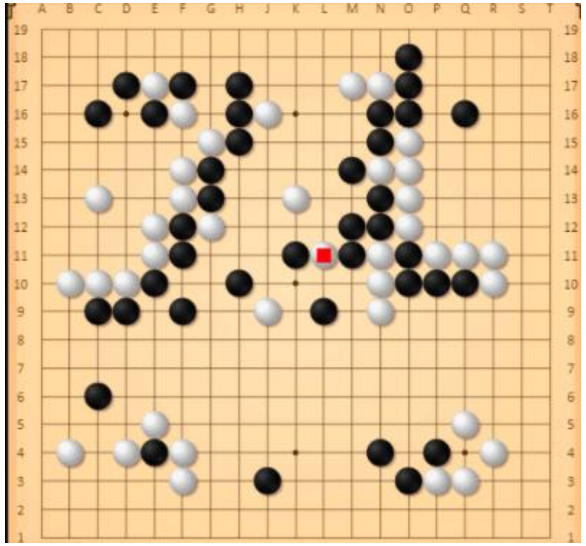
Multi-agent Reinforcement learning with Logic Programming

Jie Meng

Motivation

- We can't explain what AI agent have learned in a game

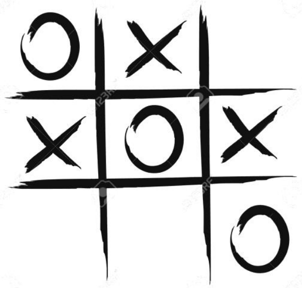
  •AlphaGo vs 李世石



- How to learn interpretable social rules
  - Combine RL and Logic rules

Application Scenario

- Tic-Tac-Toe
- Escalator
- Traffic Rules



Preliminary

- Logic Programming
- Logic Programming in Reinforcement Learning

Preliminary: Logic Programming

- Inductive Logic Programming(ILP)
  - clause: $\alpha \leftarrow \alpha_1, \ldots, \alpha_m$
  - atom: $p(t_1, t_2, \ldots, t_n)$
    where p is a n-ary predicate and $t_1, t_2, \ldots, t_n$ are terms
  - ground atom: an atom contains no variables

- ILP problem
  - An ILP problem is a tuple (*B*; *P*; *N*) of ground atoms, where:
    - B is a set of background assumptions, a set of ground atoms.
    - P is a set of positive instances - examples taken from the extension of the target predicate to be learned
    - N is a set of negative instances - examples taken outside the extension of the target predicate

Preliminary: Logic Programming

An ILP problem

$$\mathcal{B} = \{zero(0), succ(0;1), succ(1;2), succ(2;3), \ldots\}$$
$$\mathcal{N} = \{even(1), even(3), even(5), even(7), \ldots\}$$
$$\mathcal{P} = \{even(0), even(2), even(4), even(6), \ldots\}$$

The aim of ILP is to construct a logic program that explains the positive instances and rejects the negative instances.

One Solution

$$even(X) \leftarrow zero(X)$$
$$even(X) \leftarrow even(Y), succ2(Y, X)$$
$$succ2(X, Y) \leftarrow succ(X, Z), succ(Z, Y)$$

Preliminary: Logic Programming in Reinforcement Learning

- Differentiable ILP

Generating Atoms

Generating atoms from templates:

$$L = (target\ Predicate; arity; C)$$

Valuations

Given a set G of n ground atoms, a valuation is a vector $[0,1]^n$ mapping each ground atom $\gamma_i \in G$ to the real unit interval.

Example

Language template: $P_i = \{p/0, q/1, r/2\}$; $C = \{a, b\}$

$$\begin{array}{llll} \perp \mapsto 0.0 & p() \mapsto 0.0 & q(a) \mapsto 0.1 & q(b) \mapsto 0.3 \\ r(a,a) \mapsto 0.7 & r(a,b) \mapsto 0.1 & r(b,a) \mapsto 0.4 & r(b,b) \mapsto 0.2 \end{array}$$

Preliminary: Logic Programming in Reinforcement Learning

Generating Clauses

Generating atoms from templates: $C_p^1, C_p^2, \cdots C_p^n$, where C_p^i is a clause for target predicate P. e.g. $p(X) \leftarrow r(Y), q(X, Y)$

Weights

$W_p[j, k]$ represents how strongly the system believes that the pair of clauses (C_p^j, C_p^k) is the right way to define the intensional predicate p.

Updating Valuations

$V^0 = (v_0, v_1, ..., v_n) \rightarrow f_{infer}(clauses, weight, T)$
$\rightarrow$ conclusion valuations

Preliminary: Logic Programming in Reinforcement Learning

Frame

$$V^0 = \begin{bmatrix} b_1 \\ \cdots \\ b_m \\ f_1 \\ \cdots \\ f_n \end{bmatrix} = \begin{bmatrix} 1 \\ \cdots \\ 1 \\ \cdots \\ 0 \\ \cdots \\ 0 \end{bmatrix} \Rightarrow f_{infer}(\begin{bmatrix} c_1 & c_2 \\ \cdots & \cdots \\ c_k & c_m \end{bmatrix}, \mathbf{W}, T) \Rightarrow V^T \rightarrow \begin{pmatrix} prob(n_i) \\ \cdots \\ prob(p_i) \end{pmatrix}$$

Loss

Calculating the probability of the label λ given the atom α.

$$p(\lambda|\alpha, W, \Pi, \mathcal{L}, \mathcal{B}) = f_{\text{extract}}\left(f_{\text{infer}}\left(f_{\text{convert}}(\mathcal{B}), f_{\text{generate}}(\Pi, \mathcal{L}), W, T\right), \alpha\right)$$

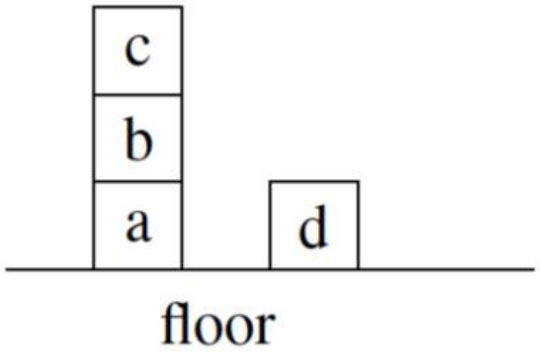
$$loss = -\mathop{\mathbb{E}}_{(\alpha,\lambda)\sim\Lambda}[\lambda \cdot \log p(\lambda|\alpha, W, \Pi, \mathcal{L}, \mathcal{B}) + (1-\lambda) \cdot \log(1-p(\lambda|\alpha, W, \Pi, \mathcal{L}, \mathcal{B}))]$$

Preliminary: Logic Programming in Reinforcement Learning

MDP with logic interpretation

A triple (M, p_S, p_A):

- $p_A : [0,1]^{|D|} \to [0,1]^{|A|}$. Maps the valuation (or score) of a set of atoms D to the probability of actions.
- $p_S : S \to 2^G$. Maps each states to ground atoms.



state: $top(d)$,
$top(c), on(d, floor)$,
$on(c, b), on(b, a)$,
$on(a, floor)$

background: $isFloor(floor)$

Figure: Block Manipuation

Problem Statement

A Markov Game is defined by a tuple $\left(\mathcal{N}, \mathcal{S}, \{\mathcal{A}^i\}_{i\in\mathcal{N}}, \mathcal{P}, \{R^i\}_{i\in\mathcal{N}}, \gamma\right)$, where $\mathcal{N}$ denotes the set of N agents. $\mathcal{A}^i$ denotes the action space of agent i. $\mathcal{P}$ is transition probability. $\mathcal{S}$ denotes the state space.

Social Rules is defined by a map, $F : S \times \{\mathcal{A}\} \rightarrow \{(0,1)\}^n$. where $\{\mathcal{A}\}$ is action space $(a_1, a_2, ..., a_n)$.

Map F can be represented logic formula, e.g. First-order logic.

Problem Statement

Under the limit of social rules, the joint policy π will be $\pi(a|s) := \prod_{i \in \mathcal{N}} \pi^i \left(a^i|s\right)|_{F(s)}$.
Optimize Value Function and get optimal π^*:

$$R = \sum_t \gamma^t R_k^t, R_k = \{0, -1, 1\}$$

$$V^i_{\pi^i,\pi^{-i}}(s) := \mathbb{E}\left[\sum_{t \geq 0} \gamma^t R^i(s_t, a_t, s_{t+1}) \middle| a_t^i \sim \pi^i(\cdot|s_t), s_0 = s\right],$$

where i represents the indices of all agents in N except agent i.

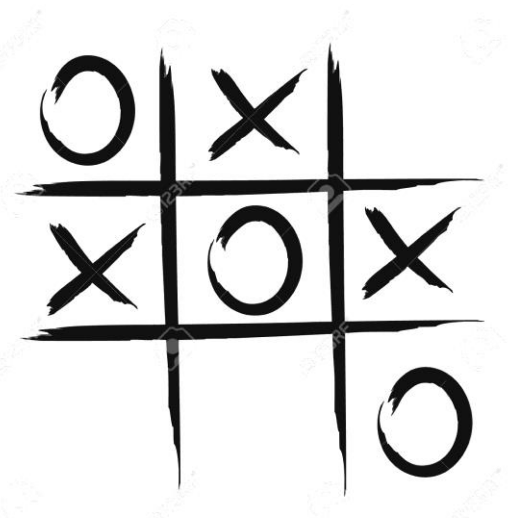
Experiment

- Tic-Tac-Toe
  - State S: (i, j) -> {-1,0,1}
    - Mine(i,j) -> 1
    - Opponent(i,j) -> -1
    - Empty(i,j) -> 0
  - Action A:

    Place(x,y)
  - Reward R:

$$R = \sum_t \gamma^t R_k^t, R_k = \{0, -1, 1\}$$

  - Auxiliary predicate: Invented(i) , i = 1,2,3,4



Experiment ---Tic-Tac-Toe

- Invalid action problem:
  - Place(x,y) :- Mine(x,y), Mine(x,z)

- One solution:
  - Give penalty for invalid action

$$R = \sum_t \gamma^t p_t R_k^t, R_k = \{0, -1, 1\}$$

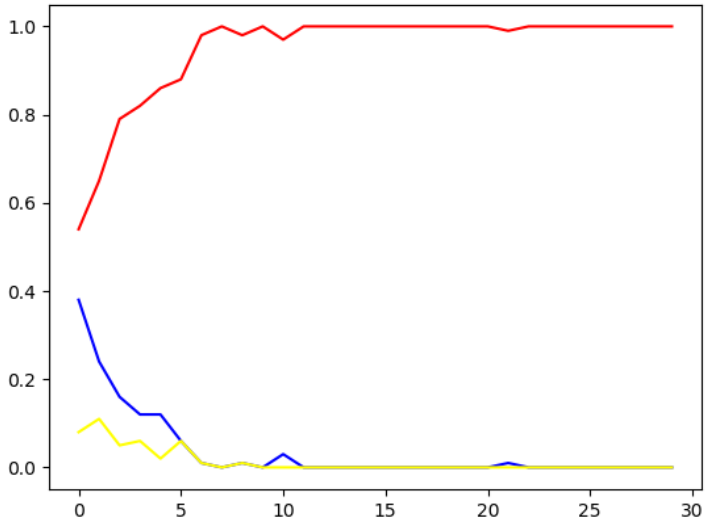
Tic-Tac-Toe: Results

- State

```
([[ 1.,  0., -1.],
  [ 0.,  0.,  0.],
  [-1.,  0.,  1.]])
```

- Logic rules:

```
place(X,Y):-invented4(X),invented4(Y)
invented4(X):-empty(X,X),succ(X,Y)
```



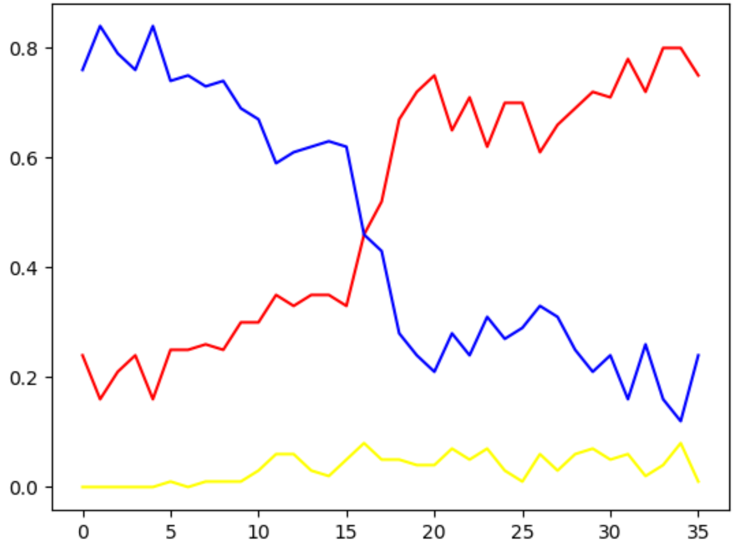
Tic-Tac-Toe: Results

- State:

```
[[-1.,  0.,  0.],
 [ 0.,  0.,  0.],
 [ 1.,  0., -1.]]
```

- Logic rules:

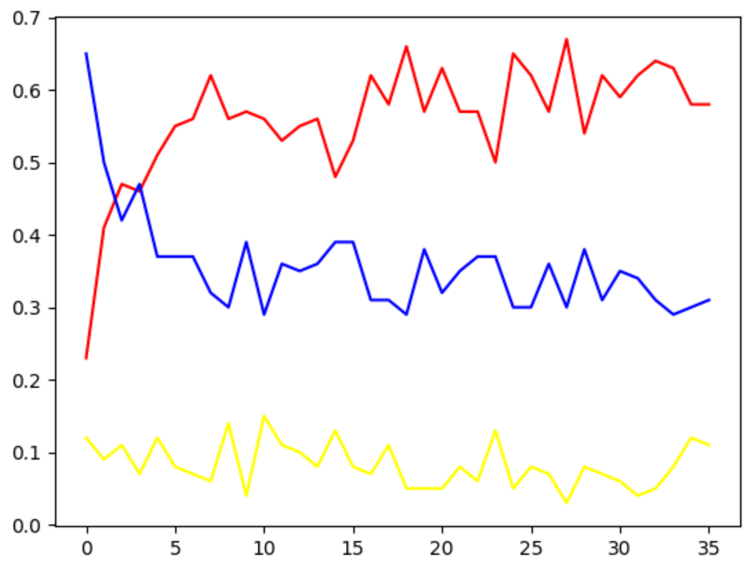
```
place(X,Y):-empty(X,X),empty(Y,Y)
```



Tic-Tac-Toe: Results

- State:

```
[[ 0.   0.   0.]
 [ 0.   1.   0.]
 [-1.   0.  -1.]]
```



- Logic rules:

```
place(X,Y):-empty(Z,Y),succ(Y,X)
```

Future Work

- Using more powerful logic representation: SDD
- Two more experiments: Escalator Traffic Rules
- Combine social rules learning and multi-agent learning

SDD: Sentential Decision Diagram