# Robot Learning

Actor-Critic Method

Prof. Songhwai Oh
ECE, SNU

Mnih, Volodymyr, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. "**Asynchronous methods for deep reinforcement learning**." In International Conference on Machine Learning, pp. 1928-1937. 2016.

# A3C

# Asynchronous Reinforcement Learning

Main idea:

- Execute multiple agents in parallel (with different exploration policies)
  - Diverse experiences
- Update parameters using accumulated gradients asynchronously

Benefits:

- Less correlation in time
- No need for experience replay
  - Hence, computation and memory requirements are small => faster
- On-policy reinforcement learning is possible
  - E.g., Sarsa, n-step, actor-critic

With the actor-critic method, continuous control is possible

# 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 $\theta$: $d\theta \leftarrow d\theta + \frac{\partial (y - Q(s,a;\theta))^2}{\partial \theta}$
  $s = s'$
  $T \leftarrow T + 1$ and $t \leftarrow t + 1$
  **if** $T \mod I_{target} == 0$ **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** $T > T_{max}$

# 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 $\theta'$: $d\theta \leftarrow d\theta + \frac{\partial \left(R - Q(s_i, a_i; \theta')\right)^2}{\partial \theta'}$
  **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 $\theta'_v = \theta_v$
  $t_{start} = t$
  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** $t - t_{start} == t_{max}$
  $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t \text{// Bootstrap from last state} \end{cases}$
  **for** $i \in \{t-1, \ldots, t_{start}\}$ **do**
    $R \leftarrow r_i + \gamma R$
    Accumulate gradients wrt $\theta'$: $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i;\theta')(R - V(s_i;\theta'_v))$
    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** $T > T_{max}$

Actor: $\pi(a|s;\theta)$
Critic: $V(s,\theta_v)$

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$$

Policy gradient

# A3C Results on Atari 2600

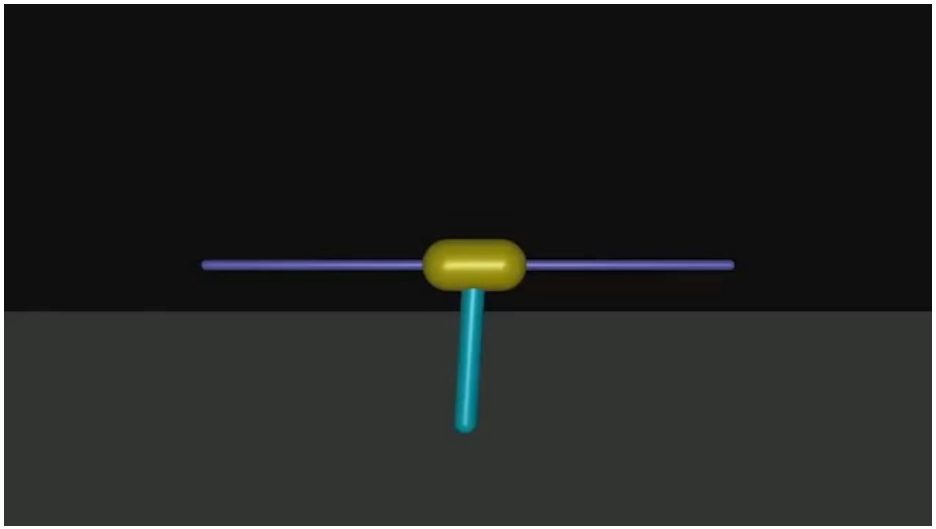
| Method | Training Time | Mean | Median |
|---|---|---|---|
| DQN | 8 days on GPU | 121.9% | 47.5% |
| Gorila | 4 days, 100 machines | 215.2% | 71.3% |
| D-DQN | 8 days on GPU | 332.9% | 110.9% |
| Dueling D-DQN | 8 days on GPU | 343.8% | 117.1% |
| Prioritized DQN | 8 days on GPU | 463.6% | 127.6% |
| A3C, FF | 1 day on CPU | 344.1% | 68.2% |
| A3C, FF | 4 days on CPU | 496.8% | 116.6% |
| A3C, LSTM | 4 days on CPU | 623.0% | 112.6% |

*Table 1.* Mean and median human-normalized scores on 57 Atari games using the human starts evaluation metric. Supplementary Table SS3 shows the raw scores for all games.

| Game | DQN | Gorila | Double | Dueling | Prioritized | A3C FF, 1 day | A3C FF | A3C LSTM |
|---|---|---|---|---|---|---|---|---|
| Alien | 570.2 | 813.5 | 1033.4 | **1486.5** | 900.5 | 182.1 | 518.4 | 945.3 |
| Amidar | 133.4 | 189.2 | 169.1 | 172.7 | 218.4 | **283.9** | 263.9 | 173.0 |
| Assault | 3332.3 | 1195.8 | 6060.8 | 3994.8 | 7748.5 | 3746.1 | 5474.9 | **14497.9** |
| Asterix | 124.5 | 3324.7 | 16837.0 | 15840.0 | **31907.5** | 6723.0 | 22140.5 | 17244.5 |
| Asteroids | 697.1 | 933.6 | 1193.2 | 2035.4 | 1654.0 | 3009.4 | 4474.5 | **5093.1** |
| Atlantis | 76108.0 | 629166.5 | 319688.0 | 445360.0 | 593642.0 | 772392.0 | **911091.0** | 875822.0 |
| Bank Heist | 176.3 | 399.4 | 886.0 | **1129.3** | 816.8 | 946.0 | 970.1 | 932.8 |
| Battle Zone | 17560.0 | 19938.0 | 24740.0 | **31320.0** | 29100.0 | 11340.0 | 12950.0 | 20760.0 |
| Beam Rider | 8672.4 | 3822.1 | 17417.2 | 14591.3 | **26172.7** | 13235.9 | 22707.9 | 24622.2 |
| Berzerk | | | 1011.1 | 910.6 | 1165.6 | **1433.4** | 817.9 | 862.2 |
| Bowling | 41.2 | 54.0 | **69.6** | 65.7 | 65.8 | 36.2 | 35.1 | 41.8 |
| Boxing | 25.8 | 74.2 | 73.5 | **77.3** | 68.6 | 33.7 | 59.8 | 37.3 |
| Breakout | 303.9 | 313.0 | 368.9 | 411.6 | 371.6 | 551.6 | 681.9 | **766.8** |
| Centipede | 3773.1 | **6296.9** | 3853.5 | 4881.0 | 3421.9 | 3306.5 | 3755.8 | 1997.0 |
| Chopper Comman | 3046.0 | 3191.8 | 3495.0 | 3784.0 | 6604.0 | 4669.0 | 7021.0 | **10150.0** |
| Crazy Climber | 50992.0 | 65451.0 | 113782.0 | 124566.0 | 131086.0 | 101624.0 | 112646.0 | **138518.0** |
| Defender | | | 27510.0 | 33996.0 | 21093.5 | 36242.5 | 56533.0 | **233021.5** |
| Demon Attack | 12835.2 | 14880.1 | 69803.4 | 56322.8 | 73185.8 | 84997.5 | 113308.4 | **115201.9** |
| Double Dunk | -21.6 | -11.3 | -0.3 | -0.8 | **2.7** | 0.1 | -0.1 | 0.1 |
| Enduro | 475.6 | 71.0 | 1216.6 | **2077.4** | 1884.4 | -82.2 | -82.5 | -82.5 |
| Fishing Derby | -2.3 | 4.6 | 3.2 | -4.1 | 9.2 | 13.6 | 18.8 | **22.6** |
| Freeway | 25.8 | 10.2 | **28.8** | 0.2 | 27.9 | 0.1 | 0.1 | 0.1 |
| Frostbite | 157.4 | 426.6 | 1448.1 | 2332.4 | **2930.2** | 180.1 | 190.5 | 197.6 |
| Gopher | 2731.8 | 4373.0 | 15253.0 | 20051.4 | **57783.8** | 8442.8 | 10022.8 | 17106.8 |
| Gravitar | 216.5 | **538.4** | 200.5 | 297.0 | 218.0 | 269.5 | 303.5 | 320.0 |
| H.E.R.O. | 12952.5 | 8963.4 | 14892.5 | 15207.9 | 20506.4 | 28765.8 | **32464.1** | 28889.5 |
| Ice Hockey | -3.8 | -1.7 | -2.5 | -1.3 | **-1.0** | -4.7 | -2.8 | -1.7 |
| James Bond | 348.5 | 444.0 | 573.0 | 835.5 | **3511.5** | 351.5 | 541.0 | 613.0 |
| Kangaroo | 2696.0 | 1431.0 | **11204.0** | 10334.0 | 10241.0 | 106.0 | 94.0 | 125.0 |
| Krull | 3864.0 | 6363.1 | 6796.1 | 8051.6 | 7406.5 | **8066.6** | 5560.0 | 5911.4 |
| Kung-Fu Master | 11875.0 | 20620.0 | 30207.0 | 24288.0 | 31244.0 | 3046.0 | 28819.0 | **40835.0** |
| Montezuma's Revenge | 50.0 | **84.0** | 42.0 | 22.0 | 13.0 | 53.0 | 67.0 | 41.0 |
| Ms. Pacman | 763.5 | 1263.0 | 1241.3 | **2250.6** | 1824.6 | 594.4 | 653.7 | 850.7 |
| Name This Game | 5439.9 | 9238.5 | 8960.3 | 11185.1 | 11836.1 | 5614.0 | 10476.1 | **12093.7** |
| Phoenix | | | 12366.5 | 20410.5 | 27430.1 | 28181.8 | 52894.1 | **74786.7** |
| Pit Fall | | | -186.7 | -46.9 | **-14.8** | -123.0 | -78.5 | -135.7 |
| Pong | 16.2 | 16.7 | **19.1** | 18.8 | 18.9 | 11.4 | 5.6 | 10.7 |
| Private Eye | 298.2 | **2598.6** | -575.5 | 292.6 | 179.0 | 194.4 | 206.9 | 421.1 |
| Q*Bert | 4589.8 | 7089.8 | 11020.8 | 14175.8 | 11277.0 | 13752.3 | 15148.8 | **21307.5** |
| River Raid | 4065.3 | 5310.3 | 10838.4 | 16569.4 | **18184.4** | 10001.2 | 12201.8 | 6591.9 |
| Road Runner | 9264.0 | 43079.8 | 43156.0 | 58549.0 | 56990.0 | 31769.0 | 34216.0 | **73949.0** |
| Robotank | 58.5 | 61.8 | 59.1 | **62.0** | 55.4 | 2.3 | 32.8 | 2.6 |
| Seaquest | 2793.9 | 10145.9 | 14498.0 | 37361.6 | **39096.7** | 2300.2 | 2355.4 | 1326.1 |
| Skiing | | | -11490.4 | -11928.0 | **-10852.8** | -13700.0 | -10911.1 | -14863.8 |
| Solaris | | | 810.0 | 1768.4 | **2238.2** | 1884.8 | 1956.0 | 1936.4 |
| Space Invaders | 1449.7 | 1183.3 | 2628.7 | 5993.1 | 9063.0 | 2214.7 | 15730.5 | **23846.0** |
| Star Gunner | 34081.0 | 14919.2 | 58365.0 | 90804.0 | 51959.0 | 64393.0 | 138218.0 | **164766.0** |
| Surround | | | 1.9 | **4.0** | -0.9 | -9.6 | -9.7 | -8.3 |
| Tennis | -2.3 | -0.7 | -7.8 | **4.4** | -2.0 | -10.2 | -6.3 | -6.4 |
| Time Pilot | 5640.0 | 8267.8 | 6608.0 | 6601.0 | 7448.0 | 5825.0 | 12679.0 | **27202.0** |
| Tutankham | 32.4 | 118.5 | 92.2 | 48.0 | 33.6 | 26.1 | **156.3** | 144.2 |
| Up and Down | 3311.3 | 8747.7 | 19086.9 | 24759.2 | 29443.7 | 54525.4 | 74705.7 | **105728.7** |
| Venture | 54.0 | **523.4** | 21.0 | 200.0 | 244.0 | 19.0 | 23.0 | 25.0 |
| Video Pinball | 20228.1 | 112093.4 | 367823.7 | 110976.2 | 374886.9 | 185852.6 | 331628.1 | **470310.5** |
| Wizard of Wor | 246.0 | 10431.0 | 6201.0 | 7054.0 | 7451.0 | 5278.0 | 17244.0 | **18082.0** |
| Yars Revenge | | | 6270.6 | **25976.5** | 5965.1 | 7270.8 | 7157.5 | 5615.5 |
| Zaxxon | 831.0 | 6159.4 | 8593.0 | 10164.0 | 9501.0 | 2659.0 | **24622.0** | 23519.0 |

# A3C Results

TORCS



MuJoCo

Labyrinth

Lillicrap, Timothy P., Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. "**Continuous control with deep reinforcement learning**." ICLR 2016

# DEEP DETERMINISTIC POLICY GRADIENT (DDPG)

# Deterministic Policy Gradient

Stochastic policy gradient (Sutton et al., 1999)

$$\nabla_\theta J(\pi_\theta) = \int_{\mathcal{S}} \rho^\pi(s) \int_{\mathcal{A}} \nabla_\theta \pi_\theta(a|s) Q^\pi(s,a) \mathrm{d}a \mathrm{d}s$$
$$= \mathbb{E}_{s\sim\rho^\pi, a\sim\pi_\theta} \left[ \nabla_\theta \log \pi_\theta(a|s) Q^\pi(s,a) \right]$$

Deterministic policy gradient (Silver et al., 2014)

$$\nabla_\theta J(\mu_\theta) = \int_{\mathcal{S}} \rho^\mu(s) \nabla_\theta \mu_\theta(s) \left. \nabla_a Q^\mu(s,a) \right|_{a=\mu_\theta(s)} \mathrm{d}s$$
$$= \mathbb{E}_{s\sim\rho^\mu} \left[ \nabla_\theta \mu_\theta(s) \left. \nabla_a Q^\mu(s,a) \right|_{a=\mu_\theta(s)} \right] \quad (9)$$

- Deterministic policy function
- Off-policy learning is possible
- Requires less samples to approximate the gradient

Actor-critic method

- Actor: Update the policy function using the policy gradient
- Critic: Update Q function using the TD error

# Review: DQN

- Experience Replay
- Target Network

**Algorithm 1: deep Q-learning with experience replay.**

Initialize replay memory $D$ to capacity $N$

Initialize action-value function $Q$ with random weights $\theta$

Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$

**For** episode $= 1, M$ **do**

  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

  **For** $t = 1$,T **do**

    With probability $\varepsilon$ select a random action $a_t$

    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$

    Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$

    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $D$

    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$

    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$

    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters $\theta$

    Every $C$ steps reset $\hat{Q} = Q$

  **End For**

**End For**

# DDPG

**Algorithm 1** DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights $\theta^Q$ and $\theta^\mu$.
Initialize target network $Q'$ and $\mu'$ with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer $R$
**for** episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state $s_1$
  **for** t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action $a_t$ and observe reward $r_t$ and observe new state $s_{t+1}$
    Store transition $(s_t, a_t, r_t, s_{t+1})$ in $R$
    Sample a random minibatch of $N$ transitions $(s_i, a_i, r_i, s_{i+1})$ from $R$
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  **end for**
**end for**

Based on the DQN architecture

# DDPG Results