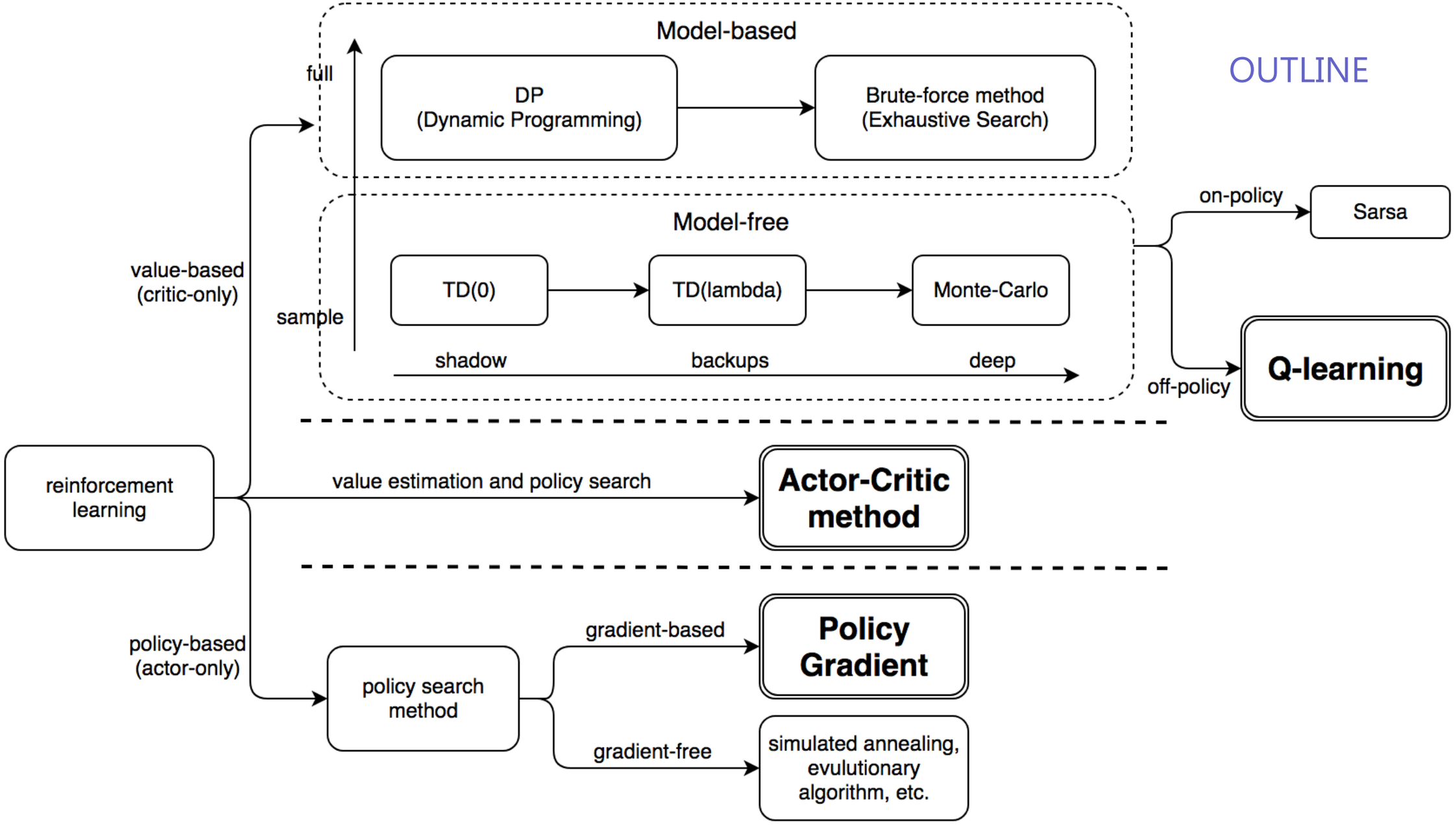
actor critic系算法简介

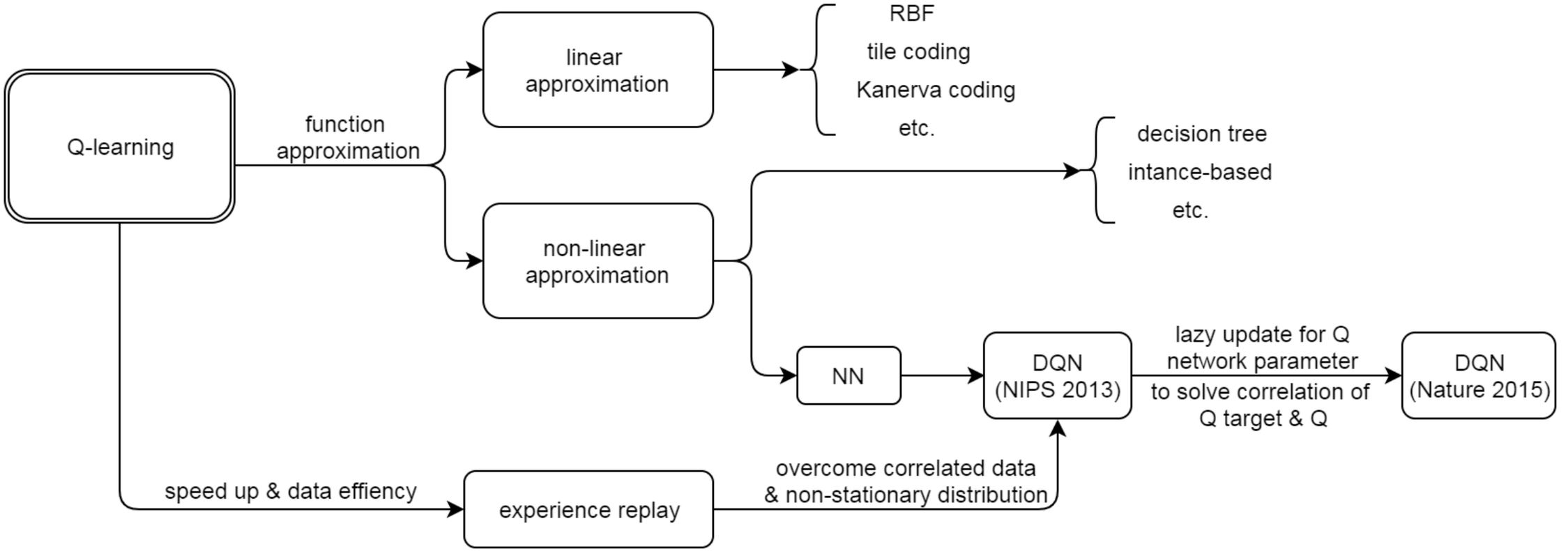
- Author： Sinyer
- sinyeratlantis@gmail.com

OUTLINE

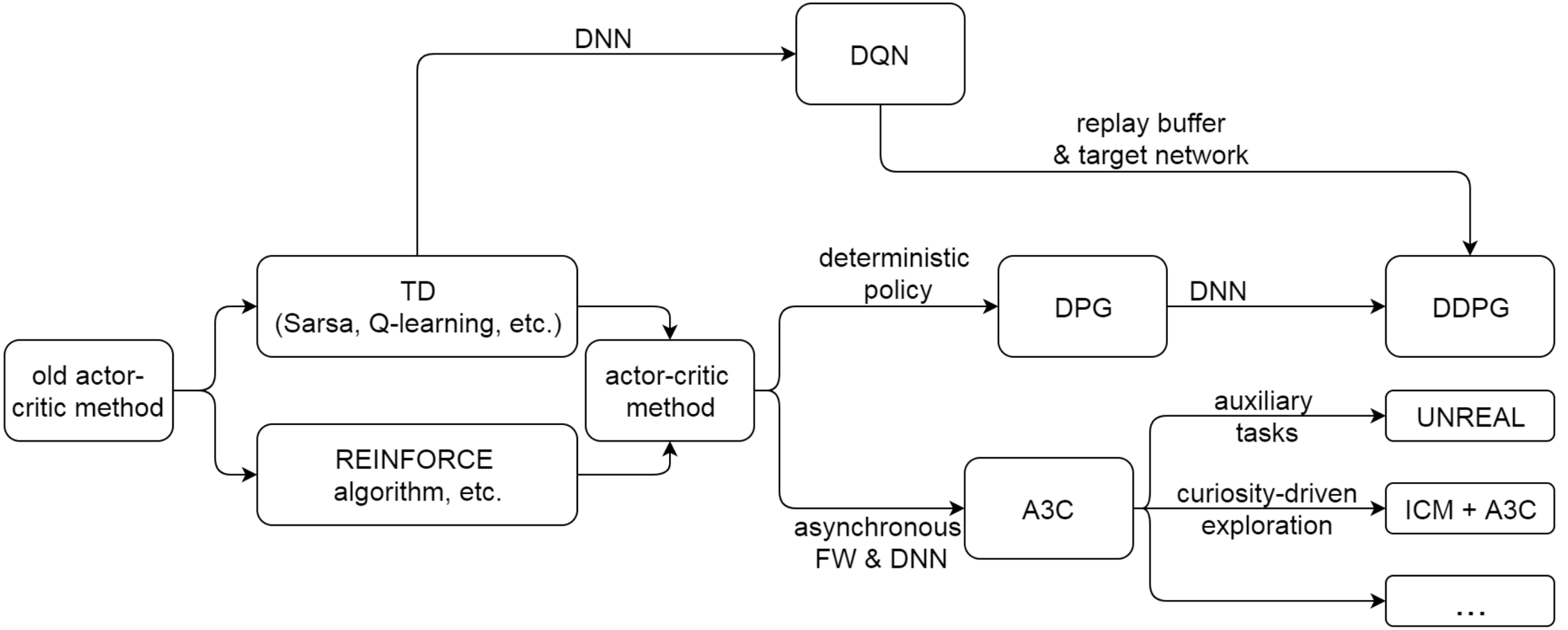
- Background
- DQN
- Double Q learning
- Prioritized Experience Replay
- Dueling architecture
- Policy Gradient
- DPG
- DDPG
- A3C
- AlphaGo


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


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
…

Background

- value-based
- policy-based
- actor-critic

1. BACKGROUND

在RL任务中，我们本质上最终要学习的是策略（Policy）。

value-based用的是间接方法，即通过学习值函数（value function）或者动作值函数（action-value function）来得到policy。

policy-based方法直接对policy进行建模和学习，因此也称为policy optimization。

policy-based方法又可分为两大类：gradient-based方法和gradient-free方法。gradient-based方法也称为policy gradient（PG）方法，分为finite difference，Monte-Carlo和Actor-Critic等。

1. BACKGROUND

Actor-Critic（AC）方法其实是policy-based和value-based方法的结合。因为它本身是一种PG方法，同时又结合了value estimation方法。

在AC框架中，actor负责通过policy gradient学习策略，而critic负责通过policy evaluation估计value function。

actor策略的更新依赖critic估计的value function，而critic估计的value function又是策略的函数。policy和value function互为依赖，相互影响，因此需要在训练过程中迭代优化。这种多元优化的迭代思想在机器学习中有很多体现。

DQN

Playing Atari with Deep Reinforcement Learning, NIPS 2013

Human-level control through deep reinforcement learning, Nature 2015

2. DQN

1989年，Watkins提出了一种model-free的基于off-policy的temporal difference（TD）方法，称为Q learning。

Q learning使用tabular method表示目标函数（V或Q函数），但tabular method表示能力有限，且多用于离散空间。考虑Function approximation方法，相比之下，FA方法可以：

- 降低输入维度，减少计算量
- 提高泛化能力，避免over-fitting
- 使目标函数对于参数可微，可用基于梯度的计算方法

2. DQN

Function approximation分为两大类：线性和非线性逼近。

- 线性逼近一般是用一系列特征的线性组合，它们的权重作为参数。优点是计算方便，算法容易实现。
- 非线性逼近，比如神经网络（Neural network，NN），其优点是表达能力大大加强，缺点是训练起来较为麻烦。

2. DQN

RL的局限之一就是需要人工设计的特征。只要特征选取合适或设计合理，很多时候算法的结果都没有显著差异。一个很自然的想法就是，通过深度学习学习这些特征。

用NN逼近目标函数（比如value function）早已有人研究，上世纪的TD-Gammon，可以说是当时RL在博弈应用的经典，即用了多层神经网络作为函数逼近，但在深度学习出现前的神经网络功能上只用于分类或回归，没有无监督学习提取特征的功能，因此难以直接处理裸数据，比如像素级数据（raw pixels）。

2. DQN

DQN中的输入采用的即是原始图像数据，这是DQN最有意义的一步。其次，它利用了experience replay实现了很好的效果。

experience replay在Long-Ji Lin 1993年的毕业论文中有较详细的介绍，其主要作用是克服经验数据的相关性（correlated data）和非平稳分布（non-stationary distribution）问题。它的做法是从以往的状态转移（经验）中随机采样进行训练。这样至少有两个好处：

- 数据利用率高，因为一个样本被多次使用。
- 连续样本的相关性会使参数更新的方差（variance）比较大，该机制可减少这种相关性。

注意这里用的是随机采样，这也给之后的改进埋下了伏笔。

2. DQN

之后Nature 2015的论文Human-level control through deep reinforcement learning对之作了改进和完善，其中对于算法上的变化最主要是引入了单独的Q function网络。

研究者在实践中发现当使用如NN这样的非线性函数逼近器逼近Q函数时RL学习过程并不稳定。这种不稳定有几种原因：

- 经验数据（即观察序列）具有相关性。
- Q function的微小改变会引起策略（policy）的巨大改变，进而改变训练数据分布，以及Q函数与Q函数目标之间的差值。

2. DQN

相关性问题可以用experience replay解决，而对于Q function的问题，论文采用了迭代式更新（iterative update）解决。

该方法令Q function的参数只在一定步数后才更新，相当于延迟更新来减少Q函数和Q函数目标间的相关性。

整个训练学习过程其实就是Q函数向目标Q函数逼近的过程，所以，若是学习体的变化影响到目标函数，势必会对收敛性造成影响。

Double Q-learning

Deep Reinforcement Learning with Double Q-learning, AAAI 2016

3. Double Q-learning

研究者发现，Q learning中的overestimation问题（在确定状态下Q值估计偏高）可能导致非最优解和学习过程稳定性下降。

最初Thrun & Schwartz开始探讨该问题，证明了在使用函数逼近器时，overestimation可能导致非最优解。之后van Hasselt发现即使用表格表示法的情况下，环境中的噪声也能导致overestimation，并且提出了解决方案Double Q-learning。

3. Double Q-learning

因为DQN基于Q-learning，所以本质上也有这个问题，因此将Double Q-learning结合进DQN可以改善其性能。

其基本思想是将选择和评估动作分离，让它们使用不同的Q函数（网络）。其中一个用于产生贪婪策略（greedy policy），另一个用于产生Q函数估计值。

实现时会有两个Q函数网络：原DQN中的Q函数网络称为在线网络（online network），后者称为目标网络（target network）。

3. Double Q-learning

由于Nature 2015提出的DQN已经引入了单独的Q目标网络，所以Double DQN对DQN架构基本不需什么改动，只需把目标网络的参数和在线网络的参数独立训练即可。

注意和本文方法相比，Nature 2015上的方法相当于是参数的延迟更新，在同一步更新的动作选取和函数估计中还是用的同一个参数。

Prioritized Experience Replay

Prioritized Experience Replay, ICLR 2016

4. Prioritized Experience Replay

在RL与DL结合的实践中起到比较关键作用的experience replay算法灵感可以说部分来自生物学，它类似于大脑中海马体在我们休息的时候将近期的经验回放加深印象的机制。最原始的RL是每次观察到一次状态转移(s, a, R, γ, S')只更新一次参数。

这样一来有几个问题：

- 参数更新具有时间上的相关性，这与随机梯度下降算法的假设不符。
- 那些出现次数少的状态转移（经验）很快就会被遗忘掉。

4. Prioritized Experience Replay

DQN中使用了experience replay来缓解这两个问题。该方法不仅稳定了函数训练过程，也提高了经验数据的利用率，缺点是需要更多内存来存储经验池（experience pool），且experience replay是对以往经验的均匀采样，忽视了经验的重要程度。

对此，我们可以采用应用在model-based的规划问题中的prioritized sweeping。直觉上，我们知道一部分经验比其它经验要对参数的训练产生更大的作用 。prioritized sweeping的基本思想是使参数更新倾向于使值估计变化更大的经验。

4. Prioritized Experience Replay

衡量经验对值估计的贡献需要有定量的测度。

在model-free的场景中，这个测度一般选用TD error。具体地，TD error越大，也就是expected learning progress越高的经验数据，可以让它们replay的次数越频繁。基于这种思想实现的greedy TD-error prioritization算法将经验数据和其TD error按序存在replay memory中，每次取最大TD error的经验进行replay，同时参数更新的量也与之同比，而新的经验会设成最大优先级以保证它至少被训练一次。

4. Prioritized Experience Replay

论文提到这种做法会导致loss of diversity问题和引入bias，文中分别用stochastic prioritization和importance sampling方法来减轻和纠正。

experience replay使得参数的更新不再受限于实际经验的顺序，prioritized experience replay继而使之不再受限于实际经验的出现频率。

Dueling Architecture

Dueling Network Architectures for Deep Reinforcement, ICML 2016

5. Dueling Architecture

论文提出了针对model-free RL的dueling network框架。它是对传统DQN架构层面上的改动，将基于状态的V函数（value function）和状态相关的advantage函数（advantage function）分离。

Advantage函数的思想基于1993年Baird提出的advantage updating。除了传统的V函数外，引入的advantage函数$A(x,u)$的定义是，当采取动作u相比于采取当前最优动作能多带来多少累积折扣回报。简单来说，就是选这个动作比当前最优动作（或其它动作）好多少。

5. Dueling Architecture

基于这个思想，同一个网络会同时估计V函数和advantage函数，它们结合起来可以得到Q函数。从架构上来说，这是一个一分为二，再合二为一的过程。它的出发点是因为对于很多状态，其实并不需要估计每个动作的值。

可以预见到，引入advantage函数后，对于新加入的动作可以很快学习，因为它们可以基于现有的V函数来学习。它直觉上的意义在于将Q函数的估计分为两步。这样，可以先估计哪些状态更能获得更多回报，而不受该状态下不同动作的干扰。

5. Dueling Architecture

V函数专注于长期目标，advantage函数专注于短期目标。这说明V函数和advantage函数分别学习到了两个层次的策略。

这种分层学习的做法有几个好处：

- V函数可以得到更多的学习机会，因为以往一次只更新一个动作对应的Q函数
- V函数的泛化性更好，当动作越多时优势越明显，直观上看，当有新动作加入时，它并不需要从零开始学习
- 因为Q函数在动作和状态维度上的绝对数值往往差很多，这会引起噪声和贪婪策略的突变，而用该方法可以改善这个问题。

Policy Gradient

Policy Gradient Methods for Reinforcement Learning with Function Approximation, Richard S. Sutton, 2000

6. Policy Gradient

PG方法是一大类基于梯度的策略学习方法。其基本做法是先定义policy为参数θ的函数，记作$\pi(\theta)$，然后定义$J(\theta)$为目标函数（Objective function / performance objective），然后，通过利用目标函数相对于参数的梯度更新参数θ，从而达到目标函数的局部最大值，记为：

$$\Delta\theta = \alpha\nabla_\theta J(\theta)$$

在Sutton的奠基性论文Policy Gradient Methods for Reinforcement Learning with Function Approximation中证明了结合可微FA的PG方法可收敛到局部最优点。

论文中提出了两种objective function定义，但结论是一致的 。以average reward formulation为例，它把J定义为平均回报（average reward），对其求导，得到了(Stochastic) Policy Graidient定理。其核心结论给出了目标函数相对于策略参数的梯度公式：

$$\frac{\partial J(\theta)}{\partial \theta} = \sum_s d^\pi(s) \sum_a \frac{\partial \pi(s,a)}{\partial \theta} Q^\pi(s,a)$$

重点是它其中不含有状态稳态分布$d^{\pi}(s)$对于策略参数的导数，这意味着参数的变化对于状态稳态分布没有影响，这样有利于通过采样来逼近和估计梯度。因为这样的话通过执行策略π来得到状态的分布就可以得到无偏估计。

注意如果其中的Q函数用真实的累积回报来估计，那就是REINFORCE算法，但结合了FA，即$Q^{w}(s,a)$后，会引入bias。因此，Sutton还证明了如果FA是compatible的，就不会有bias。

online的TD方法在有linear FA时，只有在on-policy情况下才有收敛性保证，而off-policy方法（如Q-learning）虽然在tabular情况下可以保证收敛，但是在有linear FA情况下就无法保证收敛。而least-squares方法（如LSTD、LSPI）虽能解决off-policy和linear FA下的收敛性问题，但计算复杂度比较高。

Sutton和Maei等人提出的GTD（Gradient-TD）系方法（如Greedy-GQ）解决了off-policy，FA，online算法的收敛性问题。它最小化目标MSPBE（mean-squared projected Bellman error），通过SGD优化求解。

Degris等人在论文Off-Policy Actor-Critic中将AC算法扩展到off-policy场景，提出了OffPAC算法，它是一种online、off-policy的AC算法。 OffPAC中的critic部分采用了上面提到的GTD系方法-GTD(λ)算法。

DPG

Deterministic Policy Gradient Algorithms, ICML 2014

7. DPG

论文主要提出了用于连续动作空间的deterministic policy gradient(DPG)定理，和一种学习确定性目标策略（Deterministic target policy）的off-policy AC算法。

stochastic policy $\pi_\theta(a|s)$定义为动作的分布，而deterministic policy则为状态到动作的映射 $a = \mu_\theta(s)$。

基本思想是，虽然policy可以用概率分布表示，但我们想要的只是一个动作。以往一般认为deterministic policy gradient不存在，而这篇论文将PG框架推广到deterministic policy，证明了deterministic policy gradient不仅存在，而且是model-free形式且是action-value function的梯度，形式简单。

7. DPG

Deterministic Policy Gradient（DPG）定理给出了目标函数对于策略参数的梯度：

$$\nabla_\theta J(\mu_\theta) = \int_S \rho^\mu(s) \nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)} \mathrm{d}s$$

并且证明了它是stochastic policy gradient在policy的variance趋向于0时的极限。这意味着compatible FA，natural gradients等理论都可以应用到DPG场景。

7. DPG

用stochastic policy gradient算法时，当学习进行到后期，policy分布的variance变小。因为policy对其参数的导数在均值附近会变得非常大（比如高斯分布），从而导致变化很大。

另外，在高维动作空间问题中，stochastic policy gradient定理中的梯度内积需要在高维空间中采样，而deterministic policy gradient不需要积分，有解析解。这些都是引入deterministic policy后带来的好处。

但deterministic policy会导致exploration不足。

解决方案是选择动作时用stochastic behavior policy，而学习目标为deterministic target policy。这就意味着需要off-policy。

定义behavior policy $\beta_\theta(a|s)$，target policy为$\pi_\theta(a|s)$。在off-policy和stochastic policy下，根据论文Off-Policy Actor-Critic中的结论，performance objective和其对于θ的梯度可写为：

$$\nabla_\theta J_\beta(\pi_\theta) \approx \mathbb{E}_{s\sim\rho^\beta, a\sim\beta}[\frac{\pi_\theta(a,s)}{\beta_\theta(a,s)}\nabla_\theta \log \pi_\theta(a|s) Q^\pi(s,a)]$$

7. DPG

对应的OffPAC算法中crtic部分用了gradient TD方法。actor和critic基于target policy，通过importance sampling进行估计。

论文根据DPG定理，先是基于on-policy（Sarsa）和off-policy（Q-learning）情况提出了deterministic AC算法，然后基于OffPAC算法提出了OPDAC（off-policy deterministic actor-critic algorithm）算法。OPDAC算法将前面的OffPAC中的梯度公式扩展到deterministic policy，给出off-policy deterministic policy gradient：

$$\nabla_\theta J_\beta(\mu_\theta) \approx \mathbb{E}_{s\sim\rho}[\nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)}]$$

因为FA引入的bias和off-policy引入的不稳定性，都有收敛性问题，论文最后结合compatible FA和gradient TD给出了COPDAC-GQ算法。

DDPG

Continuous Control with Deep Reinforcement Learning, ICLR 2016

8. DDPG

论文提出model-free、off-policy、actor-critic的DRL算法，称为Deep DPG，即DDPG算法，PG和DL的结合。

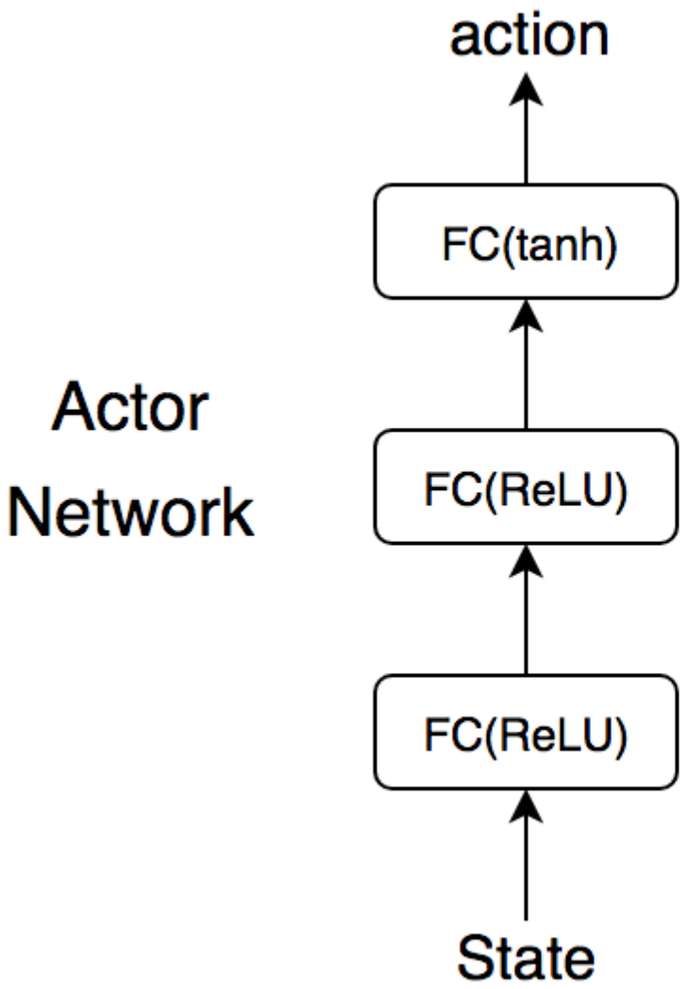
原始的DQN无法应用到连续动作空间，虽然可以通过离散化动作空间解决，但会导致维度灾难（curse of dimensionality）。

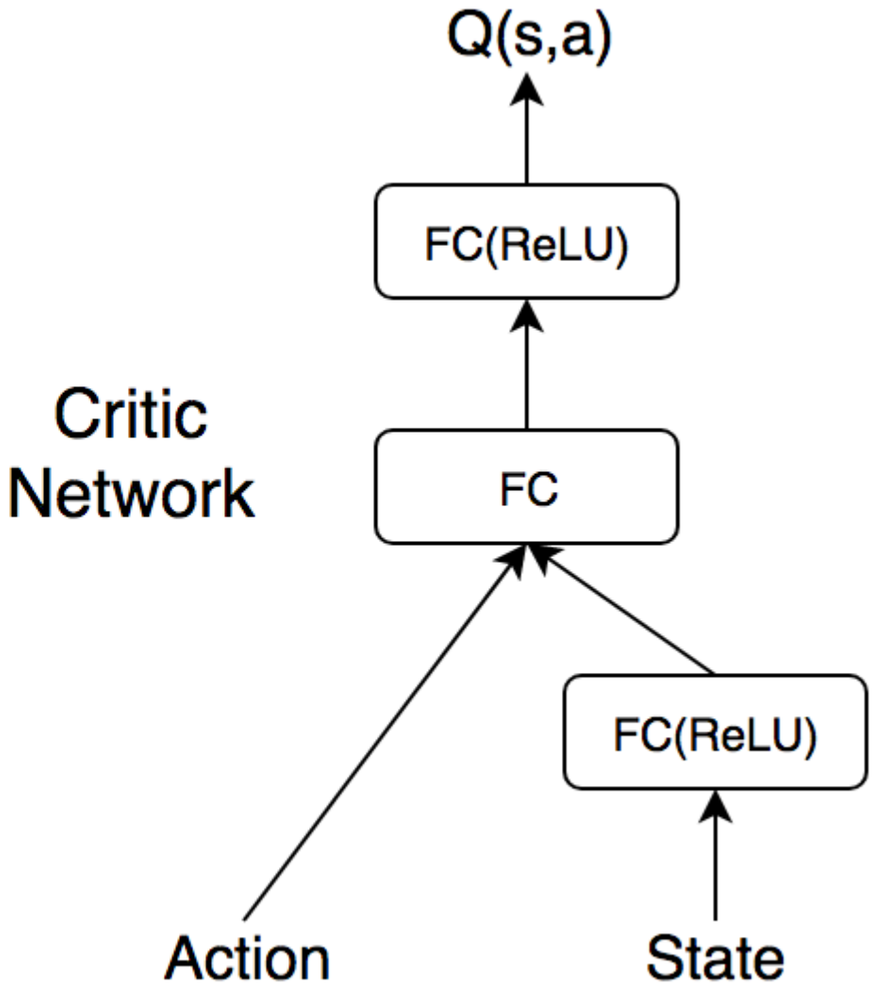
DDPG基于DPG算法，它将AC方法与DQN结合。DQN证明了非线性FA来表示value function可以收敛，通过replay buffer和target Q network解决了sample之间相互关联和Q函数更新的发散问题，使训练更加稳定。

综合以上方法，DDPG算法将DPG中的linear FA扩展到nonlinear FA，即DNN，并使之适用于online情况和高维状态动作空间。

8. DDPG

算法中actor和critic均用DNN表示，分为称为actor network和critic network。它们分别是deterministic policy μ和value-action function Q的FA，参数分别为θ^{μ}和θ^{Q}。在维度比较低时，有下面的结构：


action
FC(tanh)
Actor
Network
FC(ReLU)
FC(ReLU)
State


Q(s,a)
FC(ReLU)
Critic
Network
FC
FC(ReLU)
Action
State

8. DDPG

在迭代更新过程中，先积累experience replay buffer直到达到minibatch指定个数，然后根据sample分别更新两个DNN。先更新critic，通过loss函数L更新参数θ^Q。然后，通过critic得到Q函数相对于动作a的梯度，然后应用actor更新公式更新参数。对于θ的更新，会按比例（通过参数τ）更新到target network。这个target network会在下一步的训练中用于predict策略和Q函数值。

8. DDPG

对于连续动作空间中的exploration问题，论文使用了加一个noise process中产生的noise sample到actor policy来得到behaviour policy的方法。

这篇文章主打解决高维连续动作控制问题，最大的特点就是简单。实验包括cartpole swing-up, dexterous manipulation, legged locomotion和car driving（Torcs）等各种控制场景。

A3C

Asynchronous Methods for Deep Reinforcement Learning, ICML 2016

9. A3C

online的RL算法在和DNN简单结合后会不稳定。主要原因是观察数据往往波动很大且前后sample相互关联。

像Neural fitted Q iteration和TRPO方法通过将经验数据batch，或者像DQN中通过experience replay memory对之随机采样，这些方法有效解决了这两个问题，但是也将算法限定在了off-policy方法中。

9. A3C

论文提出了另一种思路，即通过创建多个agent，在多个环境实例中并行且异步的执行和学习。

另外，将value function的估计作为baseline可以使得PG方法有更低的variance。这种设定下，$R_t - b_t(s_t)$，即$A(s_t, s_t) = Q(a_t, s_t) - V(s_t)$就是advantage function的估计。这也是A3C名字中其中一个A - advantage的由来。

9. A3C

简单地说，每个线程都有agent运行在环境的拷贝中，每一步生成一个参数的梯度，多个线程的这些梯度累加起来，一定步数后一起更新共享参数。

它有两个显著优点：

- 它运行在单个机器的多个CPU线程上，而非使用parameter server的分布式系统，这样就可以避免通信开销和利用lock-free的高效数据同步方法（Hogwild!方法）。
- 多个并行的actor可以有助于exploration。在不同线程上使用不同的探索策略，使得经验数据在时间上的相关性很小。这样不需要DQN中的experience replay也可以起到稳定学习过程的作用，意味着学习过程可以是on-policy的。

9. A3C

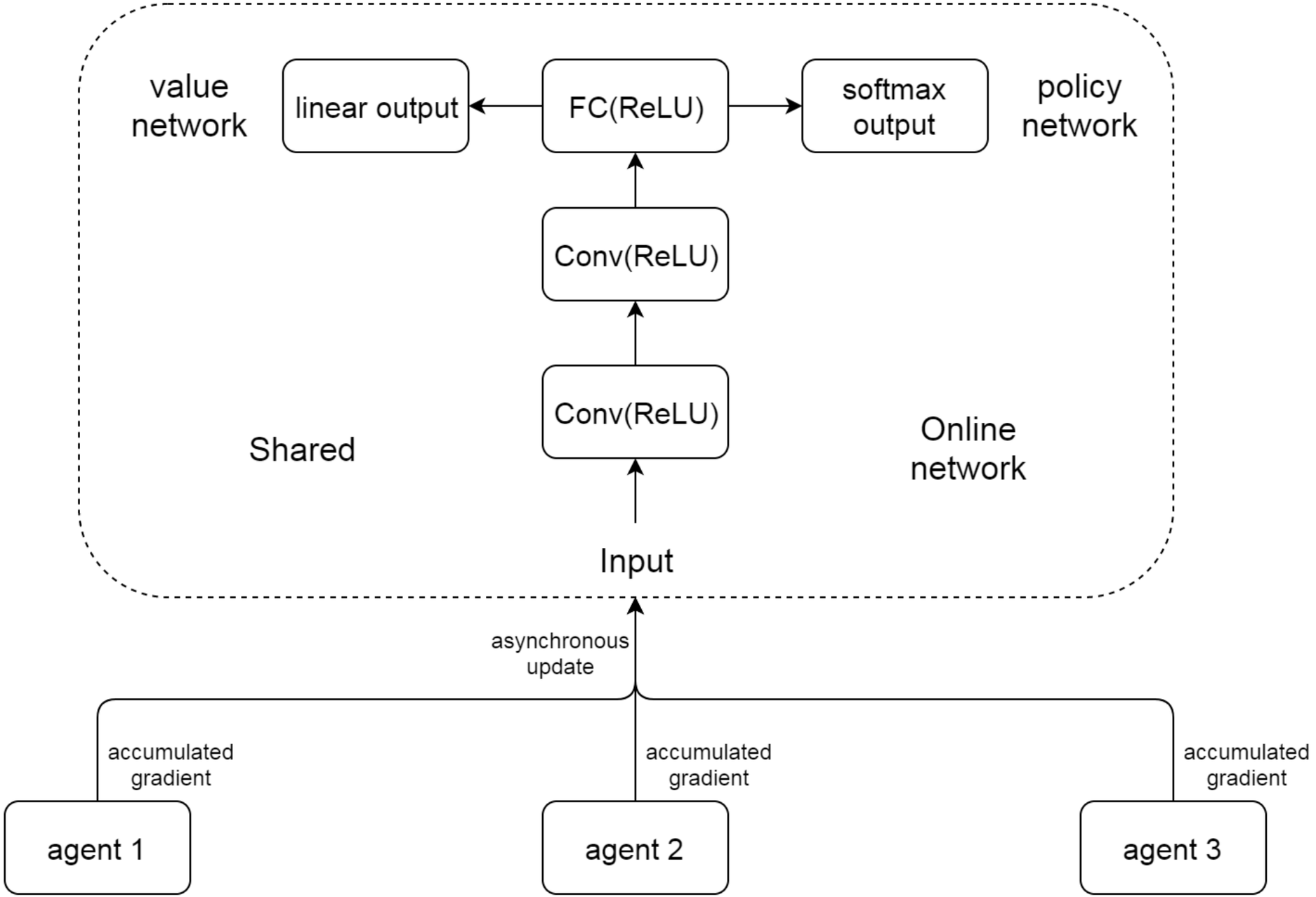
论文主要提出的算法是A3C（asynchronous advantage actor-critic）。

该算法和DDPG类似，通过DNN进行policy和value function的估计，但它没用deterministic policy，在学习过程中使用n-step回报来同时更新policy和value function。

网络结构使用了CNN，其中一个softmax output作为policy $\pi(a_t|s_t;\theta)$，而另一个linear output为value function $V(s_t;\theta_v)$，其余layer都共享。

另外，论文还使用了一个古老的技巧，即将policy的entropy加到目标函数可以避免收敛到次优确定性解。直观上，加上该正则项后目标函数更鼓励找entropy大的，即形状"扁平"的分布，这样就不容易在训练过程中聚集到某一个动作上去。在优化方法上，作者使用了基于RPMProp的一种变体。整个过程大致如图

9. A3C


value network
linear output
FC(ReLU)
softmax output
policy network
Conv(ReLU)
Conv(ReLU)
Shared
Online network
Input
asynchronous update
accumulated gradient
accumulated gradient
accumulated gradient
agent 1
agent 2
agent 3

实验证明在Atari的一些游戏中，n-steps方法比one-step方法快。另外policy-based advantage actor-critic方法比value-based method效果更好。A3C通过并行使得计算效率提升，可以用CPU达到之前用GPU达到的训练效率。

Labyrinth非常有挑战之处在于它每次episode都会生成一个新的迷宫，所以学习到的策略必须较为general。显然，走迷宫和MuJoCo不像前面的任务（Atari, TORCS都是反应式的，即看着当前的画面，就能得到最优策略），它更需要记忆，因此这里A3C也使用了RNN（在最后的hidden layer后加了额外的256个LSTM cell）。

AlphaGo

Mastering the game of Go with deep neural networks and tree search, Nature 2016

10. AlphaGo

AlphaGo通过蒙特卡洛搜索树（Monte-Carlo Tree Search，MCTS）和DRL结合使得这个大规模搜索问题在可计算范围内。它使用人工设计的特征用于policy network和value network的训练。

AlphaGo的大体思想是使用value network来评估棋盘位置，用policy network来选择动作，同时结合了监督学习和DRL技术。它还引入了一种结合MC模拟和value/policy network的搜索算法。

架构上，value network用于评估位置，policy network用于采样动作。流程上，学习分为几个阶段：

- 第一阶段，基于人类专家的数据集通过监督学习训练SL policy network，并训练一个单独的策略用于在之后的rollout中快速选取动作。然后，将SL policy network的参数来初始化RL policy network。
- 第二阶段，通过RL的policy gradient方法学习策略，通过随机梯度上升更新RL policy network参数。
- 第三阶段，学习value network用于预测RL policy network的胜算，使用SGD最小化outcome（在当前状态下围棋结束时的回报，即输赢）与V函数之间的差（用MSE表示）。
- 第四阶段，依据policy network和value network用MCST进行搜索。

在最后一个阶段中，算法会通过MC模拟从根状态来往下单向遍历搜索树，这一阶段可以分为选取（selection），扩展（expansion），评估（evaluation）和回溯（backup）四步。AlphaGo在这个MCTS算法中结合了policy和value network。

第一步选取，基于前面的value network取Q函数值最大的动作。注意这一步使用的是UCT族算法，它在动作选取时除Q函数值还会考虑一个代表未知程度的量，用于解决exploration-exploitation问题。

第二步扩展，是针对搜索树中的叶子结点，使用的是RL policy network中学习到的策略。

第三步评估有两种完全不同的方式：1. 通过value network。2.通过随机rollout直到结束（使用fast rollout policy，也就是之前的SL policy network）所得到的回报。这两者可通过参数进行调和。

第四步回溯，即根据前面的搜索进行Q函数更新。

参考文献：

- 深度增强学习漫谈 - 从DQN到AlphaGo
- 深度增强学习漫谈 - 从AC到A3C