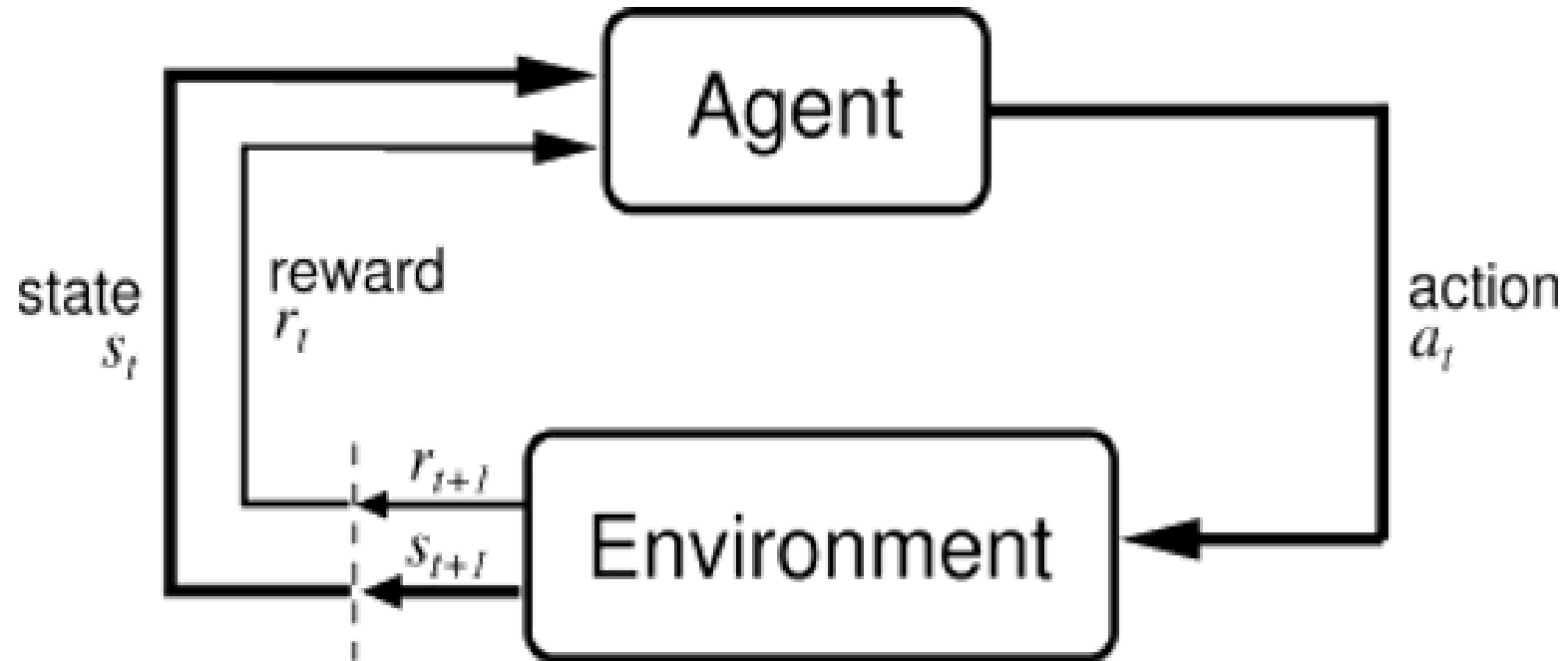


# Reinforcement learning

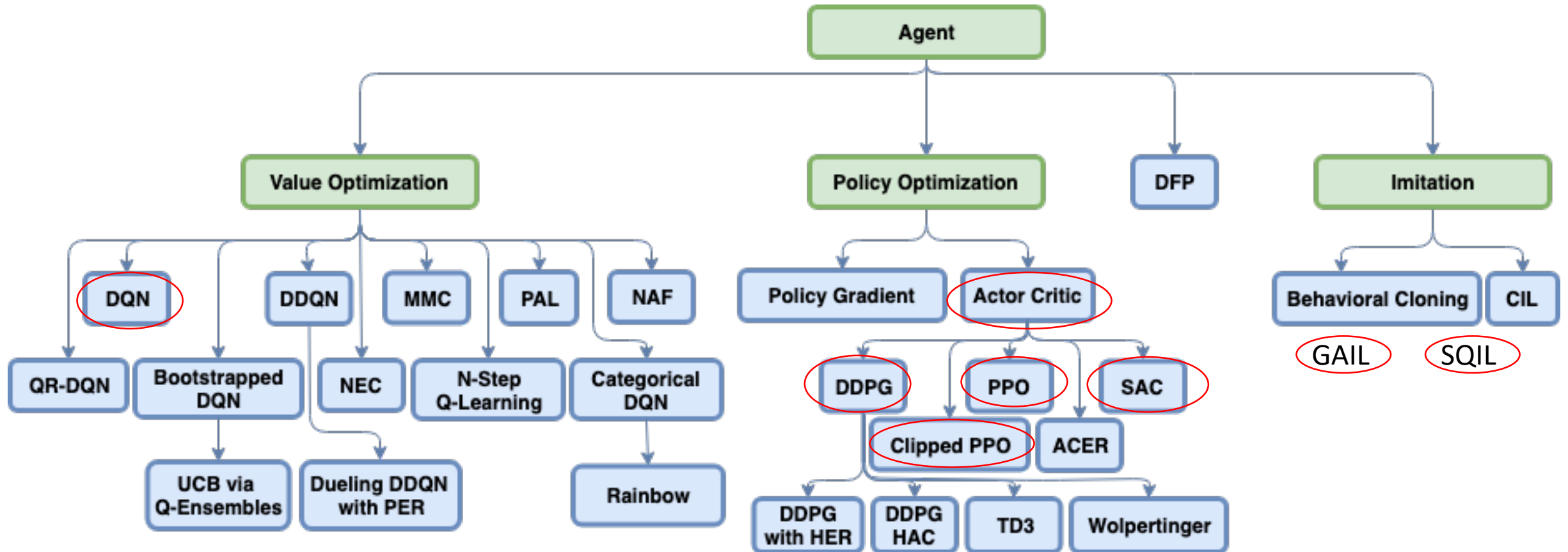


(Sutton and Barto, 1998)

# Machine learning mechanism

- Define a function to be learned:  $y^n = f(x^n)$
- Define a loss function  $\mathcal{L}(f)$  to describe the error between  $y^n$  and  $\hat{y}^n$
- Find the optimal parameters that minimize  $\mathcal{L}(f)$

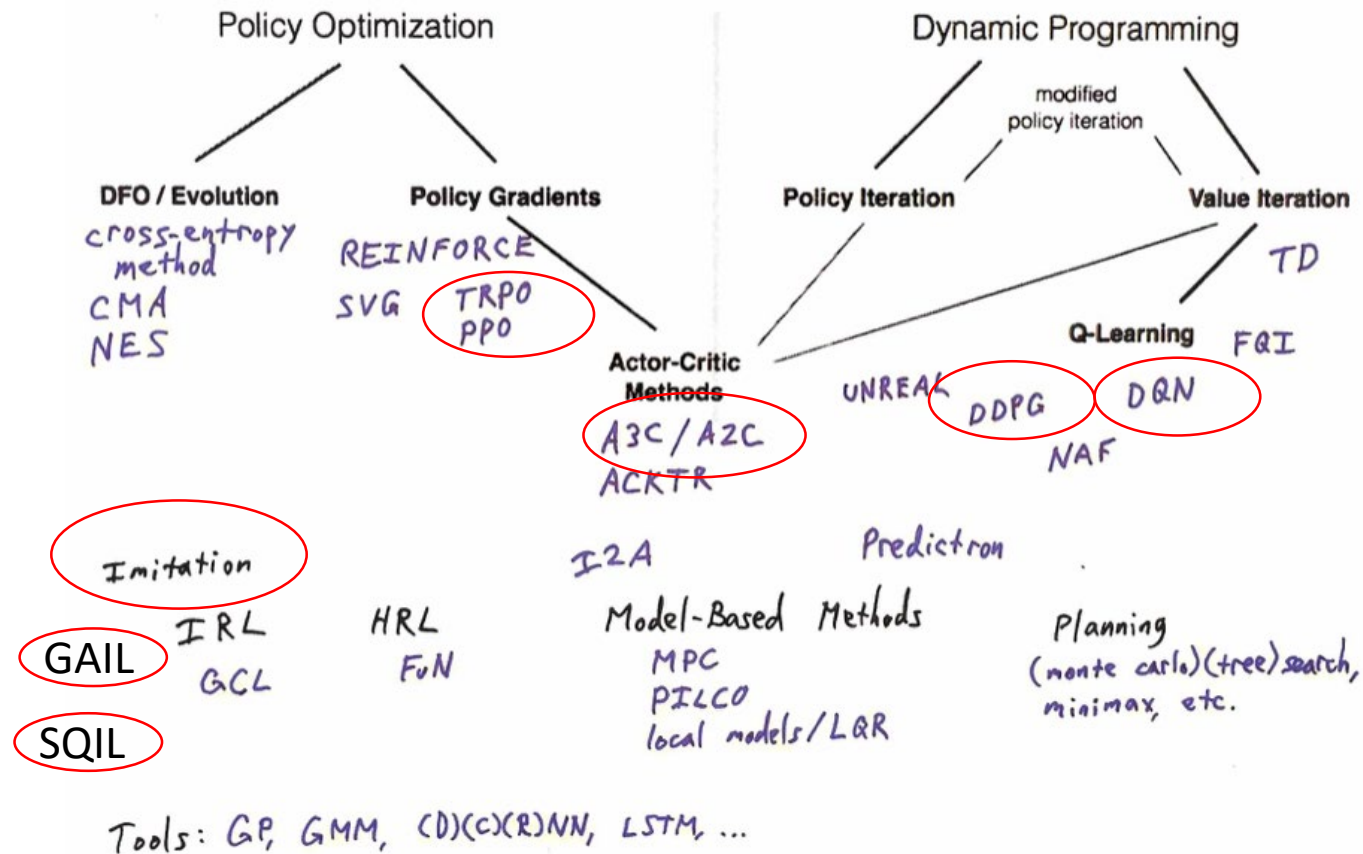
# Value optimization vs policy optimization



圖片來源: [https://nervanasystems.github.io/coach/selecting\\_an\\_algorithm.html](https://nervanasystems.github.io/coach/selecting_an_algorithm.html)

# Policy optimization vs dynamic programming

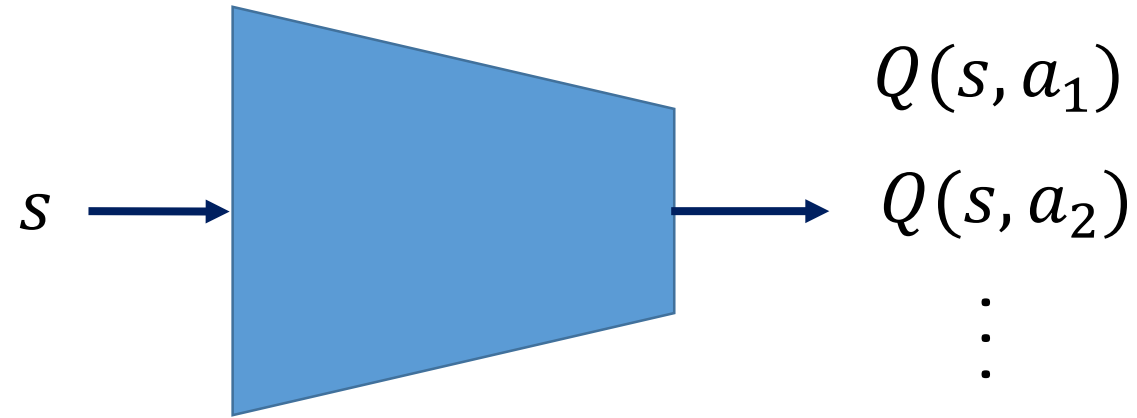
## (PO)MDP RL Algorithms Landscape



Awesome RL libs: rllkit @vitchyr, pytorch-a2c-ppo-acktr @ikostrikov, ACER @Kaixhin

圖片來源: <https://images.app.goo.gl/nPJftvi3CChYmyFDA>

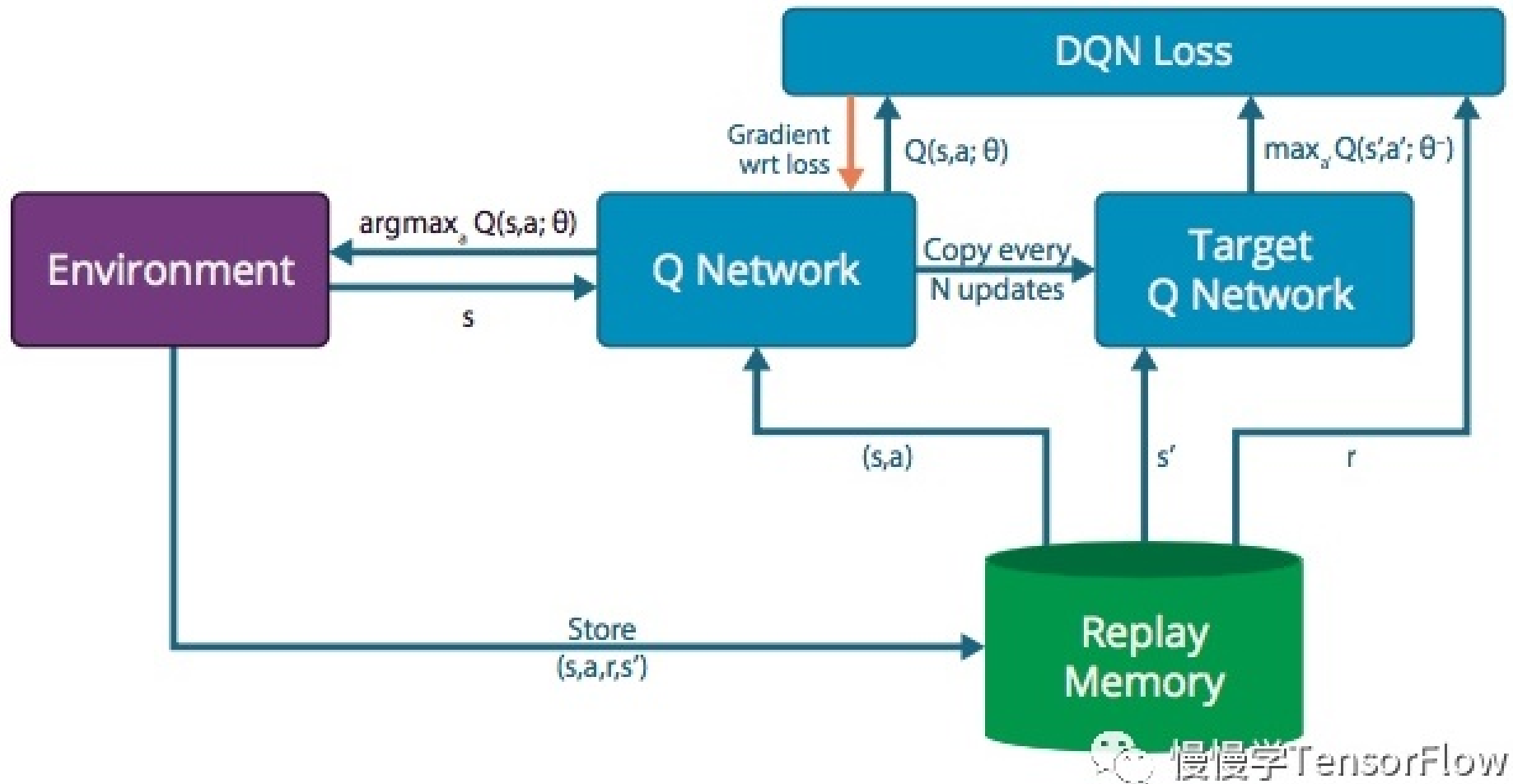
# Deep Q-Network (DQN)



**Bellman Equation:**

$$Q^*(s, a) = \sum_{s'} P(s'|s, a) \left[ R(s, a, s') + \gamma \max_{a'} Q^*(s', a') \right]$$

# Deep Q-Network (DQN)



圖片來源: [https://zhuanlan.zhihu.com/p/25546213?from\\_voters\\_page=true](https://zhuanlan.zhihu.com/p/25546213?from_voters_page=true)

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Petersen, S. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540), 529.

# Policy gradient

$$\tau = (s_1, a_1, r_1, s_2, a_2, r_2, \dots, s_T, a_T)$$

$$p_{\theta}(\tau) = p(s_1)p_{\theta}(a_1|s_1)p(s_2|s_1, a_1)p_{\theta}(a_2|s_2)p(s_3|s_2, a_2) \dots$$

$$R(\tau) = \sum_{t=1}^T r_t$$

$$\bar{R}_{\theta} = \sum R(\tau) p_{\theta}(\tau) = E_{\tau \sim p_{\theta}(\tau)}[R(\tau)]$$

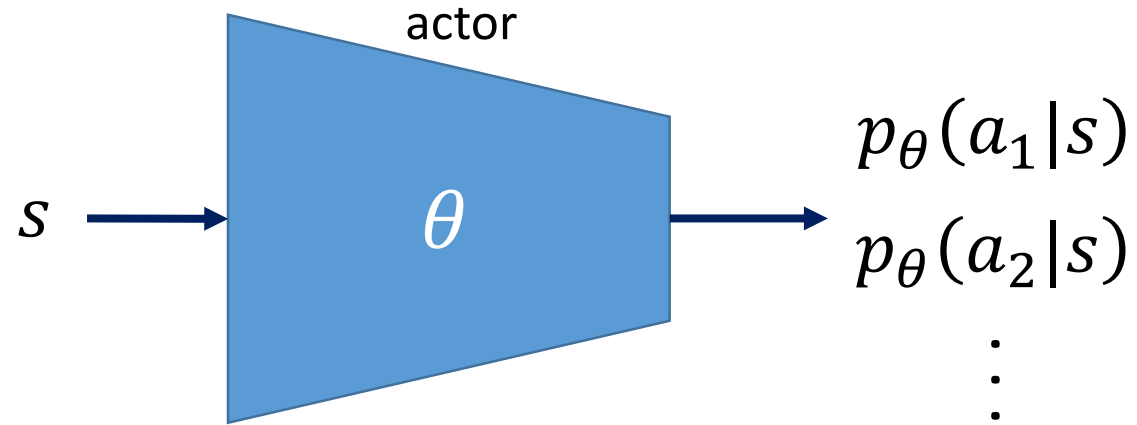
Max  $E[\bar{R}_{\theta}]$

$$\max_{\theta} E[\bar{R}_{\theta}]$$

Gradient of the  
expected value

$$\begin{aligned} \nabla \bar{R}_{\theta} &= \sum R(\tau) \nabla p_{\theta}(\tau) = E_{\tau \sim p_{\theta}(\tau)}[R(\tau) \nabla \log p_{\theta}(\tau)] \approx \frac{1}{N} \sum_{n=1}^N R(\tau^n) \nabla \log p_{\theta}(\tau^n) \\ &= \frac{1}{N} \sum_{n=1}^N \sum_{t=1}^{T_n} R(\tau^n) \nabla \log p_{\theta}(a_t^n | s_t^n) \end{aligned}$$

Use  $\nabla \bar{R}_\theta$  to update policy network



$$\theta^{\pi'} \leftarrow \theta^\pi + \eta \nabla \bar{R}_\theta$$

$$\nabla \bar{R}_\theta = \frac{1}{N} \sum_{n=1}^N \sum_{t=1}^{T_n} R(\tau^n) \nabla \log p_\theta(a_t^n | s_t^n)$$



# Tips to improve bias and reduce variance of $\nabla \bar{R}_\theta$

$$\nabla \bar{R}_\theta = \frac{1}{N} \sum_{n=1}^N \sum_{t=1}^{T_n} R(\tau^n) \nabla \log p_\theta(a_t^n | s_t^n)$$

Add a baseline to  
calculate the reward

$$\nabla \bar{R}_\theta \approx \frac{1}{N} \sum_{n=1}^N \sum_{t=1}^{T_n} (R(\tau^n) - b) \nabla \log p_\theta(a_t^n | s_t^n), \quad b \approx E[R(\tau)]$$

$$\nabla \bar{R}_\theta \approx \frac{1}{N} \sum_{n=1}^N \sum_{t=1}^{T_n} \left( \sum_{t'}^{T_n} r_{t'}^n - b \right) \nabla \log p_\theta(a_t^n | s_t^n)$$

Assign suitable time  
delayed credit

$$\nabla \bar{R}_\theta \approx \frac{1}{N} \sum_{n=1}^N \sum_{t=1}^{T_n} \left( \sum_{t'}^{T_n} \gamma^{t'-t} r_{t'}^n - b \right) \nabla \log p_\theta(a_t^n | s_t^n), \gamma < 1$$

$$A^\theta(s_t, a_t) = \left( \sum_{t'}^{T_n} \gamma^{t'-t} r_{t'}^n - b \right)$$

# Off-policy to improve efficiency of calculating $\nabla \bar{R}_\theta$

On-policy

$$\nabla \bar{R}_\theta \approx \frac{1}{N} \sum_{n=1}^N \sum_{t=1}^{T_n} A^\theta(s_t, a_t) \nabla \log p_\theta(a_t^n | s_t^n), \gamma < 1 \quad A^\theta(s_t, a_t) = \left( \sum_{t'}^{T_n} \gamma^{t'-t} r_{t'}^n - b \right)$$

Importance sampling

$$\begin{aligned} E_{x \sim p}[f(x)] &= E_{x \sim q} \left[ f(x) \frac{p(x)}{q(x)} \right] \\ \text{Var}_{x \sim q} \left[ f(x) \frac{p(x)}{q(x)} \right] &= E_{x \sim q} \left[ \left( f(x) \frac{p(x)}{q(x)} \right)^2 \right] - \left( E_{x \sim q} \left[ f(x) \frac{p(x)}{q(x)} \right] \right)^2 \\ &= E_{x \sim p} \left[ f(x)^2 \frac{p(x)}{q(x)} \right] - (E_{x \sim p}[f(x)])^2 \end{aligned}$$

Off-policy

$$\nabla \bar{R}_\theta = E_{(s_t, a_t) \sim \pi_{\theta'}} \left[ \frac{p_\theta(a_t | s_t)}{p_{\theta'}(a_t | s_t)} A^{\theta'}(s_t, a_t) \nabla \log p_\theta(a_t^n | s_t^n) \right]$$

# From $\nabla \bar{R}_\theta$ to loss function

Off-policy

$$\nabla \bar{R}_\theta = E_{(s_t, a_t) \sim \pi_{\theta'}} \left[ \frac{p_\theta(a_t | s_t)}{p_{\theta'}(a_t | s_t)} A^{\theta'}(s_t, a_t) \nabla \log p_\theta(a_t^n | s_t^n) \right]$$

Sampling efficiency

Loss function

$$J^{\theta'}(\theta) = E_{(s_t, a_t) \sim \pi_{\theta'}} \left[ \frac{p_\theta(a_t | s_t)}{p_{\theta'}(a_t | s_t)} A^{\theta'}(s_t, a_t) \right]$$

Proximal policy  
optimization (PPO)

$$J_{PPO}^{\theta'}(\theta) = J^{\theta'}(\theta) - \beta KL(\theta, \theta')$$

$$J_{PPO2}^{\theta'}(\theta) = \sum_{(s_t, a_t)} \min \left( \frac{p_\theta(a_t | s_t)}{p_{\theta'}(a_t | s_t)} A^{\theta'}(s_t, a_t), \text{clip} \left( \frac{p_\theta(a_t | s_t)}{p_{\theta'}(a_t | s_t)}, 1 - \varepsilon, 1 + \varepsilon \right) A^{\theta'}(s_t, a_t) \right)$$

# Actor-critic strategy to calculate $\nabla \bar{R}_\theta$

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

$$G_t^n = \sum_{t'}^{T_n} \gamma^{t'-t} r_{t'}^n \quad \text{unstable when sampling amount is not large enough}$$

Use expected value to reduce sampling variance

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

$V^{\pi_\theta}(s_t^n)$  Expected value of  $b$

$E[G_t^n] = Q^{\pi_\theta}(s_t^n, a_t^n)$  Expected value of  $G_t^n$

Use one neural network that estimates  $V$

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

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

$$A^\theta(s_t, a_t) = (r_t^n + V^{\pi_\theta}(s_{t+1}^n) - V^{\pi_\theta}(s_t^n))$$

# Use temporal difference to calculate $V$

$$A^\theta(s_t, a_t) = (r_t^n + V^{\pi_\theta}(s_{t+1}^n) - V^{\pi_\theta}(s_t^n))$$

Monte-Carlo approach

$$V^{\pi_\theta}(s_a) \leftrightarrow G_a$$

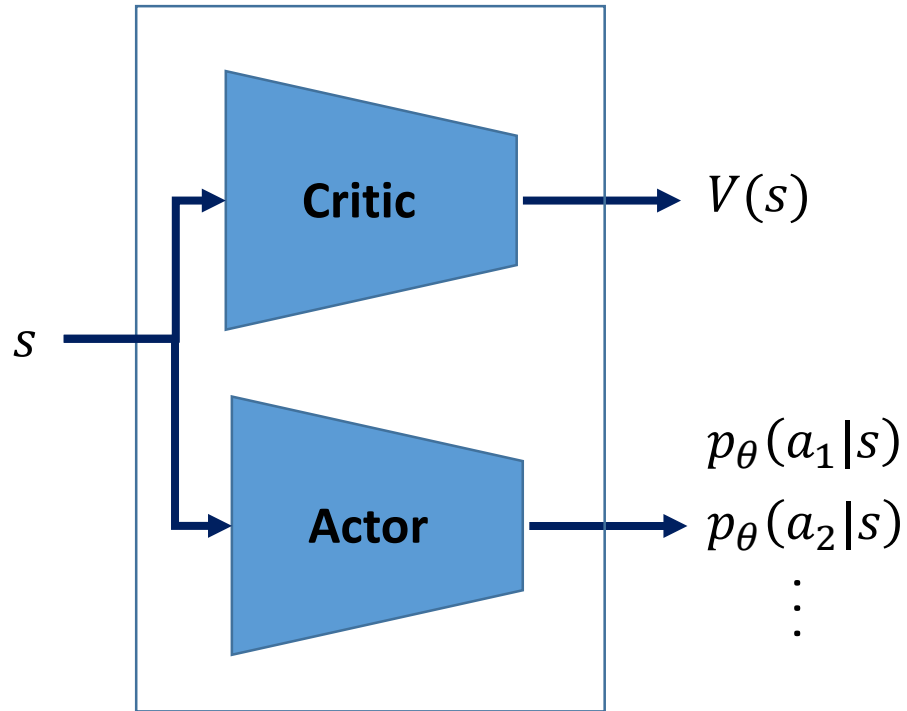
Until the end of the episode, the cumulated reward is  $G_a$

Temporal-difference  
approach

$$V^{\pi_\theta}(s_t) + r_t = V^{\pi_\theta}(s_{t+1})$$

$$V^{\pi_\theta}(s_t) - V^{\pi_\theta}(s_{t+1}) \leftrightarrow r_t$$

# Train the network



TD Error

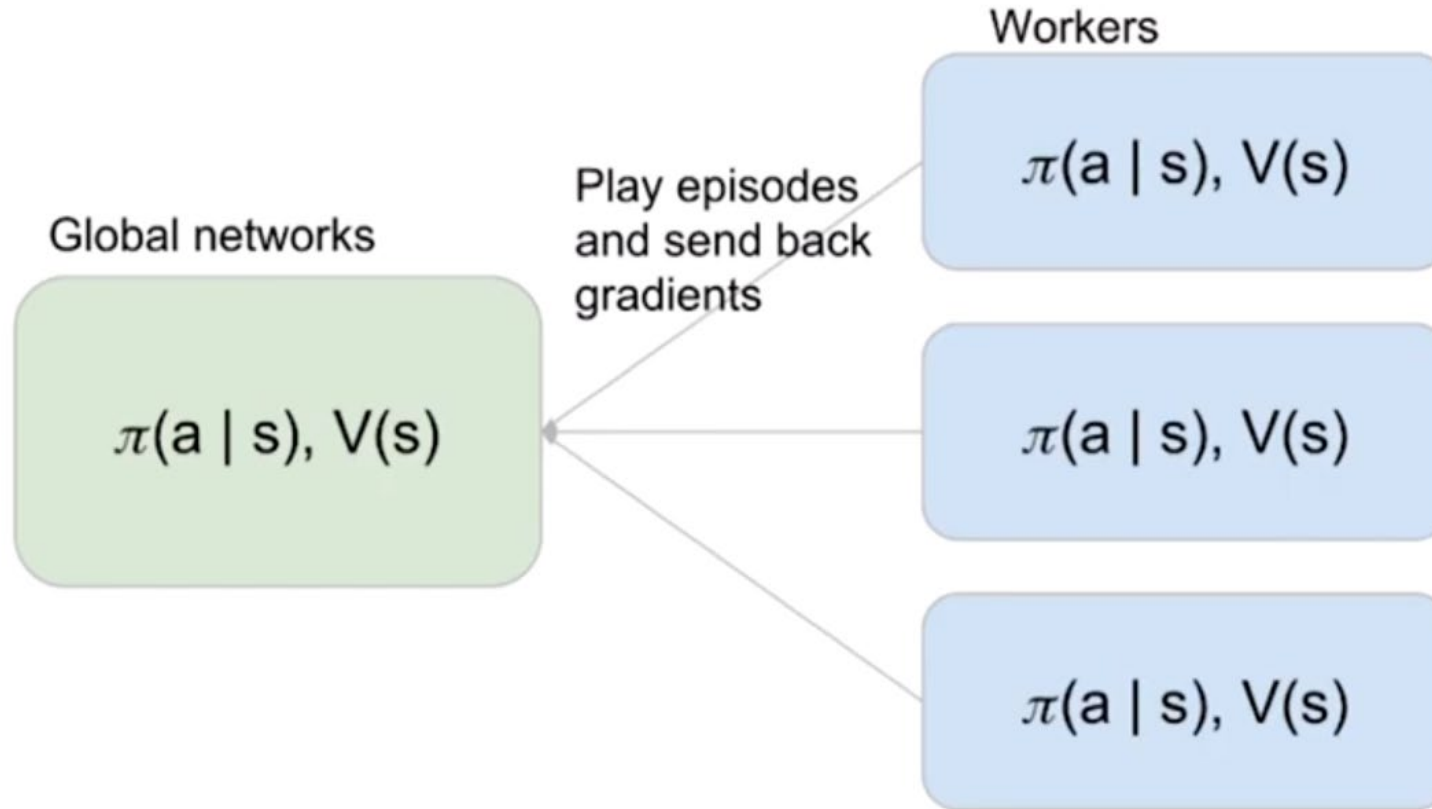
$$L = L_{\pi} + c_v L_v + c_{reg} L_{reg}$$

$$A^{\theta}(s_t, a_t) = G_t^n - V^{\pi_{\theta}}(s_t^n) = Q^{\pi_{\theta}}(s_t^n, a_t^n) - V^{\pi_{\theta}}(s_t^n) = r_t^n + \gamma V^{\pi_{\theta}}(s_{t+1}^n) - V^{\pi_{\theta}}(s_t^n)$$

$$L_v = (G_t^n - V^{\pi_{\theta}}(s_t^n))^2 = (r_t^n + \gamma V^{\pi_{\theta}}(s_{t+1}^n) - V^{\pi_{\theta}}(s_t^n))^2$$

$$L_{\pi} = \sum_{(s_t, a_t)} \min \left( \frac{p_{\theta}(a_t|s_t)}{p_{\theta'}(a_t|s_t)} A^{\theta'}(s_t, a_t), \text{clip} \left( \frac{p_{\theta}(a_t|s_t)}{p_{\theta'}(a_t|s_t)}, 1 - \varepsilon, 1 + \varepsilon \right) A^{\theta'}(s_t, a_t) \right)$$

# A3C



Reference: <https://youtu.be/iCV3vOl8IMk>

# A3C

## Stability

- Each episode will progress randomly
- Each action is sampled probabilistically
- Occasionally, performance of agent can drop off due to bad update
  - Well, this can still happen with A3C so don't think you are immune




Reference: <https://youtu.be/iCV3vOl8IMk>



# A3C

- DQN is also interested in stabilizing learning
- Techniques:
  - Freezing target network
  - Experience replay buffer
- Use experience replay to look at multiple examples per training step
- A3C simply achieves stability using a different method (parallel agents)
- Both solve the problem: how to make neural networks work as function approximators in classic RL algorithms?

# A3C

- Remember: the theory part is not new, just need to create multiple parallel agents and asynchronously update/copy parameters
  - 3 files:
    - main.py (master file; global policy and value networks)
      - Create and coordinate workers
    - worker.py (contains local policy and value networks)
      - Copy weights from global nets
      - Play episodes
      - Send gradients back to master
    - nets.py
      - Definition of policy and value networks
- 

## main.py

Instantiate global policy and value networks

Check # CPUs available, create threads and workers

Initialize global thread-safe counter, so every worker knows when to quit (when # of total steps reaches a max.)

# worker.py

```
def run():  
    in a loop:  
        copy params from global nets to local nets  
        run N steps of game (and store the data - s, a, r, s')  
        using gradients wrt local net, update the global net
```

Conceptually, it's like:

$$1) \quad g_{local} = \frac{\partial L(\theta_{local})}{\partial \theta_{local}}$$

$$2) \quad \theta_{global} = \theta_{global} - \eta g_{local}$$

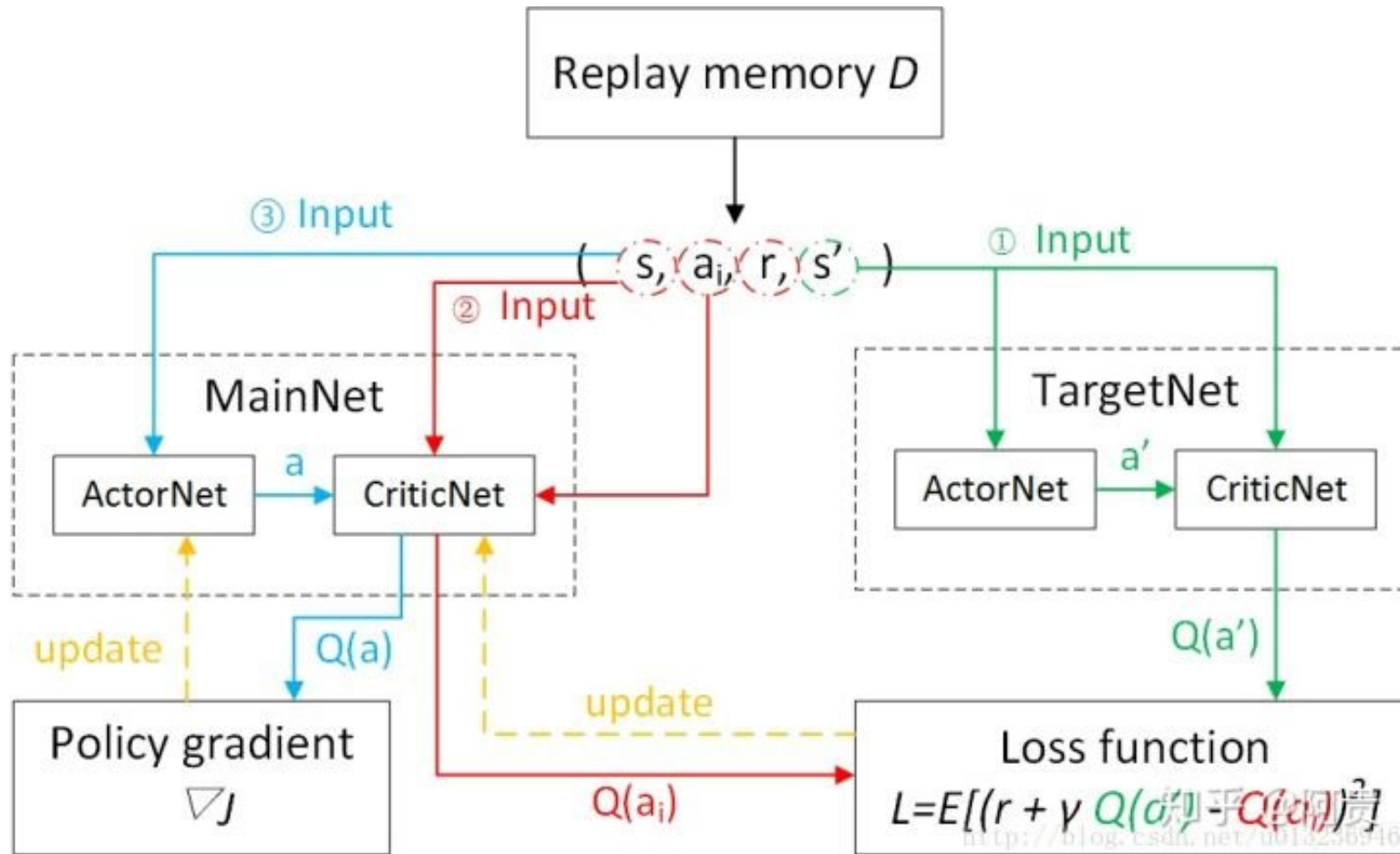
But in reality, we'll use RMSprop

Reference: <https://youtu.be/iCV3vOI8IMk>

# Multiprocessing in Python

- `mp.Queue`: a thread-safe FIFO queue for transporting training data
- `mp.Process` runs a piece of code in a child process
- PyTorch includes its own multiprocessing wrapper, same API

# Deep deterministic policy gradient (DDPG)



圖片來源: <https://zhuanlan.zhihu.com/p/47873624>