**基于DDPG算法的羊犬追逃博弈策略**

**摘 要**

**追逃微分博弈**问题，不仅在现代军事科技中发挥了越来越重要的作用，还能作为测试机器学习算法的重要案例。本文基于运动学及最优化理论知识，设计强化学习智能算法，求解羊—犬追逃博弈问题。

针对问题一，我们需要建立犬的**最优围堵策略**模型。首先，考虑犬的目标，犬希望成功围堵羊，而且犬-羊距离最小化时围堵成功可能性更大。因此，我们列出犬-羊在极坐标情形下的运动学方程，并表示出羊-犬的距离。最终，基于羊犬间的距离表达式，给出相对距离最小化的**目标规划**模型。

针对问题二，我们需要探究当犬以最优围堵策略进行围堵时，羊能成功逃逸的条件。首先依据运动学方程，求出使得羊成功逃脱的逃脱角度，求出羊离逃出点的距离和逃出时间。再通过犬距离逃出点的弧长和角速度求得犬的围堵时间。联立可得羊能成功逃逸的条件：羊的逃出时间小于犬的围堵时间。然后，我们对一犬对一羊的情形推广到多犬对一羊的情形。根据运动学方程，此时羊最优逃脱策略有两种：**直线**运动或**折线**运动。

针对问题三，我们需要通过机器学习的方法训练不具备运动学及最优化理论知识的羊，使之成功逃逸。本文基于强化学习**DDPG算法**，设计**羊的逃逸训练智能算法**。首先，结合问题一中犬的运动方程，设定初始环境状态，通过智能羊与环境的交互，得到逃脱经验并存储在记忆库。利用Python的gym库进行了200次训练，并在第40次训练后算法收敛，羊已经习得成功逃脱的决策。

针对问题四，我们需要针对问题三中的算法做出定量评价。首先，将算法中得出的**收敛步数**、**羊犬追逃轨迹、算法平均得分**作为评价指标。经过收敛步数后，羊已学会成功逃逸的决策，反映了算法的**实用性**。同时，若羊能以较短的步数达到收敛，则说明算法的**学习能力**强。至于羊犬追逃轨迹，根据收敛后的轨迹，可以初步分析羊每一步的决策，并判断该决策是否符合实际，从而评估算法的**科学性**。最终，依据羊的逃逸训练智能算法的测试结果进行评价。测试中，收敛步数为40，羊以很少的训练次数达成了训练目标，体现了算法的实用性与强学习能力。同时，依据羊—犬追逃轨迹，可以分析出羊的运动及决策都符合实际，体现了算法的科学性。而算法的平均得分，则通过随机选取状态对算法进行评价，对算法性能进行评价，能够反映算法的**有效性**。

针对问题五，我们将对问题三中羊犬追捕一对一的情况进行推广，提出多犬追捕下羊的智能逃逸算法。针对该种情形，我们对问题三算法的回报函数进行相应调整，结果在训练125次左右时开始收敛，可见算法的学习能力较强，且算法具有实用性。接着，为进一步测试并评价该算法，我们通过更改不同的两犬初始位置组合，对智能羊进行4次测试训练。4次训练中，羊逃脱所需步数分别为27、33、43和28，且从追逃轨迹中看出智能羊的运动及策略都是符合实际、可描述的，因此体现了算法的科学性。

最后，给出本文所建立模型的优缺点评价及模型的推广。

**关键词：**追逃微分博弈 目标规划 强化学习 DDPG算法

目 录

[1 问题提出 3](#_Toc87996602)

[1.1 问题背景 3](#_Toc87996603)

[1.2 问题重述 3](#_Toc87996604)

[2 问题分析 3](#_Toc87996605)

[2.1 对问题一的分析 3](#_Toc87996606)

[2.2 对问题二的分析 4](#_Toc87996607)

[2.3 对问题三的分析 4](#_Toc87996608)

[2.4 对问题四的分析 4](#_Toc87996609)

[2.5 对问题五的分析 4](#_Toc87996610)

[3 模型假设 5](#_Toc87996611)

[4 符号说明 5](#_Toc87996612)

[5 模型的建立与求解 6](#_Toc87996613)

[5.1 问题一的模型建立与求解 6](#_Toc87996614)

[5.1.1 犬的最优围堵策略 6](#_Toc87996615)

[5.2 问题二的模型建立与求解 7](#_Toc87996616)

[5.2.1 单个追捕者时羊的成功逃逸策略 7](#_Toc87996617)

[5.2.2 多个追捕者时羊的成功逃逸策略 8](#_Toc87996618)

[5.3 问题三的模型建立与求解 11](#_Toc87996619)

[5.3.1 马尔可夫决策过程（Markov Decision Processes, MDPs）错误!未找到引用源。与强化学习 11](#_Toc87996620)

[5.3.2 Deterministic Policy Gradient(DPG)与Deep Deterministic Policy Gradient (DDPG) 12](#_Toc87996621)

[5.3.3 羊的逃逸训练智能算法设计 12](#_Toc87996622)

[5.3.4 羊的逃逸训练智能算法测试 14](#_Toc87996623)

[5.4 问题四模型的建立 16](#_Toc87996624)

[5.4.1 算法评价模型 16](#_Toc87996625)

[5.4.2 羊的逃逸训练智能算法评价 17](#_Toc87996626)

[5.5 问题五的模型建立与求解 18](#_Toc87996627)

[6 模型的评价与推广 24](#_Toc87996628)

[6.1 模型的评价 24](#_Toc87996629)

[6.1.1 模型的优点 24](#_Toc87996630)

[6.1.2 模型的缺点 24](#_Toc87996631)

[6.2 模型推广 25](#_Toc87996632)

[7 参考文献 25](#_Toc87996633)

[8 附录 25](#_Toc87996634)

[8.1 羊的逃逸智能算法代码 25](#_Toc87996635)

[8.1.1 环境 25](#_Toc87996636)

[8.1.2 训练及测试 31](#_Toc87996637)

[8.1.3 DDPG算法 35](#_Toc87996638)

# 问题提出

## 问题背景

微分对策理论，是研究两个或多个决策者的控制作用同时作用于一个由微分方程描述的运动系统时，能够实现各自最优目标的决策过程的理论。[1]在经典的追逃博弈问题中，系统往往包含两类对立的博弈者，一类博弈者作为追捕者，而另一类博弈者作为逃跑者。追捕者和逃跑者在一个连续且动态变化的空间环境中活动，因此，关于双方连续动态冲突、对抗博弈的问题，可以运用微分对策进行数学描述。[2]

自然界中，速度更快的捕食者能够更加容易地捕食到速度较慢的猎物。但是，自然界中也有很多捕食者，能够通过团队合作的方式捕食到大型且速度更快的猎物。受此启发，随着现代军事科技的到来，追逃微分博弈问题在军事作战、无人机对抗、航天器交会等问题中都扮演了重要角色。因此，追逃微分博弈问题逐渐成为研究热点。

## 问题重述

现存在一羊和犬之间的追逃微分博弈问题。羊作为逃跑者，在一半径为的圆形圈内以定常速率运动，其逃逸路径上的每一点与圆心的距离随时间单调不减。若羊能逃出圆形圈，则羊胜利。犬作为追捕者，以定常速率沿着圆形圈进行围堵，以防止羊逃跑成功，犬在任意时刻都可以选择沿圆周顺时针或逆时针方向围堵。

根据问题背景，我们需要解决以下问题：

1. 基于运动学理论知识，精确建立并求解犬的最优围堵策略模型；
2. 在犬以第一问中求解出的最优围堵策略围堵的前提下，精确建模并求解出羊能够成功逃逸的条件；
3. 假设问题二中羊能够成功逃逸的条件被满足，但羊不具备运动学理论知识及最优化决策的知识，只清楚自己的能力、限制和逃逸的目标。在此基础上，给出一种机器学习方法，训练羊使得羊能够成功逃逸；
4. 建立评价体系，定量评价问题三中给出的机器学习方法的学习能力；
5. 给出更多训练羊逃逸的机器学习方法，并评价。

# 问题分析

## 对问题一的分析

问题一要求我们建立并求解犬的最优围堵策略模型。在羊—犬的追逃博弈过程中，犬希望使得两者之间的相对距离最小化而羊试图使得相对距离最大化。因此，对于犬，我们将建立使得羊与犬间相对距离最小的目标规划模型。首先，用极坐标表示羊和犬的位置，从而表示出两者间的距离。利用两者间距离最短的目标规划，即可求出能使犬的最优围堵策略模型。

## 对问题二的分析

问题二中犬将以问题一中求出的最优策略进行围堵，我们需要建模求解羊能够成功逃逸的条件。首先，假设出羊在圆周上的逃脱点，从而计算出羊沿直线逃离的时间和犬沿圆周围堵的时间。通过比较两者所要花费的时间，可以给出羊能够成功逃逸的条件。接着，我们将问题推广为有多条犬进行围堵的情况。假设追捕犬之间没有交流与合作，将羊的逃离分为沿直线运动和沿曲线运动，通过对羊到逃离点及犬到逃离点时间的比较，我们给出羊能够成功逃逸的条件。

## 对问题三的分析

问题三中，羊具有逃跑的本能以及清楚自身逃跑的目标，同时具有学习的能力，但不具备分析最佳逃跑决策的能力，因此，最佳逃跑决策需要通过后天习得，即通过一种机器学习的方法，将羊训练的具备决策的能力。羊可以利用历史经验，在新一轮的追逃博弈中，实现逃脱的最优路径与决策。沿用问题一和问题二的设定，我们提出一种机器学习的方法。得益于强化学习的思路，首先我们将羊作为一个智能体，能够在环境中自主探索，在不同环境参数的条件下，通过羊与环境的交互，得到对应不同的回报值，经过多轮的训练后，羊可以根据回报值选择最优的逃跑决策。

## 对问题四的分析

问题四要求我们给出一套评价体系来评价问题三的机器学习方法。首先，由问题三的机器学习方法，我们可以得到羊训练收敛时的步数，即当经过该步数后，羊已经习得成功逃跑的能力，通过截取收敛时犬羊追逃时的轨迹图，我们可以初步分析羊的决策，判断机器学习训练出来的结果是否符合实际。接着，我们分别调节羊的速度、狼的速度和圈的半径，得到不同对应的收敛步数，通过收敛步数的变化情况，我们可以评估机器学习的稳健性。然后，通过命令羊执行一定的步数，记录羊所有的回报值并求平均以作为算法的平均得分，对算法的性能进行测试。由收敛步数、模型的稳健性和算法的平均得分，我们可以定量评价问题三算法的优劣。

## 对问题五的分析

问题五需要将犬羊追逃推广到更一般的情形，并作出相应的定量评价。我们将上述犬羊一对一的情形，推广到犬羊多对一的情况，保留问题一中对犬的设定和最佳决策，根据多犬对一羊的情况，对问题二羊的最优逃跑策略进行适当的改良和调整，经过问题三机器学习训练后得到的结果，利用问题四来定量评估多对一情况的算法优劣。

为更直观地呈现问题分析思路，绘制流程图如图 1所示。

|  |
| --- |
| 图 1 问题分析 |

# 模型假设

1. 假设羊与犬的视野不受限，在任意位置上都能观测到对方的位置。
2. 假设羊的逃逸及犬的追捕过程中均无障碍物阻挡。
3. 假设羊的目标只有逃逸成功，犬的目标只有成功围堵羊。

# 符号说明

表 1 符号说明

|  |  |  |
| --- | --- | --- |
| 序号 | 符号 | 意义 |
| 1 |  | 外围圆周半径，即犬的活动范围 |
| 2 |  | 犬的极坐标极角 |
| 3 |  | 羊的极坐标极角 |
| 4 |  | 某一时刻羊到圆心的距离。 |
| 5 |  | 羊、犬之间的距离 |
| 6 |  | 有限的状态state空间 |
| 7 |  | 有限的行为action空间 |
| 8 |  | 回报函数 |

# 模型的建立与求解

## 问题一的模型建立与求解

### 犬的最优围堵策略

在羊与犬的追逃博弈过程中，犬的目标是成功围堵羊。因此，犬需要预测羊的行动轨迹并移动。犬希望使得两者间的相对距离最小化而羊希望两者间的相对距离最大化。我们用极坐标表示位置，假设犬在半径为的圆周上的位置为，羊在圆周内的位置为，羊与犬位置示意图如图 2所示：

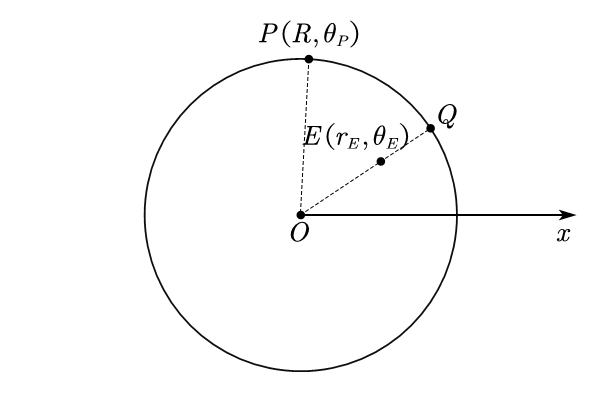


图 2 羊与犬位置示意图

其中，为与间的夹角，为与间的夹角，为某一时刻羊到圆心的距离。用向量表示速度，则有

 (1)

其中为位置矢量，为某点的极径，为径向单位矢量，为点处极角， 为横向单位矢量。

由于犬在圆周上以常速率运动，故其径向速度为0，因此犬的速度可表示为

 (2)

其中，为犬的速率，为与间的夹角。

羊的速度则可表示为

 (3)

其中，为与间的夹角，为某一时刻羊到圆心的距离。由题目得知羊逃逸路径上每一点与圆心的距离随时间单调不减，因此我们有

 (4)

犬的最优围堵策略，即花最短的时间用最短的路程追上羊。为达成此目标，需使得羊和犬之间的距离最小，即使得最小。由余弦定理，知可表示为

 (5)

综上，犬的最优围堵策略为

 (6)

当羊在约束（4）下调整逃跑角度，导致发生变化，由公式（6）可知，犬需要沿着劣弧所在方向上移动，才能抓到羊。否则，犬和羊的距离将不会缩小，与犬设定的围堵目标矛盾。

## 问题二的模型建立与求解

本问中，犬将以问题一中求出的最优策略进行围堵。考虑到实际生活中，常常会出现多个追捕者同时围堵一个逃跑者的情况，比如说罪犯的追捕过程中往往会有多个警察进行围堵，由此我们得到启发，我们将从题目中单个追捕者的情况开始讨论，进一步推广到多个追捕者的情形。

### 单个追捕者时羊的成功逃逸策略

首先，我们考虑只有两个局中人的情况：一只羊与一只犬。假设羊能够从圆周上的点处成功逃脱，作出示意图如下：

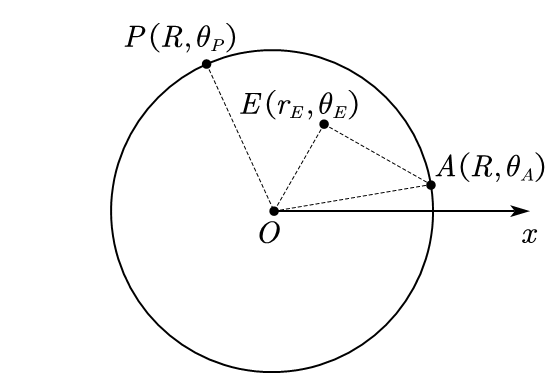


图 3 羊犬逃逸点示意图

假设当羊运动到点处后，羊选择以直线运动至处逃脱。由余弦定理，有

 (7)

称羊以直线运动逃离的时间为**最终逃离时间**，则有最终逃离时间为

 (8)

此时，犬以最优围堵策略接近羊，应是沿能够使得最小的方向，即沿劣弧前往处围堵羊。此时，犬的**最终围堵时间**为

 (9)

其中，为犬沿圆周以定常速率运动的角速度。

基于羊的最终逃离时间及犬的最终围堵时间，我们给出羊能够成功逃逸的条件：

 (10)

由公式（10）我们可以得到以下三种逃逸情形：

(i) 当羊的最终逃离时间小于犬的最终围堵时间时，羊先逃离圆形圈，逃逸成功；

(ii) 当羊的最终逃离时间大于犬的最终围堵时间时，犬能先于羊到达羊的逃出点，成功围堵，羊逃逸失败；

(iii) 当羊的最终逃离时间等于犬的最终围堵时间时，羊恰好在圆形圈上与犬相遇，刚好被犬抓捕，未能逃出圆形圈，故也逃逸失败。

因此，羊能够成功逃逸的条件为。当满足时，羊可以沿成功逃出圆形圈。

### 多个追捕者时羊的成功逃逸策略

对于有多个追捕者的情形，逃跑者通过调整逃跑方向，使得所有追捕者都无法及时到达既定位置，即逃跑者需要从任意两个追捕者中间的间隙中穿过，此时逃跑成功。因此，多个追捕者问题可以转化为双追捕者问题。

在此问题中，我们假设有两条犬与一只羊，两条犬之间相互独立，即犬之间不存在交流、合作或任务分配。羊犬间的追逃示意图如所示：

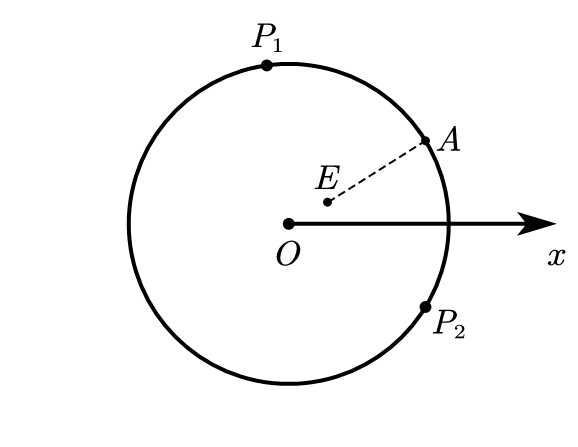


图 4 两犬一羊追逃示意图

图中与分别代表两条犬，为羊，而为羊的逃出点。对于羊来说，其最终逃跑可能是直线运动，也可能是折线运动。

**情形1：**当羊最终以**直线运动**从点逃出时，由5.2.1节中单个追捕者问题的讨论中可知，当有

 (11)

时，即羊到逃跑点的时间小于两犬到达逃跑点的时间，两犬之间存在间隙，因此羊可从间隙中成功逃脱。

而当不存在这样的逃跑点时，比起逃脱不了被捕结局的直线运动，羊会更倾向于以折线形式逃跑，说不定能换取一线生机，存在逃脱的可能性。

**情形2：**当羊逐渐接近圆周但发现可能不能成功逃脱时，即在满足约束（4）下，不存在逃脱方向，使得（11）成立。针对这种情况，我们对羊的最优逃脱策略进行以下调整：羊会选择以**折线运动**逃出。

逃脱思路是：

***Step1***羊当前不存在符合条件的逃脱方向，所以羊无论朝那个方向都是往两犬的**公共围捕区**逃跑，即不可逃脱点形成的集合SET，对，不可逃脱点需满足以下条件：在羊往点的方向直线逃跑时，至少存在两条犬都能围捕到羊。即满足以下公式：

 (12)

***Step2*** 当犬接收到此刻羊的逃跑策略时，自然会往公共围捕区方向追捕。此时羊会进行如下判断：集合Y是否为空。集合Y的定义为CIR与SET的差集，CIR为圆周上所有的点构成的集合。Y中的点即为情形1中的间隙。假如Y非空，则化归为情形1的逃跑策略；否则，转入Step1，直至Y非空为止。

为更直观呈现情形2下羊犬的追逃，假设此时羊无论朝那个方向都是往两犬的公共围捕区逃跑，如图所示：

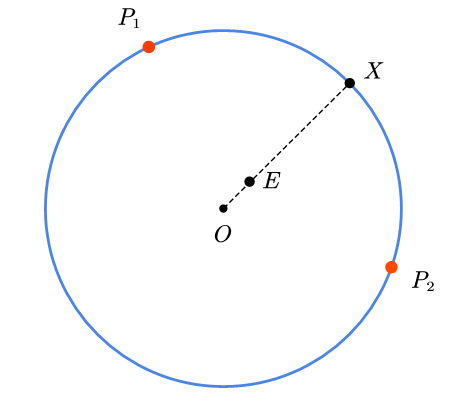


图 5 情形2下追逃示意图Ⅰ

其中蓝色线条为公共围捕区，与分别代表两条犬，为羊，而点为羊的当前的逃跑目标点。

现羊先往点方向逃跑，犬的移动方向及其公共围捕区的变化示意图如下图所示：

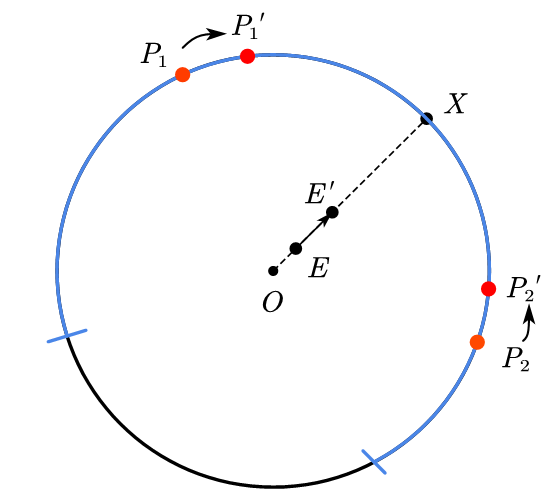


图 6 情形2下追逃示意图Ⅱ

从图 6可以看出，此时集合Y不为空，即图 6圆周上黑色弧线部分的点，因此羊会在满足约束条件（4）的情况下往间隙跑，即化归为情形1的逃跑策略。因此，羊可能的逃跑策略为：

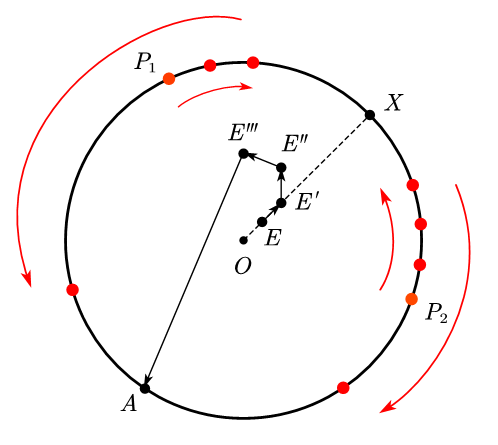


图 7 情形2下追逃示意图Ⅲ

其中，点为羊的最终逃出点，红色箭头为犬的运动方向，黑色箭头为羊的逃跑方向。

## 问题三的模型建立与求解

本问中，羊知道自己的能力、限制和目标，但它不具备运动学及最优化理论知识，我们需要通过机器学习的方法训练羊使之成功逃逸。

随着人工智能领域的飞速发展，在自主学习、自我优化方面有着巨大优势的深度强化学习也逐渐成为人们的研究热点。**强化学习**，是智能体（Agent）以“试错”的方式自主探索环境状态，并通过与环境进行交互获得回报的循环过程。[4]智能体的目标，是通过不断训练，获得最大化的长期回报。AlphaGo在围棋场上战胜人类顶尖选手正是强化学习的经典例子。本问中，羊不具备基于运动学的最优化理论知识，因此我们可以通过强化学习的机器学习方法，让羊在不断的尝试中更新自身的行为准则，从而一步步学会如何成功逃逸。

### 马尔可夫决策过程（Markov Decision Processes, MDPs）[3]与强化学习

马尔科夫决策过程，可以用于描述一个智能体(Agent)采取行动(Action)以改变自身的状态(State)，从而使得奖励(Reward)与环境(Environment)发生交互的循环过程。

在任意的马尔可夫决策过程中，智能体都存在一个确定的最优决策，强化学习的目标正是寻找给定的马尔可夫决策过程中的最优策略。[3]

马尔科夫决策过程可以由一个五元组表示。其中，表示有限的状态state空间，为有限的行为action空间，为状态转移函数，为回报函数，为折扣因子。[3]状态转移函数表示在给定当前状态和行为下，转移到下一状态的概率分布：

 (13)

其中，表示某个特定的状态，表示某个特定的行为动作，表示下一时刻可能的状态。

而回报函数表示智能体在采取某个行动后环境给予的奖励。因此，马尔可夫决策的过程中具有此特性：智能体下一时刻的状态和回报只取决于智能体当前时刻的行为和状态。

一般而言，强化学习问题是建立在马尔科夫决策过程之上的。强化学习框架流程图如图 8所示：

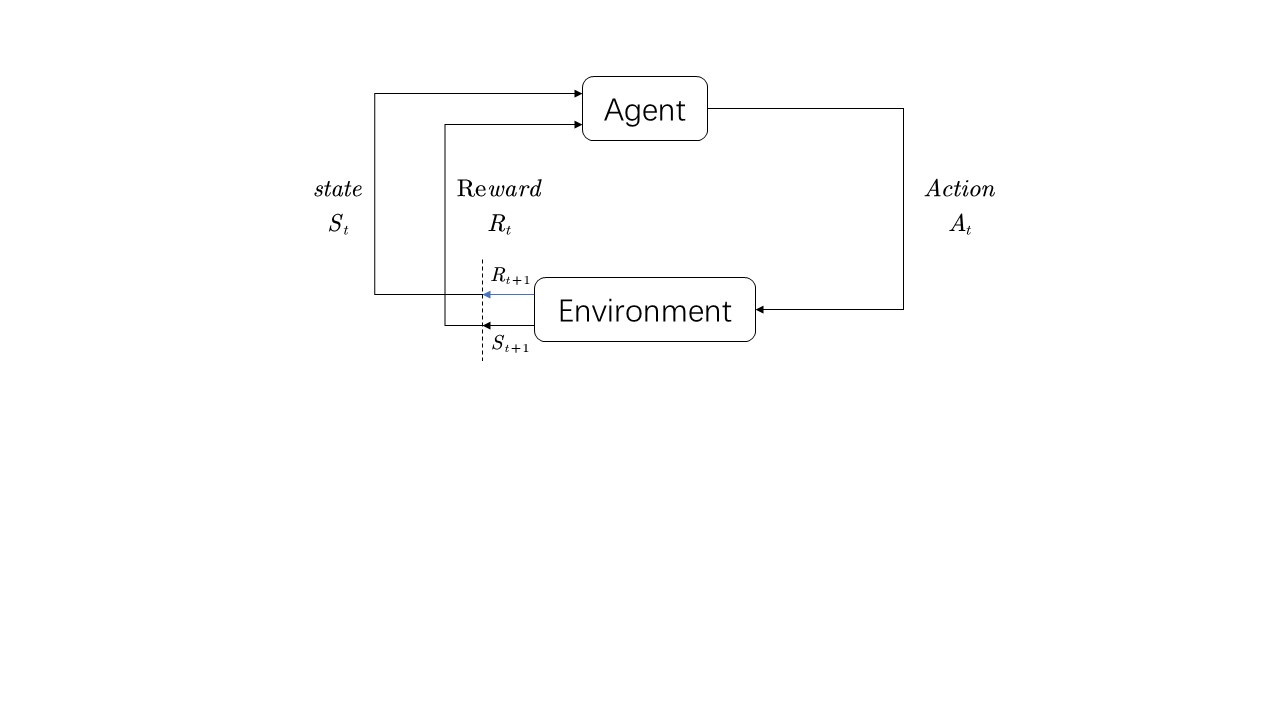


图 8 强化学习框架流程图

假设智能体处于状态时，采取了行动，根据状态转移函数，环境会转移到下一状态，同时环境会给智能体反馈一个奖励信号，即回报。[3]智能体不断循环上述过程，以期通过训练获得最大化的长期回报。

对于策略，满足马尔科夫假设：当智能体处于状态时，采取了行动的概率只与当前状态有关，和其他因素无关：



状态价值函数，可用于评估智能体的状态的价值，表示当智能体从状态开始然后执行策略时的价值[3]，即有

 (14)

其中，为最终时刻，为当前时刻，为时刻智能体得到的回报，为折扣因子。由于预测将来的回报，随着预测地越长远，对当下的影响会越来越小。所以，状态价值是从某个状态开始，直到终止状态时，所有回报的衰减和。

又由于智能体的回报与状态和行动都有关，而公式（14）没有考虑到采用的行动带来的价值影响，因此，定义动作价值函数，其表示智能体从状态做出行动并执行策略后得到的预期回报[3]。



若存在最优策略，则会使得智能体获得最大化的回报。因此最优策略下的动作价值函数可改写为Bellman最优方程如下：

 (15)

从Bellman最优方程可以看出，一个状态的价值由两部分决定：该状态的回报，以及后续衰减的状态价值之和。

### Deterministic Policy Gradient(DPG)与Deep Deterministic Policy Gradient (DDPG)

对于智能体(Agent)的行为(action)，往往可以简单分为离散的，或是连续的。而DDPG算法，是一针对连续行为的策略学习方法，由Deepmind于2016年提出[5]。该算法在DPG算法中融入了深度学习神经网络，采用卷积神经网络作为策略函数和行为值函数的模拟，即策略网络和网络，然后再使用深度学习的方法来训练上述神经网络。[5]

### 羊的逃逸训练智能算法设计

由图 2所示的羊犬追逃示意图可以看出，若将羊视为智能体，环境的状态和智能体的动作都是连续的。因此，我们基于DDPG算法[5]来定义状态空间及行为空间，并设计回报函数，从而使得羊通过学习训练后实现逃逸。

对于智能体羊E，其逃逸过程中希望与外围圆周越近越好，与犬的夹角越大越好，且离犬越远越好。通过算法获得犬P和羊E的实时位置后，我们可以以此为基础构造输入神经网络的状态量。[3]因此，对于图 2所示追逃情形，我们给出**状态空间**为

 (16)

其中，为与间的夹角，为与间的夹角，为某一时刻羊到圆心的距离。

智能体羊的**动作空间**，是其每一步可采取的动作的集合，会根据其当前所处的位置而变化。对于图 2所示追逃情形，羊的动作空间即为其可转过的角度，因此我们给出动作空间为

 (17)

示意图如下：

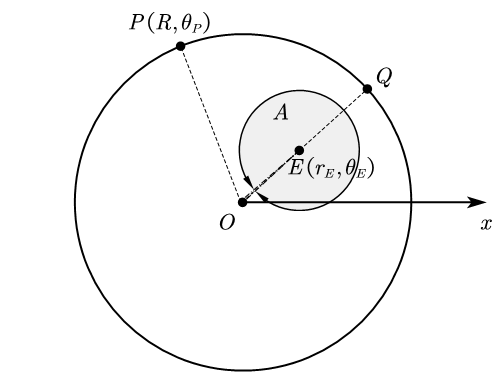


图 9 动作空间示意图

其中，犬在半径为的圆周上的位置为，羊在圆周内的位置为，灰色区域即为动作空间。

由于智能羊的目标是成功逃逸，所以对智能羊的每一次训练尝试，羊越接近圆周、与犬的夹角越大，获得的回报将会越大。因此回报函数设计为

 (18)

其中。

为更具体呈现算法，绘制流程图如图 10所示：

Step1 初始化：结合问题一中对犬围捕策略的运动学描述，我们设定犬的策略，并随机生成犬的位置，设定环境状态；

Step2 环境将state传递给智能羊，智能羊根据当前的state选择action，并与环境交互，即将action传递给环境；

Step3 环境根据action计算回报reward，通过reward来判断智能羊是否已经实现逃脱，如果已逃脱，则将此次游戏当作是成功的经验，存入记忆库中，如果未逃脱，则转入Step1和Step2，在环境与智能羊不断的交互中训练，直至智能羊成功逃脱为止；

Step4 判断记忆库是否已满，并判断是否已经达到最大游戏次数，若上述两个判断均符合，则算法结束；存在一个条件不满足，则重复上述流程，直至结束为止。

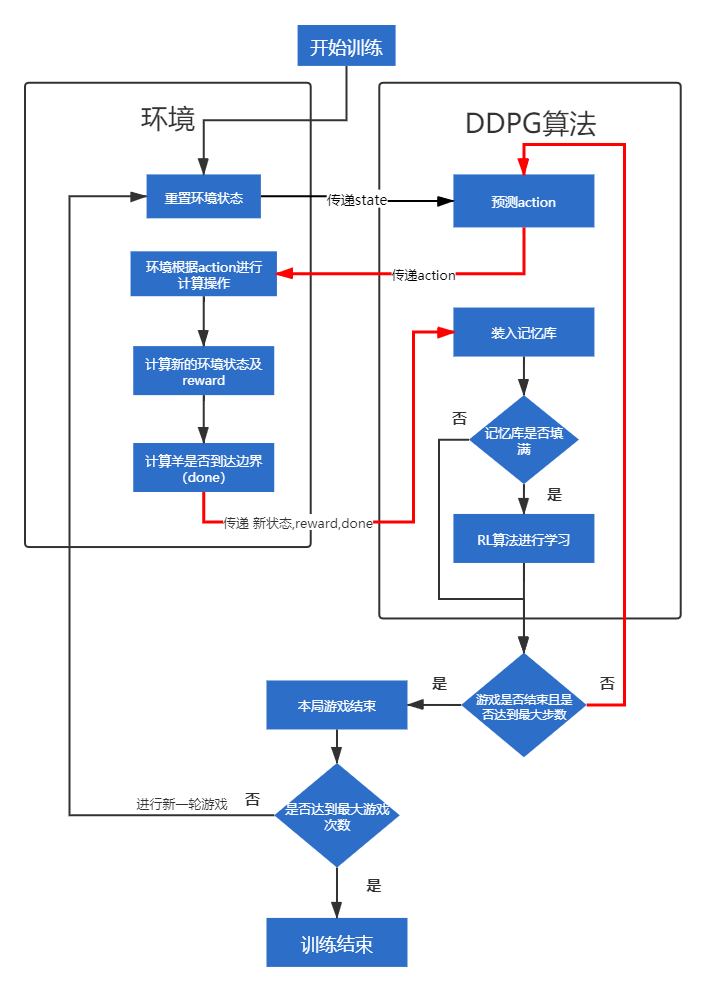


图 10 智能算法流程图

### 羊的逃逸训练智能算法测试

首先，设置实验相关参数如下：

表 2 实验相关参数

|  |  |
| --- | --- |
| 参数名 | 参数值 |
| 犬的初始位置(m, rad) | (100,) |
| 圆周半径(m) | 100 |
| 羊的初始位置(m, rad) | (0,) |
| 采样时间(s) | 0.1 |

通过Python语言进行编程，对智能羊进行200次训练，并记录每次训练后的累计回报，结果如所示。

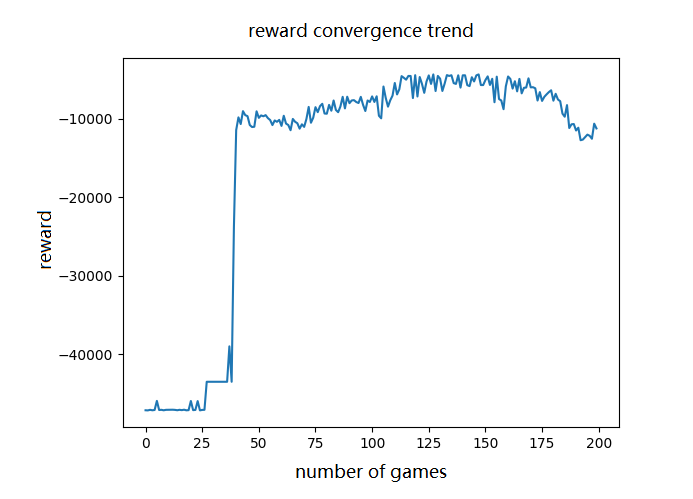
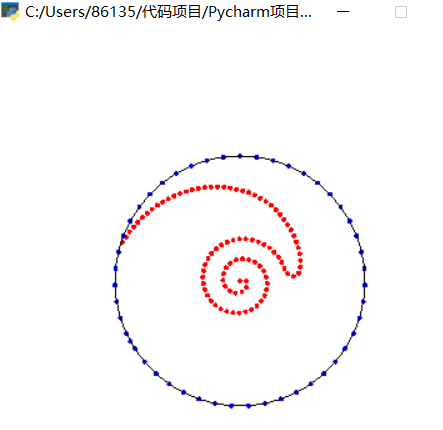


图 11 回报收敛趋势图

由图可知，在25-50次之间（大概在40次左右），回报出现了突变，突然上升到一个稳定值附近，这意味着，回报在大约训练40次左右时开始收敛。由于回报算法设计中设置了惩罚项，智能羊在逃逸失败后会受到惩罚，因此在训练175次左右之后，回报出现下降，这是由于过拟合导致。当算法收敛后，智能羊可以做出正确的决策，从而逃逸成功。训练成功后，羊的逃逸轨迹图如图 12所示：



犬初始位置

图 12 训练后羊犬逃逸轨迹图（红点为羊，蓝点为犬）

圆内的点为羊E，圆周上的点为犬P。由图6可以看出，经过训练后，羊E刚开始采取绕圈的策略，不断增大P和E之间的极角差。当极角差到达一定程度后，E突然调整策略，极角变化方向与原来相反，慢慢调整逃跑方向，迂回逃跑，最终实现逃脱。

其中，犬、羊位置的变化随时间的变化如以下三图所示（图7和图8分别是犬P和羊E的极角（单位：rad）随时间（单位：步）变化图像，图9是羊E的极径随时间变化图像）：

|  |  |
| --- | --- |
| 图 13 犬的追捕极角 | 图 14 羊的追捕极角 |

|  |
| --- |
| 图 15 羊的逃逸半径 |

结合图7、图8和图9分析，我们可以看出：0至20步长左右时，羊从初始位置出发，先沿顺时针方向运动，而此时犬相应地有两次转向，即从顺时针运动转至逆时针再转至顺时针；20步长到55步左右时，羊和犬都沿顺时针方向运动。而在这之后，羊突然转向，沿逆时针方向运动，犬在63步左右之后也开始沿逆时针方向运动。羊的突转恰好对应着图6中羊轨迹的突变。

## 问题四模型的建立

### 算法评价模型

本问中，我们需要给出一套评价体系来评价问题三的机器学习方法。我们将算法中得出的收敛步数、羊犬追逃轨迹作为指标，以对算法做出定量评价。

1. **收敛步数**

收敛步数，即为累计回报收敛时，智能羊的训练次数。若累计回报能达到收敛，则说明算法具有**实用性**。经过收敛步数后，羊已经具备成功逃逸的决策能力。若羊能以较短的步数达到收敛，则说明算法的**学习能力**强，使得智能羊能以较快的速度完成学习。同时，若对羊的速度、犬的速度和圆周半径进行调整，再通过算法得到不同的收敛步数，可以进一步评估算法的**稳健性**。

1. **羊-犬追逃轨迹**

根据收敛后羊、犬的追逃轨迹，可以初步分析羊每一步的决策，并判断该算法训练出来的智能羊做出的决策是否符合实际，从而评估算法的**科学性**。

1. **算法的平均得分**

羊进行一定的步数执行，以进行性能测试，记录羊所有reward并对其求平均。相比于用平均Q值来评价算法，平均得分可以更好地评价算法性能，同时很好地描述训练时性能的变化。因为每次测试时，平均得分都是通过随机选取状态进行评价。而平均Q值是对已选取的状态进行测试，经过一定时间后固定的状态性能会达到稳定，难以反映实际性能，例如：当出现没有出现的状态时，平均Q值就无法进行评价。[6]

### 羊的逃逸训练智能算法评价

现将5.4.1节中的算法评价模型应用至羊的逃逸训练智能算法模型中。

1. 图 11为本算法所得结果的累计回报收敛趋势图。由图可以看出，共进行200次训练，而累计回报在训练约40次时开始收敛，即收敛步数为40。可见，智能羊能以很少的训练次数完成训练目标，体现了算法的**实用性**与**强学习能力**。

2. 图 13、图 14 分别为收敛时羊、犬的极角轨迹变化图，图 12为羊-犬的追逃轨迹示意图。由图可知，0至20步长左右时，羊从初始位置出发，先沿顺时针方向运动。当羊在初始点左侧运动时，由于犬起始点与羊的起始点在同一条垂线上，因此犬也会顺时针运动。而当羊运动至初始点右侧时，犬为了缩小与羊的距离，改为逆时针运动，与图示轨迹相符。20步长到55步左右时，羊和犬都沿顺时针方向运动。而在这之后，因其离外围圆周较近，可以寻求突破口，因此羊突然转向，沿逆时针方向往外围圆周运动，犬在63步左右之后，为了更快追捕羊，也开始沿逆时针方向运动。羊的突转恰好对应着图6中羊轨迹的突变。可见，算法训练出来的智能羊的运动及策略都是符合实际的，体现了算法的**科学性**。

3. 图 16为算法的平均得分图，可以看出（1）在训练过程中，reward波动很大，这是合理的，因为羊在学习过程中不断尝试，其中某些选择不一定是一种好的决策，所以导致reward下降。reward的波动反映学习搜索的范围有足够的广度，即羊在大范围的状态中学习，而不是由于累积的经验，局限自己的选择。（2）总体上看，reward总体呈现上升趋势，说明羊学习是往有利方向学习的，而不是“越学越差”，反映算法的**有效性**。[6]

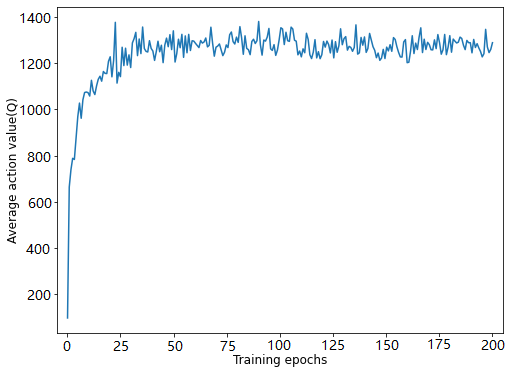


图 16 算法平均得分图

## 问题五的模型建立与求解

本问中，我们将对问题三中羊犬追捕一对一的情况进行推广，提出多犬追捕下羊的智能逃逸算法，并做出定量评价。

对于5.3.3节中的算法设计，考虑有两条犬进行追捕的情况，我们对回报函数作如下调整：

 (20)

其中，，为外围圆周半径，为羊的坐标极径，为羊的坐标极角，、分别为两犬的坐标极角。智能羊的目标是逃出外围圆周，且不被犬围堵。因此，羊离外围圆周越近、离两犬越远，回报将会越高。同时，若能让两犬尽可能靠近，使得犬的围捕区域变小甚至两犬相重合，对羊的逃脱也会更有利，因此回报也会越高。

为测试该算法，我们通过Python语言进行编程，其中，累计回报收敛趋势图如图 17所示：

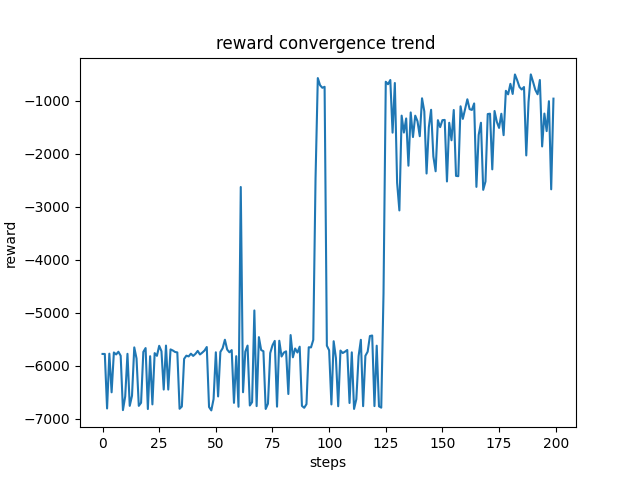


图 17 两犬累计回报收敛趋势图

由图可以看出，累计回报在约125次训练后出现了突变，突然上升到一个稳定值附近，这意味着，回报在大约训练125次左右时开始收敛。而在125此训练之后，累计回报出现上下浮动的情况，这是由于强化学习中训练数据分布的迭代性，策略更新后可能遇到一些之前没有遇到过的状态，即由于犬的位置是随机生成的，智能羊根据过往的经验，也许只能在某些犬-羊特定位置组合才能实现逃脱，而在新生成的环境中，并不能实现逃脱，所以在后续的训练中，回报下降到原先的水平。这种智能羊“学会一半”的情形一共出现了2次，在大概第60次和第100次训练中。经过125次训练后，智能羊已具备成功逃逸的能力，对于任意初始位置的犬，都能够实现逃跑目标，因此**收敛步数**约为125，可见算法的**学习能力较强**，且算法具有**实用性**。

为进一步测试并评价该算法，我们通过更改不同的两犬初始位置，对智能羊进行测试训练。选取其中有代表性的测试，训练成功后的测试结果如下：

* 测试1

|  |  |
| --- | --- |
| 图 18 测试1：羊的逃逸半径 | 图 19 测试1：羊的逃逸极角 |

|  |  |
| --- | --- |
| 图 20 测试1：犬1的追捕极角 | 图 21测试1：犬2的追捕极角 |

|  |
| --- |
| 犬1起始点  图 22测试1：羊犬围捕轨迹图（红点为羊，蓝点为犬） |

结合图12-图16可见：羊为远离两犬，一开始选择沿顺时针方向运动，此时犬一沿顺时针方向、犬二沿逆时针接近羊。约4步长后，羊为远离两犬，改为逆时针方向运动，并寻求逃出点。此时犬二要靠近羊，仍需沿逆时针方向运动，而犬一离羊较近，在羊改为逆时针方向运动时也立刻转向前往拦截羊。因犬并不知道羊的准确动向，而只是希望与羊间的距离最短，因此犬1在20步长左右开始有几次根据与羊的距离的小范围来回跑动。最终，羊避开两犬，成功逃出。

犬2起始点

* 测试2

|  |  |
| --- | --- |
| 图 23 测试2：羊的逃逸半径 | 图 24 测试2：羊的逃逸极角 |

|  |  |
| --- | --- |
| 图 25测试2：犬1的追捕极角 | 犬2起始点  图 26测试2：犬2的追捕极角 |
| 犬1起始点  图 27测试2：羊犬围捕轨迹图（红点为羊，蓝点为犬） | |

结合图17-图21，我们可以看出，根据两犬的初始位置，羊一开始选择沿顺时针方向运动，因此两犬相应地准备“夹击”，相向而行。羊运动4步长左右时，因其离犬1较近，若继续沿该方向运动很有可能被围堵，因此羊改为逆时针方向运动。因此，犬1继续沿逆时针方向运动追捕羊，而犬2在顺时针方向运动一段后，观测到羊的新动向，立刻往回跑，想要拦截羊，并根据与羊的距离有小范围来回跑动。最终，羊成功逃逸。

* 测试3

|  |  |
| --- | --- |
| 图 28测试3：羊的逃逸半径 | 图 29测试3：羊的逃逸极角 |
| 图 30测试3：犬1的追捕极角 | 图 31测试3：犬2的追捕极角 |
| 犬2起始点  犬1起始点  图 32测试3：羊犬围捕轨迹图（红点为羊，蓝点为犬） | |

结合图22-26可见：羊为远离犬，一开始选择顺时针方向运动，此时因其相对犬1为往右下方运动，故犬1沿顺时针方向运动一小段后再改为沿逆时针围堵。犬2沿顺时针方向拦截羊。约6步长后，因羊离犬1较近，羊为逃离两犬的拦截，改为逆时针方向运动，并寻求逃出点。此时犬1为拦截羊，短暂地沿顺时针方向运动后又立即改为沿逆时针方向运动，在约40步长时，犬1再次短暂改变方向后继续原方向追赶。而犬2则在发现羊往圆周逃跑时改为沿逆时针方向拦截羊，后根据羊的距离在小范围内来回跑动。最终，羊避开两犬，成功逃出。

* 测试4

|  |  |
| --- | --- |
| 图 33测试4：羊的逃逸半径 | 图 34 测试4：羊的逃逸极角 |
| 图 35测试4：犬1的追捕极角 | 图 36测试4：犬2的追捕极角 |
| 犬1起始点  犬2起始点  图 37测试4：羊犬围捕轨迹图（红点为羊，蓝点为犬） | |

结合图27-31可见：据两犬初始位置，羊首先选择沿顺时针方向运动，此时犬一沿顺时针方向拦截羊，犬二沿逆时针方向拦截羊。约3步长后，羊为逃离两犬的拦截，改为逆时针方向运动，并寻求逃出点。此时羊相对于两犬的位置未发生突变，所以两犬继续原方向拦截。在10步长时，羊相对于犬一的方向改变，犬一改为沿逆时针方向运动，并于约20步长后根据羊的位置反复变换跑动方向。犬二则始终保持原方向拦截。最终，羊成功避开两犬，成功逃出。

* 总结：

结合以上4次测试结果，我们可以得到几次测试中，智能羊逃脱所需步数，如下表：

表 3 逃脱所需步数

|  |  |
| --- | --- |
| 测试 | 逃脱步数 |
| 1 | 27 |
| 2 | 33 |
| 3 | 43 |
| 4 | 28 |

通过上述四次测试，可见算法训练出来的智能羊的运动及策略都是符合实际、可描述的，因此体现了算法的**科学性**。比较4次测试我们可以发现：（1）当初始两犬的极角差维持在一定范围的时候，羊逃脱步数会相对较小；（2）而初始两犬的极角差过小时，初始羊的策略选择往离两犬位置较远的方向跑，结果会使得两犬极角差变大，化归为（1）的情形解决，但我们通过第二问结论可以知道，对于智能羊来说，此时最佳策略应该使得两犬极角进一步减小，这样逃脱步数可以进一步减少，说明智能羊学习到逃脱的策略，但不一定是最优的逃脱策略。

# 模型的评价与推广

## 模型的评价

### 模型的优点

1. 科学性：本文研究犬羊追逃的机器学习模型时，充分结合了运动学的知识，考虑到了犬羊的运动规则。

2. 创新性：通过运动学相关的改进，本文将DDPG算法应用于一个全新的犬羊追逐问题。

3. 普适性：本文提出的犬羊机器学习算法，对于二维情形的追逐博弈具有推广价值。

### 模型的缺点

1. 多犬之间的合作围捕会使得问题更复杂，本文忽略了这种情况，只考虑犬与犬之间是独立决策的。

2. 本文只研究了单羊的逃脱情形，在多羊逃脱时，本文提出的羊与犬的规则需要相应的调整。

## 模型推广

本文在研究上由于时间的限制，存在可以继续深入讨论的地方。不仅仅在于将多对多的情形由独立推广成合作的情形，还可以将犬羊追逐从二维推广成三维。三维情形的追逐博弈，将对航空航天、机器人以及控制等领域都有重要意义。

# 参考文献

1. 微分对策

<https://baike.baidu.com/item/%E5%BE%AE%E5%88%86%E5%AF%B9%E7%AD%96/710085?fr=aladdin>

1. 刘冰雁,叶雄兵,高勇,王新波,倪蕾.基于分支深度强化学习的非合作目标追逃博弈策略求解[J].航空学报,2020,41(10):348-358.
2. 谭浪,巩庆海,王会霞.基于深度强化学习的追逃博弈算法[J].航天控制,2018,36(06):3-8+19.
3. 强化学习

<https://baike.baidu.com/item/%E5%BC%BA%E5%8C%96%E5%AD%A6%E4%B9%A0/2971075?fr=aladdin>

1. Deep Reinforcement Learning - 1. DDPG原理和算法

<https://blog.csdn.net/kenneth_yu/article/details/78478356>

1. 强化学习中性能的评价指标到底应该如何选择

<https://www.cnblogs.com/devilmaycry812839668/p/14322015.html>

# 附录

## 羊的逃逸智能算法代码

### 环境

1. **import** gym
2. **from** gym **import** spaces
3. **import** numpy as np
4. **import** math
5. **import** random
6. **from** gym.envs.classic\_control **import** rendering
8. sigma=10
10. # 将弧度转换为角度
11. **def** trans(tmp):
12. **return** 360\*(tmp/(2\*np.pi))
14. # 更新犬的状态
15. **def** change\_dog\_state(thetaP,thetaE,delta\_theta):
16. new\_thetaP=thetaP
17. clockwise = (thetaP - delta\_theta + 2 \* np.pi) % (2 \* np.pi)  # 顺时针
18. counterclockwise = (thetaP + delta\_theta + 2 \* np.pi) % (2 \* np.pi)  # 逆时针
19. **if** thetaE > thetaP:
20. **if** thetaE - thetaP >= np.pi:
21. new\_thetaP = clockwise
22. **else**:
23. new\_thetaP = counterclockwise
24. **elif** thetaE < thetaP:
25. **if** thetaP - thetaE >= np.pi:
26. new\_thetaP = counterclockwise
27. **else**:
28. new\_thetaP = clockwise
29. **return** new\_thetaP
31. # 计算夹角
32. **def** cal\_angel(theta1,theta2):
33. ans=0
34. **if** theta1 > theta2:
35. ans = theta1 - theta2
36. **if** ans > np.pi:
37. ans = 2 \* np.pi - ans  # （补）角
38. **else**:
39. ans = theta2 - theta1
40. **if** ans > np.pi:
41. ans = 2 \* np.pi - ans
42. **return** ans
44. # 判断羊是否给抓住
45. **def** catch(R,theta1,theta2,theta3):
46. x=R\*np.cos(theta1)
47. y=R\*np.sin(theta1)
48. a=R\*np.cos(theta2)
49. b=R\*np.sin(theta2)
50. A=R\*np.cos(theta3)
51. B=R\*np.sin(theta3)
52. len1=math.sqrt((x-a)\*(x-a)+(y-b)\*(y-b))
53. len2=math.sqrt((x-A)\*(x-A)+(y-B)\*(y-B))
54. **if** len1 <= sigma **and** len2 <= sigma:
55. **return** True
56. **else**:
57. **return** False
59. **class** dogSheepEnv(gym.Env):
60. **def** \_\_init\_\_(self):
61. # self.dt = 0.2  # 采样时间
62. self.dt=0.2
63. # self.thetaP=np.pi/2# 狗的极坐标
64. self.thetaP = random.uniform(0, 2 \* np.pi)# 狗1的极坐标
65. self.wP=np.pi/5# 狗的角速度
66. self.thetaP2=random.uniform(0, 2 \* np.pi)# 狗1的极坐标
67. self.vE=32# 羊的速度
68. self.thetaE=np.pi/2# 羊的极坐标
69. self.radiusE=0# 羊的极坐标半径
70. self.R=100# 圆的半径
71. self.state=np.array([self.thetaE,self.radiusE,self.thetaP,self.thetaP2])# 环境的初始状态
72. self.viewer = rendering.Viewer(400, 400)# 画板
73. self.lambda1=0.07# reward的参数1
74. self.lambda2=3.1# reward的参数2
75. self.lambda3=3.1
76. self.lambda4=6.2
78. # 自定义动作空间，观察空间
79. self.action\_space = spaces.Box(
80. # 羊的动作空间即为转动的角度，会根据当前位置进行变化
81. # 由于怕出现low比high还大的情况，我们的action\_space就不做周期处理，用的时候取余2pi就行
82. low=0, high=2\*np.pi, shape=(1,), dtype=np.float32
83. )
84. self.observation\_space = spaces.Box(
85. # 状态空间为 theta\_E,R\_E,theta\_P
86. low=np.array([0,0,0,0])
87. ,high=np.array([2\*np.pi,self.R,2\*np.pi,2\*np.pi])
88. ,dtype=np.float32
89. )
90. '''''
91. 羊接受一个动作进行位移: 使用PG算法的choose\_action
92. 犬沿劣弧进行位移
93. 接着判断游戏是否结束
94. 评价这个动作的回报
95. '''
96. **def** step(self, action):# u为action
97. # print('action: ',action)
98. # 根据action（即θ\_E'来计算新的状态）
99. self.state = self.\_get\_observation(action)
100. reward = self.\_get\_reward()
101. done = self.\_get\_done()
102. **if** done:# 如果逃脱失败，给予惩罚
103. **if** catch(self.R,self.state[0],self.state[2],self.state[3]):
104. reward=reward-1000
105. **print**('be catched')
106. **else**:
107. reward=0
108. **print**('no be catched')
109. **return** self.state,reward,done
111. # 获取reward,根据action作用之后的state来计算reward
112. **def** \_get\_reward(self):
113. # thetaP=self.state[2]
114. # thetaP2=self.state[3]
115. # thetaE=self.state[0]
116. thetaE,thetaP,thetaP2=self.state[0],self.state[2],self.state[3]
117. delta\_theta1=cal\_angel(thetaE,thetaP)# 羊与犬1的夹角
118. delta\_theta2=cal\_angel(thetaE,thetaP2)# 羊与犬2的夹角
119. delta\_theta3=cal\_angel(thetaP,thetaP2)# 两犬之间的夹角
120. # a=self.state[1]
121. # b=self.R
122. # distance=math.sqrt(a\*a+b\*b-2\*a\*b\*np.cos(delta\_theta))
123. # 羊距圆周越近越好(radiusE越大越好)，羊与犬的夹角越大越好,羊离犬越远越好
124. # print('r1: ',self.lambda1 \* abs(self.R - self.state[1]))
125. # print('r2: ',self.lambda2 \* abs(np.pi-delta\_theta1))
126. # print('r3: ',self.lambda3 \* abs(np.pi-delta\_theta2))
127. # print('r4: ',self.lambda4 \* abs(delta\_theta3))
128. **return** -(# 想要趋近于零
129. self.lambda1 \* abs(self.R - self.state[1])# 范围 [0-2\*R(200)]
130. + self.lambda2 \* abs(np.pi-delta\_theta1) # 范围 [0-100]
131. + self.lambda3 \* abs(np.pi-delta\_theta2) # 范围 [0-100]
132. + self.lambda4 \* abs(delta\_theta3)   # 范围 [0-100]
133. )
135. # 判断游戏是否结束
136. **def** \_get\_done(self):
137. **if** self.state[1]>=self.R:
138. **return** True
139. **else**:
140. **return** False
141. # 根据action修改环境，改变状态
142. **def** \_get\_observation(self,action):
143. # 已知现在的位置，首先计算位移后羊的极坐标
144. xb=self.state[1]\*np.cos(self.state[0])+self.vE\*self.dt\*np.cos(action)
145. yb=self.state[1]\*np.sin(self.state[0])+self.vE\*self.dt\*np.sin(action)
146. new\_radiusE=math.sqrt(xb\*xb+yb\*yb)
147. # 由xb和yb进行θ转换,# 返回弧度pi
148. new\_thetaE=math.atan2(yb,xb)
149. new\_thetaE=(new\_thetaE+2\*np.pi)%(2\*np.pi)
150. # 根据羊的action，选择狼的位移方向并位移
151. delta\_theta=self.wP\*self.dt
152. thetaE = self.state[0]
153. # 修改犬1的状态
154. thetaP = self.state[2]# 犬1的原状态
155. new\_thetaP=change\_dog\_state(thetaP,thetaE,delta\_theta)# 犬1的新状态
156. # 修改犬2的状态
157. thetaP2 = self.state[3]  # 犬1的原状态
158. new\_thetaP2 = change\_dog\_state(thetaP2, thetaE, delta\_theta)  # 犬1的新状态
159. # 相等的话就保持原状态
160. **return** np.array([new\_thetaE,new\_radiusE,new\_thetaP,new\_thetaP2])
162. # 重置羊和犬的状态
163. **def** reset(self):
164. thetaE=random.uniform(0, 2 \* np.pi)
165. thetaE2=(thetaE+np.pi)%(2\*np.pi)
166. self.state=np.array([0,0,thetaE,thetaE2],dtype=float)
167. **return** np.array(self.state)
169. # 画画显示犬和羊的状态
170. **def** render(self):
171. # 清空轨迹
172. # self.viewer.geoms.clear()
173. # 绘制大圆
174. ring = rendering.make\_circle(radius=self.R,res=50,filled=False)
175. transform1 = rendering.Transform(translation=(200, 200))  # 相对偏移
176. ring.add\_attr(transform1)# 让圆添加平移这个属性
177. self.viewer.add\_geom(ring)
179. # 绘制犬1
180. xP,yP=self.R\*np.cos(self.state[2]),self.R\*np.sin(self.state[2])
181. ringP = rendering.make\_circle(radius=2, res=50, filled=True)
182. ringP.set\_color(0,0,1)
183. transform\_P = rendering.Transform(translation=(200+xP, 200+yP))  # 相对偏移
184. ringP.add\_attr(transform\_P)  # 让圆添加平移这个属性
185. self.viewer.add\_geom(ringP)
186. # 绘制犬2
187. xP2, yP2 = self.R \* np.cos(self.state[3]), self.R \* np.sin(self.state[3])
188. ringP2 = rendering.make\_circle(radius=2, res=50, filled=True)
189. ringP2.set\_color(0, 0, 1)
190. transform\_P2 = rendering.Transform(translation=(200 + xP2, 200 + yP2))  # 相对偏移
191. ringP2.add\_attr(transform\_P2)  # 让圆添加平移这个属性
192. self.viewer.add\_geom(ringP2)
194. # 绘制羊
195. xE, yE = self.state[1] \* np.cos(self.state[0]), self.state[1] \* np.sin(self.state[0])
196. ringE = rendering.make\_circle(radius=2, res=50, filled=True)
197. ringE.set\_color(1, 0, 0)
198. transform\_E = rendering.Transform(translation=(200+xE, 200+yE))  # 相对偏移
199. ringE.add\_attr(transform\_E)  # 让圆添加平移这个属性
200. self.viewer.add\_geom(ringE)
202. **return** self.viewer.render()
204. # env = dogSheepEnv()
205. # while True:
206. #     env.reset()
207. #     for \_ in range(2000):
208. #         env.render()
209. #         action=random.uniform(0,2\*np.pi)
210. #         action=np.clip(action,env.state[0]-np.pi/2,env.state[0]+np.pi/2)
211. #         action=(action+2\*np.pi)%(2\*np.pi)
212. #         state, reward, done = env.step(action) # 和环境交互
213. #         if done:
214. #             break

### 训练及测试

1. **import** math
3. **import** numpy as np
5. **from** env **import** dogSheepEnv
6. **from** rl **import** DDPG
7. **import** matplotlib.pyplot as plt
8. **import** time
10. MAX\_EPISODES = 200# 比赛次数
11. MAX\_EP\_STEPS = 2000# 每把比赛的步数
12. ON\_TRAIN = False# 控制程序是进行训练还是进行测试
13. sigma=10 # 碰撞精度
15. # reward\_list=[]# 准备画图
16. # ep\_reward\_list=[]
17. thetaP\_list=[]
18. thetaP2\_list=[]
19. thetaE\_list=[]
20. rE\_list=[]
22. # 设置环境
23. env = dogSheepEnv()
24. # 设置维度
25. state\_dim = env.observation\_space.shape[0]
26. action\_dim = env.action\_space.shape[0]
27. action\_bound = env.action\_space.high
29. # 设置强化学习模型
30. rl = DDPG(action\_dim, state\_dim, action\_bound)
32. # 判断羊是否给抓住
33. **def** catch(R,theta1,theta2,theta3):
34. x=R\*np.cos(theta1)
35. y=R\*np.sin(theta1)
36. a=R\*np.cos(theta2)
37. b=R\*np.sin(theta2)
38. A=R\*np.cos(theta3)
39. B=R\*np.sin(theta3)
40. len1=math.sqrt((x-a)\*(x-a)+(y-b)\*(y-b))
41. len2=math.sqrt((x-A)\*(x-A)+(y-B)\*(y-B))
42. **if** len1 <= sigma **and** len2 <= sigma:
43. **return** True
44. **else**:
45. **return** False
47. **def** trans(tmp):
48. **return** 360\*(tmp/(2\*np.pi))
50. # def dis(R,theta1,theta2):
51. #     x=R\*np.cos(theta1)
52. #     y=R\*np.sin(theta1)
53. #     a=R\*np.cos(theta2)
54. #     b=R\*np.sin(theta2)
55. #     len=math.sqrt((x-a)\*(x-a)+(y-b)\*(y-b))
56. #     if len <= sigma:
57. #         return False
58. #     else:
59. #         return True
61. # 训练过程
62. '''''
63. env和算法的交互
64. 传state
65. 将一次学习的经历装进记忆库
67. rl模型一直将学习经历填进记忆库，直到记忆库满了才开始学习
68. 填充记忆库的过程中，环境不断地交互
69. '''
70. **def** train():
71. **for** i **in** range(MAX\_EPISODES):
72. **print**('i: ',i)
73. state = env.reset()
74. ep\_reward = 0.# 单局比赛的总reward
75. **for** j **in** range(MAX\_EP\_STEPS):
76. env.render()# 画图
77. action = rl.choose\_action(state)# 算法预测下一个动作
78. action=np.random.normal(action,scale=0.01) # 随机一下
79. # 这里限制一下动作空间
80. action=np.clip(action,env.state[0]-np.pi/2,env.state[0]+np.pi/2)
81. action=(action+2\*np.pi)%(2\*np.pi)
82. \_state, reward, done = env.step(action) # 和环境交互
83. # reward\_list.append(reward)
84. # print('reward: ',reward)
85. # print('i: ',i,' choose\_action: ', trans(action[0]),' reward: ',reward,' state: ',\_state)
86. rl.store\_transition(state, action, reward, \_state)# 把这次经历装进记忆库
87. ep\_reward += reward
88. # 记忆模块填完之后算法开始学习
89. **if** rl.memory\_full:
90. rl.learn()
91. state = \_state
92. # time.sleep(0.2)
93. **if** done **or** j == MAX\_EP\_STEPS-1: # 结束
94. # if env.state[1] >= env.R and dis(env.R,env.state[0],env.state[2]):
95. **if** (env.state[1] >= env.R) **and** (**not** catch(env.R, env.state[0], env.state[2],env.state[3])):
96. **print**('sheep win')
97. **else**:
98. **print**('dog win')
99. # ep\_reward\_list.append(ep\_reward)
100. **print**('Ep: %i | %s | ep\_r: %.1f | steps: %i' % (i, '---' **if** **not** done **else** 'done', ep\_reward, j))
101. **break**
102. rl.save() # 保存模型
104. # 测试
105. **def** eval():
106. rl.restore()# 提取模型
107. # env.render()
108. # env.viewer.set\_vsync(True)
109. # while True:
110. #     # print('新的一次')
111. #     state = env.reset()
112. #     for \_ in range(1000):
113. #         env.render()
114. #         action = rl.choose\_action(state)
115. #         action = np.random.normal(action, scale=0.01)  # 随机一下
116. #         # 这里限制一下动作空间
117. #         action = np.clip(action, env.state[0] - np.pi / 2, env.state[0] + np.pi / 2)
118. #         action = (action + 2 \* np.pi) % (2 \* np.pi)
119. #         # print('choose action: ',action,'state: ',env.state)
120. #         state, reward, done = env.step(action)
121. #         thetaE\_list.append(state[0])
122. #         rE\_list.append(state[1])
123. #         thetaP\_list.append(state[2])
124. #         if done:
125. #             if env.state[1] >= env.R and dis(env.R,env.state[0],env.state[2]):
126. #                 print('sheep win')
127. #             else:
128. #                 print('dog win')
129. #             break
130. state = env.reset()
131. **print**('thetaP: ',state[2])
132. **print**('thetaP2: ', state[3])
133. **for** \_ **in** range(1000):
134. env.render()
135. action = rl.choose\_action(state)
136. # 这里限制一下动作空间
137. action = np.clip(action, env.state[0] - np.pi / 2, env.state[0] + np.pi / 2)
138. action = (action + 2 \* np.pi) % (2 \* np.pi)
139. state, reward, done = env.step(action)
140. thetaE\_list.append(state[0])
141. rE\_list.append(state[1])
142. thetaP\_list.append(state[2])
143. thetaP2\_list.append(state[3])
144. # print('choose action: ', action,' reward: ',reward, 'state: ', env.state)
145. **if** done:
146. **break**
147. input('input: ')

150. **if** ON\_TRAIN:
151. train()
152. **else**:
153. eval()
155. # 画reward图
156. # plt.figure()
157. # len2=len(ep\_reward\_list)
158. # plt.plot(list(range(len2)),ep\_reward\_list)
159. # plt.title('reward convergence trend ')
160. # plt.xlabel('steps')
161. # plt.ylabel("reward")
162. # plt.show()
164. # 画犬1的图
165. plt.figure()
166. plt.plot(list(range(len(thetaP\_list))),thetaP\_list)
167. plt.title('pursuer1 theta')
168. plt.xlabel('steps')
169. plt.ylabel("theta")
170. plt.show()
172. # 画犬2的图
173. plt.figure()
174. plt.plot(list(range(len(thetaP2\_list))),thetaP2\_list)
175. plt.title('pursuer2 theta')
176. plt.xlabel('steps')
177. plt.ylabel("theta")
178. plt.show()
180. # 画羊的极角
181. plt.figure()
182. plt.plot(list(range(len(thetaE\_list))),thetaE\_list)
183. plt.title('escaper theta')
184. plt.xlabel('steps')
185. plt.ylabel("theta")
186. plt.show()
188. # 画羊的极径
189. plt.figure()
190. plt.plot(list(range(len(rE\_list))),rE\_list)
191. plt.title('escaper radius')
192. plt.xlabel('steps')
193. plt.ylabel("radius")
194. plt.show()

### DDPG算法

1. **import** tensorflow as tf
2. **import** numpy as np
4. #####################  hyper parameters  ####################
6. LR\_A = 0.001    # learning rate for actor
7. LR\_C = 0.001    # learning rate for critic
8. GAMMA = 0.9     # reward discount
9. TAU = 0.01      # soft replacement
10. MEMORY\_CAPACITY = 10000
11. BATCH\_SIZE = 32

14. **class** DDPG(object):
15. **def** \_\_init\_\_(self, a\_dim, s\_dim, a\_bound,):
16. self.memory = np.zeros((MEMORY\_CAPACITY, s\_dim \* 2 + a\_dim + 1), dtype=np.float32)
17. self.pointer = 0
18. self.memory\_full = False
19. self.sess = tf.Session()
20. self.a\_replace\_counter, self.c\_replace\_counter = 0, 0
22. # self.a\_dim, self.s\_dim, self.a\_bound = a\_dim, s\_dim, a\_bound[1]
23. self.a\_dim, self.s\_dim, self.a\_bound = a\_dim, s\_dim, a\_bound
24. self.S = tf.placeholder(tf.float32, [None, s\_dim], 's')
25. self.S\_ = tf.placeholder(tf.float32, [None, s\_dim], 's\_')
26. self.R = tf.placeholder(tf.float32, [None, 1], 'r')
28. with tf.variable\_scope('Actor'):
29. self.a = self.\_build\_a(self.S, scope='eval', trainable=True)
30. a\_ = self.\_build\_a(self.S\_, scope='target', trainable=False)
31. with tf.variable\_scope('Critic'):
32. # assign self.a = a in memory when calculating q for td\_error,
33. # otherwise the self.a is from Actor when updating Actor
34. q = self.\_build\_c(self.S, self.a, scope='eval', trainable=True)
35. q\_ = self.\_build\_c(self.S\_, a\_, scope='target', trainable=False)
37. # networks parameters
38. self.ae\_params = tf.get\_collection(tf.GraphKeys.GLOBAL\_VARIABLES, scope='Actor/eval')
39. self.at\_params = tf.get\_collection(tf.GraphKeys.GLOBAL\_VARIABLES, scope='Actor/target')
40. self.ce\_params = tf.get\_collection(tf.GraphKeys.GLOBAL\_VARIABLES, scope='Critic/eval')
41. self.ct\_params = tf.get\_collection(tf.GraphKeys.GLOBAL\_VARIABLES, scope='Critic/target')
43. # target net replacement
44. self.soft\_replace = [[tf.assign(ta, (1 - TAU) \* ta + TAU \* ea), tf.assign(tc, (1 - TAU) \* tc + TAU \* ec)]
45. **for** ta, ea, tc, ec **in** zip(self.at\_params, self.ae\_params, self.ct\_params, self.ce\_params)]
47. q\_target = self.R + GAMMA \* q\_
48. # in the feed\_dic for the td\_error, the self.a should change to actions in memory
49. td\_error = tf.losses.mean\_squared\_error(labels=q\_target, predictions=q)
50. self.ctrain = tf.train.AdamOptimizer(LR\_C).minimize(td\_error, var\_list=self.ce\_params)
52. a\_loss = - tf.reduce\_mean(q)    # maximize the q
53. self.atrain = tf.train.AdamOptimizer(LR\_A).minimize(a\_loss, var\_list=self.ae\_params)
55. self.sess.run(tf.global\_variables\_initializer())
57. **def** choose\_action(self, s):
58. **return** self.sess.run(self.a, {self.S: s[None, :]})[0]
60. **def** learn(self):
61. # soft target replacement
62. self.sess.run(self.soft\_replace)
64. indices = np.random.choice(MEMORY\_CAPACITY, size=BATCH\_SIZE)
65. bt = self.memory[indices, :]
66. bs = bt[:, :self.s\_dim]
67. ba = bt[:, self.s\_dim: self.s\_dim + self.a\_dim]
68. br = bt[:, -self.s\_dim - 1: -self.s\_dim]
69. bs\_ = bt[:, -self.s\_dim:]
71. self.sess.run(self.atrain, {self.S: bs})
72. self.sess.run(self.ctrain, {self.S: bs, self.a: ba, self.R: br, self.S\_: bs\_})
74. **def** store\_transition(self, s, a, r, s\_):
75. transition = np.hstack((s, a, [r], s\_))
76. index = self.pointer % MEMORY\_CAPACITY  # replace the old memory with new memory
77. self.memory[index, :] = transition
78. self.pointer += 1
79. **if** self.pointer > MEMORY\_CAPACITY:      # indicator for learning
80. self.memory\_full = True
82. **def** \_build\_a(self, s, scope, trainable):
83. with tf.variable\_scope(scope):
84. net = tf.layers.dense(s, 100, activation=tf.nn.relu, name='l1', trainable=trainable)
85. a = tf.layers.dense(net, self.a\_dim, activation=tf.nn.tanh, name='a', trainable=trainable)
86. **return** tf.multiply(a, self.a\_bound, name='scaled\_a')
88. **def** \_build\_c(self, s, a, scope, trainable):
89. with tf.variable\_scope(scope):
90. n\_l1 = 100
91. w1\_s = tf.get\_variable('w1\_s', [self.s\_dim, n\_l1], trainable=trainable)
92. w1\_a = tf.get\_variable('w1\_a', [self.a\_dim, n\_l1], trainable=trainable)
93. b1 = tf.get\_variable('b1', [1, n\_l1], trainable=trainable)
94. net = tf.nn.relu(tf.matmul(s, w1\_s) + tf.matmul(a, w1\_a) + b1)
95. **return** tf.layers.dense(net, 1, trainable=trainable)  # Q(s,a)
97. **def** save(self):
98. saver = tf.train.Saver()
99. saver.save(self.sess, './params', write\_meta\_graph=False)
101. **def** restore(self):
102. saver = tf.train.Saver()
103. saver.restore(self.sess, './params')