Chapter 6: Diffusion Models and Reinforcement Learning

Diffusion models and reinforcement learning (RL) have recently developed a productive synergy. This chapter organizes the interaction in two directions:

1. **RL for diffusion:** we treat the backward sample generation as a controlled Markov process and use policy optimization to fine-tune a pre-trained diffusion model.
2. **Diffusion for RL:** we use diffusion models as powerful parameterizations to learn policies from demonstrations and offline data, especially in high-dimensional, multi-modal action spaces.

Before diving into those synergies, we will also introduce a brief background on reinforcement learning using the language of finite-horizon episodic Markov Decision Processes (MDPs).

1 Reinforcement Learning Background

1.1 MDPs

A finite-horizon MDP is a tuple

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, \{P_h\}_{h=1}^H, \{r_h\}_{h=1}^H, H),$$

where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, H is the horizon, and for any $h \in \{1, \dots, H\}$ and state-action pair (s_h, a_h),

$$P_h(\cdot \mid s_h, a_h) \text{ is the transition kernel,} \qquad r_h(s_h, a_h) \text{ is the reward function.}$$

A policy $\pi = \{\pi_h\}_{h=1}^H$ is a series of probability distributions over actions given the current state. At step h and state s_h, the policy

$$\pi_h(\cdot \mid s_h)$$

is a conditional distribution of action a_h. Given any policy, a state-action trajectory is generated by sequentially interacting with the environment and denoted as

$$\tau = (s_1, a_1, r_1, s_2, a_2, r_2, \dots, s_H, a_H, r_H).$$

We term such a trajectory as an episode.

Value function and Q-function. For performance evaluation, we define the value function and Q-function as follows. For any step h, state-action pair (s, a), and policy π, we have

$$V_h^\pi(s) = \mathbb{E}^\pi \left[\sum_{i=h}^H r_i(s_i, a_i) \mid s_h = s \right] \quad \text{and}$$

$$Q_h^\pi(s, a) = \mathbb{E}^\pi \left[\sum_{i=h}^H r_i(s_i, a_i) \mid s_h = s, a_h = a \right].$$

Here, the superscript indicates the dependence on the policy. Indeed, the expectation under policy π implies that future actions are executed according to π. The value- and Q-function are both the expected cumulative reward starting from step h, with Q-function additionally specifying the immediate action a_h. The definition of value- and Q-function implies that

$$V_h^\pi(s) = \int_{\mathcal{A}} Q_h^\pi(s,a)\pi_h(a \mid s)\mathrm{d}a.$$

Meanwhile, we have the Bellman equation:

$$Q_h^\pi(s,a) = r_h(s,a) + \mathbb{E}_{s_{h+1}\sim P_h(\cdot|s,a)}\left[V_{h+1}^\pi(s_{h+1})\right].$$

1.2 Policy Learning

The goal of policy learning is to find a policy that maximizes the expected cumulative return. Let ρ denote the distribution of the initial state s_1. The average performance of a policy π can be written as

$$J(\pi) = \mathbb{E}_{s_1\sim\rho}\big[V_1^\pi(s_1)\big].$$

Policy optimization seeks a maximizer

$$\pi^\star \in \operatorname*{argmax}_{\pi\in\Pi} \ \mathbb{E}_{s_1\sim\rho}[V_1^\pi(s_1)],$$

where Π is a class of admissible policies.

Dynamic programming. A standard method for policy optimization in a *known* finite-horizon MDP is **dynamic programming** (backward induction). It proceeds by computing the optimal value functions $\{V_h^\star\}_{h=1}^{H+1}$ backwards in time, and then extracting an optimal policy by greedy maximization of the optimal Q-functions.

Define the terminal value function as

$$V_{H+1}^\star(s) = 0, \qquad \forall s \in \mathcal{S}.$$

For each step $h = H, H-1, \ldots, 1$, define the optimal Q-function by a one-step lookahead:

$$Q_h^\star(s,a) = r_h(s,a) + \mathbb{E}_{s'\sim P_h(\cdot|s,a)}\big[V_{h+1}^\star(s')\big], \qquad (s,a) \in \mathcal{S}\times\mathcal{A},$$

and then compute the optimal value by maximizing over actions:

$$V_h^\star(s) = \sup_{a\in\mathcal{A}} \ Q_h^\star(s,a), \qquad s \in \mathcal{S}.$$

When $\mathcal{A}$ is discrete, the supremum is a maximum; when $\mathcal{A}$ is continuous, one typically assumes the supremum is attained.

After computing $\{Q_h^\star\}$, define an optimal policy by

$$\pi_h^\star(\cdot \mid s) = \delta_{a_h^\star} \quad \text{with} \quad a_h^\star \in \operatorname*{argmax}_{a\in\mathcal{A}} \ Q_h^\star(s,a), \qquad h = 1,\ldots,H.$$

Equivalently, $\pi^\star = \{\pi_h^\star\}_{h=1}^H$ is a deterministic optimal policy (randomization can be used if the maximizer of $Q^\star$ is not unique).

Dynamic Programming

Input: horizon H, transitions $\{P_h\}_{h=1}^H$, rewards $\{r_h\}_{h=1}^H$.
Initialize: $V_{H+1}^\star(\cdot) \leftarrow 0$.
For $h = H, H-1, \ldots, 1$:

1. For every (s, a), compute

$$Q_h^\star(s,a) \leftarrow r_h(s,a) + \mathbb{E}_{s' \sim P_h(\cdot|s,a)}\big[V_{h+1}^\star(s')\big].$$

2. For every s, compute

$$V_h^\star(s) \leftarrow \max_{a \in \mathcal{A}} Q_h^\star(s,a), \qquad \pi_h^\star(\cdot \mid s) \leftarrow \delta_{a_h^\star(s)} \quad \text{with} \quad a_h^\star(s) \in \operatorname*{argmax}_{a \in \mathcal{A}} Q_h^\star(s,a).$$

Output: optimal policy $\pi^\star = \{\pi_h^\star\}_{h=1}^H$ and optimal values $\{V_h^\star\}_{h=1}^H$.

This procedure is exact when P_h and r_h are known and the required expectations and maximizations are tractable. In many modern applications, P_h is unknown and $\mathcal{S}$ (and often $\mathcal{A}$) is high-dimensional, so exact DP is infeasible; this motivates approximate methods, including parameterized policies optimized by policy gradients.

In high-dimensional complex applications, the policy is typically parameterized by a concept class such as neural networks. We denote by $\pi_\theta = \{\pi_{\theta,h}\}_{h=1}^H$ a family of policies with trainable parameter θ. A key advantage of parameterized policies is that they naturally handle continuous and high-dimensional action spaces. In this case, a widely used tool is the **policy gradient** principle, which provides an unbiased gradient estimator of $\nabla_\theta J(\pi_\theta)$ using only samples from rollouts.

Policy gradient. Let $\pi_\theta = \{\pi_{\theta,h}\}_{h=1}^H$ be a differentiable policy class. For a trajectory (episode)

$$\tau = (s_1, a_1, \ldots, s_H, a_H),$$

we denote the cumulative reward as

$$G(\tau) = \sum_{h=1}^H r_h(s_h, a_h).$$

Let $p_\theta(\tau)$ (resp. P_θ) denote the probability density/mass (resp. distribution) of trajectory τ induced by the MDP and policy π_θ. Then we have

$$J(\pi_\theta) = \int G(\tau) p_\theta(\tau) \mathrm{d}\tau.$$

Assuming we can interchange differentiation and integration, we have

$$\begin{aligned}
\nabla_\theta J(\pi_\theta) &= \nabla_\theta \int G(\tau)\, p_\theta(\tau)\, \mathrm{d}\tau \\
&= \int G(\tau)\, \nabla_\theta p_\theta(\tau)\, \mathrm{d}\tau \\
&= \int G(\tau)\, p_\theta(\tau)\, \nabla_\theta \log p_\theta(\tau)\, \mathrm{d}\tau \\
&= \mathbb{E}_{\tau \sim P_\theta}\big[\, G(\tau)\, \nabla_\theta \log p_\theta(\tau)\, \big].
\end{aligned}$$

It remains to compute $\nabla_\theta \log p_\theta(\tau)$. By the Markov property of the MDP and the definition of the policy, the trajectory density $p_\theta(\tau)$ factorizes as

$$p_\theta(\tau) = \rho(s_1) \prod_{h=1}^{H} \pi_{\theta,h}(a_h \mid s_h) p_h(s_{h+1} \mid s_h, a_h),$$

where ρ is the initial state distribution and p_h is the density of transition kernel. For the last transition density p_H, we can simply take it to be 1. Taking logarithm on both sides gives rise to

$$\log p_\theta(\tau) = \log \rho(s_1) + \sum_{h=1}^{H} \log \pi_{\theta,h}(a_h \mid s_h) \; + \; \sum_{h=1}^{H} \log p_h(s_{h+1} \mid s_h, a_h).$$

Since ρ and p_h do not depend on θ, we obtain

$$\nabla_\theta \log p_\theta(\tau) = \sum_{h=1}^{H} \nabla_\theta \log \pi_{\theta,h}(a_h \mid s_h).$$

Substituting into $\nabla_\theta J(\pi_\theta)$ yields the finite-horizon policy gradient formula:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau \sim P_\theta} \left[G(\tau) \sum_{h=1}^{H} \nabla_\theta \log \pi_{\theta,h}(a_h \mid s_h) \right].$$

We can further simplify $\nabla_\theta J(\pi_\theta)$ by introducing a lookahead cumulative reward:

$$G_h(\tau) = \sum_{i=h}^{H} r_i(s_i, a_i).$$

Then we claim that

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau \sim P_\theta} \left[\sum_{h=1}^{H} G_h(\tau) \nabla_\theta \log \pi_{\theta,h}(a_h \mid s_h) \right]. \tag{1.1}$$

Derivation of Equation (1.1). Fix a step h. We decompose the total cumulative reward into the *past* and the *future* relative to h:

$$G(\tau) = \underbrace{\sum_{i=1}^{h-1} r_i(s_i, a_i)}_{G_{<h}(\tau)} + \underbrace{\sum_{i=h}^{H} r_i(s_i, a_i)}_{G_h(\tau)}.$$

We then have

$$\begin{aligned} \mathbb{E}_{\tau \sim P_\theta} \left[G(\tau) \nabla_\theta \log \pi_{\theta,h}(a_h \mid s_h) \right] = \; & \mathbb{E}_{\tau \sim P_\theta} \left[G_{<h}(\tau) \nabla_\theta \log \pi_{\theta,h}(a_h \mid s_h) \right] \\ & + \mathbb{E}_{\tau \sim P_\theta} \left[G_h(\tau) \nabla_\theta \log \pi_{\theta,h}(a_h \mid s_h) \right]. \end{aligned}$$

We show that the first term is zero

$$\mathbb{E}_{\tau \sim P_\theta} \left[G_{<h}(\tau) \nabla_\theta \log \pi_{\theta,h}(a_h \mid s_h) \right] = 0.$$

To see this, condition on the history up to time h, i.e., define a filtration

$$\mathcal{F}_h = \sigma(s_1, a_1, \ldots, s_h),$$

where σ denotes the σ-algebra. Using the tower property of expectation, we have

$$\begin{aligned}\mathbb{E}_{\tau\sim P_\theta}\left[G_{<h}(\tau)\nabla_\theta \log \pi_{\theta,h}(a_h \mid s_h)\right] &= \mathbb{E}\left[\mathbb{E}\left[G_{<h}(\tau)\nabla_\theta \log \pi_{\theta,h}(a_h \mid s_h) \mid \mathcal{F}_h\right]\right] \\ &= \mathbb{E}\left[G_{<h}(\tau)\mathbb{E}\left[\nabla_\theta \log \pi_{\theta,h}(a_h \mid s_h) \mid \mathcal{F}_h\right]\right].\end{aligned}$$

Conditioned on $\mathcal{F}_h$, the only remaining randomness in the inner expectation is $a_h \sim \pi_{\theta,h}(\cdot \mid s_h)$, hence it holds that

$$\begin{aligned}\mathbb{E}\left[\nabla_\theta \log \pi_{\theta,h}(a_h \mid s_h) \mid \mathcal{F}_h\right] &= \int_{\mathcal{A}} \nabla_\theta \log \pi_{\theta,h}(a \mid s_h)\, \pi_{\theta,h}(a \mid s_h)\, \mathrm{d}a \\ &= \int_{\mathcal{A}} \nabla_\theta \pi_{\theta,h}(a \mid s_h)\, \mathrm{d}a \\ &= 0.\end{aligned}$$

Therefore the past reward term vanishes, and we complete the proof. □

The following box summarizes the REINFORCE algorithm based on policy gradient.

REINFORCE Algorithm

Input: step size η, number of episodes per iteration N.
Initialize: initial policy π_θ.
Repeat until convergence:

1. For $n = 1, \ldots, N$, sample an episode

   $$\tau^{(n)} = (s_1^{(n)}, a_1^{(n)}, \ldots, s_H^{(n)}, a_H^{(n)})$$

   by executing π_θ in the environment, and record rewards $r_h^{(n)} = r_h(s_h^{(n)}, a_h^{(n)})$.

2. For each episode and each step h compute $G_h^{(n)} = G_h(\tau^{(n)})$.

3. Compute gradient

   $$\widehat{\nabla_\theta J} = \frac{1}{N}\sum_{n=1}^{N}\sum_{h=1}^{H} G_h^{(n)} \nabla_\theta \log \pi_{\theta,h}\big(a_h^{(n)} \mid s_h^{(n)}\big).$$

4. Update policy by

   $$\theta \leftarrow \theta + \eta\, \widehat{\nabla_\theta J}.$$

Output: policy π_θ.

Note that the gradient computation step is to approximate the policy gradient using a finite-sample average. In addition, the policy gradient does not require the reward function to be differentiable, instead, it only requires evaluating the gradient of the log-likelihood of the policy.

Online vs. offline setting. Equation (1.1) can also be rewritten using the Q-function:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim P_\theta}\left[\sum_{h=1}^{H} Q_h^{\pi_\theta}(s_h, a_h)\nabla_\theta \log \pi_{\theta,h}(a_h \mid s_h)\right].$$

This is true by observing that $Q_h^{\pi_\theta}(s_h, a_h) = \mathbb{E}_{\tau\sim P_\theta}[G_h(\tau) \mid s_h, a_h]$. In real applications, the reward function or the Q-function is usually unknown. We need to estimate the reward (or Q-function) for policy optimization. Depending on whether active interation with the environment is allowed, policy learning problems are categorized into two types:

- In **online** settings, active interaction with the environmemt is allowed and data is collected by executing some policy and receiving feedback from the environment; so the trajectory distribution adapts as the policy changes.
- In **offline** settings, interaction with the environment is prohibitive, but we are given a pre-collected data set of trajectories collected under some behavior policy.

The estimation of reward (or Q-function) may lead to large variance in the finite-sample approximation of the policy gradient. Variance reduction is implemented for stablizing the policy learning. On a high-level, consider any measurable function $b_h(s_h)$, only depending on the state. By the same derivation of Equation (1.1), we have

$$\mathbb{E}_{\tau\sim p_\theta}\big[b_h(s_h)\,\nabla_\theta \log \pi_{\theta,h}(a_h \mid s_h)\big] = 0.$$

Therefore, it holds that

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim p_\theta}\left[\sum_{h=1}^{H}\big(Q_h^{\pi_\theta}(s_h, a_h) - b_h(s_h)\big)\,\nabla_\theta \log \pi_{\theta,h}(a_h \mid s_h)\right].$$

We choose $b_h(s_h)$ so that the variance of $\big(Q_h^{\pi_\theta}(s_h, a_h) - b_h(s_h)\big)\nabla_\theta \log \pi_{\theta,h}(a_h \mid s_h)$ is minimized, which yields the *advantage* form

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim p_\theta}\left[\sum_{h=1}^{H} A_h^{\pi_\theta}(s_h, a_h)\,\nabla_\theta \log \pi_{\theta,h}(a_h \mid s_h)\right] \quad \text{with} \quad A_h^{\pi_\theta}(s,a) = Q_h^{\pi_\theta}(s,a) - V_h^{\pi_\theta}(s).$$

The advantage form underlies actor-critic methods in both online and offline settings. Later variants lead to preference-based policy optimization methods used in LLM fine-tuning, e.g., Group-based Relative Policy Optimization (GRPO) variants.

1.3 Infinite-Horizon MDPs

Finite-horizon episodic MDPs are the most natural match to diffusion sampling (which runs for a fixed number of steps). For completeness, we briefly mention two standard infinite-horizon formulations.

Discounted setting. An infinite-horizon discounted MDP specifies a stationary transition kernel $P(\cdot \mid s, a)$ and reward $r(s, a)$, and is represented as a tuple

$$(\mathcal{S}, \mathcal{A}, P, r, \gamma),$$

where $\gamma \in (0, 1)$ is a discount factor. The value function and Q-function are now defined as

$$V^\pi(s) = \mathbb{E}^\pi\Big[\sum_{t=1}^{\infty} \gamma^t r(s_t, a_t) \mid s_1 = s\Big] \quad \text{and} \quad Q^\pi(s, a) = \mathbb{E}^\pi\Big[\sum_{t=1}^{\infty} \gamma^t r(s_t, a_t) \mid s_1 = s, a_1 = a\Big].$$

Policy learning in discounted MDP can be similarly formulated by maximizing

$$J_\gamma(\pi) = \mathbb{E}_{s_1 \sim \rho}[V^\pi(s_1)].$$

Most policy gradient methods extend to this setting with minimal changes, replacing episodic returns by discounted returns.

Average-reward setting. When discounting is undesirable (e.g., continuing tasks), a common objective is the long-run average reward

$$J_{\text{avg}}(\pi) = \lim_{T \to \infty} \frac{1}{T} \mathbb{E}^\pi\Big[\sum_{t=1}^{T} r(s_t, a_t)\Big],$$

assuming the limit exists (e.g., under ergodicity/unichain assumptions for the Markov chain induced by a stationary policy).

2 RL for Fine-tuning Diffusion Models

RL-based fine-tuning methods for pre-trained diffusion models become attractive when: (i) the reward is *non-differentiable* or only available as black-box feedback, (ii) the reward is defined on the *final sample* (or a trajectory) rather than each intermediate noisy state.

2.1 From Backward Sample Generation to MDPs

In order to apply RL-based fine-tuning methods, we need to reformulate the backward sample generation process as an MDP. Fix a time discretization $0 = t_0 < t_1 < \cdots < t_K = T$ (or directly consider the DDPM formulation). Suppose we have a metric $r(\cdot)$ for quantifying the quality of the generated samples. We define an MDP of horizon $H = K$ for representing the backward sample generation of diffusion models:

- State space $\mathcal{S}$: $s = X \in \mathbb{R}^d$.
- Action space $\mathcal{A}$: $a = X \in \mathbb{R}^d$.
- Transition kernel $\{P_h\}_{h=1}^H$: $P_h(s_{h+1} \mid s_h, a_h) = \delta_{s_{h+1} = a_h}$.

- Reward $\{r_h\}_{h=1}^H$:

$$r_h(s_h) = \begin{cases} 0 & \text{if } h \neq H \\ r(s_h) & \text{if } h = H \end{cases}.$$

- Initial distribution ρ: $\rho = \mathsf{N}(0, I)$.

Here, X represents the (noisy) data in a d-dimensional space. We note that the state space and action space are identical. In fact, at step h, the state is the current noisy data $X_{t_{h-1}}$. The action is the next-step noisy data X_{t_h}. Therefore, the transition probability is simply replacing $X_{t_{h-1}}$ by X_{t_h}—a delta function. The initial distribution is precisely the latent distribution in diffusion models. The reward is sparse and there is no intermediate reward except the terminal one. Therefore, the value function and Q-function is extremely simple, which only involves the terminal reward.

The key component in the MDP reformulation above is the policy, where we need to define the conditional distribution of X_{t_h} given $X_{t_{h-1}}$. In diffusion models, such a conditional distribution is determined by the learned score function s_θ, where θ denotes the trainable parameters. Specifically, we have

$$X_{t_h} \mid X_{t_{h-1}} \sim \mathsf{N}(\mu_\theta(X_{t_{h-1}}, t_{h-1}), \sigma_{t_{h-1}}^2 I),$$

where μ_θ is the estimated clean data using the learned score function s_θ, and $\sigma_{t_{h-1}}^2$ is the noise schedule. In this regard, a diffusion model defines the policy to be conditional Gaussian with the mean determined by the score function.

2.2 From Score Function Fine-tuning to Policy Optimization

Fine-tuning a diffusion model is to modify the score function so that the sample generation process is correspondingly steered toward the desired high-reward region. Using the MDP reformulation, we cast the score function fine-tuning as policy optimization.

Recall the policy gradient in (1.1), we need to evaluate $\nabla_\theta \log \pi_{\theta,h}$. This is straightforward as $\pi_{\theta,h}$ is Gaussian:

$$\nabla_\theta \log \pi_{\theta,h}(a_h \mid s_h) = -\frac{1}{2\sigma_{t_{h-1}}^2} \nabla_\theta \|\mu_\theta(s_h, t_{h-1}) - a_h\|_2^2.$$

We can now plug this gradient expression into the REINFORCE algorithm and update the parameters in the score function with the reward feedback.

Regularization. In many situations, we would like to balance the reward maximization and sample fidelity to prevent reward hacking. Adding some regularization on the policy becomes handy in policy optimization. One example is to impose a series of KL-divergence penalty to the expected cumulative reward $J(\pi_\theta)$:

$$\max_\theta J(\pi_\theta) - \lambda \sum_{h=1}^H \mathbb{E}[\mathrm{KL}(\pi_{\theta,h}(a_h \mid s_h), \pi_{\mathrm{pre},h}(a_h \mid s_h))],$$

where $\pi_{\text{pre},h}$ denote the policy induced by the pre-trained diffusion model. Since both $\pi_{\theta,h}$ and $\pi_{\text{pre},h}$ are Gaussian, a closed-form expression of the KL-divergence can be derived (left as an exercise, and the penalty will eventually be an ℓ^2 distance between the score functions). Alternative methods introduce trust regions on the policy or penalize the terminal distribution of the generated samples.

Initial distribution fine-tuning. In classical RL literature, the initial distribution ρ is considered as fixed. Therefore, in order to maximize the expected cumulative reward J, the policy is the decision variable. In diffusion models, however, the initial distribution can also be fine-tuned. This provides a completely new angle of fine-tuning diffusion models using the MDP reformulation. Yet, compared to the score function fine-tuning, modifying the initial distribution is not as popular.

3 Diffusion Models for Policy Learning

In classical RL, a policy $\pi(a \mid s)$ is typically parameterized by a Gaussian neural network, where the mean and variance is trainable. A Gaussian distribution is uni-modal and relatively simple. Therefore, it may break down for:

- **Perception-rich inputs:** images, videos, language instructions, long-horizon context.
- **Multi-modal behaviors:** many distinct valid actions can solve the same task.
- **Long-horizon structure:** temporally correlated action sequences (skills).

Vision-language models (VLMs) exemplify this shift by producing structured, generative outputs conditioned on rich context. In control, this suggests modeling *the conditional distribution of action sequences* given observations and goals. Diffusion models are compelling because they represent complex multimodal conditionals with stable training.

Please see slides for the remaining contents.