Lecture 7: Policy Gradient

David Silver

Lecture 10 is optional

Outline

the most practical set of methods, these methods which combine everything from this class and everything in the previous class. So, they work both value functions and with policies and try to get XXX

Policy-Based Reinforcement Learning

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

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

using linear combination of features and neural network

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

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

tells us which actions we should pick

- We will focus again on model-free reinforcement learning

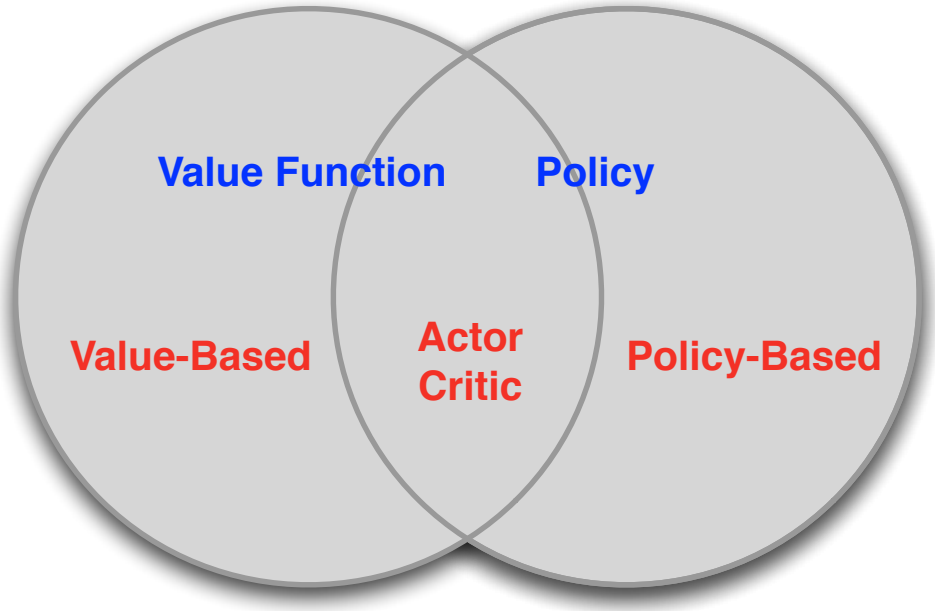
how to throw down the agent or the robot into some maze. And this thing is just wondering around just directly from experience for that anyone telling a dynamics of environment, it should be able to figure out how to adjust its parameters of its policy.

Value-Based and Policy-Based RL

The main mechanism is that we have to consider is the gradient ascent.
In other words, how can we compute the gradient follow, so it's to make this policy better.
And if we can follow that gradient, we will strictly be moving uphill in a way that improves our policy.
If we know the gradient w.r.t total reward we're going to get this MDP. We're just gonna follow that gradient in a direction to get more reward

- 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

maximization of rewards



Can you think of any advantages or disadvantages of using policy gradient compared to value based methods?
Why work the policy based methods?

Advantages of Policy-Based RL

Advantages: more stable

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

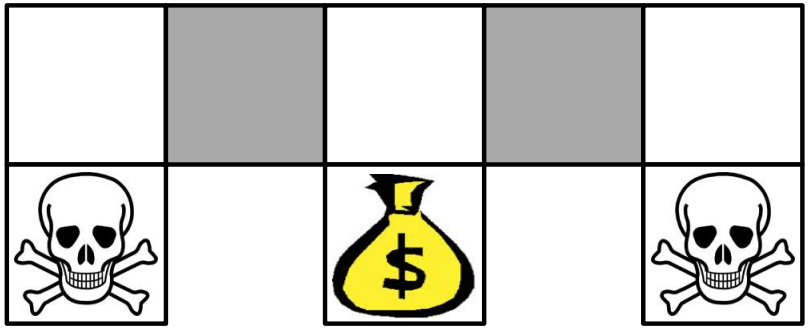
Disadvantages:

- 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)

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

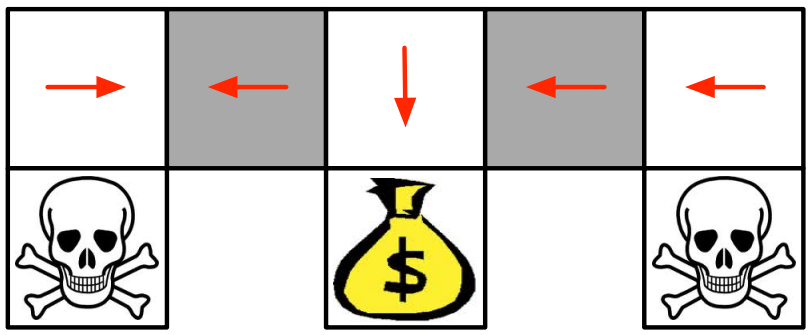
- Compare value-based RL, using an approximate value function

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

- To policy-based RL, using a parametrised policy

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

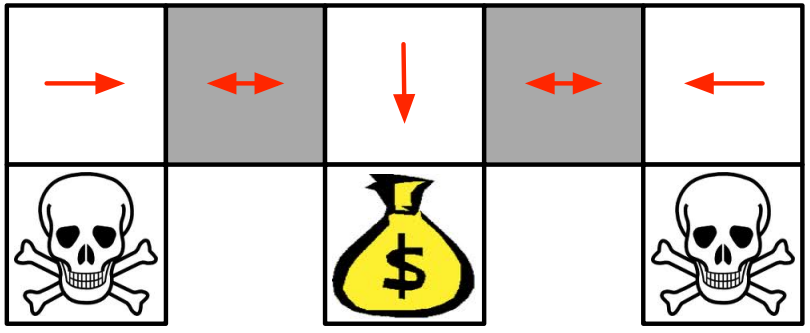
Example: Aliased Gridworld (2)



- 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)

Summary or lesson of this example(couple of slides) is to say that a stochastic policy can do better than the deterministic policy.



- 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

Let's talk about how to optimize a policy. So, what's the mean to optimize a policy. We need to know is what the object should be. Here are three different object functions we might choose. We want to know how good is this policy.

- 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 $J(\theta)$
- 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

by which I mean we're not just going to do blind optimization like genetic algorithm where you have to run or throw your robot down and wait until the end of life time to get one number that pops out.

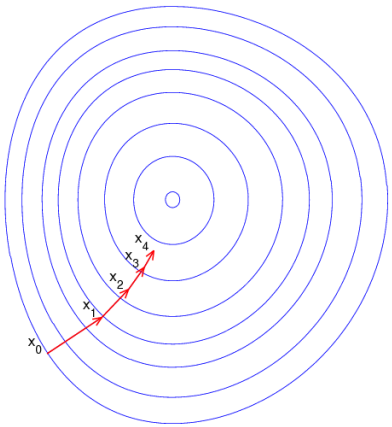
Policy Gradient

In this lecture, we're going to consider the "gradient ascent" where we have an objective function which is something like how much reward can I get out from this system. We want to make this higher.

- 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

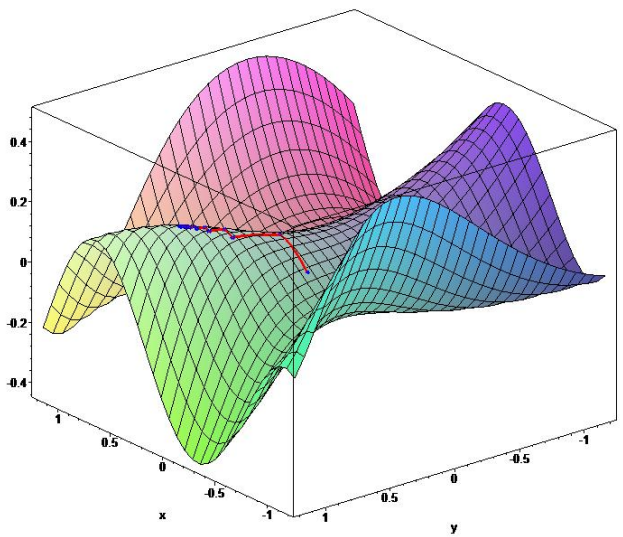
uphill

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



- 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

Numerical way to evaluate the gradient

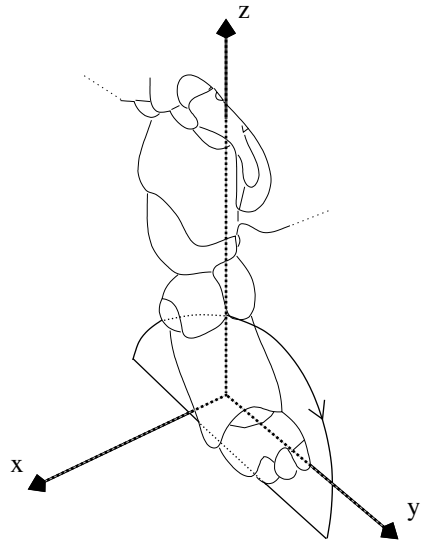
- To evaluate policy gradient of $\pi_\theta(s, a)$
- For each dimension $k \in [1, n]$
  - Estimate kth partial derivative of objective function w.r.t. θ
  - By perturbing θ by small amount ϵ in kth dimension

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

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

No value functions yet, that's the Monte-Carlo approach.

- 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

gradient log policy term is going to appear

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

I think it's useful to understand why this gradient log appears basically it comes out just from this following the likelihood ratios trick.

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

We're gonna take the expectation

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

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

The score function is just the feature for the action that we actually took - (minus) the average feature for all the actions we might have taken. So, it's basically saying how much more of this feature do I have than usual. That's score function.

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

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

We get to pick the mean and then we just add some noise to that thing to make it stochastic.
And so if we were using a Gaussian policy, what would the score function look like?
a : This is the action we actually took. This is our action a and this is the mean action.
action - mean : tells us how much more than usual we're moving with doing a particular action. And then, multiplied by the feature and then we just scale that by the variance. So both of these cases the score function takes this form of sort of how much more than usual am I doing something. And, that's quite intuitive when we start to look at these policy gradient updates.

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

get to take one step and you get one reward.
It depends on where you were and what action you took.

There's no sequence in this case, and it's just a one step.

We want to find the parameters which give us the most expected reward, so we want to find the gradient of this thing and ascend the gradient of this thing which will give us more expected reward. So, if we're thrown into any state and we pick an action according to our policy we want an expectation that to give us the most reward and so the way we're going to do that. Let's just expand this out.
So, this is the expectation over the state that we start in and this is the expectation over the actions that we pick and our own policy.

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

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

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

We're just going to plug in the likelihood ratio trick, so we're going to take the gradient of this whole thing.

the reward that we actually experienced

the score times the reward

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

This thing is an expectation under our policy so this is the expectation under the start state and an expectation under the action that we pick of the gradient of the log policy mutiplied by the reward. That's just the expectation of the score times the reward.
So, after all that we basically come back to something very simple tells us that if we want to increase our objective function and if want to get more reward we just need to move in the direction that's determined by the score times the reward.
This tells us how to adjust our policy so to get more or less of something and this tells us whether it was good or bad.

Policy Gradient Theorem

We want to do the same thing in multi-step MDPs

- 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

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

It basically tells us that again how to adjust the policy so to get more or less of that particular action multiplied by how good that particular action was.
We want to adjust the policy in the direction that does more of the good things and less of the bad things.

Monte-Carlo Policy Gradient (REINFORCE)

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

get rid of the expectation

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

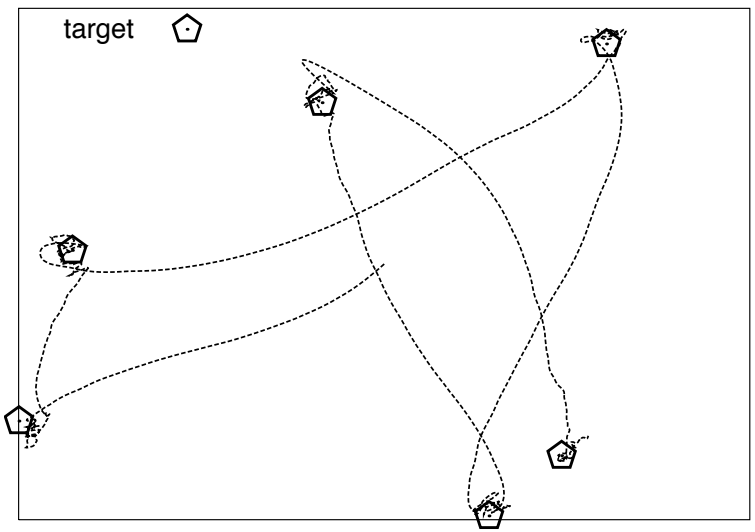
Sample Q and estimate Q by using the return as an unbiased sample of this Q. So, we're gonna be in a state we're gonna start by taking this action. We're gonna see what return we've got and we can use that as an estimate Q. We're just gonna plug that into our policy gradient to give us a direction and estimate for direction to move, and so every single episode now what we're gonna do is

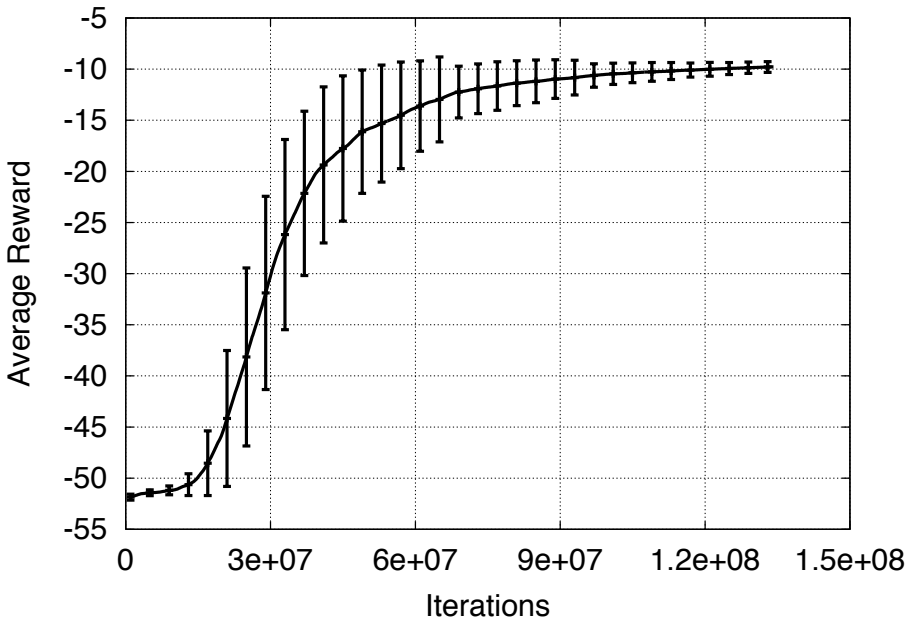
function REINFORCE
  Initialise θ arbitrarily
  for each episode $\{s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **do**
    for $t = 1$ to $T - 1$ **do**
      $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$
    end for
  end for
  return θ
end function

in each step, we're gonna adjust our parameters a little bit in the direction of this score multiplied by the return which we actually got from that point onwards.

all the way to the end of the episode Generate all of your rewards, at the end of that you know what the return was from each of your steps.

Puck World Example





- 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

It shows smooth learning but iterations, 100 millions, is very slow. And so the rest of this class is gonna be about using similar ideas but making them more efficient and how to take this very nice idea where you get this smooth learning, where you adjust the parameters of of your policy to solve a problem, make it more efficient, reduce the variance, and make it work better.

Reducing Variance Using a Critic

combine our value function approximation we did in last lecture with our policy methods

main idea

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

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

Critic doesn't actually take any decisions. It's just watching what the actor does, seeing what that's good or bad, evaluating that thing, and saying those decisions were good they got a score of a thousand or they got a score of minus a thousand

- 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 is the thing that is doing things in the world and it contains the policy. It's picking actions and making the decisions of what to do in the world.

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

original policy gradient idea which was an expectation of the score multiplied by the value function

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

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

We're gonna have our true action value function and estimate this thing using a function approximator, and then we're gonna plug this in to our policy gradient as a substitute for Q_\pi. What this means is that we have two sets of parameters and that's why these are called after critic methods we've got a critic and an actor.

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

to adjust our actor

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

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

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

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

    $w \leftarrow w + \beta \delta \phi(s,a)$ updating our critic in the direction of the TD error multiplied by the features.

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

  end for

end function

Every step of the algorithm now we don't need to wait until the end of the episode. This is now an online algorithm which we can every single step perform an update because we're using TD in our critic, not MC.

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*

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

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

It doesn't change the direction of ascent. In other words, it changes the variance of our estimator and we can reduce the variance of this thing without changing the expectation.

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

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

The particular baseline we're gonna choose is the state value function. So, what we're going to do is we're going to start off with our Q values which is our action value function and subtract off the state value function.
And what we're left with is somthing called the advantage function.
It tells us how much better than usual it(particular action a) is to take action a.

$$= \sum_{s\in\mathcal{S}} d^{\pi_\theta} B(s) \nabla_\theta \sum_{a\in\mathcal{A}} \pi_\theta(s,a)$$

$$= 0$$

our policy sums up to 1
because this is a probability distribution

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

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

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

Estimating the Advantage Function (1)

How do we estimate this advantage function? How do we do this in our critic?
There are a lot of different ways to do this actually. Suggest a couple of them.
This slide is just one way to do this would be to learn both Q and V, so our critic would basically learn Q we could also learn V using another set of parameters.

- 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,

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

From Q and V we learned, we will just take the difference between those things as our estimate of the advantage function

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

Estimating the Advantage Function (2)

There's an easier way and probably a better way. This is the probably the most commonly used variant of the actor-critic although it depends who you talk to and many people use many different variants.

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

sample of advantage function

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

- is an unbiased estimate of the advantage function

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

this is from measurability and by definition of conditional expectation.

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

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

The nice thing about this is that we only need to estimate V in our critic and we don't need to estimate Q.
We just need to estimate the state value function there's no actions that come into it.

- 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

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

Actors at Different Time-Scales

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

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

- 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

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

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

Sometimes it's a good idea to do this where we introduce bias by bootstrapping from a value function but we dramatically reduce the variance.

critic says which way to go

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

Eligibility trace now basically is an eligibility over our scores. So, it remember all of the scores that we've seen recently or frequently.

- This update can be applied online, to incomplete sequences

Alternative Policy Gradient Directions

There's a really important question in actor critic algorithms.

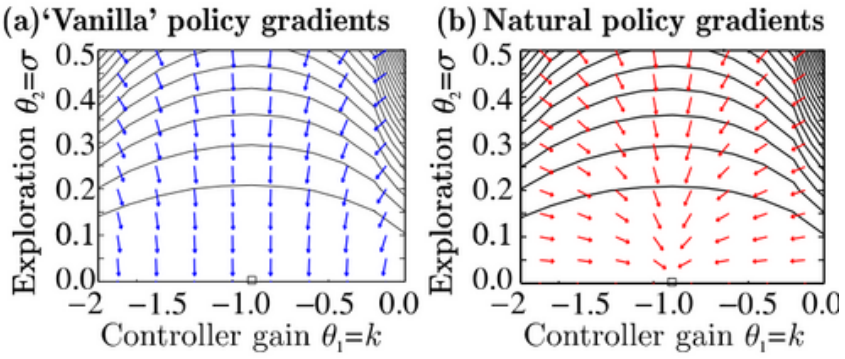
The answer is that if you choose the value function approximation carefully that you use it's possible to pick a value function approximator that doesn't introduce any bias at all. In other words, despite the fact that we don't have a true value function, they were approximating the value function. We can still follow the true gradient and be guaranteed to follow the true gradient with our policy updates. That approach is called compatible function approximation.

- 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

Recent idea which is a useful idea to know about it.
Stochastic policy or the actions which is sometimes perturbed by some noise. Estimating our policy gradients by sampling our own noise (We've got a noisy policy and we want to take an expectation over that noise.) It can be really bad idea that thing is really hard to estimate. The variance of your estmates actually starts to blow up to infinity This is sort of an unfortunate property of the policy gradient algorithms.
All we need to do is to take the gradient of our own function so we look at our critic which tells us the way which is the better actions we take. This is the deterministic policy gradient theorem, and it's very simple intuitive and in practice it seems to work a lot better than scattered policy gradient in situations where we've got continuous action spaces scales up much better to high dimensions.

Natural Policy Gradient

Will skip



- 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,

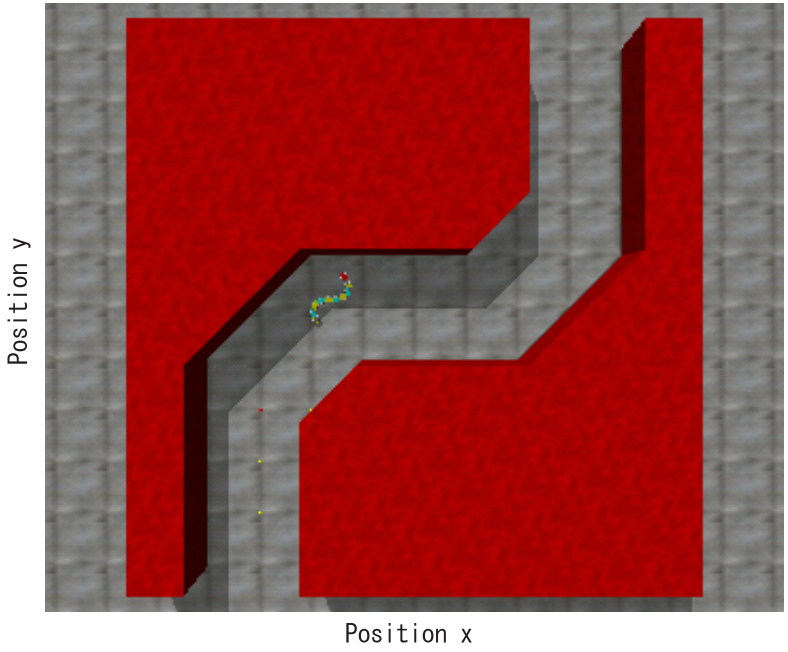
$$\nabla_w A_w(s,a) = \nabla_\theta \log \pi_\theta(s,a)$$

- So the natural policy gradient simplifies,

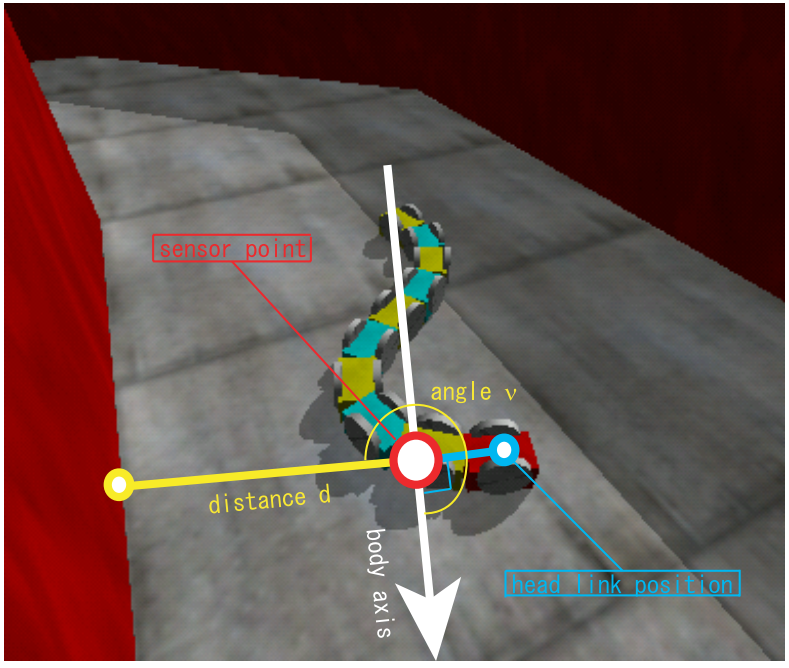
$$\begin{aligned}\nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) A^{\pi_\theta}(s,a)\right] \\ &= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) \nabla_\theta \log \pi_\theta(s,a)^T w\right] \\ &= G_\theta w \\ \nabla_\theta^{nat} J(\theta)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

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

Natural Actor Critic in Snake Domain

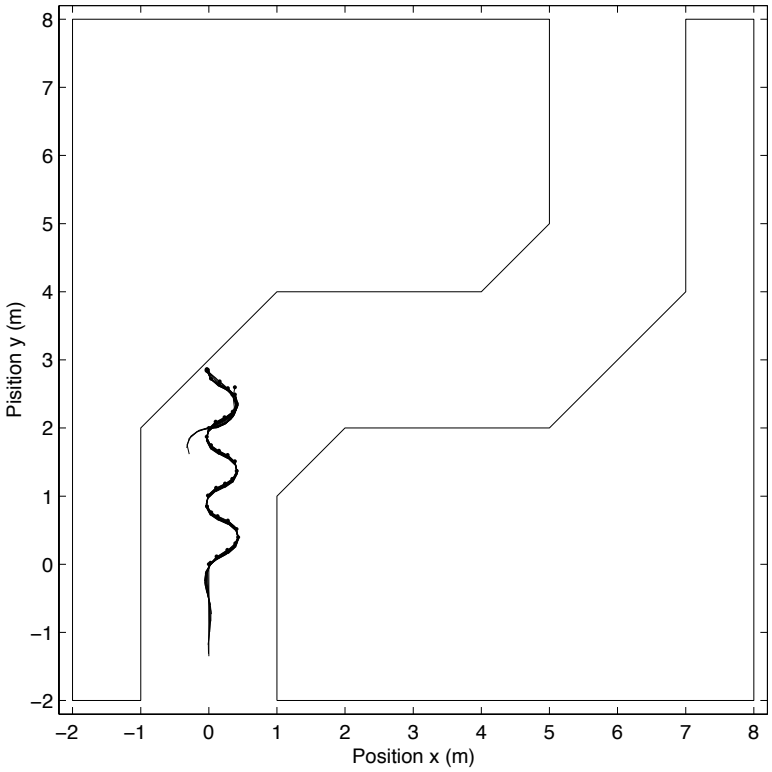


(a) Crank course

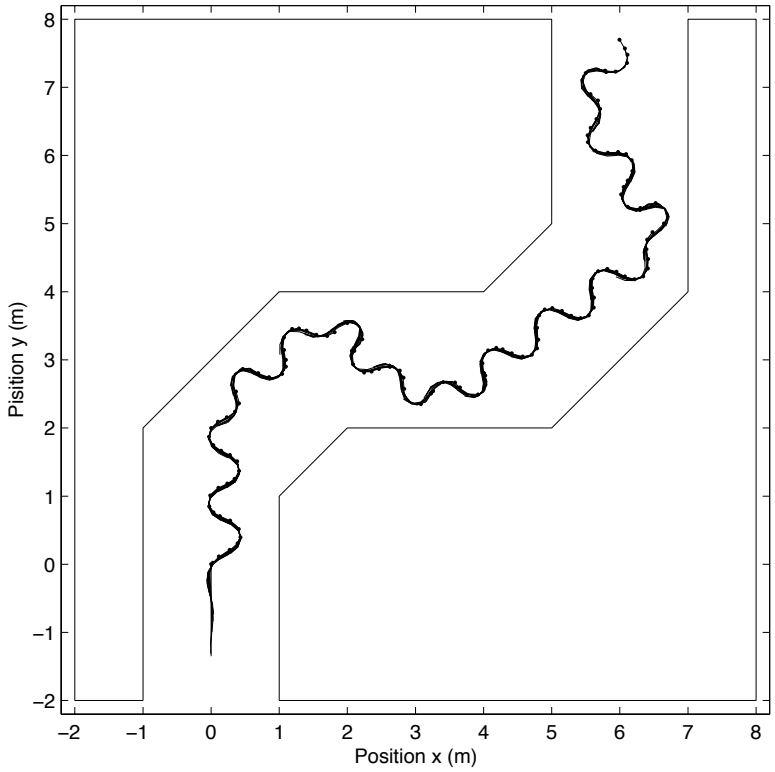


(b) Sensor setting

Natural Actor Critic in Snake Domain (2)

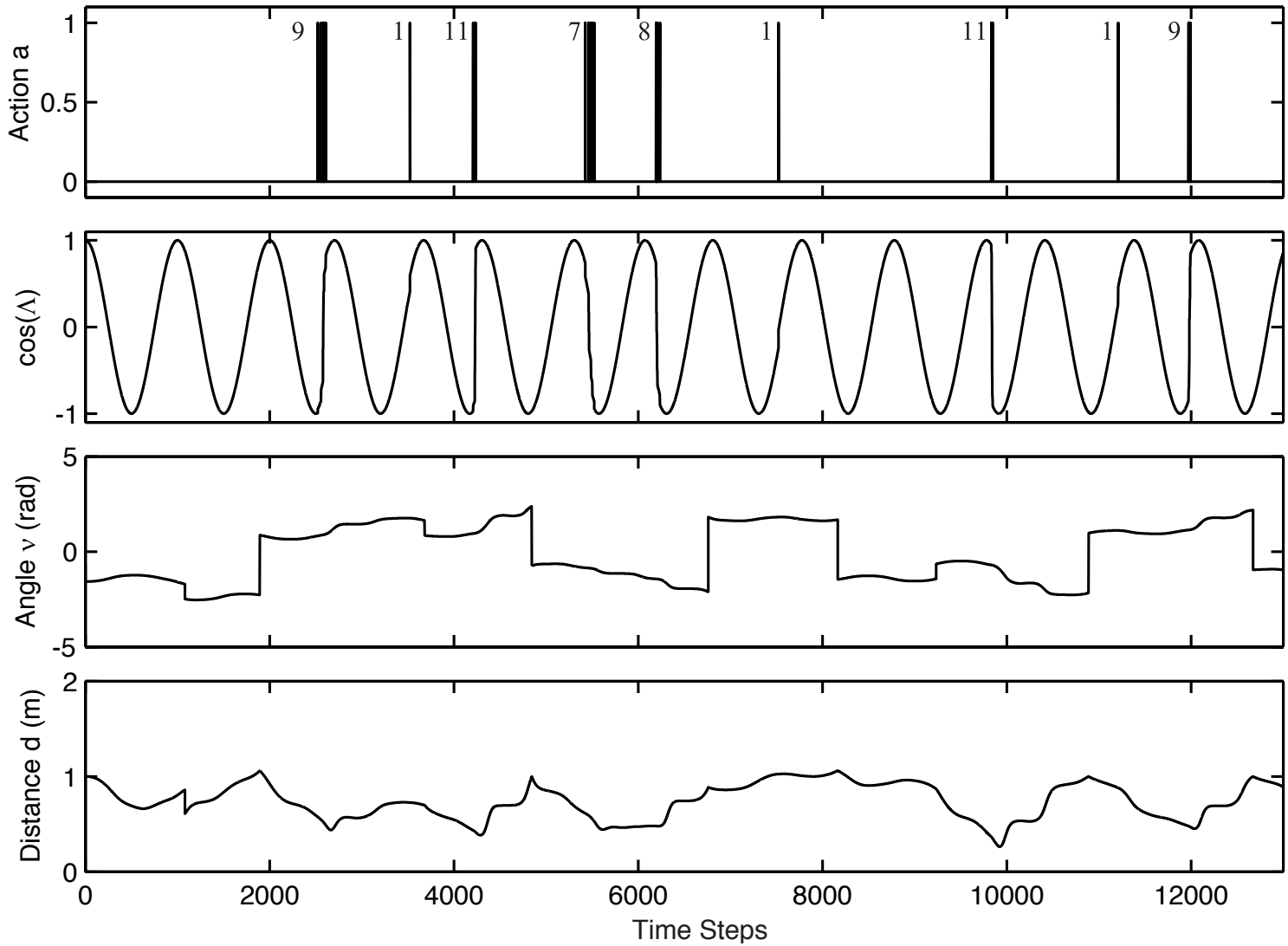


(a) Before learning



(b) After learning

Natural Actor Critic in Snake Domain (3)



Summary of Policy Gradient Algorithms

different variants at the same idea and different manipulations of trying to say move the policy in the direction that gets you more value.

- The policy gradient has many equivalent forms

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ v_t\right] \quad \text{return} \qquad \text{REINFORCE}$$

$$= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ Q^w(s,a)\right] \quad \text{value function} \qquad \text{Q Actor-Critic}$$

$$= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ A^w(s,a)\right] \qquad \text{Advantage Actor-Critic}$$

advantage function

$$= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ \delta\right] \quad \text{TD error} \qquad \text{TD Actor-Critic}$$

$$= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ \delta e\right] \qquad \text{TD}(\lambda) \text{ Actor-Critic}$$

$$G_\theta^{-1}\nabla_\theta J(\theta) = w \qquad \text{Natural Actor-Critic}$$

eligibility trace,
current score we accumulate an eligibility over all of our scores

deterministic actor critic
where we basically move
in the direction that gets us more Q.

- Each leads a stochastic gradient ascent algorithm

We just drop the expectation by sampling the thing which is inside.

- Critic uses policy evaluation (e.g. MC or TD learning) to estimate $Q^\pi(s,a)$, $A^\pi(s,a)$ or $V^\pi(s)$