Algorithm Notes

Chuan Mao

November 2025

Contents

Part I
Reinforcement Learning

1. Policy Gradient Methods

Policy Gradient (PG) methods are a class of reinforcement learning algorithms that learn a parameterized policy directly, without consulting a value function. The policy, denoted as $\pi_\theta(a|s)$, is a probability distribution over actions given a state, where θ are the parameters (e.g., the weights of a neural network). The goal is to adjust these parameters to maximize the expected total reward.

1.1. The Objective Function

The performance of a policy π_θ is measured by an objective function $J(\theta)$, which is typically defined as the expected return. For an episodic task, this is the expected return from the start state s_0:

$$J(\theta) = V^{\pi_\theta}(s_0) = \mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)] \tag{1}$$

where $\tau = (s_0, a_0, s_1, a_1, \ldots, s_T)$ is a trajectory (an episode) sampled by running the policy π_θ. The term $R(\tau) = \sum_{t=0}^{T} \gamma^t r(s_t, a_t)$ is the total discounted reward for that trajectory. The probability of observing a particular trajectory τ under policy π_θ is $p_\theta(\tau)$. The objective function can thus be written as:

$$J(\theta) = \sum_\tau p_\theta(\tau) R(\tau) \tag{2}$$

Our goal is to find the optimal parameters θ^* that maximize this objective. We can achieve this using gradient ascent, which requires us to compute the policy gradient, $\nabla_\theta J(\theta)$:

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\theta_k) \tag{3}$$

1.2. Derivation of the Policy Gradient Theorem

To derive an expression for the policy gradient, we start by taking the gradient of the objective function:

$$\nabla_\theta J(\theta) = \nabla_\theta \sum_\tau p_\theta(\tau) R(\tau) = \sum_\tau \nabla_\theta p_\theta(\tau) R(\tau) \tag{4}$$

The gradient is on $p_\theta(\tau)$, which is the probability of a trajectory. To move this into an expectation that can be estimated from samples, we use the **log-derivative trick**: $\nabla_x f(x) = f(x) \nabla_x \log f(x)$. Applying this, we get:

$$\nabla_\theta p_\theta(\tau) = p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) \tag{5}$$

Substituting this back into our gradient expression:

$$\nabla_\theta J(\theta) = \sum_\tau p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) R(\tau) \tag{6}$$

$$= \mathbb{E}_{\tau\sim\pi_\theta} \left[\nabla_\theta \log p_\theta(\tau) R(\tau) \right] \tag{7}$$

This form is useful because it's an expectation, which we can approximate by sampling trajectories. Now, let's expand $\log p_\theta(\tau)$. The probability of a trajectory is given by:

$$p_\theta(\tau) = p(s_0) \prod_{t=0}^{T-1} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t) \tag{8}$$

The log of this probability is:

$$\log p_\theta(\tau) = \log p(s_0) + \sum_{t=0}^{T-1} \log \pi_\theta(a_t|s_t) + \sum_{t=0}^{T-1} \log p(s_{t+1}|s_t, a_t) \tag{9}$$

When we take the gradient with respect to θ, the terms that do not depend on θ (the initial state probability and the environment dynamics) become zero:

$$\nabla_\theta \log p_\theta(\tau) = \nabla_\theta \left(\sum_{t=0}^{T-1} \log \pi_\theta(a_t|s_t) \right) = \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \tag{10}$$

Plugging this result back into our expectation, we get a practical form for the policy gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta} \left[\left(\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \right) R(\tau) \right] \tag{11}$$

A key insight is that an action a_t taken at time step t can only influence rewards from that point forward $(r_t, r_{t+1}, \ldots, r_T)$. Past rewards are not affected. Therefore, we can replace the total trajectory reward $R(\tau)$ with the return-from-time-t, $G_t = \sum_{k=t}^{T} \gamma^{k-t} r_k$. This reduces variance

without changing the expected gradient. This leads to the final form of the **Policy Gradient Theorem**:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) G_t\right] \tag{12}$$

This theorem provides a computable expression for the policy gradient. It tells us to increase the probability of actions that lead to high future returns. The REINFORCE algorithm is a direct application of this theorem.

2. The REINFORCE Algorithm

REINFORCE is a foundational policy gradient algorithm that uses Monte Carlo methods to estimate the policy gradient.

2.1. Principle of REINFORCE

The objective is to find the parameters θ of a policy π_θ that maximize the expected total reward $J(\theta)$. The policy gradient theorem provides a way to compute the gradient of this objective:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) G_t\right] \tag{13}$$

where τ is a trajectory $(s_0, a_0, s_1, a_1, \dots)$, and G_t is the total discounted return from time step t onwards:

$$G_t = \sum_{k=t}^{T} \gamma^{k-t} r(s_k, a_k) \tag{14}$$

In practice, the expectation is approximated by sampling N trajectories using the current policy π_θ:

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t^i|s_t^i) G_t^i \tag{15}$$

The REINFORCE Algorithm Flow:

1. **Sample:** Generate N trajectories $\{\tau_1, \tau_2, \dots, \tau_N\}$ by executing the current policy π_θ in the environment.
2. **Compute Gradient:** Estimate the policy gradient using the sampled trajectories and their returns with the formula above.
3. **Update Parameters:** Update the policy parameters using gradient ascent:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta) \tag{16}$$

2.2. Advantages and Disadvantages

- **Advantages:** It is a simple and foundational algorithm that works directly with the policy, allowing it to handle continuous action spaces and learn stochastic policies.
- **Disadvantages:** The use of Monte Carlo estimation for the return G_t introduces **high variance**, which can lead to unstable training and slow convergence. It also operates on-policy and requires full episodes for updates, making it sample-inefficient.

2.3. REINFORCE with Baseline

To reduce the high variance, a state-dependent baseline $b(s_t)$ is subtracted from the return G_t. A common and effective choice for the baseline is the state-value function, $V(s_t)$. The policy gradient becomes:

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t^i|s_t^i)(G_t^i - b(s_t^i)) \tag{17}$$

This is an unbiased estimate of the gradient because the expectation of the baseline term is zero. Using $V(s_t)$ as the baseline means we are scaling the gradient by the **Advantage Function** $A(s_t, a_t) = G_t - V(s_t)$, which measures how much better an action is compared to the average action at that state.

3. Actor-Critic Methods

Actor-Critic (AC) methods combine the strengths of policy-based methods (the Actor) and value-based methods (the Critic). The Critic learns a value function to help the Actor update its policy more effectively.

3.1. The Actor-Critic Framework

- **The Actor:** A parameterized policy $\pi_\theta(a|s)$ that controls the agent's behavior.
- **The Critic:** A parameterized value function, e.g., $V_\phi(s)$ or $Q_\phi(s, a)$, that evaluates the states or state-action pairs.

Instead of waiting for an entire episode to get the return G_t, the Critic provides a lower-variance estimate.

3.2. Advantage Actor-Critic (A2C)

In A2C, the Critic learns the state-value function $V_\phi(s)$, and this is used to compute an estimate of the advantage function. The policy gradient update for the Actor is:

$$\nabla_\theta J(\theta) \approx \mathbb{E}_{\pi_\theta} \left[\nabla_\theta \log \pi_\theta(a_t|s_t) A(s_t, a_t)\right] \tag{18}$$

We don't know the true $V(s_t)$, so we use the Critic's estimate $V_\phi(s_t)$. The advantage can be estimated using the TD error:

$$A(s_t, a_t) \approx r(s_t, a_t) + \gamma V_\phi(s_{t+1}) - V_\phi(s_t) \tag{19}$$

This formulation only requires one neural network (the Critic) to estimate the value function V_ϕ.

The A2C Algorithm Flow:

1. **Interact:** Take an action $a_t \sim \pi_\theta(a|s_t)$ and observe reward r_t and next state s_{t+1}.
2. **Compute Advantage (Actor):** Calculate the advantage estimate: $\hat{A}_t = r_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t)$.
3. **Update Actor:** Update the policy parameters θ using the gradient:

$$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a_t|s_t) \hat{A}_t \tag{20}$$

4. **Update Critic:** Update the value function parameters ϕ by minimizing a loss function, such as the mean squared error between the estimated value and the TD target:

$$\mathcal{L}(\phi) = (r_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t))^2 \tag{21}$$

Typically, the Actor and Critic networks share lower-level layers for feature extraction, with separate output heads for the policy and value function.

4. Advanced AC: Asynchronous Methods

4.1. The Problem with Simple Batch Updates

Standard A2C can be run in batches. However, if a replay buffer is used (making it off-policy), the samples collected with an old policy can make the advantage estimates inaccurate and lead to instability.

4.2. Asynchronous Advantage Actor-Critic (A3C)

A3C addresses the correlation issue not with a replay buffer, but through parallelization.

- **Architecture:** A3C uses a central *global network* and multiple parallel *worker agents*, each with its own copy of the network parameters and its own instance of the environment.
- **Asynchronous Updates:** The workers interact with their environments in parallel, collecting diverse experiences. Each worker computes gradients locally based on its experience and sends these gradients to the global network.
- **Decorrelation:** The global network receives updates asynchronously. Since the workers are exploring different parts of the state space at different times, their experiences are less correlated. This diversity of experience stabilizes the learning process, effectively replacing the need for a replay buffer.

The A3C framework leads to faster and more stable training for deep reinforcement learning agents.

5. Deterministic Policy Gradient

Deterministic Policy Gradient (DPG) and its deep variant, Deep Deterministic Policy Gradient (DDPG), are fundamental actor–critic algorithms designed for continuous control. Unlike stochastic policy gradient methods such as REINFORCE, DPG employs a deterministic policy, enabling more efficient gradient estimation in high-dimensional action spaces.

5.1. Principle of Deterministic Policy Gradient (DPG)

DPG considers a deterministic policy $\mu_\theta : \mathcal{S} \to \mathcal{A}$ that maps each state s to an action $a = \mu_\theta(s)$. The objective is to maximize the expected return under the state distribution induced by the policy:

$$J(\theta) = \mathbb{E}_{s\sim\rho^\mu}\left[Q^\mu(s, \mu_\theta(s))\right]. \tag{22}$$

The deterministic policy gradient theorem states that the gradient of this objective can be computed as:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim\rho^\mu}\left[\nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s, a)\big|_{a=\mu_\theta(s)}\right]. \tag{23}$$

Compared with stochastic policy gradient, DPG avoids sampling from a distribution $\pi_\theta(a|s)$ and instead directly differentiates through the deterministic policy. This reduces variance and is especially effective in continuous action domains.

The DPG Algorithm Flow:

1. **Collect Transitions:** Interact with the environment using the deterministic policy $a_t = \mu_\theta(s_t)$, possibly with added noise for exploration.
2. **Critic Update:** Learn the action-value function using temporal-difference learning:

$$y_t = r_t + \gamma Q^\mu(s_{t+1}, \mu_\theta(s_{t+1})). \tag{24}$$

3. **Actor Update:** Apply the deterministic policy gradient:

$$\theta \leftarrow \theta + \alpha \mathbb{E}\left[\nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s, a)|_{a=\mu_\theta(s)}\right]. \tag{25}$$

5.2. Deep Deterministic Policy Gradient (DDPG)

Deep Deterministic Policy Gradient (DDPG) extends DPG to deep neural network parameterizations and introduces two key stabilization mechanisms inspired by Deep Q-Learning (DQN): *target networks* and *experience replay*.

DDPG maintains:

- an **actor network** $\mu_\theta(s)$,
- a **critic network** $Q_w(s, a)$,
- slowly-updated **target networks**, $\mu_{\theta'}$ and $Q_{w'}$.

The critic target is computed using the target networks:

$$y_t = r_t + \gamma Q_{w'}(s_{t+1}, \mu_{\theta'}(s_{t+1})). \tag{26}$$

The critic parameters are updated by minimizing the temporal-difference loss:

$$L(w) = \mathbb{E}\left[\left(Q_w(s_t, a_t) - y_t\right)^2\right]. \tag{27}$$

The actor is updated using the deterministic policy gradient, with the critic providing the gradient signal:

$$\nabla_\theta J(\theta) \approx \mathbb{E}_{s\sim\mathcal{D}}\left[\nabla_\theta \mu_\theta(s) \nabla_a Q_w(s,a)|_{a=\mu_\theta(s)}\right], \tag{28}$$

where $\mathcal{D}$ denotes the experience replay buffer.

Target networks are updated via soft updates to improve stability:

$$\theta' \leftarrow \tau\theta + (1-\tau)\theta', \qquad w' \leftarrow \tau w + (1-\tau)w', \tag{29}$$

with $\tau \ll 1$.

6. Twin Delayed Deep Deterministic Policy Gradient (TD3)

Twin Delayed Deep Deterministic Policy Gradient (TD3) is an improved actor–critic algorithm for continuous control that builds upon DDPG. TD3 addresses several critical sources of overestimation and instability in value-based continuous-control learning. The algorithm introduces three key techniques: *Clipped Double Q-learning*, *Target Policy Smoothing*, and *Delayed Policy Updates*.

6.1. Motivation and Principle

DDPG's critic tends to suffer from overestimation bias because the target incorporates $\max_a Q(s', a)$-like structures. Overestimation in continuous action domains becomes especially problematic due to function approximation and noise. TD3 mitigates these issues by redesigning the target computation and update schedule.

TD3 maintains:

- Two critic networks $Q_{w_1}(s, a)$ and $Q_{w_2}(s, a)$;
- One actor network $\mu_\theta(s)$;
- Corresponding target networks $Q_{w_1'}$, $Q_{w_2'}$ and $\mu_{\theta'}$.

6.2. Core Techniques of TD3

(1) Clipped Double Q-Learning

Instead of using a single critic, TD3 employs two critics and uses the minimum of the two target values to reduce overestimation:

$$y_t = r_t + \gamma \min_{i=1,2} Q_{w_i'}\left(s_{t+1}, \mu_{\theta'}(s_{t+1})\right). \tag{30}$$

This modification significantly stabilizes the learning process by providing a conservative target estimate.

(2) Target Policy Smoothing

To reduce sensitivity to sharp changes in the critic and prevent exploitative deterministic actions, TD3 adds noise to the target policy:

$$\tilde{a}_{t+1} = \mu_{\theta'}(s_{t+1}) + \epsilon, \qquad \epsilon \sim \mathrm{clip}(\mathcal{N}(0, \sigma^2), -c, c). \tag{31}$$

The target becomes:

$$y_t = r_t + \gamma \min_{i=1,2} Q_{w_i'}(s_{t+1}, \tilde{a}_{t+1}). \tag{32}$$

Smoothing the target reduces variance and makes the critic less sensitive to local irregularities.

(3) Delayed Policy Updates

TD3 updates the actor (and target networks) less frequently than the critics. If critics are updated every step but policy every d steps, the actor only updates when:

$$t \mod d = 0. \tag{33}$$

This decoupling prevents the actor from being updated based on an unstable critic.

6.3. TD3 Optimization Steps

Critic Update: TD3 updates the critic networks by minimizing the TD loss:

$$L(w_i) = \mathbb{E}\left[\left(Q_{w_i}(s_t, a_t) - y_t\right)^2\right], \qquad i = 1, 2. \tag{34}$$

Actor Update: Every d steps, the actor is updated using the deterministic policy gradient:

$$\nabla_\theta J(\theta) \approx \mathbb{E}_{s\sim\mathcal{D}}\left[\nabla_\theta \mu_\theta(s) \nabla_a Q_{w_1}(s, a)\big|_{a=\mu_\theta(s)}\right]. \tag{35}$$

Target Update: Soft updates ensure stability:

$$w_i' \leftarrow \tau w_i + (1-\tau) w_i', \qquad \theta' \leftarrow \tau\theta + (1-\tau)\theta'. \tag{36}$$

6.4. The TD3 Algorithm Flow

1. **Sample Transitions:** Execute $a_t = \mu_\theta(s_t) + \mathcal{N}_t$ and store (s_t, a_t, r_t, s_{t+1}) in replay buffer $\mathcal{D}$.
2. **Critic Learning:** Sample a minibatch from $\mathcal{D}$ and update both critics using the clipped double Q target.
3. **Delayed Actor Update:** Every d steps, update the actor using the deterministic policy gradient.
4. **Soft Target Updates:** Perform soft updates on the target critic and actor networks.

TD3 significantly improves sample efficiency and stability over DDPG, making it one of the most widely used benchmarks for continuous control tasks in modern reinforcement learning.

7. Generalized Advantage Estimation (GAE)

Generalized Advantage Estimation (GAE) is a variance-reduction technique for policy gradient methods. It provides a flexible and effective way to estimate the advantage function A_t, balancing bias and variance through a tunable parameter λ. GAE is widely used in modern policy-gradient algorithms such as TRPO and PPO due to its stability and sample efficiency.

7.1. Motivation

In policy gradient algorithms, the advantage function

$$A_t = Q^\pi(s_t, a_t) - V^\pi(s_t) \tag{37}$$

provides the learning signal for updating the policy. However, estimating Q^π or A_t accurately is challenging. Monte Carlo returns exhibit high variance, while bootstrapped TD estimates introduce bias.

GAE addresses this by combining multi-step returns into a smoothed estimator controlled by $\lambda \in [0, 1]$.

7.2. TD Residual and Multi-Step Advantages

The temporal-difference residual at time t is defined as:

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t). \tag{38}$$

The k-step advantage estimator is:

$$A_t^{(k)} = \sum_{l=0}^{k-1} (\gamma)^l \delta_{t+l}. \tag{39}$$

Shorter horizons (k small) reduce variance but increase bias; longer horizons reduce bias but increase variance. GAE merges all k-step estimators into a single weighted average.

7.3. Generalized Advantage Estimation

GAE defines the advantage estimate as an exponentially-weighted sum of multi-step estimators:

$$A_t^{\mathrm{GAE}(\gamma,\lambda)} = (1-\lambda) \sum_{k=1}^{\infty} \lambda^k A_t^{(k)} = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}. \tag{40}$$

This can be computed efficiently in reverse time:

$$A_t = \delta_t + \gamma\lambda A_{t+1}. \tag{41}$$

The parameter λ controls the bias–variance trade-off:

- $\lambda = 1$: recovers Monte Carlo advantages (low bias, high variance);
- $\lambda = 0$: reduces to one-step TD advantages (high bias, low variance).

By tuning λ, GAE provides a smoothly adjustable estimation method that improves the stability of policy gradient algorithms.

7.4. The GAE Computation Flow

1. **Collect Trajectories:** Gather trajectories $\{(s_t, a_t, r_t)\}$ using the current policy.
2. **Compute TD Residuals:** For each time step:
$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t). \tag{42}$$
3. **Compute GAE Advantages:** Starting from the end of the trajectory:
$$A_t = \delta_t + \gamma\lambda A_{t+1}. \tag{43}$$
4. **Compute Returns (Optional):** For actor–critic methods:
$$G_t = A_t + V(s_t). \tag{44}$$
5. **Policy Gradient Update:** GAE advantages are used as the learning signal in:
$$\nabla_\theta J(\theta) \propto \mathbb{E}\left[\nabla_\theta \log \pi_\theta(a_t|s_t) A_t\right]. \tag{45}$$

GAE has become a standard component in modern policy gradient reinforcement learning due to its strong empirical performance, providing a principled and tunable approach to balancing bias and variance in advantage estimation.

8. Trust Region Policy Optimization (TRPO)

While A2C and A3C provide significant improvements over simpler policy gradient methods, they can still be sensitive to hyperparameters like the learning rate. A single bad update can lead to a catastrophic drop in performance from which the policy may never recover. Trust Region Policy Optimization (TRPO) addresses this by ensuring that each policy update stays within a "trust region" where performance is guaranteed to improve.

8.1. Theoretical Foundation: Monotonic Improvement Guarantee

The derivation of TRPO starts from a fundamental identity that relates the performance of two policies, an old policy π and a new policy π'. The expected return $\eta(\pi)$ of a policy is given by $\mathbb{E}_{\tau\sim\pi}[R(\tau)]$. The performance difference can be expressed in terms of the advantage function $A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s)$ of the old policy:

$$\eta(\pi') - \eta(\pi) = \mathbb{E}_{\tau\sim\pi'}\left[\sum_{t=0}^{\infty}\gamma^t A_\pi(s_t,a_t)\right] \tag{46}$$

This equation is exact but difficult to optimize directly, as the expectation depends on the trajectory distribution of the new policy π', which is unknown.

Proof of the performance-difference identity

Let the expected return of a policy be

$$\eta(\pi) = \mathbb{E}_{\tau\sim\pi}\Big[\sum_{t=0}^{\infty}\gamma^t r_t\Big],$$

and recall the advantage of the *old* policy π:

$$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s), \qquad Q_\pi(s,a) = \mathbb{E}[r_t + \gamma V_\pi(s_{t+1}) \mid s_t = s, a_t = a].$$

Consider the discounted sum of advantages along a trajectory:

$$\sum_{t=0}^{\infty}\gamma^t A_\pi(s_t,a_t) = \sum_{t=0}^{\infty}\gamma^t\big(Q_\pi(s_t,a_t) - V_\pi(s_t)\big).$$

Substitute the one-step identity $Q_\pi(s_t,a_t) = r_t + \gamma V_\pi(s_{t+1})$:

$$\sum_{t=0}^{\infty}\gamma^t A_\pi(s_t,a_t) = \sum_{t=0}^{\infty}\gamma^t r_t + \sum_{t=0}^{\infty}\gamma^{t+1}V_\pi(s_{t+1}) - \sum_{t=0}^{\infty}\gamma^t V_\pi(s_t).$$

The two V_π-series telescope:

$$\sum_{t=0}^{\infty}\gamma^{t+1}V_\pi(s_{t+1}) - \sum_{t=0}^{\infty}\gamma^t V_\pi(s_t) = -V_\pi(s_0) + \lim_{T\to\infty}\gamma^{T+1}V_\pi(s_{T+1}).$$

Under the usual assumptions (bounded V_π and $0 \le \gamma < 1$, or finite horizon), the remainder term vanishes, giving

$$\sum_{t=0}^{\infty}\gamma^t A_\pi(s_t,a_t) = \sum_{t=0}^{\infty}\gamma^t r_t - V_\pi(s_0).$$

Taking expectation with respect to trajectories sampled from π' and noting $\mathbb{E}_{\tau\sim\pi'}[\sum_{t=0}^{\infty}\gamma^t r_t] = \eta(\pi')$ and $\mathbb{E}_{s_0\sim d_0}[V_\pi(s_0)] = \eta(\pi)$ (same initial distribution d_0), we obtain

$$\mathbb{E}_{\tau\sim\pi'}\Big[\sum_{t=0}^{\infty}\gamma^t A_\pi(s_t,a_t)\Big] = \eta(\pi') - \eta(\pi).$$

This is the desired identity.

TRPO makes a local approximation by replacing the state visitation distribution of the new policy with that of the old policy. This yields a *surrogate objective* $L_\pi(\pi')$:

$$L_\pi(\pi') = \mathbb{E}_{s\sim\rho_\pi, a\sim\pi'}\left[A_\pi(s,a)\right] = \mathbb{E}_{s\sim\rho_\pi}\left[\sum_a \pi'(a|s)A_\pi(s,a)\right] \tag{47}$$

where ρ_π is the discounted state visitation frequency under the old policy π. A small improvement in this surrogate objective leads to an improvement in the true objective $\eta(\pi')$.

Derivation of the TRPO objective

To relate the true performance difference to the surrogate objective, we decompose

$$\eta(\pi') - \eta(\pi) = \sum_s \rho_{\pi'}(s) \sum_a \pi'(a|s) A_\pi(s,a).$$

Adding and subtracting $\rho_\pi(s)$ gives

$$\eta(\pi') - \eta(\pi) = \underbrace{\sum_s \rho_\pi(s) \sum_a \pi'(a|s) A_\pi(s,a)}_{L_\pi(\pi')} + \underbrace{\sum_s \big(\rho_{\pi'}(s) - \rho_\pi(s)\big) \bar{A}_{\pi'}(s)}_{\Delta(\pi,\pi')},$$

where $\bar{A}_{\pi'}(s) = \sum_a \pi'(a|s) A_\pi(s,a)$. The term $L_\pi(\pi')$ is the surrogate objective, and $\Delta(\pi,\pi')$ captures the error caused by the mismatch in state visitation distributions.
To bound this error, we use that the discounted visitation difference satisfies

$$\sum_s |\rho_{\pi'}(s) - \rho_\pi(s)| \le \frac{2\gamma}{1-\gamma} \max_s D_{\mathrm{TV}}\big(\pi'(\cdot|s),\, \pi(\cdot|s)\big),$$

which implies

$$|\Delta(\pi,\pi')| \le \frac{2\gamma}{1-\gamma} \|A_\pi\|_\infty \max_s D_{\mathrm{TV}}\big(\pi'(\cdot|s),\, \pi(\cdot|s)\big).$$

Using the Pinsker inequality

$$D_{\mathrm{TV}}(p,q) \le \sqrt{\tfrac{1}{2} D_{\mathrm{KL}}(p\|q)},$$

we obtain a KL-based bound:

$$|\Delta(\pi,\pi')| \le \frac{2\gamma}{1-\gamma} \|A_\pi\|_\infty \sqrt{\tfrac{1}{2} \max_s D_{\mathrm{KL}}\big(\pi'(\cdot|s) \,\|\, \pi(\cdot|s)\big)}.$$

This shows that if π' stays close to π in per-state KL divergence, then maximizing the surrogate $L_\pi(\pi')$ approximately improves the true return $\eta(\pi')$.
TRPO therefore imposes a trust-region constraint on the expected KL divergence between the old and new policies, ensuring that the approximation error remains small.

The core theoretical result of TRPO is a lower bound on the true performance improvement:

$$\eta(\pi') \ge L_\pi(\pi') - C \cdot D_{KL}^{\max}(\pi,\pi') \tag{48}$$

where $D_{KL}^{\max}(\pi,\pi') = \max_s D_{KL}(\pi(\cdot|s)||\pi'(\cdot|s))$ is the maximum KL divergence between the policies over any state, and C is a constant that depends on the discount factor and the maximum advantage. This inequality provides a **monotonic improvement guarantee**: by maximizing this lower bound, we ensure that every policy update improves (or at least does not degrade) the true policy performance.

8.2. The TRPO Optimization Problem

Maximizing the lower bound leads to the following constrained optimization problem:

$$\underset{\theta'}{\text{maximize}} \quad L_{\pi_\theta}(\pi_{\theta'}) \quad \text{subject to} \quad \bar{D}_{KL}(\pi_\theta, \pi_{\theta'}) \le \delta \tag{49}$$

where $\bar{D}_{KL}$ is the average KL divergence over the states visited by the old policy, and δ is the trust region radius, a small hyperparameter that limits the size of the policy update.

To solve this practically, TRPO makes two approximations:

1. The objective $L_{\pi_\theta}(\pi_{\theta'})$ is approximated by its first-order Taylor expansion: $L(\theta') \approx g^T(\theta' - \theta)$, where g is the policy gradient.
2. The KL constraint is approximated by its second-order Taylor expansion: $\bar{D}_{KL}(\pi_\theta, \pi_{\theta'}) \approx \frac{1}{2}(\theta' - \theta)^T H (\theta' - \theta)$, where H is the **Fisher Information Matrix (FIM)**.

The FIM acts as a metric on the policy parameter space, measuring the sensitivity of the policy to changes in its parameters.

The resulting optimization step is:

$$\theta' \approx \theta + \sqrt{\frac{2\delta}{g^T H^{-1} g}} H^{-1} g \tag{50}$$

The update direction $H^{-1}g$ is known as the **natural gradient**. Computing the FIM and its inverse is computationally expensive for large networks. TRPO cleverly avoids this by using the **conjugate gradient algorithm** to efficiently solve the linear system $Hx = g$ for the search direction $x = H^{-1}g$, without ever explicitly forming the matrix H. A line search is then performed along this direction to ensure both the KL constraint and the surrogate objective improvement are satisfied.

Derivation of the TRPO update

Problem. let $p = \theta' - \theta$. Consider

$$\max_p \; g^\top p \quad \text{s.t.} \quad \tfrac{1}{2} p^\top H p \le \delta,$$

where $g = \nabla_\theta L_\pi(\theta)$ and H is the (positive definite) Fisher matrix.
Lagrangian. introduce $\lambda > 0$:

$$\mathcal{L}(p, \lambda) = g^\top p - \lambda\Big(\tfrac{1}{2} p^\top H p - \delta\Big).$$

First-order condition. $\nabla_p \mathcal{L} = 0$ gives

$$g - \lambda H p = 0 \quad \Longrightarrow \quad p = \frac{1}{\lambda} H^{-1} g.$$

Enforce constraint (active). substitute into the quadratic constraint:

$$\tfrac{1}{2}\Big(\frac{1}{\lambda^2} g^\top H^{-1} g\Big) = \delta \quad \Longrightarrow \quad \lambda = \sqrt{\frac{g^\top H^{-1} g}{2\delta}}.$$

Solution. hence

$$p^\star = \theta' - \theta = \frac{1}{\lambda} H^{-1} g = \sqrt{\frac{2\delta}{g^\top H^{-1} g}}\, H^{-1} g,$$

which is the stated TRPO update direction (a scaled natural gradient).

9. Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) is a family of algorithms that aim to achieve the data efficiency and reliable performance of TRPO, but using only first-order optimization. This makes PPO simpler to implement, more general, and more computationally efficient.

9.1. From TRPO's Constraint to PPO's Objective

PPO simplifies TRPO by reframing the constrained optimization problem into an unconstrained one that is easier to solve with standard stochastic gradient methods. Instead of a hard constraint, PPO uses a penalty or a clipping mechanism to discourage policy updates that are too large.

The most common variant of PPO uses a novel **clipped surrogate objective**. Let's define the probability ratio between the new and old policies as:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \tag{51}$$

The standard policy gradient surrogate objective is $L^{PG}(\theta) = \mathbb{E}_t[r_t(\theta)\hat{A}_t]$. PPO modifies this with

a clipping mechanism:

$$L^{CLIP}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta)\hat{A}_t, \quad \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right] \tag{52}$$

Here, ϵ is a small hyperparameter (e.g., 0.2) that defines the clipping range.

9.2. Intuition Behind the Clipped Objective

The clipping mechanism serves as a proxy for the trust region constraint. The logic depends on whether the advantage $\hat{A}_t$ is positive or negative:

- **When Advantage is Positive ($\hat{A}_t > 0$):** The action taken was better than average, and the objective is to increase its probability. The term $r_t(\theta)\hat{A}_t$ increases as $\pi_\theta(a_t|s_t)$ increases. The clipping function $\text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)$ caps the ratio $r_t(\theta)$ at $1+\epsilon$. The 'min' operator ensures that the update is clipped if it becomes too large, preventing an overly aggressive policy change.
- **When Advantage is Negative ($\hat{A}_t < 0$):** The action was worse than average, and the objective is to decrease its probability. Here, we want to decrease $r_t(\theta)$. The 'clip' term bounds the ratio from below at $1-\epsilon$. Because the advantage is negative, multiplying by it flips the inequality. The 'min' operator (which becomes a 'max' for negative values) ensures that the penalty for a bad action doesn't become excessively large if the policy change is too drastic.

By taking the minimum of the normal and the clipped objectives, PPO creates a pessimistic lower bound on the unconstrained objective, discouraging large deviations from the previous policy.

9.3. The PPO Algorithm

The full PPO algorithm combines this clipped objective for the policy (actor) with a loss term for the value function (critic) and an optional entropy bonus to encourage exploration. The typical PPO training loop is as follows:

1. Run the policy $\pi_{\theta_{\text{old}}}$ in the environment for T timesteps to collect a batch of data.
2. For each timestep in the batch, compute the advantage estimate $\hat{A}_t$ (often using Generalized Advantage Estimation, GAE, for lower variance).
3. For a number of epochs, optimize the PPO surrogate objective on minibatches from the collected data using a standard optimizer like Adam. The full objective is typically:

$$L(\theta) = \mathbb{E}_t \left[L^{CLIP}(\theta) - c_1(V_\theta(s_t) - V_t^{\text{target}})^2 + c_2 S[\pi_\theta](s_t) \right] \tag{53}$$

   where the second term is the value function loss and S is an entropy bonus.

PPO's simplicity, robustness, and strong empirical performance have made it a go-to algorithm in the field of deep reinforcement learning.

Part II
Multi Agents Reinforcement Learning

10. Value-based Method

10.1. IGM Principle

The Individual-Global-Max (IGM) principle is a key requirement for centralized training with decentralized execution (CTDE). It states that the global optimal joint action is obtained when each agent greedily maximizes its own utility:

$$\arg\max_{\mathbf{a}} Q_{\text{tot}}(\boldsymbol{\tau}, \mathbf{a}) = \begin{pmatrix} \arg\max_{a^1} Q^1(\tau^1, a^1) \\ \vdots \\ \arg\max_{a^N} Q^N(\tau^N, a^N) \end{pmatrix}. \tag{54}$$

To satisfy IGM, the mixing network must impose a monotonicity constraint:

$$\frac{\partial Q_{\text{tot}}}{\partial Q_i} \geq 0, \tag{55}$$

which guarantees that independent greedy action selection aligns with maximizing Q_{tot}.

Under this framework, value decomposition follows three components:

(1) Value Factorization

$$Q_{\text{tot}} = f_\theta(Q_1, \ldots, Q_N), \tag{56}$$

where $f_\theta(\cdot)$ defines the mixing architecture.

(2) TD Target

$$y = r + \gamma \max_{\mathbf{a}'} Q_{\text{tot}}(s', \mathbf{a}'; \theta^-). \tag{57}$$

(3) Parameter Update

$$\nabla_\theta \mathcal{L} = \big(Q_{\text{tot}} - y\big) \cdot \nabla_\theta Q_{\text{tot}}. \tag{58}$$

Different multi-agent value decomposition algorithms differ mainly in the design of the mixing function $f_\theta(\cdot)$.

10.2. Linear Approximation of Q_{tot}

The joint action-value function Q_{tot} is generally a highly nonlinear function of individual utilities $\{Q^i\}$. We show that, under mild regularity assumptions, Q_{tot} can be locally approximated by a weighted linear combination of individual Q^i, providing theoretical support for value-decomposition methods such as VDN, QMIX, and Qatten.

Taylor Expansion Around the Optimal Action Assume each agent's optimal action a_0^i satisfies the first-order optimality condition

$$\frac{\partial Q^i}{\partial a^i}\Big|_{a_0^i} = 0.$$

A second-order Taylor approximation yields

$$Q^i(a^i) \approx \alpha_i + \beta_i (a^i - a_0^i)^2, \tag{59}$$

where α_i is constant and the quadratic term captures local deviations.

Linearization of Cross-Terms Consider nonlinear interactions such as $Q^i Q^j$:

$$\begin{aligned} Q^i Q^j &\approx [\alpha_i + \beta_i (a^i - a_0^i)^2][\alpha_j + \beta_j (a^j - a_0^j)^2] \\ &\approx \alpha_i \alpha_j + \alpha_i (Q^j - \alpha_j) + \alpha_j (Q^i - \alpha_i) + \cdots, \end{aligned} \tag{60}$$

implying that higher-order interaction terms can be re-expressed as linear functions of individual Q^i.

Theorem (Local Linear Decomposition) In the neighborhood of the optimal joint action, the global action-value function admits the approximation:

$$Q_{\text{tot}}(s, \mathbf{a}) \approx c(s) + \sum_{i=1}^{N} \lambda_i(s)\, Q^i(s, a^i), \tag{61}$$

where coefficients $\lambda_i(s)$ absorb higher-order derivatives from the Taylor expansion.

Connection to Qatten Because $\lambda_i(s)$ depends on the global state and implicitly captures complex higher-order interactions, Qatten implements these coefficients using multi-head attention:

$$\lambda_{i,h}(s) \propto \exp\left(e_i^\top W_{k,h}^\top W_{q,h} e_s\right),$$

yielding per-head aggregated values

$$Q^h = \sum_{i=1}^{N} \lambda_{i,h}(s)\, Q^i.$$

The final joint value is produced by state-dependent nonnegative head weights w_h:

$$Q_{\text{tot}} \approx c(s) + \sum_{h=1}^{H} w_h \sum_{i=1}^{N} \lambda_{i,h}(s)\, Q^i. \tag{62}$$

This structure provides a learnable, monotonic, and expressive approximation to the locally linear form of Q_{tot}.

10.3. ResQ / ResZ: Core Formulation and Theoretical Basis

Objective Given the joint action-value (or return distribution) $Q_{jt}(\tau, \mathbf{u})$ or $Z_{jt}(\tau, \mathbf{u})$ in a cooperative Dec-POMDP, the goal is to obtain a decomposition that: (i) admits individual greedy policies consistent with the joint optimum (IGM / DIGM), and (ii) preserves high expressiveness for non-monotonic joint utilities.

Residual Decomposition ResQ represents the joint value as a main term plus a masked residual:

$$\boxed{Q_{jt}(\tau, \mathbf{u}) = Q_{\text{tot}}(\tau, \mathbf{u}) + w_r(\tau, \mathbf{u})\, Q_r(\tau, \mathbf{u})}$$

where the ideal mask is

$$w_r(\tau, \mathbf{u}) = \begin{cases} 0, & \mathbf{u} = \bar{\mathbf{u}} := \arg\max_{\mathbf{u}} Q_{jt}(\tau, \mathbf{u}), \\ 1, & \mathbf{u} \neq \bar{\mathbf{u}}, \end{cases} \qquad Q_r(\tau, \mathbf{u}) \leq 0.$$

The main term is required to be individually decomposable, e.g. a monotonic mixer:

$$\frac{\partial Q_{\text{tot}}}{\partial Q_i} \geq 0.$$

The distributional variant (ResZ) generalizes this to random returns:

$$\boxed{Z_{jt}(\tau, \mathbf{u}) = Z_{\text{tot}}(\tau, \mathbf{u}) + w_r(\tau, \mathbf{u})\, Z_r(\tau, \mathbf{u})}$$

with $\mathbb{E}[Z_r] \leq 0$, and main term

$$Z_{\text{tot}}(\tau, \mathbf{u}) = \sum_i k_i(\tau)\, Z_i(\tau_i, u_i), \qquad k_i(\tau) \geq 0,$$

ensuring distributional monotonicity and DIGM.

Key Theoretical Insight If $Q_r \leq 0$ and Q_{tot} is constructed from individual utilities in a monotonic way, then

$$\arg\max_{\mathbf{u}} Q_{\text{tot}}(\tau, \mathbf{u}) = \arg\max_{\mathbf{u}} Q_{jt}(\tau, \mathbf{u}),$$

so the optimal joint action is recovered by individual greedy policies. Moreover, any joint function Q can be represented by choosing Q_{tot} to match Q on optimal actions while pushing all non-decomposable structure into the negative residual.

Practical Mask Approximation Since $\bar{\mathbf{u}}$ is unknown, the mask is approximated using the main term:

$$\tilde{\mathbf{u}} = \arg\max_{\mathbf{u}} Q_{\text{tot}}(\tau, \mathbf{u}), \qquad w_r(\tau, \mathbf{u}) = \begin{cases} 0, & \mathbf{u} = \tilde{\mathbf{u}}, \\ 1, & \mathbf{u} \neq \tilde{\mathbf{u}}. \end{cases}$$

Training Objective ResQ minimizes

$$\mathcal{L} = \mathcal{L}^* + \mathcal{L}_{jt},$$

where

$$\mathcal{L}^* = \mathbb{E}\big[(y - Q_{jt}(\tau, \mathbf{u}))^2\big], \qquad y = r + \gamma\, Q_{jt}(\tau', \tilde{\mathbf{u}}; \theta^-),$$

and the decomposition consistency loss is

$$\mathcal{L}_{jt} = \mathbb{E}\Big[\big(Q_{jt} - Q_{\text{tot}} - w_r Q_r\big)^2\Big].$$

ResZ uses the analogous distributional TD objective (e.g., quantile or categorical projection).

10.4. QTRAN and QPLEX: Flexible Value Decomposition Beyond Monotonicity

QTRAN: Transformation-Based Decomposition QTRAN removes the monotonicity restriction by learning a transformed joint action-value $Q'_{jt}(\tau, \mathbf{u})$ that is consistent with individual utilities. The key idea is to impose two constraints:

$$\boxed{Q'_{jt}(\tau, \mathbf{u}) = \sum_i Q_i(\tau_i, u_i) + V_{\text{tran}}(\tau)} \tag{1}$$

$$\boxed{Q'_{jt}(\tau, \bar{\mathbf{u}}) = Q_{jt}(\tau, \bar{\mathbf{u}})}, \qquad Q'_{jt}(\tau, \mathbf{u}) \geq Q_{jt}(\tau, \mathbf{u}), \tag{2}$$

where $\bar{\mathbf{u}} = \arg\max_{\mathbf{u}} Q_{jt}(\tau, \mathbf{u})$. Thus, Q'_{jt} is an upper envelope of the true joint value but matches it at the optimal joint action, ensuring optimality consistency while allowing non-monotonic interactions. Training minimizes:

$$\mathcal{L}_{\text{QTRAN}} = \left\| Q'_{jt} - \sum_i Q_i - V_{\text{tran}} \right\|^2 + \big\| Q_{jt} - Q'_{jt} \big\|^2_{\mathbf{u}=\bar{\mathbf{u}}} + \text{ReLU}\big(Q_{jt} - Q'_{jt}\big).$$

QPLEX: Duplex Dueling Decomposition QPLEX builds on dueling networks by decomposing each agent's contribution into utility and advantage:

$$Q_i(\tau_i, u_i) = V_i(\tau_i) + A_i(\tau_i, u_i) - \max_{u_i} A_i(\tau_i, u_i). \tag{3}$$

The joint action-value is mixed via a monotonic attention-based mixer:

$$\boxed{Q_{jt}(\tau, \mathbf{u}) = \sum_i w_i(\tau, \mathbf{u}_{-i})\, A_i(\tau_i, u_i) + \sum_i V_i(\tau_i),} \tag{4}$$

where the non-negative weights $w_i(\tau, \mathbf{u}_{-i})$ are produced by a multi-head attention mechanism, permitting expressive modeling of inter-agent interactions while preserving the monotonicity needed for IGM. This "duplex" structure decouples state-dependent utilities from action-dependent advantages, enabling a more flexible yet IGM-consistent value factorization.

11. Policy-based Methods

11.1. Policy-Based Methods for Cooperative MARL

Policy-based approaches optimize a factored global policy directly rather than decomposing the joint action-value. A joint policy factorizes as:

$$\pi(\mathbf{a} \mid \mathbf{o}) = \prod_i \pi_i(a_i \mid o_i), \tag{1}$$

and is trained using centralized critics that encode global information.

Centralized Critic and Value Aggregation A global value or advantage function is learned:

$$V_{\text{tot}}(s), \qquad A_{\text{tot}}(s, \mathbf{a}), \tag{2}$$

where the critic aggregates agent-level information through a mixing structure:

$$V_{\text{tot}} = f_\phi(V_1, \ldots, V_N). \tag{3}$$

Different algorithms instantiate f_ϕ differently: (i) LICA infers latent inter-agent interactions, (ii) VMIX enforces monotonic value mixing analogous to QMIX, (iii) COMA uses a counterfactual baseline to compute advantages.

Critic Learning Objective The centralized value function is trained via TD regression:

$$\mathcal{L}_v = \left(V_{\text{tot}}(s) - y\right)^2, \qquad y = r + \gamma V_{\text{tot}}(s'). \tag{4}$$

Advantage Estimation (COMA) Counterfactual advantages isolate each agent's marginal contribution:

$$A_i(a_i) = Q(s, \mathbf{a}) - \sum_{a_i'} \pi_i(a_i' \mid o_i)\, Q(s, (a_i', a_{-i}))\,. \tag{5}$$

Policy Gradient Update The global policy is optimized with:

$$\nabla_\theta J = \mathbb{E}\left[A_{\text{tot}}(s, \mathbf{a}) \nabla_\theta \sum_i \log \pi_i(a_i \mid o_i) \right], \tag{6}$$

without requiring the monotonicity or IGM conditions needed by value-decomposition methods. This enables methods such as LICA to model arbitrary agent interactions while maintaining decentralized execution.

11.2. Policy-Gradient Baselines: MAPPO, HAPPO, and HATRPO

MAPPO Multi-Agent Proximal Policy Optimization extends PPO to multi-agent settings by employing a *centralized critic* and decentralized policies. Each agent maintains its own stochastic policy

$$\pi_i(a_i|o_i), \qquad \pi(\mathbf{a}|\mathbf{o}) = \prod_i \pi_i(a_i|o_i),$$

while a shared critic estimates the global value

$$V_{\text{tot}}(s) \approx V_\phi(s).$$

The MAPPO policy update follows the clipped PPO objective:

$$\max_{\theta_i} \mathbb{E}\Big[\min\left(r_i(\theta_i) A^{\text{tot}}(s, \mathbf{a}),\ \text{clip}(r_i(\theta_i), 1-\epsilon, 1+\epsilon) A^{\text{tot}}(s, \mathbf{a}) \right) \Big],$$

where $r_i(\theta_i) = \frac{\pi_i^{\theta_i}(a_i|o_i)}{\pi_i^{\theta_i^{\text{old}}}(a_i|o_i)}$ and the advantage is computed using the centralized critic.

11.3. Topology-Aware Multi-Agent Policy Gradients (TAPE)

TAPE introduces an explicit *agent topology* to regulate how individual policies interact during gradient updates. A topology is represented as a directed graph $G = (V, E)$, with adjacency matrix $E_{ij} \in \{0, 1\}$. An edge $e_{ij} \in E$ indicates that agent i's policy update depends on agent j's utility; self-dependencies are always present. This structure forms local *coalitions* that promote intra-coalition cooperation while attenuating the propagation of cross-agent credit deficiency.

Coalition Utility For each agent i, TAPE defines a coalition-based utility:

$$U_i = \sum_{j=1}^{n} E_{ij} U_j,$$

where each local aristocrat utility U_j is given by

$$U_j(s, a_j) = Q^{\phi}_{\text{tot}}(s, a) - \sum_{a'_j} \pi_j(a'_j|\tau_j)\, Q^{\phi}_{\text{tot}}(s, (a'_j, a_{-j})).$$

This filters out contributions from agents outside the coalition and stabilizes multi-agent credit assignment.

11.3.1 Stochastic TAPE

Under stochastic policies, the resulting policy gradient can be written as

$$\nabla J_{\text{sto}}(\theta) = \mathbb{E}_{\pi}\Big[\sum_i \nabla_{\theta_i} \log \pi_i(a_i|\tau_i)\, U_i\Big] = \mathbb{E}_{\pi}\Big[\sum_{i,j} E_{ij}\, k_j(s)\, \nabla_{\theta_i} \log \pi_i(a_i|\tau_i)\, Q^{\phi}_j(s, a_j)\Big],$$

where $k_j(s) \geq 0$ are mixing-network coefficients and Q^{ϕ}_j denotes local critics. The authors prove that for tabular policies, small-step updates using this gradient guarantee monotonic policy improvement.

11.3.2 Deterministic TAPE

For deterministic policies $a_i = \pi_i(\tau_i)$, TAPE defines a coalition-filtered value:

$$Q^{\text{co}}_i(s, a) = f_{\text{mix}}\big(s,\ \mathbf{1}[E_{i1}]\, Q^{\pi_1}_1,\ \ldots,\ \mathbf{1}[E_{in}]\, Q^{\pi_n}_n\big),$$

where the mixing network masks out all critics outside agent i's coalition.

The deterministic policy gradient then becomes

$$\nabla J_{\text{det}}(\theta) = \mathbb{E}\Big[\sum_i \nabla_{\theta_i} \pi_i(\tau_i)\ \nabla_{a_i} Q^{\text{co}}_i(s, a)\big|_{a_i = \pi_i(\tau_i)}\Big],$$

optionally using softened critics $\widehat{Q}^{\phi}_i = Q^{\phi}_i - \alpha \log \pi_i$ or auxiliary features.

11.3.3 Discussion

By restricting utility propagation through a topology, TAPE balances localized cooperation and mitigated credit diffusion. Random topologies (e.g., Erdős–Rényi) enhance gradient diversity, and the theory shows that exploration diversity scales with edge probability p. Empirically, TAPE consistently improves performance over independent-learning and value-decomposition counterparts across matrix games, LBF, and SMAC benchmarks.

Part III
Generative Model

12. Flows

Normalizing Flows are a class of generative models that aim to learn a complex data distribution $p_X(x)$ by transforming a simple, known base distribution $p_Z(z)$. Typically, the base distribution is a standard normal (Gaussian) distribution, $z \sim \mathcal{N}(0, I)$.

The core of a normalizing flow is a transformation $g : \mathbb{R}^d \to \mathbb{R}^d$, parameterized by θ, which must be an invertible function, also known as a diffeomorphism. This means both g and its inverse g^{-1} must exist and be differentiable.

This invertible nature defines two primary directions of operation:

1. **Generating Direction:** To generate a new data sample x, we first sample a latent variable z from the simple base distribution, $z \sim p_Z(z)$, and then apply the transformation to it.

   $$x = g(z; \theta)$$

   This process "pushes forward" the base density p_Z to a more complex density p_X.

2. **Normalizing Direction:** To evaluate the probability density of a given data point x, we apply the inverse transformation to map it back to the latent space.

   $$z = g^{-1}(x; \theta) = f(x; \theta)$$

   This process "normalizes" the complex data distribution p_X back into the simple base distribution p_Z. This direction is crucial for training, as it allows for exact likelihood computation.

To train such a model, we need a way to compute the probability density $p_X(x)$ given $p_Z(z)$ and the transformation g. This is made possible by the change of variables theorem.

12.1. The Change of Variables Theorem

The change of variables theorem provides the mathematical foundation for calculating the probability density of a variable that is a transformation of another variable with a known density.

One-Dimensional Case. Let Z be a scalar random variable with a probability density function (PDF) $p_Z(z)$. Let $X = g(Z)$ be another random variable, where g is an invertible, differentiable, and for simplicity, monotonically increasing function. The probability density of X, $p_X(x)$, can be derived as follows.

First, consider the cumulative distribution function (CDF) of X:

$$P(X < x) = P(g(Z) < x)$$

Since g is invertible and monotonically increasing, we can apply its inverse g^{-1} to both sides of the inequality:

$$P(X < x) = P(Z < g^{-1}(x)) = \int_{-\infty}^{g^{-1}(x)} p_Z(z)dz$$

The PDF is the derivative of the CDF with respect to x. Using the Leibniz integral rule and the chain rule, we get:

$$p_X(x) = \frac{d}{dx} \int_{-\infty}^{g^{-1}(x)} p_Z(z)dz = p_Z(g^{-1}(x)) \cdot \frac{d}{dx}(g^{-1}(x))$$

If g is not necessarily monotonic, we must use the absolute value of the derivative, leading to the general form:

$$p_X(x) = p_Z(g^{-1}(x)) \left| \frac{d}{dx} g^{-1}(x) \right|$$

Using the inverse function theorem, we know that $\frac{d}{dx} g^{-1}(x) = \frac{1}{g'(g^{-1}(x))} = \frac{1}{g'(z)}$. Substituting this in gives an alternative form:

$$p_X(x) = p_Z(z) \left| g'(z) \right|^{-1} \quad \text{where } z = g^{-1}(x)$$

High-Dimensional Case. The theorem generalizes to the case where $Z \in \mathbb{R}^d$ is a random vector. Let $x = g(z)$ be an invertible and differentiable mapping from $\mathbb{R}^d$ to $\mathbb{R}^d$. The derivative is replaced by the Jacobian matrix of the transformation. The Jacobian matrix of a function $f : \mathbb{R}^d \to \mathbb{R}^d$ is given by:

$$D_f(x) = \frac{\partial f}{\partial x} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_d} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_d}{\partial x_1} & \cdots & \frac{\partial f_d}{\partial x_d} \end{pmatrix}$$

The change in volume of an infinitesimal region around x when transformed by f is given by the absolute value of the determinant of the Jacobian, $|\det(D_f(x))|$. This term acts as the rescaling factor for the probability density.

Let $z = f(x) = g^{-1}(x)$. The change of variables formula for high dimensions is:

$$p_X(x) = p_Z(f(x)) \left|\det\left(D_f(x)\right)\right|$$

Equivalently, for the generating direction $x = g(z)$:

$$p_X(x) = p_Z(z) \left|\det\left(D_g(z)\right)\right|^{-1}$$

12.2. Training via Maximum Likelihood

With the change of variables formula, we can compute the exact log-likelihood of the data. Given a dataset $\mathcal{D} = \{x^{(i)}\}_{i=1}^N$, the log-likelihood is:

$$\mathcal{L}(\theta) = \sum_{i=1}^{N} \log p_X(x^{(i)}; \theta)$$

Substituting the change of variables formula with $f = g^{-1}$:

$$\log p_X(x; \theta) = \log p_Z(f(x; \theta)) + \log \left|\det\left(D_f(x; \theta)\right)\right|$$

The training objective is to find the parameters θ that maximize this log-likelihood:

$$\theta^* = \arg\max_{\theta} \sum_{i=1}^{N} \left(\log p_Z(f(x^{(i)}; \theta)) + \log \left| \det\left(D_f(x^{(i)}; \theta) \right) \right| \right)$$

This objective presents two main challenges:

1. The function f must be efficiently invertible.
2. The determinant of the Jacobian, $\det(D_f(x))$, must be computationally tractable. A naive calculation for a dense Jacobian costs $O(d^3)$.

12.3. Constructing Flow Architectures

The key to practical normalizing flows is to design layers (invertible functions) where these two challenges are addressed. Expressive models are then built by composing these simple layers.

Composition of Flows If we have a sequence of K invertible transformations (layers) $f_1, f_2, \ldots, f_K$, their composition $f = f_K \circ \cdots \circ f_2 \circ f_1$ is also an invertible function. The application of these layers is as follows:

$$x \xrightarrow{f_1} h_1 \xrightarrow{f_2} h_2 \xrightarrow{\cdots} h_{K-1} \xrightarrow{f_K} z$$

By the chain rule, the Jacobian of the composite function is the product of the Jacobians of the individual functions:

$$D_f(x) = D_{f_K}(h_{K-1}) \cdots D_{f_2}(h_1) D_{f_1}(x)$$

A crucial property of the determinant is that $\det(AB) = \det(A)\det(B)$. This means the log-determinant of the full transformation is simply the sum of the log-determinants of each layer:

$$\log \left|\det(D_f(x))\right| = \sum_{k=1}^{K} \log \left|\det\left(D_{f_k}(h_{k-1})\right)\right|$$

This allows us to construct deep, expressive models by stacking simple layers, each with an efficiently computable Jacobian determinant.

Coupling Layers (e.g., RealNVP, NICE) Coupling layers provide a powerful way to build complex transformations from simple ones. They achieve this by splitting the input vector x into two parts, x_a and x_b, and transforming one part based on the other, while leaving the first part unchanged. A standard affine coupling layer is defined as:

$$z_a = x_a$$
$$z_b = x_b \odot \exp(s(x_a)) + t(x_a)$$

where $s(\cdot)$ and $t(\cdot)$ are arbitrary complex functions (e.g., neural networks) that output vectors of the same dimension as x_b.

- **Invertibility:** This transformation is easily invertible without needing to invert s or t.

$$x_a = z_a$$
$$x_b = (z_b - t(z_a)) \odot \exp(-s(z_a))$$

- **Jacobian:** The Jacobian matrix of this transformation is block lower-triangular:

$$D_f(x) = \begin{pmatrix} \frac{\partial z_a}{\partial x_a} & \frac{\partial z_a}{\partial x_b} \\ \frac{\partial z_b}{\partial x_a} & \frac{\partial z_b}{\partial x_b} \end{pmatrix} = \begin{pmatrix} I & 0 \\ \frac{\partial z_b}{\partial x_a} & \text{diag}(\exp(s(x_a))) \end{pmatrix}$$

  The determinant of a triangular matrix is the product of its diagonal elements.

$$\det(D_f(x)) = \det(I) \cdot \det(\text{diag}(\exp(s(x_a)))) = \prod_j \exp(s(x_a)_j) = \exp\left(\sum_j s(x_a)_j\right)$$

  The log-determinant is therefore extremely efficient to compute:

$$\log|\det(D_f(x))| = \sum_j s(x_a)_j$$

To ensure all dimensions are transformed, consecutive coupling layers typically swap the roles of which part is transformed and which is used for conditioning.

12.4. Continuous Normalizing Flows (Neural ODEs)

Instead of a discrete sequence of transformations, we can conceptualize the flow as a continuous-time process governed by an Ordinary Differential Equation (ODE). Let $z(t)$ be a continuous vector-valued function of time t. Its dynamics are defined by a neural network f:

$$\frac{dz(t)}{dt} = f(z(t), t, \theta)$$

Given an initial value $z(t_0)$, we can find the value at a later time t_1 by solving this ODE:

$$z(t_1) = z(t_0) + \int_{t_0}^{t_1} f(z(t), t, \theta) dt$$

In the context of normalizing flows, we can treat the data point x as the state at $t = t_0$ and the latent variable z as the state at $t = t_1$.

$$x = z(t_0) \quad \text{and} \quad z = z(t_1)$$

The transformation is now the ODE solver itself. Under mild Lipschitz continuity conditions on f, the solution to the ODE exists and is unique, making the transformation invertible by solving the ODE backwards in time.

The change of variables formula for this continuous transformation is given by:

$$\log p_X(x) = \log p_Z(z(t_1)) - \int_{t_0}^{t_1} \text{Tr}\left(\frac{\partial f}{\partial z(t)}\right) dt$$

where $\text{Tr}(\cdot)$ is the trace of the Jacobian matrix. This replaces the sum of log-determinants with an integral of a matrix trace. The trace is computationally much cheaper ($O(d)$ using Hutchinson's estimator) than the determinant ($O(d^3)$), making continuous flows very efficient in this regard. The main computational cost shifts to the numerical ODE solver.

13. Denoising Diffusion Probabilistic Models (DDPM)

Diffusion models are a class of likelihood-based generative models. They consist of two main processes: a fixed **forward process** that gradually adds noise to a data sample until it becomes pure noise, and a learned **reverse process** that denoises a sample from a simple distribution back into a sample from the data distribution.

13.1. Forward Process (Diffusion Process)

The forward process is a Markov chain that introduces Gaussian noise to a data sample $x_0 \sim q(x)$ over T timesteps, according to a predefined variance schedule.

Let's define a sequence of latent variables $x_1, \ldots, x_T$ where each x_t is a slightly noisier version of x_{t-1}. The transition kernel is defined as a Gaussian:

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I)$$

Here, $\{\beta_t\}_{t=1}^T$ is a variance schedule, where $\beta_t \in (0,1)$ are small constants. Let $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{k=1}^t \alpha_k$. Then the above can be rewritten as:

$$x_t = \sqrt{\alpha_t}x_{t-1} + \sqrt{1-\alpha_t}\epsilon_{t-1}, \quad \text{where } \epsilon_{t-1} \sim \mathcal{N}(0, I)$$

A significant property of this process is that we can directly sample x_t at any timestep t from the initial data point x_0, without having to iterate through the intermediate steps. Using the reparameterization trick repeatedly:

$$\begin{aligned} x_t &= \sqrt{\alpha_t}x_{t-1} + \sqrt{1-\alpha_t}\epsilon_{t-1} \\ &= \sqrt{\alpha_t}(\sqrt{\alpha_{t-1}}x_{t-2} + \sqrt{1-\alpha_{t-1}}\epsilon_{t-2}) + \sqrt{1-\alpha_t}\epsilon_{t-1} \\ &= \sqrt{\alpha_t\alpha_{t-1}}x_{t-2} + \sqrt{\alpha_t(1-\alpha_{t-1})}\epsilon_{t-2} + \sqrt{1-\alpha_t}\epsilon_{t-1} \end{aligned}$$

Since the sum of two Gaussians is also a Gaussian, we can combine the noise terms. By applying this unrolling process all the way back to x_0, we get the closed-form expression:

$$x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, \quad \text{where } \epsilon \sim \mathcal{N}(0, I)$$

This gives us the marginal distribution of x_t given x_0:

$$q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)I)$$

As $t \to T$, the term $\bar{\alpha}_T \to 0$, which means that x_T becomes nearly indistinguishable from a standard Gaussian distribution, i.e., $q(x_T|x_0) \approx \mathcal{N}(0, I)$.

13.2. Reverse Process and Objective Function

The goal of the reverse process is to learn the data distribution by reversing the noise process. Starting from $x_T \sim \mathcal{N}(0, I)$, we want to learn a model $p_\theta(x_{t-1}|x_t)$ that can generate x_0. This reverse process is also defined as a Markov chain:

$$p_\theta(x_{0:T}) = p(x_T)\prod_{t=1}^T p_\theta(x_{t-1}|x_t)$$

We train the model by maximizing the log-likelihood of the data, $\log p_\theta(x_0)$. To make this tractable, we optimize its Evidence Lower Bound (ELBO), derived using Jensen's inequality:

$$\begin{aligned} \log p_\theta(x_0) &\geq \mathbb{E}_{q(x_{1:T}|x_0)}\left[\log \frac{p_\theta(x_{0:T})}{q(x_{1:T}|x_0)}\right] \\ &= \mathbb{E}_q\left[\log p(x_T) + \sum_{t=1}^T \log p_\theta(x_{t-1}|x_t) - \sum_{t=1}^T \log q(x_t|x_{t-1})\right] \\ &= \mathbb{E}_q\left[\log \frac{p(x_T)}{q(x_T|x_0)} + \sum_{t=2}^T \log \frac{p_\theta(x_{t-1}|x_t)}{q(x_t|x_{t-1})} + \log p_\theta(x_0|x_1)\right] \end{aligned}$$

Using the property that $q(x_t|x_{t-1}) = q(x_t|x_{t-1}, x_0)$ and rewriting the reverse conditional $q(x_{t-1}|x_t, x_0)$ using Bayes' rule, we can reformulate the terms inside the sum:

$$\begin{aligned}\log \frac{p_\theta(x_{t-1}|x_t)}{q(x_t|x_{t-1})} &= \log \left(p_\theta(x_{t-1}|x_t) \cdot \frac{q(x_{t-1}|x_0)}{q(x_t|x_0)} \cdot \frac{1}{q(x_{t-1}|x_t, x_0)} \right) \\ &= \log \frac{p_\theta(x_{t-1}|x_t)}{q(x_{t-1}|x_t, x_0)} + \log \frac{q(x_{t-1}|x_0)}{q(x_t|x_0)}\end{aligned}$$

After complex cancellations, the ELBO can be expressed as a sum of KL divergences:

$$L_{VLB} = \underbrace{D_{KL}(q(x_T|x_0)\|p(x_T))}_{L_T} + \sum_{t=2}^{T} \underbrace{D_{KL}(q(x_{t-1}|x_t, x_0)\|p_\theta(x_{t-1}|x_t))}_{L_{t-1}} - \underbrace{\log p_\theta(x_0|x_1)}_{L_0}$$

The training objective is to minimize the negative ELBO. Let's analyze each term:

- L_T: The prior matching term. Since the forward process q is fixed and $p(x_T)$ is set to $\mathcal{N}(0, I)$, this term has no trainable parameters and can be ignored during training.
- L_0: The reconstruction term, which tries to reconstruct the original data x_0 from the slightly noised x_1.
- L_{t-1}: The denoising matching terms. These terms are the core of the training, aiming to make the learned reverse transition $p_\theta(x_{t-1}|x_t)$ match the true posterior $q(x_{t-1}|x_t, x_0)$.

13.3. Parameterizing the Reverse Process and Simplifying the Objective

The central challenge is to model $p_\theta(x_{t-1}|x_t)$. Since $q(x_{t-1}|x_t, x_0)$ is a Gaussian (as we will derive), we choose $p_\theta(x_{t-1}|x_t)$ to also be a Gaussian with a learned mean and a fixed variance:

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \sigma_t^2 I)$$

To set the target for our learned mean μ_θ, we first need to find the analytical form of the true posterior mean from $q(x_{t-1}|x_t, x_0)$.

13.3.1 Deriving the Posterior $q(x_{t-1}|x_t, x_0)$

Using Bayes' rule, the posterior is proportional to:

$$q(x_{t-1}|x_t, x_0) = \frac{q(x_t \mid x_{t-1}, x_0)\, q(x_{t-1} \mid x_0)}{q(x_t \mid x_0)} \propto q(x_t|x_{t-1}, x_0) q(x_{t-1}|x_0)$$

We know the forms of the two distributions on the right:

$$\begin{aligned} q(x_t|x_{t-1}, x_0) &= \mathcal{N}(x_t; \sqrt{\alpha_t} x_{t-1}, (1-\alpha_t) I) \\ q(x_{t-1}|x_0) &= \mathcal{N}(x_{t-1}; \sqrt{\bar{\alpha}_{t-1}} x_0, (1-\bar{\alpha}_{t-1}) I) \end{aligned}$$

The log of the posterior is therefore (ignoring constants):

$$\begin{aligned}\log q(x_{t-1}|x_t, x_0) &= C - \frac{1}{2(1-\alpha_t)} \|x_t - \sqrt{\alpha_t} x_{t-1}\|^2 - \frac{1}{2(1-\bar{\alpha}_{t-1})} \|x_{t-1} - \sqrt{\bar{\alpha}_{t-1}} x_0\|^2 \\ &= C - \frac{1}{2} \left(\frac{1}{1-\alpha_t} (x_t^2 - 2\sqrt{\alpha_t} x_t x_{t-1} + \alpha_t x_{t-1}^2) + \frac{1}{1-\bar{\alpha}_{t-1}} (x_{t-1}^2 - 2\sqrt{\bar{\alpha}_{t-1}} x_0 x_{t-1} + \bar{\alpha}_{t-1} x_0^2) \right)\end{aligned}$$

This is a quadratic function of x_{t-1}. We can complete the square to find the mean and variance of this Gaussian. The terms involving x_{t-1} are:

$$-\frac{1}{2} \left[\left(\frac{\alpha_t}{1-\alpha_t} + \frac{1}{1-\bar{\alpha}_{t-1}} \right) x_{t-1}^2 - 2 \left(\frac{\sqrt{\alpha_t}}{1-\alpha_t} x_t + \frac{\sqrt{\bar{\alpha}_{t-1}}}{1-\bar{\alpha}_{t-1}} x_0 \right) x_{t-1} \right]$$

The posterior is a Gaussian $\mathcal{N}(x_{t-1}; \mu_q, \tilde{\beta}_t I)$, where the variance $\tilde{\beta}_t$ is the inverse of the coefficient of x_{t-1}^2, and the mean μ_q is the coefficient of x_{t-1} multiplied by the variance. After algebraic simplification:

$$\begin{aligned}\tilde{\beta}_t &= \left(\frac{\alpha_t}{1-\alpha_t} + \frac{1}{1-\bar{\alpha}_{t-1}} \right)^{-1} = \boxed{\frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t} (1-\alpha_t)} \\ \mu_q(x_t, x_0) &= \left(\frac{\sqrt{\alpha_t}}{1-\alpha_t} x_t + \frac{\sqrt{\bar{\alpha}_{t-1}}}{1-\bar{\alpha}_{t-1}} x_0 \right) \tilde{\beta}_t = \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t} x_t + \frac{\sqrt{\bar{\alpha}_{t-1}}(1-\alpha_t)}{1-\bar{\alpha}_t} x_0\end{aligned}$$

13.3.2 Simplifying the Objective

The KL divergence term L_{t-1} between two Gaussians with means (μ_q, μ_θ) and fixed variances simplifies to a mean squared error on the means:

$$L_{t-1} = \mathbb{E}_{q(x_t|x_0)}\left[\frac{1}{2\sigma_t^2}\|\mu_q(x_t, x_0) - \mu_\theta(x_t, t)\|^2\right] + C$$

The model $\mu_\theta(x_t, t)$ must predict $\mu_q(x_t, x_0)$. However, μ_q depends on x_0, which is unknown at inference time.

KL between Gaussians ⇒ MSE on means

Assume both distributions are Gaussians with the same (isotropic) covariance $\sigma_t^2 I$:

$$q(x_{t-1} \mid x_t, x_0) = \mathcal{N}(\mu_q, \sigma_t^2 I), \qquad p_\theta(x_{t-1} \mid x_t) = \mathcal{N}(\mu_\theta, \sigma_t^2 I).$$

The closed form KL divergence between two Gaussians $\mathcal{N}(\mu_1, \Sigma)$ and $\mathcal{N}(\mu_2, \Sigma)$ with identical covariance Σ is

$$D_{\mathrm{KL}}\big(\mathcal{N}(\mu_1, \Sigma) \,\|\, \mathcal{N}(\mu_2, \Sigma)\big) = \tfrac{1}{2}(\mu_2 - \mu_1)^\top \Sigma^{-1}(\mu_2 - \mu_1).$$

Plugging $\Sigma = \sigma_t^2 I$ gives

$$D_{\mathrm{KL}}\big(q \| p_\theta\big) = \frac{1}{2\sigma_t^2}\|\mu_q - \mu_\theta\|^2.$$

Taking the expectation under $q(x_t \mid x_0)$ yields the objective term

$$L_{t-1} = \mathbb{E}_{q(x_t|x_0)}\big[D_{\mathrm{KL}}(q \| p_\theta)\big] = \mathbb{E}_{q(x_t|x_0)}\left[\frac{1}{2\sigma_t^2}\|\mu_q(x_t, x_0) - \mu_\theta(x_t, t)\|^2\right],$$

up to an additive constant C that does not depend on θ (e.g. terms from differing variances or dimension constants). Hence the KL reduces to a weighted mean-squared error on the means.

We need to re-parameterize the mean. Let's substitute $x_0 = \frac{1}{\sqrt{\bar{\alpha}_t}}(x_t - \sqrt{1-\bar{\alpha}_t}\epsilon)$:

$$\mu_q(x_t, \epsilon) = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon\right)$$

Now, instead of predicting the mean μ_q, the neural network $\epsilon_\theta(x_t, t)$ is trained to predict the noise component ϵ. The learned mean is then constructed as:

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t)\right)$$

Substituting this back into the loss term for the means:

$$\begin{aligned}\|\mu_q - \mu_\theta\|^2 &= \left\|\frac{1}{\sqrt{\alpha_t}}\left(\cdots - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon\right) - \frac{1}{\sqrt{\alpha_t}}\left(\cdots - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta\right)\right\|^2 \\ &= \frac{(1-\alpha_t)^2}{\alpha_t(1-\bar{\alpha}_t)}\|\epsilon - \epsilon_\theta(x_t, t)\|^2\end{aligned}$$

Ignoring the weighting factors which only affect learning rate, the simplified objective function for training at each step becomes:

$$L_{simple}(\theta) = \mathbb{E}_{t, x_0, \epsilon}\left[\|\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t)\|^2\right]$$

This elegant result means the entire complex training process simplifies to training a neural network to predict the noise that was added to an image at a given timestep.

Algorithm 1 Training

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

Algorithm 2 Sampling

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

Figure 1: The algorithm of DDPM.

14. Denoising Diffusion Implicit Models (DDIM)

DDIMs were developed based on a crucial observation: the DDPM training objective only depends on the marginal distribution $q(x_t|x_0)$, not the joint distribution of the entire trajectory $q(x_{1:T}|x_0)$. This insight opens the door to designing different reverse processes that are no longer strictly Markovian but still use the same trained model $\epsilon_\theta(x_t, t)$.

14.1. The Core Derivation of the DDIM Update Rule

The DDIM generative process is derived by first predicting an estimate of the clean image, $\hat{x}_0$, from the noisy image x_t. By rearranging former Equation, we get a direct way to compute this prediction:

$$\hat{x}_0 = \frac{x_t - \sqrt{1-\bar{\alpha}_t}\,\epsilon_\theta(x_t, t)}{\sqrt{\bar{\alpha}_t}} \tag{63}$$

This is a critical step. Instead of only reasoning about x_{t-1} from x_t, we first jump to an estimate of the "origin" x_0. With this $\hat{x}_0$, we can now use the forward process definition to move to any preceding timestep $t-1$. The full DDIM update equation combines these steps:

$$x_{t-1} = \underbrace{\sqrt{\bar{\alpha}_{t-1}} \left(\frac{x_t - \sqrt{1-\bar{\alpha}_t}\epsilon_\theta(x_t, t)}{\sqrt{\bar{\alpha}_t}} \right)}_{\text{Part 1: Denoised image re-scaled to } t-1} + \underbrace{\sqrt{1-\bar{\alpha}_{t-1}-\sigma_t^2} \cdot \epsilon_\theta(x_t, t)}_{\text{Part 2: Directional noise pointing to } x_t} + \underbrace{\sigma_t \epsilon}_{\text{Part 3: Random noise}} \tag{64}$$

where $\epsilon \sim \mathcal{N}(0, I)$ and σ_t is a parameter controlling stochasticity.

14.2. The Mechanism for Skipping Steps (Accelerated Sampling)

The true power of DDIM comes from the realization that because we can predict $\hat{x}_0$ from any x_t, we are not restricted to sampling the immediately preceding step x_{t-1}. We can sample from an arbitrary subsequence of timesteps $\mathcal{S} = (\tau_1, \tau_2, \dots, \tau_S)$ where $T \geq \tau_1 > \tau_2 > \dots > \tau_S > 0$ and the number of sampling steps S can be much smaller than T (e.g., $S = 50$ instead of $T = 1000$).

The generative process becomes a chain that transitions from x_{τ_i} to $x_{\tau_{i+1}}$. The generalized update rule is a direct application of Equation (7), where we replace t with τ_i and $t-1$ with τ_{i+1}:

$$x_{\tau_{i+1}} = \sqrt{\bar{\alpha}_{\tau_{i+1}}} \underbrace{\left(\frac{x_{\tau_i} - \sqrt{1-\bar{\alpha}_{\tau_i}}\epsilon_\theta(x_{\tau_i}, \tau_i)}{\sqrt{\bar{\alpha}_{\tau_i}}} \right)}_{\text{predicted } \hat{x}_0 \text{ from } x_{\tau_i}} + \sqrt{1-\bar{\alpha}_{\tau_{i+1}}-\sigma_{\tau_i}^2} \cdot \epsilon_\theta(x_{\tau_i}, \tau_i) + \sigma_{\tau_i}\epsilon \tag{65}$$

This is the mathematical embodiment of "skipping steps". The model predicts the final clean image $\hat{x}_0$ from the current noise level τ_i, and then uses that prediction to intelligently sample an image at a much earlier noise level τ_{i+1}, bypassing all intermediate steps.

14.3. Controlling Stochasticity with η

The amount of random noise in each step is controlled by σ_t. This parameter can be conveniently expressed in terms of a single hyperparameter η:

$$\sigma_t^2 = \eta \frac{(1-\bar{\alpha}_{t-1})}{(1-\bar{\alpha}_t)} \left(1 - \frac{\bar{\alpha}_t}{\bar{\alpha}_{t-1}} \right) \tag{66}$$

This parameter η smoothly interpolates between two important types of generative processes:

- When $\eta = 1$, the process becomes fully stochastic and is equivalent to the DDPM formulation. This maximizes the randomness at each step.
- When $\eta = 0$, we have $\sigma_t = 0$ for all t. The random noise term (Part 3) in the update rule vanishes. The entire process becomes **deterministic**. Given a starting noise vector x_T, the denoising trajectory is fixed, producing the exact same final image every time.

This deterministic nature is a significant advantage. It ensures that samples conditioned on the same latent variable x_T will have similar high-level features. This "consistency" property enables semantically meaningful interpolation and manipulation in the latent space.

15. A Different Perspective: Score-based Generative Models

An alternative viewpoint frames diffusion models as a specific instance of score-based generative models. The core concept in this framework is the **score function**, defined as the gradient of the log-probability density with respect to the data, $\nabla_x \log p(x)$. This function points in the direction where the data probability density increases most steeply.

The connection between diffusion models and score matching is established by **Tweedie's Formula**. For a Gaussian variable $z \sim \mathcal{N}(\mu_z, \sigma_z^2)$, the formula relates the posterior mean of μ_z to the score of the marginal distribution $p(z)$:

$$\mathbb{E}[\mu_z|z] = z + \sigma_z^2 \nabla_z \log p(z) \tag{67}$$

We can apply this to the forward process of a diffusion model, where x_t is a noisy version of the initial data x_0. The conditional distribution $q(x_t|x_0)$ is a Gaussian:

$$q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)I) \tag{68}$$

The log-likelihood is $\log q(x_t|x_0) = -\frac{1}{2(1-\bar{\alpha}_t)}\|x_t - \sqrt{\bar{\alpha}_t}x_0\|^2 + C$. The score of this conditional distribution is the gradient with respect to x_t:

$$\nabla_{x_t} \log q(x_t|x_0) = -\frac{x_t - \sqrt{\bar{\alpha}_t}x_0}{1-\bar{\alpha}_t} \tag{69}$$

Using the forward process definition $x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon_t$, we can express the numerator as $x_t - \sqrt{\bar{\alpha}_t}x_0 = \sqrt{1-\bar{\alpha}_t}\epsilon_t$. Substituting this into the score equation yields:

$$\nabla_{x_t} \log q(x_t|x_0) = -\frac{\sqrt{1-\bar{\alpha}_t}\epsilon_t}{1-\bar{\alpha}_t} = -\frac{\epsilon_t}{\sqrt{1-\bar{\alpha}_t}} \tag{70}$$

This reveals a crucial link: predicting the noise ϵ_t added at step t is equivalent to learning the score function of the noisy data distribution $p(x_t)$, up to a scaling factor. Therefore, the diffusion model's training can be interpreted as a form of score matching, and the reverse process is analogous to Langevin dynamics, which uses the learned score function to iteratively sample from the data distribution.

16. Conditional Diffusion Models

To control the generation process and create content that adheres to specific instructions (e.g., text descriptions, class labels), we introduce conditioning into the model.

16.1. Classifier-Guided Diffusion

This approach explicitly guides the sampling process using a separate, pre-trained classifier. A classifier $p_\phi(c|x_t)$ is trained to predict the correct class c from a noisy input x_t. During generation, the gradient of the classifier's log-probability is used to "steer" the denoising process toward the desired class.

The score of the joint distribution $p(x_t, c)$ can be expressed using the chain rule: $\log p(x_t, c) = \log p(x_t) + \log p(c|x_t)$. Taking the gradient gives:

$$\nabla_{x_t} \log p(x_t, c) = \nabla_{x_t} \log p(x_t) + \nabla_{x_t} \log p_\phi(c|x_t) \tag{71}$$

The unconditional score $\nabla_{x_t} \log p(x_t)$ is what the diffusion model approximates. The reverse process is then modified to incorporate the classifier's gradient. This leads to a modified noise prediction:

$$\hat{\epsilon}_\theta(x_t, c, t) = \epsilon_\theta(x_t, t) - \sqrt{1 - \bar{\alpha}_t}\omega \nabla_{x_t} \log p_\phi(c|x_t) \tag{72}$$

where ω is a guidance scale that controls the strength of the classifier's influence.

16.2. Classifier-Free Guidance

Training and maintaining a separate classifier for noisy images can be complex and computationally expensive. Classifier-Free Guidance (CFG) is a technique that achieves conditional generation with a single neural network, eliminating the need for an external classifier.

The model $\epsilon_\theta(x_t, t, c)$ is trained on conditional inputs. During training, the conditioning information c (e.g., a class label or text embedding) is randomly replaced with a null token $\emptyset$ with some probability. This forces the model to learn both the conditional and unconditional distributions simultaneously.

During inference, the model makes two predictions: one with the desired condition c, $\epsilon_\theta(x_t, t, c)$, and one without it, $\epsilon_\theta(x_t, t, \emptyset)$. The final noise prediction is an extrapolation from these two, moving further in the direction of the conditional prediction:

$$\hat{\epsilon}_\theta(x_t, t, c) = \epsilon_\theta(x_t, t, \emptyset) + w \cdot (\epsilon_\theta(x_t, t, c) - \epsilon_\theta(x_t, t, \emptyset)) \tag{73}$$

This can be rewritten for efficient implementation as:

$$\hat{\epsilon}_\theta(x_t, t, c) = (1 + w)\epsilon_\theta(x_t, t, c) - w\epsilon_\theta(x_t, t, \emptyset) \tag{74}$$

where w is the guidance weight. A higher value of w pushes the generation to adhere more strongly to the condition c. This approach is highly effective and has become the de facto standard for high-quality conditional image synthesis.