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

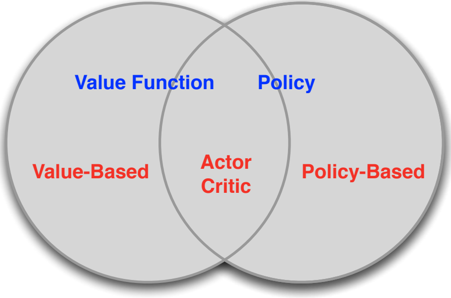
$$\pi_\theta(s, a) = \mathbb{P}\left[a \mid s, \theta\right]$$

- We will focus again on model-free reinforcement learning

Value-Based and Policy-Based RL

- Value Based
  - Learnt Value Function
  - Implicit policy (e.g. ϵ-greedy)
- Policy Based
  - No Value Function
  - Learnt Policy
- Actor-Critic

  policy value 동시에 배움
  - 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

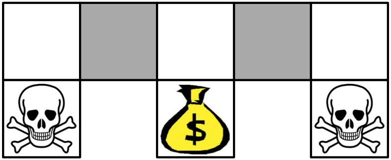
Disadvantages:

- 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)

Example: Aliased Gridworld (1)



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

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

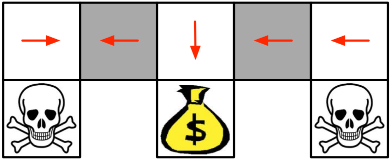
- Compare value-based RL, using an approximate value function

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

- To policy-based RL, using a parametrised policy

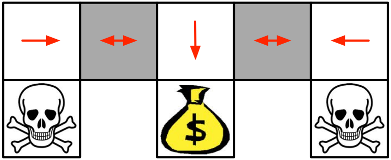
$$\pi_\theta(s, a) = g(\phi(s, a), \theta)$$

Example: Aliased Gridworld (2)



- 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

목적함수 J θ가 policy를 결정하는셈.

- 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?
- ① In episodic environments we can use the start value

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

- ② In continuing environments we can use the average value

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

stationary distribution - 각 state별로 있을 확률에 대한 확률분포.

- ③ 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

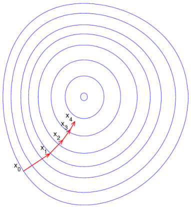
- Policy based reinforcement learning is an optimisation problem
- Find θ that maximises $J(\theta)$
- Some approaches do not use gradient
  - Hill climbing
  - Simplex / amoeba / Nelder Mead
  - Genetic algorithms
- Greater efficiency often possible using gradient
  - Gradient descent – Silver 강의는 이걸로 다.
  - Conjugate gradient
  - Quasi-newton – newton method.
- We focus on gradient descent, many extensions possible
- And on methods that exploit sequential structure

Policy Gradient

└π 가 아니라, J에 대한 gradient를 update한다, 6강- error를 minimize.

- 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

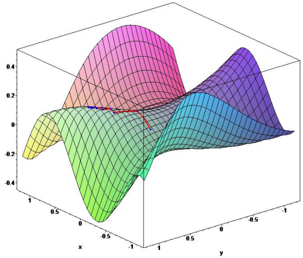
maximize
|
gradient ascent



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

- 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

Computing Gradients By Finite Differences

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

$\theta_1, \dots, \theta_n$ 까지 다 편미분

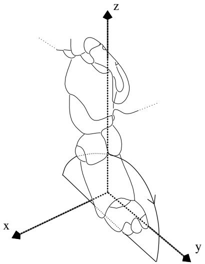
$$\left(\frac{\partial J(\theta)}{\partial \theta_1}, \dots, \frac{\partial J(\theta)}{\theta_n}\right) = \nabla J(\theta_k)$$

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

where u_k is unit vector with 1 in kth 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

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

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

$$\frac{dx}{x} = d(\log x)$$

$$\left[\frac{d}{dx} \log x = \frac{1}{x} \qquad d \log x = \frac{dx}{x} \right]$$

likelihood ratio 트릭.
$\nabla \pi$를 $\pi \nabla \log \pi$로 바꿈.
편하게 하기 위해

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

Softmax Policy

- 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

$$\pi_\theta(s, a) \propto e^{\phi(s,a)^\top \theta}$$

- The score function is

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

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

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

One-Step MDPs

어떤 확률 분포 d(s)를 따르는 state에서, 확률에 의해 나타난
첫째 state s 에서,
한 step 이동하고 reward가
끝나는 경우.

- 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

?

목적함수 ③번. –

$$
\begin{aligned}
J(\theta) &= \mathbb{E}_{\pi_\theta}\left[r\right] \\
&= \sum_{s \in \mathcal{S}} d(s) \sum_{a \in \mathcal{A}} \pi_\theta(s,a)\mathcal{R}_{s,a}
\end{aligned}
$$

likelihood ratio trick 사용하여

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \sum_{s \in \mathcal{S}} d(s) \sum_{a \in \mathcal{A}} \pi_\theta(s,a)\nabla_\theta \log \pi_\theta(s,a)\mathcal{R}_{s,a} \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) r\right]
\end{aligned}
$$

$\pi_\theta(s,a)$를 앞으로 빼내면 → $\nabla_\theta J(\theta)$를 E_{π_θ} 기댓값으로 표현가능

실제 액션으로 얻는 sample 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

r은 Q로 대체하면 된다. (증명됨)

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

앞의 목적함수 J 3개 모두 된다.

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

score function × Q

or reinforce algorithm ~ 알파고에서 쓰임.

Monte-Carlo Policy Gradient (REINFORCE)

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

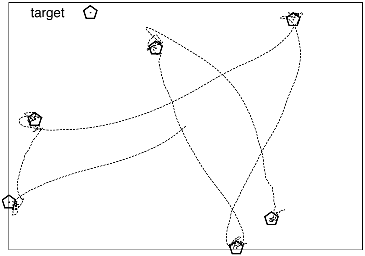
원래 G(t)로 배웠던 것, cumulative discount reward

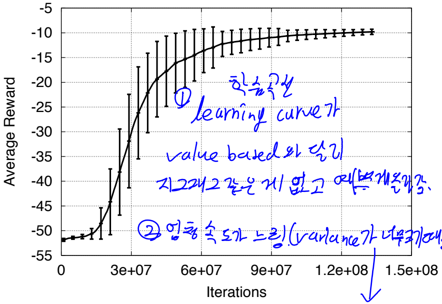
$$\Delta\theta_t = \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$$

Q는 true value인데, 모르니까 return(Q의 unviased sample임)을 쓴다.

function REINFORCE
    Initialise θ arbitrarily
    for each episode $\{s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **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





① 학습곡선 learning curve가 value based와 달리 지그재그 같은 게 없고 예쁘게 올라감.

② 엄청 속도가 느림 (variance가 너무 크기 때문)

→ variance를 어떻게 줄이기? → actor critic 방법론 등

- 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,

approximated Q true Q value

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

- 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-critic algorithms follow an *approximate* policy gradient

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

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

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(λ)
- Could also use e.g. least-squares policy evaluation

Action-Value Actor-Critic

- 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 Q-actor-critic
  Initialise s, θ
  Sample $a \sim \pi_\theta$
  for each step **do**
    Sample reward $r = \mathcal{R}_s^a$; sample transition $s' \sim \mathcal{P}_{s,\cdot}^a$
    Sample action $a' \sim \pi_\theta(s', a')$
TD error – $\delta = r + \gamma Q_w(s', a') - Q_w(s, a)$
    $\theta = \theta + \alpha \nabla_\theta \log \pi_\theta(s, a) Q_w(s, a)$
    $w \leftarrow w + \beta \delta \phi(s, a)$
    $a \leftarrow a', s \leftarrow s'$
  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*

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

Then the policy gradient is exact,

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

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,

$$
\begin{aligned}
\nabla_w \varepsilon &= 0 \\
\mathbb{E}_{\pi_\theta}\left[(Q^\theta(s,a) - Q_w(s,a))\nabla_w Q_w(s,a)\right] &= 0 \\
\mathbb{E}_{\pi_\theta}\left[(Q^\theta(s,a) - Q_w(s,a))\nabla_\theta \log \pi_\theta(s,a)\right] &= 0 \\
\mathbb{E}_{\pi_\theta}\left[Q^\theta(s,a)\nabla_\theta \log \pi_\theta(s,a)\right] &= \mathbb{E}_{\pi_\theta}\left[Q_w(s,a)\nabla_\theta \log \pi_\theta(s,a)\right]
\end{aligned}
$$

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

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

Reducing Variance Using a Baseline

state에 대한 일반적인 함수로 간주. (임의의 함수)

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

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

likelihood ratio trick을 역산중...

확률의 합이니까 1.

$\nabla_\theta 1 = 0$

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

(variance 를 줄일수 있음.

advantage function

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

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

Estimating the Advantage Function (1)

→ Q - V : 이제 V도 있으니 parameter만 세개 필요 . . .

- 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,

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

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

parameter를 줄이자!!
→ Q 없이 V에 대한
parameter만으로
advantage를 계산할수
있게함
(다음 페이지)

Estimating the Advantage Function (2)

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

s 다음 state.

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

- is an unbiased estimate of the advantage function

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

- 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

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

- 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
    $$\begin{aligned} \delta_t &= r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \\ e_t &= \gamma\lambda e_{t-1} + \phi(s_t) \\ \Delta\theta &= \alpha\delta_t e_t \end{aligned}$$

Actors at Different Time-Scales

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

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

- 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(λ), we can mix over time-scales

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

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

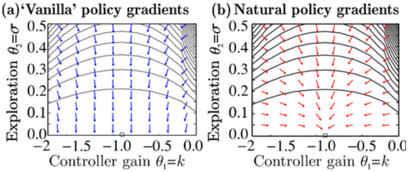
$$\begin{aligned}
\delta &= r_{t+1} + \gamma V_v(s_{t+1}) - V_v(s_t) \\
e_{t+1} &= \lambda e_t + \nabla_\theta \log \pi_\theta(s, a) \\
\Delta\theta &= \alpha \delta e_t
\end{aligned}$$

- 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

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

- 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,

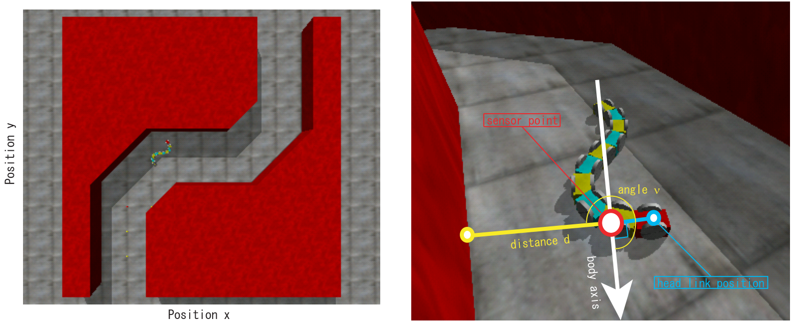
$$\nabla_w A_w(s,a) = \nabla_\theta \log \pi_\theta(s,a)$$

- So the natural policy gradient simplifies,

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

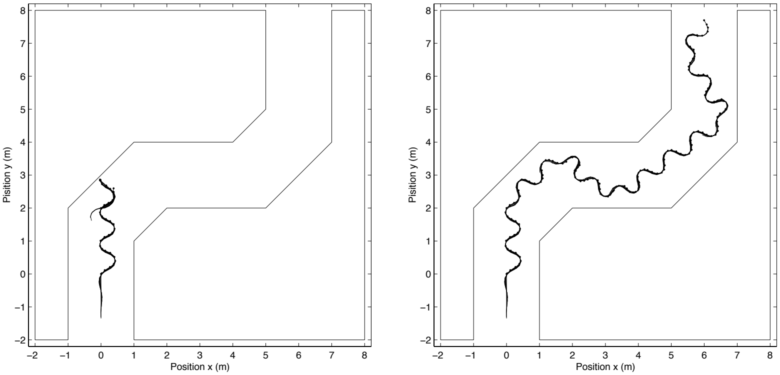
- i.e. update actor parameters in direction of critic parameters

Natural Actor Critic in Snake Domain



(a) Crank course (b) Sensor setting

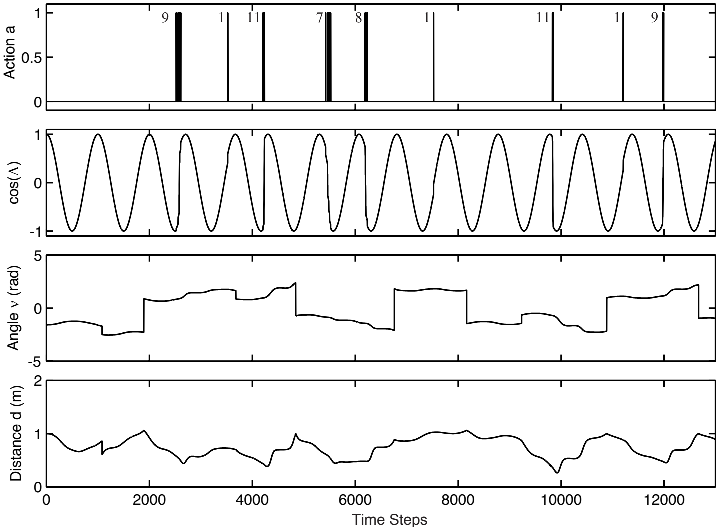
Natural Actor Critic in Snake Domain (2)



(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

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ \underbrace{v_t}_{\text{return.}}\right] && \text{~~REINFORCE~~ (only actor)} \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ Q^w(s,a)\right] && \text{Q Actor-Critic} \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ A^w(s,a)\right] && \text{Advantage Actor-Critic} \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ \delta\right] && \text{TD Actor-Critic} \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ \delta e\right] && \text{TD}(\lambda)\ \text{Actor-Critic} \\
G_\theta^{-1}\nabla_\theta J(\theta) &= w && \text{Natural Actor-Critic}
\end{aligned}
$$

- 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 $V^\pi(s)$