# Notes: Asynchronous Methods for Deep Reinforcement Learning[1]

esgl Hu

March 2, 2018

## 1 Review

The simple online RL algorithms with deep neural networks was fundamentally unstable.

The sequence of observed data encountered by an online RL agent is non-stationary, and online RL updates are strongly correlated.

**The experience replay has several drawbacks**: it uses more memory and computation per real interaction; and it requires off-policy learning algorithms that can update from data generated by an older policy.

The loss function of one-step Q-learning is

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

where $s^{'}$ is the state encounted after state $s$. One drawback of using one-step method is that obtaining a reward $r$ only directly affects the value of the state action pair $s, a$ that led to the reward. The values of other state action pairs are affected only indirectly througt the updated value $Q(s, a)$. This can make the learning process slow since many updates are required the propagate a reward to the relevant preceding states and actions. One way of propagating rewards faster is by using $n$-step returns. In $n$-step Q-learning, $Q(s, a)$ is updated toward the $n$-step return defined as $r_t + \gamma r_{t+1} + \cdots + \gamma^{n-1} r_{t+n-1} + \max_a \gamma^n Q(s_{t+n}, a)$.

Policy-based model-free methods directly parameterize the policy $\pi(a|s;\theta)$ and update the parameters $\theta$ by performing, typically approximate, gradient ascent on $\mathbb{E}[R_t]$. For example, standard **REINFORCE** updates the policy parameters $\theta$ in the direction $\nabla_\theta \log\pi(a_t|s_t;\theta)R_t$, which is an unbiased estimate of $\nabla_\theta \mathbb{E}[R_t]$. It is possible to reduce the variance of this estimate while keeping it unbiased by subtracting a learned function of the state $b_t(s_t)$, known as a baseline from the return. The resulting gradient is $\nabla_\theta log\pi(a_t|s_t;\theta)(R_t - b_t(s_t))$.

A learned estimate of the value function is commonly used as the baseline $b_t(s_t) \approx V^\pi(s_t)$ leading to a much lower variance estimate of the policy gradient. When an approximate value function is used as the baseline, the quantity $R_t - b_t$ used to scale the policy gradient can be seen as an estimate of the *advantage* of the action $a_t$ in state $s_t$, or $A(a_t, s_t) = Q(a_t, s_t) - V(s_t)$

# 2 Asynchronous one-step Q-learning

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

*// Assume global shared $\theta, \theta^{-}$, and counter $T = 0$.*
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 $\epsilon$-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 : $d\theta \leftarrow d\theta + \frac{\partial (y - Q(s,a;\theta))^2}{\partial \theta}$
    $s = s^{'}$
    $T \leftarrow T + 1$ and $t \leftarrow t + 1$
    **if** $T$ mod $I_{target} == 0$ **then**
        Update the target network $\theta^{-} \leftarrow \theta$
    **end if**
    **if** $t$ mod $I_{asyncUpdate} == 0$ or $s$ is terminal **then**
        Perform asynchronous update of $\theta$ using $d\theta$.
        Clear gradients $d\theta \leftarrow 0$.
    **end if**
**until** $T > T_{max}$

# 3 Asynchronous one-step Sarsa

The asynchronous one-step Sarsa algorithm is the same as asynchronous one-step Q-learning as given in algorithm 1 except that is uses a different target value for $Q(s, a)$. The target value used by one-step Sarsa is $r + \gamma Q(s^{'}, a^{'}; \theta^{-})$ where $a^{'}$ is the action taken in state $s^{'}$.

# 4 Asynchronous n-step Q-learning

# 5 Asynchronous advantage actor-critic

# References


[1] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. CoRR, abs/1602.01783, 2016.

**Algorithm 2** Asynchronous one-step Q-learning - pseudocode for each actor-learner thread.

*// Assume global shared parameter vector* $\theta$
*// Assume global shared target parameter vector* $\theta^-$
*// Assume global shared counter* $T = 0$
Initialize thread step count $t \leftarrow 1$
Initialize target network parameters $\theta \leftarrow \theta$
Initialize thread-specific parameters $\theta^- \leftarrow \theta$
Initialize network gradients $d\theta \leftarrow 0$
**repeat**
    Clear gradient $d\theta \leftarrow 0$
    Synchronous thread-specific parameters $\theta^{'} = \theta$
    $t_{target} = t$
    Get state $s_t$
    **repeat**
        Take action $a_t$ accordin to the $\epsilon$-greedy policy based on $Q(s_t, a_t; \theta^{'})$
        Receive reward $r_t$ and new state $s_{t+1}$
        $t \leftarrow t + 1$
        $T \leftarrow T + 1$
    **until** terminal $s_t$ or $t - t_{target} == t_{max}$
    $R = \begin{cases} 0 & \text{for terminal } s_t \\ max_a Q(s^t, a; \theta^-) & \text{for non-terminal } s_t \end{cases}$
    **for** $i \in t-1, ..., t_{start}$ **do**
        $R \leftarrow r_i + \gamma R$
        Accumulate gradients wrt $\theta^{'}$ : $d\theta + \frac{\partial(R - Q(s_i, a_i; \theta^{'}))^2}{\partial \theta^{'}}$
    **end for**
    Performance asynchronous update of $\theta$ using $d\theta$.
    **if then**$TmodI_{target} == 0$
        $\theta^- \leftarrow \theta$
    **end if**
**until** $T > T_{max}$

**Algorithm 3** Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

*// Assume global shared parameter vector $\theta$ and $\theta_v$ and global shared counter $T = 0$*
*// Assume thread-specific parameters $\theta'$ and $\theta'_v$*
Initialize thread step counter $t \leftarrow 1$
**repeat**
  Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
  Synchronous thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$
  $t_{target} = t$
  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 $t - t_{target} == t_{max}$
  $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t \text{//Bootstrap from last state} \end{cases}$
  **for** $i \in t-1, ..., t_{start}$ **do**
    $R \leftarrow r_i + \gamma R$
    Accumulate gradients wrt $\theta'$ : $d\theta + \Delta_{\theta'} log\pi(a_i|s_i;\theta')(R - V(s_i;\theta'_v))$
    Accumulate gradients wrt $\theta'_v$ : $d\theta_v + \frac{\partial(R - V(s_i;\theta'_v))^2}{\partial\theta'_v}$
  **end for**
  Perform asynchronous update of $\theta$ using $d\theta$ and of $\theta_v$ using $d\theta_v$
**until** $T > T_{max}$