第九讲：深度强化学习

主讲人 陈达贵

清华大学自动化系

在读硕士

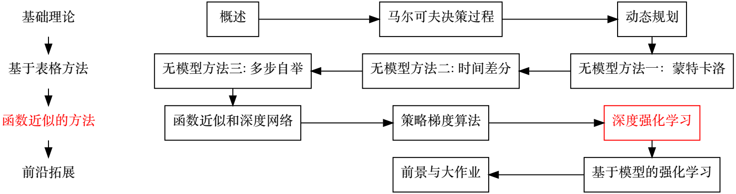
深蓝学院

shenlanxueyuan.com

强化学习理论与实践

目录

章节目录


基础理论
基于表格方法
函数近似的方法
前沿拓展
概述
马尔可夫决策过程
动态规划
无模型方法一：蒙特卡洛
无模型方法二: 时间差分
无模型方法三: 多步自举
函数近似和深度网络
策略梯度算法
深度强化学习
基于模型的强化学习
前景与大作业

本章目录

深度强化学习简介

- 深度学习 + 强化学习 = 深度强化学习
- 使用深度神经网络作为强化学习的函数近似器
- 深度学习本质上是一种表示学习，它负责将原始数据表达为和问题相关的特征
- 例：flappybird
  - 图像输入：DRL
  - 坐标输入：RL

讲解流程

- 基于值函数: 从 DQN 到 Rainbow
- 基于策略
  - 策略梯度框架
  - 信赖域优化

本节课会大量引用文献，为了方便大家得文献阅读，尽量使用和原文类似的英文表达

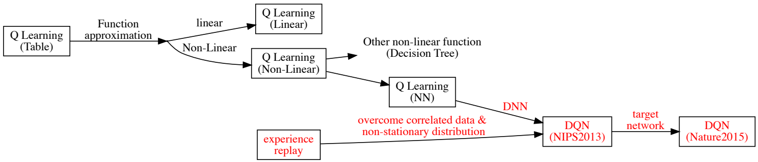
目录

DQN

- Playing Atari with Deep Reinforcement Learning (NIPS2013)
- Human-level control through deep reinforcement learning (Nature2015)
- 重要意义
  - 不用函数近似无法解决大规模问题，用函数近似训练不稳定
  - 首次证明了能够通过 raw pixels 解决游戏问题
  - 对所有游戏通用

DQN

- 关键特点
  - Q Learning+DNN
  - Experience Replay
  - Target Network



DQN

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$

For episode $= 1, M$ **do**

  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

  For $t = 1$,T **do**

    With probability ε select a random action a_t

    otherwise select $a_t = \operatorname{argmax}_a Q(\phi(s_t), a; \theta)$

    Execute action a_t in emulator and observe reward r_t and image x_{t+1}

    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

    Sample random minibatch of transitions $\left(\phi_j, a_j, r_j, \phi_{j+1}\right)$ from D

    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^{-}\right) & \text{otherwise} \end{cases}$

    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ

    Every C steps reset $\hat{Q} = Q$

  End For

End For

Double DQN

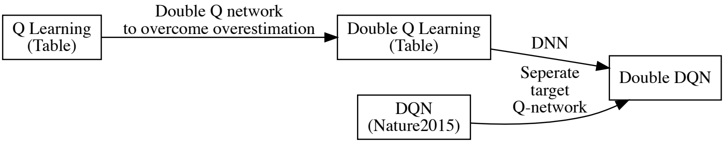
- Deep Reinforcement Learning with Double Q-learning (AAAI2016)
- Q 学习中存在过估计
  - Q 学习中的 TD 目标值 $r + \gamma \max_a Q(s', a)$ 中存在 max 操作
  - 这会引入一个正向的偏差
  - 因此建模两个 Q 网络，一个用于选动作，一个用于评估动作

$$r + \gamma Q^B(s', \arg\max_a Q^A(s', a))$$

例子

对于状态 s 下，如果对于所有的 a，真实的 $q(s, a)$ 均为 0，但是估计值由于不精确，会导致有些大于 0，有些小于 0。对估计值取最大，会引入一个正向的偏差

Double DQN



Algorithm 1 Double Q-learning

```
Initialize Q^A, Q^B, s
repeat
    Choose a, based on Q^A(s,·) and Q^B(s,·), observe r, s'
    Choose (e.g. random) either UPDATE(A) or UPDATE(B)
    if UPDATE(A) then
        Define a* = arg max_a Q^A(s', a)
        Q^A(s,a) ← Q^A(s,a) + α(s,a) (r + γQ^B(s', a*) − Q^A(s,a))
    else if UPDATE(B) then
        Define b* = arg max_a Q^B(s', a)
        Q^B(s,a) ← Q^B(s,a) + α(s,a)(r + γQ^A(s', b*) − Q^B(s,a))
    end if
    s ← s'
until end
```

Dueling DQN

- Dueling Network Architectures for Deep Reinforcement (ICML2016)
- 将 Q 函数分解成 V 函数和优势 (A) 函数

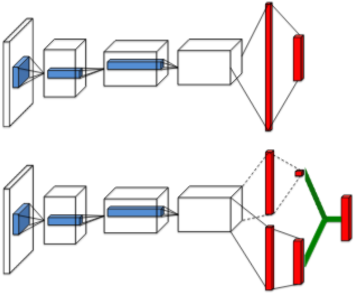
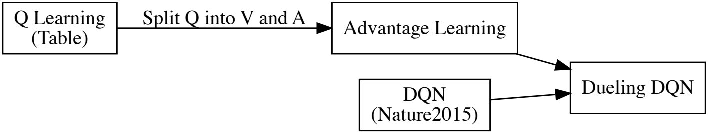


Figure 1. A popular single stream Q-network (**top**) and the dueling Q-network (**bottom**). The dueling network has two streams to separately estimate (scalar) state-value and the advantages for each action; the green output module implements equation (9) to combine them. Both networks output Q-values for each action.

Dueling DQN

- 优势
  - 对于很多状态并不需要估计每个动作的值，增加了 V 函数的学习机会
  - V 函数的泛化性能好，当有新动作加入时，并不需要重新学习
  - 减少了 Q 函数由于状态和动作维度差导致的噪声和突变



Prioritized Experience Replay

- Prioritized Experience Replay (ICLR 2016)
- DQN 算法的一个重要改进是 Experience Replay
- 训练时从 Memory 中均匀采样
- Prioritized Experience Replay 就是维护了一个带优先级的 Experience Replay
  - 不同的 Experience 的权重不同
  - 用 TD 误差去衡量权重
  - 需要使用 sum-tree 以及 binary heap data structure 去实现
  - 新的 transition 的 TD 误差会被设置为最大
  - 类似于 DP 中的优先清理
- Experience Replay 使得更新不受限于实际经验的顺序，Prioritized Experience Replay 使得更新不受限于实际经验的频率

Prioritized Experience Replay

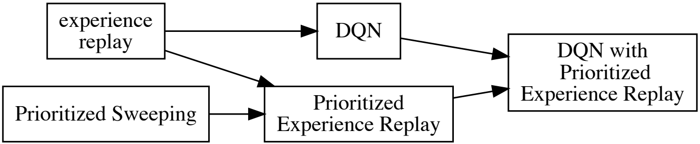
具体使用时会存在一些问题

- TD 误差对噪声敏感
- TD 误差小的 transition 长时间不更新
- 过分关注 TD 误差大的 transition 丧失了样本多样性
- 使用某种分布采样了 Experience, 会引入 Bias

解决方法

- 两种变体: $p_i = |\delta_i| + \varepsilon$ 或者 $p_i = \frac{1}{rank(i)}$
- 加入重要性采样来消除 Bias

Prioritized Experience Replay

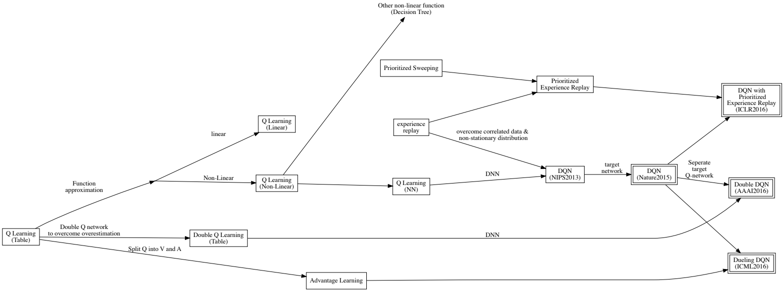


Prioritized Experience Replay

Algorithm 1 Double DQN with proportional prioritization

Input: minibatch k, step-size η, replay period K and size N, exponents α and β, budget T.
Initialize replay memory $\mathcal{H} = \emptyset$, $\Delta = 0$, $p_1 = 1$
Observe S_0 and choose $A_0 \sim \pi_\theta(S_0)$
for $t = 1$ **to** T **do**
  Observe S_t, R_t, γ_t
  Store transition $(S_{t-1}, A_{t-1}, R_t, \gamma_t, S_t)$ in $\mathcal{H}$ with maximal priority $p_t = \max_{i<t} p_i$
  if $t \equiv 0 \mod K$ **then**
    for $j = 1$ **to** k **do**
      Sample transition $j \sim P(j) = p_j^\alpha / \sum_i p_i^\alpha$
      Compute importance-sampling weight $w_j = (N \cdot P(j))^{-\beta} / \max_i w_i$
      Compute TD-error $\delta_j = R_j + \gamma_j Q_{\text{target}}(S_j, \arg\max_a Q(S_j, a)) - Q(S_{j-1}, A_{j-1})$
      Update transition priority $p_j \leftarrow |\delta_j|$
      Accumulate weight-change $\Delta \leftarrow \Delta + w_j \cdot \delta_j \cdot \nabla_\theta Q(S_{j-1}, A_{j-1})$
    end for
    Update weights $\theta \leftarrow \theta + \eta \cdot \Delta$, reset $\Delta = 0$
    From time to time copy weights into target network $\theta_{\text{target}} \leftarrow \theta$
  end if
  Choose action $A_t \sim \pi_\theta(S_t)$
end for

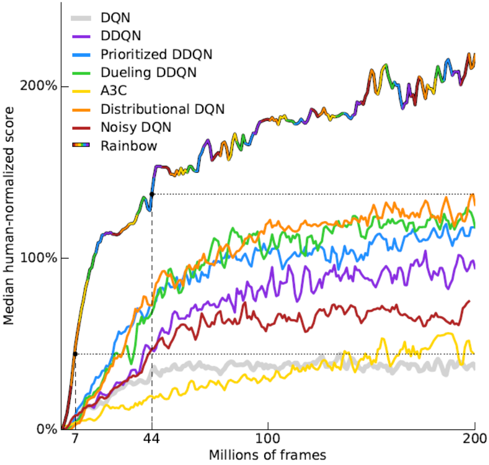
小结


Other non-linear function
(Decision Tree)
Prioritized Sweeping
Prioritized
Experience Replay
DQN with
Prioritized
Experience Replay
(ICLR2016)
Q Learning
(Linear)
linear
experience
replay
overcome correlated data &
non-stationary distribution
Function
approximation
Non-Linear
Q Learning
(Non-Linear)
Q Learning
(NN)
DNN
DQN
(NIPS2013)
target
network
DQN
(Nature2015)
Seperate
target
Q-network
Double DQN
(AAAI2016)
Q Learning
(Table)
Double Q network
to overcome overestimation
Double Q Learning
(Table)
DNN
Split Q into V and A
Advantage Learning
Dueling DQN
(ICML2016)

Rainbow

- Rainbow: Combining Improvements in Deep Reinforcement Learning
- 集合了多种 DQN 的变体
  - DQN
  - Double DQN
  - Dueling DQN
  - Prioritized Experience Replay
  - NoiseNet: (Noisy Networks for Exploration, AAAI2018)
  - Distributional RL: (A Distributional Perspective on Reinforcement Learning, ICML2017)

Rainbow


DQN
DDQN
Prioritized DDQN
Dueling DDQN
A3C
Distributional DQN
Noisy DQN
Rainbow
Median human-normalized score
200%
100%
0%
7
44
100
200
Millions of frames

目录

DPG

- Deterministic Policy Gradient Algorithms (ICML2014)
- 将策略梯度定理用到了高维和连续动作
- 常规的策略梯度方法无法用到高维和连续动作空间
  - 连续动作无法使用概率分布输出的形式
  - 高维动作空间中采样费时
  - 随机策略梯度是通过采样的方式估算策略梯度，需要在状态空间和动作空间内采样

$$\nabla_{\pi_\theta} J(\theta) = \int_{\mathcal{S}} \int_{\mathcal{A}} \rho^{\pi_\theta}(s) \nabla_\theta \log \pi_\theta(a|s) Q^{\pi_\theta}(s, a) ds\, da$$

- 直接采用确定性策略输出

$$a = \pi(s)$$

- 由此诞生几个问题:
  - 确定性策略情况下如何求策略梯度?
  - 如何探索?

DPG

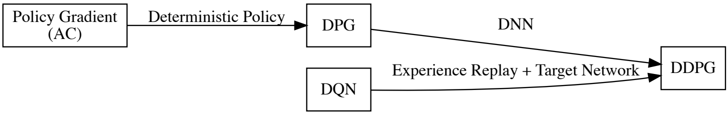
- 过去一直认为无模型情况下确定性策略梯度不存在
- DPG 证明了确定性策略梯度定理的存在, 建立了它跟 Q 函数梯度的关系

$$\nabla_\theta J(\theta) = \int_{\mathcal{S}} \rho^{\pi_\theta}(s) \nabla_\theta \pi_\theta \nabla_a Q^{\pi_\theta}(s, a)|_{a=\pi_\theta(s)} ds$$

- 只需要对状态 $\mathcal{S}$ 进行积分
- 使用 Off-policy 的方式探索并更新 (需要重要性采样)

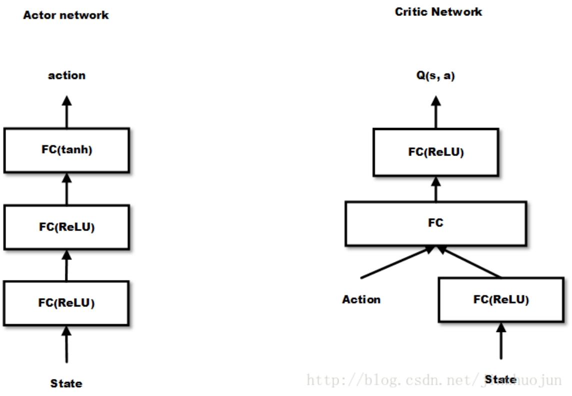
DDPG

- Continuous Control with Deep Reinforcement Learning (ICRL2016)
- 结合了 DQN 和 DPG
- 使用了 DQN 的两种技术：Experience Replay 和 Target Network
- 利用随机过程产生探索性动作



DDPG

建模方式


Actor network
action
FC(tanh)
FC(ReLU)
FC(ReLU)
State
Critic Network
Q(s, a)
FC(ReLU)
FC
Action
FC(ReLU)
State

DDPG

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

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

    Update the target networks:

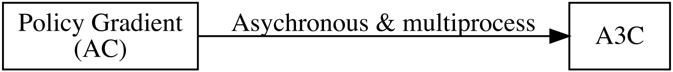
$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

A3C

- Asynchronous Methods for Deep Reinforcement Learning (ICML2016)
- Online 的算法和 DNN 结合后不稳定 (样本关联性)
- 通过创建多个 agent 在多个环境执行异步学习构建 batch
  - 来自不同环境的样本无相关性
  - 不依赖于 GPU 和大型分布式系统
  - 不同线程使用了不同的探索策略，增加了探索量



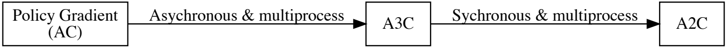
A3C

Algorithm S3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

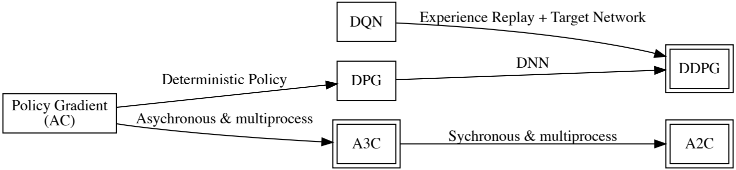
// Assume global shared parameter vectors θ and θ_v and global shared counter $T = 0$
// Assume thread-specific parameter vectors θ' and θ'_v
Initialize thread step counter $t \leftarrow 1$
repeat
    Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
    Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$
    $t_{start} = t$
    Get state s_t
    repeat
        Perform a_t according to policy $\pi(a_t|s_t;\theta')$
        Receive reward r_t and new state s_{t+1}
        $t \leftarrow t+1$
        $T \leftarrow T+1$
    until terminal s_t **or** $t - t_{start} == t_{max}$
    $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t \text{// Bootstrap from last state} \end{cases}$
    for $i \in \{t-1, \ldots, t_{start}\}$ **do**
        $R \leftarrow r_i + \gamma R$
        Accumulate gradients wrt θ': $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i;\theta')(R - V(s_i;\theta'_v))$
        Accumulate gradients wrt θ'_v: $d\theta_v \leftarrow d\theta_v + \partial\left(R - V(s_i;\theta'_v)\right)^2 / \partial\theta'_v$
    end for
    Perform asynchronous update of θ using $d\theta$ and of θ_v using $d\theta_v$.
until $T > T_{max}$

A2C

- Openai Blog(https://blog.openai.com/baselines-acktr-a2c/)
- 改异步为同步，能够更好地利用 GPU
- 在 batch_size 较大时效果好



小结


DQN
Experience Replay + Target Network
DDPG
DNN
DPG
Deterministic Policy
Policy Gradient (AC)
Asychronous & multiprocess
A3C
Sychronous & multiprocess
A2C

目录

简要介绍

策略梯度算法的更新步长很重要

- 步长太小，导致更新效率低下
- 步长太大，导致参数更新的策略比上次更差，通过更差的策略采样得到的样本更差，导致学习再次更新的参数会更差，最终崩溃

如何选择一个合适的步长，使得每次更新得到的新策略所实现的回报值单调不减

- 信赖域 (Trust Region) 方法指在该区域内更新，策略所实现的回报值单调不减

一些基础

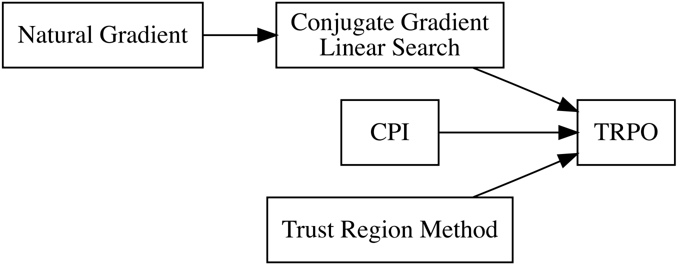
- 自然梯度:Natural Gradient Works Efficiently in Learning, 1998
  - 在黎曼空间里面，最快的下降方向不是梯度方向，而是自然梯度方向 $G^{-1}(\theta)J(\theta)$
  - 只有当坐标系统正交，才退化成欧式空间
  - 神经网络中的参数空间是黎曼空间
  - 其中 G 为 Reimannian metric tensor
  - 统计问题中，G 可以用 Hessian 矩阵去计算

一些基础

- 保守策略迭代, CPI: Approximately Optimal Approximate Reinforcement Learning, 2002
  - 给出策略性能增长的条件
    - 策略更新后的所有优势函数非负
  - 使用混合更新的方式更新策略

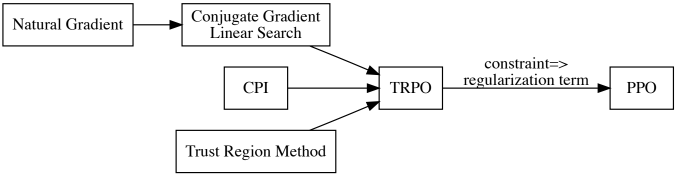
TRPO

- Trust Region Policy Optimization, ICML2015
- 以 CPI 为基础，推导出策略更新后性能的下界，通过优化下界优化原函数
- 实际操作时用 KL 散度作为约束
- 求解带约束的优化问题时，利用自然梯度
- 实际求解是利用了共轭梯度 + 线性搜索的方法，避免求自然梯度



PPO

- Proximal Policy Optimization Algorithms, 2017
- Openai blog(https://blog.openai.com/openai-baselines-ppo/)
- TRPO 太复杂，普通 PG 效果又不好
- PPO 本质上是 TRPO 的简化版
- 移除了 KL 惩罚项和交替更新
- 由于性能好，且容易实现，已经成为默认的 OPENAI 算法



PPO

$$L^{CLIP}(\theta) = \hat{E}_t\left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t)\right]$$

- θ is the policy parameter
- $\hat{E}_t$ denotes the empirical expectation over timesteps
- r_t is the ratio of the probability under the new and old policies, respectively
- $\hat{A}_t$ is the estimated advantage at time t
- ε is a hyperparameter, usually 0.1 or 0.2

其他信赖域算法

- ACKTR: Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation
- ACER: Sample Efficient Actor-Critic with Experience Replay
- GAE: High-Dimensional Continuous Control Using Generalized Advantage Estimation
- ...