# 直接偏好优化（DPO）

Fred Sun

08/12/2025

## Contents

# 1 引言与动机

## 1.1 RLHF 的回顾与局限性

在上一讲中，我们详细介绍了 RLHF（Reinforcement Learning from Human Feedback）的完整流程，包括：

1. **监督微调（SFT）**：用高质量数据微调预训练模型，得到初始策略 $\pi_{\text{SFT}}$
2. **奖励模型训练**：从人类偏好数据学习奖励函数 $r_\phi(x, y)$
3. **PPO 优化**：用奖励模型的信号，通过 PPO 算法优化策略

虽然 RLHF 在实践中取得了巨大成功（如 ChatGPT、Claude 等），但它存在一些显著的局限性：

**注记 1.1** (RLHF 的工程复杂性)**.**

1. **需要维护多个模型**：PPO 训练阶段需要同时加载 4 个模型：
   - 策略模型 $\pi_\theta$（Actor，待优化）
   - 参考模型 $\pi_{\text{ref}}$（冻结，用于 KL 惩罚）
   - 奖励模型 $r_\phi$（冻结，用于计算奖励）
   - 值函数模型 $V_\psi$（Critic，待优化）

   这对 GPU 内存提出了很高要求。
2. **训练不稳定**：PPO 涉及多个超参数（裁剪系数 $\epsilon$、KL 惩罚系数 $\beta$、GAE 参数 $\lambda$ 等），调参困难。
3. **采样效率低**：PPO 是在线算法，需要不断用当前策略采样新数据，无法充分利用离线偏好数据。
4. **实现复杂**：需要实现 GAE、裁剪目标、值函数训练等多个组件，代码复杂度高。

## 1.2 DPO 的核心思想

直接偏好优化（Direct Preference optimisation, DPO）的核心思想是：

**绕过显式的奖励建模和强化学习，直接从偏好数据优化策略。**

DPO 的关键洞察是：在 RLHF 的目标函数下，最优策略有一个闭式解，可以将奖励函数表示为策略的函数。这样，我们可以将 Bradley-Terry 偏好模型中的奖励替换为策略的函数，从而得到一个直接作用于策略参数的损失函数。

**注记 1.2** (DPO 的优势)**.**

- **简单**：只需要两个模型（策略模型和参考模型），无需奖励模型和值函数
- **稳定**：本质上是监督学习，训练稳定，易于调参
- **高效**：可以直接在离线偏好数据上训练，无需在线采样
- **理论等价**：在一定条件下，DPO 的最优解与 RLHF 的最优解等价

# 2 从 RLHF 到 DPO 的数学推导

本节是 DPO 的核心，我们将从 RLHF 的目标函数出发，一步步推导出 DPO 的损失函数。

## 2.1 RLHF 目标函数的回顾

回顾 RLHF 中 PPO 优化的目标：在最大化奖励的同时，限制策略不要偏离参考策略太远。

**定义 2.1** (RLHF 优化目标)**.**

$$\max_{\pi_\theta} \mathbb{E}_{x\sim\mathcal{D}}\mathbb{E}_{y\sim\pi_\theta(\cdot|x)}\left[r(x,y)\right] - \beta\cdot\mathbb{E}_{x\sim\mathcal{D}}\left[\mathrm{KL}(\pi_\theta(\cdot|x)\|\pi_{\text{ref}}(\cdot|x))\right] \tag{1}$$

其中：

- $r(x,y)$：奖励函数（由奖励模型给出）
- $\pi_{\text{ref}}$：参考策略（通常是 SFT 模型）
- $\beta > 0$：KL 惩罚系数，控制策略偏离程度
- $\mathcal{D}$：prompt 的分布

将目标函数展开，对于固定的 $x$：

$$\begin{aligned} J(\pi_\theta; x) &= \mathbb{E}_{y\sim\pi_\theta(\cdot|x)}[r(x,y)] - \beta\cdot\mathrm{KL}(\pi_\theta(\cdot|x)\|\pi_{\text{ref}}(\cdot|x)) \\ &= \sum_y \pi_\theta(y|x) r(x,y) - \beta\sum_y \pi_\theta(y|x)\log\frac{\pi_\theta(y|x)}{\pi_{\text{ref}}(y|x)} \end{aligned} \tag{2}$$

## 2.2 最优策略的闭式解

接下来我们求解最优策略 $\pi^*$。这是 DPO 推导的关键步骤。

**定理 2.1** (最优策略的闭式解)**.** 对于优化目标 eq. (1)，最优策略为：

$$\boxed{\pi^*(y|x) = \frac{1}{Z(x)}\pi_{\text{ref}}(y|x)\exp\left(\frac{1}{\beta}r(x,y)\right)} \tag{3}$$

其中配分函数（归一化常数）为：

$$Z(x) = \sum_y \pi_{\text{ref}}(y|x)\exp\left(\frac{1}{\beta}r(x,y)\right) \tag{4}$$

详细证明

*Proof.* 我们使用变分法求解这个约束优化问题。

**步骤 1：写出拉格朗日函数**

策略 $\pi_\theta(\cdot|x)$ 是一个概率分布，需要满足归一化约束 $\sum_y \pi_\theta(y|x) = 1$。

拉格朗日函数为：

$$\mathcal{L}(\pi_\theta, \lambda) = \sum_y \pi_\theta(y|x) r(x,y) - \beta \sum_y \pi_\theta(y|x) \log \frac{\pi_\theta(y|x)}{\pi_{\text{ref}}(y|x)} + \lambda \left(1 - \sum_y \pi_\theta(y|x)\right)$$

**步骤 2：对 $\pi_\theta(y|x)$ 求导**

对于每个 $y$，对 $\pi_\theta(y|x)$ 求偏导：

$$\frac{\partial \mathcal{L}}{\partial \pi_\theta(y|x)} = r(x,y) - \beta \left(\log \frac{\pi_\theta(y|x)}{\pi_{\text{ref}}(y|x)} + 1\right) - \lambda$$

令导数为零：

$$r(x,y) - \beta \log \frac{\pi_\theta(y|x)}{\pi_{\text{ref}}(y|x)} - \beta - \lambda = 0$$

**步骤 3：解出 $\pi_\theta(y|x)$**

整理得：

$$\begin{aligned}
\log \frac{\pi_\theta(y|x)}{\pi_{\text{ref}}(y|x)} &= \frac{1}{\beta}(r(x,y) - \beta - \lambda) \\
\frac{\pi_\theta(y|x)}{\pi_{\text{ref}}(y|x)} &= \exp\left(\frac{r(x,y) - \beta - \lambda}{\beta}\right) \\
\pi_\theta(y|x) &= \pi_{\text{ref}}(y|x) \exp\left(\frac{r(x,y)}{\beta}\right) \exp\left(-1 - \frac{\lambda}{\beta}\right)
\end{aligned}$$

**步骤 4：利用归一化条件确定常数**

由 $\sum_y \pi_\theta(y|x) = 1$：

$$\begin{aligned}
\sum_y \pi_{\text{ref}}(y|x) \exp\left(\frac{r(x,y)}{\beta}\right) \exp\left(-1 - \frac{\lambda}{\beta}\right) &= 1 \\
\exp\left(-1 - \frac{\lambda}{\beta}\right) &= \frac{1}{\sum_y \pi_{\text{ref}}(y|x) \exp\left(\frac{r(x,y)}{\beta}\right)}
\end{aligned}$$

定义配分函数：

$$Z(x) = \sum_y \pi_{\text{ref}}(y|x) \exp\left(\frac{r(x,y)}{\beta}\right)$$

则：

$$\pi^*(y|x) = \frac{1}{Z(x)} \pi_{\text{ref}}(y|x) \exp\left(\frac{r(x,y)}{\beta}\right)$$

□

**注记 2.1** (最优策略的直观理解)**.**

- 最优策略是参考策略的" 能量调整" 版本
- 奖励高的输出 $y$ 概率被放大（乘以 $\exp(r/\beta)$）
- 奖励低的输出 $y$ 概率被缩小
- $\beta$ 控制调整的幅度：$\beta$ 小时调整激进，$\beta$ 大时调整保守
- 当 $\beta \to \infty$ 时，$\pi^* \to \pi_{\text{ref}}$（完全保守）
- 当 $\beta \to 0$ 时，$\pi^*$ 趋向于只选择奖励最高的输出

### 2.3 从最优策略反解奖励函数

定理 2.1给出了*已知奖励，求最优策略*的关系。DPO 的关键步骤是**反过来**：从策略表示奖励。

**定理 2.2** (奖励函数的隐式表示)**.** 如果 $\pi^*$ 是优化目标 eq. (1)的最优策略，则奖励函数可以表示为：

$$\boxed{r(x,y) = \beta \log \frac{\pi^*(y|x)}{\pi_{\text{ref}}(y|x)} + \beta \log Z(x)} \tag{5}$$

*Proof.* 从最优策略的表达式 eq. (3)出发：

$$\pi^*(y|x) = \frac{1}{Z(x)} \pi_{\text{ref}}(y|x) \exp\left(\frac{r(x,y)}{\beta}\right)$$

两边取对数：

$$\log \pi^*(y|x) = -\log Z(x) + \log \pi_{\text{ref}}(y|x) + \frac{r(x,y)}{\beta}$$

解出 $r(x,y)$：

$$\begin{aligned} r(x,y) &= \beta \log \pi^*(y|x) - \beta \log \pi_{\text{ref}}(y|x) + \beta \log Z(x) \\ &= \beta \log \frac{\pi^*(y|x)}{\pi_{\text{ref}}(y|x)} + \beta \log Z(x) \end{aligned}$$

□

**注记 2.2** (隐式奖励的意义)**.** 公式 eq. (5)表明：

- 奖励可以用策略的对数概率比来表示
- $\beta \log \frac{\pi^*(y|x)}{\pi_{\text{ref}}(y|x)}$ 衡量最优策略相对于参考策略对输出 $y$ 的" 偏好程度"
- $\beta \log Z(x)$ 是一个只依赖于 $x$ 的常数项

关键洞察：在比较两个输出时，$\beta \log Z(x)$ 会被消去。

### 2.4 Bradley-Terry 模型回顾

在上一讲中，我们介绍了 Bradley-Terry 模型。这里简要回顾其核心内容。

**定义 2.2** (Bradley-Terry 模型)**.** 假设每个输出 $y$ 有一个潜在的质量分数 $r(x,y) \in \mathbb{R}$。当人类比较两个输出 $y_1$ 和 $y_2$ 时，偏好 $y_1$ 的概率为：

$$P(y_1 \succ y_2|x) = \frac{\exp(r(x,y_1))}{\exp(r(x,y_1)) + \exp(r(x,y_2))} = \sigma(r(x,y_1) - r(x,y_2)) \tag{6}$$

其中 $\sigma(z) = \frac{1}{1+e^{-z}}$ 是 sigmoid 函数。

该模型的直观含义是：

- 奖励差 $r(x, y_1) - r(x, y_2)$ 越大，$y_1$ 被偏好的概率越高
- 当 $r(x, y_1) = r(x, y_2)$ 时，两者被偏好的概率各为 50%
- Sigmoid 函数将奖励差映射到概率空间 $(0, 1)$

关于 Bradley-Terry 模型的详细推导（包括从随机效用模型的推导），请参见上一讲的第 10.2 节。

### 2.5 代入 Bradley-Terry 模型

现在将隐式奖励代入 Bradley-Terry 偏好模型。

**定理 2.3** (DPO 的偏好概率)**.** 将隐式奖励 eq. (5)代入 Bradley-Terry 模型，得到：

$$P(y_w \succ y_l|x) = \sigma\left(\beta \log \frac{\pi^*(y_w|x)}{\pi_{\text{ref}}(y_w|x)} - \beta \log \frac{\pi^*(y_l|x)}{\pi_{\text{ref}}(y_l|x)}\right) \tag{7}$$

*Proof.* 代入隐式奖励表达式：

$$\begin{aligned} r(x, y_w) - r(x, y_l) &= \left(\beta \log \frac{\pi^*(y_w|x)}{\pi_{\text{ref}}(y_w|x)} + \beta \log Z(x)\right) \\ &\quad - \left(\beta \log \frac{\pi^*(y_l|x)}{\pi_{\text{ref}}(y_l|x)} + \beta \log Z(x)\right) \\ &= \beta \log \frac{\pi^*(y_w|x)}{\pi_{\text{ref}}(y_w|x)} - \beta \log \frac{\pi^*(y_l|x)}{\pi_{\text{ref}}(y_l|x)} \end{aligned}$$

注意 $\beta \log Z(x)$ 项被消去了。
代入 Sigmoid：

$$P(y_w \succ y_l|x) = \sigma\left(\beta \log \frac{\pi^*(y_w|x)}{\pi_{\text{ref}}(y_w|x)} - \beta \log \frac{\pi^*(y_l|x)}{\pi_{\text{ref}}(y_l|x)}\right)$$

□

**注记 2.3** (配分函数的消去)**.** 配分函数 $Z(x)$ 的消去是 DPO 能够工作的关键。这意味着：

- 我们不需要计算难以处理的配分函数
- 偏好概率只依赖于策略在两个具体输出上的概率比
- 这使得直接优化策略成为可能

### 2.6 DPO 损失函数的推导

现在我们可以得到 DPO 的损失函数。

**定义 2.3** (DPO 损失函数)**.** 给定偏好数据集 $\mathcal{D} = \{(x^{(i)}, y_w^{(i)}, y_l^{(i)})\}_{i=1}^N$，DPO 的损失函数为负对数似然：

**期望形式：**

$$\boxed{\mathcal{L}_{\text{DPO}}(\theta) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\left[\log \sigma\left(\beta \log \frac{\pi_\theta(y_w|x)}{\pi_{\text{ref}}(y_w|x)} - \beta \log \frac{\pi_\theta(y_l|x)}{\pi_{\text{ref}}(y_l|x)}\right)\right]} \tag{8}$$

**样本估计形式：**

$$\hat{\mathcal{L}}_{\text{DPO}}(\theta) = -\frac{1}{N}\sum_{i=1}^{N}\log\sigma\left(\beta\log\frac{\pi_\theta(y_w^{(i)}|x^{(i)})}{\pi_{\text{ref}}(y_w^{(i)}|x^{(i)})} - \beta\log\frac{\pi_\theta(y_l^{(i)}|x^{(i)})}{\pi_{\text{ref}}(y_l^{(i)}|x^{(i)})}\right) \tag{9}$$

为了简化表示，定义隐式奖励：

**定义 2.4** (隐式奖励)**.**

$$\hat{r}_\theta(x,y) = \beta\log\frac{\pi_\theta(y|x)}{\pi_{\text{ref}}(y|x)} \tag{10}$$

则 DPO 损失可简写为：

$$\mathcal{L}_{\text{DPO}}(\theta) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\left[\log\sigma\left(\hat{r}_\theta(x,y_w) - \hat{r}_\theta(x,y_l)\right)\right] \tag{11}$$

**注记 2.4** (DPO 损失的结构)**.** DPO 损失与奖励模型损失结构完全相同：

$$\mathcal{L}_{\text{RM}}(\phi) = -\mathbb{E}\left[\log\sigma(r_\phi(x,y_w) - r_\phi(x,y_l))\right]$$
$$\mathcal{L}_{\text{DPO}}(\theta) = -\mathbb{E}\left[\log\sigma(\hat{r}_\theta(x,y_w) - \hat{r}_\theta(x,y_l))\right]$$

区别在于：

- 奖励模型：$r_\phi$ 是显式参数化的神经网络
- DPO：$\hat{r}_\theta = \beta\log(\pi_\theta/\pi_{\text{ref}})$ 是策略的隐式函数

# 3 DPO 损失函数的分析

## 3.1 损失函数的直观理解

**注记 3.1** (DPO 在做什么)**.** 直观地说，DPO 损失函数在做以下事情：

1. 计算策略相对于参考策略的" 偏好程度"：$\hat{r}_\theta(x,y) = \beta\log\frac{\pi_\theta(y|x)}{\pi_{\text{ref}}(y|x)}$
2. 要求 $y_w$ 的隐式奖励高于 $y_l$：$\hat{r}_\theta(x,y_w) > \hat{r}_\theta(x,y_l)$
3. 通过 Sigmoid 和 log 构造平滑的损失函数

最小化 DPO 损失会：

- **增加** $\pi_\theta(y_w|x)$ 相对于 $\pi_{\text{ref}}(y_w|x)$ 的比值
- **减少** $\pi_\theta(y_l|x)$ 相对于 $\pi_{\text{ref}}(y_l|x)$ 的比值

**例 3.1** (损失值的计算)**.** 假设 $\beta = 0.1$，对于某个样本：

- $\log\pi_\theta(y_w|x) = -10$，$\log\pi_{\text{ref}}(y_w|x) = -12$
- $\log\pi_\theta(y_l|x) = -15$，$\log\pi_{\text{ref}}(y_l|x) = -14$

计算隐式奖励：

$$\hat{r}_\theta(x,y_w) = 0.1\times(-10-(-12)) = 0.1\times 2 = 0.2$$
$$\hat{r}_\theta(x,y_l) = 0.1\times(-15-(-14)) = 0.1\times(-1) = -0.1$$

奖励差：$\Delta\hat{r} = 0.2-(-0.1) = 0.3$
损失：$-\log\sigma(0.3) = -\log(0.574)\approx 0.555$
这个损失值表示模型已经学会偏好 $y_w$，但还可以进一步优化。

### 3.2 梯度分析

理解 DPO 的梯度对于理解其训练动态至关重要。

**定理 3.1** (DPO 损失的梯度)**.** 对于单个样本 $(x, y_w, y_l)$，DPO 损失关于参数 $\theta$ 的梯度为：

$$\boxed{\nabla_\theta \mathcal{L}_{\text{DPO}} = -\beta \cdot \underbrace{(1 - \sigma(\Delta\hat{r}))}_{\text{权重}} \cdot \underbrace{(\nabla_\theta \log \pi_\theta(y_w|x) - \nabla_\theta \log \pi_\theta(y_l|x))}_{\text{方向}}} \tag{12}$$

其中 $\Delta\hat{r} = \hat{r}_\theta(x, y_w) - \hat{r}_\theta(x, y_l)$。

详细推导

*Proof.* 设 $\Delta\hat{r} = \hat{r}_\theta(x, y_w) - \hat{r}_\theta(x, y_l) = \beta\left(\log \frac{\pi_\theta(y_w|x)}{\pi_{\text{ref}}(y_w|x)} - \log \frac{\pi_\theta(y_l|x)}{\pi_{\text{ref}}(y_l|x)}\right)$
损失为 $\mathcal{L} = -\log \sigma(\Delta\hat{r})$。
**步骤 1：对 $\Delta\hat{r}$ 求导**
由链式法则：

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \Delta\hat{r}} &= -\frac{1}{\sigma(\Delta\hat{r})} \cdot \sigma(\Delta\hat{r})(1 - \sigma(\Delta\hat{r})) \\ &= -(1 - \sigma(\Delta\hat{r}))\end{aligned}$$

**步骤 2：计算 $\nabla_\theta \Delta\hat{r}$**

$$\Delta\hat{r} = \beta \log \pi_\theta(y_w|x) - \beta \log \pi_{\text{ref}}(y_w|x) - \beta \log \pi_\theta(y_l|x) + \beta \log \pi_{\text{ref}}(y_l|x)$$

由于 $\pi_{\text{ref}}$ 不依赖于 $\theta$：

$$\begin{aligned}\nabla_\theta \Delta\hat{r} &= \beta \nabla_\theta \log \pi_\theta(y_w|x) - \beta \nabla_\theta \log \pi_\theta(y_l|x) \\ &= \beta\left(\nabla_\theta \log \pi_\theta(y_w|x) - \nabla_\theta \log \pi_\theta(y_l|x)\right)\end{aligned}$$

**步骤 3：组合**

$$\begin{aligned}\nabla_\theta \mathcal{L} &= \frac{\partial \mathcal{L}}{\partial \Delta\hat{r}} \cdot \nabla_\theta \Delta\hat{r} \\ &= -(1 - \sigma(\Delta\hat{r})) \cdot \beta\left(\nabla_\theta \log \pi_\theta(y_w|x) - \nabla_\theta \log \pi_\theta(y_l|x)\right)\end{aligned}$$

□

**注记 3.2** (梯度的直观理解)**.** 梯度公式 eq. (12)有两个重要组成部分：

**1. 方向**：$\nabla_\theta \log \pi_\theta(y_w|x) - \nabla_\theta \log \pi_\theta(y_l|x)$

- 梯度下降会增加 $\log \pi_\theta(y_w|x)$（提高好输出的概率）
- 同时减少 $\log \pi_\theta(y_l|x)$（降低差输出的概率）

**2. 权重**：$(1 - \sigma(\Delta\hat{r}))$

- 当模型已经正确排序（$\Delta\hat{r}$ 大）时，权重接近 0，梯度小
- 当模型排序错误（$\Delta\hat{r}$ 小或为负）时，权重大，梯度大
- 这是一种**自适应机制**：模型在困难样本上学习更多

**注记 3.3** (与监督学习的对比)**.** 标准的监督微调（SFT）梯度为：

$$\nabla_\theta \mathcal{L}_{\text{SFT}} = -\nabla_\theta \log \pi_\theta(y|x)$$

DPO 梯度的特点：

1. **对比学习**：同时考虑正样本 $y_w$ 和负样本 $y_l$，而非只看正样本
2. **自适应权重**：$(1 - \sigma(\Delta\hat{r}))$ 根据当前预测调整学习强度
3. **相对优化**：优化的是 $y_w$ 相对于 $y_l$ 的概率比，而非绝对概率

## 3.3 $\beta$ 参数的作用

**注记 3.4** ($\beta$ 的多重角色)**.** 参数 $\beta$ 在 DPO 中扮演多个角色：

**1. 控制隐式 KL 约束的强度**

- 大 $\beta$：对偏离参考策略的惩罚大，策略变化保守
- 小 $\beta$：惩罚小，策略可以大幅偏离参考策略

**2. 缩放奖励差异**

- 隐式奖励 $\hat{r}_\theta = \beta \log(\pi_\theta / \pi_{\text{ref}})$
- $\beta$ 决定了 log 概率比转换为奖励的比例

**3. 影响优化难度**

- 大 $\beta$：Sigmoid 输入范围大，梯度可能较小
- 小 $\beta$：Sigmoid 输入范围小，需要更精细的区分

典型取值：$\beta \in [0.1, 0.5]$

## 3.4 对比学习角度理解 DPO

DPO 的损失函数与对比学习（Contrastive Learning）有着深刻的联系。本节从对比学习的视角重新理解 DPO。

### 3.4.1 DPO 损失的另一种形式

将正样本（被偏好）记为 $y^+$，负样本（不被偏好）记为 $y^-$，DPO 损失可以写成：

$$\begin{aligned}\mathcal{L}_{\text{DPO}} &= -\mathbb{E}_{(y^+,y^-,x)\sim\mathcal{D}}\left[\log P[y^+ \succ y^-]\right] \\ &= -\mathbb{E}_{(y^+,y^-,x)\sim\mathcal{D}}\left[\log \frac{\exp\left(\beta \log \frac{\pi(y^+|x)}{\pi_{\text{ref}}(y^+|x)}\right)}{\exp\left(\beta \log \frac{\pi(y^+|x)}{\pi_{\text{ref}}(y^+|x)}\right) + \exp\left(\beta \log \frac{\pi(y^-|x)}{\pi_{\text{ref}}(y^-|x)}\right)}\right]\end{aligned} \tag{13}$$

### 3.4.2 对比学习回顾

**定义 3.1** (对比学习损失)**.** 对比学习的目标是学习一种表示，使得**相似的样本彼此接近，而不相似的样本彼此远离**。

对于一个锚样本 x（anchor），假设有：

- 一个正样本 $\text{x}^+$（例如，同一图像的数据增强版本）
- 一组负样本 $\{\text{x}_i^-\}_{i=1}^m$（例如，其他图像）

为了学习一个编码器 $f$，对比学习最小化以下损失（InfoNCE loss）：

$$\ell_f(\mathrm{x}, \mathrm{x}^+, \{\mathrm{x}_i^-\}_{i=1}^m) = -\log \frac{\exp(f(\mathrm{x})^\top f(\mathrm{x}^+))}{\exp(f(\mathrm{x})^\top f(\mathrm{x}^+)) + \sum_{i=1}^m \exp(f(\mathrm{x})^\top f(\mathrm{x}_i^-))} \tag{14}$$

其中 $f(\mathrm{x})^\top f(\mathrm{x}^+)$ 是两个表示的点积，可视为相似性分数。

**注记 3.5** (对比学习的直观理解)**.** InfoNCE 损失的效果是：

- **拉近**锚样本与正样本：增大 $f(\mathrm{x})^\top f(\mathrm{x}^+)$
- **推远**锚样本与负样本：减小 $f(\mathrm{x})^\top f(\mathrm{x}_i^-)$

分母中的求和起到归一化作用，使得正样本的相似度在所有样本中占比更高。

### 3.4.3 DPO 作为对比学习

**命题 3.2** (DPO 是一种对比学习)**.** DPO 可以看成是**正负样本数量都为 1** 的对比学习，其中：

- **锚样本**：策略 $\pi(\cdot|x)$
- **正样本**：被偏好的输出 $y^+$
- **负样本**：不被偏好的输出 $y^-$
- **相似性分数**：$\beta \log \frac{\pi(y|x)}{\pi_{\text{ref}}(y|x)}$（隐式奖励）

隐式奖励 $\beta \log \frac{\pi(y|x)}{\pi_{\text{ref}}(y|x)}$ 衡量的是策略对输出 $y$ 的" 偏好程度"——值越大说明策略相对于参考策略更倾向于生成 $y$。

*Proof.* 对比 DPO 损失 eq. (13)和 InfoNCE 损失 eq. (14)：
**InfoNCE**（$m = 1$ 个负样本）：

$$\ell = -\log \frac{\exp(s^+)}{\exp(s^+) + \exp(s^-)} \tag{15}$$

其中 $s^+ = f(\mathrm{x})^\top f(\mathrm{x}^+)$，$s^- = f(\mathrm{x})^\top f(\mathrm{x}^-)$。
**DPO**：

$$\mathcal{L}_{\text{DPO}} = -\log \frac{\exp(\hat{r}^+)}{\exp(\hat{r}^+) + \exp(\hat{r}^-)} \tag{16}$$

其中 $\hat{r}^+ = \beta \log \frac{\pi(y^+|x)}{\pi_{\text{ref}}(y^+|x)}$，$\hat{r}^- = \beta \log \frac{\pi(y^-|x)}{\pi_{\text{ref}}(y^-|x)}$。
两者结构完全相同，只是相似性分数的定义不同。 □

**注记 3.6** (对比学习视角的启示)**.** 从对比学习的角度理解 DPO：

1. **目标**：让策略的" 表示"（通过隐式奖励衡量）更接近正样本、远离负样本
2. **效果**：最小化 DPO 损失会：
   - 增大 $\pi(y^+|x)$ 相对于 $\pi_{\text{ref}}(y^+|x)$ 的比值（拉近正样本）
   - 减小 $\pi(y^-|x)$ 相对于 $\pi_{\text{ref}}(y^-|x)$ 的比值（推远负样本）
3. **对比对象**：与传统对比学习不同，DPO 对比的不是样本的向量表示，而是策略对不同输出的概率比

### 3.4.4 与多负样本对比学习的联系

**注记 3.7** (扩展到多个负样本)**.**

标准 DPO 使用一个正样本和一个负样本。自然的扩展是使用多个负样本：

$$\mathcal{L}_{\text{DPO-multi}} = -\log \frac{\exp(\hat{r}(x, y^+))}{\exp(\hat{r}(x, y^+)) + \sum_{i=1}^{m} \exp(\hat{r}(x, y_i^-))} \tag{17}$$

这类似于 InfoNCE 使用更多负样本来提供更强的对比信号。实践中，可以通过从策略采样多个输出并标注偏好来构造多负样本数据。

**注记 3.8** (对比学习的温度参数)**.**

在对比学习中，通常有一个温度参数 $\tau$ 控制分布的锐度：

$$\ell = -\log \frac{\exp(s^+/\tau)}{\exp(s^+/\tau) + \exp(s^-/\tau)} \tag{18}$$

在 DPO 中，$\beta$ 扮演类似的角色：

- 小 $\beta$：分布更锐（sharp），对偏好差异更敏感
- 大 $\beta$：分布更平（soft），对偏好差异更容忍

这与对比学习中温度参数的作用一致。

| **概念** | **对比学习** | **DPO** |
|---|---|---|
| 锚样本 | 输入 $\mathrm{x}$ | 策略 $\pi(\cdot\|x)$ |
| 正样本 | 增强样本 $\mathrm{x}^+$ | 偏好输出 $y^+$ |
| 负样本 | 其他样本 $\mathrm{x}^-$ | 非偏好输出 $y^-$ |
| 相似性度量 | 表示点积 $f(\mathrm{x})^\top f(\mathrm{x}')$ | 隐式奖励 $\beta \log \frac{\pi(y\|x)}{\pi_{\text{ref}}(y\|x)}$ |
| 温度参数 | $\tau$ | $1/\beta$（或 $\beta$ 本身） |
| 优化目标 | 学习好的表示 | 学习好的策略 |

表格 1: 对比学习与 DPO 的对应关系

# 4 DPO 与 RLHF 的理论联系

## 4.1 等价性条件

DPO 的推导基于一个关键假设：我们在用最优策略 $\pi^*$ 的形式来参数化当前策略 $\pi_\theta$。

**定理 4.1** (DPO 与 RLHF 的等价性)**.** 如果以下条件满足：

1. 偏好数据由 Bradley-Terry 模型生成
2. 奖励模型完美拟合真实奖励
3. 策略优化达到全局最优

则 DPO 的最优解与 RLHF 的最优解相同。

**注记 4.1** (实践中的差异)**.** 在实践中，这些理想条件不完全满足：

- 人类偏好可能不完全符合 Bradley-Terry 模型
- 神经网络的表达能力有限，不能完美拟合
- 优化可能陷入局部最优

因此，DPO 和 RLHF 在实践中可能给出不同的结果。实验表明，在许多任务上 DPO 的效果与 RLHF 相当甚至更好。

### 4.2 DPO 的隐式奖励模型

虽然 DPO 不显式训练奖励模型，但训练好的 DPO 策略隐含了一个奖励函数。

**定义 4.1** (从 DPO 提取奖励)**.** 给定训练好的 DPO 策略 $\pi_\theta$，隐式奖励为：

$$\hat{r}(x, y) = \beta \log \frac{\pi_\theta(y|x)}{\pi_{\text{ref}}(y|x)} \tag{19}$$

这个奖励可以用于：

- 评估和比较不同输出的质量
- 作为其他任务的奖励信号
- 分析模型学到了什么偏好

**注记 4.2** (隐式奖励的特点)**.**

- **相对性**：隐式奖励是相对于参考策略定义的
- **无配分函数**：不需要计算归一化常数
- **与策略绑定**：奖励直接由策略参数决定

### 4.3 KL 散度的隐式约束

**命题 4.2** (DPO 隐式包含 KL 约束)**.** DPO 的损失函数隐式地约束策略不要偏离参考策略太远。具体地，隐式奖励可以改写为：

$$\hat{r}_\theta(x, y) = \beta \log \pi_\theta(y|x) - \beta \log \pi_{\text{ref}}(y|x) \tag{20}$$

最大化隐式奖励等价于最大化 $\log \pi_\theta(y|x)$，同时受到 $-\beta \log \pi_{\text{ref}}(y|x)$ 的锚定。

**注记 4.3** (参考策略的作用)**.** 参考策略 $\pi_{\text{ref}}$ 在 DPO 中起到关键作用：

1. **提供基准**：隐式奖励衡量的是相对于 $\pi_{\text{ref}}$ 的改进
2. **防止崩溃**：没有参考策略，模型可能学到退化的解
3. **保持能力**：$\pi_{\text{ref}}$ 通常是 SFT 模型，保证基本的语言能力

实践建议：$\pi_{\text{ref}}$ 应该是质量合理的模型（如 SFT 后的模型），不应该是随机初始化的模型。

## 5 DPO 的适用范围与多步 MDP 扩展

前面我们推导了 DPO 的损失函数，但这个推导有一个重要的前提假设：**单步 MDP**。本节将分析 DPO 在多步 MDP 下的问题，并介绍基于优势函数的替代方案。

### 5.1 DPO 推导的关键假设

回顾 DPO 推导的关键步骤：我们将隐式奖励代入 Bradley-Terry 模型后，配分函数 $Z(x)$ 被消去了。

$$r(x, y_w) - r(x, y_l) = \beta \log \frac{\pi^*(y_w|x)}{\pi_{\text{ref}}(y_w|x)} - \beta \log \frac{\pi^*(y_l|x)}{\pi_{\text{ref}}(y_l|x)} + \underbrace{\beta \log Z(x) - \beta \log Z(x)}_{=0} \tag{21}$$

这个消去能够成立，是因为在大语言模型场景下：

- 两个回答 $y_w$ 和 $y_l$ 对应**同一个**输入 $x$
- 因此它们共享**同一个**配分函数 $Z(x)$
- 相减后配分函数消去

**注记 5.1** (DPO 的单步 MDP 假设)**.**
DPO 的推导基于大语言模型的特殊结构，本质上是一个**单步 MDP**（或称为 Contextual Bandit）：

- **状态**：输入 prompt $x$
- **动作**：完整的输出序列 $y$（视为一个整体动作）
- **奖励**：序列级别的奖励 $r(x, y)$

在这个视角下，生成过程不是逐 token 的多步决策，而是一次性选择整个输出序列。

## 5.2 多步 MDP 下 DPO 的问题

现在考虑传统强化学习场景下的多步 MDP，看看 DPO 的推导会出现什么问题。

### 5.2.1 多步 MDP 的偏好建模

在多步 MDP 中，我们比较的是两条**轨迹**（trajectory），而非两个单步动作。

**定义 5.1** (轨迹偏好)**.** 设两条轨迹为：

$$\sigma^1 = \left((s_0^1, a_0^1), (s_1^1, a_1^1), \ldots, (s_{T_1-1}^1, a_{T_1-1}^1)\right) \tag{22}$$

$$\sigma^2 = \left((s_0^2, a_0^2), (s_1^2, a_1^2), \ldots, (s_{T_2-1}^2, a_{T_2-1}^2)\right) \tag{23}$$

人类偏好标签 $y \in \{0, 1, 0.5\}$ 表示：

- $y = 1$：偏好 $\sigma^1$（即 $\sigma^1 \succ \sigma^2$）
- $y = 0$：偏好 $\sigma^2$（即 $\sigma^2 \succ \sigma^1$）
- $y = 0.5$：同等偏好

**定义 5.2** (基于累积奖励的 Bradley-Terry 模型)**.** 在多步 MDP 中，假设人类偏好一条轨迹的概率与该轨迹的**累积折扣奖励**的指数成正比。基于 Bradley-Terry 模型：

$$P^*[\sigma^1 \succ \sigma^2] = \frac{\exp\left(\sum_{\sigma^1} \gamma^t r^*(s_t^1, a_t^1)\right)}{\exp\left(\sum_{\sigma^1} \gamma^t r^*(s_t^1, a_t^1)\right) + \exp\left(\sum_{\sigma^2} \gamma^t r^*(s_t^2, a_t^2)\right)} \tag{24}$$

其中 $r^*(s, a)$ 是最优奖励函数，$\gamma$ 是折扣因子。

**注记 5.2** (与大语言模型 RLHF 的区别)**.** 多步 MDP 与大语言模型 RLHF 的关键区别：

1. **多步 vs 单步**：多步 MDP 需要在每个时间步做决策，而 LLM 可以视为一次性生成整个序列
2. **不同起始状态**：多步 MDP 中两条轨迹的起始状态 $s_0^1$ 和 $s_0^2$ 可能不同，而 LLM 中两个回答共享同一个 prompt $x$
3. **轨迹长度可变**：两条轨迹的长度 $T_1$ 和 $T_2$ 可能不同

### 5.2.2 配分函数无法消去

现在尝试在多步 MDP 下应用 DPO 的推导思路。

多步 MDP 的优化目标（带 KL 约束）为：

$$\max_{\pi} \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty}\gamma^t\left(r(s_t,a_t)-\beta\log\frac{\pi(a_t|s_t)}{\pi_{\text{ref}}(a_t|s_t)}\right)\right] \tag{25}$$

通过类似的推导（利用拉格朗日乘子法和 KKT 条件），可以得到最优策略和最优奖励函数的关系：

$$r^*(s_t,a_t)=\beta\log\frac{\pi^*(a_t|s_t)}{\pi_{\text{ref}}(a_t|s_t)}+\beta\log Z(s_t) \tag{26}$$

其中 $Z(s_t)$ 是依赖于状态 $s_t$ 的配分函数。

**定理 5.1** (多步 MDP 下配分函数无法消去)**.** 将隐式奖励 (26)代入多步 Bradley-Terry 模型 (24)：

$$P^*[\sigma^1\succ\sigma^2]=\frac{\exp\left(\sum_{\sigma^1}\gamma^t\log\frac{\pi^*(a_t^1|s_t^1)}{\pi_{\text{ref}}(a_t^1|s_t^1)}+\sum_{\sigma^1}\gamma^t\log Z(s_t^1)\right)}{\exp\left(\sum_{\sigma^1}\cdots\right)+\exp\left(\sum_{\sigma^2}\gamma^t\log\frac{\pi^*(a_t^2|s_t^2)}{\pi_{\text{ref}}(a_t^2|s_t^2)}+\sum_{\sigma^2}\gamma^t\log Z(s_t^2)\right)} \tag{27}$$

由于两条轨迹经过的状态序列不同（$s_t^1\neq s_t^2$），配分函数项：

$$\sum_{\sigma^1}\gamma^t\log Z(s_t^1)\neq\sum_{\sigma^2}\gamma^t\log Z(s_t^2) \tag{28}$$

**无法消去**。因此，不能像 DPO 那样得到一个只依赖于策略的损失函数。

**注记 5.3** (为什么单步 MDP 可以消去？)**.**

回顾单步 MDP（LLM 场景）：

- 两个输出 $y_w$ 和 $y_l$ 共享同一个输入 $x$
- 配分函数只依赖于 $x$：$Z(x)$
- 相减后：$\beta\log Z(x)-\beta\log Z(x)=0$

而在多步 MDP 中：

- 两条轨迹经过不同的状态序列
- 每个状态有不同的配分函数 $Z(s_t)$
- 无法消去

## 5.3 基于优势函数的偏好建模

既然传统强化学习场景下 DPO 的推导方式不适用，研究者们提出了另一种思路：**用优势函数代替奖励函数来建模人类偏好**。

### 5.3.1 动机：优势函数作为偏好度量

**注记 5.4** (为什么考虑优势函数？)**.** 原本的 Bradley-Terry 模型使用奖励函数 $r(s,a)$ 来衡量人类偏好，但从奖励函数并不能直接得到最优策略。那么是否可以用一些与策略直接相关的量来衡量人类偏好?

强化学习中与策略挂钩的值有：

- 动作价值函数 $Q^\pi(s,a)$：在状态 $s$ 采取动作 $a$，然后按策略 $\pi$ 行动的期望回报

- 优势函数 $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$：动作 $a$ 相对于平均水平的优势

最优策略可以通过这些值导出：

$$\pi^*(a|s) = \arg\max_\pi Q^\pi(s,a) = \arg\max_\pi A^\pi(s,a) \tag{29}$$

**优势函数 $A(s,a)$ 是一个很好的衡量人类偏好的度量**：它直接反映了这个动作比平均水平好多少。

**例 5.1** (优势函数 vs 奖励函数作为偏好度量)**.** 考虑一个寻路环境：智能体每走一步得到 $-1$ 的奖励，只有到达目标点才会得到 $+100$ 的奖励。

比较两条轨迹片段：

- 轨迹 S：智能体在向目标点靠近
- 轨迹 O：智能体在远离目标点

**用奖励函数衡量**：

$$\sum_S r_t^S = -1 - 1 = -2 = \sum_O r_t^O \tag{30}$$

两条轨迹的累积奖励相同，偏好概率是一致的（50%-50%）。

**用优势函数衡量**：

- 轨迹 S 的累积优势 $\sum_S A(s_t, a_t)$ 更大（因为动作在朝正确方向走）
- 轨迹 O 的累积优势更小或为负（因为动作在朝错误方向走）

显然，用优势函数能更好地区分这两条轨迹的质量。

### 5.3.2 基于优势函数的 Bradley-Terry 模型

**定义 5.3** (基于优势函数的偏好模型)**.** 使用最优优势函数 $A^*(s,a)$ 构建 Bradley-Terry 偏好模型：

$$P^*[\sigma^1 \succ \sigma^2] = \frac{\exp\left(\sum_{\sigma^1} \gamma^t A^*(s_t^1, a_t^1)\right)}{\exp\left(\sum_{\sigma^1} \gamma^t A^*(s_t^1, a_t^1)\right) + \exp\left(\sum_{\sigma^2} \gamma^t A^*(s_t^2, a_t^2)\right)} \tag{31}$$

这里用累积折扣优势代替累积折扣奖励。

**注记 5.5** (优势函数的优点)**.**

- 优势函数直接反映动作的相对好坏
- 不需要配分函数（优势函数本身就是相对于基线的差值）
- 可以直接与策略建立联系

### 5.3.3 从优势函数到策略

关键问题是：能否像 DPO 那样，将优势函数表示为策略的函数，从而直接学习策略?

**定理 5.2** (优势函数与最优策略的关系)**.** 在 Behavior-Regularised RL（通常是 offline RL 的范畴）的设定下，最优优势函数和最优策略有直接的等式关系。

具体地，对于带 KL 约束的优化目标：

$$\max_\pi \ \mathbb{E}_\pi \left[ \sum_{t=0}^{\infty} \gamma^t \left( r(s_t, a_t) - \beta \log \frac{\pi(a_t|s_t)}{\pi_{\text{ref}}(a_t|s_t)} \right) \right] \tag{32}$$

利用拉格朗日乘子法和 KKT 条件求解，可以得到：

$$\pi^*(a|s) = \pi_{\text{ref}}(a|s) \exp\left(\frac{A^*(s,a)}{\beta}\right) \tag{33}$$

$$A^*(s,a) = \beta \log \frac{\pi^*(a|s)}{\pi_{\text{ref}}(a|s)} \tag{34}$$

证明思路

*Proof.* 这个结论可以通过软 Q-learning（Soft Q-Learning）或最大熵强化学习（Maximum Entropy RL）的框架推导。
**步骤 1**：定义软值函数和软 Q 函数。
在带熵正则化的 MDP 中，定义软值函数：

$$V^{\text{soft}}(s) = \mathbb{E}_\pi \left[ \sum_{t=0}^{\infty} \gamma^t \left( r(s_t, a_t) + \beta \mathcal{H}(\pi(\cdot|s_t)) \right) \Big| s_0 = s \right] \tag{35}$$

软 Q 函数：

$$Q^{\text{soft}}(s,a) = r(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)}[V^{\text{soft}}(s')] \tag{36}$$

**步骤 2**：最优策略的形式。
在软 Q-learning 中，最优策略为：

$$\pi^*(a|s) = \frac{\exp(Q^*(s,a)/\beta)}{\sum_{a'} \exp(Q^*(s,a')/\beta)} = \frac{1}{Z(s)} \exp\left(\frac{Q^*(s,a)}{\beta}\right) \tag{37}$$

其中 $Z(s) = \sum_{a'} \exp(Q^*(s,a')/\beta)$。
**步骤 3**：引入参考策略。
当有参考策略 $\pi_{\text{ref}}$ 时，KL 正则化项变为：

$$-\beta \,\text{KL}(\pi \| \pi_{\text{ref}}) = \beta \mathbb{E}_\pi[\log \pi_{\text{ref}}(a|s)] - \beta \mathbb{E}_\pi[\log \pi(a|s)] \tag{38}$$

相应的最优策略变为：

$$\pi^*(a|s) \propto \pi_{\text{ref}}(a|s) \exp\left(\frac{Q^*(s,a)}{\beta}\right) \tag{39}$$

**步骤 4**：优势函数的关系。
定义优势函数 $A^*(s,a) = Q^*(s,a) - V^*(s)$。由于 $V^*(s)$ 不依赖于 $a$，在归一化后：

$$\pi^*(a|s) = \pi_{\text{ref}}(a|s) \exp\left(\frac{A^*(s,a)}{\beta}\right) \cdot \frac{1}{Z'(s)} \tag{40}$$

取对数并整理，得到：

$$A^*(s,a) = \beta \log \frac{\pi^*(a|s)}{\pi_{\text{ref}}(a|s)} + \beta \log Z'(s) \tag{41}$$

关键观察：对于优势函数，由于 $\mathbb{E}_{a \sim \pi^*}[A^*(s,a)] = 0$（优势函数的定义性质），配分函数项 $\beta \log Z'(s)$ 被吸收到基线中。
在适当的归一化下，我们得到：

$$A^*(s,a) = \beta \log \frac{\pi^*(a|s)}{\pi_{\text{ref}}(a|s)} \tag{42}$$

□

**注记 5.6** (关键区别)**.** 比较奖励函数和优势函数的隐式表示：

$$r^*(s,a) = \beta \log \frac{\pi^*(a|s)}{\pi_{\text{ref}}(a|s)} + \beta \log Z(s) \quad \text{（含配分函数）} \tag{43}$$

$$A^*(s,a) = \beta \log \frac{\pi^*(a|s)}{\pi_{\text{ref}}(a|s)} \quad \text{（无配分函数）} \tag{44}$$

优势函数的表示**不含配分函数**，这是因为优势函数本身已经是相对于状态值 $V(s)$ 的差值，配分函数被自然地消去了。

### 5.3.4 代入偏好模型得到损失函数

将隐式优势函数 eq. (34)代入基于优势函数的 Bradley-Terry 模型 eq. (31)：

$$\begin{aligned} P^*[\sigma^1 \succ \sigma^2] &= \frac{\exp\left(\sum_{\sigma^1} \gamma^t A^*(s_t^1, a_t^1)\right)}{\exp\left(\sum_{\sigma^1} \gamma^t A^*(s_t^1, a_t^1)\right) + \exp\left(\sum_{\sigma^2} \gamma^t A^*(s_t^2, a_t^2)\right)} \\ &= \frac{\exp\left(\sum_{\sigma^1} \gamma^t \beta \log \frac{\pi^*(a_t^1|s_t^1)}{\pi_{\text{ref}}(a_t^1|s_t^1)}\right)}{\exp\left(\sum_{\sigma^1} \gamma^t \beta \log \frac{\pi^*(a_t^1|s_t^1)}{\pi_{\text{ref}}(a_t^1|s_t^1)}\right) + \exp\left(\sum_{\sigma^2} \gamma^t \beta \log \frac{\pi^*(a_t^2|s_t^2)}{\pi_{\text{ref}}(a_t^2|s_t^2)}\right)} \end{aligned} \tag{45}$$

**定理 5.3** (基于优势函数的 DPO 损失（多步 MDP 版本）)**.** 对于多步 MDP，使用优势函数建模偏好，可以得到类似 DPO 的损失函数：

$$\boxed{\mathcal{L}(\theta) = -\mathbb{E}_{(\sigma^1,\sigma^2,y)\sim\mathcal{D}}\left[y \log P_\theta[\sigma^1 \succ \sigma^2] + (1-y)\log P_\theta[\sigma^2 \succ \sigma^1]\right]} \tag{46}$$

其中：

$$P_\theta[\sigma^1 \succ \sigma^2] = \sigma\left(\sum_{\sigma^1} \gamma^t \beta \log \frac{\pi_\theta(a_t^1|s_t^1)}{\pi_{\text{ref}}(a_t^1|s_t^1)} - \sum_{\sigma^2} \gamma^t \beta \log \frac{\pi_\theta(a_t^2|s_t^2)}{\pi_{\text{ref}}(a_t^2|s_t^2)}\right) \tag{47}$$

**注记 5.7** (回到大语言模型场景)**.** 在大语言模型的单步 MDP 场景下，上式简化为：

- 轨迹变成单个输出：$\sigma \to y$
- 状态变成 prompt：$s \to x$
- 无需折扣：$\gamma^t = 1$

得到：

$$P^*[y_1 \succ y_2|x] = \sigma\left(\beta \log \frac{\pi^*(y_1|x)}{\pi_{\text{ref}}(y_1|x)} - \beta \log \frac{\pi^*(y_2|x)}{\pi_{\text{ref}}(y_2|x)}\right) \tag{48}$$

这正是我们之前推导的 DPO 公式。

## 5.4 总结：DPO 的本质

**注记 5.8** (DPO 的本质理解)**.** 通过本节的分析，我们可以更深入地理解 DPO：
**DPO 的本质是将 RLHF 中 Bradley-Terry 模型的人类偏好度量从奖励函数换成了优势函数。**
具体地：

- **传统 RLHF**：用奖励函数 $r(x,y)$ 衡量偏好 → 需要先训练奖励模型，再用 RL 优化
- **DPO**：用优势函数 $A(x,y) = \beta \log \frac{\pi(y|x)}{\pi_{\text{ref}}(y|x)}$ 衡量偏好 → 直接优化策略

这个转换之所以有效，是因为：

1. 优势函数可以直接用策略表示（不含配分函数）
2. 在单步 MDP 下，优势函数的表示是精确的
3. 优势函数本身就是一个好的偏好度量（反映动作相对于基线的优劣）

**注记 5.9** (多步 MDP 的挑战)**.** 虽然基于优势函数的方法在理论上可以扩展到多步 MDP，但实践中仍有挑战：

- 需要处理变长轨迹
- 折扣因子 $\gamma$ 的影响
- 轨迹级别的偏好标注成本高

这也是为什么 DPO 主要在大语言模型（可以视为单步 MDP）场景下使用的原因。

# 6 DPO 的实现细节

## 6.1 数据准备

**注记 6.1** (偏好数据格式)**.** DPO 需要的数据格式为三元组 $(x, y_w, y_l)$：

- $x$：输入 prompt
- $y_w$：人类偏好的输出（chosen）
- $y_l$：人类不偏好的输出（rejected）

数据来源可以是：

- 人工标注的偏好数据
- 从现有模型采样，然后人工比较
- 使用更强的模型（如 GPT-4）进行自动标注

## 6.2 序列概率的计算

在语言模型中，输出 $y = (y_1, y_2, \ldots, y_T)$ 是一个 token 序列。

**定义 6.1** (序列对数概率)**.** 序列的对数概率是各 token 条件对数概率之和：

$$\log \pi_\theta(y|x) = \sum_{t=1}^{T} \log \pi_\theta(y_t|x, y_{<t}) \tag{49}$$

其中 $y_{<t} = (y_1, \ldots, y_{t-1})$ 是前 $t-1$ 个 token。

**注记 6.2** (实现注意事项)**.**

- 需要正确处理 attention mask，只计算 response 部分的概率
- prompt 部分的 token 不计入概率计算
- 对于 padding token，需要 mask 掉

### 6.3 数值稳定性

**注记 6.3** (避免数值问题)**.** DPO 计算中可能遇到的数值问题：

**1. Log 概率可能很小**

- 长序列的 $\log \pi(y|x)$ 可能是很大的负数（如 -500）
- 直接计算 exp 会下溢
- 解决：始终在 log 空间计算，避免转换为概率

**2. Sigmoid 的输入可能很大**

- 当 $|\Delta \hat{r}|$ 很大时，$\sigma$ 接近 0 或 1
- $\log \sigma$ 可能产生数值问题
- 解决：使用 `logsigmoid` 函数，它在数值上更稳定

**3. 概率比可能极端**

- $\log(\pi_\theta / \pi_{\text{ref}})$ 可能很大或很小
- 解决：可以对隐式奖励进行裁剪

### 6.4 参考模型的处理

**注记 6.4** (参考模型的实现)**.** 参考模型 $\pi_{\text{ref}}$ 在训练过程中保持冻结：

- 不计算梯度，使用 `torch.no_grad()`
- 可以与训练模型共享权重初始化
- 内存优化：可以使用半精度（fp16/bf16）

内存占用：DPO 需要同时加载两个模型（$\pi_\theta$ 和 $\pi_{\text{ref}}$），但仍比 RLHF 的 4 个模型少很多。

## 7 代码实现

### 7.1 计算序列对数概率

Listing 1: 计算序列对数概率

```python
import torch
import torch.nn.functional as F

def compute_log_probs(model, input_ids, attention_mask, labels):
    """
    计算给定序列的对数概率

    参数:
        model:          语言模型
        input_ids:      输入token ids, 形状 [batch_size, seq_len]
        attention_mask: 注意力掩码, 形状 [batch_size, seq_len]
        labels:         标签 (即input_ids本身, 用于计算loss的目标)
                        形状 [batch_size, seq_len], prompt部分设为-100

    返回:
        log_probs: 每个序列的总对数概率, 形状 [batch_size]
    """
```

```
    # 获取模型输出的logits
    outputs = model(input_ids=input_ids, attention_mask=attention_mask)
    logits = outputs.logits  # [batch_size, seq_len, vocab_size]

    # Shift: 预测下一个token
    # logits: 用前T-1个位置预测后T-1个token
    shift_logits = logits[:, :-1, :]  # [batch_size, seq_len-1, vocab_size]
    shift_labels = labels[:, 1:]       # [batch_size, seq_len-1]

    # 计算每个token的对数概率
    log_probs_all = F.log_softmax(shift_logits, dim=-1)  # [batch_size,
        seq_len-1, vocab_size]

    # 获取实际token的对数概率
    # 使用gather选取对应token的概率
    log_probs_token = torch.gather(
        log_probs_all,
        dim=-1,
        index=shift_labels.unsqueeze(-1)
    ).squeeze(-1)  # [batch_size, seq_len-1]

    # 创建mask: 只计算response部分 (labels != -100)
    loss_mask = (shift_labels != -100).float()  # [batch_size, seq_len-1]

    # 对每个序列求和得到总对数概率
    log_probs = (log_probs_token * loss_mask).sum(dim=-1)  # [batch_size]

    return log_probs
```

### 7.2 DPO 损失函数

Listing 2: DPO 损失函数

```
def compute_dpo_loss(policy_model, ref_model,
                     chosen_input_ids, chosen_attention_mask, chosen_labels
                         ,
                     rejected_input_ids, rejected_attention_mask,
                        rejected_labels,
                     beta=0.1):
    """
    计算DPO损失

    参数:
        policy_model:            待训练的策略模型
        ref_model:               参考模型 (冻结)
        chosen_input_ids:        偏好输出的input ids
        chosen_attention_mask:   偏好输出的attention mask
        chosen_labels:           偏好输出的labels (prompt部分为-100)
        rejected_input_ids:      非偏好输出的input ids
        rejected_attention_mask: 非偏好输出的attention mask
        rejected_labels:         非偏好输出的labels
        beta:                    温度参数

    返回:
        loss:     DPO损失
        metrics:  用于监控的指标
    """
```

```
    # 计算策略模型的对数概率
    policy_chosen_logps = compute_log_probs(
        policy_model, chosen_input_ids, chosen_attention_mask,
            chosen_labels
    )
    policy_rejected_logps = compute_log_probs(
        policy_model, rejected_input_ids, rejected_attention_mask,
            rejected_labels
    )

    # 计算参考模型的对数概率 (不计算梯度)
    with torch.no_grad():
        ref_chosen_logps = compute_log_probs(
            ref_model, chosen_input_ids, chosen_attention_mask,
                chosen_labels
        )
        ref_rejected_logps = compute_log_probs(
            ref_model, rejected_input_ids, rejected_attention_mask,
                rejected_labels
        )

    # 计算隐式奖励
    # r_hat(x, y) = beta * log(pi_theta(y|x) / pi_ref(y|x))
    #             = beta * (log pi_theta - log pi_ref)
    chosen_rewards = beta * (policy_chosen_logps - ref_chosen_logps)
    rejected_rewards = beta * (policy_rejected_logps - ref_rejected_logps)

    # 计算奖励差
    reward_diff = chosen_rewards - rejected_rewards  # [batch_size]

    # DPO损失: -log(sigmoid(reward_diff))
    # 使用logsigmoid更数值稳定
    loss = -F.logsigmoid(reward_diff).mean()

    # 计算监控指标
    with torch.no_grad():
        # 准确率: 模型是否正确偏好chosen
        accuracy = (reward_diff > 0).float().mean()

        # 平均奖励差
        mean_reward_diff = reward_diff.mean()

        # chosen和rejected的平均奖励
        mean_chosen_reward = chosen_rewards.mean()
        mean_rejected_reward = rejected_rewards.mean()

    metrics = {
        'loss': loss.item(),
        'accuracy': accuracy.item(),
        'reward_diff': mean_reward_diff.item(),
        'chosen_reward': mean_chosen_reward.item(),
        'rejected_reward': mean_rejected_reward.item(),
    }

    return loss, metrics
```

### 7.3 数据预处理

Listing 3: DPO 数据预处理

```python
def preprocess_dpo_data(tokenizer, prompt, chosen_response, rejected_response,
                        max_length=512):
    """
    预处理单个DPO样本

    参数:
        tokenizer:          分词器
        prompt:             输入prompt
        chosen_response:    偏好的回答
        rejected_response:  非偏好的回答
        max_length:         最大序列长度

    返回:
        处理后的数据字典
    """
    # 分别tokenize prompt和response
    prompt_tokens = tokenizer(prompt, add_special_tokens=False)
    chosen_tokens = tokenizer(chosen_response, add_special_tokens=False)
    rejected_tokens = tokenizer(rejected_response, add_special_tokens=False)

    # 构建完整序列: [BOS] + prompt + response + [EOS]
    def build_sequence(prompt_ids, response_ids):
        input_ids = (
            [tokenizer.bos_token_id] +
            prompt_ids +
            response_ids +
            [tokenizer.eos_token_id]
        )

        # Labels: prompt部分设为-100 (不计算loss)
        labels = (
            [-100] * (1 + len(prompt_ids)) +  # BOS + prompt
            response_ids +
            [tokenizer.eos_token_id]
        )

        # 截断
        input_ids = input_ids[:max_length]
        labels = labels[:max_length]

        # Attention mask
        attention_mask = [1] * len(input_ids)

        return {
            'input_ids': input_ids,
            'attention_mask': attention_mask,
            'labels': labels,
        }

    chosen_data = build_sequence(prompt_tokens['input_ids'],
                                 chosen_tokens['input_ids'])
    rejected_data = build_sequence(prompt_tokens['input_ids'],
                                   rejected_tokens['input_ids'])
```

```
    return {
        'chosen_input_ids': chosen_data['input_ids'],
        'chosen_attention_mask': chosen_data['attention_mask'],
        'chosen_labels': chosen_data['labels'],
        'rejected_input_ids': rejected_data['input_ids'],
        'rejected_attention_mask': rejected_data['attention_mask'],
        'rejected_labels': rejected_data['labels'],
    }
```

### 7.4 完整训练循环

Listing 4: DPO 训练循环

```
def train_dpo(policy_model, ref_model, train_dataloader,
              num_epochs=1, learning_rate=1e-6, beta=0.1,
              device='cuda', log_interval=100):
    """
    DPO训练主函数

    参数:
        policy_model:     待训练的策略模型
        ref_model:        参考模型 (冻结)
        train_dataloader: 训练数据加载器
        num_epochs:       训练轮数
        learning_rate:    学习率
        beta:             DPO温度参数
        device:           计算设备
        log_interval:     日志打印间隔
    """
    # 移动模型到设备
    policy_model = policy_model.to(device)
    ref_model = ref_model.to(device)

    # 冻结参考模型
    ref_model.eval()
    for param in ref_model.parameters():
        param.requires_grad = False

    # 优化器
    optimizer = torch.optim.AdamW(policy_model.parameters(), lr=
        learning_rate)

    # 训练循环
    policy_model.train()
    global_step = 0

    for epoch in range(num_epochs):
        total_loss = 0
        total_accuracy = 0
        num_batches = 0

        for batch in train_dataloader:
            # 移动数据到设备
            chosen_input_ids = batch['chosen_input_ids'].to(device)
            chosen_attention_mask = batch['chosen_attention_mask'].to(
                device)
```

```
            chosen_labels = batch['chosen_labels'].to(device)
            rejected_input_ids = batch['rejected_input_ids'].to(device)
            rejected_attention_mask = batch['rejected_attention_mask'].to(
                device)
            rejected_labels = batch['rejected_labels'].to(device)

            # 计算DPO损失
            loss, metrics = compute_dpo_loss(
                policy_model, ref_model,
                chosen_input_ids, chosen_attention_mask, chosen_labels,
                rejected_input_ids, rejected_attention_mask,
                    rejected_labels,
                beta=beta
            )

            # 反向传播
            optimizer.zero_grad()
            loss.backward()

            # 梯度裁剪（可选但推荐）
            torch.nn.utils.clip_grad_norm_(policy_model.parameters(),
                max_norm=1.0)

            optimizer.step()

            # 统计
            total_loss += metrics['loss']
            total_accuracy += metrics['accuracy']
            num_batches += 1
            global_step += 1

            # 打印日志
            if global_step % log_interval == 0:
                print(f"Step {global_step}: "
                      f"loss={metrics['loss']:.4f}, "
                      f"acc={metrics['accuracy']:.2%}, "
                      f"reward_diff={metrics['reward_diff']:.3f}")

        # Epoch结束统计
        avg_loss = total_loss / num_batches
        avg_accuracy = total_accuracy / num_batches
        print(f"Epoch {epoch+1}/{num_epochs}: "
              f"avg_loss={avg_loss:.4f}, avg_acc={avg_accuracy:.2%}")

    return policy_model
```

## 7.5 使用示例

Listing 5: DPO 使用示例

```
from transformers import AutoModelForCausalLM, AutoTokenizer
from torch.utils.data import DataLoader

# 加载模型和分词器
model_name = "gpt2"  # 或其他模型
tokenizer = AutoTokenizer.from_pretrained(model_name)
tokenizer.pad_token = tokenizer.eos_token
```

```python

# 加载策略模型（待训练）
policy_model = AutoModelForCausalLM.from_pretrained(model_name)

# 加载参考模型（冻结）
ref_model = AutoModelForCausalLM.from_pretrained(model_name)

# 准备数据（示例）
train_data = [
    {
        'prompt': 'What is the capital of France?',
        'chosen': 'The capital of France is Paris.',
        'rejected': 'I dont know.',
    },
    # ... 更多数据
]

# 预处理数据
processed_data = [
    preprocess_dpo_data(tokenizer, d['prompt'], d['chosen'], d['rejected'])
    for d in train_data
]

# 创建DataLoader（需要实现collate_fn处理padding）
# train_dataloader = DataLoader(processed_data, batch_size=4, ...)

# 训练
trained_model = train_dpo(
    policy_model,
    ref_model,
    train_dataloader,
    num_epochs=3,
    learning_rate=1e-6,
    beta=0.1
)

# 保存模型
trained_model.save_pretrained("dpo_trained_model")
```

# 8 DPO 的变体与扩展

DPO 自提出以来，研究者们发现了一些问题并提出了改进方法。

## 8.1 IPO：Identity Preference optimisation

**注记 8.1** (DPO 的过拟合问题)**.** DPO 可能存在过拟合问题：当训练数据有限或噪声较大时，模型可能过度拟合偏好标签，导致：

- 隐式奖励差 $\Delta\hat{r}$ 变得非常大
- 策略变得过于极端
- 泛化能力下降

**定义 8.1** (IPO 损失函数)**.** IPO（Identity Preference optimisation）通过修改损失函数来缓解过拟合：

$$\mathcal{L}_{\text{IPO}}(\theta) = \mathbb{E}_{(x,y_w,y_l)}\left[\left(\log\frac{\pi_\theta(y_w|x)}{\pi_{\text{ref}}(y_w|x)} - \log\frac{\pi_\theta(y_l|x)}{\pi_{\text{ref}}(y_l|x)} - \frac{1}{2\beta}\right)^2\right] \tag{50}$$

IPO 使用平方损失而非对数 Sigmoid 损失，目标是让隐式奖励差接近 $\frac{1}{2\beta}$，而非无限增大。

### 8.2 KTO：Kahneman-Tversky optimisation

**注记 8.2** (不需要成对数据)**.** DPO 需要成对的偏好数据 $(y_w, y_l)$，但收集这样的数据成本较高。KTO 只需要单独标注每个输出是好还是坏。

**定义 8.2** (KTO 损失函数)**.** KTO 的损失函数分别处理正负样本：

$$\begin{aligned}\mathcal{L}_{\text{KTO}}(\theta) = \mathbb{E}_{x,y_w}\left[w(y_w)\cdot(1-\sigma(\beta\hat{r}_\theta(x,y_w)-z_{\text{ref}}))\right] \\ + \mathbb{E}_{x,y_l}\left[w(y_l)\cdot\sigma(\beta\hat{r}_\theta(x,y_l)-z_{\text{ref}})\right]\end{aligned} \tag{51}$$

其中 $z_{\text{ref}}$ 是一个参考点，$w(\cdot)$ 是权重函数（来自 Kahneman-Tversky 的前景理论）。

### 8.3 其他变体

**注记 8.3** (DPO 变体概览)**.**

**SimPO**：简化版 DPO，去掉参考模型

$$\mathcal{L}_{\text{SimPO}} = -\mathbb{E}\left[\log\sigma\left(\frac{\beta}{|y_w|}\log\pi_\theta(y_w|x) - \frac{\beta}{|y_l|}\log\pi_\theta(y_l|x) - \gamma\right)\right] \tag{52}$$

使用长度归一化的对数概率，并引入 margin 参数 $\gamma$。

**ORPO**：Odds Ratio Preference optimisation

$$\mathcal{L}_{\text{ORPO}} = \mathcal{L}_{\text{SFT}}(y_w) - \lambda\log\sigma\left(\log\frac{\text{odds}(y_w)}{\text{odds}(y_l)}\right) \tag{53}$$

将 SFT 损失和偏好损失结合，使用 odds ratio 代替概率比。

**RSO**：Statistical Rejection Sampling optimisation
通过拒绝采样从最优策略生成数据，然后用 SFT 训练。

## 9 DPO vs RLHF：比较与选择

### 9.1 优缺点对比

**注记 9.1** (全面比较)**.**

| 方面 | RLHF (PPO) | DPO |
|---|---|---|
| 实现复杂度 | 高（需要多个组件） | 低（类似监督学习） |
| 内存需求 | 高（4 个模型） | 中（2 个模型） |
| 训练稳定性 | 较难（需要调参） | 较好（更稳定） |
| 计算效率 | 低（需要采样） | 高（纯前向传播） |
| 数据利用 | 在线（需要不断采样） | 离线（直接用数据集） |
| 理论基础 | 更一般的 RL 框架 | 需要特定假设 |
| 奖励模型 | 显式，可复用 | 隐式，与策略绑定 |
| 在线学习 | 原生支持 | 不直接支持 |

### 9.2 实践中的选择建议

**注记 9.2** (何时使用 DPO？)**.**

- 计算资源有限（GPU 内存不足以加载 4 个模型）

- 有现成的高质量偏好数据集
- 追求实现简单、快速迭代
- 训练稳定性比极致性能更重要

**注记 9.3** (何时使用 RLHF？)**.**

- 需要在线学习，不断从新策略采样
- 需要复用奖励模型用于其他目的
- 偏好数据有限，需要通过采样扩充
- 任务需要更精细的奖励塑形（reward shaping）

**注记 9.4** (混合方法)**.** 实践中也可以结合两种方法：

1. 先用 DPO 做初步对齐（快速、稳定）
2. 再用 RLHF 做精细调优（如果需要）
3. 或者用 DPO 训练的模型提取奖励，用于 RLHF

# 10 总结

## 10.1 核心公式回顾

| 概念 | 公式 |
|---|---|
| RLHF 目标 | $\max_\pi \mathbb{E}[r(x,y)] - \beta \cdot \mathrm{KL}(\pi \Vert \pi_{\text{ref}})$ |
| 最优策略 | $\pi^*(y\vert x) = \frac{1}{Z(x)}\pi_{\text{ref}}(y\vert x)\exp\left(\frac{r(x,y)}{\beta}\right)$ |
| 隐式奖励 | $\hat{r}_\theta(x,y) = \beta \log \frac{\pi_\theta(y\vert x)}{\pi_{\text{ref}}(y\vert x)}$ |
| DPO 损失 | $\mathcal{L}_{\text{DPO}} = -\mathbb{E}\left[\log \sigma(\hat{r}_\theta(x,y_w) - \hat{r}_\theta(x,y_l))\right]$ |
| DPO 梯度 | $\nabla\mathcal{L} = -\beta(1-\sigma(\Delta\hat{r}))(\nabla \log \pi(y_w) - \nabla \log \pi(y_l))$ |

表格 2: DPO 核心公式

## 10.2 要点总结

1. **DPO 的核心思想**：将奖励函数隐式化为策略的函数，从而绕过显式的奖励建模和强化学习
2. **关键推导**：从 RLHF 目标的最优策略出发，反解奖励函数，代入 Bradley-Terry 模型，配分函数消去
3. **损失函数**：与奖励模型损失结构相同，但用隐式奖励代替显式奖励
4. **实现简单**：本质上是监督学习，只需要计算对数概率
5. **理论等价**：在理想条件下与 RLHF 等价，实践中效果相当
6. **变体扩展**：IPO 解决过拟合，KTO 不需要成对数据，SimPO 去掉参考模型