|  |
| --- |
| 首行 |
| 3-ALGORITHMS DOCS |
| 单位：NWPU |
| 时间2018年12月7日星期五 |

目录

[1 Vanilla Policy Gradient 1](#_Toc533002930)

[1.1 Background 1](#_Toc533002931)

[1.2 Quick Facts 1](#_Toc533002932)

[1.3 Key Equations 1](#_Toc533002933)

[1.4 Exploration vs. Exploitation 2](#_Toc533002934)

[1.5 Pseudocode(伪代码) 2](#_Toc533002935)

[1.6 Documentation 2](#_Toc533002936)

[1.7 Saved Model Contents 4](#_Toc533002937)

[1.8 References 4](#_Toc533002938)

[2 Trust Region Policy Optimization 5](#_Toc533002939)

[2.1 Background 5](#_Toc533002940)

[2.2 Quick Facts 5](#_Toc533002941)

[2.3 Key Equations 5](#_Toc533002942)

[2.4 Exploration vs. Exploitation 6](#_Toc533002943)

[2.5 Pseudocode 7](#_Toc533002944)

[2.6 Documentation 8](#_Toc533002945)

[2.7 Saved Model Contents 8](#_Toc533002946)

[2.8 References 8](#_Toc533002947)

[2.9 Relevant Papers 8](#_Toc533002948)

[2.10 Why These Papers? 8](#_Toc533002949)

[2.11 Other Public Implementations 8](#_Toc533002950)

[3 Proximal Policy Optimization（近端策略优化） 9](#_Toc533002951)

[3.1 Background 9](#_Toc533002952)

[3.2 Quick Facts 9](#_Toc533002953)

[3.3 Key Equations 10](#_Toc533002954)

[3.4 Exploration vs. Exploitation 11](#_Toc533002955)

[3.5 Pseudocode 11](#_Toc533002956)

[3.6 Documentation 12](#_Toc533002957)

[3.7 Saved Model Contents 12](#_Toc533002958)

[3.8 References 12](#_Toc533002959)

[3.9 Relevant Papers 12](#_Toc533002960)

[3.10 Why These Papers? 12](#_Toc533002961)

[3.11 Other Public Implementations 13](#_Toc533002962)

[4 Deep Deterministic Policy Gradient 14](#_Toc533002963)

[4.1 Background 14](#_Toc533002964)

[4.2 Quick Facts 14](#_Toc533002965)

[4.3 Key Equations 14](#_Toc533002966)

[4.4 The Q-Learning Side of DDPG 14](#_Toc533002967)

[4.5 The Policy Learning Side of DDPG 14](#_Toc533002968)

[4.6 Exploration vs. Exploitation 14](#_Toc533002969)

[4.7 Pseudocode 15](#_Toc533002970)

[4.8 Documentation 15](#_Toc533002971)

[4.9 Saved Model Contents 15](#_Toc533002972)

[4.10 References 15](#_Toc533002973)

[4.11 Relevant Papers 15](#_Toc533002974)

[4.12 Why These Papers? 15](#_Toc533002975)

[4.13 Other Public Implementations 15](#_Toc533002976)

[5 Twin Delayed DDPG 16](#_Toc533002977)

[5.1 Background 16](#_Toc533002978)

[5.2 Quick Facts 16](#_Toc533002979)

[5.3 Key Equations 16](#_Toc533002980)

[5.4 Exploration vs. Exploitation 16](#_Toc533002981)

[5.5 Pseudocode 16](#_Toc533002982)

[5.6 Documentation 16](#_Toc533002983)

[5.7 Saved Model Contents 16](#_Toc533002984)

[5.8 References 16](#_Toc533002985)

[5.9 Relevant Papers 16](#_Toc533002986)

[5.10 Other Public Implementations 16](#_Toc533002987)

[6 Soft Actor-Critic 17](#_Toc533002988)

[6.1 Background 17](#_Toc533002989)

[6.2 Quick Facts 17](#_Toc533002990)

[6.3 Key Equations 17](#_Toc533002991)

[6.4 Entropy-Regularized Reinforcement Learning 17](#_Toc533002992)

[6.5 Soft Actor-Critic 17](#_Toc533002993)

[6.6 Exploration vs. Exploitation 17](#_Toc533002994)

[6.7 Pseudocode 18](#_Toc533002995)

[6.8 Documentation 19](#_Toc533002996)

[6.9 Saved Model Contents 19](#_Toc533002997)

[6.10 References 19](#_Toc533002998)

[6.11 Relevant Papers 19](#_Toc533002999)

[6.12 Relevant Papers 19](#_Toc533003000)

[6.13 Other Public Implementations 19](#_Toc533003001)

# Vanilla Policy Gradient

## Background

基本策略梯度的关键思想是提高导致更高回报的行为的概率，并降低导致较低回报的行为的概率，直到您达到最优策略为止。

## Quick Facts

VPG是一种基于策略的算法.

VPG可以用于具有离散或连续操作空间的环境。

VPG的旋转实现支持MPI的并行化。

## Key Equations

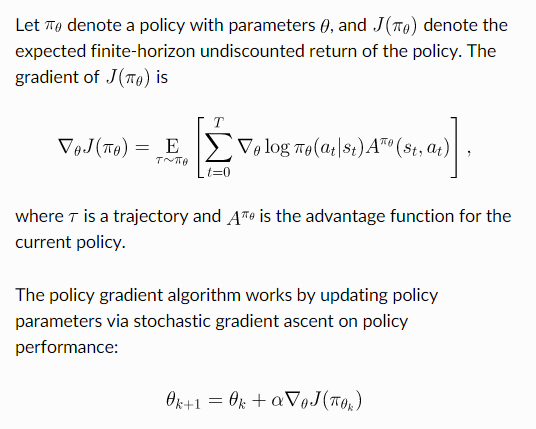


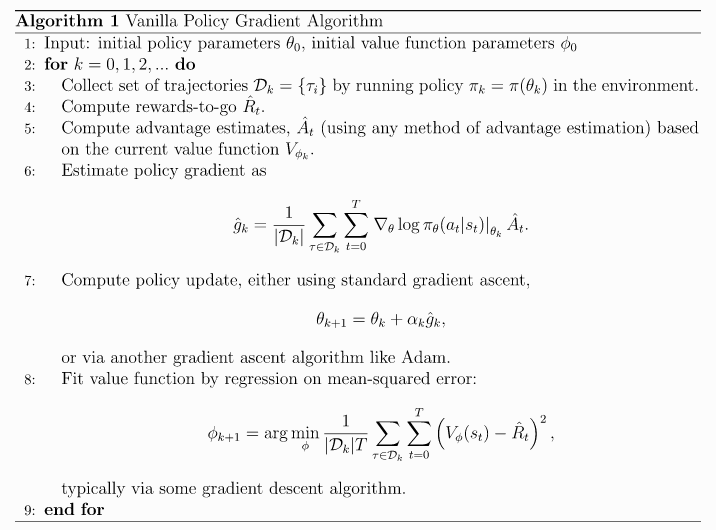
图 1-1

策略梯度实现通常基于无限水平折扣回报计算优势函数估计，尽管在其他情况下使用的是有限水平未折扣策略梯度公式。

## Exploration vs. Exploitation

VPG以一种on policy的方式训练随机策略.这意味着它根据其随机政策的最新版本，通过抽样行动进行探索。行动选择的随机性程度取决于初始条件和训练过程。在培训过程中，策略通常变得越来越少随机性，因为更新规则鼓励它利用它已经发现的奖励。这可能导致政策陷入局部最优状态。

## Pseudocode(伪代码)



是神经网络，具体参见2章3.6。

## Documentation

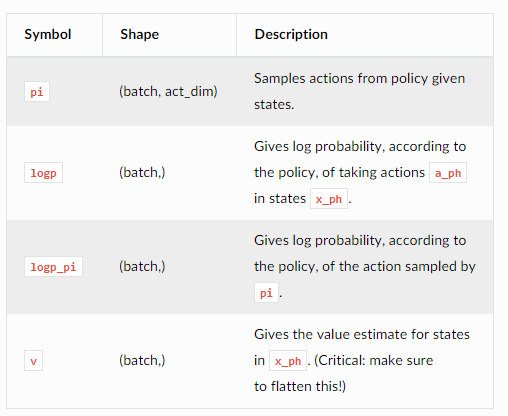
spinup.vpg(env\_fn, actor\_critic=<function mlp\_actor\_critic>, ac\_kwargs={}, seed=0, steps\_per\_epoch=4000, epochs=50, gamma=0.99, pi\_lr=0.0003, vf\_lr=0.001, train\_v\_iters=80, lam=0.97, max\_ep\_len=1000, logger\_kwargs={}, save\_freq=10)

参数:

env\_fn –创建环境副本的函数。环境必须满足OpenAI健身房API。

actor\_critic –

一个函数，它接受状态、x\_ph和action a\_ph的占位符符号，并从代理的TensorFlow计算图中返回主输出:



ac\_kwargs (dict) – Any kwargs appropriate for the actor\_critic function you provided to VPG.

seed (int) – Seed for random number generators.

steps\_per\_epoch (int) – Number of steps of interaction (state-action pairs) for the agent and the environment in each epoch.

epochs (int) – Number of epochs of interaction (equivalent to number of policy updates) to perform.

gamma (float) – Discount factor. (Always between 0 and 1.)

pi\_lr (float) – Learning rate for policy optimizer.

vf\_lr (float) – Learning rate for value function optimizer.

train\_v\_iters (int) – Number of gradient descent steps to take on value function per epoch.

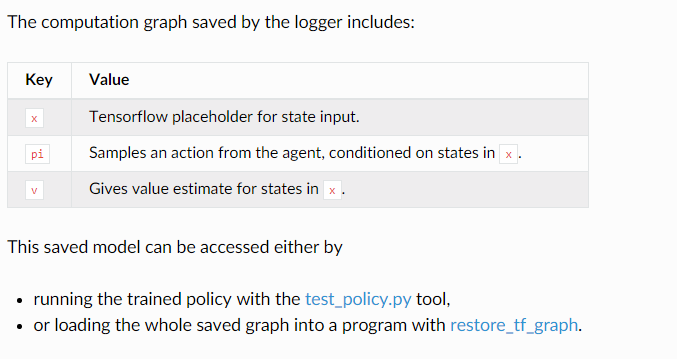
lam (float) – Lambda for GAE-Lambda. (Always between 0 and 1, close to 1.)

max\_ep\_len (int) – Maximum length of trajectory / episode / rollout.

logger\_kwargs (dict) – Keyword args for EpochLogger.

save\_freq (int) – How often (in terms of gap between epochs) to save the current policy and value function.

## Saved Model Contents



## References

Relevant Papers

* [Policy Gradient Methods for Reinforcement Learning with Function Approximation](https://papers.nips.cc/paper/1713-policy-gradient-methods-for-reinforcement-learning-with-function-approximation.pdf), Sutton et al. 2000
* [Optimizing Expectations: From Deep Reinforcement Learning to Stochastic Computation Graphs](http://joschu.net/docs/thesis.pdf), Schulman 2016(a)
* [Benchmarking Deep Reinforcement Learning for Continuous Control](https://arxiv.org/abs/1604.06778), Duan et al. 2016
* [High Dimensional Continuous Control Using Generalized Advantage Estimation](https://arxiv.org/abs/1506.02438), Schulman et al. 2016(b)

Why These Papers?

Sutton 2000之所以包括在内，是因为它是强化学习理论的永恒经典，它包含了对导致现代政策梯度的早期工作的参考。Schulman 2016(A)之所以包括在内，是因为第二章清楚地介绍了包括伪码在内的策略梯度算法的理论。段2016是一篇清晰的、最近的基准论文，它展示了深度rl设置中的香草策略梯度(例如，使用神经网络策略和adam作为优化器)与其他深度rl算法的比较。Schulman 2016(B)之所以包括在内，是因为我们的VPG实现使用了广义优势估计来计算策略梯度。

# Trust Region Policy Optimization

## Background

trpo通过采取尽可能大的步骤来提高性能来更新策略，同时满足新和旧策略允许有多近的特殊限制。约束用KL-散度来表示，KL-散度是概率分布之间距离的一种度量。

这不同于常规的策略梯度，即在参数空间中保持新旧策略的紧密性。但是，即使参数空间上看似很小的差异，在性能上也会有很大的差异-因此，一个糟糕的步骤可能会使策略绩效崩溃。这使得使用大步长与普通策略梯度一起使用是危险的，从而损害了它的样本效率。trpo很好地避免了这种崩溃，并且趋向于快速和单调地提高表现。

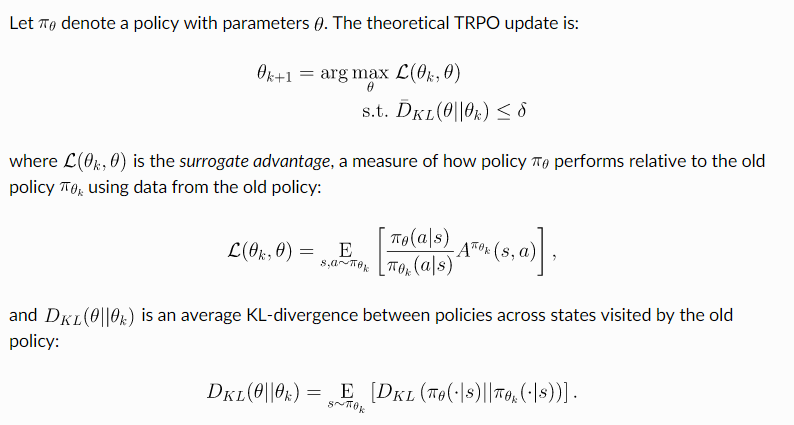
## Quick Facts

trpo是一种在策略上的算法.

TRPO可用于具有离散或连续操作空间的环境。

trpo的旋转实现支持MPI的并行化。

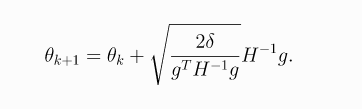
## Key Equations



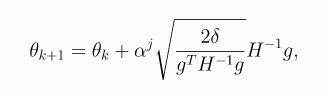
当θ=θ\_k时，你应该知道目标和约束都是零的。此外，当θ=θ\_k时，对θ的约束梯度为零。证明这些事实需要一些微妙的掌握相关的数学-这是一个练习值得做，只要你觉得准备！

理论上的trpo更新并不是最容易使用的，所以trpo做了一些近似来快速得到答案。我们把目标和约束扩展到θ\_k周围的主导秩序。

这个近似问题可以用拉格朗日对偶的方法解析求解，得到的解如下：



如果我们在这里停下来，使用这个最后的结果，算法就会精确地计算出自然政策的梯度。一个问题是，由于泰勒展开引入的逼近误差，这可能不满足KL约束，或者实际上改善了代理优势。trpo添加了对此更新规则的修改：回溯行搜索，



其中αin(0，1)是回溯系数，j是最小的非负整数，使得pi\_{theta\_{k1}满足KL约束并产生正的代理优势。

最后：计算和存储矩阵逆h^{-1}，在处理数千或数百万参数的神经网络策略时，代价高昂。Trpo采用共轭梯度算法求解x=h^{-1}的hx=g，只需要一个能计算矩阵向量积hx的函数，而不需要直接计算和存储整个矩阵h。这并不难：我们设置了一个符号运算来计算。

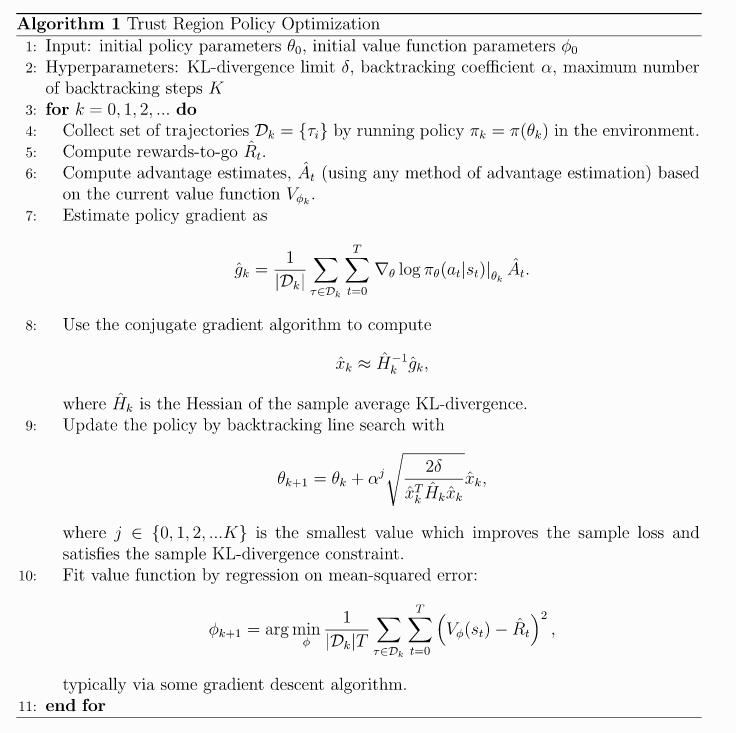


这给了我们正确的输出，而不需要计算整个矩阵。

## Exploration vs. Exploitation

trpo以一种政策上的方式训练随机政策.这意味着它根据其随机政策的最新版本，通过抽样行动进行探索。行动选择的随机性程度取决于初始条件和训练过程。在培训过程中，策略通常变得越来越少随机性，因为更新规则鼓励它利用它已经发现的奖励。这可能导致政策陷入局部最优状态。

## Pseudocode



## Documentation

## Saved Model Contents

## References

## Relevant Papers

* [Trust Region Policy Optimization](https://arxiv.org/abs/1502.05477), Schulman et al. 2015
* [High Dimensional Continuous Control Using Generalized Advantage Estimation](https://arxiv.org/abs/1506.02438), Schulman et al. 2016
* [Approximately Optimal Approximate Reinforcement Learning](https://people.eecs.berkeley.edu/~pabbeel/cs287-fa09/readings/KakadeLangford-icml2002.pdf), Kakade and Langford 2002

## Why These Papers?

Schulman 2015之所以包括在内，是因为它是描述TRPO的原始论文。Schulman 2016之所以包括在内，是因为我们的Trpo实现使用了广义优势估计来计算策略梯度。Kakade和Langford 2002之所以包括在内，是因为它包含了激励和深入联系TRPO理论基础的理论成果。

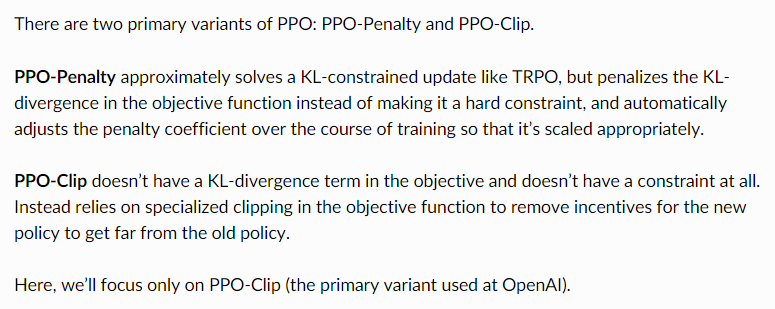
## Other Public Implementations

* [Baselines](https://github.com/openai/baselines/tree/master/baselines/trpo_mpi)
* [ModularRL](https://github.com/joschu/modular_rl/blob/master/modular_rl/trpo.py)
* [rllab](https://github.com/rll/rllab/blob/master/rllab/algos/trpo.py)

# Proximal Policy Optimization（近端策略优化）

## Background

PPO的动机与trpo相同：我们如何才能在使用现有数据的政策上采取最大可能的改进步骤，而不采取意外导致性能崩溃的步骤呢？在trpo试图用复杂的二阶方法解决这个问题的地方，ppo是一个一阶方法的家族，它使用一些其他的技巧来使新的策略接近旧的。PPO方法实现起来要简单得多，而且在经验上似乎至少与trpo一样好。



PPO有两个主要的变体：PPO-惩罚和PPO-剪辑。

PPO-惩罚近似地解决了像trpo这样的KL约束更新，但惩罚了目标函数中的KL-发散，而不是使其成为一个硬约束，并在训练过程中自动调整惩罚系数，使其适当地缩放。PPO-剪辑在目标中没有KL-发散项，而且完全没有约束.相反，它依赖于目标函数中的专门剪辑来消除对新政策的激励，使其远离旧的政策。

在这里，我们将只关注PPO-剪辑(在OpenAI中使用的主要变体)。

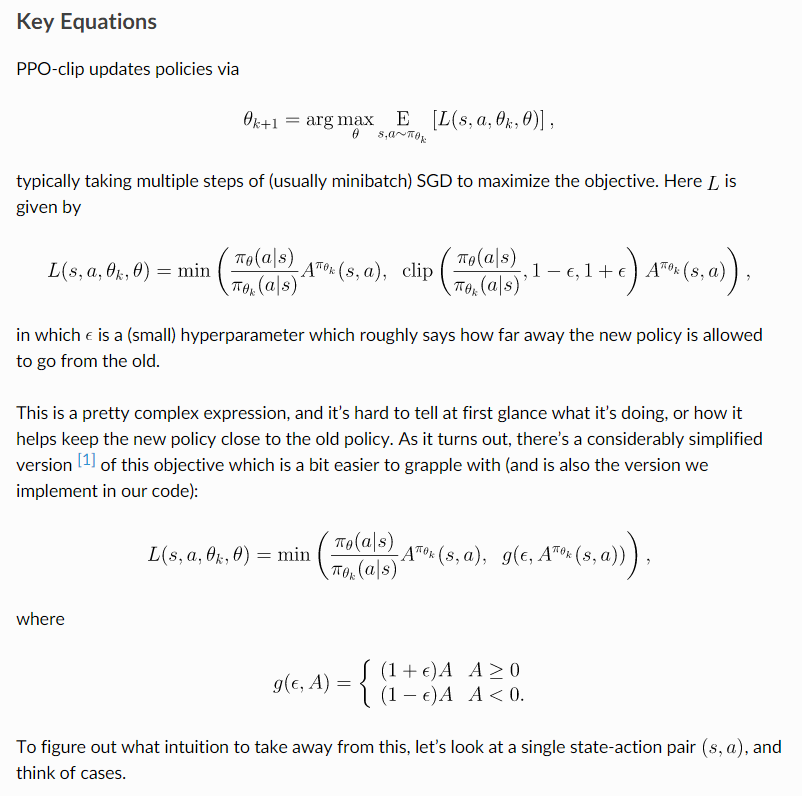
## Quick Facts

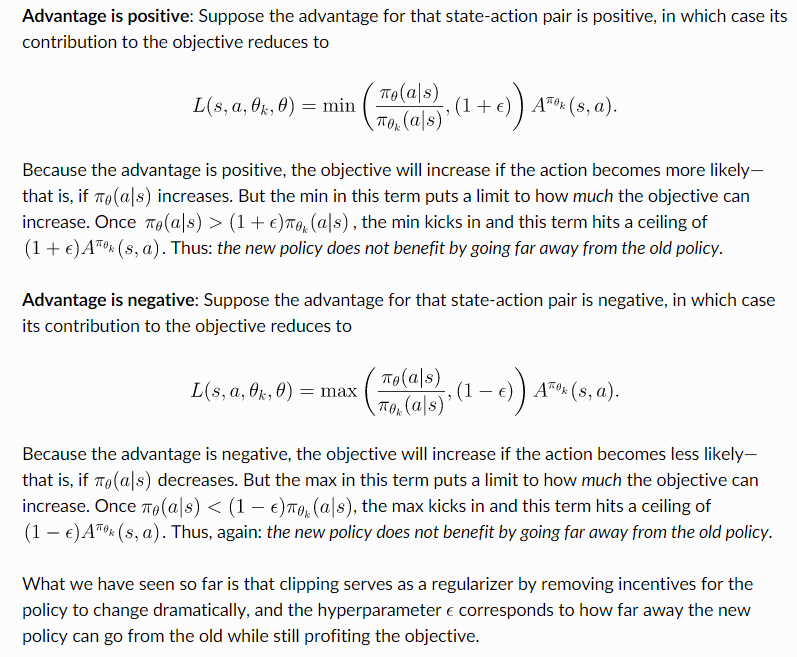
PPO是一种在策略上的算法。

PPO可用于具有离散或连续操作空间的环境。

PPO的旋转实现支持MPI的并行化。

## Key Equations





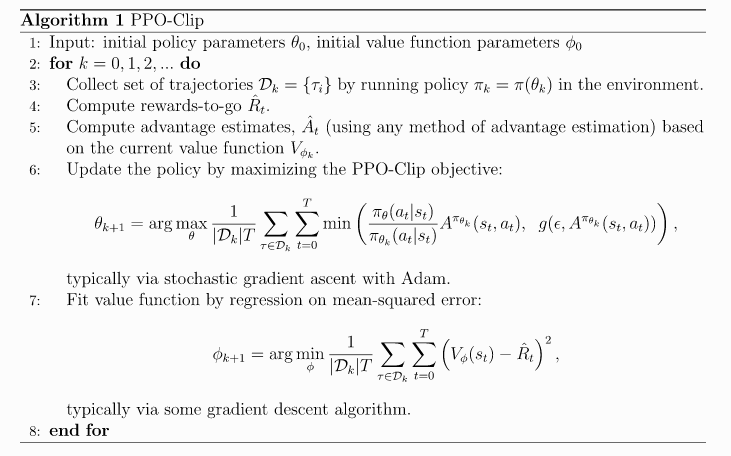
到目前为止，我们所看到的是，剪裁是一个正则化者，它消除了对政策急剧变化的激励，而超参数epsilon则对应于新政策与旧政策之间的距离，同时仍在为目标谋利。

虽然这种剪裁对于确保合理的策略更新有很大帮助，但仍然有可能产生一个与旧策略相去甚远的新策略，不同的PPO实现使用了许多技巧来避免这种情况。在这里的实现中，我们使用了一个特别简单的方法：早期停止。如果新政策与旧政策的平均KL-分歧超过一个阈值，我们就停止采取梯度步骤。当您对基本的数学和实现细节感到满意时，值得查看其他实现，看看它们是如何处理这个问题的！

## Exploration vs. Exploitation

PPO以一种政策上的方式训练随机策略.这意味着它根据其随机政策的最新版本，通过抽样行动进行探索。行动选择的随机性程度取决于初始条件和训练过程。在培训过程中，策略通常变得越来越少随机性，因为更新规则鼓励它利用它已经发现的奖励。这可能导致政策陷入局部最优状态。

## Pseudocode



## Documentation

## Saved Model Contents

## References

## Relevant Papers

* [Proximal Policy Optimization Algorithms](https://arxiv.org/abs/1707.06347), Schulman et al. 2017
* [High Dimensional Continuous Control Using Generalized Advantage Estimation](https://arxiv.org/abs/1506.02438), Schulman et al. 2016
* [Emergence of Locomotion Behaviours in Rich Environments](https://arxiv.org/abs/1707.02286), Heess et al. 2017

## Why These Papers?

舒尔曼2017年之所以包括在内，是因为它是描述PPO的原始论文。Schulman 2016之所以包括在内，是因为我们的PPO实现使用了广义优势估计来计算策略梯度。Heess 2017之所以包括在内，是因为它对PPO代理在复杂环境中学习到的行为进行了大规模的实证分析(尽管它使用的是PPO惩罚而不是PPO-剪辑)。

## Other Public Implementations

* [Baselines](https://github.com/openai/baselines/tree/master/baselines/ppo2)
* [ModularRL](https://github.com/joschu/modular_rl/blob/master/modular_rl/ppo.py) (Caution: this implements PPO-penalty instead of PPO-clip.)
* [rllab](https://github.com/rll/rllab/blob/master/rllab/algos/ppo.py) (Caution: this implements PPO-penalty instead of PPO-clip.)
* [rllib (Ray)](https://github.com/ray-project/ray/tree/master/python/ray/rllib/agents/ppo)

# Deep Deterministic Policy Gradient

## Background

## Quick Facts

ddpg是一种非策略算法.

ddpg只能用于具有连续操作空间的环境。

对于连续的动作空间，ddpg可以被认为是深度Q学习。

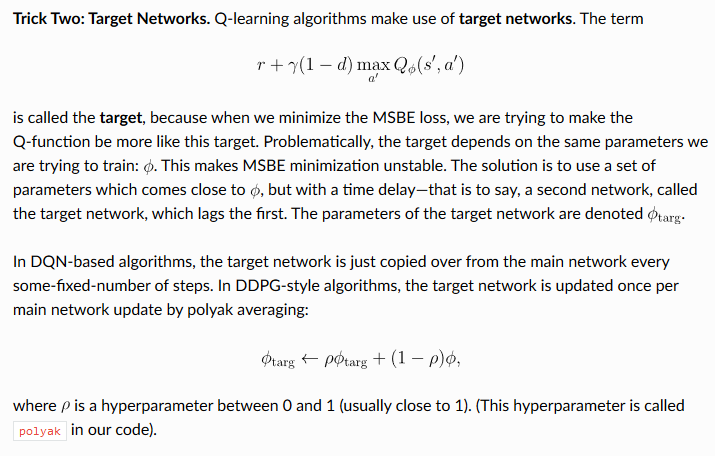
DDPG的旋转实现不支持并行化。

## Key Equations

这里，我们将解释ddpg的两个部分背后的数学：学习q函数和学习策略。

## The Q-Learning Side of DDPG

技巧一：重放缓冲区。所有用于训练深度神经网络以逼近Q^\*(s，a)的标准算法都使用了经验回放缓冲器。这是以前经验的集合{mathcal d}。为了使算法具有稳定的行为，重播缓冲区应该足够大，以包含广泛的经验，但它可能并不总是好保持一切。如果你只使用最新的数据，你就会过度适应，事情就会破裂；如果你使用太多的经验，你可能会放慢你的学习速度。这可能需要一些调整才能正确。

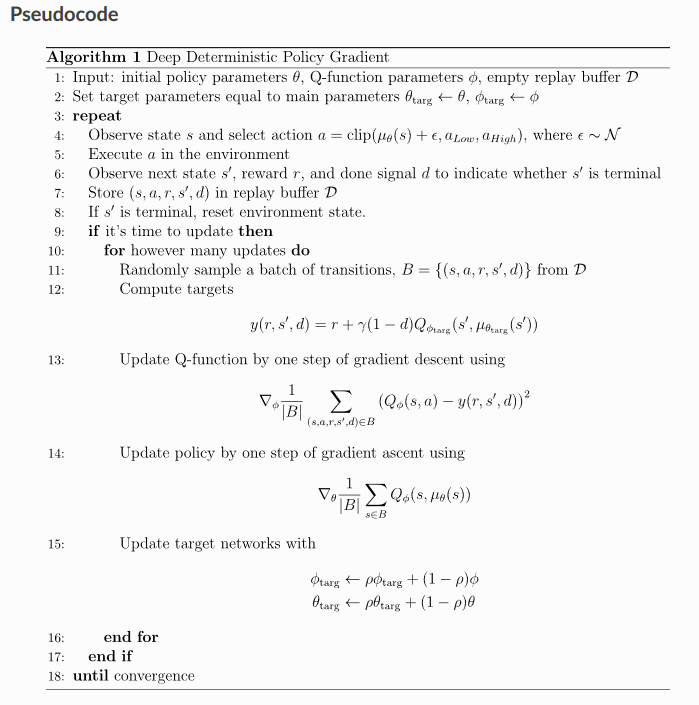


## The Policy Learning Side of DDPG

## Exploration vs. Exploitation

ddpg以一种非政策的方式训练一种确定性的政策。由于策略是确定性的，如果代理要探索on-策略，那么在一开始它可能不会尝试足够广泛的各种操作来找到有用的学习信号。为了使ddpg策略更好地探索，我们在训练时给他们的行为添加了噪音。最初的ddpg论文的作者推荐了时间相关的ou噪声，但是最近的结果表明，不相关的、平均为零的高斯噪声工作得很好。因为后者比较简单，所以更可取。为了更好地获取高质量的培训数据，你可以减少训练过程中的噪音。(在实施过程中，我们不会这样做，并在整个过程中保持固定的噪音等级。)

## Pseudocode



## Documentation

## Saved Model Contents

## References

## Relevant Papers

## Why These Papers?

## Other Public Implementations

# Twin Delayed DDPG

## Background

## Quick Facts

## Key Equations

## Exploration vs. Exploitation

## Pseudocode

## Documentation

## Saved Model Contents

## References

## Relevant Papers

## Other Public Implementations

# Soft Actor-Critic

## Background

软参与者批判性算法(Sac)是一种以非策略的方式对随机策略进行优化的算法，它在随机策略优化和ddpg式方法之间架起了一座桥梁。它并不是td3的直接继承者(大致是同时发布的)，但它结合了剪裁的双Q技巧，而且由于SAC中策略的固有随机性，它也最终受益于类似目标策略平滑之类的东西。

Sac的一个中心特征是熵正则化。该策略被训练成最大限度地在预期收益和熵之间进行权衡，熵是衡量策略中随机性的一种度量。这与勘探开发的权衡有着密切的联系：熵的增加导致了更多的探索，从而加速了以后的学习。它还可以防止策略过早地收敛到坏的局部最优。

## Quick Facts

SAC是一种非策略算法.

此处实现的Sac版本只能用于具有连续操作空间的环境。

可以实现Sac的另一个版本，它稍微改变策略更新规则，以处理离散的操作空间。

Sac的旋转实现不支持并行化。

## Key Equations

为了解释软演员批评，我们首先必须引入熵正则化的强化学习设置.在熵正则化的rl中，值函数的方程略有不同.

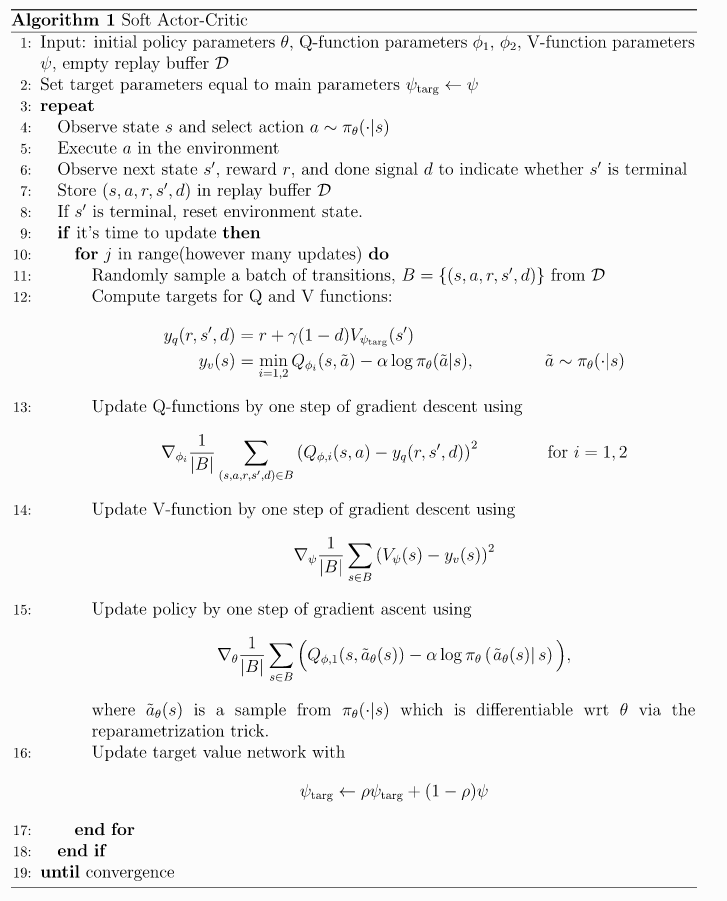
## Entropy-Regularized Reinforcement Learning

## Soft Actor-Critic

## Exploration vs. Exploitation

SAC训练具有熵正则化的随机策略，并在策略上进行探索.熵正则化系数α显着地控制着勘探开发的权衡，较高的α对应于更多的勘探，较低的α对应于更多的开采。正确的系数(导致最稳定/最高回报的学习)可能因环境而异，可能需要仔细调整。在测试时，为了了解策略如何利用它所学到的知识，我们消除了随机性，使用了平均值操作，而不是从分布中选择一个样本。这倾向于提高性能比原来的随机策略。

## Pseudocode



## Documentation

## Saved Model Contents

## References

## Relevant Papers

## [Relevant Papers](https://spinningup.openai.com/en/latest/algorithms/sac.html#id13)

* [Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor](https://arxiv.org/abs/1801.01290), Haarnoja et al, 2018

## Other Public Implementations

* [SAC release repo](https://github.com/haarnoja/sac)