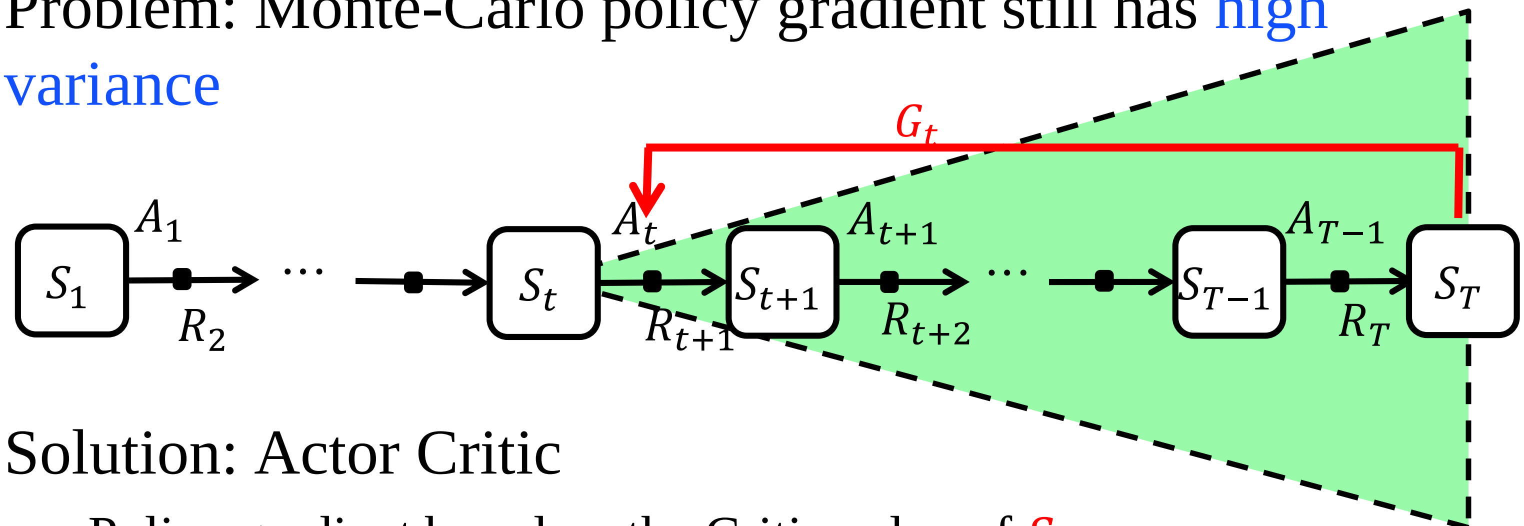
Actor-Critic

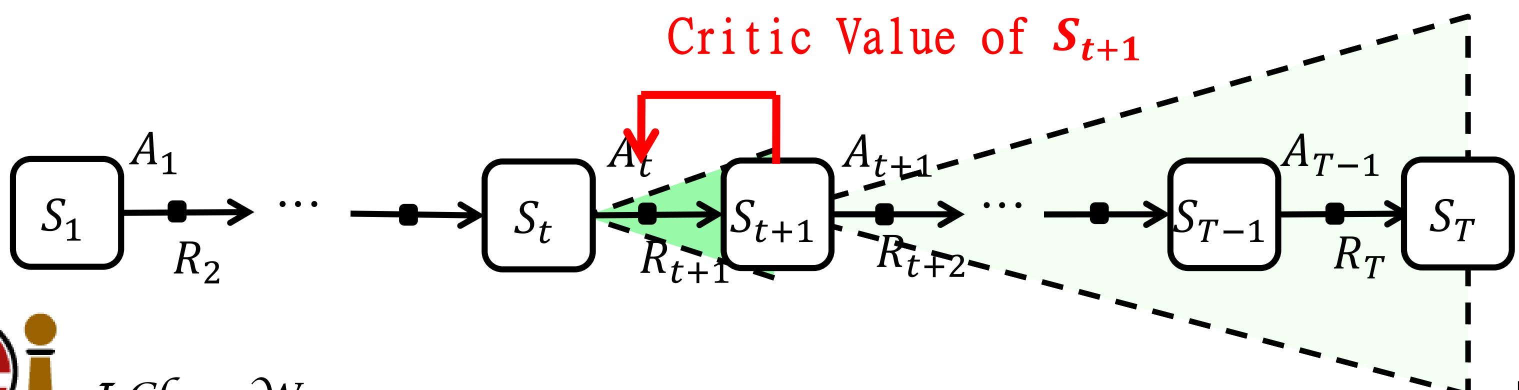
- Actor-Critic (Discrete actions)
- A3C (Asynchronous Advantage Actor-Critic)

Problem of REINFORCE

- Problem: Monte-Carlo policy gradient still has high variance



- Solution: Actor Critic
  - Policy gradient based on the Critic value of S_{t+1}



Reducing Variance Using a Critic

- We use a critic to estimate the action-value function,

$$Q_w(s_t, a_t) \approx Q^{\pi_\theta}(s, a)$$

- Actor-critic algorithms maintain two sets of parameters
  – Critic: Updates action-value function parameters w
  – Actor: Updates policy parameters θ, in direction suggested by critic
- Actor-critic algorithms follow an approximate policy gradient

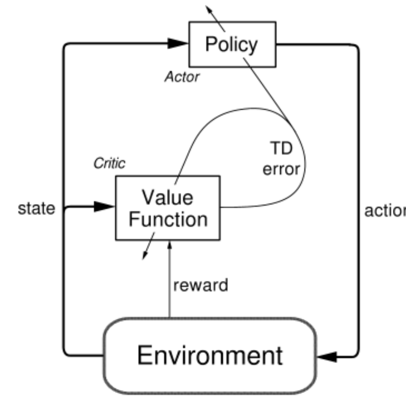
$$\nabla_\theta\ J(\theta) \approx \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s, a) \cdot Q_w(s, a)]$$

$$\Delta\theta = \alpha\nabla_\theta \log \pi_\theta(s, a) \cdot Q_w(s, a)$$

Estimating the Action-Value Function

- The critic is solving a familiar problem: policy evaluation
- But, how good is policy π_θ for current parameters θ?
- This problem was explored in previous two chapters, e.g.
  - Monte-Carlo policy evaluation
  - Temporal-Difference learning
  - TD(λ)
- Could also use e.g. least-squares policy evaluation

Actor-Critic (Discrete Action Space)



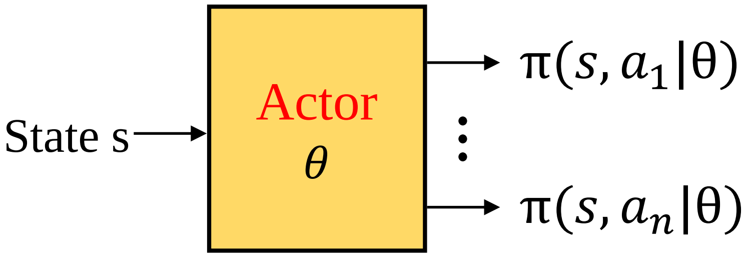
- Use two networks: an actor and a critic
  - Critic estimates the action-value function
    - Gradient:

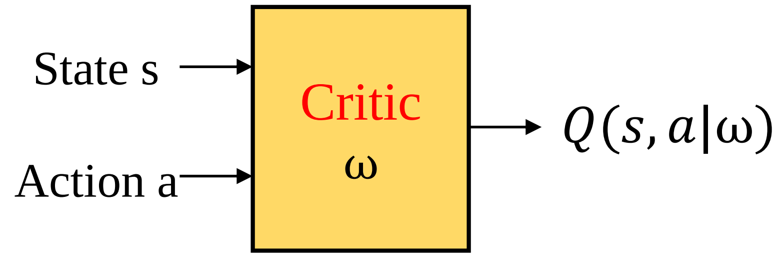
$$\nabla_\omega L_Q(s_t, a_t|\omega) = ((r_{t+1} + \gamma Q(s_{t+1}, a'|\omega)) - Q(s_t, a_t|\omega))\nabla_\omega Q(s_t, a_t|\omega)$$

  - Actor updates policy in direction suggested by critic
    - Gradient (approximate policy gradient):

$$J(\theta) = \mathrm{E}_{s,a}^{\pi_\theta}[Q(s, a|\omega)]$$

$$\nabla_\theta J(\theta) = \mathrm{E}_{s,a}^{\pi_\theta}[\boldsymbol{\nabla_\theta \log \pi(s_t, a_t|\theta)}\, Q(s_t, a_t|\omega)]$$





Actor-Critic (Discrete Action Space)

- Using linear value function approx. $Q_w(s,a) = \varphi(s,a)^T w$
  - Critic: Updates w by linear TD(0)
  - Actor: Updates θ by policy gradient

function QAC
  Initialise s, θ
  Sample $a \sim \pi_\theta$
  for each step **do**
    Sample reward $r = \mathcal{R}_s^a$; sample transition $s' \sim \mathcal{P}_{s,\cdot}^a$
    Sample action $a' \sim \pi_\theta(s', a')$
    $\delta = r + \gamma Q_w(s', a') - Q_w(s, a)$
    $\theta = \theta + \alpha \nabla_\theta \log \pi_\theta(s, a) Q_w(s, a)$
    $w \leftarrow w + \beta \delta \phi(s, a)$
    $a \leftarrow a', s \leftarrow s'$
  end for
end function

Advantage Actor-Critic

Reducing Variance Using a Baseline

- Recall: $\nabla_\theta J(\theta) = \mathrm{E}_{s,a}^{\pi_\theta}[\boldsymbol{\nabla_\theta \log \pi(s_t, a_t|\theta)}\, Q(s_t, a_t|\omega)]$
- Problem: Can we further reduce variance?
- Solution:
  - This can reduce variance, without changing expectation

$$\mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)B(s)] = \sum_{s\epsilon S} d^{\pi\theta}(s) \sum_{a} \nabla_\theta\, \pi_\theta(s,a)B(s)$$
$$= \textstyle\sum_{s\epsilon S} d^{\pi_\theta} B(s) \nabla_\theta \left(\underbrace{\boxed{\textstyle\sum_{a\epsilon A} \pi_\theta(s,a)}}_{=1\text{ (constant)}}\right)$$
$$= 0$$

  - Subtract a baseline function $B(s)$ from the policy gradient
    - A good baseline is the state value function $B(s) = V^{\pi_\theta}(s)$
- So we can rewrite the policy gradient using the advantage function $A^{\pi_\theta}(s,a)$

$$A^{\pi_\theta}(s) = Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s)$$
$$\nabla_\theta\ J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) A^{\pi_\theta}(s,a)]$$

Estimating the Advantage Function (1)

- The advantage function can significantly reduce variance of policy gradient
- So the critic should really estimate the advantage function
  - For example, by estimating both $V^{\pi_\theta}(s)$ and $Q^{\pi_\theta}(s,a)$
  - Using two function approximators and two parameter vectors,

$$V_v(s) \approx V^{\pi_\theta}(s)$$
$$Q_w(s,a) \approx Q^{\pi_\theta}(s,a)$$
$$A(s,a) = Q_w(s,a) - V_v(s)$$

- And updating both value functions by e.g. TD learning

Estimating the Advantage Function (2)

- For the true value function $V^{\pi_\theta}(s)$, the TD error δ^{π_θ}

$$\delta^{\pi_\theta} = r + \gamma V^{\pi_\theta}(s') - V^{\pi_\theta}(s)$$

  - is an unbiased estimate of the advantage function

$$\begin{aligned}\mathbb{E}_{\pi_\theta}[\delta^{\pi_\theta}|\mathrm{s},\mathrm{a}] &= \mathbb{E}_{\pi_\theta}[r + \gamma V^{\pi_\theta}(s')|s,a] - V^{\pi_\theta}(s) \\ &= Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s) \\ &= A^{\pi_\theta}(s,a)\end{aligned}$$

- So we can use the TD error to compute the policy gradient

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

- In practice we can use an approximate TD error

$$\delta_v = r + \gamma V_v(s') - V_v(s)$$

- This approach only requires one set of critic parameters v

Critics at Different Time-Scales

- Critic can estimate value function $V_\theta(s)$ from many targets at different time-scales From last lecture...
  - For MC, the target is the return v_t

$$\Delta\theta = \alpha(v_t - V_\theta(s))\phi(s)$$

  - For TD(0), the target is the TD target $\mathrm{r} + \gamma V(s')$

$$\Delta\theta = \alpha(\mathrm{r} + \gamma V(s') - V_\theta(s))\phi(s)$$

  - For forward-view TD(λ), the target is the λ-return v_t^λ

$$\Delta\theta = \alpha(v_t^\lambda - V_\theta(s))\phi(s)$$

  - For backward-view TD(λ), we use eligibility traces

$$\delta_v = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

$$e_t = \gamma\lambda\, e_{t-1} + \phi(s_t)$$

$$\Delta\theta = \alpha\delta_t e_t$$

Actors at Different Time-Scales

- The policy gradient can also be estimated at many time-scales

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

- Monte-Carlo policy gradient uses error from complete return

$$\Delta\theta = \alpha(v_t - V_v(s_t))\nabla_\theta \log \pi_\theta(s_t, a_t)$$

- Actor-critic policy gradient uses the one-step TD error

$$\Delta\theta = \alpha(r + \gamma V_v(s_{t+1}) - V_v(s_t))\nabla_\theta \log \pi_\theta(s_t, a_t)$$

- Advantage Actor-critic (A2C or A3C) policy gradient uses the (k+1)-step TD error

$$\Delta\theta = \alpha(v_t^{(k)} - V_v(s_t))\nabla_\theta \log \pi_\theta(s_t, a_t)$$

- Some policy gradient algorithms (like PPO) uses TD(λ) error

$$\Delta\theta = \alpha(v_t^{\lambda} - V_v(s_t))\nabla_\theta \log \pi_\theta(s_t, a_t)$$

Actors at Different Time-Scales

- The policy gradient can also be estimated at many time-scales

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

- Monte-Carlo policy gradient uses error from complete return

$$\Delta\theta = \alpha(v_t - V_v(s_t))\nabla_\theta \log \pi_\theta(s_t, a_t)$$

- Actor-critic policy gradient uses the one-step TD error

$$\Delta\theta = \alpha(\delta_t)\nabla_\theta \log \pi_\theta(s_t, a_t)$$

- Advantage Actor-critic (A2C or A3C) policy gradient uses the (k+1)-step TD error $=A^{(k+1)}$

$$\Delta\theta = \alpha(\delta_t + \gamma\delta_{t+1} + \cdots + \gamma^k \delta_{t+k})\nabla_\theta \log \pi_\theta(s_t, a_t)$$

- Some policy gradient algorithms (like PPO) uses TD(λ) error

$$\Delta\theta = \alpha(\delta_t + \lambda\gamma\delta_{t+1} + \cdots + (\lambda\gamma)^k \delta_{t+k} + \cdots)\nabla_\theta \log \pi_\theta(s_t, a_t)$$

$=A_t^{GAE}$: Also called GAE (Generalized Advantage Estimator)

Summary of Policy Gradient Algorithms

- The policy gradient has many equivalent forms

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\ v_t] \qquad \text{REINFORCE}$$

$$= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\ Q^w(s,a)] \qquad \text{Q Actor-Critic}$$

$$= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\ \delta] \qquad \text{TD Actor-Critic}$$

$$= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ A^{(k)}\right] \qquad \text{Advantage Actor-Critic}$$

$$= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ A^{GAE}\right] \qquad \text{TD}(\lambda)\ \text{Actor-Critic}$$

- Each leads a stochastic gradient ascent algorithm

Appendix for Advantages and TD(λ) Errors

- TD errors
- n-step TD errors
- GAE
- Eligibility Trace

Appendix: TD Errors

- TD errors:

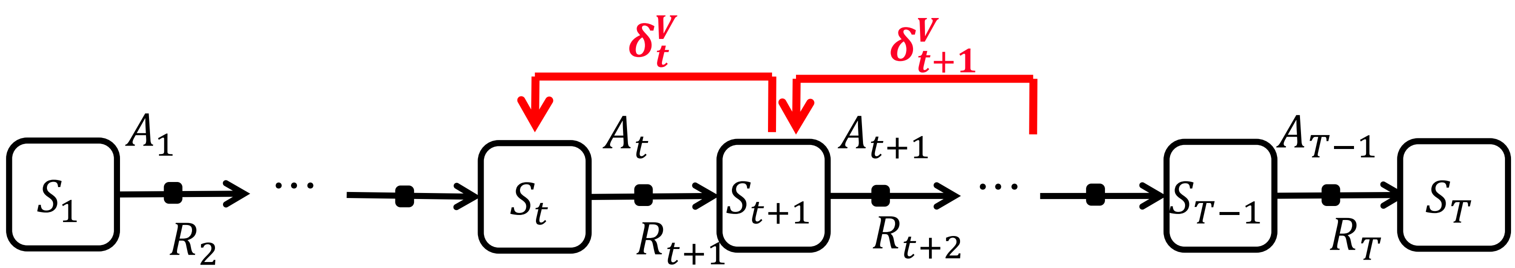
$$\delta_t^V = -V(s_t) + r_t + \gamma V(s_{t+1})$$

$$\delta_{t+1}^V = -V(s_{t+1}) + r_{t+1} + \gamma V(s_{t+2})$$

$$\delta_{t+2}^V = -V(s_{t+2}) + r_{t+2} + \gamma V(s_{t+3})$$

$$\vdots$$

$$\delta_{t+n}^V = -V(s_{t+n}) + r_{t+n} + \gamma V(s_{t+n+1})$$



Appendix: TD Errors

- Weighted TD errors:

$$\delta_t^V = -V(s_t) + r_t + \gamma V(s_{t+1})$$

$$\gamma * \delta_{t+1}^V = \gamma * (-V(s_{t+1}) + r_{t+1} + \gamma V(s_{t+2}))$$

$$\gamma^2 * \delta_{t+2}^V = \gamma^2 * (-V(s_{t+2}) + r_{t+2} + \gamma V(s_{t+3}))$$

$$\vdots$$

$$\gamma^n * \delta_{t+n}^V = \gamma^n * (-V(s_{t+n}) + r_{t+n} + \gamma V(s_{t+n+1}))$$

Appendix: n-Step TD Errors

- Sum them up, becoming n-step TD errors.

$$\delta_t^V = -V(s_t) + r_t + \cancel{\gamma V(s_{t+1})}$$

$$\gamma * \delta_{t+1}^V = \gamma * (\cancel{-V(s_{t+1})} + r_{t+1} + \cancel{\gamma V(s_{t+2})})$$

$$\gamma^2 * \delta_{t+2}^V = \gamma^2 * (\cancel{-V(s_{t+2})} + r_{t+2} + \cancel{\gamma V(s_{t+3})})$$

$$\vdots$$

$$+ \quad \gamma^n * \delta_{t+n}^V = \gamma^n * (\cancel{-V(s_{t+n})} + r_{t+n} + \gamma V(s_{t+n+1}))$$

$$\delta_t^V + \gamma\delta_{t+1}^V + \gamma^2\delta_{t+2}^V + \cdots \gamma^n\delta_{t+n}^V$$
$$= -V(s_t) + r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{n+1} V(s_{t+n+1})$$
$$= -V(s_t) + G_t^{(n+1)}$$
$$= \hat{A}_t^{(n+1)}$$

If $n \rightarrow \infty$, it becomes MC learning. Why?

Appendix: n-Step TD Errors

- n-step TD errors:

$$\hat{A}_t^{(1)} := \delta_t^V$$

$$\hat{A}_t^{(2)} := (\delta_t^V + \gamma \delta_{t+1}^V)$$

$$\hat{A}_t^{(3)} := (\delta_t^V + \gamma \delta_{t+1}^V + \gamma^2 \delta_{t+2}^V)$$

$$\vdots$$

$$\hat{A}_t^{(n)} := \sum_{k=1}^{n} \gamma^{k-1} * \delta_{t+k-1}^V$$

Appendix: n-Step TD Errors and GAE

- Weighted n-step TD errors:
  - The same trick as TD(λ)
- Then, sum them up.

$$(1-\lambda) * \hat{A}_t^{(1)} := (1-\lambda) \quad * \delta_t^V$$

$$(1-\lambda)\lambda * \hat{A}_t^{(2)} := (1-\lambda)\lambda \quad * (\delta_t^V + \gamma\delta_{t+1}^V)$$

$$(1-\lambda)\lambda^2 * \hat{A}_t^{(3)} := (1-\lambda)\lambda^2 \quad * (\delta_t^V + \gamma\delta_{t+1}^V + \gamma^2\delta_{t+2}^V)$$

$$\vdots$$

$$+ \quad (1-\lambda)\lambda^{n-1} * \hat{A}_t^{(n)} := (1-\lambda)\sum_{k=1}^{n}\gamma^{k-1} * \delta_{t+k-1}^V$$

$$\boxed{\hat{A}_t^{GAE(\gamma,\lambda)} = (1-\lambda)(\hat{A}_t^{(1)} + \lambda\hat{A}_t^{(2)} + \lambda^2\hat{A}_t^{(3)} + \cdots + \lambda^{n-1}\hat{A}_t^{(n)} + \cdots)}$$

Appendix: n-Step TD Errors and GAE

- The sum of exponentially-weighted TD residuals denoted as $\hat{A}_t^{GAE(\gamma,\lambda)}$ (actually equals to $G_t^\lambda - V(S_t)$ for $TD(\lambda)$)

$$\hat{A}_t^{GAE(\gamma,\lambda)} = (1-\lambda)\left(\hat{A}_t^{(1)} + \lambda\hat{A}_t^{(2)} + \lambda^2\hat{A}_t^{(3)} + \cdots + \lambda^{n-1}\hat{A}_t^{(n)} + \cdots\right)$$

$$= (1-\lambda)\left((\delta_t^V) + \lambda(\delta_t^V + \gamma\delta_{t+1}^V) + \lambda^2(\delta_t^V + \gamma\delta_{t+1}^V + \gamma\delta_{t+2}^V) + \cdots\right)$$

$$= (1-\lambda)\begin{pmatrix} \delta_t^V(1+\lambda+\lambda^2+\cdots) + \\ \gamma\lambda\delta_{t+1}^V(1+\lambda+\lambda^2+\cdots) + \\ (\gamma\lambda)^2\delta_{t+2}^V(1+\lambda+\lambda^2+\cdots) + \\ \cdots \end{pmatrix}$$

$$= (1-\lambda)\left(\delta_t^V\left(\frac{1}{1-\lambda}\right) + \gamma\lambda\delta_{t+1}^V\left(\frac{1}{1-\lambda}\right) + (\gamma\lambda)^2\delta_{t+2}^V\left(\frac{1}{1-\lambda}\right) + \cdots\right)$$

$$= \sum_{n=0}^{\infty}(\gamma\lambda)^n\delta_{t+n}^V = \delta_t^V + \lambda\gamma\delta_{t+1}^V + \cdots + (\lambda\gamma)^k\delta_{t+k}^V + \cdots$$

Appendix: Recall $TD(\lambda)$

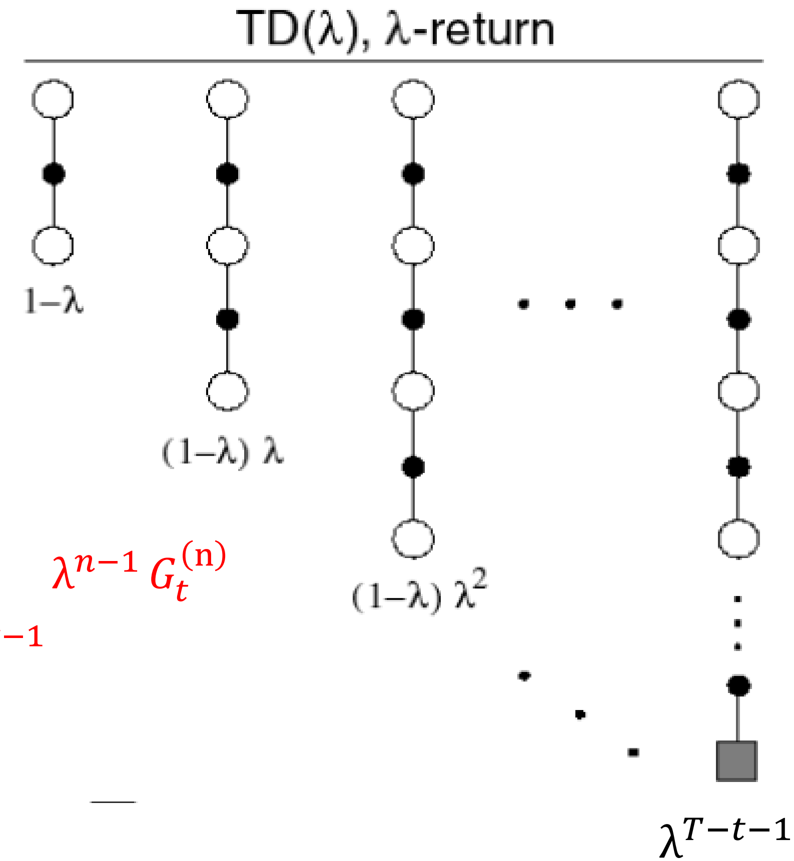
- λ-return G_t^{λ}:
  - combines all n-step returns $G_t^{(n)}$
- Using weight $(1-\lambda)\,\lambda^{n-1}$

$$G_t^{\lambda} = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1}\, G_t^{(\mathrm{n})}$$

or (in the case of termination)

$$G_t^{\lambda} = (1-\lambda)\sum_{n=1}^{T-t-1} \lambda^{n-1}\, G_t^{(\mathrm{n})} + (1-\lambda)\sum_{n=T-t-1}^{\infty} \lambda^{n-1}\, G_t^{(\mathrm{n})}$$

$$G_t^{\lambda} = (1-\lambda)\sum_{n=1}^{T-t-1} \lambda^{n-1}\, G_t^{(\mathrm{n})} + \lambda^{T-t-1} G_t$$



- Forward-view TD(λ)

$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t^{\lambda} - V(S_t)\right)$$

Appendix: GAE and Eligibility Trace

- Eligibility trace:

$$E_0(s) = 0$$

$$E_t(s) = (\gamma\,\lambda)\; E_{t-1}(s) + 1(S_t = s)$$

$$\hat{A}_t^{GAE(\gamma,\lambda)} = 1\,\delta_t^V + \lambda\gamma\delta_{t+1}^V + (\lambda\gamma)^2\delta_{t+2}^V + \cdots + (\lambda\gamma)^k\delta_{t+k}^V + \cdots$$

$$\hat{A}_{t+1}^{GAE(\gamma,\lambda)} = 1\,\delta_{t+1}^V + \lambda\gamma\delta_{t+2}^V + \cdots + (\lambda\gamma)^{k-1}\delta_{t+k}^V + \cdots$$

$$\hat{A}_{t+2}^{GAE(\gamma,\lambda)} = 1\,\delta_{t+2}^V + \cdots + (\lambda\gamma)^{k-2}\delta_{t+k}^V + \cdots$$

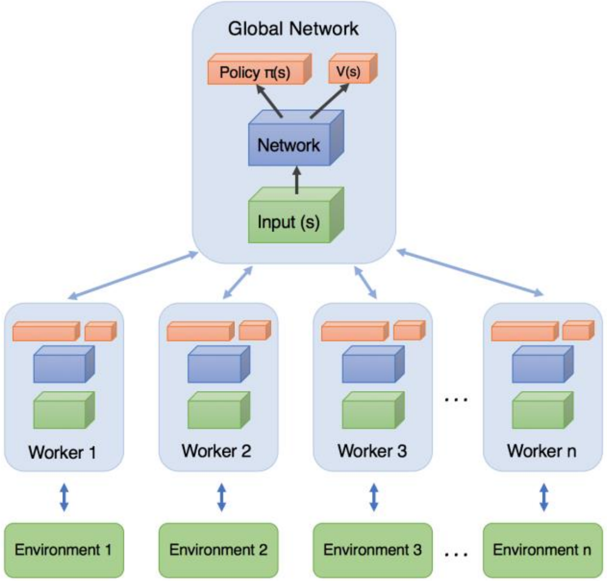
$\dots$

$E_t(s_t)$ $E_t(s_{t+1})$ $E_t(s_{t+2})$

A3C (Asynchronous Advantage Actor-Critic)

Asynchronous Advantage Actor-critic (A3C)

- Asynchronous Lock-Free Reinforcement Learning
  - Use two main ideas to make the algorithm practical:
    - Multiple threads on a single machine
    - Multiple actor-learners applying online updates in parallel (no experience replay)



Asynchronous Advantage Actor-critic (A3C)

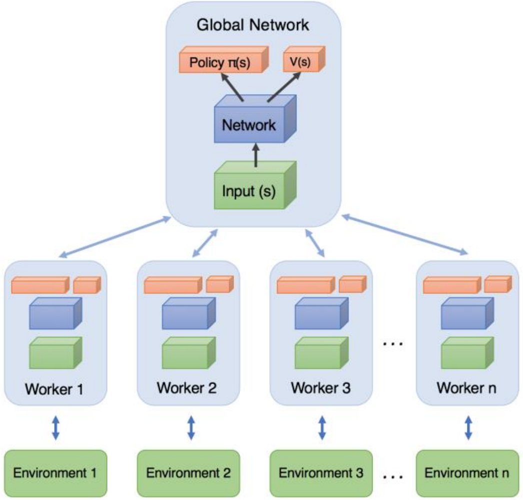
- Instead of experience replay, we asynchronously execute multiple agents in parallel.
  - Decorrelate the agents' data into a more stationary process
  - Enable a much larger spectrum of fundamental on-policy RL algorithms
- For each worker (asynchronous part):

  Copy all parameters from the global network.

  keep playing and computing gradients.

  Every N iterations:

  1. Update all gradients to the global network.
  2. Copy all new parameters from the global network

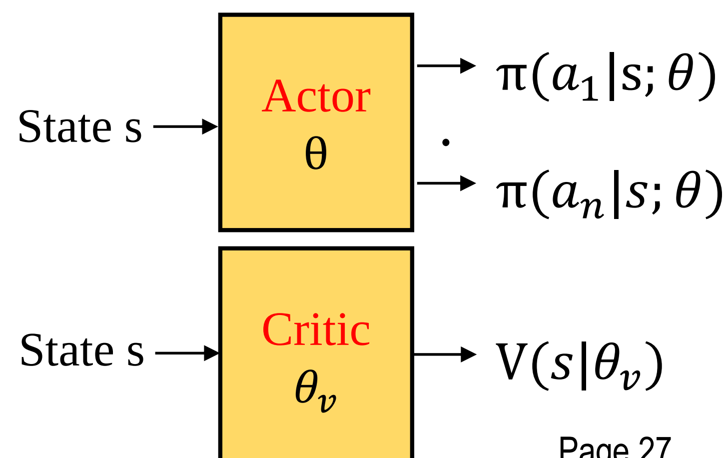


Asynchronous Advantage Actor-critic (A3C)

- Asynchronous advantage actor-critic(A3C) maintains a policy $\pi(a_t|s_t;\theta)$ and an estimate of the value function $V(s_t,\theta_v)$.
- The update performed by the algorithm can be seen as $\nabla_\theta \log \pi(a_t|s_t;\theta) A(s_t,a_t;\theta,\theta_v)$

$$\sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V(s_{t+k};\theta_v) - V(s_t;\theta_v)$$

  - Make k-step operations, and then calculate advantages backwards.
- Intuitively, the network should be

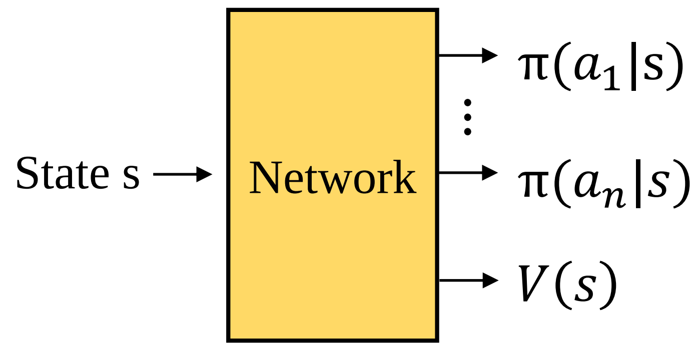


Asynchronous Advantage Actor-critic (A3C)

- Asynchronous advantage actor-critic(A3C) maintains a policy $\pi(a_t|s_t;\theta)$ and an estimate of the value function $V(s_t, \theta_v)$.
- The update performed by the algorithm can be seen as $\nabla_\theta \log \pi(a_t|s_t;\theta)A(s_t, a_t; \theta, \theta_v)$

$$\sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V(s_{t+k};\theta_v) - V(s_t;\theta_v)$$

  - Make k-step operations, and then calculate advantages backwards.
- Typically use a convolutional neural network that has two heads:
  - one softmax output for the policy $\pi(a_t|s_t;\theta)$
  - one output for the value function $V(s_t;\theta_v)$
  - all non-output layers are shared



Asynchronous Advantage Actor-critic (A3C)

repeat

Sync $\theta' = \theta, \theta'_v = \theta_v$

$t_{start} = t$

Get state s_t

repeat (note: $t = t + 1$)

Perform a_t according to policy $\pi(a_t|s_t; \theta')$

Receive s' and reward r

until terminal s_t or $t - t_{start} == t_{max}$

(note: $t = t_{start} + t_{max}$, if not terminal)

$$R = \begin{cases} 0 & \text{for terminal } s' \\ V(s_t, \theta'_v) & \text{for non-terminal } s' \end{cases}$$

for $i \in \{t-1, \cdots, t_{start}\}$ **do**

$R \leftarrow r_i + \gamma R$

Accumulate gradients wrt θ': $d\theta \leftarrow d\theta + \nabla_{\theta'} log\pi(a_i|s_i; \theta')\big(R - V(s_i; \theta'_v)\big)$

Accumulate gradients wrt θ'_v: $d\theta_v \leftarrow d\theta_v + \partial\big(R - V(s_i;\ \theta'_v)\big)^2 / \partial\theta'_v$

end for

Perform asynchronous update of θ using $d\theta$ and of θ_v using $d\theta_v$.

until $T \to T_{max}$

θ, θ_v: global shared parameters

T: global shared counter

θ', θ'_v: thread specific parameters

t: thread step counter

Experiments – A3C

Method	Training Time	Mean	Median
DQN (from [Nair et al., 2015])	8 days on GPU	121.9%	47.5%
Gorila [Nair et al., 2015]	4 days, 100 machines	215.2%	71.3%
Double DQN [Van Hasselt et al., 2015]	8 days on GPU	332.9%	110.9%
Dueling Double DQN [Wang et al., 2015]	8 days on GPU	343.8%	117.1%
Prioritized DQN [Schaul et al., 2015]	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.