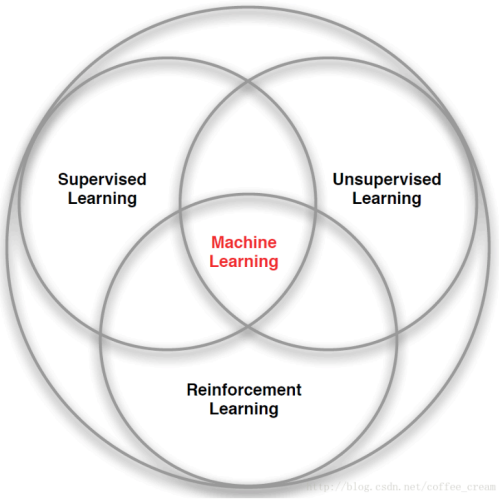
算法篇

> 本项目主要是将深度强化学习（RL）中的MADDPG——混合竞争合作环境下的多智体评论家算法接入到暴雪公司和Deepmind发布的针对*星际争霸2*的SC2LE环境下。本wiki介绍了该项目用到的基础算法——MADDPG算法，以及其在RL中的算法基础。

强化学习（RL）

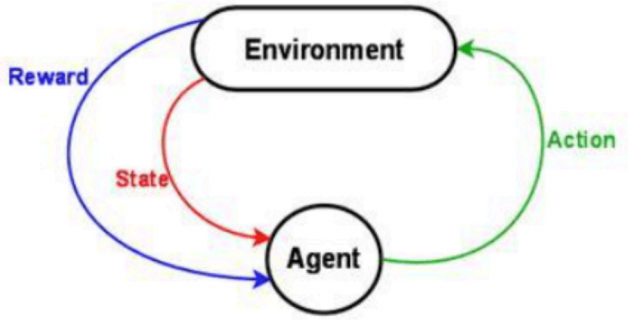
强化学习是机器学习区别于监督式学习和无监督式学习的一种新的学习方式。其他机器学习算法中学习器都是学怎么做，而在强化学习中，是在尝试的过程中学习到在特定的情境下选择哪种行动可以得到最大的回报。在很多场景中，当前的行动不仅会影响当前的回报，还会影响之后的状态和一系列的回报。



RL与监督学习、无监督学习的比较：

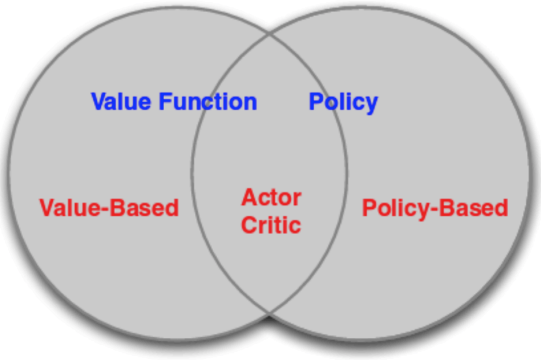
- 有监督的学习的训练集中，每个样本的观测值都已经有了标签（label）。进行学习，训练集中每一个样本的特征可以视为是对该状态（state）的描述，而标签（label）可以视为是应该执行的正确的决策或动作（action）。一个简单的例子就是通过一些已有标签（label）的肿瘤数据中，根据肿瘤的大小、形状等特征（features）判读肿瘤是良性还是恶性的。但是有监督的学习不能学习交互的情景，因为在交互的问题中获得期望行为的样例是非常不实际的，智能体（agent）只能从自己的经历（experiment）中进行学习，而经历（experiment）中采取的行为并不一定是最优的。这时利用强化学习（RL）就非常合适，因为强化学习（RL）不是利用正确的行为来指导，而是利用已有的训练信息来对行为进行评价。
- 无监督学习是指在缺乏足够的先验知识的情况下，让计算机能代我们完成这些工作，或至少提供一些帮助。根据类别未知(没有被标记)的训练样本解决模式识别中的各种问题，称之为无监督学习。一个例子就是鸡尾酒聚会中背景音乐和说话声音的提取，因为事先并不知道两种声音，没有标签（label）。因为强化学习（RL）利用的并不是采取正确行动的经历（experiment），从这一点来看和无监督的学习确实有点像。但无监督的学习的目的可以说是从一堆未标记样本中发现隐藏的结构，而强化学习（RL）的目的是最大化回报（reward）。
- 总的来说，强化学习（RL）与其他机器学习算法不同的地方在于：其中没有监督者，只有一个回报（reward）信号；反馈是延迟的，不是立即生成的；时间在强化学习（RL）中具有重要的意义；智能体（agent）的行为会影响

之后一系列的回报（reward）。一个很容易理解的强化学习的例子如前一段时间风靡一时的Alpha Go。这个机器下围棋的时候，机器为了赢得比赛需要在每一次做出决策，是选择往哪里走，然后得到的反馈并不是像监督学习一样的对或者错，这个反馈跟最后能否赢得比赛关系在现在看还不明朗。这需要机器去不断重复学习，然后改进下棋过程。



强化学习基本算法

Value-Based（或Q-Learning）基于价值的方法和Policy-Based（或Policy Gradients）基于策略的方法是强化学习中最重要的两类方法。Value-Based是预测某个状态（state）下所有策略或动作（action）的期望价值——Q值，之后通过选择最大Q值对应的策略或动作（action）执行策略，适合仅有**少量离散取值**的策略或动作（action）的环境。Policy-Based是直接预测某个状态（state）下应该采取的策略或动作（action），适合**高维连续**策略或动作（action）的环境，更通用。



强化学习的主要挑战

正如上文所言，在监督学习中，每个训练示例都有一个目标标签（label）；在非监督学习中，则没有任何标签（label）；而在强化学习中，则有**稀疏的、延时的标签**（label）——奖励（reward）。仅仅基于这些奖励（reward），代理人必须学会在环境中策略和行动（action）。

强化学习的巨大挑战主要体现在以下两方面：

- **信贷分配问题（credit assignment problem）**：有时候一瞬间得到的奖励（reward）来自一个长期的策略——就像是我们通过一学期的努力得到了期末的成绩一样，如何把得到的最终奖励（reward）分配给之前的策略和行为（action）？
- **探索和利用困境（explore-exploit dilemma）**：当我们已经有了一定的经验（experimence），这时候我们就面临着一个种权衡——是继续坚持之前的策略还是进行一下探索?

信贷分配问题（credit assignment problem）的解决办法往往是通过贴现未来回报（discounted future reward）的计算来实现的：
$R_t = r_t + \gamma(r_{t+1} + \gamma(r_{t+2} + \ldots)) = r_t + \gamma R_{t+1}$。其中贴现参数γ应该根据环境的决定性来设定。

探索和利用困境（explore-exploit dilemma）往往是通过贪心ε算法（ε-greedy exploration）处理的和经验重放（experiment replay）。贪心ε算法大致就是进行决策时，生成一个ε概率的随机方向，用来探索。而经验重发的策略不是选取最近的一批来重放，为了避免达到局部最优，经验重放选择了一小批随机经验进行重放。

深度Q学习算法

基于价值的方法——Q函数

Q函数定义为在t时刻、s_t状态下，我们执行了动作a_t的最大贴现未来回报（maximum discounted future reward）:$Q(s_t, a_t) = \max R_{t+1}$。可以直观理解Q代表的含义成，在t时刻、$s_t$状态下，我们执行了动作$a_t$之后，游戏结束时刻我们将要得到的最好分数。**尽管我们无法在s_t状态下就得到Q函数的具体值，我们之后可以通过其他方法求得——这有点像隐性函数的求解。**考虑一个状态转化$< s, a, r, s' >$，我们可以表示Q函数为形似Bellman方程的形式：
$Q(s, a) = r + Q(s', a')$。这说明我们可以通过迭代进行求解。

Q值的储存

通常把Q函数的值储存为一个矩阵：

	a_1	a_2	a_3
s_1	Q(1,1)	Q(1,2)	Q(1,3)

s_2	Q(2,1)	Q(2,2)	Q(2,3)
s_3	Q(3,1)	Q(3,2)	Q(3,3)

虽然这样表示十分直观方便，但是考虑到星际争霸游戏环境下状态和动作的种类之多，用矩阵表示显然是耗费计算力的。

深度Q神经网络

为了探索Q函数方法的更广法应用，我们引入了深度Q神经网络（DQN）的概念。神经网络特别擅长为高度结构化的数据提供良好的特性。我们可以用神经网络来表示Q函数，它以状态（四个游戏屏幕）和动作作为输入和输出相应的Q值。

DQN的损失函数是$L = \frac{1}{2}(r + Q(s'|a') - Q(s|a))^2$。

具体操作是：

1. 对当前状态s进行前馈传递，得到所有动作的预测值
2. 对下一个状态s进行前馈传递，并计算最大网络输出$max_{a'}Q(s', a')$
3. 将目标动作的Q值设置为$r + \gamma Q(s'|a')$；对于所有其他操作，设置Q值为1.得到的预测值，返回这些输出的误差为0
4. 使用反向传播更新权值

伪代码

Q学习过程的伪代码如下：

> % 初始化
> 初始化经验重放内存D
> 随机初始化状态$\forall s \in S$、动作$\forall a \in A$的Q(s,a)值
> 初始化状态s
> while（每一个轮次）
> ··% 选择执行行动
> ··选择并执行Q值最大的动作a（以ε的概率选择一个随机策略）
> ··观察奖励r和之后的状态s'
> ··把$< s, a, r, s' >$存储到经验重放内存中
> ··% 训练DQN网络
> ··从经验重放内存D中，随机选取一批状态转化$< ss, aa, rr, ss' >$
> ··计算每一小批的目标Q值

> ····如果ss'是终点状态，tt=rr
> ····否则，$tt = rr + \gamma max_{a'} Q(ss'|aa')$
> ··以损失$(r + Q(s'|a') - Q(s|a))^2$来训练深度Q网络
> ··s=s'
> end

策略梯度算法

基于策略的方法——策略梯度

基于值的方法一般是确定性的，给定一个状态就能计算出每种可能动作的奖励（确定值），但这种**确定性的方法恰恰无法处理一些现实的问题**，比如玩100把石头剪刀布的游戏，最好的解法是随机的使用石头、剪刀和布并尽量保证这三种手势出现的概率一样，因为任何一种手势的概率高于其他手势都会被对手注意到并使用相应的手势赢得游戏。

另外，**过多的状态数量也是使用基于值的方法的一个限制因素**，因为基于值的方法需要保存状态-动作的对应关系，因此很多现实问题（例如机器人控制和自动驾驶都是连续动作空间）都因为巨量的状态而无法计算。

策略梯度利用随机（stochastic）解决上面的两个问题产生的，**它能提供服从某种概率分布的随机非确定的结果**，策略梯度不计算奖励而是使用概率选择动作，这样就避免了因为计算奖励而维护状态表。策略梯度的基本原理是通过反馈调整策略，具体来说就是在得到正向奖励时，增加相应的动作的概率；得到负向的奖励时，降低相应动作的概率。用一个概率分布函数$\pi_\theta(s_t|\theta^\pi)$找到每一步的最优策略，其中$\theta^\pi$为策略函数的参数。

策略梯度学习（PG）

利用策略梯度的方法，目标就是训练出参数θ^π，让奖励的期望值最大：
$J(\theta) = argmax_\theta E[r_1 + r_1 + ... + r_t|\pi_\theta] = E_{r\,\pi_\theta(t)}[\sum_t r_t] = \int_t r(t)\pi_\theta(t)dt$。

采用梯度上升的方法，对参数θ求偏导：
$\nabla_\theta J(\theta) = \nabla_\theta \int_t r(t)\pi_\theta(t)dt = \int_t r(t)\nabla_\theta \pi_\theta(t)dt$。

根据偏导数性质$\nabla log f(x) = \frac{\nabla f(x)}{f(x)}$，
$\nabla_\theta \pi_\theta(t) = \pi_\theta(t)\frac{\nabla_\theta \pi_\theta(t)}{\pi_\theta(t)} = \pi_\theta(t)\nabla_\theta log\pi_\theta(t)$。

所以

$$\nabla_\theta J(\theta) = \int_t r(t)\nabla_\theta \pi_\theta(t)dt = \int_t r(t)\pi_\theta(t)\nabla_\theta log\pi_\theta(t)dt = E_{r\,\pi_\theta(t)}[\nabla_\theta log\pi_\theta(t)r(t)]$$
$$= E_{r\,\pi_\theta(t)}[(\sum_{t=1}^{T} log\pi_\theta(a_t|s_t))(\sum_{t=1}^{T} r(s_t, a_t))]。$$

由于策略产生的是非确定的动作，因此相同策略在多轮次中会产生不同的轨迹，为了避免个体的偏差，我们需要多次取样并取均值来提高准确性，所以，

$$\nabla_\theta J(\theta) = \frac{1}{N}\sum_{i=1}^{N}[(\sum_{t=1}^{T} log\pi_\theta(a_{i,t}|s_{i,t}))(\sum_{t=1}^{T} r(s_{i,t}, a_{i,t}))]。$$

因此，我们就得到了可计算的目标函数的导数，在轮次的反向传播 (back propagation) 中使用学习率α与$\nabla_\theta J(\theta)$的乘积作为差值更新：

$$\theta := \theta + \alpha\nabla_\theta J(\theta)$$

确定性策略梯度学习（DPG）

确定性的行为策略，每一步的行为通过函数μ直接获得确定的值：

$a_t = \mu(s_t|\theta^\mu)$，函数μ即最优行为策略，不再是一个需要采样的随机策略。确定性策略梯度学习节省了最优策略概率分布进行采样时，获得行为需要耗费的计算力。

深度确定性梯度下降（DDPG）

DDPG算法融合了Q函数和策略梯度的基础思想，采用卷积神经网络作为策略函数μ和回报函数Q
的模拟，即策略μ网络和价值Q网络；然后使用深度学习的方法来训练上述神经网络，训练的目标是使Q网络的L尽可能小、使μ网络的J尽可能大。

Q网络和策略网络

如果只使用单个神经网络的算法，学习过程很不稳定，因为Q网络的参数在频繁的梯度变换的同时，又用于计算Q网络和μ网络的梯度。基于此，DDPG分别为μ网络、Q网络各创建两个神经网络拷贝,一个叫做online，一个叫做target。

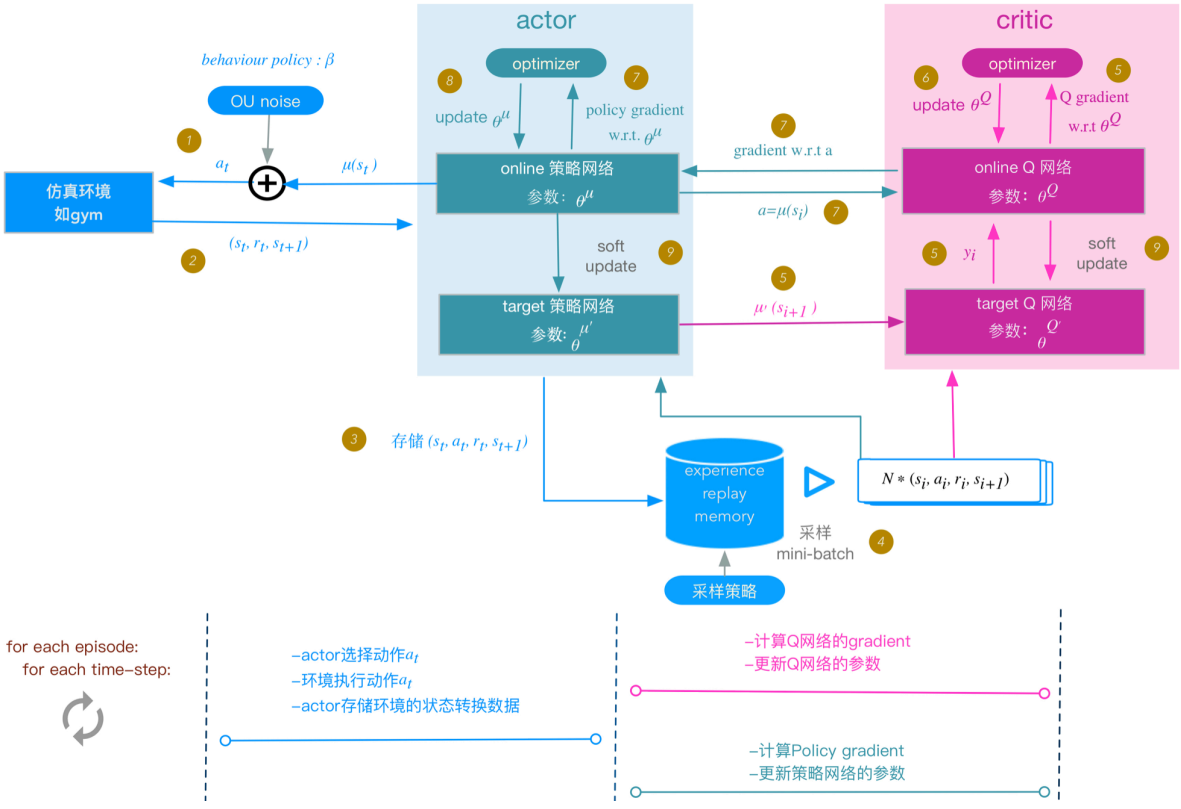
$$\text{策略网络}\begin{cases} online:\mu(s|\theta^{\mu}):gradient更新\theta^{\mu} \\ target:\mu'(s|\theta^{\mu'}):soft\ update\ \theta^{\mu'} \end{cases}$$

$$Q网络\quad\begin{cases} online:Q(s,a|\theta^{Q}):gradient更新\theta^{Q} \\ target:Q'(s,a|\theta^{Q'}):soft\ update\ \theta^{Q'} \end{cases}$$

$$soft\ update:\begin{cases} \theta^{Q'}\leftarrow\tau\theta^{Q}+(1-\tau)\theta^{Q'} \\ \theta^{\mu'}\leftarrow\tau\theta^{\mu}+(1-\tau)\theta^{\mu'} \end{cases}$$
τ一般取值0.001

优点在于target网络参数变化小，用于在训练过程中计算online网络的梯度，比较稳定，训练易于收敛。代价是参数变化小，学习过程变慢。

DDPG图解



DDPG伪代码

> % 初始化
> 初始化经验重放内存R
> 随机初始化Qonline网络和策略online网络的参数θ^{μ}和θ^{Q}，其中μ网络作为critic网络，Q网络作为actor网络

> 分别复制两个网络（参数复制），生成target网络
> while（每一轮次）
> ··% 选择执行行动
> ··UO初始化噪音$\mathcal{N}$，用于探索策略
> ··初始化状态s_t
> ··while（轮次中的每一刻）
> ····根据策略μ网络和策略探索选择和执行策略
> ····观察奖励r和之后的状态s_{t+1}
> ····把$< s_t, a_t, r_t, s_{t+1} >$存储到经验重放内存中
> ····% 训练网络
> ····从经验重放内存D中，随机选取一批N个状态转化$< s_i, a_i, r_i, s_{i+1} >$
> ····设置$y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
> ····**更新Q网络**，梯度下降损失函数$L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
> ····**更新策略网络**，梯度上升函数
> $\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla\theta^\mu \mu(s|\theta^\mu)|_{s_i}$
> ····软更新两个target网络
> ··结束
> 结束

多智体深度确定性梯度下降（MADDPG）

为了将强化学习的概念应用更具有挑战性的情景下，如多个机器人控制、语言交流、多玩家游戏等多智体竞争与合作问题。在这个项目中，我们采用了Deepmind发布的MADDPG算法，用于该星际争霸2环境一个特定场景的训练。

传统的算法用于多智能体环境下具有以下困难：

- Q-learning会受到环境不稳定性的挑战：每个智能体都在变化，而且每个智能体的角度来看，环境都会变得不稳定。
- 策略梯度（PG）方法在智能体数目增多时，会有variance变大的问题。

DDPG结合了Q学习算法和Actor-Critic的思想，MADDPG沿用了这个思想。

多智体Actor-Critic

我们通过采用分散执行，集中训练的框架来实现我们的目标。若N个智能体（actor）的策略集合分别是$\pi = \pi_1, \pi_2 ... \pi_N$，策略参数分别为

$\theta = \theta_1, \theta_2 \ldots \theta_N$，那么智能体i的策略梯度为：

$$\nabla_{\theta_j} = J(\theta_i) = E_{s\, p^{\mu}, a_i\, \pi_i}[\nabla_{\theta_i} log\pi_i(a_i|o_i) Q_i^{\pi}(X, a_1, a_2 \ldots a_N)]$$

$Q_i^{\pi}(X, a_1, a_2 \ldots a_N)]$是一个是一个集中的动作值函数，它将所有智能体的动作$a_1, a_2 \ldots a_N$和状态X（包含每个智能体的观测值和一些附加状态信息）作为输入，然后输出Q值。每个智能体的Q值是分开的，智能体有不同的奖励机制。

扩展到确定性策略，考虑N个多智体的策略μ_{θ_i}，参数为θ_i（缩写为μ_i），那么梯度为：

$$\nabla_{\theta_i} = J(\mu_i) = E_{x,a\, D}[\nabla_{\theta_i} \mu_i(a_i|o_i) \nabla_{a_i} Q_i^{\mu}(x, a_1, \ldots, a_N)|_{a_i=\mu_i(o_i)}]$$

经验重放缓冲区D包括元组$(x, x', a_1, \ldots, a_N, r_1, \ldots r_N)$，记录了所有智能体的经验，集中动作值函数$Q_i^{\mu}$按如下方式更新：
$L(\theta_i) = E_{x,a,r,x'}[(Q_i^{\mu}(x, a_1, \ldots, a_N) - y)^2]$，
$y = r_i + \gamma Q_i^{\mu'}(x', a'_1, \ldots, a'_N)|_{a'_j=\mu'_j(o_j)}$，其中$\mu' = \{\mu_{\theta'_1}, \mu_{\theta'_2}, \ldots \mu_{\theta'_N}\}$是延迟参数的目标策略集合。

其他智能体的策略推测

为了推测其他智能体策略，每个智能体可以额外保有一个与智能体j的真实策略μ_j有关的近似值$\hat{\mu}_{\phi_i^j}$，这个近似策略通过最大化智能体j的动作对数概率加上一个熵正则化项来进行学习：$L(\phi_i^j) = -E_{o_j,a_j}[log\hat{\mu}_{\phi_i^j}(a_j, o_j) + \lambda H(\hat{\mu}_{\phi_i^j})]$，其中H是策略分布的熵。

那么，集中动作值函数Q_i^{μ}更新时，可以用
$\hat{y} = r_i + \lambda Q_i^{\mu'}(x', \widehat{\mu'}_i^1(o_1), \ldots \widehat{\mu'}_i^i(o_i), \ldots \widehat{\mu'}_N^1(o_N))$。其中用$\widehat{\mu'}_i^i(o_i)$表示策略$\hat{\mu}_{\phi_i^j}$的目标网络。

智能体的策略集成

多智能体强化学习中的一个反复的问题是由于智能体不断变化的策略而导致的环境非平稳性。竞争环境下尤其如此，因为智能体可以通过过度适应竞争对手的行为而获得很强的策略。这样的策略是不可取的，因为它们很脆弱，当竞争对手改变策略时就可能会失败。

为了获得对竞争智能体策略变化更鲁棒的多智能体策略，我们提出了对K个不同的子策略进行汇总。在每个回合的博弈中，我们为每个智能体随机地选择一个特

定的子策略执行。假设策略μ_i是K个子策略的集合，子策略K由$\mu_{\theta_i^{(k)}}$表示，对于智能体，我们的最大集成化目标是：$J_e(\mu_i) = E_{k\ unif(1,K),s\ p^{\mu},a\ \mu_i^{(k)}}[R_i(s,a)]$。

由于不同的子策略将在不同的博弈回合中执行，因此我们为智能体i的每个子策略$\mu_i^{(k)}$维护一个重播缓冲区$D_i(k)$。因此，我们可以导推出集成的目标值关于$\theta_i^{(k)}$的梯度：

$$\nabla_{\theta_i^{(k)}} J_e(\mu_i) = \frac{1}{K} E_{x,a\ D_i^{(k)}}[\nabla_{\theta_i^{(k)}} \mu_i^{(k)}(a_i|o_i) \nabla_{a_i} Q^{\mu_i}(x, a_1, a_2, \ldots, a_N)|_{a=\mu_i^{(k)}(o_i)}]$$

参考

前人工作

【1】星际争霸2人工智能研究环境SC2LE初体验
【2】迈向通用人工智能：星际争霸2人工智能研究环境SC2LE完全入门指南
【3】星际争霸2之环境配置
【4】星际争霸2之MADDPG算法

参考网页

【1】强化学习基本介绍
【2】DNQ介绍
【3】Intel——DNQ
【4】MADDPG论文翻译
【5】MADDPG官方文档
【6】MADDPG简介
【7】DDPG详解
【8】深度学习A3C

参考代码

【1】openai/MADDPG
【2】Blizzard/s2client-proto
【3】deepmind/pysc2
【4】fangbrodie/pysc2_maddpg