

Some terms

Model-free: not learning the transition and reward, only need to get observations from the real world or simulator

Prediction to control - prediction into a control policy

Generalized policy iteration

Take argmax get the policy

Run the policy and estimate a new Q function

Don't have to evaluate the policy fully

Do one step and update one data (Q function)

Update greedily wrt the Q function

By being greedy (get a new policy) and improve it

Monte Carlo

Take many trajectories and average the trajectories

Temporal difference learning

Immediately update our prediction just after one step

But there is no return from the long trajectory

Current return of u

U is the estimate

Monte Carlo error: considers all the previous steps

vs.

TD error: how difference are my difference at this time step

n step TD

better tradeoff between bias and variance

When n is ∞ , it becomes Monte Carlo

Average can somewhat reduce variance (need less data to get good estimate, learn faster as it converges faster)

TD(λ)

n step is weighted by $\frac{\lambda}{n-1}$

0 forget the future take 1 step

1 close to 1 important for a long time

We are averaging the estimates (more expensive)

ADP

Run v_i or π_i

not trying to estimate the model only the values

need to have simulator of the real world to go on with RL

Build a model using function approximation

What is the transition function to use?

What reward to use

Successful has been parameterised Q function, supervised learning etc.

Model-free are easier to work with

We usually mix MC with TD in practice, less estimate and not much variance

SARSA

TD control

Uses GPI, on policy (improve the same policy)

target = $R(s) + \gamma Q(s', a')$

SARSA will converge to optimal policy on tabular cases

MC on the windy gridworld

Converges slower but eventually reaches optimal

Have to use old policy until end of the trajectory

SARSA changes policy at every time step, may wander around at the start, but will continue learning

ϵ greedy: Take the action suggested by the Q function or random exploration

Improving the current policy with ϵ greedy, when present does not converging to the optimal policy

trying to improve the policy

environment might change, world isn't quite a MDP, it can be robust, model may change over time

Might still fall off the cliff

Predictive problem learning (passive TD) has some restrictions

Q-learning

Only have the q function (acts as policy on its own)

Not in SARSA: $\gamma \max_a$

best action at the next action of Q function

E.g currently have a deployed policy, at wild change the policy with SARSA, train the target policy

Behavior policy can be some other policy that is running, generate data to train the policy

If stop learning then Q-learning will outperform Sarsa

Similar concept to ϵ greedy

Let's go to part of the space that is exploit less

Issues when state space is large

Update policy then take argmax - given the current policy from the Q function, estimate a new Q function and construct a new policy by taking argmax of Q function

Q learning can learn off policy (uses different policy)

If the count is small give bonus, if we try many times reduce bonus

Function approximation

Estimating the Q

Figure out what are the important features and extract out

E.g. for a chess, queen is worth 5 points, pawn worth, knight worth

RL will learn the parameter vector Θ

Allows generalisation observations so that it can work on entire state space

Small fraction of the state

Applications: Go, Deep neural network, OpenAI5

MCT Search on simulator, for online search value function and combine with MCTS

Do learning with function approximation

Runs in the form of ridge regression

Online learning

One example, learn an example and update the weight vector

Update theta to improve estimate, step in the direction of the negative gradient

$\hat{U}(s)$ function approximator, estimate from the function

target $u_j(s)$ At every time step it is the sum of the reward to the future, measured from the environment

It could be the q function

Semi gradient - don't include in the computation

gradient wrt the parameter Θ

we treat $\gamma\hat{U}(s')$ as a fixed target

When we move on-policy we lose most of our policy, have to do lots of tuning

e.g. doesn't converge

Deep Q-learning

Techniques to do better: Experience replay

large buffer D, old sample does not mix with the recent one

$(s_t, a_t, r_{t+1}, s_{t+1})$ max over a without the k

learn and freeze it and use as target θ^-

θ^- Q function from current - c steps

High profile achievement as one architecture can work across 50 atari games

Set a new target and change every c steps

Extract with a linear function of features

E.g. position of the ball

provably good - how long would it take to go to epsilon