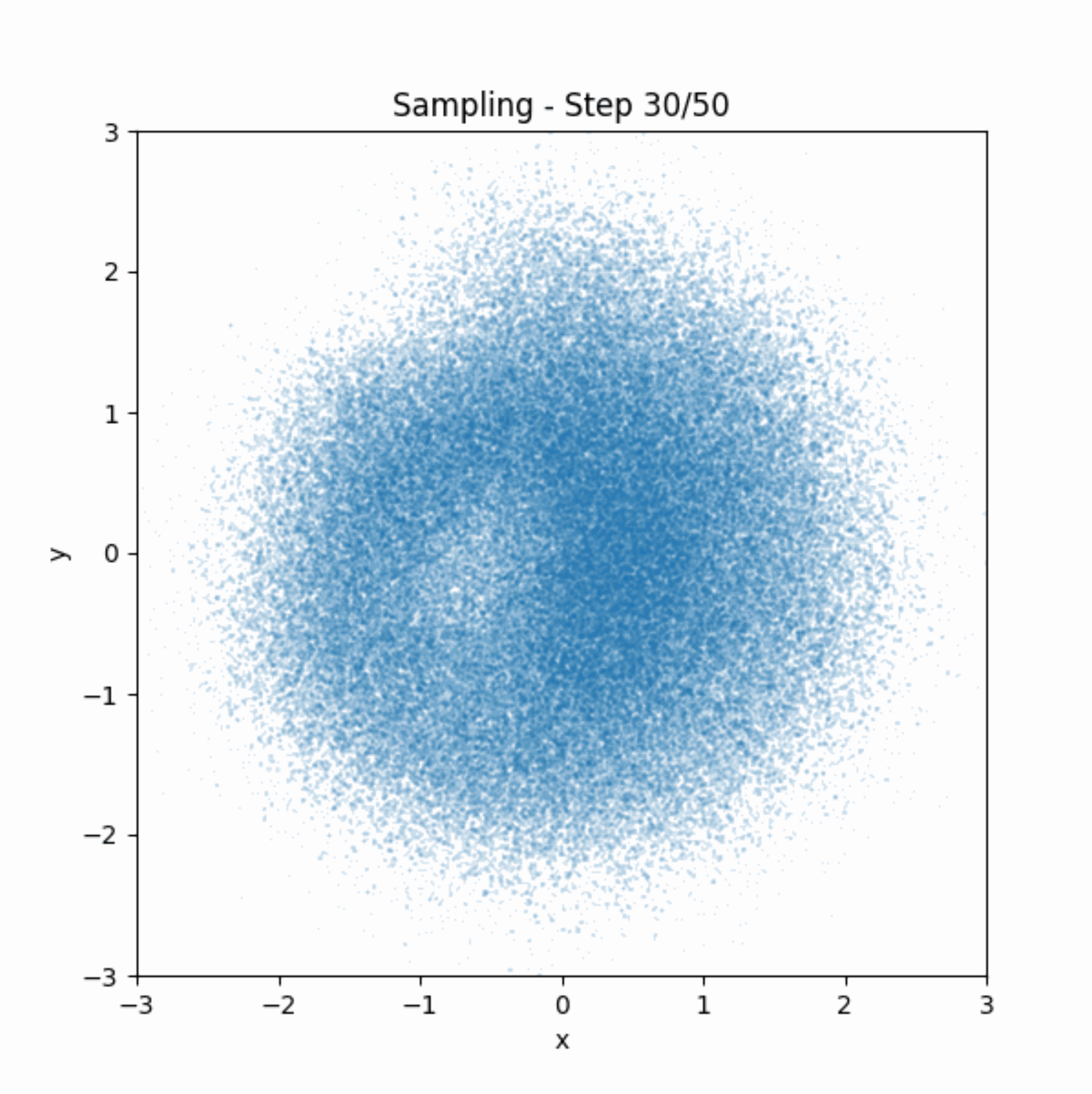
$$x_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t)\right) + \sigma_t z, \quad (z \sim \mathcal{N}(0, I))$$

I followed below sampling algorithms of original paper. We initialize x_T with a random value, and with x_t and t as inputs, the neural network output ϵ, so we use that value to calculate x_{t-1}. This is repeated until x_0 is calculated.

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 $\mathbf{x}_0$

You can try sampling by running **this code**.

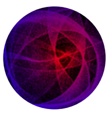

Sampling - Step 30/50
y
x
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

. . .

References

- Denoising Diffusion Probabilistic Models
- Deep Unsupervised Learning using Nonequilibrium Thermodynamics
- The Reparameterization Trick
- toy-diffusion
- Understanding the Diffusion Model

AI Python Diffusion Models Generative Ai Tools Neural Networks



Written by Hiroaki Kubo

Follow

140 Followers · 4 Following

Computer Vision Engineer. Computer Vision / Machine Learning / Computer Graphics, https://www.linkedin.com/in/hiroaki-kubo-2819951ba/