Denoising Diffusion Implicit Models

Learning with Keras Code Example

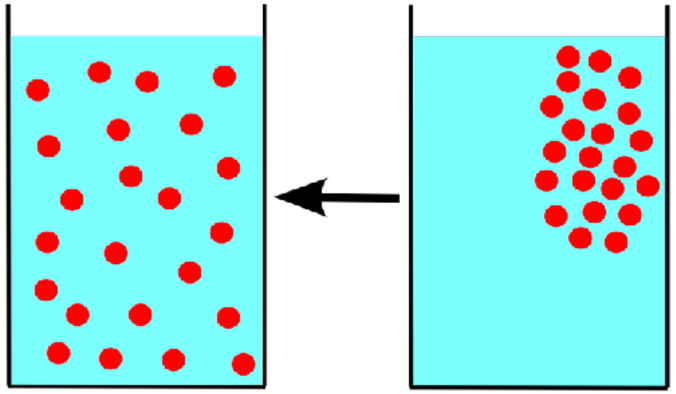
SCPARK

Diffusion Model

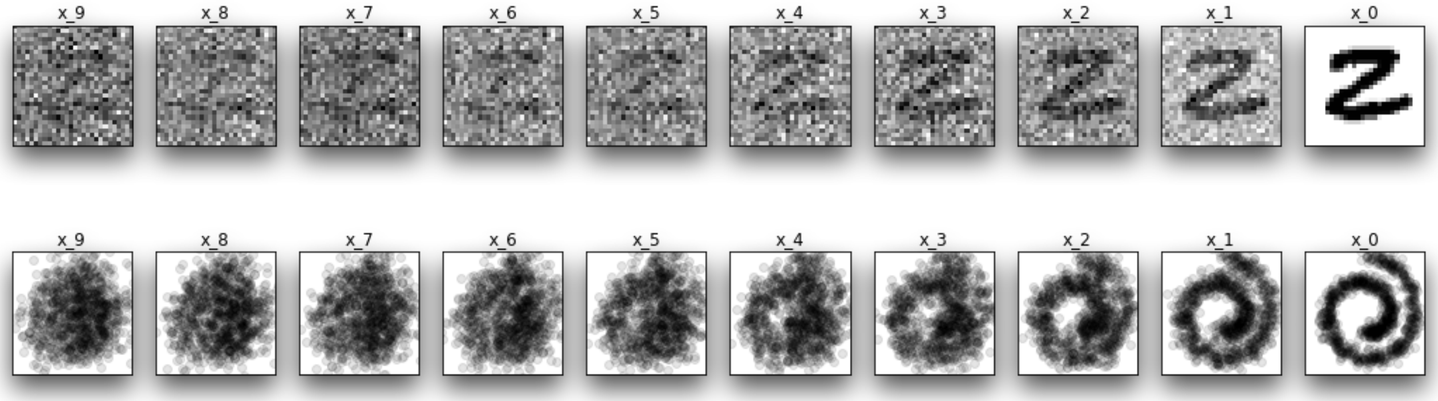
Diffusion Model

- Sohl-Dickstein, Jascha의 논문 Deep unsupervised learning using nonequilibrium thermodynamics에서 제안

- 물리에서의 diffusion 현상에 영감을 받아 모델을 설계



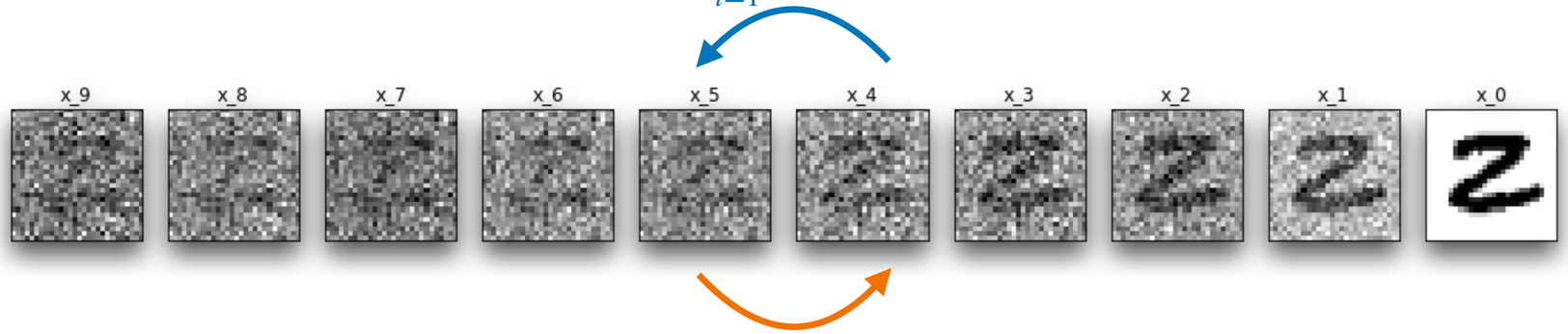
Diffusion
https://en.wikipedia.org/wiki/Diffusion
(Flipped)



Sohl-Dickstein, Jascha, et al. "Deep unsupervised learning using nonequilibrium thermodynamics." *International Conference on Machine Learning*. PMLR, 2015.

Diffusion Model

Forward Process $q\left(\mathbf{x}_{1:T} \mid \mathbf{x}_0\right) := \prod_{t=1}^{T} q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}\right), q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{x}_0\right) := \mathcal{N}\left(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \mathbf{I}\right)$



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

where $\tilde{\boldsymbol{\mu}}_t\left(\mathbf{x}_t, \mathbf{x}_0\right) := \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0 + \frac{\sqrt{\alpha_t}\left(1-\bar{\alpha}_{t-1}\right)}{1-\bar{\alpha}_t}\mathbf{x}_t$ and $\tilde{\beta}_t := \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$

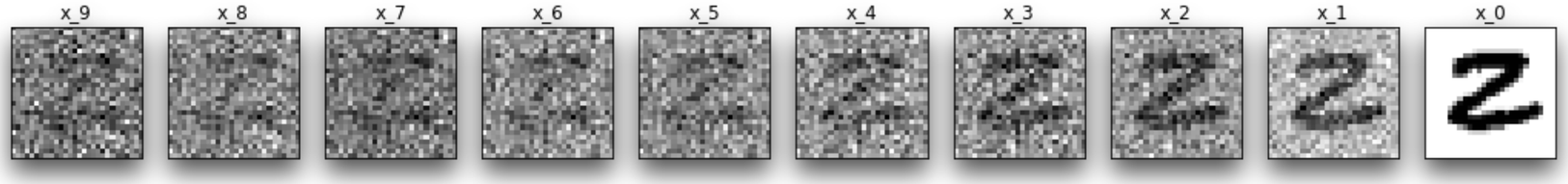
Backward Process (Neural Networks) $p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right) := \mathcal{N}\left(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta\left(\mathbf{x}_t, t\right), \boldsymbol{\Sigma}_\theta\left(\mathbf{x}_t, t\right)\right)$

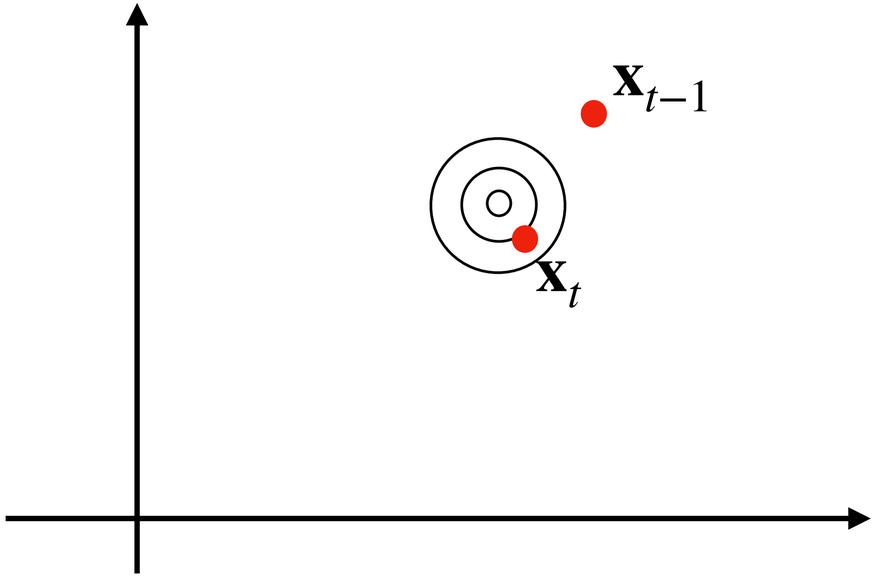
Loss Function $D_{\mathrm{KL}}\left(q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0\right) \| p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right)\right)$

Sohl-Dickstein, Jascha, et al. "Deep unsupervised learning using nonequilibrium thermodynamics." *International Conference on Machine Learning*. PMLR, 2015.

Diffusion Model - Forward Process

Forward Process $q\left(\mathbf{x}_{1:T} \mid \mathbf{x}_0\right) := \prod_{t=1}^{T} q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}\right), q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{x}_0\right) := \mathcal{N}\left(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \mathbf{I}\right)$





Distribution of $\mathbf{x}_t$ at an arbitrary timestep t in closed form

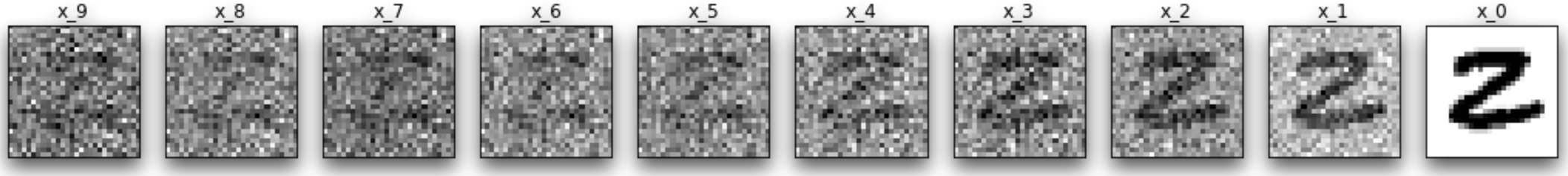
$q\left(\mathbf{x}_t \mid \mathbf{x}_0\right) = \mathcal{N}\left(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, \left(1-\bar{\alpha}_t\right)\mathbf{I}\right)$, where $\alpha_t := 1 - \beta_t$ and $\bar{\alpha}_t := \prod_{s=1}^{t} \alpha_s$

식 유도는Lil'Log 참고
https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

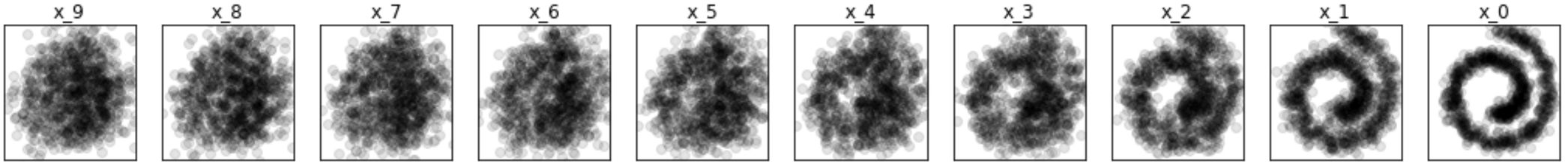
Sohl-Dickstein, Jascha, et al. "Deep unsupervised learning using nonequilibrium thermodynamics." *International Conference on Machine Learning*. PMLR, 2015.

Diffusion Model - Forward Process

MNIST single data, $\beta = 0.2$, $T = 10$



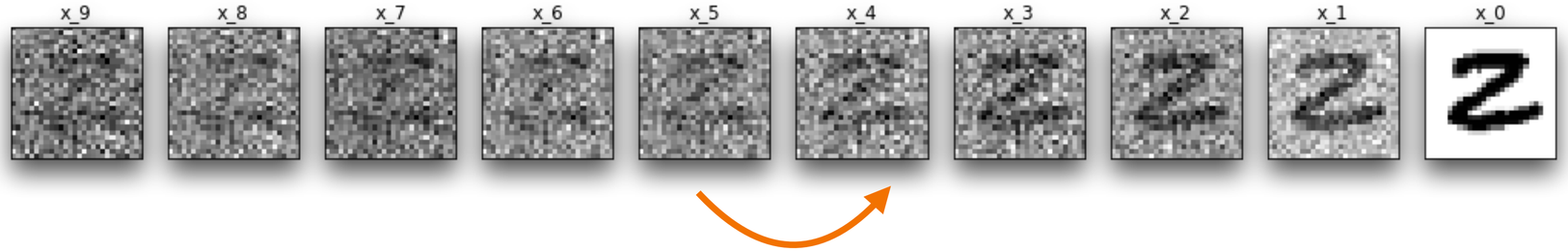
Swiss roll dataset, $\beta = 0.05$, $T = 10$



Sohl-Dickstein, Jascha, et al. "Deep unsupervised learning using nonequilibrium thermodynamics." *International Conference on Machine Learning*. PMLR, 2015.

Diffusion Model - Posterior

Forward Process $q\left(\mathbf{x}_{1:T} \mid \mathbf{x}_0\right) := \prod_{t=1}^{T} q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}\right), q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{x}_0\right) := \mathcal{N}\left(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \mathbf{I}\right)$



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

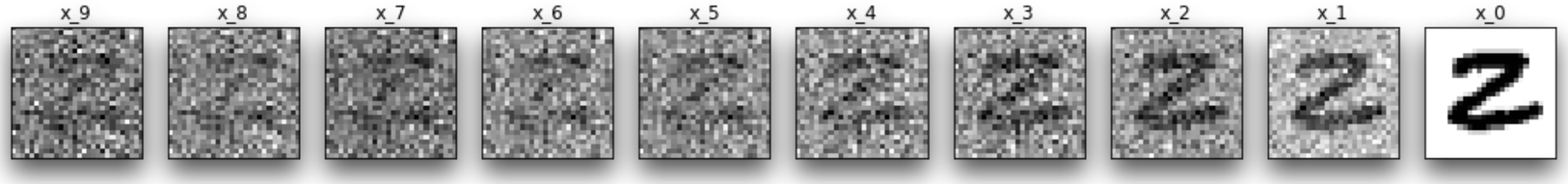
where $\tilde{\boldsymbol{\mu}}_t\left(\mathbf{x}_t, \mathbf{x}_0\right) := \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0 + \frac{\sqrt{\alpha_t}\left(1-\bar{\alpha}_{t-1}\right)}{1-\bar{\alpha}_t}\mathbf{x}_t$ and $\tilde{\beta}_t := \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$

$$q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) = \frac{q(\mathbf{x}_{t-1} \mid \mathbf{x}_0) q(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{x}_0)}{q(\mathbf{x}_t \mid \mathbf{x}_0)}$$ by Bayes' Rule

Sohl-Dickstein, Jascha, et al. "Deep unsupervised learning using nonequilibrium thermodynamics." *International Conference on Machine Learning*. PMLR, 2015.

Diffusion Model - Backward Process

Forward Process $q\left(\mathbf{x}_{1:T} \mid \mathbf{x}_0\right) := \prod_{t=1}^{T} q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}\right), q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{x}_0\right) := \mathcal{N}\left(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \mathbf{I}\right)$

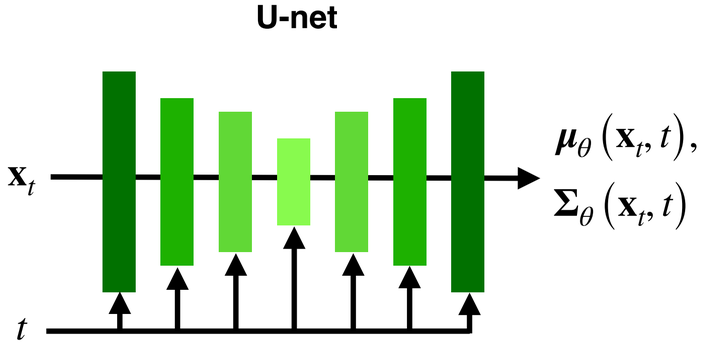


Posterior $q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0\right) = \mathcal{N}\left(\mathbf{x}_{t-1}; \tilde{\boldsymbol{\mu}}_t\left(\mathbf{x}_t, \mathbf{x}_0\right), \tilde{\beta}_t \mathbf{I}\right)$

where $\tilde{\boldsymbol{\mu}}_t\left(\mathbf{x}_t, \mathbf{x}_0\right) := \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0 + \frac{\sqrt{\alpha_t}\left(1-\bar{\alpha}_{t-1}\right)}{1-\bar{\alpha}_t}\mathbf{x}_t$ and $\tilde{\beta}_t := \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$

Backward Process (Neural Networks) $p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right) := \mathcal{N}\left(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta\left(\mathbf{x}_t, t\right), \boldsymbol{\Sigma}_\theta\left(\mathbf{x}_t, t\right)\right)$

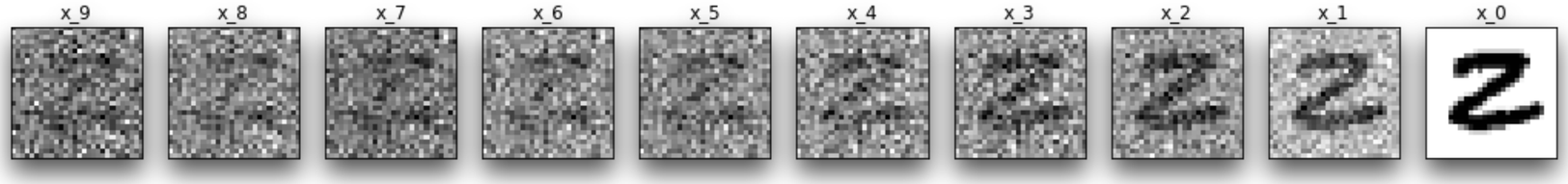
Loss Function $D_{\mathrm{KL}}\left(q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0\right) \| p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right)\right)$



Sohl-Dickstein, Jascha, et al. "Deep unsupervised learning using nonequilibrium thermodynamics." *International Conference on Machine Learning*. PMLR, 2015.

Diffusion Model - Loss Function

Forward Process $q\left(\mathbf{x}_{1:T} \mid \mathbf{x}_0\right) := \prod_{t=1}^{T} q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}\right), q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{x}_0\right) := \mathcal{N}\left(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \mathbf{I}\right)$

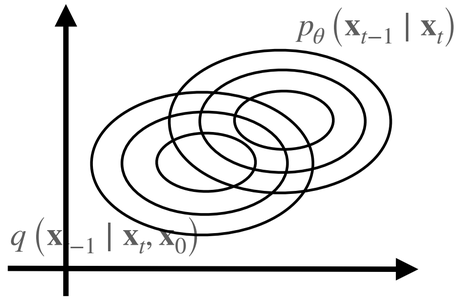


Posterior $q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0\right) = \mathcal{N}\left(\mathbf{x}_{t-1}; \tilde{\boldsymbol{\mu}}_t\left(\mathbf{x}_t, \mathbf{x}_0\right), \tilde{\beta}_t \mathbf{I}\right)$

where $\tilde{\boldsymbol{\mu}}_t\left(\mathbf{x}_t, \mathbf{x}_0\right) := \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0 + \frac{\sqrt{\alpha_t}\left(1-\bar{\alpha}_{t-1}\right)}{1-\bar{\alpha}_t}\mathbf{x}_t$ and $\tilde{\beta}_t := \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$

Backward Process (Neural Networks) $p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right) := \mathcal{N}\left(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta\left(\mathbf{x}_t, t\right), \boldsymbol{\Sigma}_\theta\left(\mathbf{x}_t, t\right)\right)$

Loss Function $D_{\mathrm{KL}}\left(q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0\right) \| p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right)\right)$



Sohl-Dickstein, Jascha, et al. "Deep unsupervised learning using nonequilibrium thermodynamics." *International Conference on Machine Learning*. PMLR, 2015.

Diffusion Model - Output Samples

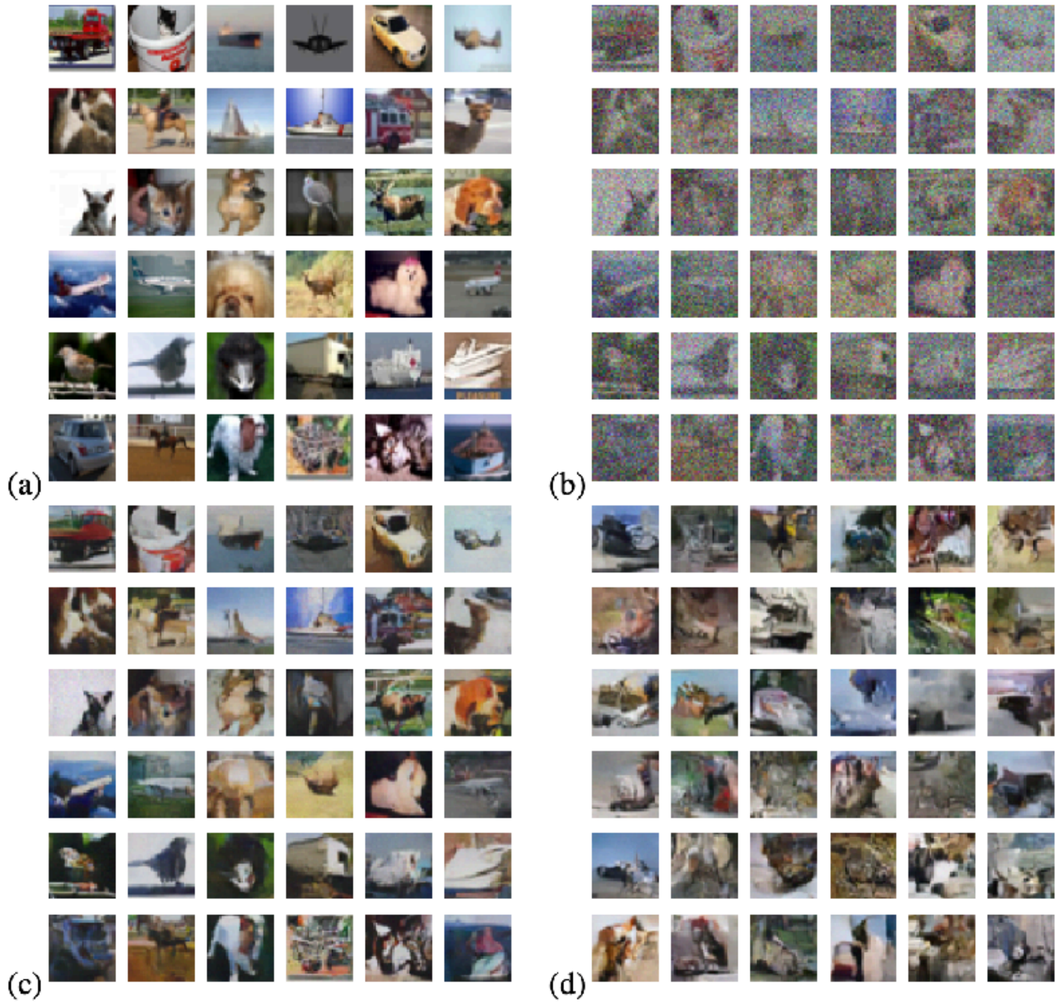


Figure 3. The proposed framework trained on the CIFAR-10 (Krizhevsky & Hinton, 2009) dataset. *(a)* Example holdout data (similar to training data). *(b)* Holdout data corrupted with Gaussian noise of variance 1 (SNR = 1). *(c)* Denoised images, generated by sampling from the posterior distribution over denoised images conditioned on the images in (b). *(d)* Samples generated by the diffusion model.

Sohl-Dickstein, Jascha, et al. "Deep unsupervised learning using nonequilibrium thermodynamics." *International Conference on Machine Learning*. PMLR, 2015.

DDPM

DDPM (Denoising Diffusion Probabilistic Models)

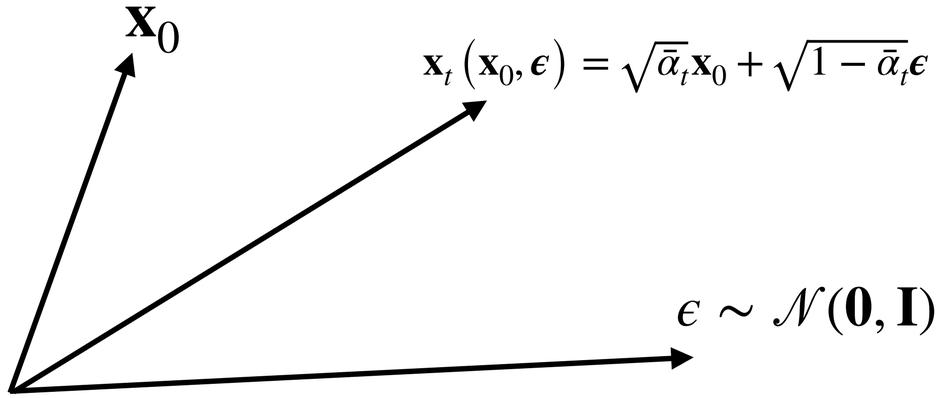
- Jonathan Ho의 논문 Denoising diffusion probabilistic models에서 제안

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

Distribution of $\mathbf{x}_t$ at an arbitrary timestep t in closed form

$q\left(\mathbf{x}_t \mid \mathbf{x}_0\right) = \mathcal{N}\left(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, \left(1-\bar{\alpha}_t\right)\mathbf{I}\right)$, where $\alpha_t := 1 - \beta_t$ and $\bar{\alpha}_t := \prod_{s=1}^{t} \alpha_s$

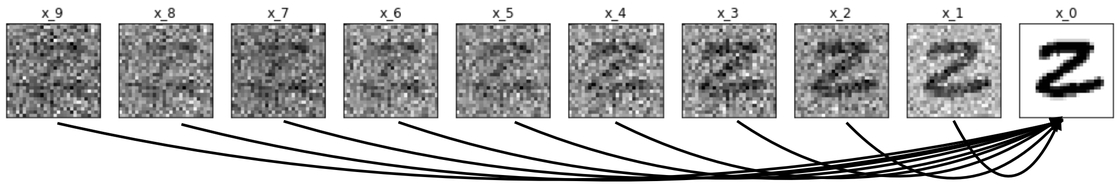
Loss Function $L_{\text{simple}}(\theta) := \mathbb{E}_{t,\mathbf{x}_0,\epsilon}\left[\left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta\left(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t\right)\right\|^2\right]$



$\mathbf{x}_t$ is a linear combination of $\mathbf{x}_0$ and ϵ

```
106     def p_loss(self, model, x_0, t, noise=None):
107         if noise is None:
108             noise = torch.randn_like(x_0)          generate ϵ
109
110         x_noise = self.q_sample(x_0, t, noise)     sample x_t
111         x_recon = model(x_noise, t)                predict ϵ
112
113         return F.mse_loss(x_recon, noise)
```

https://github.com/rosinality/denoising-diffusion-pytorch/blob/master/diffusion.py



Predict ϵ (or $\mathbf{x}_0$) at each step

Ho, Jonathan, Ajay Jain, and Pieter Abbeel. "Denoising diffusion probabilistic models." *Advances in Neural Information Processing Systems* 33 (2020): 6840-6851.

DDPM - Output Samples

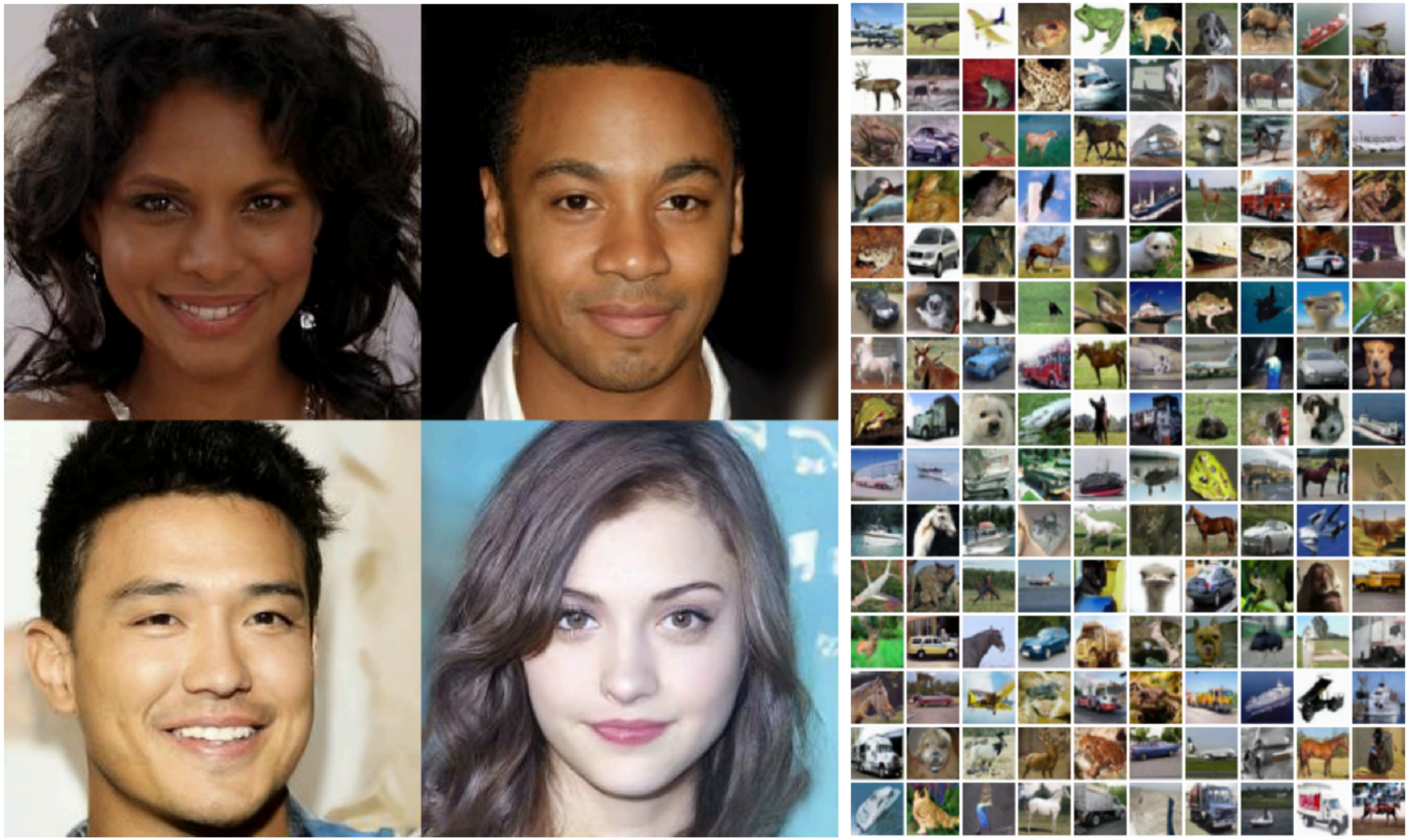
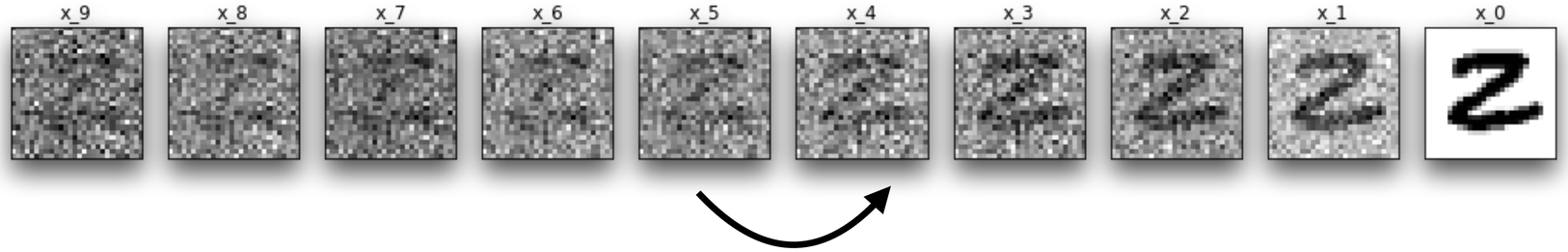


Figure 1: Generated samples on CelebA-HQ 256 × 256 (left) and unconditional CIFAR10 (right)

Ho, Jonathan, Ajay Jain, and Pieter Abbeel. "Denoising diffusion probabilistic models." *Advances in Neural Information Processing Systems* 33 (2020): 6840-6851.

DDPM의 문제점 : 너무 느리다!!!

- DDPM의 posterior는 noised image $\mathbf{x}_t$와 denoised image $\mathbf{x}_0$가 주어졌을 때, 이전 step의 noised image $\mathbf{x}_{t-1}$를 예측



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

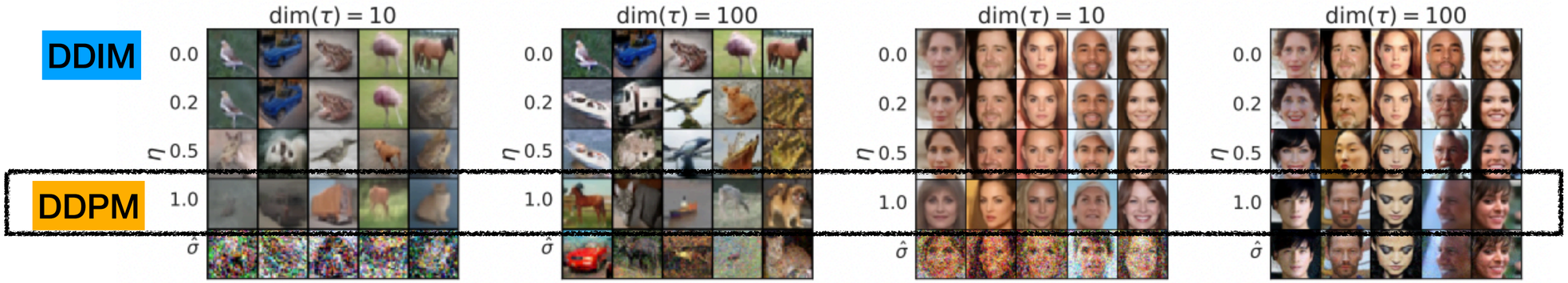
where $\tilde{\boldsymbol{\mu}}_t\left(\mathbf{x}_t, \mathbf{x}_0\right) := \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0 + \frac{\sqrt{\alpha_t}\left(1-\bar{\alpha}_{t-1}\right)}{1-\bar{\alpha}_t}\mathbf{x}_t$ and $\tilde{\beta}_t := \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$

DDPM의 문제점 : 너무 느리다!!!

- 10steps, 20step, 50steps, 100steps 씩 띄엄띄엄 예측해나가면 어떨까?

- 가능하긴 하지만 sampling하는 총 step 수가 적어짐에 따라 심각하게 quality가 저하됨

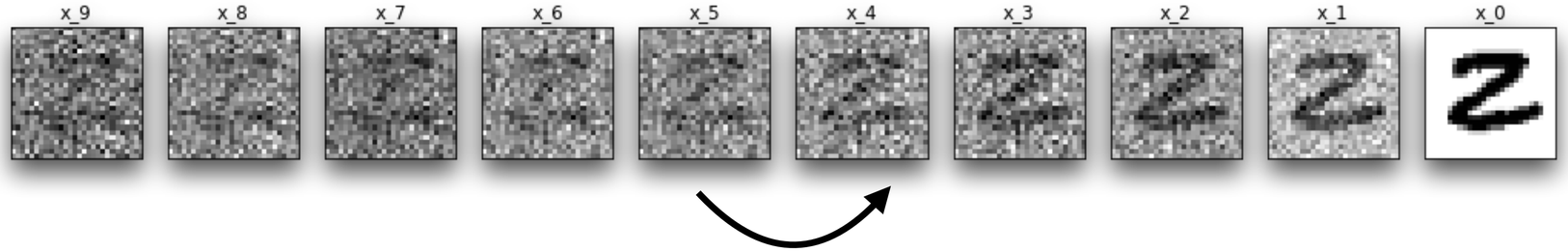
		CIFAR10 (32 × 32)					CelebA (64 × 64)				
	S	10	20	50	100	1000	10	20	50	100	1000
DDIM	0.0	**13.36**	**6.84**	**4.67**	**4.16**	4.04	**17.33**	**13.73**	**9.17**	**6.53**	3.51
	0.2	14.04	7.11	4.77	4.25	4.09	17.66	14.11	9.51	6.79	3.64
η	0.5	16.66	8.35	5.25	4.46	4.29	19.86	16.06	11.01	8.09	4.28
DDPM	1.0	41.07	18.36	8.01	5.78	4.73	33.12	26.03	18.48	13.93	5.98
	$\hat{\sigma}$	367.43	133.37	32.72	9.99	**3.17**	299.71	183.83	71.71	45.20	**3.26**



Song, Jiaming, Chenlin Meng, and Stefano Ermon. "Denoising diffusion implicit models." arXiv preprint arXiv:2010.02502 (2020).

DDIM

- Noised image $\mathbf{x}_t$와 denoised image $\mathbf{x}_0$가 주어졌을 때, $\mathbf{x}_{t-1}$를 예측하는 posterior부터 다시 설계해보자!
(DDPM은 forward process부터 설계하고 posterior를 유도해 냄)



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

where $\tilde{\boldsymbol{\mu}}_t\left(\mathbf{x}_t, \mathbf{x}_0\right) := \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0 + \frac{\sqrt{\alpha_t}\left(1-\bar{\alpha}_{t-1}\right)}{1-\bar{\alpha}_t}\mathbf{x}_t$ and $\tilde{\beta}_t := \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$

Generalized Posterior $q_\sigma\left(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, \boldsymbol{x}_0\right) = \mathcal{N}\left(\sqrt{\alpha_{t-1}}\boldsymbol{x}_0 + \sqrt{1-\alpha_{t-1}-\sigma_t^2} \cdot \frac{\boldsymbol{x}_t - \sqrt{\alpha_t}\boldsymbol{x}_0}{\sqrt{1-\alpha_t}}, \sigma_t^2 \boldsymbol{I}\right)$

* DDPM에서 보이는 $\bar{\alpha}$는 DDIM에서 α로 쓰임

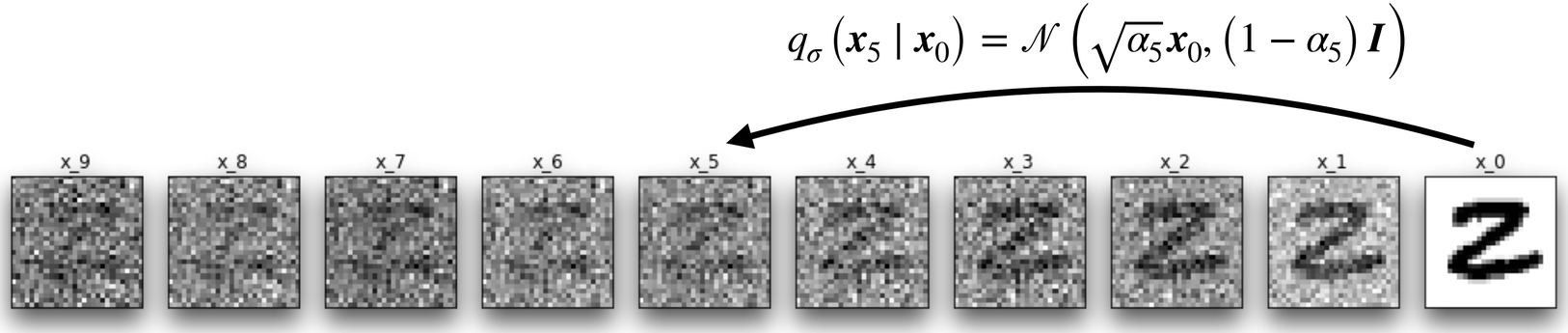
Ho, Jonathan, Ajay Jain, and Pieter Abbeel. "Denoising diffusion probabilistic models." *Advances in Neural Information Processing Systems* 33 (2020): 6840-6851.
Song, Jiaming, Chenlin Meng, and Stefano Ermon. "Denoising diffusion implicit models." arXiv preprint arXiv:2010.02502 (2020).

DDIM

- Generalized posterior $q_\sigma\left(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, \boldsymbol{x}_0\right)$는 어떻게 설계된 식인가?

>> DDPM에서 언급했던 $q_\sigma\left(\boldsymbol{x}_t \mid \boldsymbol{x}_0\right)=\mathcal{N}\left(\sqrt{\alpha_t} \boldsymbol{x}_0,\left(1-\alpha_t\right) \boldsymbol{I}\right)$식을 만족하도록 설계

>> 자세한 식 유도는 DDIM 논문 Appendix B에 언급



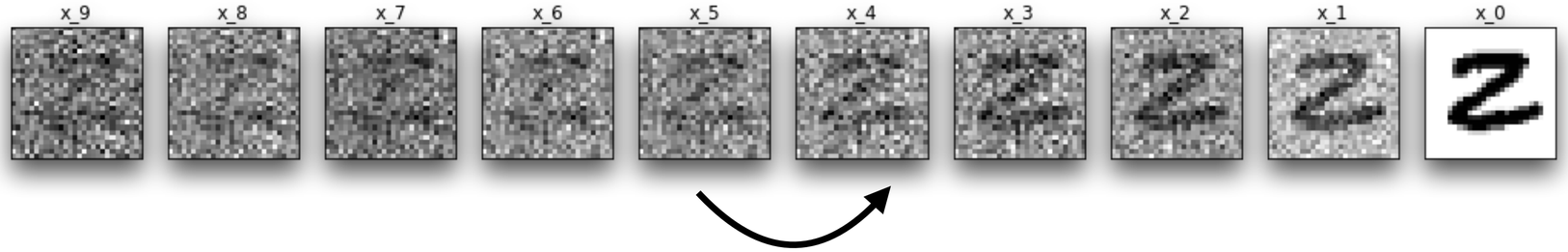
Distribution of $\mathbf{x}_t$ at an arbitrary timestep t in closed form

$$q_\sigma\left(\boldsymbol{x}_t \mid \boldsymbol{x}_0\right)=\mathcal{N}\left(\sqrt{\alpha_t} \boldsymbol{x}_0,\left(1-\alpha_t\right) \boldsymbol{I}\right)$$

Song, Jiaming, Chenlin Meng, and Stefano Ermon. "Denoising diffusion implicit models." arXiv preprint arXiv:2010.02502 (2020).

DDIM

- Generalized posterior에서 $\sigma_t = \sqrt{(1-\alpha_{t-1})/(1-\alpha_t)}\sqrt{1-\alpha_t/\alpha_{t-1}}$으로 두면 DDPM과 같아짐
- $\sigma_t = 0$ 으로 두면 posterior가 deterministic function이 되고 이것을 DDIM이라고 부름



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

where $\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0) := \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t$ and $\tilde{\beta}_t := \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$

Generalized Posterior $q_\sigma(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, \boldsymbol{x}_0) = \mathcal{N}\left(\sqrt{\alpha_{t-1}}\boldsymbol{x}_0 + \sqrt{1-\alpha_{t-1}-\sigma_t^2} \cdot \frac{\boldsymbol{x}_t - \sqrt{\alpha_t}\boldsymbol{x}_0}{\sqrt{1-\alpha_t}}, \sigma_t^2 \boldsymbol{I}\right)$

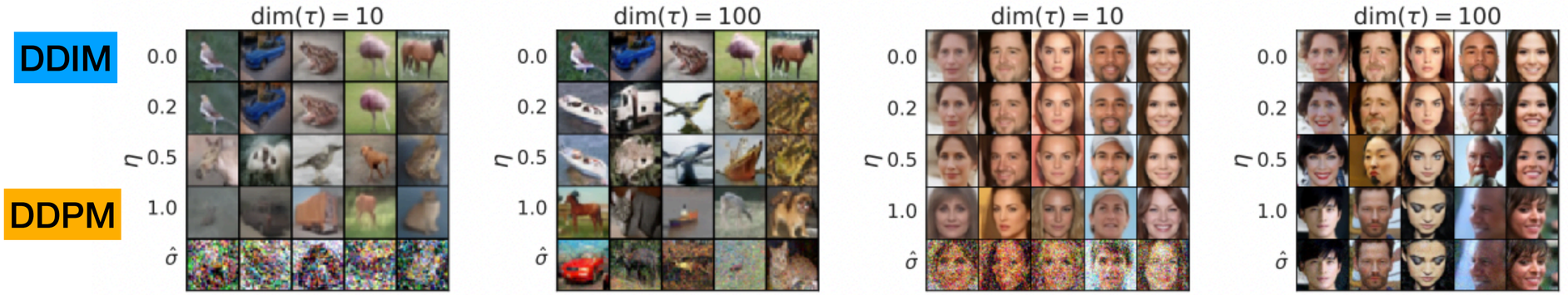
$\sigma_t = 0$ 이면 DDIM

Ho, Jonathan, Ajay Jain, and Pieter Abbeel. "Denoising diffusion probabilistic models." *Advances in Neural Information Processing Systems* 33 (2020): 6840-6851.
Song, Jiaming, Chenlin Meng, and Stefano Ermon. "Denoising diffusion implicit models." arXiv preprint arXiv:2010.02502 (2020).

DDPM vs. DDIM

- $\sigma_{\tau_i}(\eta) = \eta\sqrt{\left(1-\alpha_{\tau_{i-1}}\right)/\left(1-\alpha_{\tau_i}\right)}\sqrt{1-\alpha_{\tau_i}/\alpha_{\tau_{i-1}}}$로 두고 η를 0.0, 0.2, 0.5, 1.0로 바꿔가면서 실험
($\eta = 0$이면 DDIM, $\eta = 1$이면 DDPM)
- 1000 steps로 트레이닝된 모델을 10, 20, 50, 100, 1000 steps로 inference해가면서 실험
- DDIM이 전체적으로 FID값이 더 낮으며 sampling step 수가 적을 때 더욱 차이가 난다

		CIFAR10 (32 × 32)					CelebA (64 × 64)				
	S	10	20	50	100	1000	10	20	50	100	1000
DDIM	0.0	**13.36**	**6.84**	**4.67**	**4.16**	4.04	**17.33**	**13.73**	**9.17**	**6.53**	3.51
	0.2	14.04	7.11	4.77	4.25	4.09	17.66	14.11	9.51	6.79	3.64
η	0.5	16.66	8.35	5.25	4.46	4.29	19.86	16.06	11.01	8.09	4.28
DDPM	1.0	41.07	18.36	8.01	5.78	4.73	33.12	26.03	18.48	13.93	5.98
	$\hat{\sigma}$	367.43	133.37	32.72	9.99	**3.17**	299.71	183.83	71.71	45.20	**3.26**



Song, Jiaming, Chenlin Meng, and Stefano Ermon. "Denoising diffusion implicit models." arXiv preprint arXiv:2010.02502 (2020).

DDIM

- DDIM은 DDPM의 posterior를 일반화하고 stochastic한 항을 지우고 deterministic function으로 만든 것!

- 왜 Denoising Diffusion “Implicit” Models인가?

“Implicit generative models use a latent variable z and transform it using ***a deterministic function*** G_θ *that maps form* $\mathbb{R}^m \rightarrow \mathbb{R}^n$ *using parameters* θ*.”*

Mohamed, Shakir, and Balaji Lakshminarayanan. "Learning in implicit generative models." arXiv preprint arXiv:1610.03483 (2016).

DDIM
Keras Example

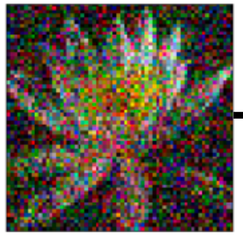
DDIM - def denoise

```
def denoise(self, noisy_images, noise_rates, signal_rates, training):
    # the exponential moving average weights are used at evaluation
1.  if training:
        network = self.network
    else:
        network = self.ema_network

    # predict noise component and calculate the image component using it
2.  pred_noises = network([noisy_images, noise_rates**2], training=training)
    pred_images = (noisy_images - noise_rates * pred_noises) / signal_rates

    return pred_noises, pred_images
```

https://keras.io/examples/generative/ddim/

1. Training network와 EMA network를 별도로 두고, training시에는 training network를, inference시에는 EMA network를 사용

2. Network에 noisy image $\mathbf{x}_t$를 넣고, predicted noise를 출력



Noisy Image $\mathbf{x}_t$

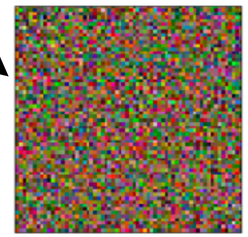
$$\mathbf{x}_t = \sqrt{\alpha_t}\mathbf{x}_0 + \sqrt{1-\alpha_t}\epsilon_t$$

Model

U-net

Predicted Image

$$\mathbf{x}_0 = \frac{\mathbf{x}_t - \sqrt{1-\alpha_t}\epsilon_\theta^{(t)}(\mathbf{x}_t)}{\sqrt{\alpha_t}}$$



Predicted Noise

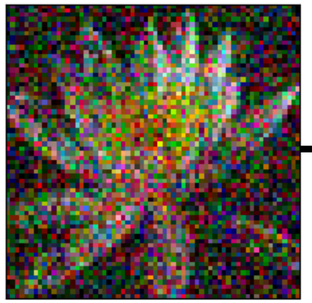
$$\epsilon_\theta^{(t)}(\mathbf{x}_t)$$

DDIM - def train_step

$$\mathbf{x}_0 = \frac{\mathbf{x}_t - \sqrt{1-\alpha_t}\epsilon_\theta^{(t)}(\mathbf{x}_t)}{\sqrt{\alpha_t}}$$

Image $\mathbf{x}_0$

α_t

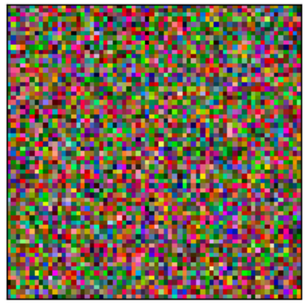


Noisy Image $\mathbf{x}_t$

$$\mathbf{x}_t = \sqrt{\alpha_t}\mathbf{x}_0 + \sqrt{1-\alpha_t}\epsilon_t$$

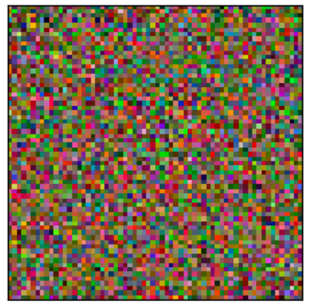
Model

Predicted Image



Noise $\epsilon_t \sim \mathbf{N}(\mathbf{0}, \mathbf{I})$

$\sqrt{1-\alpha_t}$



Predicted Noise $\epsilon_\theta^{(t)}(\mathbf{x}_t)$

MSE Loss

DDIM - def train_step

```
def train_step(self, images):
    # normalize images to have standard deviation of 1, like the noises
1.  images = self.normalizer(images, training=True)
    noises = tf.random.normal(shape=(batch_size, image_size, image_size, 3))

    # sample uniform random diffusion times
2.  diffusion_times = tf.random.uniform(
        shape=(batch_size, 1, 1, 1), minval=0.0, maxval=1.0
    )
3.  noise_rates, signal_rates = self.diffusion_schedule(diffusion_times)
    # mix the images with noises accordingly
4.  noisy_images = signal_rates * images + noise_rates * noises

    with tf.GradientTape() as tape:
        # train the network to separate noisy images to their components
5.      pred_noises, pred_images = self.denoise(
            noisy_images, noise_rates, signal_rates, training=True
        )

        noise_loss = self.loss(noises, pred_noises)  # used for training
6.      image_loss = self.loss(images, pred_images)  # only used as metric

7.  gradients = tape.gradient(noise_loss, self.network.trainable_weights)
    self.optimizer.apply_gradients(zip(gradients, self.network.trainable_weights))

    self.noise_loss_tracker.update_state(noise_loss)
    self.image_loss_tracker.update_state(image_loss)

    # track the exponential moving averages of weights
8.  for weight, ema_weight in zip(self.network.weights, self.ema_network.weights):
        ema_weight.assign(ema * ema_weight + (1 - ema) * weight)

    # KID is not measured during the training phase for computational efficiency
    return {m.name: m.result() for m in self.metrics[:-1]}
```

https://keras.io/examples/generative/ddim/

1. Image $\mathbf{x}_0$를 normalization하고, noise $\epsilon_t \sim \mathbf{N}(\mathbf{0}, \mathbf{I})$를 샘플링

2. Diffusion time $t \sim \mathbf{U}(0,1)$를 샘플링

3. Diffusion schedule에 따라 signal rate $\sqrt{\alpha_t}$와 noise rate $\sqrt{1-\alpha_t}$를 구함

4. Noisy image $\mathbf{x}_t$를 만듬. $\mathbf{x}_t = \sqrt{\alpha_t}\mathbf{x}_0 + \sqrt{1-\alpha_t}\epsilon_t$

5. Model에 noisy image $\mathbf{x}_t$를 넣어 predicted image와 predicted noise $\epsilon_\theta^{(t)}(\mathbf{x}_t)$ 출력

6. 본래의 noise ϵ와 predicted noise $\epsilon_\theta^{(t)}(\mathbf{x}_t)$ 간에 MSE loss 잡음

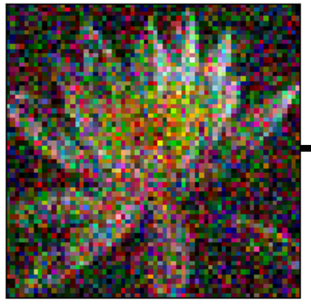
7. Gradient를 구하고 optimizer를 이용하여 weight 업데이트

8. EMA (Exponential Moving Average) model의 weight를 업데이트

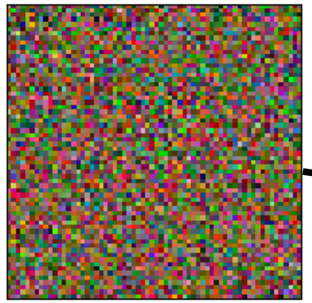
DDIM - def reverse_diffusion

$$\mathbf{x}_0 = \frac{\mathbf{x}_t - \sqrt{1-\alpha_t}\epsilon_\theta^{(t)}\left(\mathbf{x}_t\right)}{\sqrt{\alpha_t}}$$

α_{t-1}



Model



$\sqrt{1-\alpha_{t-1}}$

Noisy Image $\mathbf{x}_t$

Predicted Noise $\epsilon_\theta^{(t)}(\mathbf{x}_t)$

Noisy Image $\mathbf{x}_{t-1}$

$$\mathbf{x}_{t-1} = \sqrt{\alpha_t}\mathbf{x}_0 + \sqrt{1-\alpha_t}\epsilon_\theta^{(t)}(\mathbf{x}_t)$$

DDIM - def reverse_diffusion

```
def reverse_diffusion(self, initial_noise, diffusion_steps):
    # reverse diffusion = sampling
1.  num_images = initial_noise.shape[0]
    step_size = 1.0 / diffusion_steps

    # important line:
    # at the first sampling step, the "noisy image" is pure noise
    # but its signal rate is assumed to be nonzero (min_signal_rate)
2.  next_noisy_images = initial_noise
    for step in range(diffusion_steps):
        noisy_images = next_noisy_images

        # separate the current noisy image to its components
3.      diffusion_times = tf.ones((num_images, 1, 1, 1)) - step * step_size
        noise_rates, signal_rates = self.diffusion_schedule(diffusion_times)
4.      pred_noises, pred_images = self.denoise(
            noisy_images, noise_rates, signal_rates, training=False
        )
        # network used in eval mode

        # remix the predicted components using the next signal and noise rates
5.      next_diffusion_times = diffusion_times - step_size
        next_noise_rates, next_signal_rates = self.diffusion_schedule(
            next_diffusion_times
        )
6.      next_noisy_images = (
            next_signal_rates * pred_images + next_noise_rates * pred_noises
        )
        # this new noisy image will be used in the next step

    return pred_images
```

1. Batch size인 num_images 구하고, sampling을 수행할 step 수 diffusion_steps으로부터 step_size 구함

2. initial_noise에서부터 생성 iteration시작

3. diffusion_times는 1에서부터 0까지 step_size만큼씩 줄여감, Diffusion schedule에 따라 signal rate $\sqrt{\alpha_t}$와 noise rate $\sqrt{1-\alpha_t}$를 구함

4. Model에 noisy_images $\mathbf{x}_t$를 넣어 predicted image $\mathbf{x}_0$와 predicted noise $\epsilon_\theta^{(t)}(\mathbf{x}_t)$ 출력

5. next_diffusion_times을 구하고, diffusion schedule에 따라 signal rate $\sqrt{\alpha_{t-1}}$와 noise rate $\sqrt{1-\alpha_t}$를 구함

6. 다음 step의 Noisy image $\mathbf{x}_{t-1}$를 만듬.

$$\mathbf{x}_{t-1} = \sqrt{\alpha_{t-1}}\mathbf{x}_0 + \sqrt{1-\alpha_{t-1}}\epsilon_\theta^{(t)}(\mathbf{x}_t)$$

https://keras.io/examples/generative/ddim/