Chapter 1 Start
31 October 2025 03:37 PM

Learning is basically figuring out things through Interactions. And that basically Influences the results. We are going to design the algorithms that will learn through interaction and make decisions to achieve the end goal. The effectiveness is then calculated by the mathematical analysis.

In supervised learning, We have the labels to guide us. Here in RL, it is :
1) No right action labels
2) Must discover with trial and error,
3) We will get penalties and rewards for answers it chooses.

RL is all about learning what to do, That is figuring out the policies and taking action in each situations.
We also aim to maximize the numerical reward signal. - System always receives the feedback after each and every action.
The machine is not told on labels and stuff. Must explore and learn.
Must also find that gives long term success.

RL is a simultaneously problem, A class of method and field of studies. Meaning :
1) Problem : learning through interactions to achieve the goals.
2) Methods : Algorithms used
3) Field : The study and research area of these existing problem

RL model is all about how an agent interacts with an environment over time to achieve the goals.

The agent :
1) **Observes** the Environment - Gets a state.
2) **Acts** - Takes any actions
3) **Changes** the state of the Environment due to an action
4) **Receives the Feedback** - Rewards or not
5) **Repeats the Loop -** Till it maximizes the Long team reward.

Markov's Decision Process (MDP)

MDP Includes three aspects - Sensation, Action and the goal
1) Sensations - What the agent perceives ie the state
2) Actions - What the agents can do
3) Goal - Encoded via reward signal.

Difference between the RL and Unsupervised Learning is that, IN UL - It discovers the structures in the unlabelled data. Wheareas the RL focuses on maximizing the rewards.

The Exploration and Exploitation Trade Off

To get high rewards, The agent needs to exploit the things it knows already. It will choose the options that has worked out in the past really well. To discover a new and potential action, It must also explore and try out new actions, especially the ones it has not tried out. Exploring only can lead to doing random things, Exploiting always mean you will miss out on something better.

In definition :
Exploitation = Use current knowledge for immediate gain.
Exploration = Take risks to gather more information for the future.

Why hard ?

In a stochastic (ie mean random) task, The actions outcomes varies. To get a expected reward, You need to try multiple times.
There is no universal perfect solution, It all depends on the situation.

Supervised and Unsupervised is diff in a way. Because in 2 of them, We have the data fixed atleast. In RL we process and create the own data through

Interactions. Also RL doesn't act on one subcomponent like classification, Prediction, Optimal solution ect. Its literally on the real world. Need to interact on the entire environment and not just subskills.

RL focuses Top down :
It has goals fixed.
Can sense the Environment
Can act on it to influence it
Must operate under uncertainty

RL is a whole system, Sure works on the Subproblems too, But acts as whole always in the end.

Interdisciplinary Nature
RL involves the Interdisciplinary Nature of domains Like, maths, Psychology and Neuroscience. It also involves AI

let's give examples of the terms we used
1) Agent - Chess player, A robot, ect
2) Environment - Chessboard, Kitchen, Room with a charger
3) Goal - Winning, Battery Level, ect
4) Uncertainty - Opponents move, Battery level and usage
5) Action Impact - Actions changes the environment.
6) Feedback - Agent observes the new state, Adjusts the next move.

There is a delayed consequences, sure. But that's what drives the RL Model.

This is the essence of the temporal credit assignment - Understanding that actions now influence the rewards later. This is why they all say RL isnt reactive, It involves planning and foresight.

Monitoring and Adaptation - The agent actions can't be perfect always and predicted. The agent must constantly Observe and adjust. RL needs that continuous feedback and control.

Thus we can say that agent uses the experience to improve the performance over time.

Experience → Evaluation → Policy Improvement → Better Result and Performance

Elements of Reinforcement learning

RL has 4 main sub-elements beyond just agents and environment

Policy, Rewards signal, Value function and Model of the Environment.

1. Policy : A Policy is defined as the agents way of behaving at the given time.

This is the brain of the agent. Decides which action to take in what situation (Basically state)

π : state → action

There are 2 types of Policy :
1) Deterministic Policy - Picks the same state for a given state.
For a given battery level of 20%, Go charge and then continue is one such policy for Robots.
2) Stochastic Policy - Assigns probability to actions.
If battery at 50%, Go charge with 0.7 chance, 0.3 continue cleaning.

Policy could be simple rule tables or complex neural networks that computes actions from states.

In conclusion, Policy is literally the agents behaviour. Everything just exists to just improve the policy!

2. Reward Signal : A reward signal defines the goal of a reinforcement learning problem.

Each time the agent acts, the environment sends back a reward. This is a single number that measures how good or bad the action was.

$r_t = R(s_t, a_t)$

where s_t → state agent is currently in.
a_t → The action agent takes in the state

Where S_t → State agent is currently in
a_t → The action agent takes in the state
r_t → Reward the environment gives after action is taken.

The reward is a immediate Feedback for what just happened, It doesn't look ahead or back. It's just now! The agent will maximize the cumulative reward over time. If a action was bad or low rewards, the policy will be changed so that the agent avoids the similar action in the similar situation later.

3. Value Function : The reward signal we saw in 2 is a good way to say what's best in the immediate sense, A value function specifies what is good in the longer run.
   Value function : Instant gratification and Value function : Future satisfaction.

$V(s)$ = expected return starting from state s.
Reward → Short term goodness
Value → long term goodness.

Sometimes, The State might give a low immediate reward, But higher long term better value function or the opposite - Eating chocolate is good for now, Health wise its bad. A student studying now instead of watching some movies is good for later.

Without rewards, No learning goal. But without values, there are no strategy.

In Conclusion : RL uses the value function to maximize the rewards in a way!

The policy controls actions, The reward defines goals, The value function drives learning, and the model (if present) predicts outcomes.

4. A model is the agents internal simulation or understanding of how the environment behaves.

Model $f(s_t, a_t) \Rightarrow (s_{t+1}, r_t)$

For a given Current state st and action at, The model estimates what is next state and reward will be. Or in other words, The model lets agent imagine the outcomes without actually performing the action.

Why we need them ? - Models are used for planning, They are used to think ahead of acting. Agents can simulate the future situations using the model. Decides on the strategy before executing it.

There are two types of model : Model based, Model Free.
Model based uses a model to plan actions.
Model Free directly learns from trial and error. No predictions.

Limitation and Scope of RL

RL depends heavily on concept of states. They are the key input to policy and value function.

A state is basically everything the agent knows about the environment right now. It could also be said that whatever info the agent can sense or derive about its environment.

What RL doesn't do :
1) They don't care how you get the state - Like inputs and sensor data.
2) It assumes system gives the usable state signal
3) They only are decision making model and nothing else. Its main idea is : Given a current state, Whats the best action to take.

The difference between evolutionary methods and RL is : evolutionary methods also search for policies, but they don't have a value function or incremental learning. Its independent trial and not a continuedly learning agent. evolutionary methods just mutate to give better generation later. Not learn.

Let's use all these example for the Tic tac Toe.
Consider One player is X and other uses O.

Agent is X and an imperfect plater is O

Lets consider drawing the game and losing the game are both bad, -1 points and winning is 1 point.
RL goal here is Maximize the probability of winning P(Win)
We will be playing repeated games

There is a technique called Minimax - It says that the minimax assumes both players are perfect and it never visits a bad state at all. RL learns from however, the imperfect players which the minimax cannot exploit.
There is a also a technique called Dynamic programming. This technique needs the complete specification of the opponent.
Like the Full transition probability - How the board changes after each move.
Knowledge of the all possible outcomes.
These are very unrealistic in real world problems. They are unknown mostly in the environment that we are there.

RL is going to use the trial and error method. Observed the board states, actions and outcomes ie win or loss.
Updation of the internal evaluation of moves over many games is done too.

The RL Way of solving the Tic Tac Toe :
Each entry would estimate the probability of the eventually winning from the state at which right now we are there.
This estimate is the V(s) - State value.

All 3X in a row - Won - Value 1
All 30 in a row - Lost - Value 0
Any other board - In progress - 0.5 - 50% chance of anything happening at this point.

RL would slowly examine all the possible moves from the current board and give the value. It looks us the possible next State value V(s') in the table.
It chooses the moves according to the 2 strats.
1) Greedy Move : This is the exploit - Choose ation leading to the highest values next state - Best for predicting the winning chances
2) Explore Move : This is the explore - Choose a random move to discover new states it hasn't tried yet. This is done so that agent doesn't get stuck at a local optima.

While we are playing, we change the values of the state in which we are finding ourselves during the game. This is the core learning rule in reinforcement learning. Adjusting values estimates based on the experience.

Now, S_t → Current state → before a move

S_{t+1} → Next state → after a move

The agent update the value of $V(S_t)$ with the value of S_{t+1}.

Temporal Difference (TD) Update Rule:

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

Current State

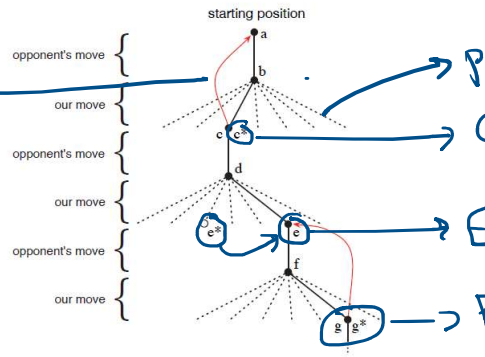
Learning Rate — How Fast we update

Error between the old Estimate and the new estimated value.

This is called bootstrapping - Learning partially from the old estimate and the next state estimated value.

Example - After a move, The new board position looks more winning - The previous state value increses slightly.

If the New board looks worse, Earlier state value decreases slightly.



Update value of V(s) from V(s')

possible ones, Not choosen

Currently the Best move.

Exploration.

Found good one

Large α → overshoots → unstable

Small α → stable → slow to adapt.

(*) α → Reduces overtime

If tuned correctly, The value V(s) will converge to true win probabilities. The policy becomes nearly optimal against the imperfect opponents. Agent action will approach the best possible moves (Exploring is exception)

The Alpha will never be completely 0. The agent can still adapt if the opponent strat changes a bit.

Also note : Any good action may have a delayed effects. Setting traps and future wins in tic tac toe is common. This will automatically we learned from temporal difference updates.

This tic tac toe was an example of the Episodic problem - Like a game win and loss.
Continuous problem like autonomous driving and robots learn continuously where decision happen nonstop.

Neural networks can act as a function approximator - It generalizes from past board positions to new, Unseen ones. Instead of remembering the each state exact value, it predicts based on features. This is how RL handle huge or continuous state space.

You pretty much cant visit all possible State - Way too many in real life. Model must learn general patterns and apply to unseen states too.