dPASP: Possíveis Aplicações

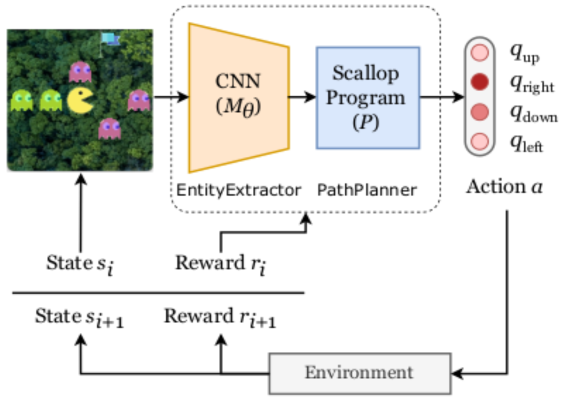
Pacman e Shogi

PacMan

- Dado um labirinto, o programa deve traçar uma rota para o pacman chegar na bandeira sem encontrar inimigos no caminho;
- Avaliação de taxa de sucesso baseia-se na quantidade de vezes que o pacman chegou na bandeira dentro de um certo tempo;
- Treinamento é realizado com conjuntos de estados em um labirinto que levam à vitória.
- Q-Learning leva 50 mil episódios para atingir 84,9% de sucesso;
- Abordagem da Scallop necessita de apenas 50 episódios para obter 99,4% de taxa de sucesso.

PacMan

(a) Three states of one gameplay session.



(b) Architecture of application with Scallop.

```
[Test Epoch 0] Success 199/200 (99.50%): 100%|██████████| 200/200 [01:37<00:00,  2.04it/s]
```

PacMan

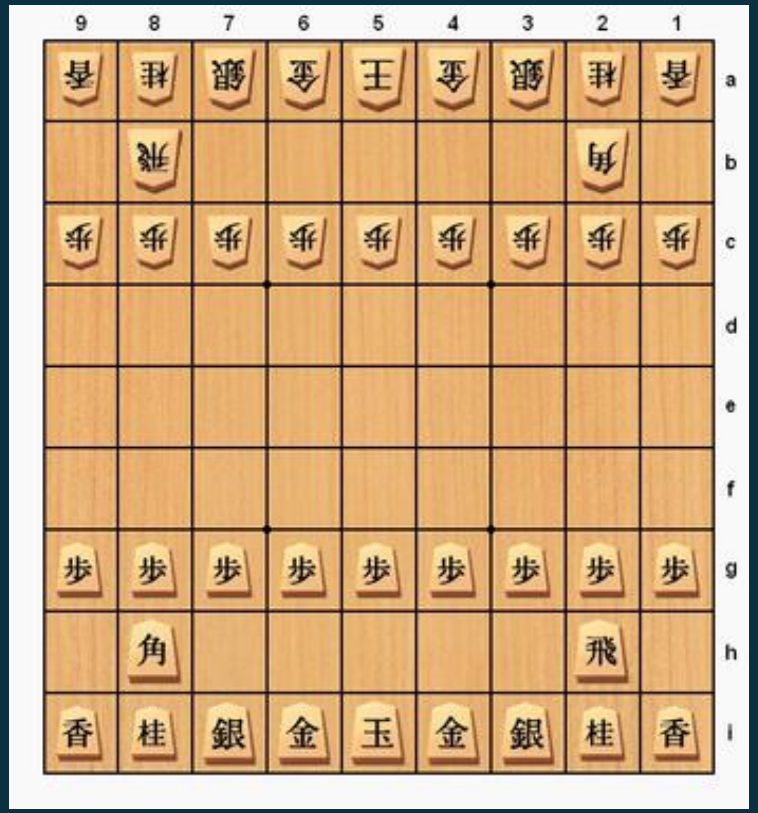
```
// Static input facts
type grid_node(x: usize, y: usize)

// Input from neural networks
type actor(x: usize, y: usize)
type goal(x: usize, y: usize)
type enemy(x: usize, y: usize)

// Possible actions to take
type Action = UP | RIGHT | DOWN | LEFT

// ========== YOUR CODE START HERE ==========

// ** Problem 1: safe_node **
// (x, y) is a safe node if it is a grid node and does not contain an enemy
type safe_node(x: usize, y: usize)
rel safe_node(x, y) = grid_node(x, y) and not enemy(x, y)

// ** Problem 2: edge **
// There is an (safe) edge between safe nodes (x1, y1) and (x2, y2) if
// taking the action `a` can move the actor from (x1, y1) to (x2, y2)
type edge(x1: usize, y1: usize, x2: usize, y2: usize, a: Action)
rel edge(x, y, x, yp, UP) :- safe_node(x, y), safe_node(x, yp), yp == y + 1
rel edge(x, y, xp, y, RIGHT) :- safe_node(x, y), safe_node(xp , y), xp == x + 1
rel edge(x, y, x, yp, DOWN) :- safe_node(x, y), safe_node(x, yp), yp == y - 1
rel edge(x, y, xp, y, LEFT) :- safe_node(x, y), safe_node(xp, y), xp == x - 1
```

```
// Given the current actor position, taking the action `a` would move the
// actor to the position (x, y)
type next_position(a: Action, x: usize, y: usize)
rel next_position(a, xp, yp) :- actor(x, y), edge(x, y, xp, yp, a)

// ** Problem 3: path **
// There is a (safe) path between safe nodes (x1, y1) and (x2, y2) if
// there is a series of safe edges connecting the two nodes.
// Note that self-path is also a safe path.
type path(x1: usize, y1: usize, x2: usize, y2: usize)
rel path(x, y, x, y) :- next_position(_, x, y)
// rel path(x1, y1, x2, y2) :- edge(x1, y1, x2, y2, _)
rel path(x1, y1, x3, y3) :- path(x1, y1, x2, y2), edge(x2, y2, x3, y3, _)

// ** Problem 5: next_action **
// We pick the action `a` as the next action if, after moving to the next
// position with `a`, we have a safe path from the next position to the goal
type next_action(a: Action)
rel next_action(a) :- next_position(a, x, y), goal(xg, yg), path(x, y, xg, yg)

// ========== YOUR CODE END HERE ==========

// Constraint violation; please keep these as is
rel too_many_goal() = n := count(x, y: goal(x, y)), n > 1
rel too_many_enemy() = n := count(x, y: enemy(x, y)), n > 5
rel violation() = too_many_goal() or too_many_enemy()
```

Ideia de projeto

- Criar um programa capaz de interpretar um tabuleiro de shogi e dizer qual a melhor jogada;
- Treinamento do modelo seria feito baseado em "exercícios" de check-mate;
- A dificuldade desses exercícios progride conforme o modelo aprende;
- Necessário modelar no dpasp:
  - O movimento de cada peça;
  - Limitação de movimentação;
  - Promoções de cada peça;
  - Condições de check-mate;
- Necessário adaptar modelo do pacman para shogi.

Ideia de projeto

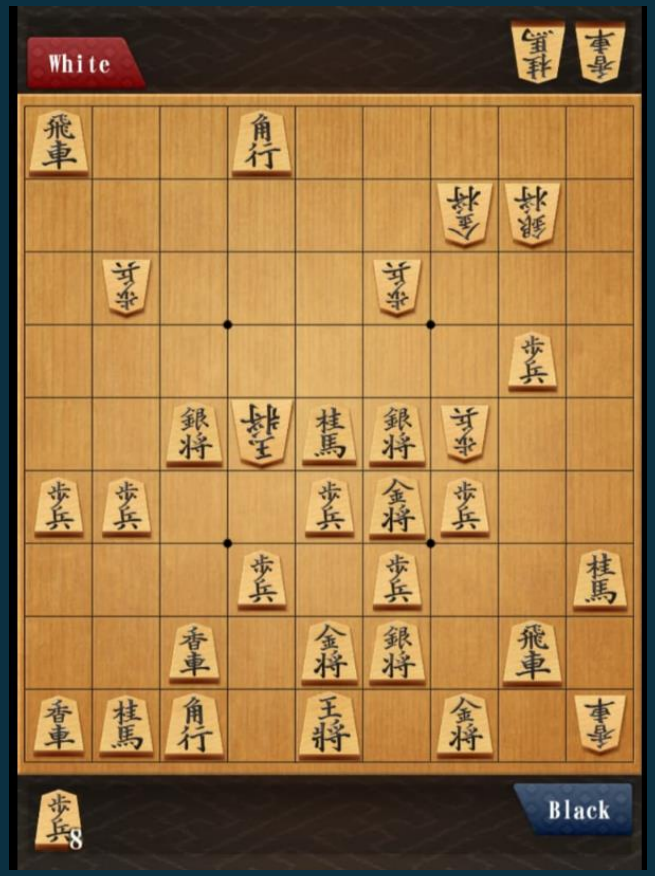


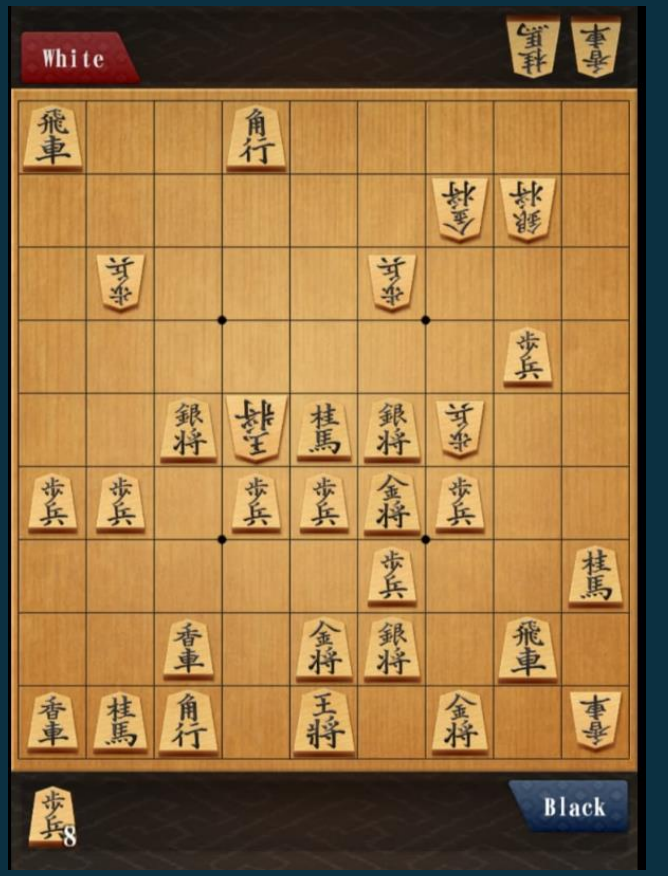
SHOGI

Name	Koma	Move	Promoted Name	Koma	Move
Reigning King (*osho*—royal general)					
Challenging King (*gyokusho*—jeweled general)					
Bishop (*kakugyo*—angle mover)			Horse (*ryuma*—swift horse)		
Rook (*hisha*—flying chariot)			Dragon (*Ryuo*—dragon king)		
Gold General (*kinsho*)					
Silver General (*ginsho*)			Promoted Silver (*narigin*)		
Knight* (*keima*—laureled horse)			Promoted Knight (*narikei*—promoted laurel)		
Lance (*kyosha*—incense chariot)			Promoted Lance (*narikyo*—promoted incense)		
Pawn (*fuhyo*—foot soldier)			Promoted Pawn (*tonkin*—reaches gold)		

*Only the knight may ignore intervening pieces.

Ideia de projeto





Referências

- Scallop: A Language for Neurosymbolic Programming;
- https://www.scallop-lang.org/doc/index.html;
- https://github.com/scallop-lang/scallop;
- https://www.scallop-lang.org/pldi23/tutorial.html#section-25.