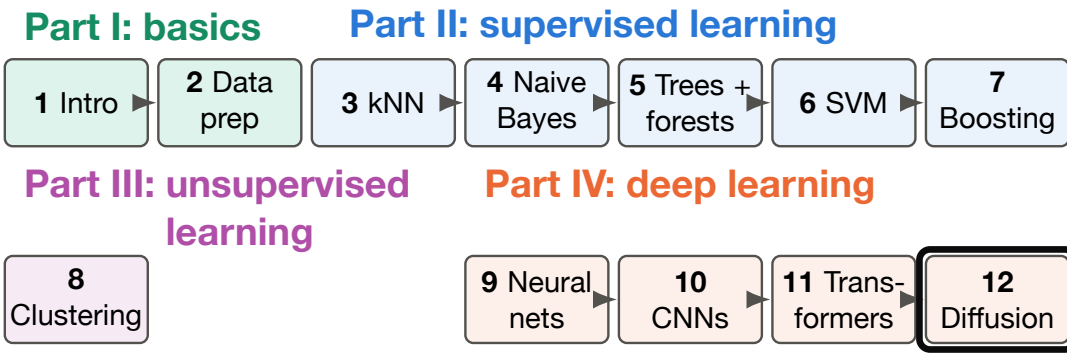
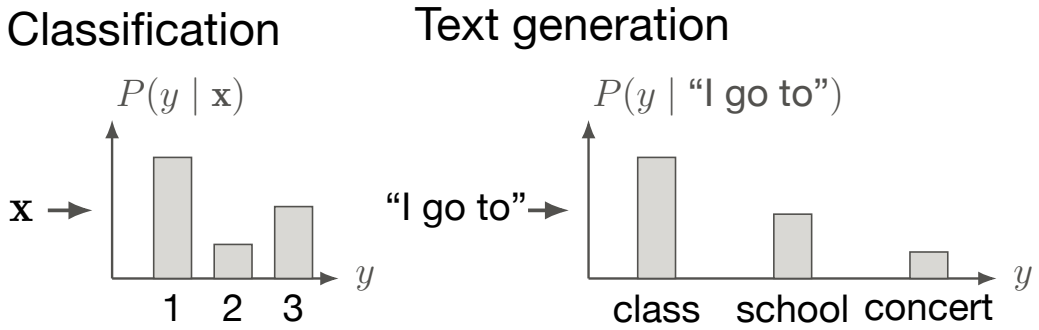
Diffusion models

229352 Lecture 12: diffusion models

Course map


Part I: basics
Part II: supervised learning
1 Intro
2 Data prep
3 kNN
4 Naive Bayes
5 Trees + forests
6 SVM
7 Boosting
Part III: unsupervised learning
Part IV: deep learning
8 Clustering
9 Neural nets
10 CNNs
11 Trans-formers
12 Diffusion

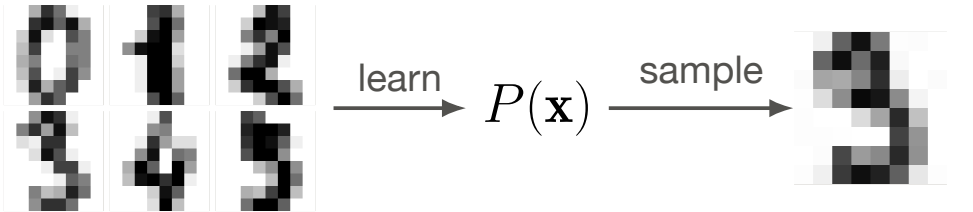
Review of supervised learning



Both models give a probability distribution of y given the input.

Text generation then samples the next word y from it.

Generative models

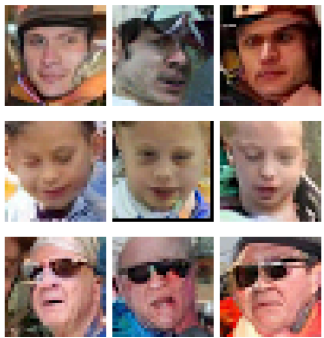


$\mathbf{x}$: an image, written as the vector of its pixel values.

A **generative model** learns the distribution $P(\mathbf{x})$ of the training images, then draws new images from it.

Progress in image generation (1)

PixelCNN, 2016



van den Oord et al., NeurIPS 2016, Fig. 4

StyleGAN, 2019



Karras, Laine and Aila, CVPR 2019, Fig. 3

Progress in image generation (2)



Imagen, 2022

Text prompt: “A photo of a Corgi dog riding a bike in Times Square. It is wearing sunglasses and a beach hat.”

Saharia et al., NeurIPS 2022; image from imagen.research.google

Progress in image generation (3)

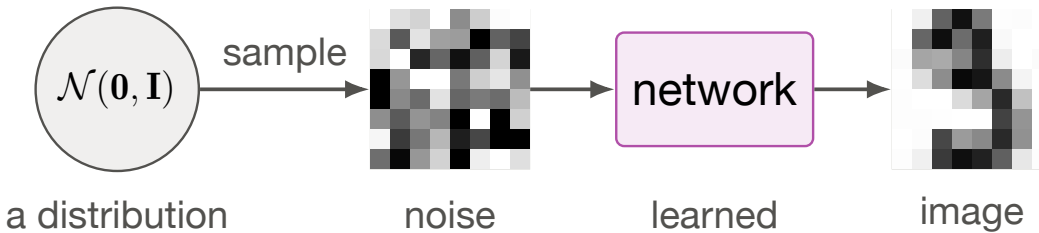


Latent diffusion, 2022

Text prompt: “A painting of a squirrel eating a burger”

Rombach et al., CVPR 2022, supplement Fig. 14 (crop)

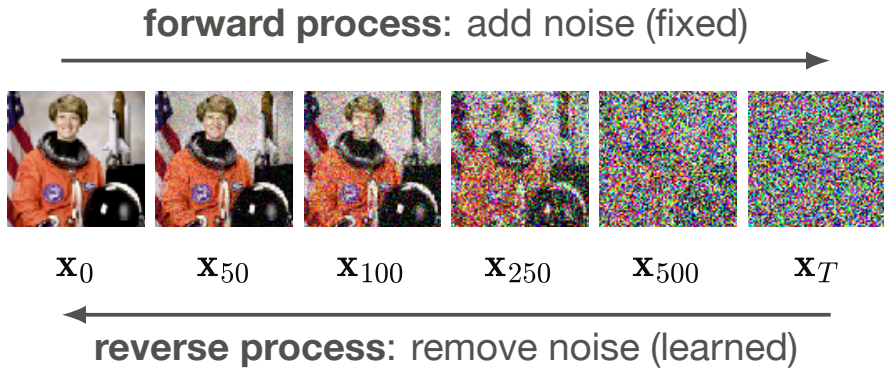
Generation as a transformation of noise



$\mathcal{N}(\mathbf{0}, \mathbf{I})$: independent Gaussian pixels, mean 0, variance 1.

A network, applied in many small steps, turns the noise into an image.

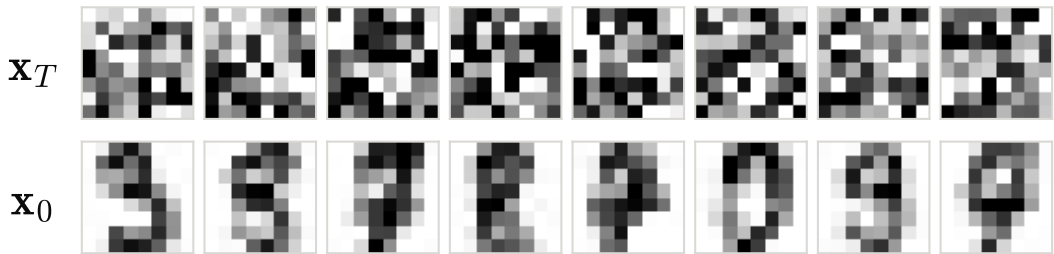
Denoising diffusion (DDPM)



$\mathbf{x}_t$: the image after t noise steps; $T = 1{,}000$ steps.

After Ho et al., NeurIPS 2020, Fig. 2. Photo: Eileen Collins (NASA), public domain.

Different noises give different images



start $\mathbf{x}_T$ and end $\mathbf{x}_0$ of eight sampling runs of our model

Different noise gives a different image.

The forward process: no training



$\mathbf{x}_0$: a training image; pixels scaled to $[-1, 1]$.

$\mathbf{x}_t$: the image after t steps, $t = 1, \ldots, T$; here $T = 1{,}000$.

No training: every step is a fixed formula.

One step of the forward process

$$\mathbf{x}_t = \sqrt{1-\beta_t}\,\mathbf{x}_{t-1} + \sqrt{\beta_t}\,\boldsymbol{\epsilon}_{t-1}, \qquad \boldsymbol{\epsilon}_{t-1} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

$\boldsymbol{\epsilon}_{t-1}$: new noise, independent of $\mathbf{x}_0$ and of the earlier noises.

β_t: the noise variance of step t; it rises linearly from $\beta_1 = 0.0001$ to $\beta_T = 0.02$.

As a distribution (q: the forward process):

$$q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) = \mathcal{N}\big(\sqrt{1-\beta_t}\,\mathbf{x}_{t-1},\ \beta_t\mathbf{I}\big)$$

Subscripts count steps; training image i would be $\mathbf{x}_0^{(i)}$.

Combining two steps

With $\alpha_t = 1 - \beta_t$, one step is

$$\mathbf{x}_t = \sqrt{\alpha_t}\,\mathbf{x}_{t-1} + \sqrt{1-\alpha_t}\,\boldsymbol{\epsilon}_{t-1}$$

The step before: $\mathbf{x}_{t-1} = \sqrt{\alpha_{t-1}}\,\mathbf{x}_{t-2} + \sqrt{1-\alpha_{t-1}}\,\boldsymbol{\epsilon}_{t-2}$.

Substitute it:

$$\mathbf{x}_t = \sqrt{\alpha_t\alpha_{t-1}}\,\mathbf{x}_{t-2} + \sqrt{\alpha_t(1-\alpha_{t-1})}\,\boldsymbol{\epsilon}_{t-2} + \sqrt{1-\alpha_t}\,\boldsymbol{\epsilon}_{t-1}$$

Two independent noise terms remain: we add them next.

Adding the two noises

The noise part: $\sqrt{\alpha_t(1-\alpha_{t-1})}\,\boldsymbol{\epsilon}_{t-2} + \sqrt{1-\alpha_t}\,\boldsymbol{\epsilon}_{t-1}$.

Independent Gaussian noises: their sum is Gaussian, with mean $\mathbf{0}$ and the sum of the variances:

$$\alpha_t(1-\alpha_{t-1}) + (1-\alpha_t) = 1 - \alpha_t\alpha_{t-1}$$

So the sum has the same distribution as $\sqrt{1-\alpha_t\alpha_{t-1}}\,\boldsymbol{\epsilon}$, with one noise $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$:

$$\mathbf{x}_t = \sqrt{\alpha_t\alpha_{t-1}}\,\mathbf{x}_{t-2} + \sqrt{1-\alpha_t\alpha_{t-1}}\,\boldsymbol{\epsilon}$$

A formula for any step

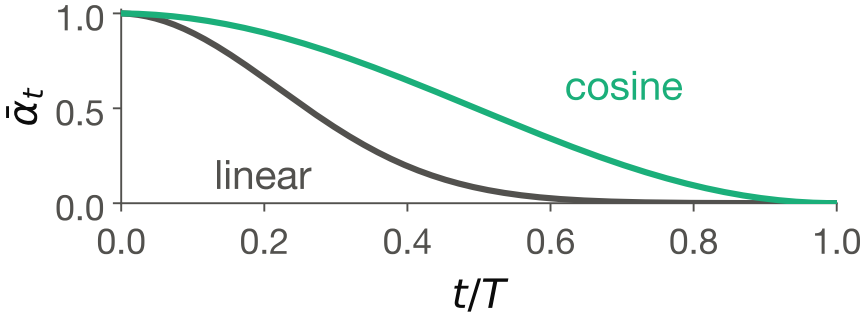
$$
\begin{aligned}
\mathbf{x}_t &= \sqrt{\alpha_t \alpha_{t-1}}\, \mathbf{x}_{t-2} + \sqrt{1 - \alpha_t \alpha_{t-1}}\, \boldsymbol{\epsilon} && \text{two steps} \\
&= \sqrt{\alpha_t \alpha_{t-1} \alpha_{t-2}}\, \mathbf{x}_{t-3} + \sqrt{1 - \alpha_t \alpha_{t-1} \alpha_{t-2}}\, \boldsymbol{\epsilon} && \text{repeat} \\
&= \sqrt{\alpha_t \cdots \alpha_1}\, \mathbf{x}_0 + \sqrt{1 - \alpha_t \cdots \alpha_1}\, \boldsymbol{\epsilon} && \text{down to } \mathbf{x}_0 \\
&= \sqrt{\bar{\alpha}_t}\, \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t}\, \boldsymbol{\epsilon} && \bar{\alpha}_t = \alpha_1 \alpha_2 \cdots \alpha_t
\end{aligned}
$$

Each "=" means equal distributions, with a new $\boldsymbol{\epsilon}$.

So $q(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}\big(\sqrt{\bar{\alpha}_t}\, \mathbf{x}_0,\ (1 - \bar{\alpha}_t)\mathbf{I}\big)$ for $t = 1, \ldots, T$.

We set $\bar{\alpha}_0 = 1$, so the formula also holds for $t = 0$.

The noise schedule

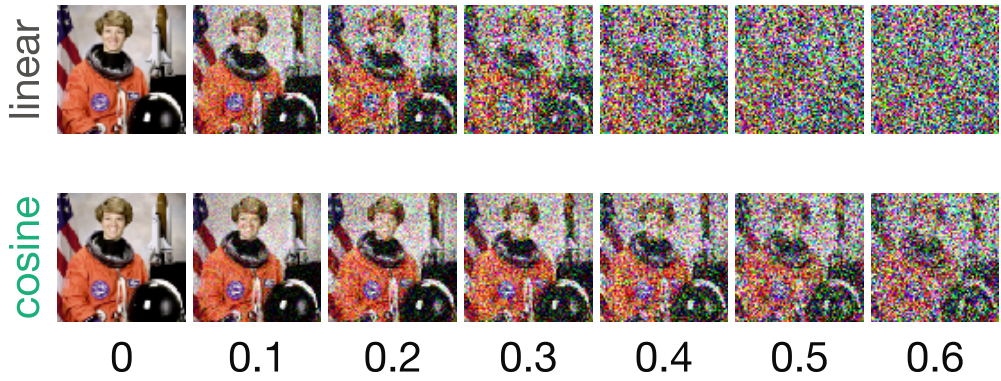


linear: as in DDPM; cosine: $\bar{\alpha}_t$ decreases more slowly

linear: $\bar{\alpha}_t < 0.01$ from $t = 674$ on, the last 33% of the steps

Cosine schedule: Nichol and Dhariwal, ICML 2021, eq. 17, $s = 0.008$

Two schedules on the photo



columns: t/T; the same noise ϵ in both rows

linear: almost gone at $t/T = 0.6$; cosine: still visible

Summary: the forward process

- The forward process adds a little Gaussian noise in each of T steps: $\mathbf{x}_t = \sqrt{1-\beta_t}\,\mathbf{x}_{t-1} + \sqrt{\beta_t}\,\boldsymbol{\epsilon}_{t-1}$.
- The noise schedule $\beta_1, \ldots, \beta_T$ is not learned.
- With $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \alpha_1 \cdots \alpha_t$: $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\,\boldsymbol{\epsilon}$ for $t = 1, \ldots, T$.
- The schedule (linear or cosine) sets the speed of fading.

Checkpoint 1

Linear schedule, $T = 1{,}000$: $\bar{\alpha}_T = 4.0 \times 10^{-5}$

Write $\mathbf{x}_T$ using $\mathbf{x}_0$ and $\boldsymbol{\epsilon}$.

What does $\mathbf{x}_T$ look like?

Checkpoint 1

Linear schedule, $T = 1{,}000$: $\bar{\alpha}_T = 4.0 \times 10^{-5}$

Write $\mathbf{x}_T$ using $\mathbf{x}_0$ and $\boldsymbol{\epsilon}$.

What does $\mathbf{x}_T$ look like?

$$\mathbf{x}_T = 0.0064\,\mathbf{x}_0 + 0.99998\,\boldsymbol{\epsilon}$$

almost pure noise $\mathcal{N}(\mathbf{0}, \mathbf{I})$: almost nothing of $\mathbf{x}_0$ is left

The reverse process

To generate, we need the reverse step, from $\mathbf{x}_t$ to $\mathbf{x}_{t-1}$.

$q(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$ would need the distribution of all images, which we do not know.

With $\mathbf{x}_0$ known: can we compute $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$?

Yes, by Bayes' theorem (Lecture 4), with $\mathbf{x}_0$ given in every term:

$$q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) = \frac{q(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{x}_0)\; q(\mathbf{x}_{t-1} \mid \mathbf{x}_0)}{q(\mathbf{x}_t \mid \mathbf{x}_0)}$$

With $\mathbf{x}_0$ given, every factor is a known Gaussian.

Deriving $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$ (part 1)

Given $\mathbf{x}_{t-1}$, the step to $\mathbf{x}_t$ does not use $\mathbf{x}_0$ (each step uses only the step before):

$$q(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{x}_0) = q(\mathbf{x}_t \mid \mathbf{x}_{t-1})$$

All three factors are Gaussians we know:

$$q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) = \mathcal{N}\big(\sqrt{\alpha_t}\,\mathbf{x}_{t-1},\ (1-\alpha_t)\mathbf{I}\big) \qquad \text{one step}$$

$$q(\mathbf{x}_{t-1} \mid \mathbf{x}_0) = \mathcal{N}\big(\sqrt{\bar{\alpha}_{t-1}}\,\mathbf{x}_0,\ (1-\bar{\alpha}_{t-1})\mathbf{I}\big) \qquad \text{from } \mathbf{x}_0 \text{ directly}$$

$$q(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}\big(\sqrt{\bar{\alpha}_t}\,\mathbf{x}_0,\ (1-\bar{\alpha}_t)\mathbf{I}\big) \qquad \text{from } \mathbf{x}_0 \text{ directly}$$

Deriving $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$ (part 2)

Look at the right side as a function of $\mathbf{x}_{t-1}$:

each: a constant times $\exp$(quadratic in $\mathbf{x}_{t-1}$)

$$q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) = \frac{\overbrace{q(\mathbf{x}_t \mid \mathbf{x}_{t-1})\, q(\mathbf{x}_{t-1} \mid \mathbf{x}_0)}}{\underbrace{q(\mathbf{x}_t \mid \mathbf{x}_0)}}$$

does not depend on $\mathbf{x}_{t-1}$

Multiply the two exponentials and complete the square in $\mathbf{x}_{t-1}$:

the result is a Gaussian density in $\mathbf{x}_{t-1}$.

Deriving $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$ (part 3)

In $\mathbf{x}_{t-1}$, each factor is a constant times $\exp\big(-\frac{\lambda}{2}\|\mathbf{x}_{t-1} - \mathbf{m}\|^2\big)$, with precision λ (1/variance) and center $\mathbf{m}$:

$$q(\mathbf{x}_t \mid \mathbf{x}_{t-1}): \quad \lambda = \frac{\alpha_t}{\beta_t}, \qquad \mathbf{m} = \frac{\mathbf{x}_t}{\sqrt{\alpha_t}}$$

$$q(\mathbf{x}_{t-1} \mid \mathbf{x}_0): \quad \lambda = \frac{1}{1-\bar{\alpha}_{t-1}}, \qquad \mathbf{m} = \sqrt{\bar{\alpha}_{t-1}}\,\mathbf{x}_0$$

The product's precision $1/\sigma_t^2$ is the sum (with $\alpha_t\bar{\alpha}_{t-1} = \bar{\alpha}_t$, $\alpha_t + \beta_t = 1$):

$$\frac{1}{\sigma_t^2} = \frac{\alpha_t}{\beta_t} + \frac{1}{1-\bar{\alpha}_{t-1}} = \frac{\alpha_t - \bar{\alpha}_t + \beta_t}{\beta_t\,(1-\bar{\alpha}_{t-1})} = \frac{1-\bar{\alpha}_t}{\beta_t\,(1-\bar{\alpha}_{t-1})}$$

Deriving $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$ (part 4)

$\tilde{\boldsymbol{\mu}}_t$ is the precision-weighted average of the centers:

$$\begin{aligned}\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0) &= \sigma_t^2 \Big(\frac{\sqrt{\alpha_t}}{\beta_t}\, \mathbf{x}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}}{1-\bar{\alpha}_{t-1}}\, \mathbf{x}_0 \Big) \\ &= \frac{\sqrt{\bar{\alpha}_{t-1}}\,\beta_t}{1-\bar{\alpha}_t}\, \mathbf{x}_0 + \frac{\sqrt{\alpha_t}\,(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\, \mathbf{x}_t\end{aligned}$$

So $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}\big(\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0),\ \sigma_t^2 \mathbf{I}\big)$, with $\sigma_t^2 = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\,\beta_t$ (σ_t^2 does not depend on $\mathbf{x}_t$ or $\mathbf{x}_0$).

At $t = 1$: $\bar{\alpha}_0 = 1$, so $\tilde{\boldsymbol{\mu}}_1 = \mathbf{x}_0$ and $\sigma_1^2 = 0$.

Reverse steps when $\mathbf{x}_0$ is known



$\mathbf{x}_T$ $\mathbf{x}_{500}$ $\mathbf{x}_{250}$ $\mathbf{x}_{100}$ $\mathbf{x}_{50}$ $\mathbf{x}_0$

start: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$; each step draws from $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$

the photo reappears; the last step gives $\mathbf{x}_0$ exactly ($\sigma_1 = 0$)

The mean written with the noise

From $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\,\boldsymbol{\epsilon}$:

$$\mathbf{x}_0 = \frac{1}{\sqrt{\bar{\alpha}_t}}\left(\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\,\boldsymbol{\epsilon}\right)$$

Substitute this $\mathbf{x}_0$ into $\tilde{\boldsymbol{\mu}}_t$ (the coefficients: next slide):

$$\tilde{\boldsymbol{\mu}}_t = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\,\boldsymbol{\epsilon}\right)$$

In words: remove the scaled noise from $\mathbf{x}_t$, then rescale.

The coefficients of the noise form

Use $\sqrt{\bar{\alpha}_t} = \sqrt{\alpha_t}\,\sqrt{\bar{\alpha}_{t-1}}$ and $\beta_t + \alpha_t = 1$.

of $\mathbf{x}_t$: $$\frac{\sqrt{\bar{\alpha}_{t-1}}\,\beta_t}{(1-\bar{\alpha}_t)\sqrt{\bar{\alpha}_t}} + \frac{\sqrt{\alpha_t}\,(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t} = \frac{\beta_t + \alpha_t - \bar{\alpha}_t}{\sqrt{\alpha_t}\,(1-\bar{\alpha}_t)} = \frac{1}{\sqrt{\alpha_t}}$$

of $\boldsymbol{\epsilon}$: $$-\frac{\sqrt{\bar{\alpha}_{t-1}}\,\beta_t}{1-\bar{\alpha}_t} \cdot \frac{\sqrt{1-\bar{\alpha}_t}}{\sqrt{\bar{\alpha}_t}} = -\frac{\beta_t}{\sqrt{\alpha_t}\,\sqrt{1-\bar{\alpha}_t}}$$

What's next?

- We can compute $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$ exactly.
- But it needs $\mathbf{x}_0$. To generate a new image, we start from noise $\mathbf{x}_T$: there is no $\mathbf{x}_0$.
- So we train a network, with weights and biases θ, to give $p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) \approx q(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$: the reverse step averaged over the images $\mathbf{x}_0$ that could have led to $\mathbf{x}_t$.
- Training compares p_θ with $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$ for random training images $\mathbf{x}_0$.

The learned reverse process

Start from noise:

$$p(\mathbf{x}_T) = \mathcal{N}(\mathbf{0}, \mathbf{I})$$

Each reverse step: a Gaussian with a learned mean:

$$p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) = \mathcal{N}\big(\boldsymbol{\mu}_\theta(\mathbf{x}_t, t),\ \sigma_t^2 \mathbf{I}\big)$$

θ: all weights and biases of the network (w in Lecture 9).

We choose σ_t^2 as in q; only the mean is learned.

The network predicts the noise

In training, $\mathbf{x}_0$ and $\boldsymbol{\epsilon}$ are known, so the target mean is

$$\tilde{\boldsymbol{\mu}}_t = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\,\boldsymbol{\epsilon}\right)$$

When we generate, only $\mathbf{x}_t$ is known, and $\mathbf{x}_t$ does not fix $\boldsymbol{\epsilon}$.

So a network $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$, the **denoiser**, predicts the noise: the average noise among the images that could give $\mathbf{x}_t$.

$$\boldsymbol{\mu}_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\,\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right)$$

Mean error and noise error

Subtract $\boldsymbol{\mu}_\theta(\mathbf{x}_t, t)$ from $\tilde{\boldsymbol{\mu}}_t$, then square:

$$\left\|\tilde{\boldsymbol{\mu}}_t - \boldsymbol{\mu}_\theta(\mathbf{x}_t, t)\right\|^2 = \frac{\beta_t^2}{\alpha_t(1-\bar{\alpha}_t)} \left\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right\|^2$$

Matching the mean is matching the noise, up to a factor that depends on t.

The loss of one image

The squared error (Lecture 9) between the noise and its prediction:

$$\ell = \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right\|^2, \qquad \mathbf{x}_t = \sqrt{\bar{\alpha}_t}\, \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t}\, \boldsymbol{\epsilon}$$

$\mathbf{x}_0$: a training image; t: a step; $\boldsymbol{\epsilon}$: a noise draw.

The simplified loss

The loss L is the average of ℓ over random training images $\mathbf{x}_0$, steps $t \in \{1, \ldots, T\}$ and noises $\boldsymbol{\epsilon}$:

$$L(\theta) = \mathbb{E}_{t, \mathbf{x}_0, \boldsymbol{\epsilon}} \Big[\Big\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta \big(\sqrt{\bar{\alpha}_t}\, \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t}\, \boldsymbol{\epsilon},\ t \big) \Big\|^2 \Big]$$

The training algorithm

Repeat a fixed number of times (or until L stops decreasing):

1. take a random training image $\mathbf{x}_0$
2. draw a step t from $\{1, \ldots, T\}$, all equally likely
3. draw noise $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
4. compute $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t}\,\boldsymbol{\epsilon}$
5. take a gradient step on $\ell = \|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\|^2$

Variables in the code

code	meaning
X	the $n = 1{,}797$ images, one per row
T	the number of steps, $T = 1{,}000$
`beta[t]`, `alpha[t]`	$\beta_t,\ \alpha_t$
`abar[t]`, `sigma[t]`	$\bar{\alpha}_t,\ \sigma_t$; `abar[0]` $= 1$
`model`	the network $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$
`gen`	random numbers with a fixed seed
`n_epochs`	2,000 epochs
`n_updates`	30,000 gradient steps

Training in practice

As in Lecture 9, each step uses a minibatch of images, each with its own t and $\boldsymbol{\epsilon}$, and sets $\theta \leftarrow \theta - \eta\, g$, with g the gradient of the average ℓ. Our code uses Adam.

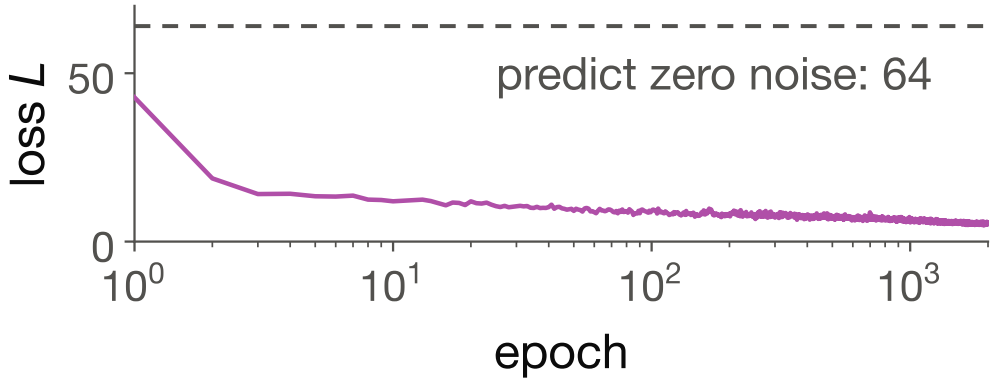
```
model = EpsNet()
opt = torch.optim.Adam(model.parameters(), lr=1e-3)
# learning-rate schedule (Lecture 9), not the noise schedule
lr_sched = torch.optim.lr_scheduler.CosineAnnealingLR(
    opt, n_updates)
losses = []
```

The training loop in PyTorch

```
for epoch in range(n_epochs):
    for i in torch.randperm(n, generator=gen).split(128):
        x0 = X[i]                              # minibatch
        t = torch.randint(1, T + 1, i.shape, generator=gen)
        eps = torch.randn(x0.shape, generator=gen)  # noise
        ab = abar[t, None]                     # abar_t
        xt = ab.sqrt() * x0 + (1 - ab).sqrt() * eps
        loss = ((eps - model(xt, t)) ** 2).sum(1).mean()
        opt.zero_grad()
        loss.backward()                        # backpropagation
        opt.step()                             # Adam
        lr_sched.step()                        # learning rate
        losses.append(loss.item())
```

`.sum(1)`: ℓ (64 pixels); `.mean()`: minibatch average

Training our model



1,797 images; linear schedule with $T = 1{,}000$; 2,000 epochs

L, the average ℓ of each epoch, falls from 42.8 to 5.2

The sampling algorithm

1. Draw noise: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2. For $t = T, T-1, \ldots, 1$:

   $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$ ($\sigma_1 = 0$: $\mathbf{z}$ is not used)

   $$\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\,\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right) + \sigma_t \mathbf{z}$$
3. Return $\mathbf{x}_0$

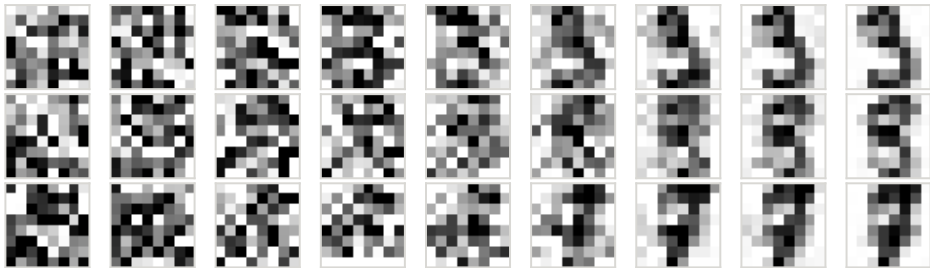
Each step draws $\mathbf{x}_{t-1}$ from $p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) = \mathcal{N}\big(\boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \sigma_t^2 \mathbf{I}\big)$.

The sampling loop in PyTorch

```
with torch.no_grad():
    x = torch.randn(100, 64, generator=gen)  # x_T
    xs = [x]
    for t in range(T, 0, -1):  # t = T, ..., 1
        eps_hat = model(x, torch.full((100,), t))
        mu = x - beta[t] / (1 - abar[t]).sqrt() * eps_hat
        mu = mu / alpha[t].sqrt()  # mu_theta(x_t, t)
        if t > 1:
            z = torch.randn(x.shape, generator=gen)
        else:
            z = 0
        x = mu + sigma[t] * z  # x_{t-1}
        xs.append(x)
```

100 images at once; `eps_hat`: $\epsilon_\theta(\mathbf{x}_t, t)$; `xs`: every $\mathbf{x}_t$.

Reverse steps of our model



three of our 100 samples, $\mathbf{x}_T$ to $\mathbf{x}_0$; clipped to $[-1, 1]$

the digit appears in the last few hundred steps

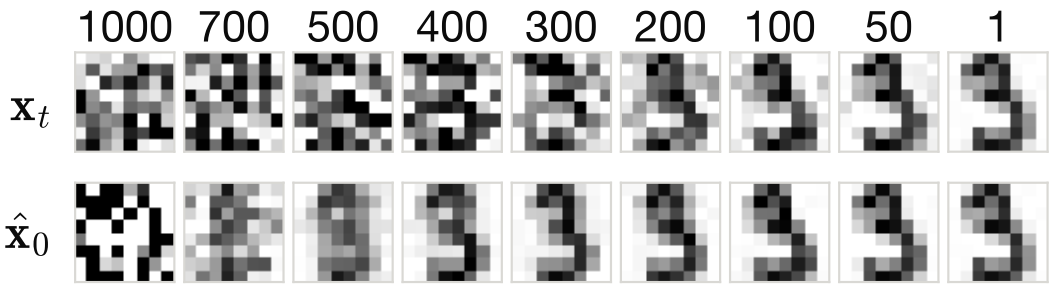
The network's estimate of $\mathbf{x}_0$

Solve $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\,\boldsymbol{\epsilon}$ for $\mathbf{x}_0$, with the predicted noise in place of $\boldsymbol{\epsilon}$:

$$\hat{\mathbf{x}}_0 = \frac{1}{\sqrt{\bar{\alpha}_t}}\Big(\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\;\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\Big)$$

$\hat{\mathbf{x}}_0$: the network's estimate, at step t, of the final image.

Our model's estimate of $\mathbf{x}_0$ at each step



first of our 100 samples; columns: t; clipped to $[-1, 1]$

$\hat{\mathbf{x}}_0$ shows the digit many steps before $\mathbf{x}_t$ does

Summary: training and sampling

- $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$: a Gaussian (Bayes' theorem) that needs $\mathbf{x}_0$.
- The learned step $p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) = \mathcal{N}\big(\boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \sigma_t^2 \mathbf{I}\big)$: a network $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$ predicts the noise.
- Training: random $\mathbf{x}_0$, t and $\boldsymbol{\epsilon}$; make $\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\|^2$ small.
- Sampling: from noise $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, take T reverse steps.

Checkpoint 2

$T = 1{,}000$ steps.

How many times does ϵ_θ run

(a) to sample one image,

(b) for one image in a training step?

Checkpoint 2

$T = 1{,}000$ steps.

How many times does ϵ_θ run

(a) to sample one image,

(b) for one image in a training step?

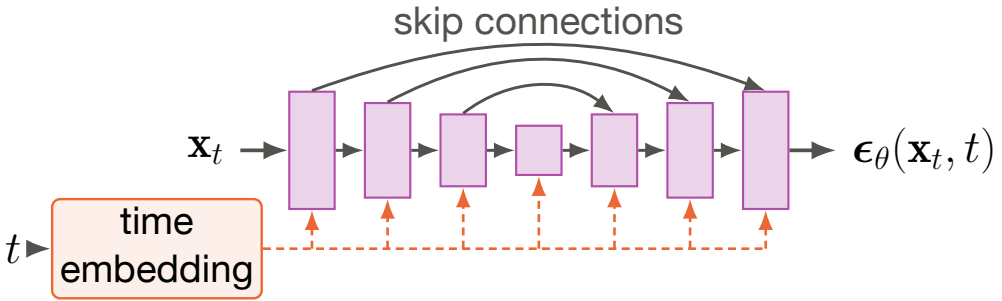
(a) 1,000 times, once per step

(b) once, at one random step t

How do we build $\epsilon_\theta(\mathbf{x}_t, t)$?

- Input: the noisy image $\mathbf{x}_t$ and the step t. Output: a noise image of the same size as $\mathbf{x}_t$.
- The network needs t: the amount of noise in $\mathbf{x}_t$ depends on t.
- For images: a **U-Net**, a convolutional network (Lecture 10).
- For our 8×8 digits: a fully connected network (Lecture 9).

A U-Net with a time embedding



Down: smaller images; up: back to the size of $\mathbf{x}_t$.

Skip connections copy details from down to up.

The **time embedding** of t is added in each (residual) block.

Ronneberger et al., MICCAI 2015 (U-Net); Ho et al., NeurIPS 2020, App. B

Our network for 8×8 digits (1)

```
class EpsNet(nn.Module):
    def __init__(self, width=512, nf=32):
        super().__init__()
        k = torch.arange(nf)
        self.freq = torch.exp(-math.log(1000) * k / nf)
        self.temb = nn.Sequential(  # time embedding
            nn.Linear(2 * nf, width), nn.SiLU(),
            nn.Linear(width, width))
        self.inp = nn.Linear(64, width)
        self.hid = nn.ModuleList(
            [nn.Linear(width, width) for _ in range(3)])
        self.tp = nn.ModuleList(
            [nn.Linear(width, width) for _ in range(3)])
        self.out = nn.Linear(width, 64)
```

`width`: nodes per layer; 1,937,984 weights and biases

Our network for 8×8 digits (2)

```
    def forward(self, x, t):  # eps_theta(x_t, t)
        a = t[:, None] * self.freq  # (batch, nf)
        e = self.temb(torch.cat([a.sin(), a.cos()], 1))
        h = self.inp(x)
        for layer, tp in zip(self.hid, self.tp):
            h = h + layer(F.silu(h + tp(e)))
        return self.out(F.silu(h))
```

a: t times 32 frequencies; e: the time embedding of t.

`h = h + layer(...)`: a residual connection.

`h + tp(e)`: the time embedding, added in each layer.

SiLU$(r) = r/(1 + e^{-r})$: an activation function.

Generating images from text

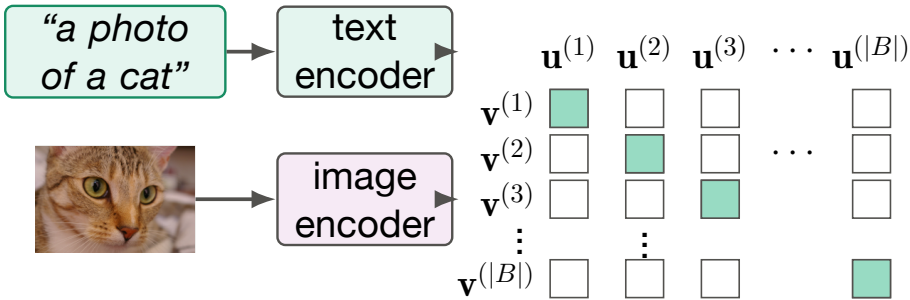
“a photograph of an astronaut riding a horse”

model



Generated with Stable Diffusion XL in NightCafe Studio by VulcanSphere, 2023; Wikimedia Commons, “Astronaut Riding a Horse (SDXL).jpg”, CC0

Matching text and images: CLIP



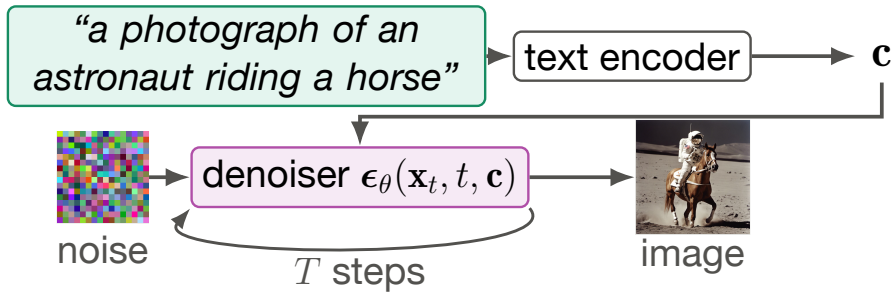
$\mathbf{v}^{(i)}$, $\mathbf{u}^{(j)}$: image i, caption j of a minibatch B of pairs.

Cell (i, j): $s_{\cos}(\mathbf{v}^{(i)}, \mathbf{u}^{(j)})$, the cosine similarity (Lecture 3).

Training makes the diagonal (matching pairs) the most similar.

After Radford et al., ICML 2021, Fig. 1. Photo: scikit-image “chelsea”, CC0.

Text-to-image diffusion



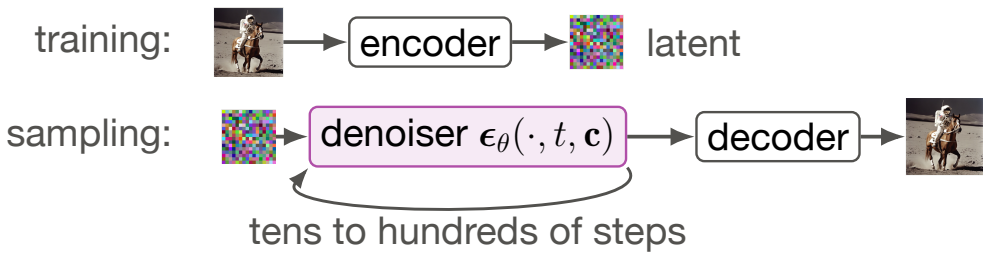
A text encoder, for example CLIP’s, gives vectors $\mathbf{c}$.

The denoiser, a U-Net as in DDPM, gets $\mathbf{c}$ in every step.

Training uses the same loss, on (image, caption) pairs.

Rombach et al., CVPR 2022; Saharia et al., NeurIPS 2022; image: VulcanSphere, CC0

Latent diffusion



Encoder: image $\rightarrow$ **latent**, a small array.

Decoder: latent $\rightarrow$ image.

Diffusion runs on latents: far fewer numbers, so it is cheaper.

Rombach et al., CVPR 2022; image: VulcanSphere, CC0

Try the demo

huggingface.co/spaces/black-forest-labs/FLUX.2-klein-9B

Prompt:

“A Corgi astronaut floating in a vast empty space”



Generated by the lecturer with the FLUX.2 [Klein] 9B demo

Summary: the network, text to image

- For images, $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$ is a U-Net with a time embedding.
- CLIP learns text and image vectors that are similar for matching pairs.
- Text to image: the prompt becomes $\mathbf{c}$, and the denoiser uses $\mathbf{c}$ in every step.
- Latent diffusion denoises a small latent array, then decodes it into an image.