CS492D: Diffusion Models and Their Applications

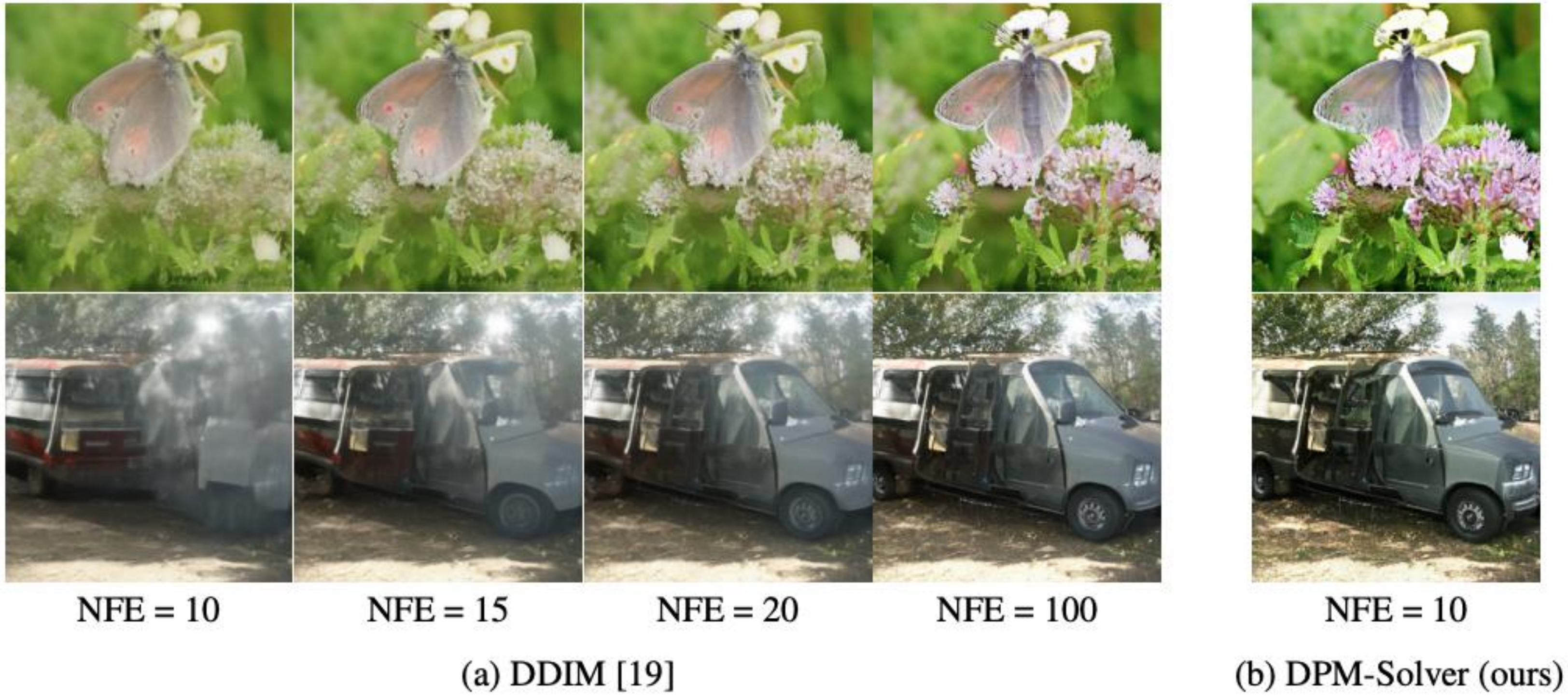
Assignment 6 Session

JUIL KOO

Fall 2024
KAIST

Introduction

In Assignment 6, we will implement **DPM-Solver**, a training-free approach to accelerate the reverse process using high-order ODE solvers.

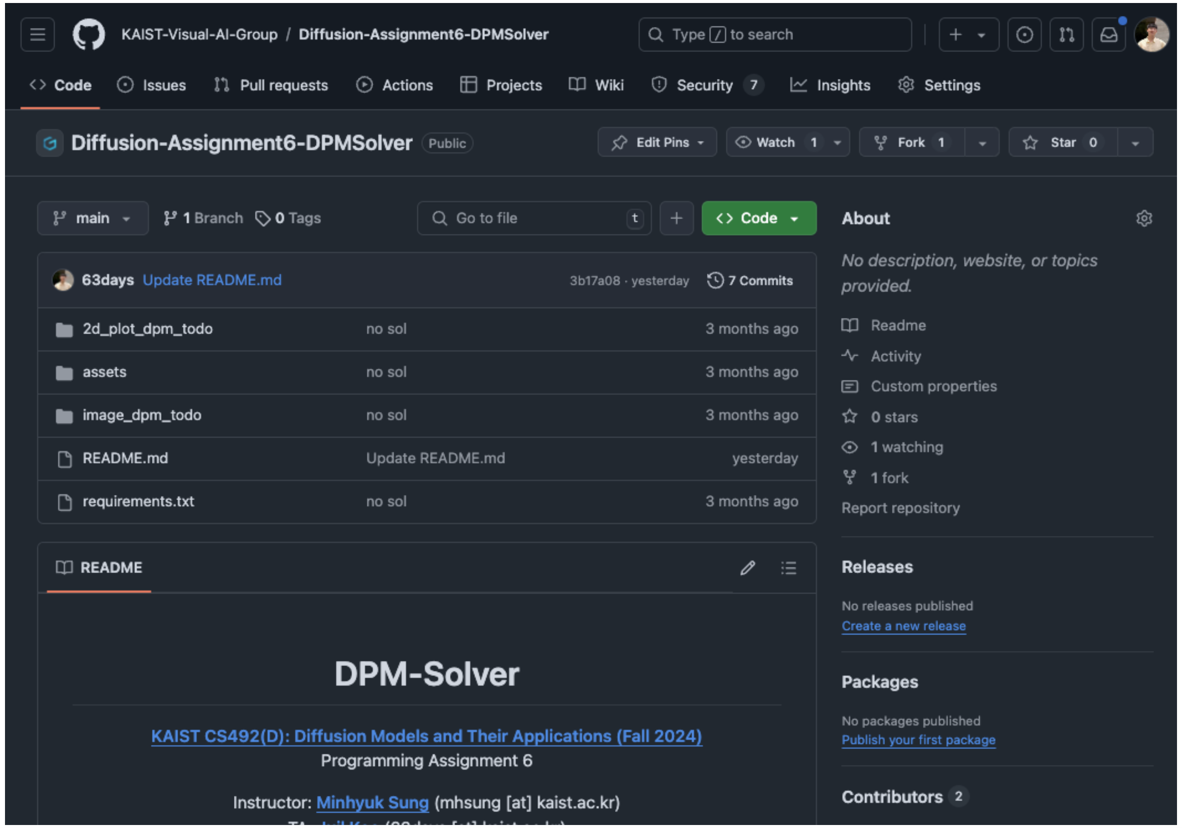


DPM-Solver, Lu *et al.*, NeurIPS 2022

Introduction

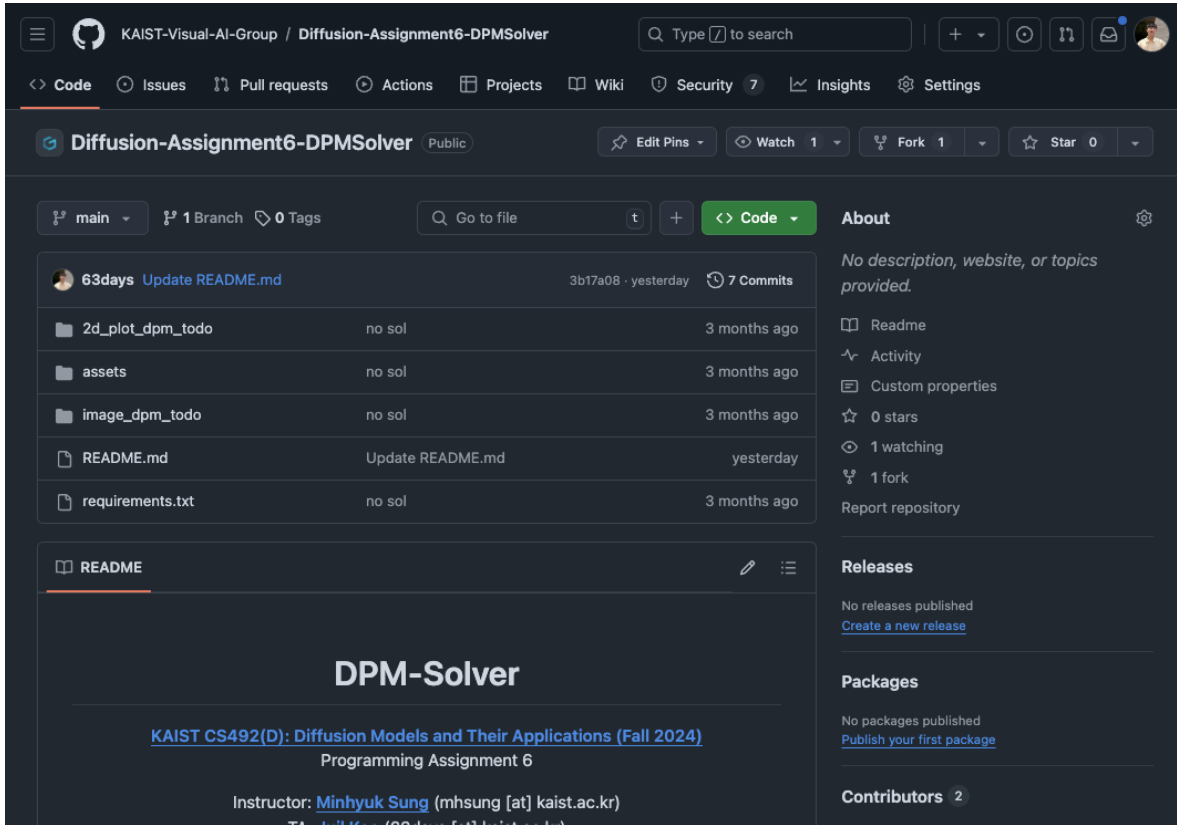
The skeleton code and instructions are available at:

https://github.com/KAIST-Visual-AI-Group/Diffusion-Assignment6-DPMSolver



Introduction

Assignment 6 builds on your implementation of Assignment 2 (DDIM-CFG).
Refer to the `README.md` for details on which files and models need to be copied from Assignment 2.



Important Notes

- Assignment 6 is due on **Dec 6th (Friday)**.
- Late submission will incur **20% penalty** for **each** late day!
- Please carefully check the README of each assignment.
- Missing items in your submission will also incur penalties.

Connectiontion Between DDPM and SDE

Forward Process: $q_{0t}(\boldsymbol{x}_t|\boldsymbol{x}_0) = \mathcal{N}(\boldsymbol{x}_t|\alpha(t)\boldsymbol{x}_0, \sigma^2(t)\mathbf{I})$

Corresponding SDE: $d\boldsymbol{x}_t = f(t)\boldsymbol{x}_t dt + g(t)d\boldsymbol{w}_t$,
where $f(t) = \frac{d \log \alpha(t)}{dt}$ and $\mathrm{g}^2(t) = \frac{d\sigma^2(t)}{dt} - 2\frac{d \log \alpha(t)}{dt}\sigma^2(t)$.

Continuous Forms of DDPM

Song *et al*. has shown the forward process has its equivalent *reverse* SDE and PF-ODE.

SDE of Forward Process: $d\boldsymbol{x}_t = f(t)\boldsymbol{x}_t dt + g(t)d\boldsymbol{w}_t$

SDE of Reverse Process: $d\boldsymbol{x}_t = [f(t)\boldsymbol{x}_t - g^2(t)\nabla_{\boldsymbol{x}} \log q_t(\boldsymbol{x}_t)]dt + g(t)d\bar{\boldsymbol{w}}_t$

PF-ODE of Reverse Process: $\frac{d\mathbf{x}_t}{dt} = f(t)\mathbf{x}_t - \frac{1}{2}g^2(t)\nabla_{\mathbf{x}} \log q_t\,(\mathbf{x}_t)$

$$= f(t)\mathbf{x}_t + \frac{g^2(t)}{2\sigma_t}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$$

Song *et al*., Score-Based Generative Modeling through Stochastic Differential Equations, ICLR 2021.

Exact Solutions of PF-ODEs

PF-ODE of Reverse Process: $\frac{d\boldsymbol{x}_t}{dt} = f(t)\boldsymbol{x}_t + \frac{g^2(t)}{2\sigma_t}\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t, t)$

The PF-ODE is a 1st-order semi-linear differential equation, whose solution can be computed. Starting from $s > t$, the solution at t is:

$$\boldsymbol{x}_t = \int_s^t \frac{d\boldsymbol{x}_\tau}{d\tau} d\tau = e^{\int_s^t f(\tau)d\tau}\boldsymbol{x}_s + \int_s^t \left(e^{\int_\tau^t f(r)dr} \frac{g^2(\tau)}{2\sigma_\tau} \boldsymbol{\epsilon}_\theta(\boldsymbol{x}_\tau, \tau) \right) d\tau \,.$$

Exact Solutions of PF-ODEs

Exact solution: $\boldsymbol{x}_t = \underbrace{e^{\int_s^t f(\tau)d\tau}\boldsymbol{x}_s}_{\text{Linear term}} + \underbrace{\int_s^t \left(e^{\int_\tau^t f(r)dr} \frac{g^2(\tau)}{2\sigma_\tau} \boldsymbol{\epsilon}_\theta(\boldsymbol{x}_\tau, \tau) \right) d\tau}_{\text{Non-linear term}}$

Exact Solutions of PF-ODEs

Exact solution: $\boldsymbol{x}_t = e^{\int_s^t f(\tau)d\tau}\boldsymbol{x}_s + \int_s^t \left(e^{\int_\tau^t f(r)dr} \frac{g^2(\tau)}{2\sigma_\tau} \boldsymbol{\epsilon}_\theta(\boldsymbol{x}_\tau, \tau) \right) d\tau$

The solution can be greatly simplified by introducing a new variable $\lambda_t \coloneqq \log\left(\frac{\alpha_t}{\sigma_t}\right)$:

$$\boldsymbol{x}_t = \frac{\alpha_t}{\alpha_s}\boldsymbol{x}_s - \alpha_t \int_s^t \left(\frac{d\lambda_\tau}{d\tau}\right) \frac{\sigma_\tau}{\alpha_\tau} \boldsymbol{\epsilon}_\theta(\boldsymbol{x}_\tau, \tau) d\tau\,.$$

Rewrite the solution again by "*change-of-variable*" for t → λ_t:

$\lambda_t = \lambda(t)$ is a strictly decreasing function of t.

$$x_t = \frac{\alpha_t}{\alpha_s}x_s - \alpha_t \int_{\lambda_s}^{\lambda_t} e^{-\lambda_\tau}\, \hat{\boldsymbol{\epsilon}}_\theta(\hat{\boldsymbol{x}}_{\lambda_\tau}, \lambda_\tau) d\lambda_\tau.$$

High-Order Solvers for PF-ODEs

Solution: $x_t = \frac{\alpha_t}{\alpha_s} x_s - \alpha_t \int_{\lambda_s}^{\lambda_t} e^{-\lambda}\, \hat{\boldsymbol{\epsilon}}_\theta(\hat{\boldsymbol{x}}_\lambda, \lambda) d\lambda$.

Approximate the integral term by Taylor expansion of $\hat{\boldsymbol{\epsilon}}_\theta(\hat{\boldsymbol{x}}_\lambda, \lambda)$:

$$\hat{\boldsymbol{\epsilon}}_\theta(\hat{\boldsymbol{x}}_\lambda, \lambda) = \sum_{k=0}^{n-1} \frac{(\lambda - \lambda_s)^k}{k!} \hat{\boldsymbol{\epsilon}}_\theta^{(k)}\left(\hat{\boldsymbol{x}}_{\lambda_s}, \lambda_s\right) + \mathcal{O}\left((\lambda - \lambda_s)^n\right).$$

n-th order derivative

$$\hat{\boldsymbol{\epsilon}}_\theta^{(n)}(\hat{\boldsymbol{x}}_\lambda, \lambda) := \frac{d^n \hat{\boldsymbol{\epsilon}}_\theta(\hat{\boldsymbol{x}}_\lambda, \lambda)}{d\lambda^n}$$

Feeding the Talyor expansion into the solution above results in:

$$x_t = \frac{\alpha_t}{\alpha_s} x_s - \alpha_t \sum_{k=0}^{n-1} \boxed{\hat{\boldsymbol{\epsilon}}_\theta^{(k)}\left(\hat{\boldsymbol{x}}_{\lambda_s}, \lambda_s\right)} \int_{\lambda_s}^{\lambda_t} e^{-\lambda} \frac{(\lambda - \lambda_s)^k}{k!} d\lambda$$

This term only needs to be approximated.

DPM-Solver-*n*

$$x_t = \frac{\alpha_t}{\alpha_{t+1}} x_{t+1} - \alpha_t \sum_{k=0}^{n-1} \hat{\boldsymbol{\epsilon}}_\theta^{(k)}\left(\hat{\boldsymbol{x}}_{\lambda_{t+1}}, \lambda_{t+1}\right) \int_{\lambda_{t+1}}^{\lambda_t} e^{-\lambda} \frac{(\lambda - \lambda_{t+1})^k}{n!} d\lambda$$

(For clarity, we now change the notation s to $t+1$.)

DPM-Solver-*n* is an approximation with the $(n-1)$-th order Taylor expansion.

DPM-Solver-1

$$x_t = \frac{\alpha_t}{\alpha_{t+1}} x_{t+1} - \alpha_t \sum_{k=0}^{n-1} \hat{\boldsymbol{\epsilon}}_\theta^{(k)}\left(\hat{\boldsymbol{x}}_{\lambda_{t+1}}, \lambda_{t+1}\right) \int_{\lambda_{t+1}}^{\lambda_t} e^{-\lambda} \frac{(\lambda - \lambda_{t+1})^k}{n!} d\lambda$$

$$\boldsymbol{x}_t = \frac{\alpha_t}{\alpha_{t+1}} \boldsymbol{x}_{t+1} - \sigma_t\left(e^{\lambda_t - \lambda_{t+1}} - 1\right) \boldsymbol{\epsilon}_\theta(\boldsymbol{x}_{t+1}, t+1)$$

$$= \underline{\frac{\alpha_t}{\alpha_{t+1}} \boldsymbol{x}_{t+1} - \alpha_t \left(\frac{\sigma_{t+1}}{\alpha_{t+1}} - \frac{\sigma_t}{\alpha_t}\right) \boldsymbol{\epsilon}_\theta(\boldsymbol{x}_{t+1}, t+1)}$$

$$\because \lambda_t \coloneqq \log\left(\frac{\alpha_t}{\sigma_t}\right)$$

DPM-Solver-1 is identical to DDIM.

DPM-Solver-2

1. Split the λ interval $(\lambda_t, \lambda_{t+1}]$ in half:

$$\tilde{\lambda} = \frac{\lambda_t + \lambda_{t+1}}{2}$$

2. Perform the first-order approximation for the first half:

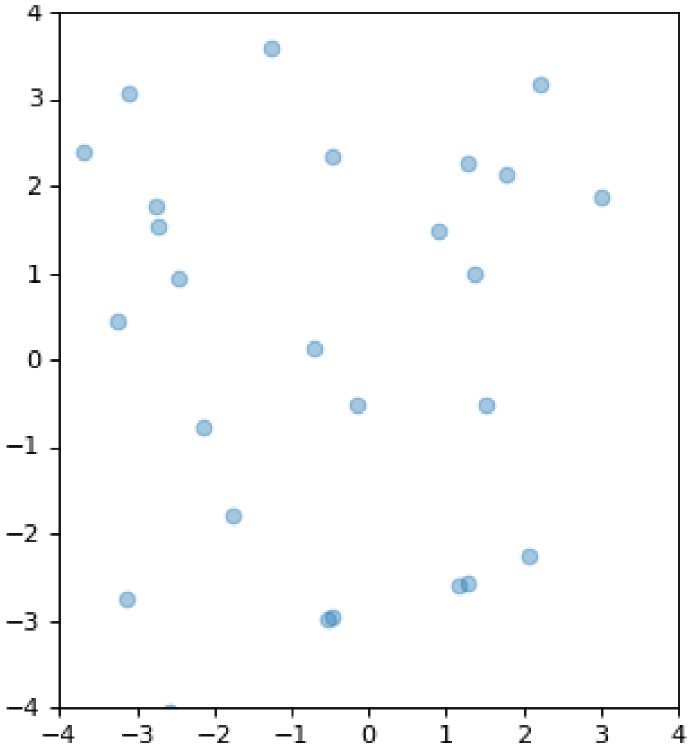
$$\boldsymbol{u} \leftarrow \frac{\alpha_{t_\lambda(\tilde{\lambda})}}{\alpha_{t+1}} \boldsymbol{x}_{t+1} - \sigma_{t_\lambda(\tilde{\lambda})} \left(e^{\frac{h_t}{2}} - 1\right) \hat{\boldsymbol{\varepsilon}}_\theta(\boldsymbol{x}_{t+1}, t+1)$$

3. Use the approximation to update $\hat{\boldsymbol{\varepsilon}}_\theta$ and compute the final $\boldsymbol{x}_t$:

$$\boldsymbol{x}_t \approx \frac{\alpha_t}{\alpha_{t+1}} \boldsymbol{x}_{t+1} - \sigma_t \left(e^{h_t} - 1\right) \hat{\boldsymbol{\varepsilon}}_\theta \left(\boldsymbol{u}, t_\lambda(\tilde{\lambda})\right)$$

Task 1

Implement DPM-Solver-1 for Swiss-Roll Data and 2D Images



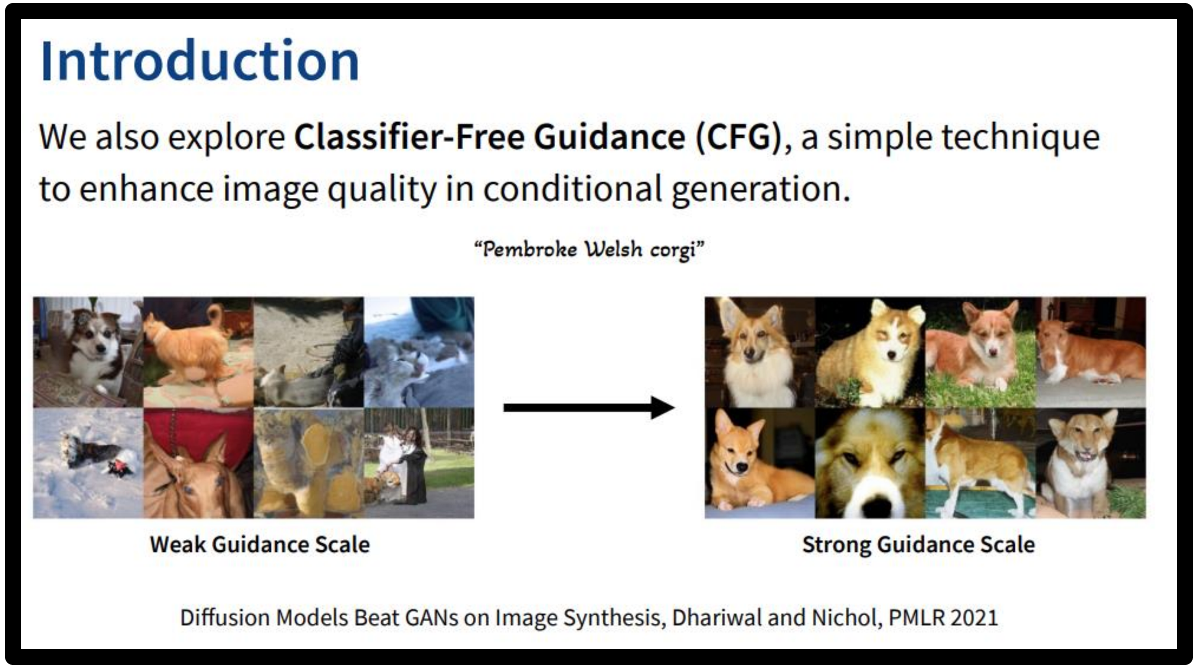
Task 1

You need to fill in the parts marked with TODO in `scheduler.py`.

```python
def first_order_step(self, x_s, s, t, eps_theta):
    """
    Implement Eq 4.1. in the DPM-Solver paper.
    Input:
        x_s (`torch.Tensor`): samples at timestep s.
        s (`torch.Tensor`): denoising starting timestep.
        t (`torch.Tensor`): denoising end timestep.
        eps_theta (`torch.Tensor`): noise prediction at s.
    Output:
        x_t (`torch.Tensor`): one step denoised sample.
    """
    assert torch.all(s > t), f"timestep s should be larger than timestep t"
    ######## TODO ########
    # DO NOT change the code outside this part.
    alpha_s = extract(self.dpm_alphas, s, x_s)
    x_t = x_s
    ######################
    return x_t
```

Task 1

For 2D image generation, use a model that you trained with a CFG setup from Assignment 2.



Task 2 (Optional)

Similarly, implement DPM-Solver-2 for Swiss-Roll Data and 2D Images.

You may want to access the noise prediction network's forward function inside a scheduler class.

```python
class DPMSolverScheduler(BaseScheduler):
    def __init__(self, num_train_timesteps, beta_1=1e-4, beta_T=0.02, mode="linear"):
        assert mode == "linear", f"only linear scheduling is supported."
        super().__init__(num_train_timesteps, beta_1, beta_T, mode)
        self._convert_notations_ddpm_to_dpm()
        # To access the model forward in high-order scheduling.
        self.net_forward_fn = None
```

```python
def build_dpm(config):
    network = SimpleNet(dim_in=2,
                        dim_out=2,
                        dim_hids=config["dim_hids"],
                        num_timesteps=config["num_diffusion_steps"]
                        )
    var_scheduler = DPMSolverScheduler(config["num_diffusion_steps"])
    var_scheduler.set_timesteps(config["num_inference_steps"])

    dpm = DiffusionModule(network, var_scheduler).to(config["device"])
    # For high-order sampling
    dpm.var_scheduler.net_forward_fn = dpm.network.forward
    return dpm

dpm = build_dpm(config)
```

Task 2 (Optional)

We also provide the `inverse_lambda()` function in `scheduler.py`, which corresponds to $t_\lambda(\lambda)$.

```
def inverse_lambda(self, lamb):
    """
    inverse function of lambda(t)
    """
    log_alpha_array = torch.log(self.dpm_alphas).reshape(1, -1)
    t_array = torch.linspace(0, 1, self.num_train_timesteps+1)[1:].reshape(1,-1).to(log_alpha_array)
    log_alpha = -0.5 * torch.logaddexp(torch.zeros((1,)).to(lamb.device), -2. * lamb)
    t = interpolate_fn(log_alpha.reshape((-1, 1)), torch.flip(log_alpha_array, [1]),
                       torch.flip(t_array, [1]))

    """
    Convert continuous t in [1 / N, 1] to discrete one in [0, 1000 * (N-1)/N]
    """
    t = ((t - 1 / self.num_train_timesteps) * 1000).long()
    return t.squeeze()
```

What to Submit

Include the following items into a PDF file: {NAME}_{SID}.pdf.

Task 1

- Sample Swiss-Roll data using DPM-Solver-1 and measure chamfer distance.
- Generate 500 images with the model trained in Assignment 2 using DPM-Solver-1, and measure FID.

Task 2 (Optional)

- Implement DPM-Solver-2 and repeat the same tasks in Task 1, but this time using DPM-Solver-2 instead of DPM-Solver-1.

What to Submit

Create a single ZIP file {NAME}_{SID}.zip including:

- The PDF file, formatted according to the guideline;
- Your implemented code.

Your score will be deducted by 10% for *each* missing item.

Please check carefully!

Grading

You will receive up to 10 points from this assignment.

For Task 1, you will receive:

- 10 points: Achieve CD lower than 40 on Swiss-Roll and achieve FID less than 30 on image generation with CFG.
- 5 points: Either achieve CD between 40 and 60 or achieve FID between 30 and 50.
- 0 points: Otherwise.

Grading: Compensation through Task 2

- **Task 2 is optional** and offers an additional 10 points.
- It follows the same criteria of Task 1, but must use DPM-Solver-2.
- These extra points can only compensate for **points lost in other assignments**, not in participation scores or projects.
- The total credits across all assignments will be 130 (20*6 + 10).
  (10 for Assignment 6 and 20 each for the others.)

Thank You