第六章 球员智能体的基本动作

本章概要

- 原子动作
- 中间动作
- 高级动作

6.1 原子动作

- catch
  - AngDeg ang = (posBall-posGolia).getDirection();
  - ang = VecPosition::normalizeAngle(ang-WM->getAgentGlobalBodyAngle());
  - soc.makeCommand(CMD_CATCH, ang);
  - ACT->putCommandInQueue(soc);
- turn
  - soc = SoccerCommand(CMD_TURN, 45);
- turn_neck
  - Soc = SoccerCommand(CMD_TURNNECK, -60);

6.1 原子动作

- move
  - Soc =SoccerCommand(CMD_MOVE, 0, 0);
- dash
  - Soc = SoccerCommand(CMD_DASH, 80, 45);
- kick
  - Soc = SoccerCommand(CMD_KICK, 100, 90);
- say
  - Soc.makeCommand(CMD_SAY, “ABC123”);

6.2 中间动作

- kickTo踢球
- mark盯防
- turnBodyToPoint转身体到某点
- moveTopos跑向某一点
- ……

6.2 中间动作- kickTo

- 踢球kickTo函数原型：SoccerCommand kickTo (VecPosition posTarget, double dEndSpeed) (函数定义在类BasicPlayer中)
- 该方法使球员智能体将球从当前位置，踢向指定的位置，并保持球的末速度为dEndSpeed。
- 通过一个单独的kick命令，球的末速度很可能达不到指定的速度，原因有两个
  - 指定的速度大于球的最大速度
  - 当前的球速结合其相对于球员智能体的位置无法使球加速到指定速度。
- 对于第一个原因，如果目标速度超过最大速度某个百分比，则踢球命令将被执行个忽略该速度。第二个原因，球员智能体使用kickBallCloseToBody将球直接踢向靠近自己的位置，这样球员智能体可以在下个周期使用更大的power将球踢出。

6.2 中间动作- kickTo

- 代码事例：将球踢向最近的队友
  - double dist;
  - ObjectT objTea = WM->getClosestInSetTo(OBJECT_SET_TEAMMATES, WM->getAgentObjectType(), &dist);//获得距离我最近的队友，距离由dist返回。
  - VecPosition posTea = WM->getGlobalPosition(objTea);
  - soc = kickTo(posTea, SS->getBallSpeedMax());//最大速度踢向队友所在点
  - ACT->putCommandInQueue(soc);
  - return soc;

6.2 中间动作-向某对象转身体

- 向某点转身体的函数原型：SoccerCommand turnBodyToObject (ObjectT o);
- 该方法将球员智能体的身体转向某对象。它接受一个场上对象作为参数，返回一个将身体转向该点的turn命令。为了实现这一目标，球员智能体在下一个周期的位置，将根据其速度被估计出来，以补偿由于惯性将球员智能体移动到一个新的位置。
- Soc = turnBodyToObject(OBJECT_BALL);
  - ACT->putCommandInQueue(soc);

6.2 中间动作-移动向某点

- 移动向某一点的函数原型：**SoccerCommand moveToPos (VecPosition posTo, AngDeg angWhenToTurn, double dDistDashBack=0.0, bool bMoveBack=false)**
- turn和dash的选择时机

6.3球员的高级动作

- 带球
- 截球
- 传球
- Fastkick
- 射门

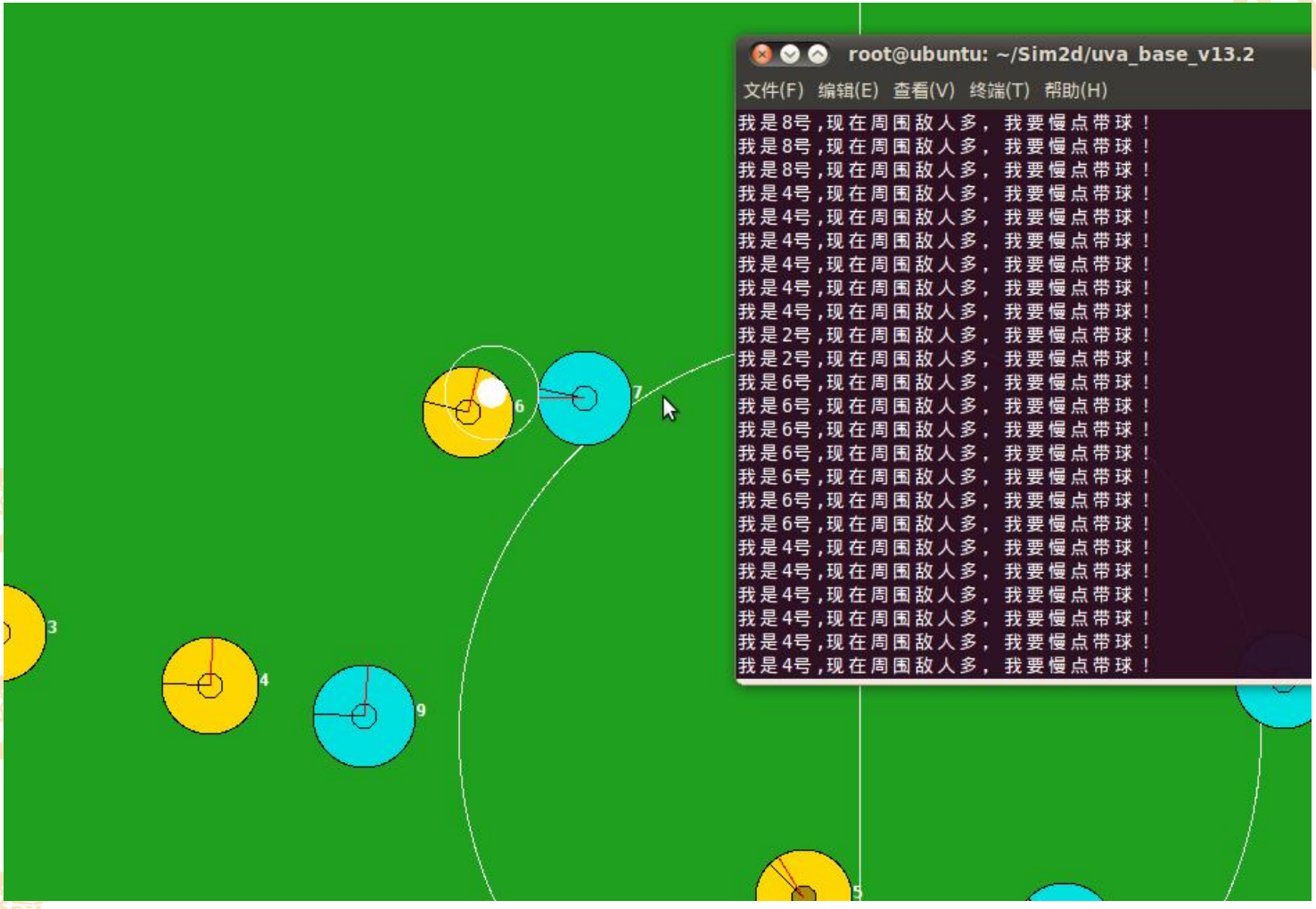
6.3 高级动作- Dribble

- 带球的原型：SoccerCommand dribble (AngDeg ang, DribbleT dribbleT) ;
- 该方法使球员智能体带球。即与球一起移动，并把球保持在一定的距离内。这实际上是重复做将球在一定的速度下踢向期望的方向，然后将其再次截住。
- 参数ang表示带球球的方向。其中第二个参数有三种形式：
  - DRIBBLE FAST: 速度较快的带球动作，球员智能体每次将球踢向相对自己较远的前方。
  - DRIBBLE SLOW: 速度较慢的带球动作，球员智能体将球保持在相对自己较近的位置。
  - DRIBBLE WITH BALL: 非常安全的带球动作，球员智能体将球保持在离自己身体非常近的位置。

6.3 高级动作- Dribble

- AngDeg angDribble = (VecPosition(52.5, 0)-posAgent).getDirection();
- Circle cir(posAgent, 6);//定义这样一个圆形区域，以我为圆心，6为半径
- int num = WM->getNrInSetInCircle(OBJECT_SET_OPPONENTS, cir);//这样就从对方球员集合中计算所有在cir中的个数。
- if (num > 0){
- cout << "我是" << WM->getPlayerNumber() << "号,现在周围敌人多，我要慢点带球！ " << endl;
- soc = dribble(angDribble, DRIBBLE_SLOW);
- }
- else{
- cout << "我是" << WM->getPlayerNumber() << "号,现在周围没人，快带球冲刺！ " << endl;
- soc = dribble(angDribble, DRIBBLE_FAST);
- }
- ACT->putCommandInQueue(soc);
- return soc;

6.3 高级动作- Dribble

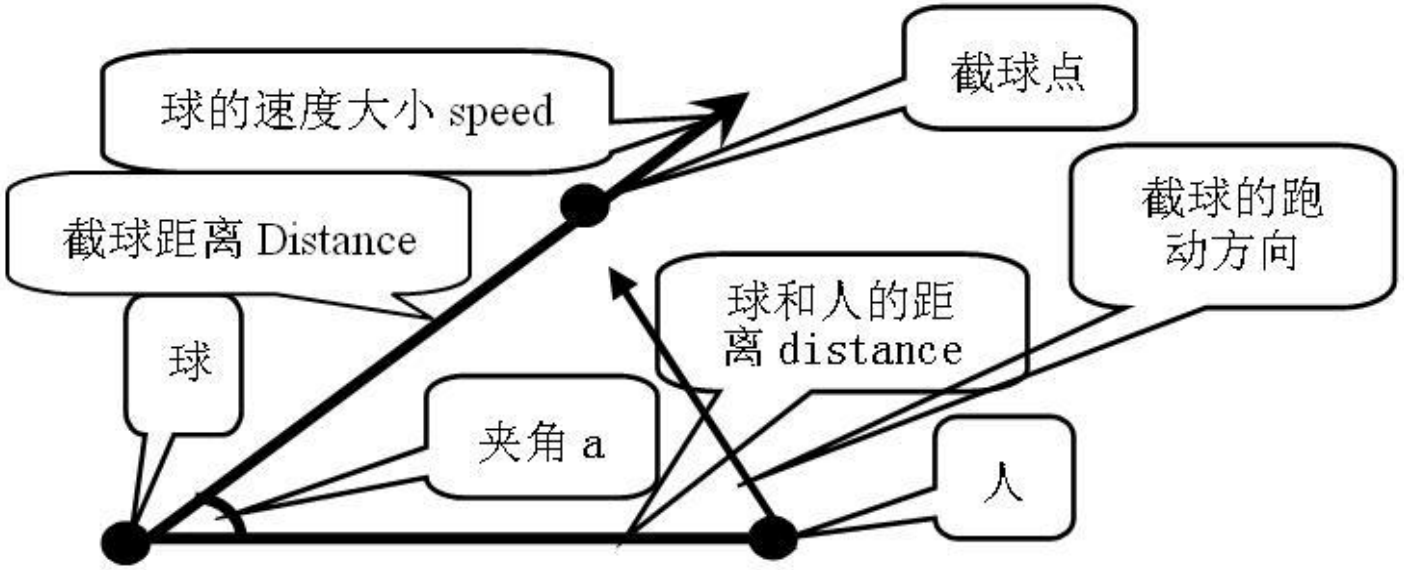


6.3.1截球（intercept）

1.问题描述

- 截球问题可以归纳成如下图的一个简单的场景：白圆圈代表球，黑色的圆圈代表球员，dist为球员到球的距离，α为球到球员之间的连线和球运动方向的夹角，speed为球的即时的运动速度。球的速度随运动衰减。截球问题归结为给定dist、α和speed，决策出队员正确的截球角度β，或者是当截到球时，球运动的距离，并给出对截球所可能花的时间的估计。具体的运动模型见相关的球员和球的运动模型。

截球示意图


球的速度大小 speed
截球点
截球距离 Distance
截球的跑动方向
球
球和人的距离 distance
夹角 a
人

6.3.1截球

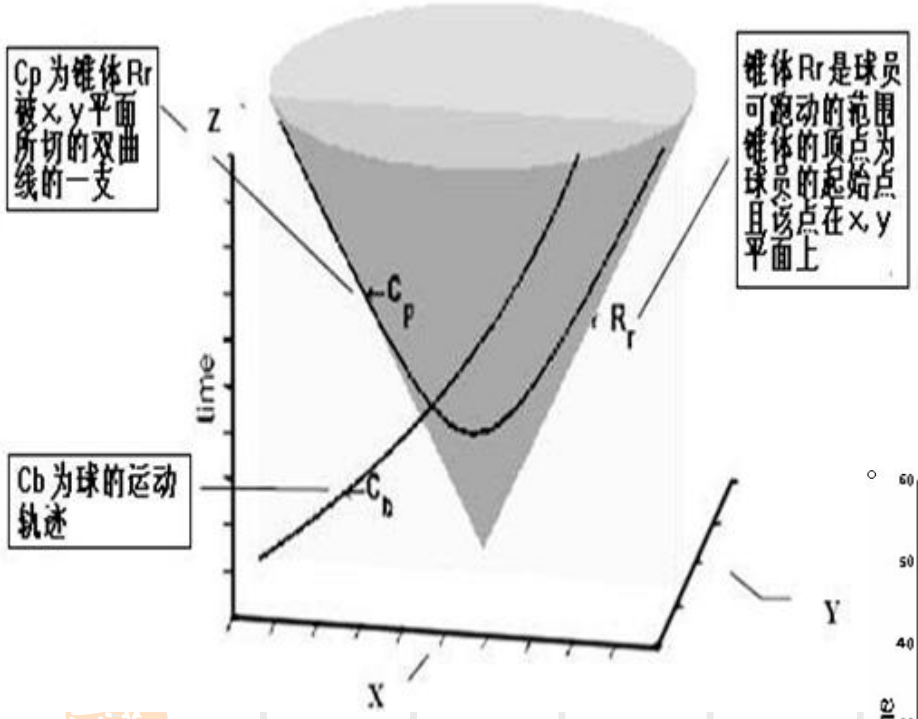
2.解决方法

- 解析法
- 通过机器学习的方法进行离线学习
- 强化学习的方法进行在线学习

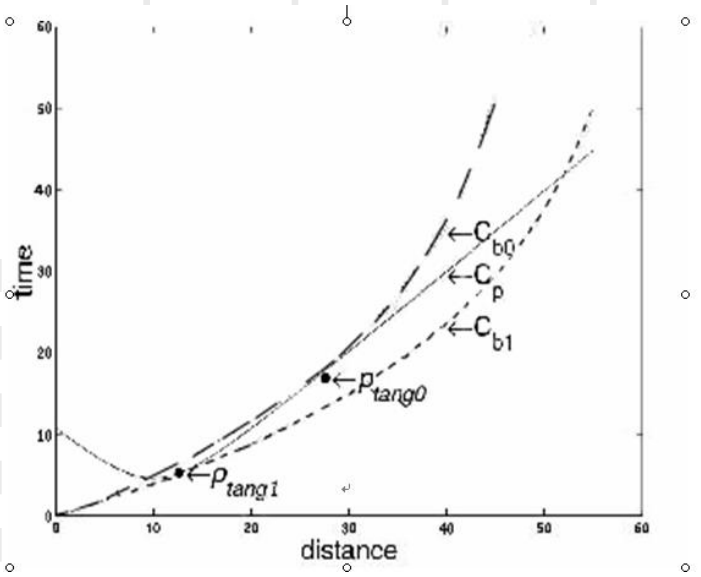
方法1：解析法

通过示意图和前面介绍的运动模型我们可以通过列出关于时间的方程，然后采用Newton迭代法求出方程的根，可以求出3个根，显然第三个根的价值不是很大，目前关注的是前2个根，然后根据高层策略选择在哪个根对应的点（前点和后点）进行截球。

解析法中的球员能够跑及的范围的三维模型



- 二维坐标系下球员能够跑及的范围是一个圆形范围
- 考虑球员的跑动速度为匀速，随时间t的增加，这个圆形的半径与t成正比增加
- 在三维坐标系下为一个锥体



- 球员的可达区域可以表示为如下方程：

$$(x-x_0)^2+(y-y_0)^2 \le (speed \cdot t + kick\_margin)^2$$

- 球的运动表示为方程：

$$\mathbf{d = initial_{speed} \cdot \frac{1-speed_{decay}^{t}}{1-speed_{decay}}}$$

- 球员的运动轨迹方程：

$$f_{cp} = \frac{\sqrt{(x - x_0)^2 + (x \cdot tga - y_0)^2} - kick\_margin}{speed}$$

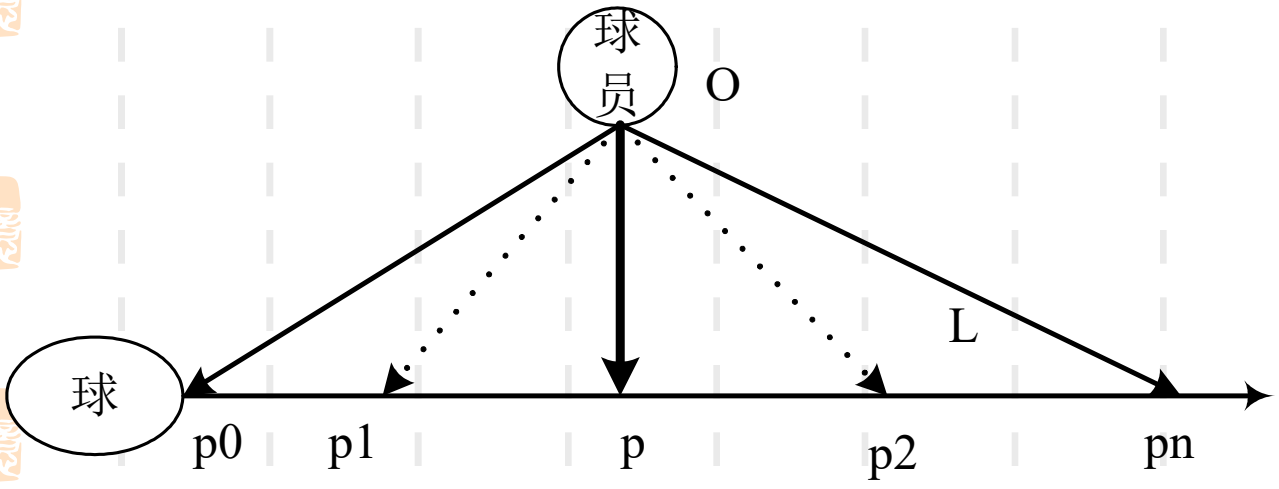
- 球的运动轨迹方程：

$$f_{cb} = \log_{Ball\_decay} 1 - \frac{x^2 (1 + tg^2 a)(1 - Ball\_decay)}{v_B}$$

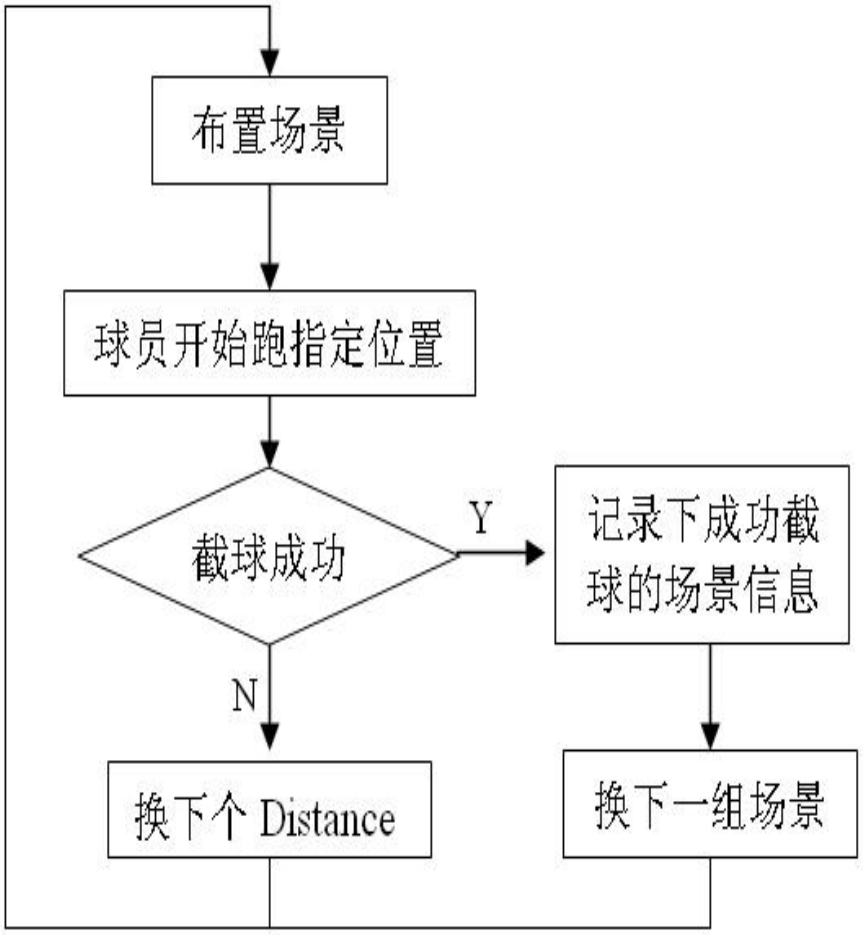
方法2:机器学习进行离线学习

- 1）决策树分类
- 2）BP网络进行函数拟合
- 3）贝叶斯网络

以BP网络为主线介绍机器学习来学习截球。



样本数据采集流程



- 截球成功条件：球速小于0.1，人球距离小于kick_able的距离；
- 场景结束条件：球速小于0.1；
- 训练时球员策略：若球kick_able，则freezeBall；否则跑到指定点；

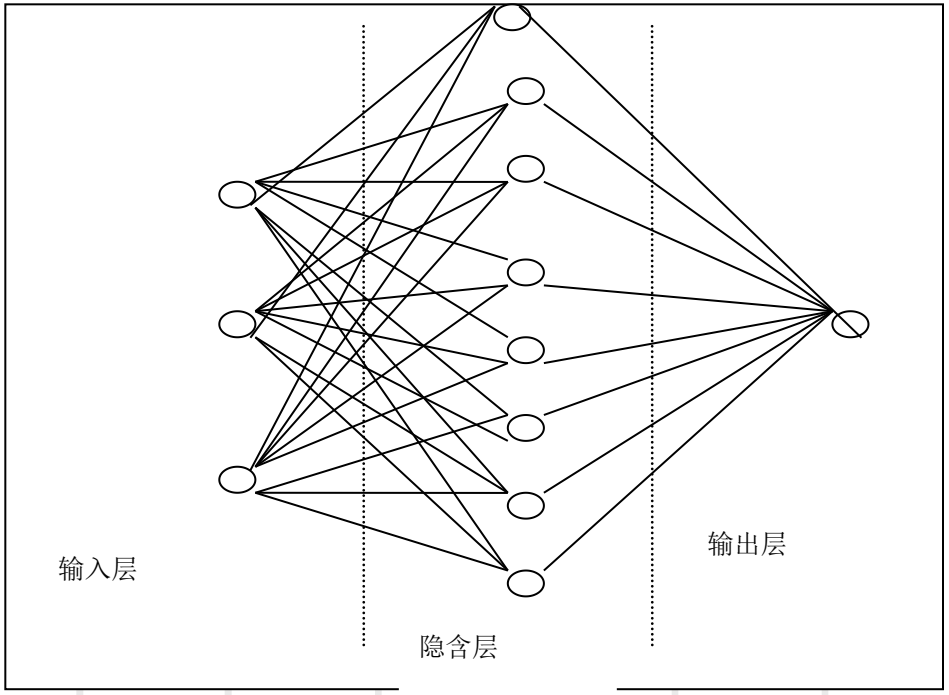
样本采集范围

① angle从0-90，离散距离15；
② distance从0-20，离散距离1；
③ speed从 0.1-2.7，离散距离0.1；

训练时采集到的部分样本数据

distance	angle	10*speed	Angle2	Time
15	45	3	30	8
10	45	4	45	10
13	45	5	22	13
13	45	6	47	12
11	45	7	80	12
11	45	8	43	17
12	45	9	26	2

BP网络学习截球



BP网络学习截球

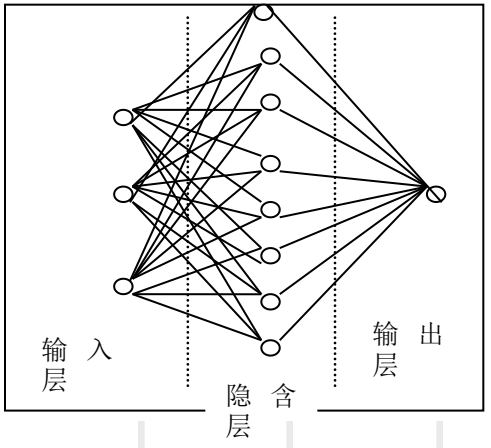
- BP网络的特点是信号由输入层单向传输到输出层，同一层神经元之间不传递信息，每个神经元与邻层所有神经元相连，连结权重用 ω_{ij}表示，各神经元的作用函数为Sigmoid函数：$f(x)=1/(1+e^{-x})$
- 它正向传播信号，反向传播误差。

BP网络学习截球

- 隐含层是BP网络的基本特征之一，事实上如果没有隐含层也就无所谓误差的反向传播了。
- 但对隐含层节点个数的选择：$\sqrt{m \times n} + 1 \sim 10$
- (m、n表示输入输出节点的个数)

信号的正向传播

- 设网络输入为x_1, x_2, x_3，输出为y。输入层各神经元的激发函数选用比例系数为1的线性函数
- 隐含层神经元的输入是：$I_i = \sum_{j=1}^{3} \omega_{ij} x_j \qquad i = 1,2,3,...8$
- 隐含层神经元的输出为：$O_i = 1/(1+e^{-I_i})$
- 输出层神经元与隐层神经元的连接权 v_i
- 则网络输出为：$y = \sum_{i=1}^{7} v_i O_i$



训练原则

- 样本　$x_{1p}, x_{2p}, x_{3p}; t_p$
- 误差　$d_p = t_p - y_p$
- 误差函数　$e_p = 1/2 \times d_p^{\,2}$
- w值随机给出，求得y_p后，误差值较大，网络计算精度不高。
- 在确定网络中隐层神经元数目的情况下，通过调整 w 的值，逐步降低误差 d_p ，以提高计算精度。

误差的反向传递

- W的修正值 $\triangle W$

$$E=\sum_{p=1}^{p} e_p$$

$$\triangle W=-\eta\frac{\partial e_p}{\partial W}$$

E<0.00001

$$\Delta W^{(n)}=-\eta\frac{\partial e_p}{\partial W}+\alpha\Delta W^{(n-1)}$$

$$\Delta v_i^{(n)}=\eta d_p\frac{\partial y_p}{\partial v_i}+\alpha\Delta v_i^{(n-1)}=\eta d_p O_{ip}+\alpha\Delta v_i^{(n-1)}$$

$$\Delta w_{ij}^{(n)}=\eta d_p\frac{\partial y_p}{\partial w_{ij}}+\alpha w_{ij}^{(n-1)}=\eta d_p v_i O_{ip}(1-O_{ip})X_{jp}+\alpha\Delta w_{ij}^{(n-1)}$$

$$W+\Delta W\rightarrow W$$

BP算法描述

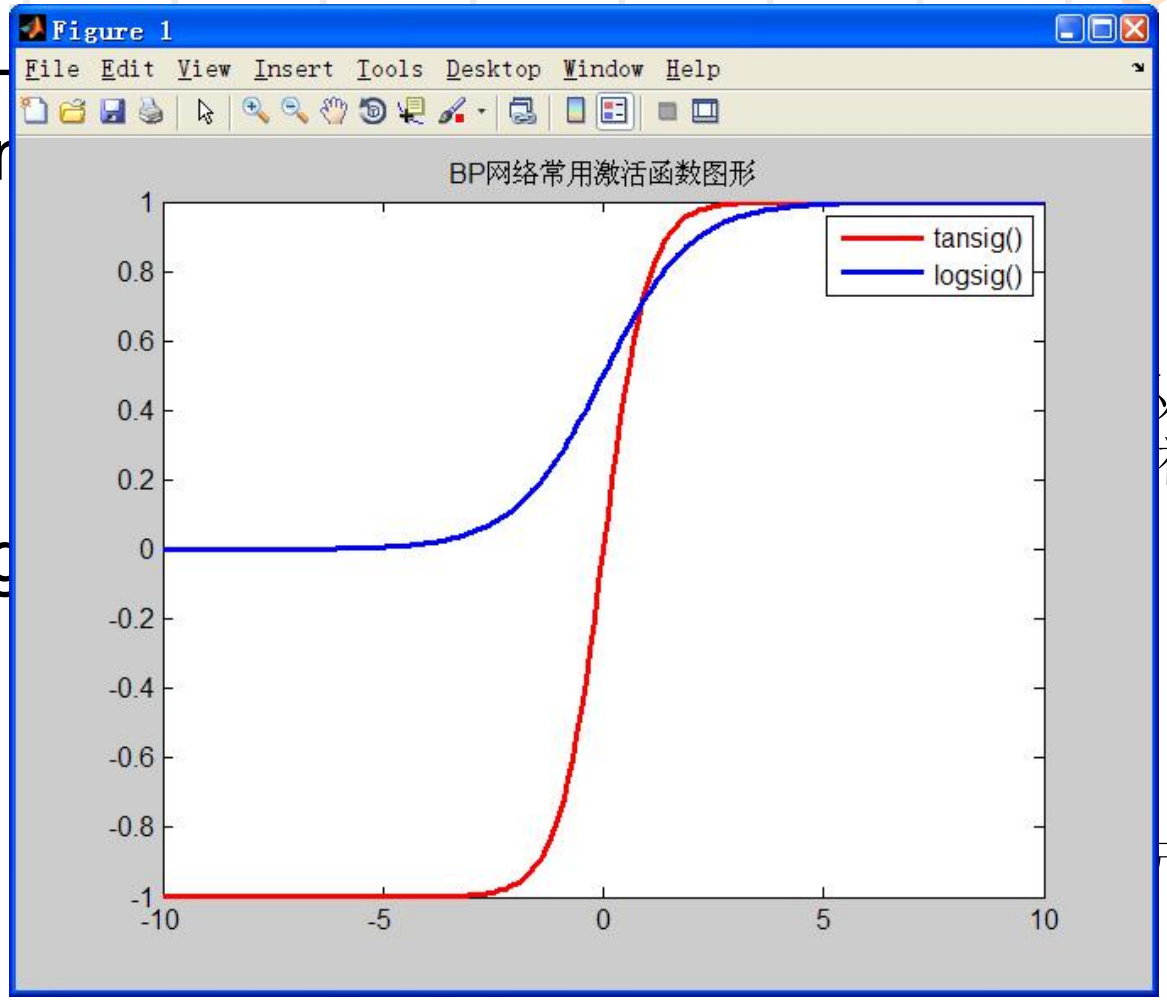
- 标准BP算法中，每输入一个样本，都要回传误差并调整权值，这种对每个样本轮训的方法称为“单样本训练”。
- 由于单样本训练遵循的是只顾眼前的“本位主义”原则，只针对每个样本产生的误差进行调整，难免顾此失彼，使训练次数增加，导致收敛速度过慢。
- 另外一种方法是，在所有样本输入之后，计算网络的总误差，再根据总误差调整权值，这种累积误差的批处理方式称为“批训练”或“周期训练”。在样本数较多时，批训练比单样本训练的收敛速度更快。

2.4.3 BP神经网络学习算法的MATLAB实现

- MATLAB中BP神经网络的重要函数和基本功能
  - newff()
    - 功能 建立一个前向BP网络
    - 格式 net = newff(PR，[S1 S2...SN1]，{TF1 TF2...TFN1}，BTF，BLF，PF)
    - 说明 net为创建的新BP神经网络；PR为网络输入取向量取值范围的矩阵；[S1 S2…SNl]表示网络隐含层和输出层神经元的个数；{TFl TF2…TFN1}表示网络隐含层和输出层的传输函数，默认为‘tansig’；BTF表示网络的训练函数，默认为‘trainlm’；BLF表示网络的权值学习函数，默认为‘learngdm’；PF表示性能数，默认为‘mse’。

2.4.3 BP神经网络学习算法的MATLAB实现

- MATL
  - tar
    -
    -
    -
  - log
    -
    -
    -



认(-∞，+∞)
神经元。

+∞)映射
元。

训练方法（续）

- 训练时，我们构造出各种情况的截球场景(传球队员固定位置，离散传球速度和传球队员和截球队员之间的相对坐标x, y)，截球队员使用各种角度截球，当成功的截球时，就将成功的数据记下。采集到的成功的数据送入神经网络用BP算法进行训练。
- 神经网络作为一个记忆的载体记录下这些成功的例子，能够进行实际各种场景的截球决策。

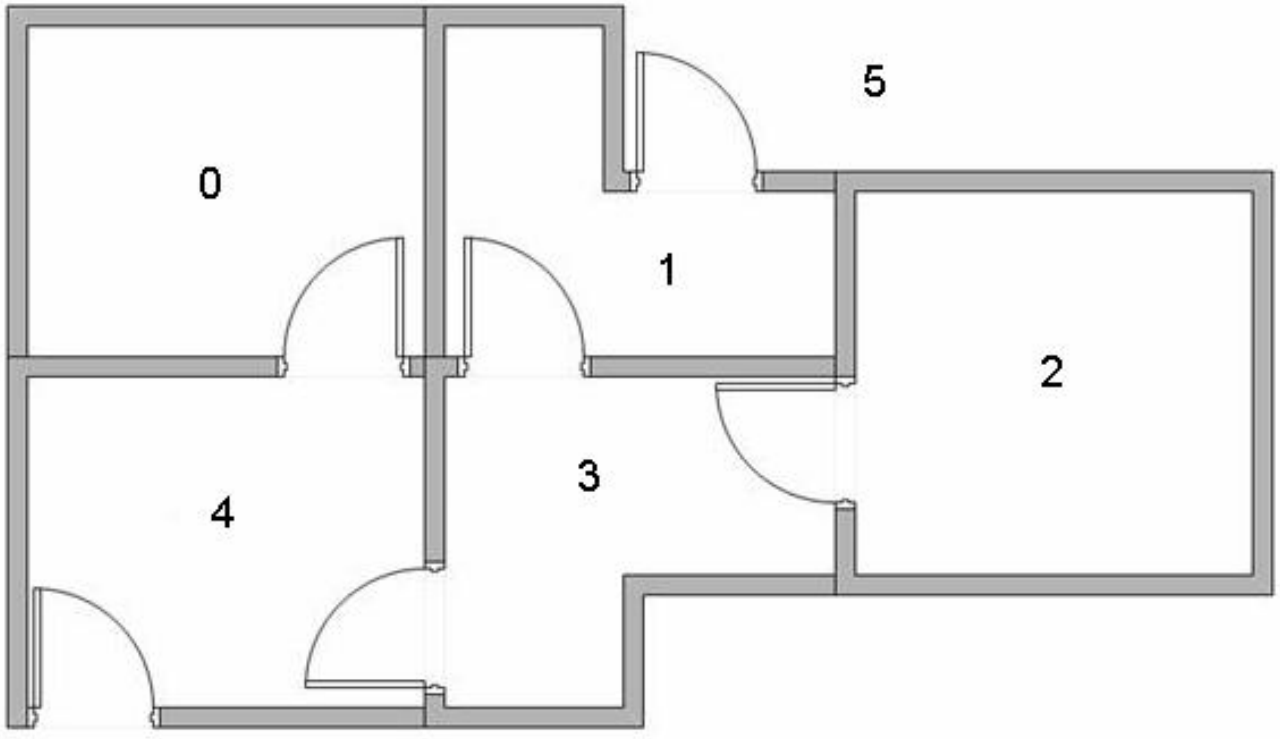
方法3：强化学习进行截球

- 强化学习是一种非监督学习的方法，它通过从环境中得到奖惩来自主发现能够得到最大奖励的策略。强化学习的特点是采用试错法搜索和延迟奖惩。
- 试错法搜索，即“吃一堑长一智”，指的是不断利用已有得知识的尝试各种动作，通过从环境中的到奖惩来修正自己的知识，从而修正策略。
- 延迟奖惩就是将得到的奖惩折算分配到每个动作中去。当一个决策过程由一系列动作组成的时候，每个动作对最后的成功都作出一定的贡献，对最后的失败也都负有一定责任。合理地把最后的奖惩分配到各个动作中，才可能学习出正确决策。

强化学习

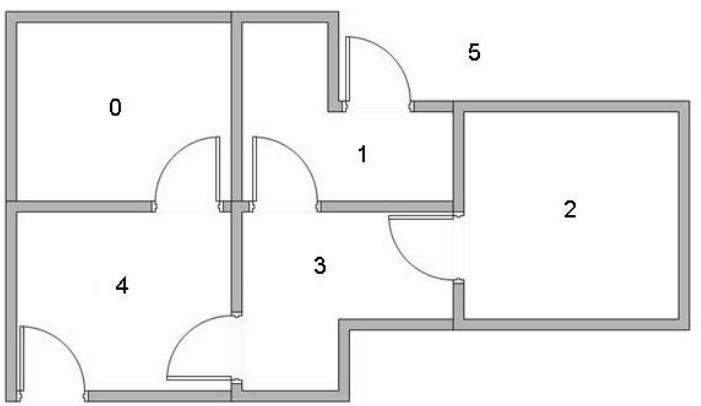
- 动态归化（Dynamic programming）
- 蒙特卡罗法（Monte Carlo）
- Q学习（Q-Learning）
- 深度强化学习（DQN）

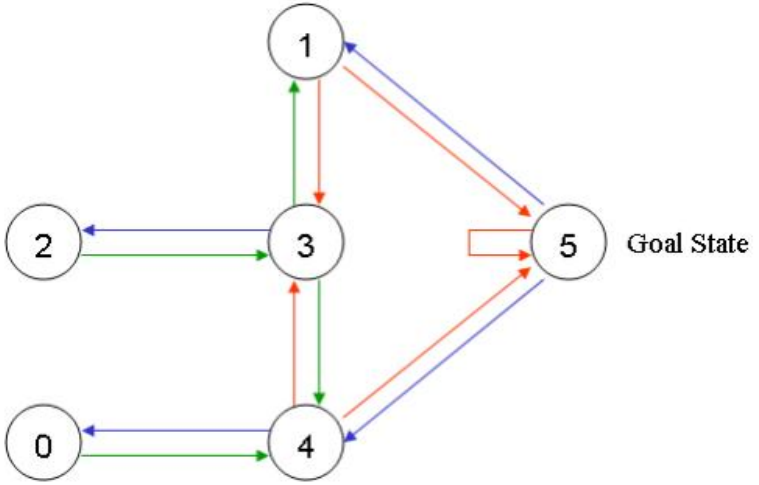
示例

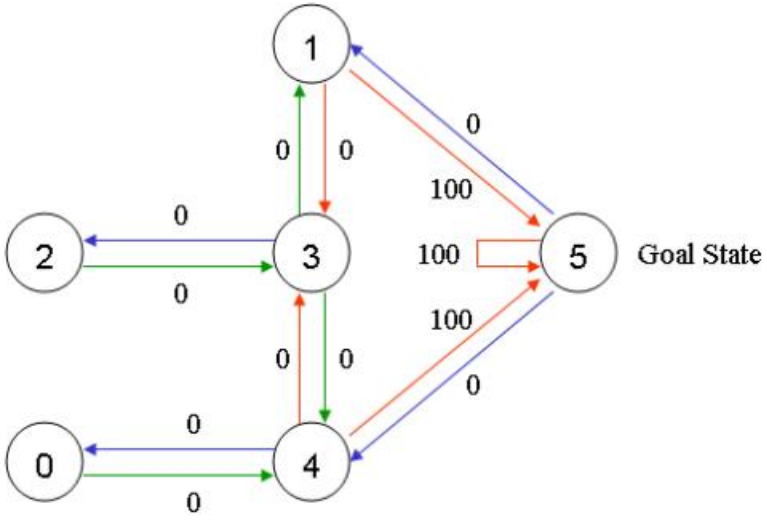


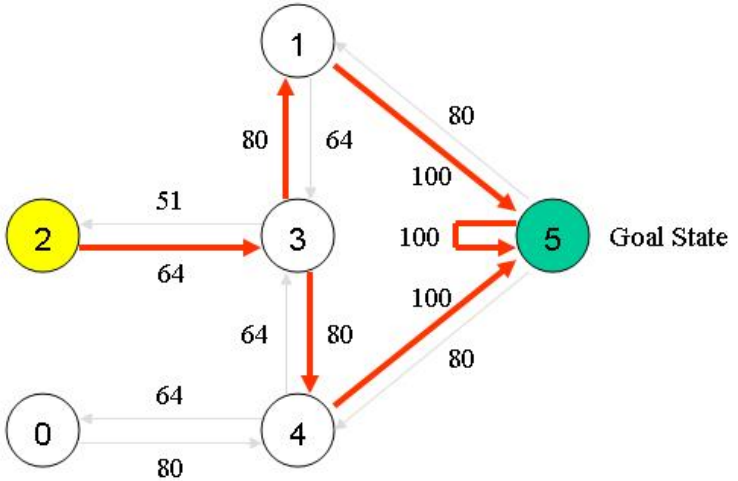
我们可能在任意一间房间中放置一个智能体（机器人），并期望该智能体能够从该房间开始走出这栋楼（可以认为是我们的目标房间）。换句话说，智能体的目的地是房间5。

示例

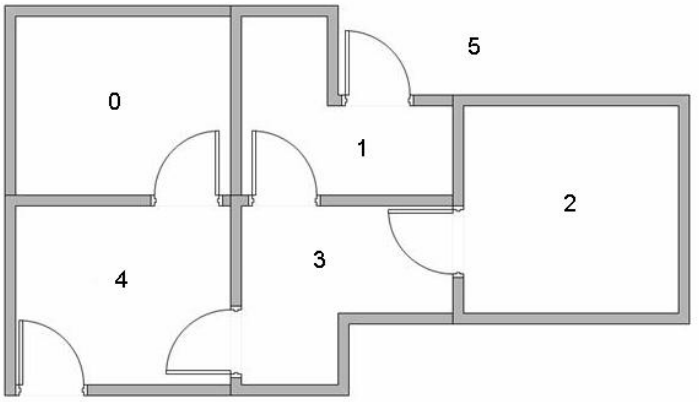

0
1
2
3
4
5


1
2
3
5
Goal State
0
4


1
0
0
0
100
2
3
100
5
Goal State
0
0
100
0
0
0
0
0
0
4

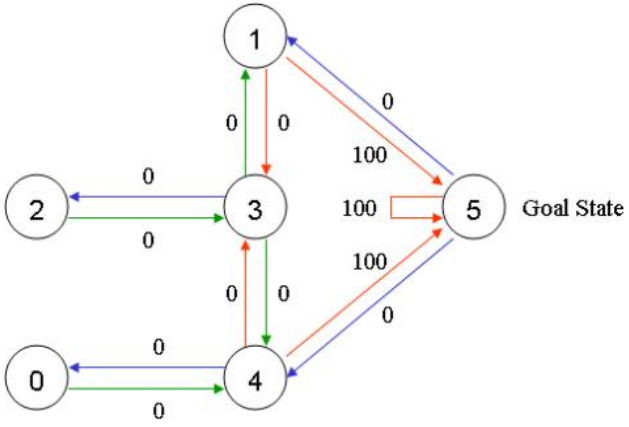

1
80
64
80
100
51
2
3
100
5
Goal State
64
100
64
80
80
64
0
4
80

示例



奖励表（回报表）

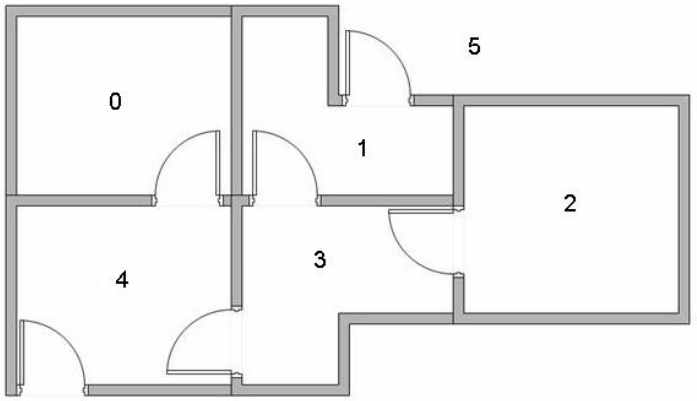
$$R = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \text{Action} \\ \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array}$$



现在，我们再增加一个相似的矩阵Q，代表智能体从经验中所学到的知识。矩阵Q的行代表智能体当前的状态，列代表到达下一个状态的可能的动作。

因为智能体在最初对环境一无所知，因此矩阵Q被初始化为0。在这个例子中，为了阐述方便，我们假设状态的数量是已知的（设为6）。如果我们不知道有多少状态时，矩阵Q在最初被设为只有一个元素。如果新的状态一旦被发现，对矩阵Q增加新的行和新的列非常简单。

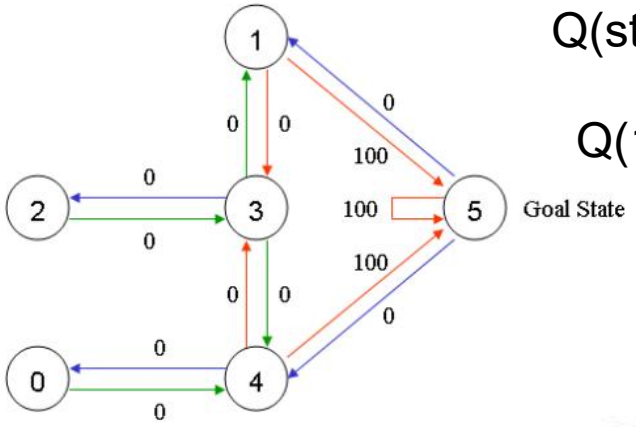
示例



Action

State	0	1	2	3	4	5
0	-1	-1	-1	-1	0	-1
1	-1	-1	-1	0	-1	100
2	-1	-1	-1	0	-1	-1
3	-1	0	0	-1	0	-1
4	0	-1	-1	0	-1	100
5	-1	0	-1	-1	0	100

$R=$



Gamma=0.8

Q(state, action) = R(state, action) + Gamma * Max[Q(next state, all actions)]

Q(state, action)=R(state, action) + Gamma * Max[Q(next state, all actions])

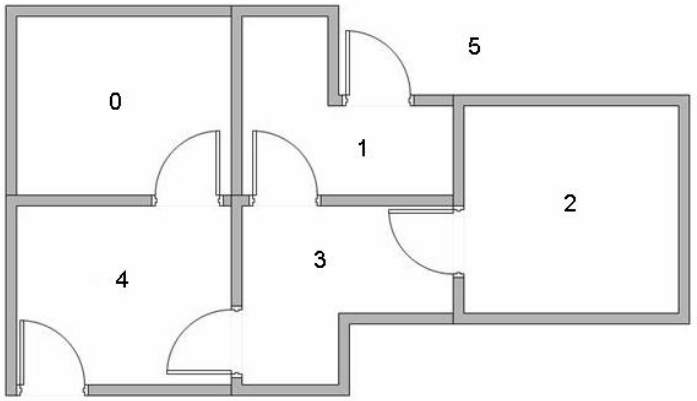
Q(1, 5) = R(1, 5) + 0.8 * Max[Q(5, 1), Q(5, 4), Q(5, 5)] = 100 + 0.8 * 0 = 100

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	0	100
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0

$Q=$

矩阵Q中的一个元素值就等于矩阵R中相应元素的值与学习变量Gamma乘以到达下一个状态的所有可能动作的最大奖励值的总和。

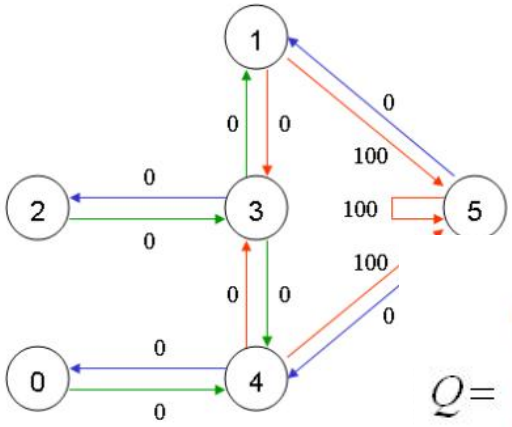
示例



Action

State	0	1	2	3	4	5
0	−1	−1	−1	−1	0	−1
1	−1	−1	−1	0	−1	100
2	−1	−1	−1	0	−1	−1
3	−1	0	0	−1	0	−1
4	0	−1	−1	0	−1	100
5	−1	0	−1	−1	0	100

$R=$



Q(state, action) = R(state, action) + Gamma * Max[Q(next state, all actions)]

Q(3, 1) = R(3, 1) + 0.8 * Max[Q(1, 3), Q(1, 5)] = 0 + 0.8 * Max(0, 100) = 80

$Q=$

	0	1	2	3	4	5
0	0	0	0	0	80	0
1	0	0	0	64	0	100
2	0	0	0	64	0	0
3	0	80	51	0	80	0
4	64	0	0	64	0	100
5	0	80	0	0	80	100

Gamma=0.8

矩阵Q中的一个元素值就等于矩阵R中相应元素的值与学习变量Gamma乘以到达下一个状态的所有可能动作的最大奖励值的总和。

示例

Q-学习算法的计算过程如下：

1、设置参数Gamma，以及矩阵R中的环境奖励值；

2、初始化Q矩

3、对每一次经

随机选择

Do while

对当前

中，选择一个

使用这

一个状态；

对下一

能的动作，获

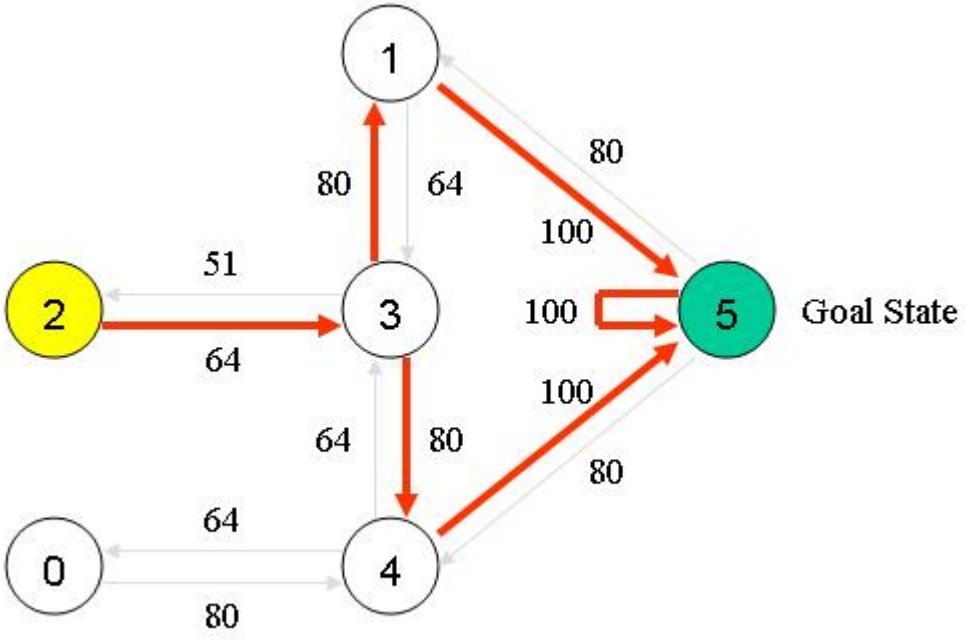
计算：

action)=R(state, action) + Gamma

Max(Q[next state, all actions])

设置下一个状态作为当前状态；

End For



为了使用矩阵Q，智能体仅仅简单地跟踪从起始状态到目标状态的状态序

阵Q中，为当前状

奖励值的动作。

如下：

=初始状态；

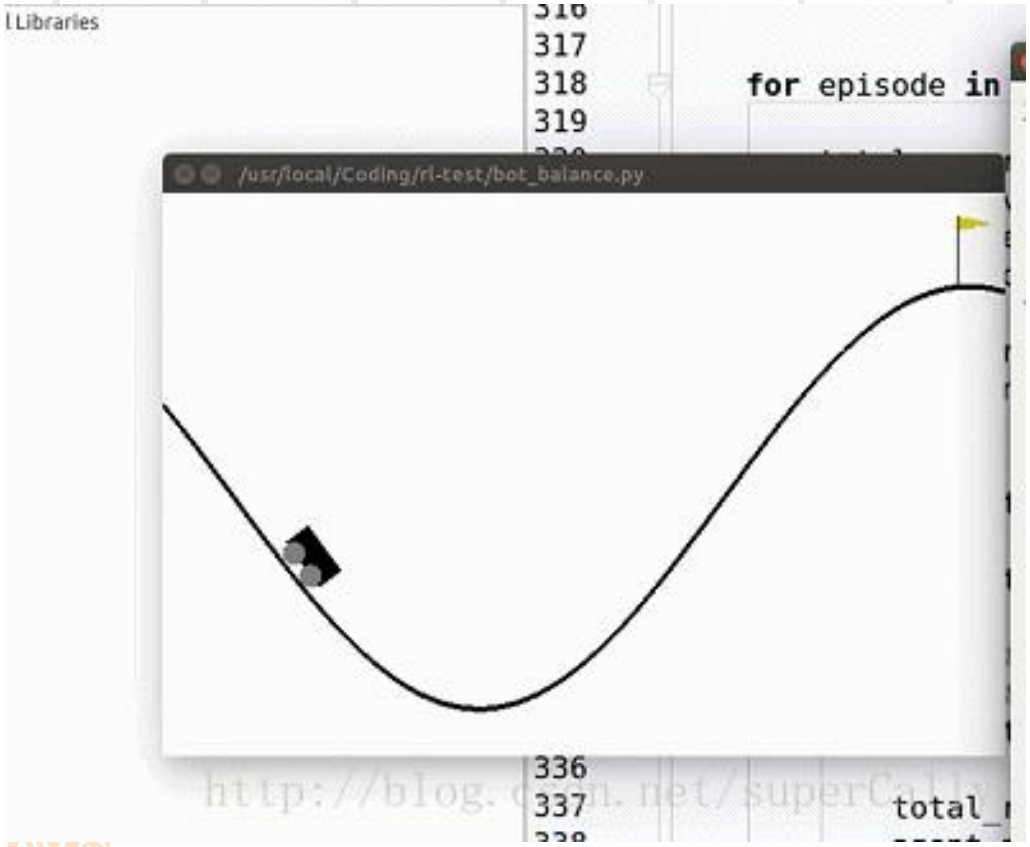
始，寻找具有最高

=下一个状态；

，直到当前状态=

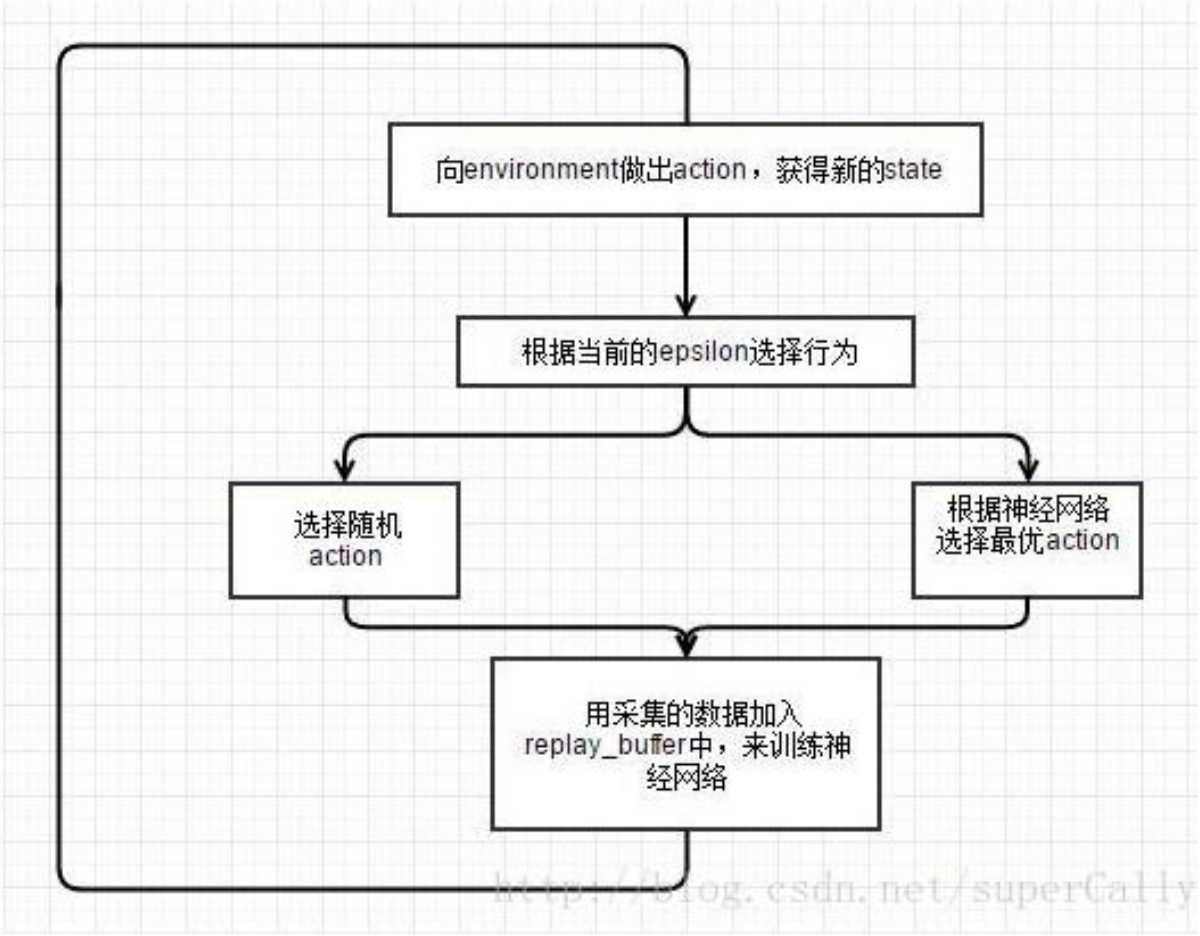
上述的算法将返回从初始状态到目标状态的状态序列。

示例（动作和状态是连续的）



小车有两个动作，在任何一个时刻可以向左运动，也可以向右运动，我们的目标是上小车走上山顶。一开始小车只能随机地左右运动，在训练了一段时间之后就可以很好地完成我们设定的目标了

Deep Q-Learning



Deep Q-Learning

1. replay_buffer（采集数据）

在不断地在系统中训练的过程中，会产生大量的训练数据。虽然这些数据并不是应对当时环境最优的策略，但是是通过与环境交互得到的经验，这对于训练系统是有非常大的帮助的。所以设置一个replay_buffer，获得新的交互数据，抛弃旧的数据，并且每次从这个replay_buffer中随机取一个batch，来训练系统。

replay_buffer中的每一条记录包含这几项：

state，表示当时系统所面临的状态

action，表示我们的agent面临系统的状态时所做的行为

reward，表示agent做出了选择的行为之后从环境中获得的收益

next_state，表示agent做出了选择的行为，系统转移到的另外一个状态

done，表示这个epsiode有没有结束

这个状态集来训练最后的神经网络

Deep Q-Learning

- **2. 神经网络**
- 因为对于某一个时刻系统的状态，我们需要估算在这个状态下，我们采取状态集S当中的每一个动作，大概会产生多大的收益
- 然后我们就可以根据我们既定的策略，在比较了收益之后，选一个动作
- 神经网络的输入，是系统的一个状态，state
- 神经网络的输出，是状态集当中的每一个动作，在当前状态下，会产生的价值
- 输入是系统给定的，输出是我们估算出来的，我们用估算的这个输出，来替代之前的输出，一步步地进行优化
- 有了这些数据，我们就可以对神经网络来做优化了
- 但是我们拿到了每个动作的价值之后，该采取怎样的策略呢？在基本的Q-Learning算法中，我们采取最最简单的epsilon-greedy策略

Deep Q-Learning

- **3. epsilon_greedy**
- 这个策略虽然简单，但是十分的有效，甚至比很多复杂的策略效果还要好
- 设置一个阈值,epsilon-boundary，比如说初始值是0.8，意思就是现在选择action的时候，80%的可能性是随机地从动作集中选择一个动作，20%的可能性是通过神经网络计算每个动作的收益，然后选最大的那一个
- 但是随着学习过程推进，我们的epsilon-boundary要越来越低，随机选择的次数要越来越少，到最后几乎不做随机的选择。

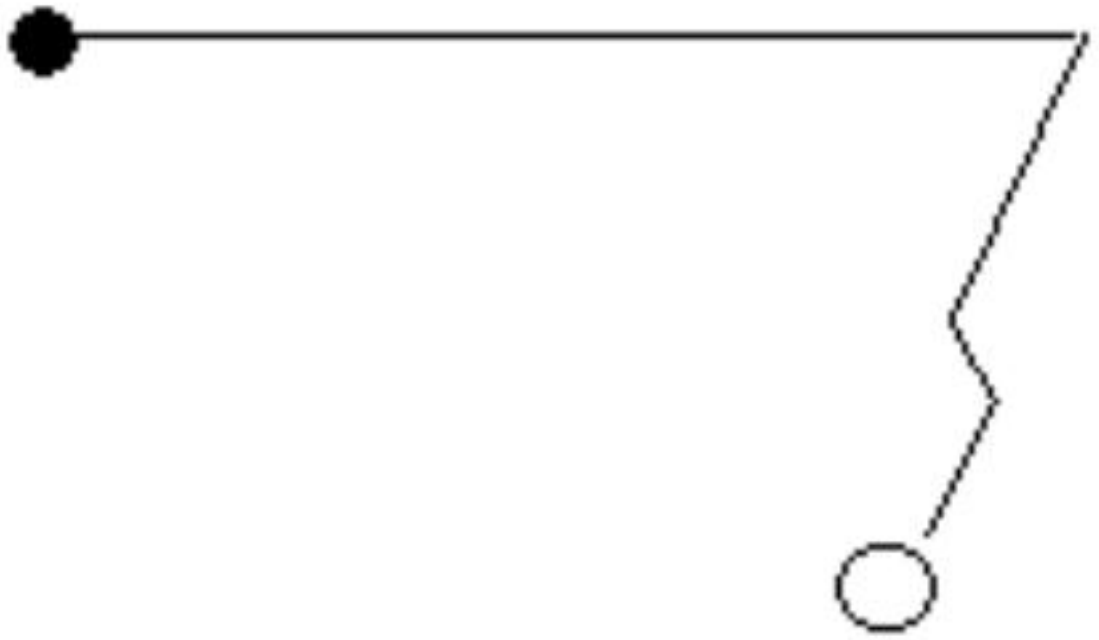
强化学习进行截球

- 在使用强化学习的时候关键是确定状态空间、动作空间、目标状态、策略函数Q（代价函数）以及价值函数V。

选择Q学习进行截球

- 首先是确定状态空间(s)，也就是world state，一般状态空间都很大，在计算和存储方面就存在很多困难，这也是目前强化学习往机器人足球中应用的难点地方；这就需要进行简化和处理。然后就是确定动作集，一般把原子动作作为动作集。目标状态是停止学习的终止条件，在学习的时候一般把得到球作为目标状态。所谓策略函数就是在当前的状态在选择动作的函数，这样的函数学要自己去设计，原则是能够把代价最小、利益最大的动作选择出来。而价值函数是在选择一个特定动作以后，是成功还是失败，相应的对这个状态下的这个动作的代价（利益）进行相应的修正，一般是加上（成功）、减去（失败）一个值，经过足够长的时间的学习就能达到一个稳态（也就是价值函数的性能较好）。

强化学习示意图



实例

- 动作集{dash,turn};
- 状态空间S={S1,….Sn};
- 一张Q表用来存储动作到状态的策略函数Q（代价函数）
- 目标状态（球在控球范围之内）
- 成功达到目标状态就进行效益分配到动作序列中，更新Q表。
- 如此迭代足够多次以后Q表仅能达到稳态。

6.3.2 传球(PASS)

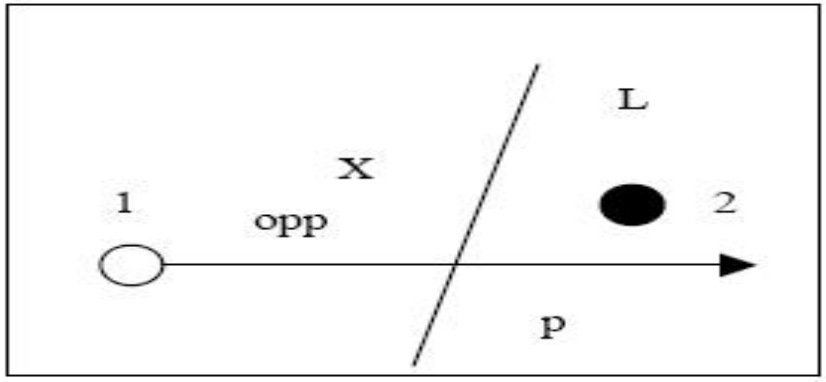
1、问题描述：

- 传球比截球复杂
- 传球方式：传给某一特定的人和传到某一点。
- 对传球进行描述的时候，可以采用这种方式：描述传球队员的周围环境，用状态S表示周围的环境或提取环境的一些特征属性向量A（a1,a2,…,an）。根据这些S或A来选择合适的传球方向和出球速度。

方法1：基于神经网络(BP)的计算学习

- 计算学习一直是机器学习的重要研究内容，它主要是通过计算的方法将那些错的很离谱的假设排除出去，通过计算机的快速计算能力得出最有可能的假设并把该假设认为是可能近似正确（probably approximately correct，PAC）。
- 基于神经网络集成的计算学习算法。以神经网络集成作为计算学习的前端，首先利用其产生计算学习所用的数据集，在产生数据集时，采用能够较好地反映神经网络集成性能的数据生成方式，使得用于计算学习的示例能够受益于神经网络集成的强泛化能力，以最终获得较高的预测精度。

基于神经网络(BP)的计算学习



- 如图，假设白圆圈表示的1号队员要把球传给用黑圆圈表示的2号队员，X表示对手。线L为对手和截球队员的垂直平分线。
- 对于垂直平分线与球轨迹的交点p以内的点，对方队员能比我方队员能先跑到；反之，交点以外的点，我方队员先跑到。
- 如果传球队员踢出速度大小合适的球，使得对手在交点以内都无法截到球，那么我方队员就必然可以比对方先截到球。如果以此速度踢出球，此队员不能在该点以前截到球，而且以小于此速度的任何速度踢出球，对手都可能在该点以内截到球,那么这个速度称为对于某个队员穿越在球运动轨迹上的某一点的穿越速度。

穿越速度

- 注意到这个分析基于上图队友在对手后面的情况。如果反过来，队友在对手前，则传球者应该以小于队友的穿越速度的速度传球，以保证队友在交点以前截到球。对上图的情景，我们把p点以前的区域称为对手的接球区域，p点以后的称为队友的接球区域。对于队友，穿越对手的穿越速度为传球给他的速度的下限。如果考虑队友后面可能有一个对手，则给他一个传球速度上限的限制。
- 上图考虑了一个队友和一个对手的简单传球场景，多个对手和队友的场景也有类似的分析。在一条传球线路上，每个队员（包括对手和队友）或者没有接球区域，或者有一个接球的区域和一个传球速度的上、下限。

学习算法

- 首先，我们利用人工神经网络中的BP网络，训练得到在特定传球路线上面传给每个球员的穿越速度。 第一步，采集样本。确定传球队员的位置和随机置接球队员的初始位置。在训练中传球者从一个较小的速度开始，沿传球线路传球，接球者利用训练好的截球技能进行截球，如果截球点在上图的p点以前，则传球者提高速度，继续尝试；否则，穿越速度为该次训练的传球速度。如此这样收集传球队员和接球之间不同的距离和角度情况下的穿越速度。
- 第二步，用人工神经网络中的BP网络拟合得到传给每个球员的穿越速度。其中输入是传球队员和接球队员的距离和传球路线的方向和传接球队员之间连线的夹角，输出是穿越速度。

学习算法

- 利用BP得出每个球员在本方传球时自己能够接球的穿越速度作为我们计算学习的基础。如果传球到一点上面，那传球路线就确定了，我们只要计算用什么样的穿越速度就可以了。如果是传给特定的球员，可以根据穿越速度淘汰掉那些接球队员没有接球区域的传球路线，选择接球队员的穿越速度区间最大的那条传球路线（主要是增强系统的抗噪音性）作为我们的目标传球路线。计算学习要学习的就是在给定了场景（主要记录的是传球队员和场上所友队员的相对位置和角度）的情况下得出最佳的传球路线。

强化学习

- 强化学习的基本思想见截球
- 传球时强化学习主要是解决状态空间庞大的问题。
- 其它方面的实现都比较简单。

6.3.3 FastKick

- 1.问题描述
  - 在Soccer Server中，队员的身体和球都使用一个圆来表示，且相互之间的位置不允许有相互重叠的部分。当球离队员的距离小于某个值时，这时队员就可以向Server发一个包括角度和力量两个参数的kick命令，对球施加一个矢量加速度。由于球的加速度有上限，且球有初始速度，因此常常无法通过一个kick命令把球加速到所希望的速度上面去，也就是一个踢球动作需要一系列周期的kick命令才能实现，这就需要Fastkick。

6.2.3 FastKick

- **2.解决方法**
- 1）直接经验式代码：将设计者的经验直接写成代码进行踢球决策。
- 2）Case-base Learning：在这种方法中，控球范围被离散成为一些点的集合，每周期的状态用4个参数来描述（球员的速度、球的相对位置、期望的出球速度、可以到达的点的集合）。然后构造了若干个Case Bases，每个Case Bases都能根据输入状态来返回一个PDL，这个PDL描述了每个可到达的点作为中间点的好坏。
- 3）强化学习：更前面提到的截球的相似，主要是通过学习提高价值函数的性能。
- 4） 清华提出的考虑对抗的强化学习—Q学习及在线规划。在这当中，用一张Q表来存储状态-动作对，再把一些状态（如这时球也对手的范围之内或球出界）屏蔽掉。然后进行训练得出实战时对抗性能更强的Q表。

6.2.4射门(shoot)

- 简单射门就是进行射门路线的选择

方法：

1、手工方法：离散射门路线，遍历

2、BP网络

3、决策树

4、强化学习（可以进行区域配合）