Asynchronous Methods for Deep Reinforcement Learning

Khodachenko Ian
(material by Mnih, Volodymyr, et al.)
Graz, 10. December 2019
Asynchronous Methods for Deep Reinforcement Learning

Overview

1. Preliminaries
   – 1-step / n-step Q-learning and SARSA
   – Actor-Critic Models
   – Stochastic Gradient Descent

2. Asynchronous Algorithms
   – 1-step Q-learning (and SARSA)
   – n-step Q-learning
   – Advantage Actor-Critic

3. Experiments
   – Asynchronous RL Framework
   – Learning Speed
   – Robustness & Scalability
   – Thread Efficiency
   – Thread Speedup
1-step / n-step Q-learning and SARSA

- 1-step Q-learning

\[
Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]
\]

- n-step Q-learning

\[
Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_t + \gamma r_{t+1} + \ldots + \gamma^{n-1} r_{t+n-1} + \gamma^n \max_a Q(s_{t+n}, a) - Q(s_t, a_t)]
\]

- SARSA

\[
Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]
\]
Actor-Critic Models (1)

- Parametrized policy $\pi(a|s; \theta)$
- Want (approximated) gradient ascent on $\mathbb{E}[R_t]$
- Like in REINFORCE update $\theta$ in the direction of
  $$\nabla_\theta \mathbb{E}[R_t] \rightarrow \nabla_\theta \log \pi(a_t|s_t; \theta)(R_t - b_t(s_t))$$
- $R_t - b_t$ can be seen as the advantage of action $a_t$ in state $s_t$, or formalized
  $$A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t) \approx R_t \approx b_t$$
- View policy $\pi$ as the actor and baseline $b_t$ the critic
Preliminaries

Actor-Critic Models (2)

- Split **agent** into **actor** and **critic**
- Actor $\rightarrow$ which action to take?
- Critic $\rightarrow$ how good was that action?

![Diagram of Actor-Critic Models](chart.png)
Stochastic Gradient Descent

- Gradient on whole data-set is expensive ($\nabla_{\theta} L(\theta)$)
- Samples approximate direction ($\nabla_{\theta} L(\theta; x^{(i)}; y^{(i)})$)
- Often used in mini-batches ($\nabla_{\theta} L(\theta; x^{(i:i+n)}; y^{(i:i+n)})$)
- Can be parallelized
Asynchronous Algorithms

Asynchronous 1-step Q-learning (and SARSA)

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

Asynchronous n-step Q-learning

Algorithm S2 Asynchronous n-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 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 $\epsilon$-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 $t - t_{start} = t_{max}$
  $R = \begin{cases} 0 & \text{for terminal } s_t \\ \max_a Q(s_t, a; \theta^-) & \text{for non-terminal } s_t \end{cases}$
  for $i \in \{t - 1, \ldots, t_{start}\}$ do
    $R \leftarrow r_i + \gamma R$
    Accumulate gradients wrt $\theta'$: $d\theta \leftarrow d\theta + \frac{\partial (R - Q(s_i, a_i; \theta'))^2}{\partial \theta'}$
  end for
  Perform asynchronous update of $\theta$ using $d\theta$.
  if $T \mod I_{target} = 0$ then
    $\theta^- \leftarrow \theta$
  end if
until $T > T_{max}$
Asynchronous Advantage Actor-Critic (A3C)

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

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

- Use multiple asynchronous actor-learners (=threads)
- Save & accumulate gradient change
- Multiple actor-learners have stabilizing effect when exploring different paths. Thus can remove replay memory (where present).
- Using RMSProp with thread-global parameters
- Training on 16 CPU cores
- Experiments on
  - Atari 2600 Games
  - TORCS Car Racing Simulator
  - MuJoCo Physics Simulator
  - new 3D environment Labyrinth
Experiments: Learning Speed

On Atari 2600, different seeds, fixed hyperparameters.

<table>
<thead>
<tr>
<th>Method</th>
<th>Training Time</th>
<th>Mean</th>
<th>Median</th>
</tr>
</thead>
<tbody>
<tr>
<td>DQN</td>
<td>8 days on GPU</td>
<td>121.9%</td>
<td>47.5%</td>
</tr>
<tr>
<td>Gorila</td>
<td>4 days, 100 machines</td>
<td>215.2%</td>
<td>71.3%</td>
</tr>
<tr>
<td>D-DQN</td>
<td>8 days on GPU</td>
<td>332.9%</td>
<td>110.9%</td>
</tr>
<tr>
<td>Dueling D-DQN</td>
<td>8 days on GPU</td>
<td>343.8%</td>
<td>117.1%</td>
</tr>
<tr>
<td>Prioritized DQN</td>
<td>8 days on GPU</td>
<td>463.6%</td>
<td>127.6%</td>
</tr>
<tr>
<td>A3C, FF</td>
<td>1 day on CPU</td>
<td>344.1%</td>
<td>68.2%</td>
</tr>
<tr>
<td>A3C, FF</td>
<td>4 days on CPU</td>
<td>496.8%</td>
<td>116.6%</td>
</tr>
<tr>
<td>A3C, LSTM</td>
<td>4 days on CPU</td>
<td>623.0%</td>
<td>112.6%</td>
</tr>
</tbody>
</table>
Experiments: Robustness & Scalability

On Atari 2600, different learning rates, random initializations.

<table>
<thead>
<tr>
<th>Method</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-step Q</td>
<td>1.0</td>
<td>3.0</td>
<td>6.3</td>
<td>13.3</td>
<td>24.1</td>
</tr>
<tr>
<td>1-step SARSA</td>
<td>1.0</td>
<td>2.8</td>
<td>5.9</td>
<td>13.1</td>
<td>22.1</td>
</tr>
<tr>
<td>n-step Q</td>
<td>1.0</td>
<td>2.7</td>
<td>5.9</td>
<td>10.7</td>
<td>17.2</td>
</tr>
<tr>
<td>A3C</td>
<td>1.0</td>
<td>2.1</td>
<td>3.7</td>
<td>6.9</td>
<td>12.5</td>
</tr>
</tbody>
</table>
Thread Efficiency (1)
On Atari 2600, best learning rate.
Experiments

Thread Efficiency (2)

On Atari 2600, best learning rate. Note how A3C has best overall scores.
Experiments

Thread Speedup (1)

On Atari 2600, best learning rate.

Khodachenko Ian (material by Mnih, Volodymyr, et al.)
Graz, 10. December 2019
Experiments

Thread Speedup (2)

On Atari 2600, best learning rate.
Note how A3C has best overall scores.
Thank you for your attention!

Questions?