Date _/_/_

Action-Values

* The value is the expected reward

$$q_*(a) \doteq \mathbb{E}[R_t | A_t = a] \quad \forall a \in \{1 \cdots k\}$$

$$= \sum_r p(r|a)\, r$$

* The goal is to maximize the expected reward

$$\text{argmax}_a \, q_*(a)$$

* Calculating $q_*(a)$

$\rightarrow q_*(a) = .5 \times -11 + .5 \times 9$

↓ prob of failure ↓ reward when fail ↓ prob of success ↓ reward when success

→ $q_*(a)$ is not known

Sample-Average Method

$$Q_t(a) = \frac{\text{sum of rewards when a taken prior to } t}{\text{number of times a taken prior to } t}$$

$$= \frac{\sum_{i=1}^{t-1} R_i}{t-1}$$

Date _/_/_

Action selection

greedy action

$$a_g = \text{argmax}_a Q(a)$$

Summary :-

→ Sample average method can be used to estimate action values.

→ The greedy action is the action with the highest value estimate.

* Incremental update rule

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

pull current reward out of the sum

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

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

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

Date _ / _ / _

Recall (current value estimate)

$$Q_n = \frac{1}{n-1} \sum_{i=1}^{n-1} R_i$$

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

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

NewEstimate ← OldEstimate + StepSize [Target − OldEstimate]

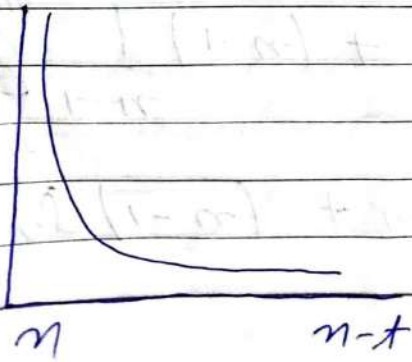
$$Q_{n+1} = Q_n + \alpha_n (R_n - Q_n)$$

$$\alpha_n \sim [0, 1]$$

$$\alpha_n = 1/n$$

Non-stationary bandit problem :-

The distribution of rewards changes with time



Date _/_/_

* Decaying past rewards

$$Q_{n+1} = Q_n + \alpha_n (R_n - Q_n)$$

$$= \alpha R_n + Q_n - \alpha Q_n$$

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

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

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

$$= \alpha R_n + (1-\alpha)\alpha R_{n-1} + (1-\alpha)^2 \alpha R_{n-2} + \cdots + (1-\alpha)^{n-1} \alpha R_1 + (1-\alpha)^n Q_1$$

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

Q_1 → initial action value

→ The first term tells us that the contribution of Q_1 decreases exponentially with time.

→ The second term tells us the rewards further back in time contribute exponentially less to the sum.

The most recent rewards contribute most to our current estimate

Date _/_/_

* Exploration & Exploitation trade-off

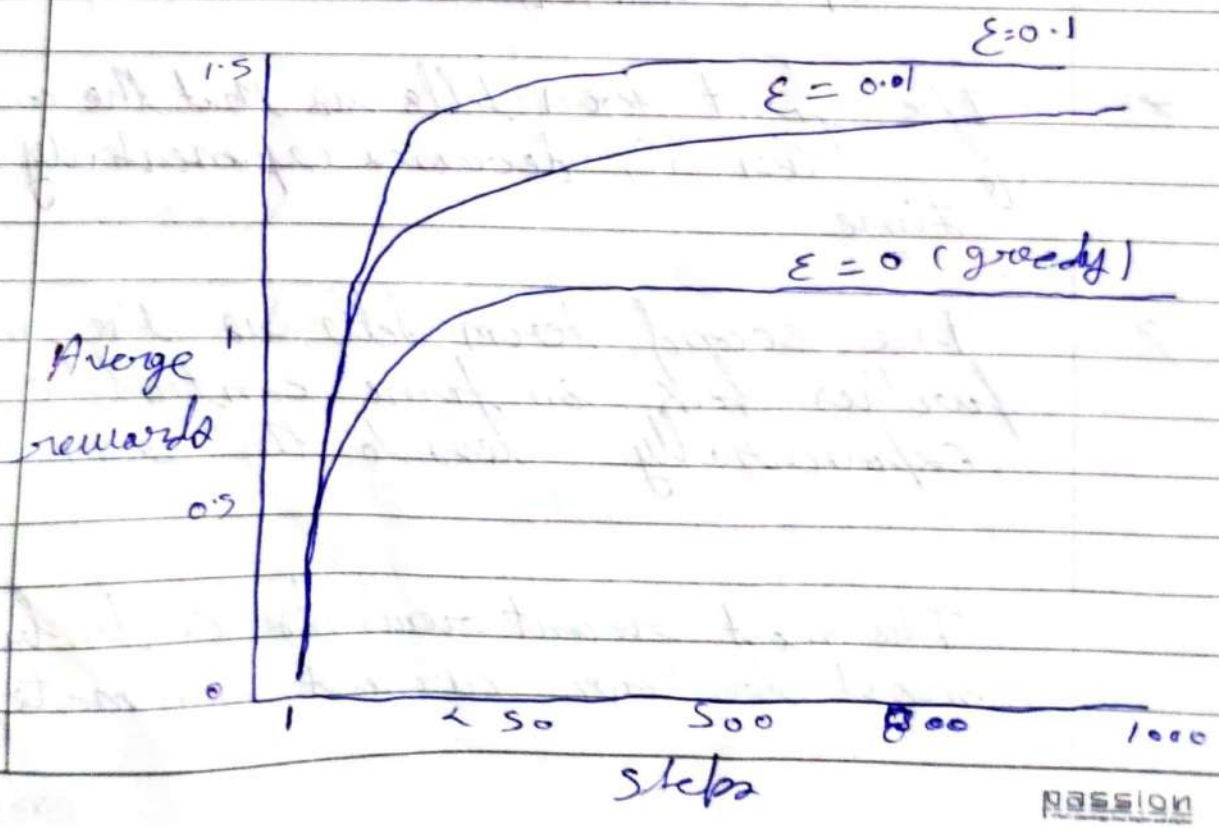
Exploration - improve knowledge for long term benefit

Exploitation - exploit knowledge for short term benefit

→ Epsilon - Greedy Action Selection

epsilon - probability of choosing to explore

$$A_t \leftarrow \begin{cases} \underset{a}{\operatorname{argmax}}\, Q_t(a) & \text{with prob } 1-\epsilon \\ a \sim \text{Uniform}(\{a_1, \ldots, a_k\}) & \text{with prob } \epsilon \end{cases}$$



2.0
0.5
1.5

2 + 0.5 (1 - 4)

Date _/_/_

Clinical Trials :-

A reward of 1 if the treatment succeeds otherwise 0

Initial Values →
$Q_1(P) = 2.0$
$Q_1(Y) = 2.0$
$Q_1(B) = 2.0$

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

Let $\alpha = 0.5$

doctor choose P

$$Q_2 = 2 + 0.5(1 - 2)$$
$$= 2 - 0.5$$
$$Q_2 = 1.5$$

Limitations of optimistic initial values

① optimistic initial values only derive early exploration

②

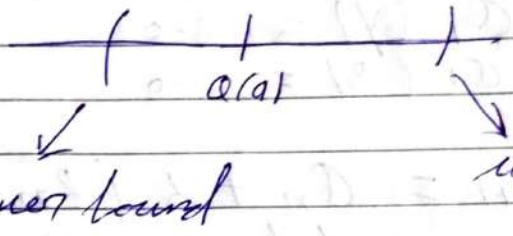
③ They are not well suited for non-stationary problem.

④ We may not know the what the optimistic initial value should be.

Date _/_/_

* Upper - Confidence Bound (UCB) Action selection

Uncertainty in estimates



→ The region in blue is the CI which represent our uncertainty.

→ If this region is large, we are uncertain that the value of action A is near or estimated value

→ If region is very small, we are very certain that the value of action A is near our estimated value.

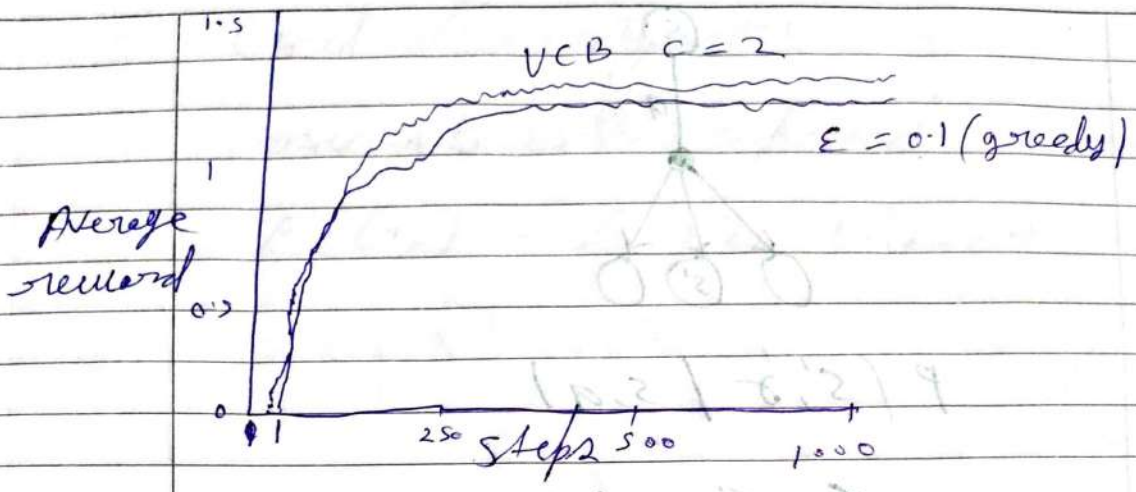
$$A_t = \text{argmax}\left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}}\right]$$

$Q_t(a)$ ↑ exploit, $\sqrt{\frac{\ln t}{N_t(a)}}$ ↑ explore

c → control the amount of exploration

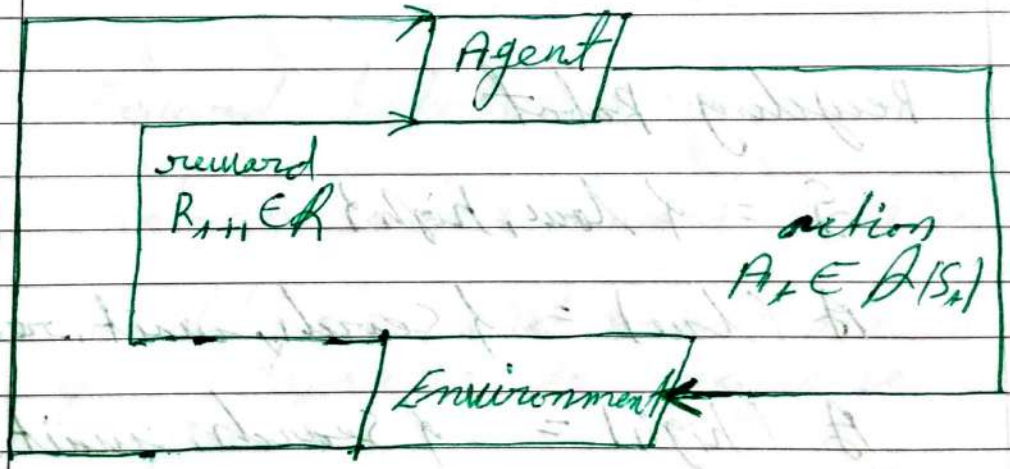
$$c\sqrt{\frac{\text{1-2 time steps}}{\text{times action taken}}}$$

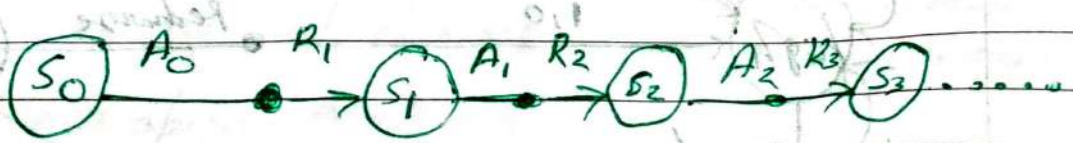
Date _/_/_



course-1 week-2

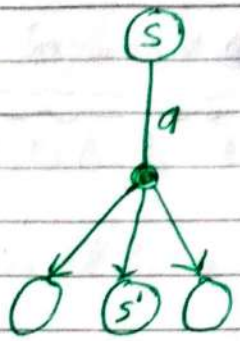
Markov Decision processes (MDP)





passion

Date _ / _ / _



$$p(s', r \mid s, a)$$

$$\sum_{s' \in S} \sum_{r \in R} p(s', r \mid s, a) = 1$$

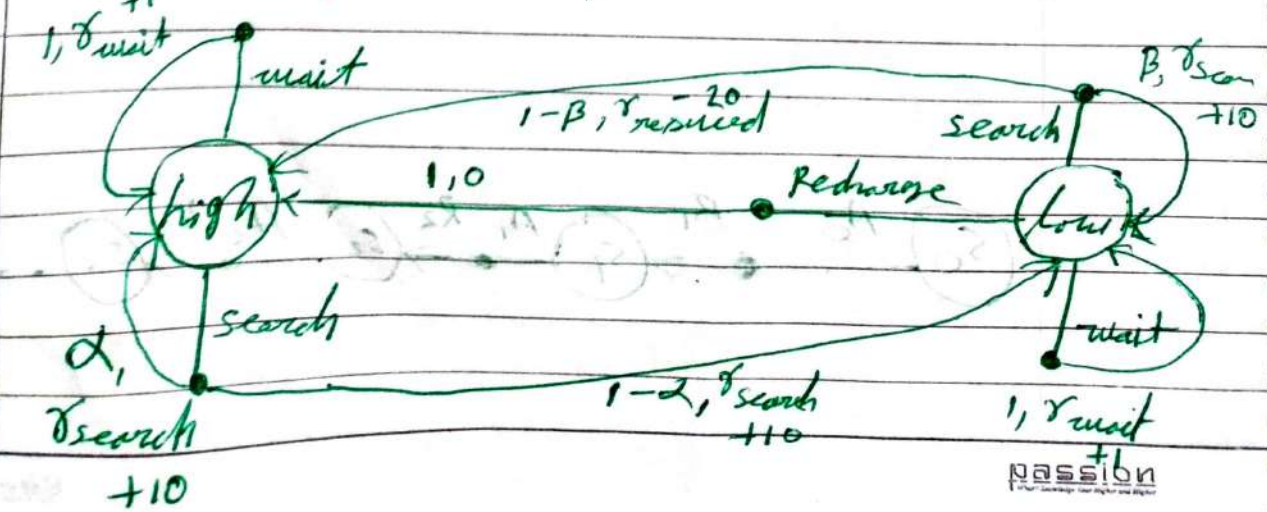
* The present state contains all the info necessary to predict the future

Recycling Robot

$S_+ = \{low, high\}$

$A(low) = \{search, wait, recharge\}$

$A(high) = \{search, wait\}$



Date _/_/_

Goal of an Agent:

maximize the expected return

$$E[G_t] = E[R_{t+1} + R_{t+2} + \cdots + R_T]$$

R_T ← final time step

* In episodic tasks, the agent-environment interaction breaks up into episodes
* Each episodes have terminal state

Reward hypothesis:

* Identify where reward signals come from
* develop algo that search the space of behaviors to maximize reward signals.

	Episodic task	Continuing task
*	Interaction breaks naturally into episodes	Interaction goes on continually
*	each episode ends in a terminal state	No terminal state
*	episodes are independent	

passion

MDP: sequential decision making

Date _/_/_

Continuing Tasks

$G_t \doteq R_{t+1} + R_{t+2} + \cdots$

Discounting

$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \cdots \gamma^{k-1} R_{t+k} + \cdots$

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

How to make sure G_t is finite ?

Discount the reward in the future by γ where

$$0 < \gamma < 1$$

Assume R_{max} is the maximum reward the agent can receive

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

$$R_{max} \sum_{k=0}^{\infty} \gamma^k$$

↑

Geometric series

$$= R_{max} \times \frac{1}{1-\gamma} \longrightarrow \text{finite}$$

Date _/_/_

Effect of γ on agent behavior

$$\gamma = 0$$

$$G_t \doteq R_{t+1}$$

Agent only cares about the immediate reward → short-sighted agent

$$\gamma \to 1$$

Agent take future rewards into account more strongly.

* Recursive nature of returns:

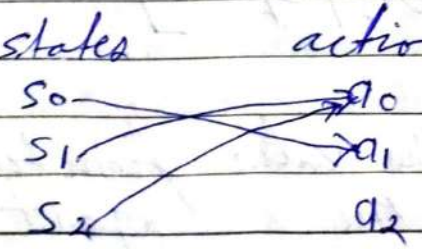
$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots$$

$$= R_{t+1} + \gamma (R_{t+2} + \gamma R_{t+3} + \cdots)$$

* $$G_t = R_{t+1} + \gamma G_{t+1}$$

+ A policy maps each state to a single action. This kind of policy is called the deterministic policy.

$$\pi(s) = a$$



State	Action
s_0	a_1
s_1	a_0
s_2	a_0

Date 20/04/2021

* A stochastic policy is one where multiple actions may be selected with non-zero probability.

$$\pi(a|s)$$

$$\sum_{a \in A(s)} \pi(a|s) = 1$$

$$\pi(a|s) \geq 0$$

* policy can only depend on the current state
(valid & invalid policies)

---- state value function ----

a state value function is the future reward an agent can expect to receive starting from a particular state & more precisely the state value function is the expected return from a given state.

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

↓

Pi indicate the value function is contingent on the agent

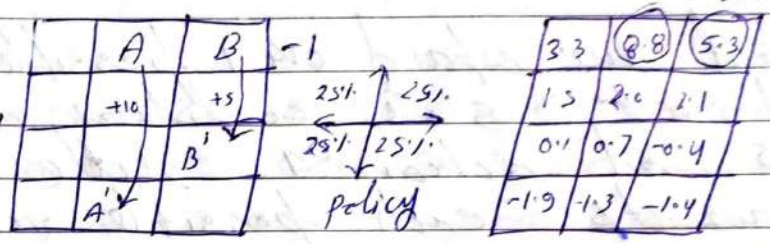
Date _/_/_

selecting actions according to π.

------ Action-value function ------

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

An action value describes what happens when the agent first selects a particular action.



* State-value functions represent the expected return from a given state under a specific policy

* Action value function represent the expected return from a given state after taking a specific action. later following a specific policy.

---- Bellman Equation ----

The Bellman equation for the state value function defines a relationship between the value of a state and the value of his possible successor state.

Date _/_/_

---- State-value Bellman Equation ----

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

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

--> first we expand the expected returns as a sum over possible action choices made by the agent

--> second we expand over possible rewards and next states conditioned on state s and action a. Little r represents each possible reward outcome.

$$= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r \mid s, a)\left[r + \gamma E_\pi[G_{t+1} \mid S_{t+1} = s']\right]$$

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

---- Action-value Bellman Equation ----

Value of a state action pair in terms of its possible successors state action pair.

In this case, the equation does not begin with the policy selecting an action because action

Date _/_/_

is already fixed as part of the state action pair. Instead we skip directly to the dynamic function p to select the immediate reward and next state s'.

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

$$= \sum_{s'} \sum_{r} p(s', r | s, a) \left[r + \gamma E_\pi[G_{t+1} | S_{t+1} = s'] \right]$$

we can't stop here, we want to recursive equation for the values of one state action pair in terms of next state action pair.

At the moment, we have the expected return given only the next state To change this, we can express the ~~exp~~ expected return from the next state as a sum of the agents possible action ~~choices~~ choices.

In particular, we can change the expectation to be conditioned on both the next state & the next action and then sum over all possible actions.

Bellman equation does not work with larger

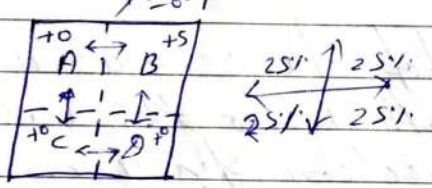
178 P

Date _/_/_

$$\sum_{s'}\sum_{r} P(s', r \mid s, a)\left[r + \gamma \sum_{a'} \pi(a' \mid s') E_\pi[G_{t+1} \mid S_{t+1}\right]$$

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

Example: Gridworld



$$V_\pi(A) \doteq E_\pi(G_t \mid S_t = A)$$
$$V_\pi(B) \doteq E_\pi(G_t \mid S_t = B)$$

$$V_\pi(A) = \sum_a \pi(a \mid A)\left(r + 0.7 V_\pi(s')\right)$$

(for each action there's only one possible associated next state and reward. That's the sum over s' and r reduces to a single value.

$$V_\pi(A) = \sum_a \pi(a \mid A)\left(r + 0.7 V_\pi(s')\right)$$

$$V_\pi(A) = \frac{1}{4}\left(5 + 0.7 V_\pi(B)\right) + \frac{1}{4} 0.7 V_\pi(C) + \frac{1}{2} 0.7 V_\pi(A)$$

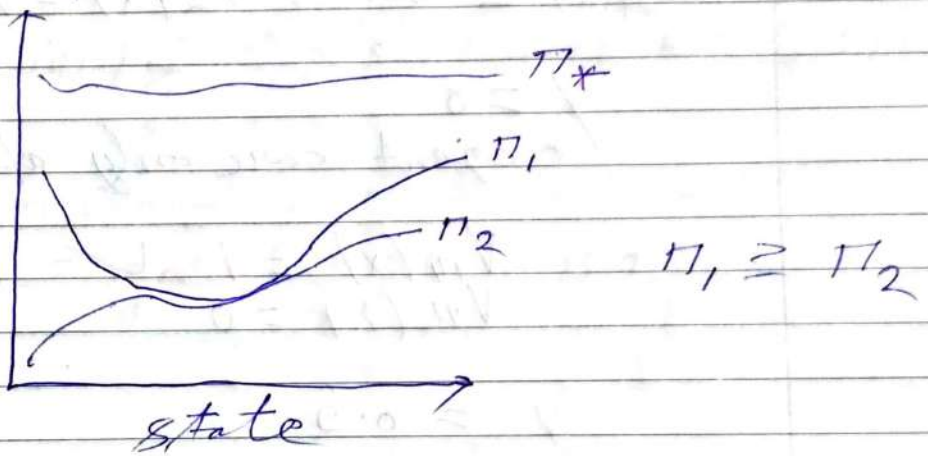
$$V_\pi(B) = \frac{1}{2}\left(5 + 0.7 V_\pi(B)\right) + \frac{1}{4} 0.7 V_\pi(A) + \frac{1}{4} 0.7 V_\pi(D)$$

Date _/_/_

$= s', A_{t+1} = a']$

$V_\pi(A) = 4.2$
$V_\pi(B) = 6.1$
$V_\pi(C) = 2.2$
$V_\pi(D) = 4.2$

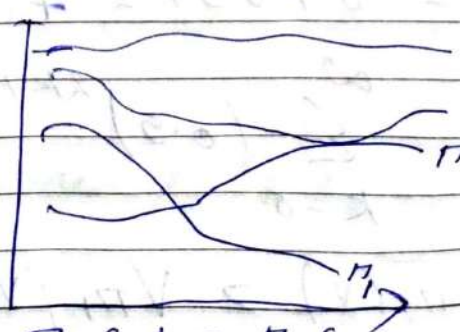
* optimal policies



$\pi_1 \geq \pi_2$

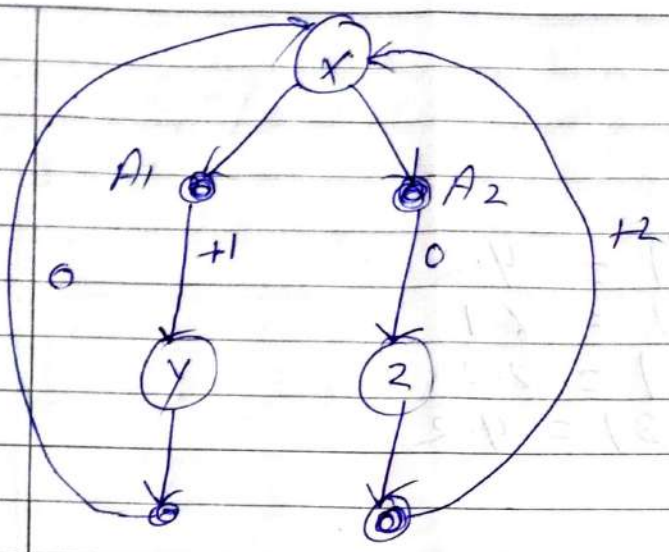
* An optimal policy π_* is as good as or better than all other policies.

* There is always an optimal policy



$\pi_3(s) = \pi_1(s)$, $\pi_3(s) =$ [illegible]

Date _/_/_



$\pi_1(x) = A_1$. $\pi_2(x) = A_2$

$\gamma = 0$

* agent care only about immediate reward

$V_{\pi_1}(x) = 1$
$V_{\pi_2}(x) = 0$

$\gamma = 0.9$

$V_{\pi_1}(x) = 1 + 0.9 * 0 + (0.9)^2 * 1 + \cdots$

$$\sum_{k=0}^{\infty} (0.9)^{2k} = \frac{1}{1-0.9^2} \simeq 5.3$$

$V_{\pi_2}(x) = 0 + 0.9 * 2 + (0.9)^2 * 0 + \cdots$

$$\sum_{k=0}^{\infty} (0.9)^{2k+1} * 2 \simeq 9.5$$

$V_{\pi_2}(x) > V_{\pi_1}(x)$

Date _/_/_

* An optimal policy is $\doteq$ the policy with the highest possible value function in all states.

V_* $v_{\pi_*}(s) \doteq \mathbb{E}_{\pi_*}[G_t \mid S_t = s] = \max_\pi v_\pi(s)$ for all $s \in \mathcal{S}$

q_* $q_{\pi_*}(s,a) = \max_\pi q_\pi(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$

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

Bellman Optimality equation for v_* & q_*

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

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

---

optimal policy

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

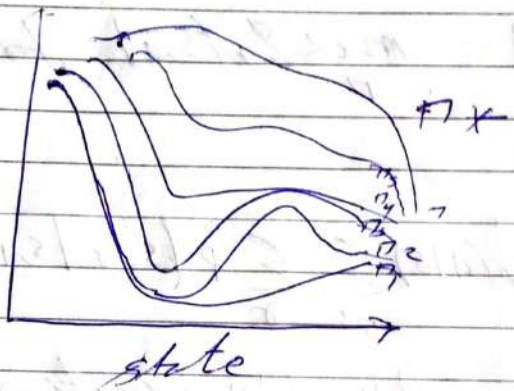
$$\pi_*(s) = \arg\max_a q_*(s,a)$$

week-04

Date _/_/_

* policy evaluation is the task of determining state-value function v_π, for a particular policy π

* Control is the task of improving an existing policy.



$\pi, P, \gamma \rightarrow$ [DP] $\rightarrow v_\pi$ policy evaluation

$P, \gamma \rightarrow$ [DP] $\rightarrow \pi_*$ control

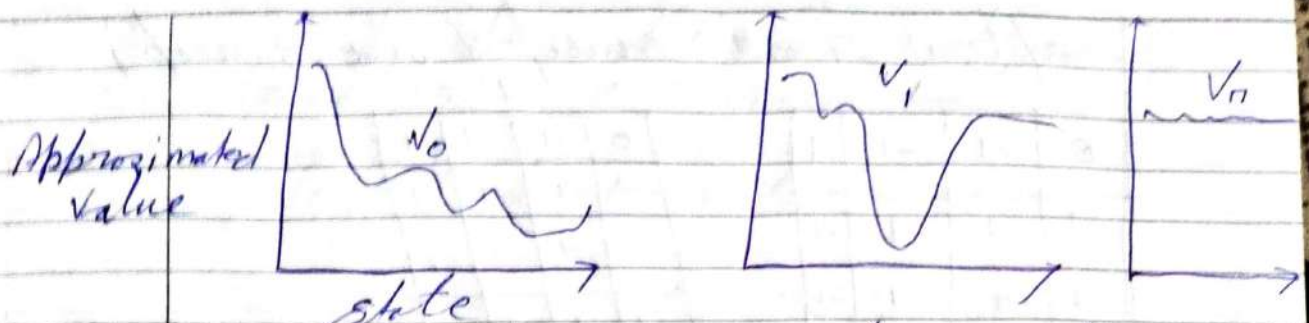
---- Iterative policy evaluation ----

use bellman equation as update rule

$$v_{k+1}(s) \leftarrow \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r | s, a)[r + \gamma v_k(s')]$$

(sequence of better estimation for value function)

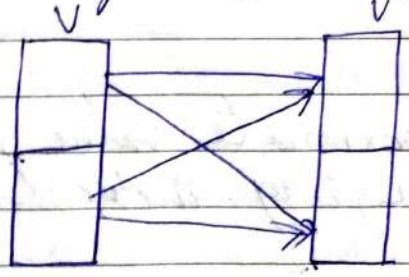
Date _/_/_

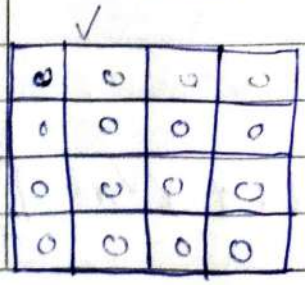


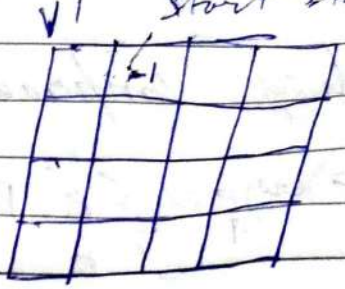
for any V_0

$$\lim_{k \to \infty} V_k = V_n$$

We store two arrays. each has one entry for every state. One array which we label V, stores the current approximation value function. Another array V' stores the updated value. By using 2 arrays, we can compute new values from the old one state at a time without the old values being changed in the process.







$0.25 * (-1 + 0) + 0.25 * (-1 + 0) + 0.25 * (-1 + 0) + 0.25 * (-1 + 0) =$
-1

Date _/_/_

after first sweep (one sweep)

V

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

V'

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

* for multiple sweeps algorithm

Input π, the policy to be evaluated

$V \leftarrow \vec{0}$, $V' \leftarrow \vec{0}$

Loop:

$\Delta \leftarrow 0$

Loop for each $s \in S$:

$V'(s) \leftarrow$ (formula written above)

$\Delta \leftarrow \max(\Delta, |V'(s) - V(s)|)$

$V \leftarrow V'$

until $\Delta < \phi$ (a small +ve number)

Output $V \approx V_\pi$

once the approximate value function stops changing, we've converge to V_π

* policy Improvement

$$\pi(s) = \arg\max_a \sum_{s'} \sum_r p(s', r | s, a)[r + \gamma V_\pi(s')] \text{ for all } s \in S$$

passion

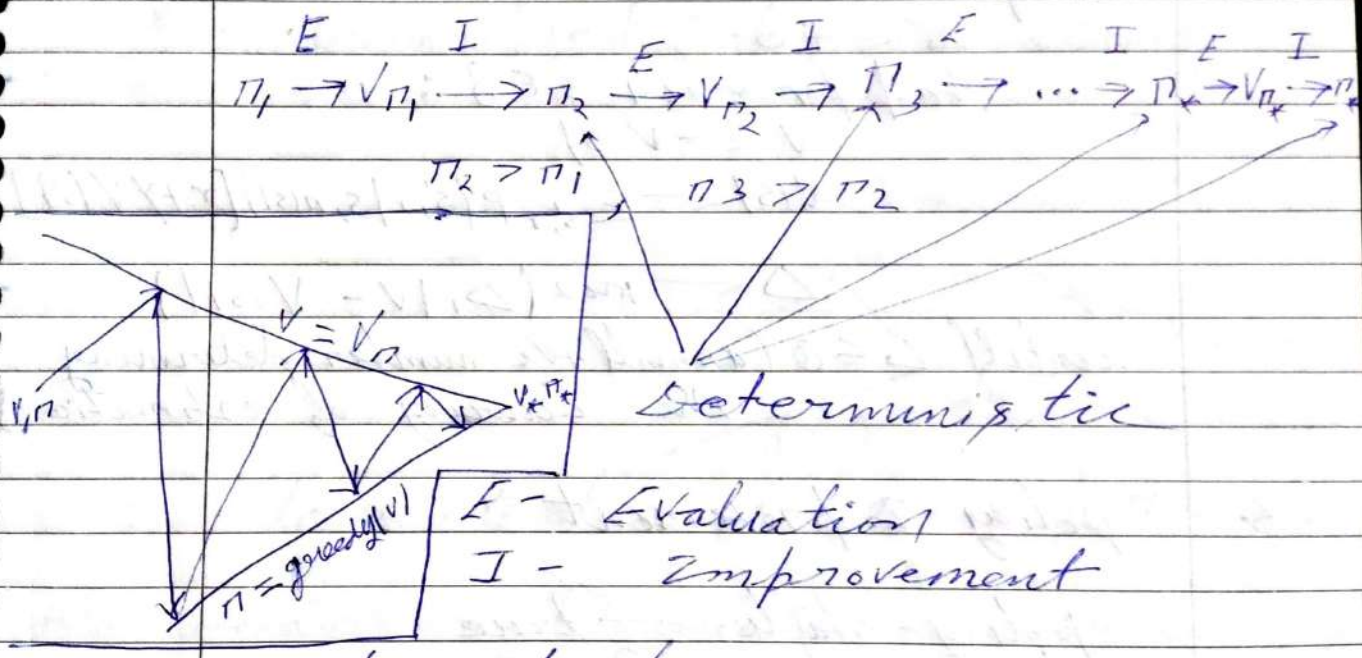
Date _/_/_

→ V_π obeys the bellman optimality equation → π is optimal

Policy Improvement Theorem

$q_\pi(s, \pi'(s)) \geq q_\pi(s, \pi(s))$ for all $s \in S$

* The new policy is a strict improvement over π unless π is already optimal.



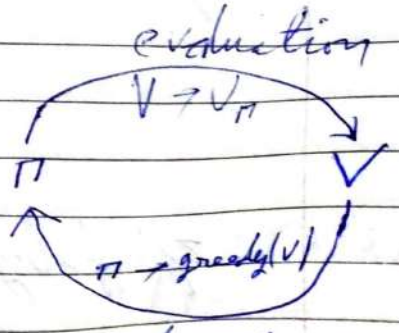
Policy Iteration

$\pi_2 \not\longleftarrow V_{\pi_1}$

$\pi_2 \longleftarrow V_{\pi_1}$

$\pi_2 \longrightarrow V_{\pi_2}$

$\pi_* \longleftrightarrow V_*$



Date _/_/_

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization

$V(s) \in \mathbb{R}$ and $\pi(s) \in A(s)$ arbitrarily for all $s \in S$

2. Policy Evaluation

Loop:

$\Delta \leftarrow 0$

Loop for each $s \in S$:

$v \leftarrow V(s)$

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

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

until $\Delta < \theta$ (a small +ve number determining the accuracy of estimation)

3. Policy Improvement

policy-stable $\leftarrow$ true

For each $s \in S$:

old-action $\leftarrow \pi(s)$

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

If old-action $\neq \pi(s)$, then policy-stable $\leftarrow$ false

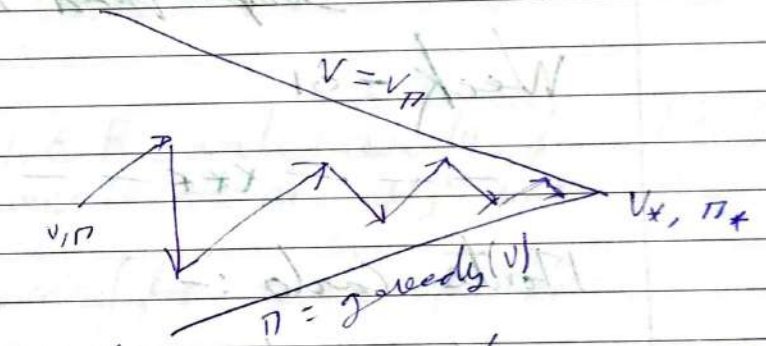
If policy-stable, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Date _/_/_

* Policy iteration works by alternating policy evaluation and policy improvement

* Policy iteration follows a sequence of better and better policies and value functions until it reaches the optimal policy and associated optimal value function.

* Generalized policy iteration



* Value iteration, for estimating $\pi \approx \pi_*$

Algo parameter: a small threshold $\theta > 0$ determining accuracy of estimation initialize $V(s)$, for all $s \in S^+$, except that $V(terminal) = 0$

Loop:

$\Delta \leftarrow 0$

Loop for each $s \in S$:

$v \leftarrow V(s)$

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

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

until $\Delta < \theta$

output a deterministic policy $\pi \approx \pi_*$, such that

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

Noah

Date _ / _ / _

Summary

* Value Iteration allows us to combine policy evaluation and improvement into a single update.

* Asynchronous dynamic programming method give us the freedom to update states in any order

Course-2

Sample based learning methods

Week-01

— — — XXX — — —

Monte Carlo :-

Monte carlo methods estimate value by averaging over a large number of random samples

So a monte carlo method for learning a value function would first observe multiple returns from the same state. Then it average those observed returns to estimate the expected return from that state. As the number of samples

Date _/_/_

increases, the average tends to get closer and closer to the expected return. The more returns the agent observe from a state, the more likely it is that the sample average is close to the state value.

MC prediction, for estimating $V = V_\pi$

Input: a policy π to be evaluated

Initialize:

$V(s) \in R$, arbitrarily, for all $s \in S$

Returns(s) $\leftarrow$ an empty list, "

Loop forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, \ldots$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots 0$

$G \leftarrow \gamma G + R_{t+1}$

Append G to Returns(S_t)

$V(S_t) \leftarrow$ average(Returns(S_t))

Date _/_/_

Computing returns efficiently

$\gamma = 0.5$

$G_0 = R_1 + \gamma G_1$

$G_1 = R_2 + \gamma G_2$

$G_2 = R_3 + \gamma G_3$

$G_3 = R_4 + \gamma G_4$

$G_4 = R_5 + \gamma G_5$

$G_5 = 0$

* compute backward to save computation

$G_5 = 0$

$G_4 = R_5 + 0.5 * 0 = 2$

$G_3 = R_4 + \gamma R_5 = 2$

$G_2 = R_3 + \gamma R_4 + \gamma^2 R_5 = 8$

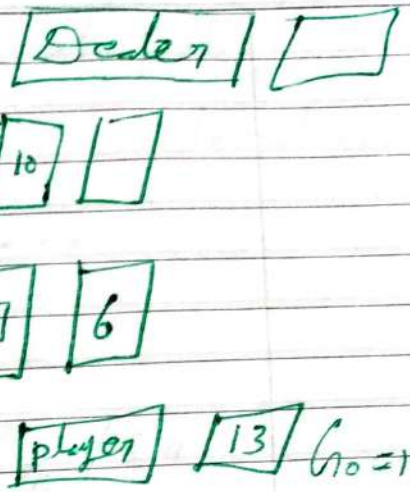
$G_1 = R_2 + \gamma R_3 + \gamma^2 R_4 + \gamma^3 R_5 = 8$

$G_0 = R_1 + \gamma R_2 + \gamma^2 R_3 + \gamma^3 R_4 + \gamma^4 R_5 = 7$

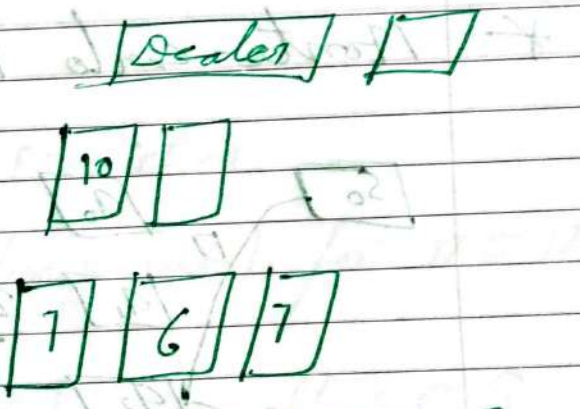
s	Returns (s)	V(s)
A	Returns (A) = [1]	1
B	Returns (B) = [1]	1

$R = -1, 0, +1$

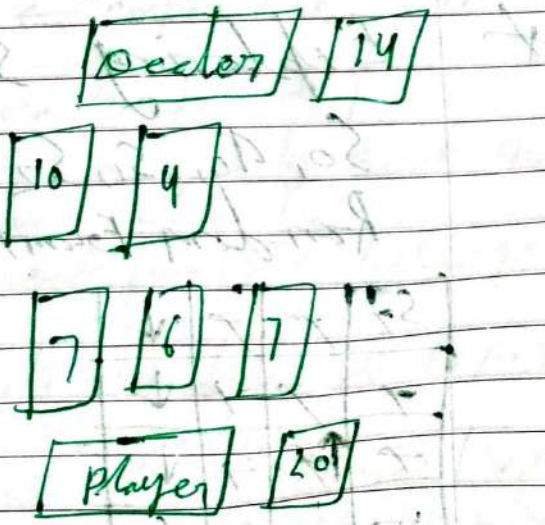
Date _/_/_



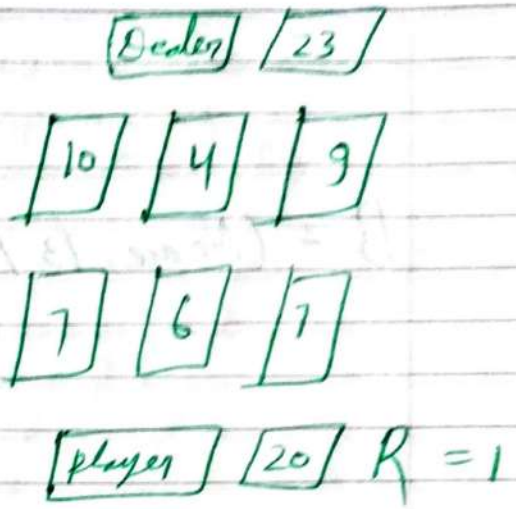
$B = (No\ ace, 13, 10)$



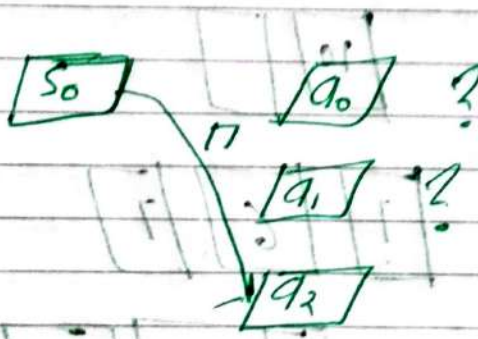
$A = (No\ Ace, 20, 10)$



Date _ / _ / _



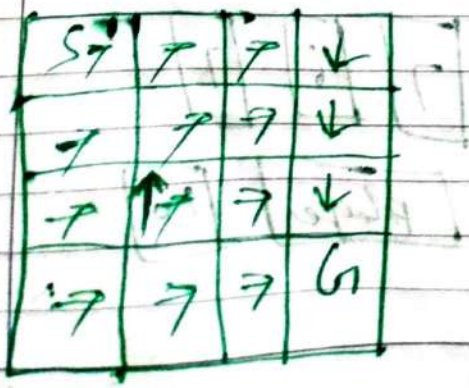
* Monte-carlo Action-values

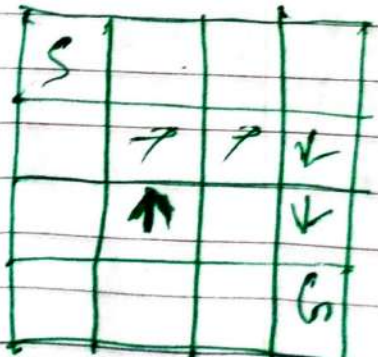


* Exploring Starts

$S_0, a_0, S_1, S_2 \ldots$

Random From S and P





Date _/_/_

Monte Carlo Generalized policy Iteration

$\pi_0 \rightarrow \pi_1 \rightarrow \pi_2 \rightarrow \cdots$

Improvement:

$\pi_{k+1}(s) \doteq \underset{a}{\operatorname{argmax}}\, q_{\pi_k}(s,a)$

Evaluation:

Monte carlo prediction

Monte carlo ES (Exploring Starts), for $\pi \simeq \pi_*$

Initialize:
$\pi(s) \in A(s)$, for all $s \in S$
$Q(s,a) \in R$, for all $s \in S, a \in A(s)$
Returns$(s,a) \leftarrow$ empty list, //

Loop forever (for each episode)
choose $S_0 \in S, A_0 \in A(S_0)$, randomly such that all pairs have probability > 0
Generate an episode from S_0, A_0, following $\pi: S_0, A_0, \ldots$
$G \leftarrow 0$
Loop for each step of episode, $t = T-1, T-2, \ldots, 0$
$G \leftarrow \gamma G + R_{t+1}$
Append G to Returns(S_t, A_t)
$Q(S_t, A_t) \leftarrow$ average(Returns(S_t, A_t))
$\pi(S_t) \leftarrow \underset{a}{\operatorname{argmax}}\, Q(S_t, a)$

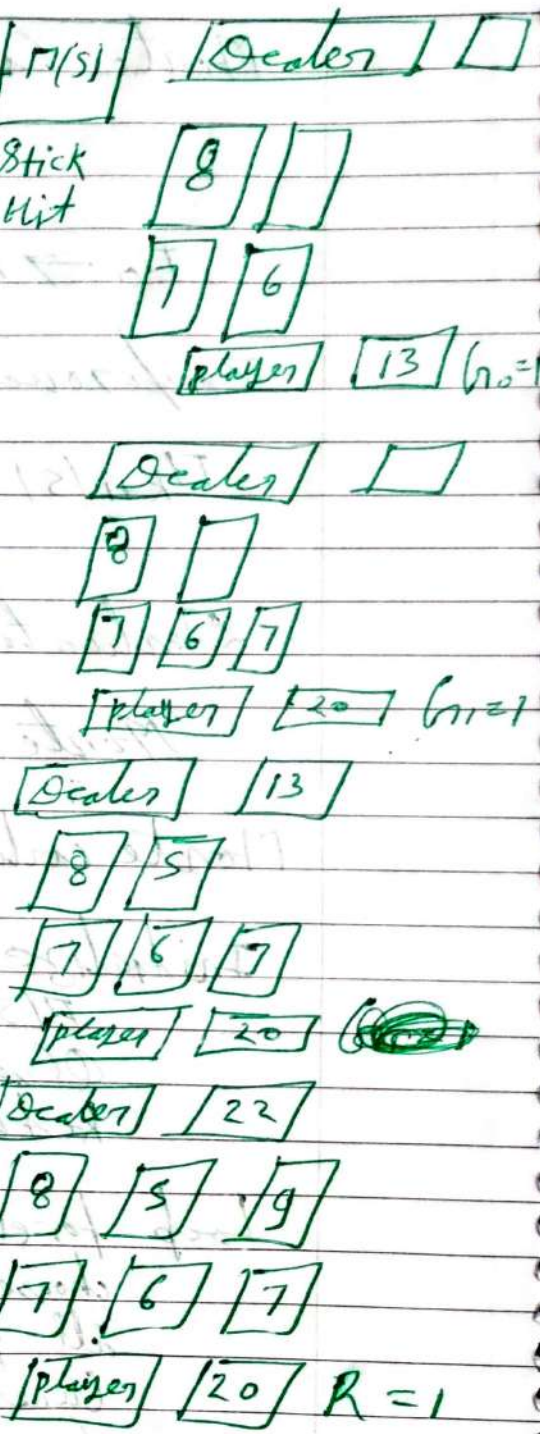
Date _ / _ / _

S, A	Returns(s,A)	Q(S,A) Hit	Q(S,A) Stick	Π(S)

X, Stick Ret(X, stick)=[] 0 1 Stick
Y, Hit Ret(Y, Hit)=[] 1 0 Hit

Y = (Noace, 13, 8)

X = (NoAce, 20, 8)



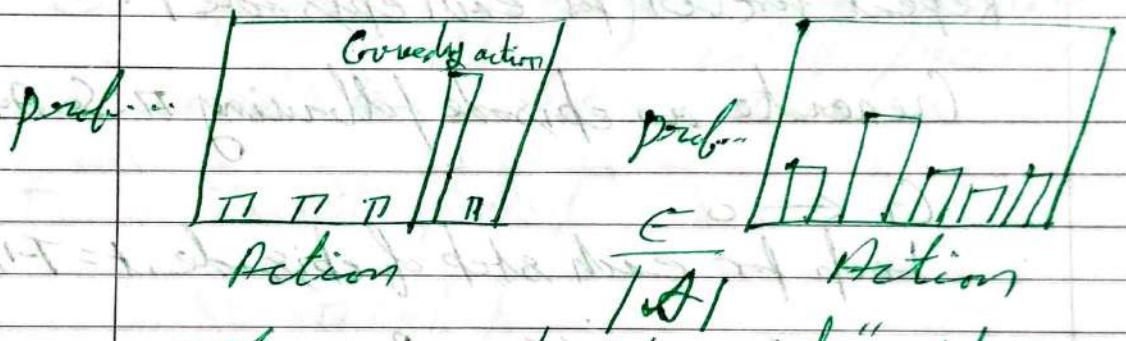
Date _/_/_

Epsilon-Soft policies

ϵ greedy policies are a subset of a larger class of policies called ϵ soft policies. ϵ soft policies take each action with prob at least ϵ over the number of actions.

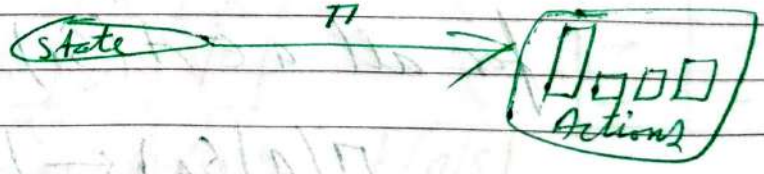
ϵ-Greedy Exploration

ϵ-Greedy policies $\subset$ ϵ-soft policies



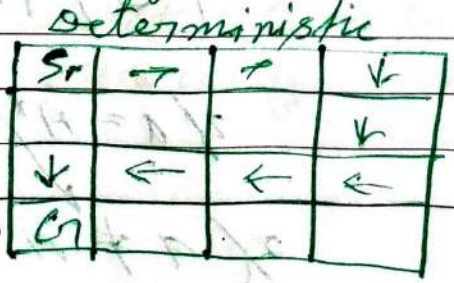
Thus we don't need "Exploring starts"

ϵ-soft policies are always stochastic

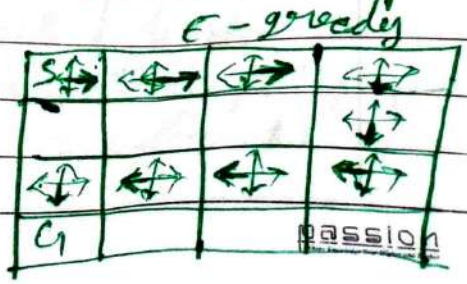


ϵ-greedy policies and deterministic policies

Deterministic



ϵ-greedy



Date _/_/_

* ϵ-soft policies may not be optimal

MC control (for ϵ-soft policies), $\pi \approx \pi_*$

Algo parameter: small $\epsilon > 0$

Initialize

$\pi \leftarrow$ ϵ-soft policy

$Q(s,a) \in \mathbb{R}$, for all $s \in S, a \in A(s)$

Returns$(s,a) \leftarrow$ empty list

Repeat forever (for each episode):

Generate an episode following π: $S_0, A_0, \ldots$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots$

$G \leftarrow \gamma G + R_{t+1}$

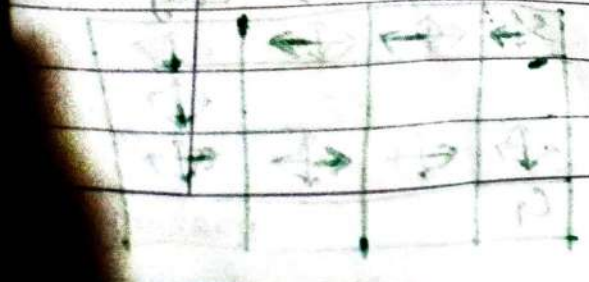
Append G to Returns(S_t, A_t)

$Q(S_t, A_t) \leftarrow$ average (Returns(S_t, A_t))

$A^* \leftarrow \underset{a}{\text{argmax}}\, Q(S_t, a)$

for all $a \in A(S_t)$:

$\pi(a|S_t) \leftarrow \begin{cases} 1-\epsilon+\epsilon/|A(S_t)| & \text{if } a = A^* \\ \epsilon/|A(S_t)| & \text{if } a \neq A^* \end{cases}$



Date _/_/_

ϵ-Soft policy

→ Values based on suboptimal policy
→ Actions " " " "

* On-Policy & Off-policy

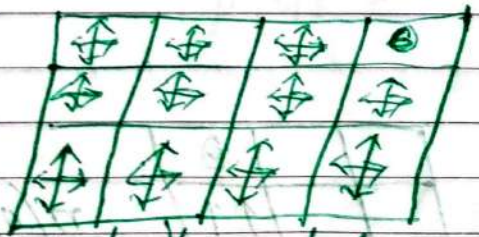
⊙ On-Policy : Improve and evaluate the policy being used to select actions.

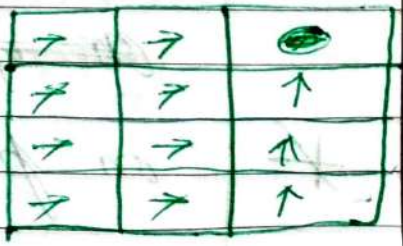
⊙ Off-policy : Improve and evaluate a different policy from the one used to select actions.

* Target policy

$\pi(a|s)$

→ learns values for this policy
→ for example the optimal "





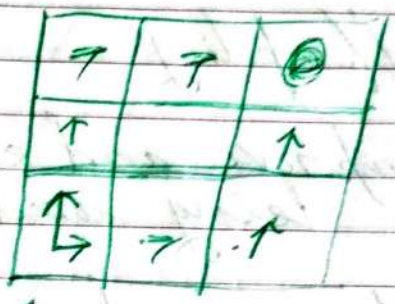
* Behavior policy $b(a|s)$

→ Select actions from this policy
→ Generally an exploratory policy (uniform random ")

Date _/_/_

* One key rule of off policy learning is that the behavior policy must cover the target policy.

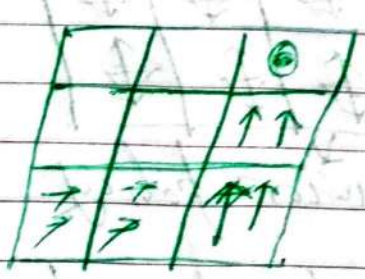
$\pi(a|s) > 0$ where $b(a|s) > 0$



consider this state with the behavior policy always goes up but the target policy goes to right agent can not learn the correct action value for that state bcz never observed samples of what would happen if it goes right

~~Off~~

* on policy



$\pi(a|s) = b(a|s)$

Date _/_/_

+ Importance sampling uses samples from one probability distribution to estimate the expectation of a different distribution.

+ Derivation of importance sampling

Sample: $x \sim b$

Estimate: $E_\pi[X]$

$$E_\pi[X] \doteq \sum_{x \in X} x\,\pi(x)$$

$$= \sum_{x \in X} x\,\pi(x)\,\frac{b(x)}{b(x)}$$

$$= \sum_{x \in X} x\,\rho(x)\,b(x)$$

$\rho(x) = \frac{\pi(x)}{b(x)}$ importance sampling ratio

$$= E_b[X\rho(X)]$$

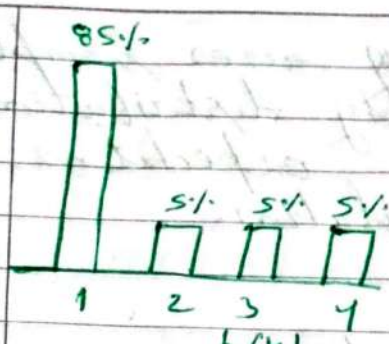
$$E[X] \approx \frac{1}{n}\sum_{i=1}^{n} x_i$$

$$\approx \frac{1}{n}\sum_{i=1}^{n} x_i\,\rho(x_i)$$

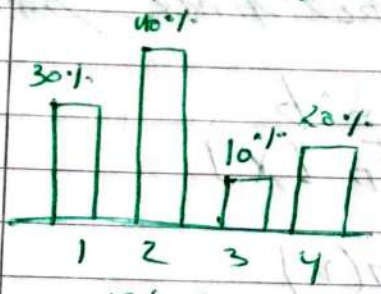
$x_1 \sim b$

passion

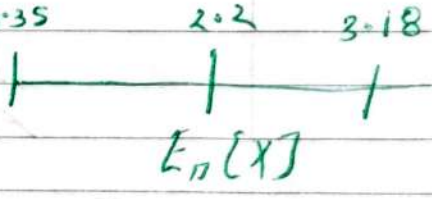
Date _ / _ / _



→ n = 1 → X = [1] → X = [1,3] → [1,3,1]
b(4) = .85
n = 3
b(4) = .05



→ π(1) = .3 →
p(1) = .1



$$\frac{1}{n}\sum_{n} \frac{\pi(n)}{b(n)} \rightarrow 1 \times \frac{.3}{.85} = 0.35$$

$$\frac{\left(1 \times \frac{.3}{.85}\right) + \left(3 \times \frac{.1}{.05}\right)}{2} = 3.18$$

$$\frac{\left(1 \times \frac{.3}{.85}\right) + \left(3 \times \frac{.1}{.05}\right) + \left(1 \times \frac{.3}{.85}\right)}{2} = 2.24$$

* Off-Policy Monte Carlo

$$\rho = \frac{P(\text{trajectory under } \pi)}{P(\text{trajectory under } b)}$$

Date _/_/_

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

Off-policy Trajectories

$$P(A_t, S_{t+1}, A_{t+1}, S_{t+2}, \dots, S_T | S_t, A_{t:T}) =$$

$$b(A_t|S_t)\,P(S_{t+1}|S_t, A_t)\; b(A_{t+1}|S_{t+1})\,P(S_{t+2}|S_{t+1}, A_{t+1}) \cdots P(S_T|S_{T-1}, A_{T-1})$$

$$\prod_{k=t}^{T-1} b(A_k|S_k)\,P(S_{k+1}|S_k, A_k)$$

$$P(\text{trajectory under } b)$$

$$\rho_{t:T-1} \doteq \frac{P(\text{traj under } \pi)}{P(\text{traj under } b)}$$

$$\doteq \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)\,P(S_{k+1}|S_k, A_k)}{b(A_k|S_k)\,P(S_{k+1}|S_k, A_k)}$$

Date _ / _ / _

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

$$\mathbb{E}_b[\rho_{t:T-1} G_t | S_t = s] = v_\pi(s)$$

Off-policy MC prediction, for $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

$V(s) \in \mathbb{R}$, arbitrarily, for all $s \in S$

Returns(s) ← an empty list,))

Loop forever (for each episode):

Generate an episode following b $S_0, A_0, R_1, S_1 \ldots$

$G \leftarrow 0 \quad W \leftarrow 1$

Loop for each step of episode, $t = T-1, \ldots 0$

$G \leftarrow \gamma W G + R_{t+1}$

Append G to Returns (S_t)

$V(S_t) \leftarrow$ average (Returns(S_t))

$W \leftarrow W \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$

Date _/_/_

Computing $\rho_{t:T-1}$ incrementally

$$\rho_{t:T-1} \doteq \rho_t \rho_{t+1} \rho_{t+2} \cdots \rho_{h-2} \rho_{h-1}$$

$$W_1 \leftarrow \rho_{t-1}$$

$$W_2 \leftarrow \rho_{t-1} \rho_{t-2}$$

$$W_3 \leftarrow \rho_{T-1} \rho_{T-2} \rho_{T-3}$$

$$W_{t+1} \leftarrow W_t \rho_t$$

---

* Temporal Difference →

$$V(S_t) \leftarrow V(S_t) + \alpha [G_t - V(S_t)]$$

TD can update its value estimates on each step of the episode. It does not have to wait for the episode to complete. It just has to remember the previous state.

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

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

$$V_\pi(s) = R_{t+1} + \gamma V_\pi (S_{t+1})$$

$$V(S_t) \leftarrow V(S_t) + \alpha [R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

passion

Date _/_/_

Tabular TD(0) for estimating V_π

Input: the policy π to be evaluated

Algo params: step size $\alpha \in (0,1]$

Initialize $V(s)$, for all $s \in S^+$, arbitrarily except that $V(\text{terminal}) = 0$

Loop for each episode:

Initialize S

Loop for each step of episode

$A \leftarrow$ action given by π for S

Take action A, observe R, S'

$V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$

$S \leftarrow S'$

until S is terminal

* Temporal Difference learning - a way to incrementally estimate the return through bootstrapping.

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

Date _/_/_

leave	exit highway	exit secondary road	enter home street	arrive home
30	35	15	10	3
	5	15	10	3

$\alpha = 1$

$\gamma = 1$

$G_0 = 5 + 15 + 10 + 10 + 3 = 43$

$V(\text{leave}) \leftarrow V(\text{leave}) + \alpha [G_0 - V(\text{leave})]$

$V(\text{leave}) \leftarrow 30 + 1 [43 - 30]$

(43) — 5 —

$G_1 = 15 + 10 + 10 + 3 = 38$

$V(\text{exit}) \leftarrow V(\text{exit}) + \alpha [G_1 - V(\text{exit})]$

$V(\text{exit}) \leftarrow 35 + 1 [38 - 35]$

(43) — 5 — (38) —

$G_2 = 10 + 10 + 3 = 23$

$V(\text{exit highway}) \leftarrow V(\text{exit highway}) + \alpha [G_2 - V(\text{exit highway})]$

$V(\text{exit highway}) \leftarrow 15 + 1 [23 - 15]$

(43) — 5 — (38) — 15 — (23)

$G_3 = 10 + 3 = 13$

$V(\text{secondary road}) \leftarrow V(\text{secondary road}) + \alpha [G_3 - V(\text{secondary road})]$

$V(\text{secondary road}) \leftarrow 10 + 1 [13 - 10]$

(43) — 5 — (38) — 15 — (23) — 10 — (13)

$G_4 = 3$

$V(\text{enter home st}) \leftarrow V(\text{enter home}) + \alpha [G_4 - V(\text{enter home})]$

$V(\text{enter home st}) \leftarrow 3 + 1 [3 - 3]$

(43) — 5 — (38) — 15 — (23) — 10 — (13) — 10 — (3) — 3 —

must wait until the end of the episode before learning can begin.

Date _/_/_

leave | exit

(30) —5— (35)

$$V(\text{leave}) \leftarrow V(\text{leave}) + \alpha[R_{30} + \gamma V(\text{exit})_{35} - V(\text{leave})_{30}]$$

(40) —5— (35) —15— (15)

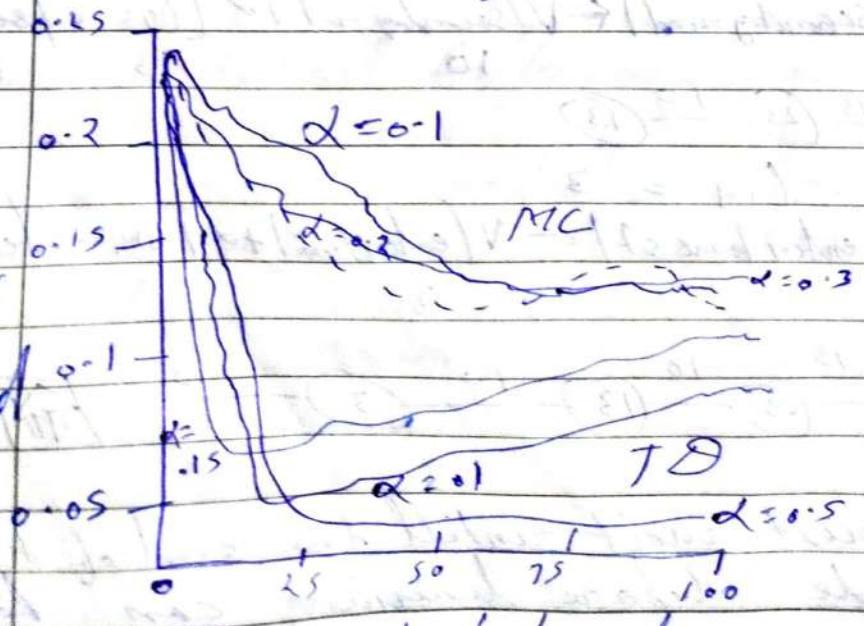
$$V(\text{exit}) \leftarrow V(\text{exit})_{35} + \alpha[R_{15} + \gamma V(\text{exit})_{15} - V(\text{exit})_{35}]$$

(40) —5— (30) —15— (20) —10— (13) —10— (3) —3— [TH]

We can learn online without waiting for the episode to end.

* Advantages of TD methods

(1) Don't require a model of the environment
(2) Online and incremental.
(3) ~~Conv~~ Converge faster than MC Methods



TD learns more slowly but achieves lower final error

passion

Date _/_/_

SARSA :-

From state-values to action-values

state-to-state

$V(s)$ [| | | |]

state-action to state-action

$Q(s,a)$ [| | | | | | | | |]

The Sarsa algorithm

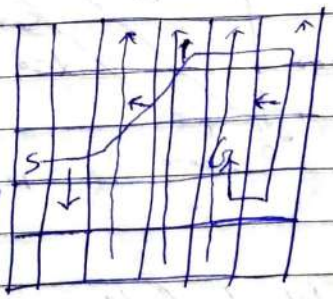
$S_t \; A_t \; R_{t+1} \; S_{t+1} \; A_{t+1}$

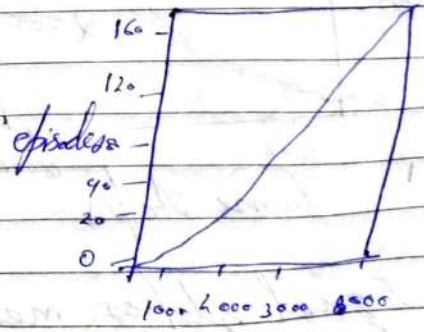
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right)$$

In Sarsa, the agent needs to know its next state action pair before updating its value estimates. That means it has to commit to its next action before the update.

An epsilon of 0.1 means we take a random action 1 in every 10 steps.

The windy Gridworld





Sarsa :

ϵ 0.1

α 0.5

Date _ / _ / _

Q - learning

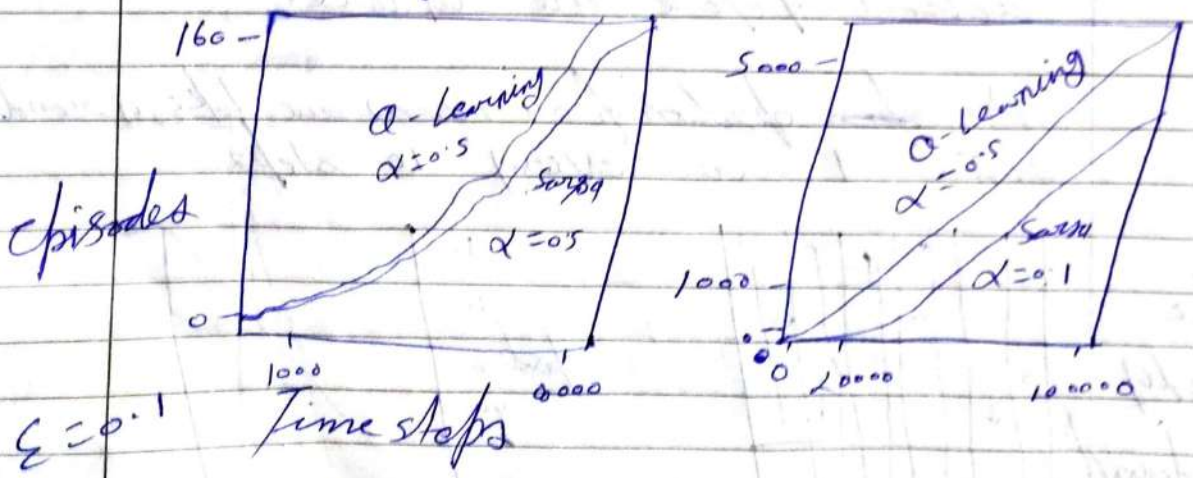
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha (R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t))$$

Sarsa is sample-based version of policy iteration which uses Bellman equations for action values, that each depend on a fixed policy.

Q learning is a ~~sample~~ sample-based version of value iteration which iteratively applies the Bellman optimality equation.

Sarsa - Policy Iteration

Q learning - Value Iteration

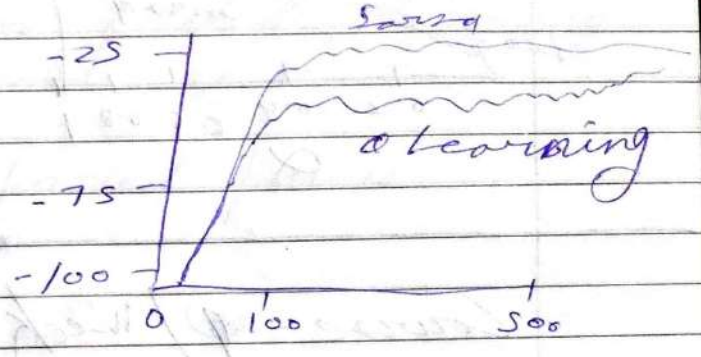


Equal slopes mean that both agents are completing episodes at the same time.

Sarsa learns same policy as Q-learning, but more slowly

Date _/_/_

The cliff walking environment



→ Q ~~learning~~ learning is off-policy without using importance sampling.

→ learning on-policy or off-policy may perform differently in control.

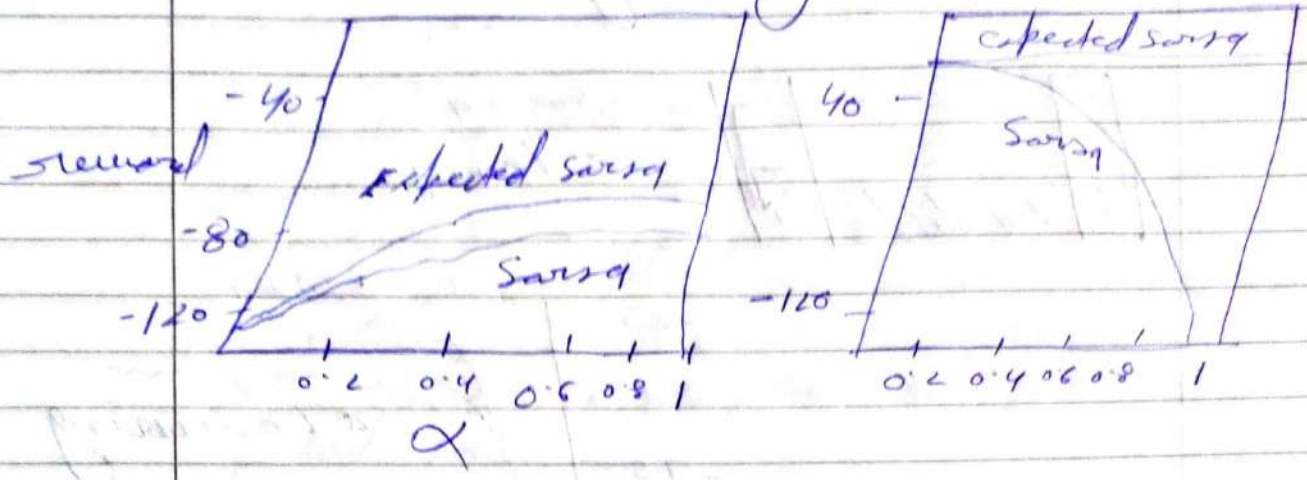
Expected Sarsa algo-

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left(R_{t+1} + \gamma \sum_{a'} \pi(a'|S_{t+1}) Q(S_{t+1}, a') - Q(S_t, A_t)\right)$$

The algo is nearly identical to Sarsa, except the TD error uses the expected estimate of the next action value instead of a sample of the next action value. That means that on every time step, the agent has to average the next state's action values according to how likely they are under the policy

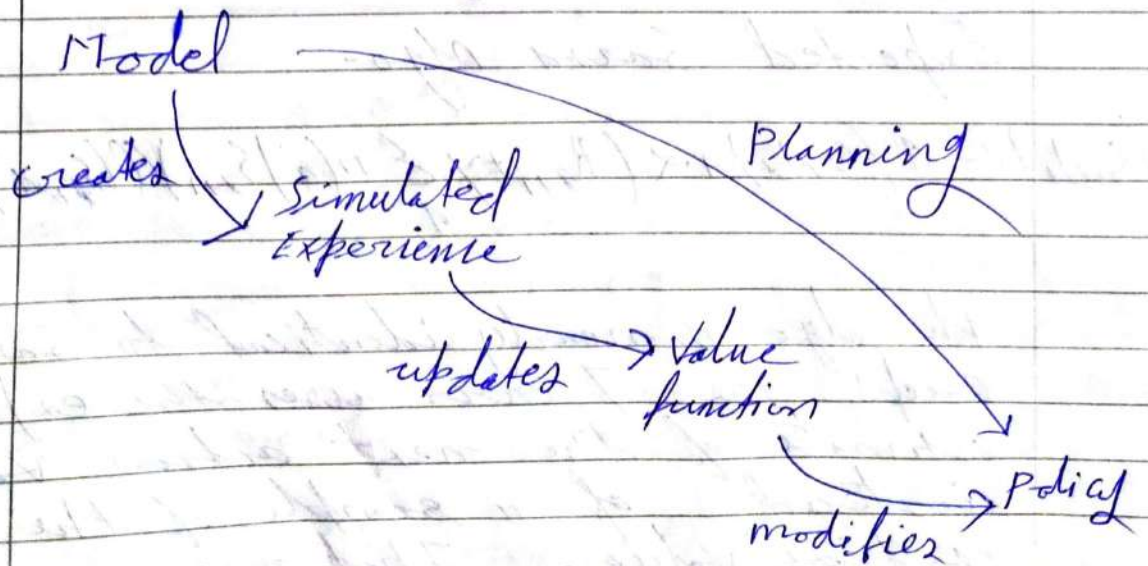
Date _/_/_

The cliff walking environment



Course - 3 / Week - 4

Planning refers to the process of using a model to improve a policy.

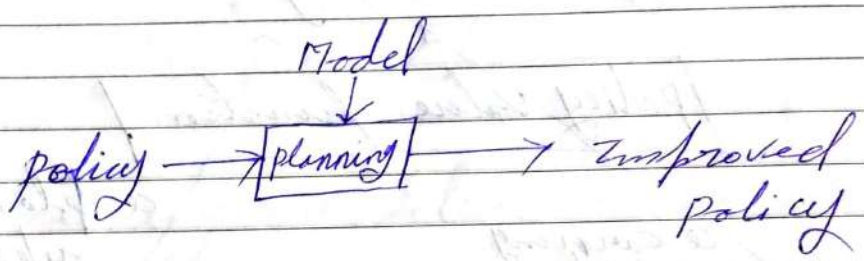


Date _/_/_

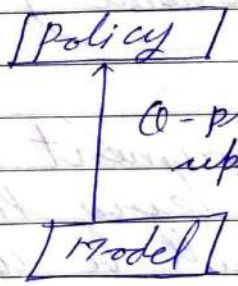
* Sample models procedurally generate samples, without explicitly storing the probability of each outcome

* Distribution models contain a list of all outcomes and their probabilities Thus require a lot of memory but gives good result.

Planning improved policies



connection with Q-learning



Random-sample one-step tabular Q-planning

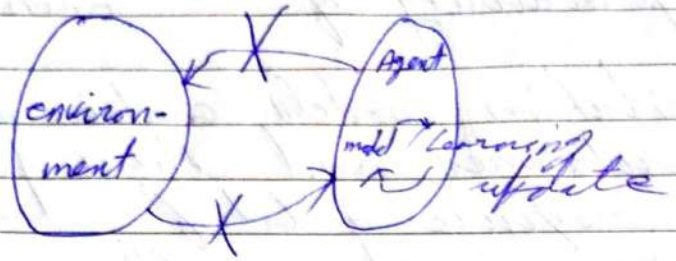
1. Sampling $S \to s \to$ [model] $\to s'$, R ; $A \to a \to$ [model] $\to R$

2. Q-learning update $Q(s,a) \leftarrow Q(s,a) + \alpha [R + \gamma \max_a Q(s',a) - Q(s,a)]$

3. Greedy policy improvement $\pi(s) = \operatorname{argmax}_a Q(s,a)$

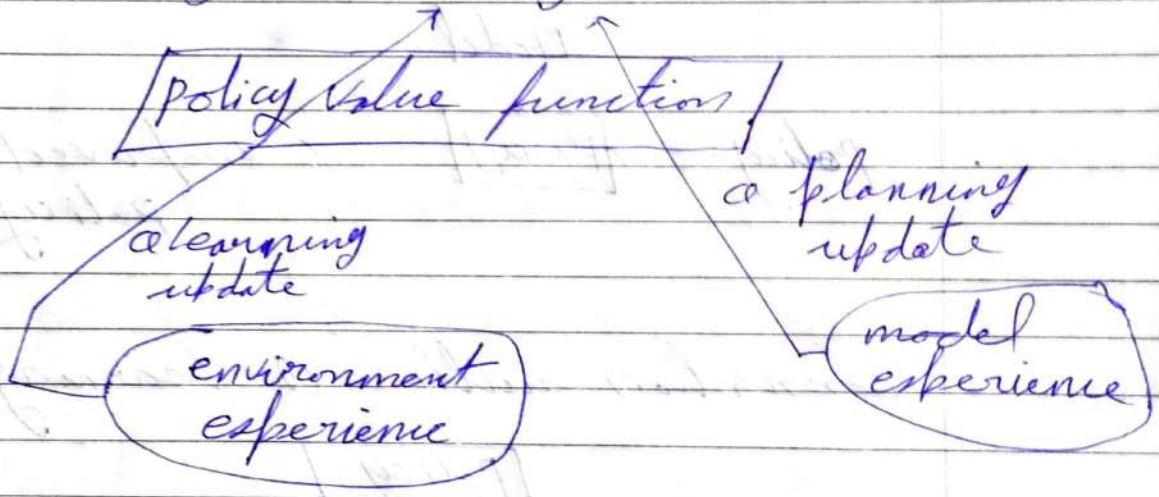
Date _/_/_

* planning only uses imagined experience



--- The Dyna Architecture ---

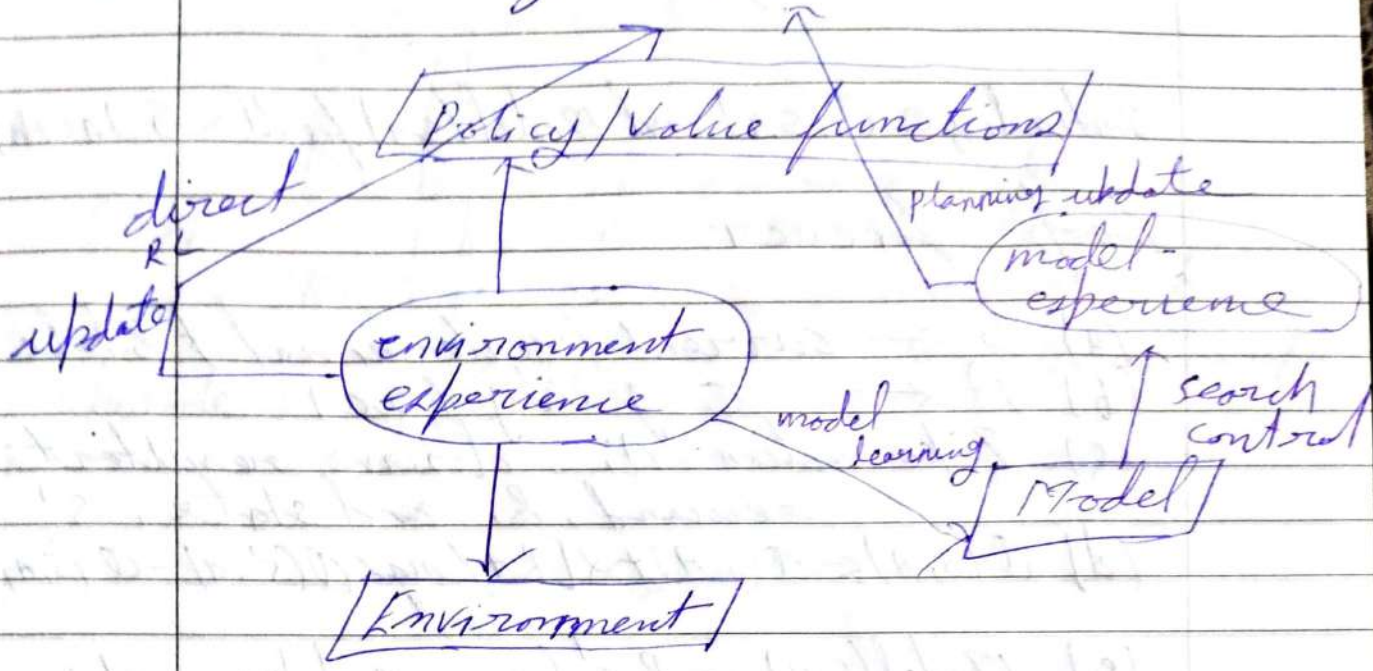
Combining Q-learning & Q-planning



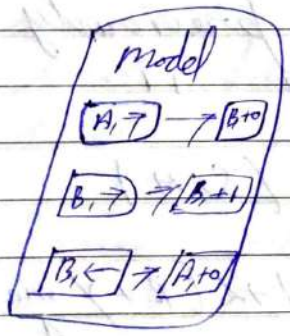
The environment experience can be used to learn the model. This model will be used to generate model experience. In addition, we want to control how the model generates this simulated experience, what states the agent will plan from. We call this process search control.

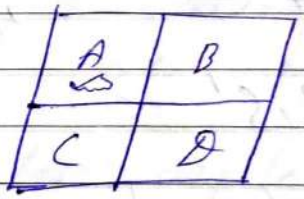
Date _/_/_

The Dyna Architecture



Learning a deterministic Model





Date _/_/_

Tabular Dyna-Q

Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in S$ & $a \in A(s)$

Loop forever:

(a) $S \leftarrow$ current (nonterminal) state

(b) $A \leftarrow \varepsilon$ greedy (S, Q)

(c) Take action A; observe resultant reward, R, and state, S'

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

(e) $Model(S,A) \leftarrow R, S'$ (assuming deterministic environment)

(f) Loop repeat n times:

$S \leftarrow$ random previously observed state

$A \leftarrow$ " action " " taken in S

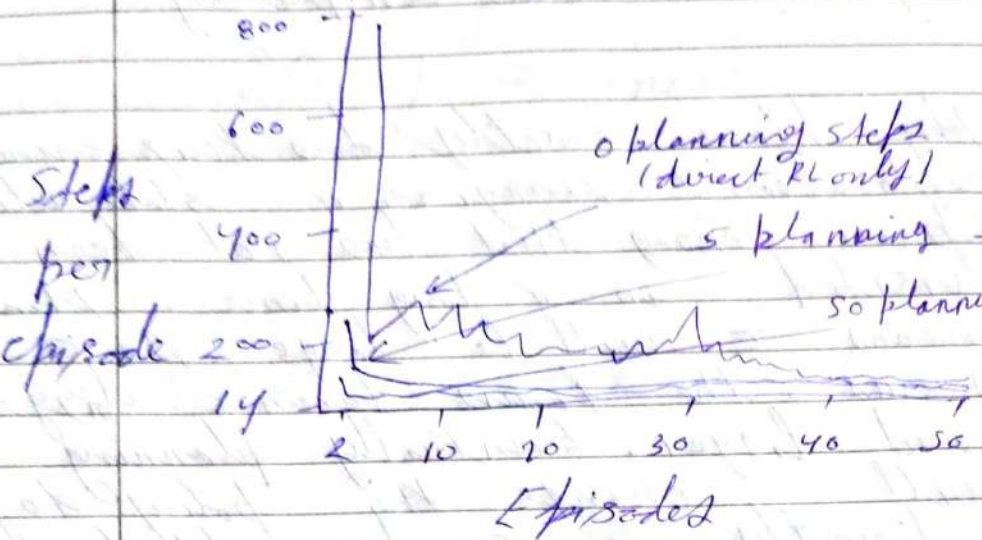
$R, S' \leftarrow Model(S,A)$

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

In this algo, search control selects a previously visited state action pair at random. It must be a state action pair the agent has seen before, otherwise the model would not know what happens next.

Date _/_/_

More planning → faster learning



Model Inaccuracy: when the environment changed, the model will be incorrect. It will remain incorrect until the agent revisits that part of the environment that changed and updates the model.

Bonus reward for exploration

$$\text{New reward} = r + k\sqrt{\tau}$$

- r: Actual reward
- k: Small constant
- τ: time steps since transition was last tried

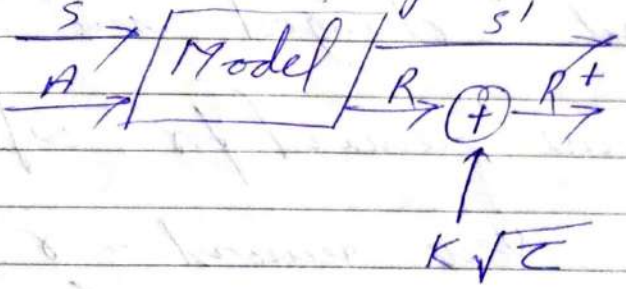
Kappa is a small constant that controls the influence of the bonus on the planning update.

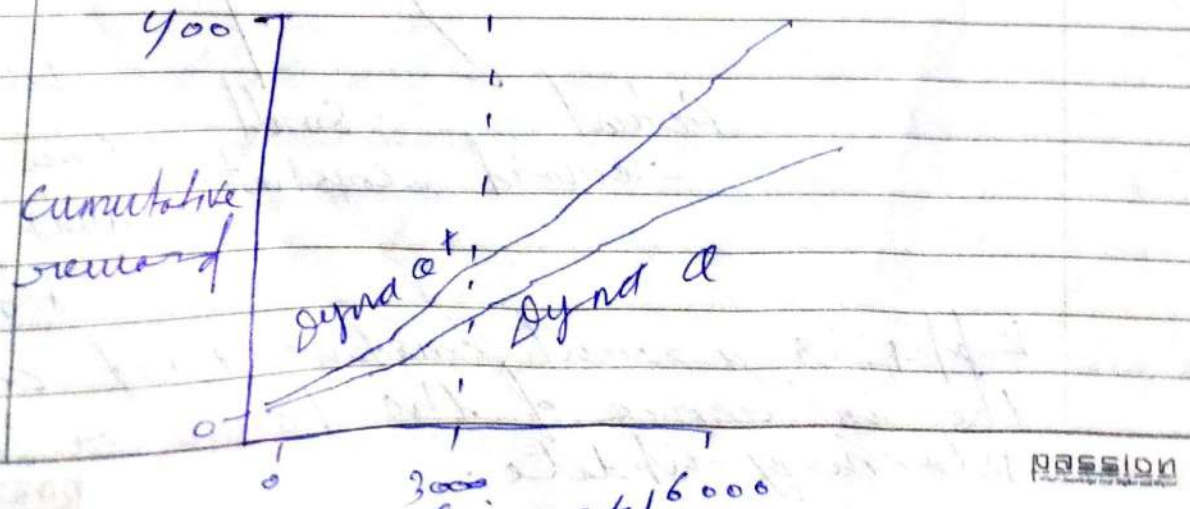
Date _/_/_

If kappa was zero, we would ignore bonus completely.

How exactly does this encourage exploration? Imagine a state action pair that has not been visited in a long time. That means τ will be large. As τ grows, the bonus becomes bigger and bigger. Eventually planning will change the policy to go directly to s due to large bonus. When the agent finally visit state s, it might see a big reward or it might be disappointed.

The Dyna-Q+ algorithm





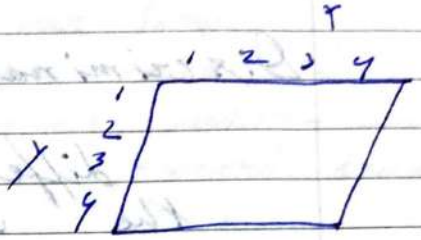
</>

Course-3, Week 1 Date _/_/_

parameterizing the value function

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

weights



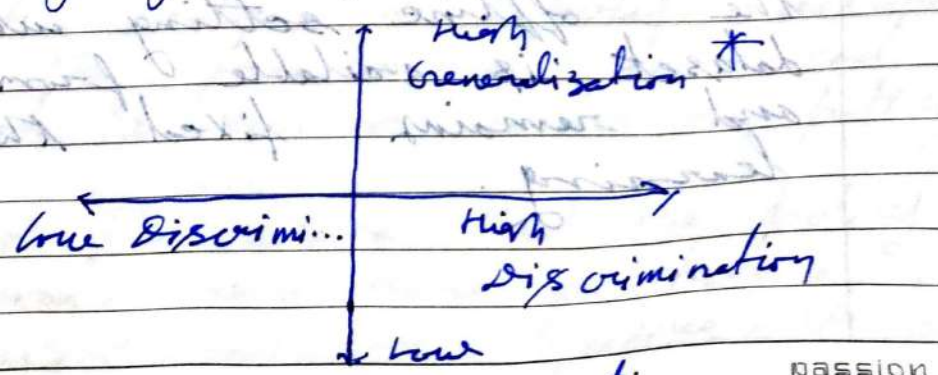
$\hat{v}(s, w) \doteq w_1 x + w_2 y$

we only have to store the 2 weights

$\hat{v}(s, w) \doteq \sum w_i x_i(s)$

This simply means that the value of each state, is computed as the sum of the weights multiplied by some fixed attributes of the state called feature.

* linear function approximation relies on having good features.



Date _/_/_

Generalization → we may not have to visit every state as much to get this values correct if we can learn its value from similar states

Discrimination → ability to make the values for two states different to distinguish b/w the values for these 2 states.

* framing policy Evaluation as Supervised learning

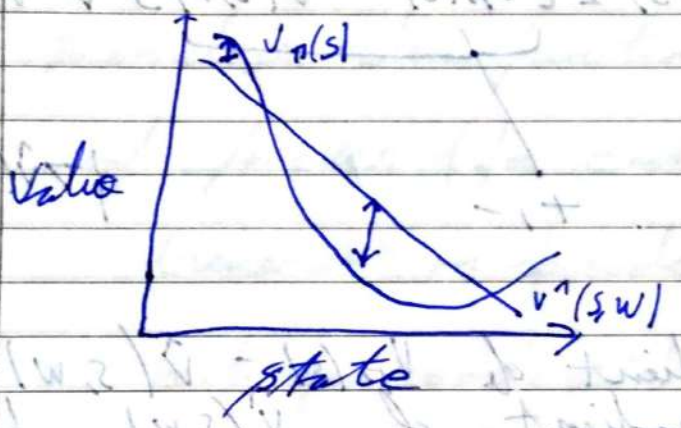
{ (S1, G1), (S2, G2), (S3, G3), ... } { (S1, R1 + γV̂(S1, w)), (S2, R2 + γV̂(S2, w)), ... } TD / Monte carlo

→ In reinforcement learning, an agent interacts with an environment and continually generates new data. This is often called the online setting. To distinguish it from the offline setting where the full dataset is available from the start and remains fixed throughout learning.

Date _/_/_

* TD methods use bootstrapping.
meaning that our target now depend on our own estimate. These estimation changes as learning progresses and so our target continually change. This is different than supervised learning where we have access to ground truth.

* The Mean Squared Value Error



$$\sum_s \mu(s)\left[V_\pi(s) - \hat{V}(s,w)\right]^2$$

$\mu(s)$ tells us how much we care about each state. A natural measure is the fraction of time each state is visited under the policy

That The states that the policy spends more time in have a higher weight in the objective. We care

Date _/_/_

less about errors in the states the policy visits less frequently.

$$\overline{VE} = \sum_s \mu(s)\left[V_\pi(s) - \hat{v}(s,w)\right]^2$$

* Gradient of the Mean Squared Value Error

$$\nabla \sum_{s \in S} \mu(s)\left[V_\pi(s) - \hat{v}(s,w)\right]^2$$

$$= -\sum_{s \in S} \mu(s)\, 2\left[V_\pi(s) - \hat{v}(s,w)\right] \nabla \hat{v}(s,w)$$

$+, -$

$\nabla \hat{v}(s,w) = x(s)$

x The gradient of $V_\pi(s) - \hat{v}(s,w)$ equals the gradient of $\hat{v}(s,w)$ because changing w does not change $V_\pi(s)$

* If the difference is +ve, it means the true value is higher than our estimate, so we should change the weights in the direction that increases our estimate. If the current error is -ve, we should change the weights in opposite direction.

Date _/_/_

for example: S_1

$$w_2 \doteq w_1 + \alpha [v_\pi(S_1) - \hat{v}(S_1, w_1)] \nabla \hat{v}(S_1, w_1)$$

Gradient Monte carlo algo :-

Input : the policy π to be evaluated

Input : a differentiable function $\hat{v} : S \times \mathbb{R}^d \to \mathbb{R}$

Algo parameter : step size $\alpha > 0$

Initialize value function weights $w \in \mathbb{R}^d$

Loop forever (for each episode) :

Generate an episode $S_0, A_0, R_1, \ldots R_T, S_T$ using π

loop for each step of episode $t = 0, 1, \ldots T-1$

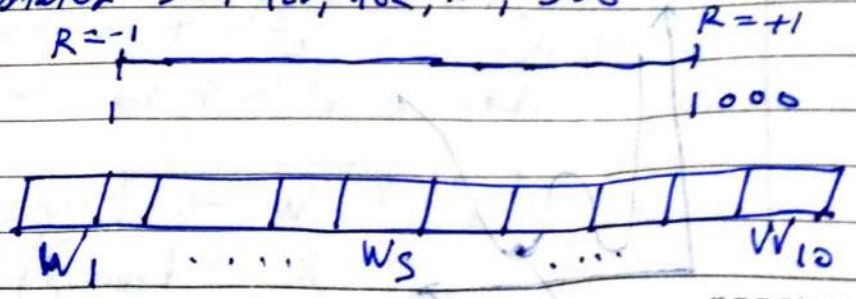
$$w \leftarrow w + \alpha [G_t - \hat{v}(S_t, w)] \nabla \hat{v}(S_t, w)$$

policy

50% $\longleftrightarrow$ 50% $\alpha = 2 * 10^{-5}$

Return 1, 1, 1, ..., 1

Visited states 500, 423, 482, ..., 936



Date _/_/_

The TD update for function approxi--

$$w \leftarrow w + \alpha [U_t - \hat{v}(S_t, w)] \nabla \hat{v}(S_t, w)$$

$$U_t \doteq R_{t+1} + \gamma \hat{v}(S_{t+1}, w).$$

$U_t \rightarrow$ biased $\rightarrow$ w may not converge to a local optimum.
because U_t (target) depends on our estimate of the value in the next state. This means our update could be biased because the estimate in our target may not be accurate.

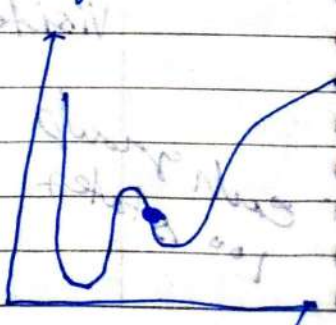
* semi-gradient TD(0)

Input: Initialize s
Loop for each episode
  Loop for each step of episode
    choose $A \sim \pi(\cdot|s)$
    Take action A, observe R, s'
    $w \leftarrow w + \alpha [R + \gamma \hat{v}(s', w) - \hat{v}(s, w)] \nabla \hat{v}(s, w)$
    $s \leftarrow s'$
  until s is terminal

Gradient Monte carlo

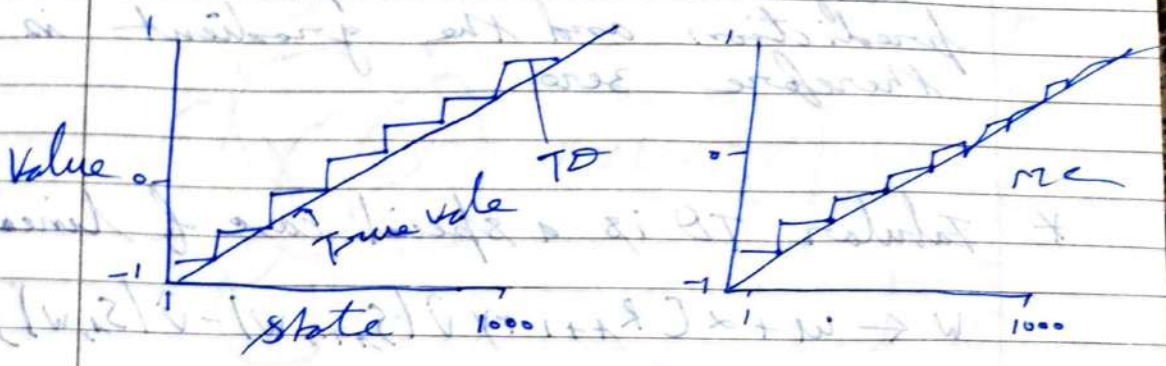
local minimum

Semigradient TD

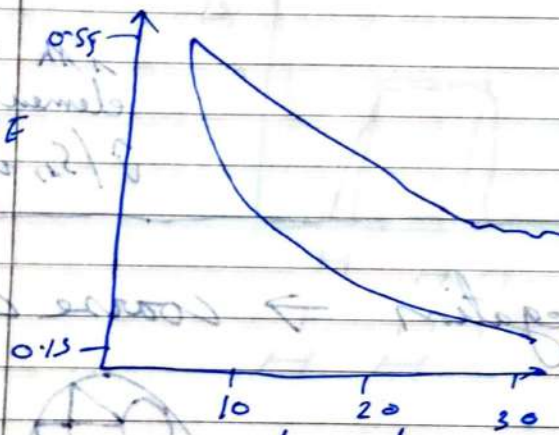


Date _/_/_

Long run performance



* MC better than TD



* TD converge faster than MC

TD update with linear function Appr...

$$w \leftarrow w + \alpha \delta_t \nabla \hat{v}(S_t, w)$$
$$w \leftarrow w + \alpha \delta_t x(S_t)$$

$$\hat{v}(S_t, w) \doteq w^T x(S_t)$$
$$\nabla \hat{v}(S_t, w) = x(S_t)$$

Date _/_/_

If the feature is zero, then that weight has no impact on the prediction and the gradient is therefore zero.

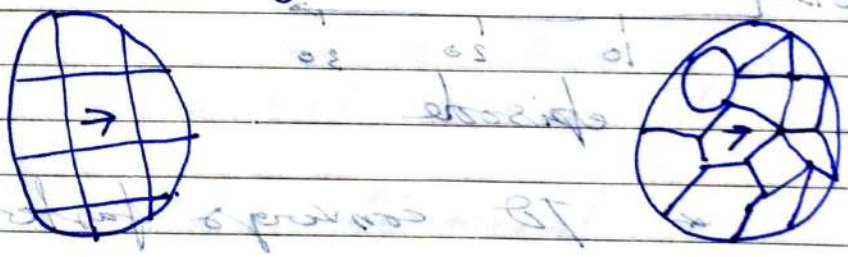
* Tabular TD is a special case of linear TD

$$w \leftarrow w + \alpha [R_{t+1} + \gamma \hat{v}(S_{t+1}, w) - \hat{v}(S_t, w)] x(S_t)$$

$$w_i \leftarrow w_i + \alpha [R_{t+1} + \gamma \hat{v}(S_{t+1}, w) - \hat{v}(S_t, w)] 1_{x(S_i)} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix} \leftarrow i\text{th element}$$

$$\hat{v}(S_t, w) = w_i$$

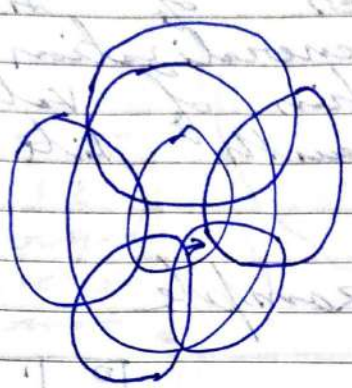
state aggregation → coarse coding



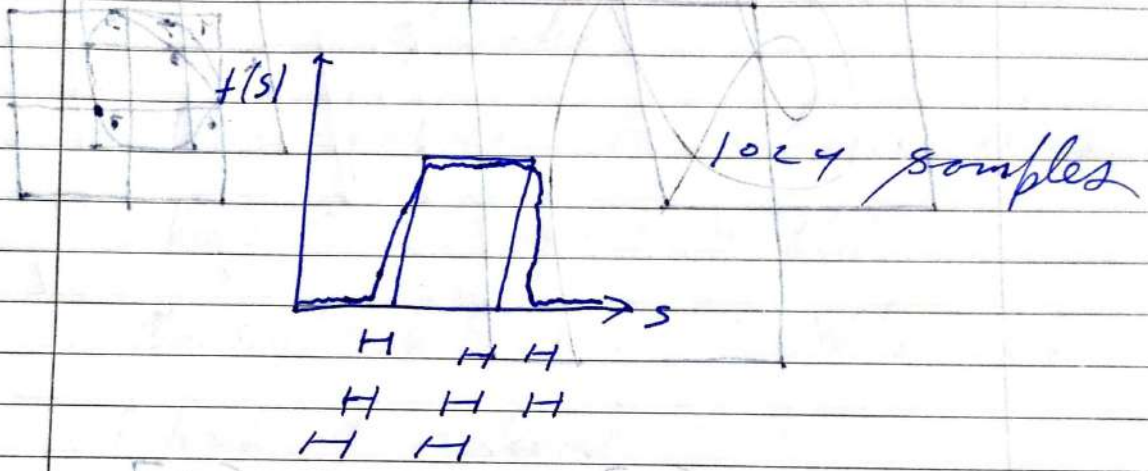
By allowing overlaps, we obtain a more flexible class of feature representations called course coding.

Date _/_/_

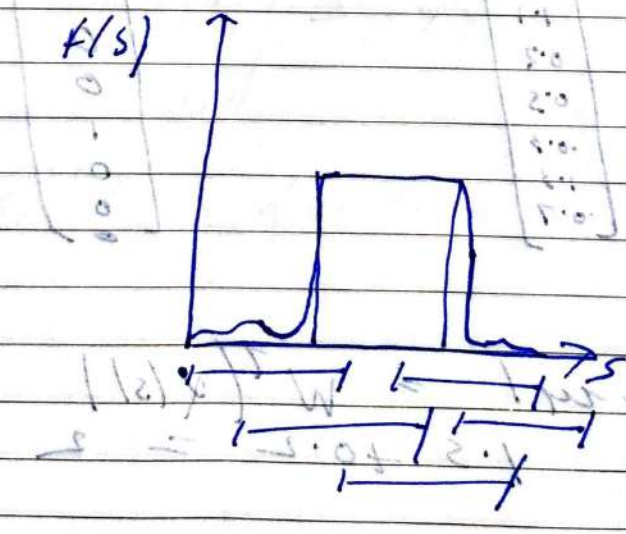
coarse coding



→ x[s] →



1024 samples

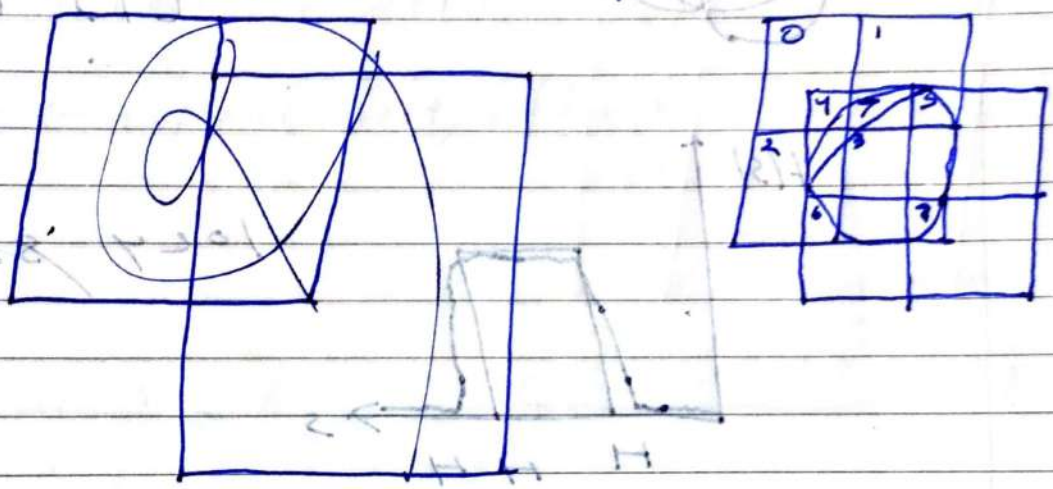


10240 samples

Date _/_/_

* The shape, size of tiles, and the number of tilings affects the generalization and discrimination of value function approximation with tile coding.

A simple example



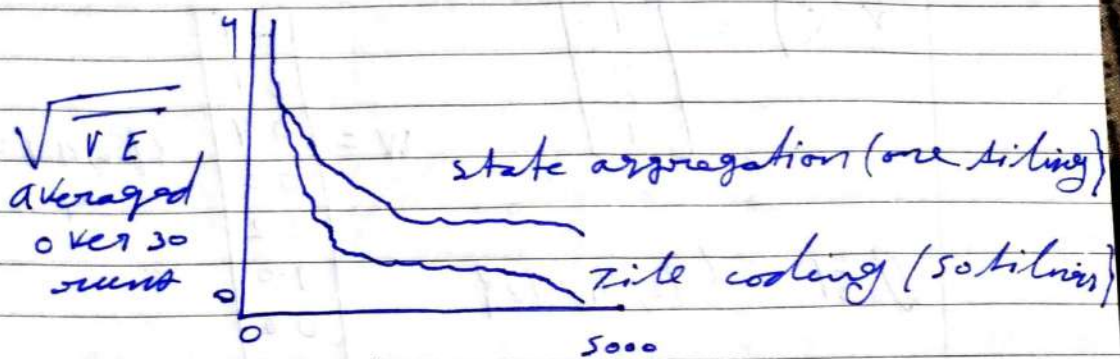
$$W = \begin{bmatrix} 0.0 \\ 1.5 \\ 1.1 \\ 0.7 \\ 0.5 \\ -0.5 \\ 1.2 \\ -0.7 \end{bmatrix} \qquad x(s) = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\hat{v}(s, w) = w^T x(s)$$

$$1.5 + 0.2 = 2$$

Date _/_/_

Tile coding vs state aggregation



State-values to action-values

$$V_\pi(s) \approx \hat{v}(s, w) \doteq w^T x(s)$$

$$q_\pi(s,a) \approx \hat{q}(s,a,w) \doteq w^T x(s,a)$$

Repeating actions

$$x(s) = \begin{bmatrix} x_0(s) \\ x_1(s) \\ x_2(s) \\ x_3(s) \end{bmatrix} \rightarrow x(s,a) = \begin{bmatrix} x_0(s) \\ x_1(s) \\ x_2(s) \\ x_3(s) \\ x_0(s) \\ x_1(s) \\ x_2(s) \\ x_3(s) \\ x_0(s) \\ x_1(s) \\ x_2(s) \\ x_3(s) \end{bmatrix} \begin{matrix} \} a_0 \\ \\ \\ \\ \} a_1 \\ \\ \\ \\ \} a_2 \\ \\ \\ \end{matrix} \quad x(s,a_0) = \begin{bmatrix} x_0(s) \\ x_1(s) \\ x_2(s) \\ x_3(s) \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\mathcal{A}(s) = \{a_0, a_1, a_2\}$$

Date _/_/_

computing action-values

$$x(s_0) = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} \qquad W = \begin{bmatrix} 0.7 \\ 0.1 \\ 0.4 \\ 0.3 \\ 2.2 \\ 1.0 \\ 0.6 \\ 1.8 \\ 1.3 \\ 1.1 \\ 0.9 \\ 0.7 \end{bmatrix} \qquad x(s_0, a_0) = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$A(s) = \{a_0, a_1, a_2\}$$

$$\hat{q}(s_0, a_0, w) = 0.7 + 0.3 = 1$$

$$\hat{q}(s_0, a_1, w) = 2.2 + 1.8 = 4$$

$$\hat{q}(s_0, a_2, w) = 1.3 + 1.7 = 3$$

Date _/_/_

* Episodic Sarsa with function approx

Episodic semi-gradient Sarsa for estimating $\hat{q} \approx q_*$

Input: a differentiable action value function parameterization $\hat{q}: S \times A \times \mathbb{R}^d \to \mathbb{R}$

Algo parameters: step size $\alpha > 0$, small $\epsilon > 0$

Initialize value function weights $w \in \mathbb{R}^d$ arbitrarily (e.g., $w = 0$)

Loop for each episode:

$S, A \leftarrow$ initial state and action of episode (eg: ϵ-greedy)

Loop for each step of episode:

Take action A, observe R, S'

If S' is terminal:

$w \leftarrow w + \alpha [R - \hat{q}(S, A, w)] \nabla \hat{q}(S, A, w)$

Go to next episode

Choose A' as a function of $\hat{q}(S', \cdot, w)$ (eg., ϵ-greedy)

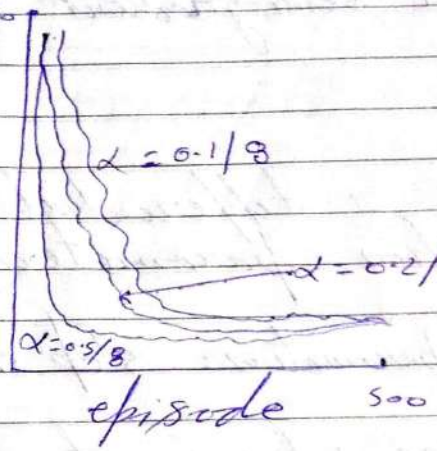
$w \leftarrow w + \alpha [R + \gamma \hat{q}(S', A', w) - \hat{q}(S, A, w)] \nabla \hat{q}(S, A, w)$

$S \leftarrow S'$

$A \leftarrow A'$

Date _/_/_

Learning curves



Expected Sarsa with function Approximation

Sarsa :

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

Expected Sarsa :

$$w \leftarrow w + \alpha \left(R_{t+1} + \gamma \sum_{a'} \pi(a' | S_{t+1}) \hat{q}(S_{t+1}, a', w) - \hat{q}(S_t, A_t, w) \right) \nabla \hat{q}(S_t, A_t, w)$$

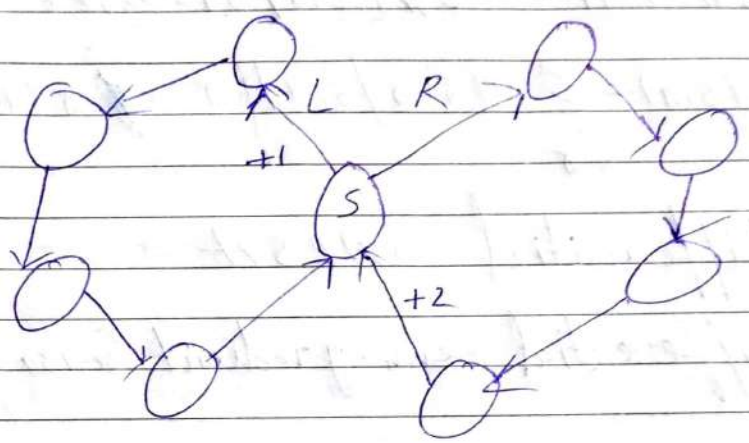
Q-learning

$$w \leftarrow w + \alpha \left(R_{t+1} + \gamma \max_{a'} \hat{q}(S_{t+1}, a', w) - \hat{q}(S_t, A_t, w) \right) \nabla \hat{q}(S_t, A_t, w)$$

Date _/_/_

The Average Reward objective

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



$r[\pi_L] = 1/5 = 0.2$

$r[\pi_R] = 2/5 = 0.4$

Returns for Average Reward

$$G_t = R_{t+1} - r(\pi) + R_{t+2} - r(\pi) + R_{t+3} - r(\pi) + \cdots$$

$r(\pi_L) = 0.2$ →

$G_t = 1 - 0.2 +$
$0 - 0.2 +$
$0 - 0.2 +$
$0 - 0.2 +$
$0 - 0.2 +$
$\vdots \quad H \to \infty$
$= 0.4$

$G_t = 0 - 0.2 +$
$0 - 0.2 +$
$0 - 0.2 +$
$0 - 0.2 +$
$2 - 0.2 +$
0.4
$= 1.4$

$r[\pi_R] = 0.4$ →

$G_t = 1 - 0.4 +$
$0 - 0.4 +$
$0 - 0.4 +$
$0 - 0.4 +$
$0 - 0.4 +$
$= -1.8$

$G_t = 0 - 0.4 +$
$0 - 0.4 +$
$0 - 0.4 +$
$0 - 0.4 +$
$2 - 0.4 +$
$H \to \infty$
$= -0.8$

Date _ / _ /

Value functions for Average Reward

$G_t = R_{t+1} - \bar{r}(\pi) + R_{t+2} - \bar{r}(\pi) + \cdots$

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

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

* Differential SARSA

Differential semi-gradient Sarsa for estimating $\hat{q} \approx q_*$

Input: $\hat{q} : S \times A \times R^d \to R$

Algo params: step sizes $\alpha, \beta > 0$

Initialize value function weights $w \in R^d$ arb

" average reward estimate $\bar{R} \in R$ "

Initialize state S, and action A

Loop for each step:

Take action A, observe R, S'

Choose A' as a function of $\hat{q}(S', \cdot, w)$

$\delta \leftarrow R - \bar{R} + \hat{q}(S', A', w) - \hat{q}(S, A, w)$

$\bar{R} \leftarrow \bar{R} + \beta\delta$

$w \leftarrow w + \alpha\delta \nabla \hat{q}(S, A, w)$

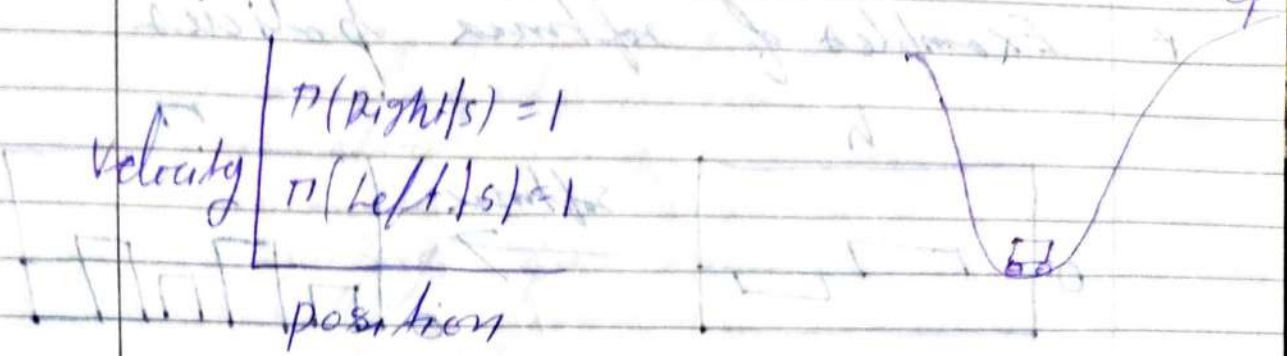
$S \leftarrow S'$

$A \leftarrow A'$

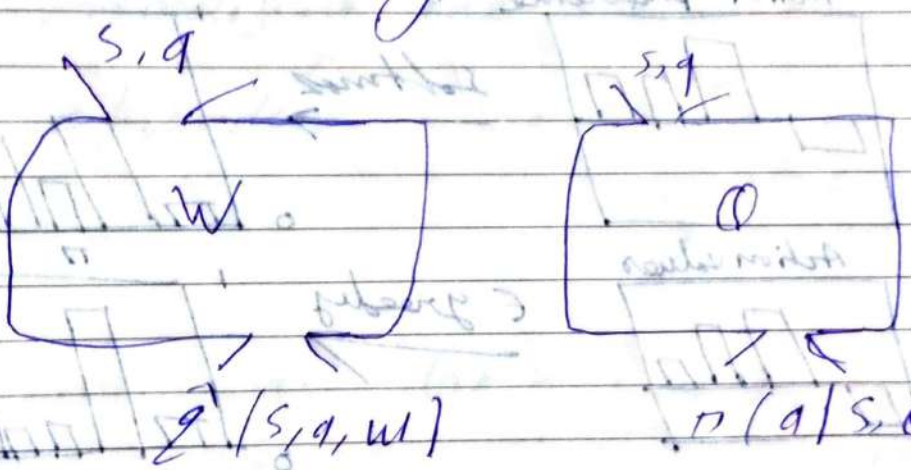
Passion

Date _/_/_

* An example of a policy without action-values



parameterizing policies directly



The Softmax policy parameterization

$$\pi(a|s,\theta) \doteq \frac{e^{h(s,a,\theta)}}{\sum_b e^{h(s,b,\theta)}}$$

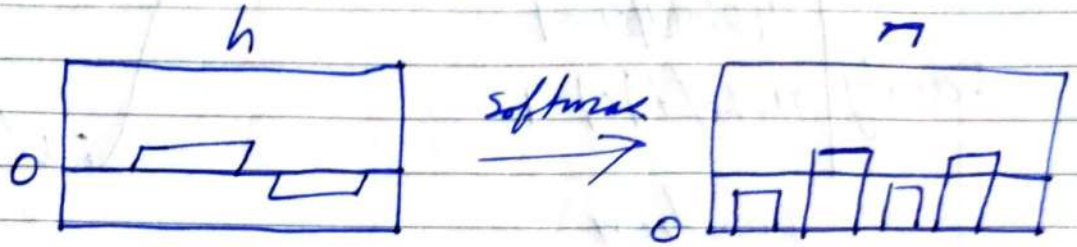
$h(s,a,\theta)$ ← action preference

$\sum_{b \in \mathcal{A}}$

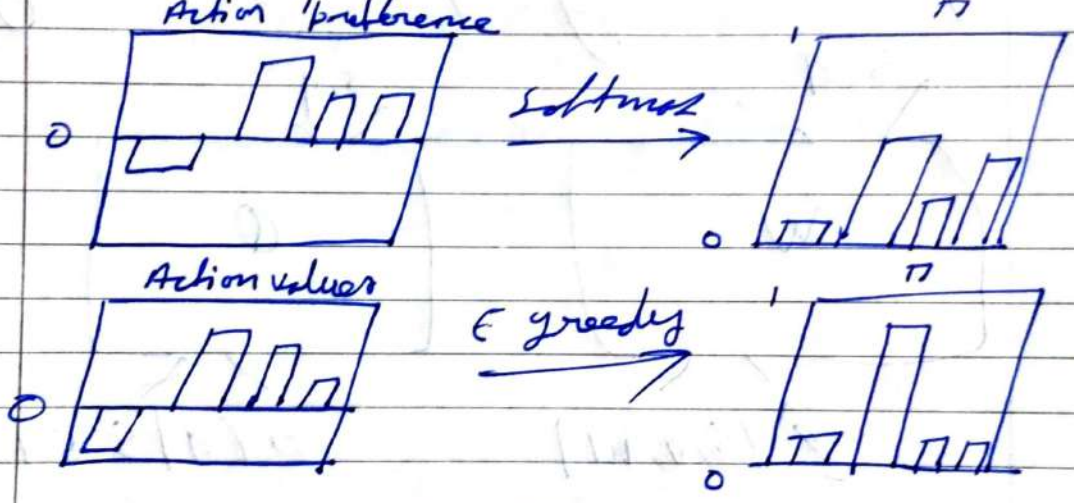
Date _/_/_

~~Transpose Convolution~~

* Examples of softmax policies



* Action preferences are not action values



* Advantages of policy parameterization

Epsilon step chooses a random action to ensure continual exploration However ε probabilities puts a cap on how good the resulting policy can be.

we cntd. of course switch to greedy policy when

Date _/_/_

exploration is done.

We can avoid this issue with parameterized policies. The policy can start off stochastic to guarantee exploration. Then as learning progresses, the policy can naturally converge towards a deterministic greedy policy.

* Formalizing the goal as an objective

Episodic: $G_t = \sum_{t=0}^{T} R_t$

Continuing: $G_t = \sum_{t=0}^{\infty} \gamma^t R_t = \sum_{t=0}^{\infty} R_t = r(\pi)$

The Average Reward objective

$$r(\pi) = \sum_s \mu(s) \sum_a \pi(a|s,\theta) \sum_{s',r} P(s',r|s,a)\, r$$

$\mathbb{E}_\pi[R_t]$ ↓ $\mathbb{E}[R_t|S_t=s, A_t=a]$

$\mathbb{E}_\pi[R_t|S_t=s]$

Date _ / _ / _

$$\nabla \gamma(\pi) = \nabla \sum_s \mu(s) \sum_a \pi(a|s,\theta) \sum_{s',\gamma} p(s',\gamma|s,a)\gamma$$

s ↑ a s',γ

depends on θ

$$\nabla \gamma(\pi) = \sum_s \mu(s) \sum_a \nabla \pi(a|s,\theta) q_\pi(s,a)$$

* Unbiasedness of the stochastic samples

$$\nabla \gamma(\pi) = E_\mu \left[\sum_a \nabla \pi(a|s,\theta) q_\pi(s,a) \right]$$

$S_0, A_0, R_1, S_1, A_1, \cdots S_t, A_t, R_{t+1}, \cdots$

Getting stochastic samples with one action

$$\sum_a \nabla \pi(a|s,\theta) q_\pi(s,a)$$

$$= \sum_a \pi(a|s,\theta) \frac{1}{\pi(a|s,\theta)} \nabla \pi(a|s,\theta) q_\pi(s,a)$$

$$= E_\pi \left[\frac{\nabla \pi(A|s,\theta)}{\pi(A|s,\theta)} q_\pi(s,A) \right]$$

passion

Stochastic Gradient Ascent for policy parameters

Date _/_/_

$$\theta_{t+1} \doteq \theta_t + \alpha \frac{\nabla \pi(A_t|S_t,\theta_t)\, q_\pi(S_t,A_t)}{\pi(A_t|S_t,\theta_t)}$$

$$\theta_{t+1} \doteq \theta_t + \alpha \nabla \ln \pi(A_t|S_t,\theta_t)\, q_\pi(S_t,A_t)$$

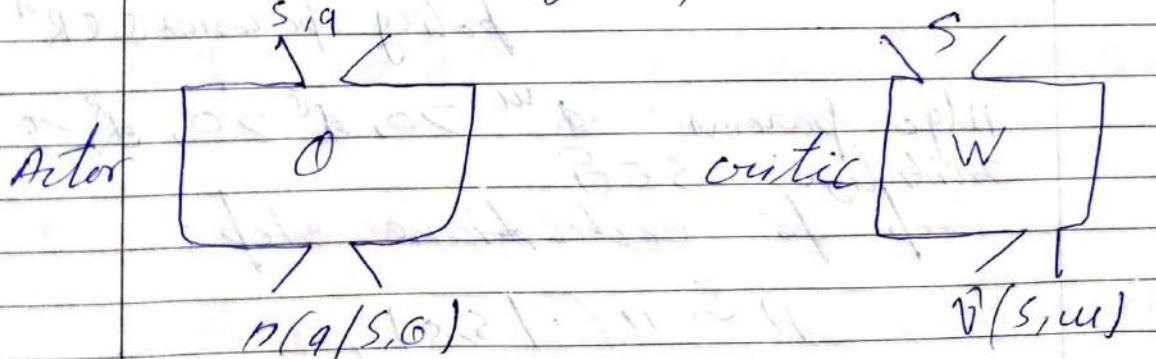
$$\nabla \log(f(x)) = \frac{\nabla f(x)}{f(x)}$$

~~Approximating~~

* Actor - critic

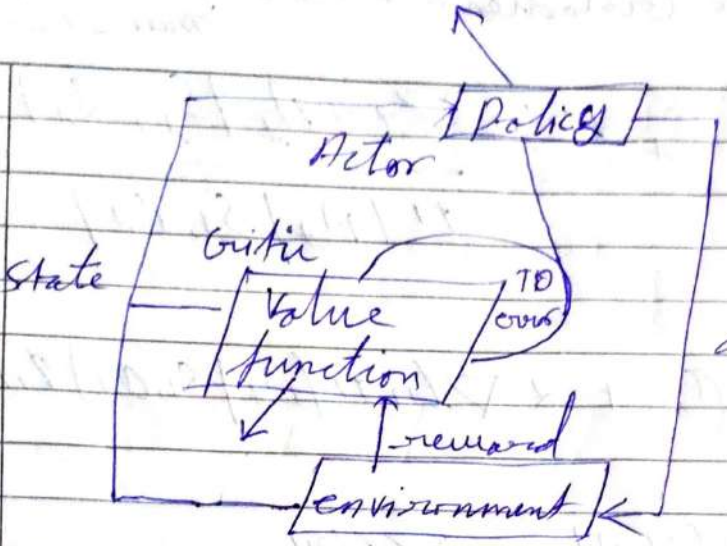
Approximating the action value in policy update

$$\theta_{t+1} \doteq \theta_t + \alpha \nabla \log \pi(A_t|S_t,\theta_t)\left[R_{t+1} - \bar{R} + \hat{v}(S_{t+1},w)\right]$$



* The actor is continually changing the policy to exceed the critics expectation; and the critic is constantly updating its value function to evaluate the actor changing policy.

Date _/_/_



Actor-Critic (continuing), for estimating $\pi_\theta \approx \pi_*$

Input: a differentiable policy parametrization $\pi(a|s,\theta)$

Input: a state-value function parameterization $\hat{v}(s,w)$

Initialize $\bar{R} \in R$ to 0

Initialize state-value weights $w \in R^d$ & policy params $\theta \in R^d$

Algo params: $\alpha^w > 0$, $\alpha^\theta > 0$, $\alpha^R > 0$

Initialize $S \in \mathcal{S}$

Loop for each time step:

$A \sim \pi(\cdot|S,\theta)$

Take action A, observe S', R

$\delta \leftarrow R - \bar{R} + \hat{v}(S',w) - \hat{v}(S,w)$

$\bar{R} \leftarrow \bar{R} + \alpha^R \delta$

$w \leftarrow w + \alpha^w \delta \nabla \hat{v}(S,w)$

$\theta \leftarrow \theta + \alpha^\theta \delta \nabla \log \pi(A|S,\theta)$

$S \leftarrow S'$

Date _/_/_

* Actor-Critic with Softmax policies

$$\theta \leftarrow \theta + \alpha^{\theta} \delta \nabla \log \pi(A|S,\theta)$$

$$\pi(a|S,\theta) \doteq \frac{e^{h(S,a,\theta)}}{\sum_{b \in A} e^{h(S,b,\theta)}}$$

$$\hat{v}(S,w) \doteq w^T x(S)$$

$$h(S,a,\theta) \doteq \theta^T x_h(S,a)$$

$$w \leftarrow w + \alpha^{w} \delta \nabla \hat{v}(S,w)$$

$$\nabla \hat{v}(S,w) = x(S)$$

$$\theta \leftarrow \theta + \alpha^{\theta} \delta \nabla \log \pi(A|S,\theta)$$

$$w \leftarrow w + \alpha^{w} \delta x(S)$$

$$\nabla \log \pi(a|S,\theta) = x_h(S,A) - \sum_b \pi(b|S,\theta) x_h(S,b)$$

* Gaussian policy

$$\pi(a|S,\theta) \doteq \frac{1}{\sigma(S,\theta)\sqrt{2\pi}} \exp\left(-\frac{(a-\mu(S,\theta))^2}{2\sigma(S,\theta)^2}\right)$$

$$\mu(S,\theta) \doteq \theta_\mu^T x(S)$$

$$\sigma(S,\theta) \doteq \exp(\theta_\sigma^T x(S))$$

passion

Date _/_/_

$$\theta = \begin{bmatrix} \theta_\mu \\ \theta_\sigma \end{bmatrix}$$

* Gradient of the log of the Gaussian Policy

$$\nabla \log \pi(a|s,\theta_\mu) = \frac{1}{\sigma(s,\theta)^2}(a - \mu(s,\theta))x(s)$$

$$= \left(\frac{(a - \mu(s,\theta))^2}{\sigma(s,\theta)^2} - 1 \right) x(s)$$

* Advantages of Continuous Actions

(1) It might not be straightforward to choose a proper discrete set of actions

(2) Continuous action allows us to generalize over action

For example, the gaussian policies smoothly assigns probability density to nearby actions. Imagine an action is found to be good and the update increases density for that action. The density for a nearby action will also increase with the agent generalizing that those actions are likely also to be good.