Lecture 8: Policy Gradient

Hado van Hasselt

Outline

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

Vapnik's rule

Never solve a more general problem as an intermediate step.
(Vladimir Vapnik, 1998)

If we care about optimal behaviour: why not learn a policy directly?

General overview

- Model-based RL:
  - \+ 'Easy' to learn a model (supervised learning)
  - \- 'Wrong' objective
  - \- Non-trivial going from model to policy (planning)
- Value-based RL:
  - \+ Closer to true objective
  - o Harder than learning models, but perhaps easier than policies
  - \- Still not exactly right objective
- Policy-based RL:
  - \+ Right objective!
  - \- Learning & evaluating policies can be inefficient (high variance)
  - \- Ignores other learnable knowledge, which may be important

Policy-Based Reinforcement Learning

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

$$v_\theta(s) \approx v_\pi(s)$$
$$q_\theta(s, a) \approx q_\pi(s, a)$$

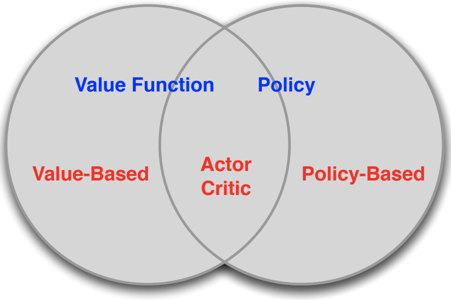
- A policy was generated directly from the value function
  - e.g. using ϵ-greedy
- In this lecture we will directly parametrize the policy

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

- We focus 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
  - 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
- Sometimes policies are simple while values and models are complex

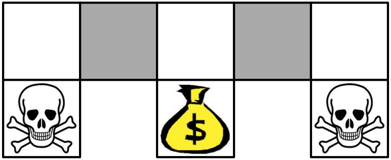
Disadvantages:

- Susceptible to local optima
- Learning a policy can be slower than learning values

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)

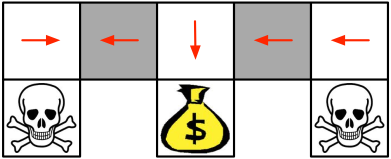


- The agent cannot differentiate the grey states
- Consider features of the following form (for all N, E, S, W)

$$\phi(s,a) = (\overbrace{\underbrace{1}_{\text{N}}\ \underbrace{0}_{\text{E}}\ \underbrace{1}_{\text{S}}\ \underbrace{0}_{\text{W}}}^{\text{walls}}\ \overbrace{\underbrace{0}_{\text{N}}\ \underbrace{1}_{\text{E}}\ \underbrace{0}_{\text{S}}\ \underbrace{0}_{\text{W}}}^{\text{actions}})$$

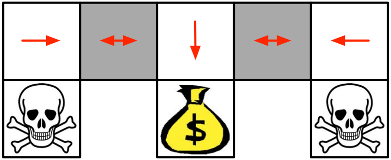
- Compare deterministic and stochastic policies

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
- So it will traverse the corridor for a long time

Example: Aliased Gridworld (3)



- An optimal stochastic policy moves randomly E or W in grey states

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

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

Policy Objective Functions

- 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

$$J_1(\theta) = v_{\pi_\theta}(s_1)$$

- 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 optimization problem
- Find θ that maximises $J(\theta)$
- Some approaches do not use gradient
  - Hill climbing
  - Genetic algorithms
- Greater efficiency often possible using gradient
  - Gradient descent
  - Conjugate gradient
  - Quasi-newton
- We will focus on stochastic gradient descent

Policy Gradient

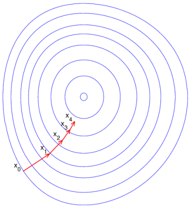
- Let $J(\theta)$ be any policy objective function
- Policy gradient algorithms search for a local maximum in $J(\theta)$ by ascending the gradient of the policy, w.r.t. parameters θ

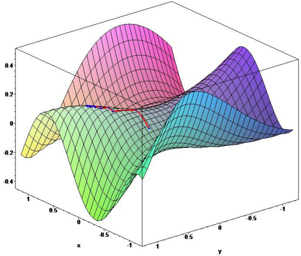
$$\Delta\theta = \alpha \nabla_\theta J(\theta)$$

- 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





Gradients on parameterized policies

- We need to compute an estimate of the policy gradient
- Assume policy π_θ is differentiable almost everywhere
  - E.g., π_θ is a linear function, or a neural network
  - Or perhaps we may have a parameterized class of controllers
- Goal is to compute

$$\nabla_\theta J(\theta) = \nabla_\theta \mathbb{E}_d[v_{\pi_\theta}(S)] \,.$$

- We will use Monte Carlo samples to compute this gradient
- So, how does $\mathbb{E}_d[v_{\pi_\theta}(S)]$ depend on θ?

Monte Carlo Policy Gradient

- Consider a one-step case such that $J(\theta) = \mathbb{E}[R(S, A)]$, where expectation is over d (states) and π (actions)
- We cannot sample R_{t+1} and then take a gradient: R_{t+1} is just a number that does not depend on θ
- Instead, we use the identity:

$$\nabla_\theta \mathbb{E}[R(S, A)] = \mathbb{E}[\nabla_\theta \log \pi(A|S) R(S, A)] \, .$$

  (Proof on next slide)
- The right-hand side gives an expected gradient that can be sampled

Monte Carlo Policy Gradient

$$
\begin{aligned}
\nabla_\theta \mathbb{E}[R(S,A)] &= \nabla_\theta \sum_s d(s) \sum_a \pi_\theta(a|s) R(s,a) \\
&= \sum_s d(s) \sum_a \nabla_\theta \pi_\theta(a|s) R(s,a) \\
&= \sum_s d(s) \sum_a \pi_\theta(a|s) \frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)} R(s,a) \\
&= \sum_s d(s) \sum_a \pi_\theta(a|s) \nabla_\theta \log \pi_\theta(a|s) R(s,a) \\
&= \mathbb{E}[\nabla_\theta \log \pi_\theta(A|S) R(S,A)]
\end{aligned}
$$

Monte Carlo Policy Gradient

$$\nabla_\theta \mathbb{E}[R(S,A)] = \mathbb{E}[\nabla_\theta \log \pi_\theta(A|S) R(S,A)] \quad \text{(see previous slide)}$$

- This is something we *can* sample
- Our stochastic policy-gradient update is then

$$\theta_{t+1} = \theta_t + \alpha R_{t+1} \nabla_\theta \log \pi_{\theta_t}(A_t|S_t) \, .$$

- In expectation, this is the actual policy gradient
- So this is a stochastic gradient algorithm

Softmax Policy

- Consider a softmax policy on linear values as an example
- Weight actions using linear combination of features $\phi(s,a)^\top \theta$
- Probability of action is proportional to exponentiated weight

$$\pi_\theta(s,a) \frac{e^{\phi(s,a)^\top \theta}}{\sum_b e^{\phi(s,b)^\top \theta}}$$

- The gradient of the log probability 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 common
- E.g., mean is a linear combination of state features $\mu(s) = \phi(s)^\top \theta$
- Lets assume variance may be fixed at σ^2 (can be parametrized as well, instead)
- Policy is Gaussian, $a \sim \mathcal{N}(\mu(s), \sigma^2)$
- The gradient of the log of the policy is then

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

Policy Gradient Theorem

- The policy gradient theorem generalizes this 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

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(A|S)\ q_{\pi_\theta}(S, A)\right]$$

Monte-Carlo Policy Gradient (REINFORCE)

- Using return $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$ as an unbiased sample of $q_{\pi_\theta}(S_t, A_t)$
- Update parameters by stochastic gradient ascent

$$\Delta\theta_t = \alpha \nabla_\theta \log \pi_\theta(A_t|S_t) G_t$$

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**
      $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(A_t|S_t) G_t$
    end for
  end for
  return θ
end function

Labyrinth

Reducing Variance Using a Critic

- Monte-Carlo policy gradient can have high variance, because we use full return
- We can use a critic to estimate the action-value function,

$$q_w(s, a) \approx q_{\pi_\theta}(s, a)$$

- 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
- One Actor-critic algorithm: 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
- What is the value of policy π_θ for current parameters θ?
- This problem was explored in previous lectures, e.g.
  - Monte-Carlo policy evaluation
  - Temporal-Difference learning
  - TD(λ)

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 Sarsa(0)
  - **Actor** Updates θ by policy gradient

function QAC
  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 $A' \sim \pi_\theta(S')$
    $w \leftarrow w + \beta(R + \gamma q_w(S',A') - q_w(S,A))\phi(S,A)$ [Sarsa(0)]
    $\theta = \theta + \alpha \nabla_\theta \log \pi_\theta(s,a) q_w(s,a)$ [Policy gradient update]
    $a \leftarrow a', s \leftarrow s'$
  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:
  - 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,

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(A|S)\ q_w(S,A)\right]$$

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,

$$\begin{aligned} \nabla_w \varepsilon &= 0 \\ \mathbb{E}_{\pi_\theta}\left[(q_{\pi_\theta}(S,A) - q_w(S,A))\nabla_w q_w(S,A)\right] &= 0 \\ \mathbb{E}_{\pi_\theta}\left[(q_{\pi_\theta}(S,A) - q_w(S,A))\nabla_\theta \log \pi_\theta(A|S)\right] &= 0 \\ \mathbb{E}_{\pi_\theta}\left[q_{\pi_\theta}(S,A)\nabla_\theta \log \pi_\theta(A|S)\right] &= \mathbb{E}_{\pi_\theta}\left[q_w(S,A)\nabla_\theta \log \pi_\theta(A|S)\right] \end{aligned}$$

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

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

Compatible Function Approximation

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

$$\varepsilon = \mathbb{E}_{\pi_\theta}\left[\left(q_{\pi_\theta}(S, A) - q_w(S, A)\right)^2\right]$$

- Unfortunately, 1) only holds for certain function classes
- Unfortunately, 2) is never quite true
- These are typical limitations of theory; be aware of the assumptions

Reducing Variance Using a Baseline

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

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

- So we can rewrite the policy gradient as

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

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,

$$
\begin{aligned}
v_\xi(s) &\approx v_{\pi_\theta}(s) \\
q_w(s, a) &\approx q_{\pi_\theta}(s, a) \\
A(s, a) &= q_w(s, a) - v_\xi(s)
\end{aligned}
$$

- 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 δ^{π_θ}

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

- is an unbiased estimate of the advantage function

$$\begin{aligned}
\mathbb{E}_{\pi_\theta}\left[\delta^{\pi_\theta} | s, a\right] &= \mathbb{E}_{\pi_\theta}\left[R_{t+1} + \gamma v_{\pi_\theta}(S_{t+1}) | s, a\right] - v_{\pi_\theta}(s) \\
&= q_{\pi_\theta}(s, a) - v_{\pi_\theta}(s) \\
&= A_{\pi_\theta}(s, a)
\end{aligned}$$

- 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

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

- This approach only requires one set of critic parameters ξ

Critics at Different Time-Scales

- Critic can estimate value function $v_\xi(s)$ from many targets at different time-scales
- From last lecture...
  - For MC, the target is the return G_t
  $$\Delta\theta = \alpha({\color{red}G_t} - v_\xi(s))\phi(s)$$
  - For TD(0), the target is the TD target $r + \gamma v_\xi(s')$
  $$\Delta\theta = \alpha({\color{red}R_{t+1} + \gamma v_\xi(S_{t+1})} - v_\xi(s))\phi(s)$$
  - For forward-view TD(λ), the target is the λ-return G_t^λ
  $$\Delta\theta = \alpha({\color{red}G_t^\lambda} - v_\xi(s))\phi(s)$$
  - For backward-view TD(λ), we use eligibility traces
  $$\begin{aligned}\delta_t &= R_{t+1} + \gamma v_\xi(s_{t+1}) - v_\xi(s_t)\\ e_t &= \gamma\lambda e_{t-1} + \phi(S_t)\\ \Delta\theta &= \alpha\delta_t e_t\end{aligned}$$

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

$$\Delta\theta = \alpha(G_t - v_\xi(s_t))\nabla_\theta \log \pi_\theta(s_t,a_t)$$

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

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

Policy Gradient with Eligibility Traces

- Just like forward-view TD(λ), we can mix over time-scales

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

- Like backward-view TD(λ), we can also use eligibility traces
  - By equivalence with TD(λ), substituting $\phi(s) = \nabla_\theta \log \pi_\theta(s, a)$

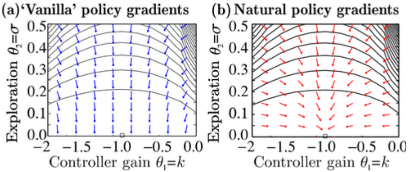
$$\begin{aligned}
\delta_t &= R_{t+1} + \gamma v_\xi(S_{t+1}) - v_\xi(S_t) \\
e_{t+1} &= \gamma\lambda e_t + \nabla_\theta \log \pi_\theta(A_t|S_t) \\
\Delta\theta &= \alpha\delta_t e_t
\end{aligned}$$

- 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 direction that maximally ascends objective function, 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 linear function approximation,

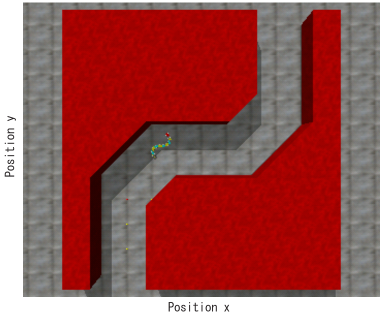
$$\phi(s, a) = \nabla_\theta \log \pi_\theta(s, a)$$

- So the natural policy gradient simplifies,

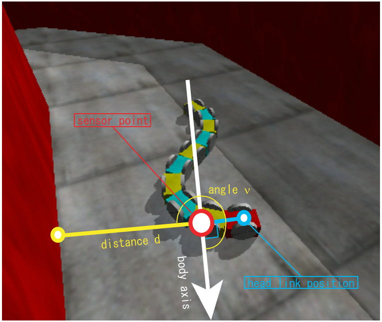
$$\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s, a) A_{\pi_\theta}(s, a)\right] \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s, a) \phi(s, a)^T w\right] \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s, a)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

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

Natural Actor Critic in Snake Domain

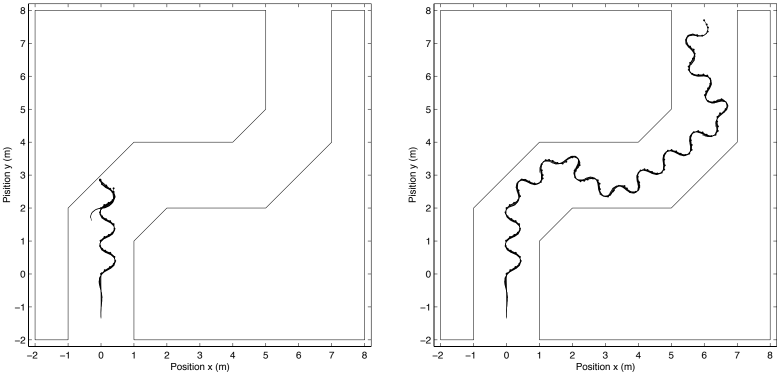


(a) Crank course



(b) Sensor setting

Natural Actor Critic in Snake Domain (2)



(a) Before learning (b) After learning

Summary of Policy Gradient Algorithms

- The policy gradient has many equivalent forms

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta} \left[\nabla_\theta \log \pi_\theta(A|S) \; G_t \right] && \text{REINFORCE} \\
&= \mathbb{E}_{\pi_\theta} \left[\nabla_\theta \log \pi_\theta(A|S) \; q_w(S, A) \right] && \text{Q Actor-Critic} \\
&= \mathbb{E}_{\pi_\theta} \left[\nabla_\theta \log \pi_\theta(A|S) \; A_w(S, A) \right] && \text{Advantage Actor-Critic} \\
&= \mathbb{E}_{\pi_\theta} \left[\nabla_\theta \log \pi_\theta(A|S) \; \delta_t \right] && \text{TD Actor-Critic} \\
&= \mathbb{E}_{\pi_\theta} \left[\delta_t e_t \right] && \text{TD}(\lambda) \text{ Actor-Critic} \\
G_\theta^{-1} \nabla_\theta J(\theta) &= w && \text{(linear) Natural Actor-Critic}
\end{aligned}
$$

- 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 $v_\pi(s)$