

Lecture 7: Policy Gradient

David Silver

Outline

- 1 Introduction
- 2 Finite Difference Policy Gradient
- 3 Monte-Carlo Policy Gradient
- 4 Actor-Critic Policy Gradient

Policy-Based Reinforcement Learning

- In the last lecture we approximated the value or action-value function using parameters θ ,

$$V_{\theta}(s) \approx V^{\pi}(s)$$

$$Q_{\theta}(s, a) \approx Q^{\pi}(s, a)$$

- A policy was generated directly from the value function
 - e.g. using ϵ -greedy
- In this lecture we will directly parametrise the **policy**

$$\pi_{\theta}(s, a) = \mathbb{P}[a \mid s, \theta]$$

- We will focus again on **model-free** reinforcement learning

Value-Based and Policy-Based RL

advantage of policy gradient over value fn:

- policy can be more compact when the actual reward might be hard to calculate
- has better convergence properties (following gradient is more stable)

■ Value Based

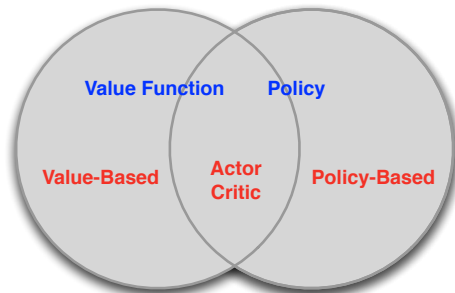
- Learnt Value Function
- Implicit policy (e.g. ϵ -greedy)

■ Policy Based

- No Value Function
- Learnt Policy

■ Actor-Critic

- Learnt Value Function
- Learnt Policy



Advantages of Policy-Based RL

Advantages:

- Better convergence properties
- Effective in high-dimensional or continuous action spaces
- Can learn stochastic policies

Q-learning requires the $\max_a \{Q(s,a)\}$ ->
is complex when working with continuous action spaces

Disadvantages:

- Typically converge to a local rather than global optimum
- Evaluating a policy is typically inefficient and high variance

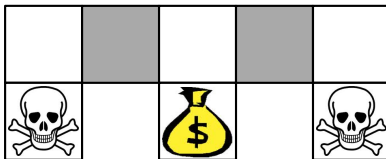
Example: Rock-Paper-Scissors



- Two-player game of rock-paper-scissors
 - Scissors beats paper
 - Rock beats scissors
 - Paper beats rock
- Consider policies for *iterated* rock-paper-scissors
 - A deterministic policy is easily exploited
 - A uniform random policy is optimal (i.e. Nash equilibrium)

optimal behaviour is stochastic, so that stochastic policy is advantageous

Example: Aliased Gridworld (1)



- The agent cannot differentiate the grey states
- Consider features of the following form (for all N, E, S, W)

$$\phi(s, a) = \mathbf{1}(\text{wall to } N, a = \text{move } E)$$

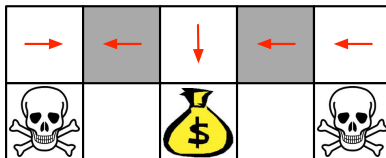
- Compare value-based RL, using an approximate value function

$$Q_{\theta}(s, a) = f(\phi(s, a), \theta)$$

- To policy-based RL, using a parametrised policy

$$\pi_{\theta}(s, a) = g(\phi(s, a), \theta)$$

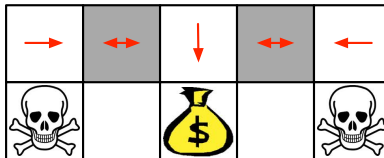
Example: Aliased Gridworld (2)



deterministic policy is bad
because it cannot differentiate
states that look the same

- Under aliasing, an optimal **deterministic** policy will either
 - move W in both grey states (shown by red arrows)
 - move E in both grey states
- Either way, it can get stuck and *never* reach the money
- Value-based RL learns a near-deterministic policy
 - e.g. greedy or ϵ -greedy
- So it will traverse the corridor for a long time

Example: Aliased Gridworld (3)



- An optimal **stochastic** policy will randomly move E or W in grey states

$$\pi_{\theta}(\text{wall to N and S, move E}) = 0.5$$

$$\pi_{\theta}(\text{wall to N and S, move W}) = 0.5$$

- It will reach the goal state in a few steps with high probability
- Policy-based RL can learn the optimal stochastic policy

Policy Objective Functions

policy objective functions is our optimization function. It can measure three different things given on this slide

evaluate existing policy π_θ finding best θ

- Goal: given policy $\pi_\theta(s, a)$ with parameters θ , find best θ
- But how do we measure the quality of a policy π_θ ?
- In episodic environments we can use the **start value**

$$J_1(\theta) = V^{\pi_\theta}(s_1) = \mathbb{E}_{\pi_\theta} [v_1]$$

- In continuing environments we can use the **average value**

$$J_{avV}(\theta) = \sum_s d^{\pi_\theta}(s) V^{\pi_\theta}(s)$$

- Or the **average reward per time-step**

$$J_{avR}(\theta) = \sum_s d^{\pi_\theta}(s) \sum_a \pi_\theta(s, a) \mathcal{R}_s^a$$

- where $d^{\pi_\theta}(s)$ is **stationary distribution** of Markov chain for π_θ

Policy Optimisation

- Policy based reinforcement learning is an **optimisation** problem
- Find θ that maximises $J(\theta)$ **objective function**
- Some approaches do not use gradient
 - Hill climbing
 - Simplex / amoeba / Nelder Mead **gradient-free optimization methods**
 - Genetic algorithms
- Greater efficiency often possible using gradient
 - Gradient descent
 - Conjugate gradient
 - Quasi-newton **with little computation the simple gradient descent is an effective way to solve the optimisation problem**
- We focus on gradient descent, many extensions possible
- And on methods that exploit sequential structure

Policy Gradient

- 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

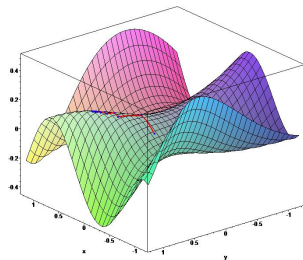
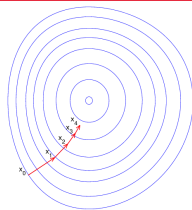
$$\Delta\theta = \alpha \nabla_{\theta} J(\theta)$$

- 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

gradient ascent here, since
maximisation of $J(\theta)$ is the goal



Computing Gradients By Finite Differences

- To evaluate policy gradient of $\pi_{\theta}(s, a)$
- For each dimension $k \in [1, n]$
 - Estimate k th partial derivative of objective function w.r.t. θ
 - By perturbing θ by small amount ϵ in k th dimension

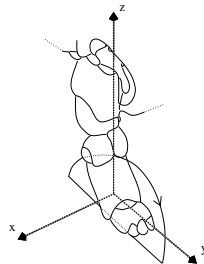
$$\frac{\partial J(\theta)}{\partial \theta_k} \approx \frac{J(\theta + \epsilon u_k) - J(\theta)}{\epsilon}$$

where u_k is unit vector with 1 in k th component, 0 elsewhere

- Uses n evaluations to compute policy gradient in n dimensions
- Simple, noisy, inefficient - but sometimes effective
- Works for arbitrary policies, even if policy is not differentiable

this approach tends to collapse in environments of higher dimensionality

Training AIBO to Walk by Finite Difference Policy Gradient



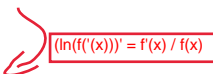
- Goal: learn a fast AIBO walk (useful for Robocup)
- AIBO walk policy is controlled by 12 numbers (elliptical loci)
- Adapt these parameters by finite difference policy gradient
- Evaluate performance of policy by field traversal time

AIBO Walk Policies

- Before training
- During training
- After training

Score Function

- We now compute the policy gradient *analytically*
- Assume policy π_θ is differentiable whenever it is non-zero
- and we know the gradient $\nabla_\theta \pi_\theta(s, a)$
- **Likelihood ratios** exploit the following identity

$$\begin{aligned}\nabla_\theta \pi_\theta(s, a) &= \pi_\theta(s, a) \frac{\nabla_\theta \pi_\theta(s, a)}{\pi_\theta(s, a)} \\ &= \pi_\theta(s, a) \nabla_\theta \log \pi_\theta(s, a)\end{aligned}$$


$(\ln(f'(x)))' = f'(x) / f(x)$

- The **score function** is $\nabla_\theta \log \pi_\theta(s, a)$

maximizes the likelihood of a is this term
How to adjust policy to get more reward

Softmax Policy

softmax is used to squash real values to values between (0, 1)

alternative to epsilon greedy

- We will use a softmax policy as a running example
- Weight actions using linear combination of features $\phi(s, a)^\top \theta$
- Probability of action is proportional to exponentiated weight

proportional to some exponential value $\pi_\theta(s, a) \propto e^{\phi(s, a)^\top \theta}$

- The score function is

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

Gaussian Policy

- In continuous action spaces, a Gaussian policy is natural
- Mean is a linear combination of state features $\mu(s) = \phi(s)^\top \theta$
- Variance may be fixed σ^2 , or can also be parametrised
- Policy is Gaussian, $a \sim \mathcal{N}(\mu(s), \sigma^2)$
- The score function is

$$\nabla_{\theta} \log \pi_{\theta}(s, a) = \frac{\overbrace{(a - \mu(s))}^{\text{how much more are we taking a particular action}} \underbrace{\phi(s)}_{\text{feature of that state}}}{\underbrace{\sigma^2}_{\text{scale it by the variance}}}$$

One-Step MDPs

- Consider a simple class of **one-step** MDPs
 - Starting in state $s \sim d(s)$
 - Terminating after one time-step with reward $r = \mathcal{R}_{s,a}$
- Use likelihood ratios to compute the policy gradient

start with expectation

$$J(\theta) = \mathbb{E}_{\pi_{\theta}} [r]$$

$$= \sum_{s \in \mathcal{S}} d(s) \sum_{a \in \mathcal{A}} \pi_{\theta}(s, a) \mathcal{R}_{s,a}$$

multiply with gradient on both sides and use log identity
-> possible because only policy depends on theta

use gradient

$$\nabla_{\theta} J(\theta) = \sum_{s \in \mathcal{S}} d(s) \sum_{a \in \mathcal{A}} \pi_{\theta}(s, a) \nabla_{\theta} \log \pi_{\theta}(s, a) \mathcal{R}_{s,a}$$

recover expectation

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

expectation of the score times the reward
-> we need to move in the direction of the score times the reward

Policy Gradient Theorem

- The policy gradient theorem generalises the likelihood ratio approach to multi-step MDPs
- Replaces instantaneous reward r with long-term value $Q^\pi(s, a)$
- Policy gradient theorem applies to start state objective, average reward and average value objective

Theorem

*For any differentiable policy $\pi_\theta(s, a)$,
for any of the policy objective functions $J = J_1, J_{avR}$, or $\frac{1}{1-\gamma} J_{avV}$,
the policy gradient is*

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

how to adjust the policy in order to move closer to the particular action that was taken when the action returned positive action-value fn value

Monte-Carlo Policy Gradient (REINFORCE)

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

v_t accumulated reward
from timestep t

$$\Delta\theta_t = \alpha \nabla_{\theta} \log \pi_{\theta}(s_t, a_t) v_t$$

function REINFORCE

Initialise θ arbitrarily

forward view algorithm - run complete episodes following policy π_{θ}

for each episode $\{s_1, a_1, r_2, \dots, s_{T-1}, a_{T-1}, r_T\} \sim \pi_{\theta}$ **do**

for $t = 1$ to $T - 1$ **do**

$\theta \leftarrow \theta + \alpha \nabla_{\theta} \log \pi_{\theta}(s_t, a_t) v_t$

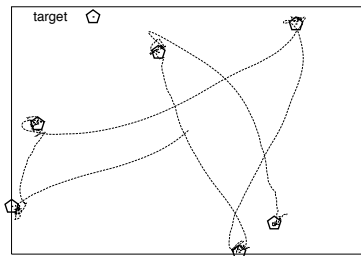
end for

end for

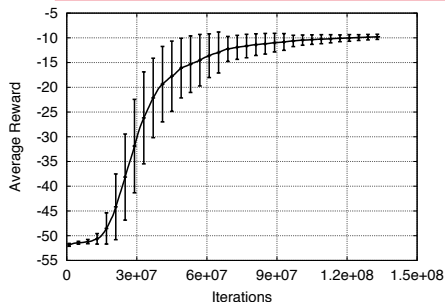
return θ

end function

Puck World Example



smooth learning curve in comparison to value iteration
but it is slow
-> make gradient policy ideas more efficient is the goal of this class



- Continuous actions exert small force on puck
- Puck is rewarded for getting close to target
- Target location is reset every 30 seconds
- Policy is trained using variant of Monte-Carlo policy gradient

Reducing Variance Using a Critic

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

combine value fn approximation
and use it in our policy gradient algorithm

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

- Actor-critic algorithms maintain *two* sets of parameters

Critic Updates action-value function parameters w

Actor Updates policy parameters θ , in direction suggested by critic

actor - takes decisions in
the environment

critic - views what the
actor does and evaluates it

- Actor-critic algorithms follow an *approximate* policy gradient

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

$$\Delta \theta = \alpha \nabla_\theta \log \pi_\theta(s, a) Q_w(s, a)$$

Estimating the Action-Value Function

- The critic is solving a familiar problem: policy evaluation
- How good is policy π_θ for current parameters θ ?
- This problem was explored in previous two lectures, e.g.
 - Monte-Carlo policy evaluation
 - Temporal-Difference learning
 - $TD(\lambda)$
- Could also use e.g. least-squares policy evaluation

Action-Value Actor-Critic

w - critic parameters

θ - actor parameters

- Simple actor-critic algorithm based on action-value critic
- Using linear value fn approx. $Q_w(s, a) = \phi(s, a)^\top w$
 - Critic** Updates w by linear TD(0)
 - Actor** Updates θ by policy gradient

function QAC

Initialise s, θ

Sample $a \sim \pi_\theta$

online algorithm because TD(0) is used

for each step **do**

Sample reward $r = \mathcal{R}_s^a$; sample transition $s' \sim \mathcal{P}_{s'}^a$.

Sample action $a' \sim \pi_\theta(s', a')$

$\delta = r + \gamma Q_w(s', a') - Q_w(s, a)$

$\theta = \theta + \alpha \nabla_\theta \log \pi_\theta(s, a) Q_w(s, a)$

$w \leftarrow w + \beta \delta \phi(s, a)$

$a \leftarrow a', s \leftarrow s'$

step size controls the smoothness of that algorithm. imagine an infinite step size - that can be understood as acting greedily on state-action fn

end for

end function

Bias in Actor-Critic Algorithms

- Approximating the policy gradient introduces bias
- A biased policy gradient may not find the right solution
 - e.g. if $Q_w(s, a)$ uses aliased features, can we solve gridworld example?
- Luckily, if we choose value function approximation carefully
- Then we can avoid introducing any bias
- i.e. We can still follow the *exact* policy gradient

Compatible Function Approximation

Theorem (Compatible Function Approximation Theorem)

If the following two conditions are satisfied:

- 1 Value function approximator is *compatible* to the policy

$$\nabla_w Q_w(s, a) = \nabla_\theta \log \pi_\theta(s, a)$$

- 2 Value function parameters w minimise the mean-squared error

$$\varepsilon = \mathbb{E}_{\pi_\theta} [(Q^{\pi_\theta}(s, a) - Q_w(s, a))^2]$$

Then the policy gradient is exact,

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

Proof of Compatible Function Approximation Theorem

If w is chosen to minimise mean-squared error, gradient of ε w.r.t. w must be zero,

$$\nabla_w \varepsilon = 0$$

$$\mathbb{E}_{\pi_\theta} [(Q^\theta(s, a) - Q_w(s, a)) \nabla_w Q_w(s, a)] = 0$$

$$\mathbb{E}_{\pi_\theta} [(Q^\theta(s, a) - Q_w(s, a)) \nabla_\theta \log \pi_\theta(s, a)] = 0$$

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

So $Q_w(s, a)$ can be substituted directly into the policy gradient,

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

Reducing Variance Using a Baseline

- We subtract a baseline function $B(s)$ from the policy gradient
- This can reduce variance, without changing expectation

$$\begin{aligned}
 \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(s, a) B(s)] &= \sum_{s \in \mathcal{S}} d^{\pi_{\theta}}(s) \sum_a \nabla_{\theta} \pi_{\theta}(s, a) B(s) \\
 &= \sum_{s \in \mathcal{S}} d^{\pi_{\theta}}(s) B(s) \nabla_{\theta} \sum_{a \in \mathcal{A}} \pi_{\theta}(s, a) \\
 &= 0
 \end{aligned}$$

instead of judging absolute scores, baselines can be used to judge scores relatively e.g. $[-1, 1]$, e.g. use average reward as baseline or value fn to get the advantage fn

- A good baseline is the state value function $B(s) = V^{\pi_{\theta}}(s)$
- So we can rewrite the policy gradient using the **advantage function** $A^{\pi_{\theta}}(s, a)$

$$A^{\pi_{\theta}}(s, a) = Q^{\pi_{\theta}}(s, a) - V^{\pi_{\theta}}(s)$$

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

if $A > 0$, the gradient moves policy toward that action
if $A < 0$, the gradient moves policy away from that action

Estimating the Advantage Function (1)

- The advantage function can significantly reduce variance of policy gradient
- So the critic should really estimate the advantage function
- For example, by estimating *both* $V^{\pi_\theta}(s)$ and $Q^{\pi_\theta}(s, a)$
- Using two function approximators and two parameter vectors,

$$\begin{aligned}V_v(s) &\approx V^{\pi_\theta}(s) \\ Q_w(s, a) &\approx Q^{\pi_\theta}(s, a) \\ A(s, a) &= Q_w(s, a) - V_v(s)\end{aligned}$$

- And updating *both* value functions by e.g. TD learning

Estimating the Advantage Function (2)

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

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

- is an unbiased estimate of the advantage function

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

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

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

- In practice we can use an approximate TD error

$$\delta_v = r + \gamma V_v(s') - V_v(s)$$

only estimate V

- one set of parameters
- generate TD error
- move towards the direction of the TD error with gradient

- This approach only requires one set of critic parameters v

Critics at Different Time-Scales

- Critic can estimate value function $V_\theta(s)$ from many targets at different time-scales From last lecture...

- For MC, the target is the return v_t

$$\Delta\theta = \alpha(v_t - V_\theta(s))\phi(s)$$

- For TD(0), the target is the TD target $r + \gamma V(s')$

$$\Delta\theta = \alpha(r + \gamma V(s') - V_\theta(s))\phi(s)$$

- For forward-view TD(λ), the target is the λ -return v_t^λ

$$\Delta\theta = \alpha(v_t^\lambda - V_\theta(s))\phi(s)$$

- For backward-view TD(λ), we use eligibility traces

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

$$e_t = \gamma\lambda e_{t-1} + \phi(s_t)$$

$$\Delta\theta = \alpha\delta_t e_t$$

Actors at Different Time-Scales

- The policy gradient can also be estimated at many time-scales

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

- Monte-Carlo policy gradient uses error from complete return

$$\Delta\theta = \alpha(v_t - V_v(s_t)) \nabla_{\theta} \log \pi_{\theta}(s_t, a_t)$$

- Actor-critic policy gradient uses the one-step TD error

$$\Delta\theta = \alpha(r + \gamma V_v(s_{t+1}) - V_v(s_t)) \nabla_{\theta} \log \pi_{\theta}(s_t, a_t)$$

Policy Gradient with Eligibility Traces

- Just like forward-view $TD(\lambda)$, we can mix over time-scales

$$\Delta\theta = \alpha(v_t^\lambda - V_v(s_t))\nabla_\theta \log \pi_\theta(s_t, a_t)$$

- where $v_t^\lambda - V_v(s_t)$ is a biased estimate of advantage fn
- Like backward-view $TD(\lambda)$, we can also use eligibility traces
 - By equivalence with $TD(\lambda)$, substituting $\phi(s) = \nabla_\theta \log \pi_\theta(s, a)$

$$\delta = r_{t+1} + \gamma V_v(s_{t+1}) - V_v(s_t)$$

$$e_{t+1} = \lambda e_t + \nabla_\theta \log \pi_\theta(s, a)$$

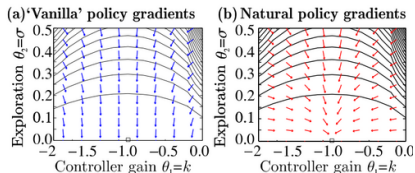
$$\Delta\theta = \alpha \delta e_t$$

- This update can be applied online, to incomplete sequences

Alternative Policy Gradient Directions

- Gradient ascent algorithms can follow *any* ascent direction
- A good ascent direction can significantly speed convergence
- Also, a policy can often be reparametrised without changing action probabilities
- For example, increasing score of all actions in a softmax policy
- The vanilla gradient is sensitive to these reparametrisations

Natural Policy Gradient



- The **natural policy gradient** is parametrisation independent
- It finds ascent direction that is closest to vanilla gradient, when changing policy by a small, fixed amount

works only for continous action space

$$\nabla_{\theta}^{nat} \pi_{\theta}(s, a) = G_{\theta}^{-1} \nabla_{\theta} \pi_{\theta}(s, a)$$

- where G_{θ} is the Fisher information matrix

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

Natural Actor-Critic

- Using compatible function approximation,

$$\nabla_w A_w(s, a) = \nabla_\theta \log \pi_\theta(s, a)$$

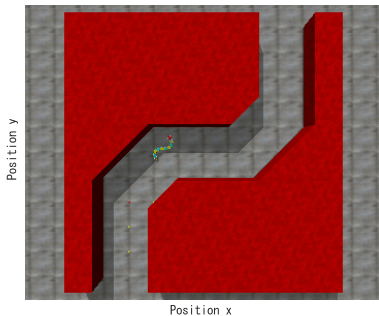
- So the natural policy gradient simplifies,

$$\begin{aligned}\nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta} [\nabla_\theta \log \pi_\theta(s, a) A^{\pi_\theta}(s, a)] \\ &= \mathbb{E}_{\pi_\theta} \left[\nabla_\theta \log \pi_\theta(s, a) \nabla_\theta \log \pi_\theta(s, a)^T w \right] \\ &= G_\theta w\end{aligned}$$

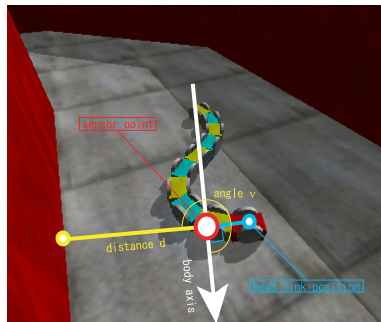
$$\nabla_\theta^{\text{nat}} J(\theta) = w$$

- i.e. update actor parameters in direction of critic parameters

Natural Actor Critic in Snake Domain

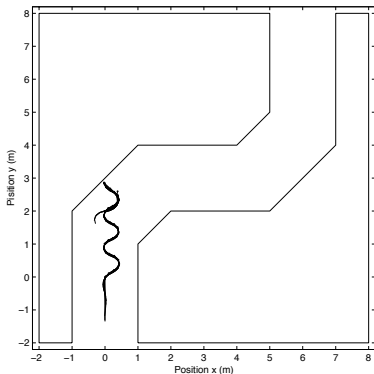


(a) Crank course

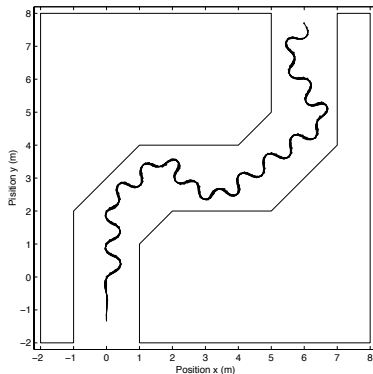


(b) Sensor setting

Natural Actor Critic in Snake Domain (2)

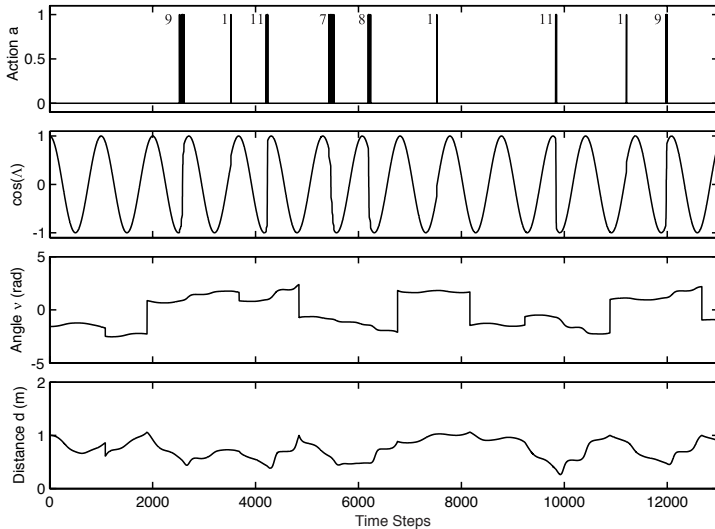


(a) Before learning



(b) After learning

Natural Actor Critic in Snake Domain (3)



Summary of Policy Gradient Algorithms

- The **policy gradient** has many equivalent forms

$\nabla_{\theta} J(\theta) = \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(s, a) \mathbf{v}_t]$	REINFORCE
$= \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(s, a) \mathbf{Q}^w(s, a)]$	Q Actor-Critic
$= \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(s, a) \mathbf{A}^w(s, a)]$	Advantage Actor-Critic
$= \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(s, a) \delta]$	TD Actor-Critic
$= \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(s, a) \delta \mathbf{e}]$	TD(λ) Actor-Critic
$G_{\theta}^{-1} \nabla_{\theta} J(\theta) = \mathbf{w}$	Natural Actor-Critic

- Each leads a stochastic gradient ascent algorithm
- Critic uses **policy evaluation** (e.g. MC or TD learning) to estimate $Q^{\pi}(s, a)$, $A^{\pi}(s, a)$ or $V^{\pi}(s)$