

Actor-Critic

Hung-yi Lee

Asynchronous Advantage Actor-Critic (A3C)

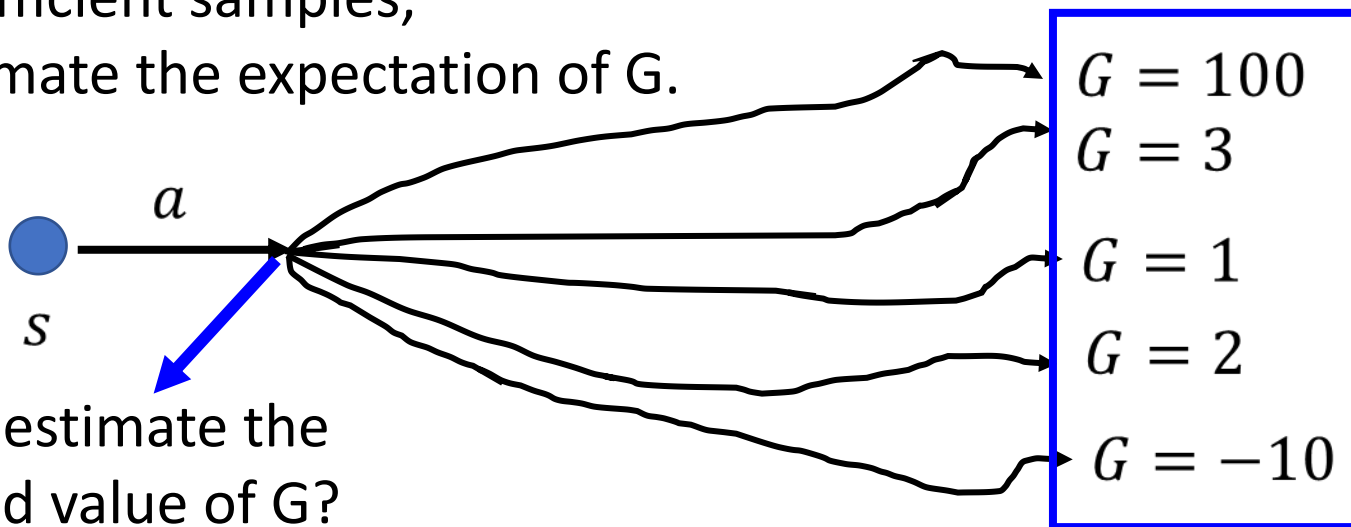
Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, Koray Kavukcuoglu, “Asynchronous Methods for Deep Reinforcement Learning”, ICML, 2016

Review – Policy Gradient

$$\nabla \bar{R}_\theta \approx \frac{1}{N} \sum_{n=1}^N \sum_{t=1}^{T_n} \left(\underbrace{\sum_{t'=t}^{T_n} \gamma^{t'-t} r_{t'}^n}_{G_t^n : \text{obtained via interaction}} - \underbrace{b}_{\text{baseline}} \right) \nabla \log p_\theta(a_t^n | s_t^n)$$

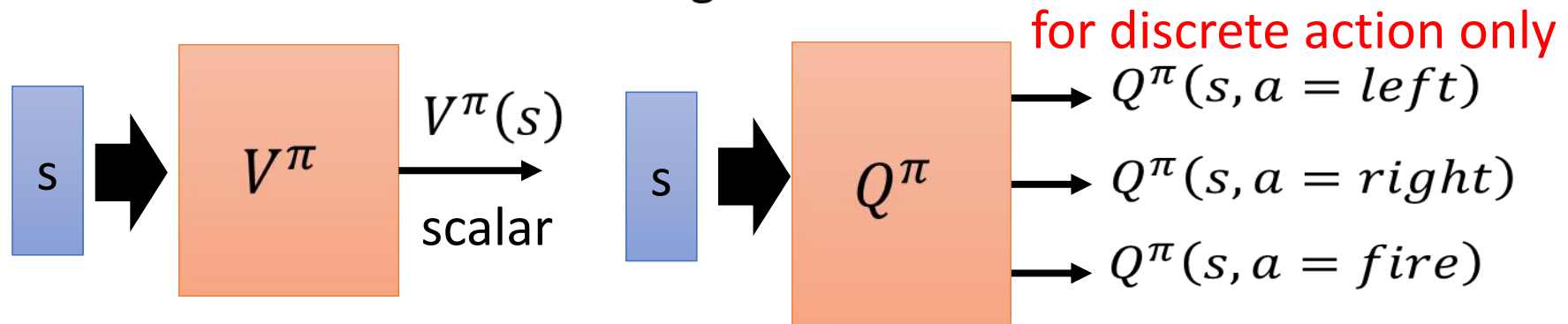
Very unstable

With sufficient samples,
approximate the expectation of G.



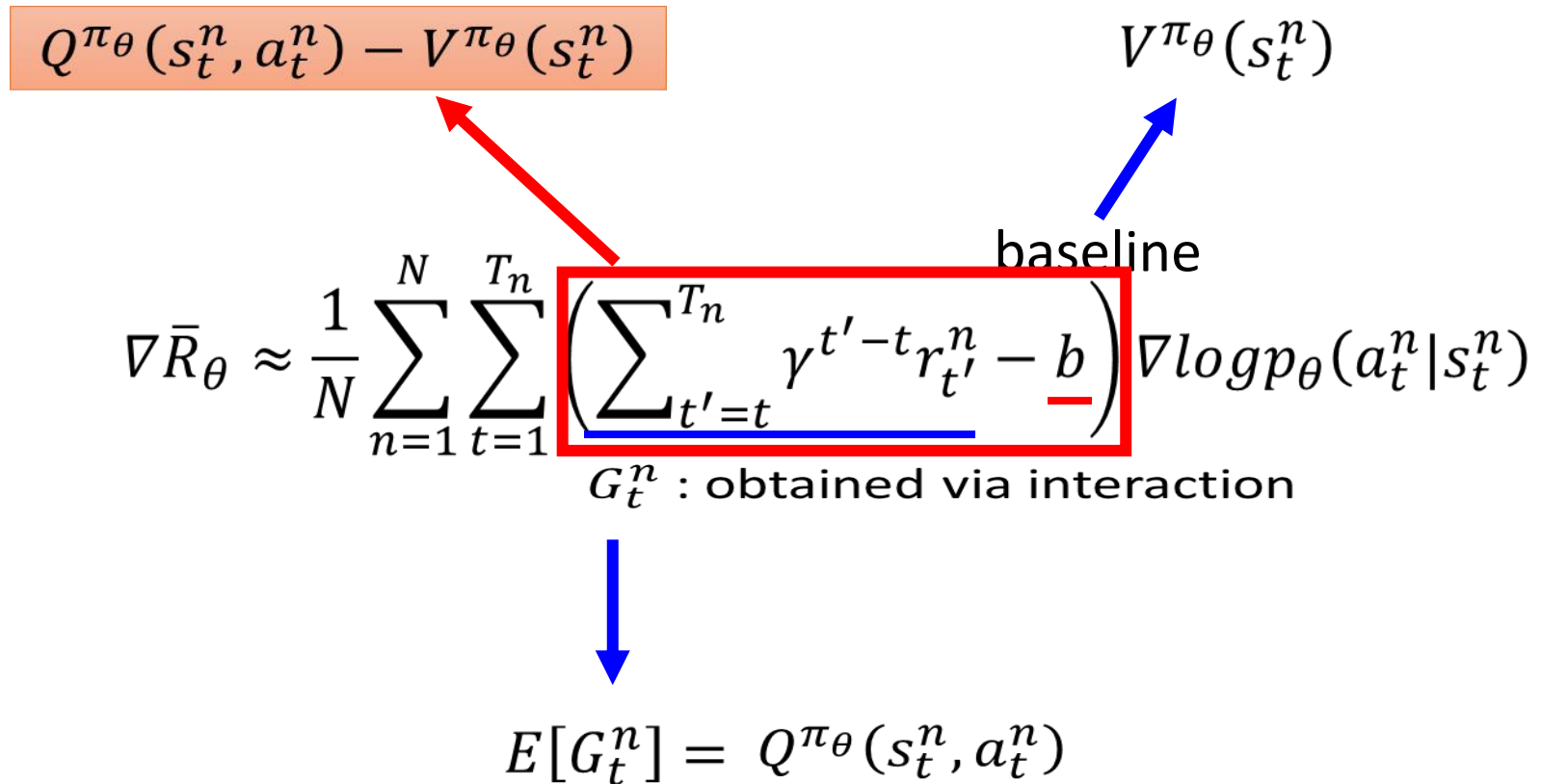
Review – Q-Learning

- State value function $V^\pi(s)$
 - When using actor π , the *cumulated* reward expects to be obtained after visiting state s
- State-action value function $Q^\pi(s, a)$
 - When using actor π , the *cumulated* reward expects to be obtained after taking a at state s



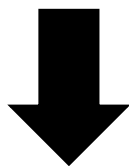
Estimated by TD or MC

Actor-Critic



Advantage Actor-Critic

$$Q^{\pi}(s_t^n, a_t^n) - V^{\pi}(s_t^n)$$



$$r_t^n + V^{\pi}(s_{t+1}^n) - V^{\pi}(s_t^n)$$

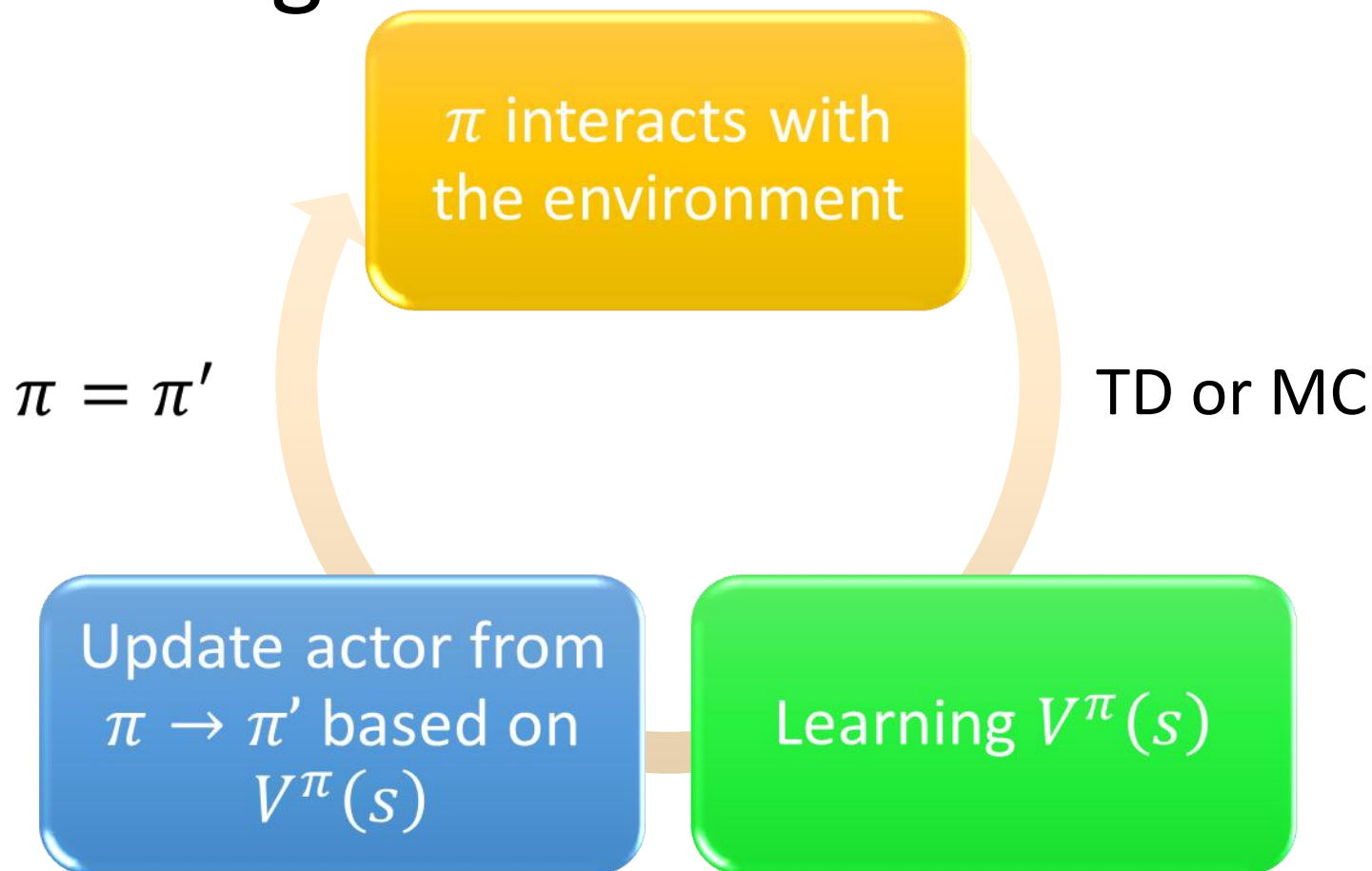
Estimate two networks? We can only estimate one.

Only estimate state value
A little bit variance

$$Q^{\pi}(s_t^n, a_t^n) = E[r_t^n + V^{\pi}(s_{t+1}^n)]$$

$$Q^{\pi}(s_t^n, a_t^n) = r_t^n + V^{\pi}(s_{t+1}^n)$$

Advantage Actor-Critic

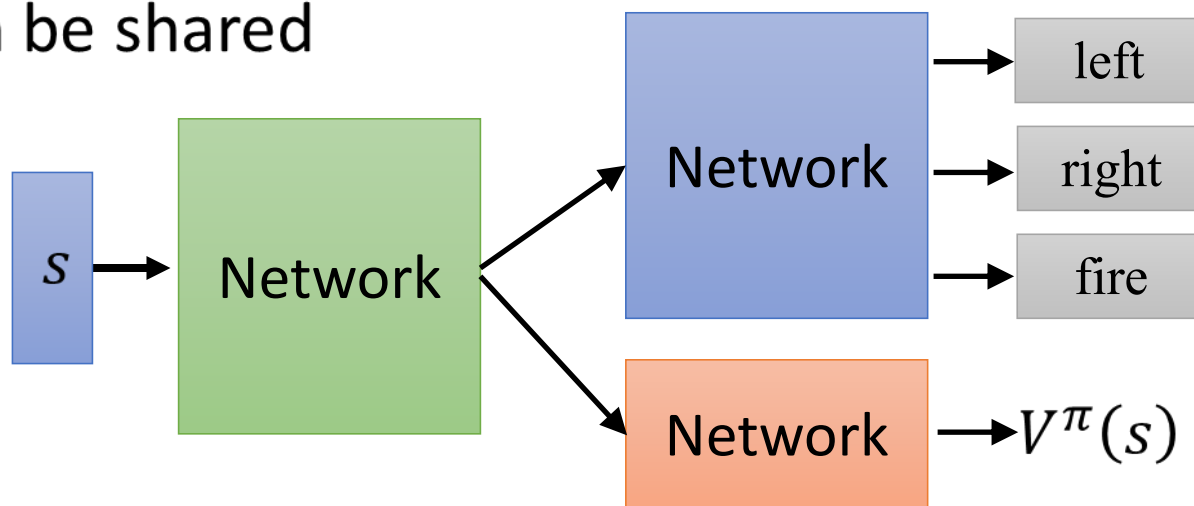


$$\nabla \bar{R}_\theta \approx \frac{1}{N} \sum_{n=1}^N \sum_{t=1}^{T_n} (r_t^n + V^\pi(s_{t+1}^n) - V^\pi(s_t^n)) \nabla \log p_\theta(a_t^n | s_t^n)$$

Advantage Actor-Critic

- Tips

- The parameters of actor $\pi(s)$ and critic $V^\pi(s)$ can be shared



- Use output entropy as regularization for $\pi(s)$
 - Larger entropy is preferred \rightarrow exploration

Asynchronous Advantage Actor-Critic (A3C)

The idea is from 李思叡

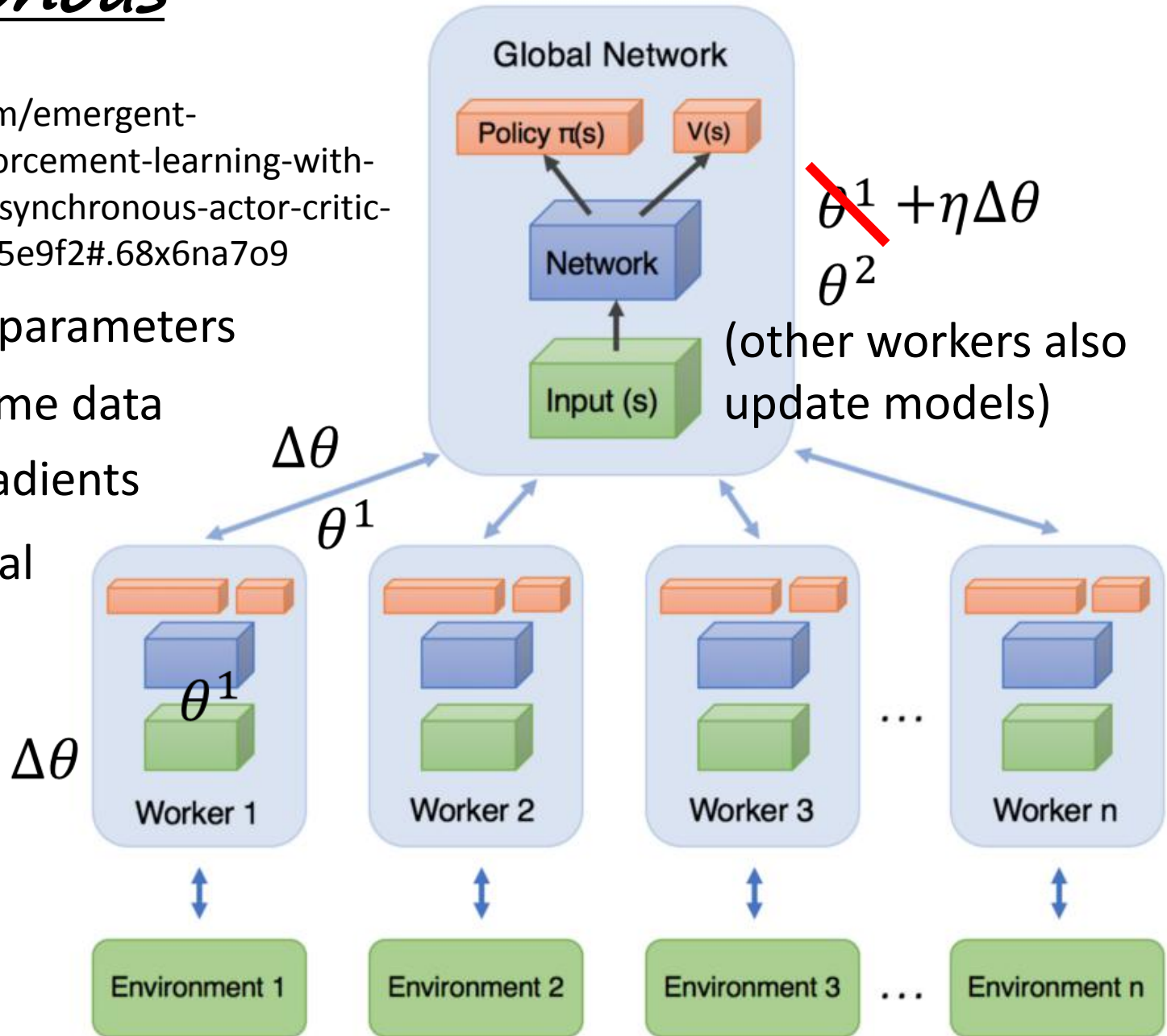


Asynchronous

Source of image:

<https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-8-asynchronous-actor-critic-agents-a3c-c88f72a5e9f2#.68x6na7o9>

1. Copy global parameters
2. Sampling some data
3. Compute gradients
4. Update global models



Pathwise Derivative Policy Gradient

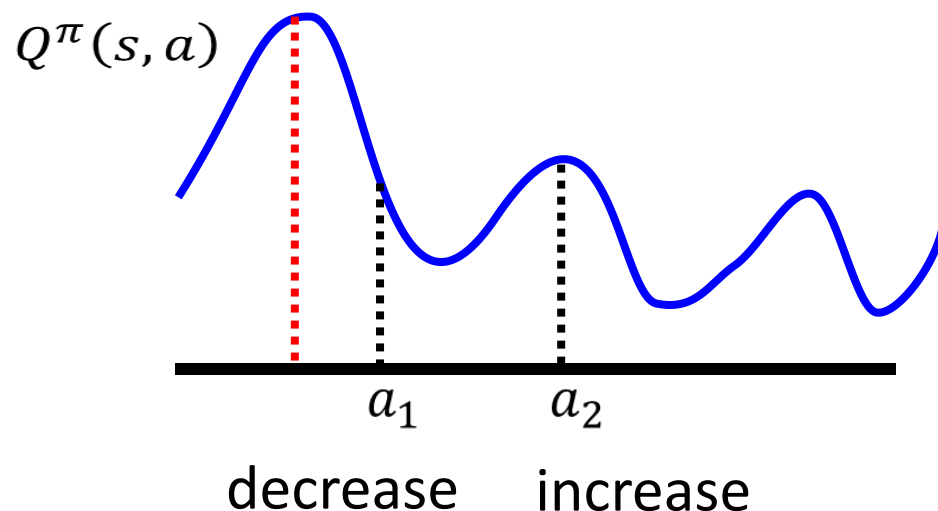
David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, Martin Riedmiller,
“Deterministic Policy Gradient Algorithms”, ICML, 2014

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess,

Tom Erez, Yuval Tassa, David Silver, Daan Wierstra, “CONTINUOUS CONTROL WITH DEEP REINFORCEMENT LEARNING”, ICLR, 2016

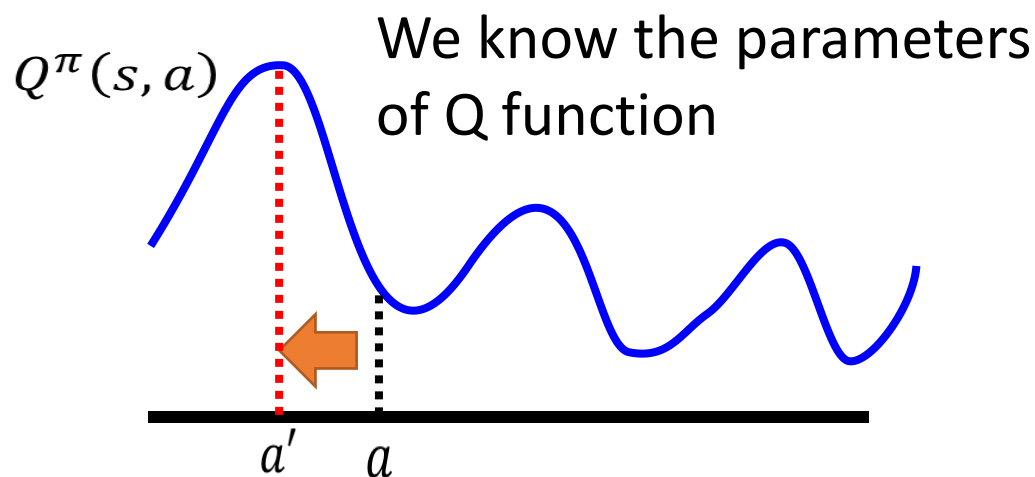
Another Way to use Critic

Original Actor-critic



Pathwise derivative policy gradient

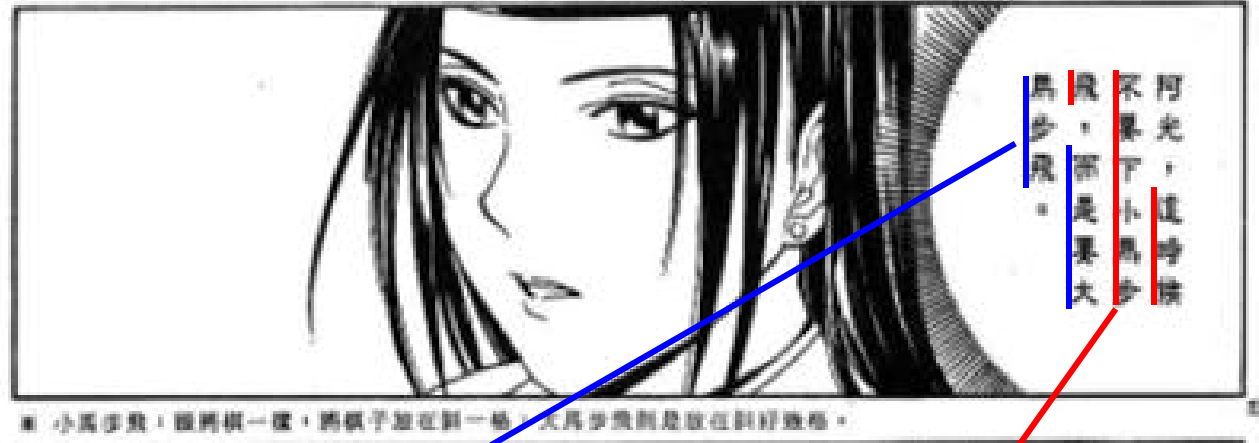
From Q function we know that taking a' at state s is better than a



Actor



Critic



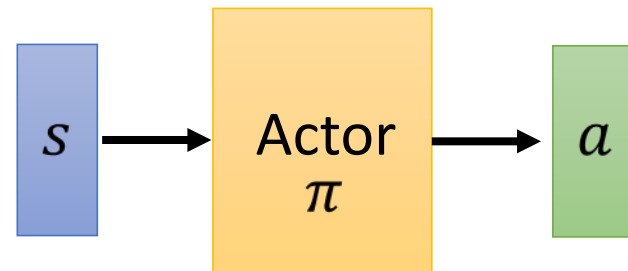
Pathwise derivative
policy gradient

Original Actor-critic

<http://www.cartomad.com/comic/109000081104011.html>

Action a is a *continuous vector*

$$a = \arg \max_a Q(s, a)$$



Actor as the solver of this optimization problem

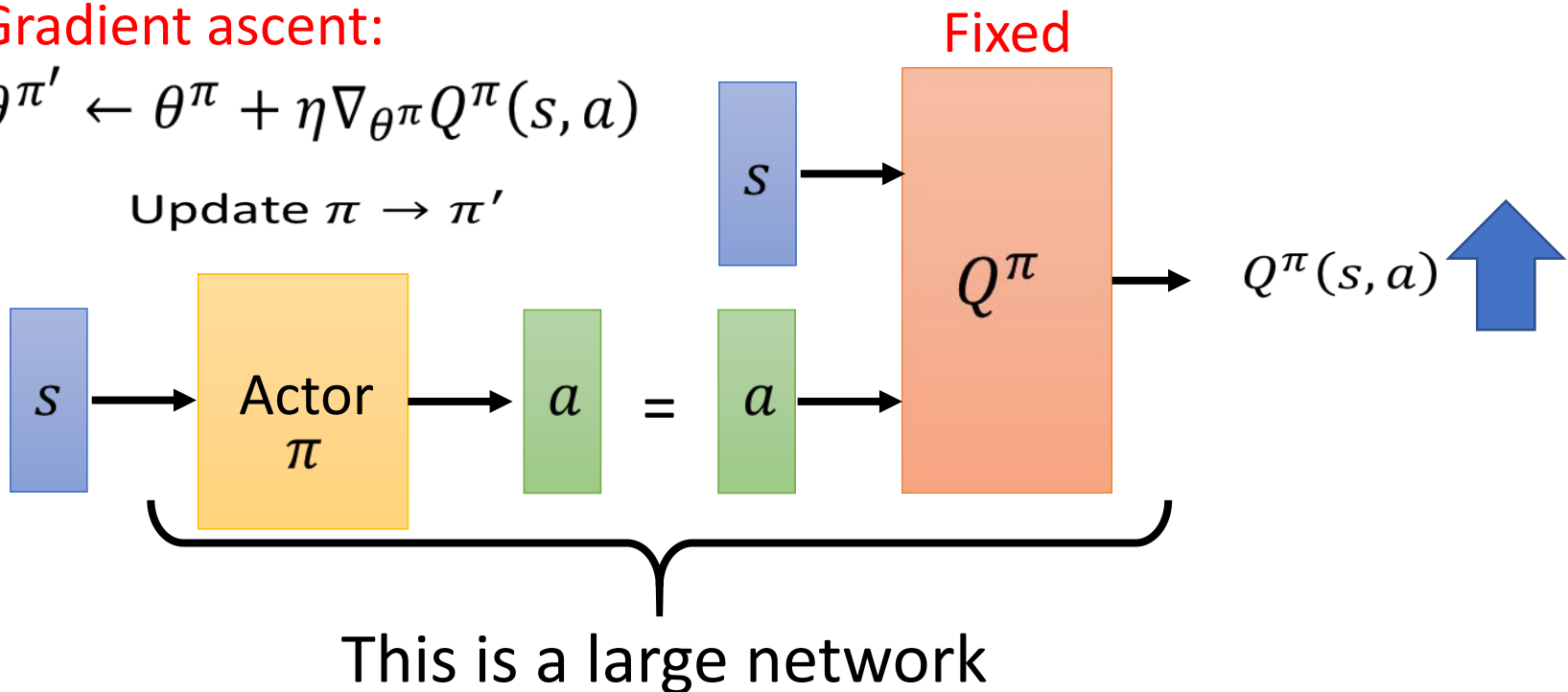
Pathwise Derivative Policy Gradient

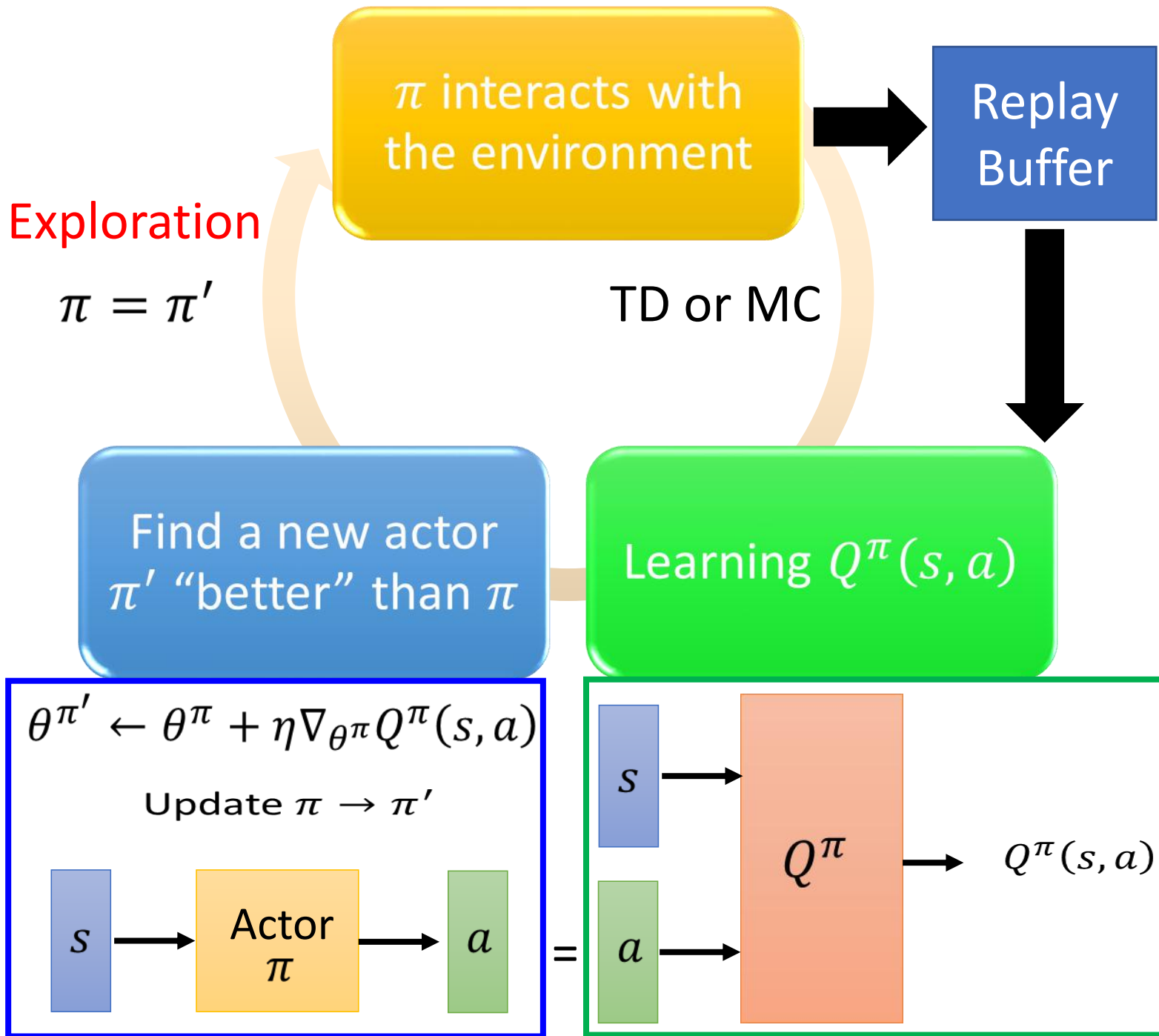
$$\pi'(s) = \arg \max_a Q^\pi(s, a) \quad \leftarrow \text{a is the output of an actor}$$

Gradient ascent:

$$\theta^{\pi'} \leftarrow \theta^\pi + \eta \nabla_{\theta^\pi} Q^\pi(s, a)$$

Update $\pi \rightarrow \pi'$





Q-Learning Algorithm

- Initialize Q-function Q , target Q-function $\hat{Q} = Q$

- For each time step t
 - Given state s_t , take action a_t based on Q (exploration)
 - Obtain reward r_t , and reach new state s_{t+1}
 - Store (s_t, a_t, r_t, s_{t+1}) into buffer
 - Sample (s_i, a_i, r_i, s_{i+1}) from buffer (usually a batch)
 - Target $y = r_i + \max_a \hat{Q}(s_{i+1}, a)$
 - Update the parameters of Q to make $Q(s_i, a_i)$ close to y (regression)
 - Update the parameters of π to maximize $Q(s_i, \pi(s_i))$
 - Every C steps reset $\hat{Q} = Q$
 - Every C steps reset $\hat{\pi} = \pi$

Q-Learning Algorithm ➡ Pathwise Derivative Policy Gradient

- Initialize Q-function Q , target Q-function $\hat{Q} = Q$, actor π , target actor $\hat{\pi} = \pi$
- In each episode
 - For each time step t
 - Given state s_t , take action a_t based on Q (exploration)
 - 1 • Obtain reward r_t , and reach new state s_{t+1}
 - Store (s_t, a_t, r_t, s_{t+1}) into buffer
 - Sample (s_i, a_i, r_i, s_{i+1}) from buffer (usually a batch)
 - Target $y = r_i + \max_a \hat{Q}(s_{i+1}, a)$
 - Update the parameters of Q to make $Q(s_i, a_i)$ close to y (regression)
 - 2 • Update the parameters of π to maximize $Q(s_i, \pi(s_i))$
 - Every C steps reset $\hat{Q} = Q$ $\hat{Q}(s_{i+1}, \hat{\pi}(s_{i+1}))$
 - Every C steps reset $\hat{\pi} = \pi$

3

4

Connection with GAN

Method	GANs	AC
Freezing learning	yes	yes
Label smoothing	yes	no
Historical averaging	yes	no
Minibatch discrimination	yes	no
Batch normalization	yes	yes
Target networks	n/a	yes
Replay buffers	no	yes
Entropy regularization	no	yes
Compatibility	no	yes

David Pfau, Oriol Vinyals, “Connecting Generative Adversarial Networks and Actor-Critic Methods”, arXiv preprint, 2016