Weekly report of lessons

Name: Mayank Kumar
Roll No: 19CS30029
The week: 01-11-2021 to 07-11-2021

The topics covered:

Reinforcement Learning - Learning with a critic, A simple learner: K-Arm Bandit, Nondeterministic: K-Arm Bandit, More complex environment, Learning a Markov Decision Process (MDP), Learning a policy, Learning optimal policy, Model-based learning: Value iteration algorithm, Policy iteration algorithm, Temporal difference algorithm, Deterministic Environment, Nondeterministic Environment, Large number of states and actions

Summary topic wise:

Learning with a critic: The learner interacts with an environment at some state, may change the state, and gets feedback (reward/penalty) while trying to reach a goal (a state). It learns a series of actions and maximizes the total rewards from any state.

A simple learner: K-Arm Bandit: Action (a) pulls a lever to win a reward r_a and the task is to maximize the reward by deciding which lever to pull. Supervised Learning predicts the lever leading to maximum earning, as labeled by a teacher. Reinforcement learning tries different levers and keeps track of the best. Q(a): value of action a. Initial Q(a) = 0 $\forall$ a. Select a' if Q(a') = max Q(a). Store r_a after each a, Q(a) = r_a

Nondeterministic: K-Arm Bandit: Action (a) pulls a lever to win a reward r with probability p(r|a).
$Q_t(a)$: estimate of value of action a at time t (average of all past rewards when a was chosen). Delta Rule: $Q_{t+1}(a) \leftarrow Q_t(a) + \eta\,[r_{t+1}(a) - Q_t(a)]$, η: learning factor with decreasing time. Select a' if Q(a') = max Q(a).

More complex environment: The environment is more complex when there are multiple states, action (a_t) affects the next state (s_{t+1}), reward is nondeterministic ($p(r|s_i, a_j)$), it learns $Q(s_i, a_j)$ which is value of taking action a_j in state s_i and the rewards may be delayed.

Learning a Markov Decision Process (MDP): At each discrete time t, the agent senses current state $s_t \in S$ and chooses action $a_t \in A$. $p(r_{t+1}|s_t, a_t)$, $p(s_{t+1}|s_t, a_t)$. It then receives reward r_t and changes state to s_{t+1}. The sequence of actions from start to terminal state is called episode or trial. Policy is the mapping from states to actions (π: S→A). Value of policy: $V^{\pi}(s_t) = E[r_{t+1} + r_{t+2} + \ldots + r_{t+T}]$; T is finite horizon.

Learning a policy: For a model with given $p(r_{t+1}|s_t, a_t)$ and $p(s_{t+1}|s_t, a_t)$, discounted value with infinite horizon: $V^{\pi}(s_t) = E[r_{t+1} + \gamma r_{t+2} + \ldots]$, optimal policy π^*: $V^*(s_t) = \max_{\pi} [V^{\pi}(s_t)]$.
Alternatively, we can learn $Q(s_t, a_t)$ and $V^*(s_t) = \max_a [Q(s_t, a_t)] = \max_a E[r_{t+1} + \gamma V^{\pi}(s_{t+1})]$

Learning optimal policy: $V^*(s_t) = max_{a_t}(E[r_{t+1}] + \gamma \sum_{s_{t+1}} P(s_{t+1}|s_t, a_t)\, V^*(s_{t+1}))$

$Q^*(s_t, a_t) = max_{a_t}(E[r_{t+1}] + \gamma \sum_{s_{t+1}} P(s_{t+1}|s_t, a_t)\, max_{a_{t+1}}(Q^*(s_{t+1}, a_{t+1})))$

$\pi^*(s_t)$: choose a* providing $V^*(s_t)$ or if $Q^*(s_t, a^*) = \max_a Q^*(s_t, a)$.

Model-based learning: Given $p(r_{t+1}|s_t, a_t)$ and $p(s_{t+1}|s_t, a_t)$ for a model, solve for optimal value function and policy by using dynamic programming:
$V^*(s) = \max_a (E[r|s, a] + \gamma \Sigma_{s'} P(s'|s, a)\, V(s'))$ $\pi^*(s) = \text{argmax}_a (E[r|s, a] + \gamma \Sigma_{s'} P(s'|s, a)\, V(s'))$
The two approaches are

- Value iteration algorithm: Iterate V(s) to arbitrary values. Repeat
  For all $s \in S$
  For all $a \in A$
  $Q(s, a) = max_a(E[r|s, a] + \gamma \Sigma_{s'} P(s'|s, a)\, V(s'))$
  $V(s) \leftarrow \max_a Q(s, a)$
  $\pi^*(s) = argmax_a(E[r|s, a] + \gamma \Sigma_{s'} P(s'|s, a)\, V(s'))$
  Until V(s) converge
- Policy iteration algorithm: Iterate π to arbitrary values. Repeat
  $\pi \leftarrow \pi'$
  Compute values using π

$V^{\pi}(s) = max_a(E[r|s, \pi(s)] + \gamma \Sigma_{s'} P(s'|s, \pi(s)) V^{\pi}(s'))$

Improve policy at each stage

$\pi'(s) = argmax_a(E[r|s, a] + \gamma \Sigma_{s'} P(s'|s, a) V(s'))$

Until $\pi = \pi'$

Temporal difference algorithm: In temporal difference algorithm, the environment is explored for value of next state and reward. This information is used to update value of current state, and examine difference between current estimate of value and discounted value of next state and the reward received.

Deterministic Environment: For any state-action pair, $Q(s_t, a_t) = r_{t+1} + \gamma max_{a_{t+1}}(Q(s_{t+1}, a_{t+1})))$. Update at every exploration by adding immediate reward with discounted estimate of next state-action pair, and converge when all the pairs are stable.

Nondeterministic Environment: For a state-action pair, vary reward or next state by keeping a running average of values. Delta rule: $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \eta [r_{t+1} + \gamma max_a\{Q(s_{t+1}, a)\} - Q(s_t, a_t)]$.
The next action is chosen randomly by ε-Greedy sampling.

Large number of states and actions: It is not feasible through tabular search and it uses regression to predict Q values when current value, reward and next state are given.

Concepts challenging to comprehend:
None

Interesting and exciting concepts:
None

Concepts not understood:
None

Any novel idea of yours out of the lessons:
Reinforcement Learning learns by focusing on the important parts of the space while ignoring the rest, and this makes it advantageous over the classical Dynamic Programming approach. We can see Q-learning as some stochastic approximations to dynamic programming. Reinforcement Learning can generalize and learn even faster when it represents knowledge by applying some function approximation methods. Further, we can decompose the main problem into a set of sub-problems. This is even more beneficial as learning a simpler subproblem is definitely faster, and now, the policies learned for a subproblem can be shared for multiple problems, which speeds up the learning of a new problem as well.
