Actor-Critic Methods (A2C, A3C)

Alina Vereshchaka

CSE4/510 Reinforcement Learning
Fall 2019

avereshc@buffalo.edu

October 31, 2019

*Slides are adopted from Deep Reinforcement Learning by Sergey Levine & Policy Gradients by David Silver

Table of Contents

Types of RL algorithms

$$\theta^* = \arg\max_\theta R_{\tau \sim p_\theta(\tau)} \left[\sum_t r(s_t, a_t) \right]$$

- Model-based RL:

Types of RL algorithms

$$\theta^* = \arg\max_{\theta} R_{\tau \sim p_\theta(\tau)} \left[\sum_t r(s_t, a_t) \right]$$

- Model-based RL: estimate the transition model and then:
  - Use it for planning (no explicit policy)
  - Use it to improve a policy

Types of RL algorithms

$$\theta^* = \arg\max_{\theta} R_{\tau \sim p_\theta(\tau)} \left[\sum_t r(s_t, a_t) \right]$$

- Model-based RL: estimate the transition model and then:
  - Use it for planning (no explicit policy)
  - Use it to improve a policy
- Value-based:

Types of RL algorithms

$$\theta^* = \arg\max_{\theta} R_{\tau \sim p_\theta(\tau)} \left[\sum_t r(s_t, a_t) \right]$$

- Model-based RL: estimate the transition model and then:
  - Use it for planning (no explicit policy)
  - Use it to improve a policy
- Value-based: estimate value function or Q-function of the current policy (no explicit policy)
- Policy-gradient:

Types of RL algorithms

$$\theta^* = \arg\max_{\theta} R_{\tau \sim p_\theta(\tau)} \left[\sum_t r(s_t, a_t) \right]$$

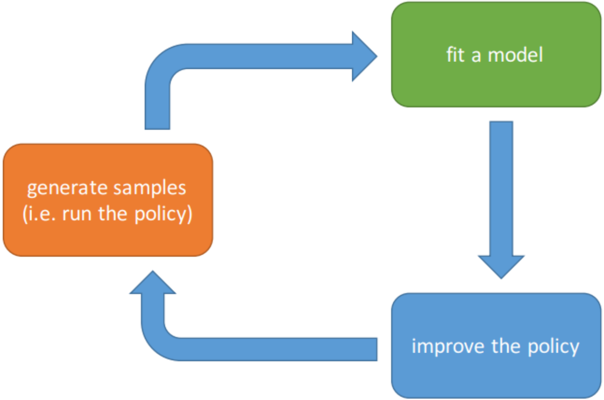
- Model-based RL: estimate the transition model and then:
  - Use it for planning (no explicit policy)
  - Use it to improve a policy
- Value-based: estimate value function or Q-function of the current policy (no explicit policy)
- Policy-gradient: directly differentiate the objective
- Actor-critic:

Types of RL algorithms

$$\theta^* = \arg\max_\theta R_{\tau \sim p_\theta(\tau)} \left[\sum_t r(s_t, a_t) \right]$$

- Model-based RL: estimate the transition model and then:
  - Use it for planning (no explicit policy)
  - Use it to improve a policy
- Value-based: estimate value function or Q-function of the current policy (no explicit policy)
- Policy-gradient: directly differentiate the objective
- Actor-critic: estimate value function or Q-function of the current policy, use it to improve the policy

Model-based Algorithms

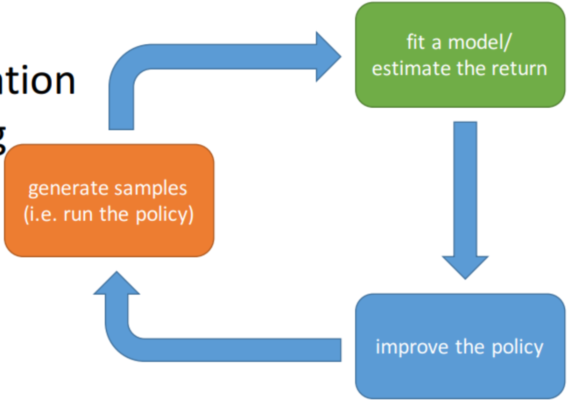


learn $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$

a few options

Examples:

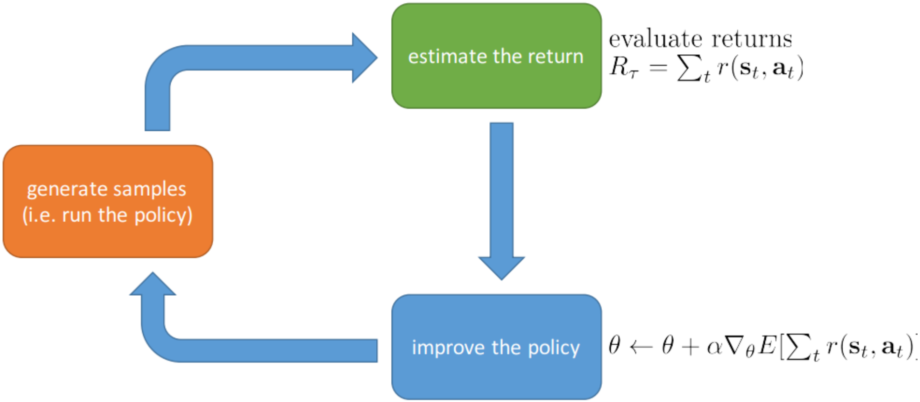
- Value-Iteration
- Q-Learning
- DQN



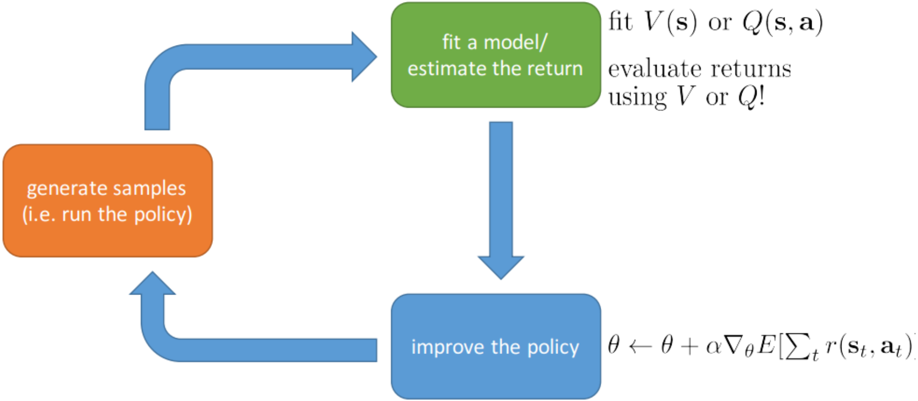
fit $V(\mathbf{s})$ or $Q(\mathbf{s},\mathbf{a})$

set $\pi(\mathbf{s}) = \arg\max_{\mathbf{a}} Q(\mathbf{s},\mathbf{a})$

Direct Policy Gradient



Actor-critic: Value Function + Policy Gradients



Comparison: Sample Efficiency

- **Sample efficiency:** How many samples do we need to get a good policy?

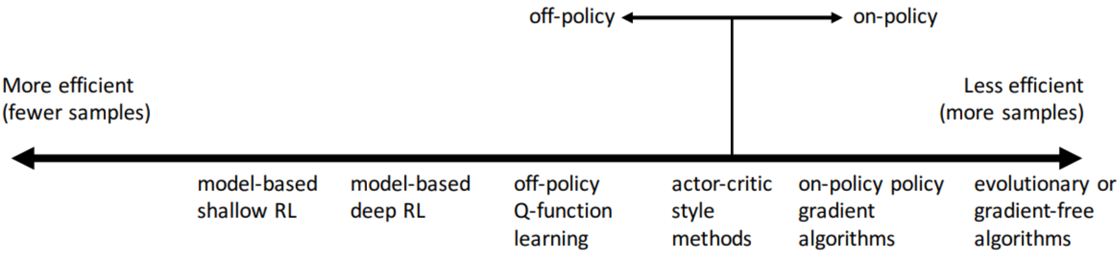
Comparison: Sample Efficiency

- **Sample efficiency:** How many samples do we need to get a good policy?
- Most important questions: Is the algorithm off policy?
  - Off policy: able to improve the policy without generating new samples from that policy

Comparison: Sample Efficiency

- **Sample efficiency:** How many samples do we need to get a good policy?
- Most important questions: Is the algorithm off policy?
  - Off policy: able to improve the policy without generating new samples from that policy
  - On policy: each time the policy is changed, even a little bit, we need to generate new samples

Comparison: Sample Efficiency



REINFORCE (Monte-Carlo Policy Gradient)

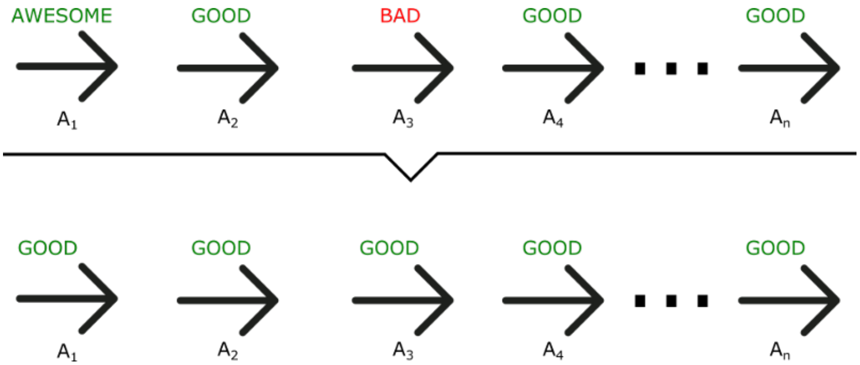
- Update parameters by stochastic gradient ascent
- Using policy gradient theorem
- Using return G_t as an unbiased sample of $Q^{\pi_\theta}(s_t, a_t)$

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

REINFORCE, A Monte-Carlo Policy-Gradient Method (episodic)

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta}), \forall a \in \mathcal{A}, s \in \mathcal{S}, \boldsymbol{\theta} \in \mathbb{R}^n$
Initialize policy weights $\boldsymbol{\theta}$
Repeat forever:
    Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
    For each step of the episode $t = 0, \ldots, T-1$:
        $G_t \leftarrow$ return from step t
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\gamma^t G_t \nabla_{\boldsymbol{\theta}} \log \pi(A_t|S_t, \boldsymbol{\theta})$

REINFORCE: Problem



Solution

Policy Update: $$\Delta\theta = \alpha * \nabla_\theta * (log\, \pi(S_t, A_t, \theta)) * R(t)$$

New update: $$\Delta\theta = \alpha * \nabla_\theta * (log\, \pi(S_t, A_t, \theta)) * \boxed{Q(S_t, A_t)}$$

Table of Contents

Actor-Critic

- Monte-Carlo policy gradient still has high variance
- We can use a critic to estimate the action-value function:

$$Q_w(s, a) \approx Q_{\pi_\theta}(s, a)$$

Actor-Critic

- Monte-Carlo policy gradient still has high variance
- We can use a critic to estimate the action-value function:

$$Q_w(s, a) \approx Q_{\pi_\theta}(s, a)$$

- Actor-critic algorithms maintain *two* sets of parameters
  - Critic Updates action-value function parameters w

Actor-Critic

- Monte-Carlo policy gradient still has high variance
- We can use a critic to estimate the action-value function:

$$Q_w(s, a) \approx Q_{\pi_\theta}(s, a)$$

- 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

- Monte-Carlo policy gradient still has high variance
- We can use a critic to estimate the action-value function:

$$Q_w(s, a) \approx Q_{\pi_\theta}(s, a)$$

- 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

$$\nabla_\theta J(\theta) \approx E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s, a) Q_w(s, a)]$$

Actor-Critic

- Monte-Carlo policy gradient still has high variance
- We can use a critic to estimate the action-value function:

$$Q_w(s, a) \approx Q_{\pi_\theta}(s, a)$$

- 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

$$\nabla_\theta J(\theta) \approx E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s, a) Q_w(s, a)]$$
$$\Delta\theta = \alpha \nabla_\theta \log \pi_\theta(s, a) Q_w(s, a)$$

Actor-Critic

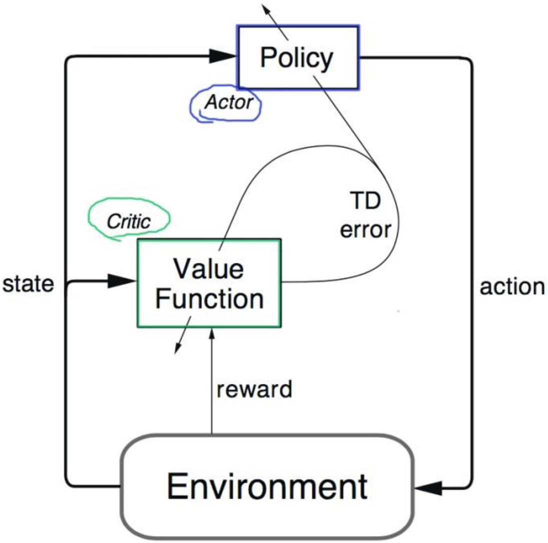
Actor-Critic

- The actor is the policy $\pi_\theta(a|s)$ with parameters θ which conducts actions in an environment.

Actor-Critic

- The actor is the policy $\pi_\theta(a|s)$ with parameters θ which conducts actions in an environment.
- The critic computes value functions to help assist the actor in learning. These are usually the state value, state-action value, or advantage value, denoted as $V(s)$, $Q(s, a)$, and $A(s, a)$, respectively.

Actor-Critic



Actor-Critic

- The critic is solving a familiar problem: policy evaluation
- How good is policy π_θ for current parameters θ?

Actor-Critic

- The critic is solving a familiar problem: policy evaluation
- How good is policy π_θ for current parameters θ?
- To estimate, use any policy evaluation method:
  - Monte-Carlo policy evaluation
  - Temporal-Difference learning
  - Least-squares policy evaluation

Estimating the TD Error

- For the true value function $V_{\pi_\theta}(s)$, the TD error δ_{π_θ}

$$\delta_{\pi_\theta} =$$

Estimating the TD Error

- For the true value function $V_{\pi_\theta}(s)$, the TD error δ_{π_θ}

$$\delta_{\pi_\theta} = r + \gamma V_{\pi_\theta}(s') - V_{\pi_\theta}(s)$$

Estimating the TD Error

- 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

$$\mathbb{E}_{\pi_\theta}[\delta_{\pi_\theta}|s, a] = \mathbb{E}_{\pi_\theta}\left[r + \gamma V_{\pi_\theta}(s')|s, a\right] - V_{\pi_\theta}(s)$$

Estimating the TD Error

- 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

$$\begin{aligned} \mathbb{E}_{\pi_\theta}[\delta_{\pi_\theta}|s, a] &= \mathbb{E}_{\pi_\theta}\left[r + \gamma V_{\pi_\theta}(s')|s, a\right] - V_{\pi_\theta}(s) \\ &= Q_{\pi_\theta}(s, a) - V_{\pi_\theta}(s) \end{aligned}$$

Estimating the TD Error

- 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

$$\begin{aligned}
\mathbb{E}_{\pi_\theta}[\delta_{\pi_\theta}|s,a] &= \mathbb{E}_{\pi_\theta}\left[r + \gamma V_{\pi_\theta}(s')|s,a\right] - V_{\pi_\theta}(s) \\
&= Q_{\pi_\theta}(s,a) - V_{\pi_\theta}(s) \\
&= A_{\pi_\theta}(s,a)
\end{aligned}$$

Estimating the TD Error

- 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

$$\begin{aligned}
\mathbb{E}_{\pi_\theta}[\delta_{\pi_\theta}|s,a] &= \mathbb{E}_{\pi_\theta}\left[r + \gamma V_{\pi_\theta}(s')|s,a\right] - V_{\pi_\theta}(s) \\
&= Q_{\pi_\theta}(s,a) - V_{\pi_\theta}(s) \\
&= A_{\pi_\theta}(s,a)
\end{aligned}$$

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

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

Estimating the TD Error

- 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

$$\begin{aligned}\mathbb{E}_{\pi_\theta}[\delta_{\pi_\theta}|s,a] &= \mathbb{E}_{\pi_\theta}\left[r + \gamma V_{\pi_\theta}(s')|s,a\right] - V_{\pi_\theta}(s) \\ &= Q_{\pi_\theta}(s,a) - V_{\pi_\theta}(s) \\ &= A_{\pi_\theta}(s,a)\end{aligned}$$

- 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, that requires one set of parameters w

$$\delta_w = r + \gamma V_w(s') - V_w(s)$$

Actor-Critic: Critic (Linear TD(0)) + Actor (policy gradient)

One-step Actor–Critic (episodic), for estimating $\pi_{\boldsymbol{\theta}} \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
  Initialize S (first state of episode)
  $I \leftarrow 1$
  Loop while S is not terminal (for each time step):
    $A \sim \pi(\cdot|S, \boldsymbol{\theta})$
    Take action A, observe S', R
    $\delta \leftarrow R + \gamma \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla \ln \pi(A|S, \boldsymbol{\theta})$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$

REINFORCE with Baseline (episodic), for estimating $\pi_{\boldsymbol{\theta}} \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Algorithm parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
  Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ $\qquad (G_t)$
    $\delta \leftarrow G - \hat{v}(S_t,\mathbf{w})$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S_t,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \gamma^t \delta \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$

Advantage Actor Critic (A2C)

- The advantage function can significantly reduce variance of policy gradient

Advantage Actor Critic (A2C)

- 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 $Q_{\pi_\theta}(s, a)$

Advantage Actor Critic (A2C)

- 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,

$$V_v(s) \approx V_{\pi_\theta}(s)$$

Advantage Actor Critic (A2C)

- 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,

$$V_v(s) \approx V_{\pi_\theta}(s)$$
$$Q_w(s, a) \approx Q_{\pi_\theta}(s, a)$$

Advantage Actor Critic (A2C)

- 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,

$$V_v(s) \approx V_{\pi_\theta}(s)$$
$$Q_w(s, a) \approx Q_{\pi_\theta}(s, a)$$
$$A(s, a) = Q_w(s, a) - V_v(s)$$

Advantage Actor Critic (A2C)

- 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

Summary of Policy Gradient Algorithms

- The policy gradient has many equivalent forms

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

Summary of Policy Gradient Algorithms

- The policy gradient has many equivalent forms

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

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

Summary of Policy Gradient Algorithms

- The policy gradient has many equivalent forms

$$\begin{aligned}\nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) G_t] && \text{REINFORCE} \\ &= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) Q_w(s,a)] && \text{Q Actor-Critic} \\ &= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) A_w(s,a)]\end{aligned}$$

Summary of Policy Gradient Algorithms

- The policy gradient has many equivalent forms

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) G_t] \quad \text{REINFORCE}$$
$$= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) Q_w(s,a)] \quad \text{Q Actor-Critic}$$
$$= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) A_w(s,a)] \quad \text{Advantage Actor-Critic (A2C)}$$
$$= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) \delta]$$

Summary of Policy Gradient Algorithms

- The policy gradient has many equivalent forms

$$\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) G_t] && \text{REINFORCE} \\
&= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) Q_w(s,a)] && \text{Q Actor-Critic} \\
&= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) A_w(s,a)] && \text{Advantage Actor-Critic (A2C)} \\
&= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) \delta] && \text{TD Actor-Critic}
\end{aligned}$$

- Each leads a stochastic gradient ascent algorithm
- Critic uses policy evaluation (e.g. MC or TD learning) to estimate $Q_\pi(s,a)$, $A_\pi(s,a)$ or $V_\pi(s)$.

Table of Contents

Asynchronous Advantage Actor Critic (A3C)

A3C stands for Asynchronous Advantage Actor Critic

- **Asynchronous:** the algorithm involves executing a set of environments in parallel to increase the diversity of training data, and with gradient updates performed in a Hogwild! style procedure. No experience replay is needed, though one could add it if desired.

A3C stands for Asynchronous Advantage Actor Critic

- **Asynchronous:** the algorithm involves executing a set of environments in parallel to increase the diversity of training data, and with gradient updates performed in a Hogwild! style procedure. No experience replay is needed, though one could add it if desired.
- **Advantage**: the policy gradient updates are done using the advantage function $A(s, a)$

Asynchronous Advantage Actor Critic (A3C)

A3C stands for Asynchronous Advantage Actor Critic

- **Asynchronous:** the algorithm involves executing a set of environments in parallel to increase the diversity of training data, and with gradient updates performed in a Hogwild! style procedure. No experience replay is needed, though one could add it if desired.
- **Advantage**: the policy gradient updates are done using the advantage function $A(s, a)$
- **Actor:** this is an actor-critic method which involves a policy that updates with the help of learned state-value functions.

Asynchronous Advantage Actor Critic (A3C)

- A3C (Mnih et al. 2016) idea: Sample for data can be parallelized using several copies of the same agent

Asynchronous Advantage Actor Critic (A3C)

- A3C (Mnih et al. 2016) idea: Sample for data can be parallelized using several copies of the same agent
  - use N copies of the agents (workers) working in parallel collecting samples and computing gradients for policy and value function

Asynchronous Advantage Actor Critic (A3C)

- A3C (Mnih et al. 2016) idea: Sample for data can be parallelized using several copies of the same agent
  - use N copies of the agents (workers) working in parallel collecting samples and computing gradients for policy and value function
  - After some time, pass gradients to a main network that updates actor and critic using the gradients of all agents

Asynchronous Advantage Actor Critic (A3C)

- A3C (Mnih et al. 2016) idea: Sample for data can be parallelized using several copies of the same agent
  - use N copies of the agents (workers) working in parallel collecting samples and computing gradients for policy and value function
  - After some time, pass gradients to a main network that updates actor and critic using the gradients of all agents
  - After some time the worker copy the weights of the global network

Asynchronous Advantage Actor Critic (A3C)

- A3C (Mnih et al. 2016) idea: Sample for data can be parallelized using several copies of the same agent
  - use N copies of the agents (workers) working in parallel collecting samples and computing gradients for policy and value function
  - After some time, pass gradients to a main network that updates actor and critic using the gradients of all agents
  - After some time the worker copy the weights of the global network
- This parallelism decorrelates the agents' data, so no Experience Replay Buffer needed

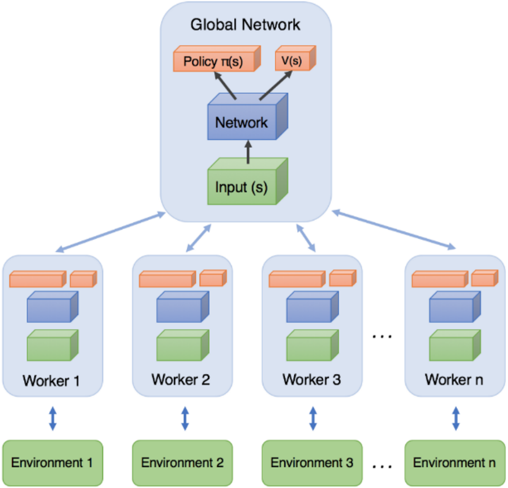
Asynchronous Advantage Actor Critic (A3C)

- A3C (Mnih et al. 2016) idea: Sample for data can be parallelized using several copies of the same agent
  - use N copies of the agents (workers) working in parallel collecting samples and computing gradients for policy and value function
  - After some time, pass gradients to a main network that updates actor and critic using the gradients of all agents
  - After some time the worker copy the weights of the global network
- This parallelism decorrelates the agents' data, so no Experience Replay Buffer needed
- Even one can explicitly use different exploration policies in each actor-learner to maximize diversity

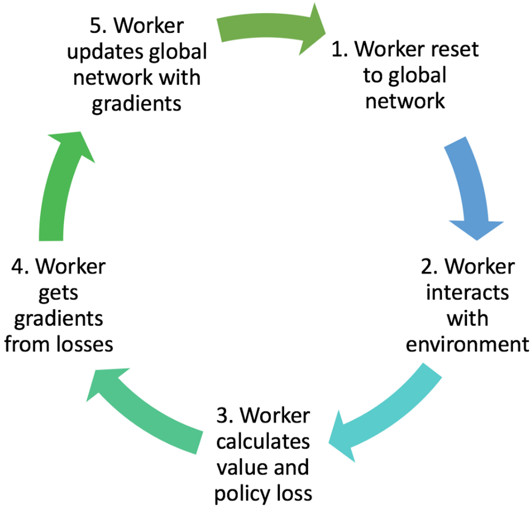
Asynchronous Advantage Actor Critic (A3C)

- A3C (Mnih et al. 2016) idea: Sample for data can be parallelized using several copies of the same agent
  - use N copies of the agents (workers) working in parallel collecting samples and computing gradients for policy and value function
  - After some time, pass gradients to a main network that updates actor and critic using the gradients of all agents
  - After some time the worker copy the weights of the global network
- This parallelism decorrelates the agents' data, so no Experience Replay Buffer needed
- Even one can explicitly use different exploration policies in each actor-learner to maximize diversity
- Asynchronism can be extended to other update mechanisms (SARSA, Q-learning, etc) but it works better in Advantage Actor critic setting

Asynchronous Advantage Actor Critic (A3C)



Asynchronous Advantage Actor Critic (A3C)



Asynchronous Advantage Actor Critic (A3C)

Algorithm S3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

// Assume global shared parameter vectors θ and θ_v and global shared counter $T = 0$
// Assume thread-specific parameter vectors θ' and θ'_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 θ': $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i;\theta')(R - V(s_i;\theta'_v))$
        Accumulate gradients wrt θ'_v: $d\theta_v \leftarrow d\theta_v + \partial\left(R - V(s_i;\theta'_v)\right)^2 / \partial\theta'_v$
    end for
    Perform asynchronous update of θ using $d\theta$ and of θ_v using $d\theta_v$.
until $T > T_{max}$