Deep learning

x.y. Denoising Diffusion

François Fleuret

https://fleuret.org/dlc/

UNIVERSITÉ DE GENÈVE

The denoising diffusion (Ho et al., 2020) is a generative model that consists in (1) generating data with a diffusion process, and (2) training a stochastic denoising auto-encoder to reverse it.

The synthesis starts by sampling a random signal corresponding to the limit of the diffusion process, and iterates the denoising auto-encoder for the same number of iterations.

Given a data-set $\mathscr{D} \subset \mathbb{R}^D$, and

$$0 < \beta_t < 1, t = 1, \dots, T,$$

let q be a distribution over $\mathbb{R}^{D \times T}$ of a "diffusion process", defined as

$$\begin{aligned} x_0 &\sim \mathscr{U}(\mathscr{D}) \\ \forall t = 1, \dots, T, \ \epsilon_t &\sim \mathscr{N}(0, \mathbf{I}) \\ x_t &= \sqrt{1 - \beta_t}\, x_{t-1} + \sqrt{\beta_t}\, \epsilon_t \end{aligned}$$

with x_0 and the ϵ_ts independent.

Given a data-set $\mathscr{D} \subset \mathbb{R}^D$, and

$$0 < \beta_t < 1, t = 1, \dots, T,$$

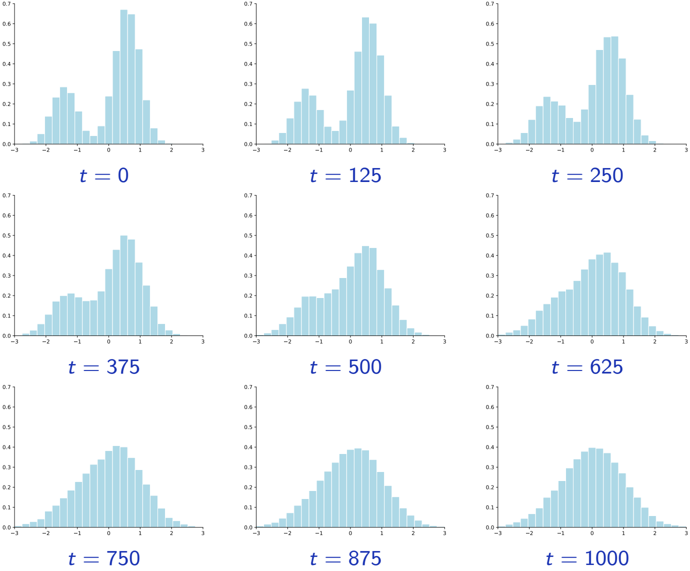
let q be a distribution over $\mathbb{R}^{D \times T}$ of a “diffusion process”, defined as

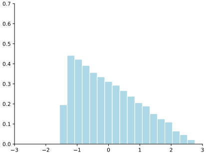
$$\begin{aligned} x_0 &\sim \mathscr{U}(\mathscr{D}) \\ \forall t = 1, \dots, T,\ \epsilon_t &\sim \mathscr{N}(0, \mathbf{I}) \\ x_t &= \sqrt{1 - \beta_t}\, x_{t-1} + \sqrt{\beta_t}\, \epsilon_t \end{aligned}$$

with x_0 and the ϵ_ts independent.

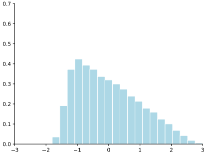
The re-scaling factors are such that if $\mathbb{E}[x_0] = 0$ and $\mathbb{V}[x_0] = \mathbf{I}$, we have

$$\forall t = 0, \dots T,\ \mathbb{E}[x_t] = 0, \mathbb{V}[x_t] = \mathbf{I}.$$

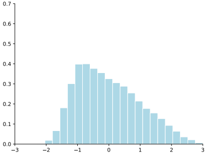

$t = 0$
$t = 125$
$t = 250$
$t = 375$
$t = 500$
$t = 625$
$t = 750$
$t = 875$
$t = 1000$



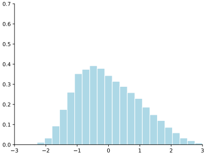
$t = 0$



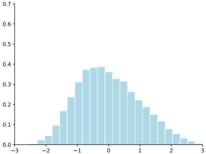
$t = 125$



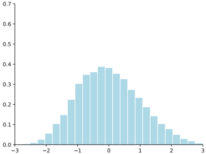
$t = 250$



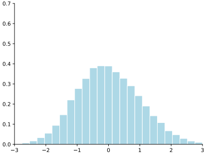
$t = 375$



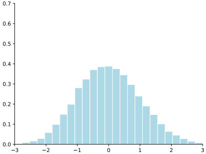
$t = 500$



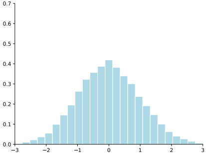
$t = 625$



$t = 750$



$t = 875$



$t = 1000$

Thanks to the independence of the successive steps, and since a sum of independent Gaussians is Gaussian, we can sample directly $x_t \mid x_0$ for any t.

If we define

$$\alpha_t = 1 - \beta_t$$
$$\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$$

we have

$$\forall t > 0, \; q(x_t \mid x_0) = \mathcal{N}\left(x_t; \sqrt{\bar{\alpha}_t}\, x_0, (1 - \bar{\alpha}_t)\, \mathbf{I}\right).$$

This shows in particular that even with small β_t, for T large enough, the distribution $x_T \mid x_0$ is $\simeq \mathcal{N}(0, \mathbf{I})$ and does not depend on the data distribution

Let p_θ be the distribution of the denoising process, which is Markovian too

$$\begin{aligned}
\log p_\theta(x_{0:T}) &= \log p_\theta(x_T) + \sum_{t=1}^{T} \log p_\theta(x_{t-1} \mid x_{t:T}) \\
&= \log p_\theta(x_T) + \sum_{t=1}^{T} \log p_\theta(x_{t-1} \mid x_t).
\end{aligned}$$

Let p_θ be the distribution of the denoising process, which is Markovian too

$$\begin{aligned}\log p_\theta(x_{0:T}) &= \log p_\theta(x_T) + \sum_{t=1}^{T} \log p_\theta(x_{t-1} \mid x_{t:T}) \\ &= \log p_\theta(x_T) + \sum_{t=1}^{T} \log p_\theta(x_{t-1} \mid x_t).\end{aligned}$$

If we are interested by synthesis of clean samples, given training samples $z_1, \dots, z_N$, we want to minimize

$$-\log \prod_{n=1}^{N} p_\theta(z_n) = N\, \bar{\mathbb{E}}_{x_{0:T} \sim q}\left[-\log p_\theta(x_0)\right].$$

$$
\begin{aligned}
\mathbb{E}_{x_{0:T}\sim q}\left[-\log p_\theta(x_0)\right] &= \mathbb{E}_{x_{0:T}\sim q}\left[-\log \frac{p_\theta(x_{1:T}\mid x_0)}{p_\theta(x_{1:T}\mid x_0)} p_\theta(x_0)\right] \\
&= \mathbb{E}_{x_{0:T}\sim q}\left[-\log p_\theta(x_{0:T})\right] + \mathbb{E}_{x_{0:T}\sim q}\left[\log p_\theta(x_{1:T}\mid x_0)\right] \\
&= \mathbb{E}_{x_{0:T}\sim q}\left[-\log p_\theta(x_{0:T})\right] + \mathbb{E}_{x_0\sim q}\mathbb{E}_{x_{1:T}\sim q\mid x_0}\left[\log p_\theta(x_{1:T}\mid x_0)\right] \\
&\leq \mathbb{E}_{x_{0:T}\sim q}\left[-\log p_\theta(x_{0:T})\right] + \mathbb{E}_{x_0\sim q}\mathbb{E}_{x_{1:T}\sim q\mid x_0}\left[\log q(x_{1:T}\mid x_0)\right] \\
&= \mathbb{E}_{x_{0:T}\sim q}\left[-\log p_\theta(x_{0:T})\right] + \mathbb{E}_{x_{0:T}\sim q}\left[\log q(x_{1:T}\mid x_0)\right] \\
&= \mathbb{E}_{x_{0:T}\sim q}\left[-\log \frac{p_\theta(x_{0:T})}{q(x_{1:T}\mid x_0)}\right] \\
&= \mathbb{E}_{x_{0:T}\sim q}\left[-\log \frac{p_\theta(x_T)\prod_{t=1}^{T} p_\theta(x_{t-1}\mid x_t)}{\prod_{t=1}^{T} q(x_t\mid x_{t-1})}\right] \\
&= \mathbb{E}_{x_{0:T}\sim q}\left[-\log p_\theta(x_T) - \sum_{t=1}^{T}\log \frac{p_\theta(x_{t-1}\mid x_t)}{q(x_t\mid x_{t-1})}\right]
\end{aligned}
$$

So we get

$$\mathbb{E}_{x_{0:T} \sim q}\left[-\log p_\theta(x_0)\right] \leq \mathbb{E}_{x_{0:T} \sim q}\left[-\log p_\theta(x_T) - \sum_{t=1}^{T} \log \frac{p_\theta(x_{t-1} \mid x_t)}{q(x_t \mid x_{t-1})}\right],$$

if $p_\theta(x_T)$ is a fixed distribution that does not depend on θ, we have to minimize

$$\mathbb{E}_{x_{0:T} \sim q}\left[-\sum_{t=1}^{T} \log p_\theta(x_{t-1} \mid x_t)\right],$$

hence train a model that approximates the reverse Markov process.

So we get

$$\mathbb{E}_{x_{0:T}\sim q}\left[-\log p_\theta(x_0)\right] \leq \mathbb{E}_{x_{0:T}\sim q}\left[-\log p_\theta(x_T) - \sum_{t=1}^{T}\log\frac{p_\theta(x_{t-1}\mid x_t)}{q(x_t\mid x_{t-1})}\right],$$

if $p_\theta(x_T)$ is a fixed distribution that does not depend on θ, we have to minimize

$$\mathbb{E}_{x_{0:T}\sim q}\left[-\sum_{t=1}^{T}\log p_\theta(x_{t-1}\mid x_t)\right],$$

hence train a model that approximates the reverse Markov process.

We can take

$$p_\theta(x_{t-1}\mid x_t) = \mathcal{N}(x_{t-1};\mu_\theta(x_t),\beta_t \mathbf{I}).$$

But we can take advantage of the Gaussian distributions, in particular that:

- we can sample $q(x_0, x_t)$,

But we can take advantage of the Gaussian distributions, in particular that:

- we can sample $q(x_0, x_t)$,
- the distribution of $q(x_{t-1} \mid x_0, x_t)$ is a Gaussian

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

with

$$\begin{aligned}
\tilde{\mu}_t(x_t, x_0) &= \frac{\sqrt{\bar{\alpha}_{t-1}}\, \beta_t}{1 - \bar{\alpha}_t} x_0 + \frac{\sqrt{\alpha_t}\, (1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t} x_t \\
\tilde{\beta}_t &= \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t} \beta_t,
\end{aligned}$$

But we can take advantage of the Gaussian distributions, in particular that:

- we can sample $q(x_0, x_t)$,
- the distribution of $q(x_{t-1} \mid x_0, x_t)$ is a Gaussian

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

with

$$\tilde{\mu}_t(x_t, x_0) = \frac{\sqrt{\bar{\alpha}_{t-1}}\,\beta_t}{1 - \bar{\alpha}_t} x_0 + \frac{\sqrt{\alpha_t}\,(1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t} x_t$$
$$\tilde{\beta}_t = \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t} \beta_t,$$

- given two Gaussians μ and μ', their cross-entropy

$$\mathbb{H}(\mu, \mu') = \mathbb{E}_{x \sim \mu}\left[-\log \mu'(x)\right]$$

has a closed form.

$$\mathbb{E}_{x_{0:T} \sim q}\left[-\sum_{t=1}^{T} \log p_\theta(x_{t-1} \mid x_t)\right]$$

$$
\begin{aligned}
&= -\sum_{t=1}^{T} \mathbb{E}_{x_{0:T} \sim q}\left[\log p_\theta(x_{t-1} \mid x_t)\right] \\
&= -\sum_{t=1}^{T} \mathbb{E}_{x_0, x_{t-1}, x_t \sim q}\left[\log p_\theta(x_{t-1} \mid x_t)\right] \\
&= -\sum_{t=1}^{T} \mathbb{E}_{x_0, x_t \sim q}\left[\mathbb{E}_{x_{t-1} \sim q \mid x_0, x_t}\left[\log p_\theta(x_{t-1} \mid x_t)\right]\right] \\
&= -\sum_{t=1}^{T} \mathbb{E}_{x_0, x_t \sim q}\left[\mathbb{H}(q(x_{t-1} \mid x_0, x_t), p_\theta(x_{t-1} \mid x_t)\right] \\
&= \sum_{t=1}^{T} \frac{1}{2\sigma_t} \mathbb{E}_{x_0, x_t \sim q}\left[\left\| \tilde{\mu}_t(x_t, x_0) - \mu_\theta(x_t, t) \right\|^2\right] + \text{cst}
\end{aligned}
$$

Ho et al. (2020) re-parametrize

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

Following the setup of Ho et al. (2020), we have

$$\alpha_t = 1 - \beta_t$$
$$\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$$
$$\sigma_t = \sqrt{\beta_t}$$

```
T = 1000
beta = torch.linspace(1e-4, 0.02, T, device = device)
alpha = 1 - beta
alpha_bar = alpha.log().cumsum(0).exp()
sigma = beta.sqrt()
```

Algorithm 1 Training

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

(Ho et al., 2020)

```
for k in range(args.nb_epochs):

    optimizer = torch.optim.Adam(model.parameters(), lr = args.learning_rate)

    for x0 in train_input.split(args.batch_size):
        x0 = (x0 - train_mean) / train_std
        t = torch.randint(T, (x0.size(0),) + (1,) * (x0.dim() - 1), device = x0.device)
        eps = torch.randn_like(x0)
        xt = torch.sqrt(alpha_bar[t]) * x0 + torch.sqrt(1 - alpha_bar[t]) * eps
        output = model((xt, t / (T - 1) - 0.5))
        loss = (eps - output).pow(2).mean()

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

Algorithm 2 Sampling

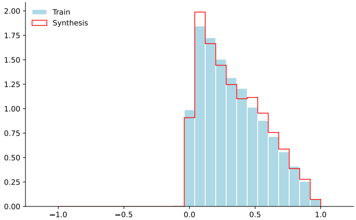
1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \ldots, 1$ **do**
3: $\quad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\quad \mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}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

(Ho et al., 2020)

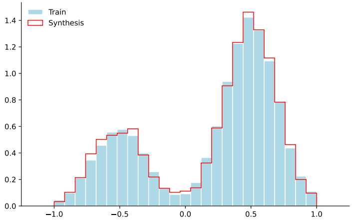
```
def generate(size, T, alpha, alpha_bar, sigma, model, train_mean, train_std):

    with torch.no_grad():

        x = torch.randn(size, device = device)

        for t in range(T-1, -1, -1):
            output = model((x, t / (T - 1) - 0.5))
            z = torch.zeros_like(x) if t == 0 else torch.randn_like(x)
            x = 1/torch.sqrt(alpha[t]) \
                * (x - (1-alpha[t]) / torch.sqrt(1-alpha_bar[t]) * output) \
                + sigma[t] * z

        x = x * train_std + train_mean

        return x
```

```
class TimeAppender(nn.Module):
    def __init__(self):
        super().__init__()

    def forward(self, u):
        x, t = u
        if not torch.is_tensor(t):
            t = x.new_full((x.size(0),), t)
        t = t.view((-1,) + (1,) * (x.dim() - 1)).expand_as(x[:,:1])
        return torch.cat((x, t), 1)
```

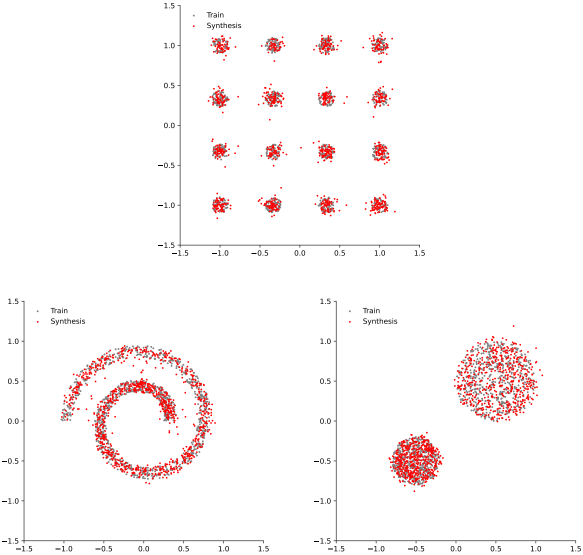
```
nh = 256

model = nn.Sequential(
    TimeAppender(),
    nn.Linear(train_input.size(1) + 1, nh),
    nn.ReLU(),
    nn.Linear(nh, nh),
    nn.ReLU(),
    nn.Linear(nh, nh),
    nn.ReLU(),
    nn.Linear(nh, train_input.size(1)),
)
```



Train
Synthesis
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
−1.0
−0.5
0.0
0.5
1.0

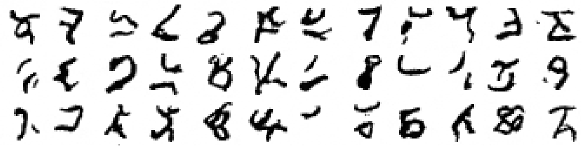


Train
Synthesis
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−1.0
−0.5
0.0
0.5
1.0

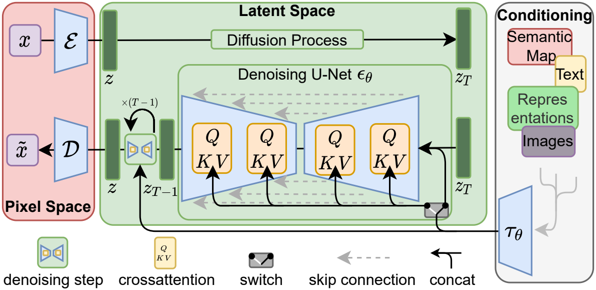

Train
Synthesis
Train
Synthesis
Train
Synthesis

```
ks, nc = 5, 64

model = nn.Sequential(
    TimeAppender(),
    nn.Conv2d(train_input.size(1) + 1, nc, ks, padding = ks//2),
    nn.ReLU(),
    nn.Conv2d(nc, nc, ks, padding = ks//2),
    nn.ReLU(),
    nn.Conv2d(nc, nc, ks, padding = ks//2),
    nn.ReLU(),
    nn.Conv2d(nc, nc, ks, padding = ks//2),
    nn.ReLU(),
    nn.Conv2d(nc, nc, ks, padding = ks//2),
    nn.ReLU(),
    nn.Conv2d(nc, nc, ks, padding = ks//2),
    nn.ReLU(),
    nn.Conv2d(nc, train_input.size(1), ks, padding = ks//2),
)
```

Generated samples



"Stable diffusion"



(Rombach et al., 2021)

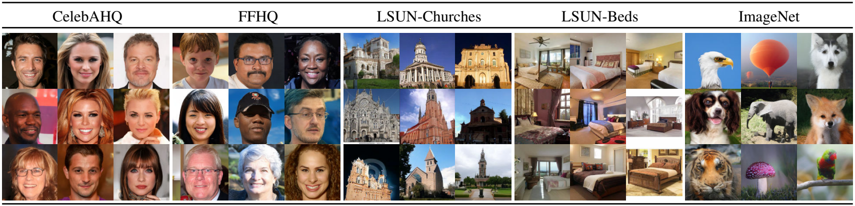


Figure 4. Samples from *LDMs* trained on CelebAHQ [39], FFHQ [41], LSUN-Churches [102], LSUN-Bedrooms [102] and class-conditional ImageNet [12], each with a resolution of 256×256. Best viewed when zoomed in. For more samples *cf*. the supplement.

(Rombach et al., 2021)

Text-to-Image Synthesis on LAION. 1.45B Model.

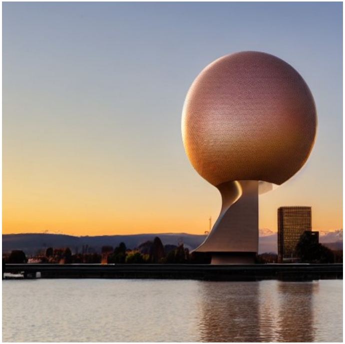
'A street sign that reads "Latent Diffusion" '	'A zombie in the style of Picasso'	'An image of an animal half mouse half octopus'	'An illustration of a slightly conscious neural network'	'A painting of a squirrel eating a burger'	'A watercolor painting of a chair that looks like an octopus'	'A shirt with the inscription: "I love generative models!" '

Figure 5. Samples for user-defined text prompts from our model for text-to-image synthesis, *LDM-8 (KL)*, which was trained on the LAION [78] database. Samples generated with 200 DDIM steps and $\eta = 1.0$. We use unconditional guidance [32] with $s = 10.0$.

(Rombach et al., 2021)

"photography of a real car made of pizzas ; very detailed, focused, beautiful light"

"still from studio ghibli movie 'the red bus in Paris '; 8 k ; very detailed, focused, colorful, trending on artstation"



“An enormous artificial intelligence in Geneva at dawn”

The End

References

J. Ho, A. Jain, and P. Abbeel. **Denoising diffusion probabilistic models**. CoRR, abs/2006.11239, 2020.

R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer. **High-resolution image synthesis with latent diffusion models**. CoRR, abs/2112.10752, 2021.