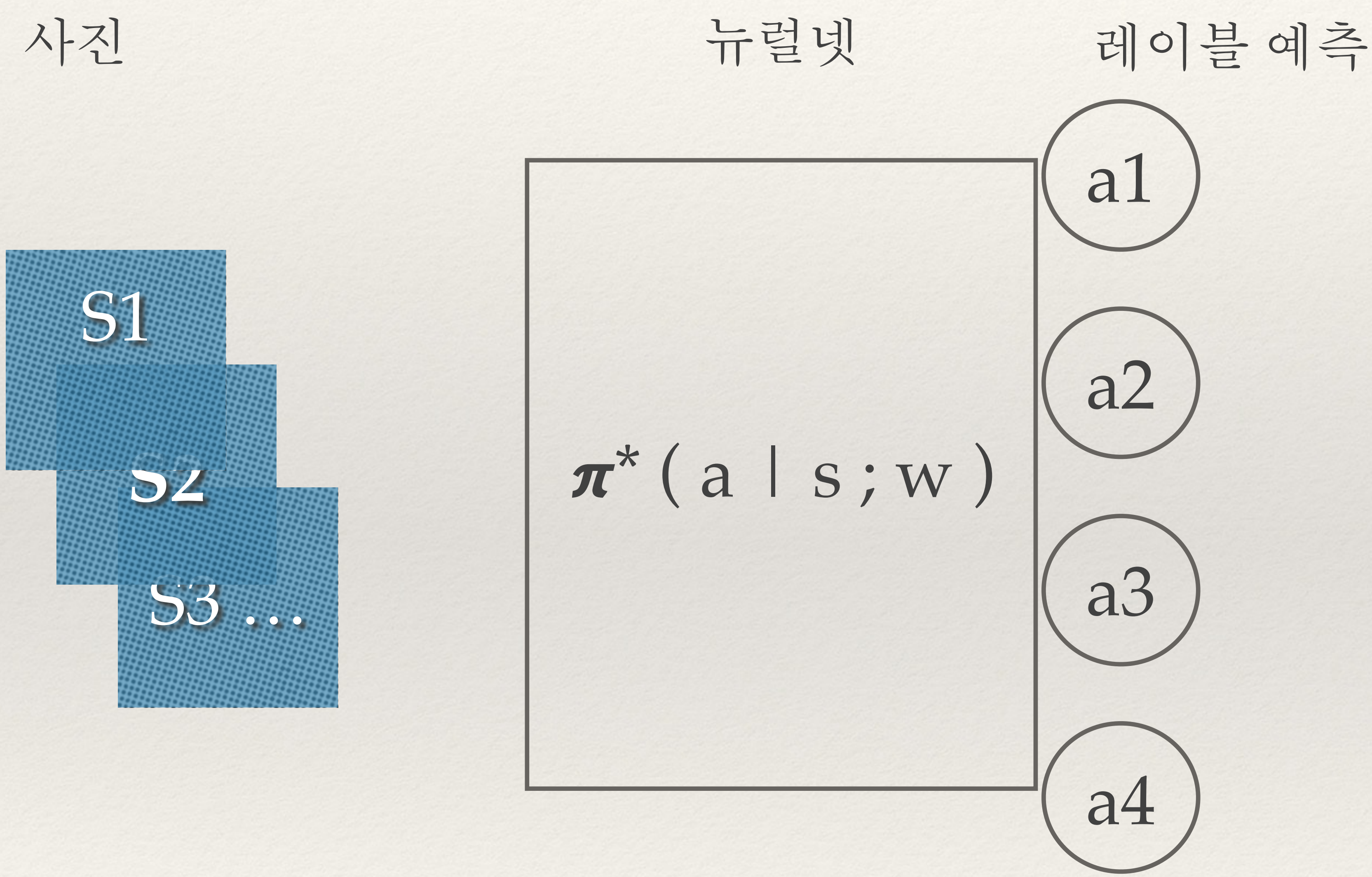
A3C

2016. 11. 21.
박성준
park@move.is
Move Inc.

Today's Agenda

- Recap on DqN
- Review of DeepMind's A3C
  - Asynchronous Methods for Deep Reinforcement Learning (https://arxiv.org/pdf/1602.01783v2.pdf)
  - Karpathy's Blog (http://karpathy.github.io/2016/05/31/rl/)
  - David Silver's own take on DqN from his course on RL (http://www0.cs.ucl.ac.uk/staff/D.Silver/web/Teaching.html)
  - Gorila (https://arxiv.org/pdf/1507.04296v2.pdf)

Recap I : CNN



Choose the architecture π

Choose loss function

- $\sum(y_i - \pi(a_i \mid s; w))^2$

Initialize π

Minimize loss function using SGD, for instance,

$w \longleftarrow w + \nabla_w \mathrm{Loss}$

Arrive at π^*

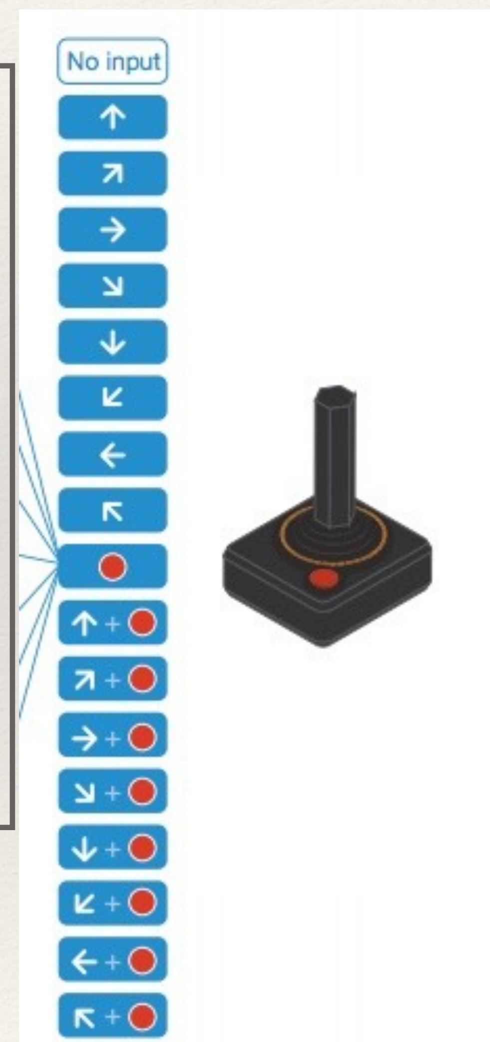
Recap II : DqN

사진

뉴럴넷

$\boldsymbol{\pi}^{*}\,(\,a \mid s\,;\theta\,)$

조작 예측



어떻게 맞는 레이블을 찾지 ?

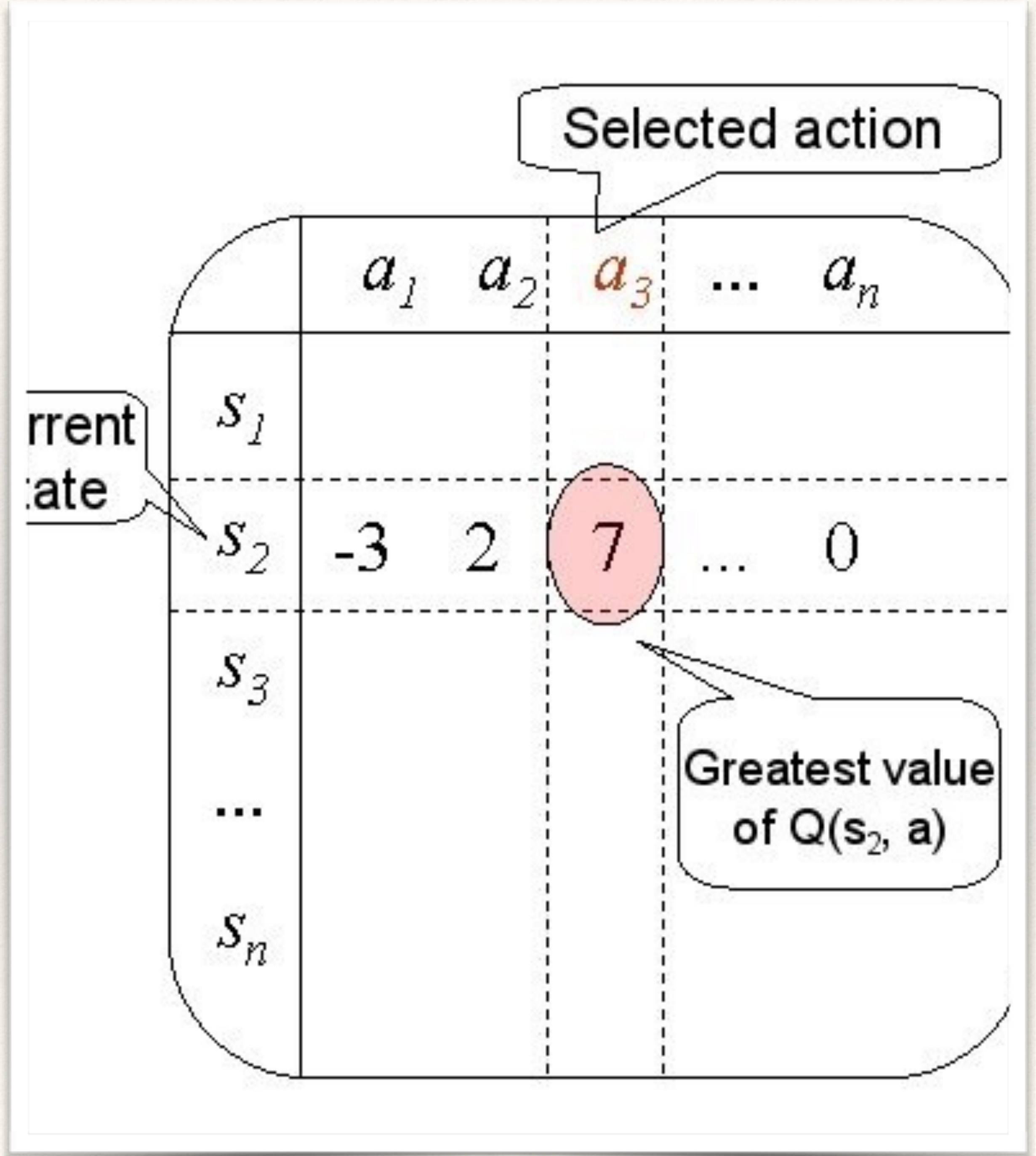
Action을 골라서 Episode를 다 플레이를 하고 나서 제일 점수를 많이 따는 조작이 맞는 조작이라 하면 되잖아.

q(a | s) 는 s 라는 화면에서 a 라는 조작을 한 후에 딴 점수의 총합이다, 라고 하면

$\boldsymbol{\pi}$ = greedy (q(a | s)) 하면 된다

q(a | s) : mathematical formalism

- q 테이블에는 s라는 state에서 a라는 action을 하고 나서 에피소드 끝까지 play를 하고 얻은 reward의 합을 채우면 된다.
- 그러고나면 $\boldsymbol{\pi}$ = greedy (q(a | s)) 하면 되고
- 그런데 이 테이블을 채울수가 없어 function approximation을 한다
- 그러자면 loss function이 필요한데 이건 Bellman equation을 응용하면 된다
- Loss=[r + max q(a′ | s′ ; θ) - q(a | s ; θ)]^2



DQN to A3C

DqN

- CNN as function approximator
- Delayed param update
- Experience replay
- q

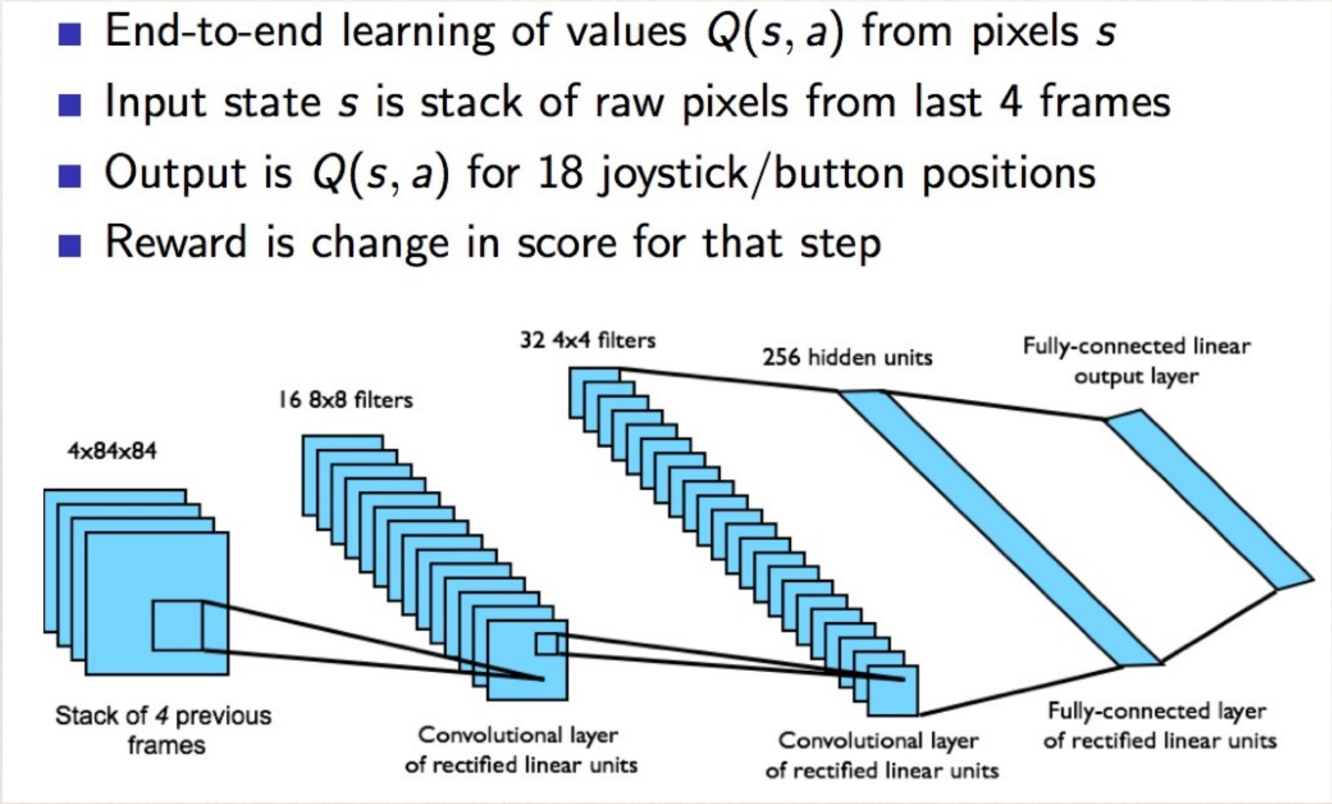
A3C

- CNN + (LSTM) as function approximator
- Delayed param update
- Threading —> async update
- π directly

$\boldsymbol{\pi}$ (a | s ; θ) : mathematical formalism

- Parameterized Policy $\boldsymbol{\pi}(a_t \mid s_t ; \theta)$ directly
- Maximize objective function $E[R_t]$ (accrued rewards) under $\boldsymbol{\pi}(a_t \mid s_t;\theta)$
- Policy Gradient Theorem + using a baseline to reduce variance —> $\nabla_\theta E[R_t] = \nabla_\theta \log \boldsymbol{\pi}(a_t \mid s_t;\theta)(R_t - V(s_t))$

Network Architecture



- Added two separate heads
- one linear head for $V(s;\theta_v)$
- Another softmax head for $\boldsymbol{\pi}(a \mid s;\theta)$
- For A3C.LSTM added a layer of 256 LSTM cells

Gorila & Async RL

Massively Parallel Methods for Deep Reinforcement Learning

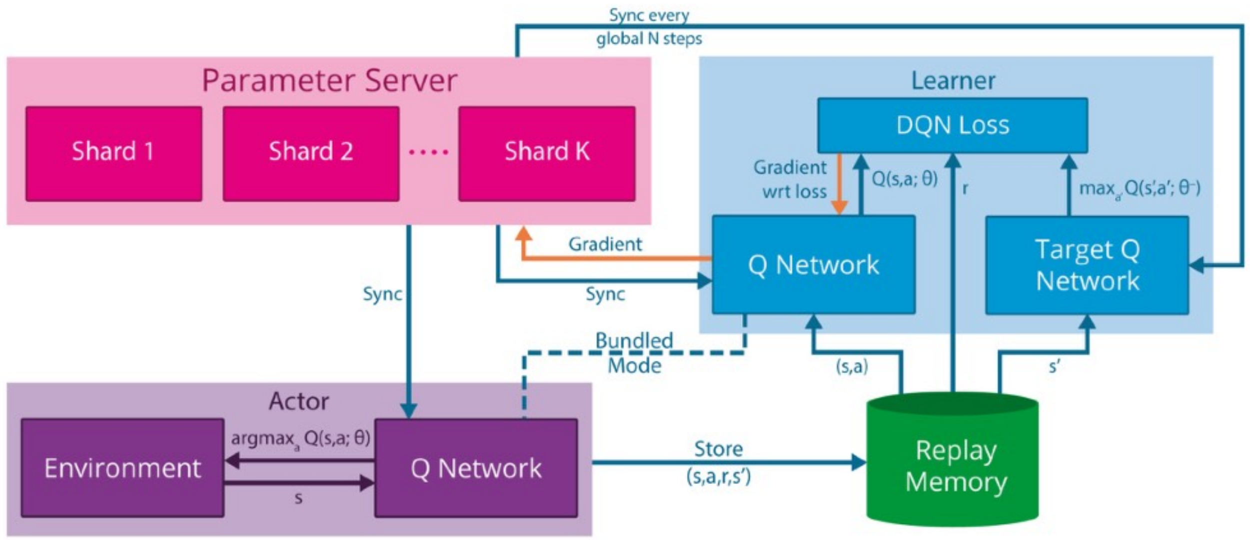


Figure 2. The Gorila agent parallelises the training procedure by separating out learners, actors and parameter server. In a single experiment, several learner processes exist and they continuously send the gradients to parameter server and receive updated parameters. At the same time, independent actors can also in parallel accumulate experience and update their Q-networks from the parameter server.

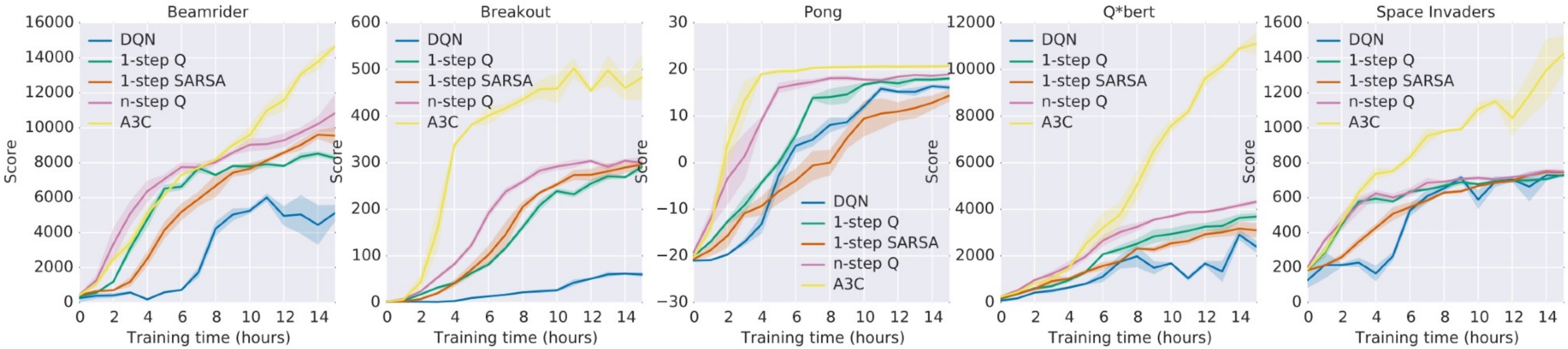
Algorithm 1 Asynchronous one-step Q-learning - pseudocode for each actor-learner thread.

// Assume global shared θ, θ^-, 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 ϵ-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 θ: $d\theta \leftarrow d\theta + \frac{\partial (y - Q(s,a;\theta))^2}{\partial \theta}$
  $s = s'$
  $T \leftarrow T + 1$ 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 θ using $d\theta$.
    Clear gradients $d\theta \leftarrow 0$.
  end if
until $T > T_{max}$

Faster & Better



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%

Back-ups

Compatible Function Approximation

Theorem (Compatible Function Approximation Theorem)

If the following two conditions are satisfied:

1. *Value function approximator is compatible to the policy*

$$\nabla_w Q_w(s,a) = \nabla_\theta \log \pi_\theta(s,a)$$

2. *Value function parameters w minimise the mean-squared error*

$$\varepsilon = \mathbb{E}_{\pi_\theta}\left[\left(Q^{\pi_\theta}(s,a) - Q_w(s,a)\right)^2\right]$$

Then the policy gradient is exact,

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ Q_w(s,a)\right]$$

Algorithm S2 Asynchronous n-step Q-learning - pseudocode for each actor-learner thread.

// Assume global shared parameter vector θ.
// Assume global shared target parameter vector θ^-.
// 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 ϵ-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 θ': $d\theta \leftarrow d\theta + \frac{\partial \left(R - Q(s_i, a_i; \theta')\right)^2}{\partial \theta'}$
    end for
    Perform asynchronous update of θ using $d\theta$.
    if $T \mod I_{target} == 0$ **then**
        $\theta^- \leftarrow \theta$
    end if
until $T > T_{max}$

Params I

- From the paper
  - 16 actor-leaner threads
  - $t_{max} = 5$ and $I_{update} = 5$
  - shared RMSProp
  - Same preprocessing as Mnih er al 2015 and action repeat of 4
  - entropy beta = 0.01
  - Initial leaning rate sampled from LogUniform(10^{-4}, 10^{-2})
  - from Mnih, use up till 256 hidden units + ReLU and then fed to the output heads (or LSTM cells)
- From : aravindsrinivas commented on Apr 25 • edited ; @miyosuda
  - I mailed the authors (from DeepMind). These are some hyper parameters that they explicitly told me in the mail:
  - The decay parameter (called alpha in the paper) for RMSProp was 0.99 and the regularization constant (called epsilon in the paper) was 0.1. The maximum allowed gradient norm was 40. The best learning rates were around 7*10^-4. Backups of length 20 were used which corresponds to setting the t_max parameter to 20.
  - Hi, I also confirmed that
  - 1) the critic learning rate must be half the actor's..
  - 2) the LR must be linearly annealed to 0 over the course of training.
  - 3) the parameters 'g' and 'theta' (moving average of RMS of gradients and of course the parameters) are shared across the threads. (Unlike your earlier version of having separate RMS moving averages). Also, there is no need of locking and updating.
  - 4) t_max = 20 means 20 perceived frames (80 with Frame skip as per game)... Not 20 states .. ie not 20 84844 tensors, but rather 20 84*84 frames...

Params II

- From muupan (https://github.com/muupan/async-rl.wiki.git) ; On the authors' implementation details: I received a confirmation by e-mail from Dr. Mnih:
- On optimization
  - They use the exact RMSprop represented by the equations (8) and (9)
  - The RMSprop parameters they used are: eta=7e-4, epsilon=0.1, alpha=0.99
  - They linearly decrease eta to zero in the course of training
  - They keep only single RMSprop 'g' while summing up the gradients of pi and V
  - They multiply the gradients of V by 0.5
  - They didn't clip losses
  - They ran it 320 million frames (= 80 million non-skipped frames) for one-day results, 1 billion frames for four-day results
- On networks
  - Pi and V share the network except the last layers
  - They initialized parameters with default Torch initialization: https://github.com/torch/nn/blob/master/Linear.lua
- On Atari
  - They clipped rewards so that they are in [-1, 1]