Final Project: Value-Iteration Method for MDP

Group 6

The Chinese University of Hong Kong, Shenzhen

Chuqiao Feng 120090272
Lang Tian 120090793
Linli Zhou 120090555
Manning Wu 120090753

April 21, 2023

Overview

Preview

- We work on implementing the value-iteration method for Markov Decision Processes.
- We ask the following question:
  Q: How to solve Markov Decision Process more efficiently?
- Our answers:(Overview)
  A1: Value Iteration with random and cyclic strategy
  A2: Value Iteration with state aggregation

Motivation

- We want to optimize sequential decision-making in reality.
- Why is the topic challenging?
  1. Curse of dimensionality
  2. The trade-off between efficiency and performance.

Problem Setup

Markov Decision Processes (MDPs)

In MDPs, we consider the following minimization problem

$$\min_{\pi \in A^S} V_\pi(i) := \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t c_{a_t}(i_t) | i_0 = i \right] \tag{1}$$

where $\{i_0, a_0, i_1, a_1, \cdots, i_t, a_t, \cdots\}$ are state-action transitions generated by the MDP under the fixed policy π, i.e. $a_t = \pi_{i_t}$, and the expectation $\mathbb{E}_\pi[\cdot]$ is over the set of (i_t, a_t) trajectories.

Notation

- S is finite state space, and the number of the total states is $|S|$
- A is finite state action space
- $\gamma \in [0, 1)$ is the discounted factor
- P is the collection of state-action-state transition probabilities, with $P(i'|i, a)$ represents the probability of going to state i' from state i when taking action a
- c is the collection of costs at different state-action pairs, i.e. we cost $c_a(i)$ if we are currently in state i and take action a

Bellman optimality equations

Bellman optimality equations (Bellman 1957)

The (optimal) value function achieved by the optimal policy satisfies

$$V^*(i) = \min_{a \in A_i} \left(c_a(i) + \gamma \sum_{i' \in S} P(i'|i, a) V^*(i') \right) \tag{2}$$

Value operator

For a given MDP, the value operator $T : \mathbb{R}^{|S|} \to \mathbb{R}^{|S|}$ is defined for all $U \in \mathbb{R}^{|S|}$ and $i \in S$ by

$$T(U)_i = \min_{a \in A_i} \left(c_a(i) + \gamma \sum_{i' \in S} P(i'|i, a) U(i') \right) \tag{3}$$

V^* is the fixed point of T.

Connection with Linear Programming(LP)

The Value operator inspires us to find an upper bound, $V \geq T(V)$, by considering a larger set of linear constraints.

$$V(i) \leq c_a(i) + \gamma \sum_{i' \in S} P(i'|i,a)V(i') \quad \forall a \in A, i \in S \tag{4}$$

Thus we can reformulate (1) as follows:

$$\begin{aligned} \min \quad & \sum_{\forall i \in S} V(i) \\ \text{s.t.} \quad & V(i) \leq c_a(i) + \gamma \textstyle\sum_{i' \in S} P(i'|i,a)V(i') \quad \forall a \in A, i \in S \end{aligned} \tag{5}$$

Value Iteration

Lemma (Contraction Mapping)

For all values $U, V \in \mathbb{R}^{|S|}$ *we have that* $\|T(U) - T(V)\|_\infty \leq \gamma \|U - V\|_\infty$ *and consequently* $\|T(U) - V^*\|_\infty \leq \gamma \|U - V^*\|_\infty$, *where* V^* *is the optimal value vector.*

- Based on Lemma 1, we can use the fixed-point iteration method, which is called **Value Iteration** in MDPs

$$V^{k+1} = TV^k$$

- We have $\|V^k - V^\star\|_\infty \leq \gamma^k \|V^0 - V^\star\|_\infty$

Lemma (Entry-wise Monotone Property)

if values $U, V \in \mathbb{R}^{|S|}$ *satisfy* $U \leq V$ *entry-wise, then* $T(U) \leq T(V)$ *entry-wise.*

(Vanilla) Value Iteration Algorithm

Algorithm 1 vanilla value iteration(S, A, c, P, γ)

Initialize $V^0 \in \mathbb{R}^{|S|}$ arbitrarily;

for $k = 0, 1, \cdots$ **do**

  for $i \in S$ **do**

    $V^{k+1}(i) = \min_{a \in A_i}\{c_a(i) + \gamma \sum_{i' \in S} P(i'|i, a)V^k(i')\}$

  end

end

Random Value Iteration

Motivation: When $|S|$ is huge, update is computationally expansive in each iteration.
Solution: Randomly select a subset of S to update.

Random Value Iteration (Random VI)

In the kth iteration, randomly select a subset of states B^k and do

$$V^{k+1}(i) = \min_{a \in \mathcal{A}_i} \{c_a + \gamma \sum_{i'} P(i'|i,a) V^k(i')\}, \ \forall i \in B^k. \tag{6}$$

Modification (Influence Tree): B^k is a random subset of states which are connected by any state in B^{k-1}

Cyclic Value Iteration

Cyclic Value Iteration (CyclicVI)

Update one state at a time in order. In the kth iteration do

- Initialize $\tilde{V}^k = V^k$.
- For $i = 1$ to $|S|$

$$\tilde{V}^k(i) = \min_{a \in \mathcal{A}_i} \{c_a(i) + \gamma \sum_{i'} P(i'|i,a) \tilde{V}^k(i')\} \tag{7}$$

- $V^{k+1} = \tilde{V}^k$.

Difference with Vanilla VI: vanilla VI is synchronous (using V^k in update), but CyclicVI is asynchronous.

Modification (Randomly Permuted CyclicVI):

Update one state at a time in random order B^k during the kth iteration.

Maze Setting and Notations

Maze Setting

- Standard Maze: 2D Maze with m^2 states (m = height = width)
- Terrain Maze: 3D Maze with m^2 states
  (Height: Initially, consider the height of each state as a random number. Then, the height of each state is updated by calculating the average of the heights around it.)

Example

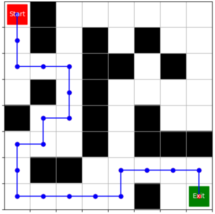

Figure: Standard Maze

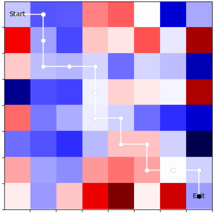

Figure: Terrain Maze

Comparison of methods in standard maze

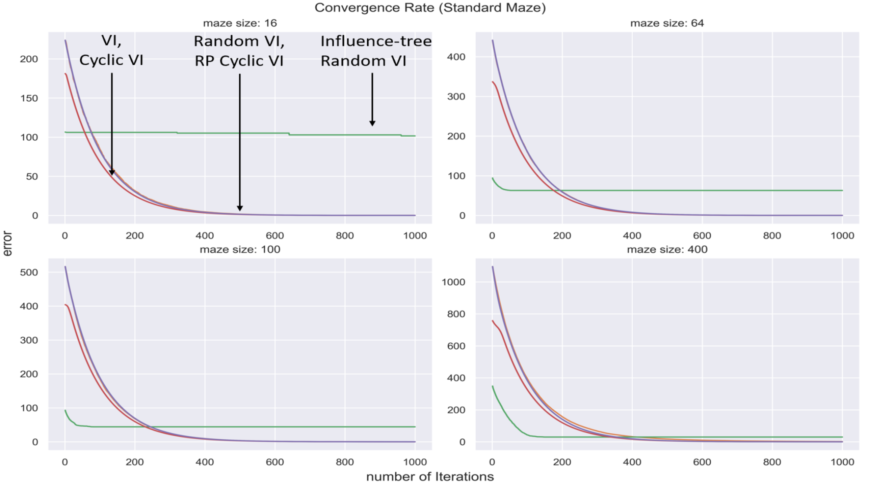

Figure: *Convergence rates of various value iteration methods in standard maze. ($\gamma = 0.99$, 20000 runs, random sample size/maze size = 1)*

Comparison of methods in terrain maze

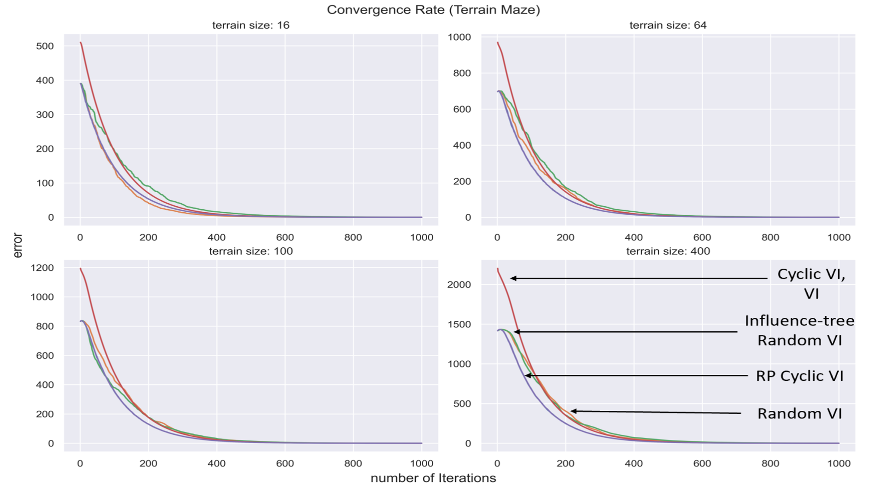

Figure: *Convergence rates of various value iteration methods in terrain maze. ($\gamma = 0.99$, 20000 runs, random sample size/maze size $= 1$)*

Results

- For both standard maze and terrain maze
  - Number of iterations to converge:
    RP Cyclic VI = VI $<$Cyclic VI $<$Random VI $<$Influence-tree Random VI
  - Random VI and Influence-tree random VI have lower computational cost.

More computation-efficient method?

- Difficulty: The state space S is huge.
- Random selection can partially reduce the computation cost, but does not utilize the problem structure information.
- New idea: state aggregation.
  - Similar states have close values (long-term rewards).
  - We can group/aggregate such states into a mega-state.
  - The state space size is reduced.
  - Mathematical meaning: piece-wise constant function approximation rather than discrete table.

State Aggregation

Mega State

A state partition $\{S_i\}_{i=1}^K$ on $\mathcal{S}$: $\mathcal{S} = \cup_{i=1}^K S_i$ and $S_i \cap S_j = \varnothing$ for $i \neq j$
Denote $W \in \mathbf{R}^K$ the cost-to-go value function for the mega state

The current value of W induces a value function $\tilde{V}(W) \in \mathbf{R}^{|\mathcal{S}|}$ on the original state space:

$$\tilde{V}(s, W) = W(j), \quad \mathit{for}\ s \in S_j$$

Adaptive Aggregation

Challenge:

- If aggregation rule W is pre-specified, we hope (and could design W such) that $\|\tilde{V}(W) - V^*\|_\infty$ is small.
- But, we do not know $V^\star$ before problem-solving, so it is hard to design W.

Solution in [Chen et al., 2021]:

- We have $V^k \to V^*$, so V^k is a surrogate of V^*.
- We can adaptively update the aggregation rule W based on V^k.

Value-based Aggregation

Algorithm 2 Value-based Aggregation

Input: $\varepsilon, \mathbf{V} = (V(1), \cdots, V(|\mathcal{S}|)^T$

$b_1 = min_{s\in|\mathcal{S}|} V(s), b_2 = max_{s\in|\mathcal{S}|} V(s), \Delta = (b_2 - b_1)/\varepsilon;$

for $i = 1, \cdots, \lceil \Delta \rceil$ **do**

$\quad \hat{S}_i = \{s|V(s) \in [b_1 + (i-1)\varepsilon, b_1 + i\varepsilon)\}, \hat{W}(i) = b_1 + (i - \frac{1}{2})\varepsilon$

Output: Return $\{S_i\}_{i=1}^K$ and W

Key idea: Discretize V^k into intervals based on $\min_s V^k(s)$ and $\max_s V^k(s)$,

Periodical Implementation

Two-Phases Algorithms:

- Phase 1 (with $\mathcal{B}$): algorithm performs global updates on $|S|$.
- Phase 2 (with $\mathcal{A}$): algorithm performs state-aggregated updates.

For a pre-specified number of iterations n, the time horizon [1, n) is divided into intervals of the form $\mathcal{B}_1, \mathcal{A}_1, \mathcal{B}_2, \mathcal{A}_2, \cdots$.

- Example:

$$\mathcal{B}_1 = \{1, 2, 3, 4\}, \quad \mathcal{A}_1 = \{5\},$$
$$\mathcal{B}_2 = \{6, 7, 8, 9\}, \quad \mathcal{A}_2 = \{10\}$$

Algorithm

Algorithm 3 Value Iteration with Adaptive Aggregation

Input: $P, c, \gamma, \varepsilon, \{\alpha_t\}_{i=1}^{\infty}, \{\mathcal{A}_i\}_{i=1}^{\infty}, \{\mathcal{B}_i\}_{i=1}^{\infty}$
Initialize $W_0 = 0$, $V_1 = 0$, $t_{sa} = 1$
for $i = 1, \cdots, n$ **do**
  if $t \in \mathcal{B}_i$ **then**
    if $t = min\{\mathbf{B}_i\}$ **then**
      $V_{t-1} = \tilde{V}(W_{t-1})$
    for $j = 1, \cdots |\mathcal{S}|$ **do**
      $V_t(j) = T_j V_{t-1}$

Continued

if $t \in \mathcal{A}_i$ **then**

  if $t = min\{A_i\}$ **then**

    Define $\{S_i\}_{i=1}^K$ *and* W_t *to be the output of Algorithm 2*

  for $j = 1, \cdots, K$ **do**

    Sample state s uniformly form collection S_j.

$$W_{t+1}(i) = (1 - \alpha_t)W_t(i) + \alpha_t T_i \tilde{V}(W)$$

  $t_{sa} = t_{sa} + 1$

if $n \in \mathcal{B}_i$ **then**

  return V_n

return $\tilde{V}(W_n)$

Experiments

Setting

- Discount factor γ: 0.99
- $|\mathcal{A}_i|$: 2 $|\mathcal{B}_i|$: 5
- Learning rate α_t: $\frac{1}{\sqrt{t}}$
- ε: 0.5
- Initialization V_0: $\mathbf{0}$

Experiments

Influence of ε: We run experiments on a 20×20 maze with different setting of ε to test the effect of ε on error.

Convergence: We test the convergence of algorithm 3 against value iteration (VI) on 20×20 standard and terrain maze.

Efficiency: We compare the computation time of algorithm3 in 4000 runs against VI on large-scale terrain maze (50×50) repeated for 20 times.

Result I

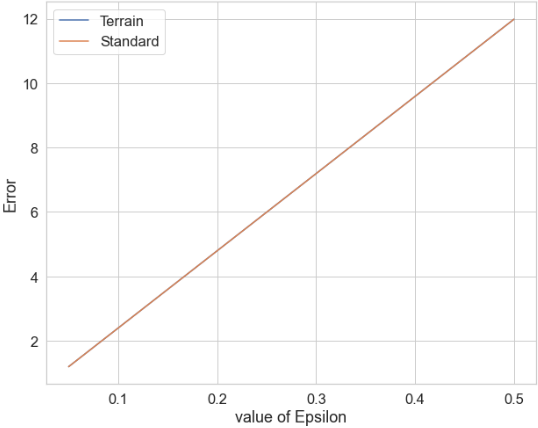

Figure: Influence of ε

$\varepsilon =$0.05, 0.2, and 0.5. The error $\|E_t\|_\infty \propto \varepsilon$.

Result II

orange line: Algorithm 3 **blue line**: Value Iteration

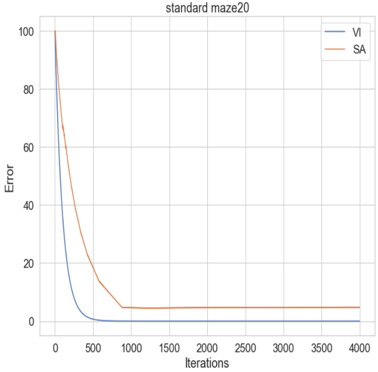

Figure: Standard maze

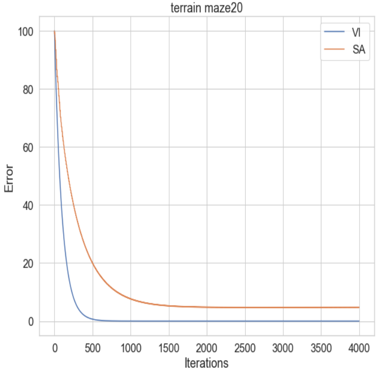

Figure: Terrain maze

The state-aggregated update $\mathcal{B}$ will increase the $\|E_t\|_\infty$.

Result III

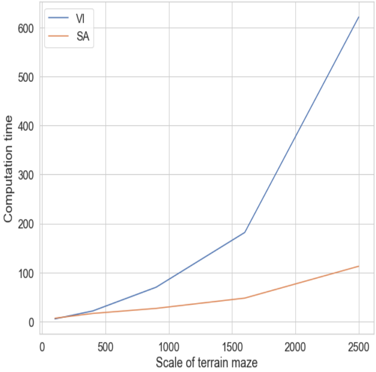

Figure: Efficiency test

Scale	Time (s) SA : VI
100	6.59 : 5.22
400	16.35 : 21.51
900	26.69 : 69.91
1600	47.68 : 181.70
2500	112.72 : 621.45

Table: Computation Time

The algorithm 3 improved efficiency in terms of computation time compared to Value Iteration.

Tic-Tac-Toe Overview

- Value Iteration
- Q-learning (Stochastic Approximation)
- Deep Reinforcement Learning (Stochastic Approximation + Function Approximation)

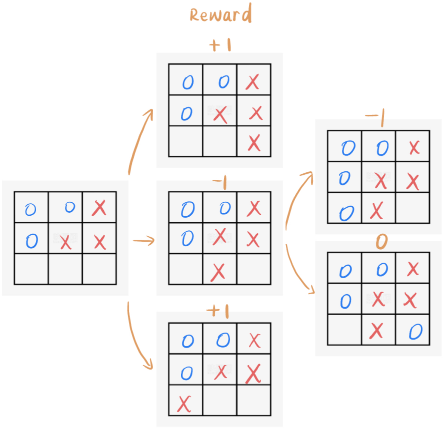

Figure: Example of rewarding state in the learning process.

Game Setting and Notations

Example: (part of a game trajectory)

States trajectory:$\{[-1,0,1,0,-1,0,1,-1,0],\ [-1,0,1,0,-1,0,1,-1,0]\}$
Values for X player: $\{0.9, 0.81\}$
(The current condition benefits player X)
Values for O player:$\{0.1, 0.18\}$
(The current condition is not beneficial for player O)

Choose Action: pick the position with the highest value in available space.

Agent's Value Function

38.2%	44.8%	44.8%
44.7%	98.8%	63.8%
40.7%	49.4%	50.6%

Agent's Value Function

$\boxed{0}$	89.2%	43.8%
44.2%	$\boxed{0}$	50.4%
86.9%	28.1%	61.1%

Figure: Choosing Action based on state-value function.

Value Iteration Method

- Retrieve the initialized state-value pairs (e.g. for player X)

$$\text{Value} = \begin{cases} 1, \text{if X wins} \\ 0, \text{if O wins} \\ 0.5, \text{otherwise} \end{cases}$$

- Apply Value Iteration over **all the states** several times
- Go until convergence (usually not more than 3 loops)

Value-Iteration

- $V_{k+1}(s) = \max_a (R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V_k(s'))$
- - Same reason as the previous problem on Bellman Equation(DP)

To retrieve the optimal policy after the value iteration:

- $\pi(s) = \text{argmax}_a R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V_{k+1}(s')$

Value Iteration Method

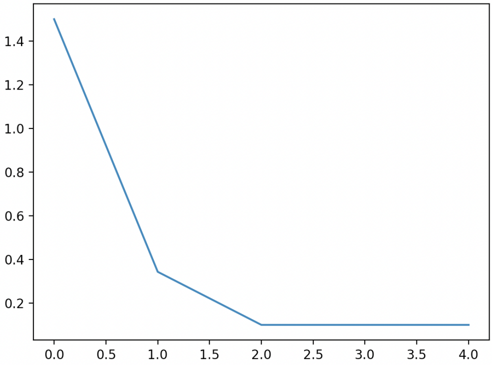

Figure: Convergence of value iteration

Problems on Time & Space Complexity

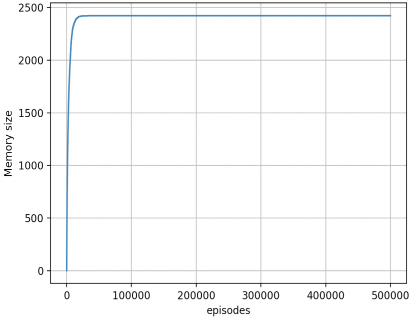

Figure: Cost of Memory Space on 3*3 game

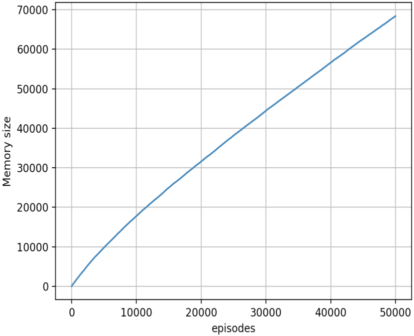

Figure: Cost of Memory Space on 4*4 game

Q-learning Algorithm

Algorithm 4 Epsilon-Greedy Q-Learning Algorithm

Input: α : *learning rate*, γ : *discount factor*, ϵ : *a small number*
Result: A Q-table containing $Q(S, A)$ pair defining estimated optimal policy $\pi^\star$
Initialize $Q(S, A)$ arbitrarily, except $Q(terminal, .)$;
$Q(terminal, .) \leftarrow 0$
for *each episode* **do**
  Initialize state S;
  for *each step in episode* **do**
    $A \leftarrow SELECT - ACTION(Q, S, \epsilon)$;
    Take action A, then observe reward R and next state S';
    $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma max_\alpha Q(S', a) - Q(S, A)]$;
    $S \leftarrow S'$

Q-learning Method

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

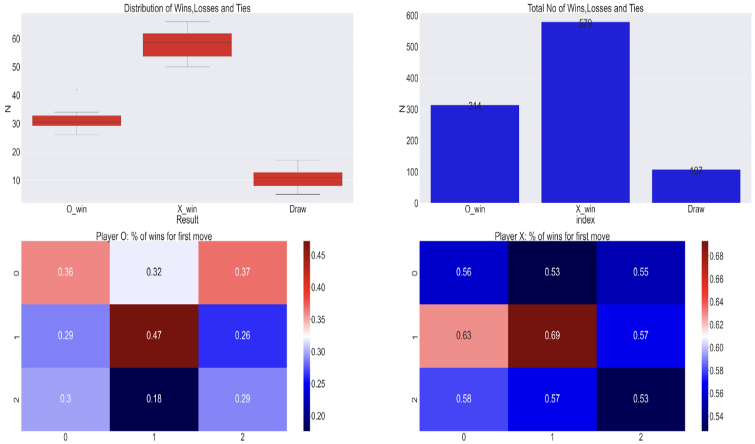

Figure: Experiment Result of Q-learning

Deep Reinforcement Learning Approach: idea

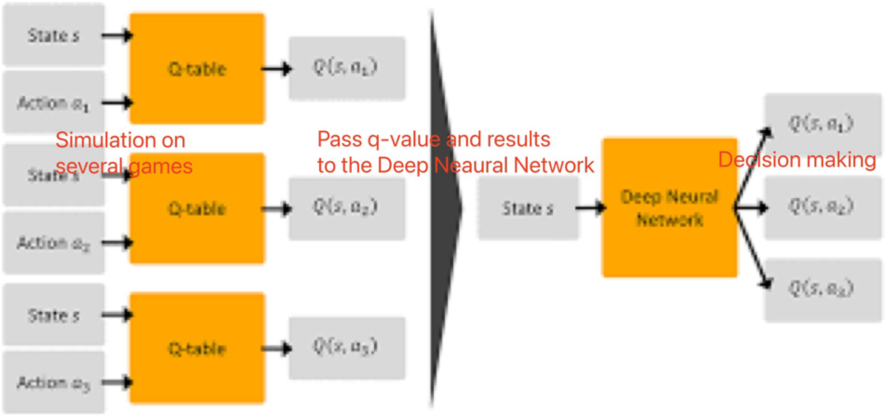

Figure: Logic of Deep Reinforcement Learning in Tic-Tac-Toe

Further: Deep Learning Approach

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 200)	2000
dropout (Dropout)	(None, 200)	0
dense_1 (Dense)	(None, 125)	25125
dense_2 (Dense)	(None, 75)	9450
dropout_1 (Dropout)	(None, 75)	0
dense_3 (Dense)	(None, 25)	1900
dense_4 (Dense)	(None, 3)	78

Figure: NN structure

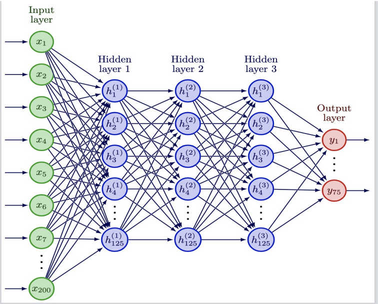

Figure: NN structure

Experiment: Deep Learning Approach

```
Results for player 1:
Wins: 976 (97.6%)
Loss: 0 (0.0%)
Draw: 24 (2.4%)
```

Figure: Trained X v.s. Random O

```
Results for player 2:
Wins: 735 (73.5%)
Loss: 45 (4.5%)
Draw: 220 (22.0%)
```

Figure: Random X v.s. Trained O

```
Results for player 1:
Wins: 294 (29.4%)
Loss: 323 (32.3%)
Draw: 383 (38.3%)

Results for player 2:
Wins: 323 (32.3%)
Loss: 294 (29.4%)
Draw: 383 (38.3%)
```

Figure: Trained X v.s. Trained O

Notice:

- Can not perform as well as Value Iteration in this tic-tac-toe condition.

Possible reasons:

- Total space is not large enough for deep learning
- Neural Network has low explainability for this model

Summary

- We discussed the value-iteration problem and it's variance on different condions
- Key take take-away message:
  - VI with Adaptive Aggregation shows improved efficiency in terms of computation time than basic VI.
  - Random Permuted Cyclic Value Iteration, potentially leads to further improvement in convergence speed. Because it explores the development of heuristic algorithms that can predict permutation.
  - For tic-tac-toe: Q-learning is better than Value Iteration in time and space complexity, DRL only performs well in a more complicated problem
- Potential future directions and limitations: State Aggregation only shows its efficiency in large-scale problems, we can try to find a way to solve this in a small-scale problem.

References

Chen, G., Gaebler, J. D., Peng, M., Sun, C., and Ye, Y. (2021).
An adaptive state aggregation algorithm for markov decision processes.
arXiv preprint arXiv:2107.11053.

Acknowledgement

Thanks to the instructions from Ziniu Li

Thank you!