ARTIFICIAL INTELLIGENCE (AG)

REINFORCEMENT LEARNING

- Reinforcement Learning: Learning with feedback to agent about what is good or bad to decide which action to take. ex. chess, Ping-pong game.
  + Task is to use observed rewards to learn an optimal (or nearly optimal) policy for an environment.
  + Two types:
    1. Passive Learning: Agency policy is fixed; the task is to learn utility of states (or state-action pairs); may involve learning a model of environment.
    2. Active Learning: Must also learn what to do; learn the policy also as there is no fixed policy. The principal issue is exploration; must explore as much as possible of its environment in order to learn how to behave in it.

* Passive Reinforcement Learning:
- Fixed policy 'π', in state 's', always executes the action π(s).
- Goal to learn how good the policy is, i.e. how good the utility function $U^{\pi}(s)$ is.
- Similar to policy evaluation task of policy iteration algorithm.
  + But, doesn't know transition model $T(s, a, s')$, which is probability of reaching state s' from 's' after doing action 'a'.
  + And doesn't know the reward function $R(s')$ which specifies reward for each state.
- Objective is to use the information about rewards to learn the expected utility $U^{\pi}(s)$ associated with each non-terminal state.
- Equation: $U^{\pi}(s) = \mathcal{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \mid \pi, s_0 = s\right]$ where, γ is called discount factor.

● Ways to implement Passive Reinforcement Learning:
1. Direct utility estimation: States that utility of a state is the expected total reward from that state onward.
   - At end of each sequence, the algorithm calculates the

observed reward-to-go for each state and updates the estimated utility for the state accordingly, just by keeping a running average for each state in a table.
- Ignores the dependence between states.

2. Adaptive Dynamic Programming (ADP): Agent tries to learn the transition model of the environment as it goes around and solving the Markov Decision Process using a dynamic programming method.
   - This adds $T(s, \pi(s), s')$ and observed rewards $R(s)$ into Bellman's Equation which is: $U^{\pi}(s) = R(s) + \gamma \sum_{s'} T(s, \pi(s), s')\ U^{\pi}(s')$.
   - Also adopt modified policy iteration approach of updating utility estimation after each change to the learned model.
3. Temporal Difference Learning:
   - Uses difference in utilities between successive states, often called temporal difference learning.
   - Given with equation: $U^{\pi}(s) \leftarrow U^{\pi}(s) + \alpha(R(s) + \gamma U^{\pi}(s') - U^{\pi}(s))$
   - Gives best of both the previous learning techniques.
   - Does not need model to perform its update.
   - First defines the condition that hold locally when the utility estimates that correct and then to write an update equation that moves the ~~update~~ estimates towards this ideal equilibrium equation.

* Active Reinforcement Learning (ARL)
- Agent must decide what action to take
- First learn a complete model with outcome probabilities for all actions, rather than just the model for the fixed policy.
- Learn utilities defined by the optimal policy; i.e. obeying Bellman's equation. $U(s) = R(s) + \gamma \max_a \Sigma_{s'} P(s' | a, s)\ U(s')$.
- After learning optimal utility function, the agent can extract an optimal action by one-step look-ahead to max. expected utility.
- Agent must trade off between exploitation to max. reward and exploitation to max. long term well being.

* Exploration

- Any scheme used for ARL has to be Greedy in the limit of infinite exploration (GLIE).
- A GLIE scheme must try each action in each state, an unbounded number of times to avoid having a finite probability that an optimal action is missed because of ~~an~~ usually bad series of outcomes.
- GLIE can be implemented as:
  + By agent choosing random action a fraction $1/t$ of the time to follow the greedy policy otherwise slow.
  + By assigning weights to actions not tried often, while tending to avoid actions of low utility. Done by assigning higher utility estimate to unexplored state-action pairs.

    Equation: $U^t(s) \leftarrow R(s) + \gamma \max_a f(\sum_{s'} P(s,a,s')\, U^t(s'), N(a,s))$

    Here, $f(u,n)$ is exploration function.

    Determines how greed (preference for higher 'u') is traded against curiousity (preference for lower 'n')

* Learning an Action - Value Function

- Learns an action-value representation instead of learning utilities. Method known as Q-learning.
- Motion $Q(a,s)$ to denote value of doing action 'a' in state 's'.

  $\therefore U(s) = \max_a Q(a,s)$
- Does not need a model for either learning or action selecting.
- Therefore, called model free method.

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

  The updated equation is:

  $$Q(a,s) \leftarrow Q(a,s) + \alpha\,(R(s) + \gamma \max_{a'} Q(a',s') - Q(a,s))$$

* Generalization in RI

- One way Function approximation: It means using any sort of representation for the function other than table.

+ Makes it practical to represent very large state spaces.
+ Takes less space and allows for inductive generalization over I/P states.
+ Chances of failure or longer delay for convergence.
+ Makes more sense to use online learning algorithm.
+ Apply Widrow-Hoff rule or delta rule for decreasing error.

$$Q_i \leftarrow Q_i - \alpha \frac{\partial \mathcal{E}_j(s)}{\partial Q_i} = Q_i + \alpha (u_j(s) - U_Q(s)) \frac{\partial U_Q(s)}{\partial Q_i}$$

+ Function Approximation allows reinforcement learner to generalize from its experience.
+ Applying linearity of function parameters Tn and a learning equation becomes

$$Q_i - Q_i + \alpha [R(s) + \gamma U_Q(s') - \hat{U}(s)] (\partial \hat{U} Q(s) / \partial Q_i)$$ for utilities and

$$Q_i \leftarrow Q_i + \alpha [R(s) + \gamma \max_{a'} \hat{Q}_\theta(a', s') - \hat{Q}_\theta(a,s)] \left(\frac{\partial \hat{Q}_\theta(a,s)}{\partial Q_i} \right)$$ for Q-values.