

Deep Reinforcement Learning - 1. DDPG原理和算法

2017年11月08日 14:23:24 kenneth_yu 阅读数: 15862

Deep Reinforcement Learning - 1. DDPG原理和算法

Deep Reinforcement Learning - 1 DDPG原理和算法

背景描述

DDPG的定义和应用场景

DDPG算法相关基本概念定义

DDPG实现框架和算法

DDPG对于DPG的关键改进

下一篇

以下用RL作为Reinforcement Learning 的简称。

背景描述

概括来说，RL要解决的问题是：让agent学习在一个环境中的如何行为动作(act)，从而获得最大的奖励值总和(total reward)。这个奖励值一般与agent定义的任务目标关联。

agent需要的主要学习内容：第一是行为策略(action policy)，第二是规划(planning)。

其中，行为策略的学习目标是最优策略，也就是使用这样的策略，可以让agent在特定环境中的行为获得最大的奖励值，从而实现其任务目标。

行为(action)可以简单分为：

- 连续的：如赛车游戏中的方向盘角度、油门、刹车控制信号，机器人的关节伺服电机控制信号。
- 离散的：如围棋、贪吃蛇游戏。Alpha Go就是一个典型的离散行为agent。

DDPG是针对连续行为的策略学习方法。

如果要了解完整和系统的RL背景知识，推荐大家看R.Sutton的这本书：《Reinforcement Learning: An Introduction, by Sutton, R.S. and Barto, A

DDPG的定义和应用场景

在RL领域，DDPG主要从：PG -> DPG -> DDPG 发展而来。

先复述一下相关的基本概念：

- s_t ：在t时刻，agent观察到的环境状态，比如观察到的环境图像，agent在环境中的位置、速度、机器人关节角度等；
- a_t ：在t时刻，agent选择的行为（action），通过环境执行后，环境状态由 s_t 转换为 s_{t+1} ；
- $r(s_t, a_t)$ 函数：环境在状态 s_t 执行行为 a_t 后，返回的单步奖励值；

上述关系可以用一个状态转换图来表示：



http://blog.csdn.net/kenneth_yu/article/details/78478356

- R_t ：是从当前状态直到将来某个状态，期间所有行为所获得奖励值的加权总和，即discounted future reward:

$$R_t = \sum_{i=t}^T \gamma^{i-t} r(s_i, a_i)$$

其中 γ 叫做discounted rate, $\in [0, 1]$, 通常取0.99.

PG

R.Sutton 在2000年提出的Policy Gradient 方法, 是RL中, 学习连续的行为控制策略的经典方法, 其提出的解决方案是:
通过一个概率分布函数 $\pi_{\theta}(s_t|\theta^{\pi})$, 来表示每一步的最优策略, 在每一步根据该概率分布进行action采样, 获得当前的最佳action取值; 即

$$a_t \sim \pi_{\theta}(s_t|\theta^{\pi})$$

生成action的过程, 本质上是一个随机过程; 最后学习到的策略, 也是一个随机策略(stochastic policy).

DPG

Deepmind的D.Silver等在2014年提出DPG: Deterministic Policy Gradient, 即确定性的行为策略, 每一步的行为通过函数 μ 直接获得确定性action:

$$a_t = \mu(s_t|\theta^{\mu})$$

这个函数 μ 即最优行为策略, 不再是一个需要采样的随机策略。

为何需要确定性的策略? 简单来说, PG方法有以下缺陷:

- 1. 即使通过PG学习得到了随机策略之后, 在每一步行为时, 我们还需要对得到的最优策略概率分布进行采样, 才能获得action的具体值; 而action通常是高维的向量, 在高维的action空间的频繁采样, 无疑是很耗费计算能力的;
- 2. 在PG的学习过程中, 每一步计算policy gradient都需要在整个action space进行积分:

$$\nabla_{\theta} = \int_S \int_A \rho(s) \pi_{\theta}(a|s) Q^{\pi}(s, a) da ds$$

(Q, ρ 参见下面DDPG部分的概念定义.)

这个积分我们一般通过Monte Carlo 采样来进行估算, 需要在高维的action空间进行采样, 耗费计算能力。

如果采取简单的Greedy策略, 即每一步求解 $argmax_a Q(s, a)$ 也不可行, 因为在连续的、高维度的action空间, 如果每一步都求全局最优解, 太耗!

在这之前, 业界普遍认为, 环境模型无关(model-free)的确定性策略是不存在的, 在2014年的DPG论文中, D.Silver等通过严密的数学推导, 证明了D其数学表示参见DDPG算法部分给出的公式 (3)。

然后将DPG算法融合进actor-critic框架, 结合Q-learning或者Gradient Q-learning这些传统的Q函数学习方法, 经过训练得到一个确定性的最优行为

DDPG

Deepmind在2016年提出DDPG, 全称是: Deep Deterministic Policy Gradient, 是将深度学习神经网络融合进DPG的策略学习方法。
相对于DPG的核心改进是: 采用卷积神经网络作为策略函数 μ 和Q函数的模拟, 即策略网络和Q网络; 然后使用深度学习的方法来训练上述神经网络。

Q函数的实现和训练方法, 采用了Deepmind 2015年发表的DQN方法, 即 Alpha Go使用的Q函数方法。

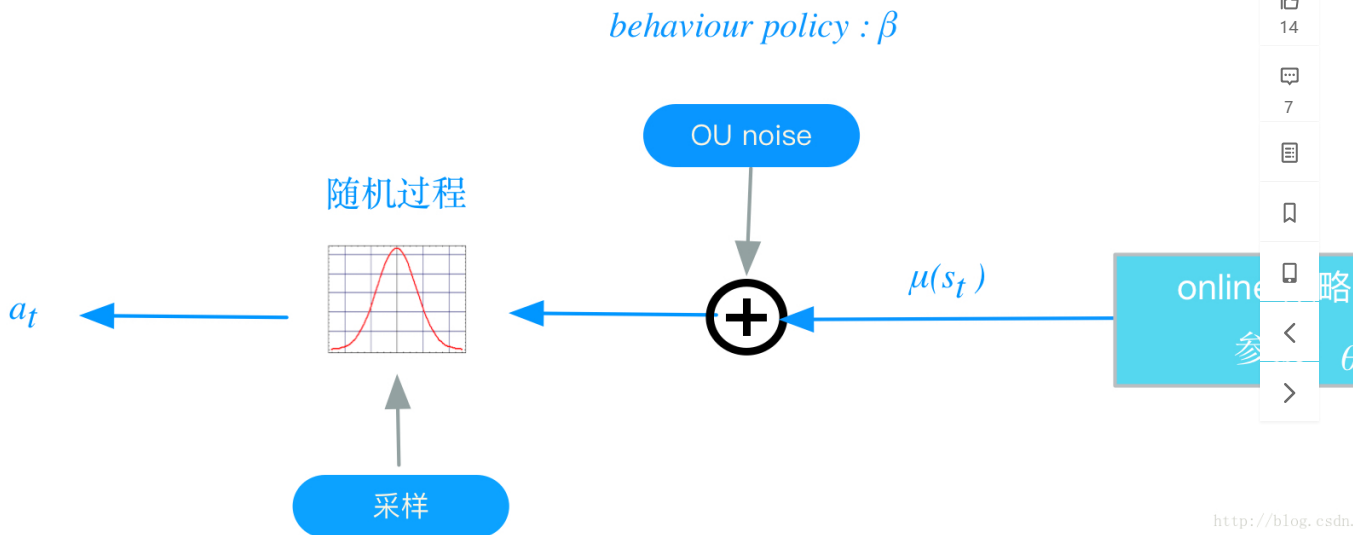
DDPG算法相关基本概念定义

我们以Open Gym 作为环境为例来讲解。

先复述一下DDPG相关的概念定义:

- 确定性行为策略 μ : 定义为一个函数, 每一步的行为可以通过 $a_t = \mu(s_t)$ 计算获得。
- 策略网络: 用一个卷积神经网络对 μ 函数进行模拟, 这个网络我们就叫做策略网络, 其参数为 θ^{μ} ;
- behavior policy β : 在RL训练过程中, 我们要兼顾2个e: exploration和exploit; exploration的目的是探索潜在的更优策略, 所以训练过程中, 我决策机制引入随机噪声:
将action的决策从确定性过程变为一个随机过程, 再从这个随机过程中采样得到action, 下达给环境执行。

过程如下图所示：



上述这个策略叫做behavior策略，用 β 来表示，这时RL的训练方式叫做off-policy.

这里与 $\epsilon - greedy$ 的思路是类似的。

DDPG中，使用Uhlenbeck-Ornstein随机过程（下面简称UO过程），作为引入的随机噪声：

UO过程在时序上具备很好的相关性，可以使agent很好的探索具备动量属性的环境。

注意：

– 这个 β 不是我们想要得到的最优策略，仅仅在训练过程中，生成下达给环境的action，从而获得我们想要的数据集，比如状态转换(transitions)、的行走路径等，然后利用这个数据集去训练策略 μ ，以获得最优策略。

– 在test 和 evaluation时，使用 μ ，不会再使用 β 。

- Q 函数: 即action-value 函数，定义在状态 s_t 下，采取动作 a_t 后，且如果持续执行策略 μ 的情况下，所获得的 R_t 期望值，用Bellman 等式来定义

$$Q^\mu(s_t, a_t) = E[r(s_t, a_t) + \gamma Q^\mu(s_{t+1}, \mu(s_{t+1}))]$$

可以看到， Q 函数的定义是一个递归表达，在实际情况中，我们不可能每一步都递归计算 Q 的值，可行的方案是通过一个函数对Bellman等式表达进行模拟。

- Q 网络: DDPG中，我们用一个卷积神经网络对 Q 函数进行模拟，这个网络我们就叫做 Q 网络，其参数为 θ^Q 。采用了DQN相同的方法。
- 如何衡量一个策略 μ 的表现：用一个函数 J 来衡量，我们叫做performance objective，针对off-policy学习的场景，定义如下：

$$\begin{aligned} J_\beta(\mu) &= \int_S \rho^\beta(s) Q^\mu(s, \mu(s)) ds \\ &= E_{s \sim \rho^\beta}[Q^\mu(s, \mu(s))] \end{aligned}$$

其中：

- s 是环境的状态，这些状态(或者说agent在环境中走过的状态路径)是基于agent的behavior策略产生的，它们的分布函数(pdf) 为 ρ^β ；
- $Q^\mu(s, \mu(s))$ 是在每个状态下，如果都按照 μ 策略选择action时，能够产生的Q值。
也即， $J_\beta(\mu)$ 是在 s 根据 ρ^β 分布时， $Q^\mu(s, \mu(s))$ 的期望值。
- 训练的目标：最大化 $J_\beta(\mu)$ ，同时最小化 Q 网络的Loss(下面描述算法步骤时会给出)。
- 最优行为策略 μ 的定义: 即最大化 $J_\beta(\mu)$ 的策略：

$$\mu = \underset{\mu}{\operatorname{argmax}} J(\mu)$$

训练 μ 网络的过程，就是寻找 μ 网络参数 θ^μ 的最优解的过程，我们使用SGA(stochastic gradient ascent)的方法。

- 最优 Q 网络定义：具备最小化的 Q 网络Loss；

训练 Q 网络的过程，就是寻找 Q 网络参数 θ^Q 的最优解的过程，我们使用SGD的方法。

DDPG实现框架和算法

online 和 target 网络

以往的实践证明，如果只使用单个“ Q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gradient update的同时，又用于计
gradient, 参见下面等式(1),(2),(3).

基于此，DDPG分别为策略网络、Q网络各创建两个神经网络拷贝,一个叫做online，一个叫做target:

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

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

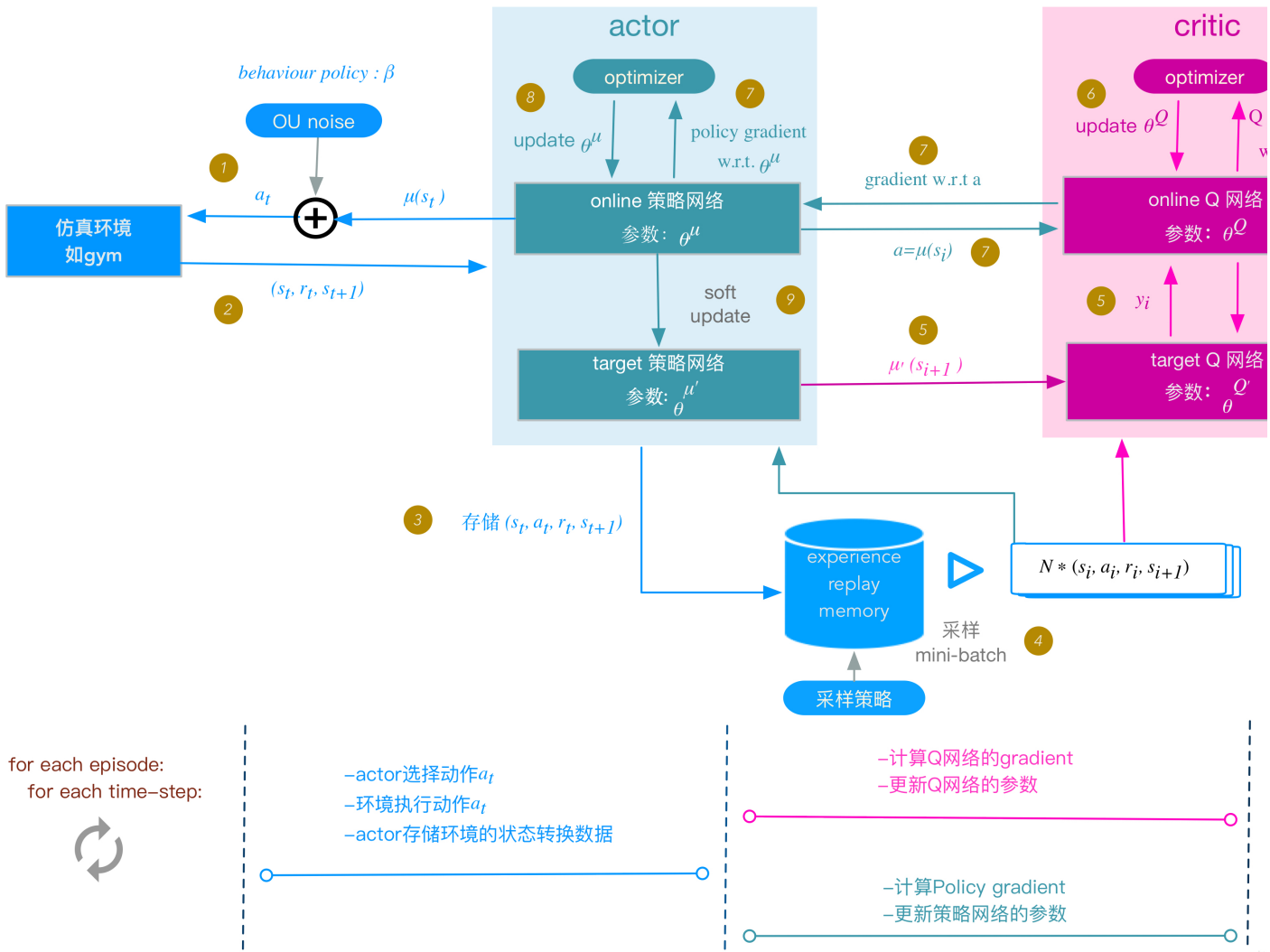
在训练完一个mini-batch的数据之后，通过SGA/SGD算法更新online网络的参数，然后再通过soft update算法更新 target 网络的参数。soft update
running average的算法：

$$\text{soft update} : \begin{cases} \theta^{Q'} \leftarrow \tau \theta^Q + (1 - \tau) \theta^{Q'} \\ \theta^{\mu'} \leftarrow \tau \theta^\mu + (1 - \tau) \theta^{\mu'} \end{cases}$$

 τ 一般取值0.001

- 优点：target网络参数变化小，用于在训练过程中计算online网络的gradient，比较稳定，训练易于收敛。
- 代价：参数变化小，学习过程变慢。

DDPG实现框架，如下图所示：



DDPG算法流程如下：

初始化actor\critic的 online 神经网络参数: θ^Q 和 θ^μ ;
将online网络的参数拷贝给对应的target网络参数: $\theta^{Q'} \leftarrow \theta^Q, \theta^{\mu'} \leftarrow \theta^\mu$;
初始化replay memory buffer R ;
for each episode:
 初始化UO随机过程;
 for t = 1, T:
 下面的步骤与DDPG实现框架图中步骤编号对应:

1. actor 根据behavior策略选择一个 a_t , 下达给gym执行该 a_t .

$$a_t = \mu(s_t | \theta^\mu) + \mathcal{N}_t$$

behavior策略是一个根据当前online策略 μ 和随机UO噪声生成的随机过程, 从这个随机过程采样 获得 a_t 的值。

2. gym执行 a_t , 返回reward r_t 和新的状态 s_{t+1} ;

3. actor将这个状态转换过程(transition): (s_t, a_t, r_t, s_{t+1}) 存入replay memory buffer R 中, 作为训练online网络的数据集。

4. 从replay memory buffer R 中, 随机采样 N 个 transition 数据, 作为online策略网络、 online Q网络的一个mini-batch训练数据。我们用 (s_i, a_i, r_i, s_{i+1}) 表 batch中的单个transition数据。

5. 计算online Q网络的 gradient:

Q 网络的loss定义: 使用类似于监督式学习的方法, 定义loss为MSE: mean squared error:

$$L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i | \theta^Q))^2 \tag{1}$$

其中, y_i 可以看做“ 标签” :

$$y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1} | \theta^{\mu'}) | \theta^{Q'}) \tag{2}$$

基于标准的back-propagation方法, 就可以求得 L 针对 θ^Q 的gradient: $\nabla_{\theta^Q} L$ 。

有两点值得注意:

- y_i 的计算, 使用的是 target 策略网络 μ' 和 target Q 网络 Q' , 这样做是为了Q网络参数的学习过程更加稳定, 易于收敛。
- 这个标签本身依赖于我们正在学习的target网络, 这是区别于监督式学习的地方。

6. update online Q: 采用Adam optimizer更新 θ^Q ;

7. 计算策略网络的policy gradient:

policy gradient的定义: 表示performance objective的函数 J 针对 θ^μ 的 gradient。根据2015 D.Silver 的DPG 论文中的数学推导, 在采用off-policy的训练方法时, policy gradient算法如下:

$$\nabla_{\theta^\mu} J_\beta(\mu) \approx E_{s \sim \rho^\beta} [\nabla_a Q(s, a | \theta^Q) |_{a=\mu(s)} \cdot \nabla_{\theta^\mu} \mu(s | \theta^\mu)] \tag{3}$$

也即, policy gradient是在 s 根据 ρ^β 分布时, $\nabla_a Q \cdot \nabla_{\theta^\mu} \mu$ 的期望值。我们用Monte-carlo方法来估算这个期望值:

在replay memory buffer中存储的(transition): (s_i, a_i, r_i, s_{i+1}) , 是基于agent的behavior策略 β 产生的, 它们的分布函数(pdf)为 ρ^β , 所以当我们从replay memory buffer中随机采样获得mini-batch数据时, 根据Monte-carlo方法, 使用mini-batch数据代入上述policy gradient公式, 可以作为对上述期望值的一个无偏差估计(biased estimate), 所以policy gradient 可以改写为:

$$\nabla_{\theta^\mu} J_\beta(\mu) \approx \frac{1}{N} \sum_i (\nabla_a Q(s, a | \theta^Q) |_{s=s_i, a=\mu(s_i)} \cdot \nabla_{\theta^\mu} \mu(s | \theta^\mu) |_{s=s_i}) \tag{4}$$

8. update online策略网络: 采用Adam optimizer更新 θ^μ ;


9. soft update target网络 μ' 和 Q' :


使用running average 的方法, 将online网络的参数, soft update给target网络的参数:


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


τ 一般取值0.001


end for time step
end for episode


14


7











总结一下：

actor-critic框架是一个在循环的episode和时间步骤条件下, 通过环境、actor和critic三者交互, 来迭代训练策略网络、Q网络的过程。

DDPG对于DPG的关键改进

2019/1/9

Deep Reinforcement Learning - 1. DDPG原理和算法 - kenneth_yu的博客 - CSDN博客

1. 使用卷积神经网络来模拟策略函数和Q函数，并用深度学习的方法来训练，证明了在RL方法中，非线性模拟函数的准确性和高性能、可收敛；而DPG中，可以看成使用线性回归的机器学习方法：使用带参数的线性函数来模拟策略函数和Q函数，然后使用线性回归的方法进行训练。

2. experience replay memory的使用：actor同环境交互时，产生的transition数据序列是在时间上高度关联(correlated)的，如果这些数据序列直接用于训练，容易overfit，不易收敛。
DDPG的actor将transition数据先存入experience replay buffer，然后在训练时，从experience replay buffer中随机采样mini-batch数据，这样采样得到的数据是不相关的。

3. target 网络和online 网络的使用， 使的学习过程更加稳定，收敛更有保障。

👍 14

💬 7

📖

🔖

📱

⏪

导致

可以

下一篇

基于tensorflow的代码实现。

🔖 收藏

🔗 分享

重大通知！1月起，上海上班族可正式在职申读985/211本科！
尚德机构 · 顶新

-  想对作者说点什么
-  weixin_37662589：深入浅出，goodood （1个月前 #7楼）
-  qq_32751937：博主用的什么画图软件啊？很精美！ （2个月前 #6楼）
-  zhouzq_890709：请问您画图软件是什么？ （2个月前 #5楼）
-  梅川鸡尾酒：不知道在论文里能不能引用作者的图呢？ （3个月前 #4楼）
-  WinddyAkoky：请问这个图是用什么画的啊 （5个月前 #3楼）
-  wsyawl：写的真好，尤其是那个图画的非常完美 （7个月前 #2楼）
-  sinat_38402582：真厉害 （8个月前 #1楼）

强化学习系列 7 : **Deep Deterministic Policy Gradient (DDPG)** 👁 6871
<>Deep Deterministic Policy Gradient (DDPG) 改进版 Deep Deterministic Poli... 来自: [女王の专属领地](#)

深度强化学习实战-Tensorflow实现**DDPG** 👁 1253
前言这是开栏以来的第一篇文章，都说万事开头难，希望开了这个头之后，专栏里能越来越多关于深... 来自: [zhi元元元](#)

深度强化学习——连续动作控制**DDPG**、NAF 👁 2.2万
传统的DQN只适用于离散动作控制，而DDPG和NAF是深度强化学习在连续动作控制上的拓展。... 来自: [草帽BOY的博客](#)

重大通知！1月起，上海上班族可正式在职申读985/211本科！
尚德机构 · 顶新

【强化学习】**DDPG(Deep Deterministic Policy Gradient)算法**详解 👁 1836
<http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html> 引自Reinforcement Learning:An Intr... 来自: [shura的技术空间](#)

(**DDPG**)深度确定策略梯度调参体会 👁 4434
花了一个星期，昨晚终于调出了还算能工作的模型，真的很难。赶紧记下来备忘。直接使用论文中的... 来自: [万德1010的博客](#)

DDPG之OU过程 👁 2336
Ornstein-Uhlenbeck过程浅析 上周在实现DDPG的过程中，发现其中用到了一个没见过的随机过... 来自: [微念的博客](#)

android google map 地图上显示叉叉问题。 👁 472
具体的问题如下图，我就不详细描述了。出现这种规律性的问题是设置地图显示样式的问题，只要不...

Deep Reinforcement learning - 2. 基于tensorflow的DDPG实现 👁 7988
TODDeep Reinforcemen learning - 2. 基于tensorflow的DDPG实现基于我上一篇博客的算法介绍... 来自: [kenneth_yu的博客](#)