12/7/1

DEEP REINFORCEMENT LEARNING

1.1 Intro. to Reinforcement Learning.

1.2. elements of R.L.

- Policy
- Reward
- Value
- Model

(2) Multiarmed Bandit Problem.

2.2 Stationary V/S Non-Stat.

same state

changing domain

circumstances

2.3 Exploration V/S exploitatn

trading

2.4 Bandit Gradient Algorithm.

(3) Markov Process.

3.2 Goals

Rewards & Returns

- Policy & value Funct
- Belman Eq.

best among all

- Optimal Soln
- Optimal Value

max or min value of function achieved by optimal solution

(4) Intro to Dynamic Programming → break down into smaller problems

- Policy Iteration
- Value Iteratn
- Generalized Policy Iteratn.

(5) MC Prediction ← - Use Monte Carlo method
- without a model
- simulate interactions
- $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$
- $V(s) = \frac{1}{N} \sum_{i=1}^{N} G_i$
- Law of large numbers
- computationally heavy

- M.C Control
- Incremental MC → online learn in RL data

(6) TD Prediction

Find optimal policy through sampling & policy iteration

- On Policy Temporal Difference Model Free (SARSA)
- Off Policy Temporal Diff. Model Free (Q-Learning)
- Expected sarsa.
- N step TD Policy evaluation
- N step SARSA
- off Policy N Step SARSA
- Tree Backup Algorithm

(7) Classific. of DRL Algo.
- Model Based V/S Model Free,
- Value Based V/S Policy Based
- On Policy V/S Off Policy.

(8) Function Approximation
- Feature Construction for Linear Methods (TILE coding, Asymmetric Tile coding)

- Linear Function Approximation
- Off Policy Func. " TD Divergence
- Deep Q Network
- Double DQN.
- Rainbow.

(9) Policy Gradient Methods
- Reinforce Algo.
- " with Baseline Algo.
- Actor Critic Method.
- Reinforce Algo. for Continuing Problem

(10) Upper Confidence Bound Action Selection
- Monte-Carlo Tree Search.
- Alpha Go Zero.
- Mu-Zero.
- Planet.

(11) Imitation Learning
- " " v/s Supervised Learn.
- Behaviour Cloning
- Inverse Reinf. Learning, GAIL.
- Data Set Augmentation, DAGGER.

OPTIONAL

Understanding Multi Agent Env.
- Co-Operative V/S Competitive Agents
- Centralized V/S Decentralized RI.
- Proximity Primal optimization (Surrogate Objective Function, Clipping).

LABS

1. Bandit Gradient Algorithm.
2. Implementing Dynamic Programming, Monte-Carlo, SARSA, Q-Learning,
3. Deep Q-Network
4. Reinforce.
5. Imitation Learning.

Exam Notes

1. On Policy Approx.
Chap 5.1, 5.2, 5.3 - DQN, DDQN.
3. 6.1, 6.2, 6.3.
4. UCB, MCTS.
5. Alpha Go Game
6. Imitation Learning
7. MARL - Proximal Policy Optimization (PPO)
8. K arm Bandit.
9. Comparison Model Based / Model Free
Value Based / Policy Based.
On Policy / Off Policy
10. Monte Carlo.

** 11. Issues with first visit, Monte Carlo Policy for a deterministic Policy.

12. Significant selection strategy in RL

13. Exploration V/S Exploitation.

REINFORCEMENT LEARN.

→ What is RL

Learn Optimal Behaviour by Interacting in an env.

→ Supervised Learning : - Labelled Data.

→ Unsupervised " " - Non- " " (Pattern Recog.)

→ Trial & Error - Reward & Penalty Based Learning.

Agent - Learner / Subject / Robot.

Env. - External system.

Action - Each time step - agent will select an action on basis of a Policy.

State - At a given time all necessary info - to make a decision [Discrete / Continuous]

[Reward] - positive reinf.

Goal of Agent is to Maximize Cumulative Rewards Over Time.

[Policy] - Agents strategy of Behaviour defined by it

- Deterministic (One state / One Action)
- Stochastic / Probabalistic Policy
- [Exploration v/s exploitation]
- Balancing exploration, Trying New Actions to learn about their outcomes
- exploitation - choosing actions of high rewards.

[Learning & Training]

RL Algo: "State - Action - Reward" sequences.

[Discount Factor]

↳ Balancing Immediate & Future Reward.

② Multiarmed Bandit Problem

- Going to the same coffee place means missing out on other option? But how will you know unless you try? But prob. of bad coffee will go up?

→ Single armed Bandit (slot machine) → Gambler doesn't known prob. of slot machine distribution, he's learning with time.

→ Multi armed Bandit (which lever to pull for max reward)

→ Action Value Function

→ Regret (Time spent on learning)

→ Greedy approach (No exploration, only exploit)

→ Epsilon Greedy approach (where Probability = 1 - Epsilon) i.e more random exploration

→ Softmax Exploration

→ Decayed Epsilon (Epsilon changes with time so we explore less & less with time)

★ Upper Confidence Bound (most used)

↳ The more uncertain, more explore that arm

↳ Choose arm, with highest variance in distribution

→ UCB

$$Q(a) + \sqrt{\frac{2\log t}{N_t(a)}}$$

→ Non stationary Bandit problem

↳ Use recency factor to give more weightage to new samples

Example online ads [MABP]

Arms, a, b, c, d, e

CTR → 10% 8% 8% 12% 6%

Obj → Max. CTR's

→ Exploration vs exploitation

→ Epsilon Greedy → more exploration as you more data, then exploit

→ Feedback → click

* Markov process

→ Next state is dependent only on current state

→ (1) State space (Sunny, cloudy, Rainy)

(2) Transition probabilities

↳ sunny → sunny = 0.7%

sunny → rainy → 0.1%

sunny → cloudy = 0.2%

(3) State transition matrix

From/to	sunny	cloudy	
sunny	0.7	0.3	
cloudy	0.2		

* Belman equation

→ relationship between state action & values of its successor in terms of future / immediate rewards

→ decomposes problem of finding policy, into smaller

* Rewards → Immediate feedback signal

Purpose → Desirable behaviors encouraging / discouraging

Returns → Cumulative total reward

↳ calculate overall value of action

→ Temporal Difference (Update at each step)

→ Method used to estimate value function such as state or action values based on experience gained

→ Combine elements of Monte Carlo methods (uses full episodes sampled) & dynamic programming (uses estimated values)

— TD Error – Difference between current & future value estimates

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

R_{t+1} = immediate reward

γ = discount factor

V_s, $V(S_{t+1})$ → value estimates for S_t, S_{t+1} etc

$\boxed{\delta_t = \text{TD error}}$

- Update Rules → Based of TD error.
- Differ based on TD(0) & action values (Q-learning, SARSA)

for
- TD(0), $V(S_t) \leftarrow V(S_t) + \alpha \delta_t$

α = learning rate

δ_t = TD error

for
- Q learning, SARSA

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \delta_t$$

where $Q(S_t, A_t)$ is the action value of state action pair (S_t, A_t), δ_t = TD error

- TD learning → Online & incremental updates, works for real time, large scale
- Don't require storing full episodic trajectories

- TD(0), Q learning, SARSA

Model Free vs Model Based

- Model Free: These learn policy or value function via interacting with env
- Model Based: learn model of the env. Then use it to make decision. Need env. dynamic info.

Value Based vs Policy Based

- Value Based: learn value functions eg. Q learning, Deep Q networks (DQN)
- Policy Based: learn optimal policy eg. Actor Critic methods

On Policy vs Off-Policy

- On Policy: eg. SARSA (Q policy TO control) & A2C (advanced Actor-Critic)
- Off-Policy: can use any policy eg. Q learning (off policy TO control) & DDPG (Deep deterministic policy gradient)

~~Policy~~

- Value function (evaluate goodness of state)
  - State value function $V(s)$
  - Action value function $Q(s,a)$
- Policies (map state to action)
  - Deterministic Policy (π)
  - Stochastic Policy (π)

Deep Q-Network → RL algorithm
↳ combines Q learning with deep neural networks
↳ approximate Q-values for given state-action pair

First understand Q learning
↳ used for learning optimal policy in MDP's without requiring a model of the environment
↳ learn action values (Q values)
↳ Q value, $Q(s,a)$ of state action pair (s,a) represents expected cum. rewards for state s, action a
↳ Goal is to learn optimal Q values for a state action pair

Q learning Process → Initialize Q values arbitrary
→ Exploration vs Exploitation → Exploration strategy (eg. ε-greedy) to discover new states while also exploiting to max. rewards

→ Update Rule (Q value iteration)

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

$Q(s,a)$ → current est
r → immediate reward
s' → next state
α → learning rate
γ → discount factor

→ Optimal policy → learn after iterations, then evaluate policy with highest Q value for S-A pair