Lecture 7: Policy Gradient methods

→ optimize the policy directly [Model free]

Previously, VFA → $V_\theta(s) \approx V^\pi(s)$

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

and policy generated based on value function.

Here, parameterize policy → $\pi_\theta(s,a) = P(a|s,\theta)$.

↳ some function approximator.

main motivation

↳ Large scale RL

Policy based

(In value based, we learnt a value function)

→ No value function

→ Learnt policy

Use Gradient Ascent to maximize reward of the policy

→ directly follow the policy gradient

(Why policy based?

→ policy can be more compact representation

→ better convergence properties

→ effective in continuous space (we had 'max' in value function)

↳ can't do in continuous action space.

→ learn stochastic policies

Disadvantages:

→ Naive can be inefficient, high variance, slower.

→ Policy can be local optimum. converged to, Why stochastic policy ?

deterministic policy → can be easily exploitable

→ eg. Rock paper scissor → if we always do 'Rock', the opponent will do paper and defeat us every time

Nash equilibrium → game theoretic equivalent of optimality.

∴ optimal behaviour → stochastic & uniform random

→ state aliasing → when markov property doesn't hold

eg. partially observable MDPs → see some part / some features that give incomplete information.

(for eg. in slide)

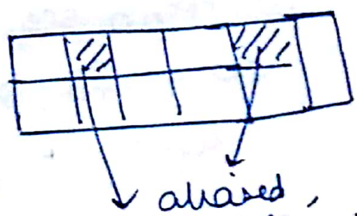
eg. we have features

$\phi(s,a) = I$ (wall to N, a = move E).

Difference betn value based and policy based RL :

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

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



aliased, can't distinguish betn them

policy will always choose the same method. ∴ can get stuck for a long time

then deterministic in the case of value based RL (greedy)

But stochastic policy → policy to randomly move E/W in marked states

$\pi(\text{move E}) = 0.5$

$\pi(\text{move W}) = 0.5$

stochastic > deterministic for state aliasing. → not a complete MDP. → where deterministic always reaches optimal policy

Policy objective functions

① → For episodic functions/env. → we can use the start value meaning that starting from s_1, the value V_1 that we can get. $J_1(\theta) = V^{\pi_\theta}(s_1) = E_{\pi_\theta}(v_1)$ → total reward

(If there is a notion of start state).

(2) For continuing env → use the average value from that point onwards

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

(3) or average reward per time step

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

(+) Same policy gradient → works for all. 3.
only distribut^n function changes → $d^{\pi_\theta}(s)$.
pr. of being in a state.

Policy-based RL → optimization problem.
→ Find θ to maximize $J(\theta)$

We'll do sequential optimizat^n → not wait until the very end before optimizing → learning within the lifetime of the agent.

Finite difference PG → search for a local max. in $J(\theta)$ by ascending the gradient of the policy w.r.t θ.

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

$$\nabla_\theta J(\theta) \xrightarrow{PG} = \begin{bmatrix} \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{bmatrix}$$

For each dimension k, perturb the parameters slightly and take difference,

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

u_k → unit vector with k^{th} component $= 1$ (rest $= 0$)

→ n evaluat^ns for n-dimensions
→ noisy & inefficient
→ works for arbitrary policy → need not be differentiable

(Finite differences).

FD → gives numerical estimate.
→ collapses for high # dimensions

MC Policy Gradient

→ assume policy differentiable whenever it is non-zero

→ Goal: compute gradient analytically

→ assume: know the gradient eg. NN/ softmax etc. as we created it

→ Likelihood ratios → $\nabla_\theta \pi_\theta(s,a) = \pi_\theta(s,a) \frac{\nabla_\theta \pi_\theta(s,a)}{\pi_\theta(s,a)}$

we want to take the expectatn. $= \pi_\theta(s,a) \nabla_\theta \log \pi_\theta(s,a)$

→ score functn → $\nabla_\theta \log \pi_\theta(s,a)$ → max likelihood.

we want to follow this gradient.

eg. Softmax policy → smoothly parameterized policy

→ linear combinations of features for actions, $\phi(s,a)^T \theta$.

Prob $\propto e^{\phi^T \theta}$ → linear softmax policy

we choose the one with higher prob.

(Discrete domain)

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

(Score functn)

actual action feature

average feature for all actions we might have taken

Gaussian policy

$\mu(s) = \phi(s,a)^T \theta$

→ fixed / parameterized

Policy is gaussian → $a \sim N(\mu(s), \sigma^2)$

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

how much more than usual is being done.

One-step MDP

→ starting in state $s \sim d(s)$

→ terminate after one time step with reward $r = R_{s,a}$

→ use likelihood ratios : (pick action that gets more reward).

$$J(\theta) = E_{\pi_\theta}[r]$$

(all 3 objective functions become the same here)

$$= \sum_{s \in S} d(s) \sum_{a} \pi_\theta(s,a) R_{s,a}$$

$$\nabla_\theta J(\theta) = \sum_{s} d(s) \sum_{a \in A} \pi_\theta(s,a) \nabla_\theta \log \pi_\theta(s,a) R_{s,a}$$

to get more reward, just more towards the directn given by score times reward.

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

→ actual that we saw ↓ sampled reward

take action, compute score × reward → gradient step.

Now, same in multistep MDP.

→ replace immediate reward with value function which gives true policy gradient. (model-free)

→ policy gradient theorem

$$\nabla_\theta J(\theta) = E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) \, Q^{\pi_\theta}(s,a)].$$

→ adjust such that it does more of the good things & less of the bad things

For supervised learning → no value functn term (main difference)

Monte Carlo Policy Gradient (REINFORCE)

→ update parameters by SGD

→ sample expectation.

→ use return v_t → unbiased sample of $Q^{\pi_\theta}(s_t, a_t)$

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

REINFORCE

Initialize θ arbitrary

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

∀ for $t = 1 \cdots T$ do

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

→ from time step t onwards (the same we have been using).

end for

end for

return θ

end function

MCPG → · curve is smooth

· slow 100 million iterations

Actor Critic methods (AC).

main idea

instead of using return, use a critic for learning

↓

VFA

MCPG → still high variance / noisy

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

2 sets of parameters

Critic → updates action-value function parameters w.

Actor → updates policy parameters θ, as judged by critic

↓

doing stuff in the world.

∴ Approximate PG. (APG)

$$\nabla_\theta J(\theta) \simeq E_\theta [\nabla_\theta \log \pi_\theta(s,a)\ Q_w(s,a)]$$

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

Critic → policy evaluation → How good is current policy π_θ for current params θ.

- MC
- TD(λ)
- Least squares

linear value fn approx → $Q_w(s,a) = \phi(s,a)^T w$

↓

Action value actor critic

Critic → update w by TD(0) (linear)

Actor → update θ by policy gradient

function QAC.

Initialize s, θ

sample $a \sim \pi_\theta(s)$

for each step do

sample reward $r = R_s^a$, sample next state $s' \sim P_{s}^a$

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

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

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

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

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

end for

end function

(online algorithm)

initialize π_θ → arbitrarily

Bias in actor-critic algorithms

→ using VFA and approximating the PG → introduces bias

→ But choosing the VFA carefully, we can avoid the bias to get the true PG. → compatible

2 conditions:

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

2) w minimizes the mean square error:

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

Then PG → exact.

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

$\nabla_w(\varepsilon) = 0$,

$$E[(Q_w(s,a) - Q_{\pi_\theta}(s,a))\nabla_w Q_w(s,a)] = 0$$

$$\Rightarrow E_{\pi_\theta}[(\quad)\nabla_\theta \log \pi_\theta(s,a)] = 0 \Rightarrow \text{prog}$$

$$\Rightarrow E_{\pi_\theta}[Q_w(s,a)\nabla_\theta \log \pi_\theta(s,a)] = E_{\pi_\theta}[Q_{\pi_\theta}(s,a)\nabla\log(\,)]$$

Softmax policy → get global optimum in table lookup state.

A C space
evaluating action
performing action.

Tricks
① Reduce variance using Baseline without changing expectat'n (i.e. ascen dir'n).

Subtract Baseline $\underline{\underline{B(S)}}$

$$E_{\pi_\theta}(\nabla_\theta \log \pi_\theta(s,a)\, B(s)) = \sum_s d^{\pi_\theta}(s) \sum_a \nabla_\theta \pi_\theta(s,a)\, B(s)$$

choose such that
$$= \sum_s d^{\pi_\theta}(s)\, B(s)\, \nabla_\theta \Big(\sum_a \pi_\theta(s,a)\Big)$$
and just → $= 0$
reduces variance
$$= 0$$

sum = 1
$\nabla_\theta 1 = 0$

So, can add / subtract any term of this form.

Good baseline ⇒ $B(s) = V^{\pi_\theta}(s)$ (state value function).

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

$$\nabla_\theta J_\theta(\theta) = E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\, A^{\pi_\theta}(s,a)]$$

How much better than usual to take that action.

How to adjust policy to achieve that action

$A^{\pi_\theta}(s,a)$ → significantly reduces variance

Ways to estimate:

① estimate both V and Q → take difference

② $V_v(s) = V^{\pi_\theta}(s)$

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

$A(s,a) = Q_w - V_v$.

update both value functions by TD.

② TD error → sample of the advantage function.

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

$$E_{\pi_\theta}[\delta^{\pi_\theta} \mid s,a] = E_{\pi_\theta}[r + \gamma V^{\pi_\theta}(s') \mid s,a] - V^{\pi_\theta}(s)$$

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

TD error is unbiased sample of advantage function (AF)

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

assuming we have the true values.

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

approximate TD error ⇒ $\delta = r + \gamma V_v(s') - V_v(s)$

only requires 1 set of parameters → v.

Critics at diff. time scales:

→ MC → v_t is target. →TD(λ) → v_t^λ → target.

→ TD(0) → $r + \gamma V(s')$.

use backward view → eligibility traces.

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

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

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

Same idea for actors

apply PG with ET.

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

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

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

apply update online

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

introduce bias by bootstrapping to reduce variance

Alternative PG directions

→ Gradient ascent algos → can follow any ascent dirⁿ

→ good dirⁿ → significantly speed convergence.

So far, stochastic policies → can be very noisy to sample and estimate (noise) → really hard to estimate.

Natural policy gradient → parameterizatⁿ independent.

→ find ascent dirⁿ closest to vanilla gradient, when changing policy by a small fixed amount!

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

work directly on ~~deter~~ limiting case, rather than adding noise in policy

G_θ → Fisher information matrix

→ start with deterministic policy to get small objective

after taking limiting case, noise → 0.

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

Flashback to compatible value functⁿs.

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

⇒ score = features

In case of deterministic policy gradient → directly take expectatⁿ of gradient of Q function.

→ limiting case of stochastic AC.

Scanned by CamScanner