### 深度强化学习算法原理和基本算法

按训练方式分类，强化学习可分为两种：与环境交互的智能体与学习器合一的on-policy训练策略，与环境交互的智能体与学习器分离的off-policy训练策略。按是否对环境建模强化学习分为Model-based方法和Model-free方法，Model-based方法需要对环境建模，根据环境模型学习策略。而Model-free方法则不对环境建模，只通过学习值函数估计策略或直接学习策略。由于对金融市场的环境建模几乎是不可实现的，因此本研究主要集中于Model-free方法。Model-free强化学习大致可以分为三种方法policy-based方法,value-based方法,actor-critic方法。Policy-based方法直接对策略进行建模学习，故又称直接强化学习（Direct Reinforcement）,思路简单直观，但训练非常耗时，value-based方法针对动作-状态值函数进行建模，根据值函数来选择策略，而Actor-Critic方法用独立的值函数来表示策略函数，其中策略函数结构称为actor，值函数结构称为critic，actor根据当前环境的状态产生动作，critic根据状态和奖励产生TD(Temporal-Difference)误差信号，并输出actor。在深度强化学习中actor和critic都由神经网络训练。Policy-base方法主要有Policy Gradient（PG）、Proximal Policy Optimization（PPO）。Value-based方法主要有Deep Q-learning Networks（DQN）、Double Q-Network（DDQN）、Prioritized replay、Dueling network。Actor-Critic方法的主要代表由Deep Deterministic Policy Gradient（DDPG），Asynchronous advantage actor-critic（A3C）、Trust Region Policy Optimization（TRPO）。Policy-based具有稳定可靠的优点，但训练速度慢。Value-based方法则抽样效率高却不稳定。DDPG，A2C/A3C方法则取得了二者的平衡。

强化学习有六个关键的概念：智能体（agent）、环境（environment）、动作（action）、状态（state）、奖励（reward）、策略（policy）。智能体根据当前的状态采取确定或随机的动作与环境交互以获取奖励，根据状态选择动作的过程被称为策略，强化学习的目标就是采取最优的策略获得最大的期望奖励。一些场景中，动作的可能选择是可数的，如游戏中的前后左右、攻击后退等，这种情况称为离散动作空间，一些场景中，动作空间的选择是不可数的，如自动驾驶，这种情况称为连续动作空间。有时动作选择是确定的、唯一的，称这种情况为确定性策略；有时动作的选择服从一定的分布，智能体根据分布随机选择动作，称这种情况为随机性策略。其中还有一些概率如观测（observation），轨迹（trajectory）等，状态是对当前环境的完全刻画，而观测则是当前状态的部分描述，并不一定包含完整信息。轨迹是一次完整训练中所有状态和动作的序列。状态的转移仅取决于上一个动作，对应于随机策略或确定性策略，状态的转移也可以是随机的或确定性的，每次状态转移的奖励仅取决于上次的状态-动作组合，智能体的目标就是最大化一个轨迹序列的最大累积期望奖励，最大累积期望奖励也分为两种：有限期无贴现奖励和无限期有贴现奖励。

强化问题的公式化描述如下：

记为动作，对于确定性动作，对于随机动作，记为轨迹，记初始状态分布为，对于确定性状态转移，对于随机状态 ，，对于有限期无贴现奖励，对于无限期有贴现奖励，假设状态转移和策略都是随机的，一个T步轨迹的概率为：

从而强化学习的目标函数即为：

优化函数目标即为：

根据动作是确定性的还是随机性的，值函数分为状态值函数和动作-状态值函数，其中状态值函数指的是从状态出发，而后一直遵循策略选择动作；而动作-状态值函数指的是从状态出发，自主选择一个动作(可能不产生自策略)，然后一直遵循策略（即每个动作都是从中抽取），因此on-policy状态值函数和动作状态值函数的联系为：

对于状态，根据状态-动作值函数可得到最优动作:

强化学习优化的基础是Bellman方程，on-policy状态值函数和off-policy动作状态值函数的bellman方程可写为：

其中是状态的转移概率分布。

从而优化问题转化为：

强化学习另一个核心的概念就是优势函数（advantage function），它衡量的是从某个状态出发，自主选择一个动作比根据策略抽取一个动作所带来的奖励有多大，即状态-动作值函数和状态函数的差：

Policy-based方法直接对策略函数建模，通过对目标函数或其代理函数进行梯度上升优化更新产生，**该类方法几乎只能执行on-policy训练，即每次训练只能基于上一次训练完成才能进行**。典型的Policy-based方法有策略梯度（Policy Gradient）、近端梯度下降优化（Proximal Policy Optimization，PPO）。其中PPO并不是直接优化目标函数，而是优化其代理函数，该代理函数是更新参数后的保守估计。

而Value-based方法则是对动作-状态值函数进行建模,通常该目标函数是以贝尔曼方程的形式优化。不同于policy-based方法，value-based方法是可以通过off-policy来训练。即每次可以用任意时点搜集的数据进行更新，不管获得数据后智能体如何对环境进行探索。对应策略则可以根据最大化Q函数获得，即：

Actor-Critic方法用于独立于值函数来表示策略函数。其中策略函数结构称为actor，值函数结构称为critic，actor根据当前环境的状态产生动作，critic根据状态和奖励产生TD(Temporal-Difference)误差信号，并输出actor。在深度强化学习中actor和critic都由神经网络训练。Actor-Critic方法的主要代表由DDPG，A2C/A3C。

Policy-based具有稳定可靠的优点，但训练速度慢。Value-based方法则抽样效率高却不稳定。DDPG，A2C/A3C方法则取得了二者的平衡。

传统的强化学习问题的求解策略通常分为三类：1，动态规划方法(Dynamic Planning)；2，蒙特卡洛法(Monte Carlo)；3，时序差分法(Temporal Difference)。动态规划法需要在每一次回溯更新某一个状态的价值时，回溯到该状态的所有可能的后续状态。导致对于复杂问题计算量很大。同时很多时候，我们连环境的状态转化模型P都无法知道，这时动态规划法根本没法使用。而蒙特卡罗法很灵活，不需要环境的状态转化概率模型，但是它需要所有的采样序列都是经历完整的状态序列。如果我们没有完整的状态序列，那么就无法使用蒙特卡罗法求解了，而时序差分就是可以不使用完整状态序列求解强化学习问题的方法。

#### 1、Policy-based算法

##### （1）、Policy Gradient（PG）

PG方法并不对值函数Q进行建模，而是直接对策略进行建模，其目标函数即：

从而VPG目标函数关于参数的梯度为：

从而策略参数的更新方程为：

VPG以on-policy的方式训练，即根据最新的随机策略梯度采样来进行动作探索，故动作选择的随机性取决于初始状态和训练过程。随着训练的轮数越来越多，对已知拥有较高奖励的概率就会越来越高，从而随机性越来越小，可能会出现训练不足的情况，从而陷入局部最优，这是VPG的一个缺点。

VPG的算法流程如下：

1, 输入：初始策略参数，初始值函数参数，训练轮数N

2, for k=0,1,2,…N:

3, 根据策略与环境交互获取训练集;

4, 计算当前奖励;

5, 基于当前值函数估计当前动作的优势函数 ;

6, 根据策略和优势函数计算策略梯度  ;

7, 根据策略梯度更新策略函数参数：;

8, 根据计算的值函数和当前奖励利用回归方法更新值函数的参数：

##### （2）、Trust Region Policy Optimization（TRPO）

TRPO的基本思想是限制每次策略梯度的更新步长，该更新由两个连续概率分布的Kullback-Leibler（KL）散度度量：

把更新限制在信任区（trust region）从而防止更新后的策略偏离前序策略太远。为参数化随机策略的表现，则它的下界函数可表示为：

其中为概率校正后的期望优势函数，使用期望优势函数而不是期望奖励函数可以减少估计方差。TRPO的原理如下：用L来近似当前策略的优势函数的准确度将随着策略远离旧策略而降低，但下降存在一个上限，记M的第二项。因此，我们可以应用Minorize-Maximization算法进行迭代优化得到，如此，的提升是单调的。但在实际中，求KL散度的最大值是非常困难的，因此实际种经常用均值代替。此外，惩罚系数C的值几乎无法使用，因为其理论结果将提供步长过小的策略更新，因此，TRPO使用作为代理目标函数，而约束策略更新的散度不超过某一确定值，记：

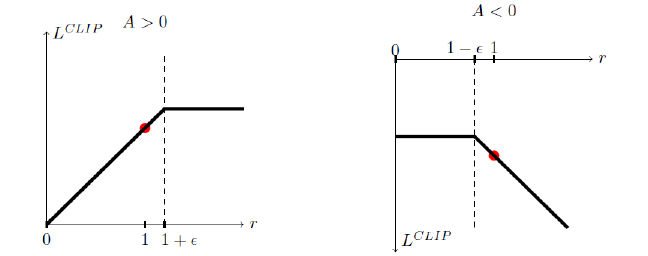
由于需要计算二阶随机梯度及其逆，所以TRPO计算较为复杂。

##### （3）、Proximal Policy Optimization（PPO）

PPO是在TRPO的基础上提出的，由于只需要计算一阶随机梯度，PPO的计算较TRPO大大简化。PPO的无约束代理目标函数为：

其中是用于限制范围的超参数。

PPO的目标函数的约束机制如下：当优势函数时，意味着当前动作对输出有正向作用,在这种情况下，如果该动作在当前策略的概率高于在前序策略的概率，则目标函数变为常数，梯度为0 ，从而防止随机梯度上升算法偏离当前策略，也就使当前动作的概率不会进一步上升。而如果，即该动作在当前策略的概率低于在前序动作的概率，策略变得更差，则PPO的目标函数的梯度仍维持为正，从而通过梯度上升算法修正偏差。当优势函数时，意味着当前动作对输出有负向影响，如果,即该动作在当前策略中的概率小于在前序策略的概率，则PPO目标函数的梯度为0，从而防止动作的概率进一步下降，保证稳定性；如果，则PPO目标函数的梯度为负，从而通过梯度上升法修正偏差。



#### 2、Value-based类算法

##### （1）、Deep Q-learning Networks（DQN）

Q-learning是最早版本的value-based深度学习方法，它并不是直接学习策略，而是对动作值函数进行学习，在传统的框架下（无论是确定性还是随机性框架）它的收敛性都是有保证的。但鉴于神经网络的强大的表示能力，因此目前Q函数通常以神经网络为学习器，当使用深度神经网络时，我们就称之为DQN(Deep Q-Networks)而非线性学习器的引入则无法保证收敛性。Q-learning对最优动作值函数的表示记为，则在第i次迭代中的损失函数可写为：

从而

根据min 原则更新网络参数。

由于Q函数并不是根据策略行为生成的状态转移更新，所以Q-learning可以由off-policy的方式进行训练。DQN通常以CNN为学习器，因为CNN具有强大的表示能力，可以学习高维状态的低维特征表示，DQN的探索-开发算法是贪心算法，使用从replay buffer中选择一小批转移样本,以的概率随机选择一个动作，从而可以减少样本的序列相关。完成学习后，智能体就可以从网络的单个前向转移中选择一个最优动作.这种程序也正**是Q-learning不能直接应用于连续动作空间的原因：value-based方法要求每一步都迭代地优化动作-值函数，这在连续动作空间是不可行的**。

DQN的算法流程如下：

1，输入迭代轮数T，状态特征维度n, 步长α，衰减因子γ, 探索率ϵ, 初始化Q网络结构, 批量梯度下降的样本数m，replay buffer ；

While i<T:

2， 在当前Q网络中输入状态s,计算所有动作对应价值函数，使用-贪 心算法选择当前动作a;

3, 在状态s下执行当前动作a,得到新的状态，和对应奖励r,及当前状态是否是终止状态表示is\_end;

4, 将存入replay buffer ;

5, 从replay buffer中抽取m个样本,计算

6， 根据更新网络参数；

7， 如果是终端状态则本轮循环结束，否则继续执行2~6；

#### 3、Actor-critic类算法

##### （1）、Asynchronous advantage actor-critic（A3C）

所谓actor-critic框架是在PG的基础上，用Q函数来估计奖励，即使用critic网络计算Q函数，来评价选择动作的优劣。从而对于actor-critic来说critic网络的损失函数为：

可以看出该损失函数与double DQN方法的损失函数一致。

而对于actor网络来说，策略的梯度则由变为：

即不是从短期的优势函数出发更新参数，而是从长期的Q函数出发更新参数。

经典的Actor-critic收敛性很差，使用on-policy的训练方法，训练速度也较慢。为解决这个问题A2C(advantage actor-critic)引入了Baseline的概念，即将Q函数更改为优势函数A作为奖励的Baseline，从而actor网络的策略梯度由又变为标准PG方法的策略梯度：

但此时critic网络的估计目标变为在既定的动作下极小化估计状态值函数V和实际值之间的差异，从而critic网络的损失函数变为：

但A2C收敛速度仍然不尽如人意，为加快收敛速度A3C使用了replay buffer技术，并基于global network思路克服了传统replay buffer的不足，即使用多线程技术 同时在多个线程里分别和环境进行交互学习，每个线程都把学习的成果汇总保存到一个公共的global network中，这个公共的global network也包括actor网络和critic网络，与各个线程里的子网络结构相同。每个线程和环境交互到一定数量后，就计算自己线程里的损失函数的梯度，但这些梯度并不更新自己线程的网络，而是更新公共部分的神经网络。每隔一段时间，各线程将自己网络的参数与公共global network的参数同步，进而指导后面的环境交互。因此global network的网络才是目标网络，各个线程里的网络主要是为了与环境进行交互。此外，A3C与A2C另一点不同的地方在于，在利用策略梯度对参数进行更新中，加入了策略的熵值，即参数更新为：

其中，为策略的熵值。

A3C方法的算法流程如下：

1. 输入公共global network的策略参数，Q函数参数，全局最大迭代次数，线程内单次最大迭代时间序列最大长度，步长、熵系数、贴现率;

while :

1. 令线程网络参数等于公共网络参数θ\_i←θ,ϕ\_i←ϕ；

while or :

3, 基于策略选择动作;

4, 观测下一个动作，并获得奖励;

5, 计算状态值函数

6， 计算;

7, 计算actor的累积梯度更新：

;

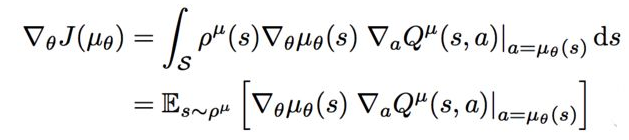
8, 计算critic网络的累积梯度更新：

9, 更新global network的参数：

##### （2）、Deep Deterministic Policy Gradient（DDPG）

DDPG是由谷歌DeepMind在David Silver et al.(2014)的确定性策略梯度的基础上提出的，是一种model-free在actor-critic框架下的深度强化学习算法。

David Silver et al.(2014)证明对于确定性策略：



对比随机策略梯度公式，确定性策略梯度公式少了对动作求期望，多了值函数Q对动作的导数。随机策略梯度里，需要对动作和状态求积分，而在确定性策略梯度里只需要对状态求积分，这就决定了求确定性策略梯度只需要更小的样本，特别是动作空间很大的时候。

在强化学习算法中，**很多时候做策略更新的时候是贪心的，即对Q值取最大值max Q(s, a)。但如果是在连续动作的强化学习问题里，对Q求最优值往往不是很容易，或者计算量很大。这也是Q-learning和DQN不好解决高维连续动作问题的原因**。而在确定性策略梯度里，其梯度的更新可以视为策略往Q值梯度的方向走，这样就一步步往Q值增大的方向走。具体而言，确定性策略梯度更新如下，然后用链式法则可得到上述的确定性策略梯度公式：

DDPG为实施更好的探索对确定性策略加入了随机噪音，但估计中使用了更容易学习的确定性目标策略。它同时学习Q函数与策略，Q函数的学习使用了off-policy数据和贝尔曼方程，而策略的学习则基于Q函数。DDPG非常适合连续动作空间环境的学习，原因在于连续的动作空间环境中，是可微的，因而可以根据梯度更新策略，由于代表了当前的最优动作，因此我们可以使用如下近似。最优动作值函数的贝尔曼形式可写作：

记为的近似，以均方贝尔曼误差（mean-squared Bellman error，MSBE）衡量接近贝尔曼方程的程度：

其中，代表该动作是否是终节点，Value-based方法都是基于对该函数求极小值。

Replay buffer

所有训练深度神经网络近似的方法都使用了replay buffer，即。如果使用最近的数据，则会导致过拟合；如果使用太多数据，则学习速度会下降。

目标网络

DDPG对贝尔曼方程中函数所拟合的部分也使用了网络结构，称为目标网络：

形式上该网络依赖于的参数，但若该网络的参数也使用与相同的参数，则拟合的结果将极不稳定。解决的办法就是使用参数的滞后，目标网络的参数记为。在DQN算法中，目标网络的参数在固定轮数之后更新，但DDPG算法中，目标网络的更新是每次主网络更新之后都更新，方法是polyak平均：

在连续动作空间中针对动作求极大值是非常困难的，DDPG使用了目标策略网络来计算某个动作的Q函数值来对进行近似，目标策略网络和目标Q函数一致，对训练过程的策略参数使用polyak平均求得，基于目标网络的设定，值函数的优化目标即：

其中为目标策略网络，其参数更新可以使用梯度上升法，这即是DDPG的策略梯度特征:

DDPG使用off-policy方法训练确定性策略，由于策略是确定性的，如果智能体进行on-policy训练，在训练的开始阶段，动作的样本量可能不足。为增加动作的多样性，DDPG在动作上增加了噪音，DDPG原作者推荐Ornstein-Uhlenbeck（OU）过程噪音，OU噪音是一种序列相关噪音，其公式为，其中是该过程的均值和标准差，是维纳过程。但最近的研究表明，简单的高斯噪音表现也非常优良。综上，DDPG的算法流程如下：

1. 输入初始策略参数，Q函数参数，replay buffer，收敛误差
2. 令目标参数等于主参数

3, while :

4, 观测状态选择动作

5, 在环境中执行动作a；

6，观测下一动作，奖励;

7，将存入replay buffer

8，如果是终点状态，重置环境状态；

9，确定是否需要更新参数 ，如果需要更新则执行：

10，for i=1,2,…,N:

11， 选择训练样本

12， 计算目标网络

13， 计算Q函数的梯度,更新critic网络；

14, 计算策略的梯度，更新actor网络；

15, 更新目标值函数网络和目标策略网络