U-6 (DL)

- **Deep Reinforcement learning**

  - combination of RL & DL
  - Type of ML algo - learns to solve a multilevel problem by trial & error
  - Machine trained on real life scenarios to make sequence of decisions
  - Receives rewards are penalities for action s/its it performs
  - Goal to max. to the total reward.
  - Deep RL → multiple layers of ANN → present in architecture to explore
  - DRL → solve wide range of complex decision making tasks.

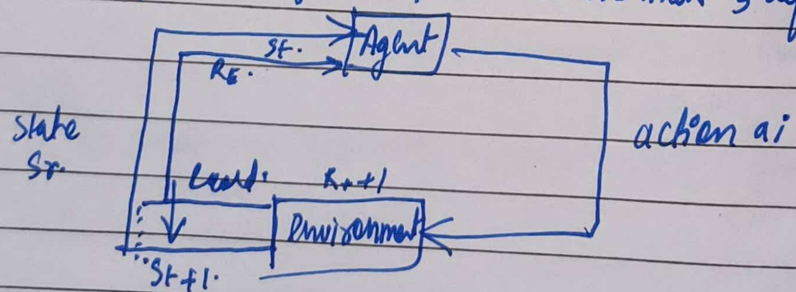
components → 1) Agent 2) environment 3) state 4) Action
5) Reward 6) value Fxn — estimates longterm reward
7) Policy optimization - goal of agent is to find policy maximize

Challenges → 1) Tradeoff b/w exploration & exploitation
Appl^n → Robotics, Game Playing, Autonomous Vehicle.

Challenge of RL
- sample efficiency
- high dimensional state
- Partial Observability
- safety & ethics
- Scalability & Comput^n Complexity

Markov Decision Process.

- mathematical framework used in to model sequential decision making problems
- provide a formal ways to represent environment & define interaction.

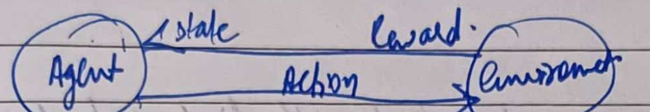


- used to formalize reinforcement learning problems.
- dynamic can be modelled → Markov Process.
- state → represents config / situation agent during decision making
- Actions → decisions or devices agent can be take at each state
- Reward fun^n → total cummulative rewards
- Trans^n Model → probabilities of trans^n from one state to another
- Discount factor → preference to present reward.

Challenges of RL - 1) Reward/ Credit Assignment
2) RL problems
3) Exploration, Exploitation Tradeoff

Page No.
Date

RL framework



eg→ self driving car, Robot locⁿ.

Dynamic Programming -

- Is an optimization method for seq. problems.
- DP algo are able to solve complex planning problems.
  - achieved with two principles -
    1) Breaking down problems into subproblems.
    2) Caching & reusing optimal solⁿ to subproblems to find overall optimal solⁿ.
  - Two strategies → 1) Policy Iteration 2) Value Iteration

Policy Iteration: compute both optimal value fxn & optimal policy.
policy iteration has 2 steps
1.) Policy evaluation
2) Policy Improvement.

a) Initialize policy policy arbitrarily.
b) Repeat until convergence.
policy evaluation
↳ evaluate V(s) for current policy it by solving.

Value Iteration:
- computes optimal values fxn for MDP
- Initializes arbitrary value fxn & then repeatedly update until it converges to optimal value.

a) Initialize value for states arbitrarily.
b) Repeat until convergence
- Bellman optimality eqn.

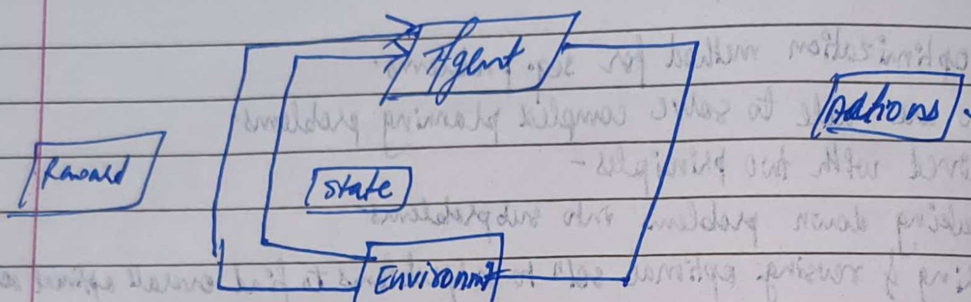
$$V(s) = \max [R(s,a) + \gamma * \Sigma P(s'|s,a) + Vs')].$$

- policy improvement -
↳ update policy by selecting actions that maximize expected values.

Provides systematic way to
Hard but best strategy for
max. cummulative

Q-Learning:-

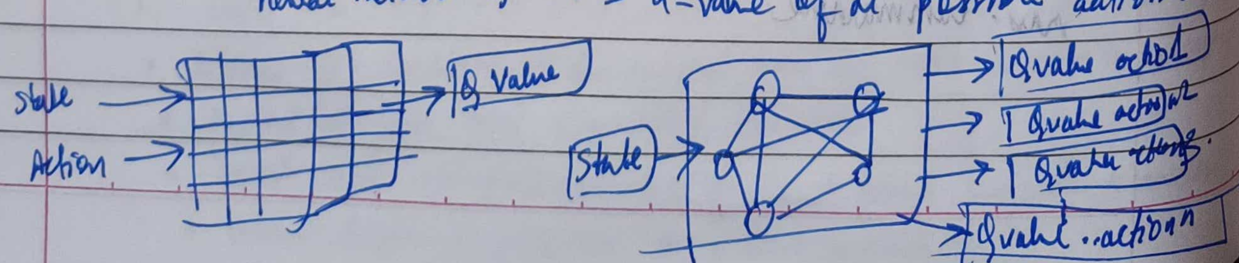
- reinforcement learning policy that will find next best action, given a current state.
- chooses this action at random & aims to max the rewards.



Adv → • long terms outcomes which are exceeding challenging to accomplish, are best acheived with strategy.
- fix mistakes during training.
- can produce ideal model to address a certain issue.
- Iteratively values-updates
- model free & handle environments with rewards.
- uses Bellman eqn to updates Q-values toward optimal policy.

Deep Q-Network(DQN)-

- only pratical for very small environment & quickly less its feasibility when no. of states & actions rto increasing.
- solving problem Deep Q-Network comes in feature.
- soln for above problem comes from relization values in matrix only have relative importance,
- basic working step for Deep Q-Learning is initial state is fed into neural network & returns Q-value of all possible actions as output.



Page No.
Date

- Deep Q network is varient of Q-Learning that uses a DNN to present Q-funct rather than simple tables of values.

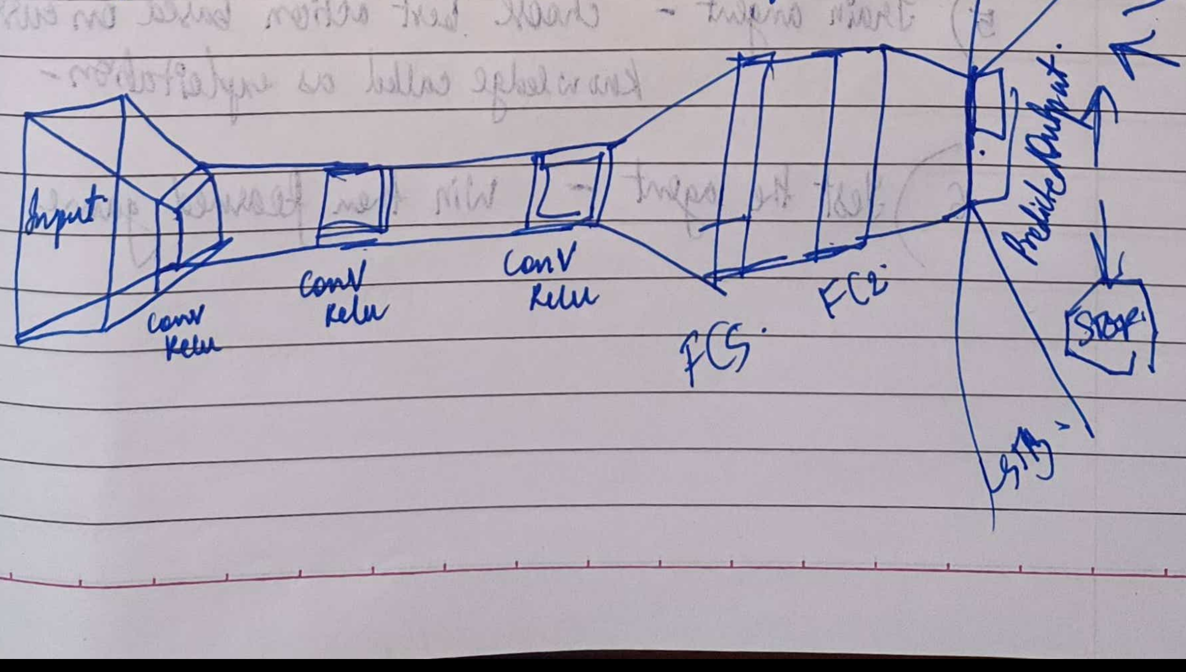
- Deep Q learning has been applied to wide range of problems, including game playing, robotics & autonomous vehicle.

Deep Q recurrent network -

- extension of Deep Q Network incorporate RNN into architecture.
- DQN uses feedforward NN to approximate Q values.

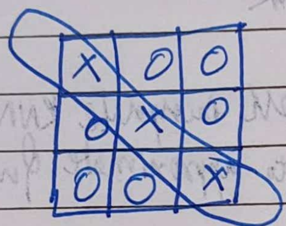
2 component - • RNN - typically LSTM or GRU.
• DQN - approx. Q values by among RNNs hidden state as addn input.

- similarly uses experience replay.
- implemented using RNN.
- Handle sequential data & provide better performance in tasks.
- DQL with RNNs input to neural network is sequence of Game states N output is sequence of Q-values, one for each state pair.
- allows agent to captures temporal dependencies & make use of past states & actions when estimating Q. values.



* Reinforcement Learning for Tic-Tac-Toe Game.

- simple game with a small state spaces, making an ideal environment for learning.
- In this game, two players take turns placing either an X or an O on a 3X3 grid.
- Goal is to place three of same symbol in a row, column or diagonal



1) Define state space - config of board ($\sim 3^9$) = 19683
   1400 unique
2) Define action space - all possible moves (9 moves n possible)
3) Define reward func^n - +1 → win, -1 → lose, 0 → other.
4) Define learning algo - Bellman eq^n:
   Q(s,a) = Q(s,a) + alpha * (reward + gamma * max[Q(s',a')] - Q(s,a))
5) Train agent - choose best action based on current knowledge called as exploitation.
6) Test the agent - Win then learned game.