RL (David Silver)

Info state / markov state

$$P(S_{t+1} \mid S_t) = P(S_{t+1} \mid S_1, S_2 \ldots S_t)$$

future is independent of the past given the present

State representation defines what happens next

Fully observable environments

agent directly observes env state

$$O_t = S_t^a = S_t^e$$

Agent state = env state = info state

Markov Decision process (MDP)

Partially observable env

agent indirectly observes env

eg: robot with a camera w/o gps

: trading agent with only current prices

: poker agent

agent state ≠ env state

agent must make their own rep. S_t^a

Partially Observable MDP (POMDP)

Sathvik Udupa

different ways:

- complete history

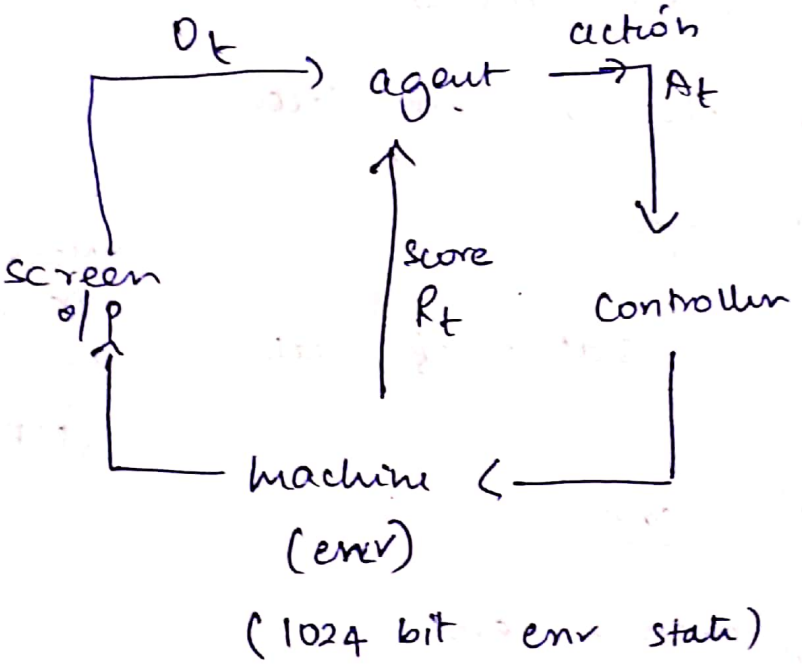
$$S_t^a = H_t$$

- Beliefs of env state

$$S_t^a = \left(P[S_t^e = s^1], \ldots P[S_t^e = s^n]\right)$$

- RNNs

Atari eg:



(1024 bit env state)

Inside an RL agent

main concepts

- Policy : how it picks action
- value : how good is it in a particular state
- Model : agent rep of the env

Policy

A map from state to action

- Deterministic policy

$$a = \pi(s)$$

- stocastic policy

$$\pi(a|s) = P(A_t|S=s)$$

Value

- Prediction of expected future reward

$$V_\pi(s) = E_\pi[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \cdots | S_t = s]$$

↳ future rewards

γ → discounting

Model

- predicts what env will do
  - transition model
    predicting next state
  - reward model
    predicting next reward.

$$P^a_{ss'} = P[S' = s' \mid S = s, A = a]$$

$$R^a_s = \mathbb{E}[R \mid s = S, A = a]$$

Characterising RL agent

(1)

- Value based
  - No policy (implicit)
  - Value function

- Policy based
  - Policy
  - No value fn

- Actor critic
  - Policy
  - Value function

(2) - Model free
  - Policy and/or Value fn
  - No Model
- Model based
  - Policy and/or Value fn
  - ~~No~~ Model

Problems in RL

- RL
  - env is unknown (robots)
  - env interacts with env
  - env improves its policy
- Planning
  - Rules known
  - env model known
  - agent performs computation
  - improves policy
  - could plan ahead using methods like tree search.

Exploration & Exploitation

- Trial & error
- goal is to discover good policy.
- exploration - more info on envi
  exploitation - exploits available info to get more reward

Markov Decision Process

- formally describes an environment for RL
- Almost all RL problems can be formalized as MDP
- Any partially observable problem can be converted into a MDP
- Bandits are MDPs with one state

State transition matrix

transition probability

$$P_{ss'} = P[S_{t+1} = s' \mid S_t = s]$$

Markov process

sequence of random states that have Markov property.

A Markov Process is a tuple (S, P)

S → set of states

P → transition probability matrix

Markov Reward Process

(S, P, R, γ)

R → Reward fn

(Immediate reward) $R_s = E[R_{t+1} \mid S_t = s]$

Return G_t → total discounted reward from time t

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{0}^{\infty} \gamma^k R_{t+k+1}$$

discount is the present value of future rewards

Why discount?

- Because we don't have a perfect model. So there will be uncertainity in the future.
- Mathematically convinient
- Avoids infinite returns

- On financial setting money now might be better than delayed money.
- Animal / human behavior.

Value fn

long term value of state s

$$V(s) = E[G_t \mid S_t = s]$$

Bellman Eq

$$V(s) = E[G_t \mid S_t = s]$$

$$= E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \mid S_t = s]$$

$$= E[R_{t+1} + \gamma [R_{t+2} + \gamma R_{t+3} + \cdots] \mid S_t = s]$$

$$= E[R_{t+1} + \gamma G_{t+1} \mid S_t = s]$$

$$= E[R_{t+1} + V(s+1) \mid S_t = s]$$

$$V = R + \gamma P V$$

immediate reward

Transition matrix

value of where we end up

$$\begin{bmatrix} V_1 \\ V_2 \\ \vdots \\ V_n \end{bmatrix} = \begin{bmatrix} R_1 \\ R_2 \\ \vdots \\ R_n \end{bmatrix} + \gamma \begin{bmatrix} P_{11} & \cdots & P_{1n} \\ & & \\ P_{n1} & \cdots & P_{nn} \end{bmatrix} \begin{bmatrix} V(1) \\ V(2) \\ \vdots \\ V(n) \end{bmatrix}$$

It can be solved directly,

$$V = R + \gamma P V$$

$$(1 - \gamma P) V = R$$

$$V = (1 - \gamma P)^{-1} R$$

$O(n^3)$ for n states

→ To solve large MDPs

- DP
- Monte-Carlo
- Temporal-Difference Learning

MDP is a Markov Reward Process with decisions

$$(S, A, P, R, \gamma)$$

$$P^a_{ss'} = P[S_{t+1} = s' \mid S_t = s, A_t = a]$$

$$R_s^a = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$$

Policy π is a distribution over actions given states.

$$\pi(a|s) = P[A_t = a | S_t = s]$$

MDP policies depend on current state

state value

$$v_\pi(s) = E_\pi [G_t | S_t = s]$$

action value

$$q_\pi(s,a) = E_\pi [G_t | S_t = s, A_t = a]$$

$$V_\pi = E_\pi [R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s]$$

$$q_\pi = E_\pi [R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a]$$

Optimal value function

$$v_*(s) = \max_\pi v_\pi(s)$$

$$q_*(s,a) = \max_\pi q_\pi(s,a)$$

optimal policy

$\pi \geq \pi'$ if $V_\pi(s) \geq V_{\pi'}(s)$ $\forall s$.

Theorem

- For any MDP, there is an optimal policy better than or equal to all the policy.

$\pi_* > \pi \quad \forall \pi$

- All optimal policies achieve the optimal value function & action-value fn.

$$V_{\pi*}(s) = V_*(s)$$

$$q_{\pi*}(s,a) = q_*(s,a)$$

An optimal policy can be found by maximum over $q_*(s,a)$

$$\pi_*(a|s) = \begin{cases} 1 & \text{if } a = \arg\max_a q_*(s|a) \\ 0 & \text{otherwise} \end{cases}$$

Bellman Optimality Eq for V_*

$$V_*(s) = \max_a q_*(s,a)$$

$$q_*(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V_*(s')$$

$$\therefore V_*(s) = \max R_s^a + \gamma \sum P_{ss'}^a V_*(s')$$

$$q_*(s,a) = R_s^a + \gamma \sum P_{ss'}^a \max_{a'} q_*(s',a')$$

Thum are non linear.

To solve it

- Value iteration
- policy iteration } DP based
- Q-learning
- Sarsa

Planning by DP

- Sequential a temporal aspect to the problem.
- Programming optimising a 'program' - a policy.
- Breaking down into subproblems and solving them.
- Can be applied on problems which have properties
  - optimal substructure
    Can solve an overall problem by solving optimal solutions of its different pieces.
  - Overlapping subproblems
    Subproblems occur again & again

- MDP satisfies both properties
- Value fn stores and recursive solutions
- For prediction :
  - i/p : MDP (S, A, P, R, γ) & π
    - MRP $(S, P^{\pi}, R^{\pi}, \gamma)$
  - o/p : value function V_{π}
- For control :
  - i/p
    - MDP (S, A, P, R, γ)
  - o/p
    - optimal V_{π}, π_*

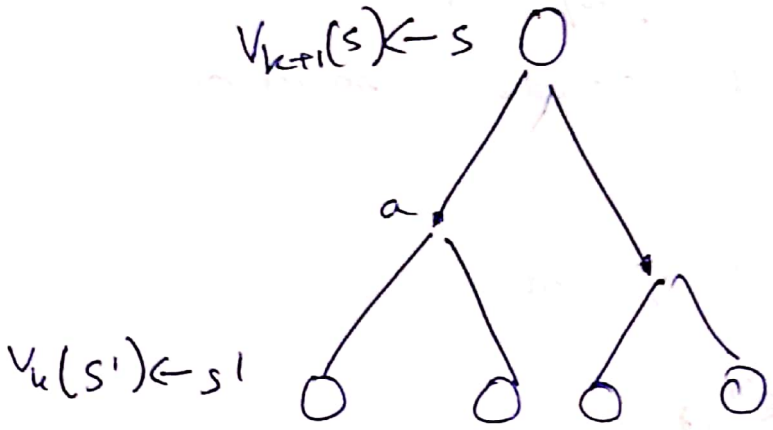
Policy Evaluation

Problem
- To Calculate a given policy π

Soln
- iterative application of Bellman Expectation eq

$V_1 \rightarrow V_2 \rightarrow \dots \rightarrow V_{\pi}$

- Using Synchronous backups
  - At every iteration $k+1$
  - For all states $s \in S$
  - Update $V_{k+1}(s)$ from $V_k(s')$

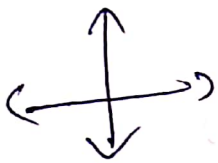


~~Replace~~ place the current value function at ~~the~~ every leaf s_k

Compute new V_k using Bellman eq (Multiply by Probability & sum it up) and repeat this

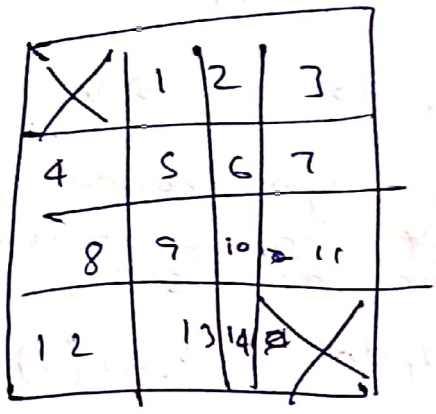
This process will converge on the true value for.

on a small grid



action

$\gamma = 1$



$\gamma = -1$ for all.

1 terminal state (2 squares)

find the avg steps taken to reach terminal with uniform random policy.

$\pi(n|\cdot) = \pi(e|\cdot) = \pi(s|\cdot) = \pi(w|\cdot) = 0.25$

V_k for the random policy

$k=0$

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

(our estimate of every single state on how much steps to reach X)

(start with 0 everywhere)

Then for each state compute Bellan expectation

$k=1$

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	0

at every state, any step will give only -1 reward

$k = 2$

0	-1.75	-2	-2
-1.75	-2	-2	-2
-2	-2	-2	-1.75
-2	-2	-1.75	0

for a.

$$= \frac{(-1 + -1) + (-1 + -1) + (-1 + -1) + (-1 + 0)}{4} = -1.75$$

$k = \infty$

0	-14	-20	-22
-14	-18	-20	-20
-20	-20	-18	-14
-22	-20	-14	0

greedy policy wrt v_k

look at v and predict what value to take.

On 3 iteration it can give optimal policy.

So value fn helps us figure out policy

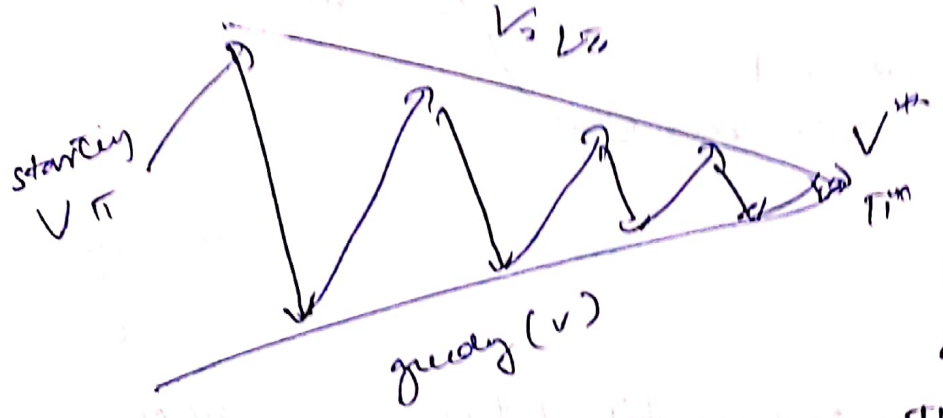
policy iteration

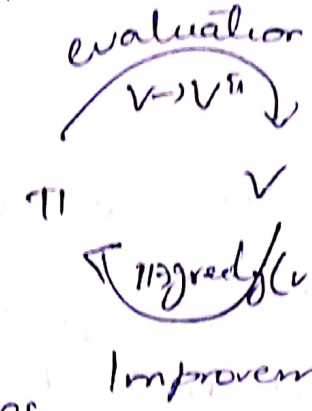
- To find the best policy
  - Evaluate the policy

$$v_\pi(s) = \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} + \dots \mid S_t = s\right]$$

  - Improve the policy by acting greedily

$$\pi' = \text{greedy}(v_\pi)$$

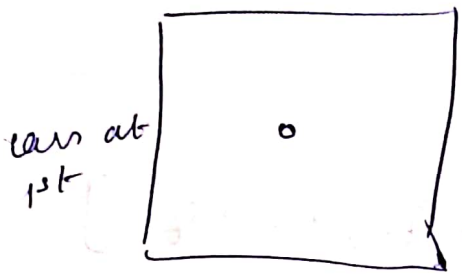




Car rental problem

States
- 2 location, max of 20 cars at each
- Action - Move up to 5 cars b/w location overnight
- Reward - $10 for each rental
- Transition - cars returned & requested randomly
  - 1st location → average requests, avg returns
  - 2nd location → Poisson with λ returns

π_0 — we know P from poisson distribution (initial data)



compute reward → take action greedily (how many cars to move from where to where) → repeat till convergence

---

- Consider a deterministic policy $a = \pi(s)$ (acting greedily always makes it deterministic)

$$\pi'(s) = \underset{a \in A}{\operatorname{argmax}}\ q_\pi(s, a)$$

- This improves value over one step

$$V_{\pi'}(s) \geq V_\pi(s)$$

$$V_\pi(s) \leq V_{\pi'}(s) = E_{\pi'}\left[R_{t+1} + \gamma V_\pi(S_{t+1}) \mid S_t\right]$$

$$\leq E_{\pi'}\left[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) \mid S_t = s\right]$$

$$\leq E_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \dots \mid S_t = s\right] = V_{\pi'}(s)$$

This will make it ~~must stop attract when~~ stop.

At some point it might stop.

if improvements stop,

$$q_\pi(s, \pi'(s)) = \max_a q_\pi(s,a) = q_\pi(s, \pi(s)) = V_\pi(s)$$

↓

This satisfy BOE

$$V_\pi(s) = \max_{a \in A} q_\pi(s,a)$$

Policy iterations solves MDPs.

Modified policy iteration

- Should we include a stopping condition?
- stop after k iteration?
- why not update policy every iteration?

Value iteration

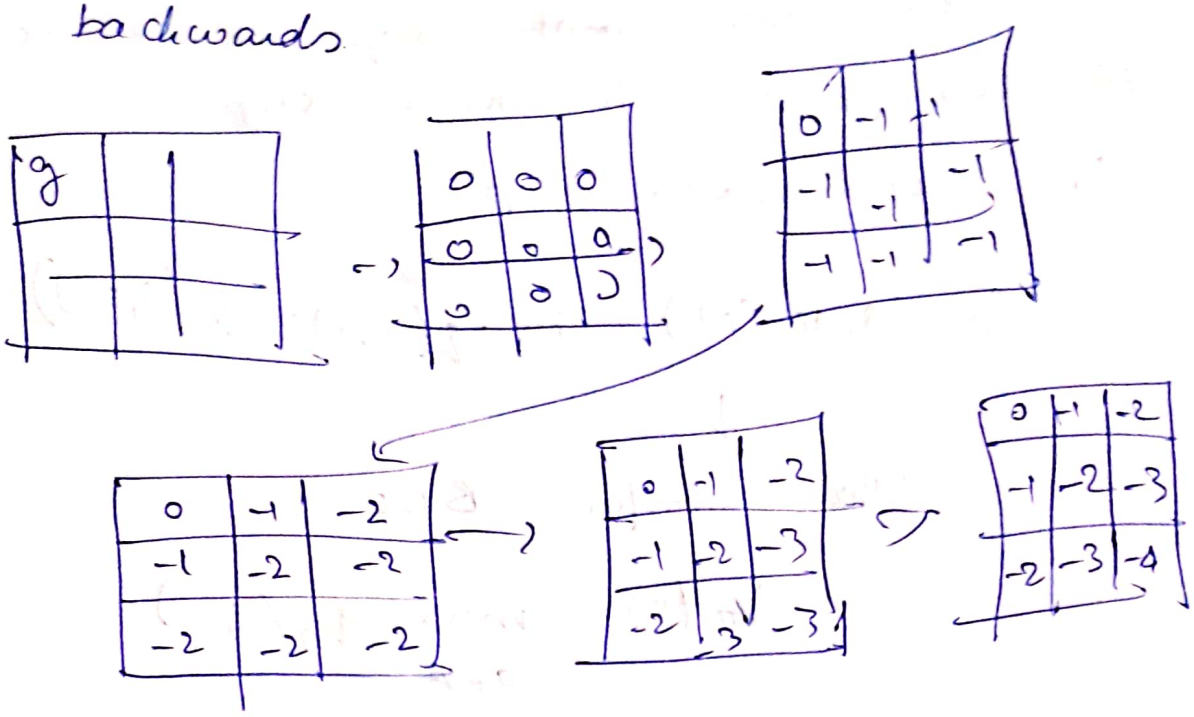
Principle of optimality:

A policy $\pi(a|s)$ achieves the optimal value from state s $v(s) = \pi_*(s)$ if and only if

- For any state s' reachable from s
- π achieves optimal value from state s'

Intuition:

Start with final rewards & work backwards



one step lookahead from the ~~final~~ terminal.

value iteration

- to find optimal policy π

Asynchronous DP

- backup states individually
- will converge if all states are selected at some point.

- in place DP
- prioritized sweeping
- Real time DP

In-place → forget about old & new value funcs
- just use the latest value.

Prioritized Sweeping
- priority queue to see which states are better than others.
- magnitude of error in Bellman equation b/w states

Real-time - Select states that agent selects

DP uses full-width backups,
(computation on all nodes)
- so sample backups

Model free Control Prediction

Monte Carlo learning

- look at complete episodic memory.
- learn V_π from episodes of experience from π

$$G_t = R_{t+1} + \gamma R_{t+2} + \dots \gamma^{T-1} R_T$$

$$V_\pi(s) = E_\pi [G_t | S_t = s]$$

In MC, empirical mean is taken instead of expectation.

First Visit MC policy evaluation

- Consider the first time one visits a state.
- increment a counter ($N(s)$)
- add up total return ($S(s) = S(s) + G_t$)
- take mean from that time
- By law of large numbers, $V(s) \to V_\pi(s)$ as $N(s) \to \infty$

Every Visit MC policy Evaluation

- Consider every visit to the state & summing

Incremental mean

$$\mu_k = \frac{1}{k}\sum_{1}^{k} x_j$$

$$= \frac{1}{k}\left[x_k + \sum_{j=1}^{k-1} x_j\right]$$

$$= \frac{1}{k}\left[x_k + (k-1)\mu_{k-1}\right]$$

$$= \mu_{k-1} + \frac{1}{k}\left(x_k - \mu_{k-1}\right)$$

- k ↙ total current steps
- x_k ↓ new step value
- μ_{k-1} ↓ previous step mean

Incremental MC updates

For each state S_t with G_t

$$N(S_t) = N(S_t) + 1$$

$$V(S_t) = V(S_t) + \frac{1}{N(S_t)}\left(G_t - V(S_t)\right)$$

To forget old episodes replace $\frac{1}{N(S_t)}$ by a constant α.

Temporal-Difference Learning.

- learns from incomplete episodes.
- It updates a guess towards a guess

$$V(S_t) = V(S_t) + \alpha\left[G_t - V(S_t)\right]$$

- Simplest TD algorithm, TD(0)

update value $V(S_t)$ towards estimated return $R_{t+1} + \gamma V(S_{t+1})$

R_{t+1} → immediate reward

$\gamma V(S_{t+1})$ → discounted value of next step.

$$V(S_t) = V(S_t) + \alpha\left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right)$$

$R_{t+1} + \gamma V(S_{t+1})$: TD target

$R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$: TD error

TD vs MC

- TD can learn before final outcome
- MC should wait till end

- TD can learn when it may not see final outcome
- MC can learn only from complete sequences
- TD works in non-terminating env
- MC works in terminating env

Bias, Variance tradeoff

- G_t is unbiased of $v_\pi(s_t)$ in MC.
- True TD is unbiased estimate of $v_\pi(s_t)$ (if v_π is known)
- TD target with $V(s_{t+1})$ is biased estimate of $v_\pi(s_t)$
- TD target has lower variance than return. (return depends on lots of things) (TD target depends on 1 random transition)

MC has good convergence due to no bias

- Not sensitive to initial values
- TD more sensitive to initial values
- TD much more efficient than MC.

- MC converges to solution with min MSE
- TD converges to soln of max likelihood Markov model
- MC more effective in non Markov env
- TD exploits Markov property, much better

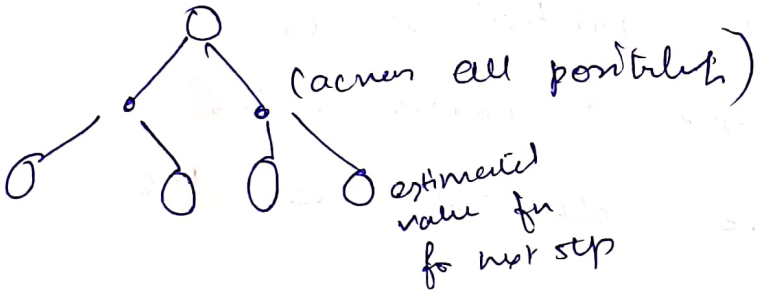
Bootstrapping - Update involves estimate (estimated value for of next step)

- MC does not bootstrap
- TD, DP bootstraps

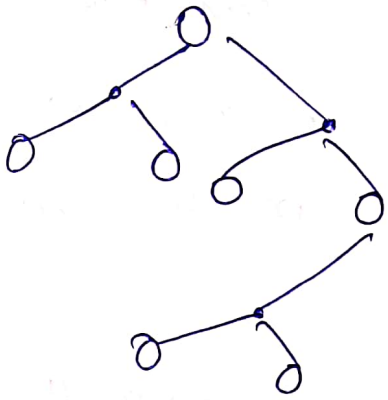
Sampling

- MC samples
- DP does not (every possibility considered)
- TD samples

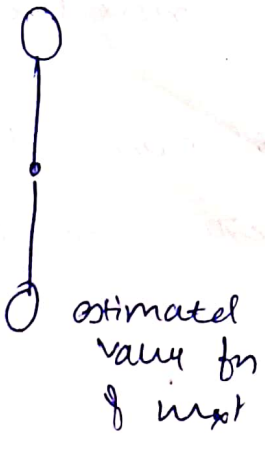
DP



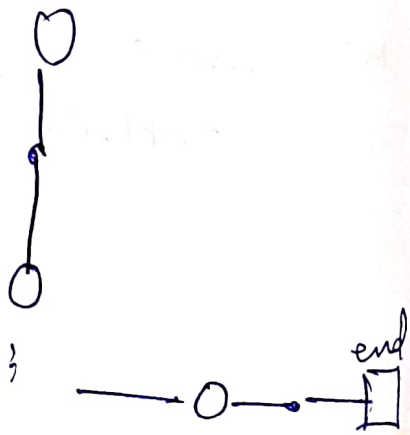
Exhaustive search



TD



MC



TD(λ)

n steps of look ahead and estimating value fn.

n = total steps, TD = MC

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \dots \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$$

$$V(S_t) = V(S_t) + \alpha\left(G_t^{(n)} - V(S_t)\right)$$

To select best n:

average over n-step returns over diff n

eg: avg of 2-step & 4-step

$$\frac{1}{2} G^{(2)} + \frac{1}{2} G^{(4)}$$

λ-return

geometrically weighted avg of all n.

$$G_t^{\lambda} = (1-\lambda) \sum_{1}^{\infty} \lambda^{n-1} G_t^{(n)}$$

geometric weights decay in efficient (low memory)

This is forward view (computation same as TD(0))

Backward view

- keeps eligibility trace $E_t(s)$ for every state s

TD error

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

$$V_t(s) = V(s) + \alpha \delta_t E_t(s)$$

when $\lambda = 0$,
it is TD(0)

when $\lambda = 1$,
credit is differed until end of episode.

Model Free Control

- On policy & Off-policy.

- On-policy MC Control

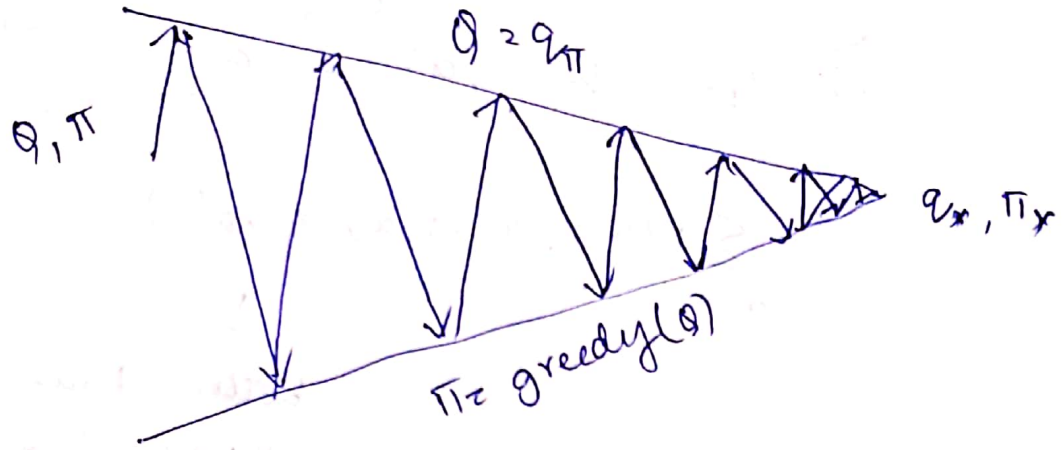
will replacing DP with MC work?

No, because

- no exploration (acting greedily will ignore a lot of state spaces)
- greedy policy over $V(s)$ requires model of MDP to act greedily. (no transition matrix available)

Alternative is to use action value functions $Q(s,a)$ is model free

$$\pi'(s) = \underset{a \in A}{\arg\max}\ Q(s,A)$$



ε - greedy exploration

- Simplest idea
- All m actions are tried with non-zero Prob
- with prob $1-\epsilon$, choose the greedy action
- with prob ϵ, choose an action at random
- This guarantees to continue to explore everything
- with ϵ-greedy, there will be step of policy improvement.

ϵ-greedy

policy Improvement proof.

$$q_\pi(s, \pi'(s)) = \sum_{a \in A} \pi'(a|s)\, q_\pi(s,a)$$

($\pi'(a|s)$: prob that we will take each action)

↓ one step by new policy

$$= \epsilon/m \sum q_\pi(s,a) + (1-\epsilon) \max_{a \in A} q_\pi(s,a)$$

(ϵ/m: probo of taking every action; $(1-\epsilon)$: prob of taking greedy action)

$$\geq \epsilon/m \sum q_\pi(s,a) + (1-\epsilon) \sum \frac{\pi(a|s) - \epsilon/m}{1-\epsilon} q_\pi$$

$$= \sum_{a \in A} \pi(a|s)\, q_\pi(s,a) = V_\pi(s)$$

↓

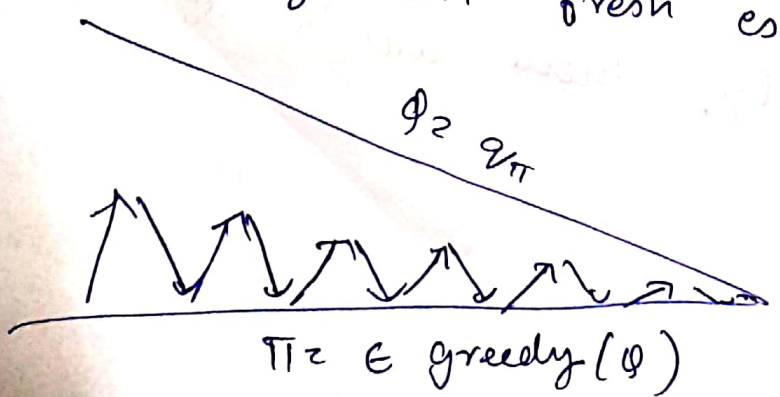
better than value of original policy

this max is considered greater than weighted sum of all the actions

$\therefore V_{\pi'}(s) \geq V_\pi(s)$

update every single episode:

- act greedily wrt fresh estimate



Greedy in Limit with Infinite Exploration (GLIE)

- All action states are infinitely explored many times.
- Policy converges on being greedy.

ϵ-greedy is GLIE if ϵ reduces to zero at $\epsilon_k = \frac{1}{k}$

GLIE MC Control

- Sample k^{th} episode $\{S_1, A_1, R_2, \cdots S_T\} \sim \pi$
- For each state & action

$$N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)} (G_t - Q(S_t, A_t))$$

- Improve policy based on new action value fn

$$\epsilon \leftarrow 1/k.$$

$$\pi \leftarrow \epsilon\text{-greedy}(Q)$$

TD ~~co~~ control advantages over MCC.

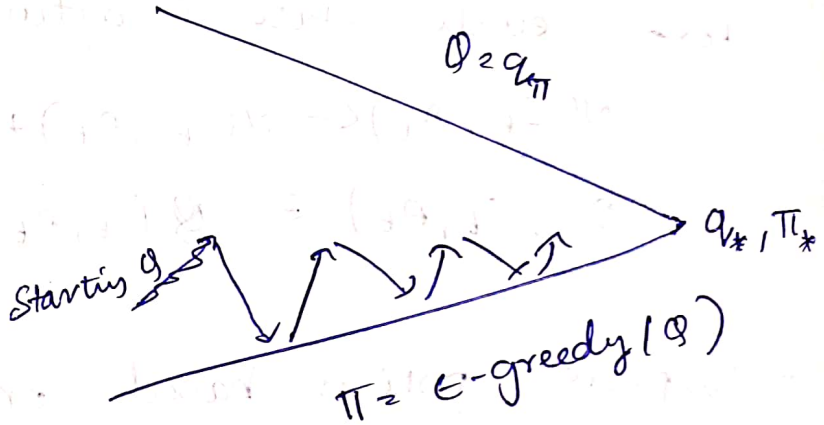
- lower variance
- Online
- Incomplete sequences

Natural idea:

- use TD instead of MC in control loop
  - Apply TD to $Q(S, A)$
  - Use ϵ-greedy
  - Update every time step.

Updating Action Value fns with Sarsa

$$Q(S,A) \leftarrow Q(S,A) + \alpha\left(R + \gamma Q(S',A') - Q(S,A)\right)$$



pseudo code

- Initialize $Q(s,a)$, arbitrarily
- Repeat (for every episode)
  - Initialize S
  - Choose A from S using Q policy derived from Q
  - Repeat (for each step)
    - take action, observe R, S'
    - Choose A' from S' using ϵ-greedy
    - update Q
    - update S, A
  - terminal S

- GLIE Sarsa will converge.
- Robbins - Monro sequence of step-sizes α_t

$$\sum_{t=1}^{\infty} \alpha_t = \infty$$

$$\sum_{t=1}^{\infty} \alpha_t^2 < \infty$$

n-step sarsa

Same as n-step TD,

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha (q_t^{(n)} - Q(S_t, A_t))$$

Sarsa-λ (same as TDλ) (this is not online)

q^λ returns combine all n-step Q-returns $q_t^{(n)}$.

Using weight $(1-\lambda)\lambda^{n-1}$

$$q_t^\lambda = (1-\lambda) \sum_{1}^{\infty} \lambda^{n-1} q_t^{(n)}$$

- Forward view Sarsa(λ)

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha (q_t^\lambda - Q(S_t, A_t))$$

- Backward View Sarsa(λ)
  - same as TD(λ) Backward
  - But sarsa(λ) has one ET for each state action.
  - $E_0(s,a) = 0$
  - $E_t(s,a) = \gamma\lambda E_{t-1}(s,a) + 1(S_t = s, A_t = a)$

Off-policy learning.

- Learning from others.
- Learning from a behavior policy μ
- Re-use experience from old policies
- Learn about optimal policy while following exploratory policy
- Learn about multiple policies while following one policy.

Importance Sampling.

- Estimate expectation of a diff distribution

$$E_{x \sim P}[f(x)] = \sum P(x) f(x)$$

$$= \sum Q(x) \frac{P(x)}{Q(x)} f(x)$$

$$= E_{x \sim Q}\left[\frac{P(x)}{Q(x)} f(x)\right]$$

Importance Sampling ~~for~~ in MC

- have to multiply IS ratio to every step
- use returns generated from μ to evaluate π
- Extremely high variance (policies don't match over sequences)
- Not used in practice.

Importance Sampling for Off-policy TD.

- Multiply for one step
- Lower Variance
- Policies need to be similar only for one step.

Q-learning

- Specific to TD(0) or SARSA(0)
- use action state $Q(s,a)$
- select next action according to behavior policy
- Consider alternative successor action if we followed target policy.

$$A_{t+1} \sim \mu(\cdot|S_t)$$

$$A' \sim \pi(\cdot|S_t)$$

- And update $Q(S_t, A_t)$ towards value of alternative action

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha\left(R_{t+1} + \gamma Q(S_{t+1}, A') - Q(S_t, A_t)\right)$$

($Q(S_t, A_t)$: μ policy; $Q(S_{t+1}, A')$: target policy)

- No Importance policy.

- target policy is a greedy policy
- both μ & π can improve
- ~~tar~~ π is made greedy wrt Q.

$$\pi(S_{t+1}) = \arg\max_{a'} Q(S_{t+1}, a')$$

- μ is ϵ-greedy wrt Q
-

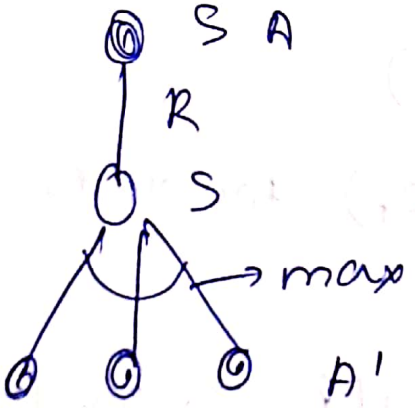
$$q_t = R_{t+1} + \gamma Q(S_{t+1}, A')$$

$$= R_{t+1} + \gamma Q(S_{t+1}, \arg\max_{a'} Q(S_{t+1}, a'))$$

$$= R_{t+1} + \max_{a'} \gamma Q(S_{t+1}, a')$$

↓

updating it bit on max Q value it can take



Value function Approx

Large scale RL

- Backgammon 10^{20} states
- go 10^{170} states
- helicopter continuous state space.

So far

- lookup table for $Q(s,a)$ or $V(s)$ for every s

Problem with large MDP

- too many states & actions to store in memory
- too slow to learn everything individually.

Solution

- Value fn approx

$$\hat{v}(s,w) \approx v_\pi(s)$$

$\downarrow$

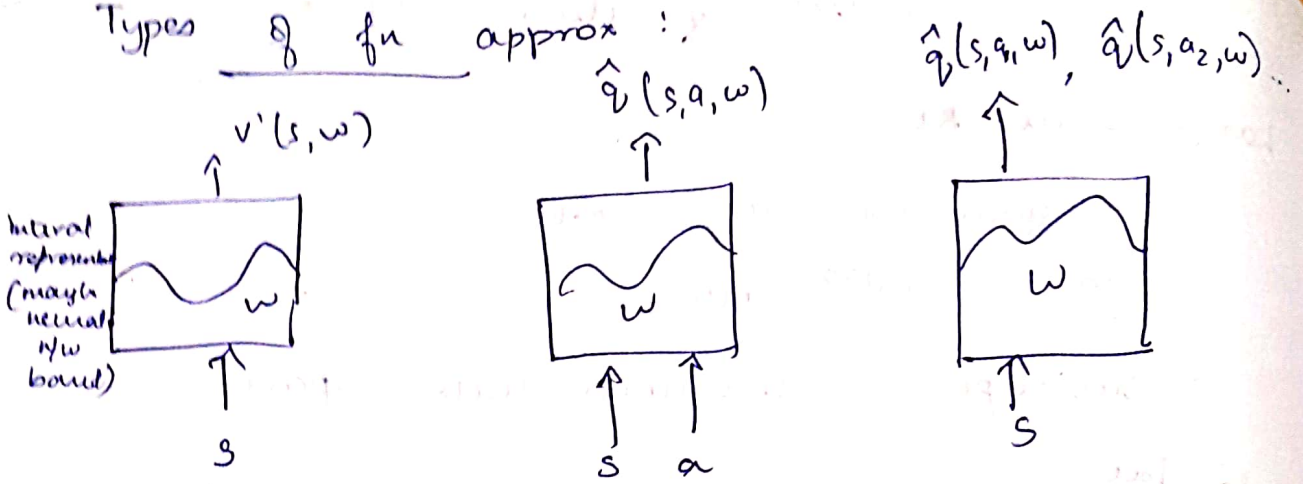
estimate of $v_\pi(s)$

$w \rightarrow$ weights

$$\hat{q}(s,a,w) \approx q_\pi(s,a)$$

- generalize seen states to unseen states
- update parameter w using TD or MC

Types of fn approx :



function approx:

- Linear combinations of features
- Neural n/w
- decision tree
- Nearest Neighbour
- Fourier / Wavelet bases

} differentiable

Incremental methods

gradient descent:

$J(w)$ diff fn param of w

$$\nabla_w J(w) = \begin{pmatrix} \frac{\partial J(w)}{\partial w_1} \\ \vdots \\ \frac{\partial J(w)}{\partial w_n} \end{pmatrix}$$

To find local minima of $J(w)$

Adjust param w in direction of -ve gradient

$$\Delta w = -\frac{1}{2} \alpha \nabla_w J(w)$$

Value fn approx using SGD

- find w minimising MSE b/w $\hat{v}$ & v_π

$$J(w) = E_\pi\left[(v_\pi(s) - \hat{v}(s,w))^2\right]$$

$$\therefore \Delta w = \alpha E_\pi\left[(v_\pi(s) - \hat{v}(s,w))\nabla_w \hat{v}(s,w)\right]$$

SGD samples the gradient

$$\Delta w = \alpha(v_\pi(s) - \hat{v}(s,w))\nabla_w \hat{v}(s,w)$$

Feature vector

Represent state by FV

$$S = \begin{pmatrix} x_1(s) \\ \vdots \\ x_n(s) \end{pmatrix}$$

x_j: distance from landmarks
: trends in stock market
: piece config in chess

Linear VFA

- linear combination of features

$$\hat{v}(s,w) = x(s)^T w = \sum_j^n x_j(s) w_j$$

$$J(w) = E_\pi\left[(v_\pi(s) - x(s)^T w)^2\right]$$

$$\nabla_w \hat{v}(s;w) = x(s)$$

$$\Delta w = \alpha(v_\pi(s) - \hat{v}(s,w))x(s)$$

update = (Step-size) × (error) × (feature value)

But there are no supervising, only rewards.

In practise, substitute a target for $V_\pi(s)$

For MC, target is G_t

$$\therefore \Delta w = \alpha(G_t - \hat{v}(s_t, w)) \nabla_w \hat{v}(s_t, w)$$

For TD, target is $R_{t+1} + \gamma \hat{v}(s_{t+1}, w)$

$$\Delta w = \alpha(R_{t+1} + \gamma \hat{v}(s_{t+1}, w) - \hat{v}(s_t, w)) \nabla_w \hat{v}(s_t,$$

For TD(λ), target is λ-return G_t^λ.

$$\Delta w = \alpha(G_t^\lambda - \hat{v}(s_t, w)) \nabla_w \hat{v}(s_t, w)$$

MC with VFA

- return is used to build some traing data

$(s_1, G_1), (s_2, G_2), \ldots (s_T, G_T)$

- Converges to global optimum

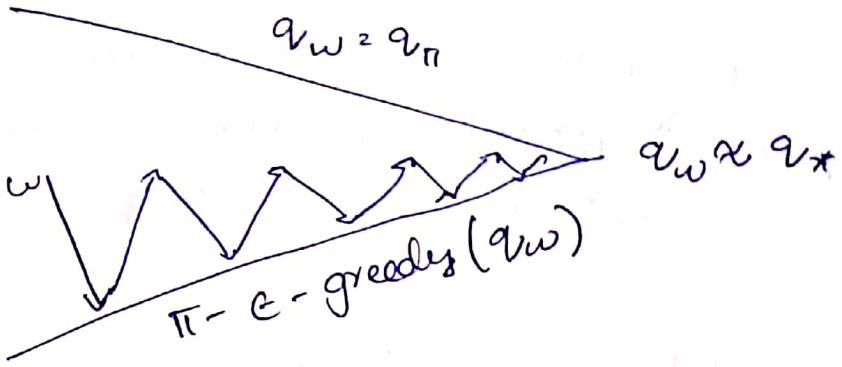
TD with VFA

target is a sample of true value $V_\pi(s_t)$

$(s_1, R_2 + \gamma \hat{v}(s_2, w)), (s_2, R_3 + \gamma \hat{v}(s_3, w)), \ldots (s_{T-1}, R_T)$

- Converges close to global optimum.

Control with VFA



Action value fn Approx

$$\hat{q}(s,A,w) = q_\pi(s,A)$$

$$J(w) = E_\pi\left[(q_\pi(s,A) - \hat{q}(s,A,w))^2\right]$$

$$x(s,A) = \begin{pmatrix} x_1(s,A) \\ \vdots \end{pmatrix}$$

Same as before for everything

Batch RL

- GD is not sample efficient
- Experience D consisting of (state, value) pairs

$$D = \{(s_1, v_1^\pi), (s_2, v_2^\pi) \ldots\}$$

Sample s,v from D

apply SGD

Exp Replay is solved using least square Prediction

DQN

- uses exp replay & fixed Q-targets
- for a while move towards q value in old n/w (keeping θ fixed) & then make θ⁻ n/w make equal to new n/w θ.

Policy gradient

directly parameterise policy instead of V_θ or Q_θ

Advantages of policy based or value based

- Better convergence
- Effective in high dim
- Can learn stochastic policies

Dis Ad

- Typically converge to lower than upper
- Evaluating a policy is typically inefficient & high variance

Policy obj functions

means of quality of policy π_θ?

- In episodic env, we start value

$J_1(\theta) = V^{\pi_\theta}(s_1) = E_{\pi_\theta}(v_1)$

- (continuing env, use avg value

$$J_{av}(\theta) = \sum_s d^{\pi\theta}(s) V^{\pi\theta}(s)$$

- avg reward per time step

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

To optimise these objectives:

- gradient free
  - Hill climbing
  - genetic algos
  - simplex / Amoeba / Nelder mead
- gradient based
  - GD
  - Conjugate Grad
  - Quasi-Newton.

policy grad

- $J(\theta)$ - obj fn
- gradient ascent to maximise local max of $J(\theta)$

$$\Delta\theta = \alpha \underbrace{\nabla_\theta J(\theta)}_{\text{policy grad}}$$

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

Computing grads by finite diff

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

- simple, noisy, inefficient
- works even if it is not diff.

Analytically compute grads

- score fn.
  - assume π_θ is diff when picking actions
  - we know $\nabla_\theta \pi_\theta(s,a)$
  - Likelihood ratio:

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

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

Score fn is $\nabla_\theta \log \pi_\theta(s,a)$

Softmax policy

$$\pi_\theta(s,a) \propto e^{\phi(s,a)^T \theta}$$

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

↳ feature we took

↳ avg of all features

gaussian policy:

(On cont. action space)

parametrise μ of the gaussian

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

Variance can be fixed or parametrised

$$a \sim N(\mu(s), \sigma^2)$$

Score fn =

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

for One-Step MDP

- start at state $s \sim d(s)$
- terminate after one step, Reward $r = R_{s,a}$

- Using likelihood ratios to compute the policy gradient

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

$$= \sum d(s) \sum \pi_\theta(s,a) R_{s,a}$$

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

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

for multi step:

- replace r with long term value $Q^\pi(s,a)$

$J = J_1$, J avg or $\frac{1}{1-\gamma} J_{avg}$

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

Monte-Carlo policy grad (Reinforce)

- update params by SGD
- use PG to sample exp.

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

for each episode $\{s_1, a_1, r_2, \ldots\} \sim \pi_\theta$ do
  for $t = 1$ to $T-1$ do
    $\theta = \theta + \Delta\theta_t$

Reducing Var using a Critic

- MC PG has high var
- use critic to estimate Q

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

- Action critic → 2 set of params
  Critic → for action value Q_w, param w
  actor → for policy, param θ, in direction suggested by critic

Approx policy grad combining both:

$$\nabla_\theta J(\theta) \approx E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) Q_w(s,a)]$$
$$\Delta\theta = \alpha \nabla_\theta \log \pi_\theta(s,a) Q_w(s,a)$$

To estimate Q_w:
- policy evaluation
- how good is π_θ under θ?
- MC or TD or TD(λ)

Bias in Actor-critic

- Approxi pg intros bias

Reducing Var using Baseline

- Subtract a Baseline fn B(s) from policy g.
- This can change var w/o changing exp.

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

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

$$= \sum d^{\pi_\theta} B(s) \nabla_\theta \sum \pi_\theta(s,a)$$

gradient becomes 0 (prob dist. add to 1)

$$= 0$$

we can use state value fn as B(s)

$$B(s) = V^{\pi_\theta}(s)$$

rewriting pg using the advantage fn $A^{\pi_\theta}(s,a)$

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

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

↓

depending on whether it is +ve or -ve, it changes the policy.

Estimating the advantage fn:

- Critic learns Q and V.

$$V_v(s) \approx V^{\pi_\theta}(s)$$

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

$$A(s,a) \approx Q_w(s,a) - V_v(s)$$

update both using TD learning

- Commonly used variant
  - TD error considered as a sample of the adv fn.

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

$$E_{\pi_\theta}[\delta^{\pi_\theta} | s,a] = A^{\pi_\theta}(s,a)$$

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

  - can use approx TD error

$$\delta_v = r + \gamma V_v(s') - V_v(s)$$

  - only 1 set of critic params need

Critics at diff Time scales.

For MC

$$\Delta\theta = \alpha(v_t - V_\theta(s))\,\phi(s)$$

For TD(0)

$$\Delta\theta = \alpha(r + \gamma V(s') - V_\theta(s))\,\phi(s)$$

For Forward TD(λ)

$$\Delta\theta \approx \alpha(v_t^\lambda - V_\theta(s))\,\phi(s)$$

For backward TD(λ)

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

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

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

Same for actor

Alternative Policy gradient directions

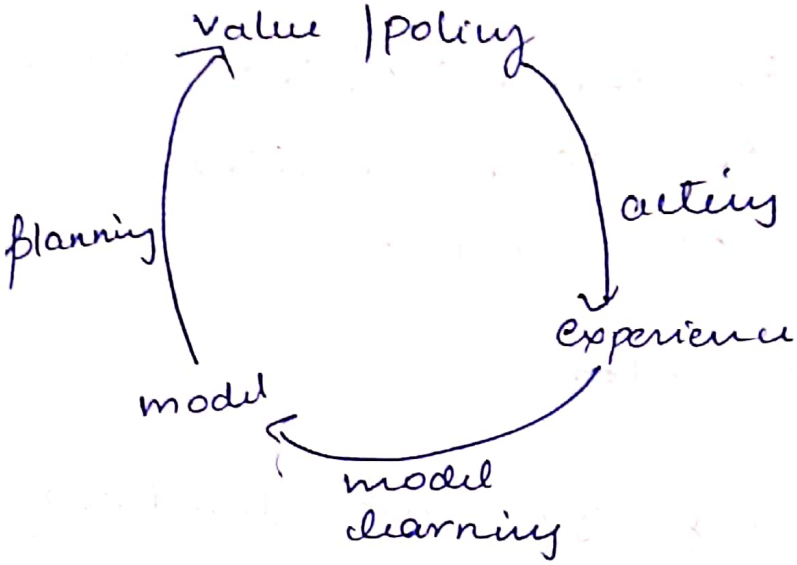
- Compactible for Approx.
  - features for critic are scores of policy.

Natural policy gradient

- parameter independent

Integrating Learning and Planning.

- learn model from experience and use planning to construct value fn or policy.
- model tells about rewards and transition probs.



Advantages:

- if learn value fn & policy directly is hard (Eg: playing Chess), model can provide more compact info
- Can learn using supervised learning
- reason about model uncertainty (choose to go areas where model is uncertain)

disadvantages:

- first learn model, then value fn (two sources of error)

Model

- representation of a MDP (S, A, P, R) parameterized by η
- assume some S and A are known
- $M_\eta = (P_\eta, R_\eta)$ → state transitions

$P_M \approx P$ $\quad R_M \approx R$

$$S_{t+1} \sim P_M(S_{t+1} | S_t, A_t)$$

$$R_{t+1} \sim R_M(R_{t+1} | S_t, A_t)$$

Typically assume S and R are indep.

$$P[S_{t+1}, R_{t+1} | S_t, A_t] = P[S_{t+1} | S_t, A_t] P[R_{t+1} | S_t, A_t]$$

goal: estimate M_η from exp. $\{S_1, A_1, R_2, \ldots S_T\}$

This is supervised

↳ took action A_1 from S_1

$S_1, A_1 \to R_2, S_2 \to$ new state

↳ Immediate reward

$S_2, A_2 \to R_3, S_3$

$\vdots$

$S_{T-1}, A_{T-1} \to R_T, S_T.$

$S, a \to r$ is a regression problem

$S, a \to S'$ is a density estimation problem.

Pick loss functions such as MSE, KLD etc.

Examples of models

- Table lookup
- Linear Expectation
- Linear gaussian
- gaussian process
- Deep belief Network

etc.

table look up model

- count visits to each action - state pairs
  → using this, build a probability distribution

or

- record experiments seen so far and randomly pick up tuples of state - action

Planning with a model

Solve MDP to find optimal value, policy to pick action

algos:

- Value iteration
- Policy iteration
- Tree search

etc

Sample based planning.

- Use the model only to generate samples
- Sample exper from model
  - Then apply model free RL to samples
    - MC control
    - sarsa
    - Q-learning
- often more efficient
- Solve for the imagined ~~eff~~ experiences.

Planning with an inaccurate model:

for an important model,

$$P_\eta, R_\eta \neq P, R$$

Model based RL is only good as the model that is learnt.

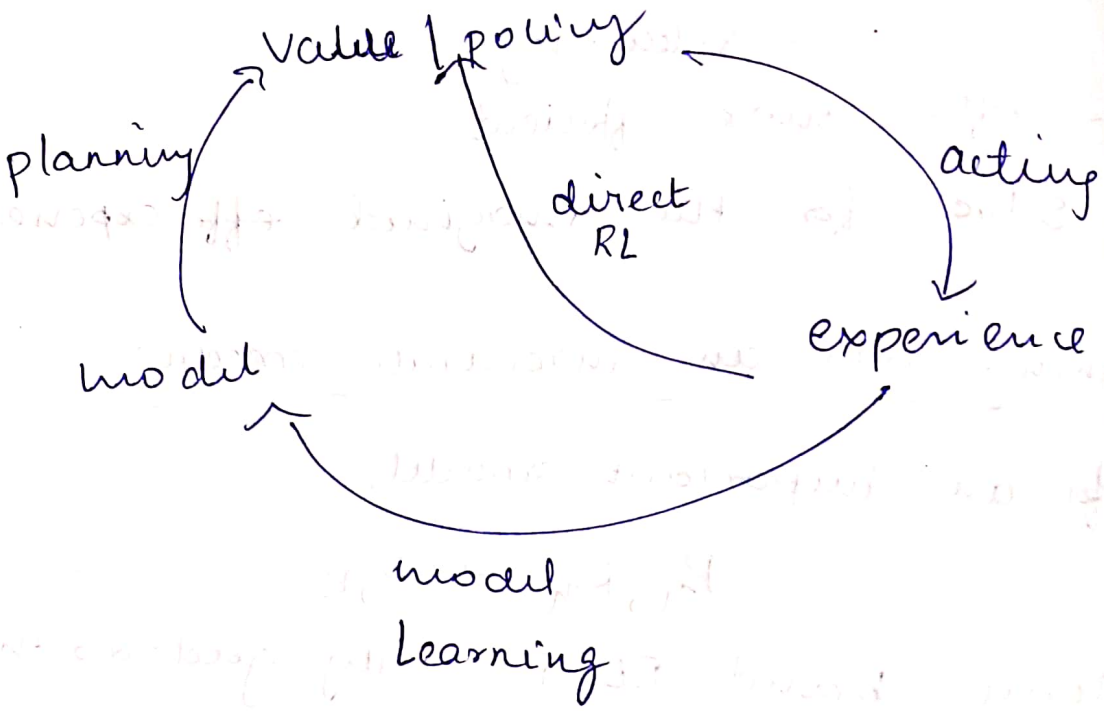
Solns:
- when model is wrong, use model-free
- reason explicitly about model uncertainity (bayesian approach)

Integrated architectures.

- two sources of experience
  - real exp, from sampled from envi (true MDP)
    $S' \sim P^a_{s,s'}$. $R = R_s^a$
  - Simulated exp, sampled from model (approx MDP)
    $S' \sim P_\eta(S'|S,A)$ $R = R_\eta(R|S,A)$

Dyna

- Learn a model from real experience.
- learn and plan value fn from real and simulated exp.



Dyna - Q

Initialize $Q(s,a)$ and $model(s,a)$

i) $S \leftarrow$ current state

ii) $A \leftarrow \epsilon$-greedy(S,Q)

iii) Execute A, observe R, state S' (see where we end up)

iv) $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
[normal Q learning]

v) $Model(S,A) \leftarrow R, S'$ update model using supervised learning on R and S'

vi) Repeat n times:
- $S \leftarrow$ random seen state and a action

- $R, S' \leftarrow$ model (S, A) Imagine the transition
- Q learning update, update Q value from where started to best Q value we ended up (S')

Simulation Based Search.

Forward search:

- this algorithms don't explore every state space
- They select the best action by lookahead.
- They build a search tree with current state S_t at the root
- Using a ~~mdp~~ model of MDP to lookahead
- No need to solve whole MDP, only starting from now.

Simulation-based Search.

- Forward search using samples
- Apply Model free RL to samples

$$\{S_t^k, A_t^k, R_{t+1}^k, \cdots, S_T^k\}_{k=1}^K \sim M_v$$

k samples.

Then,

MCC -> MC search

Simple Monte Carlo Search

- Model M_v & simulation policy π
- for each action $a \in A$
  - Simulate k episodes from state s_t
  - Evaluate actions by MC evaluation
- Select real action with max value.

Monte Carlo Tree Search (SOTA 2016) used in GO

- Model M_v
- Simulate k episodes from S_t using a current simulation policy π
- Build a search tree based on all state action pairs using MC evaluation
- After search is complete, pick actions with highest q-value in the root.
- After every simulation, π improves
- Each simulation can pick
  - Tree policy: pick actions to maximise $Q(S,A)$
  - Default policy: pick actions randomly for S,A not in tree.
- Repeat this
  - Improve tree policy with ϵ-greedy etc.
- Converges on optimal search tree

Go

position evaluation

$R_t = 0$ for all non-terminal states

$$R_T = \begin{cases} 0 & \text{lose} \\ 1 & \text{win} \end{cases}$$

policy $\pi = \{\pi_B, \pi_W\}$ for both players.

$$V_\pi(s) = E_\pi[R_T \mid S = s] = P[\text{wins} \mid S = s]$$

$$V_*(s) = \max_{\pi_B} \min_{\pi_W} V_\pi(s) \rightarrow \text{tree based.}$$

Advantage of MCTS

- highly selective best-first search
- Evaluates states dynamically.
- Uses samples to break C of Dim.
- parallelizable

TD-Search

- TD search applies Sarsa to Sub-MDPs
- For each step in Sim, update A, V by Sarsa.
- Select actions ε-greedy.
- May also use for approx for Q.

Dyna-2

- maintains 2 value fns
  - like a long term memory from real experience.
  - and a short term memory as a search tree.

Exploration and Exploitation

Exploitation: Make decision based on current info.

exploration: gather more information by doing something else.

Types of exploration

- Random exploration
  - explore random action eg: ϵ-greedy.
- Optimisim in the face of uncertainty
  - estimate uncertain in value
  - Prefn to explore states/actions with highest uncertainty.
- Information state space
  - consider agent's info as part of its state
  - lookahead to see how info helps reward.
    (computationally hard)

~~How~~ We can explore in

- State space ~~&~~ action space
- parameter space
  (parametrix policy π)
  (adv: consistent explorations)
  (disadv: doesn't know all state/action)

Multi-Armed Bandit

- simplification of MDP.
- MAB is a tuple of $\{A, R\}$
- A is a set of Actions or arms
- $R^a(r) = P[R = r \mid A = a]$ is an unknown prob distribution over rewards.
- At every t, select one action, env generates a reward.
- maximise cumulative reward over time

- ① regret: opportunity loss for one step
- ② action-value: mean reward for action a
- ③ optimal-value: best action to choose.

② $q(a) = E[R \mid A = a]$

③ $V_* = q(a^*) = \max_{a \in A} q(a)$

① $l_t = E[V_* - q(A_t)]$

↓ optimal ↳ payoff of picked action

- total regret

$$L_t = E\left[\sum_{1}^{t} V_* - q(A_t)\right]$$

- Maximin . Cumulative reward = minimin total regret

- Count $N_t(a)$ → expected no of selection for a

- gap Δa is $V_* - q(a)$

$$\therefore L_t = E\left[\sum_{1}^{t} V_* - q(A_T)\right]$$

$$= \sum_{a \in A} E[N_t(a)](V_* - q(a))$$

$$= \sum_{a \in A} E[N_t(a)]\Delta a$$

∴ Regret is a function of counts and gaps

- Every time gap is large, make sure count is low. If gap is low, let count be more.

- problem: gap are not known, because we don't know V_*

greedy algo with this

- greedy algo can pick sub-optimal action forever.
- It has linear total regret

Algorithms with Optimistic Initialisation

- really good idea
- initialise values to max possible $Q(a) = r_{max}$
- Then act greedily

$$A_t = \underset{a \in A}{\operatorname{argmax}}\, Q_t(a)$$

- Encourages exploration of things not known.
- linear total regret.
- some unlucky samples can lock out some states forever.

ϵ-greedy

- explores forever
- Constant ϵ ensures min regret
- linear total regret.

decaying ϵ-greedy

- decay ϵ.
- logarithmic regret.
- this needs advance knowledge of gaps.

- Total regret is at least log in steps

optimism in face of uncertainty

- pick more uncertain action

UCB1

$$A_t = \underset{a \in A}{\operatorname{argmax}} \left[Q_t(a) + \sqrt{\frac{2 \log t}{N_t(a)}} \right]$$

- $Q_t(a)$ → MC estimate of Q value
- t → total time steps
- $N_t(a)$ → No of times this action selected

This achieves log total regret

Bayesian Bandits

- exploit prior dists on R.
- Use posterior to guide exploration

Probability matching

- Selects action according to prob that it is the best one.
- Can be diff to compute $\pi(a)$ analytically from posterior.

Thompson sampling

- Sample based probability matching

$$\pi(a) = E\left[\mathbb{1}(Q(a) = \max Q(a')) \mid R_1, \ldots, R_{t-1} \right]$$

- Every step, pick sample from posterior
- Then pick the one which is best.

Then algo keep going forever (can't impose budget)

Info-state space

- Info gain is high in uncertain situations
- do if we know value of info, we can trade off exploration & exploitation optimally.
- Cast bandit problem as MDP instead of one-step
- At each state, there is an info state $\tilde{s}$ summarising all info seen.
- An action A causes a transition to a new info state $\tilde{s}'$ with prob $\tilde{P}^{A}_{\tilde{s}\tilde{s}}$
- So there is now an MDP $\tilde{M}$ in the augmented info state space

$$\tilde{M} = \{\tilde{S}, A, \tilde{P}, R, \gamma\}$$

- Consider Bernoulli Bandit (like coin flip).
- $R^a = B(\mu_a)$ $\mu_a \rightarrow$ prob
- find highest μ_a.
- Info state $\tilde{s} = \{\alpha, \beta\}$

  It counts
  - α counts where reward = 0
  - β counts when reward = 1

- To solve this,

- Start with a Beta(α_a, β_a) prior over p_a
- update posterior for new action
  Beta($\alpha_a + 1, \beta_a$) for r_0
  Beta($\alpha_a, \beta_a + 1$) for r_1
- this is Bayes adaptive MDP

Contextual Bandits

- add states.
- $S = P(s)$, prob distn over states.

Other approaches

- reconstruct model
- for unknown states, give r_{max}.
- Be very optimistic towards uncertainty
- Solve MDP using any algo.

game-playing

Best response is other player(s) become part of environment. This is just an MDP now.

Is a nash equilibrium among all players

NE is when every player has found best response to everyone else.

Perfect info games

- all players can access info of others

Imperfect info games

eg: poker

Value fn:

policy $\pi = \{\pi^1, \pi^2\}$

$$V(s) = E_\pi [G_t | S_t = s]$$

A minimax fn maximises ow's expected return & app minimises opponent's expected return