基于一个微分对策问题的机器学习能力定量评价

摘要

羊犬追逐-逃逸模型是一个基于最优控制和微分对策的博弈对抗问题，可以用于测试机器学习算法的计算性能。对于给定参数的羊犬追逐-逃逸模型，本文通过建立数学模型，对羊和犬取得博弈胜利的条件及制胜策略进行了详细分析，同时提出了两种机器学习算法用于求解该模型，并基于熵权法给出了一套定量评价机器学习算法性能的评价体系。

针对问题一，本文首先将问题一转换为二人非协作零和博弈问题，通过构建微分形式下的动力系统建立随机微分博弈模型，并对终端状态、终端流形、终端支付方程的基本形式进行了分析。本文认为，犬的博弈制胜策略本质上为给定约束条件下的路径规划问题，因此在随机微分博弈的基础上，本文以犬的视角通过最优化终端支付方程将该问题转换为单目标极值优化问题，并采用动态规划和马尔可夫决策的方法对模型进行求解，给出了犬的最优围堵策略。

问题二要求在犬采用最优策略围堵的条件下确定羊成功逃逸需满足的条件。由于羊的速度、犬的速度、逃逸圆的半径三者为直接影响追逃博弈终端状态的变量，因此本文通过构建速度比例系数对羊和犬的速度进行约束，并进一步引入围捕半径针对博弈即将达到终端状态的犬和羊的运动情况进行分析，最终通过分类讨论给出了羊取得博弈胜利需满足的条件。

针对问题三，由于待解决的羊犬追逐-逃逸模型本身存在着反馈稀疏、难以直接确定有监督学习的监督学习部分、单步状态生成需与环境进行交互的特征，因此本文将羊视为自主进行路径规划的智能体，采用强化学习对该模型进行求解。考虑动作状态连续问题，本文通过引入深度确定性策略梯度 (DDPG) 对羊和犬的追逃博弈进行数值模拟，给出了羊经 500 个回合训练后的训练结果，以及羊通过算法学习到的最优路径规划策略。

对于问题四，由于学术界已经存在若干通用的用于定量评价强化学习算法的数值指标，本文基于熵权法对平均总回报、平均收敛步长、总回报曲线方差、收敛步长曲线方差 4 个评价指标带来的评价信息相融合，给出了系统评价强化学习算法求解羊犬追逃博弈问题计算性能的评价体系，并针对问题三中 DDPG 的学习效果进行了定量评价。

在问题五中，本文采用不同于 DDPG 的算法：近端策略优化 (PPO) 算法再一次对羊犬追逐-逃逸模型进行求解，并给出了 PPO 与 DDPG 的学习效果对比，以及 PPO 在问题四提出的评价方法下的计算得分。本文的评价结果显示，在羊犬追逐-逃逸问题中，PPO 算法的综合计算性能要优于 DDPG。

最后，本文给出了模型的优点以及下一步的改进方向。

关键词：随机微分博弈　马尔可夫决策　强化学习　策略梯度　熵权法　PPO

1 问题的背景重述

1.1 问题背景

羊犬博弈是一类特殊的微分对策问题，机器学习方法在求解此类涉及最优控制、微分对策的问题时计算效果往往比较差，而本文提出的羊犬博弈问题可以用于测试机器学习方法的性能。

1.2 问题重述

在本文提出的羊犬博弈问题中，羊在半径为 R 的圆形圈内按照定常速率 v 向圆外进行逃逸，在逃逸路径上需满足每一点与圆心的距离随时间单调不减，但具有任意方向的转弯能力，犬沿着圆周以定常速率 V 围堵以防止羊逃逸。本题需解决的问题如下：

(1) 基于运动学建模求解犬的最优围堵策略；

(2) 建模求解当犬以 (1) 中所述最优围堵策略围堵羊时羊取得博弈胜利的条件；

(3) 提出一种机器学习方法，使得羊在满足 (2) 的条件下经训练后可以获得博弈胜利；

(4) 设计一套评价体系，用于定量评价 (3) 中给出的机器学习算法的学习能力；

(5) 提出并利用 (4) 中的评价体系定量评价更多羊逃逸的机器学习方法。

2 问题分析

2.1 问题一的分析

考虑羊犬追逐-逃逸博弈问题在时空维度下的连续性，不妨考虑构造羊和犬的随机微分对策方程，随后将最优控制转化为单目标极值优化问题求解该问题。

2.2 问题二的分析

在问题一的基础上，考虑羊和犬博弈结果与 v、V、R 之间的关联性，因此不妨构造比例系数对三个变量进行约束，随后分类讨论羊成功逃逸需满足的条件。

2.3 问题三的分析

由于羊犬博弈问题本质上属于基于序列的决策问题，同时考虑羊犬博弈问题中存在的反馈比较稀疏、有监督学习的监督部分难以直接构造、需不断与环境进行交互等若干特征，不妨采用强化学习算法指导羊的逃逸过程，并引入策略梯度使得算法能学习连续的动作状态。

2.4 问题四的分析

由于已经存在若干通用的性能指标用于分析强化学习算法的学习效果，因此不妨将综合评价方法与已经存在的评价指标相融合，从而建立一个具有强泛化能力的机器学习综合评价模型。

2.5 问题五的分析

在问题四的基础上，本题只需要进一步提出若干种用于求解羊犬追逐-逃逸问题的机器学习算法，并利用问题四提出的综合评价模型对算法进行评价即可。

3 基本假设

1. 假设羊和犬的追逃博弈为二人完全非协作博弈，在博弈过程中，羊和犬均出于最大化收益或最小化风险的目标做出决策，二者均不允许持续地做出完全随机决策，羊和犬在决策时均满足理性人假设；
2. 假设羊和犬不存在任何除观测对方的空间坐标、速度及速度方向等环境状态信息之外的任何交流，且羊和犬仅允许基于观测到的运动学状态信息进行决策；
3. 假设羊和犬的追逃博弈仅在有限时域内完成；
4. 为简化模型，假设羊和犬的速度方向均为瞬时变化的，忽略加速度对速度的影响；
5. 假设羊和犬均基于当前时刻下的观测状态采取动作以获得当下的最优相应，即犬和羊的决策过程均具有 Markov 性。

4 符号约定

符号	含义
$\odot O, \partial O, R$	基准圆的圆心、边界及半径
t_0, t_τ	羊犬追逃模型的起始时间和终止时间
$(\rho_s(t), \psi_s(t))$	羊和犬的空间极坐标
$v(v_x(t), v_y(t)), V(V_x(t), V_y(t))$	羊和犬的瞬时速度
$\theta_s(t), \theta_d(t)$	羊和犬的速度夹角
ζ	状态空间
$\Gamma = \{\Gamma^1, \Gamma^2, \Gamma^3\}$	终端状态集
Π	终端流形
$J(\cdot)$	羊犬博弈的终端支付方程
u_s, u_d	羊和犬的控制策略
$D(t)$	羊距离边界的最小距离函数
δ_t	决策时间间隔

续上表：

符号	含义
η	速度比例系数
r	羊的逃逸半径
s_i	第 i 个决策时域内的环境状态
a_i	羊在第 i 个决策时域内采取的动作
r_i	羊在第 i 个决策时域内获得的回报
$\pi(s,a)$	策略
$G(j)$	羊在一个回合内获得的总回报
$V()$	值函数
θ,ϕ	Actor 和 Critic 网络的参数
α,β	Actor 和 Critic 网络的学习率
d_k	第 k 个评价指标

5 模型的建立与求解

5.1 问题一

5.1.1 模型准备

由题意，不妨假设羊犬追逐-逃逸模型以半径为 R、圆心为 O 的圆为基准圆，记圆的边界为 ∂O，圆的内部为 $\odot O\backslash\partial O$。假设追逐-逃逸起始时刻为 t_0，终端时刻为 t_τ，羊在 t_0 时刻从圆心 O 出发，在 $\odot O\backslash\partial O$ 沿任意方向的速度 v 向外逃逸，犬随即以与圆边界相切的速度 V 沿边界 ∂O 进行围堵。不妨以羊向外逃逸的起点 O 为原点建立空间直角坐标系 xOy，构建羊犬追逃模型示意图如图1所示，图中*表示羊，*表示犬：

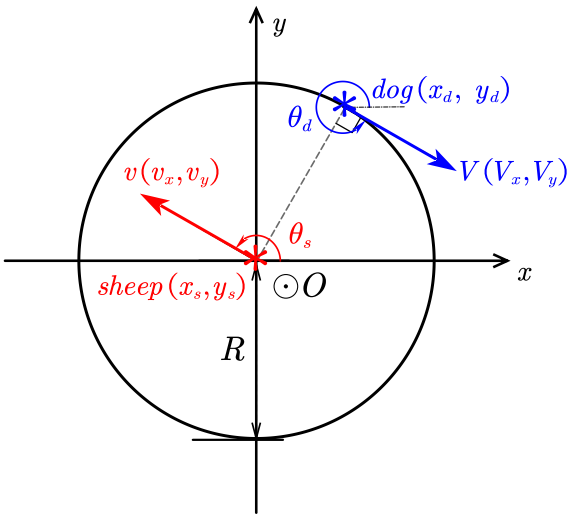


图 1: 羊犬追逐-逃逸模型的平面坐标示意图

不妨假设羊和犬在 t 时刻的空间坐标分别为 $(x_s(t),y_s(t))$、$(x_d(t),y_d(t))$，其中 $t\in[t_0,t_\tau]$。考虑如下极坐标变换：

$$\begin{cases} x=\rho cos(\psi); \\ y=\rho sin(\psi); \end{cases} \tag{5.1}$$

因此在极坐标系架构下羊和犬在 t 时刻的空间坐标还可以分别用 $(\rho_s(t),\psi_s(t))$、$(\rho_d(t),\psi_d(t))$ 表示，其中 $t\in[t_0,t_\tau]$。

为便于表述，本文将羊和犬的瞬时速度视为矢量，向坐标系 xOy 两个正交方向进行分解，其矢量分解式如式(5.2)所示：

$$\begin{cases} \vec{V}=\vec{V_x}+\vec{V_y}; \\ \vec{v}=\vec{v_x}+\vec{v_y}; \end{cases} \tag{5.2}$$

若忽略矢量符号，不妨简记羊的瞬时速度为 $v(v_x(t),v_y(t))$，犬的瞬时速度为 $V(V_x(t),V_y(t))$。

本文取水平向右为水平正方向，记羊的瞬时速度 v 与水平正方向的矢量角为 $\theta_s(t)$，犬的瞬时速度 V 与水平正方向的矢量角为 $\theta_d(t)$，其中 θ_s 和 θ_d 均为以逆时针方向为正方向的有向角，如图1所示。不妨记 $\delta\theta(t)=\theta_d(t)-\theta_s(t)$ 为羊犬速度矢量角的差值角，该角同样为以逆时针方向为正方向的有向角，该角定量描述了犬和羊在追逃过程中任意时刻 t 下瞬时移动方向的差异，其中 $t\in[t_0,t_\tau]$。恰当地建立坐标系可以更好地指导本文模型的求解。

5.1.2 基于最小化终端支付方程的最优围堵策略

本文基于最优控制 [1] 和随机微分对策方法 [2] 求解羊犬追逐-逃逸模型。假设状态空间为 $\zeta=(x_s(t),y_s(t),x_d(t),y_d(t))\in\mathbb{R}^4$，构建动力系统及其初始状态如式(5.3)所示：

$$\begin{cases} \dot{x_d}=V_x=Vcos\theta_d \\ \dot{y_d}=V_y=Vsin\theta_d \\ \dot{x_s}=v_x=vcos\theta_s \\ \dot{y_s}=v_y=vsin\theta_s \\ x_d(t_0)=x_{d,t_0},y_d(t_0)=y_{d,t_0} \\ x_s(t_0)=0,y_s(t_0)=0 \end{cases} \tag{5.3}$$

则可由如下形式定义犬羊追逐-逃逸随机微分博弈模型：

$$\dot{\zeta}=F(t,\zeta,u_d,u_s,W) \tag{5.4}$$

其中 $t\in\mathbb{R}^*$ 为博弈时间，$u_d\in\mathbb{E}^p,u_s\in\mathbb{E}^q$ 分别为犬和羊的控制策略变量；$W=W(t,\omega)\in\mathbb{E}^r$ 为某一定义在完备概率空间 (Ω,A,μ) 上的随机过程，其中 Ω 是非空抽象集，A 为 Ω 的子集上的一个 σ-代数，且 μ 是 A 上的一个概率测度。

事实上，由于羊总可以选择策略使得追逃问题在无限长时间内完成，因此本文假设羊犬追逃博弈仅在有限时间内完成，因此考虑存在最大终端时间 $t_{\tau,max}$，使得 $t_0 \leqslant t_\tau \leqslant t_{\tau,max}$，因此在本文中博弈终止时有两个条件：羊成功逃逸或羊未成功逃逸，其中羊未成功逃逸可分为两种情况：羊被犬成功围捕或羊的逃逸时间超过了最大终端时间 $t_{\tau,max}$。因此不妨建立状态集 $\Gamma = \{\Gamma^1, \Gamma^2, \Gamma^3\}$ 区分上述三种终端状态，其满足

$$\begin{cases} \Gamma^1 = \{\zeta | \text{羊成功逃逸}\} \\ \Gamma^2 = \{\zeta | x_d(t_\tau) = x_s(t_\tau), y_d(t_\tau) = y_s(t_\tau)\} \\ \Gamma^3 = \{\zeta | t_\tau > t_{\tau,max}\} \end{cases}$$

由于追逃过程中羊的优化目标是逃离 $\odot O$ 同时尽可能扩大羊与犬之间的欧氏距离，而犬的优化目标与之相反，在追逃过程中犬希望在保证轨迹满足边界 ∂O 限制的同时最小化犬与羊之间的直线距离，如果仅考虑有限终端时刻的博弈结果，该博弈存在零和博弈的特征，因此不妨构建终端支付方程如式(5.5)所示：

$$J\left(x_s(t_\tau), y_s(t_\tau), x_d(t_\tau), y_d(t_\tau)\right) = \frac{1}{2}\left[\left(x_d(t_\tau) - x_s(t_\tau)\right)^2 + \left(y_d(t_\tau) - y_s(t_\tau)\right)^2\right] \tag{5.5}$$

当仅考虑状态 Γ^2 时，即保证犬成功围捕羊、羊犬追逃博弈仅在有限时间内完成，存在终端流形 $\Pi \in \mathbb{R}^4$，满足：

$$\begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} x_d & y_d & x_s & y_s \end{bmatrix}^T = 0_{2\times 1} \tag{5.6}$$

观察可知这也是状态 Γ^2 的定义式，它等价于终端时刻羊和犬的位置坐标相同。假设羊和犬在时间 t 内的容许控制量为 $u_s(t)$ 和 $u_d(t)$，其中 $t_0 \leqslant t \leqslant t_\tau$，以终端支付方程为优化目标，则犬的单目标优化方程如式(5.7)所示：

$$\min_{u_d(\cdot)} \max_{u_s(\cdot)} J\left(x_s(\cdot), y_s(\cdot), x_d(\cdot), y_d(\cdot); u_s, u_d\right) \tag{5.7}$$

其中 $u_s(t)$ 和 $u_d(t)$ 也可以表述为羊和犬在各自在观察到对方的行动状态后做出的逃逸-追逐策略。

5.1.3 模型求解

动态规划是一种常用于解决微分对策问题的解法 [4]，可以通过将连续微分对策问题离散化化解为 N 个离散微分对策问题来求解。仅考虑有限时间内的羊犬追逃博弈问题，由于在时刻 t 犬仅能根据视觉和记忆判断 $[t_0, t]$ 时刻羊的逃逸策略 $[u_s(t_0), u_s(t)]$，而不能提前知晓羊在未发生时间区间 $[t, t_\tau]$ 内的逃逸策略 $[u_s(t_0), u_s(t)]$，而羊完全可以采取服从某一随机过程 W 的控制策略 u_s 增强其逃跑路径的随机性，从而对犬的追逐产生迷惑，提高其成功逃逸的可能性，因此犬仅能根据时刻 t 所获得的博弈信息决定当下的控制策略 $u_d(t)$。因此不妨将追逃博弈时间区间 $[t_0, t_\tau]$ 平均剖分为 N 份，每份的长

度为 δ_t，由于动态规划过程实际为 Markov 决策过程，满足以下性质：下一时刻的最优控制状态仅由当前时刻的系统状态决定，求解犬对应待优化方程(5.7)的全局最优解的最优控制策略 u_d^* 可转化为由贪心策略求解该方程在每一小段时间区间内的最优控制策略 $u_d(t_0+i\delta_t)^*, i=0,1,\ldots,N-1$，从而得到全局满意的控制策略。在某些情况下基于动态规划的算法可能求出全局最优控制策略。

考虑待优化方程(5.7), 由于题目要求羊在逃逸过程中轨迹需满足约束

$$\sqrt{x_s\left(t_0+(i+1)\delta_t\right)^2+y_s\left(t_0+(i+1)\delta_t\right)^2}\geqslant\sqrt{x_s\left(t_0+i\delta_t\right)^2+y_s\left(t_0+i\delta_t\right)^2},\ i=0,\ldots,N-1,$$

且需要在有限时间内完成逃逸动作，因此假设羊距离 ∂O 的最短距离函数为 $D(t)$，分析可知 $D(t)$ 在时域 $[t_0,t_\tau]$ 内必为单调非增连续函数。若将犬和羊的坐标由直角坐标系转换为极坐标系，显然 $D(t)=\rho_d-\rho_s$ 为单调不增量，该结论表明随博弈时间推移，羊和犬在极径坐标视角下必然逐渐接近，因此求解犬的单目标优化方程(5.7)可以转化为在任意剖分后的时间区间内最小化犬和羊的极角坐标差，即最小化以下 Markov 决策方程(5.8)：

$$\min_{u_d(\cdot)}|\psi_d(t_0+i\delta_t)-\psi_s(t_0+i\delta_t)|,\ \forall i=0,1,\ldots,N-1, \tag{5.8}$$

其中 $\psi_d(t)$ 和 $\psi_s(t)$ 分别为 t 时刻犬和羊的极角坐标（以下称位移夹角），它表示 t 时刻将犬和羊的实际位置与原点相连，所成直线与水平正方向形成的有向角，如图2所示。

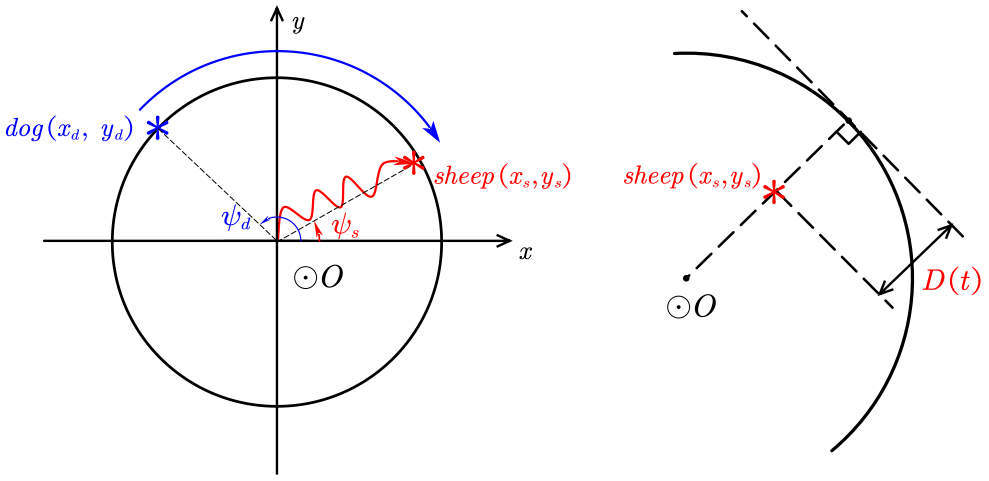


图 2: 羊犬位移夹角和最短距离函数 $D(t)$ 示意图

综上所述，本文制定**犬的最优围堵策略** u_d^* 如下：在羊犬追逐-逃逸博弈的 $i\delta_t$ 时刻，考虑最小化目标函数(5.8)，将当前时刻犬和羊的位移夹角 $\psi_d(t)$ 和 $\psi_s(t)$ 视为犬在当下博弈中获得的完全信息，并根据此信息调整追逐方向，采取当前时间步下的最优追逐策略，从而最小化在 $[i\delta_t,(i+1)\delta_t]$ 时间区间内犬和羊的位移夹角。犬依据 $i\delta_t$ 时刻捕获的位移夹角信息始终沿 ∂O 朝向缩小该夹角的方向移动，直至羊移动至犬的捕获半径内，犬将羊成功捕获，将此控制策略 $u_d(t_0+i\delta_t)$ 视为该时域内的最优控制策略。再考虑极限状态 $\delta_t\to 0$，由于极限状态下剖分区间长度 δ_t 近似为 0，离散状态的微分博弈策略

可近似视为连续状态下的随机微分策略，因此犬在连续状态下的最优随机微分策略为所有离散状态下的单步最优策略组成的集合 $u_d^* = \{u_d(t + i\delta_t)^*\}, \delta_t \to 0, i = 0, \ldots, N-1$。该控制策略的示意图如图3所示。

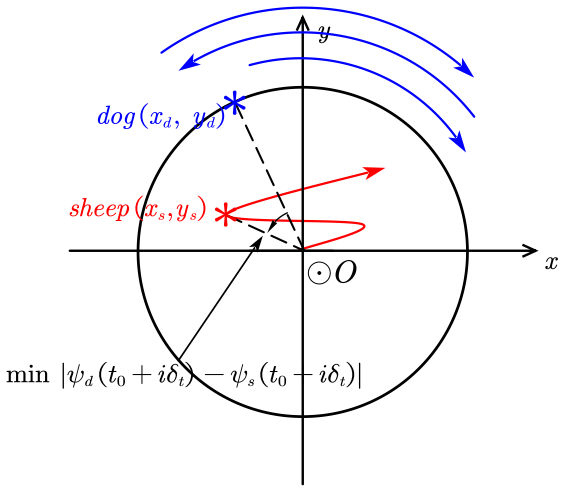


图 3: 犬的最优围捕策略示意图

文献 [3] 证明了如下结果：当存在捕获半径 r 时，考虑极大极小值均衡，追逐方在 $\mathbb{R}^2$ 平面内的捕获区间为圆形，在此捕获区间内追逐方一定可以捕获被追逐方。在本题中犬所对应的捕获区间对应为 ∂O 的一部分，在图4中用蓝色曲线表示，在此段曲线内羊沿犬能追逐的任意方向逃跑都会被捕获，因此若羊实现成功逃逸，则羊必然在犬的捕获区间以外逃逸。此事件发生的时间区间恰好对应终端状态为 Γ^1 和 Γ^2 时以终端时刻 t_τ 为末尾时刻的时间区间 $[t_\tau - \delta_t, t_\tau]$，且羊犬博弈时间有限的假设保证了该事件必然发生。绘制捕获前羊犬追逃博弈的示意图如图4所示。

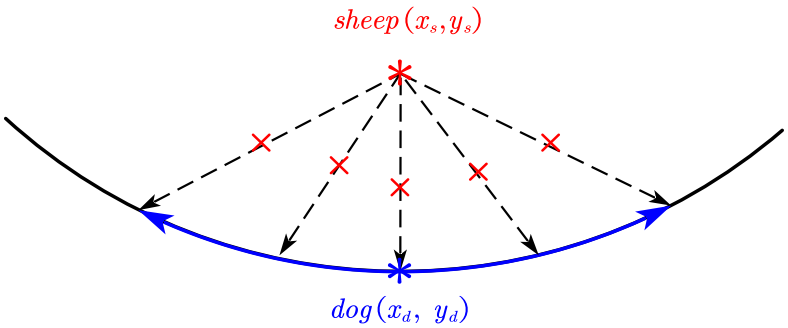


图 4: 犬的最终捕获态示意图

5.2 问题二

5.2.1 模型准备

由图4可知，如果犬具有很大的切向速度 V，当羊逃逸至 ∂O 内边缘时，犬可以在其对应边的任意位置围堵羊，使得羊完全不能逃逸。由分析可知，假设羊和犬的速度比例系数为 η：

$$\eta = \frac{v}{V};$$

羊和犬在边界处的逃逸成功条件与比例系数 η 的取值有关，且进一步分析可知 η 和 $\odot O$ 的半径 R 决定了羊犬博弈的终端状态 Γ。考虑临界条件：当羊沿直线产生长度为 R 的位移时，犬沿基准圆边界产生长度为 πR 的路程，此时速度比例系数 $\eta = \frac{v}{V} = \frac{R}{\pi R} = \frac{1}{\pi}$，因此不妨考虑针对比例系数 η 的取值进行讨论，求解在犬采用最优控制 u_d^* 的条件下，羊在追逃博弈中胜出的条件。

5.2.2 基于羊犬速度比例系数的博弈制胜条件

情况 1： $\eta > \dfrac{1}{\pi}$

当 $\eta > \frac{1}{\pi}$ 时，若犬采取最优围捕策略 u_d^* 围捕羊，羊只需采取以下控制策略 u_s 即可取得博弈胜利：

由于羊和犬的支付方程恰好相同，但具有完全相反的优化目标，考虑羊的控制策略 u_s 使得 Markov 决策方程(5.9)在第一个决策时间区间取得最大值：

$$\max_{u_s(\cdot)} |\psi_d(t_0 + i\delta_t) - \psi_s(t_0 + i\delta_t)|,\ i = 0, \tag{5.9}$$

则羊只需保持按直线朝向背向犬初始位置的方向逃逸即可保证胜出。不失一般性假设犬沿顺时针方向沿 ∂O 进行追捕，该轮博弈初始状态和终端状态示意图如图5所示。

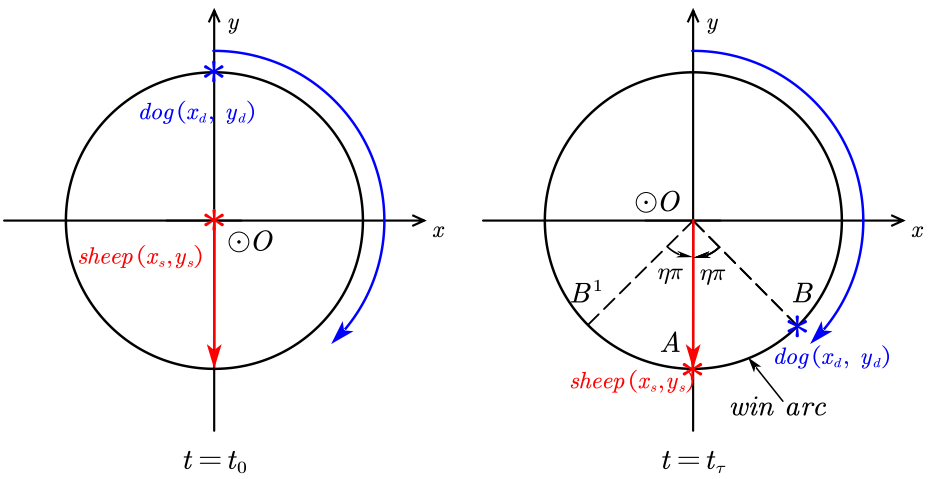


图 5: $\eta > \dfrac{1}{\pi}$ 条件下羊成功逃逸示意图

由图5可知，当犬按照始终缩小羊与犬的位移夹角选择围捕路径时，由于羊犬速度比例系数 η 的限制，当羊从起始点 O 沿背向犬的起始点的直线逃逸长度为 R 的距离到达 ∂O 上的点 A 时，犬从起始点经过路径 $\frac{R}{\eta}$ 到达点 B，显然此时羊一定胜出。进一步分析可知在该羊犬速度比例系数的限制下，在 ∂O 上存在羊的必胜弧 win arc=$\overset{\frown}{AB}$，羊只需从原点 O 出发，沿直线保持速度 v 朝向弧 $\overset{\frown}{AB}$ 上的任意一点（不包括点 B）前进，即可在终端时刻 t_τ 完全不被犬追上，从而成功出圈。必胜弧 win arc 对应的圆心角恰好为 $\eta\pi$，η 越大该弧越大，博弈的临界状态为羊沿直线 OB 进行逃逸，恰好在终端时刻 τ 被犬捕获。同理，当犬沿逆时针方向追捕时，同样存在羊的必胜弧 win arc=$\overset{\frown}{AB^1}$，羊从原点出发沿直线朝向该弧上任意一点保持匀速 v 前进（除临界点 B^1 以外）均可实现成功出圈。因此，在满足条件 $\eta=\frac{1}{\pi}$ 时，即使犬按照最优围堵策略进行围堵，羊按照以上控制策略 u_s 在博弈中一定可以获得胜利。

情况 2： $\eta \leqslant \dfrac{1}{\pi}$

当 $\eta \leqslant \frac{1}{\pi}$ 时，如果羊采用与情况 1 相同的逃逸策略，犬将先于羊到达羊途径边界 ∂O 的逃逸点，从而成功对羊实施围堵，因此羊将不能成功完成逃逸动作，该情形的示意图如图6所示。若仅考虑羊和犬在有限时间内的博弈，即仅考虑终端状态为 Γ^1 和 Γ^2 的情形，若羊采取非直线随机逃逸策略，则有可能因为犬的追逐动作受羊逃逸策略的随机性干扰从而使羊完成逃逸动作；否则，羊将难以完成逃逸。

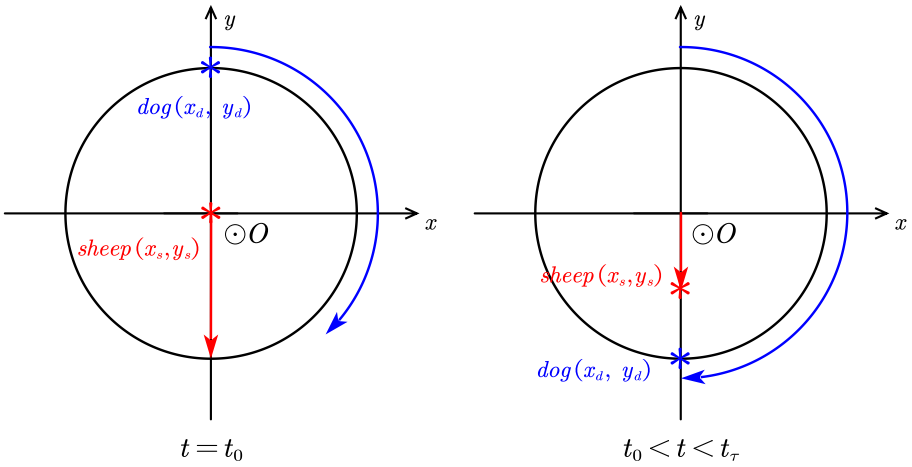


图 6: $\eta \leqslant \dfrac{1}{\pi}$ 条件下羊逃逸失败示意图

由于追逐-逃逸博弈仅在有限时间内完成，因此在终端时刻 t_τ 前必然出现如图4所示的羊即将穿越边界 ∂O 的情况，若给定羊和犬的速度条件，则终端情形可能为 Γ^1 或 Γ^2 中的任意一种，羊和犬具体执行的追逃策略与微分对策的终端状态无关，而仅仅与羊在接近突破边界 ∂O 的瞬时时刻的系统状态有关，因此不妨仅考虑该追逃系统即将到达终端时刻的情况。但由图4可知，终端状态到达 Γ^1 或 Γ^2 既取决于羊和犬的速度关系，也取决于基准圆的半径 R，因此不妨基于羊和犬的速度 v 和 V、基准圆 O 的半径 R 讨论

该随机微分博弈的终端情形。

由题目及前文分析可知，假设 δ_t 时间内羊仍按照直线逃逸，则羊逃逸途径的路程为 $v\delta_t$，犬围堵羊途径的边界长度为 $V\delta_t$，以羊在终端时刻前的坐标 $(x_s(t_\tau-\delta_t), y_s(t_\tau-\delta_t))$ 为圆心、$v\delta_t$ 为半径构建圆，羊在 δ_t 时间内可以到达该圆边界上任意一点，将此圆与基准圆 $\odot O$ 相割，使得该圆的直径恰好为相割形成的割弦，此时相割圆在圆 O 上形成的弧记为 $\overset{\frown}{GH}$。由解析几何知识可知，该圆的直径与弧 $\overset{\frown}{GH}$ 的长度存在以下关系：

$$|\overset{\frown}{GH}| = 2R\arcsin\left(\frac{2r}{2R}\right), \tag{5.10}$$

其中 $2r$ 为切割圆的直径，$|\overset{\frown}{GH}|$ 为弧 $\overset{\frown}{GH}$ 的弧长。不妨假设狗的捕获区域为弧 $\overset{\frown}{PQ}$，则临界状态为弧 $\overset{\frown}{PQ}$ 恰好与弧 $\overset{\frown}{GH}$ 重合，如图7所示。

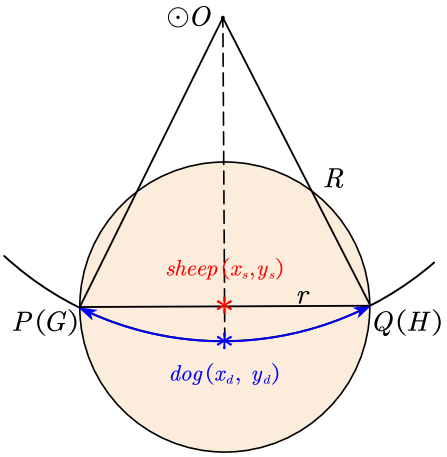


图 7: 羊恰好无法逃逸的临界状态示意图

由式(5.10)可知，当博弈进行至图7所示临界状态时，几何量与运动量具有等价关系：$|\overset{\frown}{GH}| = V\delta_t$，$2r = v\delta_t$，将几何量用运动量替换可将式(5.10)等价变换为下式：

$$V\delta_t = 2R\arcsin\left(\frac{v\delta_t}{2R}\right),$$

观察图像可知，该临界状态对应的是羊在 δ_t 时间内朝向任意方向向 $\odot O$ 外逃逸均需经过弧 $\overset{\frown}{PQ}$，因此无法逃出犬的围捕区域，$t = t_\tau$ 时博弈终端状态进入 Γ^2。由于剖分时间间隔 δ_t 的任意性，不妨取 $\delta_t = 1$，此时临界速度状态可表述为：

$$V = 2R\arcsin\left(\frac{v}{2R}\right), \tag{5.11}$$

其中 R 为 $\odot O$ 的半径。

情况 2-1： $V \geqslant 2R\arcsin\left(\frac{v}{2R}\right)$

当犬和羊的速度满足 $V \geqslant 2R\arcsin(\frac{v}{2R})$ 时，弧 $\overset{\frown}{GH} \subseteq \overset{\frown}{PQ}$，即羊在 $t_\tau - \delta_t$ 时刻沿任意方向向 $\odot O$ 外逃逸均需经过弧 $\overset{\frown}{PQ}$，羊逃逸至 ∂O 即被犬捕获，终端状态进入 Γ^2，不

存在羊逃逸成功的情形。

情况 2-2： $V < 2R\arcsin\left(\dfrac{v}{2R}\right)$

当犬和羊的速度满足 $V < 2R\arcsin(\frac{v}{2R})$ 时，弧 $\overset{\frown}{PQ} \subseteq \overset{\frown}{GH}$，此时羊可以朝向犬追捕区域以外的圆弧 $\overset{\frown}{GH} \setminus \overset{\frown}{PQ}$ 逃逸，此时犬因最大位移不足不能对羊完成围捕，羊成功完成逃逸动作，终端状态进入 Γ^1。以上两种情形的示意图如图8所示。

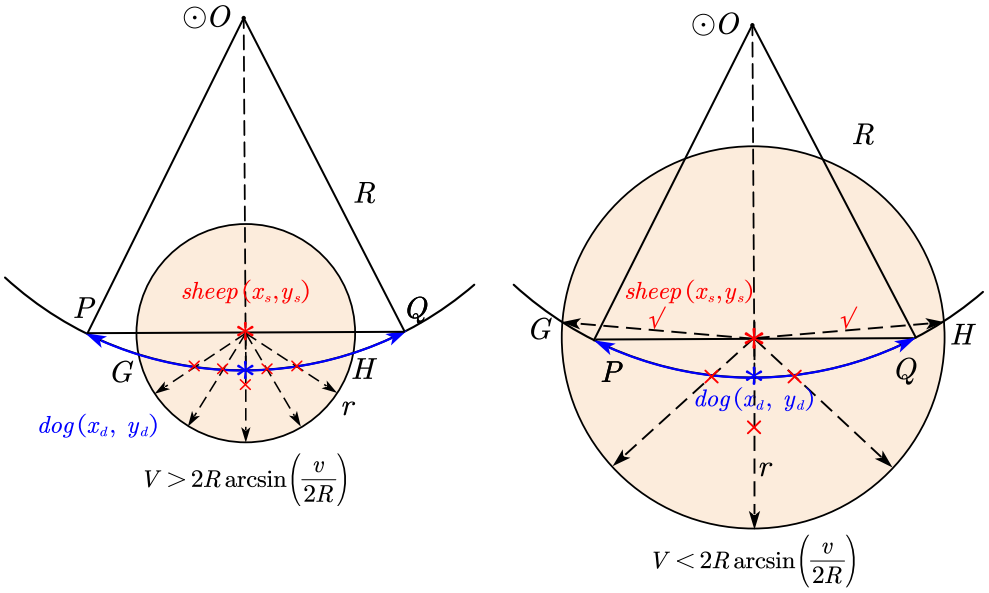


图 8: 情况 2-1、情况 2-2 的讨论示意图

综上所述，羊可以完成逃逸动作的系统条件如下式所示：

$$\begin{cases} \eta > \dfrac{1}{\pi}; \\ V < 2R\arcsin\left(\dfrac{v}{2R}\right),\ if\ \eta \leqslant \dfrac{1}{\pi}. \end{cases} \tag{5.12}$$

5.3 问题三

5.3.1 模型准备

本题要求犬采取最优追捕策略对羊进行围堵，羊以逃出基准圆为目标与犬博弈。由于待解决的羊犬追逐-逃逸模型本身存在着反馈稀疏、难以直接确定有监督学习的监督学习部分、单步状态生成需与环境进行交互的特征，，且强化学习方法在求解微分博弈问题中具有一定优势，若将羊视为智能体 Agent、犬视为可移动障碍体，则可将本题抽象为单一智能体在连续动作空间下的逃逸路径规划问题，因此本题引入深度确定性策略梯度 DDPG(Deep Deterministic Policy Gradient)[5] 用于训练羊躲避犬的追捕同时实现逃逸路径最优化，使得羊能在满足逃逸条件下成功实现逃逸。

5.3.2 基于 DDPG 的羊逃逸强化学习模型

DDPG 方法采用 Actor-Critic 框架构建强化学习神经网络。本文仍在有限博弈时间内考虑该博弈，假设第 i 个时间区间为 $[t_0+i\delta_t, t_0+(i+1)\delta_t]$，此时羊作为待学习的智能体称为 Agent，Agent 能观测到的环境状态为 s_i，在本文中即犬的位置坐标、速度方向等围捕过程中的运动学信息，在观测到环境状态 s_i 后羊随即采取动作 a_i 与环境产生交互，交互完成以后犬由状态 s_i 跳转至状态 s_{i+1}，此时系统根据第 $i+1$ 时刻的环境状态对羊的动作 a_i 基于回报函数 $r(s_i,a_i,s_{i+1})$ 进行评价，羊获得的回报 $r_{i+1}=r(s_i,a_i,s_{i+1})$ 即为羊在上一时刻采取动作 a_i 的反馈。回报函数具有以下特点：当羊采取正确动作后该函数可以给予羊一定的正向回报，而当羊采取错误动作后该函数将会给予羊一定的负向惩罚，羊在不断试错的过程中不断学习正确的动作，进而实现持续地最大化总回报 G。用 $\pi=u_s$ 表示羊的逃逸控制策略，在一般强化学习模型中，考虑实际羊逃逸策略选择的随机性，策略 $\pi(s,a)$ 是当羊观测到环境状态 s 后采取行为 a 的概率分布，即：

$$\pi(s,a)=P(a_i=a|s_i=s),\ i=0,\dots,N-1 \tag{5.13}$$

但在 DDPG 中，策略 π 为一确定性策略，这能保证学习完成后不会因随机生成实际的路径规划策略而对羊成功逃逸产生不利影响。

事实上，羊逃逸时越接近博弈结束时刻的状态相对于博弈开始的状态对于羊学习到正确的逃逸方法具有更强的重要性，因此考虑采用折扣率 $\gamma\in[0,1]$ 对羊的单步回报进行加权，计算羊的总回报 G：

$$G(j)=\sum_{i=0}^{N-1}\gamma^{N-(i+1)}r(s_i,a_i,s_{i+1}). \tag{5.14}$$

其中 j 为羊一次博弈完成后形成的轨迹。强化学习的目标函数为

$$J(\theta)=\mathbb{E}_{j\ p_\theta(j)}[G(j)]=\mathbb{E}_{j\ p_\theta(j)}[\sum_{i=0}^{N-1}\gamma^{N-(i+1)}r(s_i,a_i,s_{i+1})], \tag{5.15}$$

其中 θ 为决定策略输出的参数。策略梯度 $\nabla_\theta(s,a)$ 为

$$\nabla_\theta J(\theta)=\begin{pmatrix}\frac{\partial J(\theta)}{\partial\theta_1}\\ \vdots\\ \frac{\partial J(\theta)}{\partial\theta_n}\end{pmatrix}. \tag{5.16}$$

在 Actor-Critic 框架下，利用贝尔曼方程递归近似计算从时刻 $t_0+i\delta_t$ 开始的回报：

$$\hat{G}(j_{t_0+i\delta_t:t_\tau})=r_{i+1}+\gamma V_\phi(s_{i+1}). \tag{5.17}$$

确定性策略 Actor-Critic 的架构如下：Actor 是用于学习如何根据环境 s 给出羊的最优动作 a 从而形成确定性策略 u_s 的多层人工神经网络，Critic 是用于估计在输入动

作 a 时输出该动作的准确回报 r 的多层人工神经网络，因此 Actor-Critic 网络的学习过程分两步进行。动作 a 作为连续向量是 Actor 的优化目标：

$$a = \arg\max \log \pi(s, a), \tag{5.18}$$

值函数 $V(s_i)$ 作为优化目标用于减小学习过程中产生的方差，是 Critic 的优化目标：

$$\min \left(\hat{G}(j_{t_0 + i\delta_t : t_\tau}) - V(s_i) \right)^2, \tag{5.19}$$

假设两个网络的参数分别为 θ 和 ϕ，则两个网络在学习阶段基于前向传播分别按梯度更新参数，直到确定性策略 π 收敛（即给定初始状态 s_0，羊采取的策略从随机游走到逐渐采取稳定策略），其更新公式如式5.20所示：

$$\begin{cases} \theta \leftarrow \theta + \alpha \dfrac{\partial}{\partial \theta} \log \pi(s, a) \\ \phi \leftarrow \phi + \beta \dfrac{\partial}{\partial \phi} V(s) \end{cases} \tag{5.20}$$

其中 $\alpha > 0$ 和 $\beta > 0$ 分别为 Actor 网络和 Critic 网络的学习率。

在本文中，Actor 和 Critic 两个网络均采用三层以 Relu 作为激活函数的 Dense 层架构而成，并使用 Adam 优化器进行优化，其工作流程如图9所示。

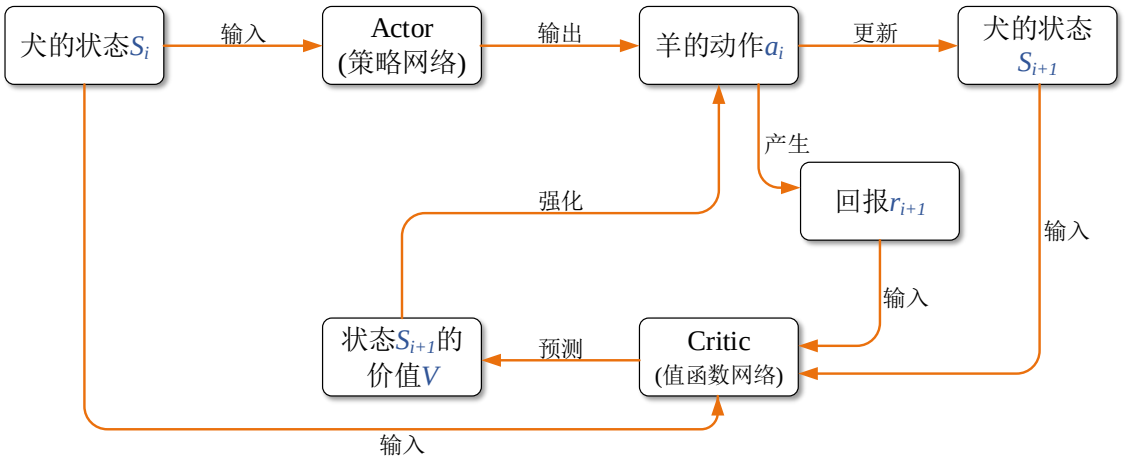


图 9: DDPG 框架下的羊犬逃逸训练流程

5.3.3 模型求解

本文设计 Actor-Critic 的学习率 $\alpha = 0.001, \beta = 0.001$，折扣率 $\gamma = 0.9$，同时设置容量为 30 的经验回放池作为羊的长期记忆，防止优化过程中目标函数因震荡导致不收敛。为指导羊通过回报的正反馈进行学习，本文设计回报函数 $r(s_i, a_i, s_{i+1})$ 如下：

$$r=\begin{cases}100.0,\ \text{if sheep win the game},\\ 1.0,\ \text{if}\,|\psi_{s,i+1}-\psi_{d,i+1}|>|\psi_{s,i}-\psi_{d,i}|\ \text{and}\ \sqrt{x_{s,i+1}^2+y_{s,i+1}^2}>\sqrt{x_{s,i}^2+y_{s,i}^2},\\ 0.5,\ \text{if}\,|\psi_{s,i+1}-\psi_{d,i+1}|\leqslant|\psi_{s,i}-\psi_{d,i}|\ \text{and}\ \sqrt{x_{s,i+1}^2+y_{s,i+1}^2}>\sqrt{x_{s,i}^2+y_{s,i}^2},\\ -50.0,\ \text{if}\sqrt{x_{s,i+1}^2+y_{s,i+1}^2}\leqslant\sqrt{x_{s,i}^2+y_{s,i}^2},\\ -100.0,\ \text{if sheep lose the game};\end{cases}\tag{5.21}$$

其中 ψ_s 和 ψ_d 分别为羊和犬的位移夹角。

由回报函数的设计可知，羊只有在满足逐步远离起点、逐步扩大与犬之间的位移夹角并赢得博弈胜利才能获得最高回报。羊在逃逸过程中如果做出反向折回 (即由于速度方向控制不当使得距离起点的距离缩小) 的动作将受到-50 的惩罚，从而羊经训练后能保持与起点的距离始终保持增大的姿态，同时为了羊最大化博弈还要学习如何设计策略 π 使得羊成功完成逃逸任务。

本文设计最大训练回合数为 500，并设置逃逸半径 $R=100$，羊的速度 $v=1$，犬的线速度 $V=\frac{50}{\pi}$，在每一训练回合开始时随机生成犬的初始状态 s_0，羊采取动作 a_0 并更新羊的位置坐标 (x_s,y_s)，随后依据 Markov 决策方程(5.10)更新犬在每一时刻下的状态 s_i，递归式地完成这一过程直到达到终端状态 Γ^1 或 Γ^2，该回合结束。在每一步环境状态 s 更新后，按照式(5.20)更新 Actor 和 Critic 的参数。在本文中，羊可采取的动作 a 为将速度夹角 θ_s 调整至 $[-\pi,\pi]$ 内的任意角度，训练完成后绘制总回报、总逃逸步数随训练回合数变化的曲线如图10和11所示。由图像可知，羊的逃逸总回报在大约训练 100 回合后逐步趋向于收敛，而羊逃逸消耗的总步数在大约训练 150 回合后逐步趋向于收敛，表明羊正在 DDPG 的引导下逐步学习如何在最大化回报的过程中形成正确的逃逸策略。

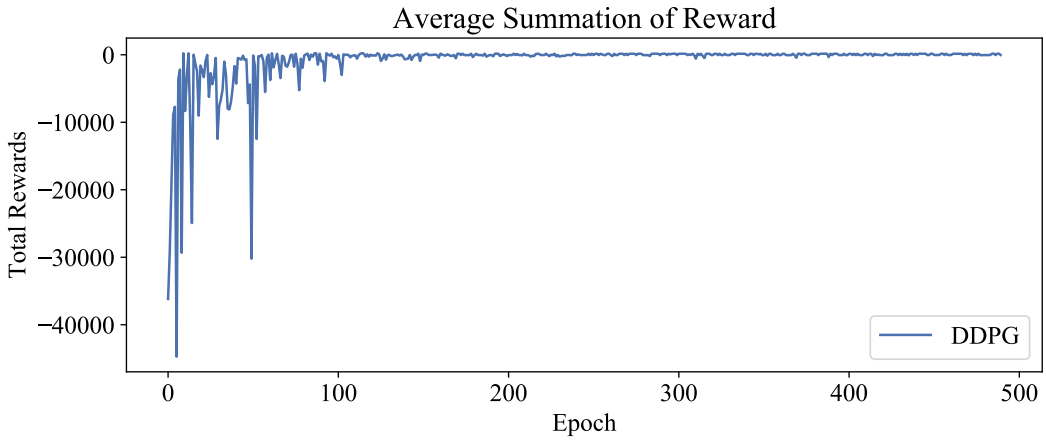


图 10: 经 DDPG 训练后羊的逃逸总回报变化曲线

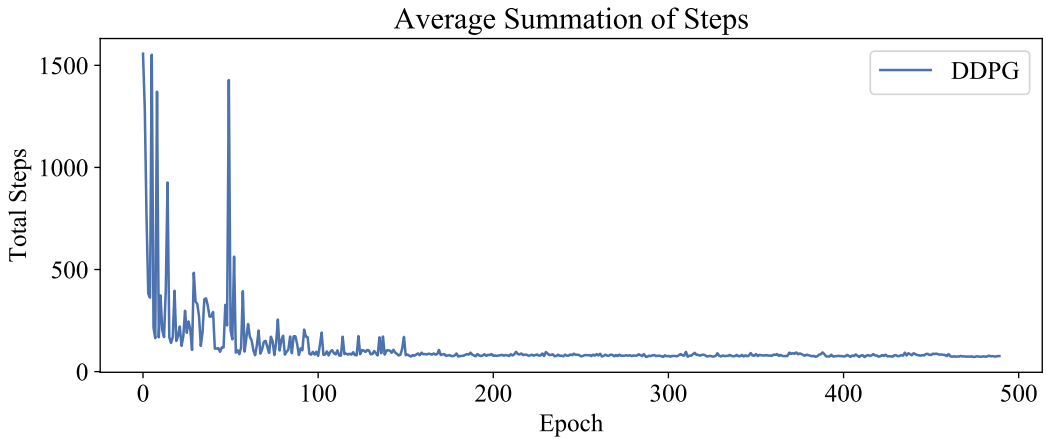
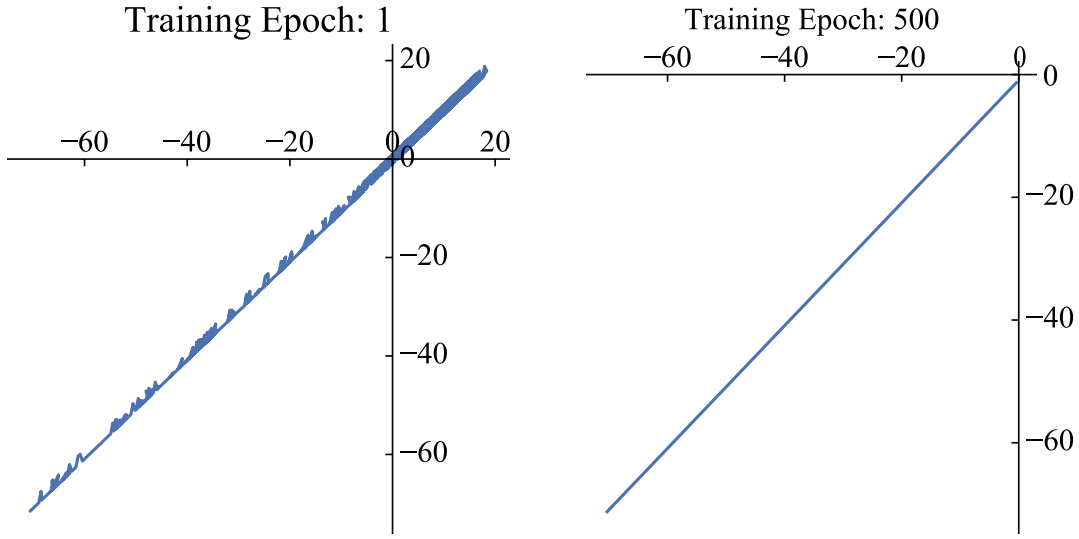


图 11: 经 DDPG 训练后羊的逃逸总步数变化曲线

图12展示了羊从原点 $(0,0)$ 出发，在第 1 次和第 500 次训练时的逃逸轨迹。其中第 1 次羊使用了 1538 步完成逃逸动作，第 500 次羊仅通过 168 步即完成逃逸。由于确定性策略 Actor-Critic 网络每进行一步就更新一次参数，且采用经验回放技术稳定逃逸的动作生成过程，在第 1 次博弈时羊即找到了沿直线规划逃逸路径的贪婪策略 π，但仍可以看出在第 1 次训练时羊的轨迹相对第 500 次具有明显的随机性抖动，完成逃逸消耗的时间步长也更多，上述结果进一步说明了本文所提算法的有效性。



(a) (b)

图 12: 第 1 次和第 500 次训练时羊的逃逸轨迹

5.4 问题四

5.4.1 模型准备

在文献 [6] 中，作者提出了用强化学习算法在测试场景下的平均得分和平均预测 Q 值作为评价指标，用来衡量该算法的学习性能。本文在文献 [6] 的基础上采用熵权法对原方法进行改进，提出了基于平均总回报和收敛步长的羊犬博弈综合评价模型。

5.4.2 基于熵权法的羊犬博弈综合评价模型

在羊犬博弈问题中，一次博弈回合中羊的博弈总回报越高，表明羊在满足约束条件下其逃逸的总体目标达成效果越好，而在一次博弈中羊的逃逸总步长越少，表明羊的逃逸时长越短，其达成躲避犬的围捕的目标越高效。在强化学习算法的实际训练过程中，有些算法在某些场景下由于神经网络的参数学习在每一单步完成后即进行，其总回报曲线往往会发生巨大抖动，表明其学习过程并不稳定，严重时可能会造成模型不收敛。基于上述原因，本文考虑平均总回报、平均收敛步长、总回报曲线标准差和收敛步长曲线标准差四类指标对模型进行综合评价。

由于采用多组实验数据可以更好地降低评价模型的方差，因此不妨随机生成 n_{tra} 组训练初始环境状态，令若干机器学习算法在每一组环境状态下进行训练。假设第 i_a 个机器学习算法 A 在第 i_t 组初始环境训练完成后的总回报变化曲线、收敛步长变化曲线分别为 $C^{(1)}_{i_a,i_t}, C^{(2)}_{i_a,i_t}$，计算训练完成 Ep 个回合后的平均总回报、平均收敛步长、总回报曲线标准差和收敛步长曲线的标准差，将上述四类指标设置记为 $d_k, k=1,2,3,4$。考虑 Ep 个训练回合后的变化曲线是由于在开始阶段算法普遍不能收敛于较优策略，因此都具有较强的抖动，可以将该部分抖动视为噪声加以删除。四类指标的权重按照熵权法予以确定。

考虑指标 $d_k, k=1,\ldots,4$，首先采用 Z-score 变换对原始数据进行归一化，使得变换后该指标的均值为 0，标准差为 1，Z-score 变换的公式如下式所示：

$$d_k^* = \frac{d_k - \mu_{d_k}}{\sigma_{d_k}}, \tag{5.22}$$

其中 μ_{d_k} 为第 k 个指标的均值，$\sigma_{\mu_{d_k}}$ 为第 k 个指标的标准差。

计算该指标提供的信息熵：

$$e_k = -K \sum_{i=1}^{m} d_{i,k}^* \ln d_{i,k}^*, \tag{5.23}$$

其中 m 为第 k 个指标包含的样本个数，$K = \ln m$。

随后可计算第 k 个指标的权重 λ_k：

$$\lambda_k = \begin{cases} \dfrac{1-e_k}{\sum_{j=1}^{4} 1-e_j}, & k=1,2; \\ -\dfrac{1-e_k}{\sum_{j=1}^{4} 1-e_j}, & k=3,4. \end{cases} \tag{5.24}$$

对指标 3 和指标 4 取负值的原因是算法在此两类指标下的得分越高，表明该算法的稳定性越差。

5.4.3 模型求解

本文随机生成 $n_{tra}=5$ 组训练环境状态，选取最大训练回合数为 500，设置 $Ep=100$，即从第 100 个回合以后使用熵权法对问题三中提出的算法进行测试，由公式(5.22)、(5.23)、(5.24)计算可得 4 类评价指标的权重如表1所示：

表 1: 四类评价指标的权重

d_1	d_2	d_3	d_4
0.295454	0.213436	-0.227366	-0.263743

由表1最终由熵权法求得该算法的得分为-0.0262930。

5.5 问题五

5.5.1 基于近端策略优化的羊逃逸路径规划模型

近端策略优化 (PPO)[7] 算法是一类常用于解决路径规划问题的算法，该方法主要架构于 Actor-Critic 网络，通过构建近似回报函数对策略进行更新。假设羊采取的逃逸控制策略为 $\pi=u_s$，则 PPO 的优化目标函数为：

$$L(\theta)=\mathbb{E}[\min\frac{\pi(s,a)}{\pi_{old}(s,a)}\hat{A},\mathrm{clip}(\frac{\pi(s,a)}{\pi_{old}(s,a)},1-\epsilon,1+\epsilon)\hat{A}] \tag{5.25}$$

其中 ϵ 为超参数，$\pi(s,a)$ 为优化后的新策略，$\pi_{old}(s,a)$ 为优化前的旧策略，$\hat{A}$ 为策略优势值，用于表示新策略相对于旧策略的优势。由于该模型同样与 DDPG 同属于基于策略梯度的强化学习算法，且均采用 Actor-Critic 网络架构，训练方式与 DDPG 类似，因此其训练过程不再赘述。

5.5.2 模型求解与定量评价

与问题三采用同样的初始环境训练 PPO 算法，其训练后羊的总回报变化曲线和收敛步数变化曲线如图13和14所示。由图像可知，PPO 算法的总回报曲线、收敛步长曲线相对于 DDPG 算法更加平滑，表明 PPO 算法相对于 DDPG 算法具有更强的稳定性，且由图像还可知 PPO 算法的收敛要早比 DDPG 算法更早，大约训练至 80 回合 PPO

算法即达到平稳收敛态。

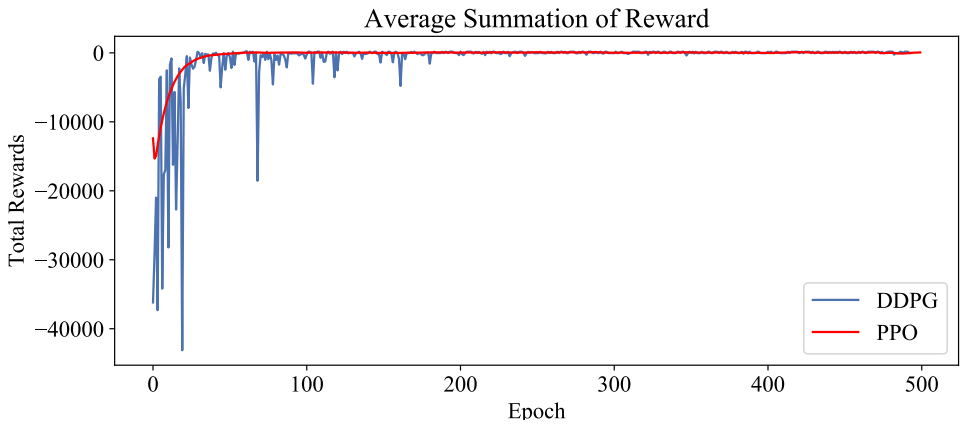


图 13: 经 DDPG 和 PPO 训练后羊的逃逸总回报变化曲线对比

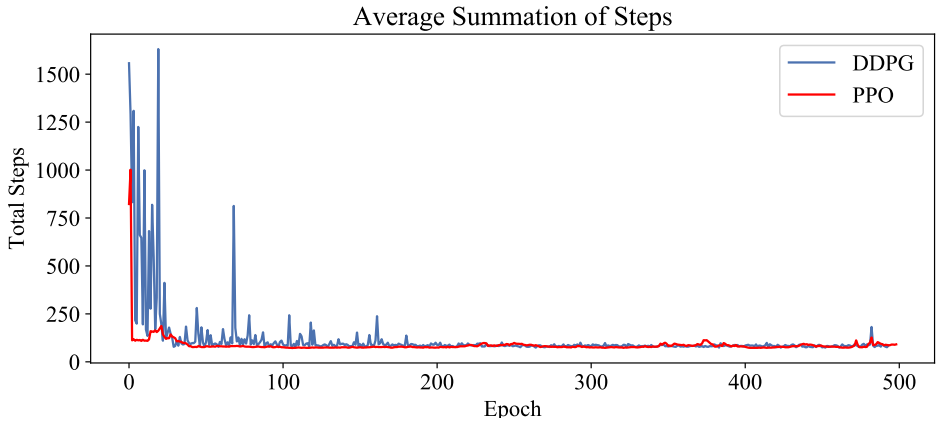


图 14: 经 DDPG 和 PPO 训练后羊的逃逸收敛步长变化曲线对比

采用问题四中提出的熵权法对 PPO 算法进行评价，PPO 算法的最终得分为 0.0097365，优于 DDPG 算法在羊犬博弈问题中的得分，这也与来自图像的直观评价相一致，进一步表明了问题四所提算法的稳定性与鲁棒性。

6 模型评价与改进

6.1 模型的优点

1. 基于随机微分对策构建羊犬博弈运动学模型，并采用动态规划和马尔可夫决策求解最优对策，降低了原始微分对策问题的求解难度。
2. 采用了基于强化学习的方法对羊进行训练，使得羊能在犬的围捕下掌握逃逸的最优策略，从而完成逃逸路径的规划，具有较强的泛化能力。
3. 在现有机器学习评价指标下基于熵权法提出了一种新的强化学习评价体系，实验结果表明该评价方法具有较好的鲁棒性。

6.2 模型的改进

1. 在使用动态规划方法求解基于随机微分对策问题时，还可以考虑将原始问题转化为线性系统的最优控制问题，通过求解 Riccati 方程的解来解决。
2. 除熵权法以外，还可以采用 TOPSIS 法、层次分析法、数据包络分析法等构造完善的强化学习算法评价体系。
3. 可以使用 DDPG、PPO 的改进算法进一步提高模型性能，比如 Twin Delayed DDPG(TD3)、PPO2、TRPO 等。
4. 可以将羊犬博弈问题推广为多智能体路径规划问题，或者将本文的模型转化为基于对抗学习的路径生成问题，进一步提高模型的现实意义。

7 参考文献

[1] Pachter M,Garcia E,Casbeer D W.Differential Game of Guarding a Target[J].Journal of Guidance Control and Dynamics,2017,40(11):1-8.

[2] 刘坤, 郑晓帅, 林业茗, 韩乐, 夏元清. 基于微分博弈的追逃问题最优策略设计 [J/OL]. 自动化学报:1-14[2021-07-29].https://doi.org/10.16383/j.aas.c200979.

[3] Venkatesan R H, Sinha N K.The Target Guarding Problem Revisited: Some Interesting Revelations[J].Ifac Proceedings Volumes,2014,47(3):1556-1561.

[4] Nisio M.Stochastic Differential Games[M].Springer Japan,2015.

[5] Lillicrap T P , Hunt J J , Pritzel A , et al. Continuous control with deep reinforcement learning[J]. Computer ence, 2015.

[6] Volodymyr M , Koray K , David S , et al. Human-level control through deep reinforcement learning[J]. Nature, 2019, 518(7540):529-33.

[7] Schulman J , Wolski F , Dhariwal P , et al. Proximal Policy Optimization Algorithms[J]. 2017.

8 附录

8.1 附录 A——Gym 平台下搭建羊犬博弈环境的 Python 源程序

```
import math
import gym
from gym import spaces, logger
from gym.utils import seeding
import numpy as np
import copy
# np.random.seed(0)

class MyEnv(gym.Env):
metadata = {
    'render.modes': ['human', 'rgb_array'],
    'video.frames_per_second': 50
}
def __init__(self):
    self.x_y_threshold = 100.0
    self.dog_rho = 100
    self.dog_theta = np.random.uniform(low=-math.pi,high = math.pi)
    # self.sheep_x = 0
    # self.sheep_y = 0
    # self.sheep_rho = 0
    # self.sheep_theta = 0
    self.dog_omega = 0.5/math.pi
    self.sheep_v =  1
    self.action_space = spaces.Box(-math.pi, math.pi, shape=(1,), dtype=np.float32)
    self.observation_space = ...
spaces.Box(low=np.array([-math.pi,-self.x_y_threshold,-self.x_y_threshold]), ...
high=np.array([math.pi, self.x_y_threshold,self.x_y_threshold]), shape=(3,), dtype=np.float64)
    self.seed()
    self.viewer  = None
    self.state   = None
    # self.n_features = self.observation_space.shape[0]
    # self.n_actions = self.action_space.shape[0]

def seed(self,  seed=None):
    self.np_random, seed = seeding.np_random(seed)
    return [seed]

def step(self,  action):
    err_msg = "%r (%s) invalid" % (action, type(action))
    assert self.action_space.contains(action), err_msg
    dog_theta, sheep_x, sheep_y = self.state[0], self.state[1], self.state[2]
    sheep_v_theta = action
    sheep_rho = np.sqrt(sheep_x**2+sheep_y**2)
    if  sheep_x > 0 :
        sheep_theta = np.arctan(sheep_y/sheep_x)
    elif  sheep_x == 0 :
        if  sheep_y > 0:
            sheep_theta = math.pi/2
        elif  sheep_y ==0 :
            sheep_theta = 0
        else  :
```

```
            sheep_theta = -math.pi/2
        else :
            sheep_theta = np.arctan(sheep_y/sheep_x)+math.pi
        sheep_x_prev = copy.deepcopy(sheep_x)
        sheep_y_prev = copy.deepcopy(sheep_y)
        sheep_rho_prev = copy.deepcopy(sheep_rho)
        sheep_theta_prev = copy.deepcopy(sheep_theta)
        # Update the coordinates of sheep
        sheep_x = sheep_x + self.sheep_v*np.cos(sheep_v_theta)
        sheep_y = sheep_x + self.sheep_v*np.sin(sheep_v_theta)
        sheep_rho = np.sqrt(sheep_x**2+sheep_y**2)
        if  sheep_x > 0 :
            sheep_theta = np.arctan(sheep_y/sheep_x)
        elif  sheep_x == 0 :
            if  sheep_y > 0:
                sheep_theta = math.pi/2
            elif  sheep_y ==0 :
                sheep_theta = 0
            else :
                sheep_theta = -math.pi/2
        else :
            sheep_theta = np.arctan(sheep_y/sheep_x)+math.pi
        # Update the coordinates of dog
        dog_theta_prev = copy.deepcopy(dog_theta)
        if  dog_theta < sheep_theta :
            dog_theta = dog_theta + self.dog_omega
        else :
            dog_theta = dog_theta - self.dog_omega

        self.state  = np.array([dog_theta, sheep_x, sheep_y]) # state

        done = bool(
            sheep_rho**2 ≥ self.x_y_threshold**2
        )

        if  not done:
            if  sheep_rho > sheep_rho_prev :
                if  abs(sheep_theta - dog_theta_prev) > abs(sheep_theta_prev - dog_theta_prev):
                    reward = 1.0
                else :
                    reward = 0.5
            else :
                reward = -50.0  # -50

        else:
            if   abs(sheep_theta_prev-dog_theta_prev)>0.25/math.pi :
                reward = 100.0
            else :
                reward = -100.0

        return self.state,   reward, done, {}

    def reset(self):
        self.state  = np.array([np.random.uniform(low=-math.pi,high = math.pi), 0, 0])
        return self.state

    def render(self,  mode='human'):
        return None
    def close(self):
```

```
109         return None
110
```

8.2 附录 B——基于 Tensorflow 搭建 DDPG 网络的 Python 源程序

```
1   # import tensorflow as tf
2   import tensorflow.compat.v1 as tf
3   tf.disable_v2_behavior()
4   import numpy as np
5   import pandas as pd
6   import matplotlib.pyplot as plt
7   import gym
8   np.random.seed(1)
9   tf.set_random_seed(1)
10  ######  hyper parameters  ######
11  MAX_EPISODES = 500
12  MAX_EP_STEPS = 2001
13  LR_A = 0.001    # learning rate for actor
14  LR_C = 0.001    # learning rate for critic
15  GAMMA = 0.9     # reward discount
16  REPLACEMENT = [
17      dict(name='soft', tau=0.01),
18      dict(name='hard', rep_iter_a=600, rep_iter_c=500)
19  ][1]            # you can try different target replacement strategies
20  MEMORY_CAPACITY = 10000
21  BATCH_SIZE = 32
22  RENDER = False
23  OUTPUT_GRAPH = True
24  # ENV_NAME = 'Pendulum-v0'
25  ENV_NAME = 'MyEnv-v0'
26  ######  Actor  ######
27  class Actor(object):
28      def __init__(self, sess, action_dim, action_bound, learning_rate, replacement):
29          self.sess   = sess
30          self.a_dim = action_dim
31          self.action_bound = action_bound
32          self.lr    = learning_rate
33          self.replacement = replacement
34          self.t_replace_counter = 0
35          with tf.variable_scope('Actor'):
36              # input s, output a
37              self.a  = self._build_net(S, scope='eval_net', trainable=True)
38              # input s_, output a, get a_ for critic
39              self.a_  = self._build_net(S_, scope='target_net', trainable=False)
40          self.e_params = tf.get_collection(tf.GraphKeys.GLOBAL_VARIABLES, ...
    scope='Actor/eval_net')
41          self.t_params = tf.get_collection(tf.GraphKeys.GLOBAL_VARIABLES, ...
    scope='Actor/target_net')
42          if self.replacement['name'] == 'hard':
43              self.t_replace_counter = 0
44              self.hard_replace = [tf.assign(t, e) for t, e in zip(self.t_params, self.e_params)]
45          else:
46              self.soft_replace = [tf.assign(t, (1 - self.replacement['tau']) * t + ...
    self.replacement['tau'] * e)
47                                   for t, e in zip(self.t_params, self.e_params)]
```

```
48      def __build_net(self, s, scope, trainable):
49          with tf.variable_scope(scope):
50              init_w = tf.random_normal_initializer(0., 0.3)
51              init_b = tf.constant_initializer(0.1)
52              net = tf.layers.dense(s, 30, activation=tf.nn.relu,
53                                    kernel_initializer=init_w, bias_initializer=init_b, name='l1',
54                                    trainable=trainable)
55              with tf.variable_scope('a'):
56                  actions = tf.layers.dense(net, self.a_dim, activation=tf.nn.tanh, ...
    kernel_initializer=init_w,
57                                            bias_initializer=init_b, name='a', trainable=trainable)
58                  scaled_a = tf.multiply(actions, self.action_bound, name='scaled_a')  # Scale ...
    output to -action_bound to action_bound
59          return scaled_a
60      def learn(self, s):    # batch update
61          self.sess.run(self.train_op, feed_dict={S: s})
62
63          if self.replacement['name'] == 'soft':
64              self.sess.run(self.soft_replace)
65          else:
66              if self.t_replace_counter % self.replacement['rep_iter_a'] == 0:
67                  self.sess.run(self.hard_replace)
68              self.t_replace_counter += 1
69      def choose_action(self, s):
70          s = s[np.newaxis, :]    # single state
71          return self.sess.run(self.a, feed_dict={S: s})[0]  # single action
72      def add_grad_to_graph(self, a_grads):
73          with tf.variable_scope('policy_grads'):
74              # ys = policy;
75              # xs = policy's parameters;
76              # self.a_grads = the gradients of the policy to get more Q
77              # tf.gradients will calculate dys/dxs with a initial gradients for ys, so this is dq/da * ...
    da/dparams
78              self.policy_grads = tf.gradients(ys=self.a, xs=self.e_params, grad_ys=a_grads)
79          with tf.variable_scope('A_train'):
80              opt = tf.train.AdamOptimizer(-self.lr)  # (- learning rate) for ascent policy
81              self.train_op = opt.apply_gradients(zip(self.policy_grads, self.e_params))
82  ######  Critic  ######
83  class Critic(object):
84      def __init__(self, sess, state_dim, action_dim, learning_rate, gamma, replacement, a, a_):
85          self.sess  = sess
86          self.s_dim = state_dim
87          self.a_dim = action_dim
88          self.lr   = learning_rate
89          self.gamma = gamma
90          self.replacement = replacement
91          with tf.variable_scope('Critic'):
92              # Input (s, a), output q
93              self.a  = a
94              self.q  = self._build_net(S, self.a, 'eval_net', trainable=True)
95
96              # Input (s_, a_), output q_ for q_target
97              self.q_ = self._build_net(S_, a_, 'target_net', trainable=False)    # target_q is based ...
    on a_ from Actor's target_net
98
99              self.e_params = tf.get_collection(tf.GraphKeys.GLOBAL_VARIABLES, ...
    scope='Critic/eval_net')
100             self.t_params = tf.get_collection(tf.GraphKeys.GLOBAL_VARIABLES, ...
    scope='Critic/target_net')
```

```
        with tf.variable_scope('target_q'):
            self.target_q = R + self.gamma * self.q_
        with tf.variable_scope('TD_error'):
            self.loss   = tf.reduce_mean(tf.squared_difference(self.target_q, self.q))
        with tf.variable_scope('C_train'):
            self.train_op = tf.train.AdamOptimizer(self.lr).minimize(self.loss)
        with tf.variable_scope('a_grad'):
            self.a_grads = tf.gradients(self.q,  a)[0]    # tensor of gradients of each sample (None, ...
    a_dim)
        if  self.replacement['name'] == 'hard':
            self.t_replace_counter = 0
            self.hard_replacement = [tf.assign(t, e) for t,  e in zip(self.t_params, self.e_params)]
        else:
            self.soft_replacement = [tf.assign(t,  (1 - self.replacement['tau'])  * t  + ...
    self.replacement['tau'] * e)
                                     for t,  e in zip(self.t_params, self.e_params)]
    def _build_net(self, s, a, scope, trainable):
        with tf.variable_scope(scope):
            init_w = tf.random_normal_initializer(0., 0.1)
            init_b = tf.constant_initializer(0.1)
            with tf.variable_scope('l1'):
                n_l1 = 30
                w1_s = tf.get_variable('w1_s', [self.s_dim, n_l1], initializer=init_w, ...
    trainable=trainable)
                w1_a = tf.get_variable('w1_a', [self.a_dim, n_l1], initializer=init_w, ...
    trainable=trainable)
                b1 = tf.get_variable('b1', [1,  n_l1], initializer=init_b,  trainable=trainable)
                net = tf.nn.relu(tf.matmul(s, w1_s) + tf.matmul(a, w1_a) + b1)
            with tf.variable_scope('q'):
                q = tf.layers.dense(net,  1,  kernel_initializer=init_w, bias_initializer=init_b, ...
    trainable=trainable)   # Q(s,a)
        return q
    def learn(self,   s,  a,  r,  s_):
        self.sess.run(self.train_op,   feed_dict={S: s, self.a: a,  R: r,  S_: s_})
        if  self.replacement['name'] == 'soft':
            self.sess.run(self.soft_replacement)
        else:
            if  self.t_replace_counter % self.replacement['rep_iter_c'] == 0:
                self.sess.run(self.hard_replacement)
            self.t_replace_counter += 1
######  Memory  ######
class Memory(object):
    def __init__(self, capacity, dims):
        self.capacity  = capacity
        self.data  = np.zeros((capacity, dims))
        self.pointer  = 0
    def store_transition(self,  s,  a,  r,  s_):
        transition = np.hstack((s, a, [r],  s_))
        index = self.pointer % self.capacity  # replace the old memory with new memory
        self.data[index,  :]  = transition
        self.pointer  += 1
    def sample(self, n):
        assert self.pointer  ≥ self.capacity,  'Memory has not been fulfilled'
        indices = np.random.choice(self.capacity, size=n)
        return self.data[indices,   :]
env = gym.make(ENV_NAME)
env = env.unwrapped
env.seed(1)
state_dim = env.observation_space.shape[0]
```

```
155 action__dim = env.action__space.shape[0]
156 action__bound = env.action__space.high
157 # all placeholder for tf
158 with tf.name__scope('S'):
159     S = tf.placeholder(tf.float32,  shape=[None, state__dim], name='s')
160 with tf.name__scope('R'):
161     R = tf.placeholder(tf.float32,  [None, 1],  name='r')
162 with tf.name__scope('S__'):
163     S__ = tf.placeholder(tf.float32, shape=[None, state__dim], name='s__')
164 sess = tf.Session()
165 # Create actor and critic.
166 # They are actually connected to each other, details can be seen in tensorboard or in this picture:
167 actor = Actor(sess, action__dim, action__bound, LR__A, REPLACEMENT)
168 critic  = Critic(sess,  state__dim, action__dim, LR__C, GAMMA, REPLACEMENT, actor.a, actor.a__)
169 actor.add__grad__to__graph(critic.a__grads)
170 sess.run(tf.global__variables__initializer())
171 M = Memory(MEMORY__CAPACITY, dims=2 * state__dim + action__dim + 1)
172 if  OUTPUT__GRAPH:
173     tf.summary.FileWriter("logs/", sess.graph)
174 var = 3  # control exploration
175 reward__list = []
176 step__list = []
177 state__list = []
178 for i  in range(MAX__EPISODES):
179     s  = env.reset()
180     ep__reward = 0
181     episode__state__list = []
182     for j  in range(MAX__EP__STEPS):
183         if  RENDER:
184             env.render()
185         # Add exploration noise
186         a = actor.choose__action(s)
187         a = np.clip(np.random.normal(a, var), -2, 2)    # add randomness to action selection for ...
    exploration
188         s__, r,  done, info = env.step(a)
189         episode__state__list.append([s__[0], s__[1][0], s__[2][0]])
190         M.store__transition(s, a, r / 10, s__)
191         if  M.pointer > MEMORY__CAPACITY:
192             var *= .9995    # decay the action randomness
193             b__M = M.sample(BATCH__SIZE)
194             b__s = b__M[:, :state__dim]
195             b__a = b__M[:, state__dim: state__dim + action__dim]
196             b__r = b__M[:, -state__dim - 1: -state__dim]
197             b__s__ = b__M[:, -state__dim:]
198             critic.learn(b__s,  b__a, b__r, b__s__)
199             actor.learn(b__s)
200         s  = s__
201         ep__reward += r
202         # if j  == MAX__EP__STEPS-1:
203         if  done == True :
204             print('Episode:',  i,  ' Reward: %i' % int(ep__reward), 'Explore: %.2f' % var, )
205             reward__list.append(ep__reward)
206             step__list.append(j)
207             state__list.append(episode__state__list)
208             if  ep__reward > -300:
209                 RENDER = True
210             break
211
```

8.3 附录 C——基于 Tensorflow 搭建 PPO 网络的 Python 源程序

```
# import tensorflow as tf
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
import numpy as np
import matplotlib.pyplot as plt
import gym
import pandas as pd

EP_MAX = 500
EP_LEN = 2001
GAMMA = 0.9
A_LR = 0.0001
C_LR = 0.0002
BATCH = 32
A_UPDATE_STEPS = 10
C_UPDATE_STEPS = 10
S_DIM, A_DIM = 3, 1
METHOD = [
dict(name='kl_pen', kl_target=0.01, lam=0.5),   # KL penalty
dict(name='clip', epsilon=0.2),                 # Clipped surrogate objective
][1]
class PPO(object):
def __init__(self):
    self.sess  = tf.Session()
    self.tfs   = tf.placeholder(tf.float32,[None,S_DIM],'state')
    #critic
    with tf.variable_scope('critic'):
        l1  = tf.layers.dense(self.tfs,100,tf.nn.relu)
        self.v  = tf.layers.dense(l1,1)  # state-value
        self.tfdc_r  = tf.placeholder(tf.float32,[None,1],'discounted_r')
        self.advantage = self.tfdc_r - self.v
        self.closs  = tf.reduce_mean(tf.square(self.advantage))
        self.ctrain_op = tf.train.AdamOptimizer(C_LR).minimize(self.closs)
    #actor
    pi,pi_params = self._build_anet('pi',trainable=True)
    oldpi,oldpi_params = self._build_anet('oldpi',trainable=False)
    with tf.variable_scope('sample_action'):
        self.sample_op = tf.squeeze(pi.sample(1),axis=0)
    with tf.variable_scope('update_oldpi'):
        self.update_oldpi_op = [oldp.assign(p) for p,oldp in zip(pi_params,oldpi_params)]
    self.tfa   = tf.placeholder(tf.float32,[None,A_DIM],'action')
    self.tfadv  = tf.placeholder(tf.float32,[None,1],'advantage')
    with tf.variable_scope('loss'):
        with tf.variable_scope('surrogate'):
            ratio  = pi.prob(self.tfa)/oldpi.prob(self.tfa)
            surr = ratio * self.tfadv
        if  METHOD['name'] == 'kl_pen':
            self.tflam  = tf.placeholder(tf.float32,None,'lambda')
            kl  = tf.distributions.kl_divergence(oldpi,pi)
            self.kl_mean = tf.reduce_mean(kl)
            self.aloss  = -tf.reduce_mean(surr-self.tflam * kl)
        else:
            self.aloss  = -tf.reduce_mean(tf.minimum(surr,
                                                  ...
```

```
    tf.clip_by_value(ratio,1.-METHOD['epsilon'],1.+METHOD['epsilon'])*self.tfadv)
55                                                  )
56          with tf.variable_scope('atrain'):
57              self.atrain_op = tf.train.AdamOptimizer(A_LR).minimize(self.aloss)
58          tf.summary.FileWriter("log/", self.sess.graph)
59
60          self.sess.run(tf.global_variables_initializer())
61  def update(self,s,a,r):
62      self.sess.run(self.update_oldpi_op)
63      adv = self.sess.run(self.advantage,{self.tfs:s,self.tfdc_r:r})     # 得到advantage value
64      # update actor
65      if  METHOD['name'] == 'kl_pen':
66          for  _ in range(A_UPDATE_STEPS):
67              _,kl = self.sess.run([self.atrain_op,self.kl_mean],
68                                   {self.tfs:s,self.tfa:a,self.tfadv:adv,self.tflam:METHOD['lam']})
69              if  kl  > 4 * METHOD['kl_target']:
70                  break
71              elif  kl  < METHOD['kl_target'] / 1.5:  # adaptive lambda, this is in OpenAI's paper
72                  METHOD['lam'] /= 2
73              elif  kl  > METHOD['kl_target'] * 1.5:
74                  METHOD['lam'] *= 2
75              METHOD['lam'] = np.clip(METHOD['lam'], 1e-4, 10)  # sometimes explode, this ...
    clipping is my solution
76      else:     # clipping method, find this is better (OpenAI's paper)
77          [self.sess.run(self.atrain_op,  {self.tfs:  s,  self.tfa:  a,  self.tfadv:  adv}) for _ in ...
    range(A_UPDATE_STEPS)]
78      # update critic
79      [self.sess.run(self.ctrain_op,  {self.tfs:  s,  self.tfdc_r:  r}) for  _ in range(C_UPDATE_STEPS)]
80  def _build_anet(self,name,trainable):
81      with tf.variable_scope(name):
82          l1  = tf.layers.dense(self.tfs,100,tf.nn.relu,trainable=trainable)
83          mu = 2 * tf.layers.dense(l1,A_DIM,tf.nn.tanh,trainable=trainable)
84          sigma = tf.layers.dense(l1,A_DIM,tf.nn.softplus,trainable=trainable)
85          norm_dist = tf.distributions.Normal(loc=mu,scale=sigma) # 一个正态分布
86      params = tf.get_collection(tf.GraphKeys.GLOBAL_VARIABLES,scope=name)
87      return norm_dist,params
88  def choose_action(self,s):
89      s  = s[np.newaxis,:]
90      a = self.sess.run(self.sample_op,{self.tfs:s})[0]
91      return np.clip(a,-np.pi,np.pi)
92  def get_v(self,s):
93      if  s.ndim < 2:s = s[np.newaxis,:]
94      return self.sess.run(self.v,{self.tfs:s})[0,0]
95  env = gym.make('MyEnv-v0').unwrapped
96  ppo = PPO()
97  all_ep_r = []
98  step_list = []
99
100 for ep in range(EP_MAX):
101 s = env.reset()
102 buffer_s, buffer_a, buffer_r = [], [],  []
103 ep_r = 0
104 done = False
105 for t  in range(EP_LEN):    # in one episode
106     if  done == False :
107         env.render()
108         a = ppo.choose_action(s) # choose action
109         s_, r,  done, _ = env.step(a)
110         buffer_s.append(s)
```

```
            buffer_a.append(a)
            buffer_r.append((r+8)/8)    # normalize reward, find to be useful
            s = s_
            ep_r += r
            # update ppo
            if (t+1) % BATCH == 0 or t == EP_LEN-1:
                v_s_ = ppo.get_v(s_)
                discounted_r = []
                for r in buffer_r[::-1]:
                    v_s_ = r + GAMMA * v_s_
                    discounted_r.append(v_s_) # v(s) = r + gamma * v(s+1)
                discounted_r.reverse()
                bs, ba, br = np.vstack(buffer_s), np.vstack(buffer_a), np.array(discounted_r)[:, ...
    np.newaxis]
                buffer_s, buffer_a, buffer_r = [], [],  []
                ppo.update(bs, ba, br)
        else:
            step_list.append(t)
            break
    if ep == 0: all_ep_r.append(ep_r)
    else: all_ep_r.append(all_ep_r[-1]*0.9 + ep_r*0.1)
    print(
        'Ep: %i' % ep,
        "|Ep_r: %i" % ep_r,
        ("|Lam: %.4f" % METHOD['lam']) if METHOD['name'] == 'kl_pen' else '',
    )

```

8.4 附录 D——熵权法求解 DDPG 和 PPO 模型得分的 Python 源程序

```
import pandas as pd
import numpy as np
# import data
score = pd.read_excel('score.xlsx')
# normalsization
score = (score-score.min())/(score.max()-score.min())
# compute information entropy
e_ = np.log(4)*score*np.log(score)
e_ = e_.sum()
e_ = e_/(e_.sum())
e_[-2:-1] = -e[-2:-1]
# compute score
score = e_*score.mean()
score = score.sum()

```