28 July.

REINFORCEMENT LEARNING __/__/__

- Sequential Decision Making - RL.

- Agent & Environment:
The agent interacts with an external environment.
Agent has an internal state z_t, which it passes to its policy π to make a decision action
$a_t = \pi(z_t)$.
Policy remains same, updating with each state.

Note: Tic Tac Toe - number of possible states, including x, o, & empty - including valid & invalid states - 19,683.

The environment responds with an observation o_{t+1}, which is used by the agent with the state update function $z_{t+1} = SU(z_t, a_t, o_{t+1})$.

Reward: Immediate gain.
Goal: Maximize reward - get long-term reward.

- Maximum expected utility principle:

$$V_\pi(s_0) = \mathbb{E}_p(a_0, s_1, a_1, \ldots, a_T, s_T | s_0, \pi)\left[\sum_{t=0}^{T} R(s_t, a_t) | s_0\right].$$

where s_0 - agent's initial state.
$R(s_t, a_t)$ - reward function.
$V_\pi(s_0)$ - value function for policy π evaluated at s_0.

Qn. Draw a markov chain which represents a reinforcement learning.

__/__/____

29 July.

- State, Action, and Reward:
Loops these 3.

Policy function π:
Decisions are taken based on the current state.

$$\pi(s,a) \mapsto [0,1].$$

maps the pair (s,a) to a probability value.

$$\pi(a|s) = \mathbb{P}(A=a \mid S=s).$$

⇒ Probability of taking an action given a particular state.

Policy function can be a deterministic function or a random function.

Reward: After performing an action.

In Super Mario, the reward might be:
+1 - collecting coins.
+10,000 - winning / saving the princess.
-10,000 - Touching the turts (?).
0 - Nothing is done.

Had it be + 10000 for collecting coins & 1000 for saving princess, Mario would've been too busy going after coins.

State Transition: Old state $\xrightarrow{\text{action}}$ New state.
State transition can be random or deterministic.
Randomness to the state transition comes from the environment.

$$p(s'|s,a) = \mathbb{P}(S'=s' \mid S=s, A=a).$$

p function outputs the probability of the new state

GOOD BOY

__/__/____

given the old state and the action.

- Two sources of Randomness:
One is from the action while the other is from the state.

→ Dot means all possible states.

$$A \sim \pi(\cdot|s).$$

Action is randomly sampled from the discrete probability distribution.

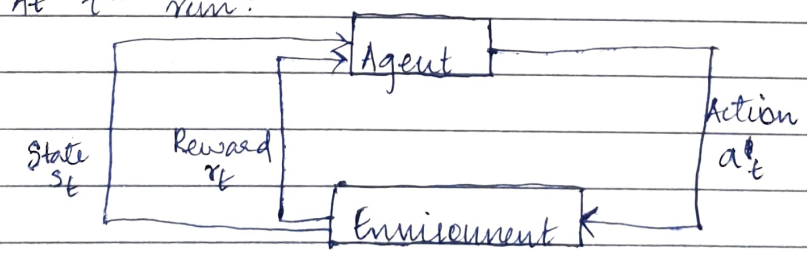
$$s' \sim p(\cdot|s,a).$$

~ means 'sampled from'.

Randomness in Action is from the policy function.
Randomness in State Transition is from the state transition probability function.

Environment generates the new state by sampling the state transition function.

At t^{th} run:



- Play the game using AI:
- Observe the state s_t, select an action ~~a_t~~
$$a_t \sim \pi(\cdot|s_t)$$
and execute a_t.
- AI will use the policy function to control the agent.
- The environment gives the new state s_{t+1} and

reward r_t:

$s_1 \rightarrow a_1 \rightarrow s_2 \rightarrow a_2 \rightarrow s_3 - \cdots s_n a_n r_n$.
$\searrow r_1$ $\searrow r_2$

$s_1 a_1 r_1, s_2 a_2 r_2, \ldots, s_n a_n r_n$
← episode →

What is a good policy?
A good policy leads to a big cumulative reward.

$$\sum_{t=1}^{n} \gamma^{t-1} r_t.$$

where γ (gamma) is the discounting factor.
r_t is the reward.

31 July.

RL uses the rewards to guide learning of the policy function. Many algorithms are devised to learn good policy functions.
We record the trajectories as we train with samples.
We let the agent interact with the environment and record the trajectories.

- Rewards and Returns:

Return (cumulative future reward).

Let U_t be the return at time t.

$U_t = R_t + R_{t+1} + \cdots + R_n$.
= sum of all future rewards from time t to the end.

__/__/____

Since future is uncertain, we introduce 'discounted return'.

Discounted return:

$$U_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \cdots + \gamma^{n-t} R_n.$$

where γ (gamma) is the discounting factor, b/w the range $[0, 1]$.

The above is called the weighted sum of rewards from t to end.
At the end of the game, $R_t, R_{t+1}, \cdots$, we can calculate the discounted return.

Randomness:
At any point of time 't', the reward $R_t, R_{t+1}, \cdots$ is random.
Reward R_i depends on State S_i and Action A_i.
State is random:

$$S_i \sim p(\cdot \mid S_{i-1}, A_{i-1}).$$

Action is also random:

$$A_i \sim \pi(\cdot \mid S_i)$$

where π is the policy.
Action is sampled from all the possibilities in the current state.

If either S_i or A_i is random, then R_i is also random.
Because reward R_i depends on S_i & A_i.
U_t depends on $R_t, R_{t+1}, \cdots$.
In turn, U_t depends on all the states & actions at t.

When a game ends, we know all the returns and U_t is just a number.

- Action-Value function:
  $Q_\pi(s,a)$.

At time t, U_t is random.

$$U_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \cdots + \gamma^{n-t} R_n.$$

How is the current situation being evaluated?

→ Expectation: $E[X] = \sum x\, p(x)$.

Take the expectation of U_t, by integrating out the randomness, we will get a real number that reflects how good the current situation is:

$$Q_\pi(s,a) = E\left[U_t \mid S_t = s_t, A_t = a_t \right].$$

It depends on the current state s_t and action a_t.

To compute the expectation, we need the probability density function (PDF) of S and A.

$Q_\pi(s,a)$ depends on the policy function.

$Q_\pi(s,a)$ is known as <u>critic</u>, since it evaluates the ~~presence of~~ performance of the agent.

$Q_\pi(s_t,a_t)$ depends on s_t, a_t, π and p.
$Q_\pi(s_t,a_t)$ is independent of S_{t+1} to S_n and A_{t+1} to A_n.

SIT DOWN
: BE HUMBLE :

__/__/____

- State - Value function:
  Assesses the how good a state is.

5 Aug.

1 Don't Remember - It was an online class.

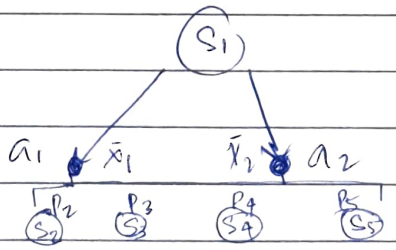
6 Aug.

- Model based Vs Model free Approach:

Model based: We have exact knowledge of what we are trying to do, lack of uncertainty.
Eg: Tic Tac Toe.

Model free: Random moves; ~~Not much info about~~
Eg: Mario.
Reinforcement learning.
Not much info about the env/agent.

Model based: Rewards from the env is known.
State transition probabilities are known.
Eg:



Model free: Rewards from the env is unknown.
State transition probabilities are unknown.

Exploration has to be done, repeatedly doing 'state - action - reward' loop.
Markovian approach ??.

__/__/____

Predictive Problem

↓ Policy

Evaluate the policy

↓

Corresponding value function.

Control Problem (DYNAMIC)

↓

Find the best Policy

↓

Optimal policy.

- No input is provided.
- Goal: explore the policy space & find the optimal policy.

- Classification of RL Algorithms:

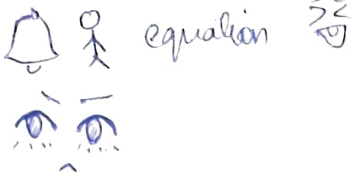
	Predictive	Control
Model based planning	Dynamic programming - policy evaluation.	- Dynamic prog - value iteration. - DP policy iteration.
Model Free RL	- Monte Carlo prediction. - Temporal Difference (TD). - TD(λ) backward.	- Monte carlo control. - SARSA. - SARSA backward - Q-learning. - Policy Gradient. - Actor Critic.

Interaction with the environment makes RL interesting!
???

Trajectory of interaction:

$s_1 a_1 r_1, s_2 a_2 r_2, \ldots$ → sequence can be in any order like this.

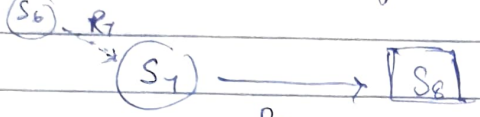
Eg: $s_1 a_1 r_1, s_3 a_3 r_3, s_7 a_7 r_7 \ldots$ etc.



__/__/____

- Bellman Equation:

Work backwards from a terminal state.



R_8 ⟶ Reward from taking an action to reach S_8.

$S_1 a_1 r_1 \cdots a_7 S_8 r_8$.

Return: All cumulative reward till that reward.

U_t - (cummulative) return - discounted cummulative sum of all previous rewards.

12 Aug

Did I miss any class? IDK .

- Multi-Armed Bandits.

I can't focus at all, my head ~~hurt~~ hurts because of someone's too-strong perfume.

Regret = Difference b/w best possible expected reward & the actual reward we got.

$$\text{Regret}(T) = T_{p^*} - \text{Reward}(T).$$

$p^* = \max_{i \in (1,2,\cdots,k)} p_i$: highest expected reward from one of the k arms.

$$\text{Avg\_regret}(T) = p^* - \frac{\text{Reward}(T)}{T}.$$

__/__/____

- Greedy (Naive) Algorithm.
- Epsilon - Greedy Algorithm.
- Upper Confidence Bound (UCB) Problem 4.
- Thompson Sampling Algorithm.

20 Aug. GAME THEORY

Missed about 15 minutes of the class.
I have very little idea of what's going on.

- Solutions:
  Dominant Strategy - 2 types.
  - Pure nash equilibrium - prisoner's dilemma.
  - ?
  - Mixed Strategy Nash equilibrium.
    Matching pennies: Player 1 wins if both are heads or both are tails.
    Player 2 wins if one head & one tail is observed.

- Games with no Nash Equilibria:
  Finite set of players & finite strategy sets is needed for nash equilibrium.

OOPS! Wrong Subject :\

* check moodle for the lecture notes.
* I missed atleast 1 ~~cla~~ prev class.
(car rental problem)

21 Aug.

- Planning by Dynamic Programming:

DP is used for optimization.
Problems → Subproblems
For each subproblem, we try to find an optimal solution.
Done using recurrence.

- Markov Decision process.
A set of states an agent can explore.

Requirements for DP:
- Optimal Substructure: decomposing into subproblems.
- Overlapping subproblems: subproblems recurs, cache solution to use again.
* Markov decisions satisfies both.

6 Oct

- Monte Carlo Method in RL:

- Evaluation and Improvement (?) of policies (πs):

- On-policy MC Method:
1. Generate experiences using the current policy (π).
2. Update the policy π using experiences generated from above.
3. The policy is improved incrementally, and same policy is used to generate new experiences.

Eg:
Goal: Evaluate a target policy π, always choose action A in the given state to move to the next state.
Action $\in \{A, B\}$.
$S = \{S_1, S_2, \ldots, S_n\}$. π.

__/__/____

S	Action	Reward	Next State
S_1	A	10	S_2
S_2	A	10	S_1
S_1	A	10	S_3
⋮			

Estimate the value $V\pi(s,a) = \sum_t r_i / N$.
where N - no. of experiences.

$$V\pi(s,a) = \sum_t r_i / N = (10+10+\cdots)/N.$$

Behaviour policy (β).

- Off-policy MC method:

1. Generate exp or take actions to move from one state to another, using a different policy (β).
2. Estimate the value of target policy π using the experiences from above.
3. Target policy π is improved without being used to generate experiences.

Eg:

Same kind as previous example.
Actions $\in \{A, B\}$.

S	Action	Reward	Next State.
S_1	A	10	S_2
S_2	B	5	S_1
S_1	A	25	S_6
⋮			

Importance Sampling : ~~Identify~~ The technique of identifying experiences relevant to π (target policy).

Weight = $P(A|S,\pi) / P(A|S,\beta)$.
[A : the target policy says 'always choose A'].

__/__/____

~~$V_\pi(S,a) = 1/2/3 = 3/2 = 1.5$~~

Weight $= 1/2/3 = 3/2 = 1.5$.

$V_\pi(S,a) = (10 \times 1.5) + (5 \times 1.5) + (25 \times 1.5) + \cdots \quad /N.$

9 Oct.

- Derivation of Importance Sampling:

$$\text{Weight} = \frac{P(A|S,\pi)}{P(A|S,\beta)} = \frac{\pi}{\beta} = \rho.$$

$X \leftarrow$ r.v.

Sampled from the probability distribution 'β' and we want to estimate the expected value of X w.r.t the target distribution π.

$$\mathbb{E}_\pi[x] = \sum_{x \in X} x\,\pi(x).$$

Important Sampling ratio 'ρ'.

$$\mathbb{E}_\pi[x] = \sum_{x \in X} x\,\pi(x)$$

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

$$= \sum_{x \in X} \underbrace{x\rho(x)}_{\text{a new variable.}}\beta(x).$$

~~$\mathbb{E}_{x \in \alpha}$~~

~~$\Rightarrow \mathbb{E}_X[x]$~~

$$\Rightarrow \mathbb{E}_{\beta(x)}[x\rho(x)] = \sum_{x \in X} x\rho(x)\beta(x).$$

PUNOCCHIO

Assignment :- GYM? Jim? Use it.
- Try to implement RL: MC or TD?

__/__/__

Now we can sample x from β and estimate its' expected value under π.

Here, we can change the policy if things don't work out, so getting more experience.

- Temporal Difference Learning (TD):

From the experiences, use "bootstrap", rather than waiting till the end of the episode.

Estimating how good the policy is on the go, like dynamic progamming.

- Generate experiences and use it to improve the policy; but before the end of the episode.

Can be compared with Monte Carlo - the opposite.

2 ways to do bootstrap:
TD(0) update, TD(λ) update.

1. TD(0) update:
It updates the value of the current state S_t based on the immediate reward R_{t+1} and the estimated value of next state $V(S_{t+1})$.

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

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

$V(S_t)$: Value at a given state

α : learning rate

:Maximize:

Thonda vanikkunnu !!!

__/__/__

~~S_t~~

TD Error:

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

Goal is to minimize the error.

TD(0) ← TD(λ) → MC.

[Wasn't paying attention 😶]

Optimization, control are the key points in RL.

- SARSA (On policy Control Algorithm):

It evaluates policy π that is used to take actions (behaviour). The update uses action A_{t+1} actually taken in the next state S_{t+1}.

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

ε-Greedy policy can be used here (?).

$S_{t+1} \leftarrow SB_t$.

- Q-Learning (Off-Policy Control Algorithm):

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

13 Oct.

- Missed A Class. 😔

EAR RINGING STRATEGY

__/__/__

14 Oct.

- Function Approximation:

Im too tired to write, sleepy.

In RL, an agent explores the states & gets rewards. This is stored in a look up table.
Supervised learning.

- Gradient Descent ~~Algo~~ Method:

SGD: $w_{t+1} = w_t - \alpha \nabla L(w_t)$.

For TD update:

$$w_{t+1} = w_t - \alpha \underbrace{(\text{Target} - \hat{V}(s_t, w_t))}_{\text{TD error.}} \nabla \hat{v}(s_t, w_t).$$

- Linear Function Approximation:

Vector estimate $\hat{v}(s_t, w_t) = w^T x(s) = \sum_t w_t x_t(s)$.

$$\nabla \hat{v}(s_t, w_t) = x(s).$$

↳ feature vector.

TD(0) update:

$$w_{t+1} = w_t - \alpha \left[R_{t+1} + \gamma \hat{v}(s_{t+1}, w_t) - \hat{v}(s_t, w_t)\right] \cdot x(s_t).$$

- Control Algorithm:
Find optimal policy π^* by approximating action value function q.
(Instead of prediction (state value)).

$$w_{t+1} = w_t + \alpha \left[R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, w_t) - \hat{q}(S_t, A_t, w_t) \right] \cdot \nabla \hat{q}(S_t, A_t, w_t).$$

- Q-Learning (Off Policy):

$$w_{t+1} = w_t + \alpha \left[R_{t+1} + \gamma \max_{a'} \hat{q}(S_{t+1}, a', w_t) - \hat{q}(S_t, A_t, w_t) \right] \cdot \nabla \hat{q}(S_t, A_t, w_t).$$