Distributed Deep Reinforcement Learning
Lucia Liu

Distributed Prioritized Experience Replay
Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado van Hasselt, David Silver

Distributed Distributional Deterministic Policy Gradients
Gabriel Barth-Maron, Matthew W. Hoffman, David Budden, Will Dabney, Dan Horgan, Dhruva TB, Alistair Muldal, Nicolas Heess, Timothy Lillicrap
Approaches to Distributed Architecture

• Distribution through parallelizing computation of gradients

• Distribution of generation and selection of experience data

• Deep Q-Network (DQN)

• Deep Deterministic Policy Gradient (DDPG)

Dean et al. in NIPS 2012: Large scale distributed deep networks.
Prioritized Experience Replay

1. Greedy TD-error prioritization
2. Stochastic prioritization
   - Proportional
   - Rank-based

Schaul et al. in ICLR 2016: Prioritized Experience Replay
Distributed Prioritized Experience Replay (Ape-X)
Algorithm 1 Actor

1: procedure Actor(B, T) ▷ Run agent in environment instance, storing experiences.
2: \( \theta_0 \leftarrow \text{LEARNER PARAMETERS}(\) ▷ Remote call to obtain latest network parameters.
3: \( s_0 \leftarrow \text{ENVIRONMENT INITIALIZE}(\) ▷ Get initial state from environment.
4: for \( t = 1 \) to \( T \) do
5: \( a_{t-1} \leftarrow \pi_{\theta_{t-1}}(s_{t-1}) \) ▷ Select an action using the current policy.
6: \( (r_t, \gamma_t, s_t) \leftarrow \text{ENVIRONMENT STEP}(a_{t-1}) \) ▷ Apply the action in the environment.
7: \( \text{LOCAL BUFFER}.\text{ADD}((s_{t-1}, a_{t-1}, r_t, \gamma_t)) \) ▷ Add data to local buffer.
8: if \( \text{LOCAL BUFFER}.\text{SIZE}() \geq B \) then ▷ In a background thread, periodically send data to replay.
9: \( \tau \leftarrow \text{LOCAL BUFFER}.\text{GET}(B) \) ▷ Get buffered data (e.g. batch of multi-step transitions).
10: \( p \leftarrow \text{COMPUTE PRIORITIES}(\tau) \) ▷ Calculate priorities for experience (e.g. absolute TD error).
11: \( \text{REPLY}.\text{ADD}(\tau, p) \) ▷ Remote call to add experience to replay memory.
12: end if
13: Periodically \( \theta_t \leftarrow \text{LEARNER PARAMETERS}(\) ▷ Obtain latest network parameters.
14: end for
15: end procedure
Algorithm 2 Learner

1: procedure LEARNER(T)                      ▷ Update network using batches sampled from memory.
2:     \( \theta_0 \leftarrow \text{INITIALIZENETWORK()} \)                          ▷ Update the parameters \( T \) times.
3:     for \( t = 1 \) to \( T \) do
4:         \( id, \tau \leftarrow \text{REPLAY.SAMPLE()} \) ▷ Sample a prioritized batch of transitions (in a background thread).
5:         \( l_t \leftarrow \text{COMPUTELoss}(\tau; \theta_t) \) ▷ Apply learning rule; e.g. double Q-learning or DDPG
6:         \( \theta_{t+1} \leftarrow \text{UPDATEPARAMETERS}(l_t; \theta_t) \)
7:         \( p \leftarrow \text{COMPUTEPRORITIES()} \) ▷ Calculate priorities for experience, (e.g. absolute TD error).
8:         \( \text{REPLAY.SETPRIORITY}(id, p) \) ▷ Remote call to update priorities.
9:     \text{PERIODICALLY}(\text{REPLAY.REMOVETO FIT()}) ▷ Remove old experience from replay memory.
10:    end for
11: end procedure
Advantages

Shared, centralized replay memory

Prioritization

Off-Policy

High priority data discovered by any actor benefits whole system
Ape-X DQN

• Loss:  
  \[ l_t(\theta) = \frac{1}{2}(G_t - q(S_t, A_t, \theta))^2 \]

\[ G_t = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n \underbrace{q(S_{t+n}, \text{argmax}_a q(S_{t+n}, a, \theta), \theta^-)}_{\text{double-Q bootstrap value}} \]

\[ \quad \underbrace{\text{multi-step return}}_{\text{double-Q bootstrap value}} \]

• Behavior policy: Different policy for each actor, \( \varepsilon \)-greedy
Ape-X DPG

• Actor policy network + Q-network

• Q-network:
  • Action-value estimate $q(s,a,\psi)$
  • Loss:

$$l_t(\psi) = \frac{1}{2}(G_t - q(S_t, A_t, \psi))^2$$

• Policy network:
  • Action $A_t = \pi(S_t, \phi)$
  • Gradient $\nabla_\phi q(S_t, \pi(S_t, \phi), \psi)$

$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n q(S_{t+n}, \pi(S_{t+n}, \phi^{-}), \psi^{-}) \quad \text{multi-step return}$$
Results: Atari games

Blue: Ape-X DQN
Orange: A3C
Purple: Rainbow
Green: DQN
## Results: Atari games

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Training Time</th>
<th>Environment Frames</th>
<th>Resources (per game)</th>
<th>Median (no-op starts)</th>
<th>Median (human starts)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ape-X DQN</td>
<td>5 days</td>
<td>22800M</td>
<td>376 cores, 1 GPU</td>
<td>434%</td>
<td>358%</td>
</tr>
<tr>
<td>Rainbow</td>
<td>10 days</td>
<td>200M</td>
<td>1 GPU</td>
<td>223%</td>
<td>153%</td>
</tr>
<tr>
<td>Distributional (C51)</td>
<td>10 days</td>
<td>200M</td>
<td>1 GPU</td>
<td>178%</td>
<td>125%</td>
</tr>
<tr>
<td>A3C</td>
<td>4 days</td>
<td>—</td>
<td>16 cores</td>
<td>—</td>
<td>117%</td>
</tr>
<tr>
<td>Prioritized Dueling DQN</td>
<td>9.5 days</td>
<td>200M</td>
<td>1 GPU</td>
<td>172%</td>
<td>115%</td>
</tr>
<tr>
<td>Gorila DQN c</td>
<td>~4 days</td>
<td>—</td>
<td>unknown</td>
<td>96%</td>
<td>78%</td>
</tr>
<tr>
<td>UNREAL d</td>
<td>—</td>
<td>250M</td>
<td>16 cores</td>
<td>331% d</td>
<td>250% d</td>
</tr>
</tbody>
</table>
Results: Atari games
Results: Atari games

![Graphs showing the performance of different Atari games with varying replay capacities.](image)
Results: Continuous Control
Distributed Distributional Deterministic Policy Gradients

Gabriel Barth-Maron, Matthew W. Hoffman, David Budden, Will Dabney, Dan Horgan, Dhruva TB, Alistair Muldal, Nicolas Heess, Timothy Lillicrap
Distributed Distributional DDPG (D4PG)

- Based on DDPG algorithm with 4 extensions

- Distributional critic update
- Distributed parallel actors
- N-step returns
- Prioritization of experience replay
Distributional critic

- Distributional update with random variable $Z_{\pi} \rightarrow Q_{\pi}(x, a) = \mathbb{E}[Z_{\pi}(x, a)]$
- Distributional Bellman operator
  \[(\mathcal{T}_{\pi} Z)(x, a) = r(x, a) + \gamma \mathbb{E}[Z(x', \pi(x')) | x, a]\]
- Loss
  \[L(w) = \mathbb{E}_{\rho} \left[ d(\mathcal{T}_{\pi, w}(x, a), Z_w(x, a)) \right] \]
- Gradient for actor update
  \[\nabla_{\theta} J(\theta) \approx \mathbb{E}_{\rho} \left[ \nabla_{\theta} \pi_{\theta}(x) \nabla_{a} Q_{w}(x, a) | a = \pi_{\theta}(x) \right], \]
  \[= \mathbb{E}_{\rho} \left[ \nabla_{\theta} \pi_{\theta}(x) \mathbb{E}[\nabla_{a} Z_{w}(x, a)] | a = \pi_{\theta}(x) \right].\]

N-step returns

• Replacing Bellman operator with

\[
\left( T^{N}_\pi Q \right)(x_0, a_0) = r(x_0, a_0) + \mathbb{E} \left[ \sum_{n=1}^{N-1} \gamma^n r(x_n, a_n) + \gamma^N Q(x_N, \pi(x_N)) \mid x_0, a_0 \right]
\]
Architecture variants
Results: Standard Control
Results: Parkour
Video
Thank you!
References

• Horgan et al. 2018. Distributed prioritized experience replay.
• Barth-Maron et al. 2018. Distributed distributional deterministic policy gradients.
• Dean et al. 2012. Large scale distributed deep networks.
• Schaul et al. 2016. Prioritized experience replay.
• Mnih et al. 2015. Human-level control through deep reinforcement learning.