

Reinforcement Learning

Exercise 6: Policy Approximation

09.06.2023

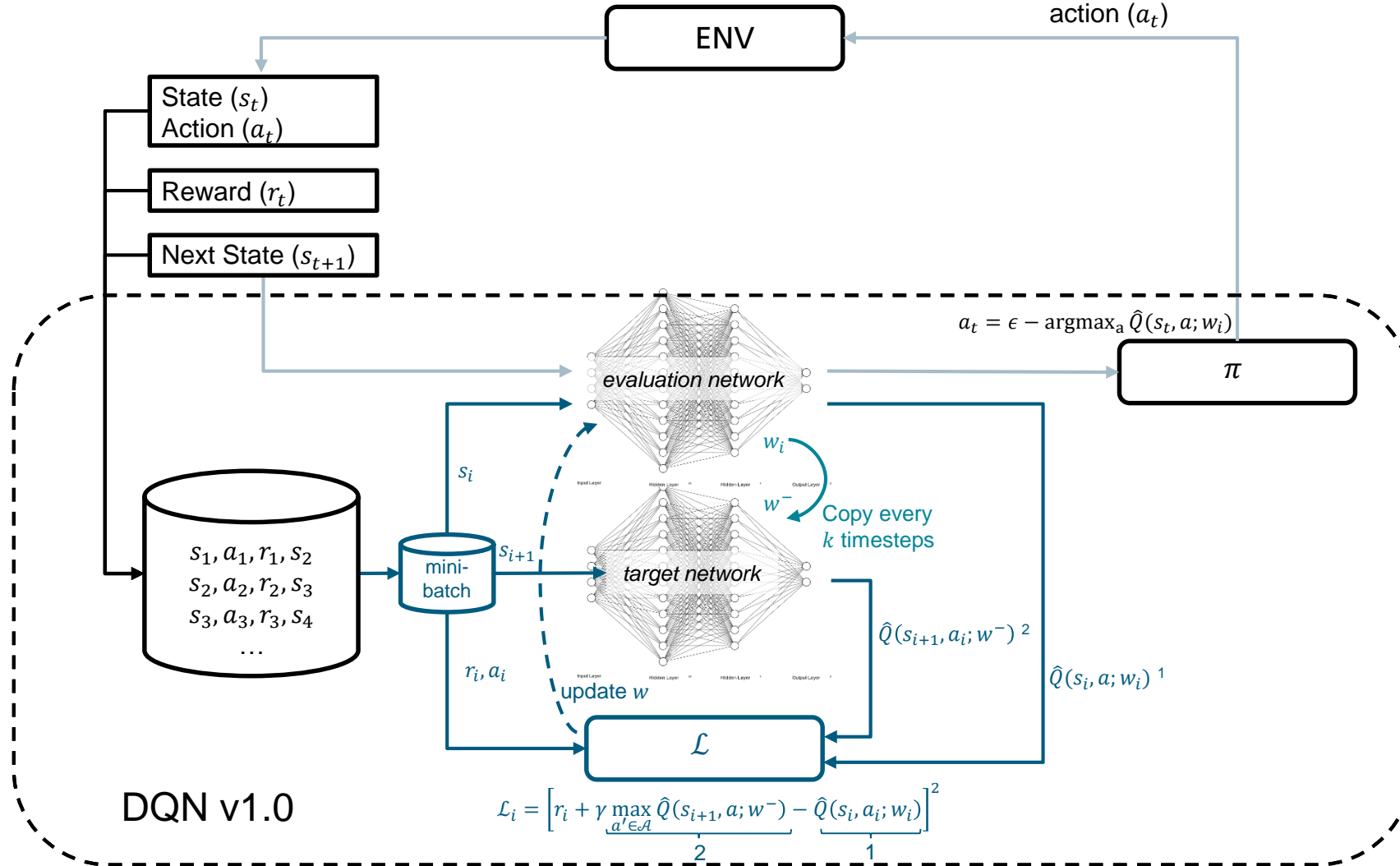
Nico Meyer

Exercise Sheet 6

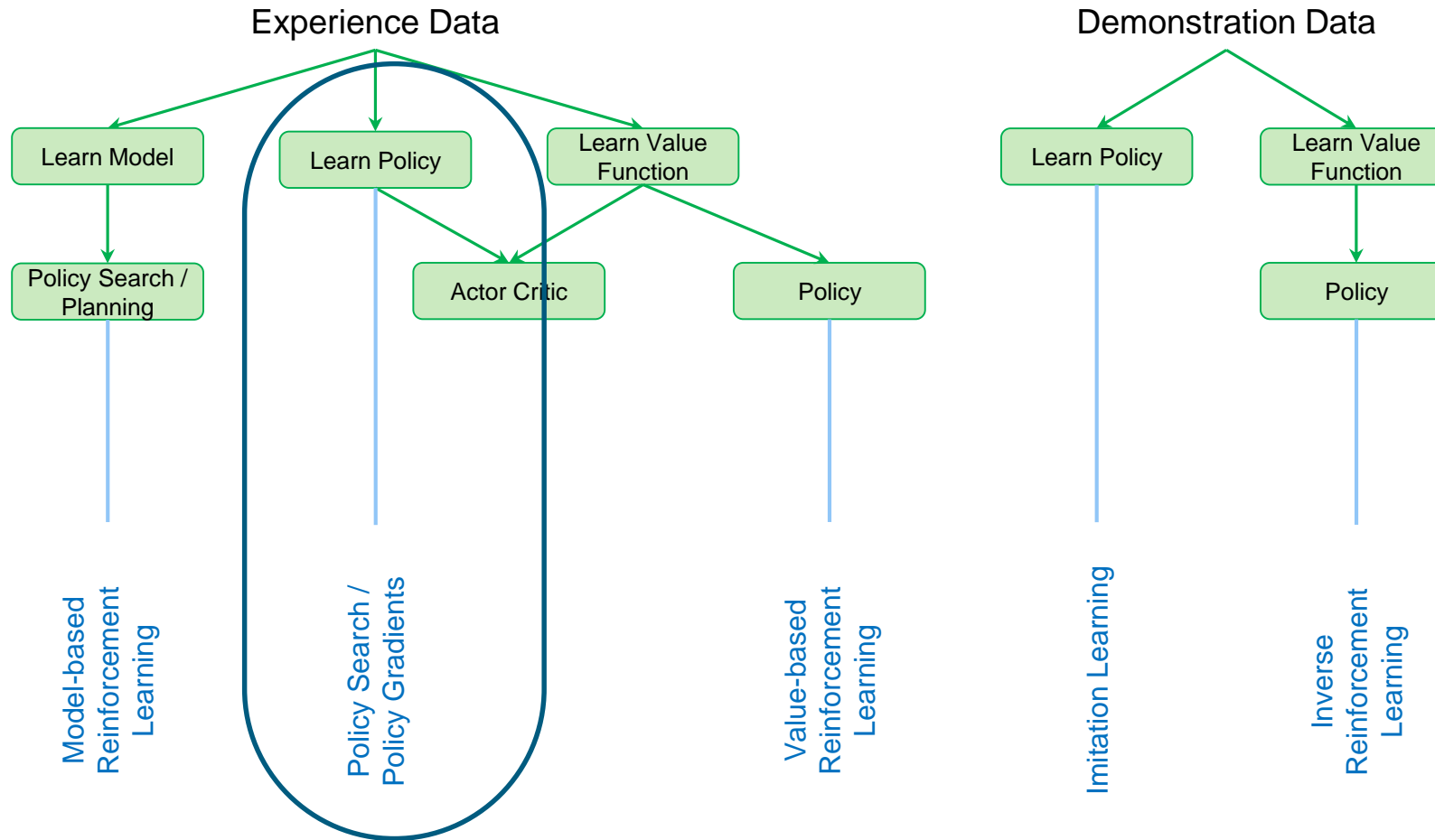
Value Function Approximation



Deep Q-Networks (DQNs)



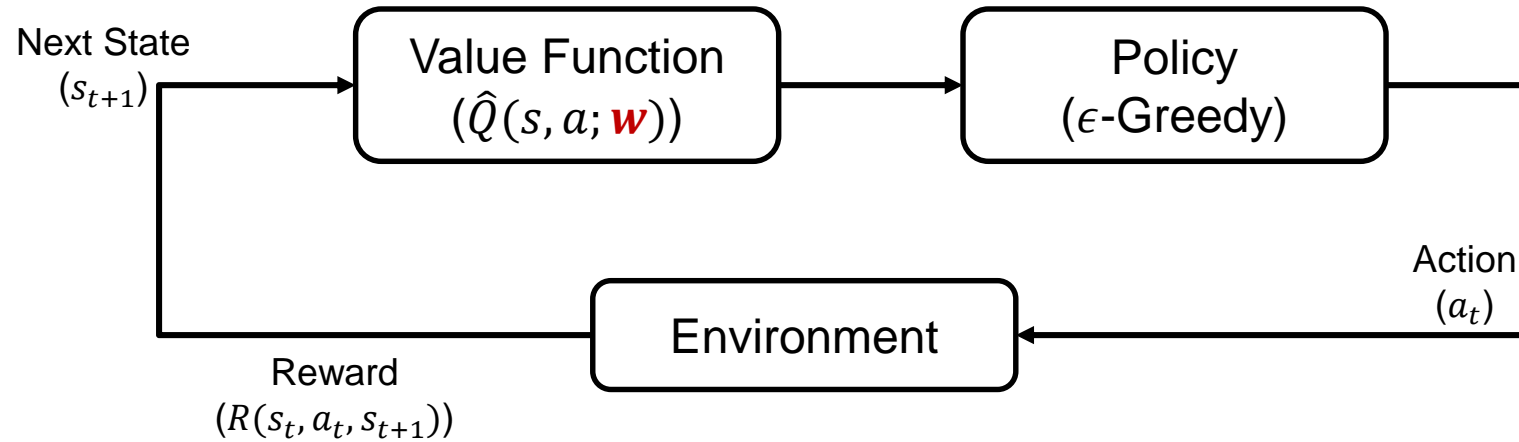
Policy-based Reinforcement Learning



Policy-based Reinforcement Learning

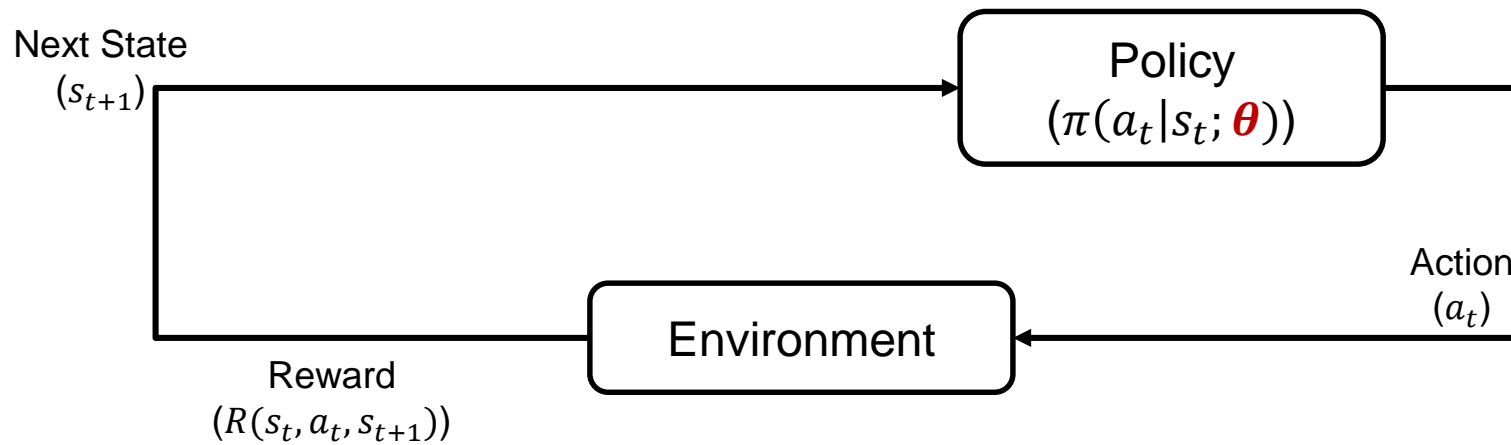
Change of pipeline

So far:



Goal: find w that approximates the true Q -function

Now:



Goal: find θ that maximizes long term reward

Policy-based Reinforcement Learning

Advantages and Disadvantages

Advantages:

- **Good convergence properties**
- Easily extended to high-dimensional or continuous state and action spaces
- Can learn ***stochastic policies***
- Sometimes policies are simple while values and models are complex
 - e.g., rich domain, but optimal is always to go left

Disadvantages:

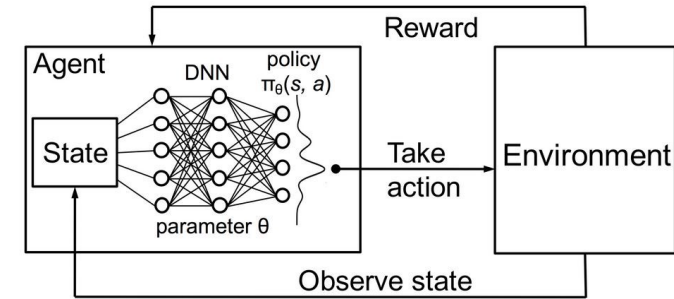
- Susceptible to local optima (especially with non-linear FA)
- Obtained knowledge is specific, does not always generalize well
- Ignores a lot of information in the data (when used in isolation)

Policy-based Reinforcement Learning

Stochastic policies

We have seen deterministic policies like this:

State gives $Q(s, a; w)$ and we selected $\pi(a|s)$ by $\operatorname{argmax}_a Q(s, a; w)$

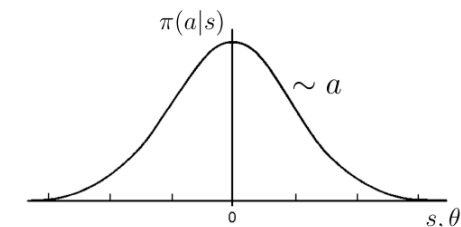


<https://towardsdatascience.com/self-learning-ai-agents-iv-stochastic-policy-gradients-b53f088fce20>

Instead, stochastic policies do something like this:

$$\pi(a|s) = \mathbb{P}[a|s; \theta]$$

(policy is represented as a probability distribution)



➤ optimal policy might be stochastic

Policy-based Reinforcement Learning

Optimizing via gradient ascent

- Our goal is to maximize the expected reward:

$$G(\tau) := \sum_{t=0}^{T-1} \gamma^t R(s_t, a_t)$$
$$\max_{\theta} \mathbb{E}_{\pi_{\theta}} G(\tau)$$

(where π_{θ} is a parameterized policy, e.g., a neural network)

- But how do we maximize this?

→ **Gradient Ascent!** Suppose we know how to calculate the gradient w.r.t. the parameters:

$$\nabla_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}} G(\tau)$$

- Then we can update our parameters θ in the direction of the gradient:

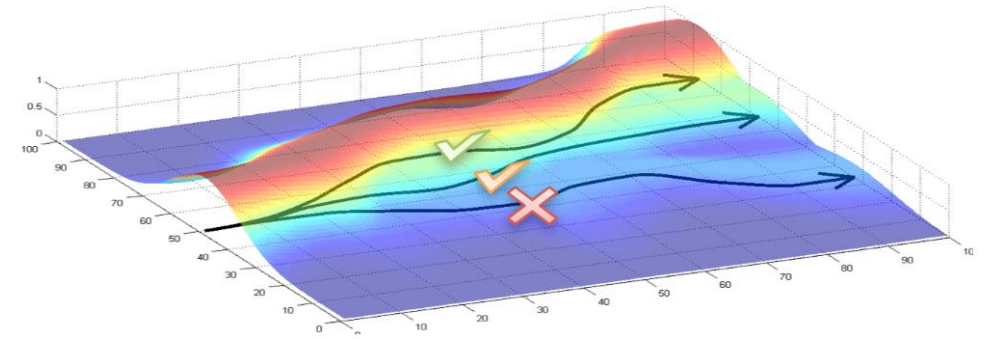
$$\theta \leftarrow \theta + \alpha \nabla_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}} G(\tau)$$

Policy Gradient

often in literature
referred to as $\nabla_{\theta} J(\pi_{\theta})$

Policy-based Reinforcement Learning

REINFORCE



Pieter Abbeel. DeepRL Bootcamp 4A Policy Gradients.

REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for π_*

Input: a differentiable policy parameterization $\pi(a|s, \theta)$

Algorithm parameter: step size $\alpha > 0$

Initialize policy parameter $\theta \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):

Generate an episode $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \theta)$

Loop for each step of the episode $t = 0, 1, \dots, T - 1$:

$$G \leftarrow \sum_{k=t+1}^T \gamma^{k-t-1} R_k \quad (G_t)$$

$$\theta \leftarrow \theta + \alpha \gamma^t G \nabla \ln \pi(A_t|S_t, \theta)$$

Intuition: The gradient tries to

- Increase probability of paths with positive G
- Decrease probability of paths with negative G

Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.

Exercise Sheet 7.1

Vanilla Policy Gradient (VPG)



Actor Critic Approaches

Introduce critic that estimates Q

- The policy gradient we used so far (without baseline to begin with):

$$\begin{aligned}\nabla_{\theta} \mathbb{E}_{\pi_{\theta}} G(\tau) &\approx \frac{1}{L} \sum_{\tau} \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) G(\tau) \\ &\approx \frac{1}{L} \sum_{\tau} \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \underbrace{\sum_{t'=t}^T \gamma^{t'-t} R(s_{t'}, a_{t'})}_{= Q^{\pi}(s_t, a_t)}\end{aligned}$$

- Use e.g. a neural network to approximate Q!
- In practice: estimate $v^{\pi}(s_t; \phi)$ explicitly, and then sample

$$q^{\pi}(s_t, a_t) \approx G_t^{(n)}$$

$$\text{i.e. } \hat{G}_t^{(1)} = R_t + \gamma v^{\pi}(s_{t+1}; \phi)$$

Actor Critic Approaches

Advantage Actor Critic (A2C)

- Introduce a baseline:

$$\begin{aligned}\nabla_{\theta} \mathbb{E}_{\pi_{\theta}} G(\tau) &\approx \frac{1}{L} \sum_{\tau} \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \underbrace{\sum_{t'=t}^T \gamma^{t'-t} R(s_{t'}, a_{t'}) - b(s_t)}_{= Q^{\pi}(s_t, a_t)} \\ &= \frac{1}{L} \sum_{\tau} \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \boxed{(\hat{G}_t - b(s_t))} \\ &\quad := A^{\pi}(s_t, a_t)\end{aligned}$$

- Calculate via TD error:

$$\begin{aligned}A^{\pi}(s_t, a_t) &= Q^{\pi}(s_t, a_t) - V^{\pi}(s_t) \\ &= r + \gamma \cdot v^{\pi} - v^{\pi}(s_t)\end{aligned}$$

- Or multi-step TD error: "Generalized Advantage Estimation (GAE)"

Exercise Sheet 7.2

Advantage Actor Critic (A2C)



Better Control of Improvement Steps

Potential problems with gradient-based updates

- Note: the advantage function (which is a noisy estimate) may not be accurate
 - Too large steps may lead to a disaster (even *if* the gradient is *correct*)
 - Too small steps are also bad
 - Mathematical formulization:
 - First-order derivatives approximate the (parameter) surface to be flat
 - But if the surface exhibits high curvature it gets dangerous
 - Projection: small changes in parameter space might lead to large changes in policy space!
 - **Parameters θ get updated to areas too far out of the range from where previous data was collected**
- Regularize updates to the policy parameters such that the policy does not change too much



Images taken from https://medium.com/@jonathan_hui/rl-trust-region-policy-optimization-trpo-explained-a6ee04e0e0e9 and <http://www.taiwanoffthebeatentrack.com/2012/08/23/mount-hua-华山-the-most-dangerous-hike-in-the-world/>

Better Control of Improvement Steps

Natural Policy Gradients

TRPO tries to approximate inverse of Fisher information (i.e. Hessian)

- First-order derivatives approximate the (parameter) surface to be flat
- But if the surface exhibits high curvature it gets dangerous
- **Small changes in parameter space might lead to large changes in policy space!**

What we essentially do

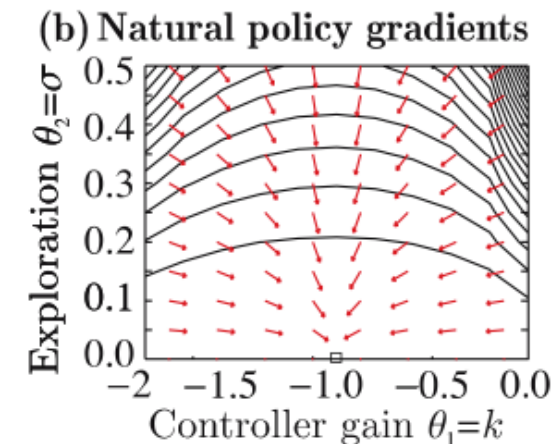
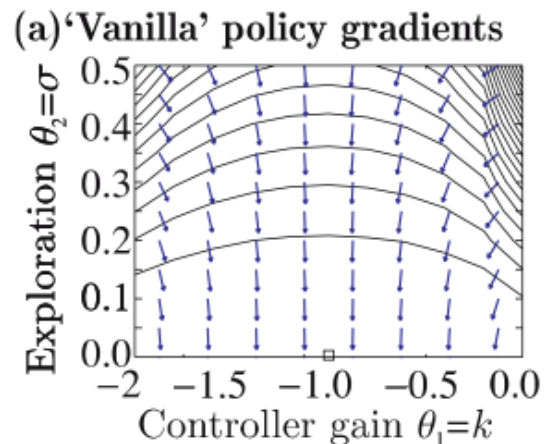
(optimization perspective on 1st order gradient descent)

$$\theta' \leftarrow \arg \max_{\theta'} (\theta' - \theta)^T \nabla_{\theta} J(\theta), \text{ subject to } \|\theta' - \theta\|^2 \leq \epsilon$$

What we want to do

(incorporate 2nd order information)

$$\theta' \leftarrow \arg \max_{\theta'} (\theta' - \theta)^T \nabla_{\theta} J(\theta), \text{ subject to } \|\theta' - \theta\|_F^2 \leq \epsilon$$
$$\rightarrow \theta \leftarrow \theta + \alpha \mathbf{F}^{-1} \nabla_{\theta} J(\theta) \text{ with e.g. KL-divergence}$$



Peters et al.: Natural Actor-Critic. 2018

Better Control of Improvement Steps

Proximal Policy Optimization (PPO)

- The main motivation behind PPO is the same as for TRPO:
 - Make the biggest possible improvement step
 - Do not step too far such that the performance accidentally collapses
- PPO addresses the shortcomings of TRPO:
 - PPO uses 1st order methods with a few tricks
 - Significantly simpler to implement
 - Shows similar performance to TRPO (empirically)

removes the incentive for moving r_t outside of the interval $[1 - \epsilon, 1 + \epsilon]$

- PPO-Clip: The PPO objective we want to maximize is given by

$$= g(\epsilon, \hat{A}_t(s, a))$$

$$g(\epsilon, \hat{A}_t) = \begin{cases} (1 + \epsilon)A, & \text{if } A \geq 0 \\ (1 - \epsilon)A, & \text{if } A < 0 \end{cases}$$

$$L(\theta) = \hat{\mathbb{E}}_t [\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_t)]$$

where ϵ is a hyperparameter (i.e., 0.1 or 0.2) that defines how far π_{new} may go away from π_{old}

➤ the final objective is a lower bound (i.e., a pessimistic bound) on the unclipped objective

Better Control of Improvement Steps

Automated Tuning of the gradient step *without calculating the Hessian*

Algorithm 1 PPO-Clip

- 1: Input: initial policy parameters θ_0 , initial value function parameters ϕ_0
- 2: **for** $k = 0, 1, 2, \dots$ **do**
- 3: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
- 4: Compute rewards-to-go \hat{R}_t .
- 5: Compute advantage estimates, \hat{A}_t (using any method of advantage estimation) based on the current value function V_{ϕ_k} .
- 6: Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg \max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T \min \left(\frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

typically via stochastic gradient ascent with Adam.

- 7: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg \min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T \left(V_{\phi}(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.

- 8: **end for**
-

Advantage Clipping for conservative policy updates

See also: <https://spinningup.openai.com/en/latest/algorithms/ppo.html>

Thank you for your attention!