Diffuser

윤승제

Planning with Diffusion for Flexible Behavior Synthesis

Michael Janner [*1] **Yilun Du** [*2] **Joshua B. Tenenbaum** [2] **Sergey Levine** [1]

Abstract

Model-based reinforcement learning methods often use learning only for the purpose of estimating an approximate dynamics model, offloading the rest of the decision-making work to classical trajectory optimizers. While conceptually simple, this combination has a number of empirical shortcomings, suggesting that learned models may not be well-suited to standard trajectory optimization. In this paper, we consider what it would look like to fold as

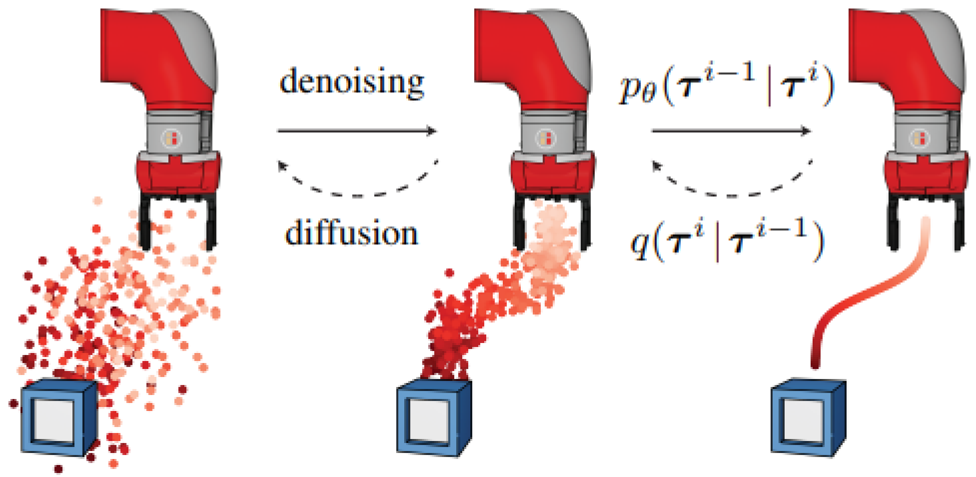


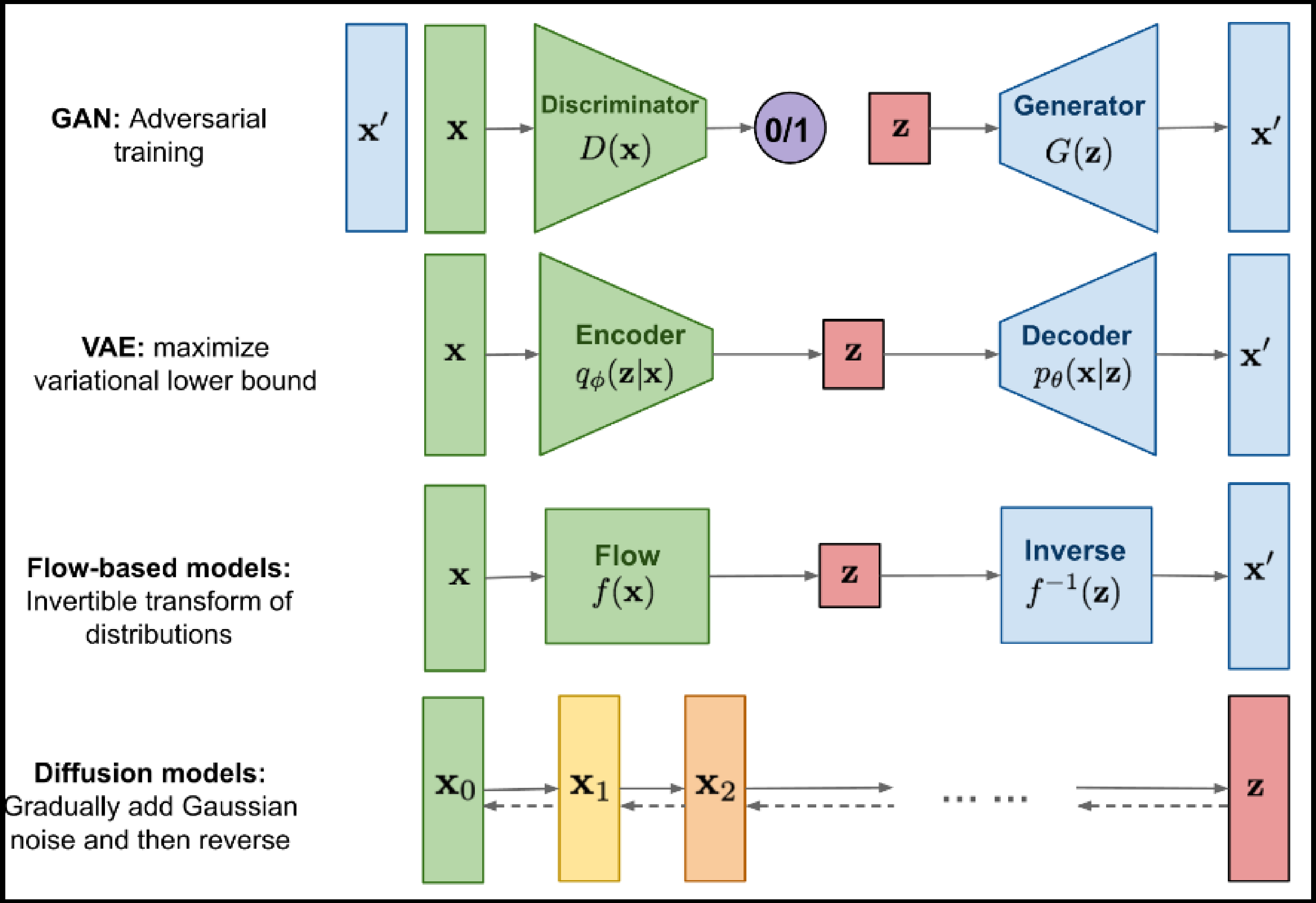
Figure 1. Diffuser is a diffusion probabilistic model that plans by iteratively refining trajectories.

Contents

Background

Overview of DDPM

- Generative model



https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

Denoising Diffusion Probabilistic Models

Denoising Diffusion Probabilistic Models

Jonathan Ho
UC Berkeley
jonathanho@berkeley.edu

Ajay Jain
UC Berkeley
ajayj@berkeley.edu

Pieter Abbeel
UC Berkeley
pabbeel@cs.berkeley.edu

Overview of DDPM

- Denoising diffusion probabilistic model
  - Forward q : noise 추가
  - Reverse p : noise 에서 원본 이미지로 denoising

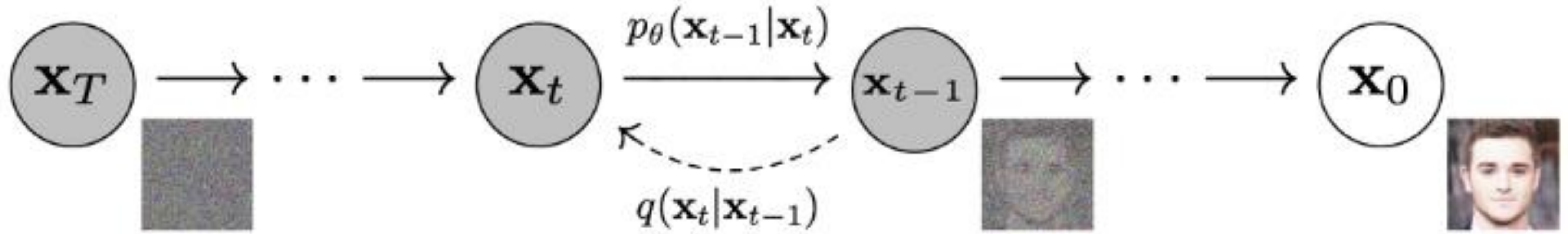


Figure 2: The directed graphical model considered in this work.

Denoising Diffusion Probabilistic Models 2020

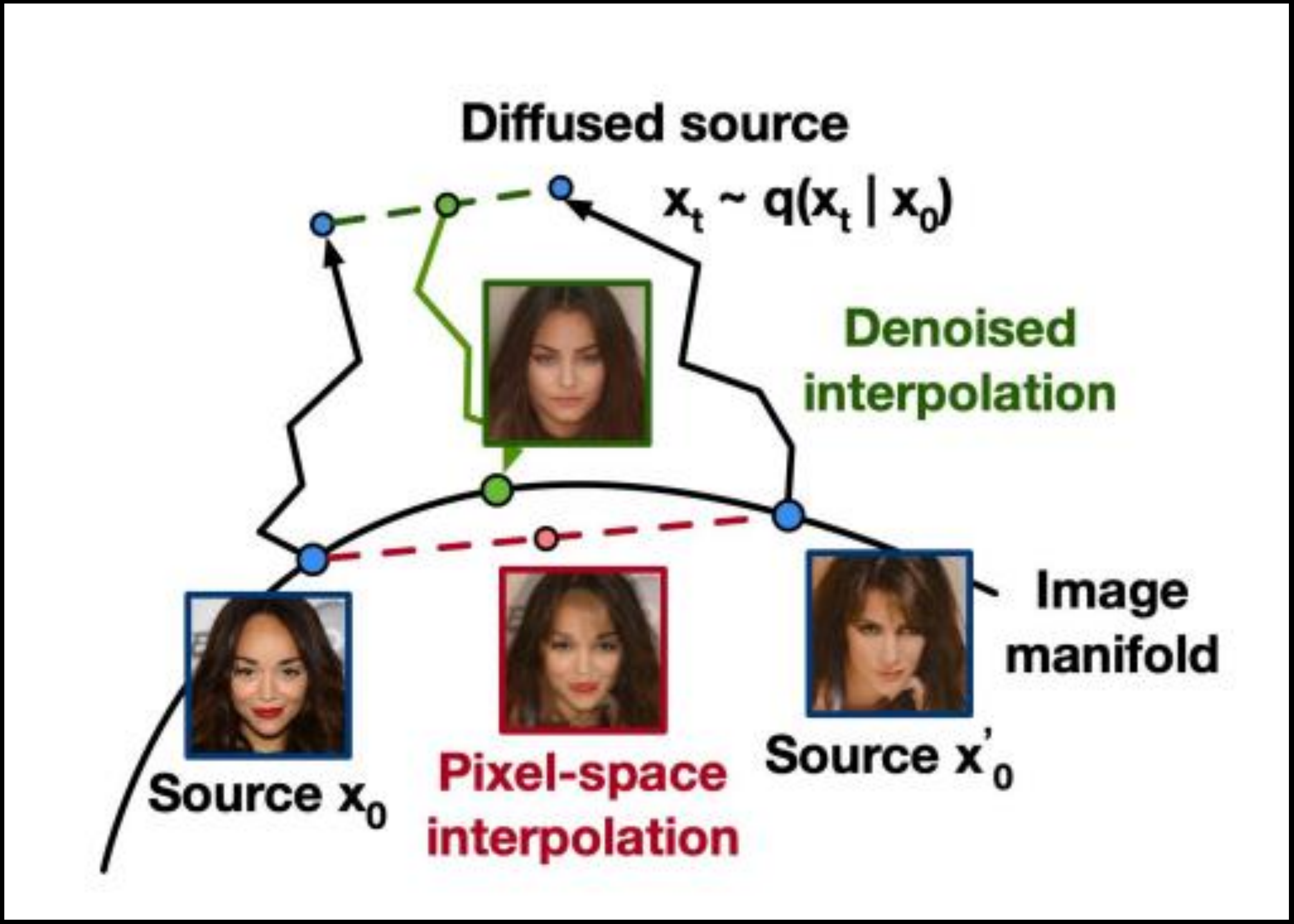
Overview of DDPM

- Denoising diffusion probabilistic model
  - 본래 noise 공간에서 large scale 먼저 거시적인 부분부터 생성
  - 거시적인 요소들이 생성되고 난 후 점차 미시적인 부분도 구체화되면서 생성

Figure 6: Unconditional CIFAR10 progressive generation ($\hat{\mathbf{x}}_0$ over time, from left to right). Extended samples and sample quality metrics over time in the appendix (Figs. 10 and 14).

Overview of DDPM

- Interpolation of DDPM
  - 픽셀 상에서의 interpolation 이 아닌 diffusion 된 space 상에서 interpolation



Training process of DDPM

- How to train the forward/reverse process
  - 최종 프로세스

Algorithm 1 Training

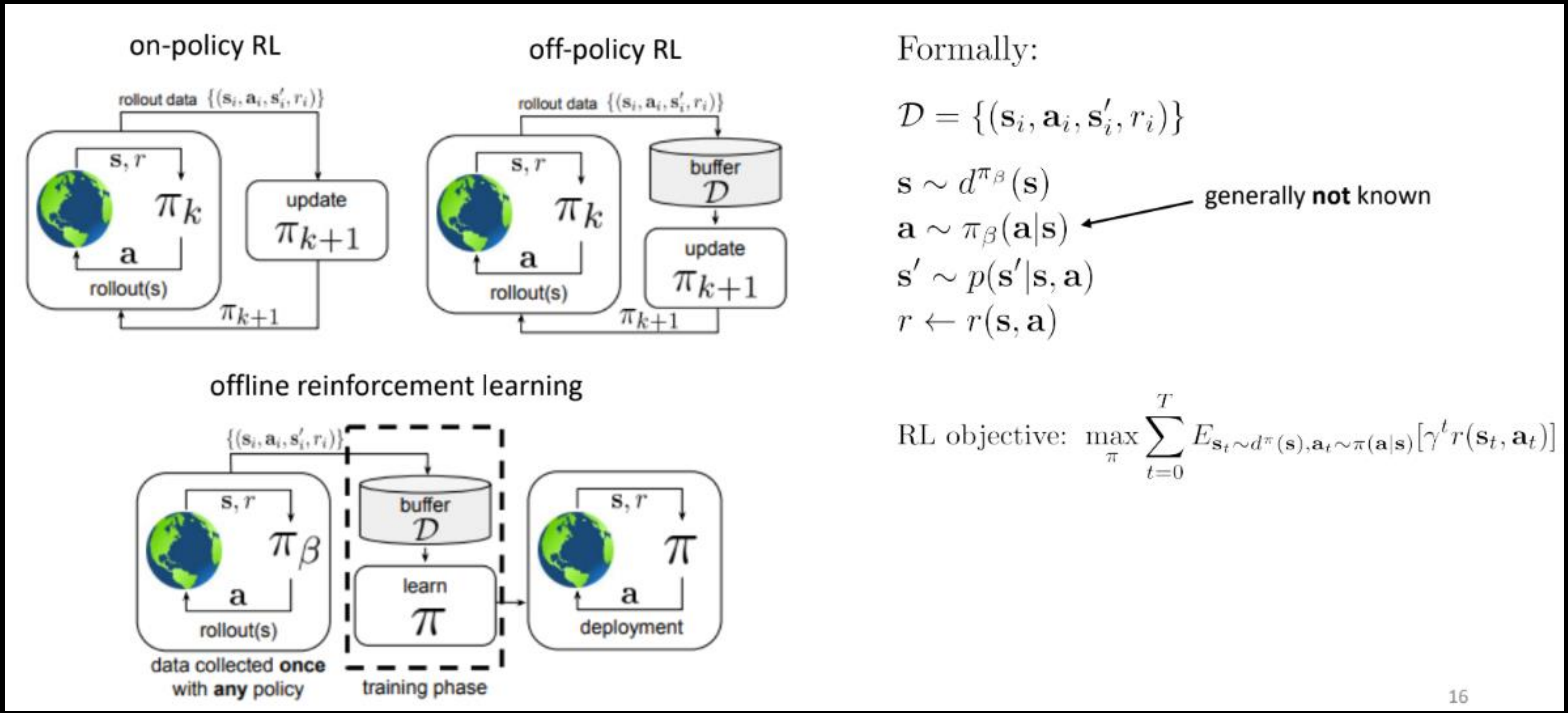
repeat
  $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
  $t \sim \mathrm{Uniform}(\{1, \ldots, T\})$
  $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  Take gradient descent step on
  $$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0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

Algorithm 2 Sampling

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

Offline RL

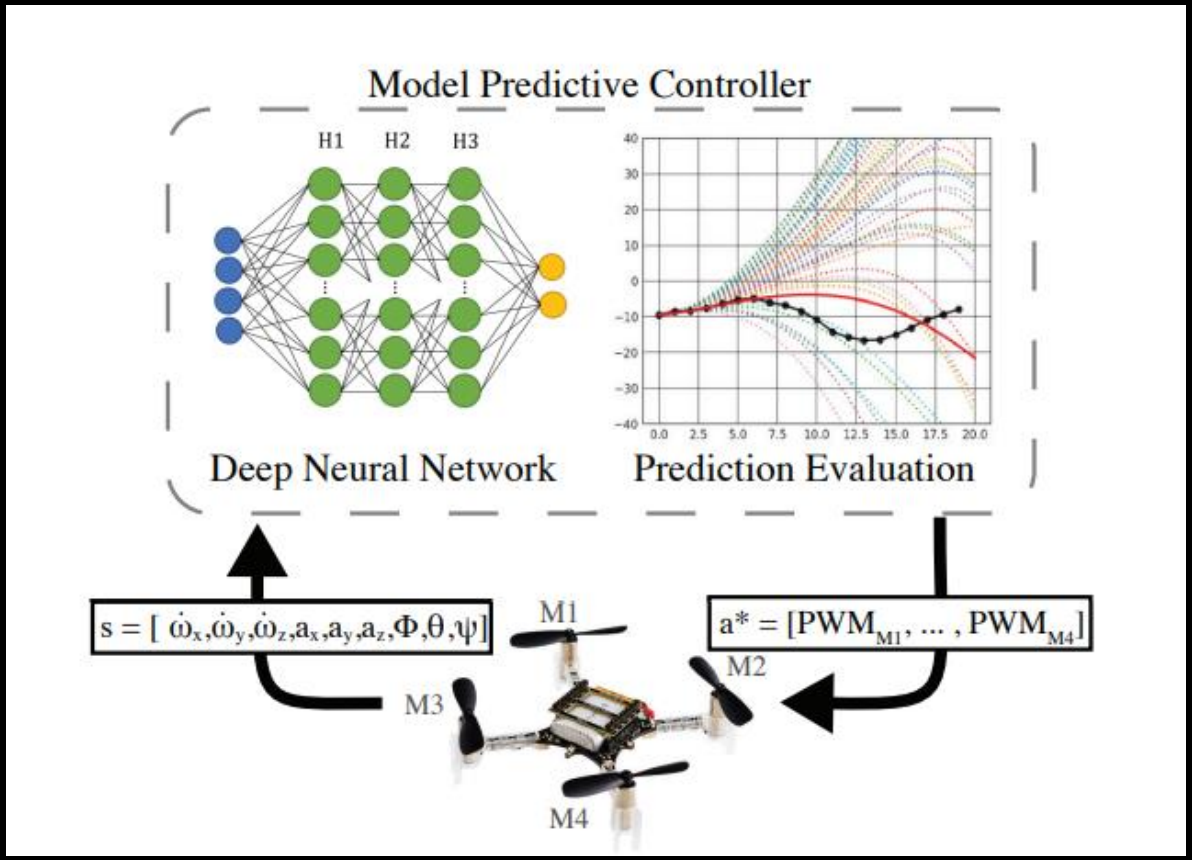
- Training process of offline RL
  - Sim, env와의 interaction 이 없음
  - Buffer 내에 없는 Counterfactual 한 scene에 대응하기 위해 uncertainty estimation & Q regularization 에 집중



https://sites.google.com/view/offlinerltutorial-neurips2020/home

Model based RL

- Examples of model based RL
  - Transition model을 supervised learning 하는 것이 일반적
  - Scalable for several task and sample efficient
  - 기존 one step prediction + markovian이라 Long term prediction 을 신뢰하긴 어려움
  - Planning 과정에서 생성된 trajectory가 실제 환경의 trajectory space에 adversarial 할수도



Rambert al el, “Low Level Control of a Quadrotor with Deep Model-Based Reinforcement Learning”, IROS19

Diffuser : DDPM based trajectory synthesis

Diffuser

- Intuition
  - Janner 왈 : Transition model sl & predictive planning을 generative model로 대체?

- Offload as much of MBRL into contemporary generative modeling as possible

replace **prediction and planning** with big generative model

Algorithm 1 Model-based RL (idealized)

Inputs: Dataset of transitions $\mathcal{D} = \{(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}), \ldots\}$, reward function $r(\cdot, \cdot)$, current state $\mathbf{s}_0$

1: Train a predictive model

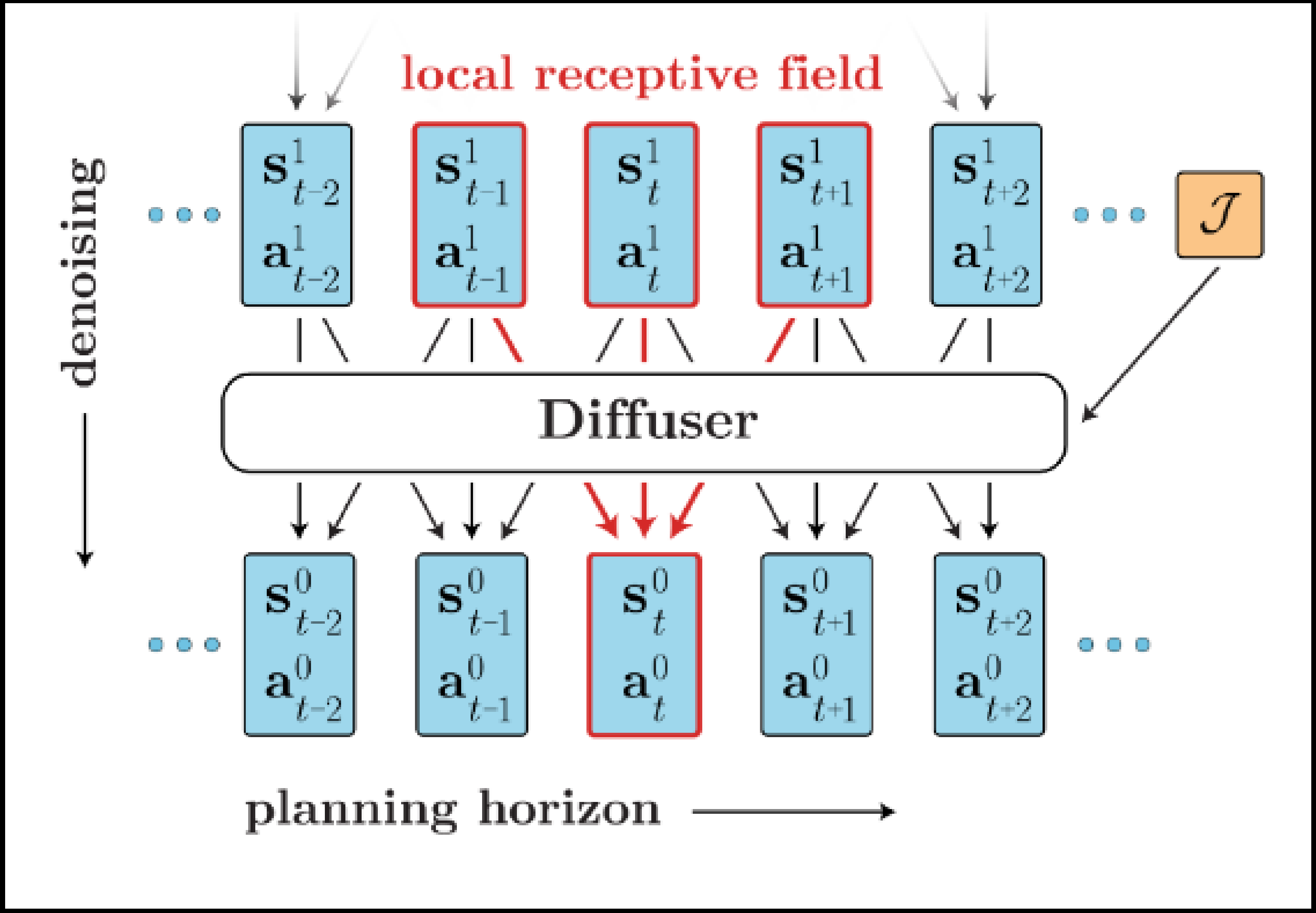
$$\underset{\boldsymbol{f}}{\text{minimize}} \; \mathbb{E}_{\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1} \sim \mathcal{D}} \left[\| \mathbf{s}_{t+1} - \boldsymbol{f}(\mathbf{s}_t, \mathbf{a}_t) \| \right]$$

2: Use model to evaluate potential plans $\mathbf{a}_{0:T}$, selecting the best one:

$$\underset{\mathbf{a}_{0:T}}{\text{maximize}} \; r(\mathbf{s}_0, \mathbf{a}_0) + r(\mathbf{s}_1, \mathbf{a}_1) + r(\mathbf{s}_2, \mathbf{a}_2) + \ldots$$

Diffuser

- Structure of diffuser
  - Single channel image array 처럼 s, a trajectory noise array 로 초기화.
  - Diffuser를 통해서 reward maximizing 한 방향으로 denoising.
  - Not used rnn but local receptive field -> rnn 특유의 memoryless 문제에 자유롭다.



Diffuser

- Structure of diffuser
  - Unet 구조
  - Group Norm
  - Mish activation

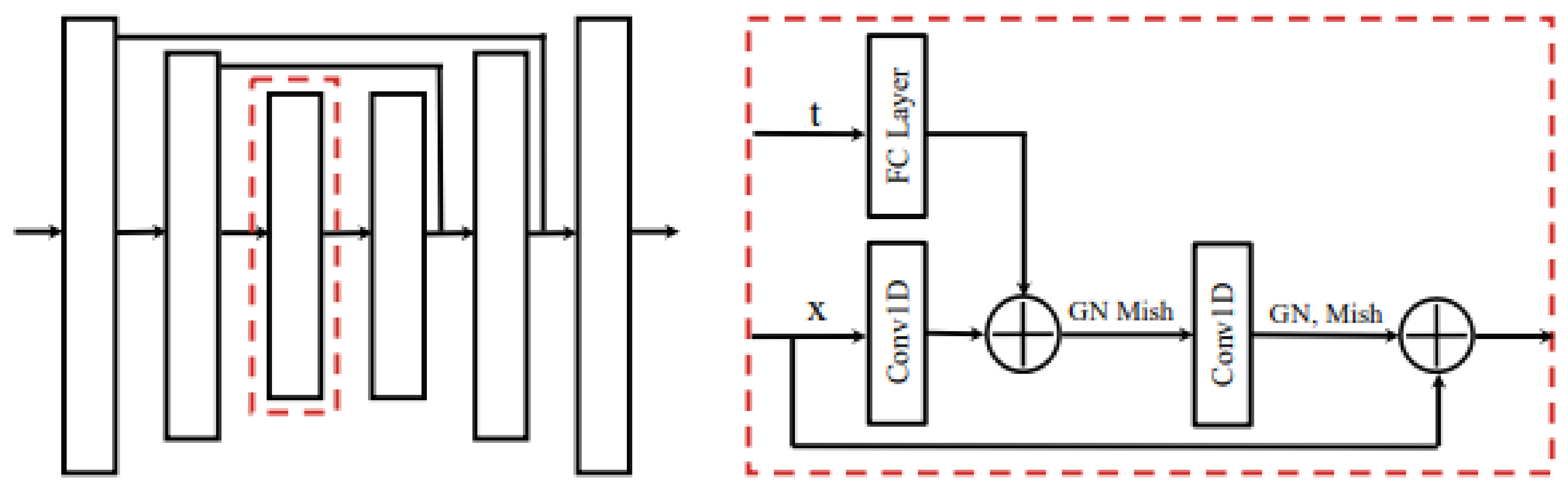


Figure A1. Diffuser has a U-Net architecture with residual blocks consisting of temporal convolutions, group normalization, and Mish nonlinearities.

Diffuser

- Planning with diffusion
  - 아래 식처럼 trajectory에 대한 분포를 정의

$$\tilde{p}_\theta(\boldsymbol{\tau}) \propto p_\theta(\boldsymbol{\tau})h(\boldsymbol{\tau}).$$

  - $p_\theta(\tau)$: physically realistic
  - $h(\tau)$: guidance (higher rewards)

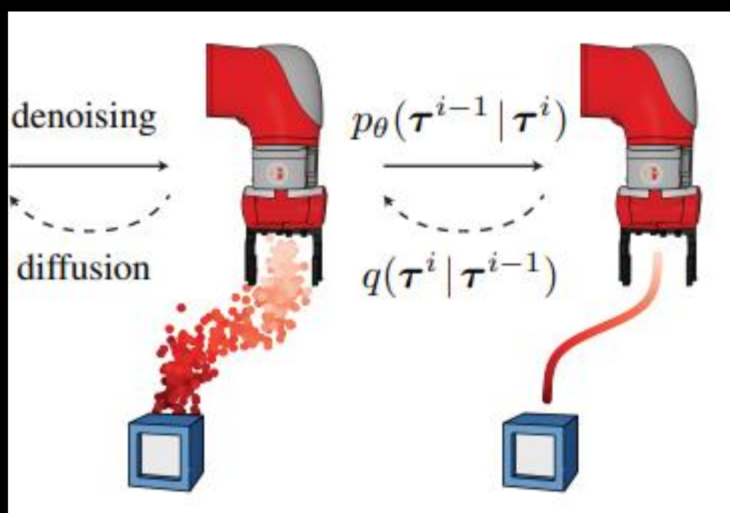
- Guided sampling
  - $h(\tau) = p(O_{1:t}|\tau)$, O_i 는 binary, optimal이면 1 아니면 0
  - $p(O_t = 1) = exp(r(s_t, a_t))$

- Goal conditioned RL as inpainting

$$h(\boldsymbol{\tau}) = \delta_{\mathbf{c}_t}(\mathbf{s}_0, \mathbf{a}_0, \ldots, \mathbf{s}_T, \mathbf{a}_T) = \begin{cases} +\infty & \text{if } \mathbf{c}_t = \mathbf{s}_t \\ 0 & \text{otherwise} \end{cases}$$

Diffuser

- Guided sampling
  - Guide operator $\mathcal{J}$는 0부터 T 까지 trajector에서 얻은 return 총합
  - $\mathcal{J}$를 최대화 하는 방향으로 gradient를 구하고, denoising 하는 reverse step p 에서 mean을 guide



$$p_\theta(\boldsymbol{\tau}^{i-1} \mid \boldsymbol{\tau}^i, \mathcal{O}_{1:T}) \approx \mathcal{N}(\boldsymbol{\tau}^{i-1}; \mu + \Sigma g, \Sigma)$$

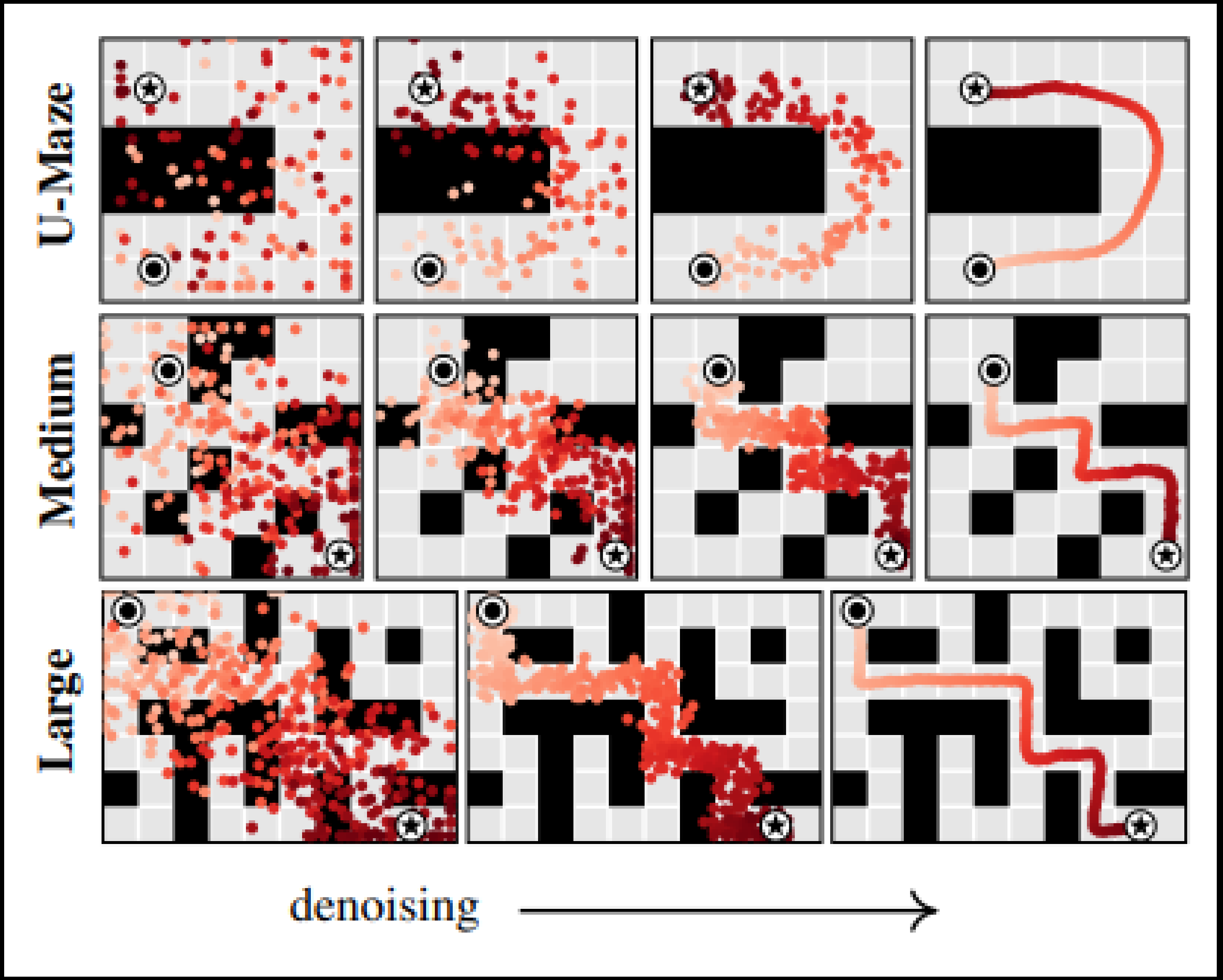
$$\begin{aligned} g &= \nabla_{\boldsymbol{\tau}} \log p(\mathcal{O}_{1:T} \mid \boldsymbol{\tau})|_{\boldsymbol{\tau}=\mu} \\ &= \sum_{t=0}^{T} \nabla_{\mathbf{s}_t, \mathbf{a}_t} r(\mathbf{s}_t, \mathbf{a}_t)|_{(\mathbf{s}_t, \mathbf{a}_t)=\mu_t} = \nabla \mathcal{J}(\mu). \end{aligned}$$

Algorithm 1 Guided Diffusion Planning

```
Require Diffuser μ_θ, guide 𝒥, scale α, covariances Σ^i
while not done do
    Observe state s; initialize plan τ^N ~ 𝒩(0, I)
    for i = N, ..., 1 do
        // parameters of reverse transition
        μ ← μ_θ(τ^i)
        // guide using gradients of return
        τ^{i-1} ~ 𝒩(μ + αΣ∇𝒥(μ), Σ^i)
        // constrain first state of plan
        τ^{i-1}_{s_0} ← s
    Execute first action of plan τ^0_{a_0}
```

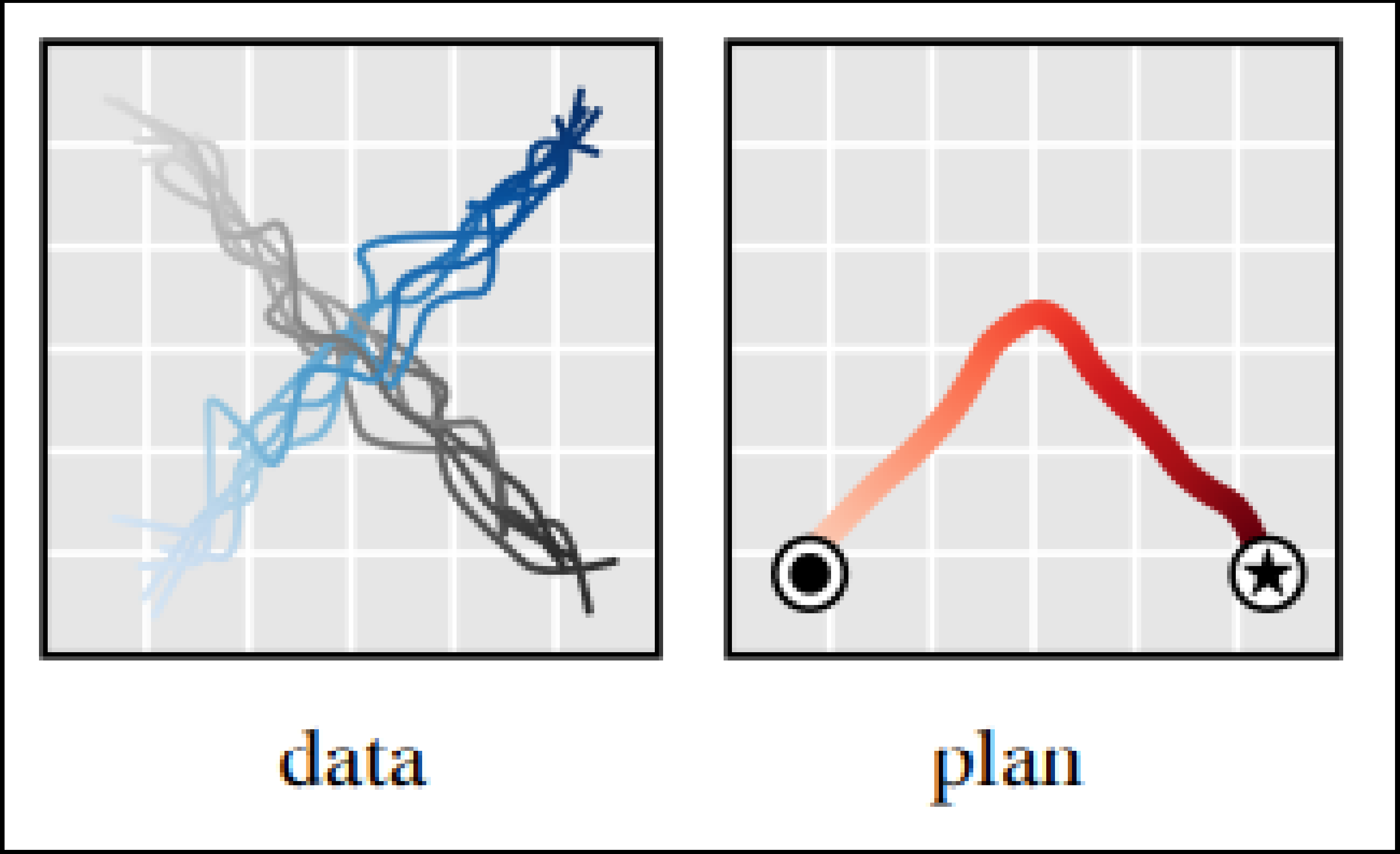
Properties of diffuser

- Learned long-horizon planning
  - 기존 model based RL과 달리 single step으로 dynamics를 예측하지 않고, horizon T 만큼의 noise array를 denoising 해서 기존보다 long-term prediction 을 효율적으로 처리
  - Starting points와 goal을 다르게 세팅해도 trajectory 생성 가능
  - Sparse reward 환경에서 강점을 가진다.



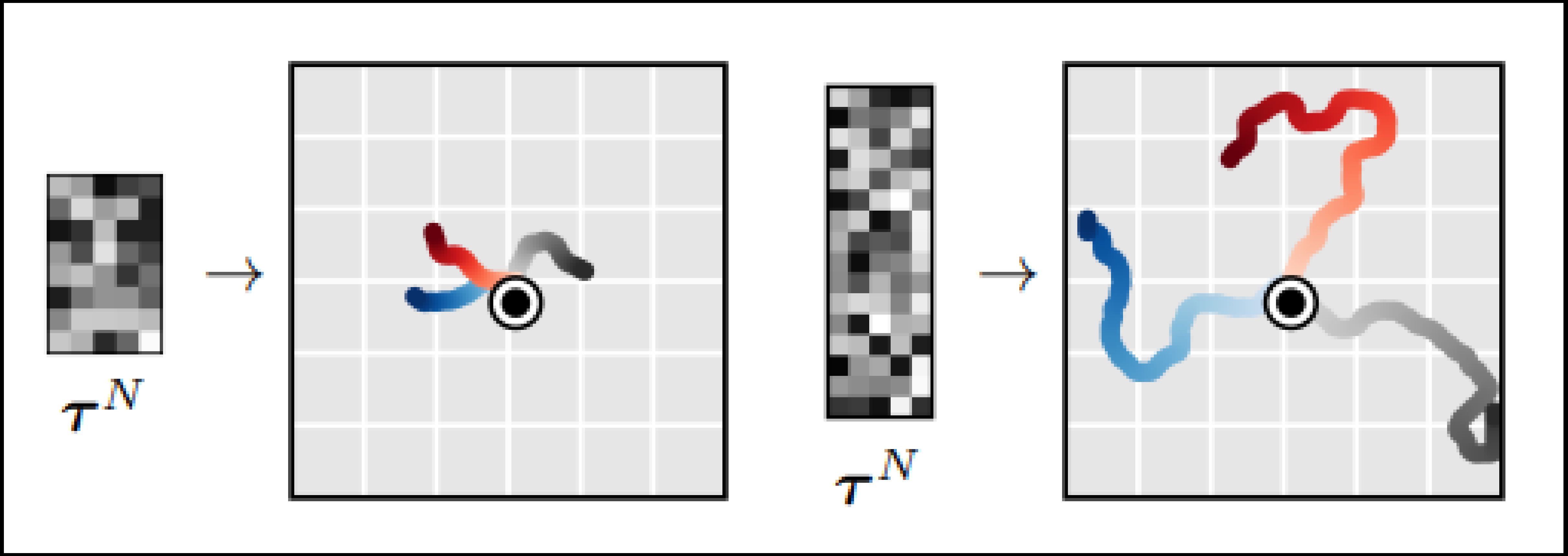
Properties of diffuser

- Temporal compositionality
  - Autoregressive 한 요소 (rnn) 없고, non-markovian 이지만, denoising 과정에서 계속 local receptive field로 temporal 한 부분을 refine 해주는 방식이라, markovian 처럼 거동 가능
  - 데이터 내에서 존재하는 temporal, transition 특성들을 학습해서 그 특성 안에 feasible한 trajectory 로 생성



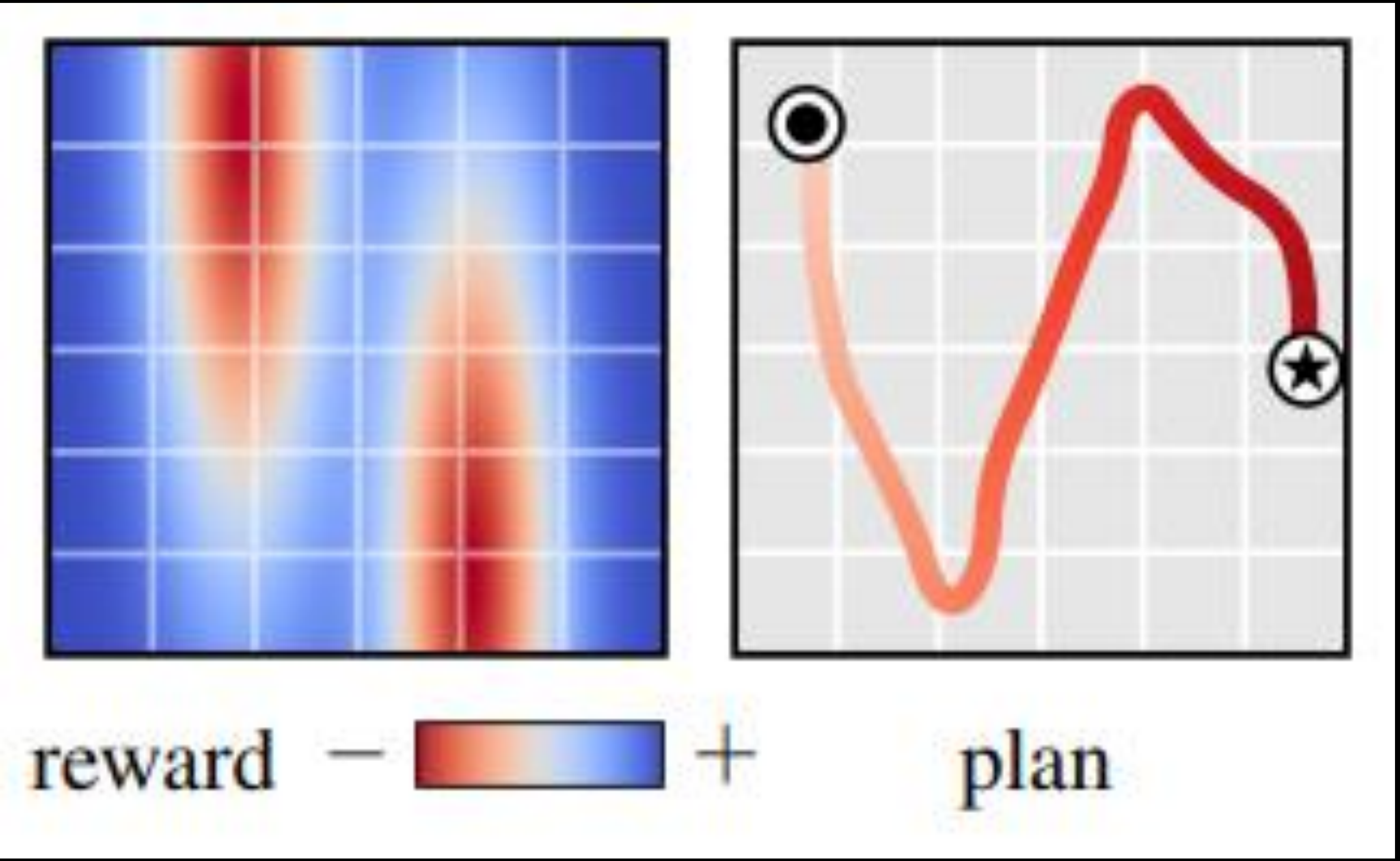
Properties of diffuser

- Variable length planning
  - Latent noise array의 길이에 따라서 생성되는 trajectory의 길이가 결정됨



Properties of diffuser

- Task compositionality
  - 새로운 reward function 으로 부터 학습 과정에 들어가지 않았던 새 task에 대해 composition 이 가능



Results of Diffuser

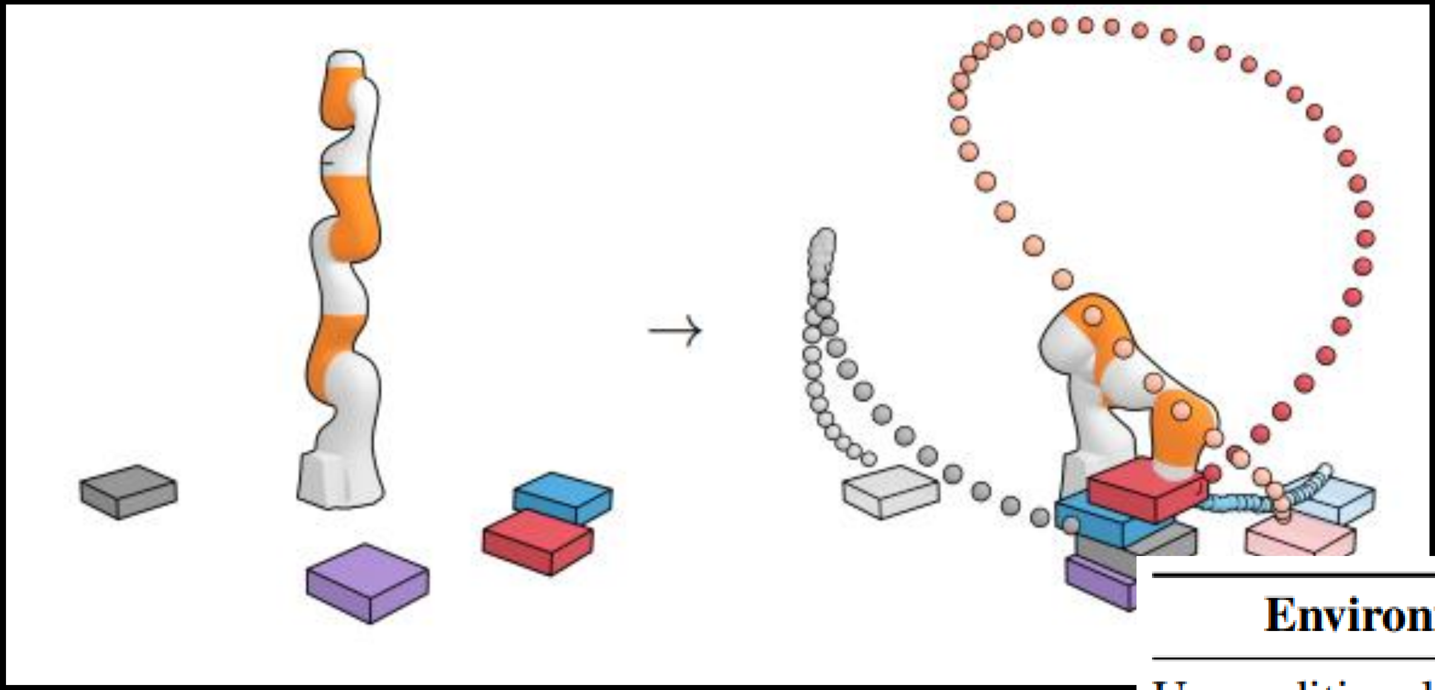
Results of diffuser

- Multi task planning
  - Baseline인 BCQ, CQL, IQL 보다 좋은 성능
  - BCQ, CQL, IQL 모두 value network에 goal condition을 input으로 줘야하는 형식이라 multi task 시 학습을 다시 해야 하나, Diffuser는 그럴 필요 없음.
  - Denoing 과정에서 s_0, s_T 을 원하는 값으로 계속 fix 해주면 multi task 에도 바로 사용 가능.

Environment		BCQ	CQL	IQL	Diffuser
Maze2D	U-Maze	12.8	5.7	47.4	**113.9** ±3.1
Maze2D	Medium	8.3	5.0	34.9	**121.5** ±2.7
Maze2D	Large	6.2	12.5	58.6	**123.0** ±6.4
Single-task Average		9.1	7.7	47.0	**119.5**
Multi2D	U-Maze	-	-	24.8	**128.9** ±1.8
Multi2D	Medium	-	-	12.1	**127.2** ±3.4
Multi2D	Large	-	-	13.9	**132.1** ±5.8
Multi-task Average		-	-	16.9	**129.4**

Results of diffuser

- Test time flexibility
  - 환경은 block stacking
  - 학습 과정에 없었던 새로운 task에서도 잘 되느냐
  - 100이 perfect
  - 학습에 없었던 새로운 task에 전혀 동작을 못하는 BCQ, CQL에 비해 diffuser는 40-60 사이의 성능을 보임 (중간에 constraint가 틀렸다던가, task 가 완료가 안 되서 감점된 듯)



Environment	BCQ	CQL	Diffuser
Unconditional Stacking	0.0	24.4	**58.7** ± 2.5
Conditional Stacking	0.0	0.0	**45.6** ± 3.1
Rearrangement	0.0	0.0	**58.9** ± 3.4
Average	0.0	8.1	**54.4**

Reference

- Reference
  - https://lilianweng.github.io/posts/2021-07-11-diffusion-models/
  - https://arxiv.org/abs/2006.11239
  - https://diffusion-planning.github.io/
  - https://arxiv.org/abs/1901.03737

END