Policy-based and Actor-Critic Deep Reinforcement Learning Methods

CS 4641 B: Machine Learning (Summer 2020)

Miguel Morales

07/15/2020

Admin

- Homework 4 has been released. It covers the two previous lectures.
- Instructions for the Project presentation have been released. Check Piazza for details.
- Make sure to contribute to Piazza/Lecture discussions.
- This is the final lecture.
- Grading of HW1, HW2, Project Proposal is complete.
- HW3 grading to be completed soon.
- Project presentations moved by 1 week.
  - Presentation due the 27th, report the 29th.
- No office hours, but make sure to create a Private Piazza post if you need to reach out to me.
- Course website has been updated to reflect some of these: https://mimoralea.github.io/cs4641B-summer2020/

Outline

Policy-based Methods

Actor-Critic Methods

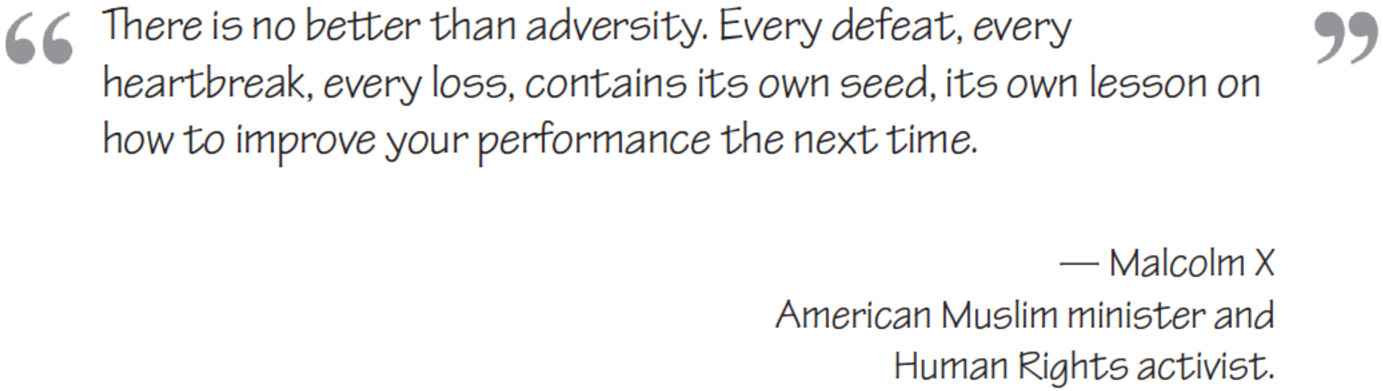
Advanced DRL Topics

Outline

Policy-based Methods

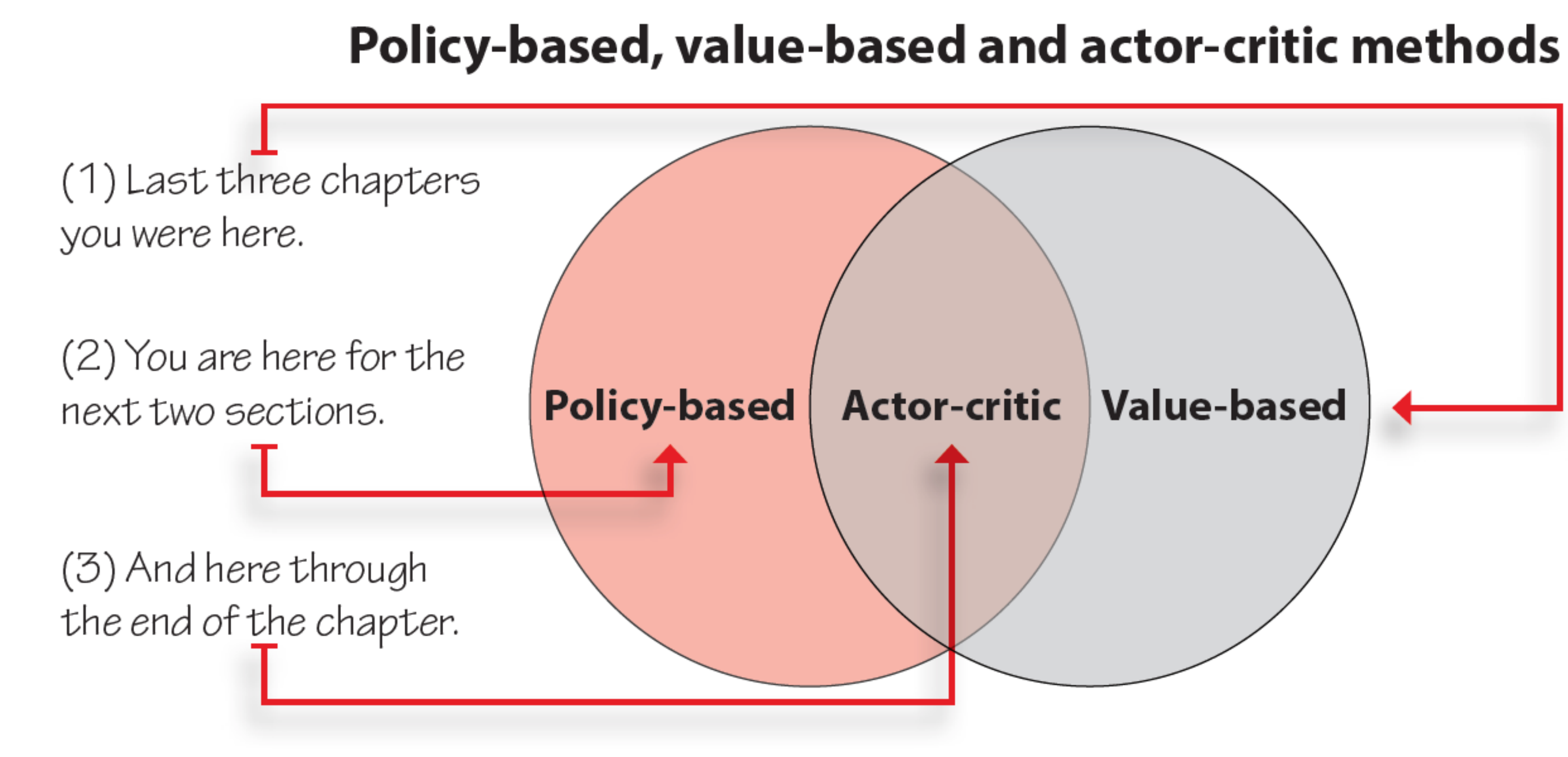
Actor-Critic Methods

Advanced DRL Topics



> “There is no better than adversity. Every defeat, every heartbreak, every loss, contains its own seed, its own lesson on how to improve your performance the next time.”
>
> — Malcolm X
> American Muslim minister and
> Human Rights activist.

Policy-based Methods



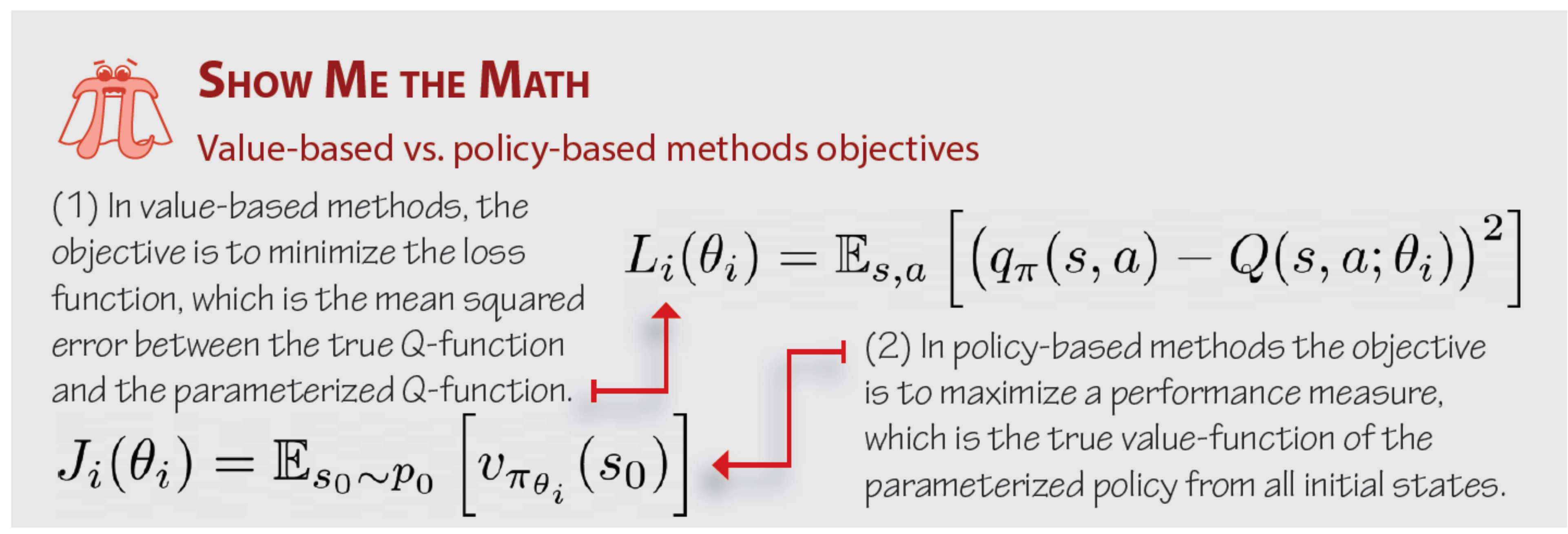
Policy-based, value-based and actor-critic methods

(1) Last three chapters you were here.

(2) You are here for the next two sections.

(3) And here through the end of the chapter.

Value-based vs. Policy-based objectives



Show Me the Math

Value-based vs. policy-based methods objectives

(1) In value-based methods, the objective is to minimize the loss function, which is the mean squared error between the true Q-function and the parameterized Q-function.

$$L_i(\theta_i) = \mathbb{E}_{s,a}\left[\left(q_\pi(s,a) - Q(s,a;\theta_i)\right)^2\right]$$

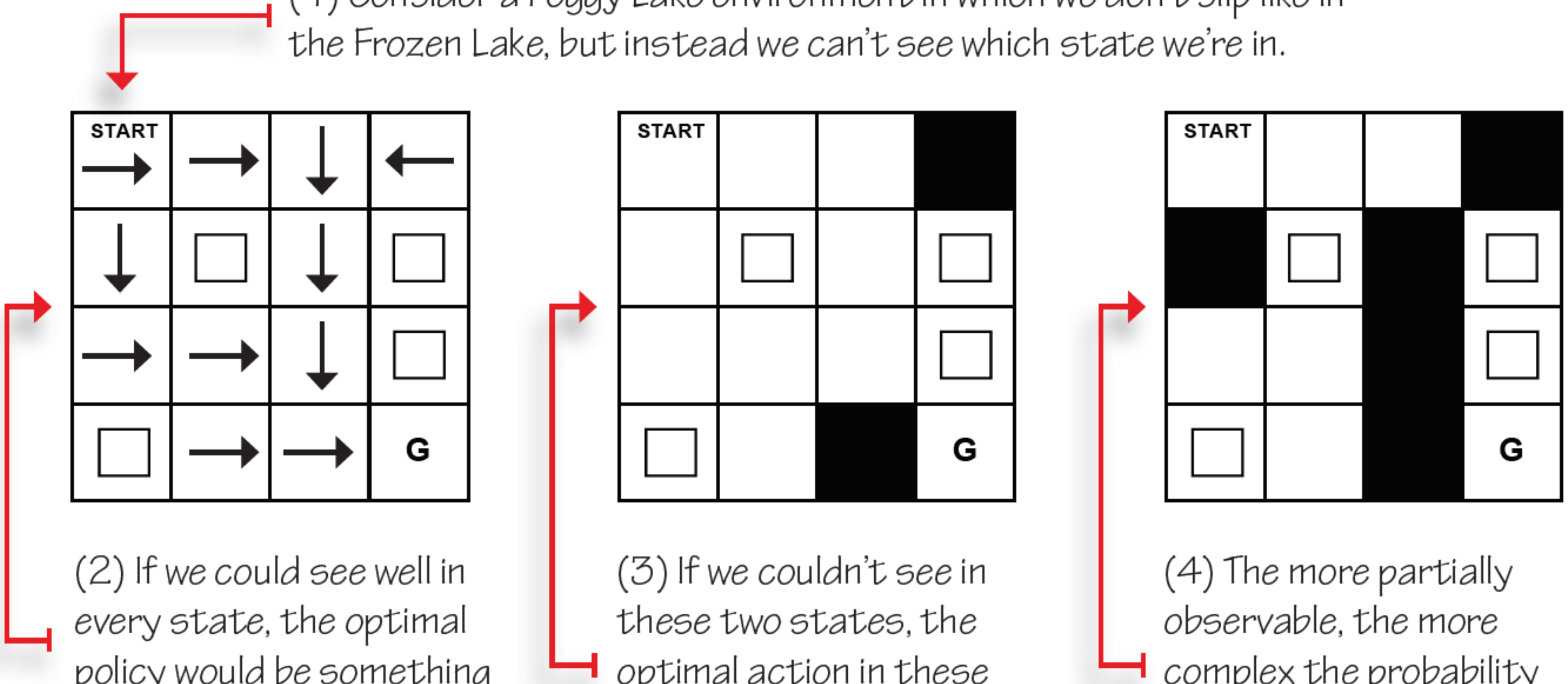
$$J_i(\theta_i) = \mathbb{E}_{s_0 \sim p_0}\left[v_{\pi_{\theta_i}}(s_0)\right]$$

(2) In policy-based methods the objective is to maximize a performance measure, which is the true value-function of the parameterized policy from all initial states.

Policy-based methods motivation

Learning stochastic policies could get us out of trouble

(1) Consider a Foggy Lake environment in which we don't slip like in the Frozen Lake, but instead we can't see which state we're in.

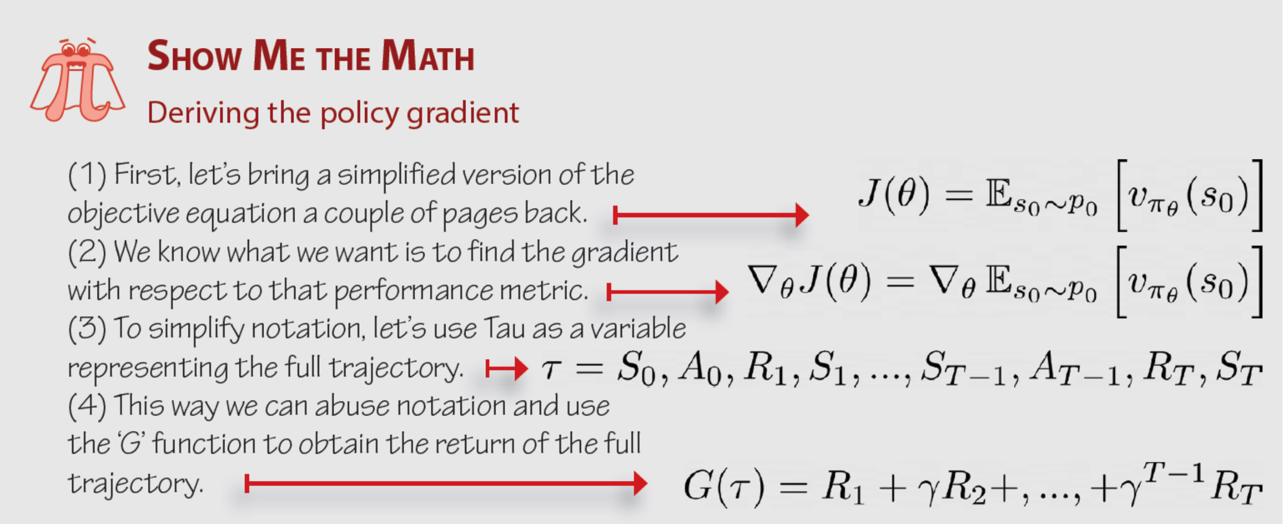


(2) If we could see well in every state, the optimal policy would be something like this.

(3) If we couldn't see in these two states, the optimal action in these states would be something like 50% left and %50 right.

(4) The more partially observable, the more complex the probability distribution to learn for optimal action selection.

Deriving the policy gradient



Show Me the Math

Deriving the policy gradient

(1) First, let's bring a simplified version of the objective equation a couple of pages back.

$$J(\theta) = \mathbb{E}_{s_0\sim p_0}\left[v_{\pi_\theta}(s_0)\right]$$

(2) We know what we want is to find the gradient with respect to that performance metric.

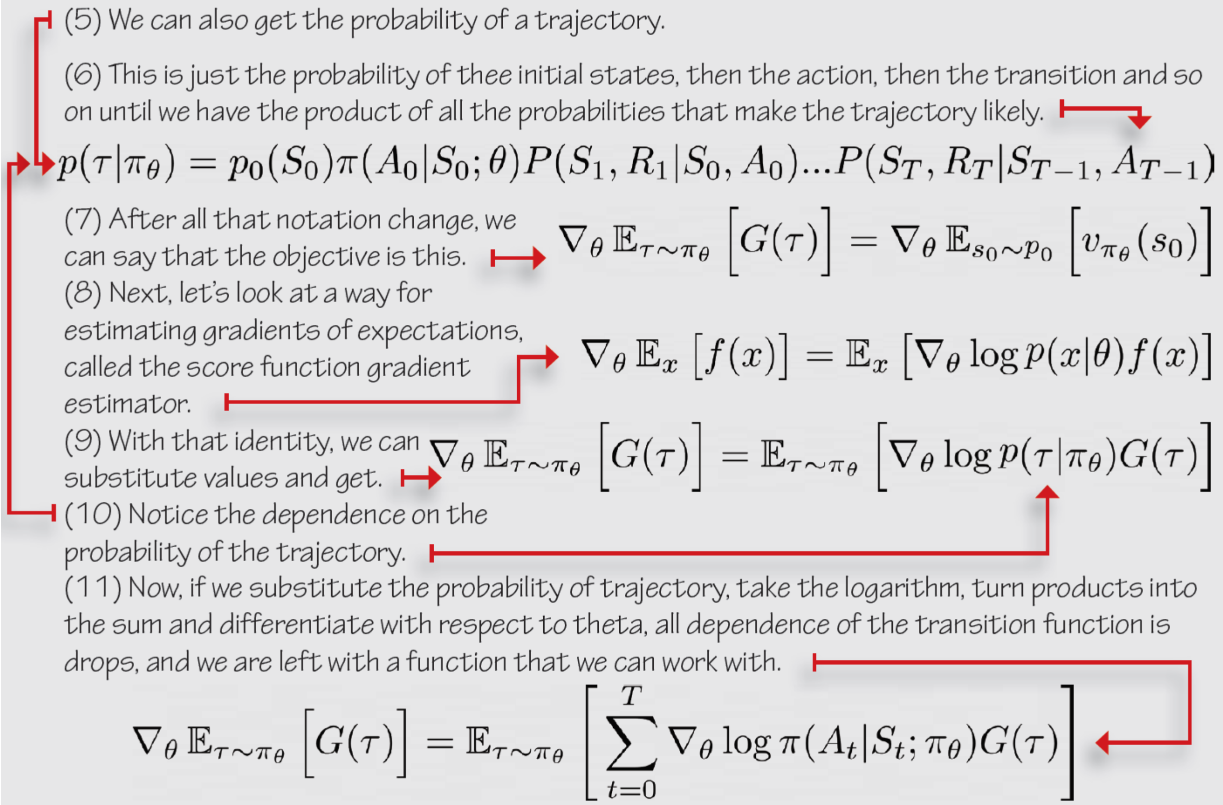
$$\nabla_\theta J(\theta) = \nabla_\theta\, \mathbb{E}_{s_0\sim p_0}\left[v_{\pi_\theta}(s_0)\right]$$

(3) To simplify notation, let's use Tau as a variable representing the full trajectory.

$$\tau = S_0, A_0, R_1, S_1, \ldots, S_{T-1}, A_{T-1}, R_T, S_T$$

(4) This way we can abuse notation and use the 'G' function to obtain the return of the full trajectory.

$$G(\tau) = R_1 + \gamma R_2 +, \ldots, +\gamma^{T-1} R_T$$



(5) We can also get the probability of a trajectory.

(6) This is just the probability of thee initial states, then the action, then the transition and so on until we have the product of all the probabilities that make the trajectory likely.

$$p(\tau|\pi_\theta) = p_0(S_0)\pi(A_0|S_0;\theta)P(S_1, R_1|S_0, A_0)\ldots P(S_T, R_T|S_{T-1}, A_{T-1})$$

(7) After all that notation change, we can say that the objective is this.

$$\nabla_\theta\, \mathbb{E}_{\tau\sim\pi_\theta}\left[G(\tau)\right] = \nabla_\theta\, \mathbb{E}_{s_0\sim p_0}\left[v_{\pi_\theta}(s_0)\right]$$

(8) Next, let's look at a way for estimating gradients of expectations, called the score function gradient estimator.

$$\nabla_\theta\, \mathbb{E}_x\left[f(x)\right] = \mathbb{E}_x\left[\nabla_\theta \log p(x|\theta) f(x)\right]$$

(9) With that identity, we can substitute values and get.

$$\nabla_\theta\, \mathbb{E}_{\tau\sim\pi_\theta}\left[G(\tau)\right] = \mathbb{E}_{\tau\sim\pi_\theta}\left[\nabla_\theta \log p(\tau|\pi_\theta) G(\tau)\right]$$

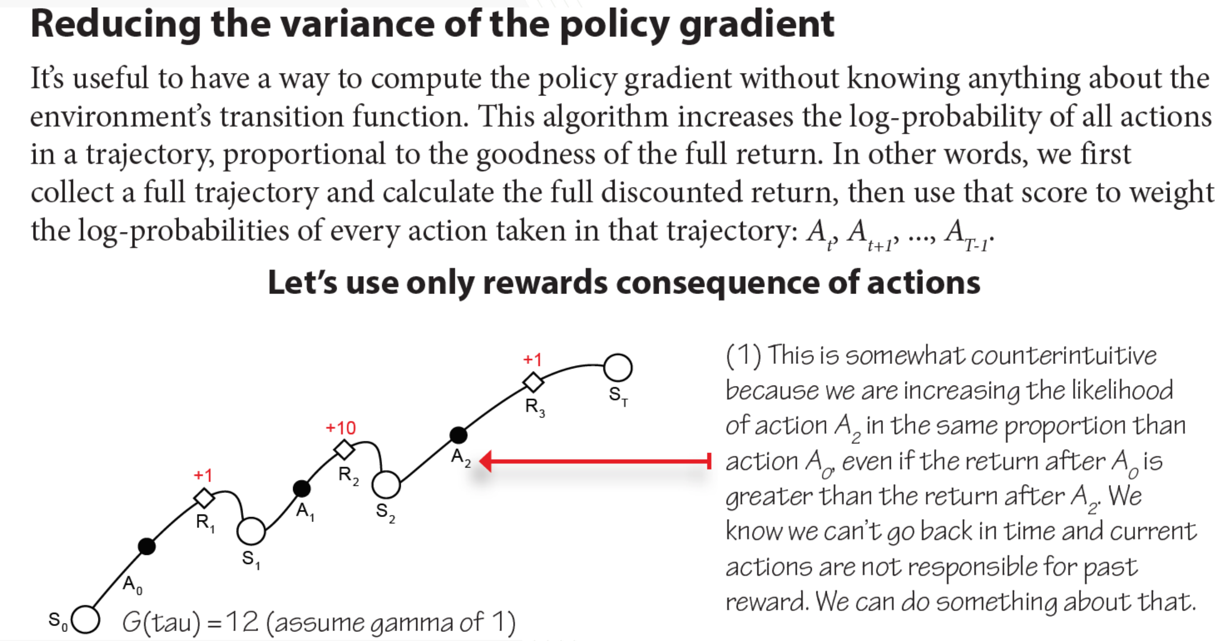
(10) Notice the dependence on the probability of the trajectory.

(11) Now, if we substitute the probability of trajectory, take the logarithm, turn products into the sum and differentiate with respect to theta, all dependence of the transition function is drops, and we are left with a function that we can work with.

$$\nabla_\theta\, \mathbb{E}_{\tau\sim\pi_\theta}\left[G(\tau)\right] = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi(A_t|S_t;\pi_\theta) G(\tau)\right]$$

http://blog.shakirm.com/2015/11/machine-learning-trick-of-the-day-5-log-derivative-trick/

Using future rewards only



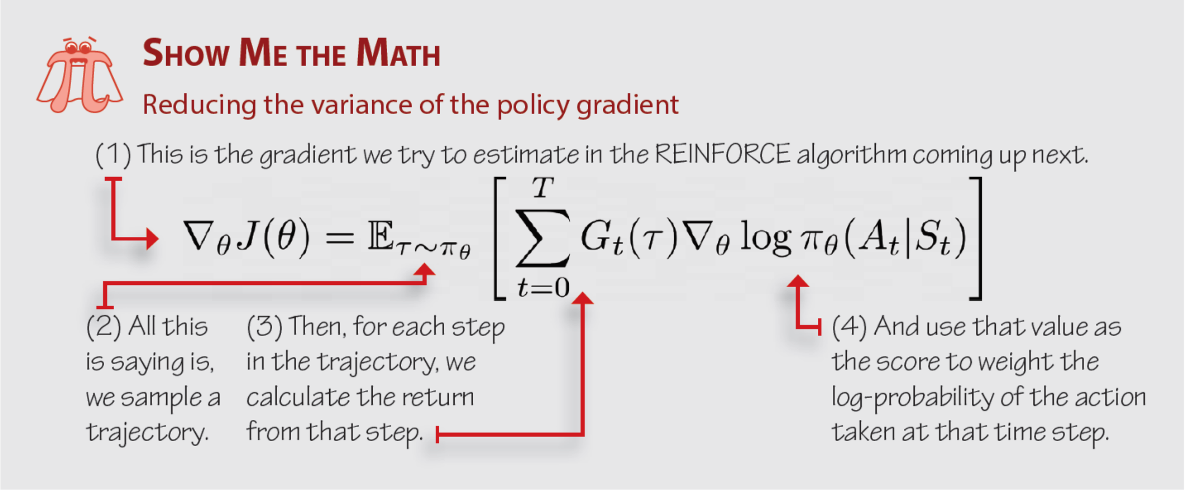
Reducing the variance of the policy gradient

It's useful to have a way to compute the policy gradient without knowing anything about the environment's transition function. This algorithm increases the log-probability of all actions in a trajectory, proportional to the goodness of the full return. In other words, we first collect a full trajectory and calculate the full discounted return, then use that score to weight the log-probabilities of every action taken in that trajectory: A_t, A_{t+1}, ..., A_{T-1}.

Let's use only rewards consequence of actions

G(tau) = 12 (assume gamma of 1)

(1) This is somewhat counterintuitive because we are increasing the likelihood of action A_2 in the same proportion than action A_0, even if the return after A_0 is greater than the return after A_2. We know we can't go back in time and current actions are not responsible for past reward. We can do something about that.



Show Me the Math

Reducing the variance of the policy gradient

(1) This is the gradient we try to estimate in the REINFORCE algorithm coming up next.

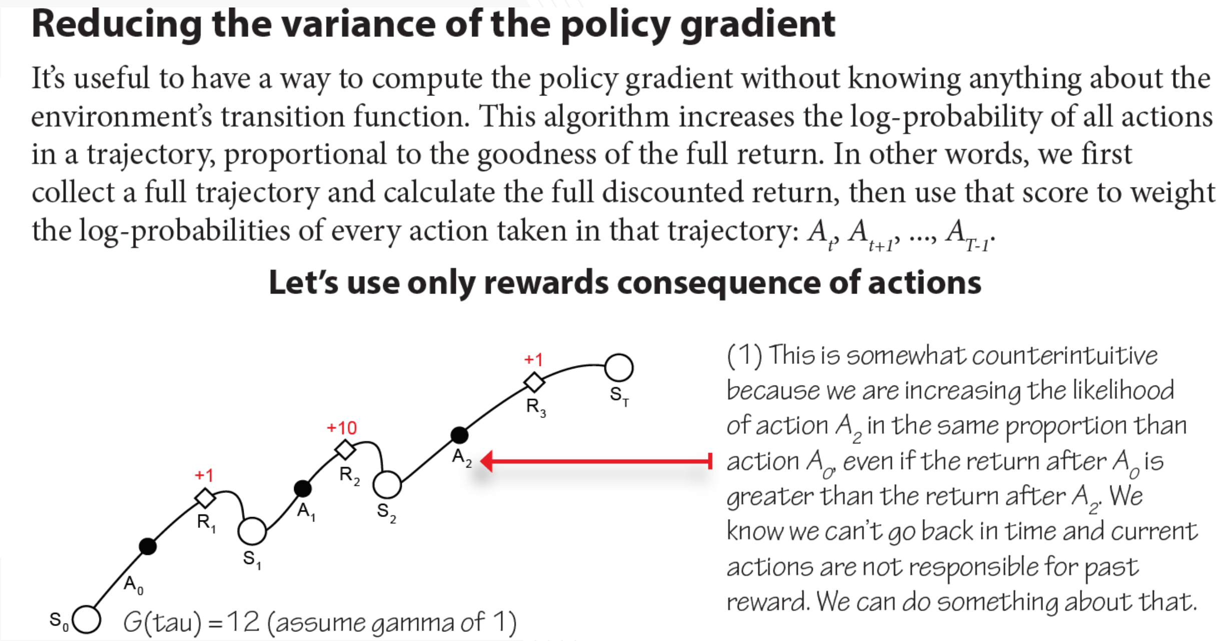
$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} G_t(\tau)\nabla_\theta \log \pi_\theta(A_t|S_t)\right]$$

(2) All this is saying is, we sample a trajectory.

(3) Then, for each step in the trajectory, we calculate the return from that step.

(4) And use that value as the score to weight the log-probability of the action taken at that time step.

Using future rewards only



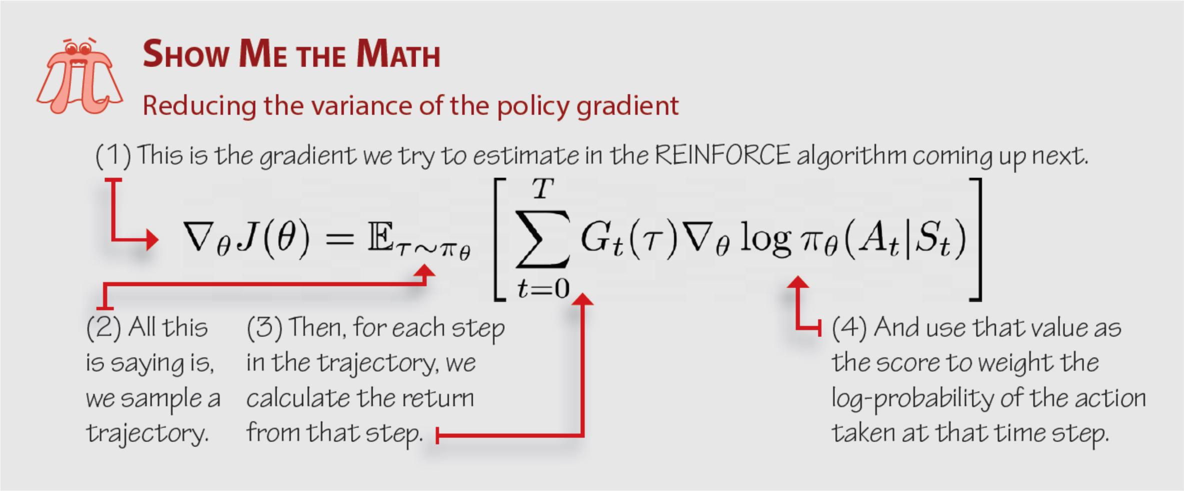
Reducing the variance of the policy gradient

It's useful to have a way to compute the policy gradient without knowing anything about the environment's transition function. This algorithm increases the log-probability of all actions in a trajectory, proportional to the goodness of the full return. In other words, we first collect a full trajectory and calculate the full discounted return, then use that score to weight the log-probabilities of every action taken in that trajectory: A_t, A_{t+1}, ..., A_{T-1}.

Let's use only rewards consequence of actions

$G(\text{tau}) = 12$ (assume gamma of 1)

(1) This is somewhat counterintuitive because we are increasing the likelihood of action A_2 in the same proportion than action A_0, even if the return after A_0 is greater than the return after A_2. We know we can't go back in time and current actions are not responsible for past reward. We can do something about that.



Show Me the Math

Reducing the variance of the policy gradient

(1) This is the gradient we try to estimate in the REINFORCE algorithm coming up next.

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} G_t(\tau)\nabla_\theta \log \pi_\theta(A_t|S_t)\right]$$

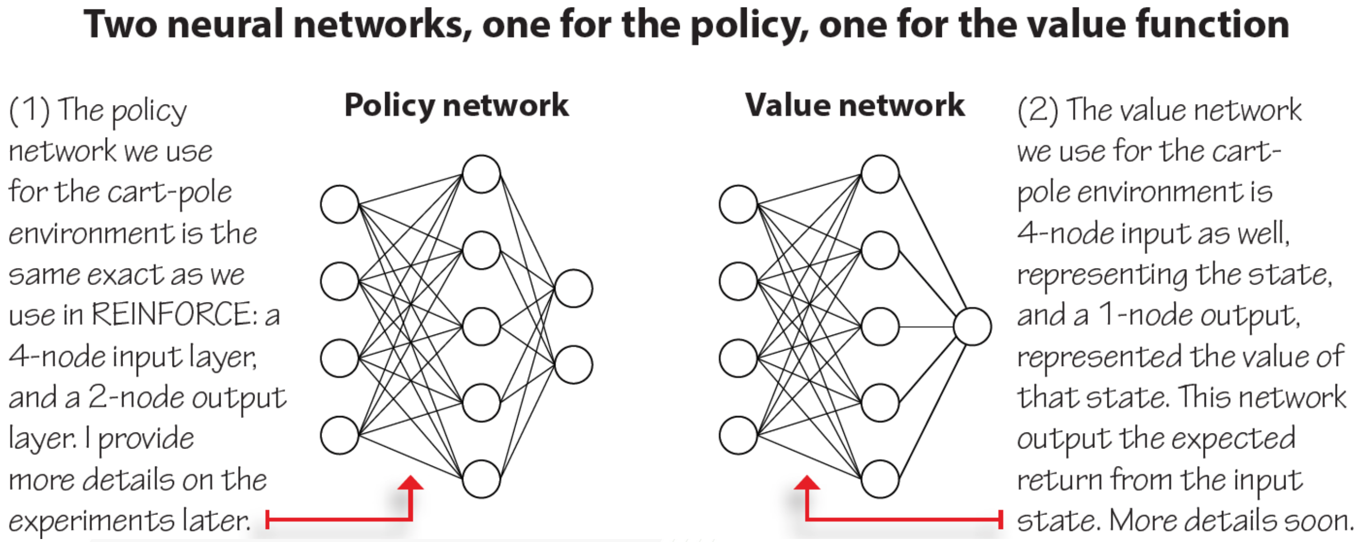
(2) All this is saying is, we sample a trajectory.

(3) Then, for each step in the trajectory, we calculate the return from that step.

(4) And use that value as the score to weight the log-probability of the action taken at that time step.

This is an agent called REINFORCE

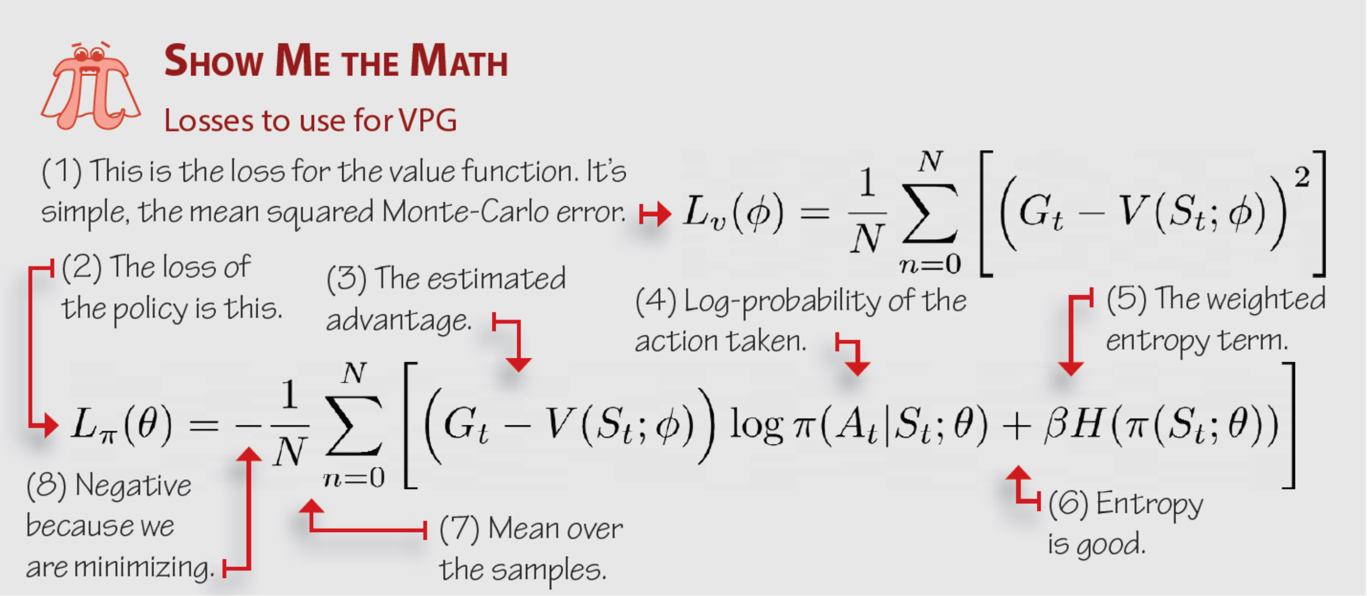
Vanilla Policy Gradient: How about learning a value function?



Two neural networks, one for the policy, one for the value function

(1) The policy network we use for the cart-pole environment is the same exact as we use in REINFORCE: a 4-node input layer, and a 2-node output layer. I provide more details on the experiments later.

(2) The value network we use for the cart-pole environment is 4-node input as well, representing the state, and a 1-node output, represented the value of that state. This network output the expected return from the input state. More details soon.



Show Me the Math

Losses to use for VPG

(1) This is the loss for the value function. It's simple, the mean squared Monte-Carlo error.

$$L_v(\phi) = \frac{1}{N}\sum_{n=0}^{N}\left[\left(G_t - V(S_t;\phi)\right)^2\right]$$

(2) The loss of the policy is this.

(3) The estimated advantage.

(4) Log-probability of the action taken.

(5) The weighted entropy term.

$$L_\pi(\theta) = -\frac{1}{N}\sum_{n=0}^{N}\left[\left(G_t - V(S_t;\phi)\right)\log \pi(A_t|S_t;\theta) + \beta H(\pi(S_t;\theta))\right]$$

(8) Negative because we are minimizing.

(7) Mean over the samples.

(6) Entropy is good.

Outline

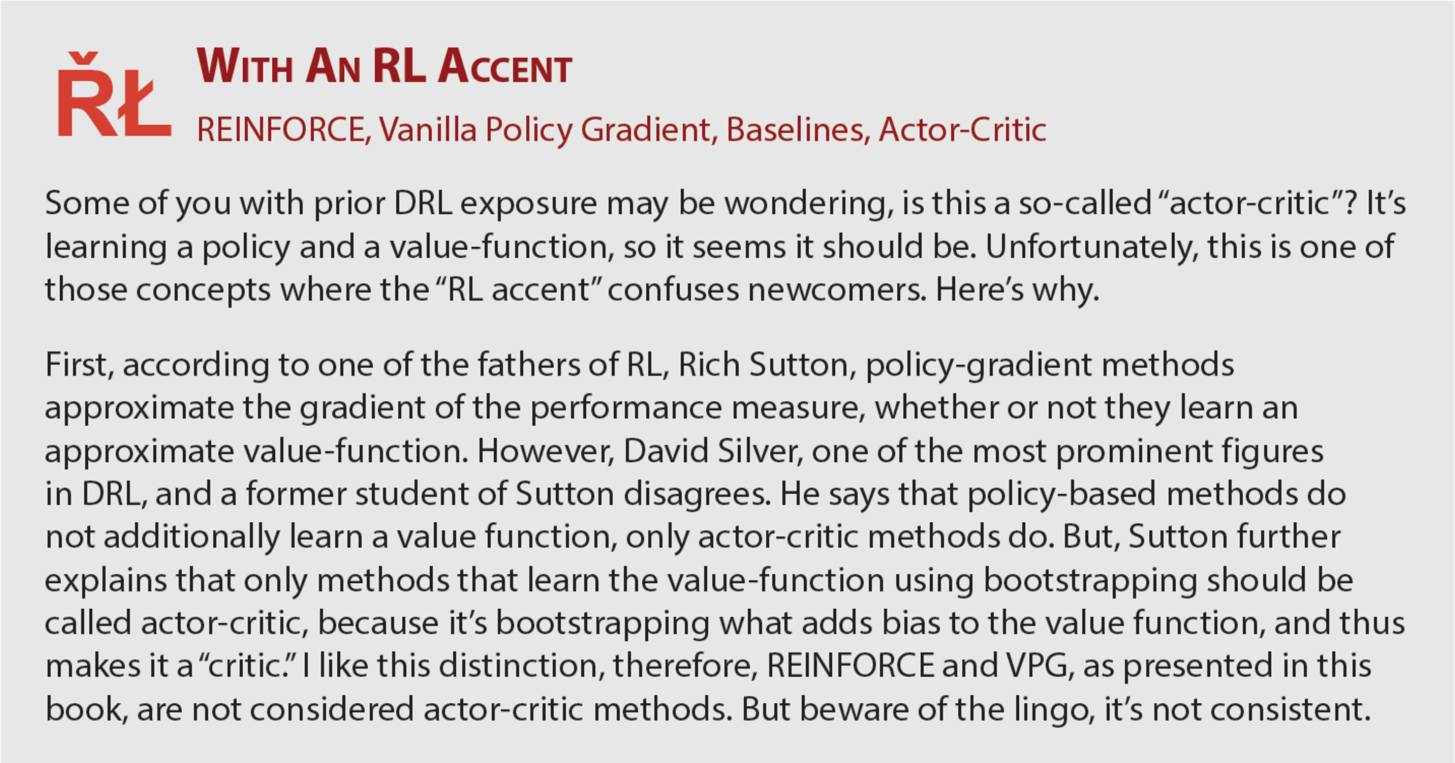
Policy-based Methods

Actor-Critic Methods

Advanced DRL Topics

> “Criticism may not be agreeable, but it is necessary. It fulfills the same function as pain in the human body. It calls attention to an unhealthy state of things.”
>
> — Winston Churchill
> British politician, army officer, writer, and
> Prime Minister of the United Kingdom

Actor-Critic Methods



With An RL Accent

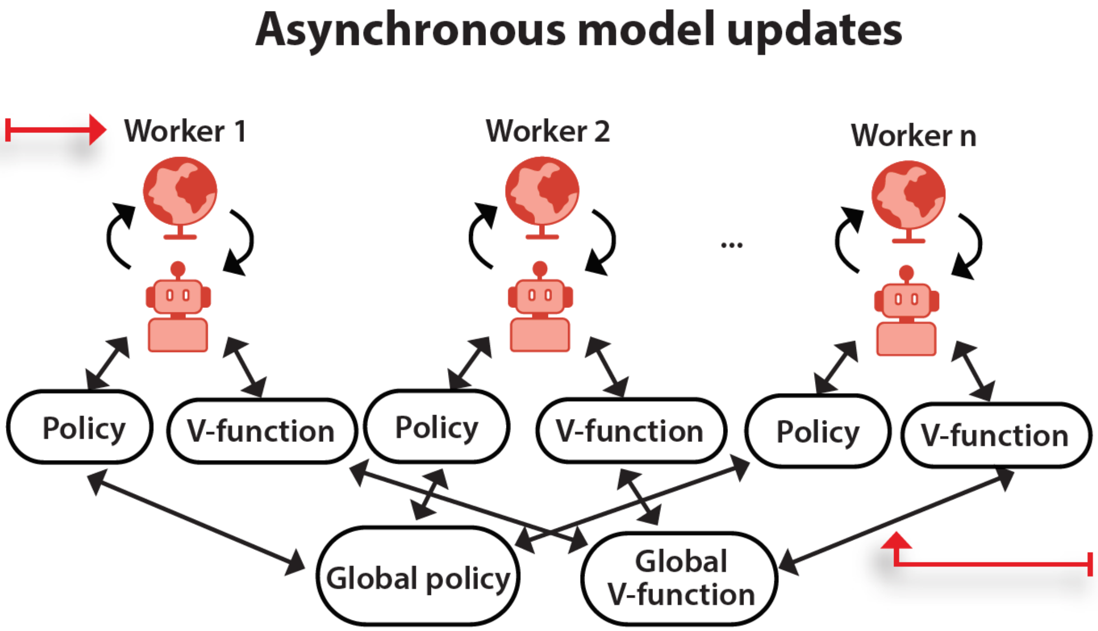
REINFORCE, Vanilla Policy Gradient, Baselines, Actor-Critic

Some of you with prior DRL exposure may be wondering, is this a so-called "actor-critic"? It's learning a policy and a value-function, so it seems it should be. Unfortunately, this is one of those concepts where the "RL accent" confuses newcomers. Here's why.

First, according to one of the fathers of RL, Rich Sutton, policy-gradient methods approximate the gradient of the performance measure, whether or not they learn an approximate value-function. However, David Silver, one of the most prominent figures in DRL, and a former student of Sutton disagrees. He says that policy-based methods do not additionally learn a value function, only actor-critic methods do. But, Sutton further explains that only methods that learn the value-function using bootstrapping should be called actor-critic, because it's bootstrapping what adds bias to the value function, and thus makes it a "critic." I like this distinction, therefore, REINFORCE and VPG, as presented in this book, are not considered actor-critic methods. But beware of the lingo, it's not consistent.

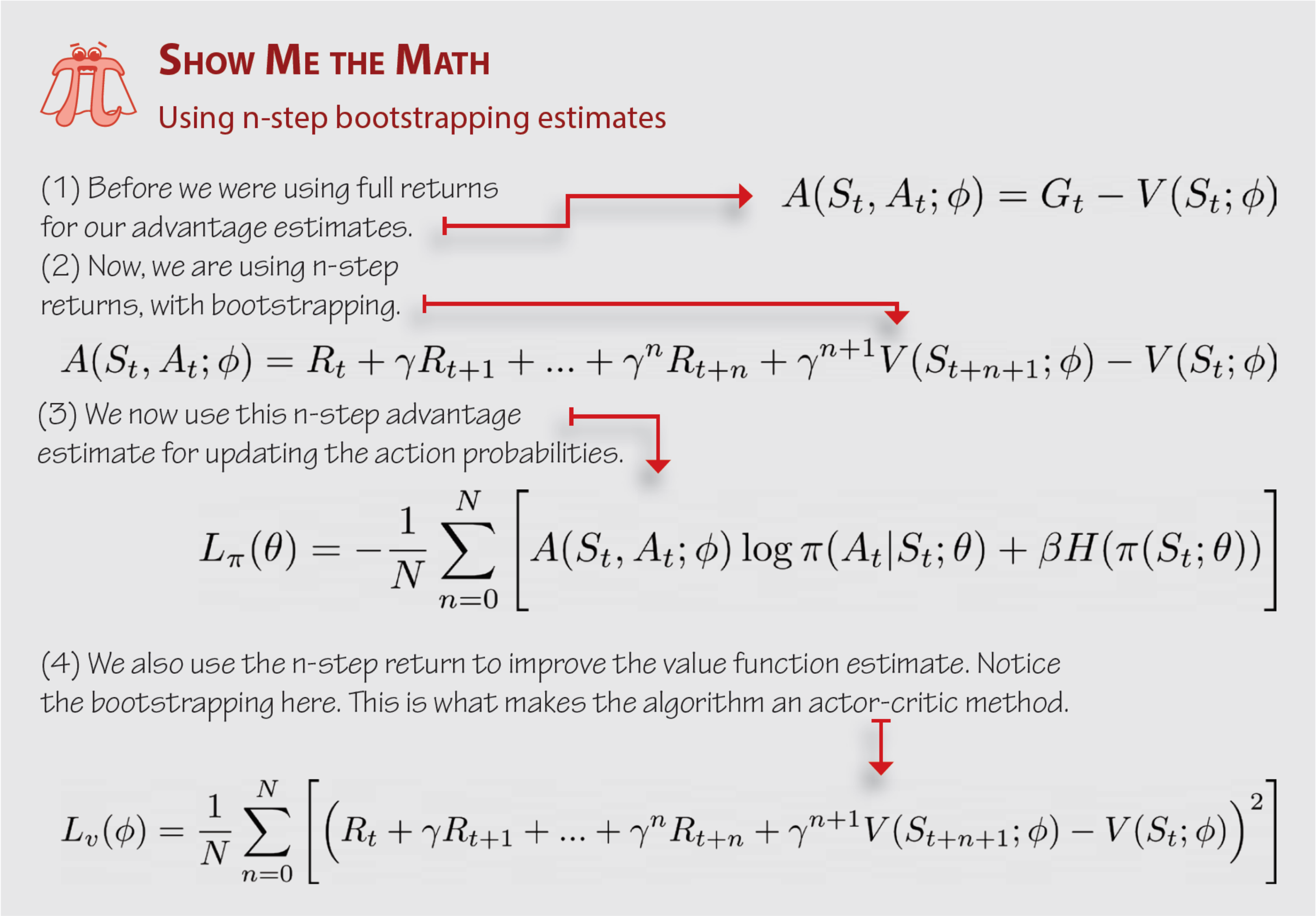
A3C: Parallel policy updates

(1) In A3C, we create multiple worker-learners. Each of them creates an instance of the environment, and the policy and V-function neural network weights use for generating experiences.



(2) After an experience batch is collected, each worker updates the global model asynchronously, without coordination with other workers. Then, they reload their copy of the models and keep at it.

A3C: Using n-step bootstrapping estimates



Show Me the Math

Using n-step bootstrapping estimates

(1) Before we were using full returns for our advantage estimates.

$$A(S_t, A_t; \phi) = G_t - V(S_t; \phi)$$

(2) Now, we are using n-step returns, with bootstrapping.

$$A(S_t, A_t; \phi) = R_t + \gamma R_{t+1} + ... + \gamma^n R_{t+n} + \gamma^{n+1} V(S_{t+n+1}; \phi) - V(S_t; \phi)$$

(3) We now use this n-step advantage estimate for updating the action probabilities.

$$L_\pi(\theta) = -\frac{1}{N} \sum_{n=0}^{N} \left[A(S_t, A_t; \phi) \log \pi(A_t | S_t; \theta) + \beta H(\pi(S_t; \theta)) \right]$$

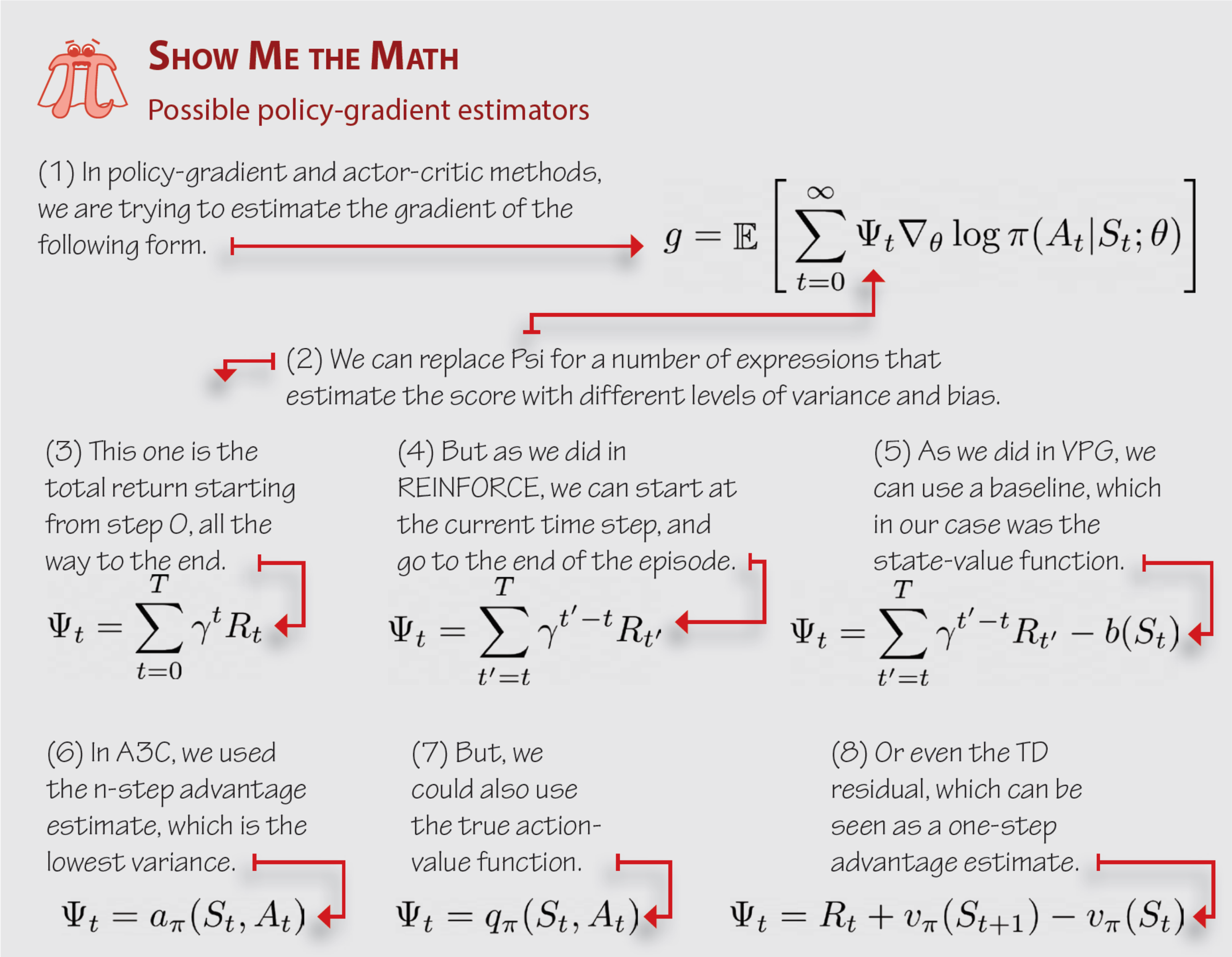
(4) We also use the n-step return to improve the value function estimate. Notice the bootstrapping here. This is what makes the algorithm an actor-critic method.

$$L_v(\phi) = \frac{1}{N} \sum_{n=0}^{N} \left[\left(R_t + \gamma R_{t+1} + ... + \gamma^n R_{t+n} + \gamma^{n+1} V(S_{t+n+1}; \phi) - V(S_t; \phi) \right)^2 \right]$$

A3C: Non-blocking model updates

- One of the most critical aspects of A3C is that its network updates are asynchronous and lock-free.
- Having a shared model creates a tendency for competent software engineers to want a blocking mechanism to prevent workers from overwriting other updates.
- Interestingly, A3C uses an update-style called a Hogwild!, which is being shown not only to achieve a near-optimal rate of convergence but also outperform alternative schemes that use locking by an order of magnitude.

GAE: TD(lambda) target for advantages



Show Me the Math

Possible policy-gradient estimators

(1) In policy-gradient and actor-critic methods, we are trying to estimate the gradient of the following form.

$$g = \mathbb{E}\left[\sum_{t=0}^{\infty} \Psi_t \nabla_\theta \log \pi(A_t|S_t;\theta)\right]$$

(2) We can replace Psi for a number of expressions that estimate the score with different levels of variance and bias.

(3) This one is the total return starting from step 0, all the way to the end.

$$\Psi_t = \sum_{t=0}^{T} \gamma^t R_t$$

(4) But as we did in REINFORCE, we can start at the current time step, and go to the end of the episode.

$$\Psi_t = \sum_{t'=t}^{T} \gamma^{t'-t} R_{t'}$$

(5) As we did in VPG, we can use a baseline, which in our case was the state-value function.

$$\Psi_t = \sum_{t'=t}^{T} \gamma^{t'-t} R_{t'} - b(S_t)$$

(6) In A3C, we used the n-step advantage estimate, which is the lowest variance.

$$\Psi_t = a_\pi(S_t, A_t)$$

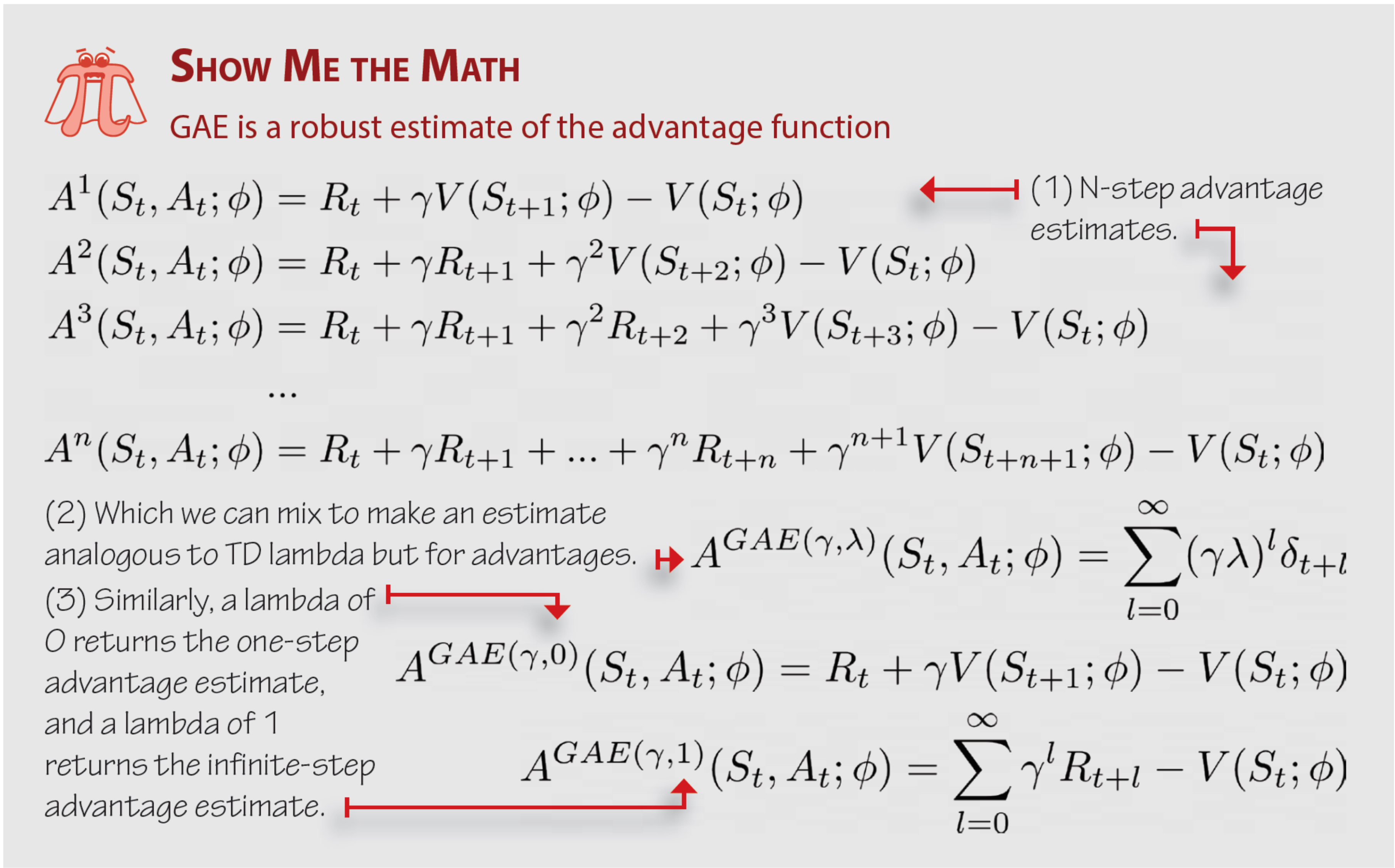
(7) But, we could also use the true action-value function.

$$\Psi_t = q_\pi(S_t, A_t)$$

(8) Or even the TD residual, which can be seen as a one-step advantage estimate.

$$\Psi_t = R_t + v_\pi(S_{t+1}) - v_\pi(S_t)$$

GAE: GAE(lambda), GAE(0), GAE(1)



Show Me the Math

GAE is a robust estimate of the advantage function

$$A^1(S_t, A_t; \phi) = R_t + \gamma V(S_{t+1}; \phi) - V(S_t; \phi)$$

$$A^2(S_t, A_t; \phi) = R_t + \gamma R_{t+1} + \gamma^2 V(S_{t+2}; \phi) - V(S_t; \phi)$$

$$A^3(S_t, A_t; \phi) = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^3 V(S_{t+3}; \phi) - V(S_t; \phi)$$

$$\dots$$

$$A^n(S_t, A_t; \phi) = R_t + \gamma R_{t+1} + \dots + \gamma^n R_{t+n} + \gamma^{n+1} V(S_{t+n+1}; \phi) - V(S_t; \phi)$$

(1) N-step advantage estimates.

(2) Which we can mix to make an estimate analogous to TD lambda but for advantages.

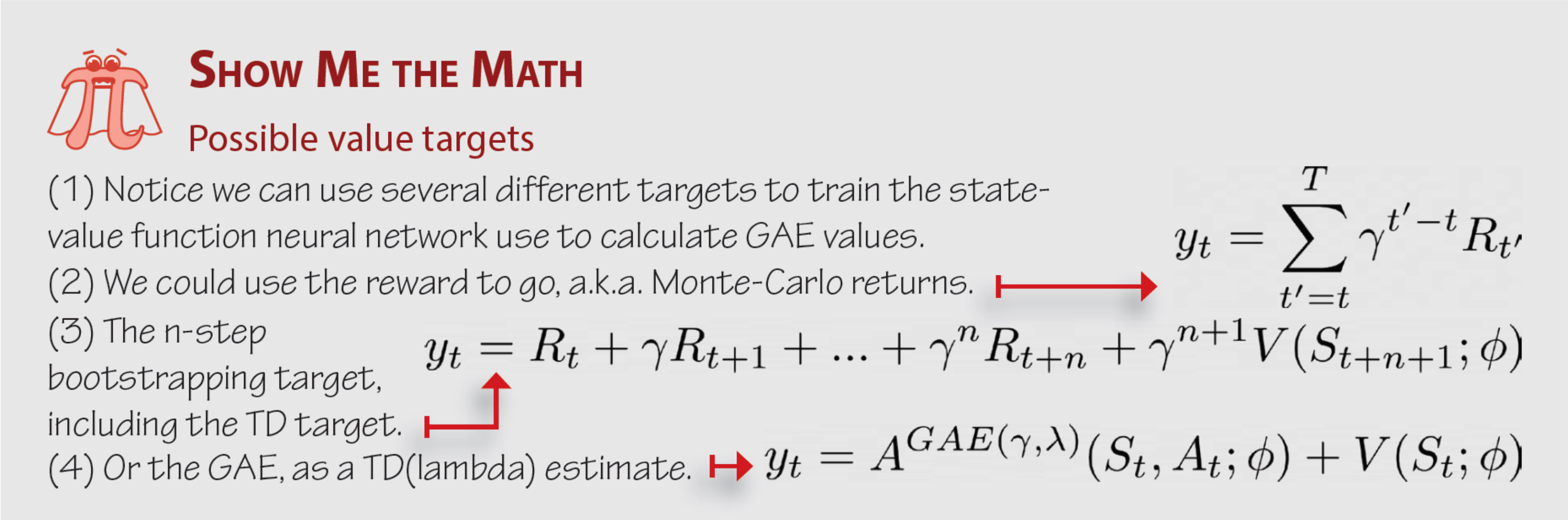
$$A^{GAE(\gamma,\lambda)}(S_t, A_t; \phi) = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}$$

(3) Similarly, a lambda of 0 returns the one-step advantage estimate, and a lambda of 1 returns the infinite-step advantage estimate.

$$A^{GAE(\gamma,0)}(S_t, A_t; \phi) = R_t + \gamma V(S_{t+1}; \phi) - V(S_t; \phi)$$

$$A^{GAE(\gamma,1)}(S_t, A_t; \phi) = \sum_{l=0}^{\infty} \gamma^l R_{t+l} - V(S_t; \phi)$$

GAE: How to train V?



Show Me the Math

Possible value targets

(1) Notice we can use several different targets to train the state-value function neural network use to calculate GAE values.

(2) We could use the reward to go, a.k.a. Monte-Carlo returns.

$$y_t = \sum_{t'=t}^{T} \gamma^{t'-t} R_{t'}$$

(3) The n-step bootstrapping target, including the TD target.

$$y_t = R_t + \gamma R_{t+1} + \ldots + \gamma^n R_{t+n} + \gamma^{n+1} V(S_{t+n+1}; \phi)$$

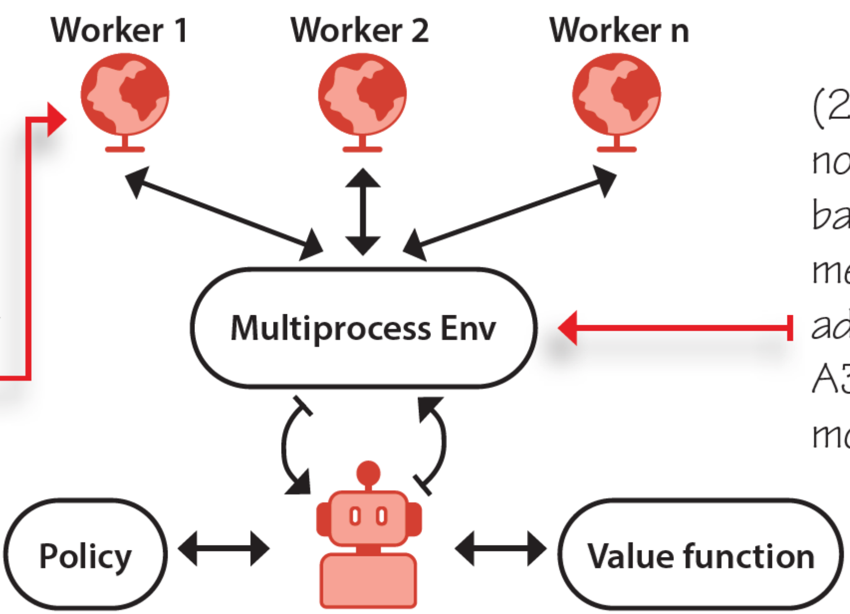
(4) Or the GAE, as a TD(lambda) estimate.

$$y_t = A^{GAE(\gamma,\lambda)}(S_t, A_t; \phi) + V(S_t; \phi)$$

A2C: Synchronous policy updates

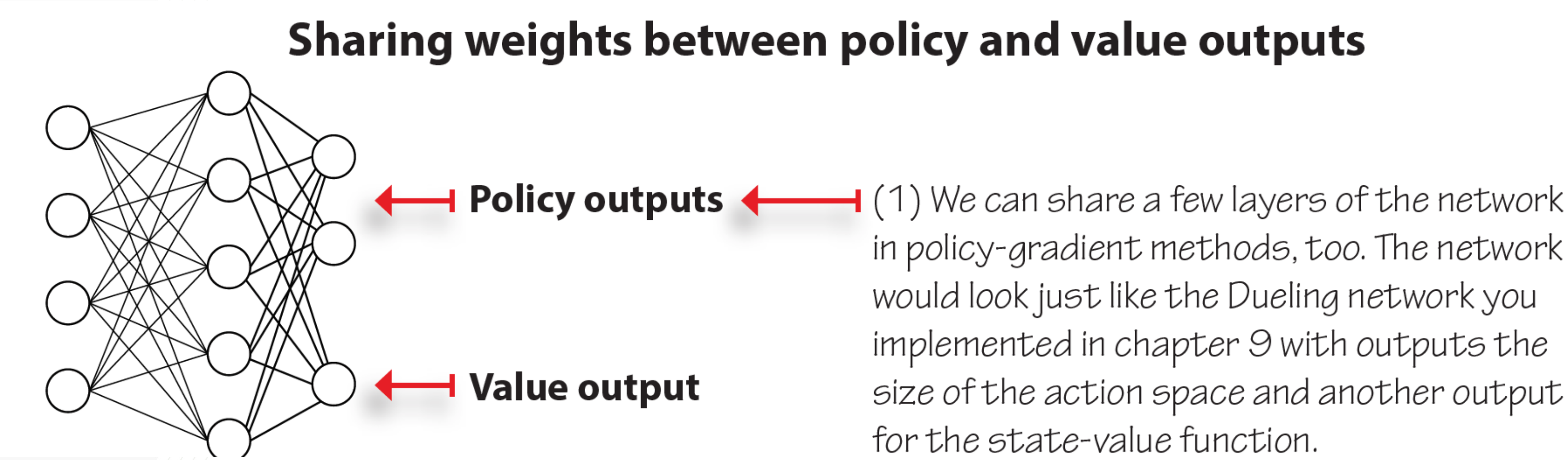
Synchronous model updates

(1) In A2C, we have a single agent driving the interaction with the environment. But, in this case the environment is a multi-process class that gathers samples from multiple environments at once.



(2) The neural networks now need to process batches of data. Which means in A2C we can take advantage of GPUs, unlike A3C in which CPUs are the most important resource.

A2C: Weight sharing model



Sharing weights between policy and value outputs

Policy outputs

Value output

(1) We can share a few layers of the network in policy-gradient methods, too. The network would look just like the Dueling network you implemented in chapter 9 with outputs the size of the action space and another output for the state-value function.

More policy-based and actor-critic methods

- DDPG: Deep Deterministic Policy Gradient
- TD3: Twin Delayed DDPG
- SAC: Soft-Actor Critic
- TRPO: Trust-Region Policy Optimization
- PPO: Proximal Policy Optimization

https://github.com/mimoralea/gdrl

Outline

Policy-based Methods

Actor-Critic Methods

Advanced DRL Topics

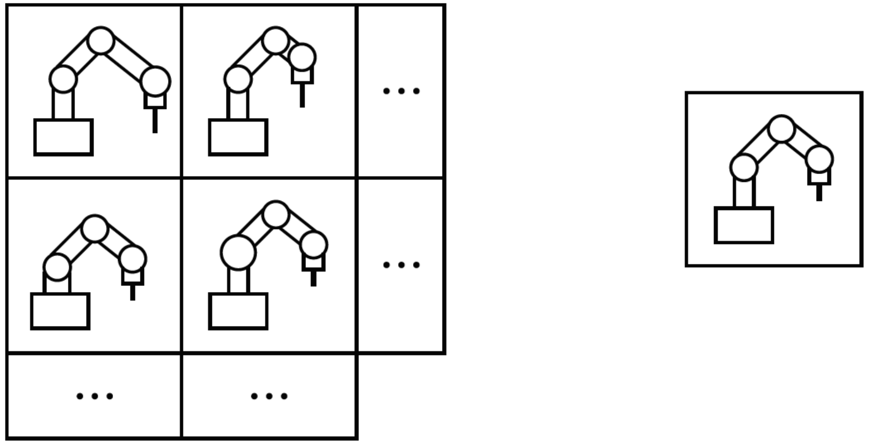
> "Our ultimate objective is to make programs that learn from their experience as effectively as humans do."
>
> — John McCarthy
> Founder of the field of Artificial Intelligence
> Inventor of the Lisp programming Language

Transfer learning

Sim-to-real transfer learning task is a common need in the real world

Domain randomization in simulation at training time

Better generalization at test time in the real world



(1) Lots of people think you need a high-fidelity simulation to transfer and agent from simulation to the real world, but that is, in fact, not true!

(2) What works better is to have a flexible simulator so that you can randomize the parameters during training, and the agent is forced to generalize better.

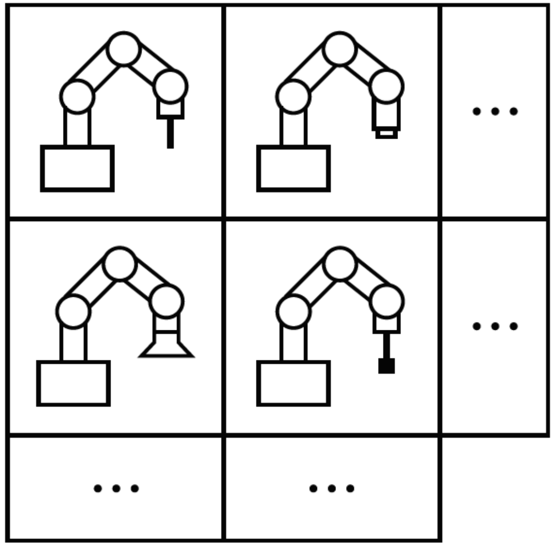
(3) Then, the real-world looks just like another variation of the simulation.

Multi-task learning

Multi-task learning consists of training on multiple related tasks, and testing on a new one

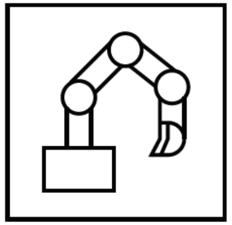
(1) Multi-tasks learning is the transfer of a policy trained in multiple tasks, either simultaneously or sequentially, to another task.

Multiple related tasks at training time



(2) In this example, I use 4 different end effectors, but in reality, the task doesn't need to be too different. This could be very related tasks.

Better generalization at test time



(3) The idea is the agent should perform better at the target tasks, either with no or some fine-tuning.

Other areas of research

- Curriculum learning: Learn on progressively more difficult tasks.
- Meta learning: Learning to learn. How to solve a task with n-shot trials (including zero-shot).
- Hierarchical reinforcement learning: Learning a hierarchy of policies.
- Multi-agent reinforcement learning: How does the challenge change when multiple agents are learning simultaneously?
- Explainable AI, Safety, Fairness, Ethical Standards: How to we make sure we are preparing for the inclusion of AI systems to our daily lives?

Thank you!