Elements common to all control tasks

$S_t$ - state, $A_t$ - Action $R_t$ - Reward, Agent

Environment,

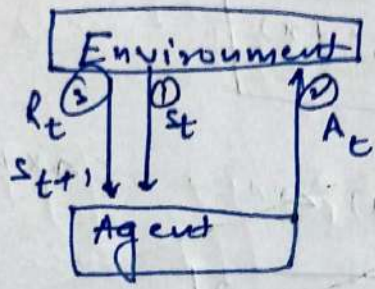
Markov decision process MDP

Discrete time stochastic control process

Time moves toward infinite interval $t \in [1, 2, 3, -]$

future states depends only partially on action taken

It is based on decision making to reach the target state




$(S, A, R, P)$

Set of possible state

Probability of passing from one state to another when taking each possible action

set of actions that can be taken in each of the states

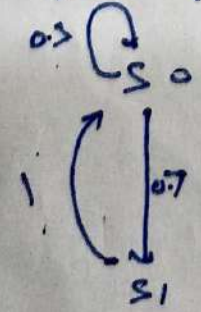
set of Rewards for each $(s, a)$ pair

- The process has no memory

$$P[S_{t+1} | S_t = s_t] = P[S_{t+1} | S_t, S_{t-1} = s_{t-1} \dots S_0 = s_0]$$

- The next state depends only on the current state not on the previous state

- If a process meets this property, it is known as Markov process




Different Action reward

Types of Markov decision process

finite (game) | infinite (car driving)

Terminating: Episodic at certain condition like chess | continuing

Trajectory and episode → start to target

Trajectory :- Elements that are generated when the agent moves from one state to another

$$\tau = s_0, a_0, R_1, s_1, a_1, R_2, s_2, a_2, R_3, s_3 \ldots$$

Episode: Trajectory from the initial state of the task to a terminal state

$$\tau = s_0, a_0, R_1, s_1, a_1, R_2, s_2, \ldots R_T, s_T$$

Reward Vs Return

The goals of the task are represented by the rewards ($R_t$)

We want to maximize the sum of rewards

A short term reward can worsen long-term result

Reward

$R_t$

Returns

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \ldots + R_T$$

Discount factor ($\gamma$) The agent has no incentive to get to the goal through the shortest route

$$G_0 = R_1 + R_2 + R_3 + R_4 + \ldots + R_T$$

We will multiply future rewards by a discount factor $\gamma$

$$G_0 = R_1 + \gamma R_2 + \gamma^2 R_3 + \ldots + \gamma^{T-t-1} R_T$$

$$\gamma \in [0, 1]$$

higher the exponent more value of reward is reduced

If $\gamma = 0$, all future rewards will be 0

If $\gamma = 1$, rewards will not be discounted

- we want to maximize the long term discounted sum

**Policy** $\pi(s)$ function that decides what action to taken in a particular state

$$\pi : S \mapsto A$$

Probability of taking action a in state s

$\pi(a|s)$

Action a taken in state s

$\pi(s)$

stochastic

$\pi(s) = [p(a_1), p(a_2), \ldots p(a_n)]$

$\pi(s) = [0.3, 0.2, 0.5]$

Deterministic

$\pi(s) \rightarrow a$

$\pi(s) = a_1$

we want to maximize the sum of discounted rewards

we must find the optimal policy $\pi_*$

**state value and action value**

**state value** $V_\pi(s) = \mathbb{E}[G_t | s_t = s]$

$V_\pi(s) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \ldots \gamma^{T-t-1} R_t | s_t = s]$

following policy $\pi$

q - value of a state-action pair

$q_\pi(s,a) = \mathbb{E}[G_t | s_t = s, A_t = a]$

$q_\pi(s,a) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-t-1} R_t | s_t = s, A_t = a]$

following policy $\pi$

Bellman equation $v(s)$ :- searching optimal policy

$$v_\pi(s) = \mathbb{E}[G_t | s_t = s]$$

$$= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-t-1} R_t | s_t = s]$$

$$= \mathbb{E}[R_{t+1} + \gamma G_{t+1} | s_t = s]$$

$$= \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')]$$

following policy $\pi$

Bellman equation for $q(s,a)$

$$q_\pi(s,a) = \mathbb{E}[G_t | s_t = s, A_t = a]$$

$$= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T | s_t = s, A_t = a]$$

$$= \mathbb{E}[R_{t+1} + \gamma G_{t+1} | s_t = s, A_t = a]$$

$$= \sum_{s',r} p(s',r|s,a)\left[r + \gamma \sum_{a'} \pi(a'|s') q_\pi(s',a')\right]$$

following policy $\pi$

## Solving a Markov decision process

The optimal value of a state is the expected return following the optimal policy

$$v_*(s) = \mathbb{E}_\pi[G_t | s_t = s]$$

$$q_*(s,a) = \mathbb{E}_\pi[G_t | s_t = s, A_t = a]$$

The optimal policy $\pi_*$ policy. it's the one that chooses action that maximize $v(s)$, or $q(s,a)$

$$\pi_*(s) = \arg\max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_*(s)]$$

$$\pi_*(s) = \arg\max_a q_*(s,a)$$

To find the optimal policy $\pi_*$, we must know the optimal value

To find the optimal $v_*$ or $q_*$ value we must know the optimal policy

Bellman optimality equations

$$V_*(s) = \max_a \sum_{s',r} p(s',r|s,a)\left[r + \gamma V_*(s')\right]$$

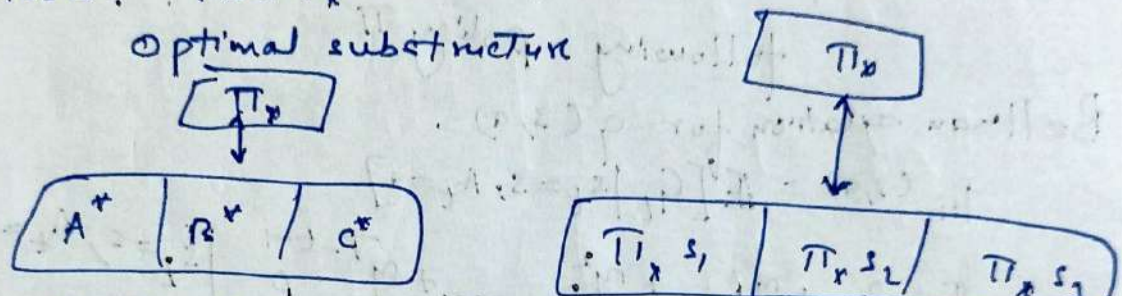
$$q_*(s,a) = \sum_{s',r} p(s',r|s,a)\left[r + \gamma \max_{a'} q_*(s',a')\right]$$

following the optimal policy $\pi_*$

Dynamic programming Method that find the solution to a problem by breaking it down into smaller, easier problems

Our task: Find $\pi_*$

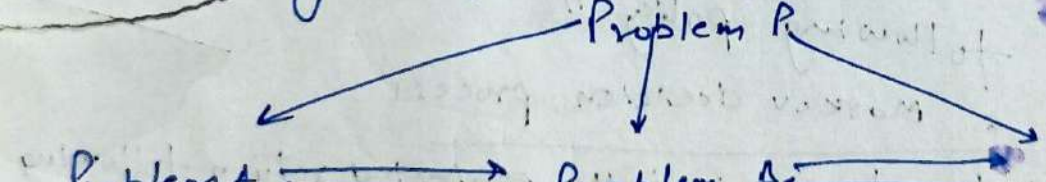
optimal substructure




The optimal solution to all subproblems produce the optimal solution to the original problem

$\pi_* \Longleftrightarrow \pi_*(s)$ $\quad q_* \Longleftrightarrow q_*(s,a)$ $\quad V \Longleftrightarrow V_*(s)$

Overlapping subproblems




The solutions to the subproblems are mutually dependent

$$V_*(s) = \max_a \sum_{s',r} p(s',r|s,a)\left[r + \gamma V_*(s')\right]$$

$$q_*(s,a) = \sum_{s',r} p(s',r|s,a)\left[r + \gamma \max_{a'} q_*(s',a')\right]$$

DP turns these eq. into update rules

$$V_{k+1}(s) = \max_a \sum_{s',r} p(s',r|s,a)\left[r + \gamma V_k(s')\right]$$

$$V(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a)\left[r + \gamma V(s')\right]$$

Model of the environment

$$V_{k+1} \leftarrow \max_a \sum_{s',r} p(s',r|s,a)\left[r + \gamma V(s')\right]$$

We need to know in advance how the state changes and what rewards we get from performing each action in each state

DP: solves problems using expected values, not trial and error

Value iteration: We find the optimal policy $\pi_*$:

$$\pi_*(s) = \arg\max_a \sum_{s',r} p(s',r|s,a)\left[r + \gamma v_*(s')\right]$$

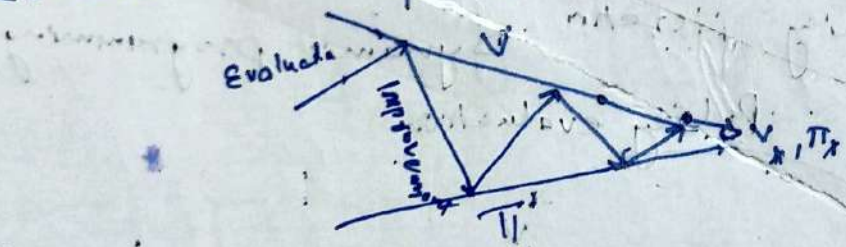
... but for that you have to find $v_*$...

update rule

$$V(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a)\left[r + \gamma V(s')\right]$$

Moves towards the value under the optimal policy

Policy iteration: process that alternately improves the estimated value & policy




Policy evaluation

Iterative policy evaluation

Iterative approximates the value of a given policy $v_\pi$

$$v_{\pi_i}(s) = \sum_a \pi(s|a) \sum_{s',r} p(s',r|s,a)\left[r + \gamma v_{\pi_i}(s')\right]$$

$$V(s) \leftarrow \sum_a \pi(s|a) \sum_{s',r} p(s',r|s,a)\left[r + \gamma V(s')\right]$$

Each iteration gets closer to $v_\pi$

$$v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_\pi$$

Policy improvement: Does the policy improve if we change the first action?

$$q_\pi(s,a) = \sum_{s',r} p(s',r|s,a)\left[r + \gamma v_\pi(s')\right]$$

if $q_\pi(s, \pi'(s)) \geq v_\pi(s)$ then

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

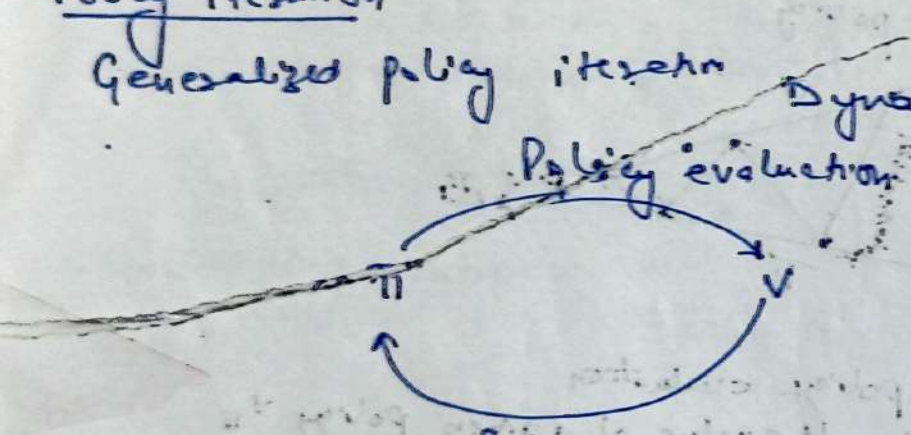
$\pi$ and $\pi'$ differ only in the action a they take in state $s$.

New policy! $\pi'(s) = \arg\max_a \sum_{s',r} p(s',r|s,a)\left[r + \gamma v_\pi(s')\right]$

Policy-stable ← true
for $s \in S$ do
  old action ← $\pi(s)$
  $\pi(s) \leftarrow \arg\max_{a \in A(s)} \sum_{s',r} p(s',r|s,a)\left[r + \gamma V(s')\right]$
  if old action $\neq \pi(s)$ then
    policy-stable ← false
  end if
end for

Policy iteration

Generalized policy iteration




Policy iteration results in the following iterative process

then try to replicate the results of dynamic programming!

— More efficiently
— Without a model

Disadvantage of dynamic programming

High computation cost
complexity grows very rapidly with the no of states
In each sweep we update all the states

Monte carlo method family of methods that learn optimal $V_*(s)$ or $q_*(s,a)$ value based on samples

The agent will use a policy $\pi$ to tackle the task for an entire episode

$$S_0, A_0, R_1, S_1, A_1, \ldots S_T, A_{T-1}, R_T$$

They approximate the values by interacting with the environment to generate sample returns and averaging them

$$V_\pi(s) = \mathbb{E}_\pi [G_t | S_t = s] \qquad G_t = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}$$

$$V_\pi(s) = \frac{1}{N} \sum_{k=1}^{N} G_{sk}$$

$$q_\pi(s,a) = \mathbb{E}_\pi [G_t | S_t = s, A_t = a] \qquad G_t = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}$$

$$Q_\pi(s,a) = \frac{1}{N} \sum_{k=1}^{N} G_{s,a_k}$$

Laws of large numbers

In the limit, this succession of return samples $G_{s_1}, G_{s_2} \ldots G_{s_N}$ converges to its expected value $V_\pi(s)$

$$P\left(\lim_{n \to \infty} \bar{G}_s = V_\pi(s)\right) = 1$$

Advantages of Monte carlo Method

Dynamic programming

$$V_*(s) = \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_*(s')]$$

- The estimate of a state value does not depend on the rest
- The cost of estimating a state value is independent of the total no. of states
- we can focus the estimations on the states that help us solve the task

$S_0, A_0, R_1, S_1, A_1, R_2 \dots, R_T$

from the generated trajectory, we will calculate the returns for each moment of time t

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \gamma^{T-t-1} R_T$$

$$G_{t+1} = R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \dots \gamma^{T-t-1} R_T$$

Our ~~old~~ strategy Use these returns to update the value function and through it, the policy $\pi$

To improve the policy, we can use $v(s)$ argmax

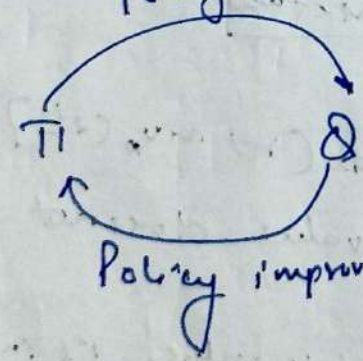
$$\pi'(s) = \arg\max_a \sum_{s',r} p(s', r | s, a) [r + \gamma v_\pi(s')]$$

$v(s)$ requires knowing the effects of taking each action beforehand

~~To~~ The environment dynamics are implicit in $q_\pi(s,a)$

$$q_\pi(s,a) = \sum_{s',r} p(s', r | s, a) [r + \gamma v_\pi(s')]$$

Instead of $V(s)$ we keep a table with $Q(s,a)$




$\pi_0 \to Q_{\pi_0} \to \pi_1 \to Q_{\pi_1} \dots$

$\dots Q_{\pi_*} \to \pi_*$

The importance of exploration.

$Q(s,a)$ is a estimate $\pi'(s) = \arg\max_a Q(s,a)$

The estimate will improve as we obtain new samples, but they might not be perfect

If action a is optimal but has a bad estimate $Q(s,a)$ we'll never pick it

The only way to avoid this is to explore all actions every once in a while and update their estimate $Q(s,a)$

## How can we ~~update~~ maintain the exploration?

Two options

exploring ~~state~~ starts

$s_0 \sim S,\ A_0 \sim A(s_0)$

stochastic policies

$\pi(a|s) > 0,\ \forall a \in A(s)$

stochastic policies

On-policy learning

Generates samples using the same policy $\pi$ that we're going to optimize

off-policy learning

Generates samples with an exploratory policy b different from the one we're going to optimize

## On-policy Monte Carlo

$\epsilon$-greedy policy

with probability $\epsilon$ select a random action

with probability $1-\epsilon$ selected the action with highest $Q(s,a)$

$$\pi(a|s) = \begin{cases} 1-\epsilon+\epsilon_r & a = a^* \\ \epsilon_r & a \neq a^* \end{cases} \qquad \epsilon_r = \frac{\epsilon}{|A|}$$

$|A| = 4,\ \epsilon = 0.2$

$$\pi(a|s) = \begin{cases} 1 - 0.2 + 0.05 = 0.85 & a = a^* \\ 0.05 & a \neq a^* \end{cases}$$

$$\epsilon_r = \frac{0.2}{4} = 0.05$$

# off policy Monte Carlo

off-policy strategy

Exploratory policy
$b(a|s)$

Target policy
$\pi(a|s)$

off-policy strategy
$b(a|s)$

Generates the episode we are going to update $q(s,a)$ with $S_0, A_0, R_1, S_1, A_1, \ldots, R_T$

$\pi(a|s)$

Policy to be optimized through $q(s,a)$ values!

$$\pi(s) \leftarrow \arg\max_a q(s,a)$$

The policy $b$ has to be able to explore all the actions that $\pi$ it contains!

If $\pi(a|s) > 0$, then $b(a|s) > 0$.

The ~~average~~ returns will not approximate the value under $\pi$ but under $b$!

$$\mathbb{E}_b[G_t | S_t = s, A_t = a] = q_b(s,a)$$

Importance sampling:- statistical technique for estimating the expected values of a distribution by working with samples from another distribution

$$w_t = \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}$$

By correcting the returns using importance sampling we will approximate the value under $\pi$!

$$\mathbb{E}[w_t G_t | S_t = s] = V_\pi(s)$$

Update Rule

for each $q(s,a)$, we'll keep a list of observed returns $[G_1, G_2, G_3, \ldots G_N]$

Each time we need to update $Q(s,a)$ we'll recompute the average

$$Q(s,a) \leftarrow \frac{1}{N} \sum_{k=1}^{N} G_k$$

update rule for $Q(s,a)$

$$Q(s,a) \leftarrow Q(s,a) + \frac{w_t}{C(s,a)} [G_t - Q(s,a)]$$

where $C(s,a) = \sum_{k=1}^{n} w_k$

## Temporal difference methods

- Family of methods that learn the optimal $V_*(s)$ or $q_*(s,a)$ values based on experience
- combination of Monte carlo methods and dynamic programming

The agent learns from example

$S_0, A_0, R_1, S_1, A_1, \ldots, R_T$

They do not need a model of the environment

$$\pi'(s) = \arg\max_a \sum_{s',r} p(s',r|s,a)[r + \gamma V(s')]$$

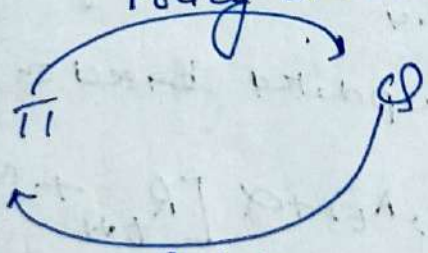
$$\pi'(s) = \arg\max_a Q(s,a)$$

They use bootstrapping!

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

update rule for the SARSA algorithm

Generalize Policy iteration




Monte carlo methods wait until the return $G_t$ is available before updating $Q(s,a)$.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \gamma^{T-t-1} R_T$$

we update $Q$ and $\pi$ every time the agent takes an action.

Temporal difference methods

we keep a table with q-value estimates for each $S_t, A_t$ pair:

$$Q(S_t, A_t)$$

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a]$$

$$q_\pi(s,a) = \sum_{r,s'} p(r,s' \mid s,a)\left[r + \gamma v_\pi(s')\right]$$

$$q_\pi(s,a) = \sum_{r,s'} p(r,s' \mid s,a)\left[r + \gamma \sum_{a'} \pi(a' \mid s') q_\pi(s',a')\right]$$

$$q_\pi(s,a) = \sum_{r,s'} p(r,s' \mid s,a)\left[r + \gamma \sum_{a'} \pi(a' \mid s') q_\pi(s',a')\right]$$

$R_{t+1}$ $S_{t+1}$ $A_{t+1}$ $Q$

Compare the two estimates:

$$\boxed{R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)}$$

Temporal difference error:

Estimates are updated based on temporal difference error:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right]$$

update rule for the SARSA methods

Similar to constant $\alpha$ Monte carlo!

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [G_t - Q(S_t, A_t)]$$

Estimating!

$$\hat{G}_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$$

Allow us to update $Q(S_t, A_t)$ at time $t+1$.

$$Q(S_t, A_t) \leftarrow (1-\alpha)(S_t, A_t) + \alpha [R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})]$$

The update moves $Q(S_t, A_t)$ $\alpha$ percent in the direction of $R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$

Monte carlo vs temporal differnce method

SARSA on-policy temporal difference learning

$\epsilon$-greedy policy

Probability $\epsilon$ : $a \sim A(s)$ random

Probability $1-\epsilon$ : $a = \underset{a}{\text{argmax}}\, Q(s,a)$

The name sarsa comes from the five values involved in the update rule!

$$S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

The $\epsilon$-greedy policy $\pi$ picks the next action, used to update $Q(S_t, A_t)$

Q learning off-policy temporal difference learning.

exploratory policy $b(s)$

Target policy

$$\pi(s) = \underset{a}{\text{argmax}}\, Q(s,a)$$

update rule!

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma Q(S_{t+1}, \pi(S_{t+1})) - Q(S_t, A_t)]$$

update $Q(S_t, A_t)$ biased on best available action.

$$A_{t+1} = \pi(S_{t+1}) = \text{argmax}_a Q(S_{t+1}, a)$$

Advantages Vs Montecarlo

They allow us to update $Q(s,a)$ while experience is being collected

In practice they converge faster

Advantages Vs dynamic programming

More efficient, focus the effort on the states that lead to goals

Don't require a model of the environment.

n step temporal difference method

Family of algorithms that learn based on experience using the n-step bootstraping technique

update rule of the SARSA algorithm:

$$Q(S_t, A_t) \rightarrow Q(S_t, A_t) + \alpha [\underbrace{R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})}_{\text{update target}} - Q(S_t, A_t)]$$

Bootstrapping

$$R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$$

1 step bootstrapping

After performing an action, we replace the remaining rewards by the $Q(S_{t+1}, A_{t+1})$ estimate

2-step bootstrapping:-

$$R_{t+1} + \gamma R_{t+2} + \gamma^2 Q(S_{t+2}, A_{t+2})$$

3-step bootstrapping

$$R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 Q(S_{t+3}, A_{t+3})$$

n step bootstrapping

$$R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n Q(S_{t+n}, A_{t+n})$$

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n Q(S_{t+n}, A_{t+n})$$

n-step return estimate

$$Q(S_t, A_t) \rightarrow Q(S_t, A_t) + \alpha [G_{t:t+n} - Q(S_t, A_t)]$$

Since we need to observe n rewards, we need to wait until time $t+n$ to update $Q(S_t, A_t)$.

SARSA

$$G_{t:t+1} = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$$

$$Q(S_t, A_t) \rightarrow Q(S_t, A_t) + \alpha [G_{t:t+1} - Q(S_t, A_t)]$$

$$G_{t:t+2} = R_{t+1} + \gamma R_{t+2} + \gamma^2 Q(S_{t+2}, A_{t+2})$$

$$G_{t:t+3} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 Q(S_{t+3}, A_{t+3})$$

....

If $n \geq T$:

$$G_{t:t+n} = G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n}$$

$$Q(S_t, A_t) \rightarrow Q(S_t, A_t) + \alpha [G_{t:t+n} - Q(S_t, A_t)]$$

Monte carlo

$G_{t:t+n} = G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n}$

$Q(S_t, A_t) \longrightarrow Q(S_t, A_t) + \alpha [G - Q(S_t, A_t)]$

How does n affect the learning process?

$G_{t:t+1} = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$ . $G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n} R_{t+n}$

$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n} Q(S_{t+n}, A_{t+n})$

How does the choice of n affect the learning process?

Bias: $Q(S_{t+1}, A_{t+1})$ is an estimate of future rewards. The estimate improve throughout the learning process

$R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$

$Q(S_{t+1}, A_{t+1})$ introduces bias it's the estimate

$R_{t+1} + \gamma R_{t+2} + \gamma^2 Q(S_{t+2}, A_{t+2})$

$R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 Q(S_{t+3}, A_{t+3})$

$R_{t+1} + \gamma R_{t+2} + \gamma^{n-1} R_{t+n} + \gamma^n Q(S_{t+n}, A_{t+n})$

the higher n, the more heavily discounted the estimate $Q(S_{t+n}, A_{t+n})$ by $\gamma^n$.

The higher the n, the lower the bias

Variance: Each reward is a random variable that depends on the state s and action a preceding it

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n}$$

Even if they have the same expected value $G_t$ and -es will be very different from each other

- The higher the n, the higher the variance
- The higher n is, the lower the bias but the higher the variance

n-step SARSA

Combining SARSA with n-step bootstrapping

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^n Q(S_{t+1}, A_{t+n})$$

update rule:

$$Q(S_t, A_t) \longrightarrow Q(S_t, A_t) + \alpha \left[ G_{t:t+n} - Q(S_t, A_t) \right]$$

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^n Q(S_{t+1}, A_{t+n})$$

On-policy learning strategy with $\epsilon$-greedy policy

Probability $\epsilon$

$a \sim A(s)$ randomly

Probability $1-\epsilon$

$a = \arg\max_a Q(s, a)$

## n- step SARSA in action

**Continuous state spaces** $S = (0, 0)$

Solutions:

- Transforming the states (section 7)
- Using other algorithms (Section 8-11)

Transforming the states:
Convert the continuous state space into a discrete state space

$$R \rightarrow \{S_0, S_1, S_2, \ldots S_N\}$$

- state aggregation
- Tile coding

**state aggregation**

- We went from having infinite states to having only 10

## State aggregation

2 D state aggregation, 1D state aggregation

— Aggregate the state and generate a grid to having a single state

**Tile coding** Average of the estimate

Creation of each tiling

step 1: Divide each state dimension in segments.

step 2: Expand or shrink the value range

step 3: Displace the segments.

step 4: Repeat steps 1 to 3 n times.

## function approximators

continuous state space

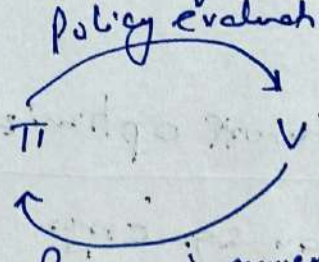
Disadvantages of state aggregation and tiling coding
- → Limited precision.
- → complexity $O(n^k)$.

— $n$ = segments, $k$ = dimensions

we need precise alternative of limited complexity

**function approximators** (state value function)

How do we observe the value function?




the agent learns based on experience

The function $V_*(s)$ and $q_*(s,a)$ are not known in advance

$$f_1(s/w) \rightarrow f_2(s/w) \rightarrow \cdots \rightarrow f_n(s/w) \simeq v_*(s)$$

Two examples

- state

$s = [s_1, s_2, s_3 \ldots s_n]$

Parameter vector

$w = [w_1, w_2, w_3 \ldots w_n]$

Linear approximation

$f(s/w) = \hat{v}(s/w) = w \cdot s^T$

$f(s) = \hat{v}(s)$

$= w_1 s_1 + w_2 \cdot s_2 + w_3 \cdot s_3 + \cdots + w_4 \cdot s_4$

Inputs

$\phi(s) = [s_1, s_1^2, \ldots, s_1^k, \ldots, s_1^k, \ldots, s_1^i, \ldots, s_n]$

Parameter vector:

$w = [w_1, w_2, w_3, \ldots w_n]$

Polynomial approximator:

$f(s|w) = \hat{v}(s|w)$

$= w \cdot \phi(s)^T$

$f(s) = \hat{v}(s) = w_1 s_1 + w_2 \cdot s_1^2 + w_3 \cdot s_1^3 \ldots$

Advantages of using function approximators:

Efficiency $w = [w_1, w_2, w_3, \ldots, w_n]$

They only require memory to save their parameters

Flexibility, Precision

Artificial neural network

They serve multiple purpose including function approximation.

Deep sarsa

SARSA + Neural network

$S = [s_1, s_2]$

$\hat{q} = [\hat{q}(s, a_1), \hat{q}(s, a_2), \ldots]$

Neural network optimization

Mean sq. error

$$L(\theta) = \frac{1}{N} \sum_{i=1}^{N} [y_i - \hat{y}_i]^2$$

we want to minimize the square of the errors of the neural network estimates

$$L(\theta) = \frac{1}{|k|} \sum_{i=1}^{|k|} [R_i + \gamma \hat{q}(s_i', A_i' | \theta_{targ}) - \hat{q}(s_i, A_i | \theta)]^2$$

Target: $y_i = R + \gamma \hat{q}(s_i', A_i' | \theta_{targ})$ — value towards which we want to push the estimate

Estimate: $\hat{y} = \hat{q}(s_i, A_i | \theta)$ — estimate of the q value of a state-action pair

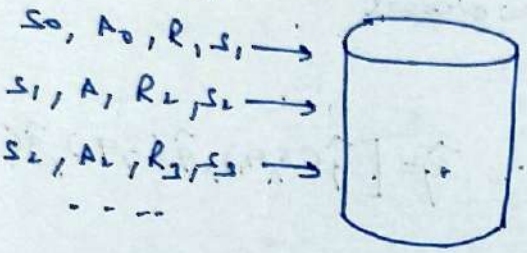
we calculate the gradient vector of the cost function with respect to the w parameters

$$\nabla L(\theta) = \left[\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2} \dots \frac{\partial L}{\partial w_n}\right]$$

with the gradient vector, we will make a SGD step

$$\theta \leftarrow \theta - \alpha \nabla L(\theta)$$

## Replay memory




Memory that stores the state transitions that the agent experiences

The memory has a limited size and when it fills up, it replaces old transitions with new ones

$k = (s, t, R, s') \sim B$

To update the NN, we randomly choose a batch of transitions from the memory

The batch of the transitions obtained from the memory is used to calculate the cost function and update the $\theta$ parameters

## Target network

Boot strapping + function approximators

$$y_i = R_i + \gamma \hat{q}(S_i', A_i' | \theta_{targ})$$

when a value is changed, nearby values will also be affected

By modifying a $\hat{q}(S_i, A | \theta)$ estimate we also modify its $\hat{q}(S_i', A_i' | \theta_{target})$ target.

we make a copy of the neural network to calculate the target

$\theta_{targ} \longleftrightarrow \theta$

This nn does not change with SGD. Its $\theta$ parameters remain the same.

Each k episode, we copy the parameters from the agent network to the target network

$$\theta_{targ} \leftarrow \theta$$

## Deep Q-learning

Q-Learning + Neural Network

$s = [s_1, s_2]$ $\quad\quad$ $\hat{q} = [\hat{q}(s,a_1), \hat{q}(s,a_2), \hat{q}(s,a_3)]$

Experiance replay

Policy gradient method $\quad$ Value based method $\rightarrow$ function approximators

Value table

$a = \arg\max_a Q(s,a)$ $\quad\quad$ $a = \arg\max_a \hat{q}(s,a|\theta)$

The policy is defined based on these values.

Policy gradient method :- They use a function approximator to estimate the probability to taking each action

$$\pi(a|s,\theta) \in [0,1]$$

Input $\quad\quad$ output

$s = [s_1, s_2]$ $\quad\quad$ $\pi(s|\theta) = [p(a_1), p(a_2), p(a_3)]$

The nn is the policy

Advantages Value based method cannot represent the stochastic policy in a simple way.

Greedy policy

$$\pi(a'|s) \begin{cases} 1 & \text{if } a' = \arg\max_a \hat{q}(s,a|\theta) \\ 0 & \text{else.} \end{cases}$$

Epsilon-greedy policy

$$\pi(a'|s) \begin{cases} 1 - \varepsilon + \frac{\varepsilon}{|A|} & \text{if } a' = \arg\max_a \hat{q}(s,a|\theta) \\ \frac{\varepsilon}{|A|} & \text{else} \end{cases}$$

Si $\pi^*(s) = [0.7, 0.3]$, How do we represent it?

The policy changes more smoothly during learning.

Value based methods

When the maximum q-value changes they choose a new action 100% of the time $a = \arg\max_a \hat{q}(s,a)$

Policy gradient method

The probability of choosing an action changes in small increments

$$a \sim \pi(s|\theta)$$

# The Bellman Equation

Concept

S - state  a - Action  R - Reward

$\gamma$ - Discount

$$V(s) = \max_a (R(s,a) + \gamma V(s'))$$

## Markov decision Process (MDP)

Deterministic search

↓

100%

Nondeterministic search

10% ↙  ↓  ↘ 10%

80%

$$0.8 * V(s_1') + 0.1 * V(s_2') + 0.1 V(s_3')$$

$$V(s) = \max_a (R(s,a) + \overbrace{\gamma V(s')})$$

$$V(s) = \max_a \left( R(s,a) + \gamma \sum_{s'} P(s,a,s') V(s') \right)$$

## Policy vs Plan

## Adding a Living Penalty

## Q Learning Intuation

$$V(s) = \max_a \left( R(s,a) + \gamma \sum_{s'} P(s,a,s') V(s') \right)$$

$$Q(s,a) = R(s,a) + \gamma \sum_{s'} P(s,a,s') V(s')$$

$$V(s') \rightarrow \max_{a'} Q(s',a')$$

$$Q(s,a) = R(s,a) + \gamma \sum_{s'} P(s,a,s') \max_{a'} Q(s',a')$$

## Temporal difference :

$$Q(s,a) = R(s,a) + \gamma \sum_{s'} P(s,a,s') \max_{a'} Q(s',a')$$

$$Q(s,a) = R(s,a) + \gamma \max_{a'} Q(s',a')$$

| Before | After |
|---|---|
| $Q(s,a)$ | $R(s,a) + \gamma \max_{a'} Q(s',a')$ |

$$TD(q,s) = R(s,a) + \gamma \max_{a'} Q(s',a') - Q(s,a)$$

$$Q(s,a) = Q(s,a) + \alpha TD(q,s)$$

$$Q_t(s,a) = Q_{t-1}(s,a) + \alpha TD_t(a,s)$$

$$Q_t(s,a) = Q_{t-1}(s,a) + \alpha\left(R(s,a) + \gamma \max_{a'} Q(s',a') - Q_{t-1}(s,a)\right)$$

## RL | Multiarm Bandits

Very close to real not exactly same to real returns rate

$\epsilon$ - Greedy approach

Reward, Action, Value of action (Q)

$$Q_{n+1} = Q_n + \frac{1}{n}\left(R_n - Q_n\right)$$

$$Q_{n+1} = Q_n + \alpha\left(R_n - Q_n\right)$$

$$= \alpha R_n + (1-\alpha) Q_n$$

$$= \alpha R_n + (1-\alpha)\left[Q_{n-1} + \alpha\left(R_{n-1} - Q_{n-1}\right)\right]$$

$$= \alpha R_n + (1-\alpha)\left[\alpha R_{n-1} + (1-\alpha) Q_{n-1}\right]$$

$$= \alpha R_n + (1-\alpha)\alpha R_{n-1} + (1-\alpha) Q_{n-1}$$

$$= (1-\alpha)^n Q_1 + \sum_{i=1}^{n} \alpha (1-\alpha)^{n-i} R_i$$

Naive strategy

1. Explore only

| 100 | 100 | 100 |
|---|---|---|
| × 10 | × P | × 5 |
| 1000 | 100P | 500 |

2300

$[P = 700]$

2. Exploid only

$R_1 = 7 \quad R_2 \to 8 \quad R_3 \to 5$

$20 + 297(P) = 2394$

$[P = 330]$

3. $\varepsilon$ - Greedy

$\varepsilon = 10\%$

↳ 30 Explore

↳ 270 Exploit

Result 2907 Avg.

$[P \approx 100]$

Zero regret strategy

$P/T \to 0$

$T \to \infty$

$Q_t(a) \doteq$ Sum of rewards when a taken prior to t

number of times a taken prior to t

$$= \frac{\sum_{i=1}^{t-1} R_i \cdot 1_{A_i = a}}{\sum_{i=1}^{t-1} 1_{A_i = a}}$$

Greedy action selection rule

$$A_t \doteq \underset{a}{\text{argmax}}\, Q_t(a)$$

New Estimate ← old estimate + step size [Target − Old estimate]

$$Q_{n+1} = \frac{1}{n} \sum_{i=1}^{n} R_i$$

$$= \frac{1}{n} \left( R_n + \sum_{i=1}^{n-1} R_i \right)$$

$$= \frac{1}{n} \left( R_n + (n-1) \frac{1}{n-1} \sum_{i=1}^{n-1} R_i \right)$$

$$= \frac{1}{n} \left( R_n + (n-1) Q_n \right)$$

$$= \frac{1}{n}\left(R_n + n Q_n - Q_n\right)$$

$$= Q_n + \frac{1}{n}\left[R_n - Q_n\right]$$

Exponential Recency - Weighted Average

$$Q_{n+1} = Q_n + \alpha\left[R_n - Q_n\right]$$

$$= \alpha R_n + (1-\alpha) Q_n$$

$$= \alpha R_n + (1-\alpha)\left[\alpha R_{n-1} + (1-\alpha) Q_{n-1}\right]$$

$$= \alpha R_n + (1-\alpha)\alpha R_{n-1} + (1-\alpha)^2 Q_{n-1}$$

$$= \alpha R_n + (1-\alpha)\alpha R_{n-1} + (1-\alpha)^2 \alpha R_{n-2} +$$

$$\cdots + (1-\alpha)^{n-1}\alpha R_1 + (1-\alpha)^n Q_1$$

$$= (1-\alpha)^n Q_1 + \sum_{i=1}^{n} \alpha (1-\alpha)^{n-i} R_i$$

UCB

$$A_t \doteq \arg\max_a \left[ Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right]$$

Gradient Bandit Algorithm

$$Pr\{A_t = a\} \doteq \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}} \doteq \pi_t(a)$$

Gradient Bandits. Update with stochastic Gradient Ascent

$$H_{t+1}(A_t) \doteq H_t(A_t) + \alpha(R_t - \bar{R}_t)(1 - \pi_t(A_t))$$

$$H_{t+1}(a) \doteq H_t(a) - \alpha(R_t - \bar{R}_t)\pi_t(a)$$

## Upper confidence Bound (UCB)

hope around estimate value

$$Q_A + \sqrt{\frac{2 \ln N}{n_A}}$$

$$Q_B + \sqrt{\frac{2 \ln N}{n_B}}$$

$$Q_C + \sqrt{\frac{2 \ln N}{n_C}}$$

Q value :- simply average reward from machine

$n_A$ = no. of times machine A is played

$N$ = Total plays in all the machine

So $N = n_A + n_B + n_C$

Thompson sampling

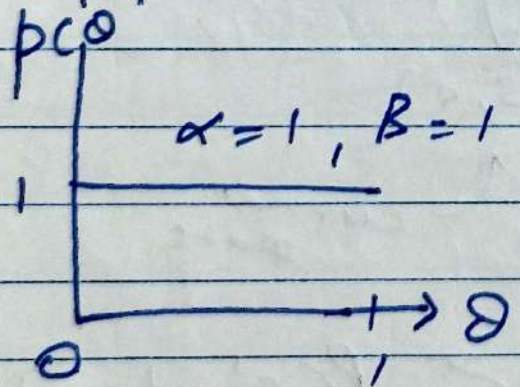
$\beta$- Distribution

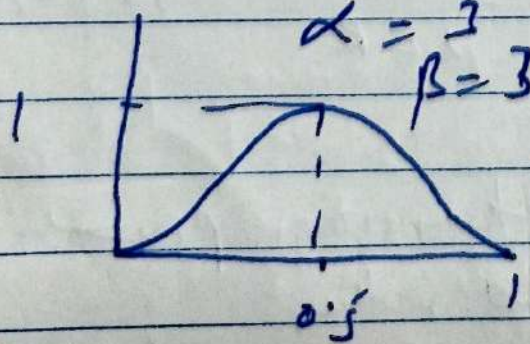
$$p(\theta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \theta^{\alpha-1}(1-\theta)^{\beta-1}$$

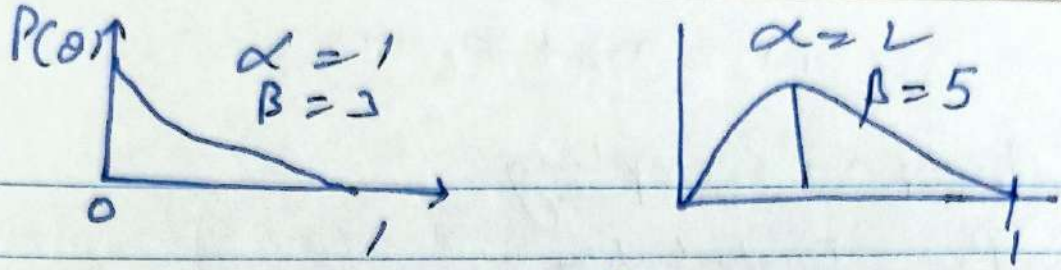
$$\theta \in [0,1]$$

where, $\Gamma(.)$ gamma function

$\alpha, \beta \rightarrow$ shape parameters $> 0$






Markov Decision Process

1. Agent
2. Environment
3. Reward
4. state

Policy iteration  value iteration

Policy $P(a|s)$

Translation dynamics

$$P(s', r | s, a)$$

(next state  current state

$$P(s'|s,a) = \sum_{r \in R} P(s', r | s, a)$$

↑ state transition probability

Total discount return

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^{N-1} R_{t+N}$$

Value function:
(with respect to a policy $\pi$)

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid s_t = s]$$

$$V_\pi(s) = E_\pi[G_t \mid s_t = s]$$

$$= E_\pi[R_{t+1} + \gamma G_{t+1} \mid s_t = s]$$

$$= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, r)$$

$$\left[r + \gamma E_\pi[G_{t+1} \mid s_{t+1} = s']\right]$$

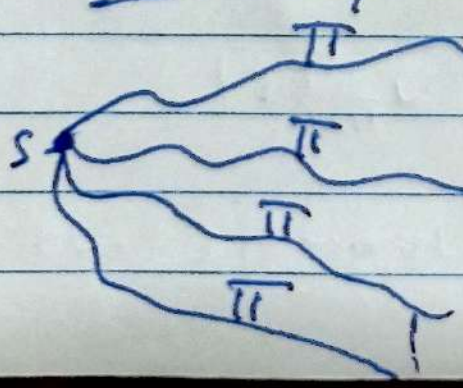
Current state — Policy — dynamics

$$\Rightarrow v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)$$

$$\left[r + \gamma v_\pi(s')\right]$$

next state's value

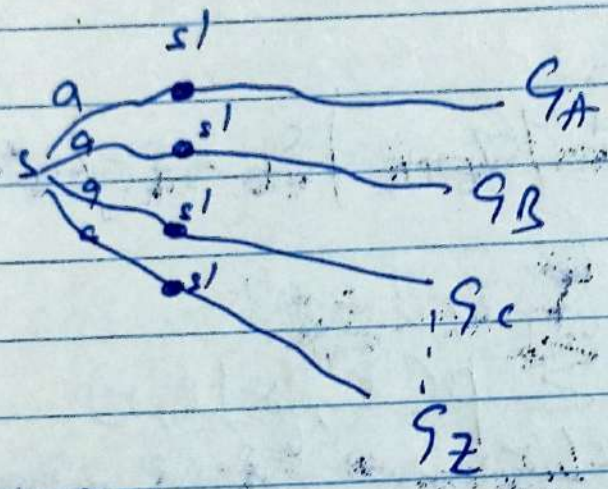
Bellman equation



$$V_\pi(s) = \frac{1}{26} \sum (G_A + G_B + \dots + G_Z)$$

Value of an action:

$$q_\pi(s,a) = E_\pi[G_t \mid s_t = s, A_t = s]$$




$$\hat{q}_{T_1}(s,a) = \frac{1}{26} \sum (G_A + G_B + \dots G_Z)$$

optimal state value and optimal action value

optimal → Maximum possible value of a state / action.

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

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

# UCB derivation

$$L_t = \sum_a N_t(a)\,\Delta_a$$

$$\Delta_{a^*} = 0$$

$$\forall_a = a^* \quad N_t(a)\,\Delta_a \leq X_a \log t$$

$$A_t = \arg\max_a Q_t(a) + U_t(a)$$

$$m \leq t \quad N_m(a)\,\Delta_a \leq X_a \log m \leq X_a \log t$$

$$n \in \{m+1, \ldots t\} : N_n(a)\,\Delta_a > X_a \log n$$

$$\mathbb{E}[N_t(a)] = \mathbb{E}\left[\sum_{n=1}^{t} I(A_n = a)\right]$$

$$= \mathbb{E}\left[N_m(a) + \sum_{n=m+1}^{t} I(A_n = a)\right]$$

$$= \mathbb{E}\left[X_a \cdot \frac{\log t}{\Delta_a} + \sum_{n=m+1}^{t} I(A_n = a)\right]$$

$$= X_a \frac{\log t}{\Delta_a} + \sum_{n=m+1}^{t} \mathbb{E}\left[I(A_n = a) \,\middle|\, N_n(a)\,\Delta_a > X_a \log n\right]$$

$P(A_n = a \quad | N_n(a)\Delta_a > x_a \log n]$

$P(A_n = a)$? Given $N_n(a)\Delta_a > x_a \log n$

$\leq P(Q_t(a) + U_t(a) \geq Q(a^*) + U_t(a^*)$

Simplified the equation

$= P(Q + U \geq Q^* + U^*) \leq 1$

$= P(Q+U \geq Q^* + U^* \mid Q^* + U^* \leq y) \cdot \underbrace{P(Q^* + U^* \leq y)}_{\leq \frac{1}{n}}$

$+ \underbrace{P(Q+U \geq Q^* + U^* \mid Q^* + U^* > y)}_{\frac{1}{n}} \cdot \underbrace{P(Q^* + U^* > y)}_{}$

$y \doteq q(a^*)$

$q(a^*)$

$P(Q^* + U^* \leq q^*) \leq e^{-N_n(a^*) U_n(a^*)^2} = \frac{1}{n}$

$\sum_{n=1}^{t} \frac{1}{n} \leq \log t+1$

$$e^{-Nu^2} = \frac{1}{n} \Rightarrow u = \sqrt{\frac{\log n}{N}}$$

$$P(\Phi + u \geq \Phi^* + u^* \mid \Phi^* + u^* > q^*) \leq \frac{1}{n}?$$

$$\leq P(\Phi + u \geq q^*)$$

$$= P(-\Phi - u \leq -q^* + q - q)$$

$$= -\Delta_q$$

$$= P(-\Phi - \Delta_q - u \leq q)$$

$$\Delta_q \geq 2u \; ?$$

$u$?

$$N \cdot \Delta_q > x_q \log n$$

$$\Delta_q > x_q \frac{\log n}{N}$$

$$x_q \, Ut(q)^2$$

$$x_q = \frac{4}{\Delta_q} \Rightarrow \Delta_q^2 > 4u\,t(q)^2$$

$$\Rightarrow \Delta_q > 2u$$

$\leq P(-\delta + 2u - u) \leq -\eta)$

$= P(-\delta + u \leq -\eta) \quad | \; u = \sqrt{\frac{\log \frac{4}{\delta}}{N}}$

$< e^{-N \cdot u^2} = \frac{1}{n}$

Transfer learning — All weights

Deep learning in hindi

— Used in mostly in CNN vision

— " Alexnet 8layer

Imagenet 1M, 1000class

→ 1000 image

→ Ordinary system

Less data

↙ ↘

Don't use deep learning

- enabling mechanism

use Transfer learning

— weight same

weights and architecture same