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

ゲームと人工知能

- 人工知能研究
  - 計算機で知的な情報処理ができるようにしたい
- ゲーム AI
  - 限定された「世界」
  - ルールが明確に定義
  - 手法の性能評価が比較的容易
  - 強い人間プレイヤの存在

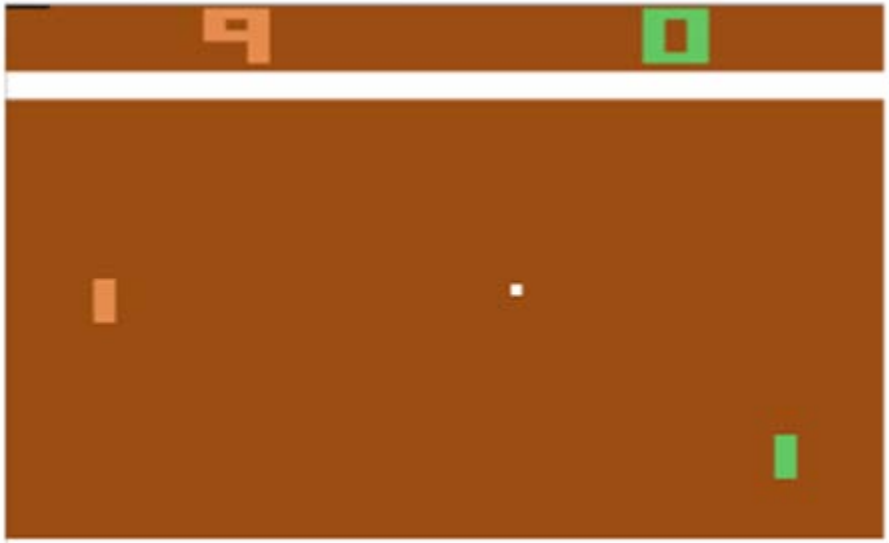
深層強化学習
(DEEP REINFORCEMENT LEARNING)

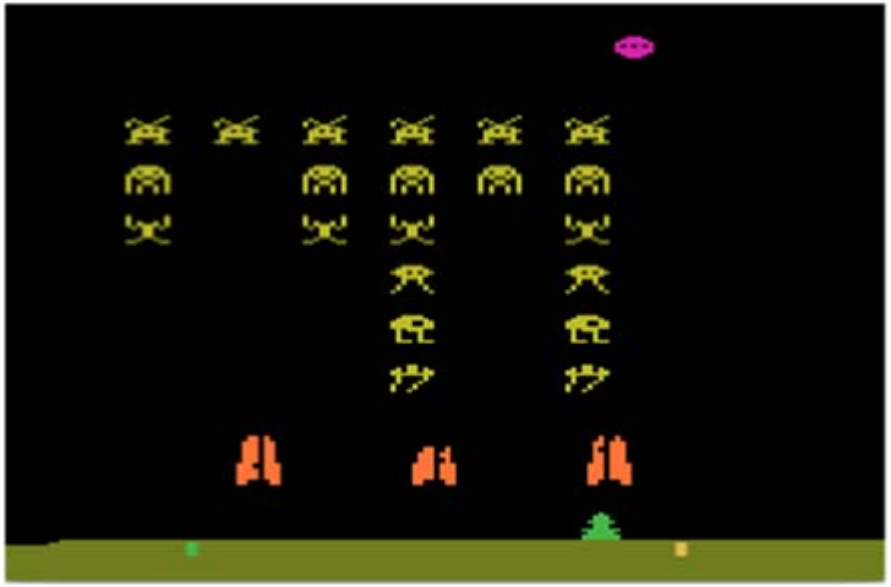
Atari 2600

- 家庭用ビデオゲーム機
  - 1977年発売

https://en.wikipedia.org/wiki/Atari_2600

Atari 2600 Games





強化学習とは

- 強化学習（reinforcement learning）
  - 機械学習の枠組みのひとつ
  - 「環境」と「報酬」を定義

    → 賢く行動する「エージェント」を自動構築

  - エージェント
    - 受け取る「報酬」を（長い目で見て）最大化するような行動戦略を探す

強化学習とは

- 強化学習の特長
  - 教師付き学習と異なり、環境（シミュレータ）と報酬さえ定義すればエージェントが自分で試行錯誤を繰り返してひとりでに賢くなる

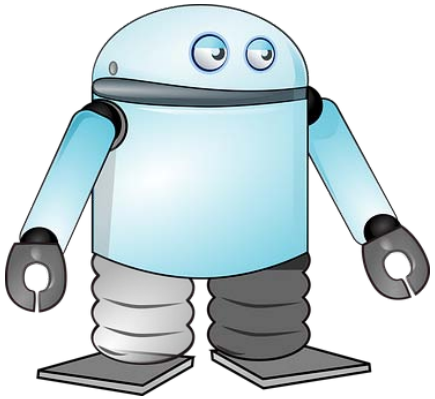
- 応用
  - ロボット、ドローン等の制御
  - ゲームAI
  - 対話・質問応答システム
  - 構造予測問題
  - etc.

(Ng et al., 2004)

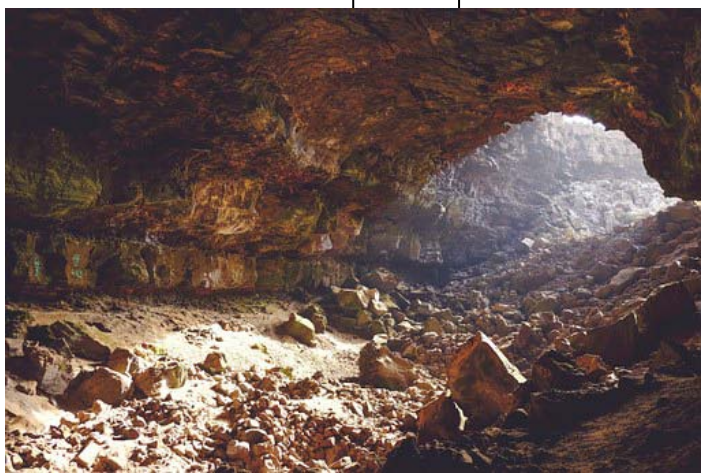
強化学習

状態 s

報酬 r



行動 a



エージェント

環境

マルコフ決定過程
（Markov Decision Process, MDP）

- マルコフ決定過程
  - 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

マルコフ決定過程

- 例

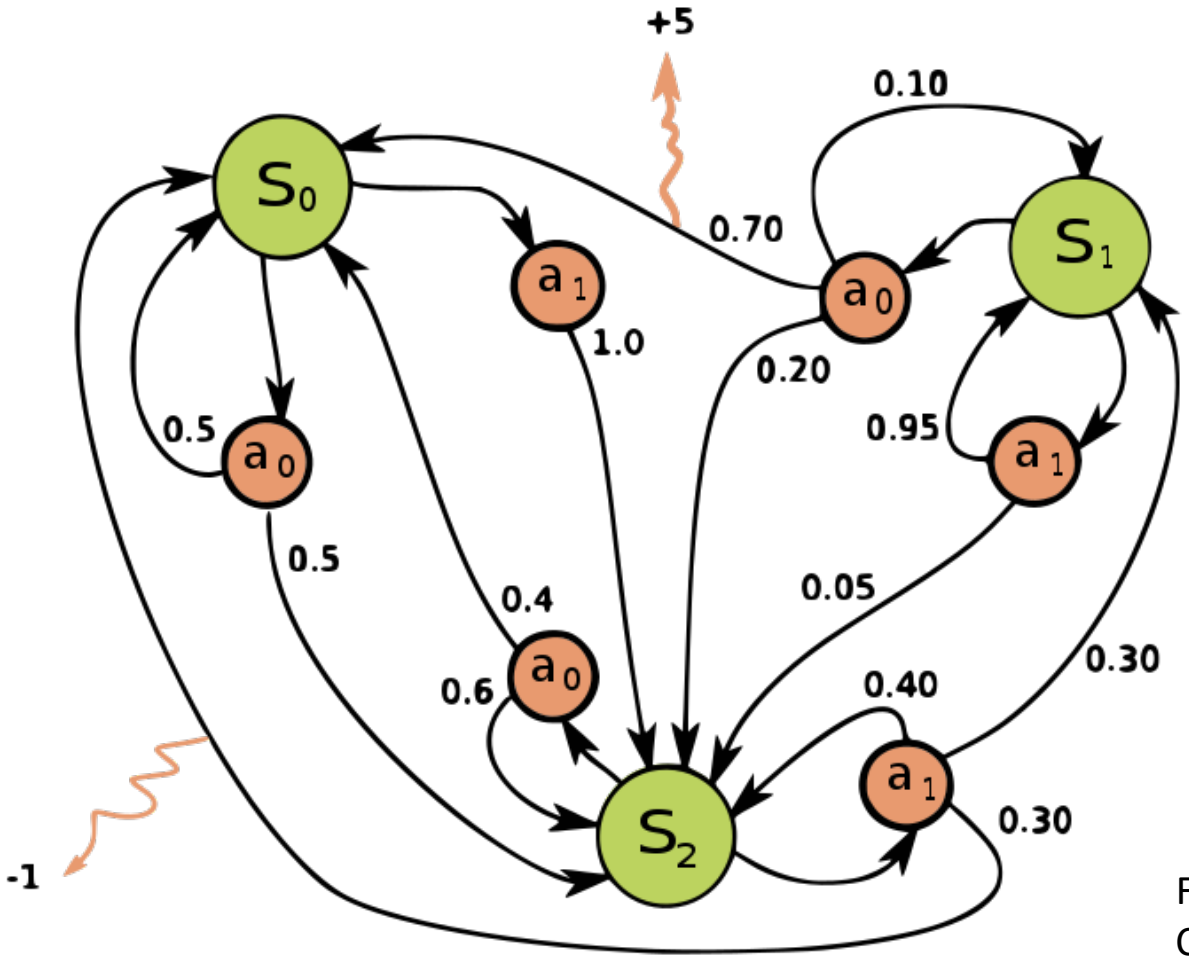


Figure by Waldoalvarez
CC BY-SA 4.0

https://en.wikipedia.org/wiki/Markov_decision_process

リターン

- エージェントの目的
  - 期待リターン（return）を最大化したい
  - リターン：現在から未来にわたる累積報酬

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

  - 割引率 γ $(0 < \gamma < 1)$
    - 未来の報酬をどれだけ重視するかを決めるパラメータ

行動価値関数

- 最適行動価値関数 $Q^*(s, a)$
  - 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リターンの期待値
  - これがわかれば MDP は解決！
    - 各状態で $Q^*(s, a)$ が最大の行動をとればよいので

- $Q^*(s, a)$ が満たす式（Bellman 方程式）

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

Q学習

- Q学習（Q-learning）[Watkins, 1989]
  - オンライン学習によって最適行動価値関数を推定
  - エージェントが行動するたびに $Q^*(s, a)$ の推定値を更新

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

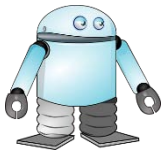
一歩先で得られる
より正確な推定値

現在の
推定値

Q学習（Q-learning）

- アルゴリズム
  - $Q(s,a)$ を適当に初期化
  - 以下のエピソードを繰り返す
    - S を初期状態に設定
    - 以下のステップを終了状態に到達するまで繰り返す
      - 現在の状態 S で選択可能な行動から、適当な方策によって行動 A を選択
      - 行動 A を行い、R と S' を観測
      - $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
      - $S \leftarrow S'$

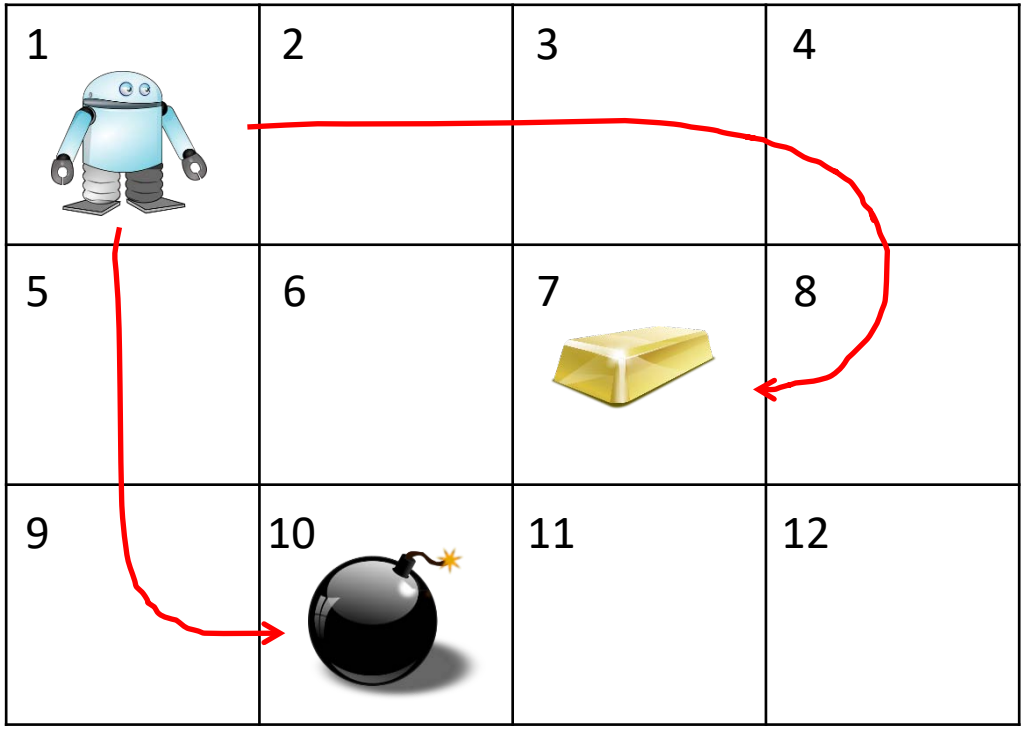
初期状態

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	

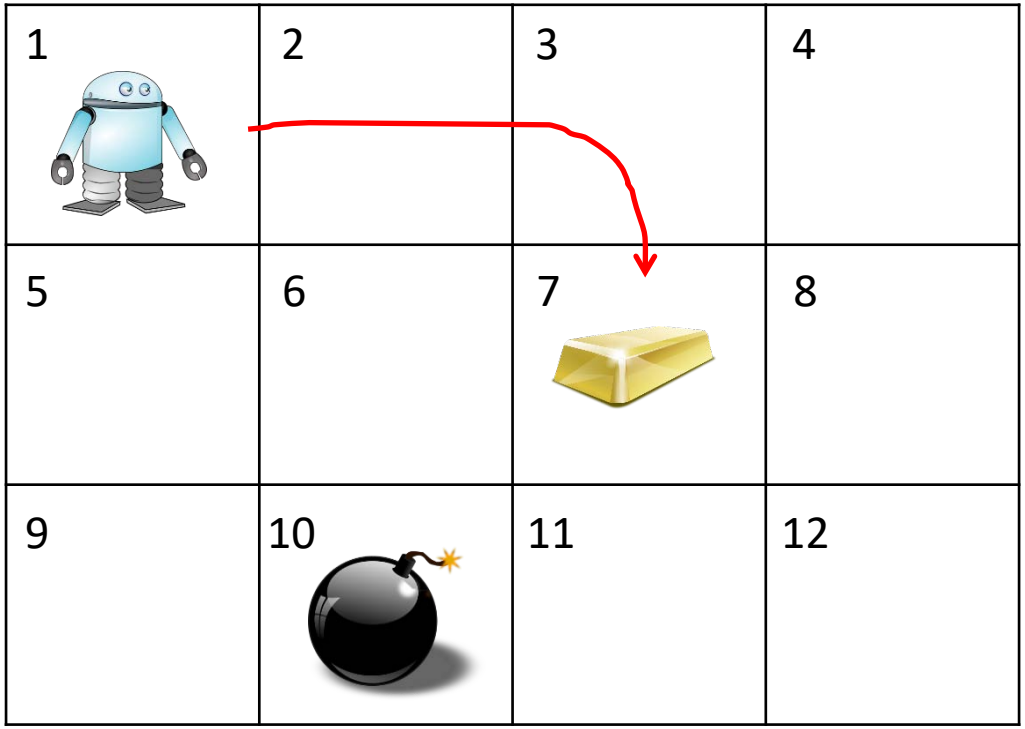
状態7と状態10を経験



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	

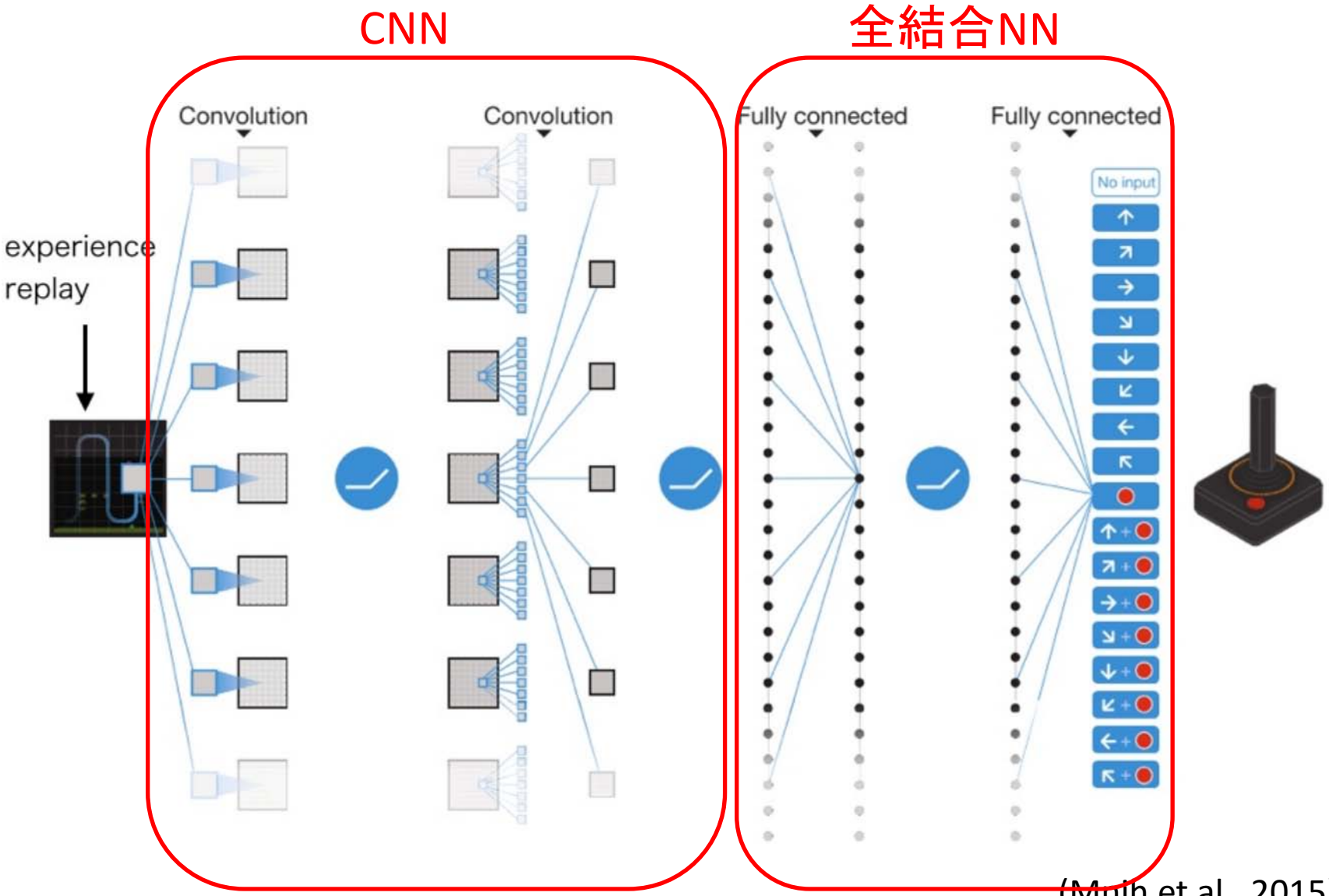
関数近似によるQ学習

- 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
  - $Q(s, a; \theta_i)$が$\gamma + \max_{a'} Q(s', a'; \theta_{i-1})$を予測できるように学習

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

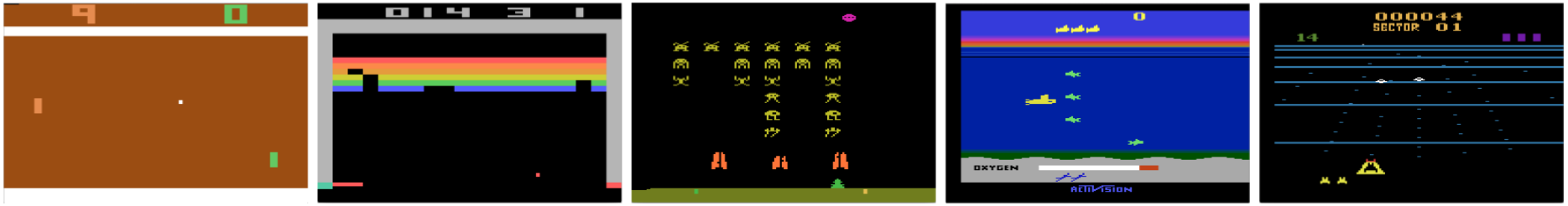
Deep Q-Network



(Mnih et al., 2015)

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

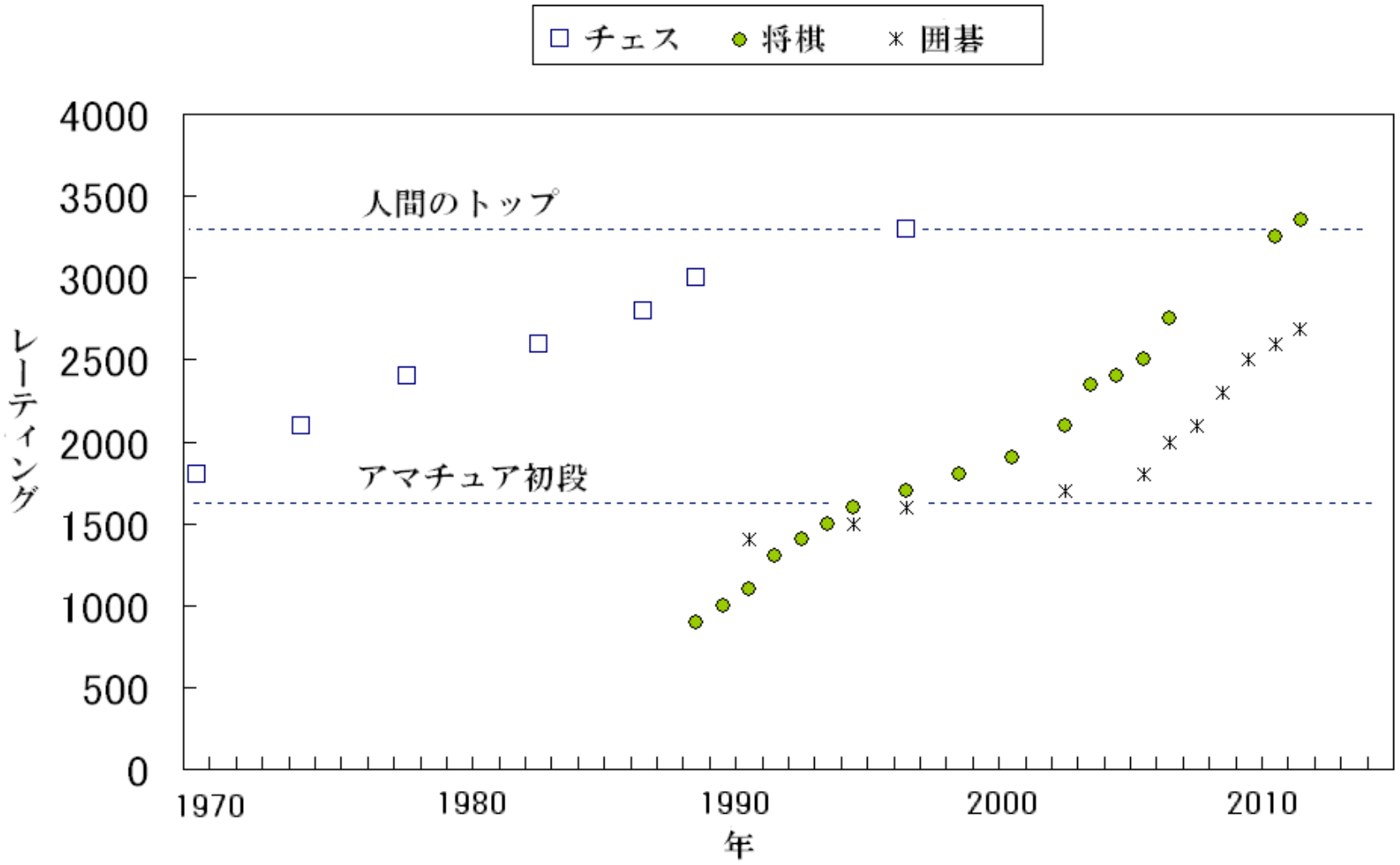
コンピュータ囲碁

AlphaGo vs 李世ドル

コンピュータ囲碁

- コンピュータ囲碁の進歩
  - 初段手前でしばらく停滞
  - モンテカルロ木探索アルゴリズムの登場（2006年ごろ）
  - アマチュアトップレベルに
  - 再び停滞（～2015年）
  - AlphaGo 登場（2016）
- 難しさ
  - 合法手が多い
  - 評価関数の設計が難しい
    - 地が確定するのは最後
    - 石の生死の判定
    - 離れた場所にある石の影響
    - etc

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

レーティング

- 棋力を数値化する手法
  - Elo rating system

$$E_A = \frac{1}{1+10^{(R_B - R_A)/400}}$$

E_A ： プレイヤAが勝つ確率

R_A ： プレイヤAのレーティング

R_B ： プレイヤBのレーティング

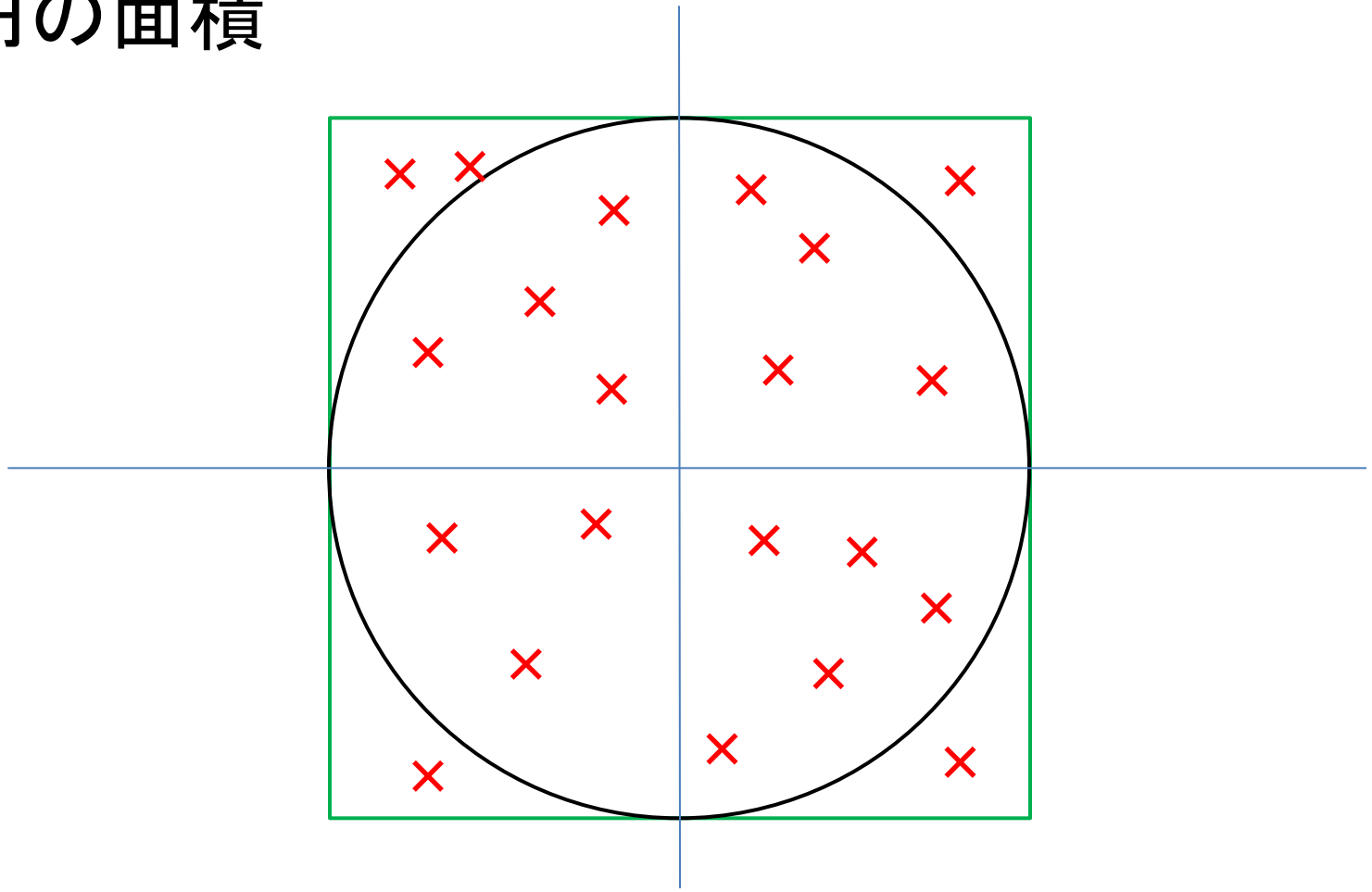
  - 対局相手と勝ち負けの履歴から最尤推定

モンテカルロ木探索（MCTS）

- モンテカルロ木探索（Monte Carlo Tree Search, MCTS）
  - AI 研究に大きな影響
    - 囲碁で大成功
    - 他のゲーム、プランニング、制御、最適化問題などへの応用が進む
  - 特長
    - ドメイン知識が不要
    - 他の手法がうまくいかない難しい問題で成功
    - 計算パワーの向上がそのまま性能の向上につながる

モンテカルロ法

- 円の面積

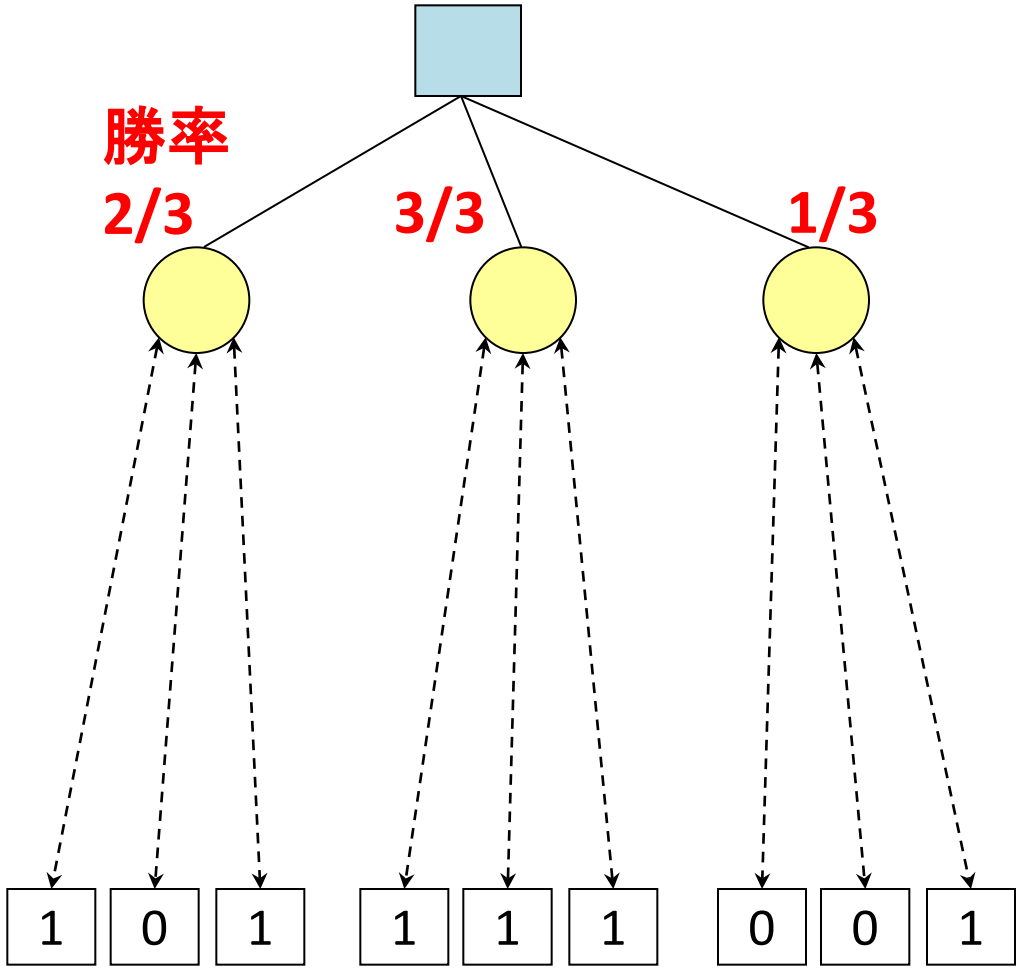


原始モンテカルロ法

- 各合法手からランダムプレイ
  - 評価関数不要

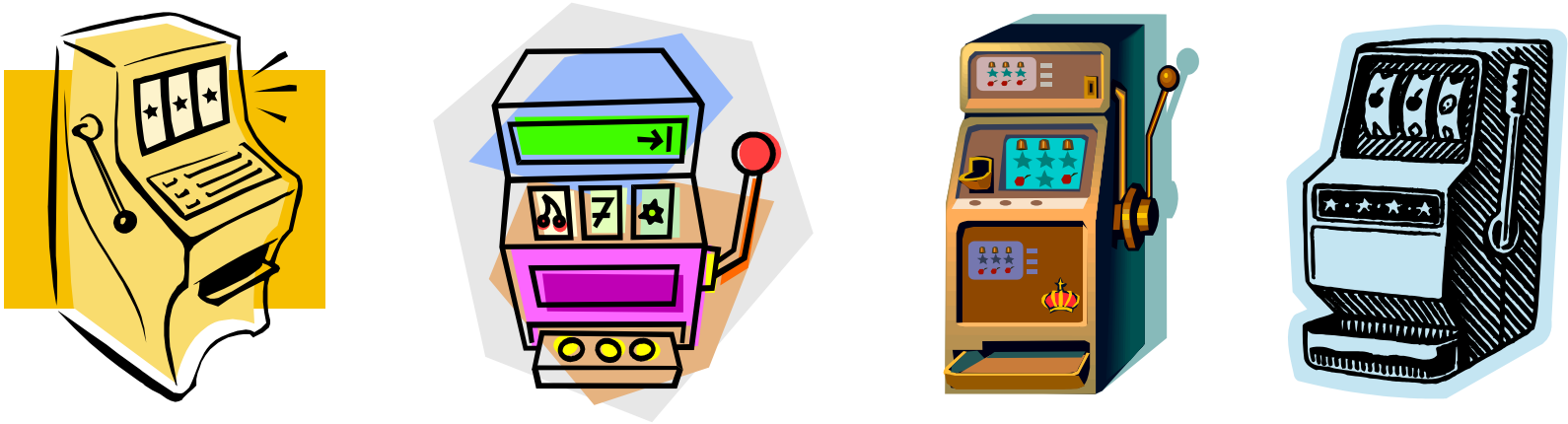
- 勝率の一番高い手を選ぶ

- ダメな手に対しも多くの試行を行うので効率が悪い



多腕バンディット問題

- 多腕バンディット問題（multi-armed bandits）



- どのスロットマシンにお金をつぎこむべきか？
  - 儲かるマシンに集中したい（exploitation）
  - 儲かるマシンを見つけたい（exploration）

UCB

- Upper Confidence Bounds (UCBs)
  - 多腕バンディット問題の近似解法
  - Regret が O(ln *n*)

$$\mathrm{UCB1} = \overline{X}_j + \sqrt{\frac{2\ln n}{n_j}}$$

総試行回数

マシン *j* のこれまでの報酬の平均

マシン *j* の試行回数

利用（exploitation）

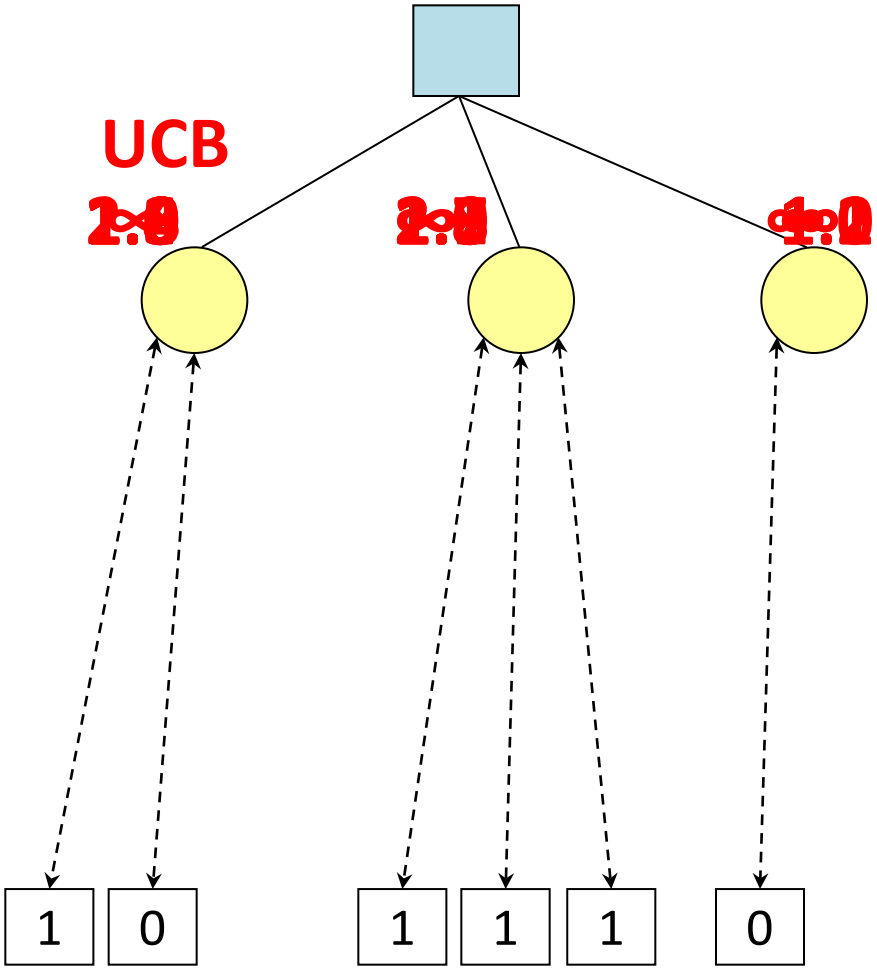
探索（exploration）

UCB 例

- 各イテレーションで UCB 値が最も高い手を選ぶ

$$\mathrm{UCB1} = \overline{X}_j + \sqrt{\frac{2\ln n}{n_j}}$$

- 有望な手に関して多くの試行



UCT

- UCB の問題
  - ２手目以降のプレイアウトに無駄が多い
  - 相手の悪手に期待するような手を選ぶ

- UCT (UCB applied to Trees)
  - Kocsis & Szepesvari (ECML 2006)
  - UCB を各ノードで適用
  - 勝率等を各ノードに保存した木を成長させる
  - MINMAX値に収束

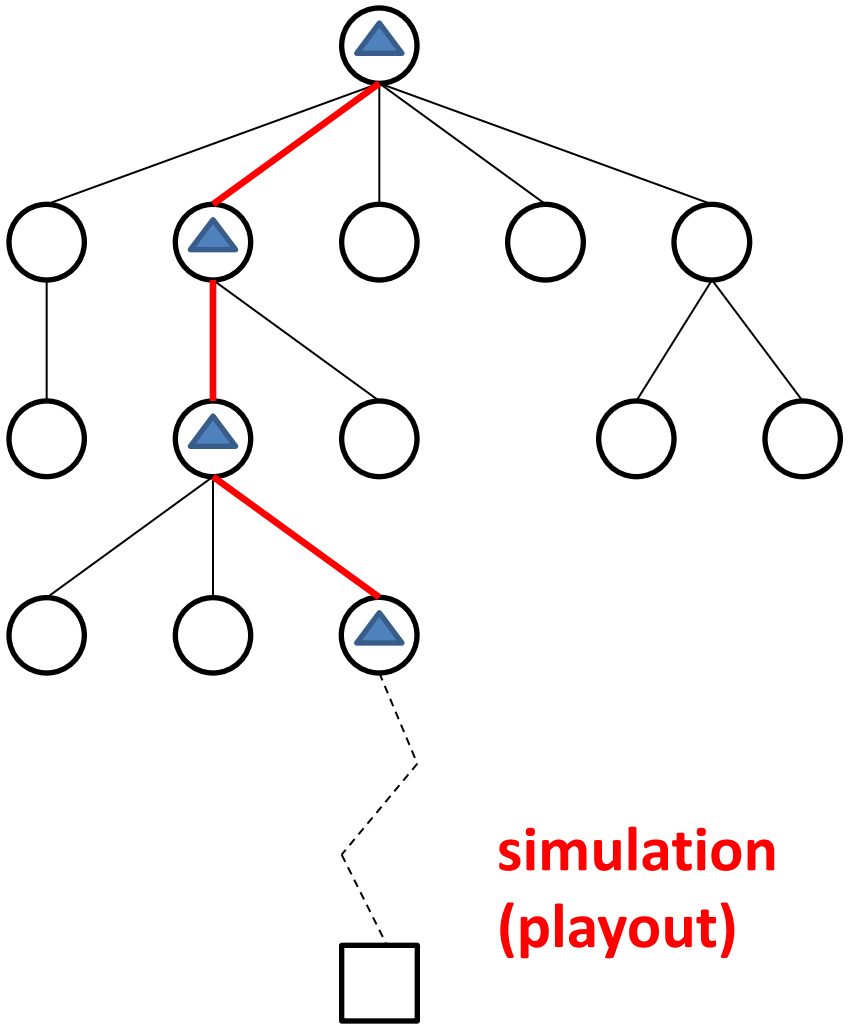
MCTSの基本動作

- 各イテレーション

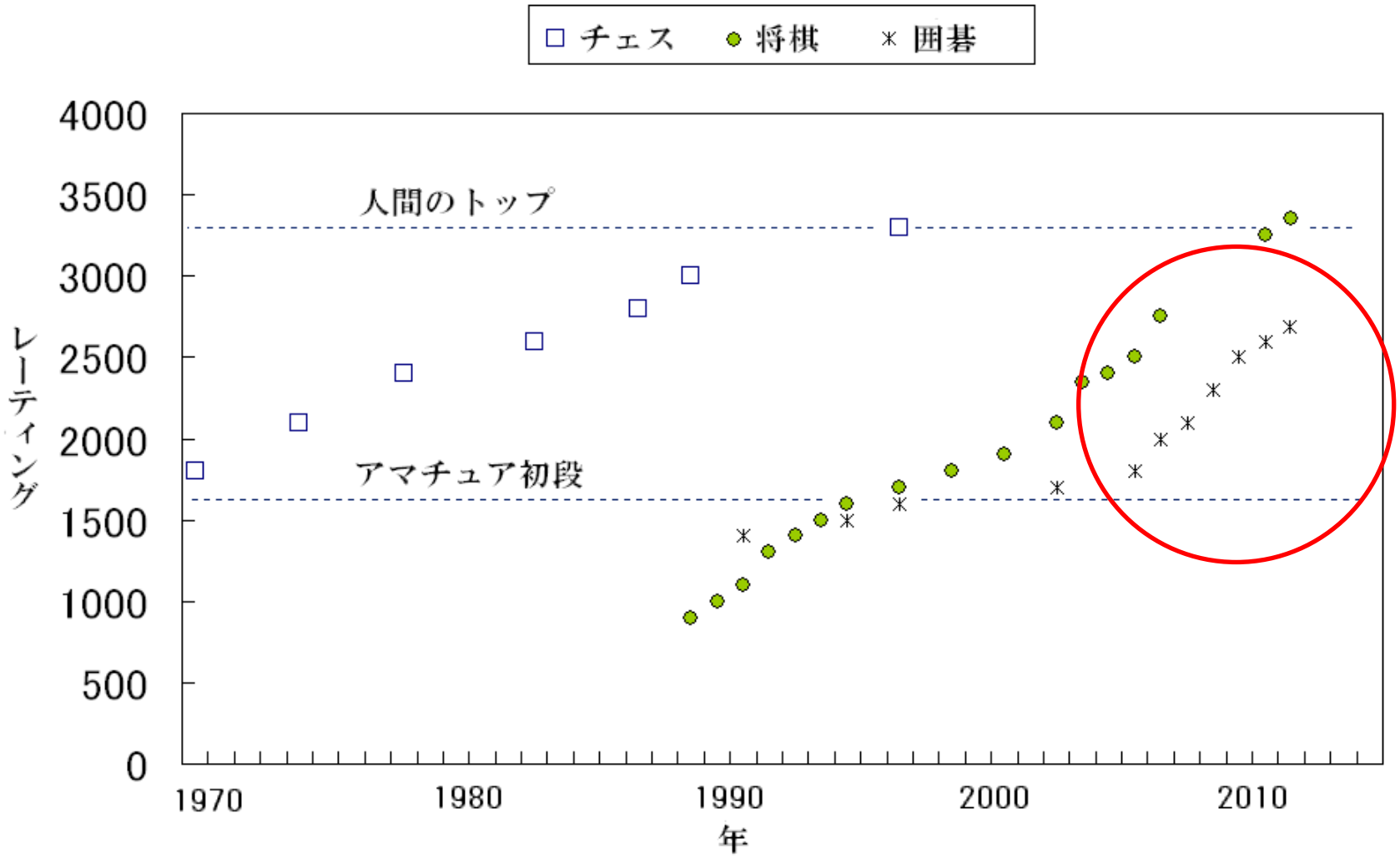
1. Selection
2. Expansion
3. Simulation
4. Backpropagation

- UCT 値が最大の子ノードを再帰的に選択

$$\mathrm{UCT} = \overline{X}_j + 2C_p\sqrt{\frac{2\ln n}{n_j}}$$



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

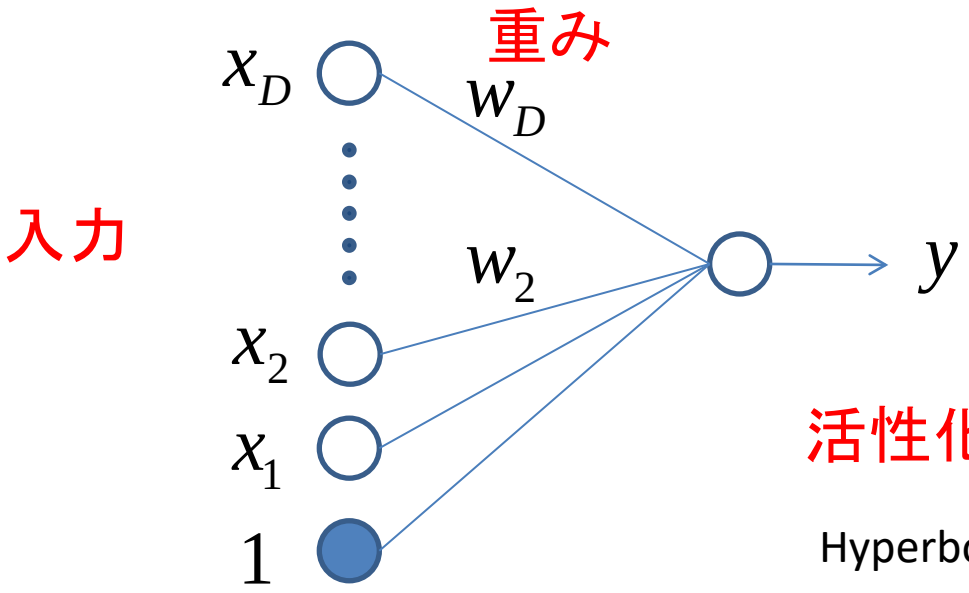
AlphaGo

- Google DeepMind が開発
- 従来の囲碁プログラムを圧倒的に上回る棋力
  - 500勝1敗（？）

- 深層学習（deep learning）による強化学習
  - ポリシーネットワーク
    - 個々の指し手が選択される確率
  - バリューネットワーク
    - 局面の評価（勝敗の予測）

ニューラルネットワーク

- ニューロン

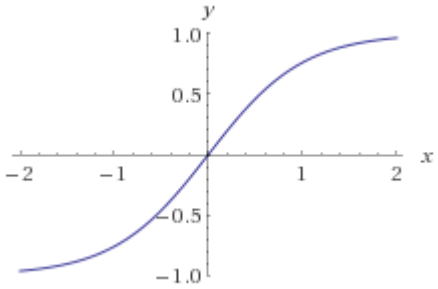


入力の線形和に非線形な
活性化関数を適用

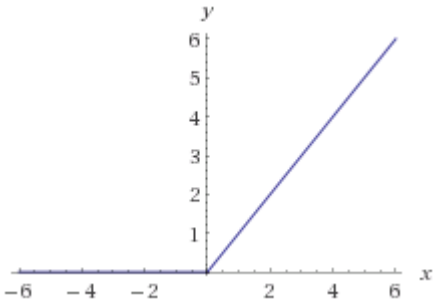
$$y = f\left(\sum_{i=0}^{D} w_i\, x_i\right)$$

活性化関数

Hyperbolic tangent

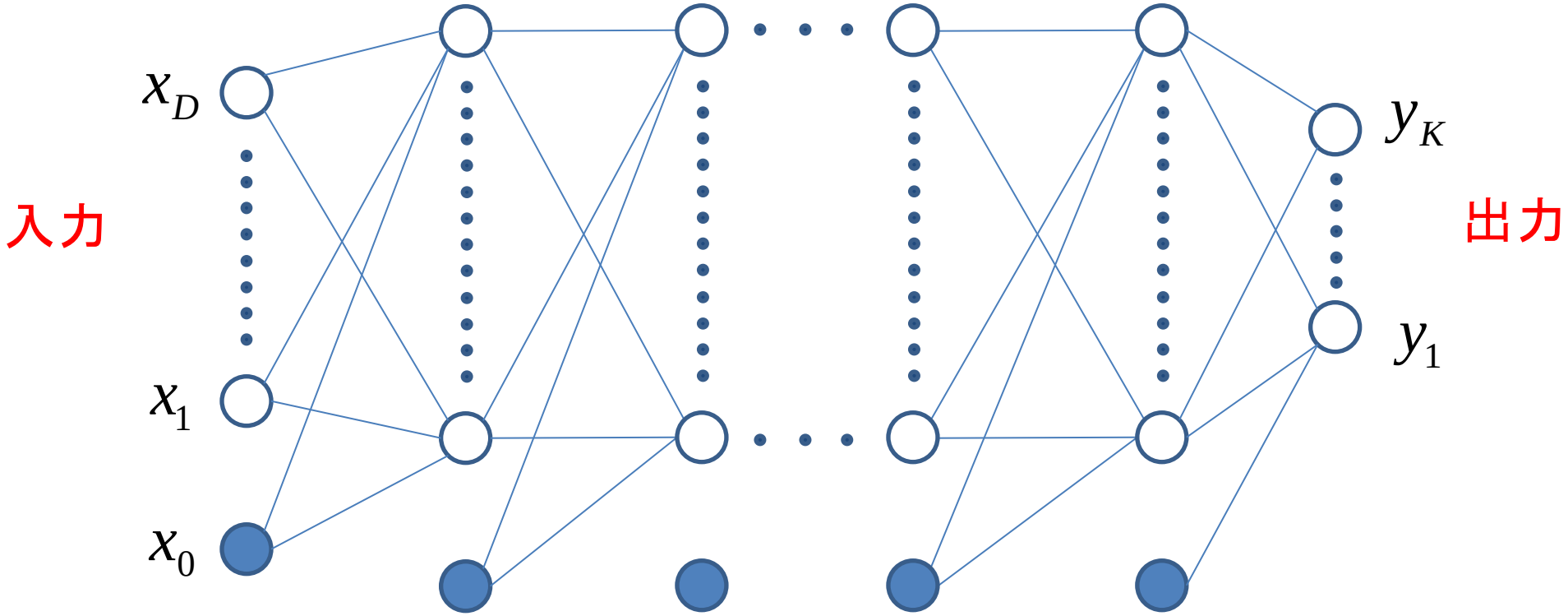


ReLU (Rectified Linear Unit)



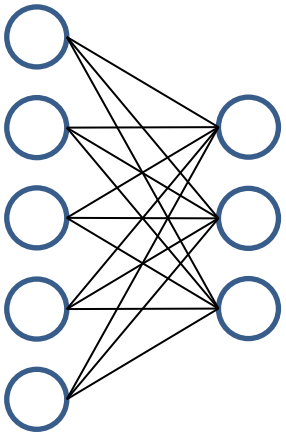
多層ニューラルネットワーク

- 多数の入出力のペアから入出力関係を学習



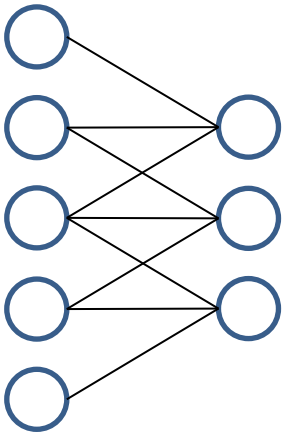
畳み込みニューラルネットワーク（Convolutional Neural Network, CNN）

- 全結合



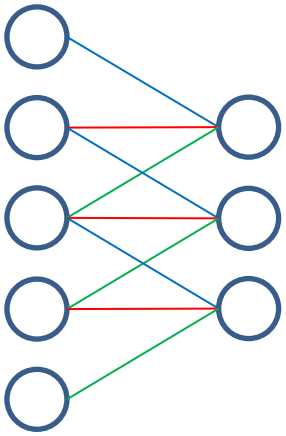
パラメータ数
5 x 3 = 15

- 局所的結合



パラメータ数
3 x 3 = 9

- パラメータ共有

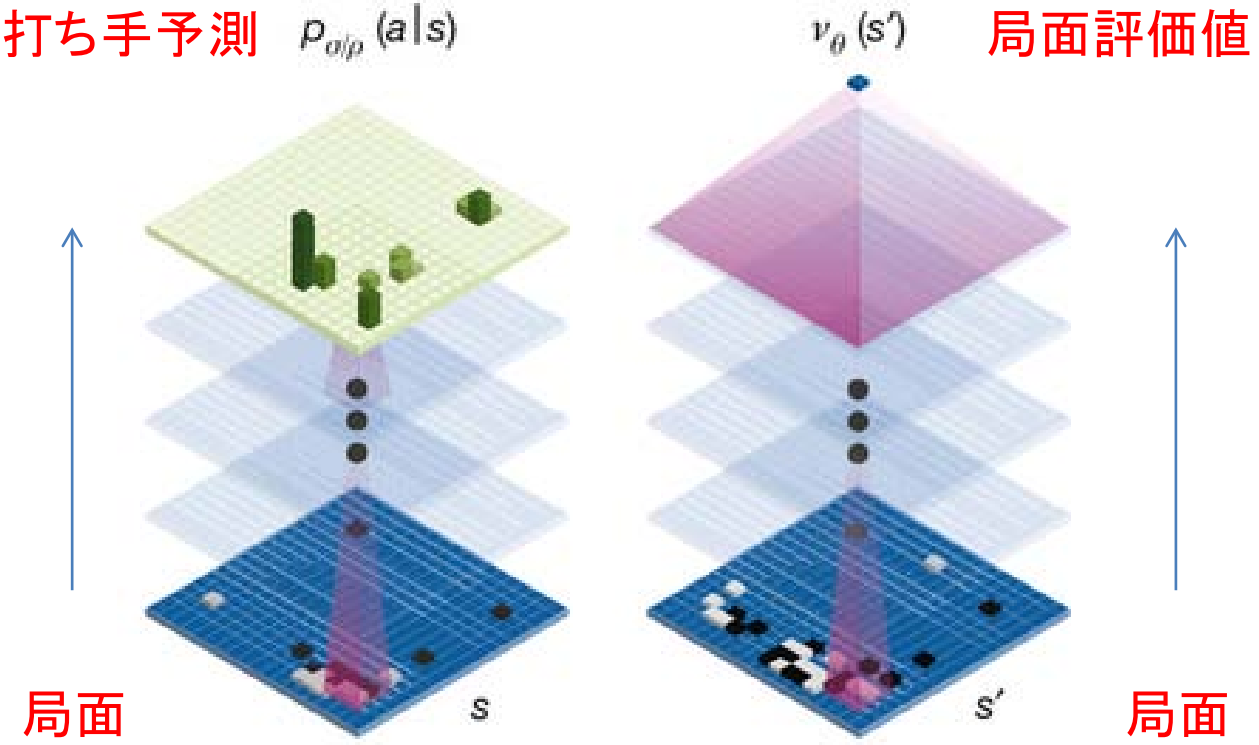


パラメータ数
3

パラメータ数を減らすことにより過学習を回避
画像認識、テキスト分類などに有効

AlphaGo

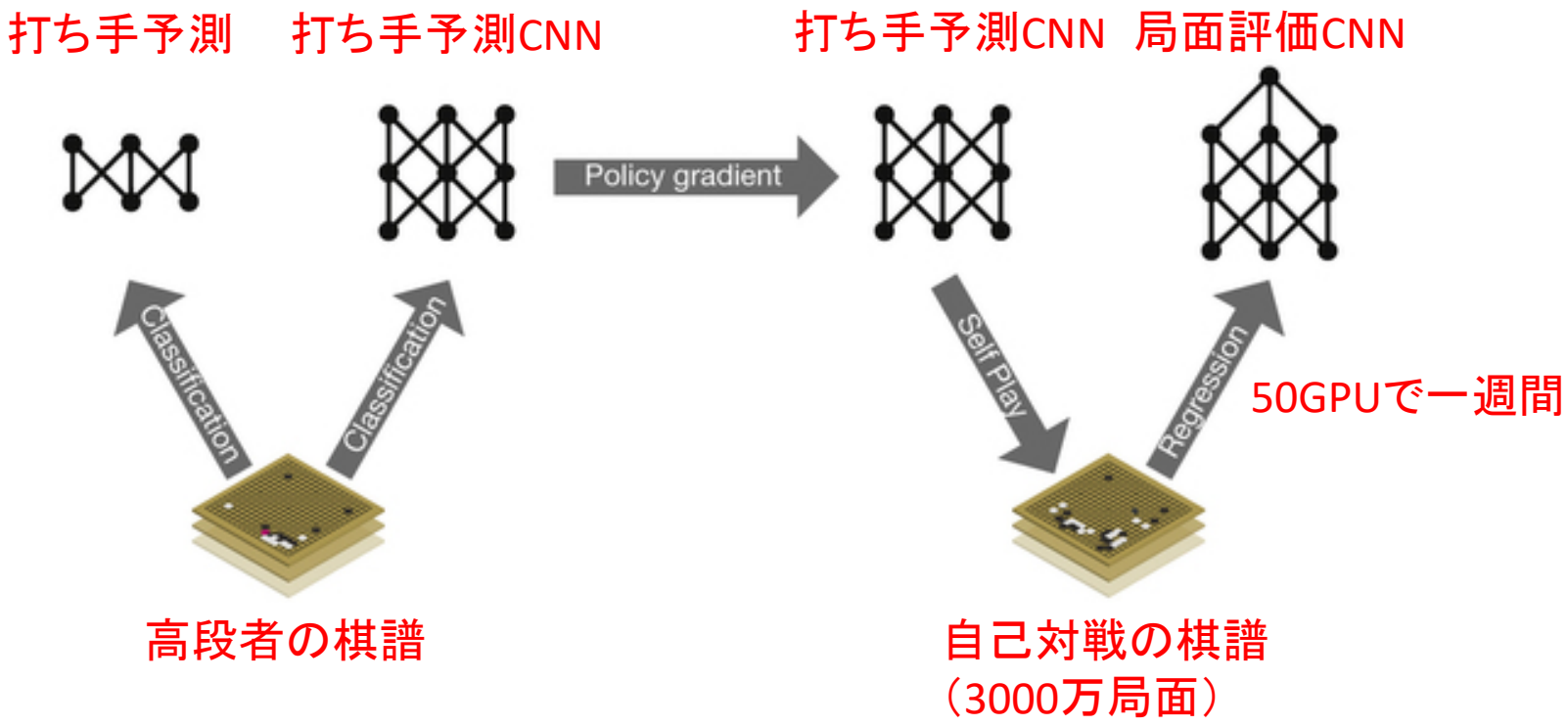
- CNN による打ち手予測、局面評価
  - 入力 19x19x48　深さ12層



D Silver *et al. Nature* **529,** 484–489 (2016) doi:10.1038/nature16961

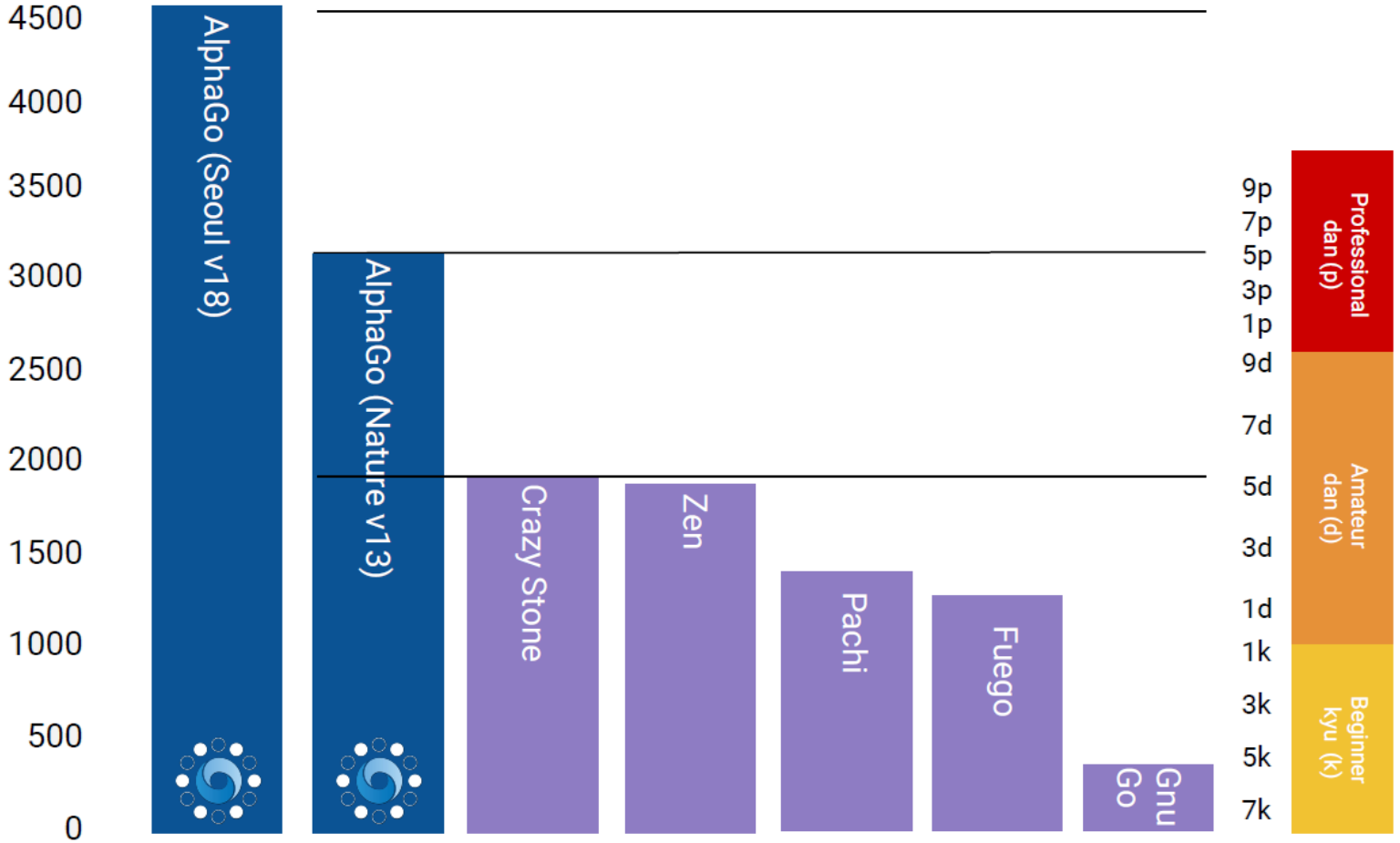
AlphaGo

- ニューラルネットワークの学習
  - 高段者の棋譜による教師付き学習＋強化学習



D Silver *et al. Nature* **529,** 484–489 (2016) doi:10.1038/nature16961

AlphaGo の棋力



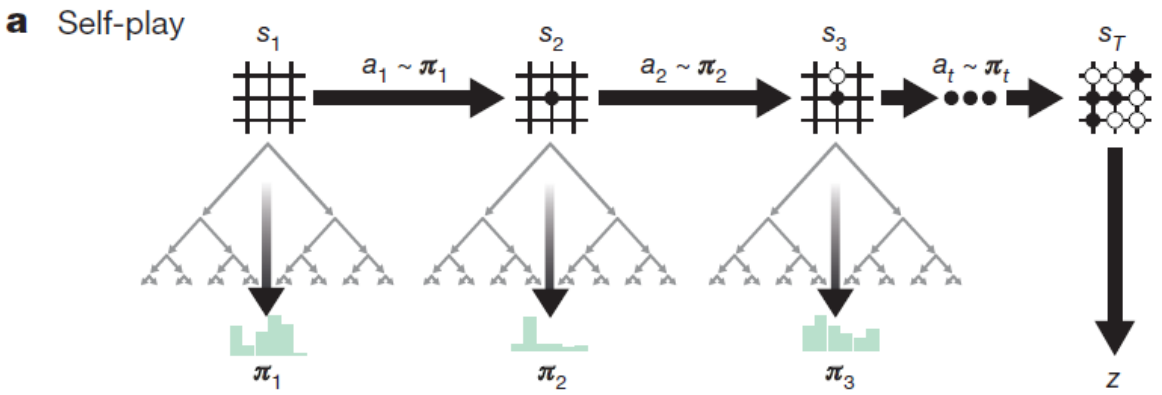
(D Silver, ICML Tutorial 2016)

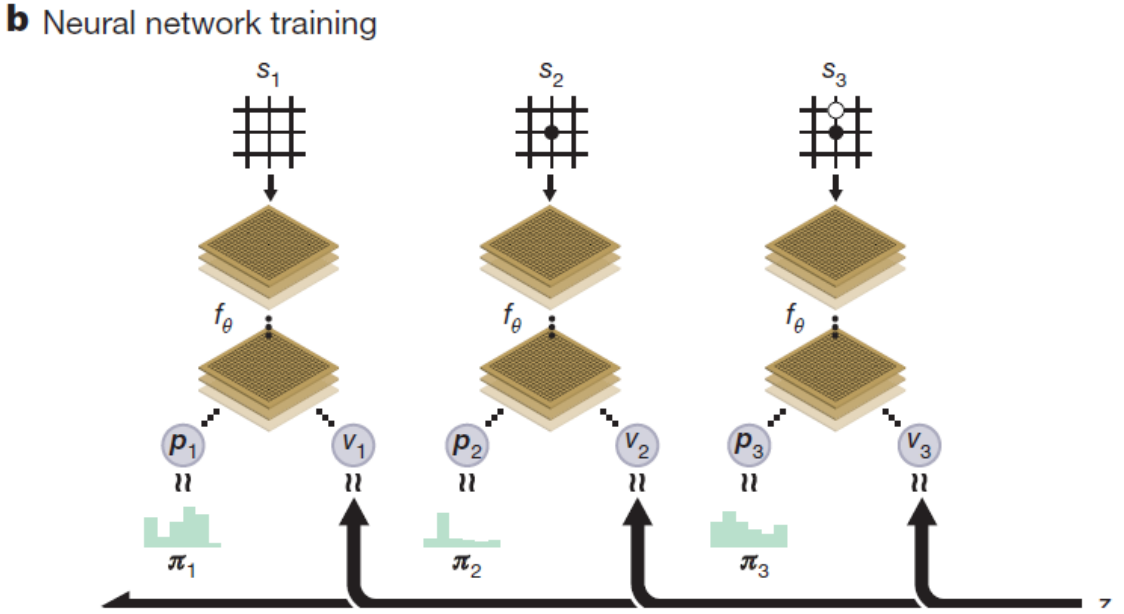
AlphaGo Zero [Silver et al., 2017]

- 2017年10月に発表された AlphaGo の後継

- 特徴
  - 囲碁の知識を（ルール以外は）ほとんど利用しない
    - プロ棋士の棋譜は使わない
    - ニューラルネットワークへの入力は黒白の石の配置だけ
  - ランダムなパラメータから強化学習
    - AlphaGo の棋力を36時間で上回る
  - ニューラルネットワーク専用ハードウェアを利用
    - Tensor Processing Unit (TPU)

自己対戦による学習





・ニューラルネットワーク

$$(\boldsymbol{p}, v) = f_\theta(s)$$

局面 s から合法手の確率分布 $\boldsymbol{p}$ と局面評価 v を計算

・学習の目的関数

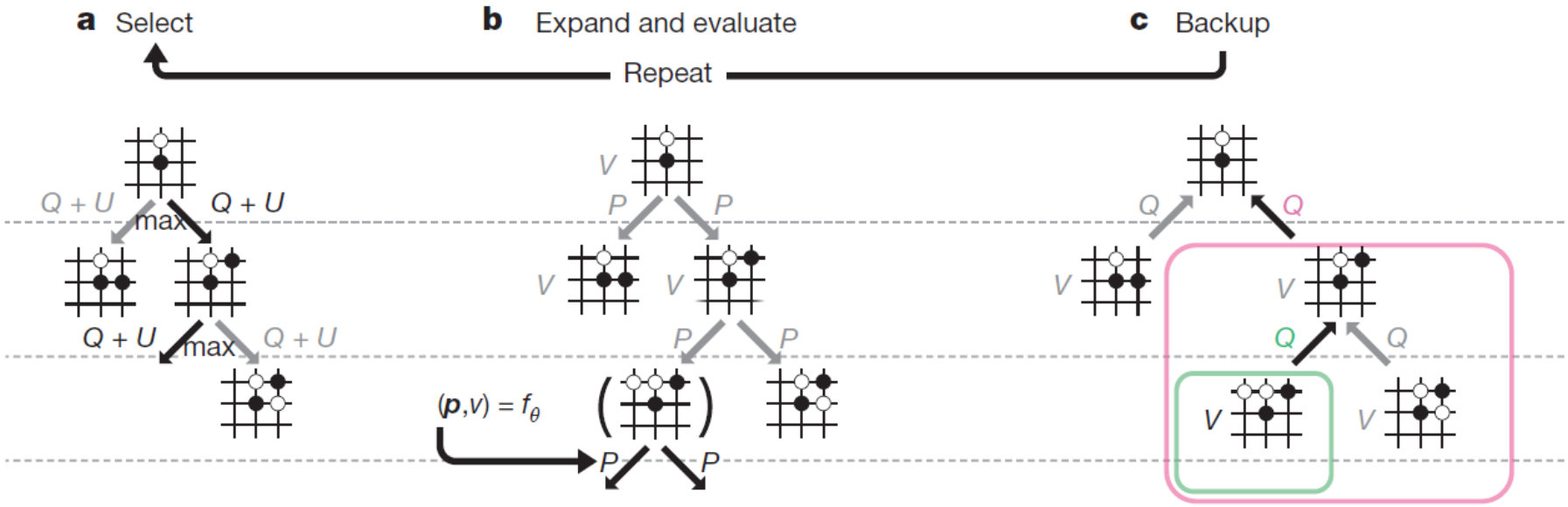
$$l = (z - v)^2 - \boldsymbol{\pi}^{\mathrm{T}} \log \boldsymbol{p} + c\|\theta\|^2$$

局面評価 v と実際の勝ち負けのずれが少ないほど良い

確率分布 $\boldsymbol{p}$ とモンテカルロ木探索によって得られた分布 π とのずれが少ないほど良い

(Silver et al., 2017)

モンテカルロ木探索



手の選択の基準

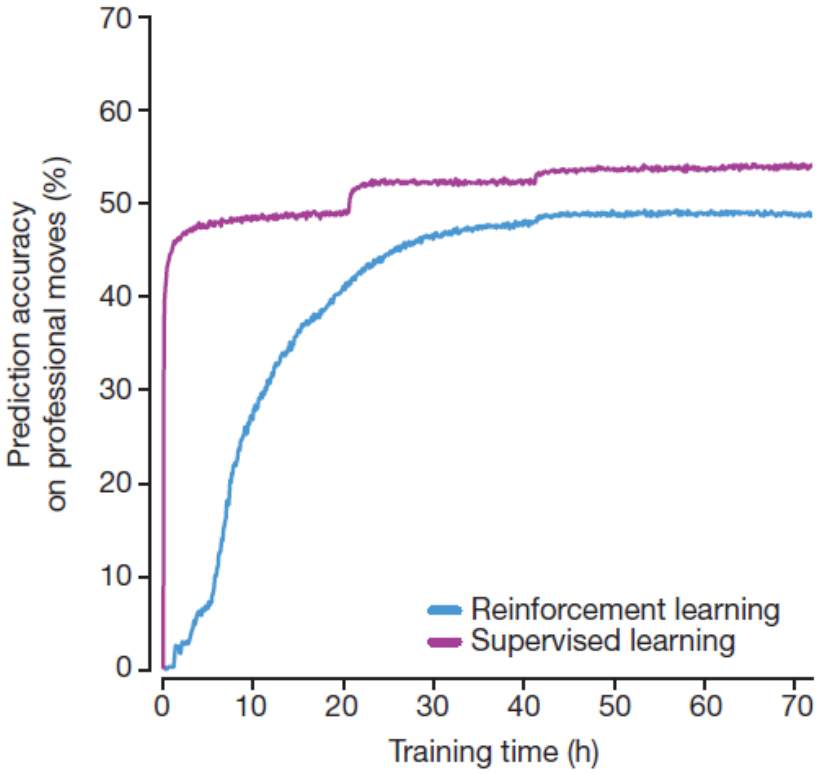
$$Q(s,a)+cP(s,a)\frac{\sqrt{\sum_b N(s,b)}}{1+N(s,a)}$$

$$Q(s,a)=\frac{1}{N(s,a)}\sum_{s'|s,a\to s'}V(s')$$

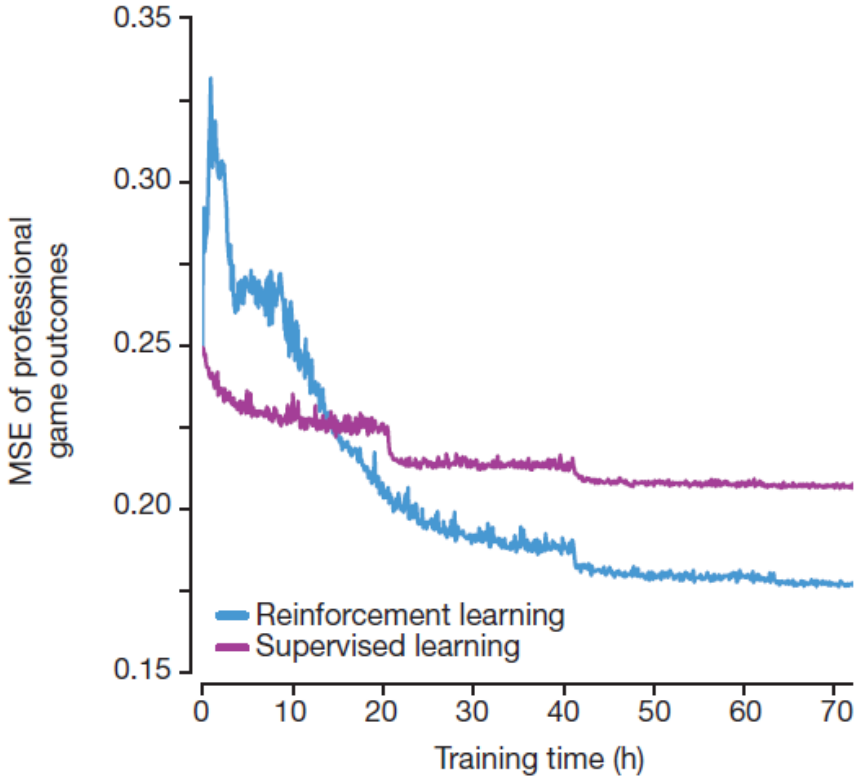
(Silver et al., 2017)

ニューラルネットワークの予測性能

・プロ棋士の手を予測

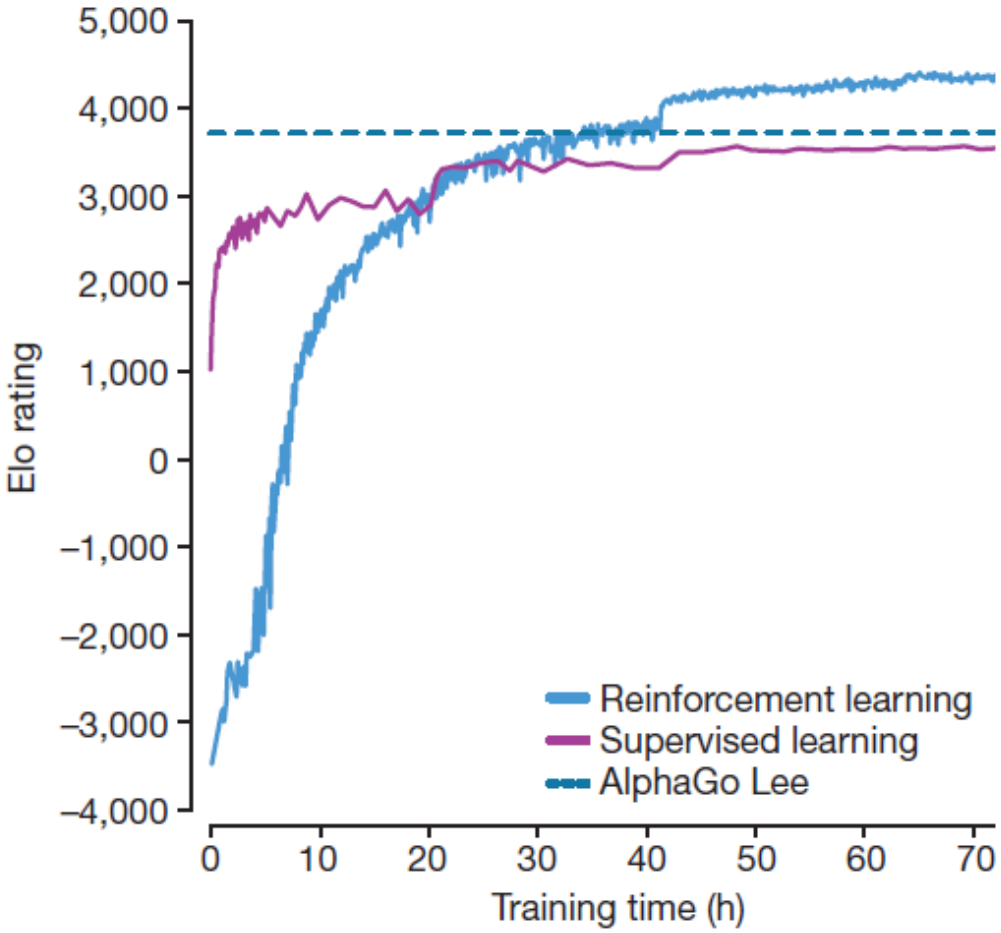


・プロ棋士の棋譜の勝敗を予測



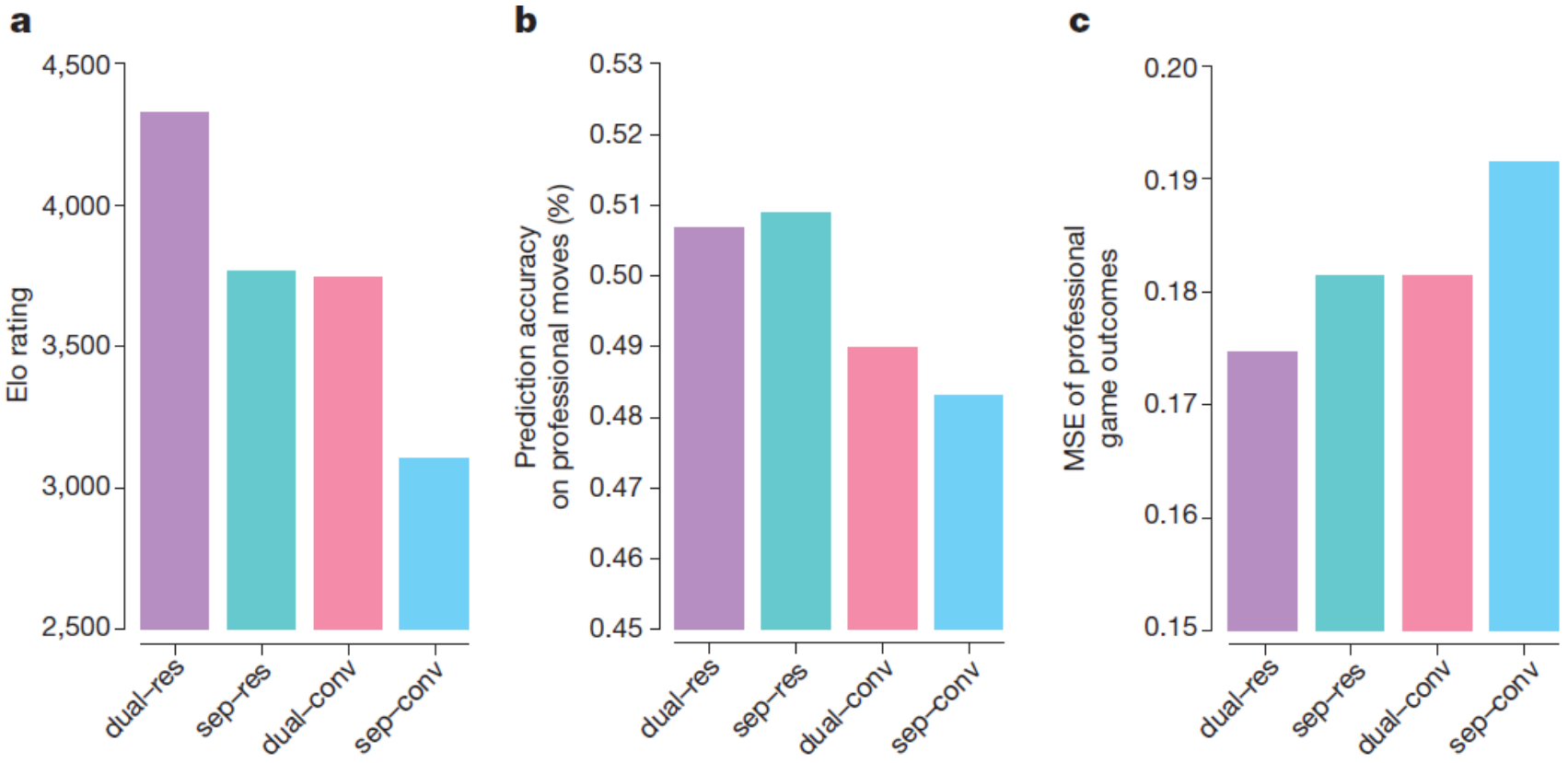
(Silver et al., 2017)

AlphaGo Zero の強化学習



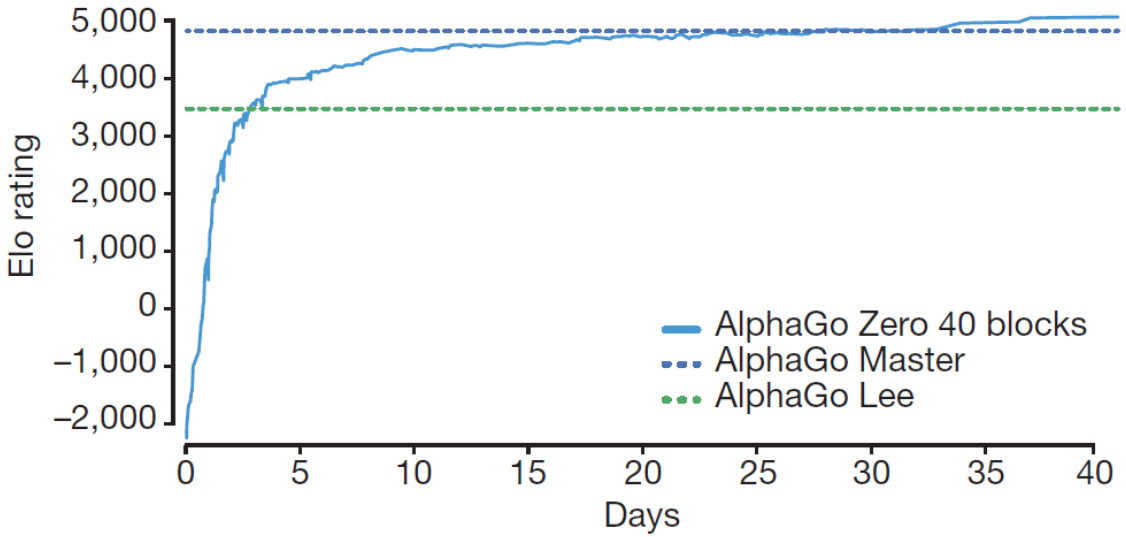
(Silver et al., 2017)

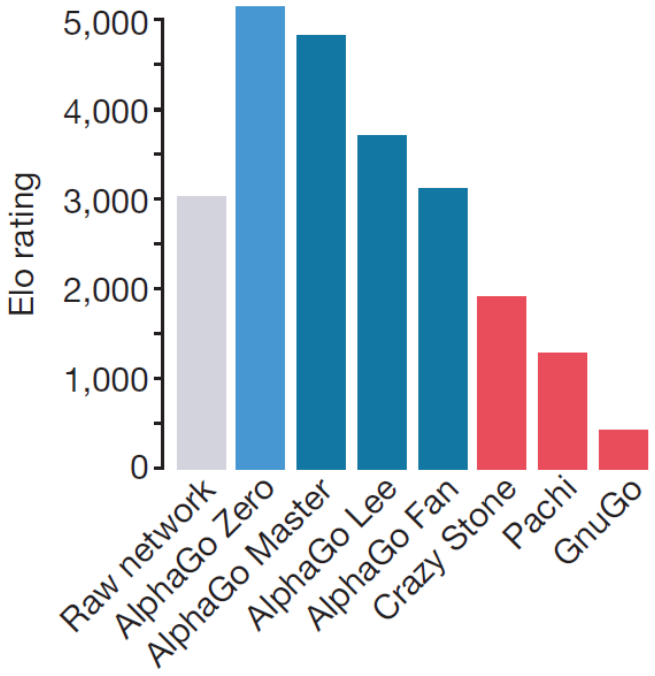
マルチタスク学習及びResNetの効果



(Silver et al., 2017)

AlphaGo Zero の強さ

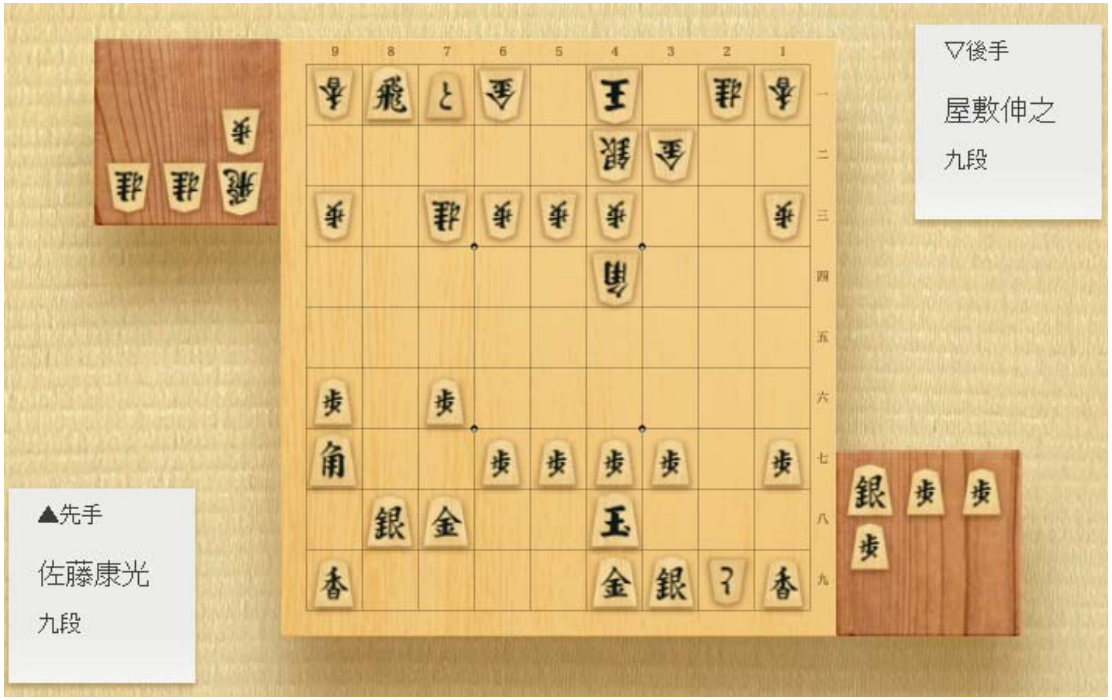




(Silver et al., 2017)

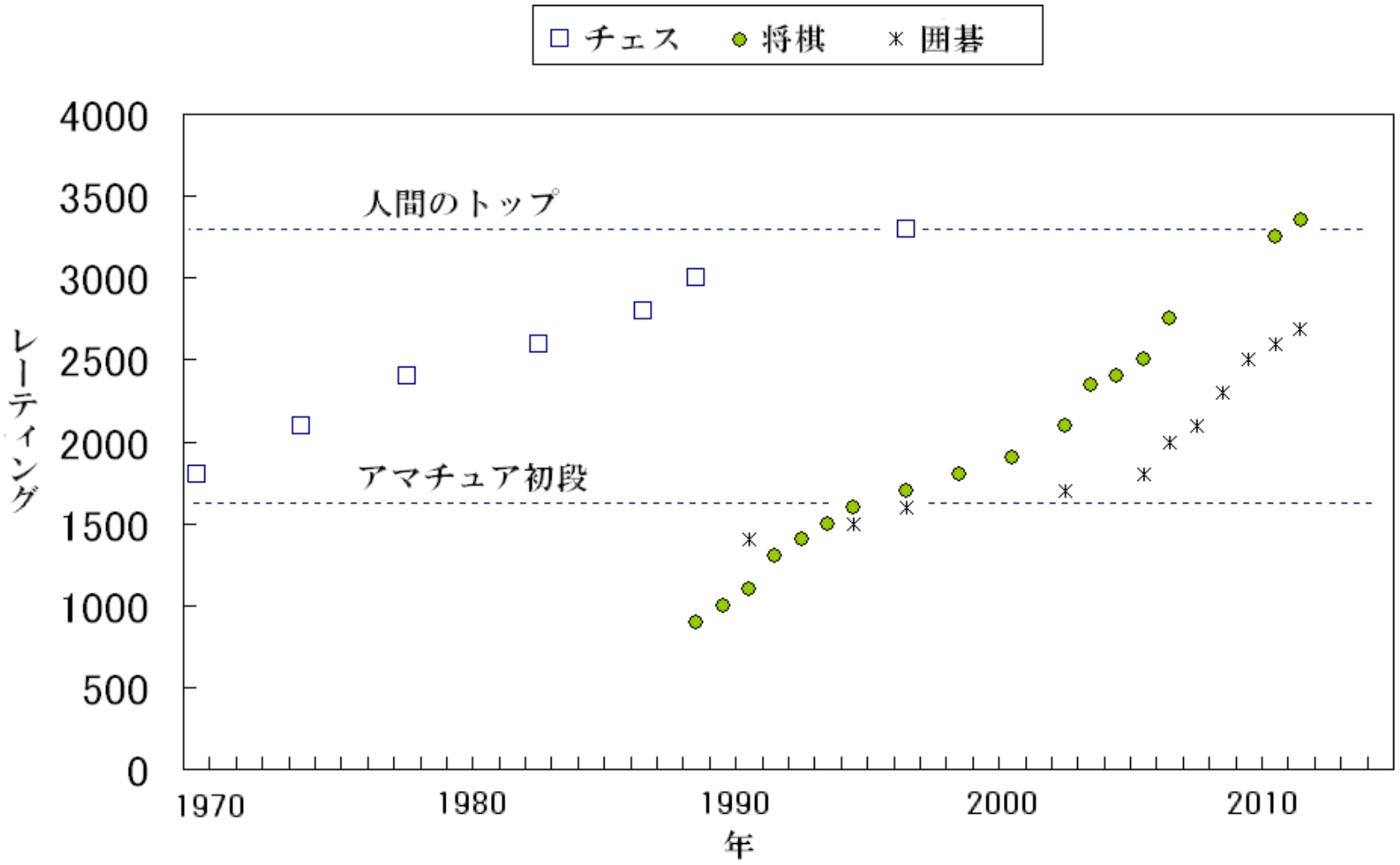
コンピュータ将棋

将棋



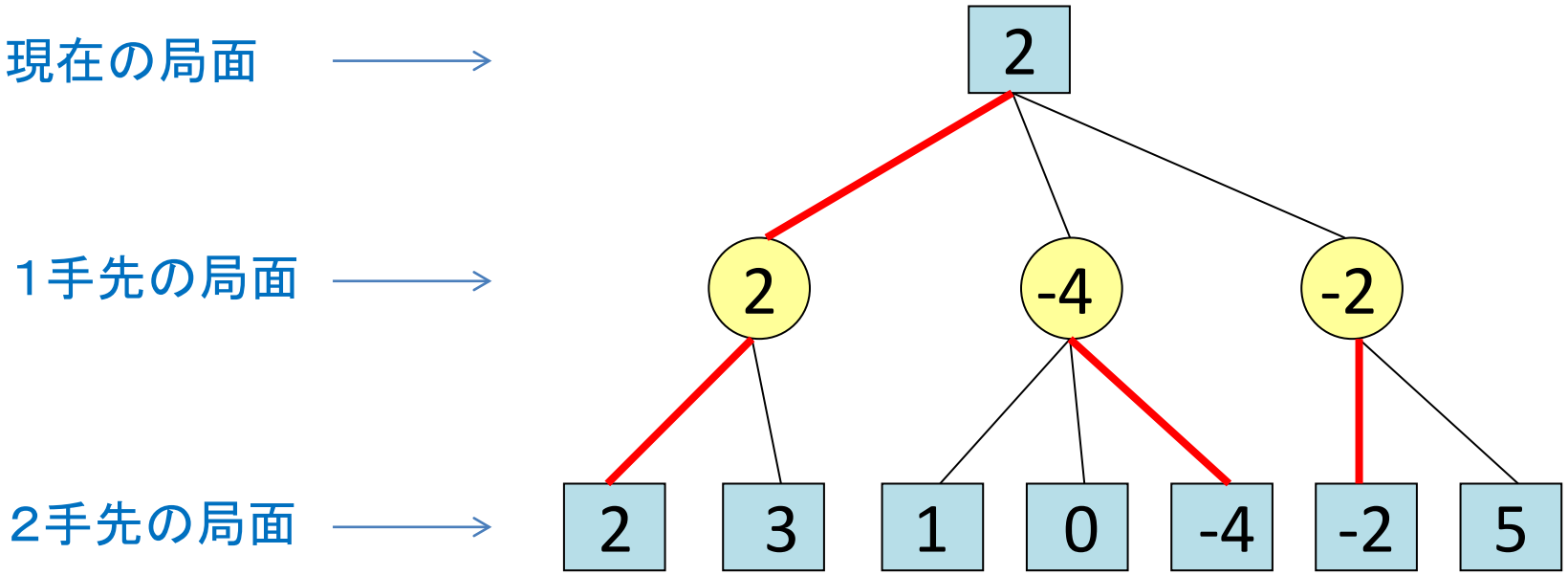
- Japanese chess
- 持ち駒のルール（取った駒が再利用できる）
- 将棋人口（1年に1回以上指した15歳以上の人の数）：700万人

コンピュータチェス・将棋・囲碁



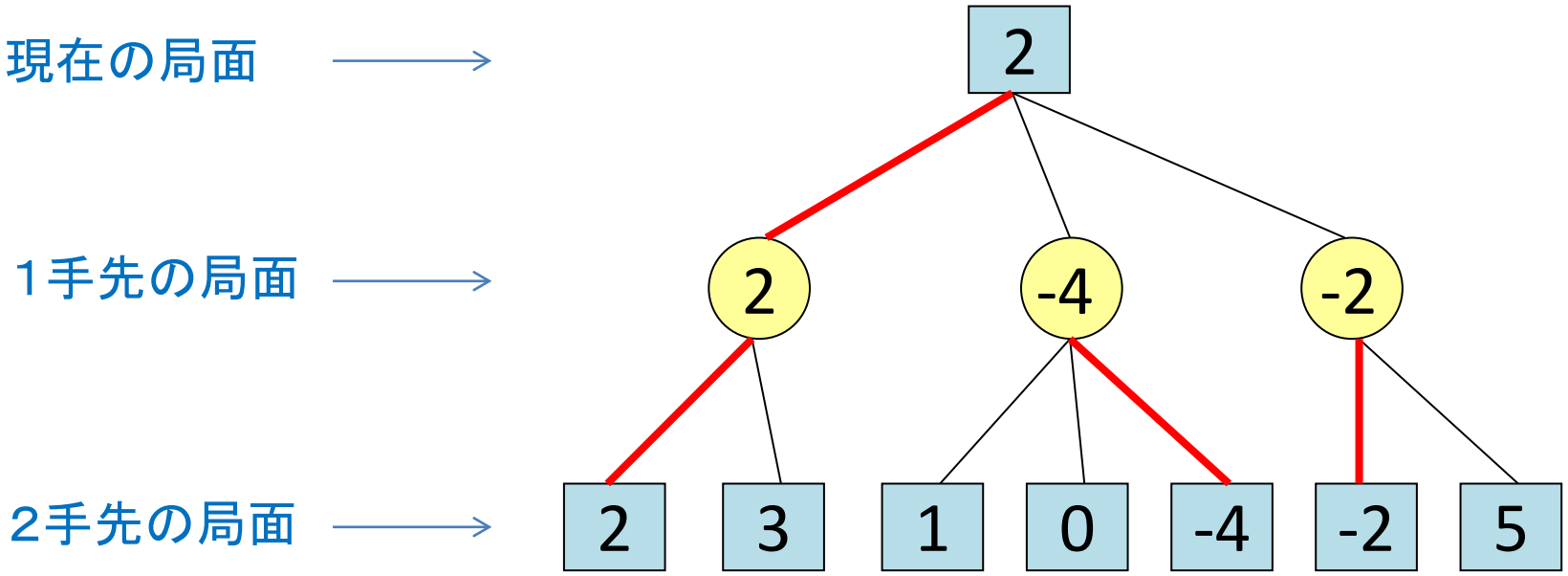
FPGAで将棋プログラムを作ってみるブログ
http://blog.livedoor.jp/yss_fpga/archives/53897129.html

コンピュータの思考法の原理



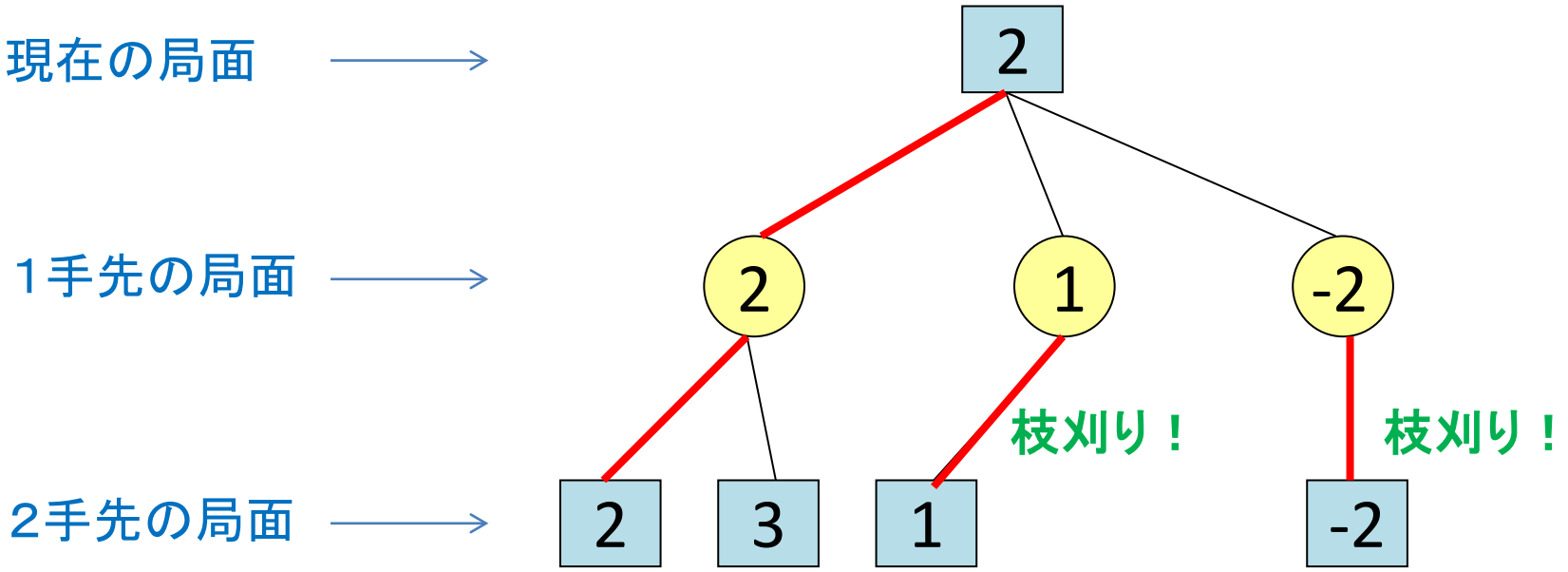
- **評価関数**によって末端局面の有利・不利の度合いを数値化
- お互いが自分にとって最も都合の良い手を選ぶと仮定して逆算（ミニマックス探索）

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

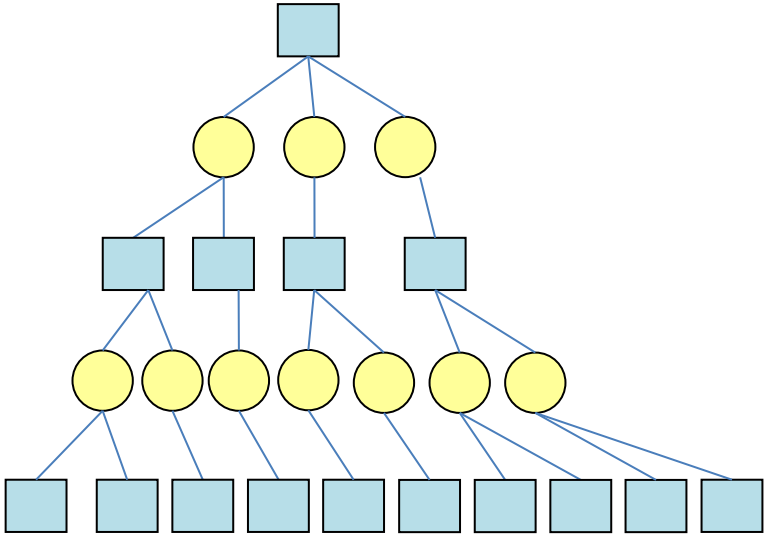
反復深化

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

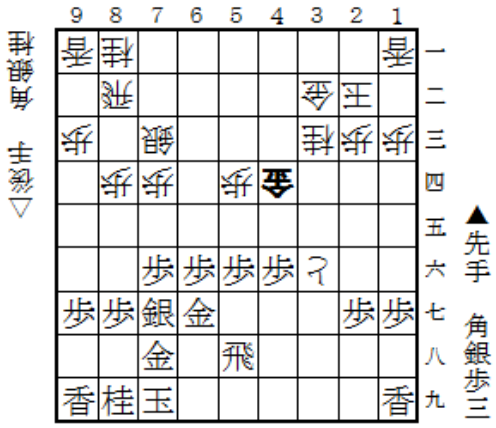


- 探索の最大深さを徐々に深くしていく
- 時間がなるまで繰り返す

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

駒の損得

- 駒の価値

駒種	価値
王	∞
飛	920
角	750
金	610
銀	570
桂	310
香	270
歩	100

駒種	価値
竜	1280
馬	1150
成銀	590
成桂	610
成香	630
と	660

評価関数

- 線形モデル
  - 重みベクトルと特徴ベクトルの内積でスコアを計算

重みベクトル

$$v(t) = \mathbf{w}^T \phi(t)$$

局面 t のスコア

局面 t の特徴ベクトル

特徴ベクトル

- 駒の損得を表現する特徴ベクトル

$$\begin{pmatrix} 1 \\ -1 \\ \vdots \\ 3 \end{pmatrix}$$

← （先手の飛車の数）－（後手の飛車の数）

← （先手の角の数）－（後手の角の数）

← （先手の歩の数）－（後手の歩の数）

重みベクトル

- それぞれの特徴の重要さを表す

$$\begin{pmatrix} 920 \\ 750 \\ \vdots \\ 100 \end{pmatrix}$$

← 飛車を1枚得していることの「重み」

← 角を1枚得していることの「重み」

← 歩を1枚得していることの「重み」

スコアの計算

- 重みベクトルと特徴ベクトルの内積をとる

$$\begin{pmatrix} 920 & 750 & \cdots & 100 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \\ \vdots \\ 3 \end{pmatrix} = -250$$

→ 後手が少し（歩2.5枚ぶん）有利

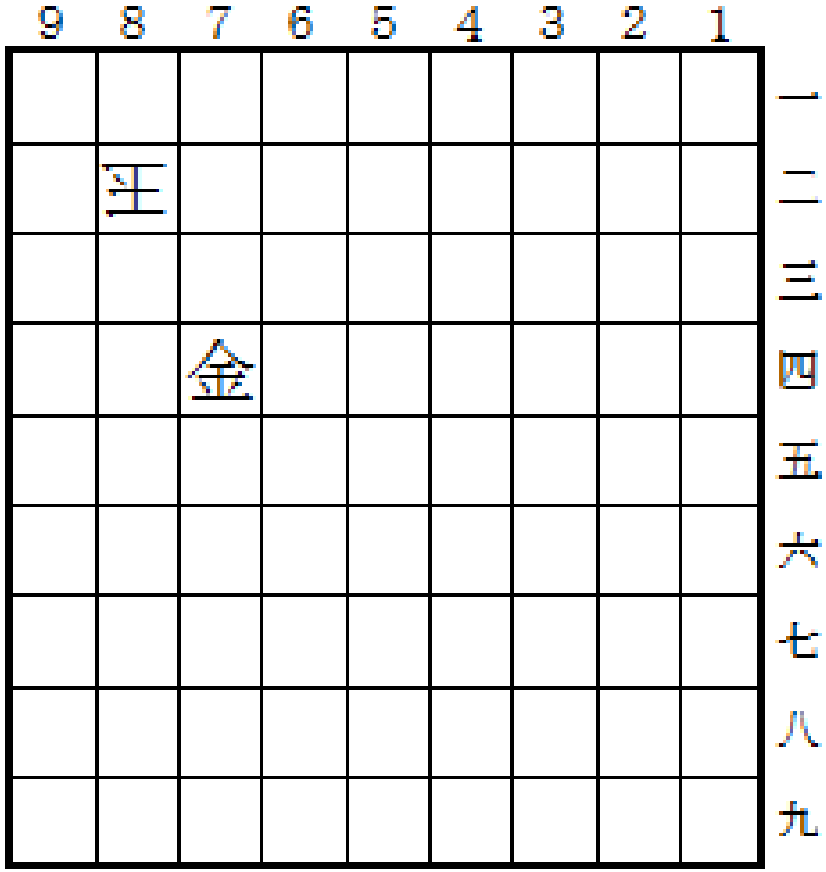
駒の働き

- 序盤は駒の損得だけ考えていればよい
- 終盤は「駒の損得より速度」
  - 隅のほうの駒を取りにいっている間に自分の玉が追い詰められる
    ⇒　昔の将棋ソフトの典型的な負けパターン

- 駒の働きを考慮することが非常に重要

駒の働きを表現する特徴ベクトル

- 例） 盤上の2つの駒の位置関係



どんな駒が　どこにいる

$(14 \times 81)^2 = 1285956$

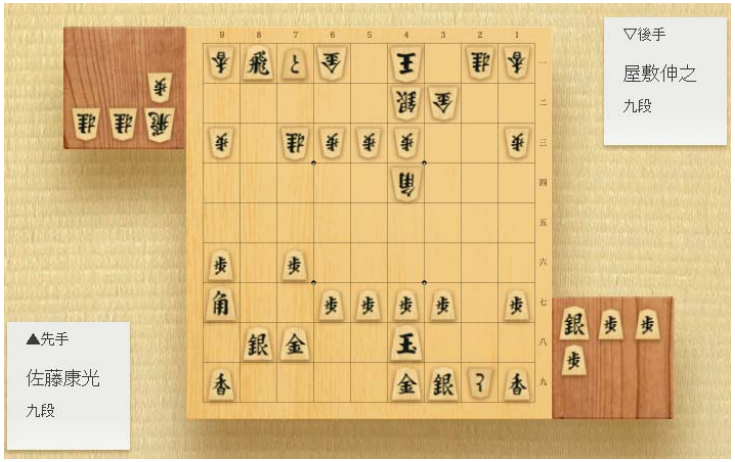
129万次元の特徴ベクトル

→重みベクトルを手作業で決めるのは無理！

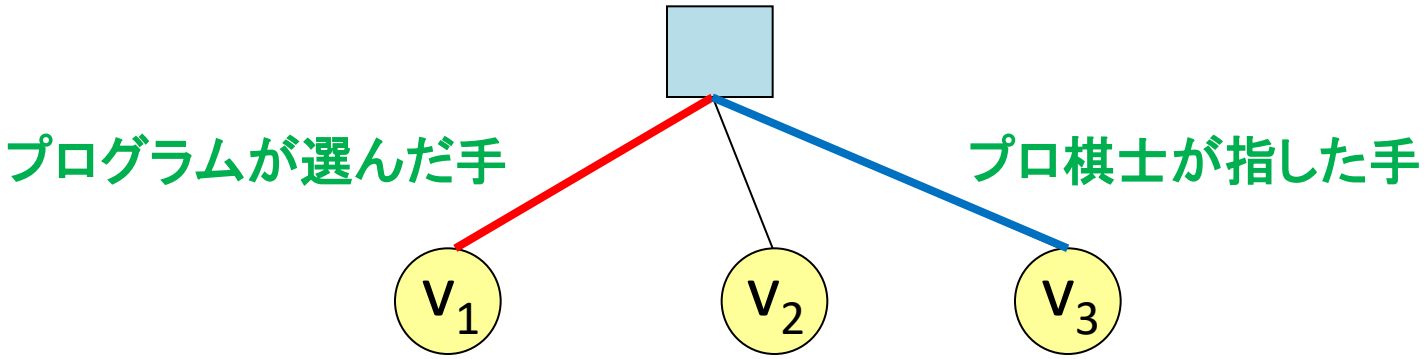
比較学習（comparison training）

- コンピュータチェスの評価関数の自動学習
  - Tesauro (2001), DeepThought
  - 教師付き学習
    - エキスパートの棋譜
  - 多少効果あり

- 将棋では大成功
  - 別名 Bonanza メソッド (保木, 2006)

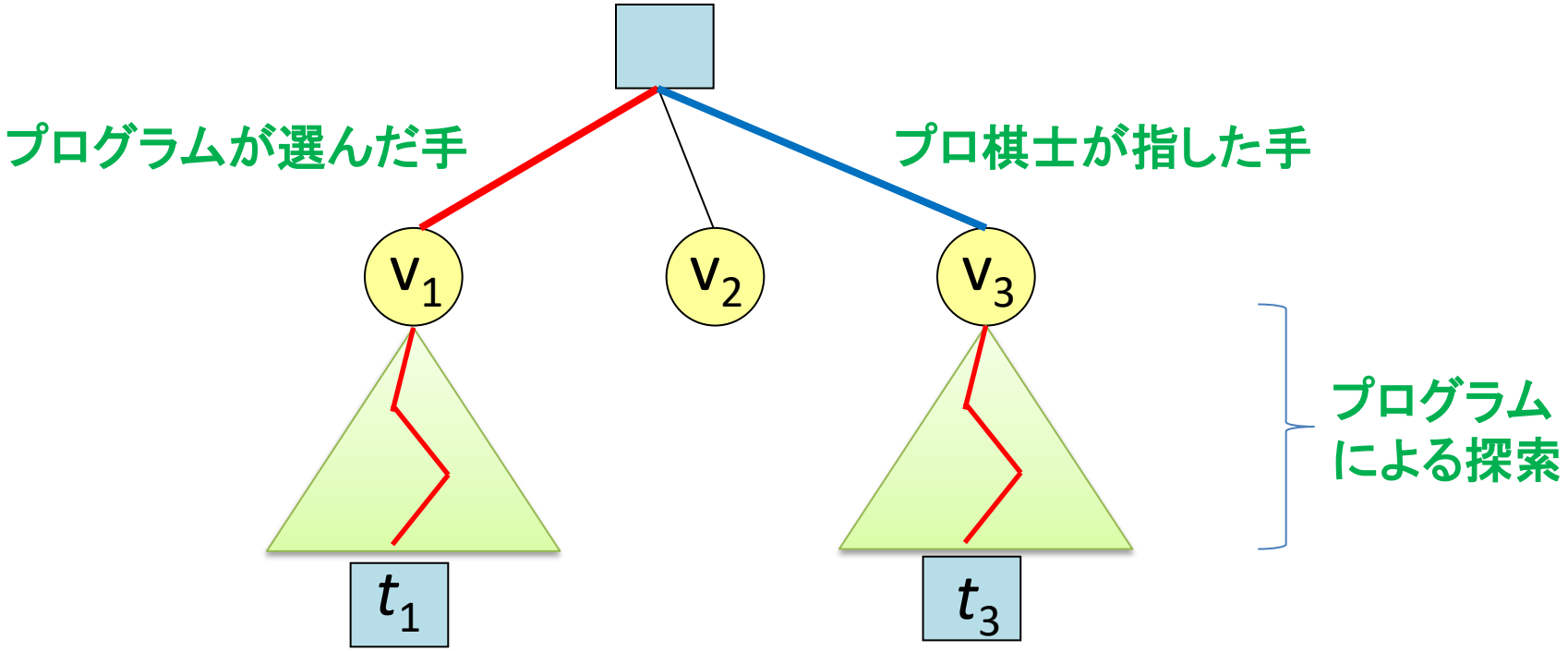


（先読みなしの）比較学習



- プロ棋士が指した手と同じ手を指すように調整
- $v_3 > v_1$ となるように重みベクトルを調整

先読み＋学習



- （t_3 の評価値）>（t_1 の評価値）となるように重みベクトルを調整
- プロ棋士の読みをプログラムの読みで代用

評価関数の学習法の例

- パーセプトロン学習
  1. 学習データ中の局面をランダムにひとつ選ぶ
  2. 最善手の選択を間違えるなら重みベクトルを更新
  3. 以上、繰り返し
- 更新式（正解手のスコアを超える手が1つの場合）

$$\mathbf{w} \leftarrow \mathbf{w} + \phi(t^*) - \phi(\hat{t})$$

正解手からの探索での
末端局面

間違えて選ぶ手からの
探索での末端局面

なぜ学習できるのか？

- 更新後のスコアの差を計算してみると・・・

正解手の更新後のスコア

$$\phi(t^*)(\mathbf{w}+\phi(t^*)-\phi(\hat{t}))=\mathbf{w}\phi(t^*)+\phi(t^*)^2-\phi(t^*)\phi(\hat{t})$$

間違えて選んだ手の更新後のスコア

$$\phi(\hat{t})(\mathbf{w}+\phi(t^*)-\phi(\hat{t}))=\mathbf{w}\phi(\hat{t})+\phi(t^*)\phi(\hat{t})-\phi(\hat{t})^2$$

両者の差

$$\underbrace{\mathbf{w}\phi(t^*)-\mathbf{w}\phi(\hat{t})}_{\text{もともとの差}}+\underbrace{(\phi(t^*)-\phi(\hat{t}))^2}_{\text{必ず正}}$$

AlphaZero (Silver et al., 2018)

- AlphaGo Zero の技術をチェス、将棋に適用
  - 事前知識が（ほぼ）ゼロの状態から強化学習

COMPUTER SCIENCE

A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play

David Silver[1,2*†], Thomas Hubert[1*], Julian Schrittwieser[1*], Ioannis Antonoglou[1], Matthew Lai[1], Arthur Guez[1], Marc Lanctot[1], Laurent Sifre[1], Dharshan Kumaran[1], Thore Graepel[1], Timothy Lillicrap[1], Karen Simonyan[1], Demis Hassabis[1†]

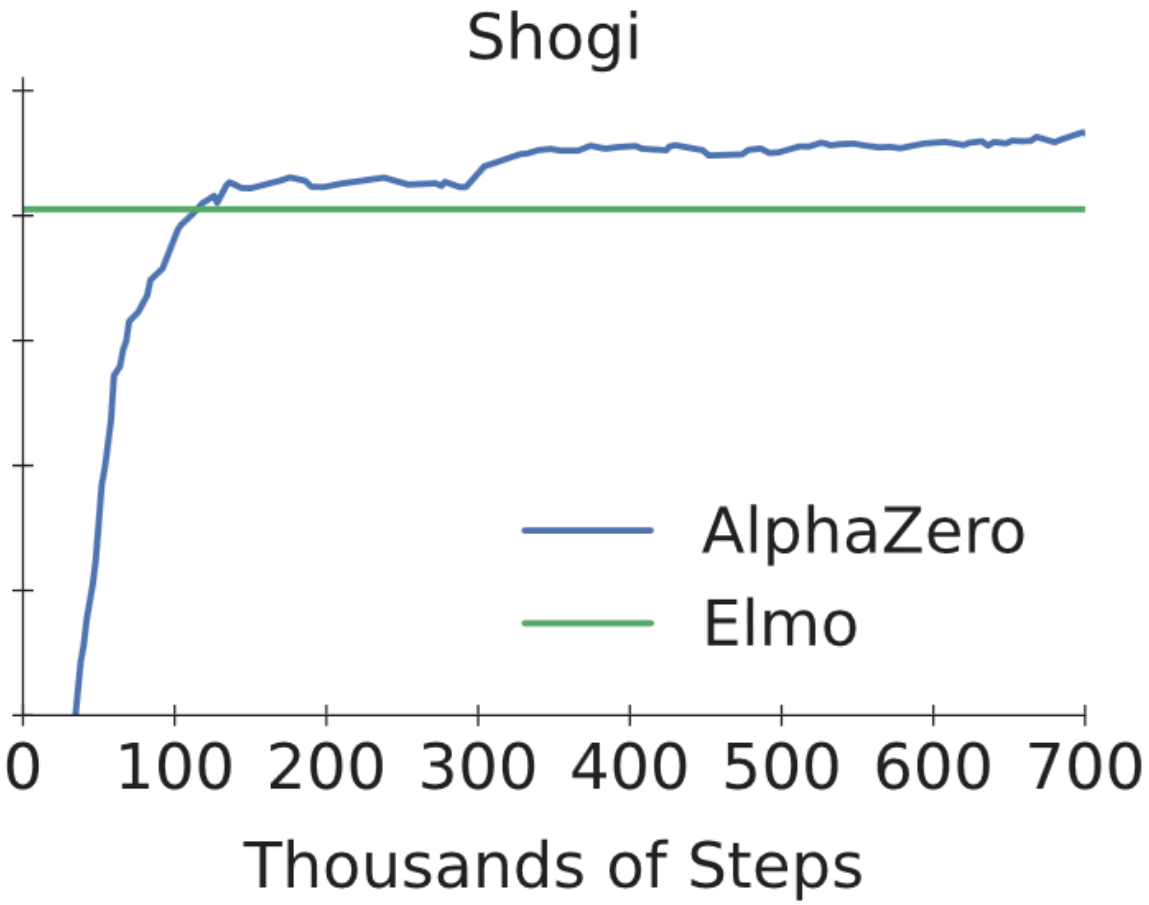
The game of chess is the longest-studied domain in the history of artificial intelligence. The strongest programs are based on a combination of sophisticated search techniques,

programmers, combined with a high-performance alpha-beta search that expands a vast search tree by using a large number of clever heuristics and domain-specific adaptations. In (*10*) we describe these augmentations, focusing on the 2016 Top Chess Engine Championship (TCEC) season 9 world champion Stockfish (*11*); other strong chess programs, including Deep Blue, use very similar architectures (*1*, *12*).

In terms of game tree complexity, shogi is a substantially harder game than chess (*13*, *14*): It is played on a larger board with a wider variety of pieces; any captured opponent piece switches sides and may subsequently be dropped anywhere on the board. The strongest shogi programs, such as the 2017 Computer Shogi Association (CSA) world champion Elmo, have only recently de-

Silver *et al.*, *Science* **362**, 1140–1144 (2018) 7 December 2018

AlphaZero (Silver et al., 2018)



羽生竜王のコメント

"Some of its moves, such as moving the King to the centre of the board, go against shogi theory and - from a human perspective - seem to put AlphaZero in a perilous position. But incredibly it remains in control of the board. Its unique playing style shows us that there are new possibilities for the game."

Yoshiharu Habu, 9-dan professional, only player in history to hold all seven major shogi titles

https://deepmind.com/blog/alphazero-shedding-new-light-grand-games-chess-shogi-and-go/

将棋ソフトの新しい役割

- 将棋プログラムは強くなりすぎた
  - もはやプロでも全く勝てない
- 人間の上達を助けるツール
  - 「悪手」「好手」の指摘
  - 検討のツール
  - 指導対局
- 観戦のサポート

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

コンピュータポーカーの進歩

- 2008年
  - Heads-up Limit Texas hold'em でトッププレイヤに勝利
- 2015年
  - Heads-up Limit Texas hold'em が解かれる
- 2017年
  - Heads-up No-Limit Texas hold'em でトッププレイヤに勝利

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

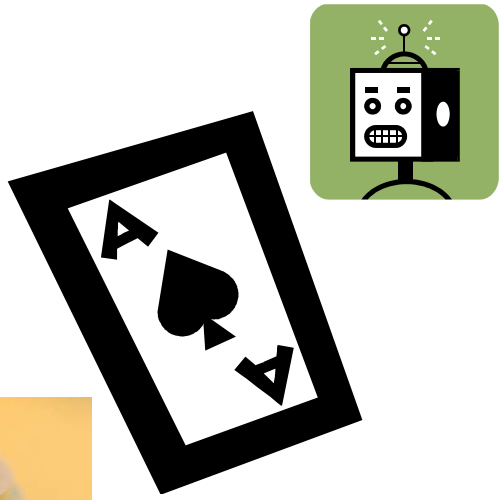
- グー、チョキ、パーで利得が違う場合
  - グーで勝ったら３点
  - チョキで勝ったら２点
  - パーで勝ったら１点

- 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

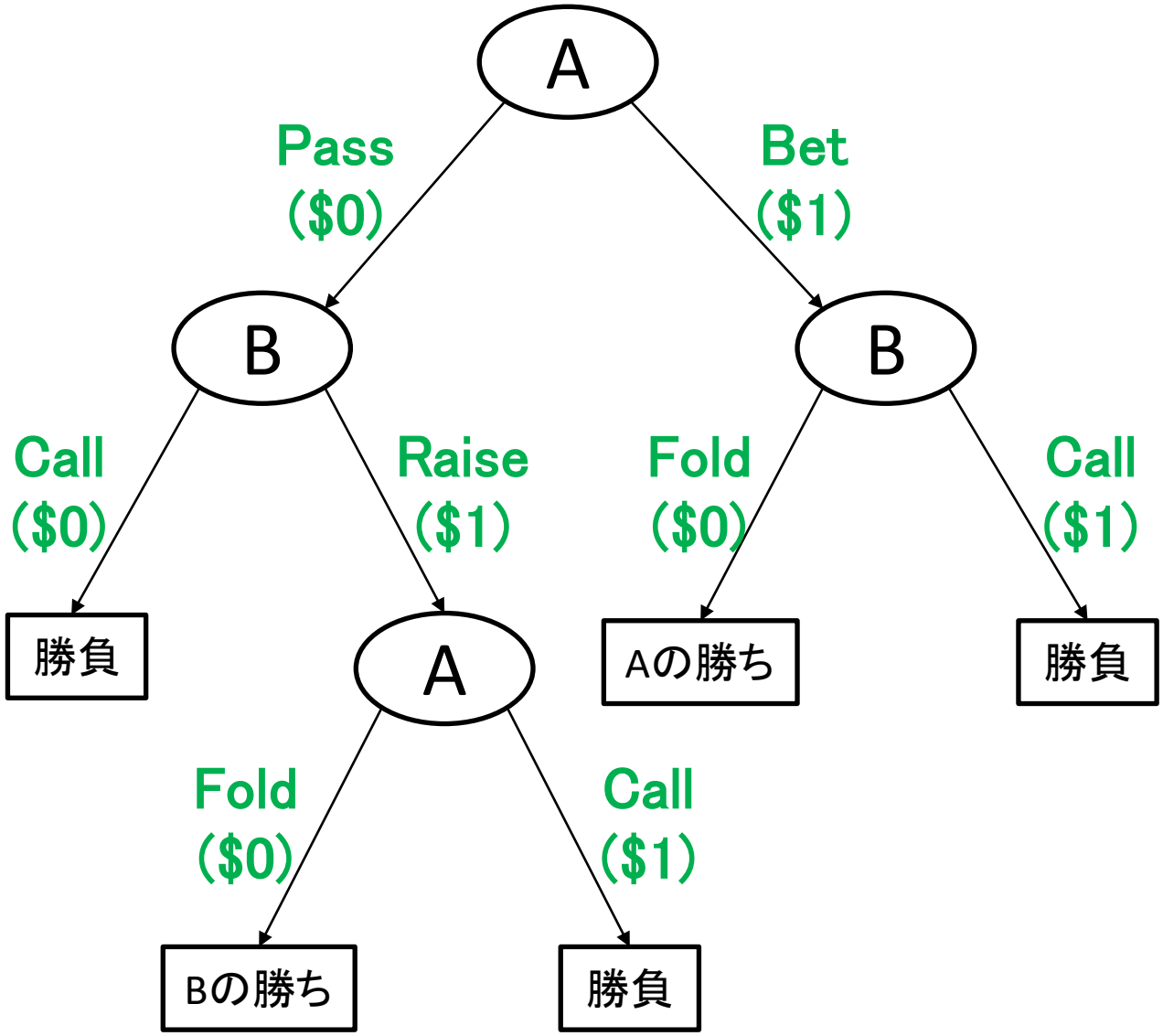
- 単純化されたポーカー
  - 1対1
  - カードは1枚
  - 強いカードを持っている方が勝ち
  - Ante（最低掛け金）は $1



- ラウンドの進行
  - プレイヤ A の手番
    - Pass or Bet $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/

One-card Poker



ナッシュ均衡戦略(プレイヤA)

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/

ナッシュ均衡戦略（プレイヤB）

Pass 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

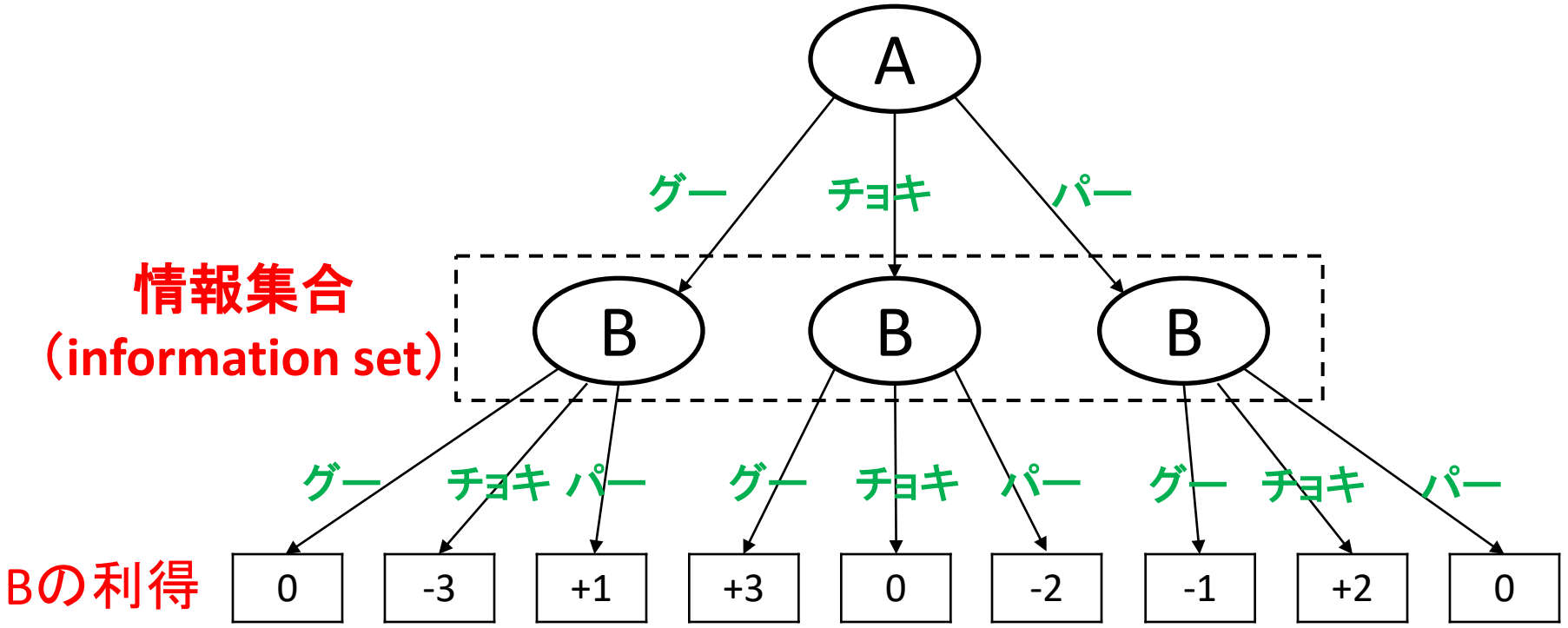
http://www.cs.cmu.edu/~ggordon/poker/

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

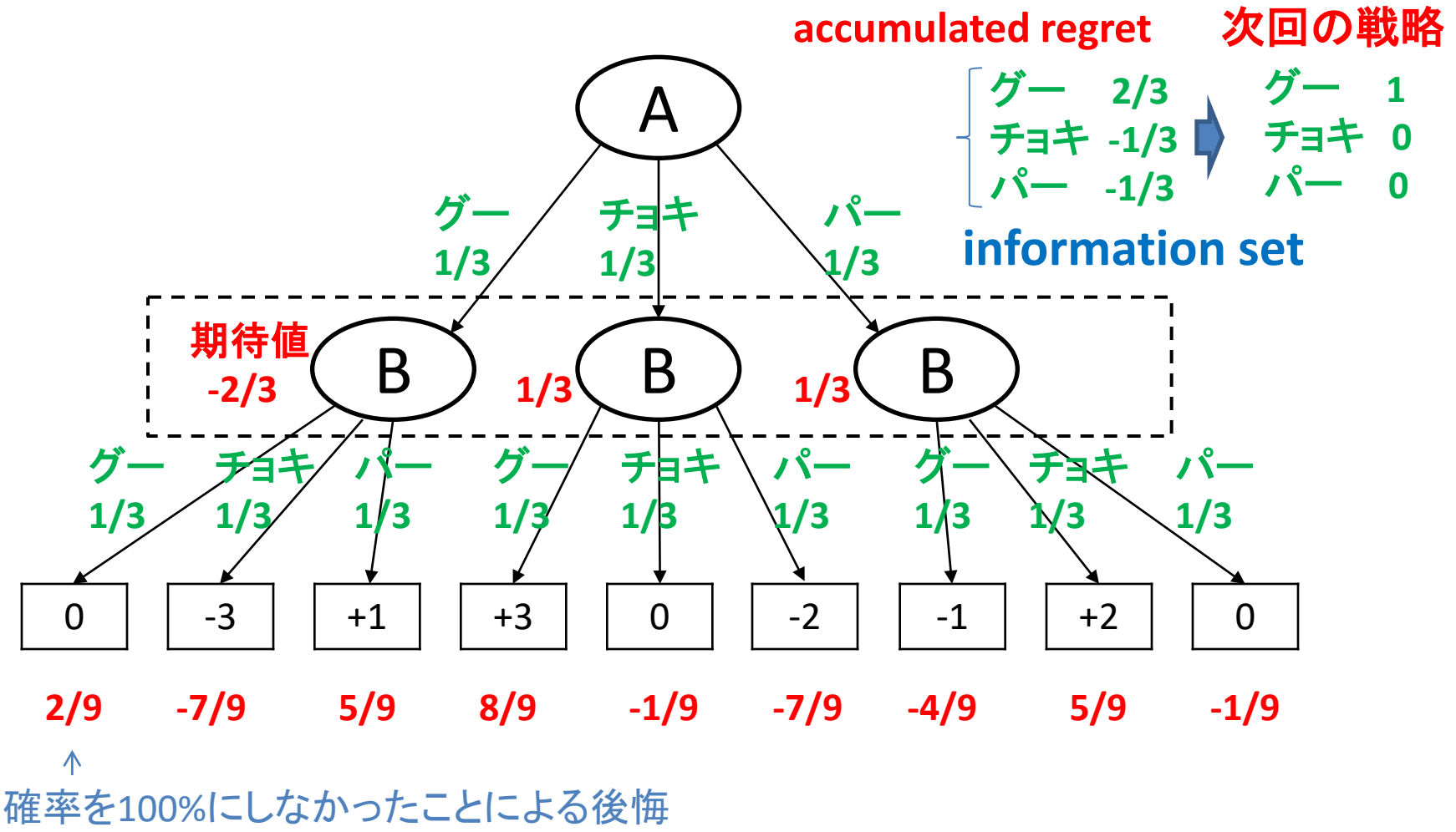
$$R_i^T = \frac{1}{T} \max_{\sigma_i^* \in \Sigma_i} \sum_{t=1}^{T} \left(u_i\left(\sigma_i^*, \sigma_{-i}^t\right) - u_i\left(\sigma^t\right) \right)$$

  - 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