コンピュータゲームプレイヤ

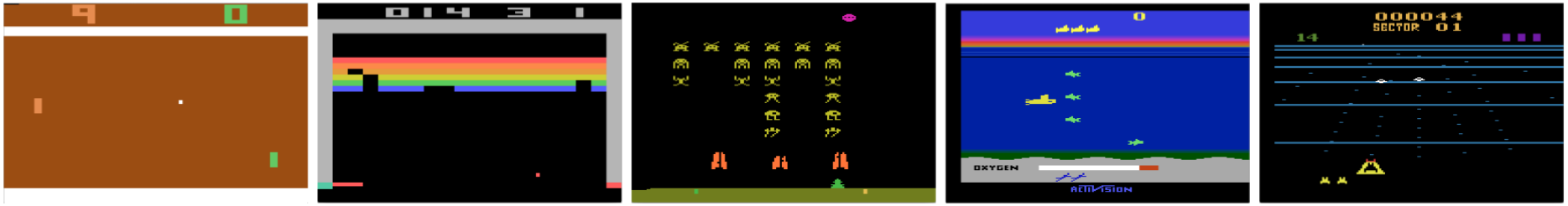
鶴岡 慶雅

工学部 電子情報工学科
情報理工学系研究科 電子情報学専攻

DEEP Q-NETWORK (DQN)

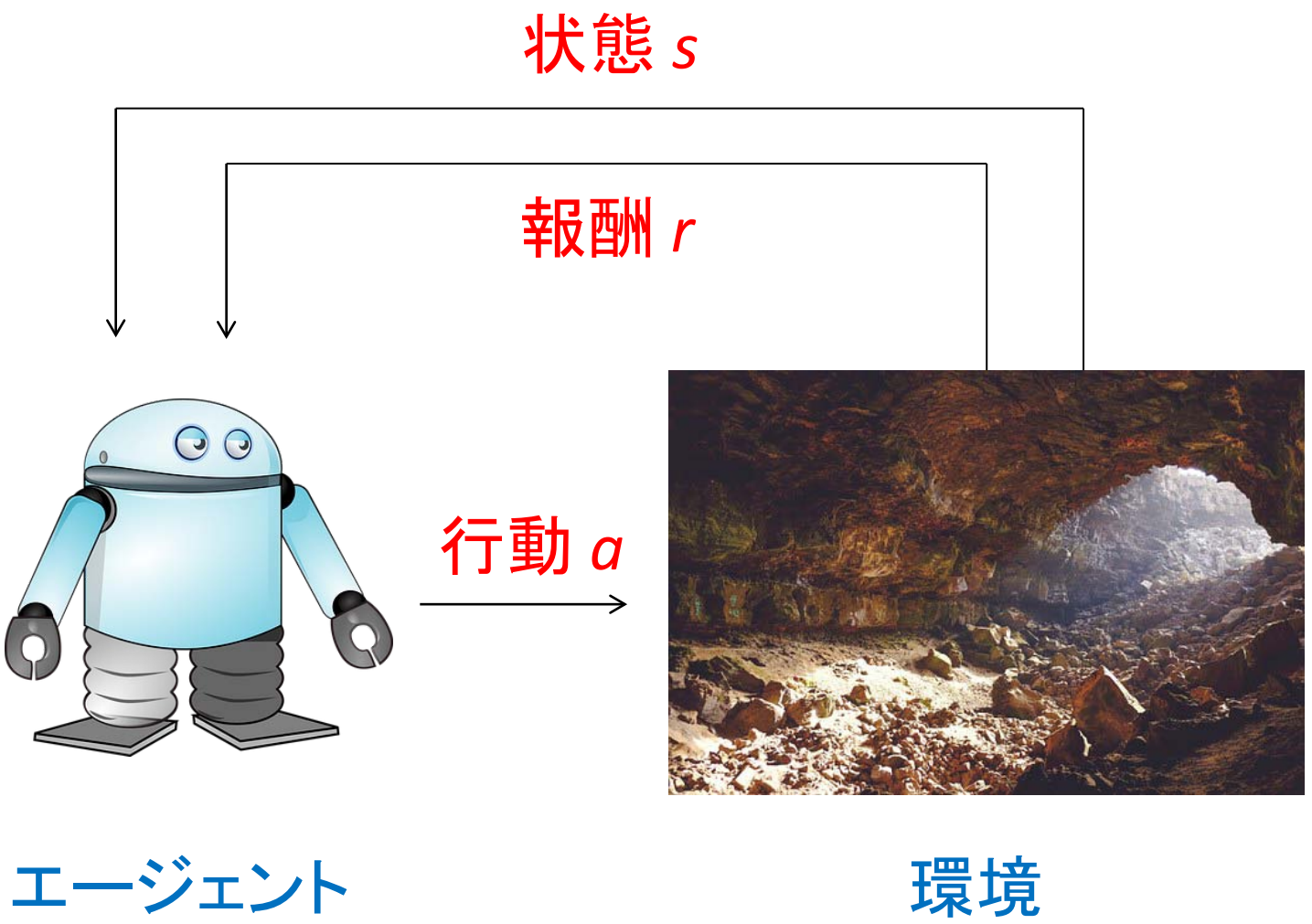
Deep Q-Network (Mnih et al., 2015)

- Atari 2600 Games
  - ブロック崩し、スペースインベーダー、ピンポン、etc.



- 同一のプログラムですべてのゲームを学習
  - CNN＋強化学習（Q-Learning）
  - https://www.youtube.com/watch?v=AVg_YIp09ps

強化学習（Reinforcement Learning, RL）



MDP

- マルコフ決定過程（Markov decision Process）
  - 状態集合 S
  - 行動集合 A
  - 状態遷移関数 $P(s'|s,a)$
    - 状態 s において行動 a とった場合に状態 s' に遷移する確率
  - 報酬関数 $R(s,a,s')$
    - 状態 s から行動 a によって状態 s' に遷移したときに得られる報酬

強化学習

- エージェントの目的
  - 現在から未来にわたる累積報酬を最大化

$$g_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

- Bellman 方程式

$$Q^*(s,a) = \sum_{s'} P(s'|s,a)\left(R(s,a,s') + \gamma \max_{a'} Q^*(s',a')\right)$$

状態 s で行動 a をとり、その後最善の行動をとり続けた場合に得られる報酬の期待値

Q学習

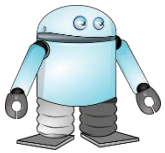
- Q(*s*, *a*) を学習
  - *Q*(*s*, *a*): 状態 *s* で行動 *a* をとった場合に将来得られる報酬の総和の期待値（の予測値）
  - 行動するたびに予測値を更新

$$Q(s_t, a) \leftarrow Q(s_t, a) + \alpha\left(r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right)$$

一歩先で得られる
より正確な予測値

現在の
予測値

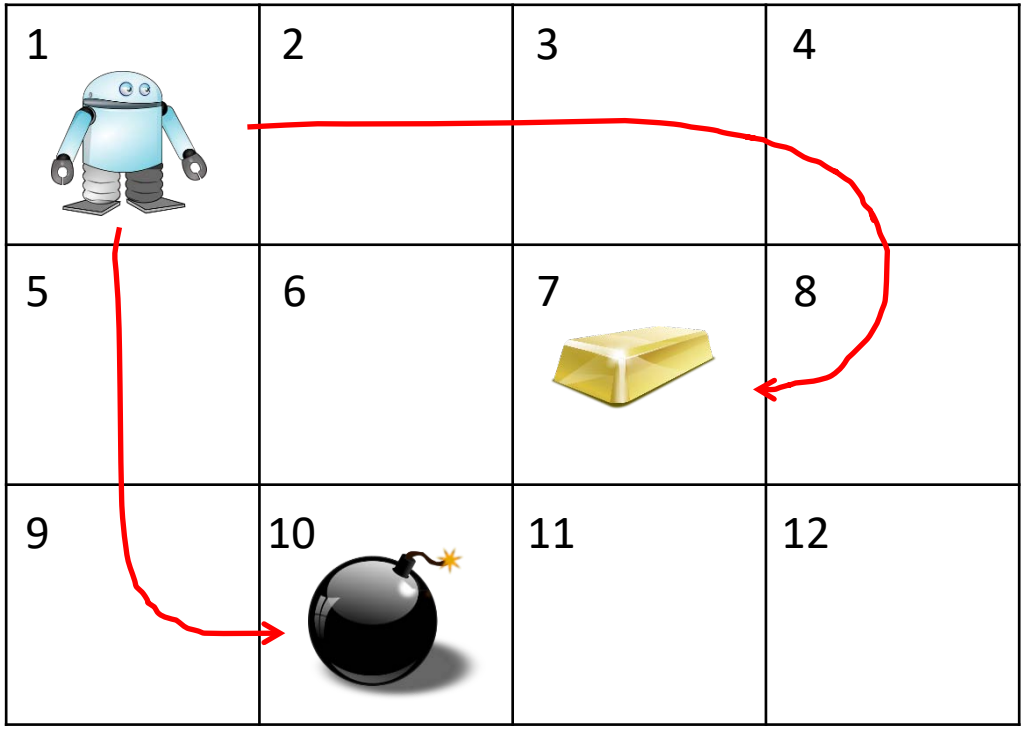
初期状態

1 	2	3	4
5	6	7 	8
9	10 	11	12

初期状態

	Up	Down	Left	Right	End
1	0	0	0	0	
2	0	0	0	0	
3	0	0	0	0	
4	0	0	0	0	
5	0	0	0	0	
6	0	0	0	0	
7					0
8	0	0	0	0	
9	0	0	0	0	
10					0
11	0	0	0	0	
12	0	0	0	0	

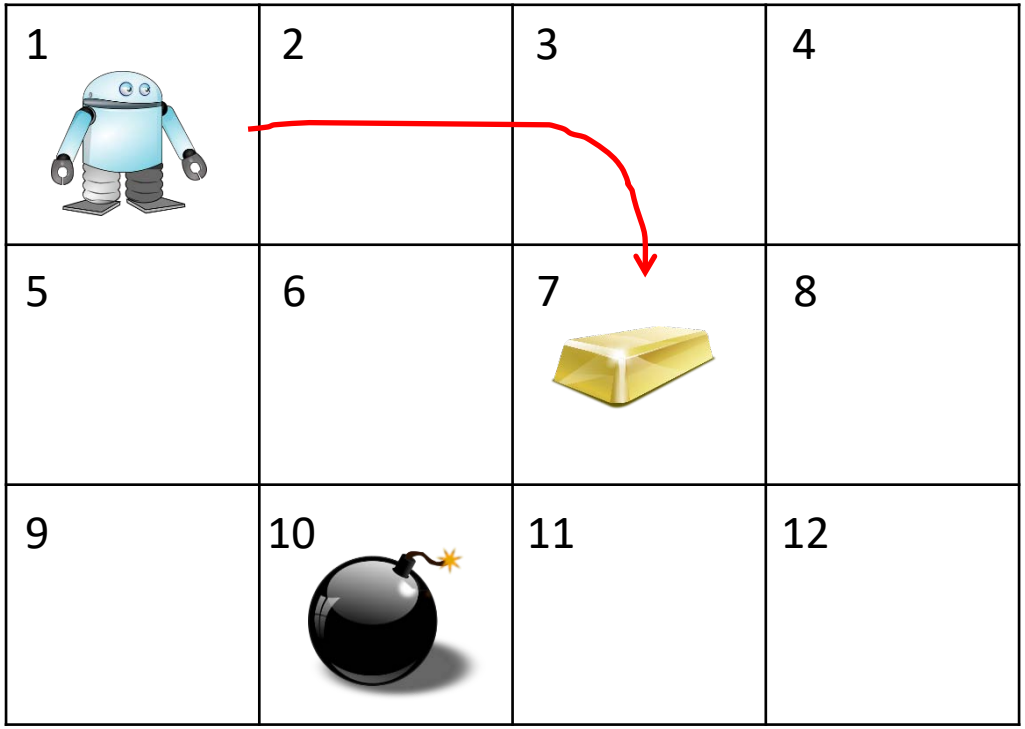
状態7と状態10を経験



状態7と状態10を経験した後

	Up	Down	Left	Right	End
1	0	0	0	0	
2	0	0	0	0	
3	0	0	0	0	
4	0	0	0	0	
5	0	0	0	0	
6	0	0	0	0	
7					**100**
8	0	0	0	0	
9	0	0	0	0	
10					**-100**
11	0	0	0	0	
12	0	0	0	0	

状態3を経由して状態7に到達



状態3を経由して状態7に到達

	Up	Down	Left	Right	End
1	0	0	0	0	
2	0	0	0	0	
3	0	**90**	0	0	
4	0	0	0	0	
5	0	0	0	0	
6	0	0	0	0	
7					100
8	0	0	0	0	
9	0	0	0	0	
10					-100
11	0	0	0	0	
12	0	0	0	0	

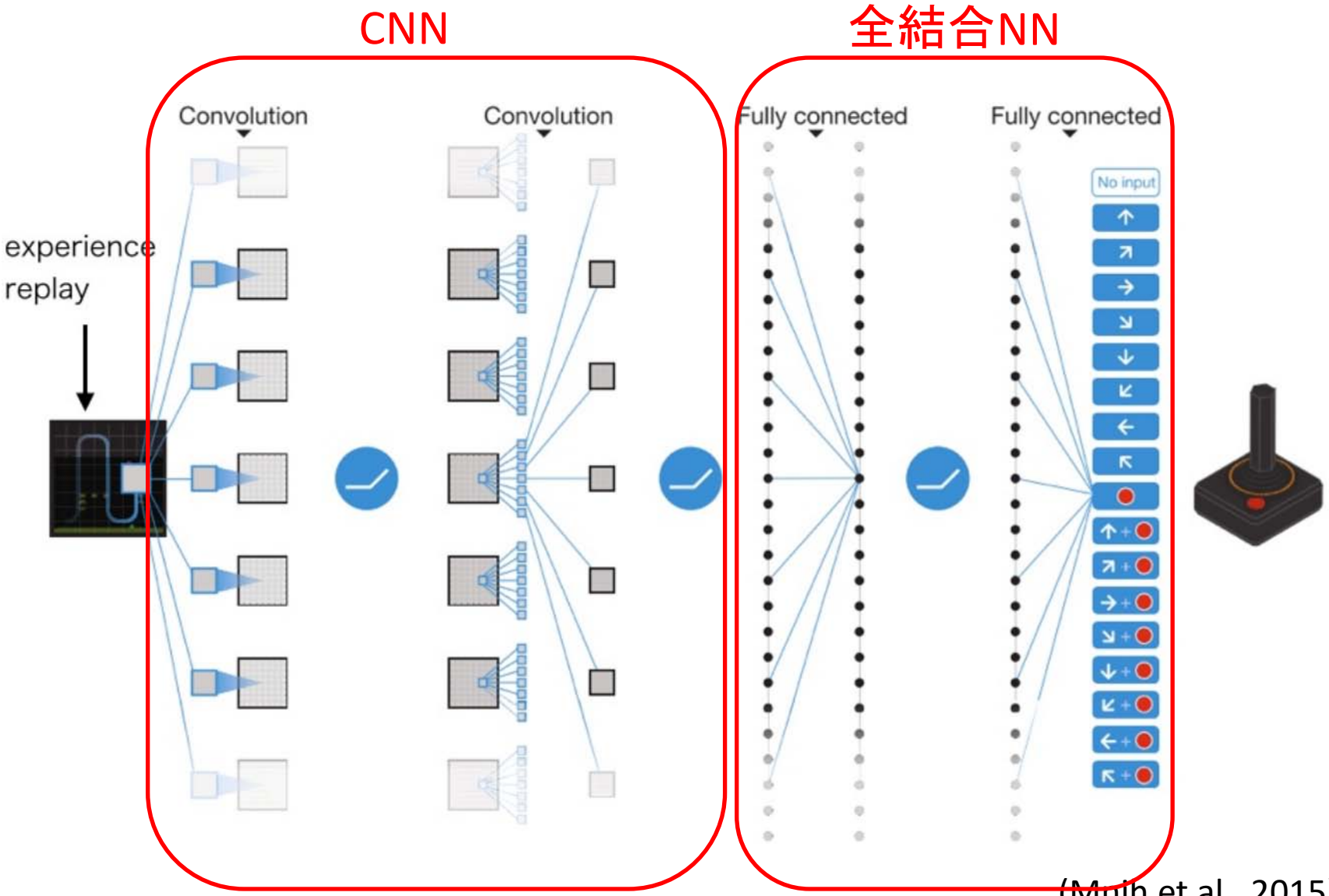
関数近似によるQ学習

- テーブルによるQ学習の問題
  - メモリ使用量が状態空間の大きさに比例
  - 汎化能力がない

- 関数近似によるQ学習
  - ニューラルネットワーク等で $Q(s,a;\theta)$ を実現
  - 最小化

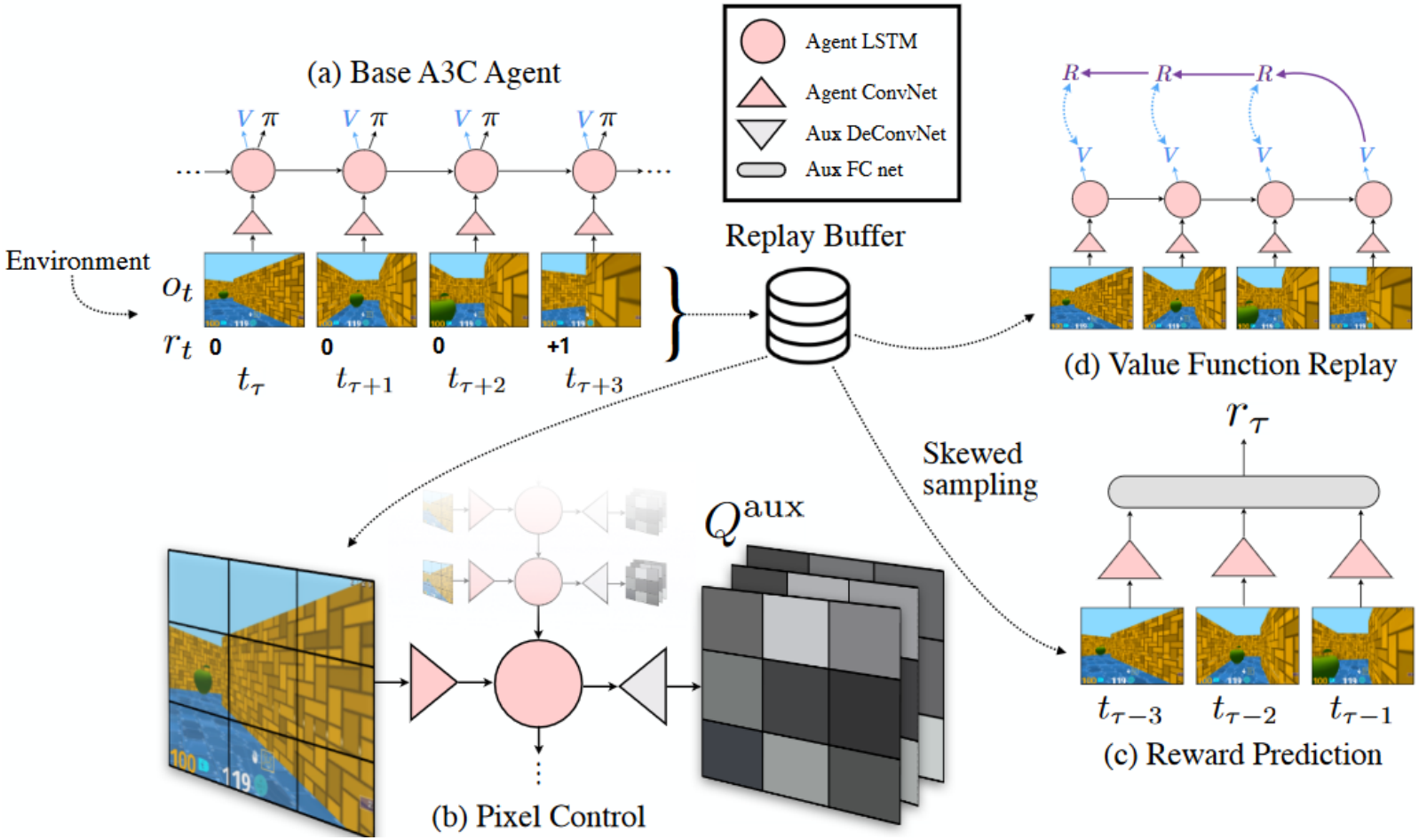
$$L(\theta_i) = \mathrm{E}\left[\left(r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) - Q(s,a)\right)^2\right]$$

Deep Q-Network



(Mnih et al., 2015)

Reinforcement Learning with Unsupervised Auxiliary Tasks (Jaderberg et al., 2016)



コンピュータポーカー

Texas Hold’em

- Texas Hold’em
  - 最も人気のあるポーカーのひとつ

ゲーム理論超入門

- 利得表・戦略・ゼロサム

じゃんけんゲーム

		プレイヤBの戦略		
		グー	チョキ	パー
プレイヤAの戦略	グー	0	+1	−1
	チョキ	−1	0	+1
	パー	+1	−1	0

- 純粋戦略（pure strategy）
  - ある戦略を確定的に選ぶ

- 混合戦略（mixed strategy）
  - 戦略を**確率的**に選ぶ
  - 例 ［グー（0.5） チョキ（0.3） パー（0.2）］

ナッシュ均衡

じゃんけんゲーム

		プレイヤBの戦略		
		グー	チョキ	パー
プレイヤAの戦略	グー	0	+1	−1
	チョキ	−1	0	+1
	パー	+1	−1	0

- ナッシュ均衡（Nash equilibrium）
  - どのプレイヤも自分（だけ）が戦略を変更することによって得をすることがない状態
  - 戦略の組が互いに最適応答になっている
- じゃんけんゲーム
  - ナッシュ均衡は純粋戦略では存在しない
  - 混合戦略 ［グー（1/3） チョキ（1/3） パー（1/3）］

問題

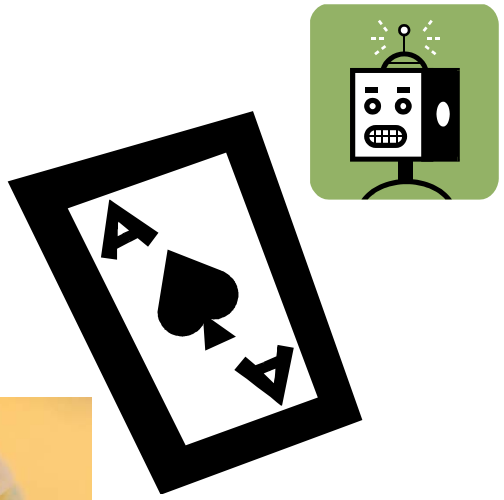
- グー、チョキ、パーで利得が違う場合
  - グーで勝ったら３点
  - チョキで勝ったら２点
  - パーで勝ったら１点

- ナッシュ均衡戦略は？
  ① グーの確率＞チョキの確率＞パーの確率
  ② パーの確率＞チョキの確率＞グーの確率
  ③ それ以外

  答え　③ グー（1/3） チョキ（1/6） パー（1/2）

One-card Poker

- 極限まで単純化されたポーカー
  - 1対1
  - カードは1枚
  - 強いカードを持っている方が勝ち



- ラウンド
  - 最低掛け金は $1
  - プレイヤ A の手番
    - Bet $0 or $1
  - プレイヤ B の手番
    - Call, Raise or Fold
  - (プレイヤ B が Raise した場合のみ)プレイヤ A の手番
    - Call or Fold

http://www.cs.cmu.edu/~ggordon/poker/

プレイヤAのナッシュ均衡戦略

1st round

カード	Bet する確率
2	0.454
3	0.443
4	0.254
5	0.000
6	0.000
7	0.000
8	0.000
9	0.422
10	0.549
J	0.598
Q	0.615
K	0.628
A	0.641

2nd round

カード	Bet する確率
2	0.000
3	0.000
4	0.169
5	0.269
6	0.429
7	0.610
8	0.760
9	1.000
10	1.000
J	1.000
Q	1.000
K	1.000
A	1.000

http://www.cs.cmu.edu/~ggordon/poker/

プレイヤBのナッシュ均衡戦略

Bet 0$ に対して

カード	Bet する確率
2	1.000
3	1.000
4	0.000
5	0.000
6	0.000
7	0.000
8	0.000
9	1.000
10	1.000
J	1.000
Q	1.000
K	1.000
A	1.000

Bet 1$ に対して

カード	Bet する確率
2	0.000
3	0.000
4	0.000
5	0.251
6	0.408
7	0.583
8	0.759
9	1.000
10	1.000
J	1.000
Q	1.000
K	1.000
A	1.000

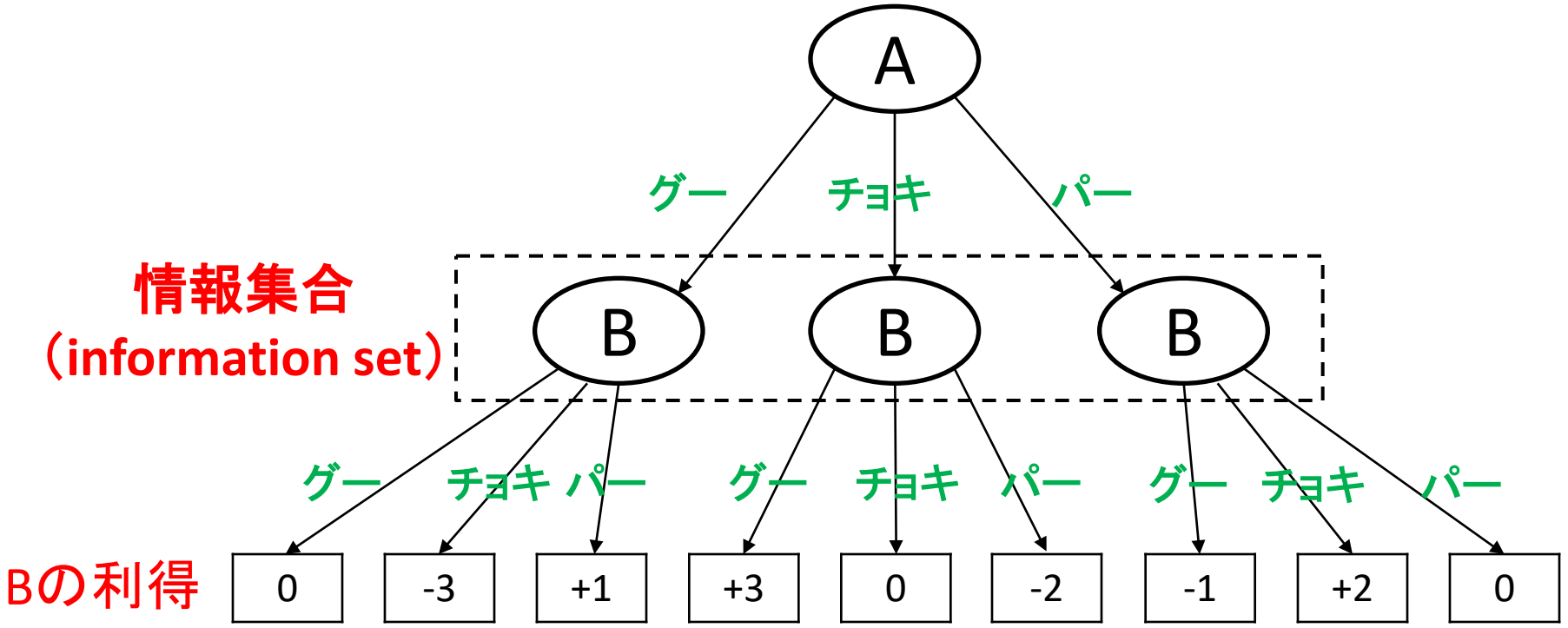
http://www.cs.cmu.edu/~ggordon/poker/

ナッシュ均衡

- ポーカーの場合
  - Rhode Island Hold'em
    - カード3枚のポーカー
    - 9億行 x 9億列　⇒　抽象化　100万行 x 100万列
  - Texas Hold'em
    - 相当に粗い抽象化をしないと解けない

展開形による表現

- 展開形（extensive-form）



Counterfactual Regret Minimization (CFR)

- Average overall regret

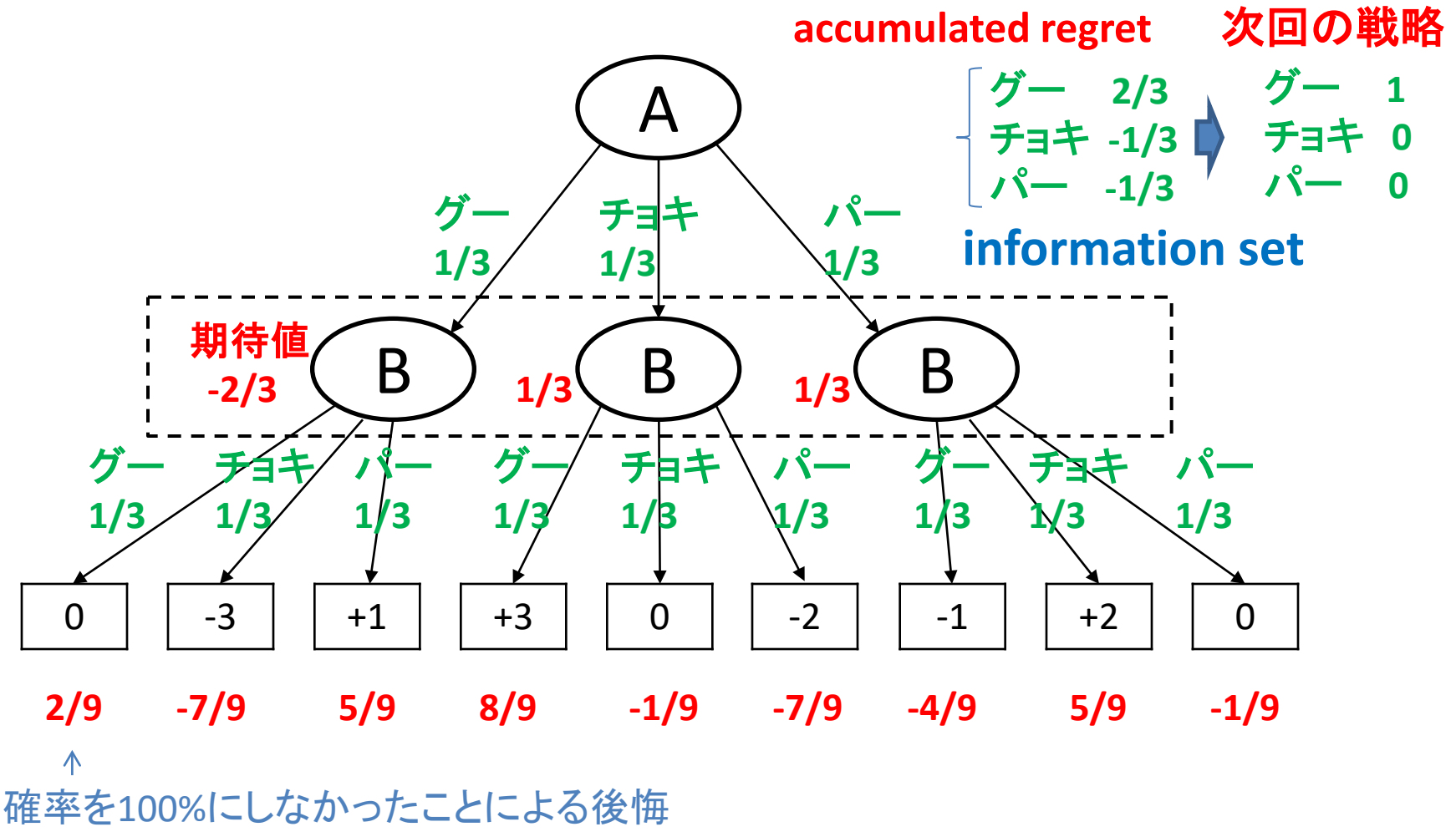
$$R_i^T = \frac{1}{T} \max_{\sigma_i^* \in \Sigma_i} \sum_{t=1}^{T} \left(u_i\left(\sigma_i^*, \sigma_{-i}^t\right) - u_i\left(\sigma^t\right) \right)$$

  - Regret: 結果的に見てベストであった戦略との効用の差
  - Regret が 0 に近づく
    ⇒　平均戦略によるナッシュ均衡

- 情報集合（information set）と overall regret
  - 個々の情報集合で独立に regret を最小化
  - Regret matching によって各プレイヤの戦略を更新

Regret matching 例

- 階段じゃんけん（Bからみた効用）



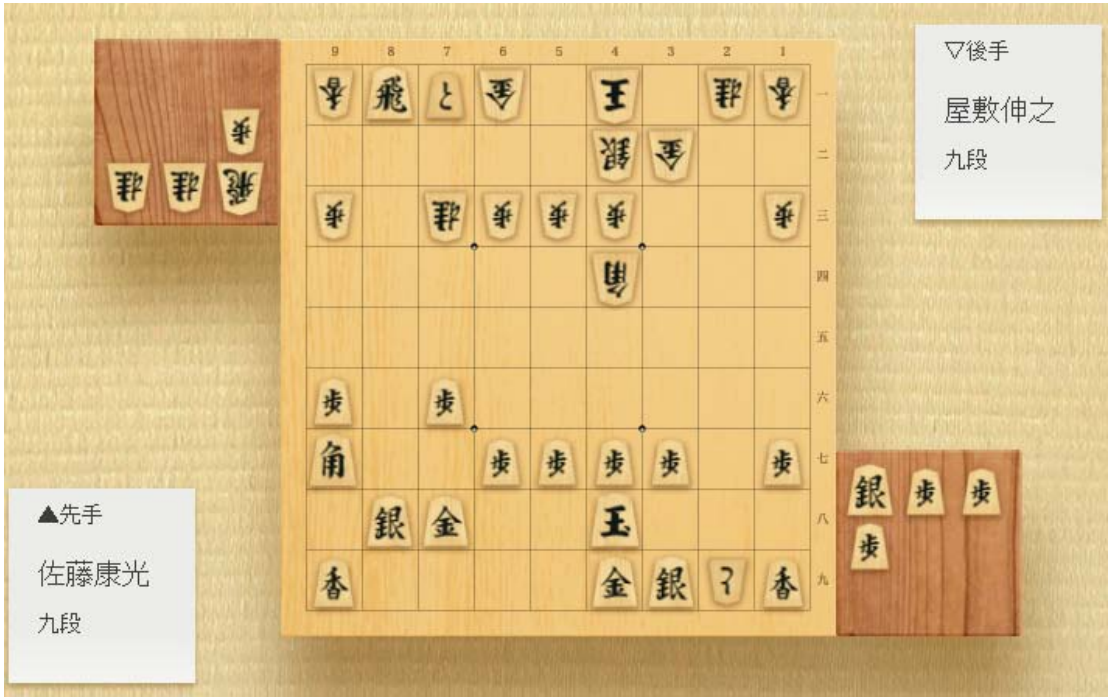
↑
グーの確率を100%にしなかったことによる後悔

vs世界チャンピオン

- Heads-up Limit Texas hold'em
  - 1対1
  - 掛け金は離散的に上昇
- Polaris 2.0
  - University of Alberta
  - CFR
- 2008 Gaming Life Expo
  - 3 wins, 2 losses, 1 tie

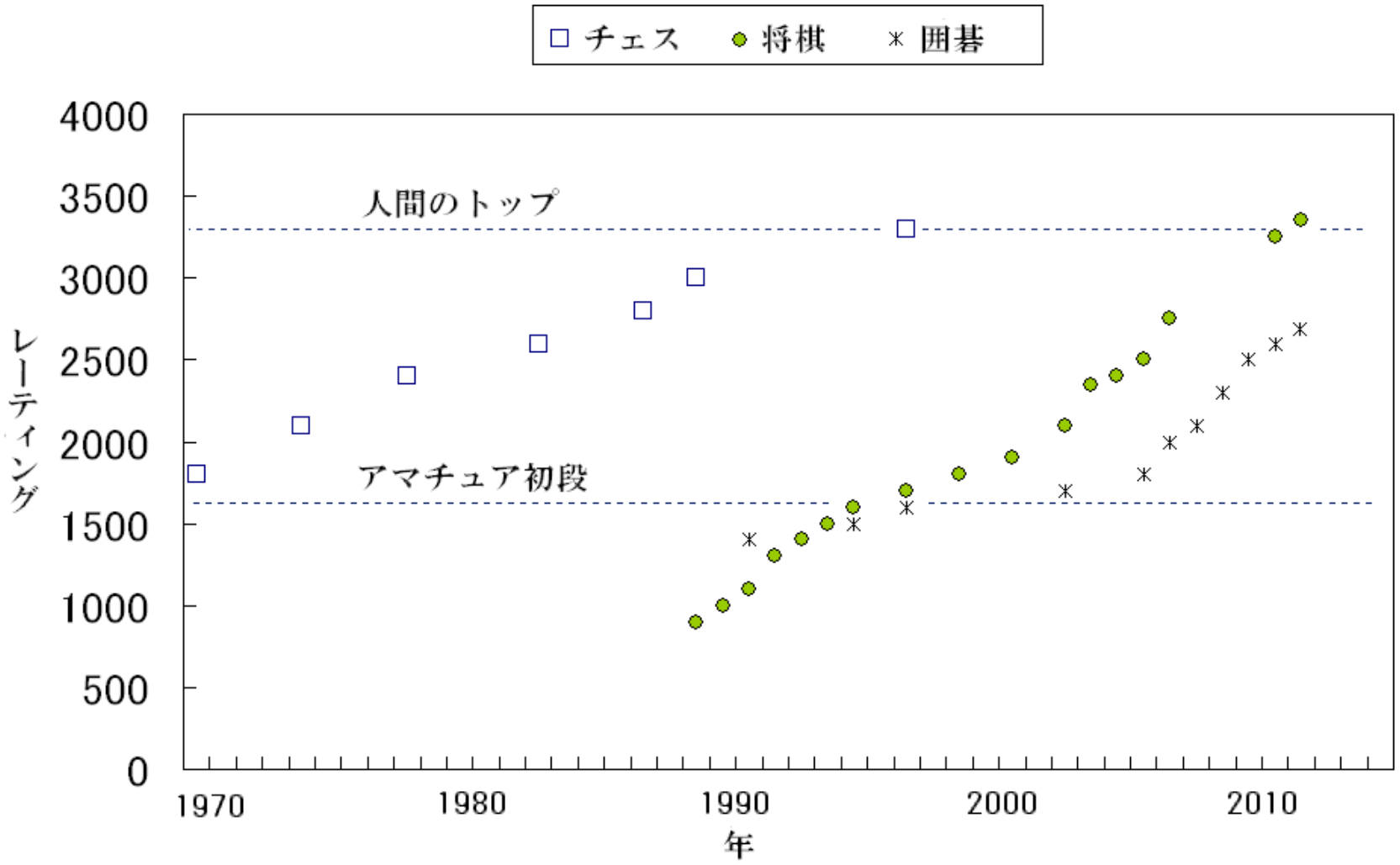
コンピュータ将棋

将棋



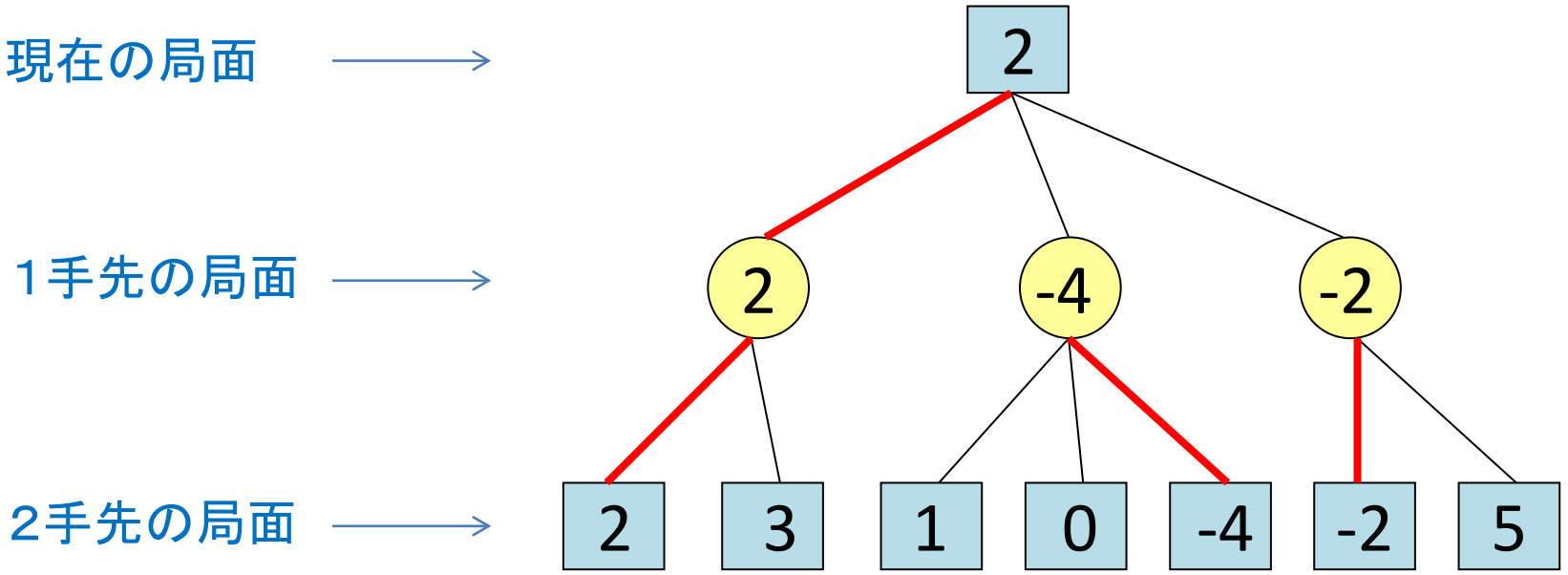
- Japanese chess
- 持ち駒のルール（取った駒が再利用できる）
- 将棋人口（1年に1回以上指した15歳以上の人の数）：700万人

コンピュータチェス・将棋・囲碁



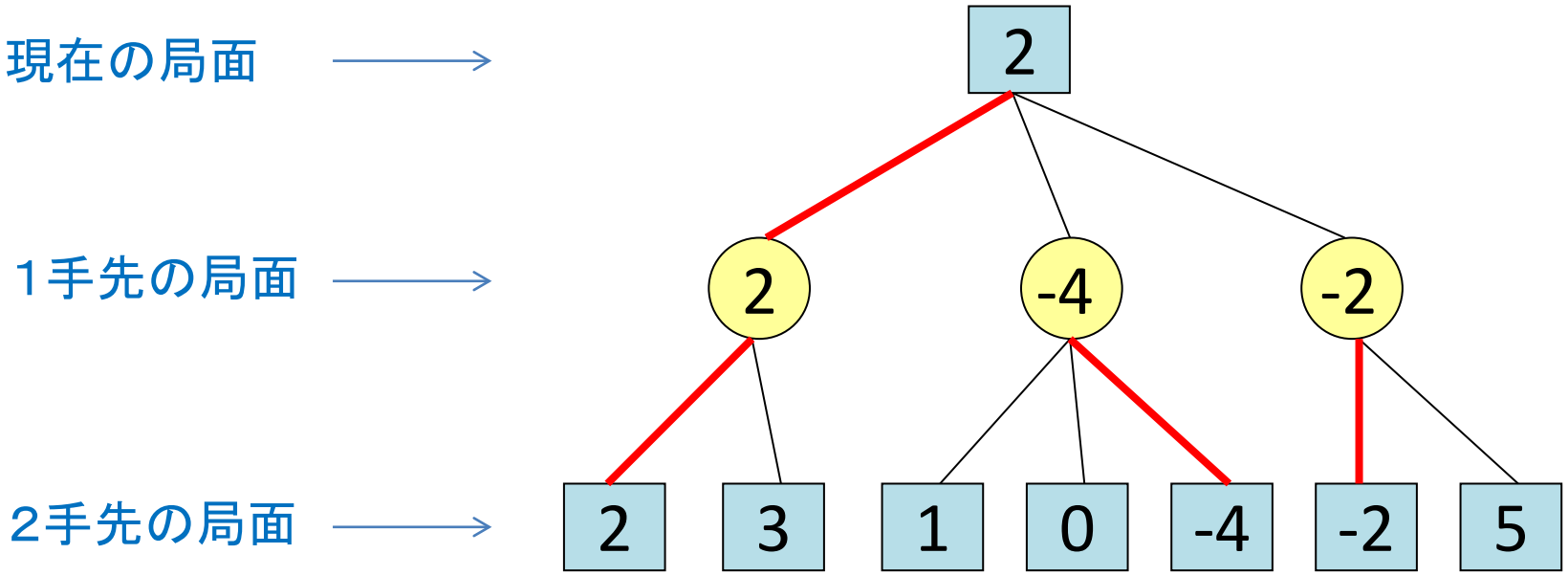
FPGAで将棋プログラムを作ってみるブログ
http://blog.livedoor.jp/yss_fpga/archives/53897129.html

コンピュータの思考法の原理



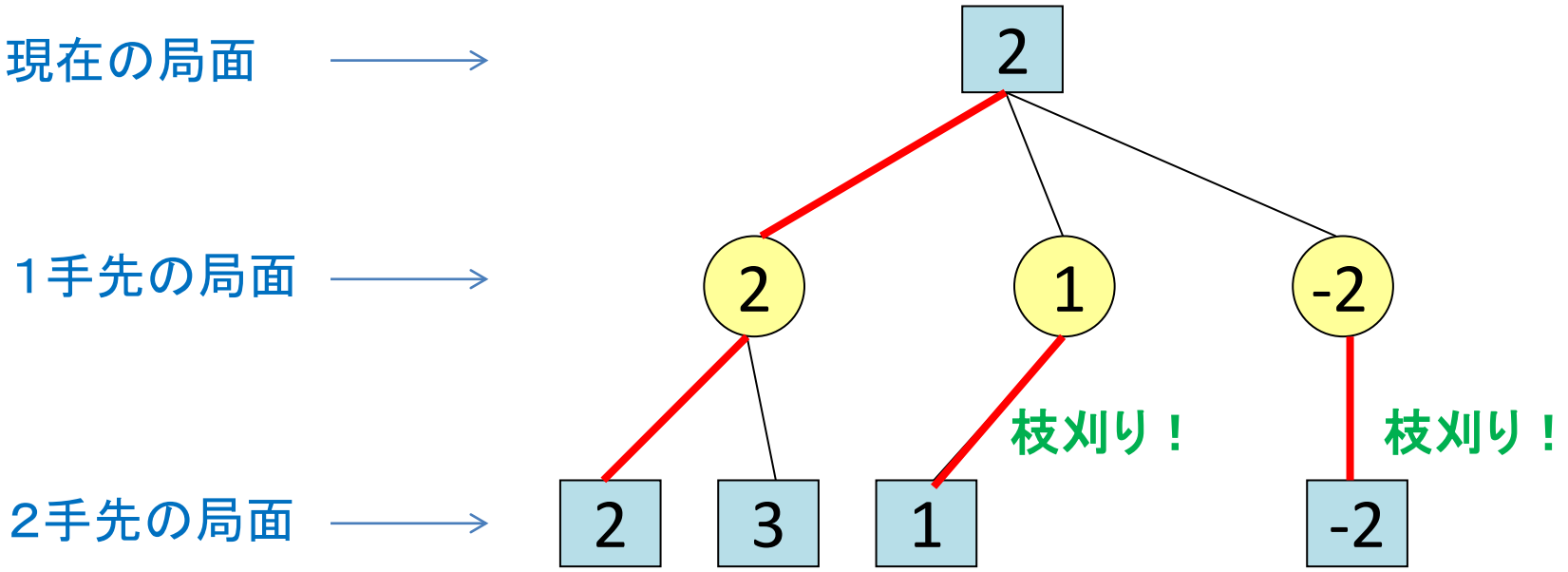
- **評価関数**によって末端局面の有利・不利の度合いを数値化
- お互いが自分にとって最も都合の良い手を選ぶと仮定して逆算（ミニマックス探索）

深さ優先探索



- 関数の再帰呼び出しで簡単に実装できる
- 省メモリ

枝刈り



- 探索ノード数を大幅に減らせる
- 現在局面で選択する手と評価値は変わらない

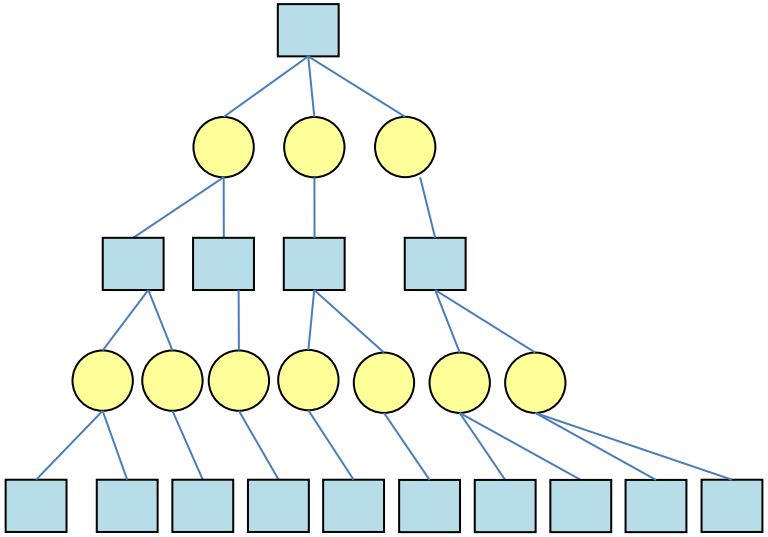
反復深化

最大深さ1で探索

最大深さ2で探索

最大深さ3で探索

最大深さ4で探索

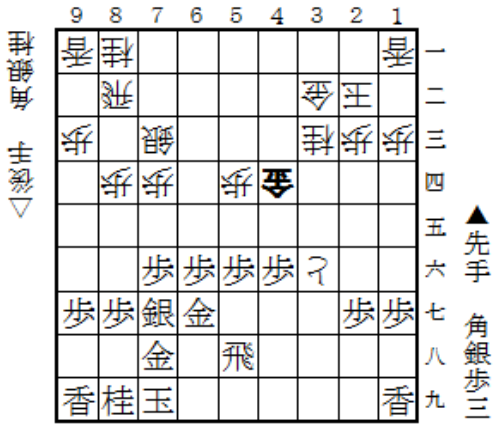


- 探索の最大深さを徐々に深くしていく
- 時間がなるまで繰り返す

評価関数

- 局面の有利／不利を数値化
  - 互角ならゼロ
  - 先手が有利ならプラス
  - 後手が有利ならマイナス

- 重要な要素
  - 駒の損得
  - 駒の働き
  - 玉の危険度
  - 序盤の駒組み



＋320点