Multi Agent Systems

- Lab 3 -

MDP Value and Policy Iteration Analysis

Slides adapted from Reinforcement Learning class by Vien Ngo, University of Stuttgart, 2016

Markov decision process

- A reinforcement learning problem that satisfies the Markov property is called a Markov decision process, or MDP.
- MDP = $\{\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{P}_0, \gamma\}$.
  - $\mathcal{S}$: consists of all possible states.
  - $\mathcal{A}$: consists of all possible actions.
  - $\mathcal{T}$: is a transition function which defines the probability $\mathcal{T}(s', s, a) = Pr(s'|s, a)$.
  - $\mathcal{R}$: is a reward function which defines the reward $\mathcal{R}(s, a)$.
  - $\mathcal{P}_0$: is the probability distribution over initial states.
  - $\gamma \in [0, 1]$: is a discount factor.

State Value Function

The **value** (*expected discounted* return – for infinite horizon settings) of a policy *π* when started in state *s:*

$$V^{\pi}(s) = \mathrm{E}_{\pi}\{r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots \mid s_0 = s\}$$

where 0 < γ < 1 is the *discounting* factor

Optimality: policy ***π**** *is optimal iff*

$$\forall_s : \; V^{\pi^*}(s) = V^*(s) \qquad \text{where } \; V^*(s) = \max_{\pi} V^{\pi}(s)$$

Bellman Principle of Optimality

$$V^*(s) = \max_a \left[R(a,s) + \gamma \sum\nolimits_{s'} P(s' \mid a, s)\, V^*(s') \right]$$

$$\pi^*(s) = \operatorname*{argmax}_a \left[R(a,s) + \gamma \sum\nolimits_{s'} P(s' \mid a, s)\, V^*(s') \right]$$

Value Iteration

Given the Bellman equation

$$V^*(s) = \max_a \left[R(a,s) + \gamma \sum_{s'} P(s' \mid a, s)\, V^*(s') \right]$$

$\rightarrow$ iterate

$$\forall_s : \; V_{k+1}(s) = \max_a \left[R(a,s) + \gamma \sum_{s'} P(s' | \pi(s), s)\, V_k(s') \right]$$

stopping criterion:

$$\max_s |V_{k+1}(s) - V_k(s)| \leq \epsilon$$

Policy Iteration

Value Iteration computes **V*** directly

To **evaluate** a given policy π, one needs to compute V^{π}, that is iterate using π instead of max_a

$$\forall_s : V_{k+1}(s) = R(\pi(s), s) + \gamma \sum_{s'} P(s'|\pi(s), s)\, V_k(s')$$

Counts as one iteration

Optimal policy can then be computed in iterative way

Policy Iteration

1) Initialise π_0 in a given way (e.g. randomly)
2) Iterate
   - Policy Evaluation: compute compute V^{π_k}
   - Policy Improvement: $\pi_{k+1}(s) = \underset{a \in A}{argmax} [R(a,s) + \gamma \sum_{s'} P(s'|a,s) V^{\pi_k}(s)]$

Value Iteration variants

Gauss-Seidel Value Iteration

- Standard VI algorithm updates all states at next iteration using **old** values at previous iteration (each iteration finishes when all states get updated).

Algorithm 1 Standard Value Iteration Algorithm

1: **while** (!converged) **do**
2: $V_{old} = V$
3: **for** (each $s \in \mathcal{S}$) **do**
4: $V(s) = \max_a \{R(s,a) + \gamma \sum_{s'} P(s'|s,a) V_{old}(s')\}$ ← Counts as one iteration

- Gauss-Seidel VI updates each state using values from previous computation.

Algorithm 2 Gauss-Seidel Value Iteration Algorithm

1: **while** (!converged) **do**
2: **for** (each $s \in \mathcal{S}$) **do**
3: $V(s) = \max_a \{R(s,a) + \gamma \sum_{s'} P(s'|s,a) V(s')\}$ ← Counts as one iteration

Value Iteration variants

Prioritised Sweeping

- Similar to Gauss-Seidel VI, but the sequence of states in each iteration is proportional to their update magnitudes (Bellman errors).
- Define Bellman error as $E(s; V_t) = |V_{t+1}(s) - V_t(s)|$ that is the change of s's value after the most recent update.

Algorithm 3 Prioritised Sweeping VI Algorithm

1: Initialize $V_0(s)$ and priority values $H_0(s), \forall s \in \mathcal{S}$.
2: **for** $k = 0, 1, 2, 3, \ldots$ **do**
3: pick a state to update (with the highest priortiy): $s_k \in \arg\max_{s \in \mathcal{S}} H_k(s)$
4: value update: $V_{k+1}(s_k) = \max_{a \in \mathcal{A}} \left[R(s_k, a_k) + \gamma \sum_{s'} P(s'|s_k, a_k) V_k(s')\right]$ Counts as one iteration
5: for $s \neq s_k$: $V_{k+1}(s) = V_k(s)$
6: update priority values: $\forall s \in \mathcal{S}, H_{k+1}(s) \leftarrow E(s; V_{k+1})$ (Note: the error is w.r.t the future update).

OpenAI Gym Test Environments

- Two test MDP environments based on OpenAI Gym[1]:
  - Taxi-v3[2]
    - 4 locations
    - Pickup passenger at one location and drop him off at another
    - 6 actions: move NORTH, SOUTH, EAST, WEST + PICK_UP + DROP_OFF
    - Rewards: +20 for successful drop-off, -1 per movement, -10 for illegal pick-up or drop-off
  - FrozenLake-v1 (8x8)[3]
    - Agent controls movement over a grid
    - Some tiles walkable, some lead agent to fall into water; agent has a start and goal tile
    - Movement of agent with uncertainty (due to slippery ice)
    - Rewards: +1 if agent finds correct path, 0 otherwise

1. Requires installing in a Python environment using python setup.py install
2. Instantiate using env = gym.make(“Taxi-v3”)
3. Instantiate using env = gym.make("FrozenLake-v1", map_name="8x8", is_slippery=True)

OpenAI Gym Test Environments

- For Value / Policy Iteration we do not need to **run the game;** we only need its *properties*
- For an instantiated environment: e.g. env = gym.make(“Taxi-v3”)
  - Set of states: **env.observation_space** → a Discrete(n) (i.e. a range of discrete states numbered 0 to n-1)
  - Set of actions: **env.action_space** → a Discrete(m) (i.e. a range of discrete actions numbered 0 to m-1)
  - Transition Probabilities: **env.P[state_idx][action_idx]** = (**prob**, **new_state**, **reward**, terminated)

Tasks

- For each game consider the following values for convergence criteria
  - max_iterations: 5*10^5
  - epsilon_threshold: 10^-2 or 10^-3
  - Discount factor: 0.9
- **Step 1**: compute **V*** using the **standard Value Iteration Algorithm**
- **Step 2:** Run the two variants of Value Iteration **(Gauss-Seidel VI, Prioritized Sweeping VI)** and compute the **number of iterations** until convergence to **V*. ATTENTION:** pass **V*** as a parameter to Gauss-Seidel VI and Prioritized-Sweeping VI
- **Step 3:** Run 5 instantiations (random policy init each time) for **Policy Iteration** for each game: compute the **average of number of iterations** until convergence. **ATTENTION:** pass V* as a parameter to Policy Iteration
- **Plot convergence graph**
  - X axis: number of iterations **(NOTE!** An **iteration** is considered **an update to a state** in the value function**, i.e. an update to V(s))**
  - Y axis: $||V - V^*||_2$
- **Analyze convergence speed properties**