RL

Unit - 4

Bootstrapping

→ in RL refers to a technique where an agent updates its estimate of state or action values using current or previous estimates rather than relying solely on complete trajectories or final outcomes.

→ This approach combines observed rewards with existing value predictions to refine its policy iteratively.

Mechanism

→ In bootstrapping, the agent updates the value function $V(s)$ or $Q(s,a)$ using the Bellman

equations, which express the relationship between the value of a state and the value of subsequent states.

for example

in Temporal Difference (TD) learning, the update rule for a state s_t is

$$V(s_t) \leftarrow V(s_t) + \alpha \left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \right]$$

$\alpha \rightarrow$ learning rate.

$\gamma \rightarrow$ discount factor.

$r_{t+1} + \gamma V(s_{t+1})$ is bootstrapped estimate of the return.

TD(0) Algorithm

TD(0) (Temporal Difference Learning with a 0-step lookhead) is a model-free, online RL algorithm that updates value estimates incrementally by combining observed rewards with current value predictions (bootstrapping). It operates on a per-time step basis without waiting for an entire Episode to complete.

TD(0) update Rule

The value of state $V(S_t)$ is updated using

$$V(S_t) \leftarrow V(S_t) + \alpha \left[r_{t+1} + \gamma V(S_{t+1}) - V(S_t) \right]$$

α - learning rate $(0 < \alpha \leq 1)$

γ - Discount factor $(0 \leq \gamma \leq 1)$

r_{t+1} - immediate reward after transitionins from s_t.

$V(s_{t+1})$ estimated value of the next state.

Algorithm

→ initialize $V(s)$

→ for each episode:

→ observe Initial state s_t.

→ while s_t is not terminal.

→ take action a_t (using policy).

→ observe reward r_{t+1} and next state s_{t+1}

→ update $V(s_t)$ using the TD(0) rule

→ set $s_t \leftarrow s_{t+1}$.

Ex:-

current state S_t : $V(S_t) = 5$.

Next state S_{t+1} : $V(S_t + 1) = 6$,

Observe reward

$r_{t+1} = 2$, $\alpha = 0.1, \gamma = 0.9$

Update

$V(S_t) \leftarrow 5 + 0.1 [2 + 0.9 \times 6 - 5]$

$= 5 + 0.1 [2 + 5.4 - 5]$

$= 5 + 0.1 \times 2.4 = 5.24$.

Advantage

1. Efficiency
2. Low variance.
3. Works in non-Terminating Environment.

Disadvantages

(1) Biar
(2) slower convergence.

Convergence of Monte carlo and Batch TD(0) Algorithm,

Both are used in RL to estimate value functions, Here, we focus on Monte carlo and Batch TD(0) alg.

(1) Monte carlo method

→ It learn value function by averaging the returns observed after visiting a state. They require complete episodes to update value estimate.

Convergence

MC methods converges to the optimal value function under conditions :-

→ sufficient exploration.

→ infinite visits to each state-action pair.

→ the policy evaluated is fixed

Advantage

→ unbiased estimates because they use actual returns

→ simple to implement.

Disadv

→ Requires complete episode before updating.

→ High variance

(2) Batch TD(0) method

TD(0) is a bootstrapping method that updates value estimate based on the observed reward and the estimated value of the next state. Batch TD(0) applies TD(0) updates to a fixed dataset (batch) of experiences.

Convergence

Batch TD(0) converges to the value function that minimizes the mean squared error on the training data.

→ The batch of data is fixed and finite.

→ The markov decision process (MDP) is finite and deterministic.

→ The step-size parameter (learning rate) satisfy the Robbins-monro conditions.

Advantages

→ can update value estimates after each step (online learning).

→ Lower variance compared to mc methods.

Disadvan

→ Biased estimate due to bootstrapping

→ Requires careful tuning of the learning rate.

key diff

Monte carlo	Batch TD(0)
Convergence target optimal value function	value function minimizing MSE on batch.
Bias/variance unbiased, high variance	Biased, lower variance.
Data Requirement Requires complete episodes	can use incomplete episodes
Speed of learning Slower due to high variance	Faster due to bootstrapping

① Monte Carlo method

you're playing a game where you move through rooms and collect rewards. the goal is to estimate the value of each room (state) based on the total reward you collect.

State :- Room A, Room B, Room C.

Action :- move left, move Right

Reward

→ moving from Room A to Room B give +0 reward.

→ moving from Room A to Room C give +1 reward.

→ Room B and Room C are terminal state (game end).

Episode - 1

Path :- A → C (reward = 1)

total return from Room A

$G(A) = 1$

Episode - 2

Path :- A → B (reward = 0)

Total return from Room A :

$G(A) = 0$

Episode - 3

Path : A → C (reward = 1)

Total return for Room A :

$G(A) = 1$

Value Estimation for Room A

$$V(A) = \frac{1+0+1}{3} = \frac{2}{3}$$

TD(0) method

Batch of Experience

(A, Right, 1, C)

(A, Left, 0, B)

(A, Right, 1, C)

step 1: initial $V(A) = 0, V(B) = 0, V(C) = 0$

step 2: for each experience, compute

TD target:

TD target $= r + \gamma V(s')$

Assume discount factor $\gamma = 1$)

For (A, Right, 1, C)

TD target $= 1 + 1 \cdot V(C) = 1 + 0 = 1$

for (A, Left, 0, B):

TD target $= 0 + 1 \cdot V(B) = 0 + 0 = 0$

For (A, Right, 1, C):

TD target $= 1 + V(C) = 1 + 0 = 1$

update V(A) at the end of 2D dataset

$$V(A) = \frac{1+0+1}{3} = \frac{2}{3}$$

Model-Free control in RL

* Model-free control in RL refers to a class of algorithms where the agent learns to make optimal decisions without explicitly constructing a model of the environment. Instead, the agent directly interacts with the environment, learns from the experience (state, action, reward, next state), and improves its policy over time.

Common model-free control Algorithms

(1) Monte Carlo Methods

→ learn value functions or policies by averaging returns from complete episodes.

→ suitable for episodic tasks.

→ Example:- Monte carlo control. where the agent updates the policy after each episode.

(2) Temporal Difference (TD) methods:-

→ learn value functions or policie by bootstrapping from current estimates.

→ suitable for both episodic and continous tasks.

→ Example

→ SARSA

→ Q-learning.

(3) Policy Gradient Method

→ Directly optimize the policy by maximizing the expected return using gradient ascent.

Eg:-

→ REINFORCE Algorithm

→ Actor-Critic Method.

Advantages

(1) Simplicity.

(2) Applicability

(3) Scalability.

Disadvantages

(1) Slow inefficiency

(2) Exploration Challenges

(3) Convergence Issues

Q - Learning

* Q-learning is a model free, off-policy reinforcement learning algorithm that learns the optimal action-value function $Q(s,a)$, represents the expected utility of taking action a in state s.

* goal is to maximize cumulative reward by iteratively updating a Q-table, which guide the agent's decisions.

Key components

Q-table :- Matrix storing $Q(s,a)$ value for state-action pairs

Bellman Equation :- Basis for Q-value ~~for~~ update.

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

α : learning rate ($0 \leq \alpha \leq 1$)

γ : discount factor

$R(s,a)$ → immediate reward after taking action a in state s

$\max_{a'} Q(s', a')$ → maximum feature reward estimate from next state s'

Algorithm

① initialize Q-table → Typically with zeros or small random values.

② Select Action → use an exploration strategy to choose action a in state s.

③ Execute Action → observe reward r and next state s'.

(4) Update q-value :- Apply the Bellman equation to adjust $q(s,a)$

(5) Repeat :- Transition to s' and iterate until convergence or terminal state.

Ex :- Simple Gird world

Environment

States :- s_0 (Start), s_1 (Goal)

Actions :- Right (R), Down (D)
(but Down D is invalid in s_0)

Rewards :- $s_0 \rightarrow s_1$ via Right : +1 (goal reached)

→ invalid operation (Down in s_0) : 0 (no movement)

Terminal state : s_1 (episode ends)

Step-1 :- initializ Q-Table

State	Right (R)	Down (D)
S_0	0	0
S_1	0	0

step 2 :- Agent interation (one Episode)

Parameters

→ learning rate (α) = 0.1

→ Discount factor (γ) = 0.9

→ Exploration (ϵ) = 0.1 (10% chance of random action).

Episode flow

① Start at S_0

② Choose Action

→ with 10% Chance (ϵ),
Explore: Down (D) invalid

→ with 90% chance,
exploit : Right (R) (optimal)

Assum the agent explores and selects Down (D) first.

(3) Observe Reward

- Reward = 0 (no movement)
- next state = s_0 (stays in place)

(4) Update Q-value for $s_0 \to D$

$$Q(s_0, D) = Q(s_0, D) + \alpha [R + \gamma \max_{a'} (s_0, a') - Q(s_0, D)]$$

$$Q(s_0, D) = 0 + 0.1 [0 + 0.9 \times \max(0,0) - 0]$$

$$Q(s_0, D) = 0$$

No change since s_0 has no future reward yet.

(5) next step

→ ~~Agent~~ Agent tries again in s_0

→ Now, Exploit Right (R) (Greedy way)

6. Observe Result

→ Reward = +1 (goal reached)

→ Next state = S_1 (terminal).

(7) update Q-value for $S_0 \to R$:-

$$Q(S_0, R) = Q(S_0, R) + \alpha \left[R + \gamma \max_a Q(S_0, a') - Q(S_0, R) \right]$$

$$Q(S_0, R) = 0 + 0.1 \left[1 + 0.9 \times \max (Q(S_1, a')) - 0 \right]$$

$$Q(S_0, R) = 0 + 0.1 \times 1 = 0.1$$

Update Q-table

State	Right (R)	Down (D)
S_0	0.1	0
S_1	0	0

Final Q-table (After convergence)

State	Right (R)	Down (D)
S_0	1.0	0
S_1	0	0

The agent learns that Right from S_0 is optimal to reach the goal.

Episode 1 → 0.1
Episode 2 → 0.19
" → "
Episode n → 1.0

Exploration vs Exploitation

① Exploration

* trying new actions to discover their rewards
* Helps the agent learn about the environment.

(2) Exploitation

→ uses known information to choose the best action.

→ maximizes immediate reward on current knowledge.

Trade-off

There should be no too much Exploration and too much Exploitation

if Too much Exploration → Agent wastes time on bad actions.

if too much Exploitation → Agent may miss better actions.

Ex:

Exploration → trying a new path to find shortcut

Exploitation → trying the known fastest path.

SARSA

→ SARSA (State - Action - Reward - Action) is an on-policy TD learning algorithm. it learns the Q-values $Q(s,a)$ for a policy while following the same policy, incorporating exploration into updates. The name reflects the sequence of steps: (s, a, r, s', a')

Key components

→ **on-policy learning** updates Q-values based on actions taken by the current policy

→ **update rule**

$$Q(s,a) \leftarrow Q(s,a) + \alpha [R + \gamma Q(s',a') - Q(s,a)]$$

α → learning rate

γ → discount factor

$Q(s',a')$ → Q-value of the next action selected by the policy

3. Algorithm steps

1. ~~Initi~~ Initialize $Q(s, a)$ arbitrarily (e.g. → zeros)

2. for each episode
   a. start ~~with~~ in state s.
   b. choose action a with the current policy (e.g. ϵ-greedy)
   c. Repeat until terminal state:-
      → take action a, observe reward R, and next state s'.
      → choose next action a' with the current policy.
      → update $Q(s, a)$ using the SARSA rule
      → $s \leftarrow s'$, $a \leftarrow a'$

SARSA vs Q-learning

SARSA	Q-learning
on-policy (learn the policy being used)	off-policy (learns optimal policy).
uses $Q(s', a')$ (actual next action)	uses $\max_{a'} Q(s', a')$.
~~Exploration~~ Accounts for exploration in updates	ignores exploration in updates.
~~Risk sensitivity~~ Safer in risky environments	more aggressive

Ex:-
Same for Q-learning Example just use formula

$$Q(s_0, a) = Q(s_0, a) + \alpha [R + \gamma Q(s_0, a) - Q(s_0, a)]$$

Expected SARSA

→ it is on-policy TD learning algorithm. it extends SARSA by using the expected value of the Q-value for the next state, weighted by the current policy action probabilities. This reduce variance compared to SARSA, which relies on a single sampled action.

(2) Key Components

→ On-policy learning
→ Expected value
→ Update Rule

$$q(s,a) \leftarrow q(s,a) + \alpha \left[R + \gamma \sum_{a'} \pi(a'|s') \, q(s',a') - q(s,a) \right]$$

$\pi(a'|s')$ → probability of choosing action a' in state s' under the current policy.

Algorithm

(1) Initialize $q(s,a)$ → zero.

(2) for each episode.

a) start in state s.

b) choose action a using the current policy

c) Repeat until terminal state.

→ take action a, observe reward R, and next state s'.

→ compute the expected q-value for s'.

$$E[\varphi(s', \cdot)] = \sum_{a'} \pi(a'|s') \varphi(s', a')$$

→ Update $\varphi(s, a)$:

$$\varphi(s,a) \leftarrow \varphi(s,a) + \alpha[R + \gamma E[\varphi(s', \cdot)] - \varphi(s,a)].$$

→ choose next action a' using the policy.

→ $s \leftarrow s'$, $a \leftarrow a'$.

Ex:- Same example as the φ-learning but use formulas

Agent in s_0 choose Down (D)

Reward 0, Next state s_0

Expected φ-value for s_0.

$E[\varphi(s_0, \cdot)] = (0.9 \times 0) + (0.1 \times 0) = 0$

update $\varphi(s_0, R)$:-

$$\varphi(s_0, R) = 0 + 0.1[0 + 0.9 \times 0 - 0]$$
$$= 0$$

$\alpha = 0.1$
$\gamma = 0.9$
$\epsilon = 0.1$

So for Right (R)
Reward +1
Expected φ-value for $s_1 : 0$ (termind state).

Expected SARSA	SARSA	φ-learning
Expectation over policy's actions	sampled next action.	max φ-value of next state
lower (uses expectation)	Higher (uses single sample)	Highest (uses max operator).
onpolicy	onpolicy	Off-policy

Short

① Episode

→ A sequence of interactions from an initial state to a terminal state

Ex:- A robot navigations from start to goal, ending when it succeeds or fails.

② Policy

→ The agent's strategy for choosing actions in a state

Types

→ Deministic: $\pi(s) = a$ (fixed action for a state)

→ stochastic: $\pi(a|s)$ = probability of action a in state s.

(3) Convergence

→ when the algorithm stabilizes (Q-values or policy stop changing significantly).

Ex:- Q-learning converges to the optimal Q-table after many Episodes.

(4) on-policy vs off-policy

on-policy	off-policy
learns the policy being followed (eg, SARSA)	learns the optimal policy independently of actions taken. (eg:- Q-learning)
updates we actions from the current policy	update we hypothetical actions (max Q value).
Ex:- SARSA	Ex:- Q learning.