Lecture 1: Introduction to diffusion models

Chenyang Yuan

January 6, 2025

Outline

- Applications of diffusion models
- Course logistics
- What are diffusion models?
- Training
- Sampling

Text-to-image generation

Image credit: SDXL: Improving Latent Diffusion Models for High-Resolution Image Synthesis https://arxiv.org/pdf/2307.01952

Sketch-to-Image: coarse-grained control

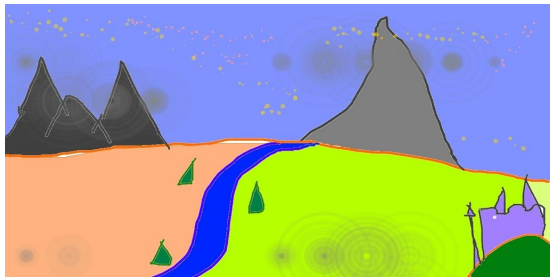


Image credit: https://github.com/CompVis/stable-diffusion

ControlNet: fine-grained control

Figure 1: Controlling Stable Diffusion with learned conditions. ControlNet allows users to add conditions like Canny edges (top), human pose (bottom), *etc*., to control the image generation of large pretrained diffusion models. The default results use the prompt "a high-quality, detailed, and professional image". Users can optionally give prompts like the "chef in kitchen".

Image credit: Adding Conditional Control to Text-to-Image Diffusion Models https://arxiv.org/pdf/2302.05543

Image editing and composition

Figure 1. With just a few images (typically 3-5) of a subject (left), *DreamBooth*—our AI-powered photo booth—can generate a myriad of images of the subject in different contexts (right), using the guidance of a text prompt. The results exhibit natural interactions with the environment, as well as novel articulations and variation in lighting conditions, all while maintaining high fidelity to the key visual features of the subject.

Image credit: DreamBooth: Fine Tuning Text-to-Image Diffusion Models for Subject-Driven Generation https://arxiv.org/pdf/2208.12242

Diffusion “prior” for image restoration

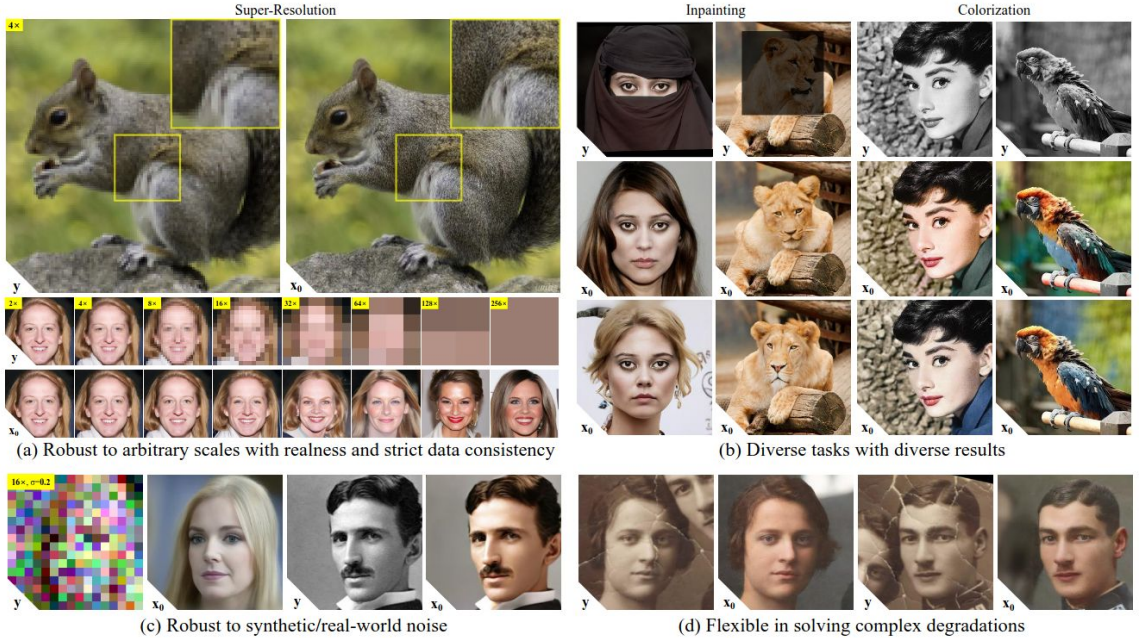


Image credit: Zero-Shot Image Restoration Using Denoising Diffusion Null-Space Model https://arxiv.org/pdf/2212.00490

Generating visual illusions

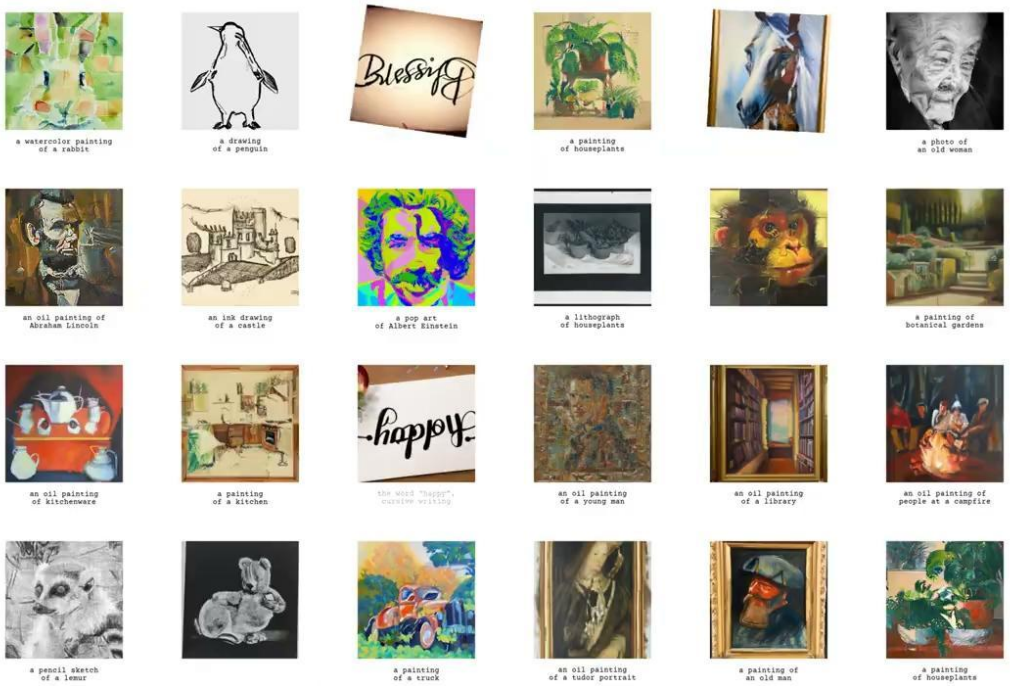
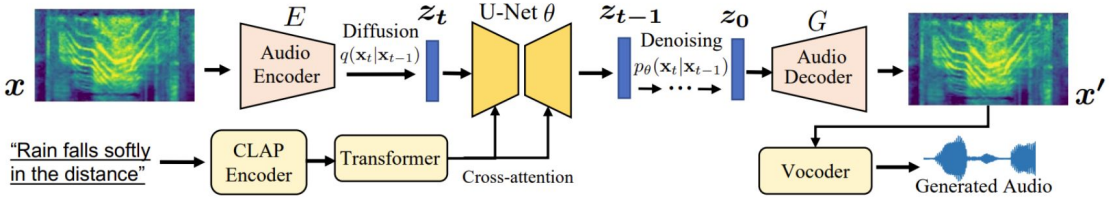


Image credit: Visual anagrams https://dangeng.github.io/visual_anagrams/

Text-to-audio generation



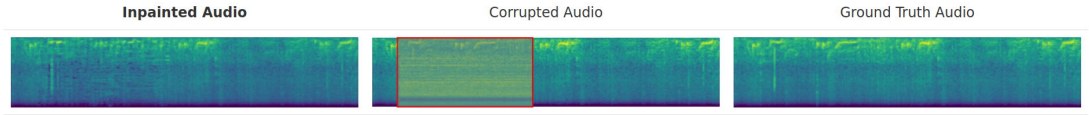


Image credit: Make-An-Audio: Text-To-Audio Generation with Prompt-Enhanced Diffusion Models https://text-to-audio.github.io/

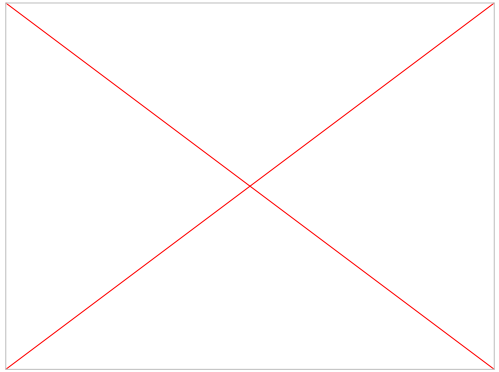
Video generation

A girl is running across a beach and holding a kite. She's wearing jean shorts and a yellow t-shirt. The sun is shining down.

Video credit: https://ai.meta.com/research/movie-gen/

Low-angle tracking shot, 18mm lens. The car drifts, leaving trails of light and tire smoke, creating a visually striking and abstract composition. The camera tracks low, capturing the sleek, olive green muscle car as it approaches a corner. As the car executes a dramatic drift, the shot becomes more stylized. The spinning wheels and billowing tire smoke, illuminated by the surrounding city lights and lens flare, create streaks of light and color against the dark asphalt. The cityscape – yellow cabs, neon signs, and pedestrians – becomes a blurred, abstract backdrop. Volumetric lighting adds depth and atmosphere, transforming the scene into a visually striking composition of motion, light, and urban energy.

Video credit: https://deepmind.google/technologies/veo/veo-2/



In an ornate, historical hall, a massive tidal wave peaks and begins to crash. Two surfers, seizing the moment, skillfully navigate the face of the wave.

Video credit: https://openai.com/index/video-generation-models-as-world-simulators/

Novel view synthesis

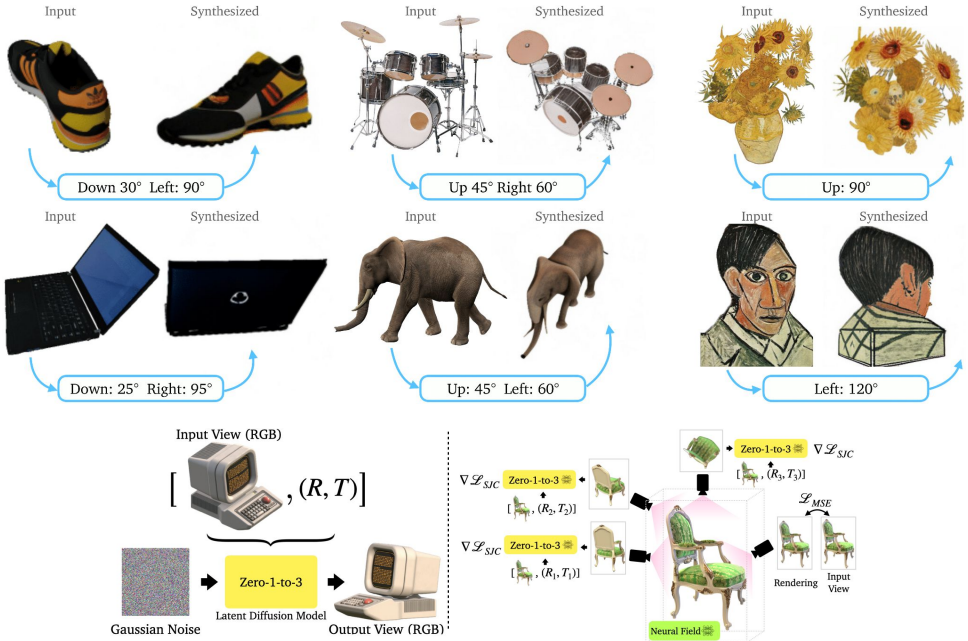


Image credit: Zero-1-to-3: Zero-shot One Image to 3D Object https://zero123.cs.columbia.edu/

3D shape generation

Video credit: DreamFusion https://dreamfusion3d.github.io/

Video credit: SV3D https://sv3d.github.io/

Video game simulation

Image/video credit: Diffusion Models Are Real-Time Game Engines
https://gamengen.github.io/

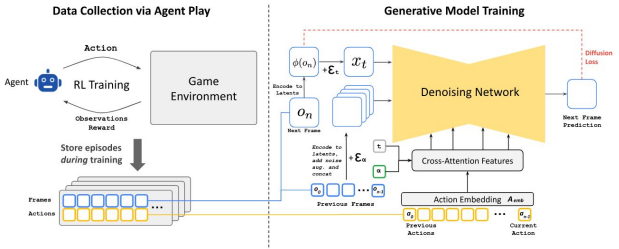
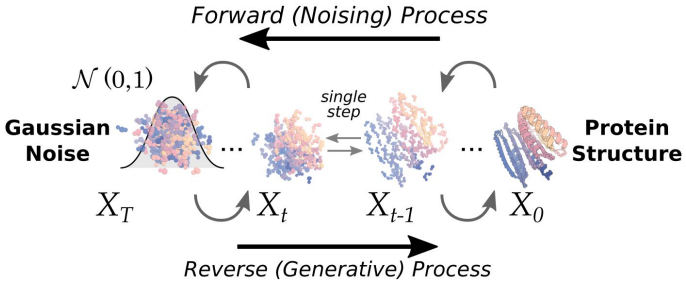
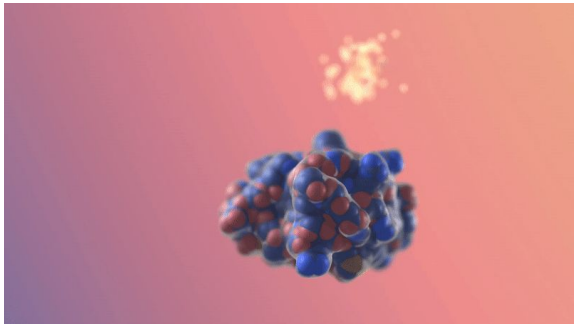
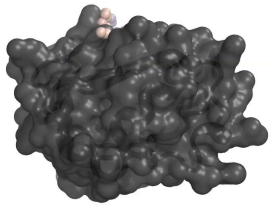


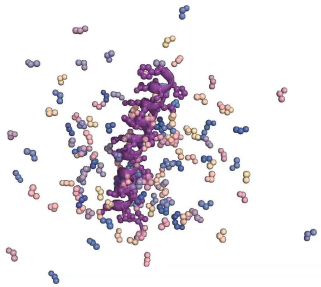
Figure 3: **GameNGen** method overview. v-prediction details are omitted for brevity.

Protein synthesis



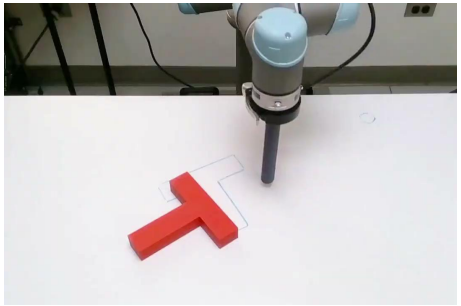


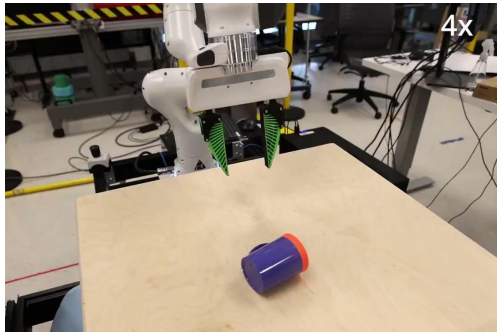




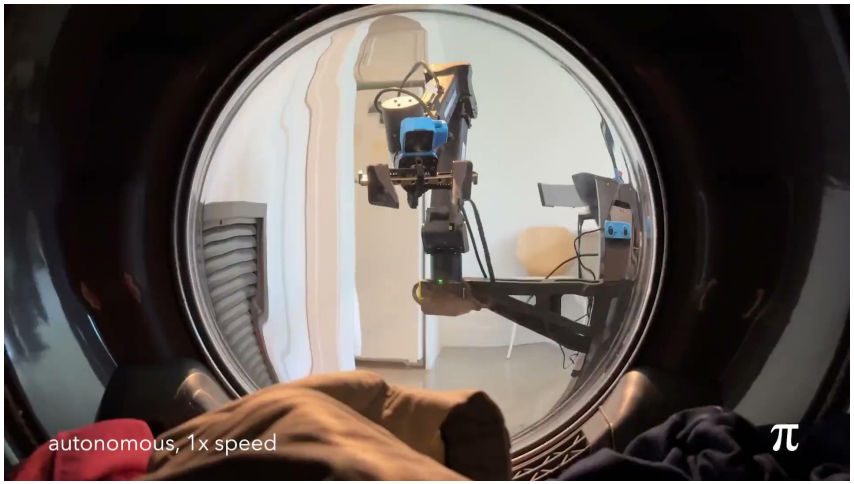
Image/video credit: RFDiffusion https://www.bakerlab.org

Imitation learning in robotics





Video credit: https://diffusion-policy.cs.columbia.edu/



Video credit: https://www.physicalintelligence.company/blog/pi0

Course outline

1. Introduction (Chenyang)
2. Interpretations / perspectives (Cole)
3. Conditioning and guidance (Boyuan)
4. Generalization (Chris)
5. Advanced applications (Artem)
6. Additional topics, summary and conclusion (Chenyang)

Course Staff

Chenyang Yuan

Cole Becker

Boyuan Chen

Artem Lukoianov

Christopher Scarvelis

Course logistics

Lectures: M/W/F 10am-11am, Jan 6 -17, 32-144

Video will be recorded and posted on webpage

Webpage: https://6-s183-diffusion.github.io/

All course materials will be posted here

Piazza: https://piazza.com/class/m5b9cuy61u412k

Office hours: Thu 1/9 and Tue 1/14

Email: 6.s183-diffusion-instructors@mit.edu

Mailing list: https://mailman.mit.edu/mailman/listinfo/6.s183-diffusion

Course grading

Grading with be P/D/F, 18 points and above required to pass

Attendance/participation (6 points)

- Complete short survey sent after each lecture to receive credit

Problem Sets (12 points)

- Two problem sets (Pset 1 released today, due Friday 1/10)
- Graded on completion

Mini Project (4 points)

- Open-ended project (see website for project ideas)
- Report due end of class

What are diffusion models?

Inspired by diffusion processes

(Brownian motion)

Forward process adds noise to data

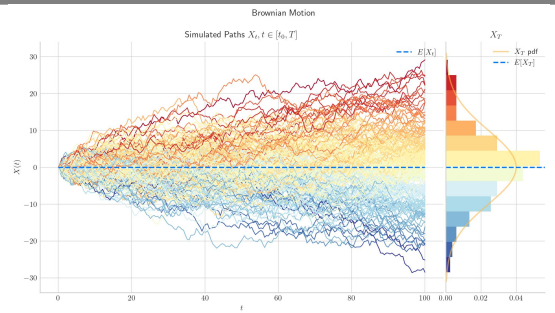


Image credit:
https://quantgirluk.github.io/Understanding-Quantitative-Finance/brownian_motion.html

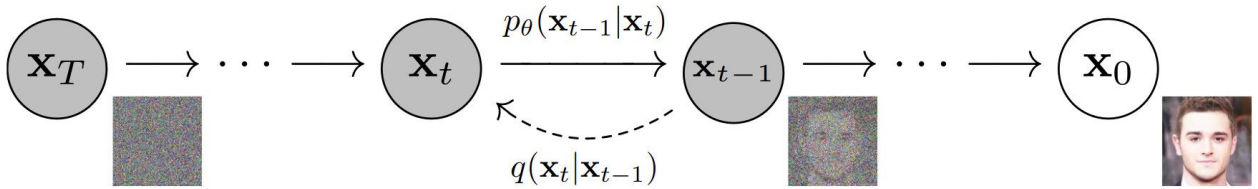


Image credit: Denoising Diffusion Probabilistic Models https://arxiv.org/pdf/2006.11239

Diffusion models learn the reverse process to recover data from noise

Training diffusion models

Denoising diffusion models estimate a noise vector $\epsilon \in \mathbb{R}^n$ from a given noise level $\sigma > 0$ and noisy input $x_\sigma \in \mathbb{R}^n$ such that for some x_0 in the data manifold $\mathcal{K}$,

$$x_\sigma \approx x_0 + \sigma\epsilon$$

A *denoiser* $\epsilon_\theta : \mathbb{R}^n \times \mathbb{R}_+ \to \mathbb{R}^n$ is learned by minimizing

$$L(\theta) := \mathbf{E}_{x_0,\sigma,\epsilon} \left\| \epsilon_\theta(x_0 + \sigma\epsilon, \sigma) - \epsilon \right\|^2$$

- x_0 is sampled from training data
- σ is sampled from *training noise schedule* (more on this later)
- ϵ is sampled from $N(0, I_n)$ (i.i.d. Gaussian)

Training noise schedules

What levels of noise to use?

In practice, noise levels range from $\sigma_{min} = 0.01$ to $\sigma_{max}=100$

Sampled uniformly from a *noise schedule* (typically 100-1000 discrete noise levels)

$$\sigma \sim \{\sigma_1, \ldots, \sigma_{1000}\}$$

Typically follow a log-scale $\lambda = \log(\sigma)$

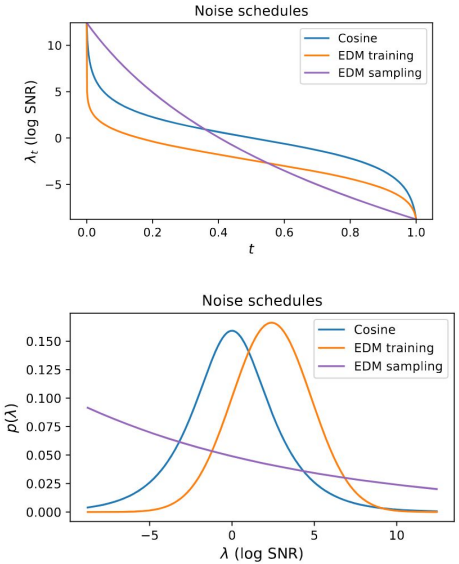
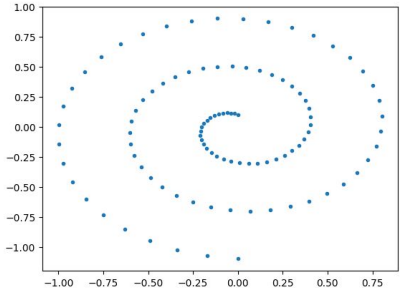


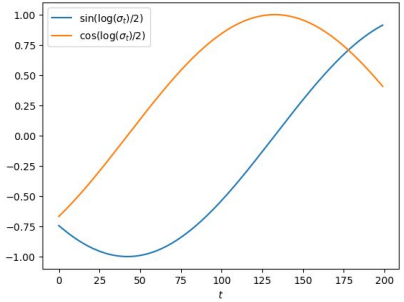
Image credit: Understanding Diffusion Objectives as the ELBO with Simple Data Augmentation https://arxiv.org/pdf/2303.00848

Toy model

2D spiral dataset



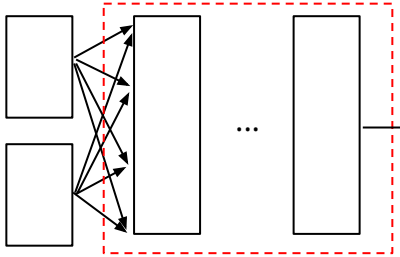
2D noise-level embedding



Add noise

Noise-level embedding

Fully Connected MLP



Predict Noise

```python
def get_sigma_embeds(sigma):
    sigma = sigma.unsqueeze(1)
    return torch.cat([torch.sin(torch.log(sigma)/2),
                      torch.cos(torch.log(sigma)/2)], dim=1)

class TimeInputMLP(nn.Module):
    def __init__(self, dim, hidden_dims):
        super().__init__()
        layers = []
        for in_dim, out_dim in pairwise((dim + 2,) + hidden_dims):
            layers.extend([nn.Linear(in_dim, out_dim), nn.GELU()])
        layers.append(nn.Linear(hidden_dims[-1], dim))
        self.net = nn.Sequential(*layers)
        self.input_dims = (dim,)

    def rand_input(self, batchsize):
        return torch.randn((batchsize,) + self.input_dims)

    def forward(self, x, sigma):
        sigma_embeds = get_sigma_embeds(sigma)         # shape: b x 2
        nn_input = torch.cat([x, sigma_embeds], dim=1) # shape: b x (dim + 2)
        return self.net(nn_input)

model = TimeInputMLP(dim=2, hidden_dims=(16,128,128,128,128,16))
```

Commonly used model architectures

Convolutional U-Nets

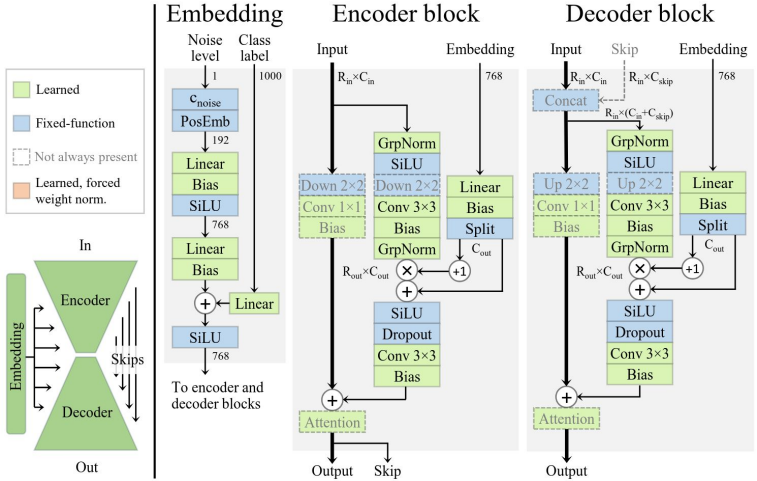


Image credit: Analyzing and Improving the Training Dynamics of Diffusion Models
https://arxiv.org/pdf/2312.02696

Patch-wise Transformers

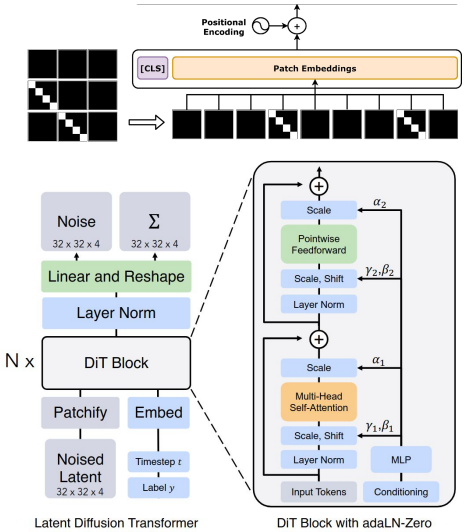
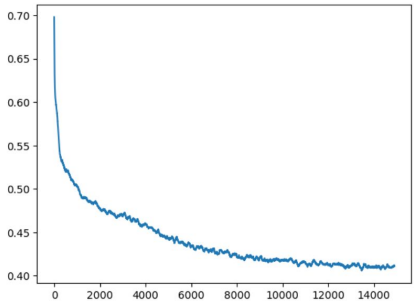


Image credit: Scalable Diffusion Models with Transformers
https://arxiv.org/pdf/2212.09748

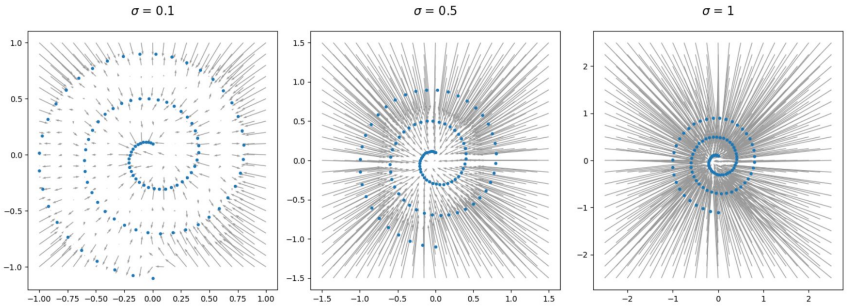
Training the toy model

```python
def training_loop(loader  : DataLoader,
                  model   : nn.Module,
                  schedule: Schedule,
                  epochs  : int = 10000):
    optimizer = torch.optim.Adam(model.parameters())
    for _ in range(epochs):
        for x0 in loader:
            optimizer.zero_grad()
            sigma, eps = generate_train_sample(x0, schedule)
            eps_hat = model(x0 + sigma * eps, sigma)
            loss = nn.MSELoss()(eps_hat, eps)
            optimizer.backward(loss)
            optimizer.step()
```



Training loss over 15000 epochs, smoothed with moving average

The learned denoiser $\epsilon_\theta(x, \sigma)$ can be visualized as a vector field parameterized by the noise level σ, by plotting $x - \sigma\epsilon_\theta(x, \sigma)$ for different x and levels of σ.



Plot of predicted $\hat{x}_0 = x - \sigma\epsilon_\theta(x, \sigma)$ for different x and σ

The model predicts expected x0 given x and σ

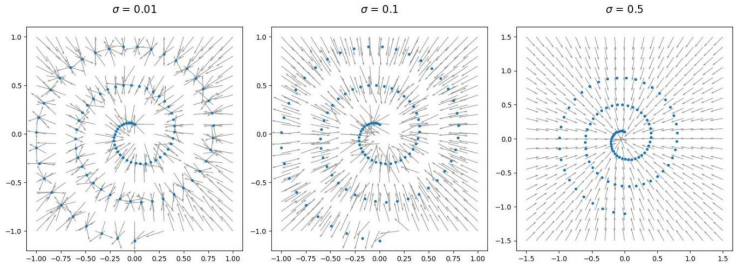
What did the model learn?

The *ideal denoiser* is the minimizer of training loss, a function of data distribution $\mathcal{K}$ and noise level σ

$$\epsilon^* := \arg\min_{\epsilon_\theta} \mathbf{E}_{x_0,\sigma,\epsilon} \left\| \epsilon_\theta(x_0 + \sigma\epsilon, \sigma) - \epsilon \right\|^2$$

For finite $\mathcal{K}$, there is a closed-form solution:

$$\epsilon^*(x_\sigma, \sigma) = \frac{\sum_{x_0 \in \mathcal{K}} (x_\sigma - x_0) \exp(-\left\| x_\sigma - x_0 \right\|^2 / 2\sigma^2)}{\sigma \sum_{x_0 \in \mathcal{K}} \exp(-\left\| x_\sigma - x_0 \right\|^2 / 2\sigma^2)}$$



Plot of direction of $\epsilon^*(x, \sigma)$ for different x and σ

What did the model learn?

The *distance function* $\mathrm{dist}_{\mathcal{K}} : \mathbb{R}^n \to \mathbb{R}$ to a set $\mathcal{K} \subseteq \mathbb{R}^n$, is defined via

$$\mathrm{dist}_{\mathcal{K}}(x) := \inf\{\|x - x_0\| : x_0 \in \mathcal{K}\}$$

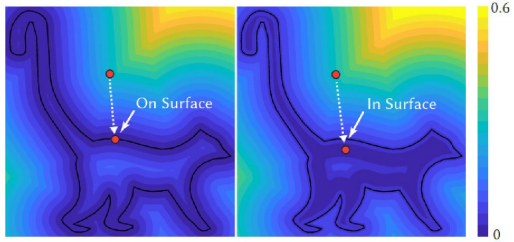
Intuitively, $\mathrm{proj}_{\mathcal{K}}(x) - x$ is the direction of steepest descent (i.e. neg. gradient) of $\mathrm{dist}_{\mathcal{K}}(x)$.

$$\nabla \tfrac{1}{2}\mathrm{dist}_{\mathcal{K}}(x)^2 = x - \mathrm{proj}_{\mathcal{K}}(x)$$

Distance function is not differentiable everywhere, $\nabla \mathrm{dist}_{\mathcal{K}}(x)$ is not continuous, thus hard to learn!

Solution: σ-smoothed distance function

$$\mathrm{dist}_{\mathcal{K}}(x, \sigma) := \underset{x_0 \in \mathcal{K}}{\mathrm{softmin}_{\sigma^2}} \|x_0 - x\|^2 = -\sigma^2 \log\left(\sum_{x_0 \in \mathcal{K}} \exp\left(-\frac{\|x_0 - x\|^2}{2\sigma^2}\right)\right)$$



Smoothed distance function has continuous gradients

Image credit: Fast Evaluation of Smooth Distance Constraints on Co-Dimensional Geometry https://arxiv.org/pdf/2108.10480

Theorem

For all $\sigma > 0$ and $x \in \mathbb{R}^n$, we have

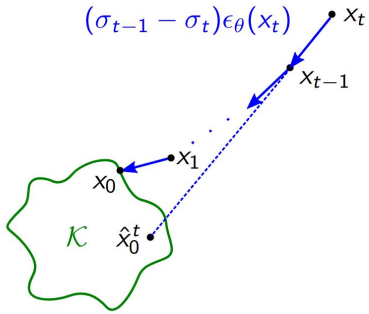
$$\frac{1}{2}\nabla_x \mathrm{dist}^2_{\mathcal{K}}(x, \sigma) = \sigma \epsilon^*(x, \sigma).$$

Denoiser model learns the gradient of a σ-smoothed distance function

Sampling from diffusion models

Given noisy x_σ and noise level σ, the learned denoiser $\epsilon_\theta(x_\sigma, \sigma)$ estimates

$$x_0 \approx \hat{x}_0(x_\sigma, \sigma) := x_\sigma - \sigma\epsilon_\theta(x_\sigma, \sigma).$$



Sampling algorithms (e.g. DDIM) construct a sequence $\hat{x}_0^t := \hat{x}_0(x_t, \sigma_t)$ of estimates from a sequence of points x_t using the update:

$$x_{t-1} = x_t + (\sigma_{t-1} - \sigma_t)\epsilon_\theta(x_t, \sigma_t)$$

Implementing the sampler

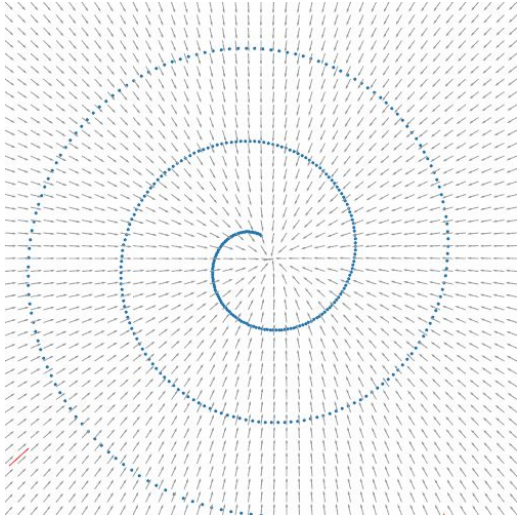
Algorithm 1 DDIM sampler

for $t = N, \ldots, 1$ **do**

$$x_{t-1} \leftarrow x_t + (\sigma_{t-1} - \sigma_t)\epsilon_\theta(x_t, \sigma_t)$$

return x_0

```python
@torch.no_grad()
def samples_ddim(model      : torch.nn.Module,
                 sigmas     : torch.FloatTensor,
                 xt         : Optional[torch.FloatTensor] = None,
                 batchsize  : int = 1):
    model.eval()
    accelerator = Accelerator()
    xt = model.rand_input(batchsize).to(accelerator.device) * sigmas[0]
    for i, (sig, sig_prev) in enumerate(pairwise(sigmas)):
        xt = xt - (sig - sig_prev) * model.predict_eps(xt, sig.to(xt))
    return xt
```



Probabilistic sampling

Deterministic (DDIM) update:

$$x_{t-1} = x_t + (\sigma_{t-1} - \sigma_t)\epsilon_\theta(x_t, \sigma_t)$$

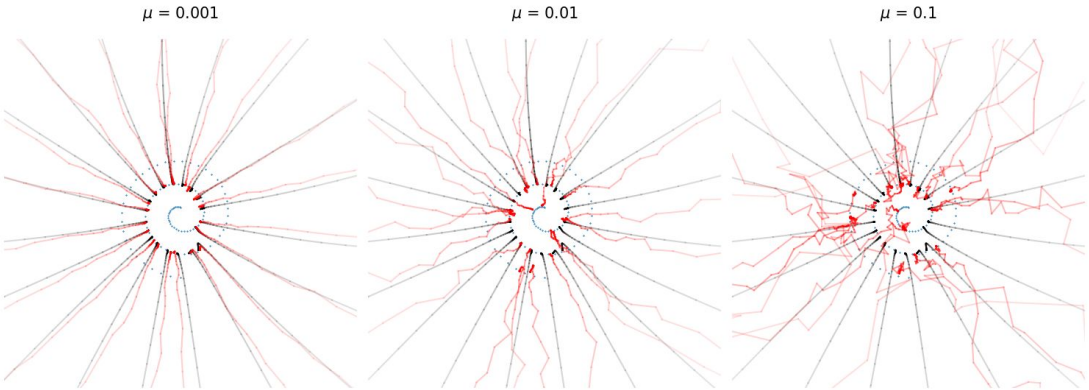
Probabilistic (DDPM) update:

$$x_{t-1} = x_t + (\sigma_{t'} - \sigma_t)\epsilon_\theta(x_t, \sigma_t) + \eta w_t$$

Where $w_t \sim \mathcal{N}(0, I)$, $\sigma_{t-1} = \sigma_t^{\mu}\sigma_{t'}^{1-\mu}$ and $\eta = \sqrt{\sigma_{t-1}^2 - \sigma_{t'}^2}$.

(Matches norm of update in expectation if $\mathbb{E}\,\|w_t\|^2 = \|\epsilon_\theta(x_t, \sigma_t)\|^2$)

Note: $0 \leq \mu < 1$ and $\sigma_{t'} < \sigma_{t-1} < \sigma_t$



Speeding up sampling

Algorithm 1 DDIM sampler

for $t = N, \dots, 1$ **do**

$x_{t-1} \leftarrow x_t + (\sigma_{t-1} - \sigma_t)\epsilon_\theta(x_t, \sigma_t)$

return x_0

Algorithm 2 Our sampler

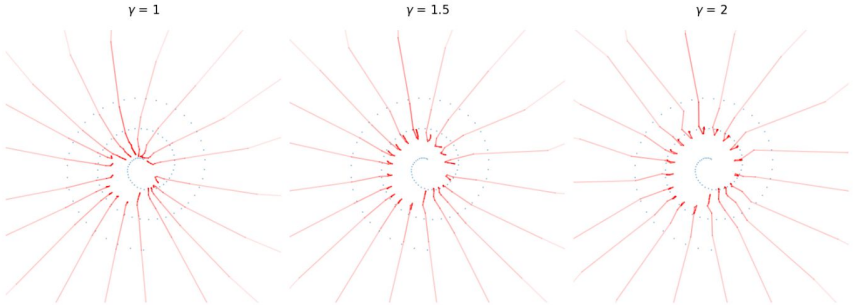
$x_{N-1} \leftarrow x_N + (\sigma_{N-1} - \sigma_N)\epsilon_\theta(x_N, \sigma_N)$

for $t = N-1, \dots, 1$ **do**

$\bar{\epsilon}_t \leftarrow 2\epsilon_\theta(x_t, \sigma_t) - \epsilon_\theta(x_{t+1}, \sigma_{t+1})$

$x_{t-1} \leftarrow x_t + (\sigma_{t-1} - \sigma_t)\bar{\epsilon}_t$

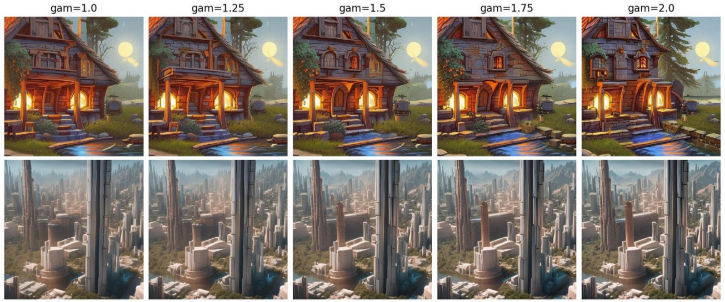
return x_0



Interpreting and Improving Diffusion Models from an Optimization Perspective https://arxiv.org/abs/2306.04848

Implementing the full sampler

```python
@torch.no_grad()
def samples(model      : nn.Module,
            sigmas     : torch.FloatTensor, # Iterable with N+1 values for N sampling steps
            gam        : float = 1.,        # Suggested to use gam >= 1
            mu         : float = 0.,        # Requires mu in [0, 1)
            xt         : Optional[torch.FloatTensor] = None,
            batchsize  : int = 1):
    xt = model.rand_input(batchsize) * sigmas[0]
    eps = None
    for i, (sig, sig_prev) in enumerate(pairwise(sigmas)):
        eps, eps_prev = model(xt, sig.to(xt)), eps
        eps_av = eps * gam + eps_prev * (1-gam)  if i > 0 else eps
        sig_p = (sig_prev/sig**mu)**(1/(1-mu)) # sig_prev == sig**mu sig_p**(1-mu)
        eta = (sig_prev**2 - sig_p**2).sqrt()
        xt = xt - (sig - sig_p) * eps_av + eta * model.rand_input(batchsize).to(xt)
        yield xt
```



Text-to-image samples using Stable Diffusion

Additional reading/resources

Blogpost: https://chenyang.co/diffusion.html

Code: https://github.com/yuanchenyang/smalldiffusion/

Elucidating the Design Space of Diffusion-Based Generative Models
https://arxiv.org/abs/2206.00364

Analyzing and Improving the Training Dynamics of Diffusion Models
https://arxiv.org/abs/2312.02696

See course website for more readings

Pset 1 released today, due Friday 1/10