Lecture 7: Policy Gradient

David Silver

Outline

Policy-Based Reinforcement Learning

Two types of Value Function approximation: 1) State function V, 2) Q function

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

we calculated this using a NN

$$V_\theta(s) \approx V^\pi(s)$$
$$Q_\theta(s, a) \approx Q^\pi(s, a)$$

instead of the Value function

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

input = state
output = probability distribution over the actions

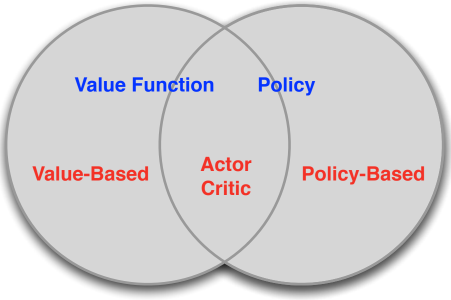
$$\pi_\theta(s, a) = \mathbb{P}\left[a \mid s, \theta\right]$$

- We will focus again on model-free reinforcement learning

Value-Based and Policy-Based RL

We can take the parameterized policy and adjust those parameters to improve the reward. We do this by doing gradient ascent.

- 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

you don't need the Max() like you do in Value functions. If you have a trillion actions, it will be expensive to take the max

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

Why would it be desirable to have a stochastic policy?

Here's an example...

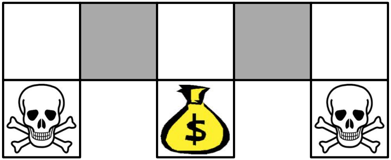
The stochastic policy (of playing random hands) is optimal

- 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)

Which will do better?

A value-based algo?
A policy-based algo?



state aliasing -
partially observable
MDP

- 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

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

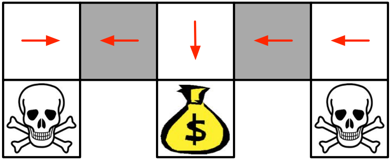
- To policy-based RL, using a parametrised policy

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

Example: Aliased Gridworld (2)

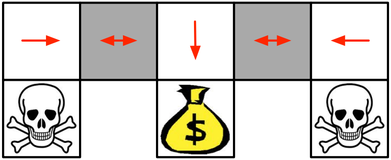
Deterministic policy, do the same thing every time for a given state.

So here, the two grey squares are the same "state" (based on the features we used), so the policy will be to either go left all the time or right all the time



- 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

- 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

what's the expected value for this start state given the policy

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

- In continuing environments we can use the average value

the probability we're in a given state multiplied by the value you'll receive onwards from being in that state

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

probability of being in state s

- Or the average reward per time-step

want to get the highest reward per time step

for each timestep, get the highest immediate reward

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

- 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
  - Conjugate gradient
  - Quasi-newton
- We focus on gradient descent, many extensions possible
- And on methods that exploit sequential structure

Policy Gradient

Finite Difference Policy Gradient: This is the simplest version

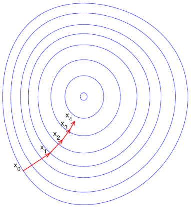
- Let $J(\theta)$ be any policy objective function
- Policy gradient algorithms search for a *local* maximum in $J(\theta)$ by ascending the gradient of the policy, w.r.t. parameters θ

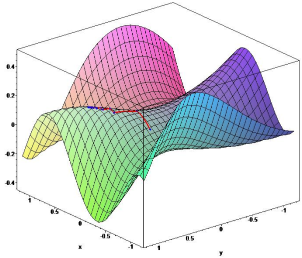
$$\Delta\theta = \alpha \nabla_\theta J(\theta)$$

- 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





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

so you just perturb your theta, then see the difference in total reward.
you have to do this for each dimension individually...

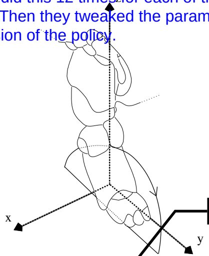
$$\frac{\partial J(\theta)}{\partial \theta_k} \approx \frac{J(\theta + \epsilon u_k) - J(\theta)}{\epsilon}$$

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

So they ran the robot and calculated its time. Then they slightly tweaked one parameter, then ran it again and time it. They did this 12 times for each of the 12 dimensions. Then they tweaked the parameters to get the next version of the policy.



12 dimensions

- 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

Using the brute-force method may work, but it's not practical if you have high dimensions. So now let's look at one way to analytically compute the gradient.

- 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

we can multiply and divide by our gradient.

then we do this trick to get the gradlog of our policy

This new form helps us take the expectation because we can sample from it

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

- 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

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

"how much more of this feature do I have than usual"

If this is positive, we have a certain feature more than usual. If it gets a good reward, we want to adjust the policy to do more of this.

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

goal: pick the action that maximizes your reward

- 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

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

r tells us whether it was good or bad, and how to adjust the parameters

expectation of the score * reward

Q gives us the long term reward over the horizon

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

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

Monte-Carlo Policy Gradient (REINFORCE)

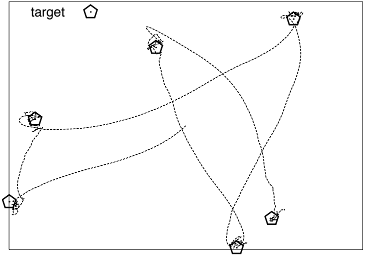
- Update parameters by stochastic gradient ascent
- Using policy gradient theorem
- Using return v_t as an unbiased sample of $Q^{\pi_\theta}(s_t, a_t)$

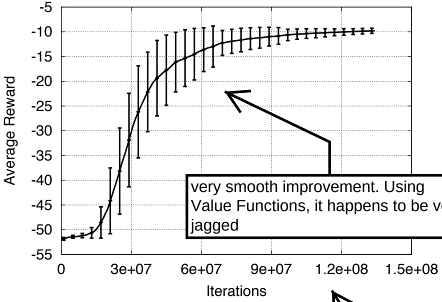
$$\Delta\theta_t = \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$$

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**
            $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$
        end for
    end for
    return θ
end function

Puck World Example

Monte Carlo PG is very slow





very smooth improvement. Using Value Functions, it happens to be very jagged

sample inefficient!

- 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

Main idea: instead of using the return to estimate action-value function, we will estimate the action-value function explicitly using a function estimator (aka Critic). Now we have two sets of parameters.

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

watches and evaluates the policy

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

So we plug this into our PG

- 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

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

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

We adjust the policy/actor in the direction that, according to the critic, will get more reward

actually doing things in the world.
Actually taking actions

Estimating the Action-Value Function

We want to evaluate our current policy. We aren't trying to estimate Q*! We're trying to estimate Q_pi

- 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)

can use our favorite policy evaluator

- Could also use e.g. least-squares policy evaluation

Basic Idea:
Actor uses the policy to take actions
Critic evaluates the policy

The actor then tries to improve the policy based on the feedback from the critic

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

we now don't have to wait until the end of the episode to do an update. We can do an update every step since we're using TD as the critic

function QAC
  Initialise s, θ
  Sample $a \sim \pi_\theta$
  for each step **do**
    Sample reward $r = \mathcal{R}_s^a$; sample transition $s' \sim \mathcal{P}_{s,\cdot}^a$ so we do 1 transition, get the reward r and state s'
    Sample action $a' \sim \pi_\theta(s', a')$
    $\delta = \boxed{r + \gamma Q_w(s', a')} - Q_w(s, a)$ get TD error
    $\theta = \theta + \alpha \nabla_\theta \log \pi_\theta(s, a) Q_w(s, a)$ update policy/actor
    $w \leftarrow w + \beta \delta \phi(s, a)$ update critic
    $a \leftarrow a', s \leftarrow s'$
  end for
end function

what we actually got

what we originally would have predicted

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

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

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

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

V(s) is just the avg(Q(s,a))

"how much better is it than usual to take action a"

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

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

Estimating the Advantage Function (1)

How do we estimate the Advantage Function? There's a lot of different ways....We will investigate acouple

One way would be to learn both Q and V. We'd have params for Q and separate params for V

- 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

TD error (delta): If we knew the true value of V_pi, then the TD error is the reward we just got plus the discounted true value of the next state minus to true value of our current state

Estimating the Advantage Function (2)

This is easier and more commonly used!

- 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

expected TD error

this is literally the equation for Q

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

the TD error is an unbiased estimate of the Advantage function, so we can just move the policy in the direction of the TD error

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

this means that we only need to estimate V_pi, not Q_pi

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

- 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...

you can make any of these your targets for the update function

  - For MC, the target is the return v_t

$$\Delta\theta = \alpha({\color{red}v_t} - V_\theta(s))\phi(s)$$

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

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

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

$$\Delta\theta = \alpha({\color{red}v_t^\lambda} - V_\theta(s))\phi(s)$$

  - 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

return minus some baseline

- 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

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

plug in TD target

Policy Gradient with Eligibility Traces

How to make our actor make use of critics from many different steps all the way into the future

- Just like forward-view TD(λ), 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(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

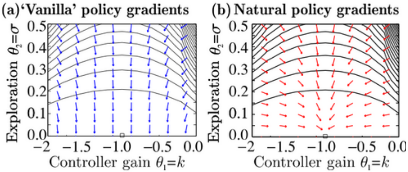
Alternative Policy Gradient Directions

- 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

so far we've followed these stochastic policies, but this can be noisy.

NPG:
Start out with deterministic policy
Only works for continuous action spaces



- 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,

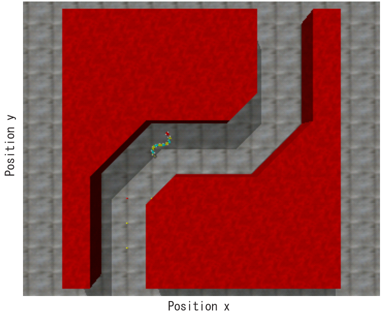
$$\nabla_w A_w(s,a) = \nabla_\theta \log \pi_\theta(s,a)$$

- So the natural policy gradient simplifies,

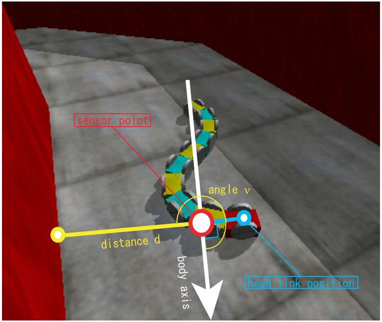
$$\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) A^{\pi_\theta}(s,a)\right] \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) \nabla_\theta \log \pi_\theta(s,a)^T w\right] \\
&= G_\theta w \\
\nabla_\theta^{nat} J(\theta)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

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

Natural Actor Critic in Snake Domain

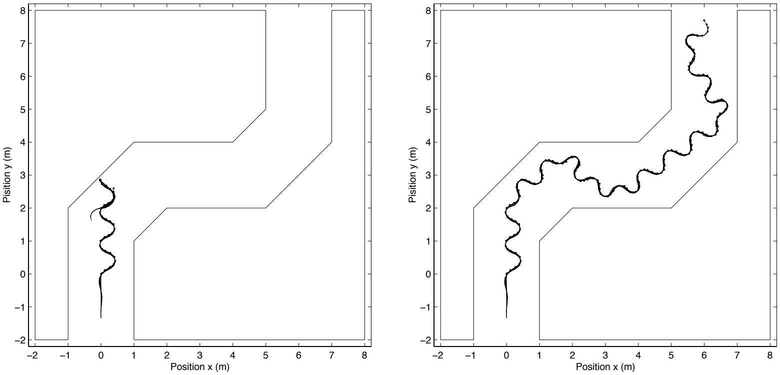


(a) Crank course



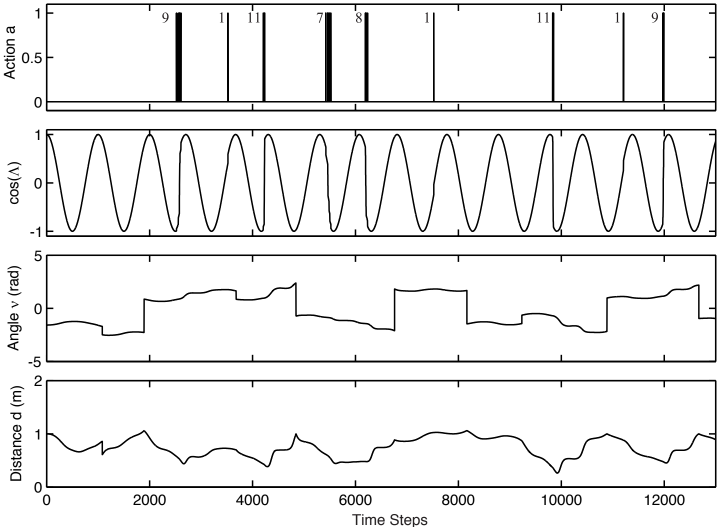
(b) Sensor setting

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

one update per episode...

everything else can do one update per step

These are all slight variants of the same idea

- The policy gradient has many equivalent forms

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

- 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 $V^\pi(s)$ use your favorite algo