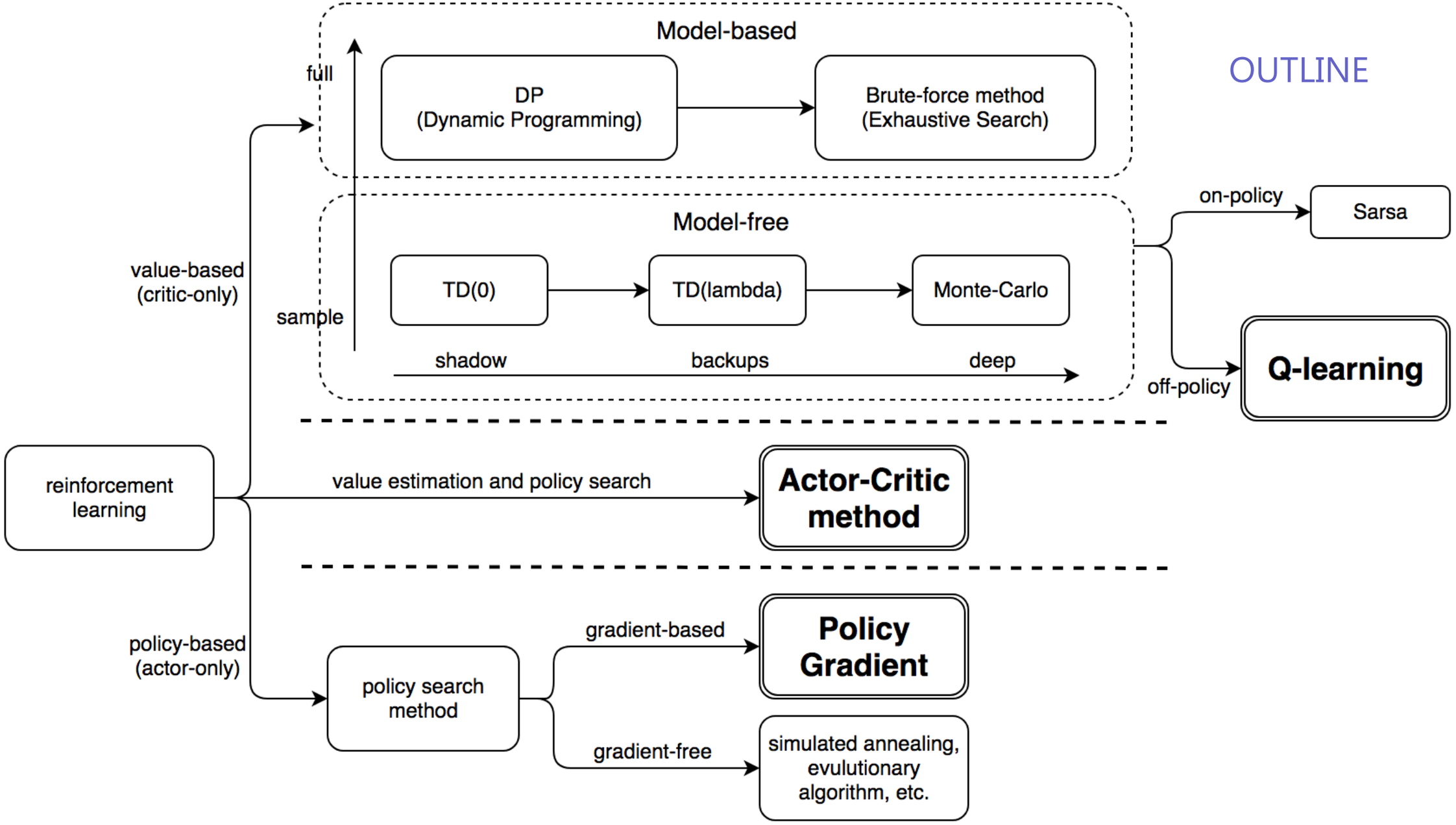
强化学习引论

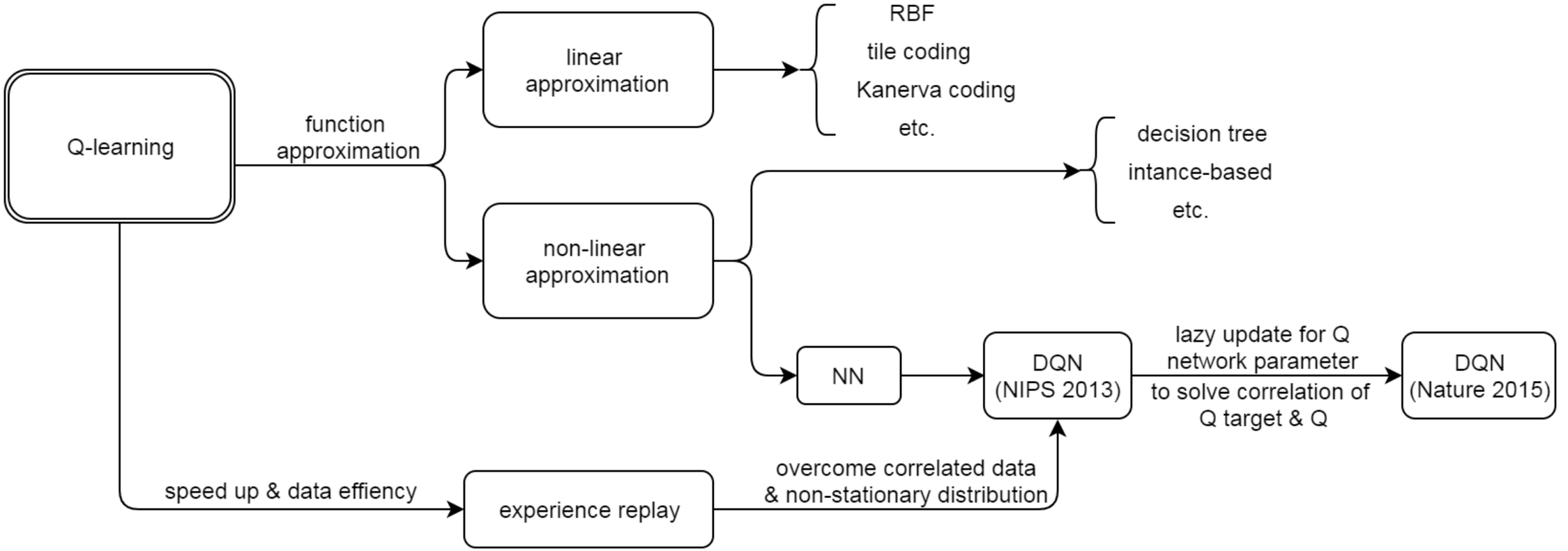
- Author： Sinyer
- sinyeratlantis@gmail.com

OUTLINE

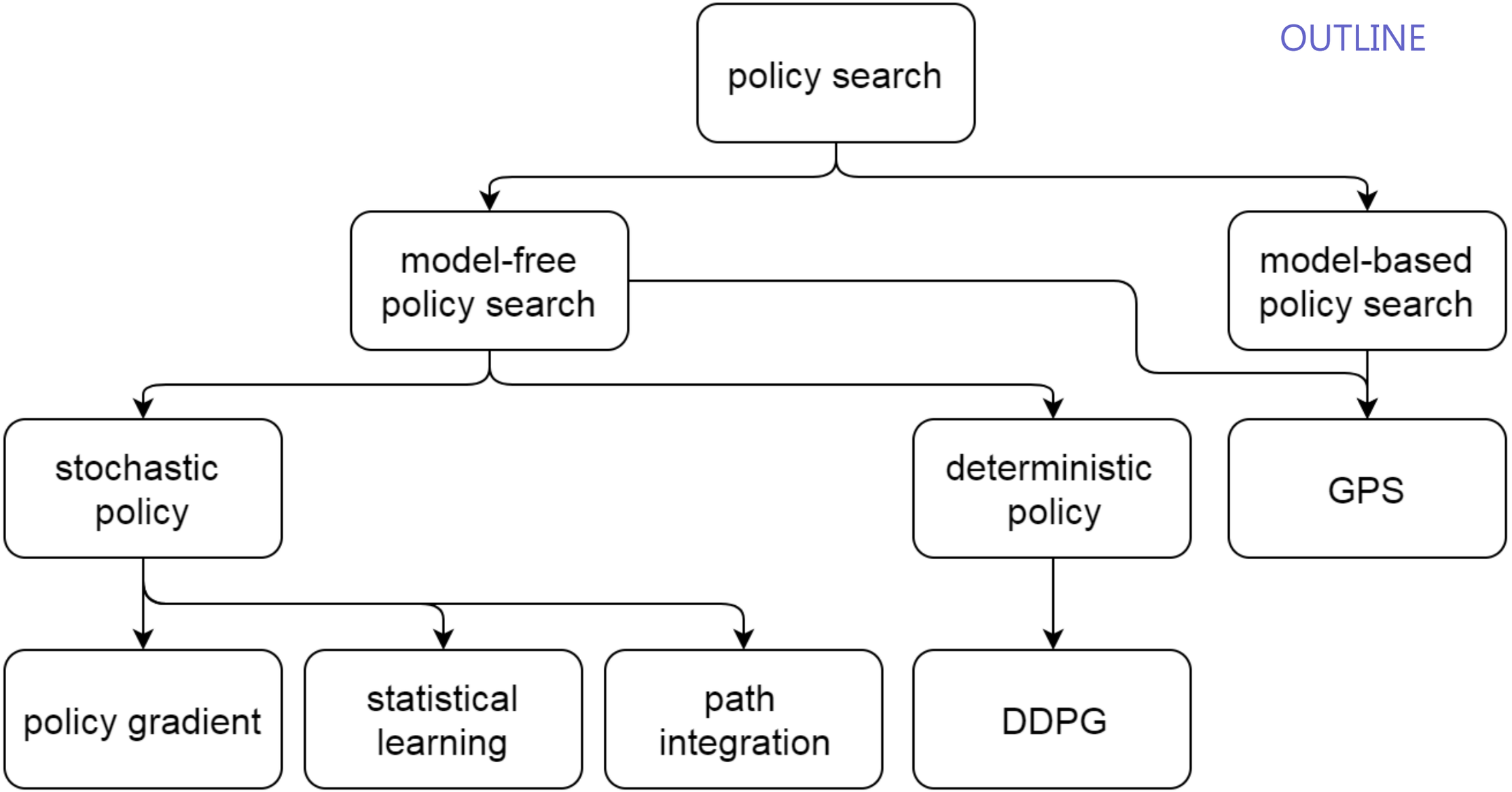
- Introduction
- Dynamic Programming
- Monte-Carlo Method
- Temporal-Difference Method
- Policy Gradient


OUTLINE
Model-based
full
DP
(Dynamic Programming)
Brute-force method
(Exhaustive Search)
Model-free
sample
TD(0)
TD(lambda)
Monte-Carlo
shadow
backups
deep
on-policy
Sarsa
off-policy
Q-learning
value-based
(critic-only)
reinforcement
learning
value estimation and policy search
Actor-Critic
method
policy-based
(actor-only)
policy search
method
gradient-based
Policy
Gradient
gradient-free
simulated annealing,
evulutionary
algorithm, etc.

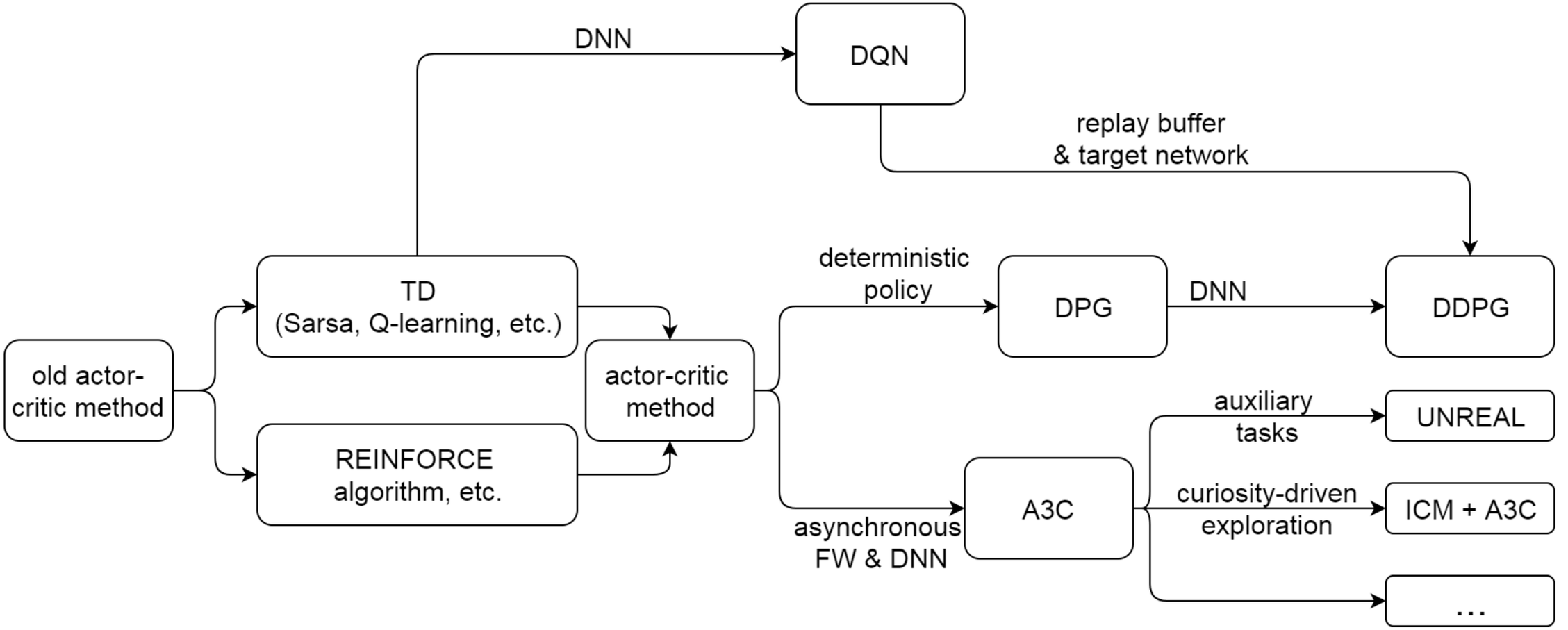
OUTLINE

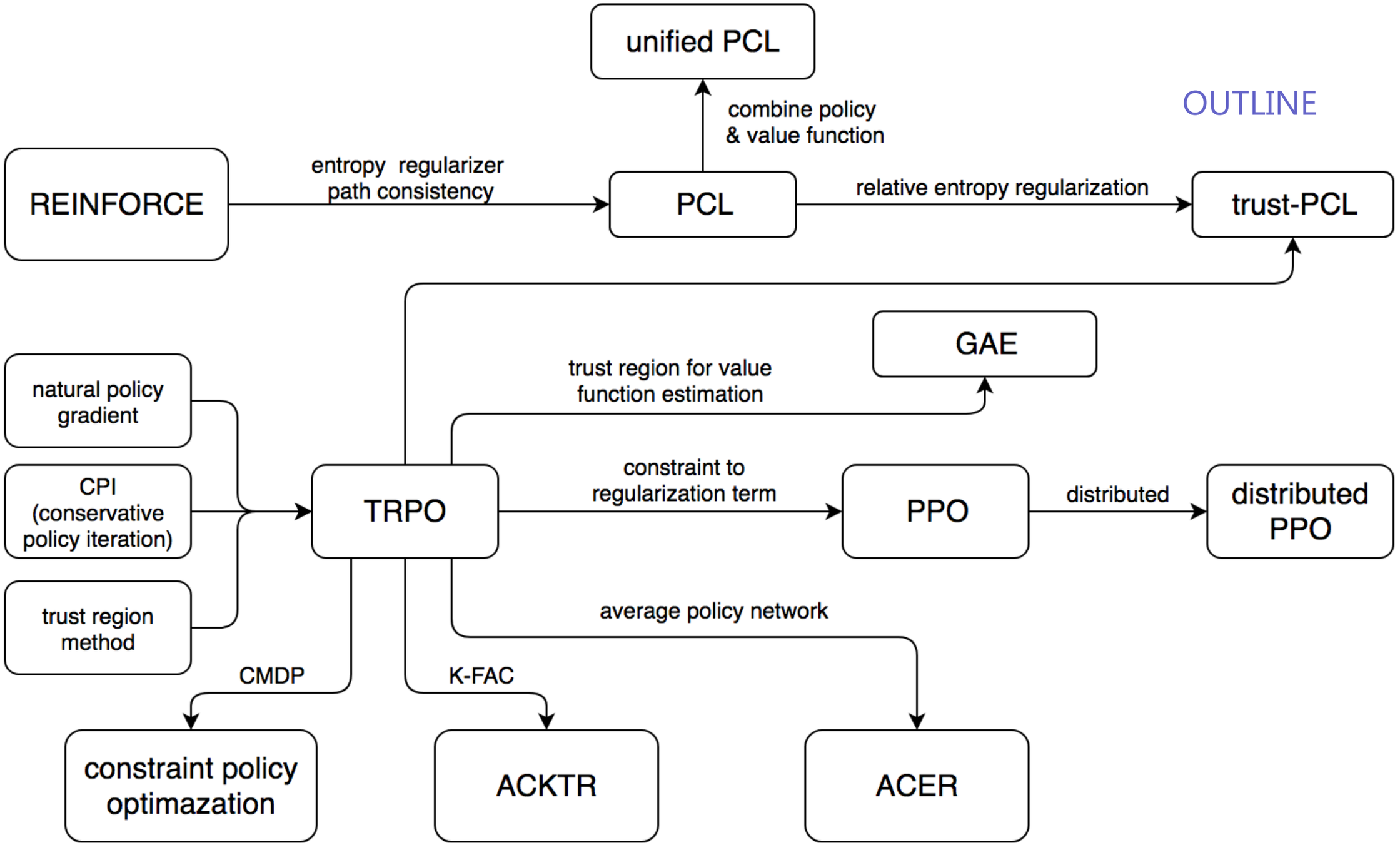

Q-learning
function approximation
linear approximation
RBF
tile coding
Kanerva coding
etc.
non-linear approximation
decision tree
intance-based
etc.
NN
DQN (NIPS 2013)
lazy update for Q network parameter
to solve correlation of Q target & Q
DQN (Nature 2015)
speed up & data effiency
experience replay
overcome correlated data & non-stationary distribution

OUTLINE


policy search
model-free policy search
model-based policy search
stochastic policy
deterministic policy
GPS
policy gradient
statistical learning
path integration
DDPG

OUTLINE


DNN
DQN
replay buffer
& target network
TD
(Sarsa, Q-learning, etc.)
old actor-
critic method
REINFORCE
algorithm, etc.
actor-critic
method
deterministic
policy
DPG
DNN
DDPG
auxiliary
tasks
UNREAL
A3C
curiosity-driven
exploration
ICM + A3C
asynchronous
FW & DNN
…


OUTLINE
unified PCL
combine policy
& value function
REINFORCE
entropy regularizer
path consistency
PCL
relative entropy regularization
trust-PCL
GAE
trust region for value
function estimation
natural policy
gradient
CPI
(conservative
policy iteration)
trust region
method
TRPO
constraint to
regularization term
PPO
distributed
distributed
PPO
average policy network
CMDP
K-FAC
constraint policy
optimazation
ACKTR
ACER

除了以上框架，广义的基于模型的方法，个人认为是一个比较独立的领域，与监督学习、无监督学习、甚至计算机视觉等方向关联较为紧密，而与强化学习的初衷：序列决策问题相离较远，在这里不做过多讨论，还有依赖监督数据，多智能体（属于工程问题）等问题也不做讨论。这些领域包括引导策略搜索GPS（通过轨迹优化产生监督数据进行策略搜索），逆向强化学习（通过监督数据学习reward函数），PILCO及其扩展等。

OUTLINE

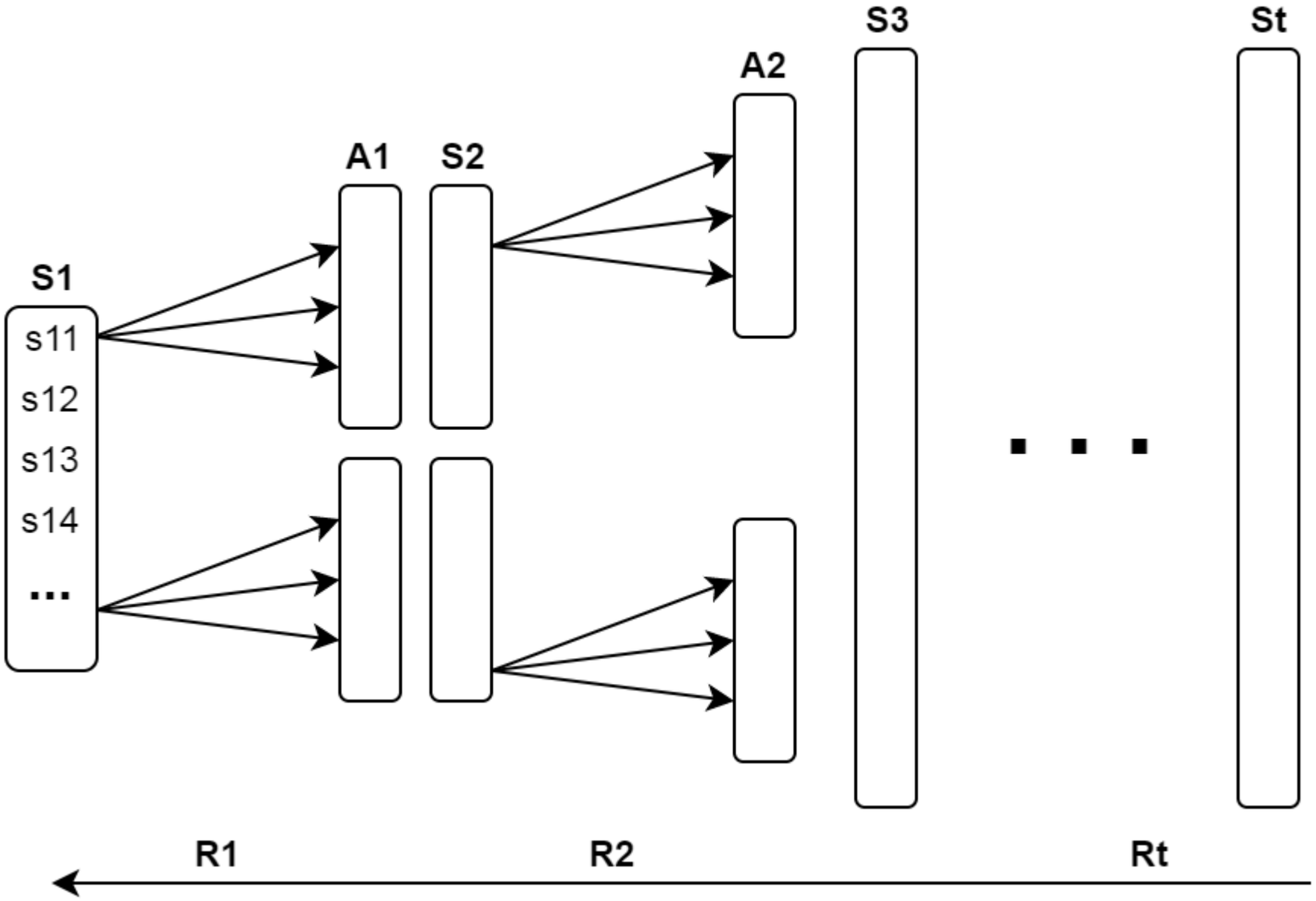
主流算法为DPG系算法以及PPO系算法，本系列文档主要即介绍这两系算法及其理论基础与扩展。

本文档主要讲解这些算法最基本的理论基础。

对于文档内容，在下尽力追求客观，但限于自身能力，其中免不了含有大量个人的理解，加之强化学习并非本人专精领域，若有疏漏，还请见谅，欢迎各位指出，与在下讨论交流。

Introduction

1. Introduction



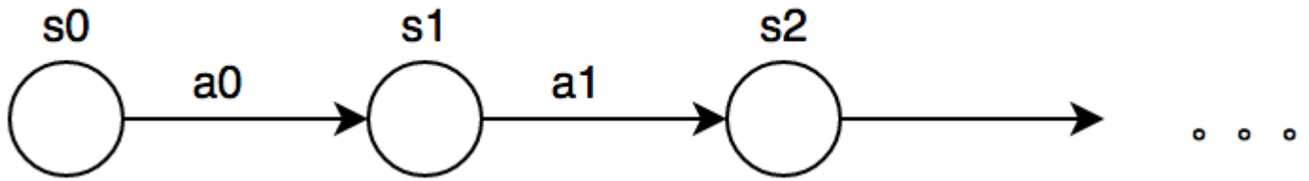
上图为离散单向的状态转移图，所有的状态为状态空间，所有的动作为动作空间。由一个状态采取动作，转移到下一个状态，不断进行，其中状态转移会返回即时奖励，一个策略决定如何选择动作。当先前的状态对现在的状态没有影响时，动作的选择仅与当前状态和策略相关，此即MDP，Markov Decision Process。

强化学习的目标，就是要解决从初始状态到最终状态获得最大累积奖励的序列决策问题。

1. Introduction

最直观的方法就是穷举法，计算出每种序列的累积奖励，然后找出最大值。考虑减少计算量，可以采用迭代的方法估计最优决策。

首先是动态规划法。考虑下图简单情形：



设定每一个状态均有一个状态价值，定义其为其累计奖励：

$$v(s_t) = r_t + r_{t+1} + r_{t+2} + \cdots$$

显然，最优的策略就是状态转移时选择状态价值最大的状态。

考虑所有下一状态价值已知的话，当前状态价值可表示为：

$$v(s) = r + v(s')$$

实际上，每个状态下的动作选择往往不唯一，我们也才需要选择策略。状态价值定义上为某条确定路径的累积奖励，所以，要得到定义的状态价值，动作选择必须是确定的。不难发现，不同的动作选择，状态价值也不一样，所以说，状态价值是关于策略的函数。

扩展到广义的状态价值的定义，我们策略的输出不一定必须是确定的动作，而是可以表示为动作概率的分布$\pi(a|s)$。概率是确定的，所以我们可以定义状态价值为不同路径累积奖励依概率的期望：

$$v_\pi(s) = \mathbb{E}[G_t|S_t = s]$$

展开为：

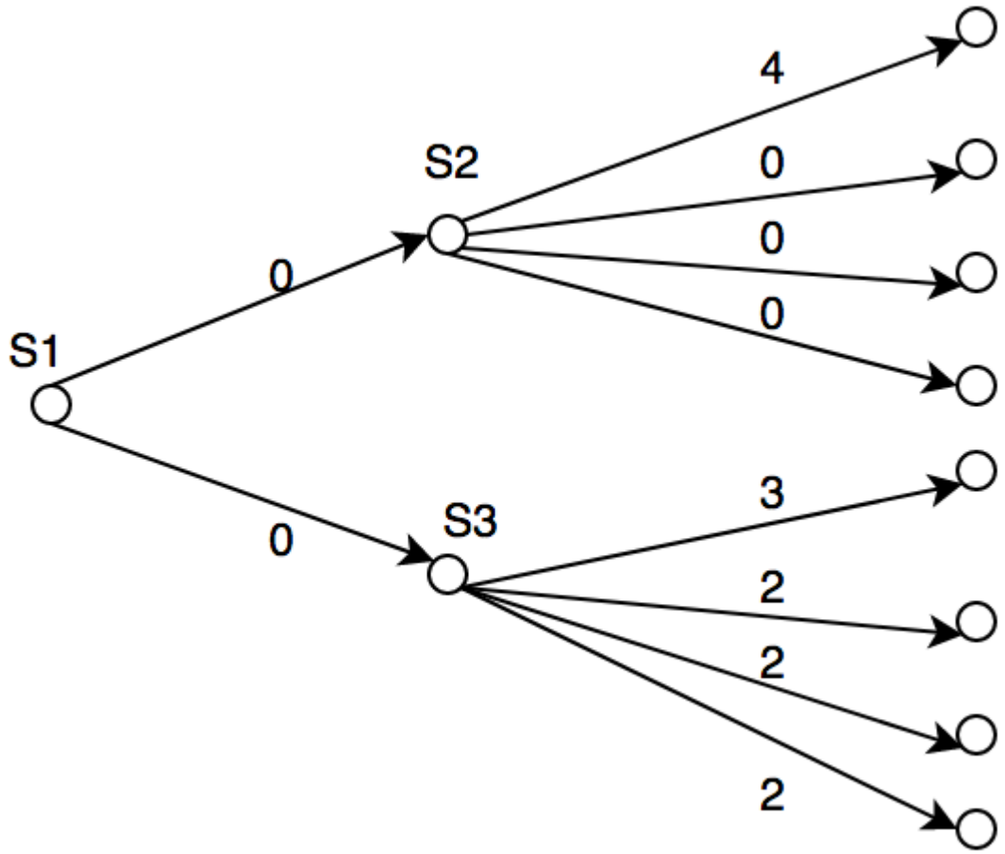
$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + v_\pi(s')]$$

即当前状态价值为所有下一状态的价值与其奖励之和的概率加权和。

这里，本人曾陷入一个误区，认为在状态转移不唯一的情况下时，只需要考虑均匀分布的情况，在每种选择概率都一样的情况下，状态价值理应是准确无误的，若是均匀分布都不准确，其他分布怎么会更准确呢。这里便反映了当初本人对状态函数理解不够深刻。

状态函数衡量的，一定是确定序列的累积奖励。在策略不同序列也不同的情况下，状态函数也不是唯一的，只有给出了策略，序列分布确定了，才能给出唯一的期望状态价值。也就是，均匀分布下的状态价值的确计算无误，但仅属于均匀分布策略。

举个简单的例子，如下图所示：



若按均匀分布计算状态价值，则S2价值为1，S3价值为2.25，则S1选择S3然后选择奖励为3的路径，策略完成，但实际我们发现选S2奖励为4的路线总奖励更高，这就说明了我们的决策，一定求得的是单条路径的最高奖励，而不是所有路径奖励的平均。

其实，我们从上图便可以看出，若是对于单条路径而言，S2的状态价值应该为4，S3的状态价值应该为3，均匀概率的选择，使状态价值被无用路径的价值拉低了，真正反映状态价值的，应该是最优路径。

从这里我们便可看出，能够用于决策的状态价值，应该是最优状态价值，即最优路径下的状态价值，我们用$v^*(s)$表示，此时对应的策略，应是最优策略π^*。

但问题是，不给定策略则无法计算状态价值，而给定策略后计算出的状态价值却只属于该策略，更不用提依据状态价值来得到最优策略了。那，究竟如何使用状态价值呢？

我们发现，状态价值最准确的衡量，即是最大的衡量，若是对每个状态价值都有$v_\pi(s) \leq v_{\pi'}(s)$，那$\pi'$显然是更优的策略，这样，我们就得到了提升策略的依据：

$$v_*(s) = \max_\pi v_\pi(s)$$

由$v_\pi(s)$的定义可得：

$$v_*(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + v_*(s')]$$

对单向序列决策问题，若s'全部为终止状态时，$v_*(s) = \max_a \sum_{s',r} p(s', r|s, a) \cdot r$，由此迭代即可得到最经典的自底向上的动态规划法。

另外，考虑连续任务或带回路的序列决策问题，容易出现初始状态的价值接近无穷大，对以后的优化很不利，所以目前主流的做法是用一个折扣因子乘以未来状态价值以衰减未来的影响，避免状态价值发散。折扣因子应视情况而定。

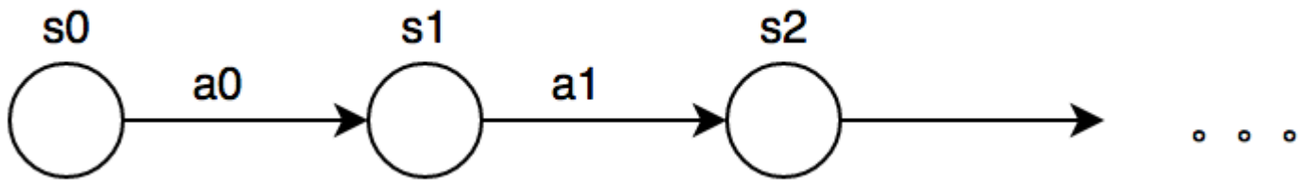
Dynamic Programming

2. DP

实际中，带回路的序列决策问题很常见，状态的最优价值可能依赖于自身的价值，要求明确所有下一状态的最优价值的自底向上法显然行不通。为了解决这个问题，我们采用了一种近似拟合最优策略的方法，即强化学习中的动态规划法。（这里文献中的说法是max操作是非线性方程，往往没有闭环解，所以常常使用迭代的方法来解）

动态规划法通过迭代去近似拟合状态的价值，具体而言，相比比较所有策略的穷举法，动态规划不直接得到最优策略，而是初始化一个随机策略，然后通过优化它逼近最优策略。

首先是策略评估。



依旧是简单情况，假定序列$(s_0, a_0, s_1, a_1, ...)$为整个任务唯一的序列，累积奖励即为$\sum r$。如上图所示，action即为向右转移，在这种唯一情况下，状态价值即累积奖励。若令终止状态为s_3，令转移奖励分别为0，1，2，则$v(s_0) = 3$，$v(s_1) = 3$，$v(s_2) = 2$。

动态规划首先初始化所有状态价值为0，对每一个状态，直接通过$v(s) = r + v(s')$更新当前状态的价值，若表示初始状态为$t = 0$，则上式可扩展为$v_{t+1}(s) = r + v_t(s')$，如：

$$
\begin{aligned}
v_0(s_0) &= 0 + 0 = 0 \\
v_0(s_1) &= 1 + 0 = 1 \\
v_0(s_2) &= 2 + 0 = 2 \\
v_1(s_0) &= 0 + v_0(s_1) = 1 \\
v_1(s_1) &= 1 + v_0(s_2) = 3 \\
v_1(s_2) &= 2 + v_0(s_3) = 2
\end{aligned}
$$

$$
\begin{aligned}
v_2(s_0) &= 0 + v_1(s_1) = 3 \\
v_2(s_1) &= 1 + v_1(s_2) = 3 \\
v_2(s_2) &= 2 + v_1(s_3) = 2 \\
v_3(s_0) &= 0 + v_2(s_1) = 3 \\
v_3(s_1) &= 1 + v_2(s_2) = 3 \\
v_3(s_2) &= 2 + v_2(s_3) = 2
\end{aligned}
$$

可以看到，通过不断地迭代，状态价值收敛为真实状态价值。即对于一个策略，通过不断地迭代，状态价值将收敛于该策略下的真实状态价值。

评估完状态价值，我们便需要提升我们的策略：

$$v_*(s) = \max_{\pi} v_\pi(s)$$

我们直接采用贪婪策略的方法按当前状态价值直接提升。

policy iteration在每一次迭代都需要进行policy evaluation，也就是说每次迭代均需要对所有状态集进行遍历，这是非常耗时的，实际上policy evaluation没有必要一定converge到v_π，有几种方法能提前终止policy evaluation而不影响policy iteration的收敛，其中在policy evaluation仅对每个状态更新一次的方法，我们称之为value iteration，通过Bellman optimality equation也可以得出value iteration方法。

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $v \leftarrow V(s)$
| $V(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$
| $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \arg\max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$

Monte-Carlo Method

3. MC

动态规划需要对环境完全的知识，但对于一个未知的MDP，则无法用DP的方法求解，这类问题我们称为Model-free prediction。在model-free的情况下，直接估算$Q(S,A)$更常见，因为即便估算出来了$V(S)$，没有model还是不知道如何选择action（如何生成policy）。

有两类方法用于求解这类问题：

- Monte-Carlo Learning（no bootstrapping）
- Temporal-Difference Learning（with bootstrapping）

Monte Carlo Prediction（policy evaluation）

先按照要evaluate的policy π 的指示走到episode结束，然后用empirical mean return来估算expected return。

每个状态下的所有动作都需要估计其价值，所以所有状态访问无限多次，以保证收敛到V_π，通常采用两种方法：

- exploring starts，即开始的(s, a)要随机初始化，保证所有(s, a)都能访问到，但有些实际环境难以应用。
- 在每个state，采用类似于 ε - greedy的方法（而不是采用greedy方法选择action），从而保证所有action，即所有(s, a)可以选择到。

Monte Carlo Control

基于generalized policy iteration（GPI）

policy evaluation时：根据Monte Carlo Prediction方法，计算出$Q_\pi(S,A)$

policy improvement时：

- 基于exploring starts方式。
- 基于 ε - greedy方式。

基于exploring starts方式，可以找到 Q^*/π^*，但是基于 ε - greedy方式，只能找到所有 ε - greedy policies里面最优的 π^* ，好处是不再需要exploring starts。

Off-policy Monte Carlo Prediction：

根据behavior policy b 产生的样本估算target policy π 的 V_π。此时要考虑两个policies产生同一个样本的概率是不同的，用Importance Sampling来平衡对应的return，Importance Sampling比率最终仅取决于两项policies，而不取决于MDP。

$$\rho_{t:T-1} \doteq \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}$$

两种使用Importance Sampling的方法：

- Ordinary/Average Importance Sampling
- Weighted Importance Sampling

这两种方法均可通过incremental的方式on-line实现。

实际上，我们非常倾向于选择加权估计，因为它通常具有明显较低的方差。 尽管如此，我们也不会完全放弃ordinary importance sampling，因为使用function approximation更容易扩展到approximate methods。

Off-policy Monte Carlo Control：

Prediction过程基于behavior policy b 学习 Q_π。

对target policy π 做policy improvement：

$$\pi(S_t) = \arg\max_a Q_\pi(S_t, a)$$

MC-Learning的本质是每个样本的权重一样大，但是对于incremental的更新形式，$(G_t - V(S_t))$ 的step-size却越来越小，其具体形式为 $lr_t = 1/N(S_t)$。

而对于non-stationary的问题，让 $lr_t = constant$ 反而更合适一些，因为 $lr_t = constant$ 本质上意味着离现在越近的reward对于$V(s)$的贡献越大，而实际问题中更多的是non-stationary的，所以直接设置 $lr_t = constant$ 很常见。

Temporal-Difference Method

TD-Learning的特点：走一步，用下一个状态的估值来estimate当前状态的估值。

- MC method：利用采样得到的总回报G_t来估计状态价值的期望

  $$V(S_{t+1}) = V(S_t) + \alpha[G_t - V(S_t)]$$

- TD(0) / one-step TD method：

  利用采样的得到的$R_{t+1} + \gamma V(S_{t+1})$来估计状态价值的期望，且$V(S_{t+1})$也是同样的估计。

  TD-target = $R_{t+1} + \gamma V(S_{t+1})$

  TD-error = $R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$

对于采样平均方法，考虑next sate value为0的简单情况，对n次采样，有：

$$Q_{n+1} = \frac{1}{n}\sum_{i=1}^{n} R_i = Q_n + \frac{1}{n}[R_n - Q_n]$$

TD-learning可以看成如下的形式：

$$NewEstimate \leftarrow OldEstimate + StepSize[Target - OldEstimate]$$

本质上对TD-error的降低等同于让target的估计向target的方向前进一步。

令α表示$\frac{1}{n}$，可得：

$$Q_{n+1} \doteq Q_n + \alpha[R_n - Q_n] = (1-\alpha)^n Q_1 + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} R_i$$

α即step-size，可以看出来constant step-size本质上是对不同step得到的reward进行加权平均，而且离现在越近，权重越大。

在实际实现过程中，我们会考虑逐步减小参数step-size α 的值，假设n时刻的step-size为 $\alpha(n)$，为保证Q的计算收敛，我们可以利用随机近似理论中的一个结论：

- $\sum_n \alpha(n) == \infty$（$\alpha(n)$足够大能够纠正任何初始状态与随机波动）
- $\sum_n \alpha(n)^2 < \infty$ （步长逐渐减少）

sample-average中，$\alpha(n) = 1/n$ 满足步长递减条件，能够保证收敛，但是 $\alpha(n) = \alpha$ 的恒定步长却不满足这个条件，所以不能保证收敛，但constant step-size $\alpha(n) = \alpha$ 意味着weighted R_i，而且R_i离现在越近权重越大，使Q值更强烈地响应更近的reward。

实践中常采用 $\alpha(n) = \alpha$ 的恒定步长，满足上面收敛条件的step-size方法常用于理论工作。

控制部分我们在Q-learning部分会详细讲解。

Policy Gradient

5. PG

策略梯度算法学习可直接选择动作的参数化策略，其中值函数仍可用于学习策略权重，但不是动作选择所必需的。

其优点：

- 更好的收敛特性（步长足够小，策略梯度保证不断优化策略，不过容易陷入局部最优）
- 在高维或连续动作空间中有效（Q值需要max over action）
- 可以学习随机策略（max Q over action可以近似看做是确定性策略）

其缺点：

- 通常收敛到局部而非全局最优
- 估计策略通常是低效和高方差的

在episodic任务中，policy gradient的性能评估基于每一episode初始状态的值 $\eta(\theta) = V_{\pi,\theta}(s_0)$。

对其求梯度可得：

$$\nabla\eta(\theta) = \sum_s d_\pi(s) \sum_a q_\pi(s,a) \nabla_\theta \pi(a|s,\theta)$$

其中，$d_\pi(s)$为状态稳态分布。

REINFORCE: Monte Carlo Policy Gradient

更新方式为：

$$\theta_{t+1} \doteq \theta_t + \alpha \gamma^t G_t \frac{\nabla_\theta \pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)}$$

REINFORCE的更新增加了这个方向上的权重向量，与return成正比，与动作选择的概率成反比。 前者保证它使权重在产生最高回报的动作的方向上移动得最多。 后者则因为对于被频繁选择的动作，更新往往会朝着他们的方向，即使他们没有产生最高的回报也可能赢。

作为一种随机梯度法，REINFORCE具有良好的理论收敛性（G_t是无偏的，REINFORCE将渐近收敛到局部最小值）。 这保证了在α足够小的情况下，期望性能能够改善，并且对于不断减小的α，在标准随机逼近条件下，策略能收敛到局部最优。 但是，作为蒙特卡罗方法，REINFORCE可能具有很高的方差，因此学习速度较慢。

REINFORCE with Baseline

baseline可以是任何函数，即使是一个随机变量，只要它不随方差降低而变化（从而加快学习速度）

baseline的一个自然的选择是估计状态值$V(S_t;w)$。在一些状态，所有动作都具有较高的价值，我们需要高baseline来区分较高价值的动作和较不重要的动作。在其他状态，所有动作的价值都较低，所以也设置较低的baseline。

REINFORCE with baseline是无偏的，并且会渐近地收敛到局部最小值，但是像所有的蒙特卡罗方法一样，它往往学习速度慢（高方差），并且不便于处理online learning和连续性问题。

5. PG

REINFORCE算法即Policy Gradient系方法的前身，关于Policy Gradient系方法的详细内容及引申，我们会在后面的文档中作为DPG系方法的理论基础进行阐述。

参考文献：

> Reinforcement Learning: An Introduction, Richard S. Sutton, 2018