(70) Policy - based methods.

↓

RL is ultimately about learning an optimal policy from interaction with the environment

↓

till now, saw value based methods, where we need to estimate an optimal action value function, to estimate an optimal policy.

↓

okay for env. with small state spaces ← but for large state space environments

↓

some sort of approximation has been formulated till now.

↓ discretization → tile coding, radial basis function

↓ function approximation → learn to represent the action value function in the form of a neural network

→ allows non-linear function approximation

↓

env. state as i/p to neural network

↓

o/p was value of each possible action

↓

and as per the previous method, action is selected on the basis of these values, which can done, either on the basis of greedy or ε - greedy.

← (s) Q table

↓

neural net (action values)

↓

choose based on preferred technique.

rest process can stay same.

* But optimal action value function has been estimated before estimation of optimal policy.

↓

can policy be found directly? without worrying about the value function at all.

↓

yes → policy based methods.

→ neural network to approximate policy → state as i/p.

↓

in the o/p layer, rather than return action values, it can return probabilities of the actions, & accordingly sample from those. → sampling: stochastic policy.

greedy ↓ deterministic policy ←

↓

i.e. 2 possible actions, 2 nodes in the final layer.

↓

agent uses this policy to interact with the env. by passing the state to the network.

↓

we need to then select appropriate weights for the network, so that for each state that is passed o/p is action probabilities where the optimal action is most likely to be selected.

↓

initially random values and then fine-tuned to maximize reward as more interaction takes place, gradually mastering the task.

(72)

* for continous action space.

↓

neural network has one node for each action entry (or index)

(i.e. for each constituent of the action vector)

↓

for example: bipedal walker

Num	Name	min	max
0	Hip-1 (Z/V)	-1	1
1	Knee-1 (Z/V)	-1	1
2	Hip-2 (Z/V)	-1	1
3	Knee-2 (Z/V)	-1	1

↓

action: vector of four numbers, so the o/p layer of the policy network will have four nodes

↓

since every value must be a number between -1 and 1, a tanh activation will be added.

↓

similarly for continous mountain car benchmark

* Hill climbing: optimization technique used to find maximum of a function

agent's goal: to maximize the expected return for any environment, J

weights in the network: θ

mathematical relationship b/w θ & expected return J

as θ → encodes the policy → determines the action
(affects the policy)
↓
influences reward
↓
used to get expected return, J

$$J(\theta) = \sum_{\tau} P(\tau; \theta) R(\tau)$$ → it exists (mathematical relation b/w θ, and J)

↓

goal is to find θ, that maximize J

↓

consider a case where the neural net has only 2 weights : θ_1, θ_2

↓

~~app~~ estimating θ_1, θ_2 values to maximize the reward, can be done using technique called gradient ascent.

i.e. rather than moving in the direction opp. to the gradient (as done in gradient descent) to find minima, we move in direction of the gradient.

↓

begins with an initial guess.

↓

this process is repeated

↓

to do this, gradients need to be evaluated. (later)

↓

Now: hill climbing

(74) As with gradient ascent, initiate with a random set of values for θ. collecting an episode corresponding to the policy for the weights of the network, and then the record is returned.

↓

add random noise to weights → gaussian or some other model → another set of candidate values

↓

if better good. we change the current weights to the new ones,

↓

else we go back to our current net weights.

↓

repeated till optimal policy not approximated.

Hill climbing may lead to being stuck in a local minima.

(•) psuedocode : Hill climbing :

i) initialize the weights θ in the policy arbitarily

ii) collect an episode with θ, and record the return G.

$\theta_{best} \leftarrow \theta$

$G_{best} \leftarrow G$

iii) Repeat until env. solved :

* Random noise ~~added~~ added to θ_{best}, to get a new set of weights. θ_{new}.
* collect an episode with θ_{new}, to get G_{new}
* If $G_{new} > G_{best}$:

  $\theta_{best} \leftarrow \theta_{new}$, $G_{best} \leftarrow G_{new}$.

* general class of approaches, that find arg $\max_\theta J(\theta)$ through randomly perturbing the most recent best estimate as stochastic policy search. Likewise J, can be refered to as an objective function (we would like to maximize it)

↓

one improvement could be to select a number of neighbouring policies at each iteration and then picking the best among them.

↓

gives the agent an idea about the neighbourhood of the policy.

↓

steepest ascent hill climbing

(reduces the risk of selecting a next policy that may lead to a suboptimal solution)

- getting stuck in local minimas → a) random restarts

b) simulated annealing

↓

predefined schedule to control how the policy space is explored

↓

initially : large noise parameter (broad neighbourhood to explore)

↓

gradually noise is reduced as we get closer & closer to the optimal solution

↓

analogous to kneeling iron : heating it up and letting it cooldown gradually.

(76) * adaptive noise

↓

as better policy is estimated, radius for search can be reduced. for generating the next policy

↓

translates to reducing or decaying the variance of the Gaussian noise we add.,

but if donot find a better policy, it is better to increase the search radius, and to continue exploring from the current best policy.

↓

reduces the probability of it getting stuck.

→ Black-box optimization:

hill climbing

→ steepest ascent hill climbing

↓

techniques used so far, do not have any idea about their usecase, and are as such not dependent on the use case. These algorithms just care about approximating the θ that maximize J, and can be applied to any similar case.

↓

other black box optimization techniques:

i) cross - entropy method

ii) evolution strategies.

i) cross - entropy method:

leveraging useful information from the weights that aren't selected as the best

↓

rather than selecting the best, selecting the top 10 or 20% and then taking its average.

ii) evolution strategies. → name originally comes from biological evolution

↓

best policy : weighted sum of all candidate policies, where policies with higher return has higher say or get a higher weight.

↓

most successful individuals have more influence on the next generation

* why policy-based methods?

i) simplicity → directly getting to the problem at hand, rather than worrying on computation of some intermediate step.

↓

also avoids storing additional data, that may or may not be useful

ii) stochastic policies.

iii) continous action space.

↓

(well suited)

(value based isn't)

* being able to get a true stochastic policy

↓

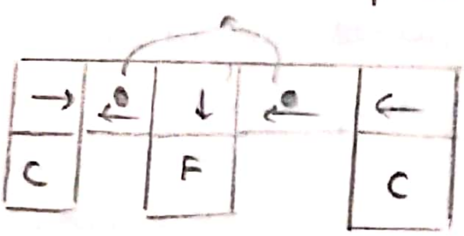
ε-greedy is sort of a hack

aliased states : 2 or more states that are perceived to be identical, but are actually different.

↓

partial observability of the environment

these are the problematic states for blindfolded monkey.



↓

all features are identical

↓

action values are also equal since they map to the same state, ∴ corresponding best action will become

← will be wrong for atleast 1 aliased state

↓

oscillations

↳ he could come out using ε-greedy

but this can be really inefficient & could take a lot of time.

↓

and high epsilon could lead to bad actions.

↓

equal prob to both actions!

↓

value based approach tends to learn a deterministic or near-deterministic policy, whereas a policy based approach can learn the desired stochastic policy.

finding the most profitable or rewarding action over a continuous action space is an optimization problem in itself.

↓

(the more features the action vector has, harder it gets to find the other corresponding maxima)

↓

hence it is better that we can map a given state to an action directly.

↓

even ~~tho~~ if the resulting policy is a bit more complex, computation required would drop significantly. enabled by a policy based method.