Asynchronous Methods for Deep Reinforcement Learning

Chen Gong

10 October 2019

1 Conference Name

International Conference on Machine Learning 2016

2 Thesis statement

研究人员发现简单的在线强化学习和深度神经网络结合起来并不是稳定的，所有有大量的研究人员提出办法来解决这个问题。但是这些方法都有一个共同的特点，在线的强化学习 agent 遇到的观察数据序列是不稳定的，并且在线强化学习的更新是强烈相关的。那么，可以建立一个经验回放池 (Experience Replay)，将 agent 产生的数据放在其中，这样可以从不同的时间步骤上进行批处理或者随机采样。这样的方法可以降低不稳定性，而且消除更新的相关性，但是同时限制了这个算法只能用于 off-policy 的强化学习算法中。

使用 Experience Replay 的方法主要有以下几个缺陷：

1. 每一次交互会消耗更多的内存和计算；

2. 会使 off-policy 学习算法从更老的策略中产生的数据上进行更新。

本文的主要亮点在于，提出了一种很不同的流程来做深度强化学习。不是采用 Experience Replay 的方法，而是采用的异步的平行执行多个智能体的方式来解决上述的问题。这样在任意一个时间段，并行的 agent 都会经历不同的状态，可以解决不稳定的问题。同时，由于是不同的 agent 产生的数据，也很好的解决了更新的相关性的问题。这个 idea 还可以运用于一些 on-policy 的问题中。

文中提出的并行的强化学习结构也提供了一些实际的好处，之前的研究者总是使用一些固定的计算方法，如 GPU 或者大量分布式结构，本文的算法可以在单机上用多核 CPU 来执行。并且取得了比之前基于 GPU 算法更好的效果。

3 Methods

在描述 Deep Q-learning Network 的背景算法中，该部分提出了 Q-learning 中的几个问题，并提出了具体的解决方法。

3.1 One-Step vs N-Step

利用 One-Step 的方法更新价值函数 $Q(s,a)$ 中有一些缺点，因为得到一个奖励 r 仅仅影响了得到该奖励的状态动作对 (s, a) 的动作，而其他的状态动作仅仅间接的通过更新 $Q(s,a)$ 来优化。这样就使得学习的过程变得很缓慢，因为许多更新都要传播一个 r 给相关的动作和状态。

而改进的方法是利用了 N-step 的快速传播奖赏的方法，在这种方法中将对 loss function 做如下的改变：

$$L_i(\theta_i) = \mathbb{E}(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i))^2 \tag{1}$$

转换为：

$$L_i(\theta_i) = \mathbb{E}(r_i + \gamma r_{i+1} + \cdots + \gamma^{n-1} r_{i+n-1} + r^n \max_{a} Q(s_{i+n}, a; \theta_{i+n}) - Q(s, a; \theta_i))^2 \tag{2}$$

这样利用 N-step 的更新方式，Q 函数是朝向 N-step 返回的方向进行更。新这样就使得一个奖赏 r 直接地影响了 n 个正在进行的状态和动作的值。这也使得给相关的奖赏更新更加有效，更加快速。

3.2 优势函数 (Advantage Function)

优势函数可以被定义为：

$$A(a_t, s_t) = Q(a_t, s_t) - V(s_t) \tag{3}$$

在 policy gradient 算法中，策略梯度的计算为 $\nabla_\theta \log \pi(a_t|a_t;\theta)(R_t - b_t(s_t))$。为了缩减方差采用了优势函数的估计方法，使用 $Q(a_t, s_t)$ 函数来代替 r，其中 $\mathbb{E}_{a_t}(Q(a_t, s_t)) = V(s_t)$，那么 $V(s_t)$ 可以看成 policy 部分的 baseline。并且，由于在优势函数的计算中，由于需要用神经网络来近似两个函数 Q 函数和 V 函数，所以利用 $r_t + V(s_{t+1})$ 来近似 Q 函数。于是，优势函数可以近似为：

$$A(a_t, s_t) = r_t + V(s_{t+1}) - V(s_t) \tag{4}$$

然后，利用 N-step 的方法来代替 One-step 的更新策略，于是损失函数可以被定义为

$$L_t(\theta_t) = \mathbb{E}(\sum_{i=0}^{k-1} \gamma^i r_{t+1} + \gamma^k V(s_{t+k}; \theta_t) - V(s_t; \theta_v))^2 \tag{5}$$

3.3 异步的强化学习框架

至于文中提出的多线程的各种算法的变种，是一个很强大的工程 trick，利用这样的方式可以实现和 Experience Replay 类似的效果，利用多组数据记录，较好的解决了训练不稳定的问题。而且消除了数据之间的关联性，便于参数的更新。同时异步的框架可以较好的使用于 on-policy 的强化学习算法中，解决了 Experience Replay 只能用于 off-policy 算法中的缺点。在论文中讨论了四种算法的异步变形，即：One-Step Sarsa，One-Step Q-learning，N-Step Q-leanring 和 Advantage Actor-Critic 算法。文中介绍的选择这四种 RL 算法的目的是证明 RL 算法可以训练深度神经网络策略而不用花费太多的计算资源。在文章的实验表明，异步的学习方法使用 CPU 的计算工具不依赖于 GPU，但是可以实现比 GPU 运算更快的速度和更好的效果。

并且在作者选择的四种 RL 的算法中，Actor-Critic 算法是 on-policy 策略搜索算法，而 Q-learning 算法是 off-policy 基于价值函数的方法，利用两种主要的 idea 来实现四种算法来证明异步算法的有效性。

在异步的多个 Actor 学习者中，利用单机的多 CPU 线程，将这些 Actor 保持在一个机器上，就省去了多个 learner 之间通信的花销。而且多个并行的 Actor 可以更可能的去探索环境的不同的部分。这样我们还可以显示的利用每一个 Actor 不同的探索策略来最大化这个多样性。通过在不同的线程中不同的探索策略，并行的及时的改变参数，更可能在时间上不相关，相比较与单个 agent 采用 online 的更新方式，效率更高。而且，不需要采用经验回放，依赖于并行的 actor 采用不同的探索策略来扮演 Experience Replay 的角色。

除了使学习过程更加的稳定，利用多个并行的 Actor 还有以下的两个优势：

1. 训练时间的大幅度降低，因为时间大致和并行的 Actor 的个数呈线性关系。
2. 因为我们不在依赖于 Experience Replay 来稳定学习，我们可以将这种思想运用到 on-policy 的 RL 方法中。

3.4 优化

在此论文中使用的是标准的非中心化的 RMSProp 更新方法 (这种优化算法的核心思想我不是很懂)：

$$g = \alpha g + (1-\alpha)\triangle\theta^2 \quad and \quad \theta \longleftarrow \theta - \eta\frac{\triangle\theta}{\sqrt{g+\varepsilon}} \tag{6}$$

4 Conclusion and Discussion

论文中对四种强化学习算法使用了异步的更新方法，实验结果证明可以在不同的领域内稳定的训练神经网络。而且可以通用于 off-policy 和 on-policy 的 RL 算法中，并可以同时很好的解决离散和连续的控制问题。而在这四种异步算法中，Asynchronous Advantage Actor-Critic 算法 (A3C) 利用最短的数时间实现了 state-of-the-art 的效果。

论文中作者提出畅想，这个结构的使用于已存在的各种强化学习算法中可以获得立竿见影的效果。我觉得也可以。而且 n-step 更新步骤和优势函数的使用获得了很好的效果。

论文中最后提出，通过包含对状态值和优势函数分离数据流的竞争的结构可以产生更准确地 Q 值的估计。这样可以同时提升 Value-based 和 Policy-based 的方法，并且更加利于神经网络表示共同的特征。(Dueling Double Q-learning Network)

5 My Reflections

论文中采用异步的方法来代替 Experience Replay，这绝不表明 Experience Replay 是没有用的。其实论文中提出了一个想法：在每一个 Actor-learner 上加一个 Experience Replay，无疑这样会进一步的提高数据的利用率，获得更快的训练时间，减小和环境交互的时间。但是这样会进一步的增加内存的负担。

同时，在优势函数的设定上还可以继续做文章，如通用优势函数的使用等等，对算法性能做进一步的提高。而在 Value-Based 的方法中，如何减少估计的偏差是一个无法避免的问题，可以使用本论文中提出的方法结合在线的时间差分算法来做进一步的优化。