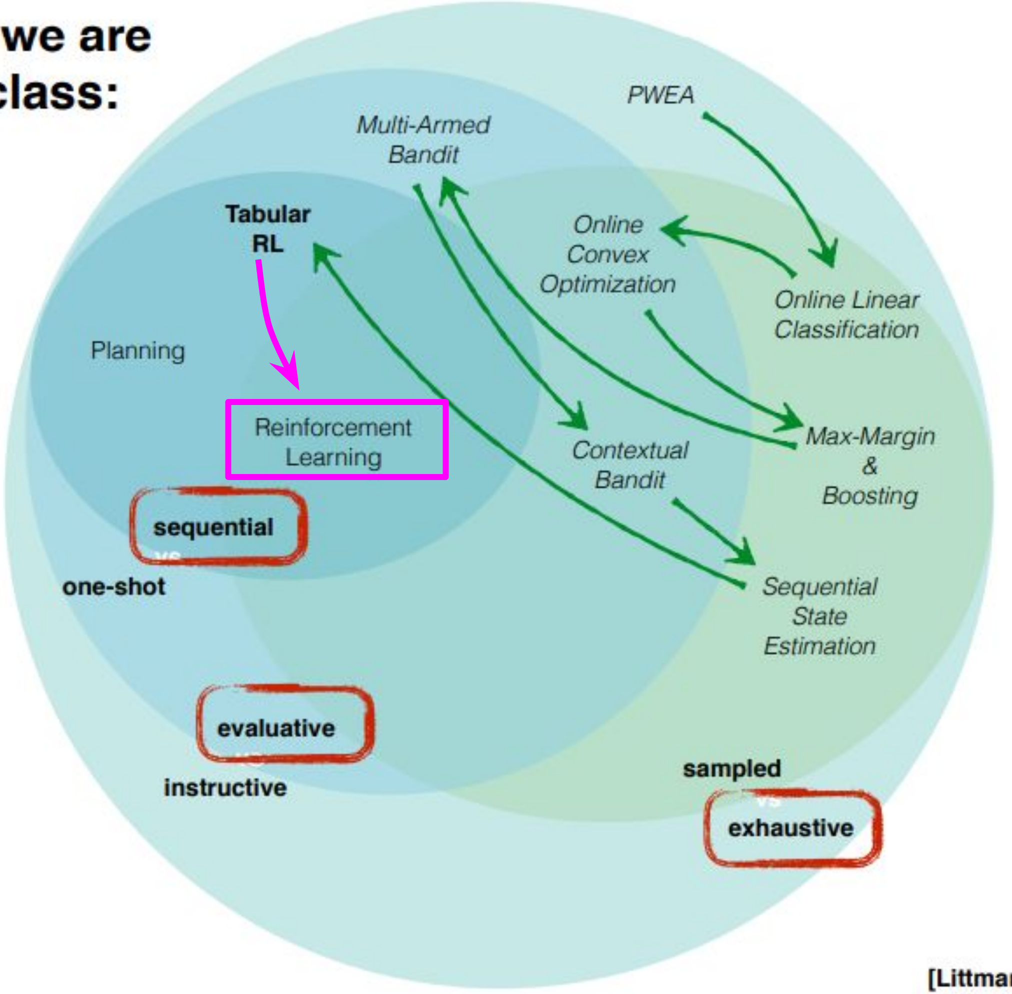
A3C: Asynchronous Advantage Actor Critic

16-831 Statistical Techniques in Robotics

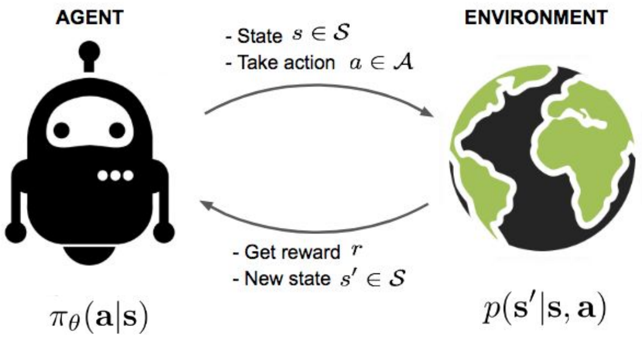
Carnegie Mellon University (Malhar Bhoite, Vinay Varma, Ashish Roongta)

Where we are in the class:



[Littman Nature 2015]

Reinforcement Learning (Recap)



Markov decision process $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{T}, r\}$

$\mathcal{S}$ – state space — states $s \in \mathcal{S}$ (discrete or continuous)

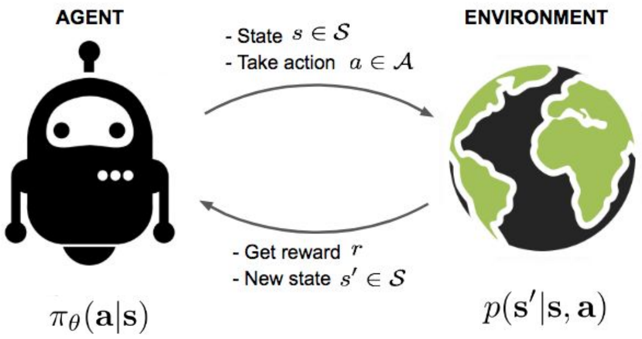
$\mathcal{A}$ – action space — actions $a \in \mathcal{A}$ (discrete or continuous)

r – reward function — $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$

$r(s_t, a_t)$ – reward

Objective:
Learn to take actions that give the most reward

Reinforcement Learning (Recap)



Markov decision process $\quad \mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{T}, r\}$

$\mathcal{S}$ – state space $\quad$ states $s \in \mathcal{S}$ (discrete or continuous)

$\mathcal{A}$ – action space $\quad$ actions $a \in \mathcal{A}$ (discrete or continuous)

r – reward function $\quad r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$

$r(s_t, a_t)$ – reward

Objective:
Learn to take actions that give the most reward

What are the various techniques to solve RL we have learnt so far?

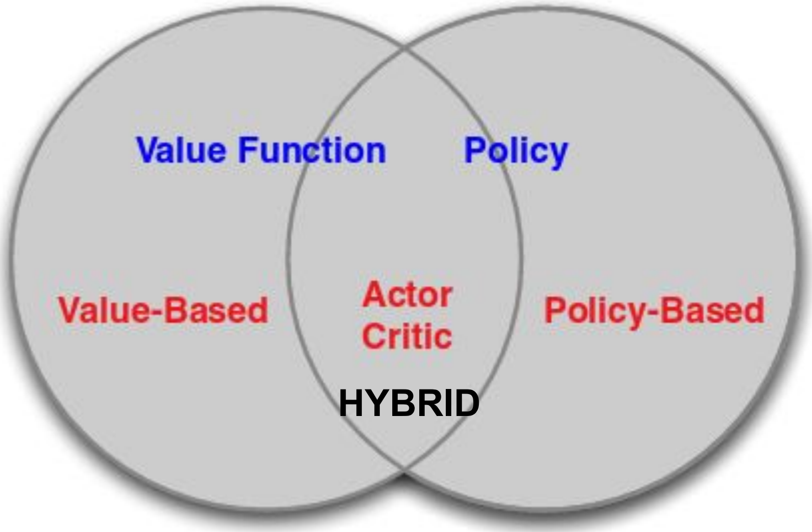
Reinforcement Learning (Recap)

Methods for Solving Reinforcement Learning (Learning the Policy)

Policy Gradient Methods

Temporal Difference Learning
(Value-Based Methods)

Actor-Critic Methods



Can we combine the strengths of both methods?

Roadmap

Methods for Solving Reinforcement Learning (Learning the Policy)

Policy Gradient Methods

Temporal Difference Learning
(Value-Based Methods)

Strengths

Actor-Critic Methods

Strengths

Improve Math

Advantage Actor-Critic

Improve Computation

Asynchronous Advantage Actor-Critic (A3C)

Asynchronous Methods for Deep Reinforcement Learning

Volodymyr Mnih[1] VMNIH@GOOGLE.COM
Adrià Puigdomènech Badia[1] ADRIAP@GOOGLE.COM
Mehdi Mirza[1,2] MIRZAMOM@IRO.UMONTREAL.CA
Alex Graves[1] GRAVESA@GOOGLE.COM
Tim Harley[1] THARLEY@GOOGLE.COM
Timothy P. Lillicrap[1] COUNTZERO@GOOGLE.COM
David Silver[1] DAVIDSILVER@GOOGLE.COM
Koray Kavukcuoglu [1] KORAYK@GOOGLE.COM
[1] Google DeepMind
[2] Montreal Institute for Learning Algorithms (MILA), University of Montreal

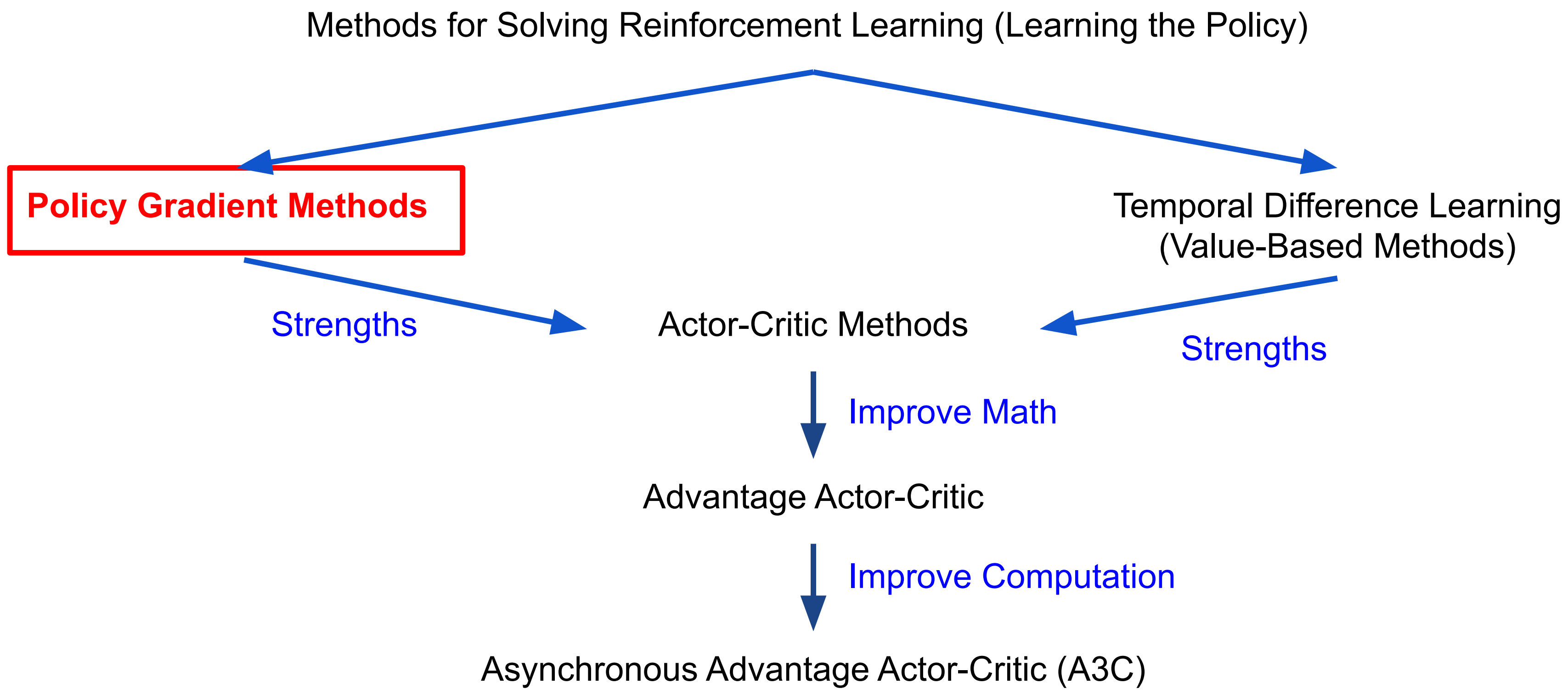
Abstract

We propose a conceptually simple and lightweight framework for deep reinforcement learning that uses asynchronous gradient descent for optimization of deep neural network controllers. We present asynchronous variants of four standard reinforcement learning algorithms and show that parallel actor-learners have a stabilizing effect on training allowing all four methods to successfully train neural network controllers. The best performing method, an asynchronous variant of actor-critic, surpasses the current state-of-the-art on the Atari domain

line RL updates are strongly correlated. By storing the agent's data in an experience replay memory, the data can be batched (Riedmiller, 2005; Schulman et al., 2015a) or randomly sampled (Mnih et al., 2013; 2015; Van Hasselt et al., 2015) from different time-steps. Aggregating over memory in this way reduces non-stationarity and decorrelates updates, but at the same time limits the methods to off-policy reinforcement learning algorithms.

Deep RL algorithms based on experience replay have achieved unprecedented success in challenging domains such as Atari 2600. However, experience replay has several drawbacks: it uses more memory and computation per real interaction; and it requires off-policy learning algorithms

Roadmap


Methods for Solving Reinforcement Learning (Learning the Policy)
Policy Gradient Methods
Temporal Difference Learning
(Value-Based Methods)
Strengths
Actor-Critic Methods
Strengths
Improve Math
Advantage Actor-Critic
Improve Computation
Asynchronous Advantage Actor-Critic (A3C)

Policy Gradient Theorem (Recap)

1. Start by Defining a Parameterized Policy $\boldsymbol{\pi_\theta}$

Objective: Maximize Expected Rewards

2. Define an Objective Function ***J(θ)***

$$J(\theta) = \sum_{s \in S} d^\pi(s) V^\pi(s) = \sum_{s \in S} d^\pi(s) \sum_{a \in A} \pi_\theta(a|s) Q^\pi(s,a)$$

3. Calculate the Gradient of the Objective Function $\nabla_\theta J(\theta)$

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

4. Plug values in the update rule $\theta \leftarrow \alpha \nabla_\theta J(\theta)$

5. Repeat till converged

Recall:

$$V^\pi(\mathbf{s}_t) = E_{\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}[Q^\pi(\mathbf{s}_t, \mathbf{a}_t)]$$

$$Q^\pi(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{T} E_{\pi_\theta}\left[r(\mathbf{s}_{t'}, \mathbf{a}_{t'})|\mathbf{s}_t, \mathbf{a}_t\right]$$

Policy Gradient Theorem (Recap)

1. Start by Defining a Parameterized Policy $\boldsymbol{\pi_\theta}$

Objective: Maximize Expected Rewards

2. Define an Objective Function $\boldsymbol{J(\theta)}$

$$J(\theta) = \sum_{s \in S} d^\pi(s) V^\pi(s) = \sum_{s \in S} d^\pi(s) \sum_{a \in A} \pi_\theta(a|s) Q^\pi(s, a)$$

3. Calculate the Gradient of the Objective Function $\boldsymbol{\nabla_\theta J(\theta)}$

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

4. Plug values in the update rule $\boldsymbol{\theta \leftarrow \alpha \nabla_\theta J(\theta)}$

What are these terms?

5. Repeat till converged

https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html

Policy and the Gain

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

State-Action Value Function

Policy

Policy and the Gain

$$\nabla_\theta J(\theta) = \mathrm{E}_{\pi_\theta}[\nabla_\theta \boxed{log\pi_\theta(s,a)}\boxed{R(s,a)}]$$

‘Policy’
Function Approximator
(Eg. Logistic Regressor, Neural Network, etc.)

State →Action

‘Gain’ Term (General)
Value Function - Estimates expected return
Scalar value

Intuitively, this **evaluates** the policy

Policy and the Gain

$$\nabla_\theta J(\theta) = \mathrm{E}_{\pi_\theta}[\nabla_\theta \boxed{log\pi_\theta(s,a)}\boxed{R(s,a)}]$$

'Policy'
Function Approximator
(Eg. Logistic Regressor, Neural Network, etc.)

State →Action

'Gain' Term (General)
Value Function - Estimates expected return
Scalar value

Intuitively, this **evaluates** the policy

Policy and the Gain

$$\nabla_\theta J(\theta) = \mathrm{E}_{\pi_\theta}[\nabla_\theta \boxed{log\pi_\theta(s,a)} \boxed{R(s,a)}]$$

'Policy'
Function Approximator
(Eg. Logistic Regressor, Neural Network, etc.)

State →Action

'Gain' Term (General)
Value Function - Estimates expected return
Scalar value

Intuitively, this **evaluates** the policy

What are the Different ways of estimating this gain term?

Policy Evaluation Methods (Estimating Gain)

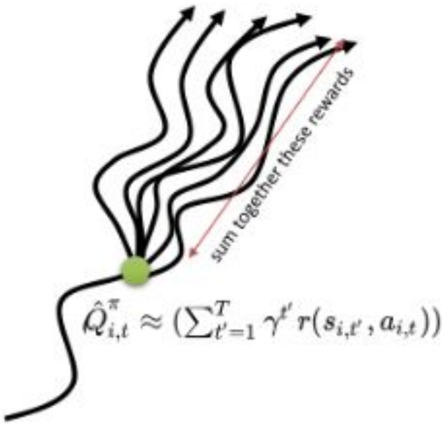
$$\nabla_\theta J(\theta) = \mathrm{E}_{\pi_\theta}[\nabla_\theta log \pi_\theta(s,a) \boxed{R(s,a)]}$$

- $\sum_{t'=t}^{T} \gamma^{t'-t} r(s_{i,t'}, a_{i,t'})$ Total Reward of Simulated Trajectories (Monte Carlo Rollout)
- $\sum_{t'=t}^{T} \gamma^{t'-t} r(s_{i,t'}, a_{i,t'}) - b(s_t)$ Monte Carlo Rollout with Baseline
- $Q^\pi(s_t, a_t)$ State-Action Value Function
- $A^\pi(s_t, a_t)$ Advantage Function
- $r_t + \gamma V^\pi(s_{t+1}) - V^\pi(s_t)$ TD Residual

Empirically Calculate Gain (REINFORCE)

Simulate episodes and calculate ***empirical mean return*** *instead of* ***expected return:***

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \boxed{\hat{Q}^{\pi}_{i,t}}$$



http://rail.eecs.berkeley.edu/deeprlcourse-fa17/f17docs/lecture_5_actor_critic_pdf.pdf

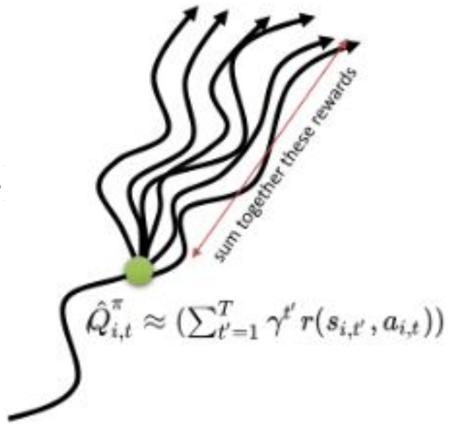
Empirically Calculate Gain (REINFORCE)

Simulate episodes and calculate ***empirical mean return*** *instead of* ***expected return:***

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\boxed{\hat{Q}^{\pi}_{i,t}}$$

Problems with Monte Carlo Rollout/ REINFORCE:

- Calculating the Gain term is **slow** (Wait till every episode terminates
- Empirical mean is unbiased, but has **high variance**



http://rail.eecs.berkeley.edu/deeprlcourse-fa17/f17docs/lecture_5_actor_critic_pdf.pdf

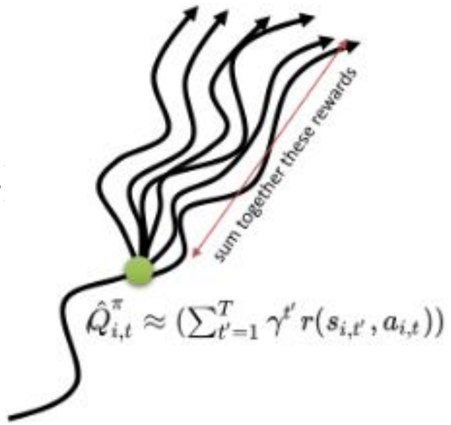
Empirically Calculate Gain (REINFORCE)

*Simulate episodes and calculate **empirical mean return** instead of **expected return:***

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \boxed{\hat{Q}^\pi_{i,t}}$$

Problems with Monte Carlo Rollout/ REINFORCE:

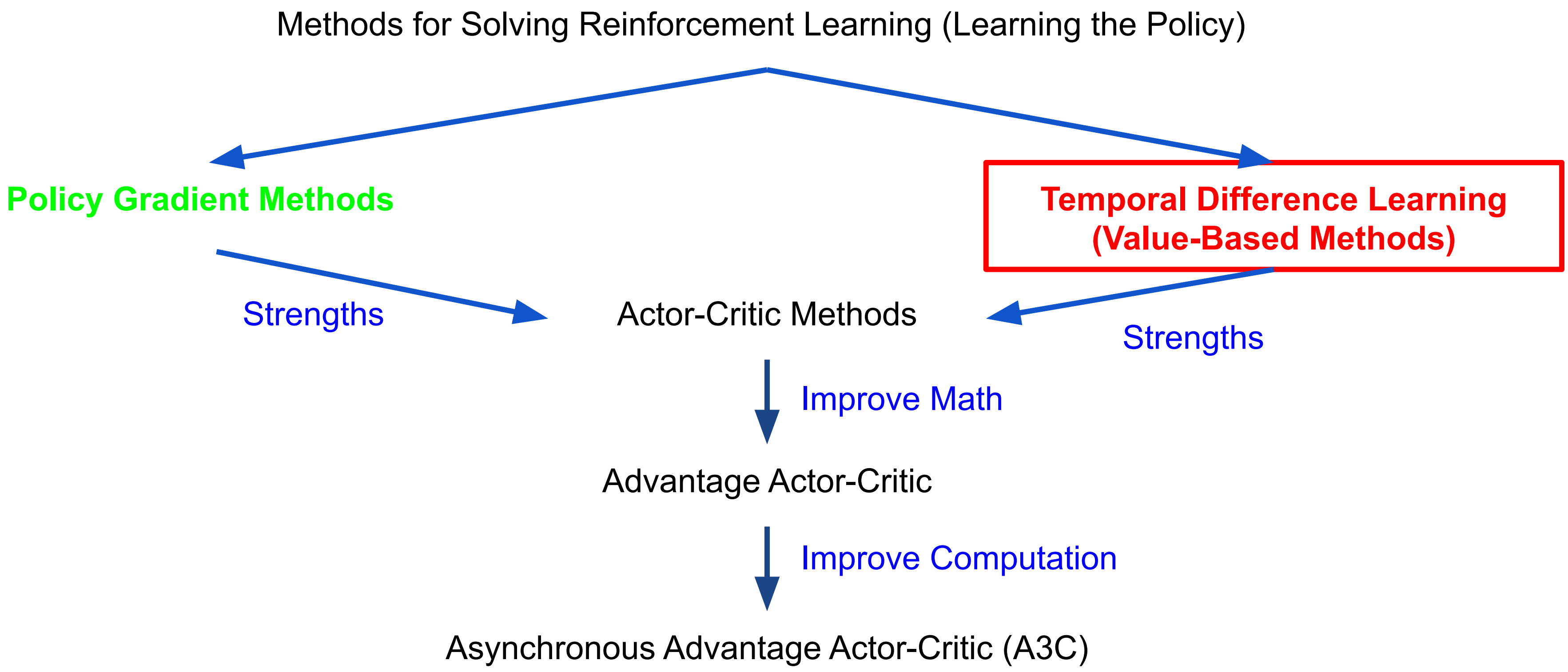
- Calculating the Gain term is **slow** (Wait till every episode terminates
- Empirical mean is unbiased, but has **high variance**



Any faster way of estimating the Gain term?

http://rail.eecs.berkeley.edu/deeprlcourse-fa17/f17docs/lecture_5_actor_critic_pdf.pdf

Roadmap



Methods for Solving Reinforcement Learning (Learning the Policy)
Policy Gradient Methods
Temporal Difference Learning
(Value-Based Methods)
Strengths
Actor-Critic Methods
Strengths
Improve Math
Advantage Actor-Critic
Improve Computation
Asynchronous Advantage Actor-Critic (A3C)

Temporal Difference Methods (Recap)

TD Methods give **another way to estimate** the **value function**

$$\delta = \underline{R_{t+1} + \gamma V^{\pi} s_{t+1}} - V^{\pi} s_t \qquad \Big| \qquad \delta = \underline{R_{t+1} + \gamma Q^{\pi} s_{t+1}} - Q^{\pi} s_t$$

Value Estimate after one step, TD(0)

Richard S. Sutton and Andrew G. Barto. [Reinforcement Learning: An Introduction; 2nd Edition](). 2017.

Temporal Difference Methods (Recap)

TD Methods give **another way to estimate** the **value function**

$$\delta = \underline{R_{t+1} + \gamma V^{\pi} s_{t+1}} - V^{\pi} s_t \qquad \Big| \qquad \delta = \underline{R_{t+1} + \gamma Q^{\pi} s_{t+1}} - Q^{\pi} s_t$$

Value Estimate after one step, TD(0)

- Computationally much faster : One-Step vs Complete Episode (Monte Carlo)
- **Low variance**
- Can learn in environments without a final outcome

Richard S. Sutton and Andrew G. Barto. Reinforcement Learning: An Introduction; 2nd Edition. 2017.

Temporal Difference Methods (Recap)

TD Methods give **another way to estimate** the **value function**

$$\delta = \underline{R_{t+1} + \gamma V^{\pi} s_{t+1}} - V^{\pi} s_t \qquad \Big| \qquad \delta = \underline{R_{t+1} + \gamma Q^{\pi} s_{t+1}} - Q^{\pi} s_t$$

Value Estimate after one step, TD(0)

- Computationally much faster : One-Step vs Complete Episode (Monte Carlo)
- **Low variance**
- Can learn in environments without a final outcome

But we're dealing with very large state and action spaces - what frameworks can we use?

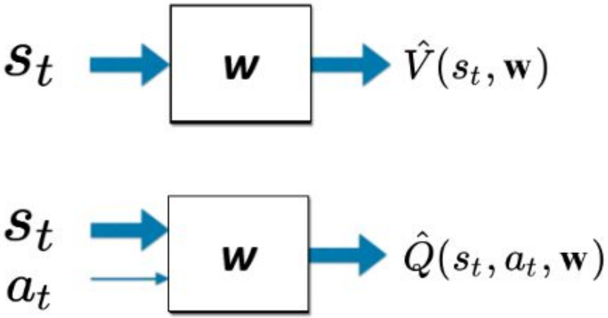
Richard S. Sutton and Andrew G. Barto. Reinforcement Learning: An Introduction; 2nd Edition. 2017.

Value Function Approximation

$$\delta = R_{t+1} + \gamma V^{\pi} s_{t+1} - V^{\pi} s_t \qquad \delta = R_{t+1} + \gamma Q^{\pi} s_{t+1} - Q^{\pi} s_t$$

New Function Approximator to Estimate the Value Function/ Evaluate Policy

*Parameters **w***



http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/FA.pdf

Value Function Approximation (General Method)

$$C(w) = \mathbb{E}_\pi[(\underbrace{V_\pi(s)}_{\text{Target/True Value Function}} - \underbrace{\hat{V}(s, \mathbf{w})}_{\text{Approximate Value Function}})^2]$$

Notation Alert: ^ for Value Function Approx.

http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/FA.pdf

Value Function Approximation (General Method)

$$C(w) = \mathbb{E}_\pi[(\underline{V_\pi(s)} - \underline{\hat{V}(s,\mathbf{w})})^2]$$

Notation Alert: ^ for Value Function Approx.

Target/True Value Function

Approximate Value Function

$$\Delta\mathbf{w} = -\frac{1}{2}\alpha\nabla_\mathbf{w}C(\mathbf{w}) = \alpha\mathbb{E}_\pi[(V_\pi(s) - \hat{V}(s,\mathbf{w}))\nabla_\mathbf{w}\hat{V}(s,\mathbf{w})]$$

*Solve using **Gradient Descent***

http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/FA.pdf

Value Function Approximation (General Method)

$$C(w) = \mathbb{E}_\pi[(\underbrace{V_\pi(s)}_{\text{Target/True Value Function}} - \underbrace{\hat{V}(s, \mathbf{w})}_{\text{Approximate Value Function}})^2]$$

Notation Alert: ^ for Value Function Approx.

$$\Delta \mathbf{w} = -\frac{1}{2}\alpha \nabla_{\mathbf{w}} C(\mathbf{w}) = \alpha \mathbb{E}_\pi[(V_\pi(s) - \hat{V}(s, \mathbf{w}))\nabla_{\mathbf{w}} \hat{V}(s, \mathbf{w})]$$

*Solve using **Gradient Descent***

$$\Delta \mathbf{w} = \alpha[(V_\pi(s) - \hat{V}(s, \mathbf{w}))\nabla_{\mathbf{w}} \hat{V}(s, \mathbf{w})]$$

Update parameters

http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/FA.pdf

Value Function Approximation (TD(0) Update)

$$C(w) = \mathbb{E}_\pi[(\underline{V_\pi(s)} - \underline{\hat{V}(s,\mathbf{w})})^2]$$

Notation Alert: ^ for Value Function Approx.

Target/True Value Function

Approximate Value Function

$$\Delta\mathbf{w} = -\frac{1}{2}\alpha\nabla_{\mathbf{w}}C(\mathbf{w}) = \alpha\mathbb{E}_\pi[(V_\pi(s) - \hat{V}(s,\mathbf{w}))\nabla_{\mathbf{w}}\hat{V}(s,\mathbf{w})]$$

*Solve using **Gradient Descent***

$$\Delta\mathbf{w} = \alpha[(V_\pi(s) - \hat{V}(s,\mathbf{w}))\nabla_{\mathbf{w}}\hat{V}(s,\mathbf{w})]$$

Update parameters

Target Value → Value Function using TD Estimation

$$\Delta\mathbf{w} = \alpha[\underline{R_{t+1} + \gamma\hat{V}(s_{t+1},\mathbf{w}) - \hat{V}(s,\mathbf{w})})\nabla_{\mathbf{w}}\hat{V}(s,\mathbf{w})]$$

Same logic applicable to Q Value Function

http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/FA.pdf

To Summarize...

Value-Based Methods (e.g. DQN)

Strength:
Fast Estimators of Action Quality
(given state, action)

Weaknesses:
Cannot deal with very large/continuous action spaces (*argmax over actions!*)

Policy Gradient Methods (e.g. REINFORCE)

Strength:
Tackle Large Action Spaces

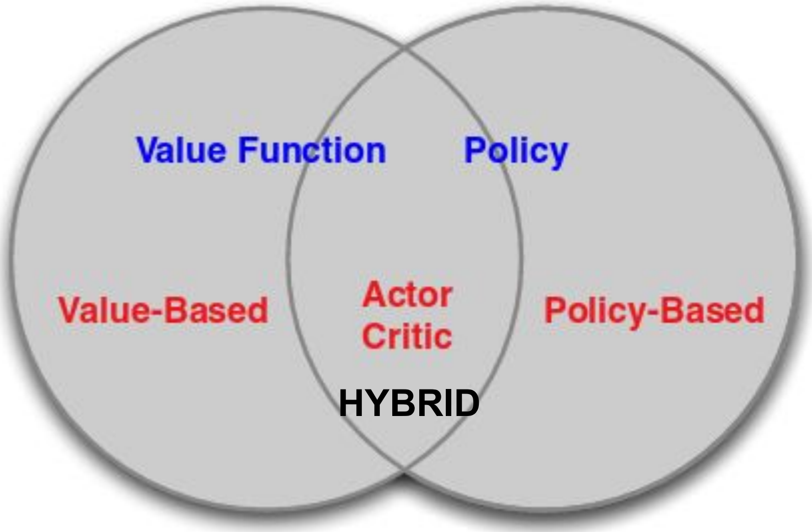
Weaknesses:
Action quality estimation is slow (Monte Carlo rollout)

An Idea - Combine Strengths

Value-Based Methods (e.g. DQN)

Strength:
Fast Estimators of Action Quality
(given state, action)

Weaknesses:
Cannot deal with very large/continuous action spaces (*argmax over actions!*)



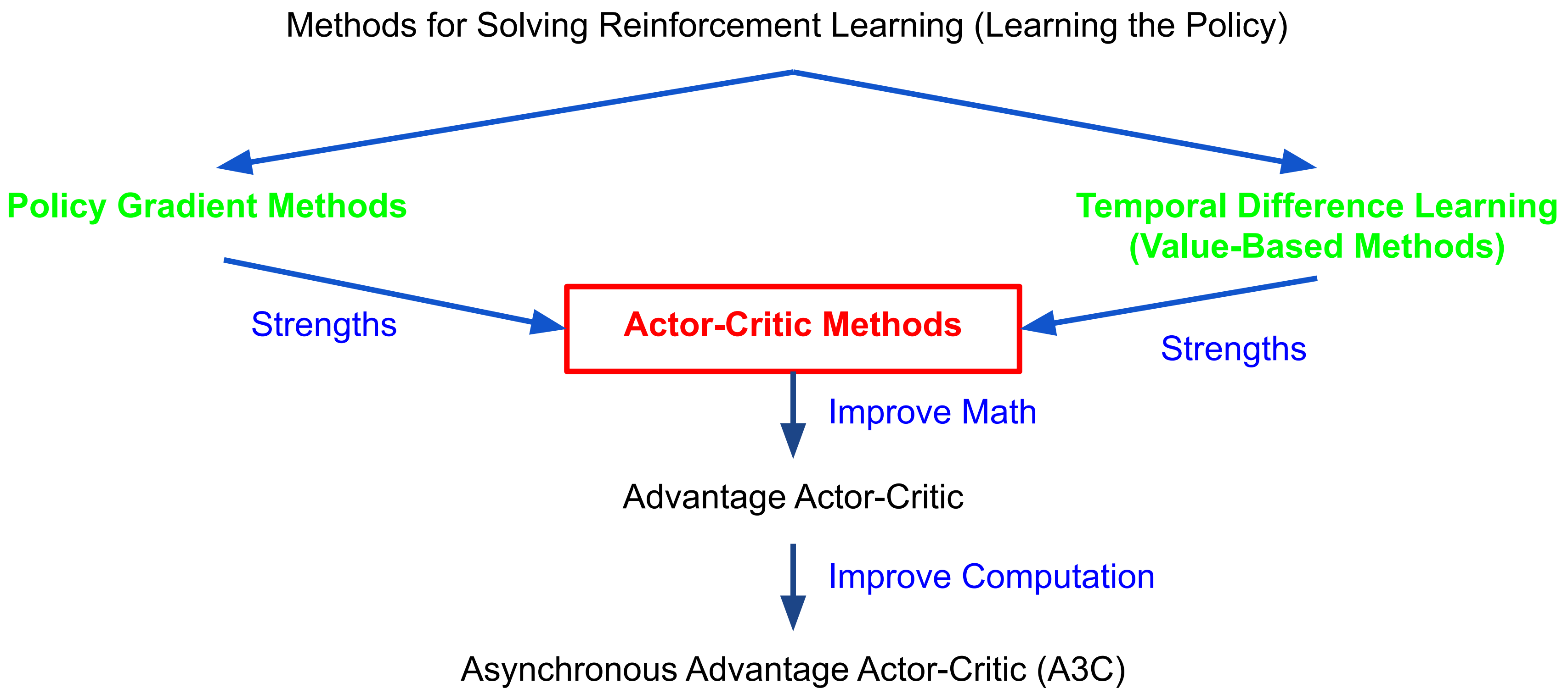
Can we combine the strengths of both methods?

Policy Gradient Methods (e.g. REINFORCE)

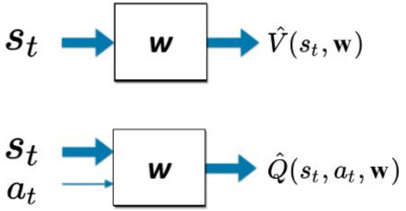
Strength:
Tackle Large Action Spaces

Weaknesses:
Action quality estimation is slow (Monte Carlo rollout)

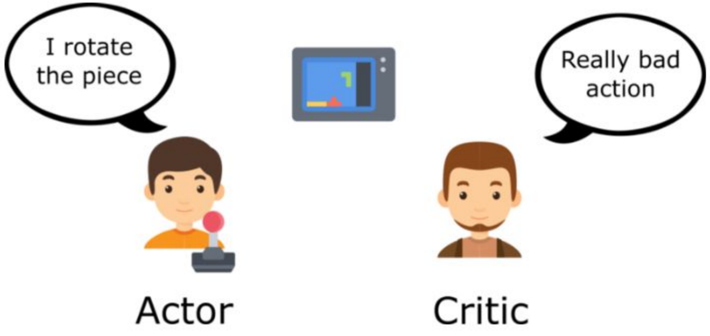
Roadmap


Methods for Solving Reinforcement Learning (Learning the Policy)
Policy Gradient Methods
Temporal Difference Learning
(Value-Based Methods)
Strengths
Actor-Critic Methods
Strengths
Improve Math
Advantage Actor-Critic
Improve Computation
Asynchronous Advantage Actor-Critic (A3C)

Actor-Critic Methods: Intuition

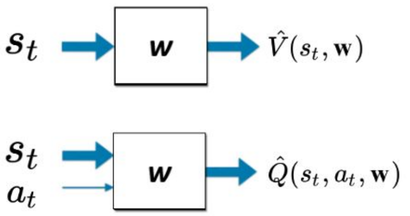


Take Actions Using Policy Gradient Methods
State →Action



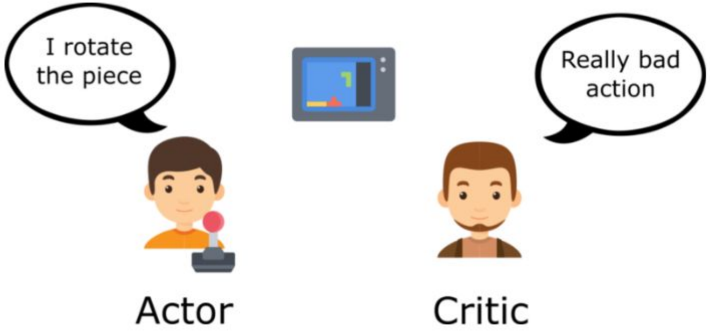
Evaluate Policy Using TD Learning
State → ***Value Function***

Actor-Critic Methods: Intuition



Take Actions Using Policy Gradient Methods
State →Action

Use 'Policy' Function Approximator with Parameter θ

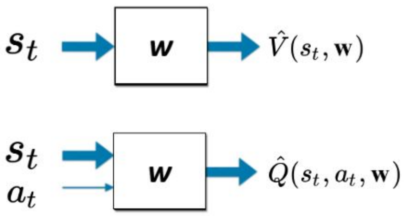


Evaluate Policy Using TD Learning
State → *Value Function*

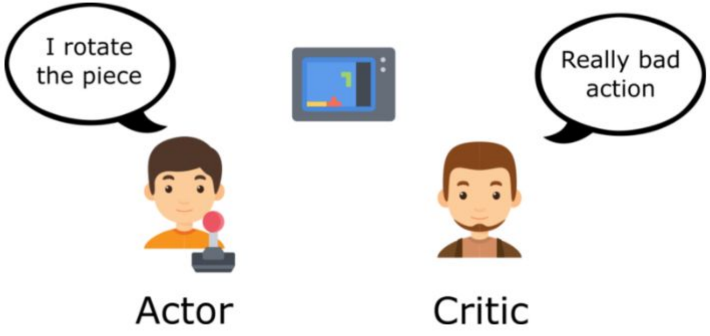
Use 'Value' Function Approximator with parameters w to calculate Value Function (Q or V)

$$\nabla_\theta J(\theta) = \mathrm{E}_{\pi_\theta}[\nabla_\theta \boxed{log \pi_\theta(s, a)} \boxed{R(s, a)}]$$

Actor-Critic Methods: Intuition



Take Actions Using Policy Gradient Methods
State →Action



Evaluate Policy Using TD Learning
State → ***Value Function***

Use 'Policy' Function Approximator with Parameter θ

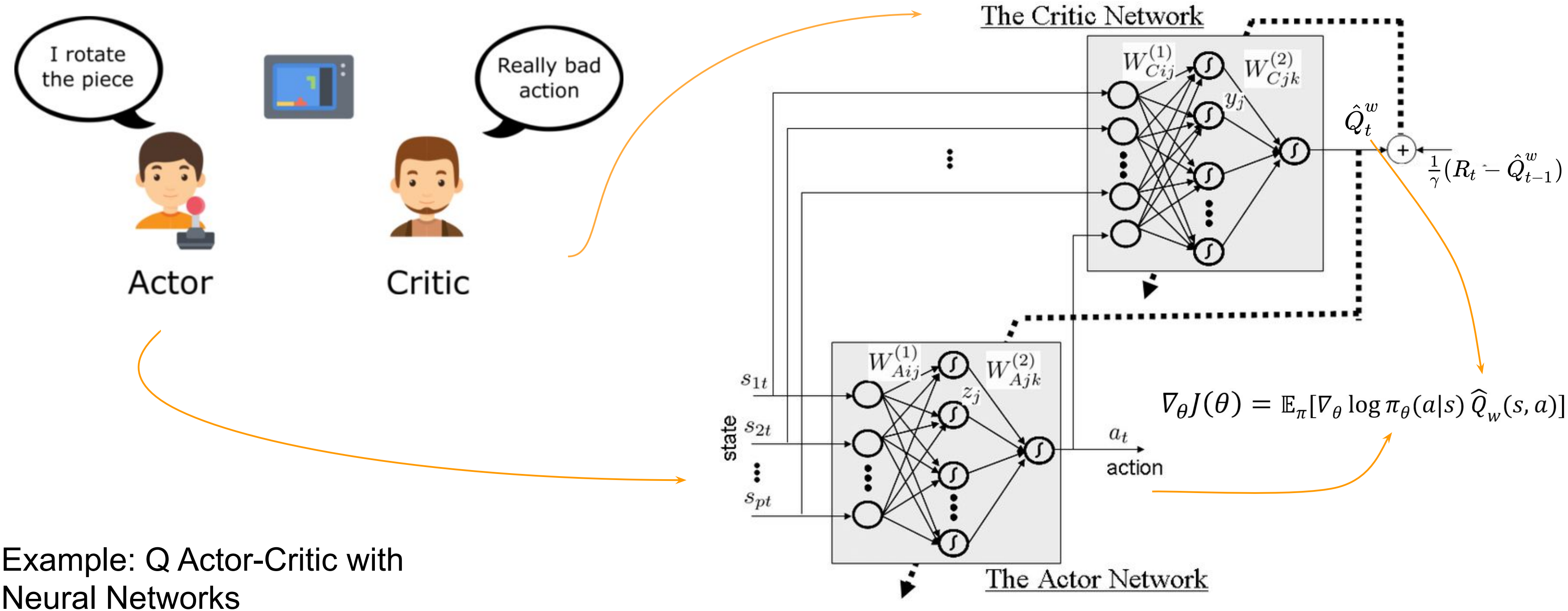
Use 'Value' Function Approximator with parameters w to calculate Value Function (Q or V)

$$\nabla_\theta J(\theta) = \mathrm{E}_{\pi_\theta}[\nabla_\theta \boxed{log \pi_\theta(s,a)} \boxed{R(s,a)}]$$

Q Actor-Critic

Advantage Actor-Critic

Actor-Critic Methods: Visualization



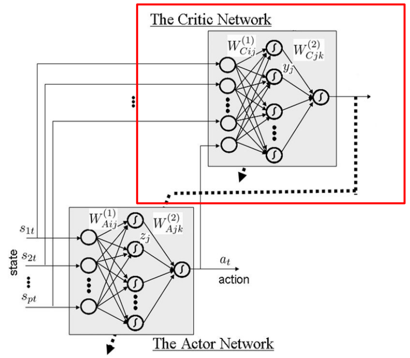
Example: Q Actor-Critic with Neural Networks

Q Actor Critic

Actor Update (Gradient Ascent)

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

Replace Gain Term



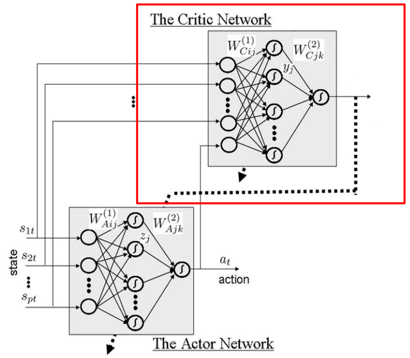
$$Q^{\pi_\theta} \approx \hat{Q}_w(s,a)$$

Use Value calculated from Value function approximator

$$\nabla_\theta J(\theta) \approx [\nabla_\theta log\pi_\theta(s,a)\boxed{\hat{Q}_w(s,a)]}$$

$$\Delta\theta = \alpha\nabla_\theta \log \pi_\theta(s,a)\hat{Q}_w(s,a)$$

Q Actor Critic



Actor Update (Gradient Ascent)

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

Replace Gain Term

$$Q^{\pi_\theta} \approx \hat{Q}_w(s,a)$$

Use Value calculated from Value function approximator

$$\nabla_\theta J(\theta) \approx [\nabla_\theta log\pi_\theta(s,a)\boxed{\hat{Q}_w(s,a)]}$$

$$\Delta\theta = \alpha\nabla_\theta \log \pi_\theta(s,a)\hat{Q}_w(s,a)$$

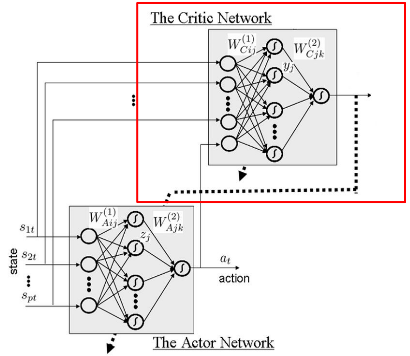
Critic Update (Gradient Descent)

$$\Delta \mathbf{w} = \alpha[\underline{R_{t+1} + \gamma\hat{Q}(s',a',\mathbf{w}) - \hat{Q}(s,a,\mathbf{w})})\nabla_{\mathbf{w}}\hat{Q}(s,a,\mathbf{w})]$$

One step TD error

Gradient Of Value Function

Advantage Actor Critic



Actor Update (Gradient Ascent)

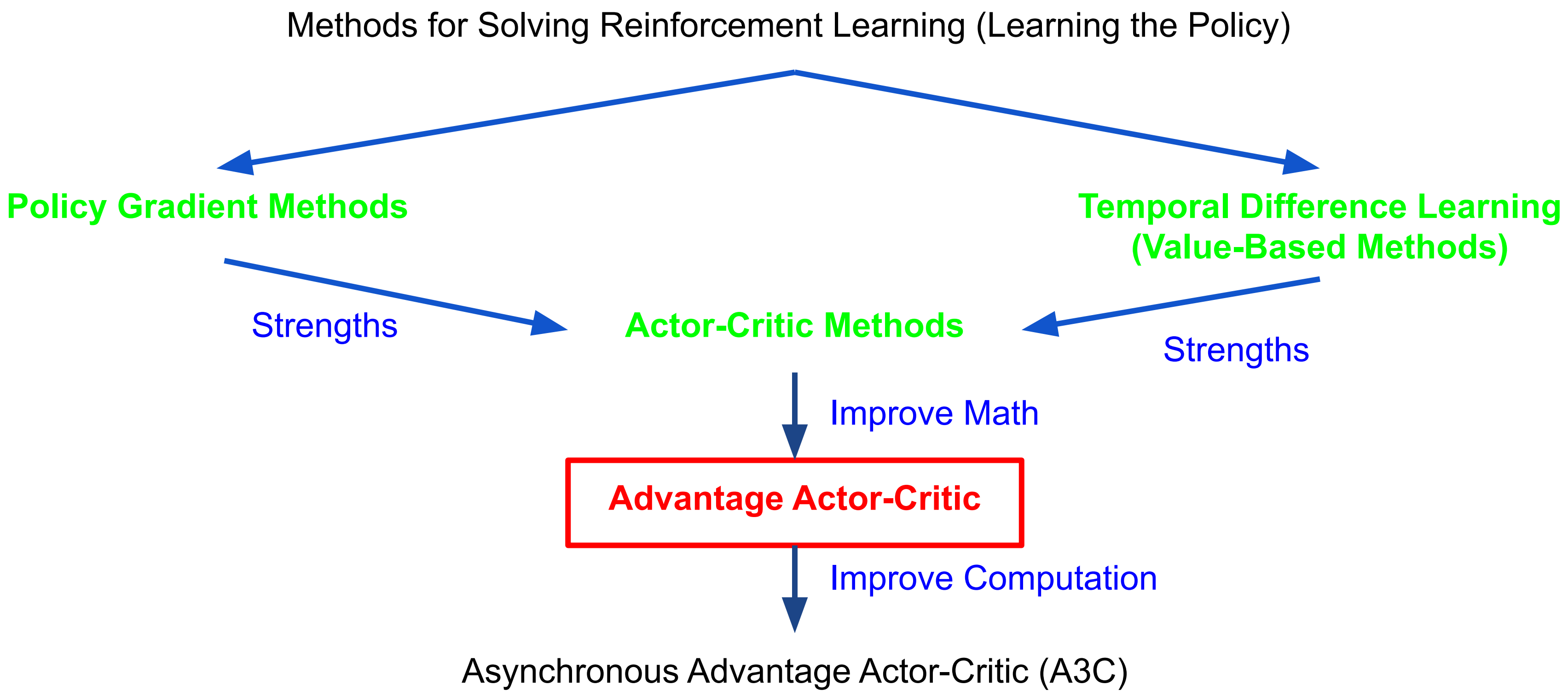
$$\nabla_\theta J(\theta) = \mathbb{E}_\pi[\nabla_\theta \log \pi_\theta(a|s)\boxed{\hat{A}_w(s,a)]}$$

Critic Update (Gradient Descent)

$$\Delta \mathbf{w} = \alpha[\underbrace{R_{t+1} + \gamma \hat{V}(s_{t+1}, \mathbf{w}) - \hat{V}(s, \mathbf{w})}_{\text{One step TD error}})\nabla_{\mathbf{w}} \hat{V}(s, \mathbf{w})]$$

Gradient Of Value Function

Roadmap


Methods for Solving Reinforcement Learning (Learning the Policy)
Policy Gradient Methods
Temporal Difference Learning
(Value-Based Methods)
Strengths
Actor-Critic Methods
Strengths
Improve Math
Advantage Actor-Critic
Improve Computation
Asynchronous Advantage Actor-Critic (A3C)

Advantage

$$A(\mathrm{s}, \mathrm{a}) = Q(s, a) - V(s)$$

Advantage

$$A(\mathrm{s}, \mathrm{a}) = Q(s, a) - \boxed{V(s)}$$

$$\sum_a \pi(a|s) Q(s, a)$$

Intuitively, this means how much better it is to take a specific action compared to the average, general action at the given state

Advantage Actor Critic

Policy Gradient:

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

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

Advantage Actor Critic

Policy Gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi[\nabla_\theta \log \pi_\theta(a|s)\, \cancel{Q_{\pi_\theta}(s,a)}^{A_{\pi_\theta}(s,a)}]$$

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

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

Advantage Actor Critic

Why??

Is this a different Policy Gradient??

Policy Gradient:

$$A_{\pi_\theta}(s,a)$$

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

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

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

Baseline: Variance reduction

$$\mathrm{Var}(f-\phi) = \mathrm{Var}(f) - \underbrace{2\mathrm{Cov}(f,\phi)}_{\text{Large when } f,\ \phi \text{ strongly correlated}} + \mathrm{Var}(\phi)$$

$\longrightarrow$ $\mathrm{Var}(f-\phi) < \mathrm{Var}(f)$

If ϕ and f are strongly correlated, so that the covariance term on the right hand side is greater than the variance of ϕ, then a variance improvement has been made over the original estimation problem.

This is a generic approach to reducing variance of Monte carlo estimates for integrals

How does variance reduction help here?

Better estimates of policy gradient

Greensmith, Evan, Bartlett, Peter L., and Baxter, Jonathan. "Variance reduction techniques for gradient estimates in reinforcement learning."

Advantage Actor Critic

Why??

Variance reduction

Is this a different Policy Gradient??

Policy Gradient:

$$A_{\pi_\theta}(s,a)$$

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

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

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

Reward shaping

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_\pi[\nabla_\theta \log \pi_\theta(a|s)\,(Q_{\pi_\theta}(s,a) - V_{\pi_\theta}(s))] \\
&= \mathbb{E}_\pi[\nabla_\theta \log \pi_\theta(a|s)\, Q_{\pi_\theta}(s,a)] \; - \; \mathbb{E}_\pi[\nabla_\theta \log \pi_\theta(a|s)\, V_{\pi_\theta}(s)]
\end{aligned}
$$

Baseline

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi[\nabla_\theta \log \pi_\theta(a|s)\,(Q_{\pi_\theta}(s,a) - V_{\pi_\theta}(s))]$$
$$= \boxed{\mathbb{E}_\pi[\nabla_\theta \log \pi_\theta(a|s)\, Q_{\pi_\theta}(s,a)]} - \mathbb{E}_\pi[\nabla_\theta \log \pi_\theta(a|s)\, V_{\pi_\theta}(s)]$$

Policy gradient with Q function

Baseline: Unbiased estimate

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_\pi[\nabla_\theta \log \pi_\theta(a|s)\,(Q_{\pi_\theta}(s,a) - V_{\pi_\theta}(s))] \\
&= \boxed{\mathbb{E}_\pi[\nabla_\theta \log \pi_\theta(a|s)\, Q_{\pi_\theta}(s,a)]} - \boxed{\mathbb{E}_\pi[\nabla_\theta \log \pi_\theta(a|s)\, V_{\pi_\theta}(s)]}
\end{aligned}
$$

Policy gradient with Q function

$$
\mathbb{E}_\pi\left[\nabla_\theta \log \pi_\theta(a|s)\, V_{\pi_\theta}(s)\right] = 0
$$

Unbiased estimate of the policy gradient for any baseline shift *b(s)* in reward. Here,

b(s) = V(s)

Baseline: Unbiased estimate

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi[\nabla_\theta \log \pi_\theta(a|s)\,(Q_{\pi_\theta}(s,a) - V_{\pi_\theta}(s))]$$
$$= \boxed{\mathbb{E}_\pi[\nabla_\theta \log \pi_\theta(a|s)\, Q_{\pi_\theta}(s,a)]} - \boxed{\mathbb{E}_\pi[\nabla_\theta \log \pi_\theta(a|s)\, V_{\pi_\theta}(s)]}$$

Policy gradient with Q function

$$\mathbb{E}_\pi\left[\nabla_\theta \log \pi_\theta(a|s)\, V_{\pi_\theta}(s)\right] = 0$$

Unbiased estimate of the policy gradient for any baseline shift *b(s)* in reward. Here,

b(s) = V(s)

Proof

Baseline: Unbiased estimate

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_\pi[\nabla_\theta \log \pi_\theta(a|s)\,(Q_{\pi_\theta}(s,a) - V_{\pi_\theta}(s))] \\
&= \boxed{\mathbb{E}_\pi[\nabla_\theta \log \pi_\theta(a|s)\, Q_{\pi_\theta}(s,a)]} - \boxed{\mathbb{E}_\pi[\nabla_\theta \log \pi_\theta(a|s)\, V_{\pi_\theta}(s)]}
\end{aligned}
$$

Policy gradient with Q function

$$
\mathbb{E}_\pi\left[\nabla_\theta \log \pi_\theta(a|s)\, V_{\pi_\theta}(s)\right] = 0
$$

Unbiased estimate of the policy gradient for any baseline shift *b(s)* in reward. Here,

b(s) = V(s)

Proof

Solve HW4!

Advantage Actor Critic

Why??
Variance reduction
Is this a different Policy Gradient??
Unbiased estimate of policy gradient

Policy Gradient:

$$A_{\pi_\theta}(s,a)$$

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

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

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

Estimating the Advantage function

For the true value function $V^{\pi_\theta}(s)$ the TD error:

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

is an unbiased estimate of the advantage function:

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

Estimating the Advantage function

For the true value function $V^{\pi_\theta}(s)$ the TD error:

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

is an unbiased estimate of the advantage function:

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

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

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

Estimating the Advantage function

For the true value function $V^{\pi_\theta}(s)$ the TD error:

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

is an unbiased estimate of the advantage function:

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

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

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

In practice we can use an approximate TD error

$$\delta^{\pi_\theta} \approx r + \gamma \hat{V}_\emptyset(s') - \hat{V}_\emptyset(s)$$

Advantage Actor-Critic algorithm

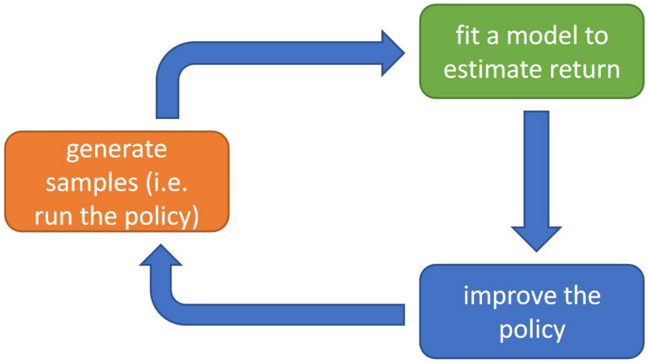
batch actor-critic algorithm:

1. sample $\{\mathbf{s}_i, \mathbf{a}_i\}$ from $\pi_\theta(\mathbf{a}|\mathbf{s})$ (run it on the robot)
2. fit $\hat{V}^\pi_\phi(\mathbf{s})$ to sampled reward sums
3. evaluate $\hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i) = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \hat{V}^\pi_\phi(\mathbf{s}'_i) - \hat{V}^\pi_\phi(\mathbf{s}_i)$
4. $\nabla_\theta J(\theta) \approx \sum_i \nabla_\theta \log \pi_\theta(\mathbf{a}_i|\mathbf{s}_i) \hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i)$
5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

online actor-critic algorithm:

1. take action $\mathbf{a} \sim \pi_\theta(\mathbf{a}|\mathbf{s})$, get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$
2. update $\hat{V}^\pi_\phi$ using target $r + \gamma \hat{V}^\pi_\phi(\mathbf{s}')$
3. evaluate $\hat{A}^\pi(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \hat{V}^\pi_\phi(\mathbf{s}') - \hat{V}^\pi_\phi(\mathbf{s})$
4. $\nabla_\theta J(\theta) \approx \nabla_\theta \log \pi_\theta(\mathbf{a}|\mathbf{s}) \hat{A}^\pi(\mathbf{s}, \mathbf{a})$
5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

fit $\hat{V}^\pi$



$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

$$y_{i,t} \approx r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \gamma \hat{V}^\pi_\phi(\mathbf{s}_{i,t+1})$$

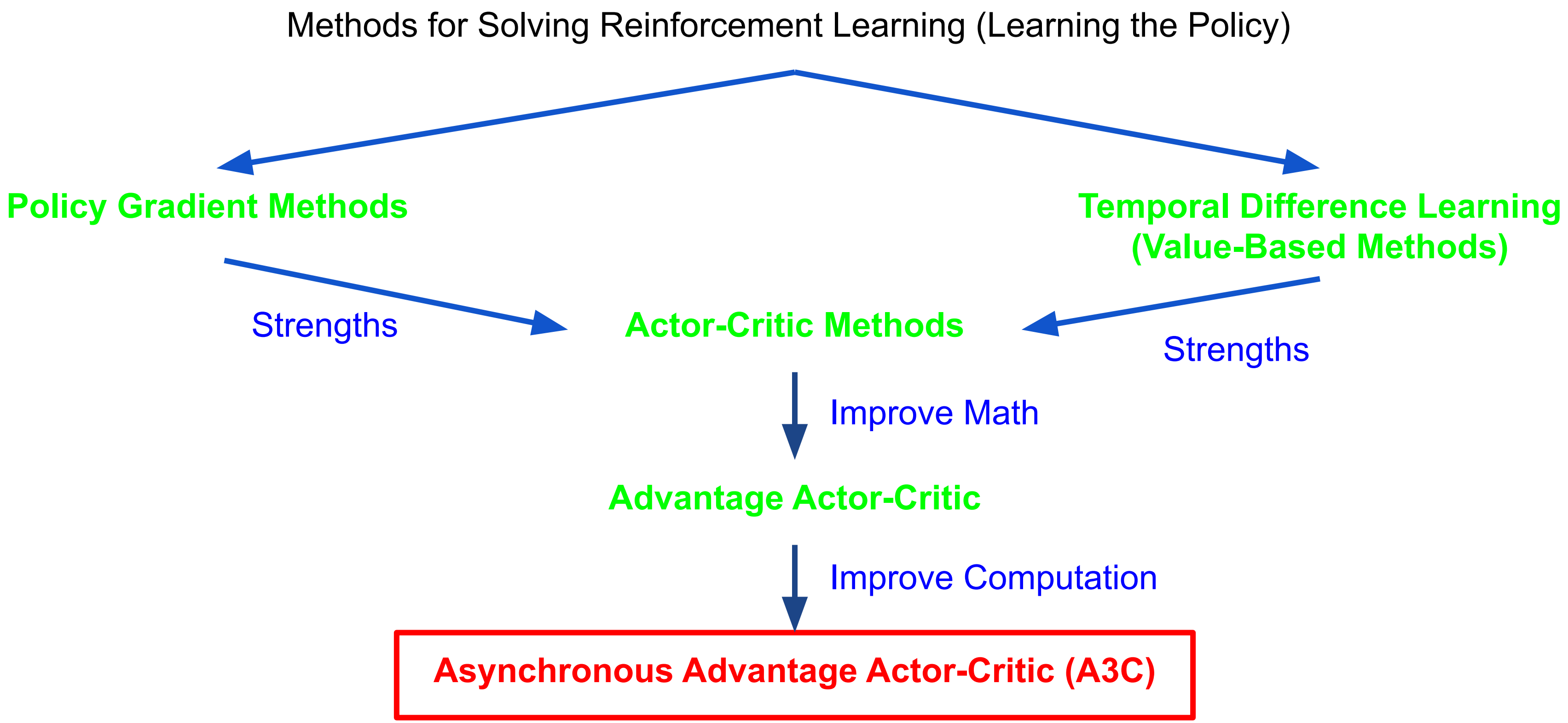
$$\mathcal{L}(\phi) = \frac{1}{2} \sum_i \left\| \hat{V}^\pi_\phi(\mathbf{s}_i) - y_i \right\|^2$$

Recap

$$\nabla_\theta J(\theta) = \mathrm{E}_{\pi_\theta}[\nabla_\theta log \pi_\theta(s,a) \boxed{R(s,a)}]$$

- $\sum_{t'=t}^{T} r(s_{i,t'}, a_{i,t'})$ Total Reward of Simulated Trajectories (Monte Carlo Rollout)
- $\sum_{t'=t}^{T} r(s_{i,t'}, a_{i,t'}) - b(s_t)$ Monte Carlo Rollout with Baseline
- $Q^\pi(s_t, a_t)$ State-Action Value Function
- $A^\pi(s_t, a_t)$ Advantage Function
- $r_t + \gamma V^\pi(s_{t+1}) - V^\pi(s_t)$ TD Residual

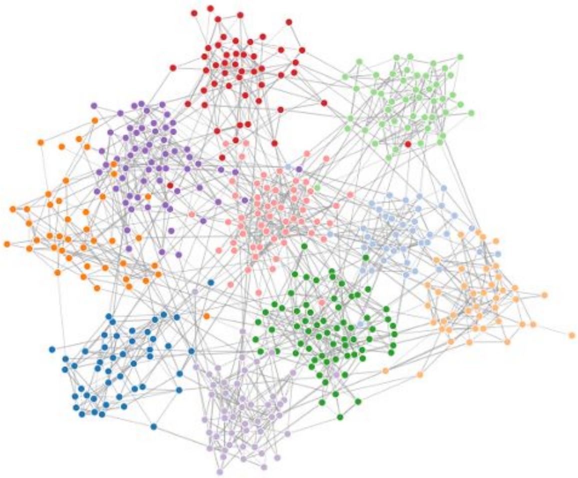
Roadmap


Methods for Solving Reinforcement Learning (Learning the Policy)
Policy Gradient Methods
Temporal Difference Learning
(Value-Based Methods)
Strengths
Actor-Critic Methods
Strengths
Improve Math
Advantage Actor-Critic
Improve Computation
Asynchronous Advantage Actor-Critic (A3C)

Asynchronous Advantage Actor-Critic

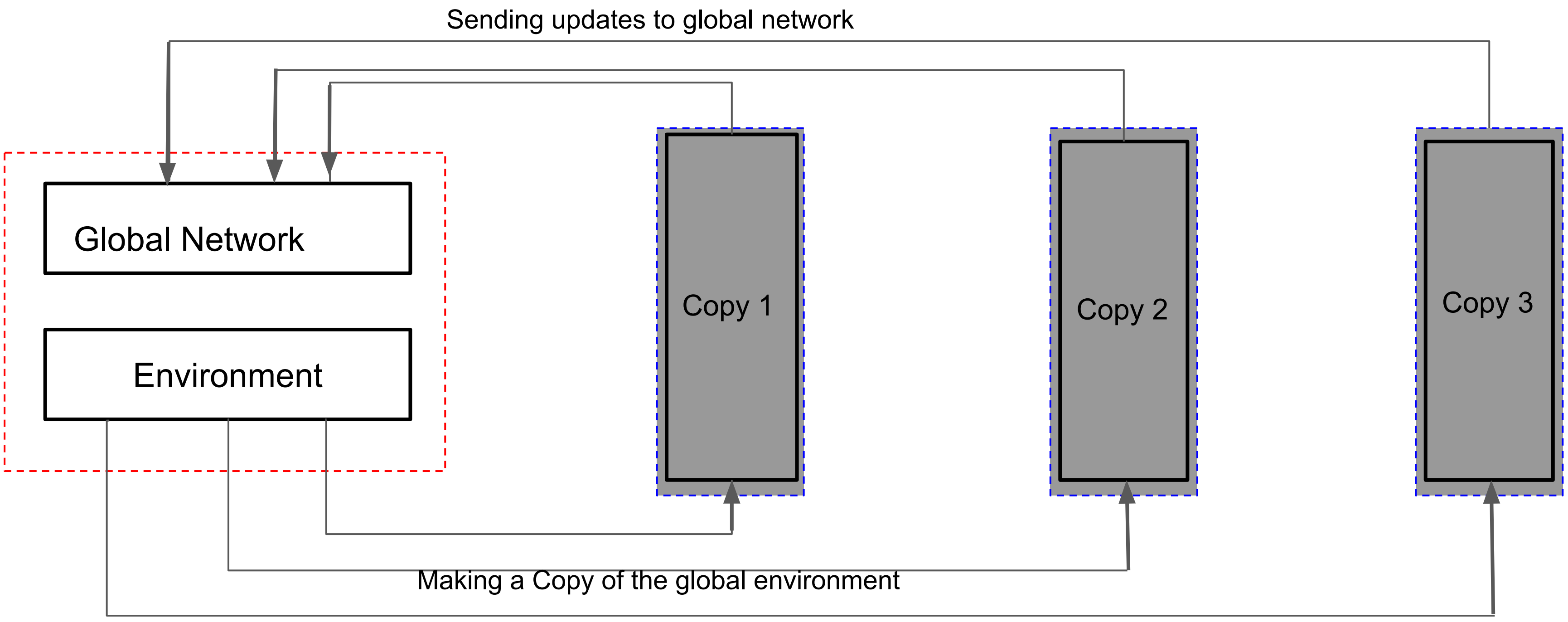
Why do we need Asynchronous Actor-Critic Methods - Breaking Sequential Correlation

- Learning from strongly correlated sequences (MDP) is inefficient.
- Heavily correlated sequences lead to strong bias in trained parameters.
- Sequential correlation makes learning process unstable



Asynchronous

- Parallel multi-agents with no coordination between them.



Asynchronous (No Coordination)

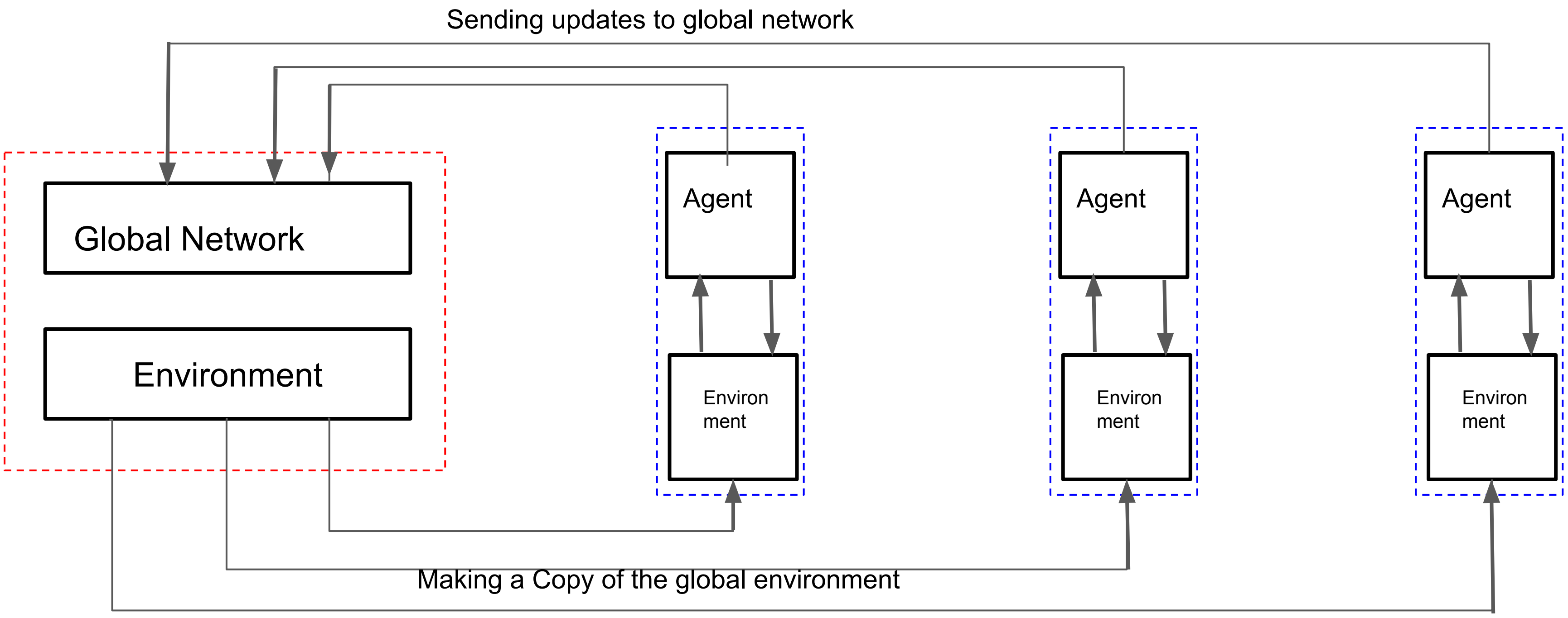
- Motivation:

  Find RL algorithms that can train deep neural network policies reliably (breaking sequential correlation) without large resources.

- Asynchronously execute parallel agents,each with its own copy of the environment.
- The gradients are asynchronously sent to global network (central copy of the model).

Asynchronous

- Parallel multi-agents with no coordination between them.

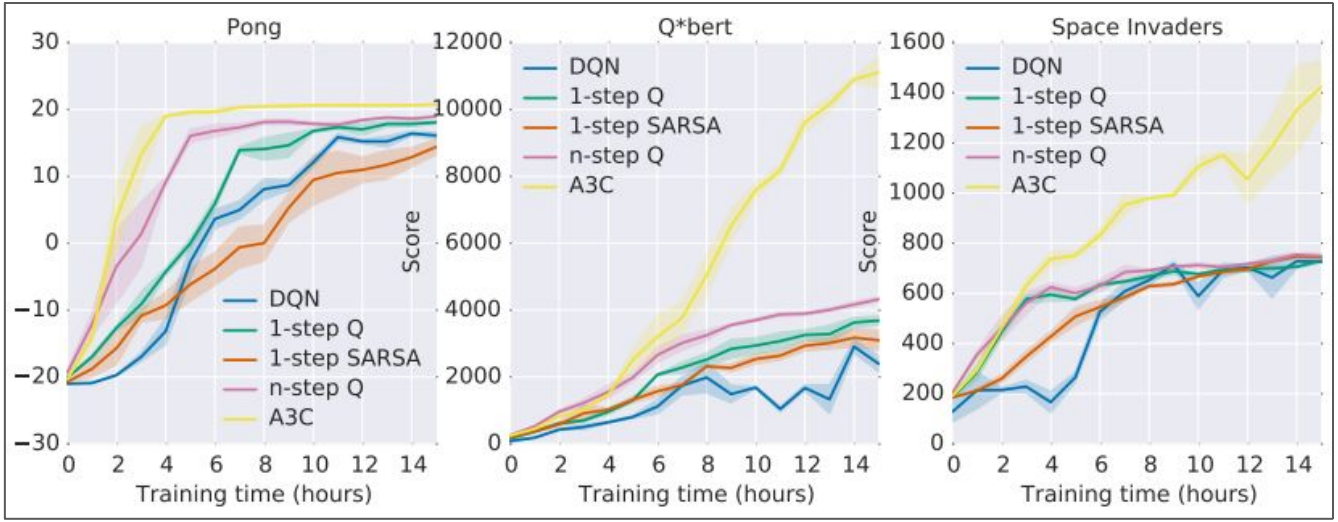


Advantages of Asynchronous

- Multiple actors applying asynchronous updates is less likely to be correlated than single agent.
- Can be applied to most standard reinforcement algorithms.
- They are able to train variety of networks stably.
- Works for both discrete as well as continuous methods.

Advantages of Asynchronous

- Multiple actors applying asynchronous updates is less likely to be correlated than single agent.
- Can be applied to most standard reinforcement algorithms.
- They are able to train variety of networks stably.
- Works for both discrete as well as continuous methods.



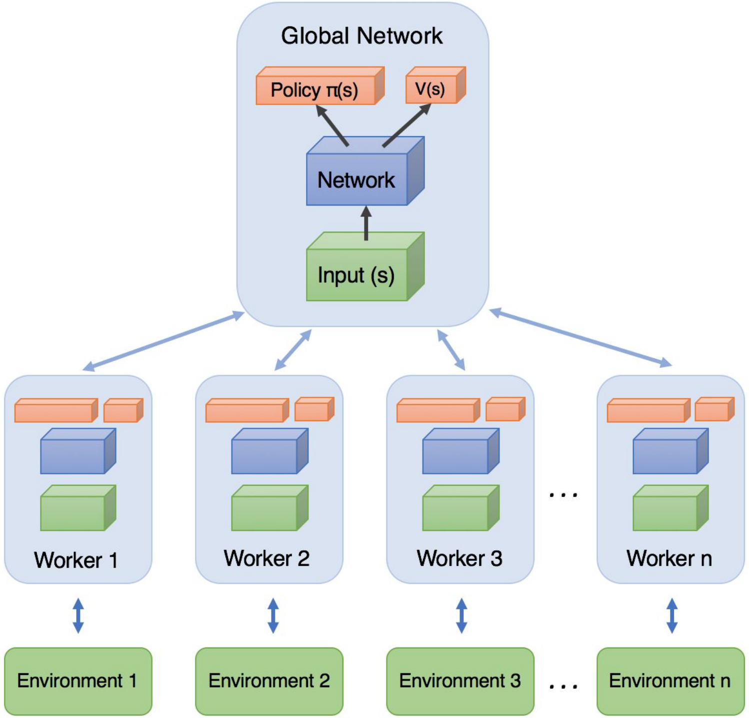
Asynchronous Advantage Actor-Critic

A3C (Asynchronous Advantage Actor-Critic)

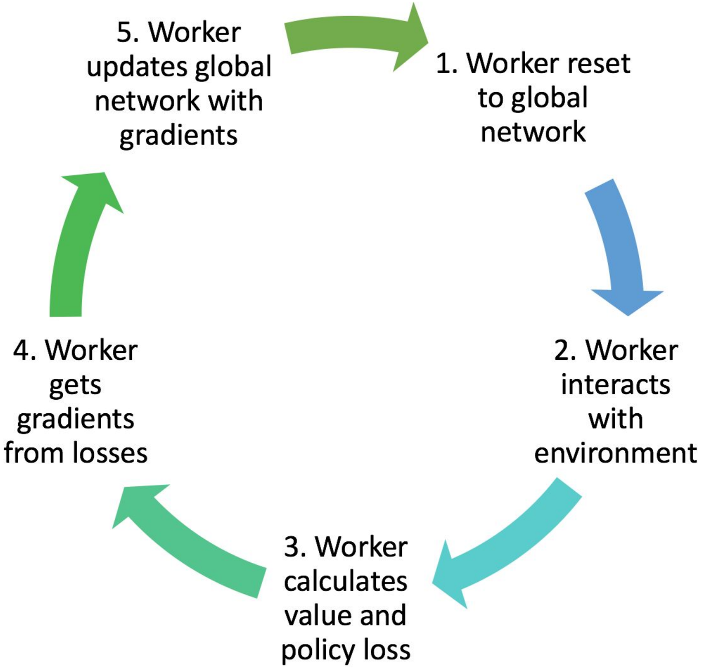
- Multiple actors are trained in **parallel**
- Each agent asynchronously updates the global network.
- Multiple CPU threads for seperate actor copies.

A3C (Asynchronous Advantage Actor-Critic)

- Multiple actors are trained in **parallel**
- Each agent asynchronously updates the global network.
- Multiple CPU threads for seperate actor copies.

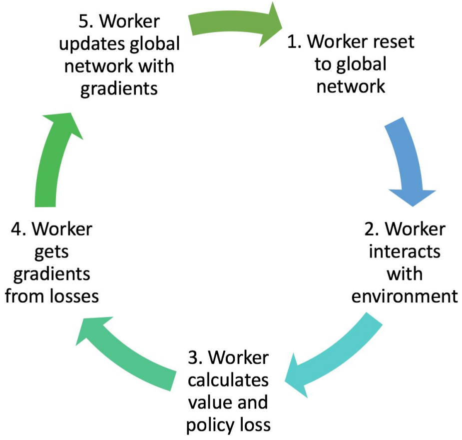


Algorithm outline



For Each Agent

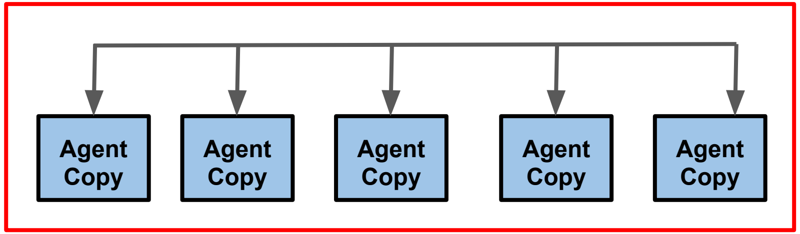
Algorithm outline



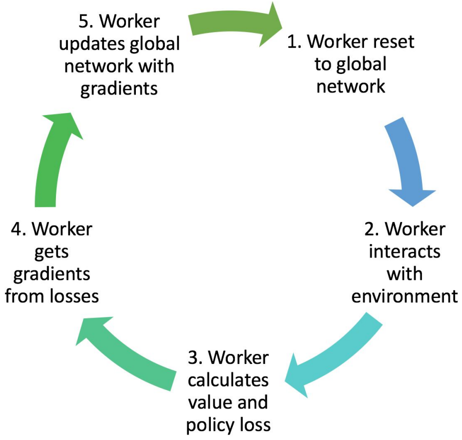
For Each Agent

Algorithm outline

#Assume global shared parameter vectors θ.
#Assume thread-specific parameter vectors θ'_k for each worker agent k.

1. Reset gradients: $d\theta \leftarrow 0$ and $d\theta'_k \leftarrow 0$
2. Duplicate global parameters to thread-specific parameters $\theta'_k = \theta$
3. 

4. $\theta \leftarrow \theta + \alpha \nabla_{\theta'_k} J(\theta'_k)$



For Each Agent

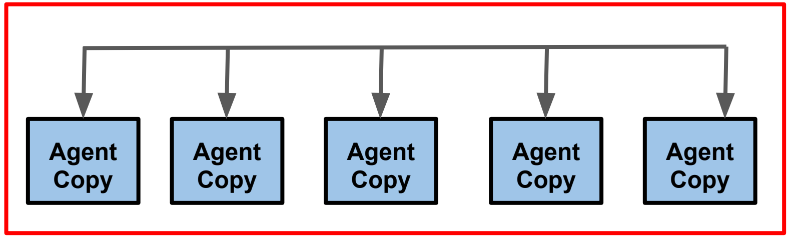
Algorithm outline

Agent Copy

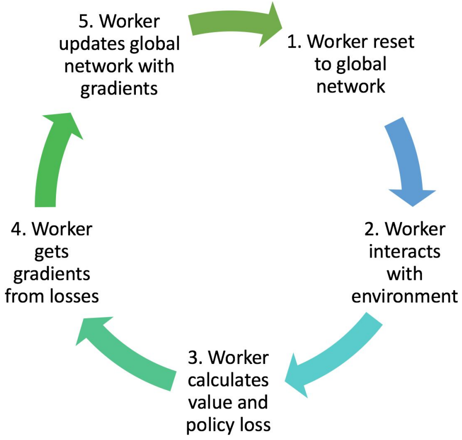
3. (a) sample $\{s_i, a_i\}$ from $\pi_{\theta'_k}(a|s)$ (run it on each agent)
   (b) fit $\hat{V}^{\pi}_{\Phi}(s)$ to sampled reward sums.
   (c) evaluate $\hat{A}^{\pi}(s_i, a_i) = r(s_i, a_i) + \gamma \hat{V}^{\pi}_{\Phi}(s'_i) - \hat{V}^{\pi}_{\Phi}(s_i)$
   (d) $\nabla_{\theta'_k} J(\theta'_k) \approx \Sigma_i \nabla_{\theta' + k} log \pi_{\theta'_k}(a_i|s_i) \hat{A}^{\pi}(s_i, a_i)$

#Assume global shared parameter vectors θ.
#Assume thread-specific parameter vectors θ'_k for each worker agent k.

1. Reset gradients: $d\theta \leftarrow 0$ and $d\theta'_k \leftarrow 0$
2. Duplicate global parameters to thread-specific parameters $\theta'_k = \theta$
3.



4. $\theta \leftarrow \theta + \alpha \nabla_{\theta'_k} J(\theta'_k)$

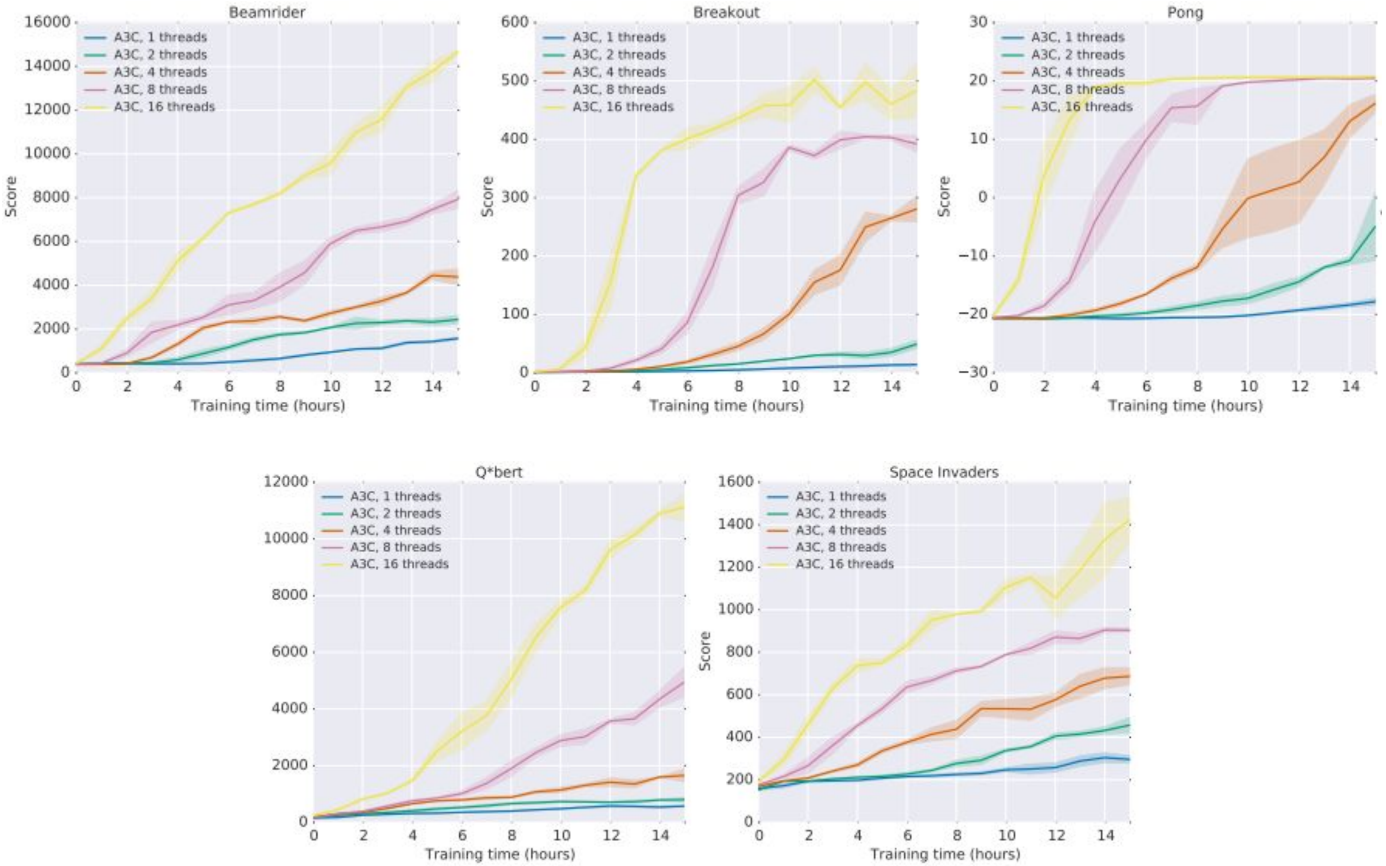


For Each Agent

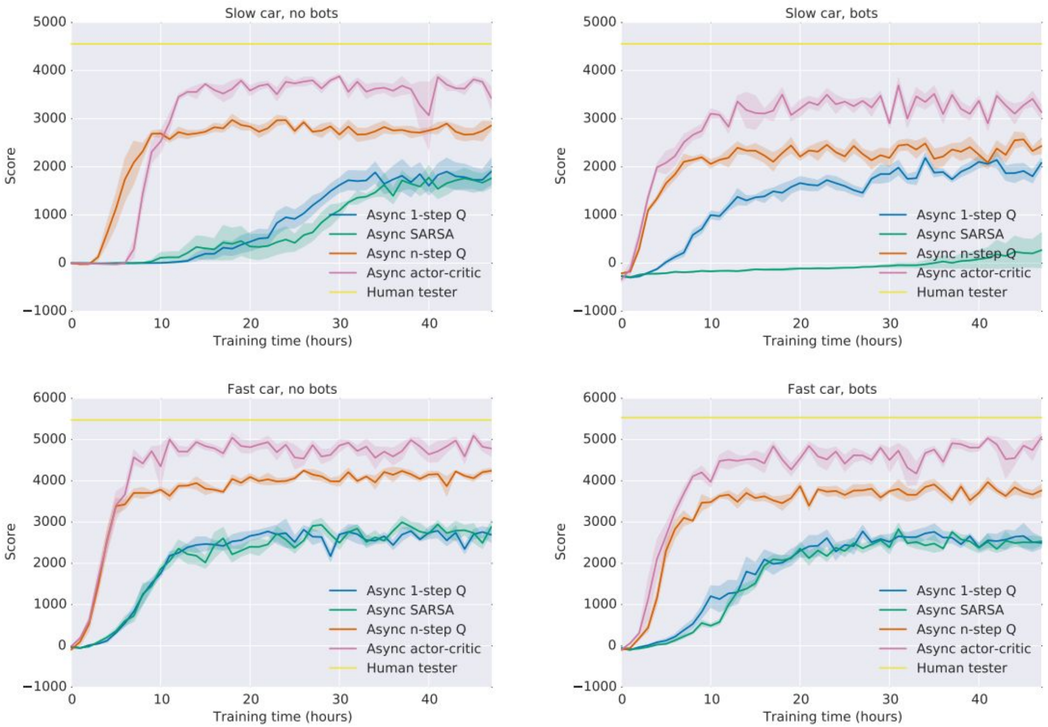
Advantages

- Stabilizes training (by breaking correlation).
- Using parallel actor-learners to update a shared model has a stabilizing effect.
- Multiple actors running in parallel explore different parts of the environment.
- Different exploration policies in each actor-learner can be explicitly implemented to maximize this diversity.
- Reduced training time (roughly linear in the number of parallel actor-learners).
- No GPU required. Multi-core CPU works.
- Eliminate the need of experience replay.

Reduced training time-linear in number of parallel actors



Performance of A3C



Asynchronous Methods for Deep Reinforcement Learning

Volodymyr Mnih[1] VMNIH@GOOGLE.COM
Adrià Puigdomènech Badia[1] ADRIAP@GOOGLE.COM
Mehdi Mirza[1,2] MIRZAMOM@IRO.UMONTREAL.CA
Alex Graves[1] GRAVESA@GOOGLE.COM
Tim Harley[1] THARLEY@GOOGLE.COM
Timothy P. Lillicrap[1] COUNTZERO@GOOGLE.COM
David Silver[1] DAVIDSILVER@GOOGLE.COM
Koray Kavukcuoglu[1] KORAYK@GOOGLE.COM

[1] Google DeepMind
[2] Montreal Institute for Learning Algorithms (MILA), University of Montreal

Abstract

We propose a conceptually simple and lightweight framework for deep reinforcement learning that uses asynchronous gradient descent for optimization of deep neural network controllers. We present asynchronous variants of four standard reinforcement learning algorithms and show that parallel actor-learners have a stabilizing effect on training allowing all four methods to successfully train neural network controllers. The best performing method, an asynchronous variant of actor-critic, surpasses the current state-of-the-art on the Atari domain while training for half the time on a single multi-core CPU instead of a GPU. Furthermore, we show that asynchronous actor-critic succeeds on a wide variety of continuous motor control problems as well as on a new task of navigating random 3D mazes using a visual input.

1. Introduction

Deep neural networks provide rich representations that can enable reinforcement learning (RL) algorithms to perform effectively. However, it was previously thought that the combination of simple online RL algorithms with deep neural networks was fundamentally unstable. Instead, a variety of solutions have been proposed to stabilize the algorithm (Riedmiller, 2005; Mnih et al., 2013; 2015; Van Hasselt et al., 2015; Schulman et al., 2015a). These approaches share a common idea: the sequence of observed data encountered by an online RL agent is non-stationary, and online RL updates are strongly correlated. By storing the agent's data in an experience replay memory, the data can be batched (Riedmiller, 2005; Schulman et al., 2015a) or randomly sampled (Mnih et al., 2013; 2015; Van Hasselt et al., 2015) from different time-steps. Aggregating over memory in this way reduces non-stationarity and decorrelates updates, but at the same time limits the methods to off-policy reinforcement learning algorithms.

Deep RL algorithms based on experience replay have achieved unprecedented success in challenging domains such as Atari 2600. However, experience replay has several drawbacks: it uses more memory and computation per real interaction; and it requires off-policy learning algorithms that can update from data generated by an older policy.

In this paper we provide a very different paradigm for deep reinforcement learning. Instead of experience replay, we asynchronously execute multiple agents in parallel, on multiple instances of the environment. This parallelism also decorrelates the agents' data into a more stationary process, since at any given time-step the parallel agents will be experiencing a variety of different states. This simple idea enables a much larger spectrum of fundamental on-policy RL algorithms, such as Sarsa, n-step methods, and actor-critic methods, as well as off-policy RL algorithms such as Q-learning, to be applied robustly and effectively using deep neural networks.

Our parallel reinforcement learning paradigm also offers practical benefits. Whereas previous approaches to deep reinforcement learning rely heavily on specialized hardware such as GPUs (Mnih et al., 2015; Van Hasselt et al., 2015; Schaul et al., 2015) or massively distributed architectures (Nair et al., 2015), our experiments run on a single machine with a standard multi-core CPU. When applied to a variety of Atari 2600 domains, on many games asynchronous reinforcement learning achieves better results, in far less

Proceedings of the 33rd International Conference on Machine Learning, New York, NY, USA, 2016. JMLR: W&CP volume 48.

Asynchronous Methods for Deep Reinforcement Learning

Volodymyr Mnih[1] VMNIH@GOOGLE.COM
Adrià Puigdomènech Badia[1] ADRIAP@GOOGLE.COM
Mehdi Mirza[1,2] MIRZAMOM@IRO.UMONTREAL.CA
Alex Graves[1] GRAVESA@GOOGLE.COM
Tim Harley[1] THARLEY@GOOGLE.COM
Timothy P. Lillicrap[1] COUNTZERO@GOOGLE.COM
David Silver[1] DAVIDSILVER@GOOGLE.COM
Koray Kavukcuoglu[1] KORAYK@GOOGLE.COM

[1] Google DeepMind
[2] Montreal Institute for Learning Algorithms (MILA), University of Montreal

Abstract

We propose a conceptually simple and lightweight framework for deep reinforcement learning that uses asynchronous gradient descent for optimization of deep neural network controllers. We present asynchronous variants of four standard reinforcement learning algorithms and show that parallel actor-learners have a stabilizing effect on training allowing all four methods to successfully train neural network controllers. The best performing method, an asynchronous variant of actor-critic, surpasses the current state-of-the-art on the Atari domain while training for half the time on a single multi-core CPU instead of a GPU. Furthermore, we show that asynchronous actor-critic succeeds on a wide variety of continuous motor control problems as well as on a new task of navigating random 3D mazes using a visual input.

1. Introduction

Deep neural networks provide rich representations that can enable reinforcement learning (RL) algorithms to perform effectively. However, it was previously thought that the combination of simple online RL algorithms with deep neural networks was fundamentally unstable. Instead, a variety of solutions have been proposed to stabilize the algorithm (Riedmiller, 2005; Mnih et al., 2013; 2015; Van Hasselt et al., 2015; Schulman et al., 2015a). These approaches share a common idea: the sequence of observed data encountered by an online RL agent is non-stationary, and online RL updates are strongly correlated. By storing the agent's data in an experience replay memory, the data can be batched (Riedmiller, 2005; Schulman et al., 2015a) or randomly sampled (Mnih et al., 2013; 2015; Van Hasselt et al., 2015) from different time-steps. Aggregating over memory in this way reduces non-stationarity and decorrelates updates, but at the same time limits the methods to off-policy reinforcement learning algorithms.

Deep RL algorithms based on experience replay have achieved unprecedented success in challenging domains such as Atari 2600. However, experience replay has several drawbacks: it uses more memory and computation per real interaction; and it requires off-policy learning algorithms that can update from data generated by an older policy.

In this paper we provide a very different paradigm for deep reinforcement learning. Instead of experience replay, we asynchronously execute multiple agents in parallel, on multiple instances of the environment. This parallelism also decorrelates the agents' data into a more stationary process, since at any given time-step the parallel agents will be experiencing a variety of different states. This simple idea enables a much larger spectrum of fundamental on-policy RL algorithms, such as Sarsa, n-step methods, and actor-critic methods, as well as off-policy RL algorithms such as Q-learning, to be applied robustly and effectively using deep neural networks.

Our parallel reinforcement learning paradigm also offers practical benefits. Whereas previous approaches to deep reinforcement learning rely heavily on specialized hardware such as GPUs (Mnih et al., 2015; Van Hasselt et al., 2015; Schaul et al., 2015) or massively distributed architectures (Nair et al., 2015), our experiments run on a single machine with a standard multi-core CPU. When applied to a variety of Atari 2600 domains, on many games asynchronous reinforcement learning achieves better results, in far less

Proceedings of the 33rd International Conference on Machine Learning, New York, NY, USA, 2016. JMLR: W&CP volume 48.

6. Conclusions and Discussion

We have presented asynchronous versions of four standard reinforcement learning algorithms and showed that they are able to train neural network controllers on a variety of domains in a stable manner. Our results show that in our proposed framework stable training of neural networks through reinforcement learning is possible with both value-based and policy-based methods, off-policy as well as on-policy methods, and in discrete as well as continuous domains. When trained on the Atari domain using 16 CPU cores, the proposed asynchronous algorithms train faster than DQN trained on an Nvidia K40 GPU, with A3C surpassing the current state-of-the-art in half the training time.

One of our main findings is that using parallel actor-learners to update a shared model had a stabilizing effect on the learning process of the three value-based methods we considered. While this shows that stable online Q-learning is possible without experience replay, which was used for this purpose in DQN, it does not mean that experience replay is not useful. Incorporating experience replay into the asynchronous reinforcement learning framework could

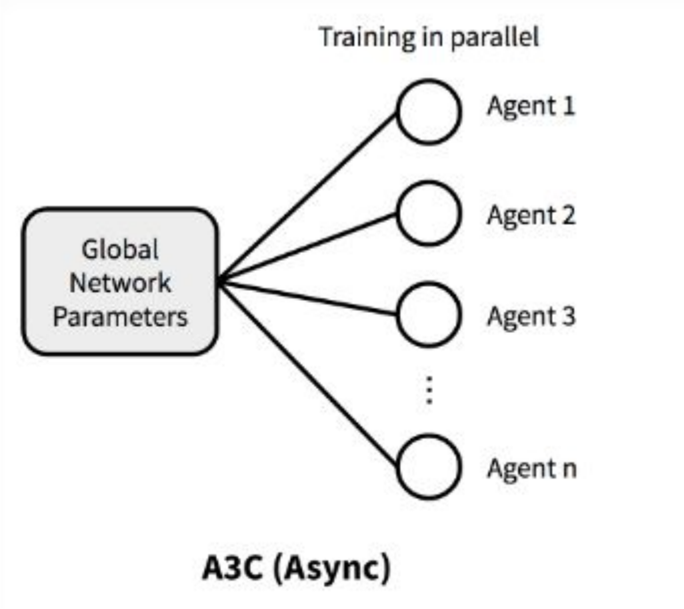
Limitations of A3C

- Since updates are asynchronous, every update may not be necessarily an optimal update towards convergence.

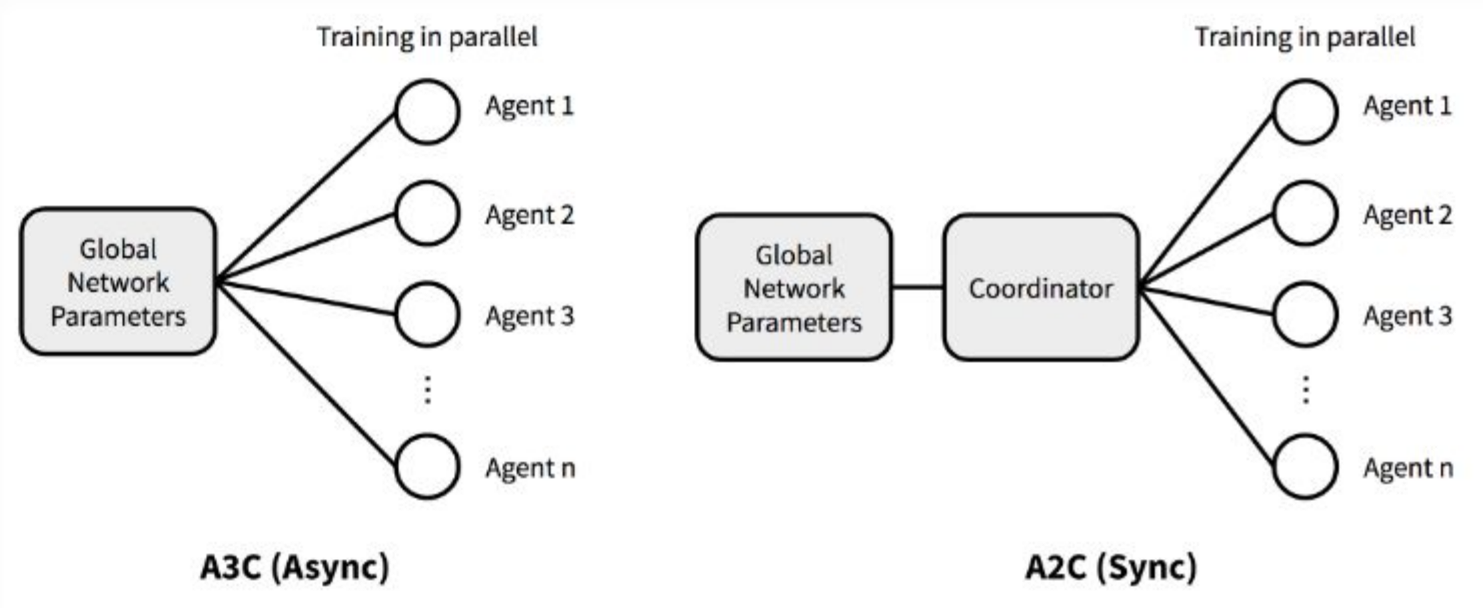
  [Each agent updates the global network while other agents are still working with the outdated copy of environment]

- The issue of choosing right features for the actor and critic network remains a challenge.

A3C vs A2C


Training in parallel
Global
Network
Parameters
Agent 1
Agent 2
Agent 3
Agent n
A3C (Async)

A3C vs A2C


Training in parallel
Agent 1
Agent 2
Agent 3
Agent n
Global Network Parameters
A3C (Async)
Training in parallel
Agent 1
Agent 2
Agent 3
Agent n
Global Network Parameters
Coordinator
A2C (Sync)

References

- Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., … Kavukcuoglu, K. (2016). Asynchronous Methods for Deep Reinforcement Learning. Retrieved from http://arxiv.org/abs/1602.0178
- Actor-Critic Algorithms, CS 294-112: Deep Reinforcement Learning Sergey Levine. Retrieve from http://rail.eecs.berkeley.edu/deeprlcourse-fa17/f17docs/lecture_5_actor_critic_pdf.pdf
- Lecture 7: Policy Gradient, David Silver http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/pg.pdf
- Lecture 6: Value Function Approximation, David Silver http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/FA.pdf
- Policy Gradient Algorithms, Lilian Weng https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html
- Richard S. Sutton and Andrew G. Barto. Reinforcement Learning: An Introduction; 2nd Edition. 2017.
- An Introduction to Policy Gradients with Doom https://medium.freecodecamp.org/an-introduction-to-policy-gradients-with-cartpole-and-doom-495b5ef2207f
- 10703 (Spring 2018): Deep RL and Control http://www.cs.cmu.edu/~rsalakhu/10703/