Recitation 13: Reinforcement Learning

Ken, Bhuvan

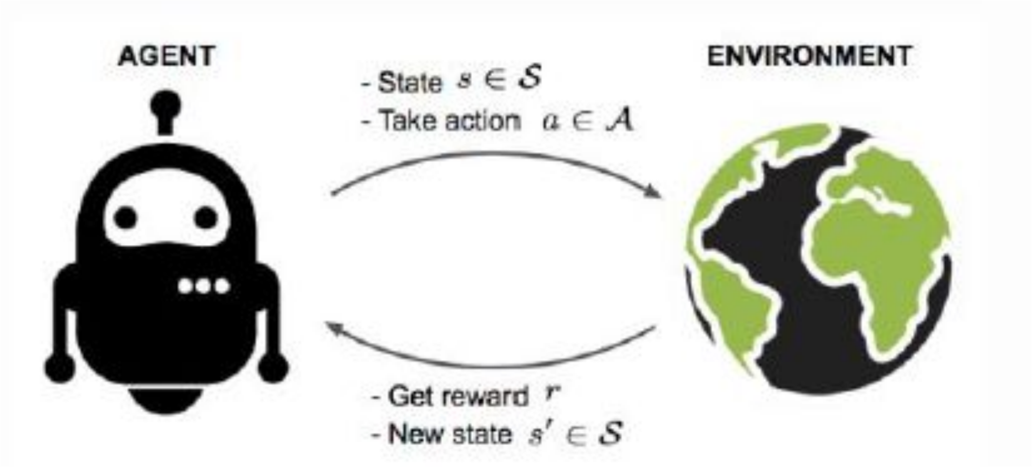
Reinforcement Learning

Learning Paradigms in Machine Learning:

- Supervised Learning
- Unsupervised Learning
- **Reinforcement Learning**

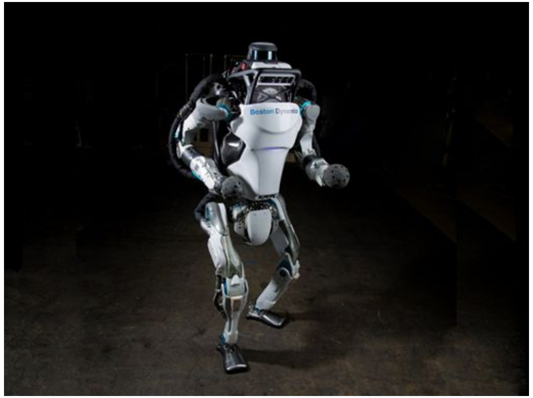
Reinforcement Learning

Learning to make decisions



Reinforcement Learning: Applications

- Games, Robotics, Control, Computer Vision, NLP ...



Markov Decision Process

- S: finite state space
- A: finite action space
- P: state transition model: p(s'|s, a)
- R: reward model: r(s, a, s')

Value Function, Q Function and Bellman Equation

What is a value function?

- Determines how valuable a given state is, for the agent.
- The value function depends on the policy using which the agent performs actions
- The value at a particular state using a policy **π** is given by:

$$V^{\pi}(s) = \mathbb{E}[\sum_{i=1}^{T} \gamma^{i-1} r_i] \quad \forall s \in \mathbb{S}$$

- Among all value-functions, there exists an optimal value function whose value is greater that other functions for all states. The optimal policy **π*** corresponds to the optimal value

$$V^{*}(s) = \max_{\pi} V^{\pi}(s) \quad \forall s \in \mathbb{S} \qquad \pi^{*} = \arg\max_{\pi} V^{\pi}(s) \quad \forall s \in \mathbb{S}$$

Value Function, Q Function and Bellman Equation

What is the Q-value function?

- Determines how valuable taking an action a is, from a given state s
- $V^*(s)$ can be obtained by finding the maximum over all possible $Q^*(s,a)$ values
- The $Q^*(s, a)$ is equal to the summation of immediate reward after performing action a while in state s and the discounted expected future reward after transition to a next state s'.
- If we know the optimal Q-function we can extract the optimal policy by choosing the action that maximises Q for a state s

Value Function, Q Function and Bellman Equation

The Bellman Equation:

$$Q^*(s,a) = R(s,a) + \gamma\, \mathbb{E}_{s'}[V^*(s')]$$

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in \mathbb{S}} p(s'|s,a) V^*(s')$$

$$\text{Since,}$$

$$V^*(S) = \max_a Q^*(s,a)$$

$$V^*(S) = \max_a \left[R(s,a) + \gamma \sum_{s' \in \mathbb{S}} p(s'|s,a) V^*(s') \right]$$

Value Iteration

- Computed the optimal state value function by improving the value of V(s) iteratively from a random start value
- Repeatedly updates Q(s,a) and V(s) until convergence and it is guaranteed to converge to optimal values.

Initialize $V(s)$ to arbitrary values
Repeat
  For all $s \in S$
    For all $a \in \mathcal{A}$
      $Q(s,a) \leftarrow E[r|s,a] + \gamma \sum_{s' \in S} P(s'|s,a)V(s')$
    $V(s) \leftarrow \max_a Q(s,a)$
Until $V(s)$ converge

Policy Iteration

- In value iteration, since the agent is optimising for the optimal policy, it might converge before value function.
- In Policy iteration, instead of repeatedly improving the value function, the policy is redefined at each step and the value is computed until convergence.

Initialize a policy π' arbitrarily
Repeat
  $\pi \leftarrow \pi'$
  Compute the values using π by
    solving the linear equations
      $V^{\pi}(s) = E[r|s, \pi(s)] + \gamma \sum_{s' \in S} P(s'|s, \pi(s))V^{\pi}(s')$
  Improve the policy at each state
    $\pi'(s) \leftarrow \arg\max_a (E[r|s, a] + \gamma \sum_{s' \in S} P(s'|s, a)V^{\pi}(s'))$
Until $\pi = \pi'$

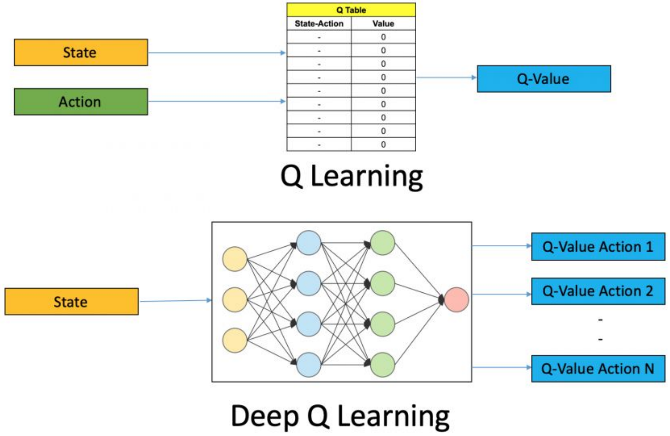
Q Learning

- Policy and Value iteration can be used when the agent has prior knowledge about the effects of its actions and the environment (offline planning)
- What if the agent only knows a set of possible states and actions and can observe the environment current state?
  - The agent must actively learn through its interactions with the environment
- Q-Learning a model-free learning algorithm that does not assume anything about the state-transition or rewards
- Q-learning tries to approximate the Q value of state-action pairs from the samples of Q(s,a) that were observed during the interaction with the environment.

Deep Q Learning

Why deep Q learning?

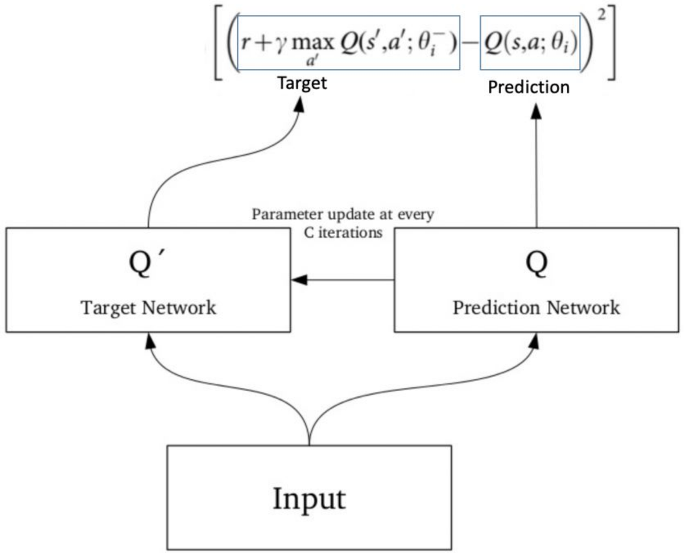
- If the number of actions and states in an environment are huge, tabulation becomes cumbersome due to both memory and time constraints
- Neural models can be used to approximate Q-values instead
- The state is given as the input and the Q-value of all possible actions is generated as the output



Deep Q Learning

What happens in DQNs?

- The past experiences are stored in a memory buffer and the next action is predicted by the Q network
- Loss is calculated as the mean squared error of the predicted Q value and a target Q value (Q*)
- For calculating the target Q value we can use a separate target network that can reduce divergence
- Target network has the same architecture as the Q-value prediction network but with the parameters frozen
- For every x iterations we copy the parameters from the prediction network to the target network
- This stabilizes training and reduces variability



Deep Q Learning

DQN steps summarized:

- Collect transitions from the environment to train the DQN.
- Select an action using the Epsilon-Greedy policy, i.e., select a random action versus maximum Q value action with a probability epsilon.
- Perform the action in a state s and move to a new state s’ and store this transition in the memory buffer <s,a,r,s’>
- Sample a batch of transitions from the replay buffer and calculate the loss
- Perform a gradient descent with respect to the actual network parameters to minimise the loss
- After every x steps copy actual network weights to the target network weights and repeat this for M episodes