(35) * Temporal Difference Methods - TD learning

↓

Monte Carlo method required episodes after which the Q table values were estimated and accordingly Monte-Carlo needed these breaks, which is not in temporal difference learning,

since many tasks don't work episodically and continuous learning is essential.

↓ in previous module (monte carlo control)

↓ now temporal Difference control.

↓

main idea : if the agent is playing chess, it doesn't have to wait until the end of the episode to see if its won the game or not.

↓

it will be able to estimate the probability at every move, whether its winning the game or not.

↓ or

in case of a self driving car, if training occurs in episodes and episode ends at a crash, then the car would need crashes to happen to be able to improve (which is obviously very expensive)

(Monte carlo approach issue)

↓

TD learning instead of updating values, whenever interaction ends, amends its prediction at every step.

↓

can be applied to both continous and episodic tasks, while Monte carlo can be implemented only on episodic tasks.

* TD control sarsa.

↓

update the Q table as the episode is unfolding

↓

$S_0^1, \rightarrow, -1, S_1^2, \rightarrow$: all info. req. to update the Q table under TD sarsa.

36

* In Monte Carlo, we had the future reward knowledge of the episode to figure out the discounted return for the current state, action in Q table to find the alternate estimate (G_t)

↓

But such knowledge is not available in TD learning, rather we have knowledge only about the current & next time step.

↓

so we pick the estimate for the t+1 timestep state and action pair from the Q table, and accordingly find G_t, using which we can update the Q table and so on as the episode progresses.

based on the Q table, we select the action for the state based on ε-greedy policy

↓

each action is selected for the modified Q table using ε greedy factor for the state.

	↑	↓	←	→
1	+7	+6	+5	+6
2	+8	+7	+9	+8
3	+10	+8	+9	+9

t_0 t_1

1, →, −1, 2, →

+6 current estimate

−1, +8 → +7 alternative estimate

↓

we move little towards +7,

$Q_t \rightarrow +6 + \alpha(7-6)$

$\rightarrow +6 + \alpha(1)$

$\rightarrow +6 + 0.2(1)$

$+6.2$

(taking $\alpha = 0.2$)

	↑	↓	←	→
1	+7	+6	+5	+6.2
2	+8	+7	+9	+8
3	+10	+8	+9	+9

↓

Same process is repeated for the next timestep.

2, →, −1, 3, ↑

$\underbrace{2, \rightarrow}_{t_1}, \underbrace{-1, 3, \uparrow}_{t_2}$

$$Q_t \leftarrow Q_t + \alpha (G_t - Q_t)$$

↓

$$G_t = R_{t+1} + \gamma Q_t(S_{t+1}, A_{t+1}) = -1 + 1(+10) = +9$$

↓

$$+8.2 \leftarrow +8 + 0.2(+9 - (+8))$$

	↑	↓	←	→
1	+7	+6	+5	+6.2
2	+8	+7	+9	+8.2
3	+10	+8	+9	+9

updated Q table after taking two time steps into consideration from Monte Carlo Control

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha (G_t - Q(S_t, A_t))$$

→ complete episode sampled

→ for each state action pair, we get $Q(S_t, A_t)$ (current estimate), whereas G_t (alternative estimate) is calculated using discounting of future rewards already known due to sampling of complete episode.

→ These are used to update the Q table

↓

in TD control, (sarsa 0) ← name of algorithm

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha (R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$$

↓

The reward that was estimated by having a knowledge of the complete episode, is now estimated using the current time step & next time step state & action (which is a much shorter window than before)

Q table is updated at each time step, rather than the end of an episode

↓

This approach can be employed in both continuous and episodic tasks.

↓

ϵ - greedy policy to select action for a state at every time step.

Scanned by CamScanner

{ sarsa (state, action, reward, state, action) }.

↓

This algorithm can be summed up as : for every time step $t \geq 0$, the agent : i) takes the action A_t (from current state S_t) that is ϵ-greedy wrt Q-table (chosen from previous time step)

ii) receives the reward R_{t+1} and next state S_{t+1}

↓

contributes to the cummulative reward.

iii) chooses the next action A_{t+1} (from next state S_{t+1}) that is ϵ-greedy wrt Q table

(which will be used as current state for next time step)

iv) uses the information in the tuple ($S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}$) to update the entry $Q(S_t, A_t)$ in the Q-table corresponding to the current state S_t and action A_t

TD Control : Q-learing (sarsamax)

tries to converge to
ϵ-greedy

In sarsa 0 algorithm of TD control, the updation of the Q table took place after the action for the next state is chosen using ϵ-greedy., and that state, action pair was used for the next time step.

↓

directly tries to target for optimal

initialization stays the same

Whereas in sarsamax, the updation takes place after the reward and the corresponding state is selected. The ~~state~~, action for the state is selected using the greedy policy to update the Q table. Once the Q table is updated, the new values are used to find the action for the current state using ϵ-greedy policy.

↓

The updation step becomes,

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right)$$

↓

$S_0 \; A_0 \; R_1 \; S_1 \; A_1 \; R_2 \; S_2 \; A_2 \; R_3 \; S_3 \; A_4$

$$\dashrightarrow Q(S_0, A_0) \leftarrow Q(S_0, A_0) + \alpha \left(R_1 + \gamma Q(S_1, A_1) - Q(S_0, A_0)\right)$$

$\pi \leftarrow \epsilon\text{-greedy}(Q)$

↓

sarsa 0

$S_0, A_0, R_1, S_1 \mid A_1 \; R_2 \; S_2$

$$\rightarrow Q(S_0, A_0) \leftarrow Q(S_0, A_0) + \alpha \left(R_1 + \gamma \max_{a \in A} Q(S_1, a) - Q(S_0, A_0) \right)$$

↓

sarsamax

↙

here A_1 is chosen using ϵ-greedy applied on the updated Q table from $S_t, A_t, R_{t+1}, S_{t+1}$

↓

S_1, A_1, R_2, S_2

↓

$$Q(S_1, A_1) \leftarrow Q(S_1, A_1) + \alpha \left(R_2 + \gamma \max_{a \in A} Q(S_2, a) - Q(S_1, A_1) \right)$$

↓

A_2 is then selected using ϵ-greedy policy. taking into account the updated Q table

TD control : Expected sarsa (closely resembles sarsamax)

↓

difference in updation step

(max prob to max action)

↑

to check expectation

↑ if an action occurring

↓

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(R_{t+1} + \gamma \sum_{a \in A} \pi(a / S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t) \right)$$

	↑	↓	←	→
1	+7	+6	+5	+6
2	+8	+7	+9	+8
3	+10	+8	+9	+9

$1, \rightarrow, -1, 2, \rightarrow$

$+6 + 0.1(-1 + (0.1 \times 8 + 0.1 \times 7 + 0.7 \times 9) + 0.1 \times 8) - 6)$

$= 6.16$

(42) * Optimism

↓

rather than initializing the Q table at 0, initializing to large values can improve performance.

↓

the initialized Q table is referred to as optimistic, since action-value estimates are guaranteed to be larger than true action values.

↓

For example, if all possible rewards that can be received by the agent are -ve, then initializing every estimate in the Q table to zero is a good technique.