Asynchronous Methods for Deep Reinforcement Learning

presented by Jason TOKO

背景与动机

- 传统经验认为，online RL方法与DNN相结合会导致不稳定。
- 主要原因：
  - 观察数据的不稳定性；
  - 样本间的相关性。
- 解决方法：
  - 使用Experience replay（经验回放）方法，可以减少不稳定性以及消除样本相关性。
- 限制：
  - 每次交互都需要大量的内存和计算；
    - 硬件要求较高，传统DRL方法依赖于GPU或者大型分布式架构等
  - 只能应用off-policy的方法进行学习。

异步RL框架

- 本文提出的异步RL框架，解决了经验回放存在的问题：
  - 1、异步地执行多个agent， 通过并行的agent经历的不同状态，去除训练过程中产生样本之间的相关性；
  - 2、只需一个标准的多核CPU即可实现算法，在效果、时间和资源消耗上都优于传统方法；
  - 3、框架的通用性：不仅适用于off-policy、value-based方法，也适用于on-policy、policy-based方法，适用于离散和连续动作空间。
- 主要算法：
  - Asynchronous one-step Q-learning
  - Asynchronous one-step Sarsa
  - Asynchronous n-step Q-learning
  - Asynchronous advantage actor-critic

Asynchronous one-step Q-learning

Algorithm 1 Asynchronous one-step Q-learning - pseudocode for each actor-learner thread.

// Assume global shared θ, θ^-, and counter $T = 0$. 全局共享参数θ，θ-，T

Initialize thread step counter $t \leftarrow 0$
Initialize target network weights $\theta^- \leftarrow \theta$
Initialize network gradients $d\theta \leftarrow 0$
Get initial state s
repeat
    Take action a with ϵ-greedy policy based on $Q(s, a; \theta)$
    Receive new state s' and reward r

$$y = \begin{cases} r & \text{for terminal } s' \\ r + \gamma \max_{a'} Q(s', a'; \theta^-) & \text{for non-terminal } s' \end{cases}$$

    Accumulate gradients wrt θ: $d\theta \leftarrow d\theta + \frac{\partial (y - Q(s,a;\theta))^2}{\partial \theta}$
    $s = s'$
    $T \leftarrow T + 1$ and $t \leftarrow t + 1$
    if $T \mod I_{target} == 0$ **then**
        Update the target network $\theta^- \leftarrow \theta$
    end if
    同步更新target network参数
    if $t \mod I_{AsyncUpdate} == 0$ or s is terminal **then**
        Perform asynchronous update of θ using $d\theta$.
        Clear gradients $d\theta \leftarrow 0$.
    end if
    异步更新main network参数
until $T > T_{max}$

Asynchronous one-step Sarsa

Algorithm 1 Asynchronous one-step Sarsa - pseudocode for each actor-learner thread.

// Assume global shared θ, θ^-, and counter $T = 0$. 全局共享参数θ，θ-，T

Initialize thread step counter $t \leftarrow 0$
Initialize target network weights $\theta^- \leftarrow \theta$
Initialize network gradients $d\theta \leftarrow 0$
Get initial state s
repeat
  Take action a with ϵ-greedy policy based on $Q(s, a; \theta)$
  Receive new state s' and reward r
  $$y = \begin{cases} r & \text{for terminal } s' \\ r + \gamma \boxed{Q(s',a';\theta^-)} & \text{for non-terminal } s' \end{cases}$$
  Accumulate gradients wrt θ: $d\theta \leftarrow d\theta + \frac{\partial (y - Q(s,a;\theta))^2}{\partial \theta}$
  $s = s'$
  $T \leftarrow T + 1$ and $t \leftarrow t + 1$
  if $T \mod I_{target} == 0$ **then**
    Update the target network $\theta^- \leftarrow \theta$
  end if
  同步更新target network参数
  if $t \mod I_{AsyncUpdate} == 0$ or s is terminal **then**
    Perform asynchronous update of θ using $d\theta$.
    Clear gradients $d\theta \leftarrow 0$.
  end if
  异步更新main network参数
until $T > T_{max}$

Asynchronous n-step Q-learning

- one-step方法中，每个r只作用于产生它的(s,a)的更新中，因而学习速度较慢
- 改进方法：n-step Q-learning：

$$TargetQ = r_t + \gamma r_{t+1} + \cdots + \gamma^{n-1} r_{t+n-1} + \gamma^n \max_{a'} Q(s', a'; \theta_i^-)$$

Asynchronous n-step Q-learning

Algorithm S2 Asynchronous n-step Q-learning - pseudocode for each actor-learner thread.

// Assume global shared parameter vector θ.
// Assume global shared target parameter vector θ^-.
// Assume global shared counter $T = 0$.
Initialize thread step counter $t \leftarrow 1$
Initialize target network parameters $\theta^- \leftarrow \theta$
Initialize thread-specific parameters $\theta' = \theta$
Initialize network gradients $d\theta \leftarrow 0$
repeat
    Clear gradients $d\theta \leftarrow 0$
    Synchronize thread-specific parameters $\theta' = \theta$
    $t_{start} = t$
    Get state s_t
    repeat
        Take action a_t according to the ϵ-greedy policy based on $Q(s_t, a; \theta')$
        Receive reward r_t and new state s_{t+1}
        $t \leftarrow t + 1$
        $T \leftarrow T + 1$
    until terminal s_t **or** $t - t_{start} == t_{max}$
    $$R = \begin{cases} 0 & \text{for terminal } s_t \\ \max_a Q(s_t, a; \theta^-) & \text{for non-terminal } s_t \end{cases}$$

全局共享参数θ，θ-，T

线程专用参数θ’

产生n步序列

Asynchronous n-step Q-learning

for $i \in \{t-1, \ldots, t_{start}\}$ **do**

$R \leftarrow r_i + \gamma R$

Accumulate gradients wrt θ': $d\theta \leftarrow d\theta + \frac{\partial\left(R - Q(s_i, a_i; \theta')\right)^2}{\partial \theta'}$

end for

累积n步梯度

Perform asynchronous update of θ using $d\theta$.

异步更新main network参数

if $T \mod I_{target} == 0$ **then**

$\theta^- \leftarrow \theta$

end if

同步更新target network参数

until $T > T_{max}$

Asynchronous advantage actor-critic(A3C)

- Advantage function（优势函数）：

$$A(a_t, s_t) = Q(a_t, s_t) - V(s_t)$$

- 优势函数表现了动作 a_t 在状态 s_t 下的优劣程度~
- Advantage actor-critic：

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s, a)\ A^{\pi_\theta}(s, a)\right]$$

- 在A3C算法中，可用DNN去估计优势函数：

$$A(s_t, a_t; \theta', \theta'_v) = \sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V(s_{t+k}; \theta'_v) - V(s_t; \theta'_v)$$

- 实际中，引入熵项可避免过早收敛到次优解：

$$\nabla_{\theta'} \log \pi(a_t | s_t; \theta')(R_t - V(s_t; \theta_v)) + \beta \nabla_{\theta'} H(\pi(s_t; \theta'))$$

Asynchronous advantage actor-critic(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** $t - t_{start} == t_{max}$

产生n步序列

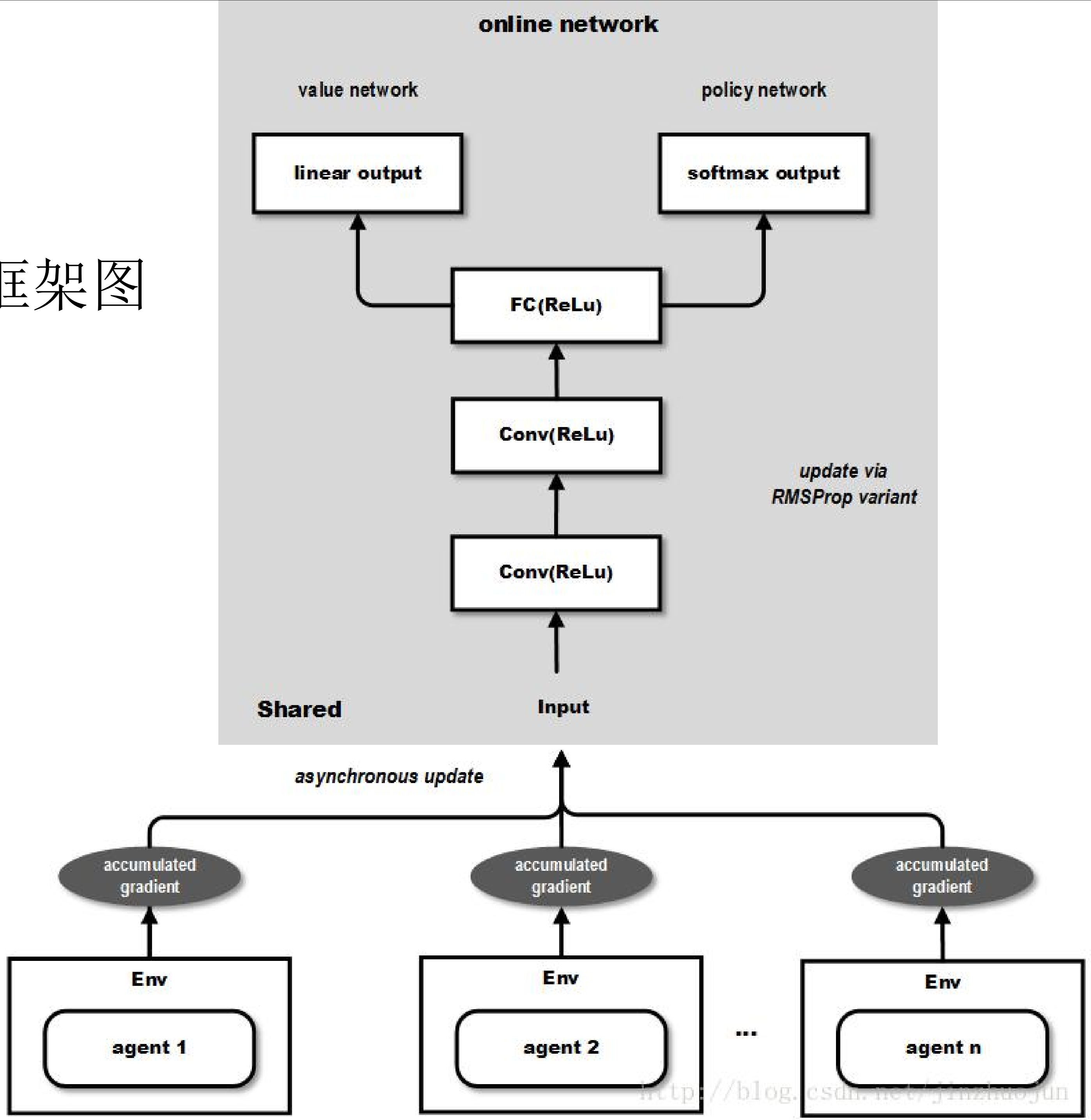
$$R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t \text{// Bootstrap from last state} \end{cases}$$

  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

累积n步
梯度

  Perform asynchronous update of θ using $d\theta$ and of θ_v using $d\theta_v$.
until $T > T_{max}$

A3C 框架图



online network
value network
policy network
linear output
softmax output
FC(ReLu)
Conv(ReLu)
update via
RMSProp variant
Conv(ReLu)
Shared
Input
asynchronous update
accumulated gradient
accumulated gradient
accumulated gradient
Env
agent 1
Env
agent 2
...
Env
agent n

总结

- 异步RL框架的着重点主要在于工程上的设计与优化。
- 异步框架可以解决DNN作为FA的不稳定性问题。
- 异步框架通用性很强，适用于on-policy和off-policy方法，适用于离散和连续动作空间。
- 我
- 随
- 便
- 填
- 充
- 一
- 下

- 引用：
- http://blog.csdn.net/u013236946/article/details/73195035
- http://blog.csdn.net/jinzhuojun/article/details/72851548