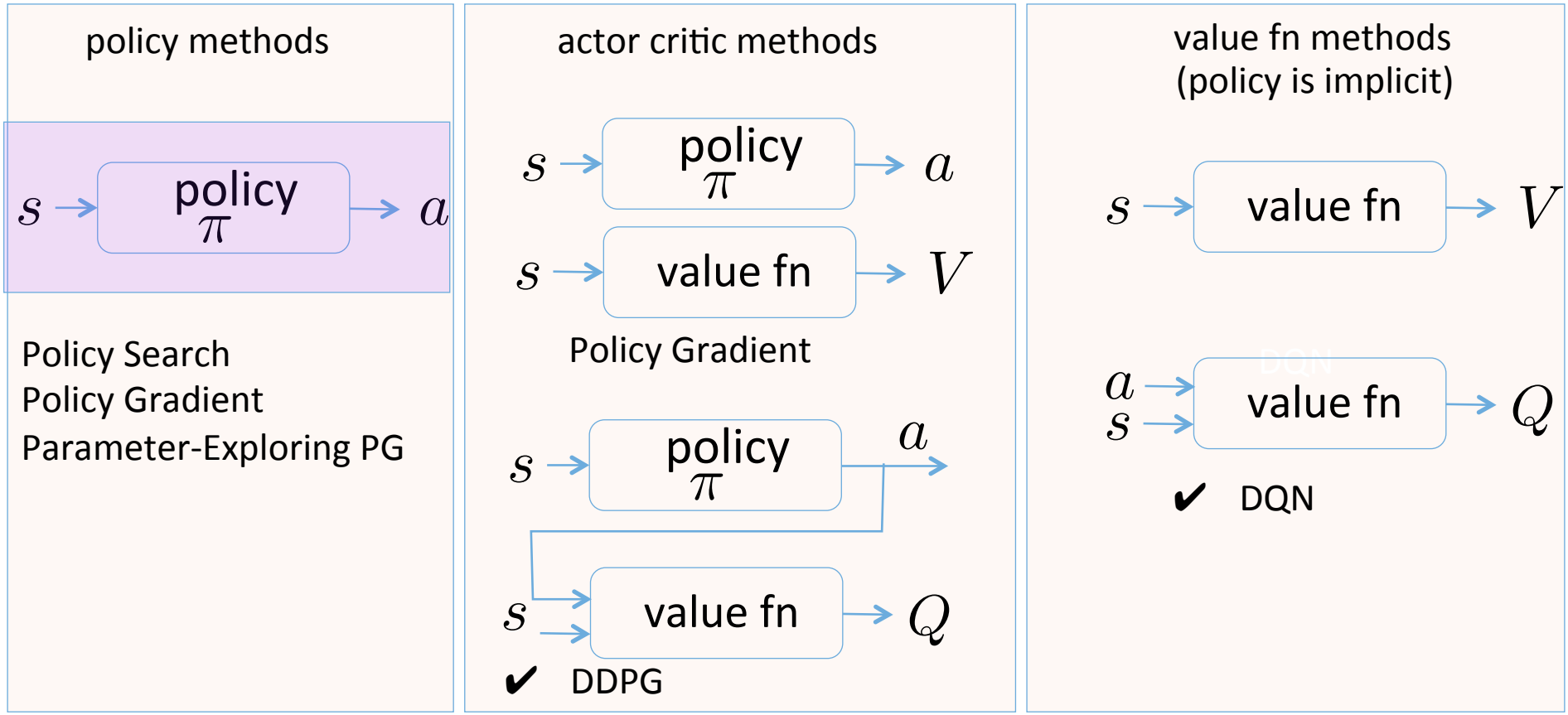
Policy-based Methods

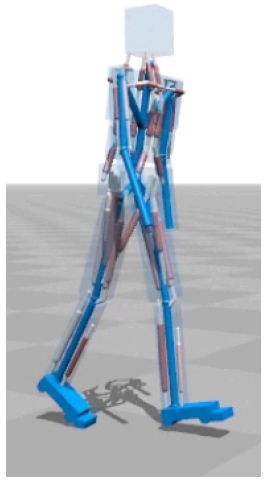


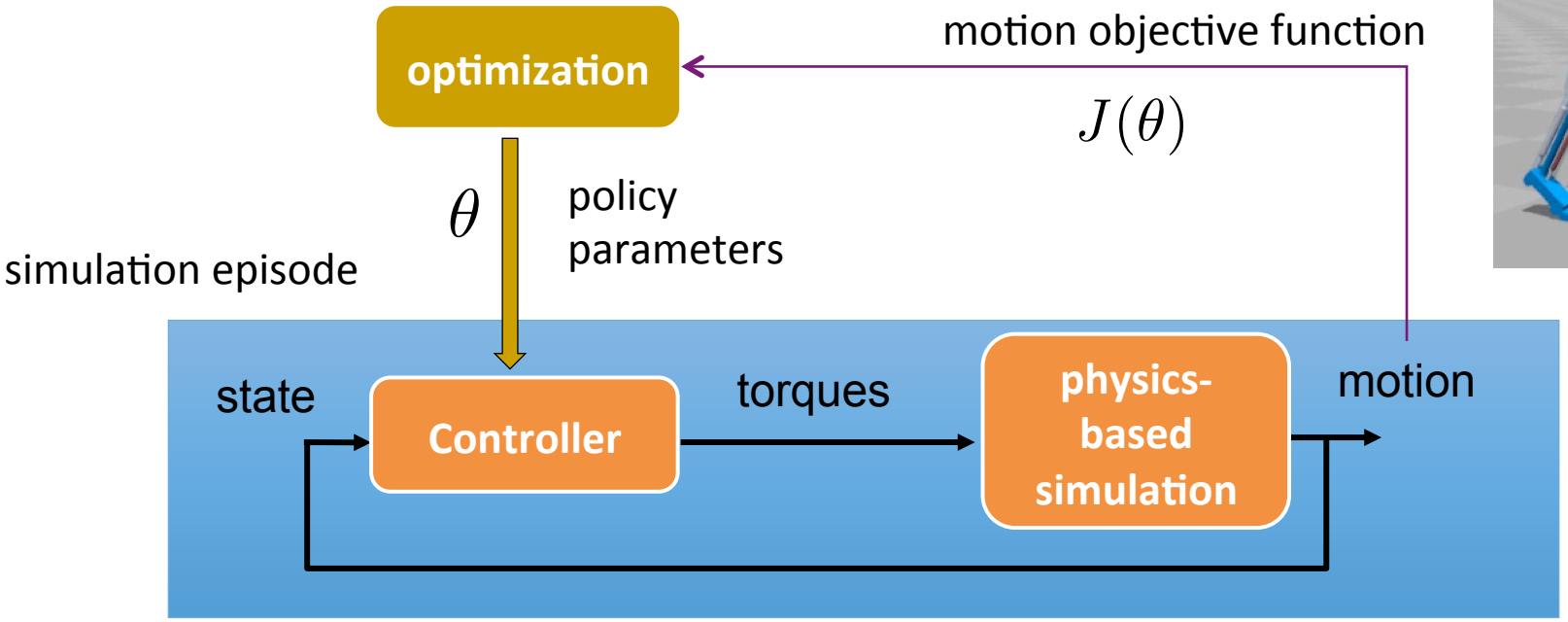
Why policy-based methods?

- to handle continuous actions
- good solutions can (sometimes) be described using simple policies

Policy Search / Hill Climbing / Black Box / Evolutionary / Gradient-Free / Derivative-Free Optimization / Parameter-Exploring Policy Gradients

Optimize $J(\theta)$ using the total episode returns





Policy optimization problem

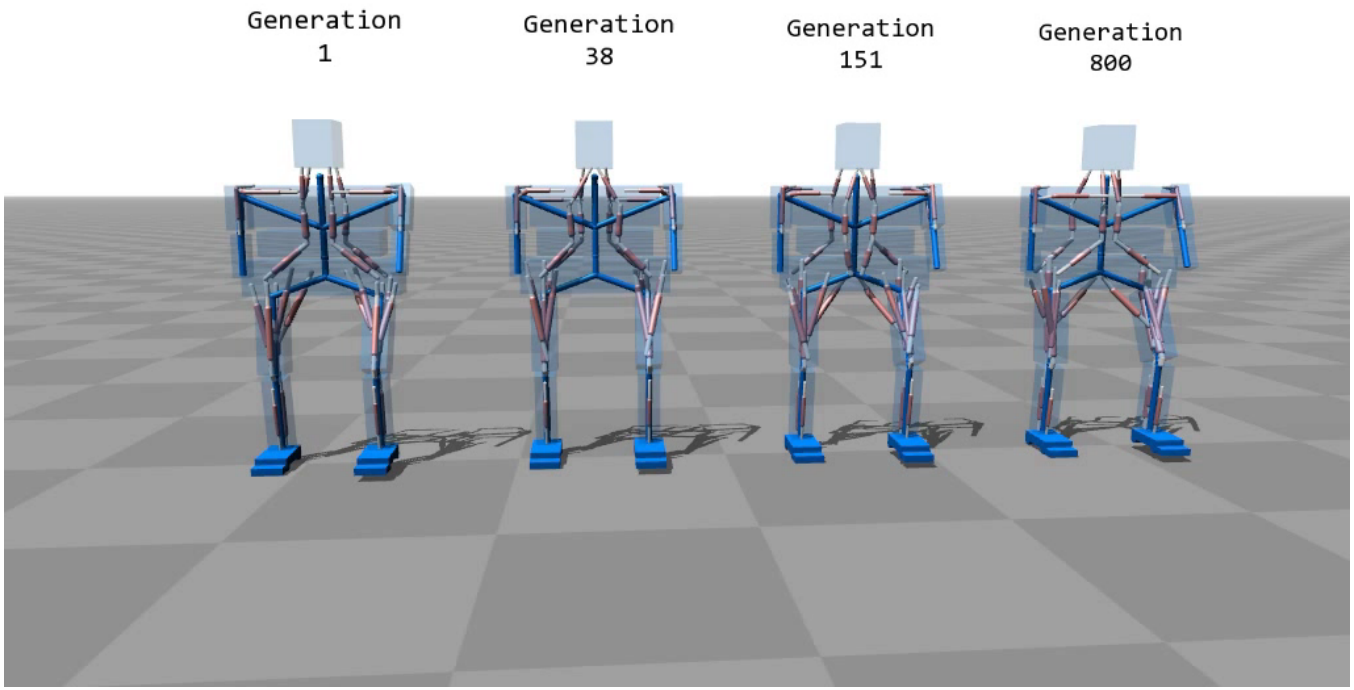
policy parameters:

Subject	Parameters
Muscle physiology	3–30 *
Muscle geometry	12–39 *
State transition	3
Target features	14
Feedback control	14–63 *
Initial character state	6

objective:

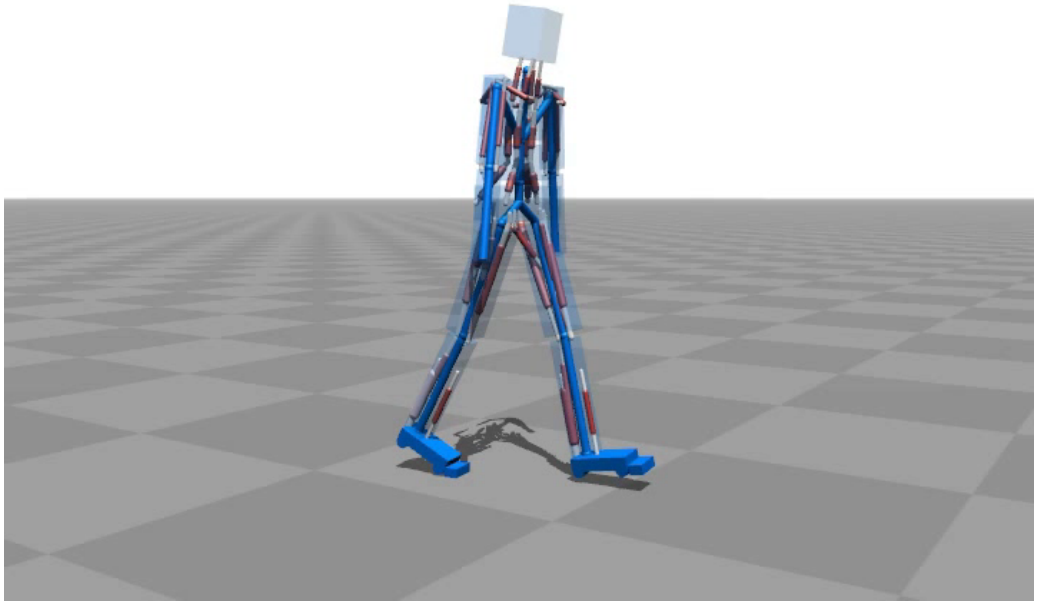
$$\bar{E}(\mathbf{K}) = \bar{E}_{\text{speed}} + \bar{E}_{\text{headori}} + \bar{E}_{\text{headvel}} + \bar{E}_{\text{slide}} + \bar{E}_{\text{effort}}$$

Biomechanical Locomotion of Imaginary Creatures

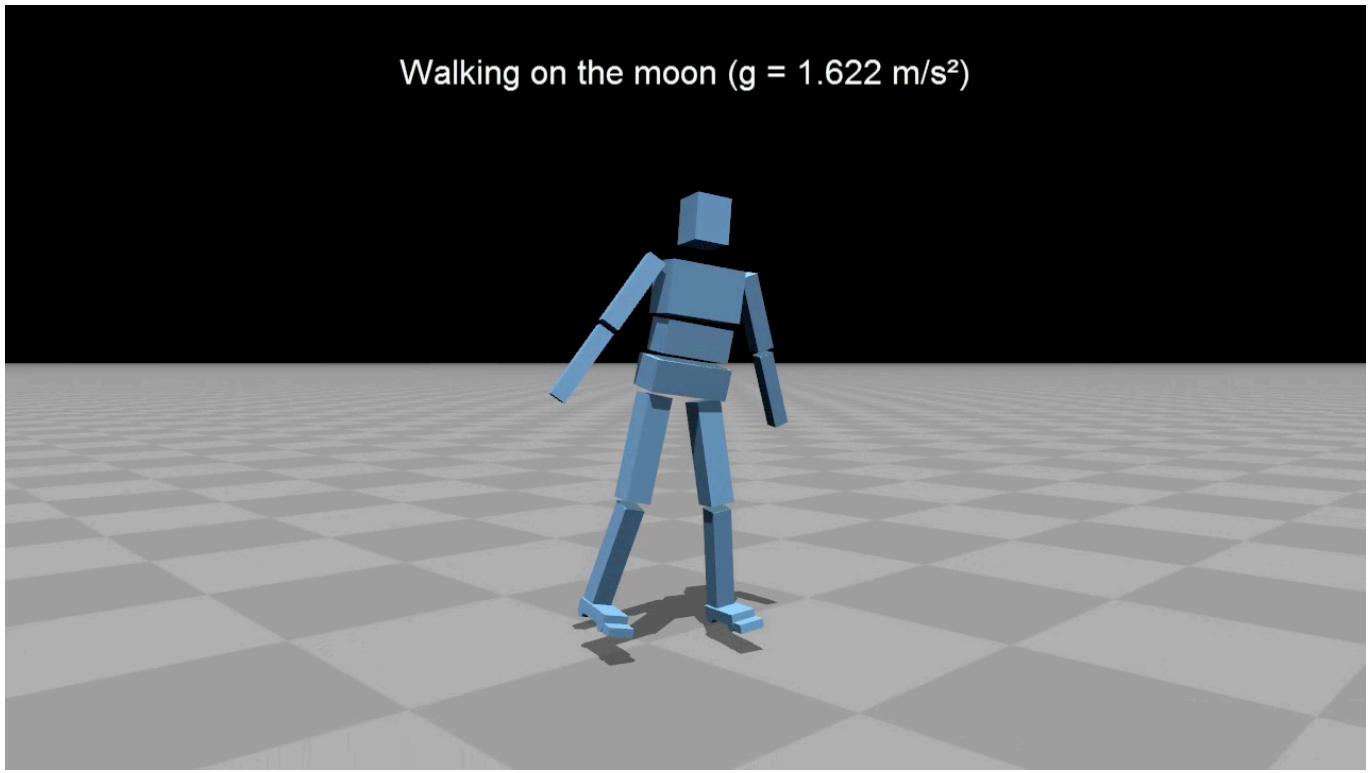


[Flexible Muscle-based Locomotion for Bipedal Creatures, SIGGRAPH ASIA 2013]

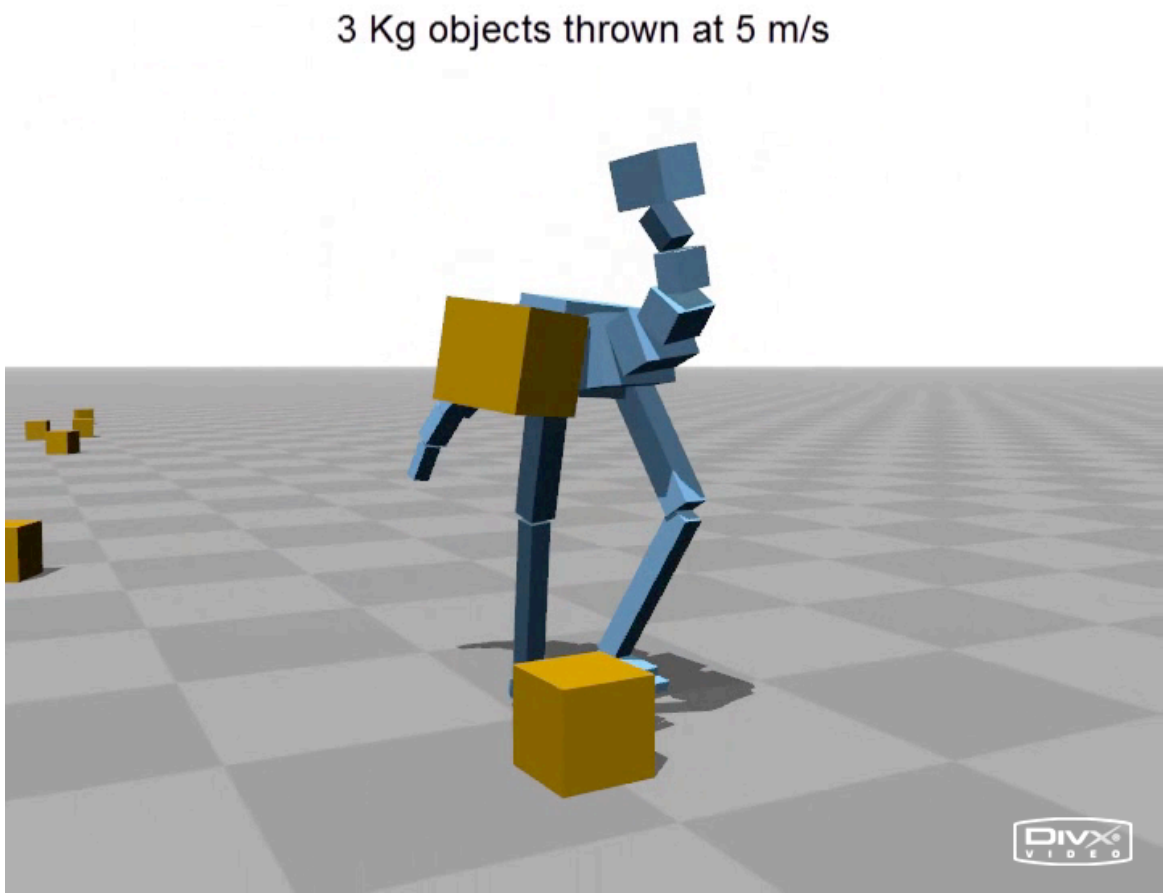
Optimization for Various Speeds



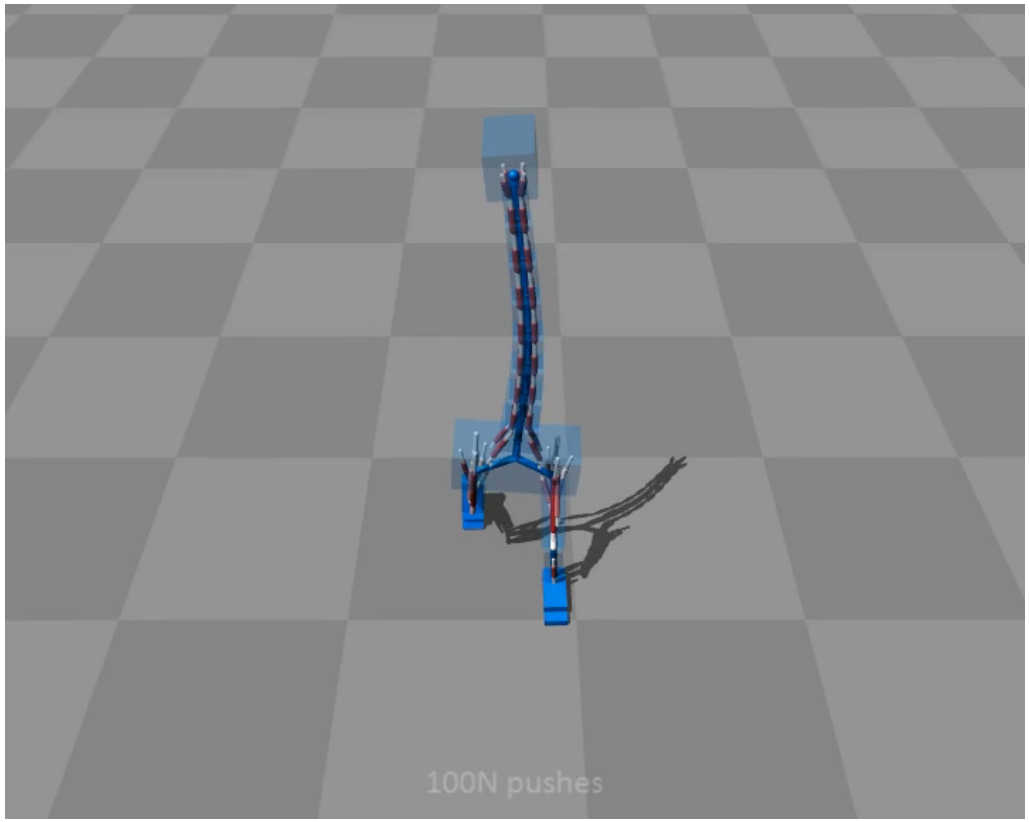
Moon Walk



Robustness



Codesign of Body and Control

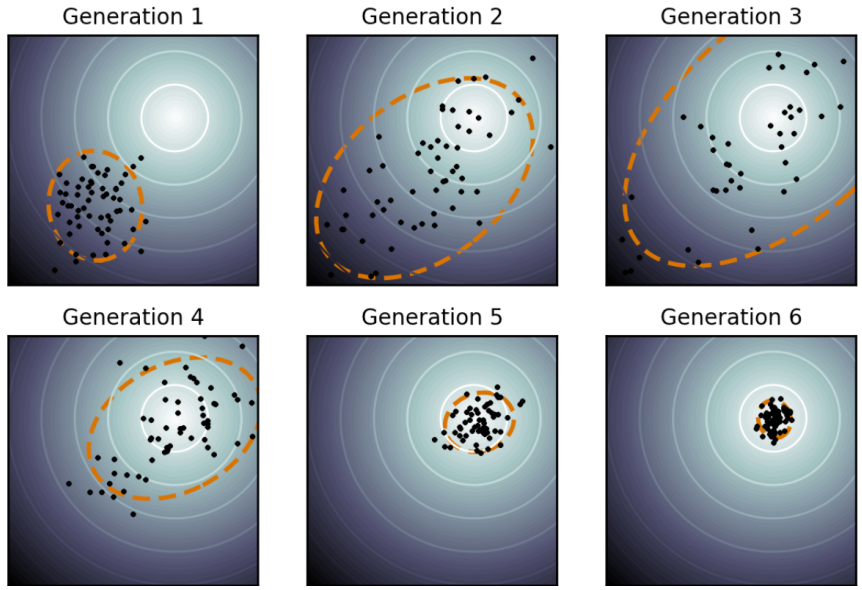


Example Gradient-Free Methods:

CEM: Cross Entropy Method
CMA: Covariance Matrix Adaptation

- Treat the policy parameters directly as defining a high-dimensional search space.
- Model the space of candidate solutions for a given generation using a Normal (or multivariate Gaussian) distribution in this space.
- Evaluate a sample population for their fitness, using 1+ episodes per candidate.
- Keep a set of the best ("elite") samples.
- Move the distribution to fit those elite samples, i.e., maximum likelihood
- repeat for the next generation

CMA
[Wikipedia]



Cross-Entropy Method

$$\max_{\theta} U(\theta) = \max_{\theta} \mathrm{E}[\sum_{t=0}^{H} R(s_t)|\pi_\theta]$$

- Views *U* as a black box
- Ignores all other information other than *U* collected during episode

= evolutionary algorithm

population: $P_{\mu^{(i)}}(\theta)$

CEM:

for iter i = 1, 2, …
  for population member e = 1, 2, …
    sample $\theta^{(e)} \sim P_{\mu^{(i)}}(\theta)$
    execute roll-outs under $\pi_{\theta^{(e)}}$
    store $(\theta^{(e)}, U(e))$
  endfor
  $\mu^{(i+1)} = \arg\max_{\mu} \sum_{\bar{e}} \log P_{\mu}(\theta^{(\bar{e})})$
    where $\bar{e}$ indexes over top p %
endfor

[Abbeel]

Cross-Entropy Method

- Can work embarrassingly well

Method	Mean Score	Reference
Nonreinforcement learning		
Hand-coded	631,167	Dellacherie (Fahey, 2003)
Genetic algorithm	586,103	(Böhm et al., 2004)
Reinforcement learning		
Relational reinforcement learning+kernel-based regression	≈50	Ramon and Driessens (2004)
Policy iteration	3183	Bertsekas and Tsitsiklis (1996)
Least squares policy iteration	<3000	Lagoudakis, Parr, and Littman (2002)
Linear programming + Bootstrap	4274	Farias and van Roy (2006)
Natural policy gradient	≈6800	Kakade (2001)
CE+RL	21,252	
CE+RL, constant noise	72,705	
CE+RL, decreasing noise	348,895	

István Szita and András Lörincz. "Learning Tetris using the noisy cross-entropy method". In: *Neural computation* 18.12 (2006), pp. 2936–2941

Approximate Dynamic Programming Finally Performs Well in the Game of Tetris

Victor Gabillon
INRIA Lille - Nord Europe, Team SequeL, FRANCE
victor.gabillon@inria.fr

Mohammad Ghavamzadeh*
INRIA Lille - Team SequeL & Adobe Research
mohammad.ghavamzadeh@inria.fr

Bruno Scherrer
INRIA Nancy - Grand Est, Team Maia, FRANCE
bruno.scherrer@inria.fr

[NIPS 2013]

Closely Related Approaches

CEM:
for iter i = 1, 2, …
for population member e = 1, 2, …
sample $\theta^{(e)} \sim P_{\mu^{(i)}}(\theta)$
execute roll-outs under $\pi_{\theta^{(e)}}$
store $(\theta^{(e)}, U(e))$
endfor
$\mu^{(i+1)} = \arg\max_{\mu} \sum_{\bar{e}} \log P_{\mu}(\theta^{(\bar{e})})$
where $\bar{e}$ indexes over top p %
endfor

[Abbeel]

- Reward Weighted Regression (RWR)
  - Dayan & Hinton, NC 1997; Peters & Schaal, ICML 2007

$$\mu^{(i+1)} = \arg\max_{\mu} \sum_{e} q(U(e), P_{\mu}(\theta^{(e)})) \log P_{\mu}(\theta^{(e)})$$

- Policy Improvement with Path Integrals (PI^2)
  - PI2: Theodorou, Buchli, Schaal JMLR2010; Kappen, 2007; (PI2-CMA: Stulp & Sigaud ICML2012)

$$\mu^{(i+1)} = \arg\max_{\mu} \sum_{e} \exp(\lambda U(e)) \log P_{\mu}(\theta^{(e)})$$

- Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES)
  - CMA: Hansen & Ostermeier 1996; (CMA-ES: Hansen, Muller, Koumoutsakos 2003)

$$(\mu^{(i+1)}, \Sigma^{(i+1)}) = \arg\max_{\mu,\Sigma} \sum_{\bar{e}} w(U(\bar{e})) \log \mathcal{N}(\theta^{(\bar{e})}; \mu, \Sigma)$$

- PoWER
  - Kober & Peters, NIPS 2007 (also applies importance sampling for sample re-use)

$$\mu^{(i+1)} = \mu^{(i)} + \left(\sum_{e} (\theta^{(e)} - \mu^{(i)}) U(e) \right) / \left(\sum_{e} U(e) \right)$$

Cross-Entropy / Evolutionary Methods

- Full episode evaluation, parameter perturbation
- Simple
- Main caveat: best when intrinsic dimensionality not too high
  - i.e., number of population members comparable to or larger than number of (effective) parameters

→ in practice OK if low-dimensional θ and willing to do do many runs

→ Easy-to-implement baseline, great for comparisons!

[Abbeel]

Considerations

- Pros:
  - Work with arbitrary parameterization, even non-differentiable
  - Embarrassingly easy to parallelize
- Cons:
  - Not very sample efficient since ignores temporal structure

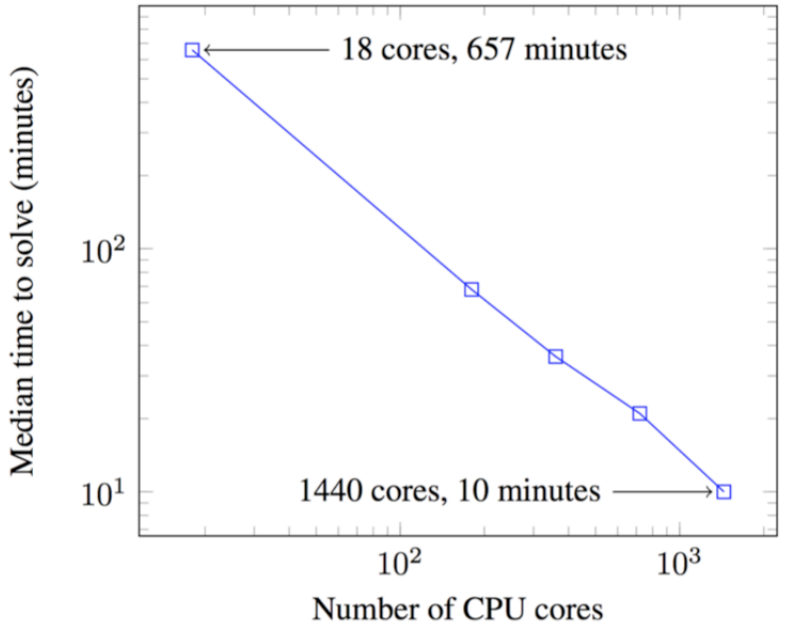
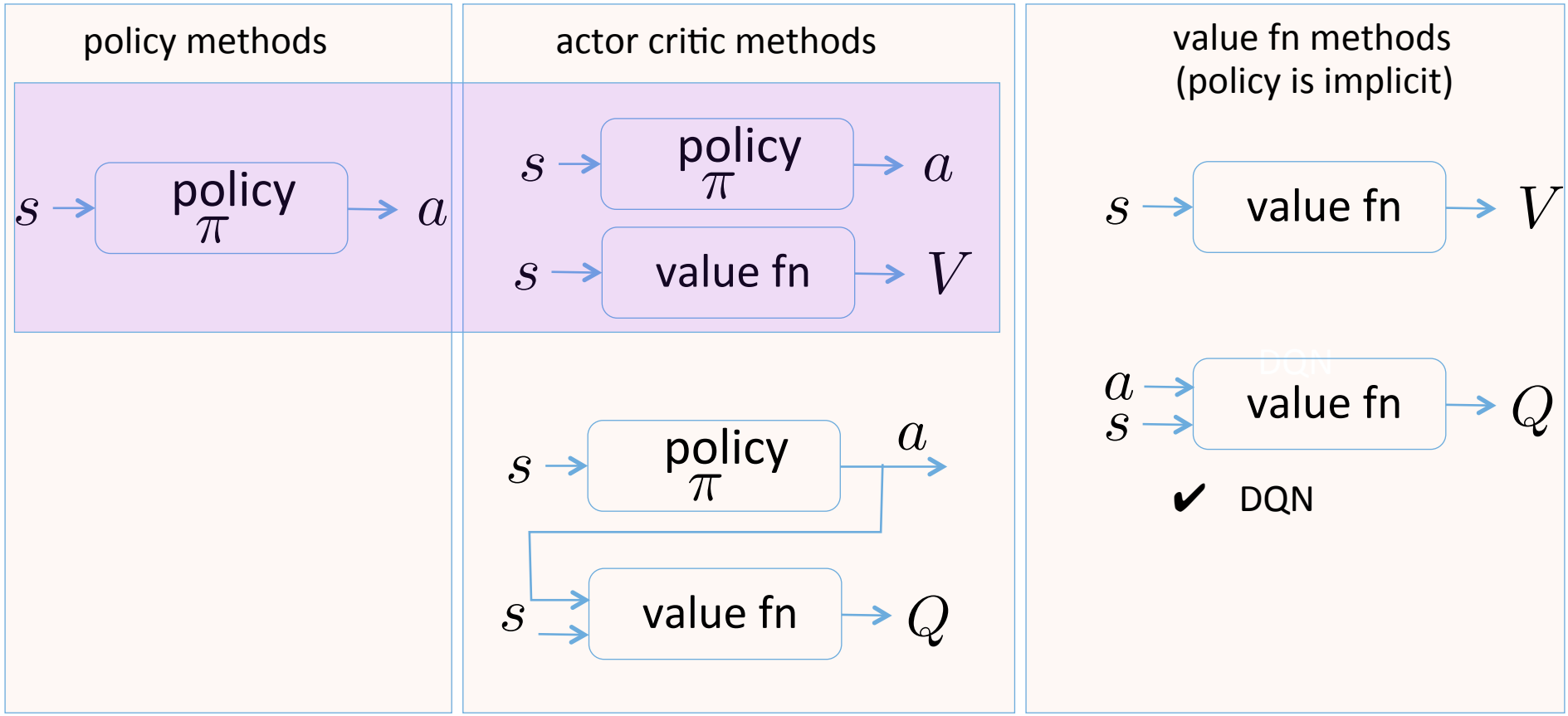


Figure 1. Time to reach a score of 6000 on 3D Humanoid with different number of CPU cores. Experiments are repeated 7 times and median time is reported.

[Salimans, Ho, Chen, Sutskever, 2017]

Policy Gradient Methods



Policy Gradient Algorithms

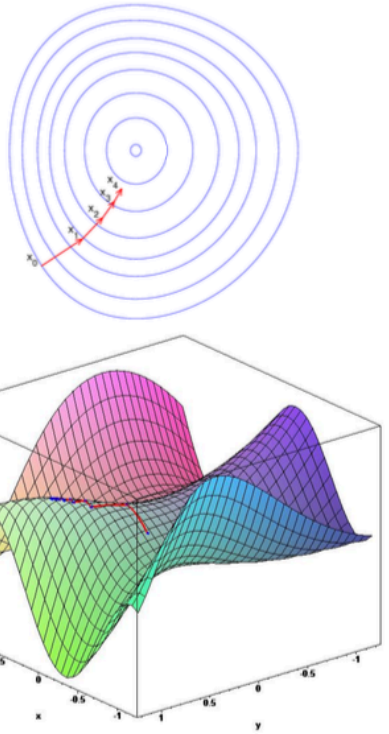
- Let $J(\theta)$ be any policy objective function
- Policy gradient algorithms search for a *local* maximum in $J(\theta)$ by ascending the gradient of the policy, w.r.t. parameters θ

$$\Delta\theta = \alpha \nabla_\theta J(\theta)$$

- Where $\nabla_\theta J(\theta)$ is the policy gradient

$$\nabla_\theta J(\theta) = \begin{pmatrix} \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{pmatrix}$$

- and α is a step-size parameter



[Abbeel]

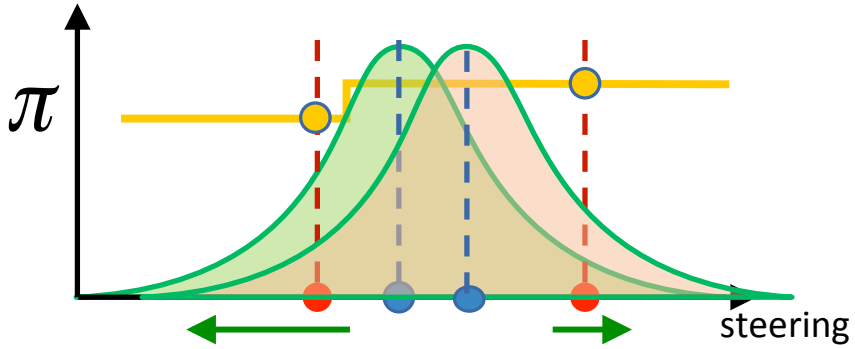
Computing the Policy Gradient via finite-differences

$$\frac{\partial \mathcal{J}(\theta)}{\partial \theta_k} \approx \frac{\mathcal{J}(\theta + \epsilon u_k) - \mathcal{J}(\theta)}{\epsilon}$$

- very slow
  - one gradient computation requires n+1 policy evaluations for n policy parameters
- better idea: get a (noisy) policy gradient for each policy time step!

The intuition:
Consider a one-step reward problem

"make the sampled rewards more likely"

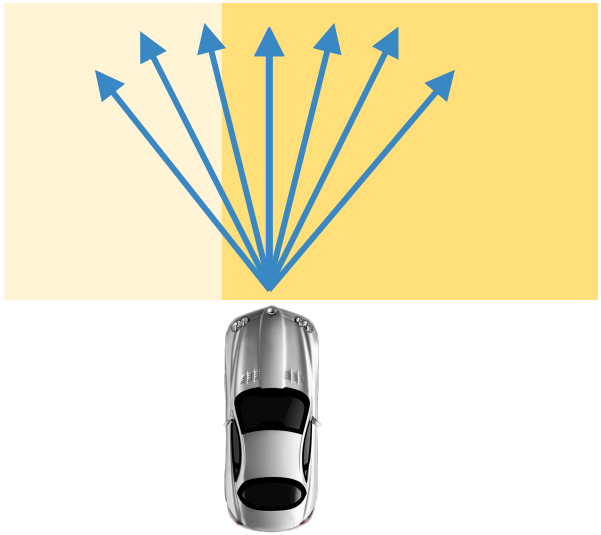


$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) r\right]$$

gradient vectors (score function)

rewards

$$\Delta\theta = \alpha \nabla_\theta J(\theta)$$



Gradient wrt mean of log Normal distribution

$$N(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

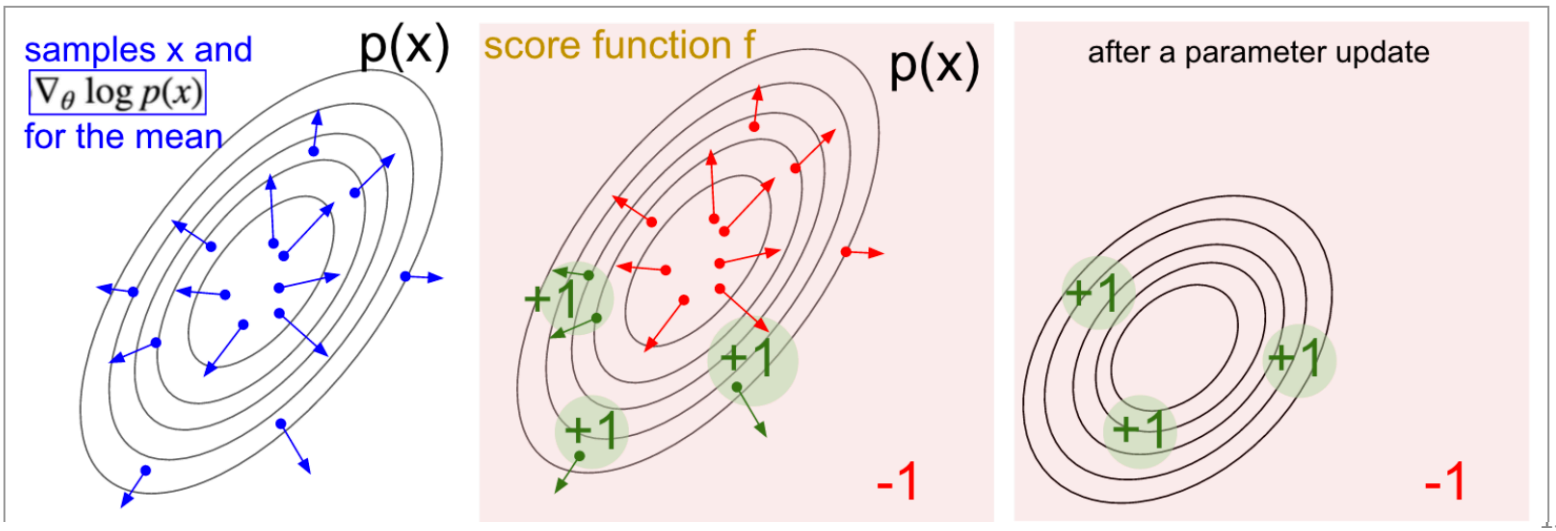
$$\begin{aligned}
\nabla_\mu \ln(N(x)) &= \nabla_\mu \ln(\frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}) \\
&= \nabla_\mu(-\frac{1}{2}\ln(2\pi\sigma^2) - \frac{(x-\mu)^2}{2\sigma^2}) \\
&= \frac{1}{\sigma^2}(x-\mu)
\end{aligned}$$

Intuition in 2D

- Policy is Gaussian, $a \sim \mathcal{N}(\mu(s), \sigma^2)$
- The score function is

$$\nabla_\theta \log \pi_\theta(s, a) = \frac{(a - \mu(s))\nabla_\theta \mu_\theta(s)}{\sigma^2}$$

for a symmetric Gaussian, the blue arrows are just radial vectors from the mean action to the sampled action



The math: Policy Gradient Theorem

$$\nabla_\theta J(\theta) = \nabla_\theta \left[\sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} Q^\pi(s,a) \pi_\theta(a|s) \right]$$

$$\propto \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} Q^\pi(s,a) \nabla_\theta \pi_\theta(a|s)$$

this step uses Likelihood ratios – see below

$$\propto \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} Q^\pi(s,a) \, \pi_\theta(s,a) \nabla_\theta \log \pi_\theta(s,a) = \mathbb{E}\left[\sum_{t=0}^{\infty} Q^\pi(s,a) \nabla_\theta \log \pi_\theta(a_t \mid s_t) \right]$$

- Likelihood ratios exploit the following identity

$$\nabla_\theta \pi_\theta(s,a) = \pi_\theta(s,a) \frac{\nabla_\theta \pi_\theta(s,a)}{\pi_\theta(s,a)}$$
$$= \pi_\theta(s,a) \nabla_\theta \log \pi_\theta(s,a)$$

$$\frac{d}{dx}\left(\ln(x)\right) = \frac{1}{x}$$

$$\frac{d}{dx}\left(\ln f(x)\right) = \frac{f'(x)}{f(x)}$$

Policy Gradient Derivation

[Pieter Abbeel]

We let τ denote a state-action sequence $s_0, u_0, \ldots, s_H, u_H$. We overload notation: $R(\tau) = \sum_{t=0}^{H} R(s_t, u_t)$.

$$U(\theta) = \mathrm{E}[\sum_{t=0}^{H} R(s_t, u_t); \pi_\theta] = \sum_{\tau} P(\tau; \theta) R(\tau)$$

In our new notation, our goal is to find θ:

$$\max_{\theta} U(\theta) = \max_{\theta} \sum_{\tau} P(\tau; \theta) R(\tau)$$

[Abbeel]

$$U(\theta) = \sum_\tau P(\tau;\theta) R(\tau)$$

Taking the gradient w.r.t. θ gives

$$\begin{aligned}
\nabla_\theta U(\theta) &= \nabla_\theta \sum_\tau P(\tau;\theta) R(\tau) \\
&= \sum_\tau \nabla_\theta P(\tau;\theta) R(\tau) \\
&= \sum_\tau \frac{P(\tau;\theta)}{P(\tau;\theta)} \nabla_\theta P(\tau;\theta) R(\tau) \\
&= \sum_\tau P(\tau;\theta) \frac{\nabla_\theta P(\tau;\theta)}{P(\tau;\theta)} R(\tau) \\
&= \sum_\tau P(\tau;\theta) \nabla_\theta \log P(\tau;\theta) R(\tau)
\end{aligned}$$

Approximate with the empirical estimate for m sample paths under policy π_θ:

[Aleksandrov, Sysoyev, & Shemeneva, 1968]
[Rubinstein, 1969]
[Glynn, 1986]
[Reinforce, Williams 1992]
[GPOMDP, Baxter & Bartlett, 2001]

$$\nabla_\theta U(\theta) \approx \hat{g} = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \log P(\tau^{(i)};\theta) R(\tau^{(i)})$$

[Abbeel]

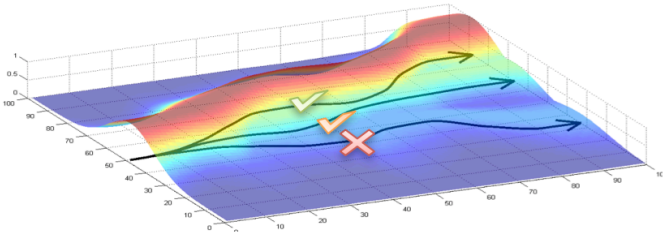
$$\nabla U(\theta) \approx \hat{g} = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \log P(\tau^{(i)}; \theta) R(\tau^{(i)})$$

- Valid even when
  - R is discontinuous and/or unknown
  - Sample space (of paths) is a discrete set

[Abbeel]

$$\nabla U(\theta) \approx \hat{g} = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \log P(\tau^{(i)}; \theta) R(\tau^{(i)})$$

- Gradient tries to:
  - Increase probability of paths with positive R
  - Decrease probability of paths with negative R



! Likelihood ratio changes probabilities of experienced paths, does not try to change the paths (<-> Path Derivative)

Importance Sampling

$$\mathbb{E}_p[f(x)] = \mathbb{E}_q\left(\frac{f(\boldsymbol{X})p(\boldsymbol{X})}{q(\boldsymbol{X})}\right)$$

Derivation from Importance Sampling

[Abbeel]

$$U(\theta) = \mathbb{E}_{\tau \sim \theta_{\text{old}}} \left[\frac{P(\tau|\theta)}{P(\tau|\theta_{\text{old}})} R(\tau) \right]$$

$$\nabla_\theta U(\theta) = \mathbb{E}_{\tau \sim \theta_{\text{old}}} \left[\frac{\nabla_\theta P(\tau|\theta)}{P(\tau|\theta_{\text{old}})} R(\tau) \right]$$

$$\nabla_\theta \, U(\theta)|_{\theta=\theta_{\text{old}}} = \mathbb{E}_{\tau \sim \theta_{\text{old}}} \left[\frac{\nabla_\theta \, P(\tau|\theta)|_{\theta_{\text{old}}}}{P(\tau|\theta_{\text{old}})} R(\tau) \right]$$

$$= \mathbb{E}_{\tau \sim \theta_{\text{old}}} \left[\nabla_\theta \log P(\tau|\theta)|_{\theta_{\text{old}}} \, R(\tau) \right]$$

Decomposing path into state sequences [Abbeel]

$$\nabla_\theta \log P(\tau^{(i)};\theta) = \nabla_\theta \log \left[\prod_{t=0}^{H} \underbrace{P(s_{t+1}^{(i)}|s_t^{(i)},u_t^{(i)})}_{\text{dynamics model}} \cdot \underbrace{\pi_\theta(u_t^{(i)}|s_t^{(i)})}_{\text{policy}} \right]$$

$$= \nabla_\theta \left[\sum_{t=0}^{H} \log P(s_{t+1}^{(i)}|s_t^{(i)},u_t^{(i)}) + \sum_{t=0}^{H} \log \pi_\theta(u_t^{(i)}|s_t^{(i)}) \right]$$

$$= \nabla_\theta \sum_{t=0}^{H} \log \pi_\theta(u_t^{(i)}|s_t^{(i)})$$

$$= \sum_{t=0}^{H} \underbrace{\nabla_\theta \log \pi_\theta(u_t^{(i)}|s_t^{(i)})}_{\text{no dynamics model required!!}}$$

[Abbeel]

The following expression provides us with an unbiased estimate of the gradient, and we can compute it without access to a dynamics model:

$$\hat{g} = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \log P(\tau^{(i)}; \theta) R(\tau^{(i)})$$

Here:

$$\nabla_\theta \log P(\tau^{(i)}; \theta) = \sum_{t=0}^{H} \underbrace{\nabla_\theta \log \pi_\theta(u_t^{(i)} | s_t^{(i)})}_{\text{no dynamics model required!!}}$$

Unbiased means:

$$\mathrm{E}[\hat{g}] = \nabla_\theta U(\theta)$$

REINFORCE

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

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Algorithm parameter: step size $\alpha > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ (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 $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ (G_t)
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$

[Sutton Barto, p328]

Policy gradient methods maximize the expected total reward by repeatedly estimating the gradient $g := \nabla_\theta \mathbb{E}\left[\sum_{t=0}^{\infty} r_t\right]$. There are several different related expressions for the policy gradient, which have the form

$$g = \mathbb{E}\left[\sum_{t=0}^{\infty} \Psi_t \nabla_\theta \log \pi_\theta(a_t \mid s_t)\right], \tag{1}$$

Many choices of "compatible" rewards

where Ψ_t may be one of the following:

1. $\sum_{t=0}^{\infty} r_t$: total reward of the trajectory.
2. $\sum_{t'=t}^{\infty} r_{t'}$: reward following action a_t.
3. $\sum_{t'=t}^{\infty} r_{t'} - b(s_t)$: baselined version of previous formula.
4. $Q^\pi(s_t, a_t)$: state-action value function.
5. $A^\pi(s_t, a_t)$: advantage function.
6. $r_t + V^\pi(s_{t+1}) - V^\pi(s_t)$: TD residual.

The latter formulas use the definitions

$$V^\pi(s_t) := \mathbb{E}_{\substack{s_{t+1:\infty},\\ a_{t:\infty}}}\left[\sum_{l=0}^{\infty} r_{t+l}\right] \qquad Q^\pi(s_t, a_t) := \mathbb{E}_{\substack{s_{t+1:\infty},\\ a_{t+1:\infty}}}\left[\sum_{l=0}^{\infty} r_{t+l}\right] \tag{2}$$

$$A^\pi(s_t, a_t) := Q^\pi(s_t, a_t) - V^\pi(s_t), \quad \text{(Advantage function)}. \tag{3}$$

[Schulman 2016]

Various Policy Gradient Algorithms

- REINFORCE: use Monte Carlo policy returns

1. Initialize θ at random
2. Generate one episode $S_1, A_1, R_2, S_2, A_2, \ldots, S_T$
3. For t=1, 2, … , T:
    1. Estimate the the return G_t since the time step t.
    2. $\theta \leftarrow \theta + \alpha\gamma^t G_t \nabla \ln \pi(A_t|S_t, \theta)$.

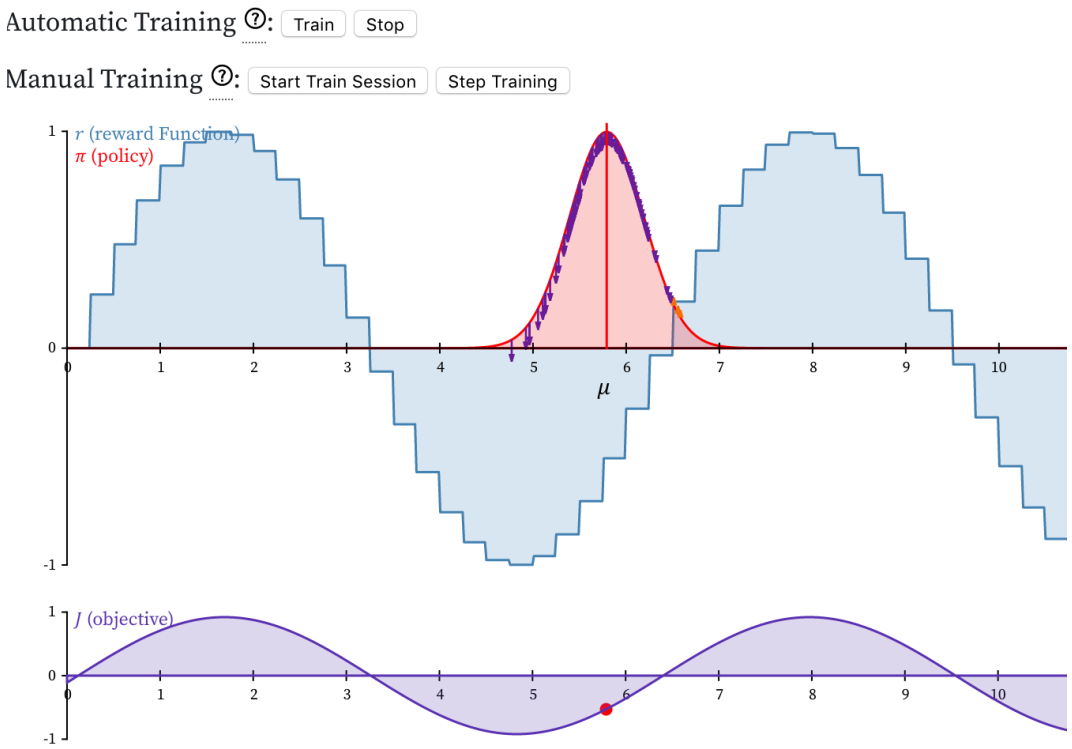
Actor-Critic Policy Gradient Algorithm

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 $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla \ln \pi(A|S,\boldsymbol{\theta})$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$

Policy Gradient Demos

[Farzad Abdolhosseini]
https://observablehq.com/@farzadab/policy-gradients



What's in a step-size?

- Terrible step sizes, always an issue, but how about just not so great ones?

- Supervised learning
  - Step too far → next update will correct for it
- Reinforcement learning
  - Step too far → terrible policy
  - Next mini-batch: collected under this terrible policy!
  - Not clear how to recover short of going back and shrinking the step size

[Abbeel]

The following methods propose ways of avoiding unsafe step sizes:

TRPO: Trust-Region Policy Optimization
PPO: Proximal Policy Optimization

[Schulman]

Reducing reinforcement learning to optimization

- Much of modern ML: reduce learning to numerical optimization problem
  - Supervised learning: minimize training error
- RL: how to *use all data so far and compute the best policy*?
  - Q-learning: can (in principle) include all transitions seen so far, however, we're optimizing the wrong objective
  - Policy gradient methods: yes stochastic gradients, but no optimization problem*
  - This lecture: write down an optimization problem that allows you to do a small update to policy π based on data sampled from π (*on-policy* data)

[Schulman]

What Loss to Optimize?

- Policy gradients

$$\hat{g} = \hat{\mathbb{E}}_t\Big[\nabla_\theta \log \pi_\theta(a_t \mid s_t)\hat{A}_t\Big]$$

- Can differentiate the following loss [practical implementation with Autodiff]

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t\Big[\log \pi_\theta(a_t \mid s_t)\hat{A}_t\Big].$$

but don't want to optimize it too far

- Equivalently differentiate

$$L^{IS}_{\theta_{\text{old}}}(\theta) = \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\hat{A}_t\right].$$

at $\theta = \theta_{\text{old}}$, state-actions are sampled using θ_{old}. (IS = importance sampling)

Just the chain rule: $\nabla_\theta \log f(\theta)\big|_{\theta_{\text{old}}} = \frac{\nabla_\theta f(\theta)\big|_{\theta_{\text{old}}}}{f(\theta_{\text{old}})} = \nabla_\theta\left(\frac{f(\theta)}{f(\theta_{\text{old}})}\right)\Big|_{\theta_{\text{old}}}$

Optimization Methods

- Line search methods
  - find direction
  - select step length
- Trust region methods
  - fix the max step length (defines the “trust region”)
  - find best point in that region
  - locally approximate objective f: linear or quadratic

Trust Region Policy Optimization

[Schulman]

- Define the following trust region update:

$$\begin{aligned} \underset{\theta}{\text{maximize}} \quad & \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \hat{A}_t \right] \\ \text{subject to} \quad & \hat{\mathbb{E}}_t[\text{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]] \leq \delta. \end{aligned}$$

- Also worth considering using a penalty instead of a constraint

$$\underset{\theta}{\text{maximize}} \quad \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \hat{A}_t \right] - \beta \hat{\mathbb{E}}_t[\text{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]]$$

- Method of Lagrange multipliers: optimality point of δ-constrained problem is also an optimality point of β-penalized problem for some β.
- In practice, δ is easier to tune, and fixed δ is better than fixed β

[Schulman]

Trust Region Policy Optimization: Pseudocode

- Pseudocode:

  for iteration=$1, 2, \ldots$ **do**

  Run policy for T timesteps or N trajectories

  Estimate advantage function at all timesteps

$$\underset{\theta}{\text{maximize}} \sum_{n=1}^{N} \frac{\pi_\theta(a_n \mid s_n)}{\pi_{\theta_{\text{old}}}(a_n \mid s_n)} \hat{A}_n$$

$$\text{subject to} \quad \overline{\text{KL}}_{\pi_{\theta_{\text{old}}}}(\pi_\theta) \leq \delta$$

  end for

- Can solve constrained optimization problem efficiently by using conjugate gradient
- Closely related to natural policy gradients (Kakade, 2002), natural actor critic (Peters and Schaal, 2005), REPS (Peters et al., 2010)

[Schulman]

"Proximal" Policy Optimization: KL Penalty Version

- Use penalty instead of constraint

$$\underset{\theta}{\text{maximize}} \sum_{n=1}^{N} \frac{\pi_\theta(a_n \mid s_n)}{\pi_{\theta_{\text{old}}}(a_n \mid s_n)} \hat{A}_n - C \cdot \overline{\text{KL}}_{\pi_{\theta_{\text{old}}}}(\pi_\theta)$$

- Pseudocode:

  for iteration=1, 2, . . . **do**
    Run policy for T timesteps or N trajectories
    Estimate advantage function at all timesteps
    Do SGD on above objective for some number of epochs
    If KL too high, increase β. If KL too low, decrease β.
  end for

- $\approx$ same performance as TRPO, but only first-order optimization

[Schulman]

Limitations of TRPO

- Hard to use with architectures with multiple outputs, e.g. policy and value function (need to weight different terms in distance metric)
- Empirically performs poorly on tasks requiring deep CNNs and RNNs, e.g. Atari benchmark
- CG makes implementation more complicated

[Schulman]

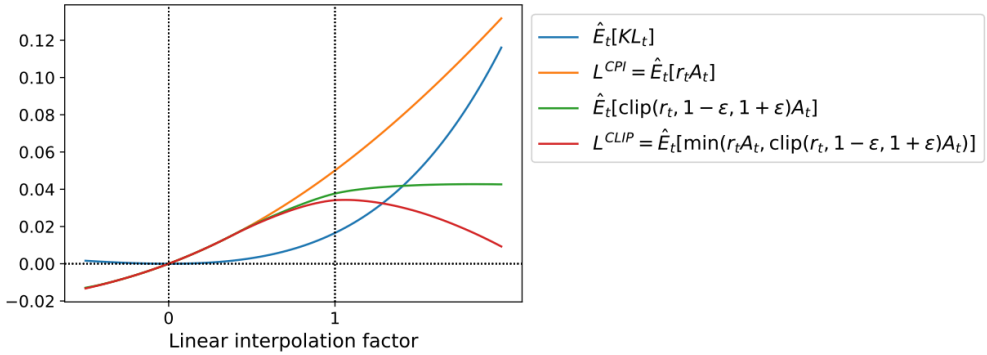
Proximal Policy Optimization: Clipping Objective

- Recall the surrogate objective

$$L^{IS}(\theta) = \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\hat{A}_t\right] = \hat{\mathbb{E}}_t\left[r_t(\theta)\hat{A}_t\right]. \tag{1}$$

- Form a lower bound via clipped importance ratios

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right] \tag{2}$$



- Forms pessimistic bound on objective, can be optimized using SGD

PyTorch Implementation

```
prob = Fnn.softmax(pi)
log_prob = Fnn.log_softmax(pi)
action_prob = prob.gather(1, action)

prob_old = Fnn.softmax(pi_old)
action_prob_old = prob_old.gather(1, action)

ratio = action_prob / (action_prob_old + 1e-10)

advantage = (advantage - advantage.mean()) / (advantage.std() + 1e-5)

surr1 = ratio * advantage
surr2 = torch.clamp(ratio, min=1. - clip, max=1. + clip) * advantage

policy_loss = -torch.min(surr1, surr2).mean()
```

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

Interpreting the PPO objective

[OpenAI: https://spinningup.openai.com/en/latest/algorithms/ppo.html]

positive advantage $$L(s, a, \theta_k, \theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, (1+\epsilon)\right) A^{\pi_{\theta_k}}(s, a)$$

negative advantage $$L(s, a, \theta_k, \theta) = \max\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, (1-\epsilon)\right) A^{\pi_{\theta_k}}(s, a)$$

What we have seen so far is that clipping serves as a regularizer by removing incentives for the policy to change dramatically, and the hyperparameter ϵ corresponds to how far away the new policy can go from the old while still profiting the objective.

[Schulman]

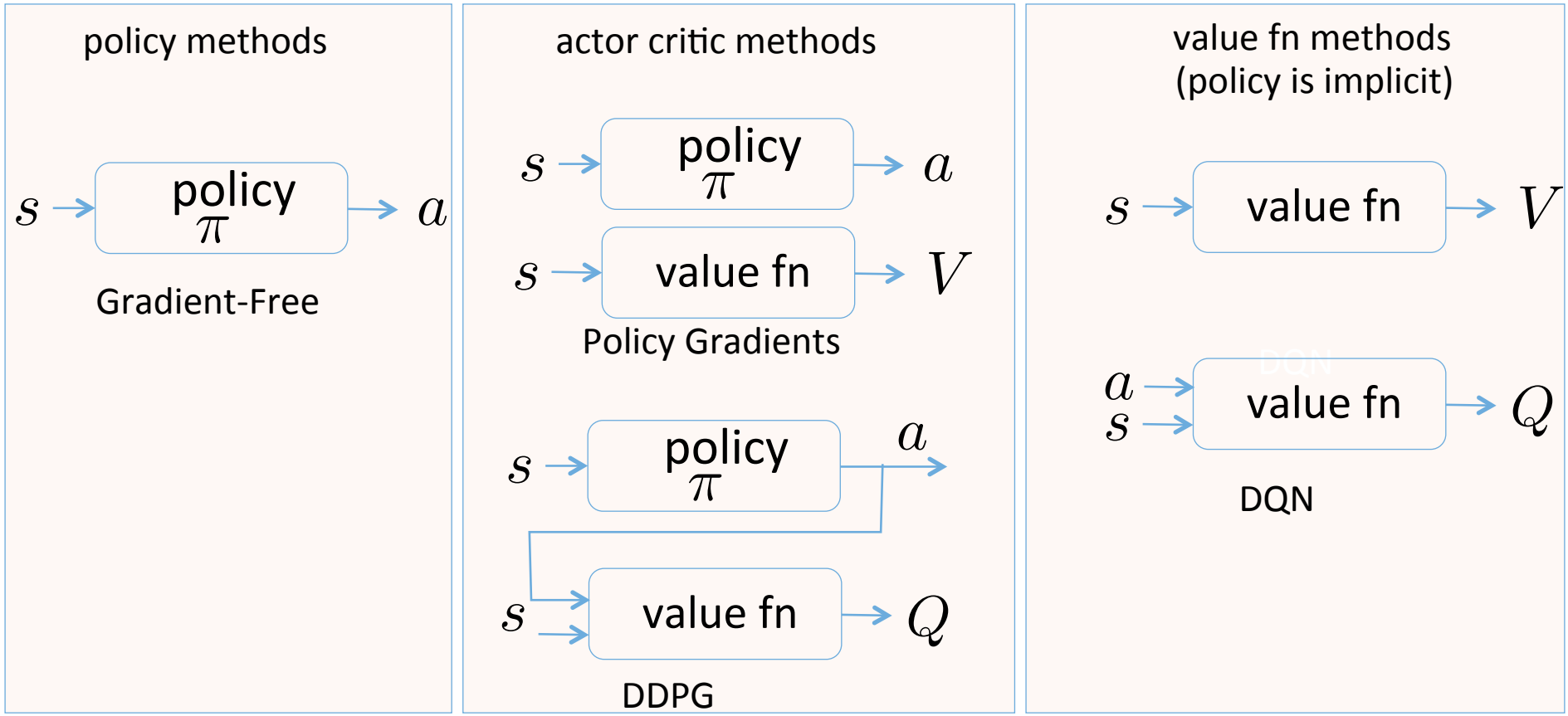
Proximal Policy Optimization

- Pseudocode:

  for iteration=$1, 2, \ldots$ **do**
  Run policy for T timesteps or N trajectories
  Estimate advantage function at all timesteps
  Do SGD on $L^{CLIP}(\theta)$ objective for some number of epochs
  end for

- A bit better than TRPO on continuous control, much better on Atari
- Compatible with multi-output networks and RNNs

SUMMARY


policy methods
s
policy
π
a
Gradient-Free
actor critic methods
s
policy
π
a
s
value fn
V
Policy Gradients
s
policy
π
a
s
value fn
Q
DDPG
value fn methods
(policy is implicit)
s
value fn
V
a
s
value fn
Q
DQN

Practical Advice for working with RL

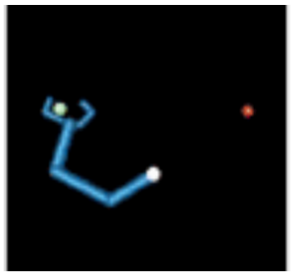
- Techniques from supervised learning don't necessarily work in RL:
  - batch norm, dropout, big networks
- use small test problems; experiment quickly
- interpret and visualize the learning process: state visitation, value fn
- new task? make it easier until there are signs of life
- env design: visualize random policy
- explore sensitivity to parameters
- use multiple random # seeds
- automate experiments; consider using cloud computing
- much more at the link below

~[Schulman https://drive.google.com/file/d/0BxXI_RttTZAhc2ZsblNvUHhGZDA/view]

Local minina

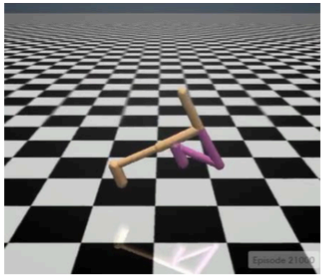
- RL can (and will often) get stuck in local minima
- mitigate this via
  - sufficient exploration
  - early termination
  - reward shaping
  - entropy bonus
  - demonstrations

Rewards functions in theory and practice



$$r(\mathbf{x}, \mathbf{u}) = \begin{cases} 1 \text{ if object at target} \\ 0 \text{ otherwise} \end{cases}$$

$$\begin{aligned} r(\mathbf{x}, \mathbf{u}) = &- w_1 \|p_{\text{gripper}}(\mathbf{x}) - p_{\text{object}}(\mathbf{x})\|^2 + \\ &- w_2 \|p_{\text{object}}(\mathbf{x}) - p_{\text{target}}(\mathbf{x})\|^2 + \\ &- w_3 \|\mathbf{u}\|^2 \end{aligned}$$



$$r(\mathbf{x}, \mathbf{u}) = \begin{cases} 1 \text{ if walker is running} \\ 0 \text{ otherwise} \end{cases}$$

$$\begin{aligned} r(\mathbf{x}, \mathbf{u}) = &w_1 v(\mathbf{x}) + \\ &w_2 \delta(|\theta_{\text{torso}}(\mathbf{x})| < \epsilon) + \\ &w_3 \delta(h_{\text{torso}}(\mathbf{x}) \geq h) \end{aligned}$$

[Levine]

Another view of policy gradients

Our first generic candidate for solving reinforcement learning is *Policy Gradient*. I find it shocking that Policy Gradient wasn't ruled out as a bad idea in 1993. Policy gradient is seductive as it apparently lets one fine tune a program to solve any problem without any domain knowledge. Of course, anything that makes such a claim must be too general for its own good. Indeed, if you dive into it, **policy gradient is nothing more than random search dressed up in mathematical symbols and lingo**.

Lots of papers have been applying policy gradient to all sorts of different settings, and claiming crazy results, but I hope that it is now clear that they are just dressing up random search in a clever outfit. When you end up with a bunch of papers showing that genetic algorithms are competitive with your methods, this does not mean that we've made an advance in genetic algorithms. It is far more likely that this means that your method is a lousy implementation of random search.

[An Outsider's Tour of Reinforcement Learning
http://www.argmin.net/2018/06/25/outsider-rl/ – Ben Recht]

Simple random search of static linear policies is competitive for reinforcement learning

Horia Mania hmania@berkeley.edu
Aurelia Guy lia@berkeley.edu
Benjamin Recht brecht@berkeley.edu

Department of Electrical Engineering and Computer Science
University of California, Berkeley

Policy Gradient changes the objective

$$\text{maximize}_u \quad R(u) \qquad \text{vs} \qquad \text{maximize}_{p(u)} \quad \mathbb{E}_p[R(u)]$$

I don't think I can overemphasize the point that policy gradient and RL are not magic. I'd go as far as to say that policy gradient and its derivatives are legitimately bad algorithms. In order to make them work well, you need lots of tricks. Algorithms which are hard to tune, hard to reproduce, and don't outperform off the shelf genetic algorithms are bad algorithms.

[An Outsider's Tour of Reinforcement Learning
http://www.argmin.net/2018/06/25/outsider-rl/ − Ben Recht]

Intuition in 2D

- Policy is Gaussian, $a \sim \mathcal{N}(\mu(s), \sigma^2)$
- The score function is

$$\nabla_\theta \log \pi_\theta(s, a) = \frac{(a - \mu(s))\nabla_\theta \mu_\theta(s)}{\sigma^2}$$

for a symmetric Gaussian, the blue arrows are just radial vectors from the mean action to the sampled action