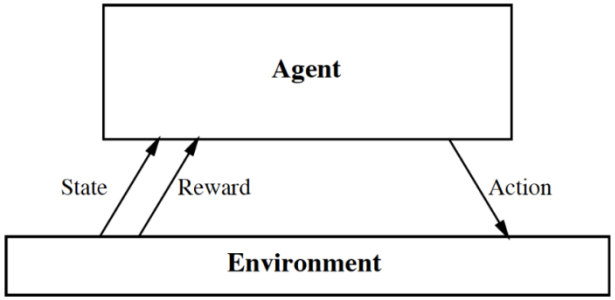
強化学習

伊庭　斉志

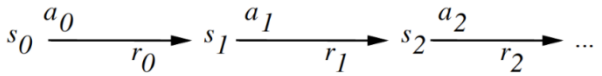
学習のいろいろ

- 教師あり学習
  - すぐにフィードバックがある(入力に対しての正解・不正解)
  - classification, regression



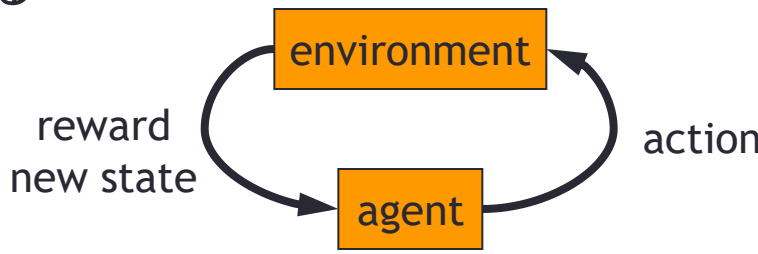
- 教師なし学習
  - フィードバックがない
  - clustering

- 強化学習
  - おくれてフィードバック(報酬:reward)がある
  - より一般的
  - 環境との相互作用により目的を達成する
  - エピソード: 与えられた問題の初期状態からタスクの完了または失敗による終了状態までの一連の試行



強化学習が必要な例

- 強化学習
  - おくれてフィードバック（報酬：reward）がある
  - より一般的
  - 環境との相互作用により目的を達成する



- こんな状況を考えよう
  - 自分で準電子ながら部屋を掃除するロボット
  - ロボットサッカー
  - 株にどう投資するか？
  - ヘリコプターをどう飛ばすか？
    - などなど

なにがむつかしいのか？

われわれは環境について何も知らない

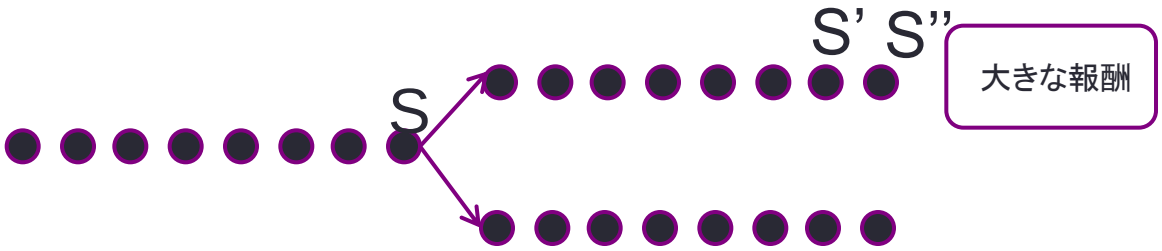
しかしそれでも状況・報酬を観察して行動しなくてはならない

- 行動の結果が不確定かもしれない
- 環境を完全には認識できないかもしれない
- 報酬が確率的かもしれない
- 報酬はしばしば遅れる (例：迷路での餌探し)
- 環境が自分の行動にどう変化するかの知識（モデル）を持たないかもしれない
- 報酬がどう払われるかの知識（モデ
- 環境は学習途中に変化するか
- exploration v.s. exploitation

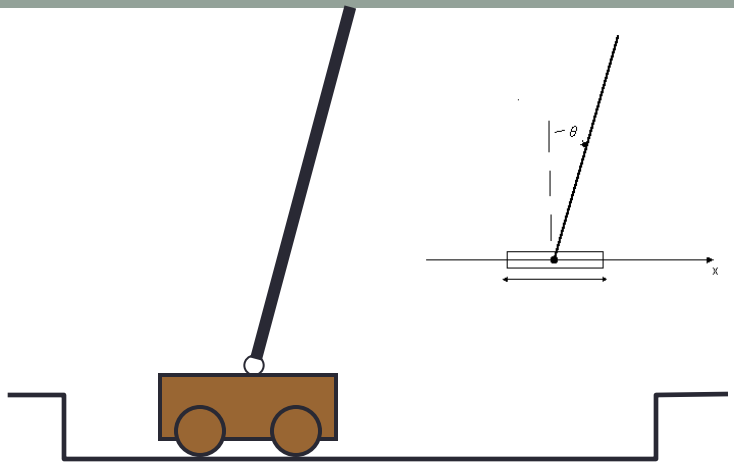
Credit-assignment problem

Credit Assingment Problem

- 遅れた報酬は学習を困難にする
- 状態Sでの選択が重要であったが、状態S'での行動が状態S''での大きな報酬をもたらした
- このような状態にどう対処すべきか?
- 例: ゲームでの間違った手

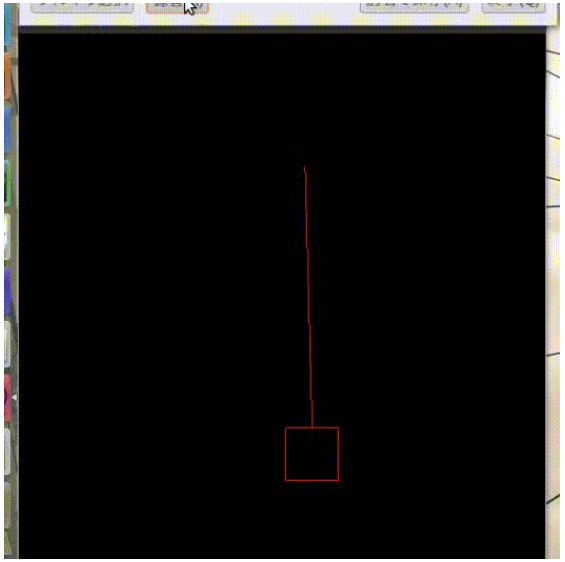


例をみてみよう



- pole-balancing
- TD-Gammon [Gerry Tesauro]
- helicopter [Andrew Ng]

- “good” や “bad” を教える教師が存在しない
  - 報酬“10” はいいのか悪いのか?
  - 報酬は遅れて与えられる



- 制御理論に類似する
  - より一般的で少ない制約である

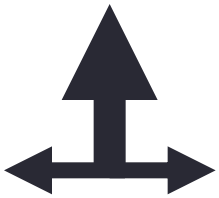
- 環境を探索して、経験から学べ
  - 盲目的な探索ではなく、より賢くなれ

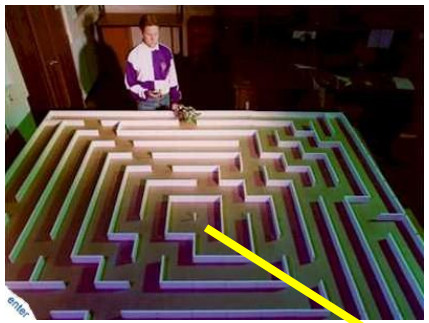
部屋の中のロボット

1				+1
2				-1
3	START			
	1	2	3	4

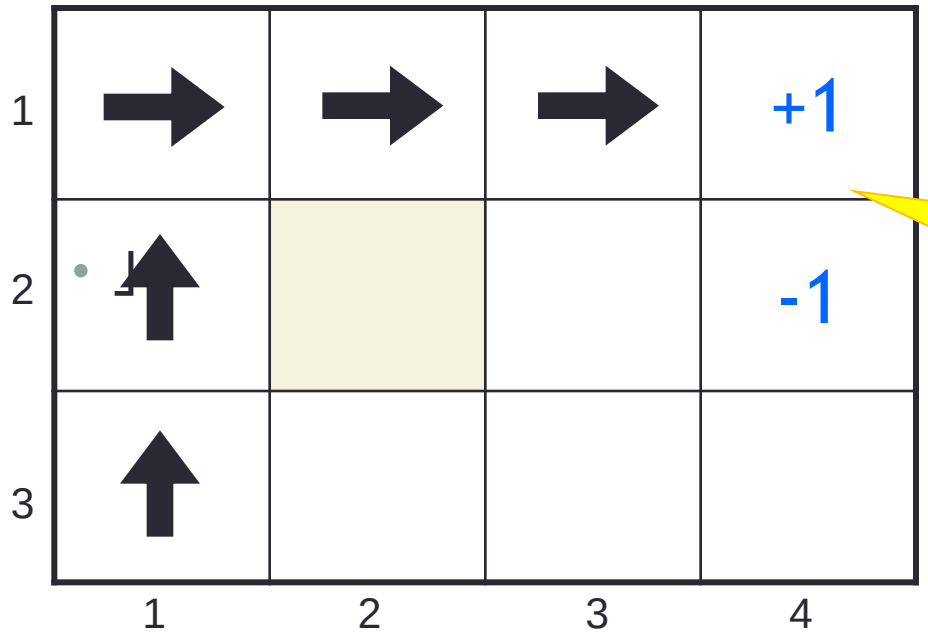
- 報酬 : +1 @ [4,3], -1 @ [4,2]

行動: UP, DOWN, LEFT, RIGHT





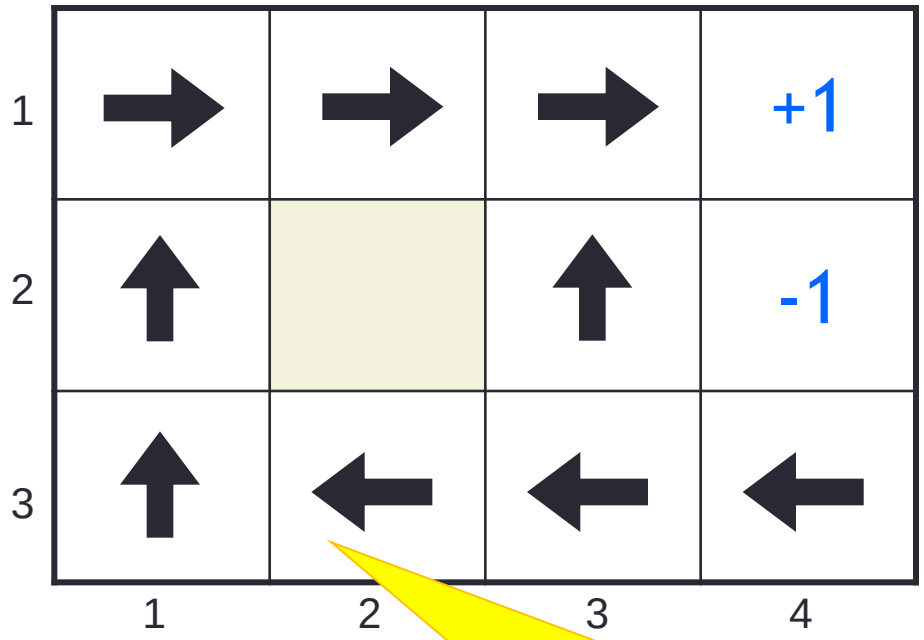
正解は？



+1からの逆方向
探索などで得られる

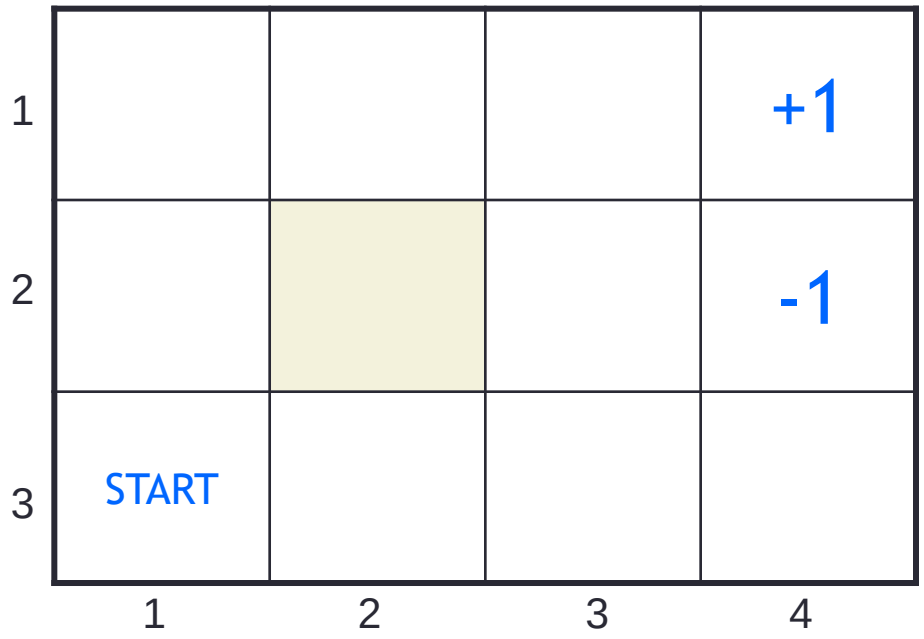
- 行動が決定的のときのみ
  - 行動が確率的なときはどうするか？
- 解/方策
  - 各状態から行動・方策へのマッピング

部屋の中のロボット(不確実な環境)



最適な行動列は存在しない
最も効率的な行動な方策が見つかるか？

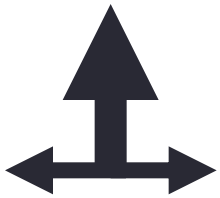
部屋の中のロボット（不確実な環境）



行動: UP, DOWN, LEFT, RIGHT

目標: UP

80% move UP
10% move LEFT
10% move RIGHT

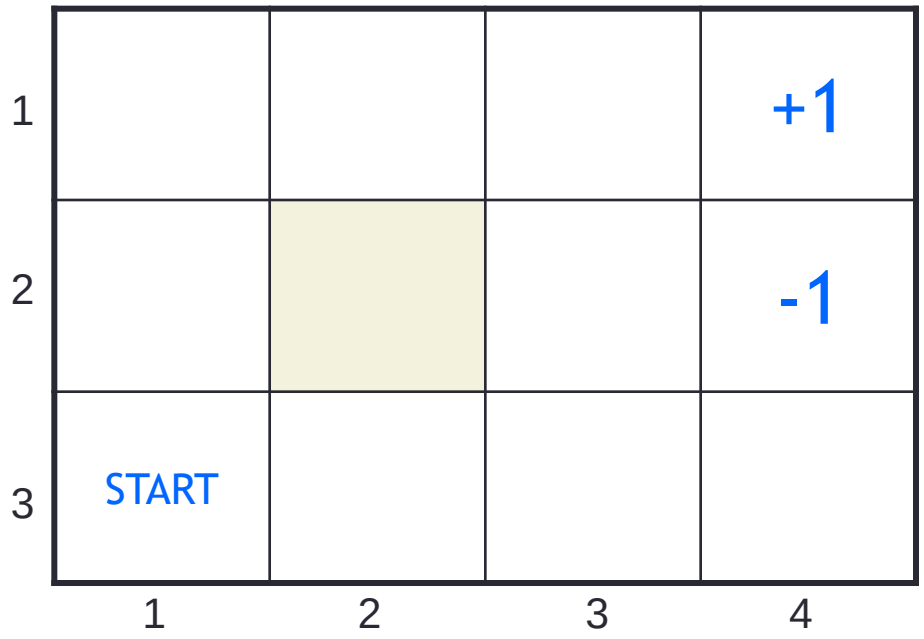


報酬 +1 @ [4,3], -1 @ [4,2]
各ステップでの報酬（一歩動くことのコスト）は -0.04

- エピソードと訓練例

(1,1)→(1,2)→(1,3)→(1,2)→(1,3)→(2,3)→(3,3) →(3,4) **+1**
0.57 0.64 0.72 0.81 0.9
(1,1)→(1,2)→(1,3)→(2,3)→(3,3)→(3,2)→(3,3)→(3,4) **+1**
(1,1)→(2,1)→(3,1)→(3,2)→(4,2) **-1**

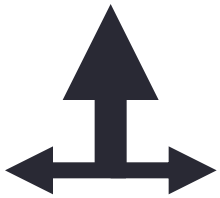
部屋の中のロボット（不確実な環境）



行動: UP, DOWN, LEFT, RIGHT

目標：UP

80% move UP
10% move LEFT
10% move RIGHT



報酬 +1 @ [4,3], -1 @ [4,2]
各ステップでの報酬（一歩動くことのコスト）は -0.04

- U(s) = sからスタートしたときに期待利得
  - 単にsの報酬ではなく、将来に通る状態も考慮する
- U(1,1)=R(1,1)+0.8 * U(1,2)+0.1 * U(2,1)+0.1 * U(1,1)

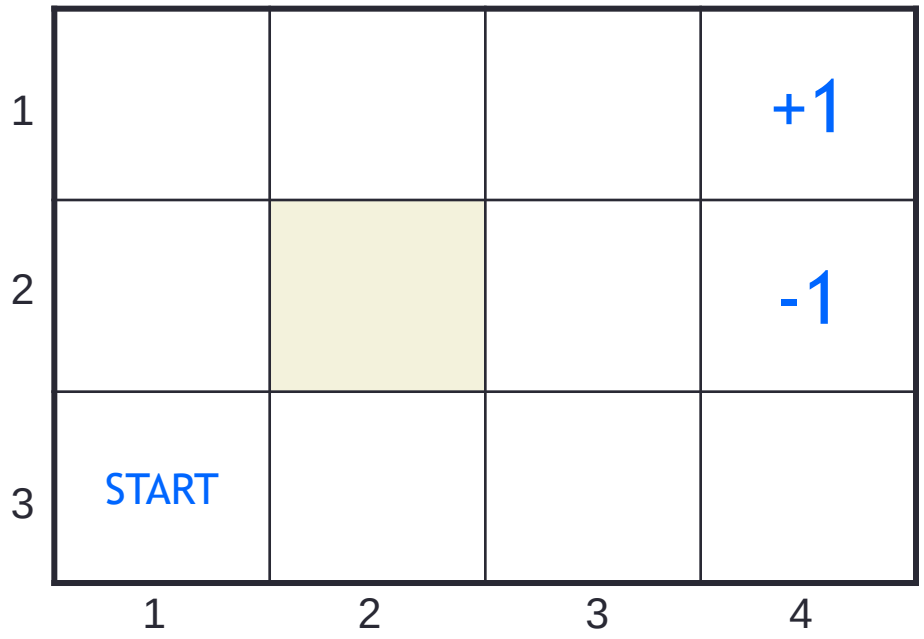
現在の報酬

UPの確率

RIGHTの確率

LEFTの確率
壁に衝突
結果的にstay

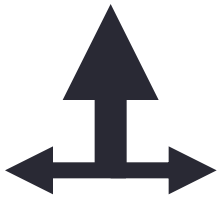
部屋の中のロボット（不確実な環境）



行動: UP, DOWN, LEFT, RIGHT

目標：UP

80% move UP
10% move LEFT
10% move RIGHT



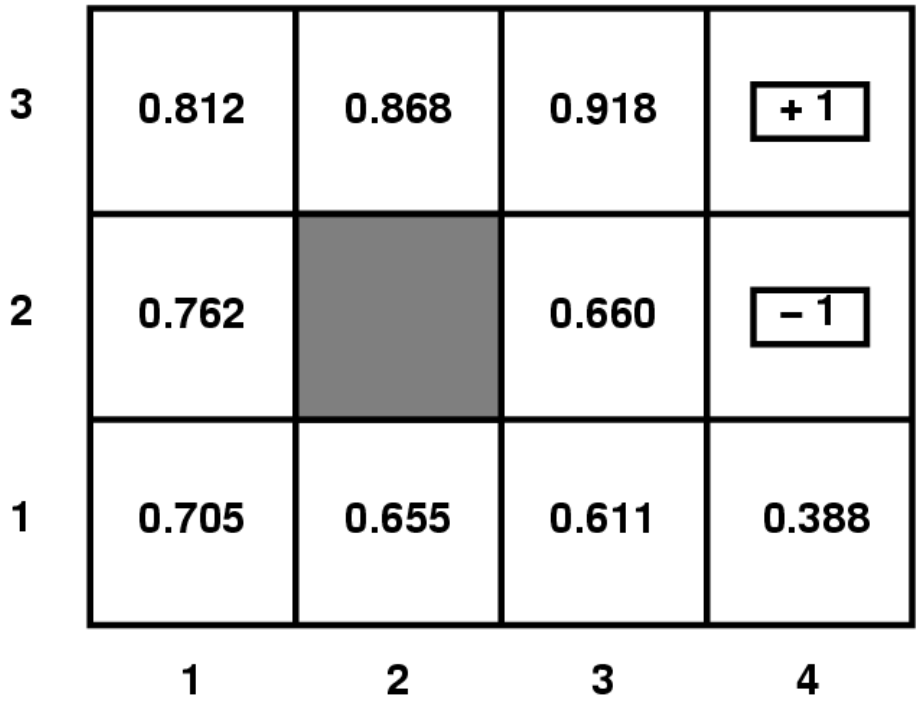
報酬 +1 @ [4,3], -1 @ [4,2]
各ステップでの報酬（一歩動くことのコスト）は -0.04

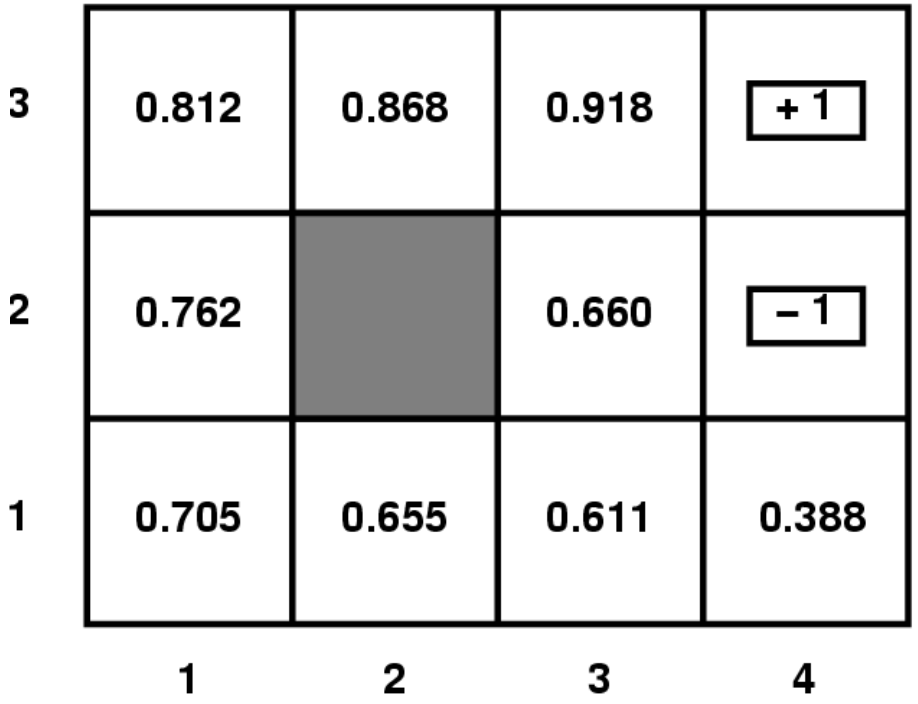
- U(s) = sからスタートしたときに期待利得
  - 単にsの報酬ではなく、将来に通る状態も考慮する
- $U(1,1)=R(1,1)+\gamma * \{0.8 * U(1,2)+0.1 * U(2,1)+0.1 * U(1,1)\}$

割引率

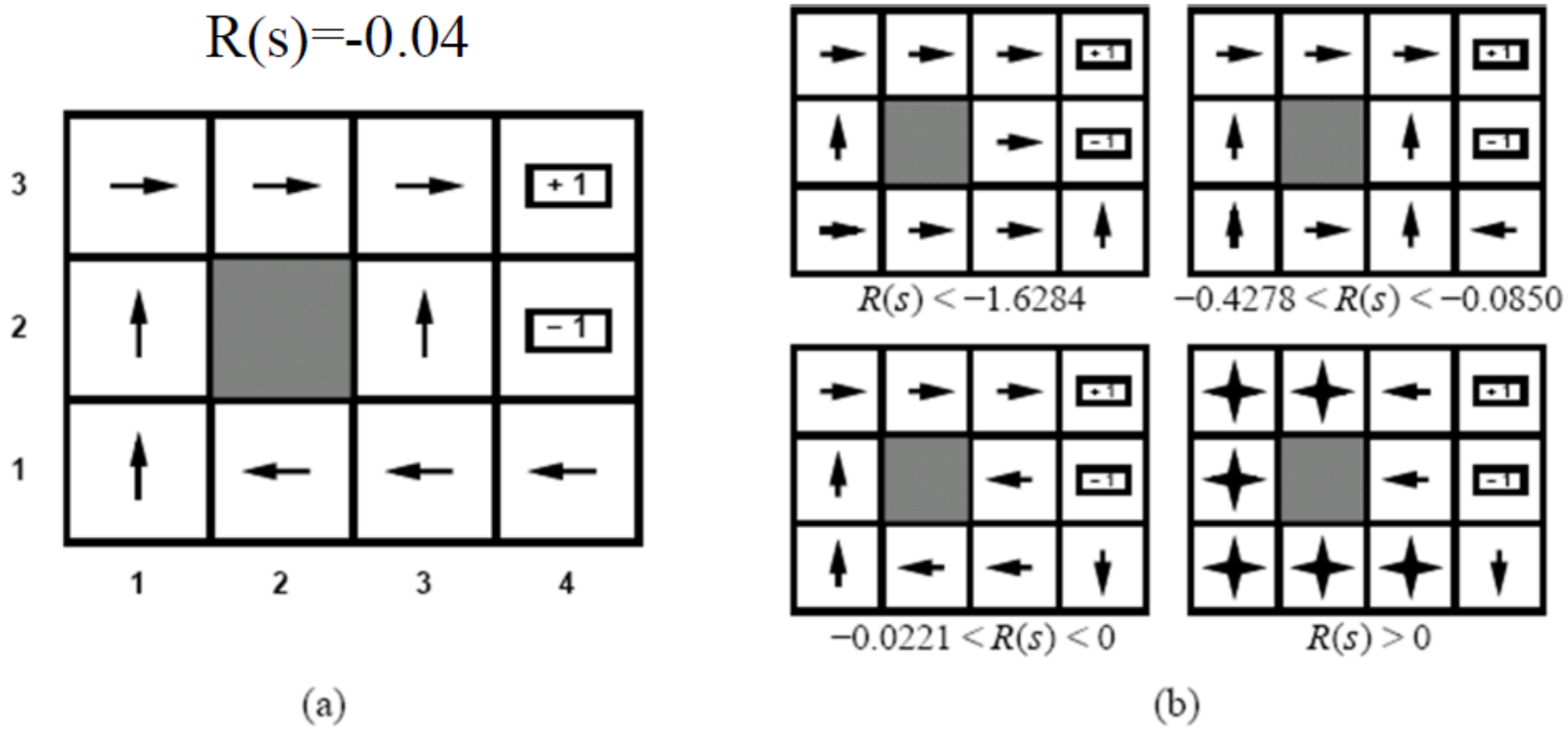
Bellman方程式

各ステップでの報酬: -0.04



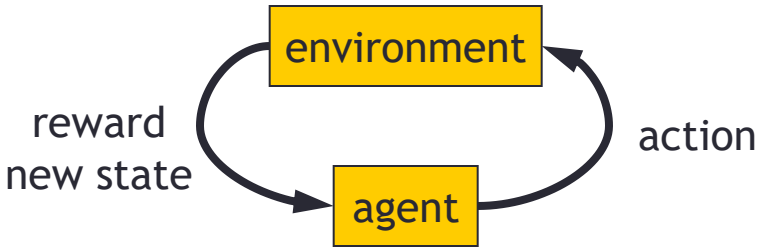


報酬でまとめてみる



マルコフ決定過程(MDP)

- 状態の集合 S, 行動の集合 A, 初期状態 S_0
- 推移モデル P(s,a,s')
  - P([1,1], up, [1,2]) = 0.8
- 報酬関数 r(s)
  - r([4,3]) = +1



- 目標: 報酬の合計を最大にすること
- エピソード
  - 与えられた問題の初期状態からタスクの完了または失敗による終了状態までの一連の試行
- 方策: S から A へのマッピング π
  - π(s) か π(s,a) (決定的 vs. 確率的)
- 強化学習
  - 推移と報酬は通常は得られない
  - 経験に基づいて方策をどう変えるか？
  - 環境をいかに探索するか？

問題の定義

•方策：　環境の状態からエージェントがとるべき行動へのマッピング．
例：もし部屋が汚れていれば、掃除せよ。もしバッテリーが不足していれば、充電せよ。

$$\pi : S \rightarrow A$$

状態価値関数

• 何を最適すべきか？　将来の期待収益

$$V^{\pi}(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots$$

$$= \sum_{i=0}^{\infty} \gamma^i r_{t+i} \qquad 0 \leq \gamma < 1$$

直近の報酬の方が未来の報酬よりも価値がある．

γ（割引率）が0なら 目先のことしか考えない

Bellman方程式

- 状態価値関数: $V^{\pi}(s)$
  - s　にはじまり　π　をとったときの期待収益

- $V^{\pi}(s)$ は以下の再帰関係を満たす（Bellman方程式という）

$$
\begin{aligned}
V^{\pi} &= E_{\pi}\{\sum_{k=0}^{\infty}\gamma^{k} r_{t+k+1} \mid s_t = s\} \\
&= E_{\pi}\{r_{t+1} + \sum_{k=0}^{\infty}\gamma^{k} r_{t+k+2} \mid s_t = s\} \\
&= \sum_{a}\pi(s,a)\sum_{s'} P^{a}_{ss'}\{r^{a}_{ss'} + \gamma E_{\pi}\{\sum_{k=0}^{\infty}\gamma^{k} r_{t+k+2} \mid s_{t+1} = s'\}\} \\
&= \sum_{a}\pi(s,a)\sum_{s'} P^{a}_{ss'}\{r^{a}_{ss'} + \gamma V^{\pi}(s')\}
\end{aligned}
$$

方策πが状態sで行動aをとる確率

- ただし、

$$
\begin{aligned}
P^{a}_{ss'} &= P\{s_{t+1} = s', s_t = s, a_t = a\} \quad \text{状態sと行動aから状態s'に推移する確率} \\
r^{a}_{ss'} &= E\{r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'\} \quad \text{状態sと行動aの次状態s'での報酬の期待値}
\end{aligned}
$$

価値関数

- 期待収益 $V^{\pi}(s)$

- 最適の方策$\pi^*(s)$ は何か? $V^* = \max_s V^{\pi}(s)$

- 解答: 以下を最大にする行動となる

$$\pi^*(s) = \arg\max_a \left[r(s,a) + \gamma V^*(\delta(s,a)) \right]$$

- 以下の2つの仮定が必要である

- 行動“a” を 状態“s” で実行したときの報酬 $r(s,a)$ が既知
- 行動“a” を 状態“s” で実行したときの次状態 $s_{t+1} = \delta(s_t, a)$ が既知

- Bellman方程式で動的計画法を用いて解ける
- しかしこの仮定は現実的でない

- より実際的には、$r(s,a)$ と δ を学習していくべきである

動的計画法による学習

Step1 全ての s において $V(s)$ の値を初期化する (通常は 0). $V(s)$ の初期値を $V_0(s)$ と表す.

価値反復法

Step2 全ての s に対して以下を実行する.

$$V_k(s) = \max_a \sum_{s'} P^a_{ss'} \{ r^a_{ss'} + \gamma V_{k-1}(s') \} \qquad (1.20)$$

Step3 $\max_s \mid V_k(s) - V_{k-1}(s) \mid$ の値が十分に小さな値となるまで Step2 を繰り返す.

- 以下の2つの仮定が必要である
- 行動“a” を 状態“s” で実行したときの報酬 $r(s,a)$ が既知
- 行動“a” を 状態“s” で実行したときの次状態 $s_{t+1} = \delta(s_t, a)$ が既知

- しかしこの仮定は現実的でない

- 状態集合全体に対して繰り返し計算を行うため，問題の複雑化に伴って状態空間が指数関数的に増加する

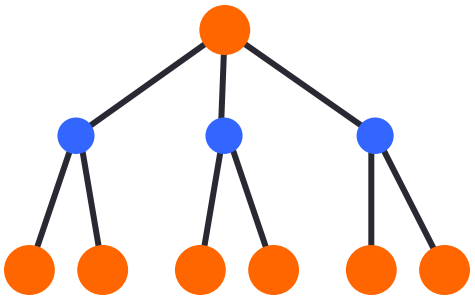
- より実際的には、$r(s,a)$ と δ を学習していくべきである

方策の評価と改善

- 方策の評価: π -> V^{π}
  - Bellman 方程式
  - 再帰的な過程

$$V_{k+1}(s) = \sum_{a} \pi(s,a) \sum_{k'} P^{a}_{ss'} \left[r^{a}_{ss'} + \gamma V_k(s') \right]$$

  - 任意の価値関数 V_0 ではじめて、V_k が収束するまで実行する



- 方策の改善: V^{π} -> π'

$$\pi'(s) = \arg\max_{a} Q^{\pi}(s,a)$$

$$= \arg\max_{a} \sum_{s'} P^{a}_{ss'} \left[r^{a}_{ss'} + \gamma V^{\pi}(s') \right]$$

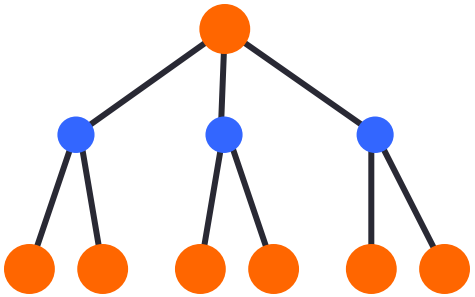
  - π' がπ より良い, または π' が最適である (つまり $\pi = \pi$')

方策･価値の繰り返し

- 方策の繰り返し

$$\pi_0 \rightarrow^E V^{\pi_0} \rightarrow^I \pi_1 \rightarrow^E V^{\pi_1} \rightarrow^I \ldots \rightarrow^I \pi^* \rightarrow^E V^*$$

  - 2重のネストした繰り返し;　遅すぎる
  - V^{π_k} に収束する必要はない
    - そこにすすめばいい



- 価値 $V_{k+1}(s) = \max_a \sum_{s'} P^a_{ss'} \left[r^a_{ss'} + \gamma V_k(s') \right]$

  - Bellman 方程式を更新に用いる
  - V^*　に収束する

TD（Temporal Difference）学習

- 動的計画法の計算量を減らしたい
- 最終結果を待たずに，評価を途中で更新しながら$V^{\pi}(s)$の推定を行う手法
- 基本的な方針
  - 各行動ごとに価値関数の局所的な更新を行う
  - 推移確率の完全な評価を試みない

TD（Temporal Difference）学習

- モンテカルロ法と動的計画法の統合
  - モンテカルロ法的: モデルを必要とせず、経験から学ぶ
  - 動的計画法的: 成功者の価値から学ぶ
  - モンテカルロ法よりも普通は早くなる
- 定数のαによるモンテカルロ法:
  - 更新すべきエピソードの終わりまで待つ

$$V(s_t) \leftarrow V(s_t) + \alpha \left[R_t - V(s_t)\right]$$

目標

- もっとも簡単な TD　:　TD(0)
  - 成功者に基づいて、各ステップで更新する

$$V(s_t) \leftarrow V(s_t) + \alpha \left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right]$$

TD（Temporal Difference）学習

- 更新規則　：　Bellman方程式を近似する

$$V(s_t) \leftarrow V(s_t) + \alpha \left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \right]$$

学習率

割引率

次状態に基づく、利得の（ノイズのある）サンプル値

- 更新では（ノイズのある）利得サンプルの平均を維持する。
- 学習率がサンプルに従って減っていく（例：1/n）のなら、利得の評価値は真の値に収束する。
- VがBellman方程式を満たすと更新は0となる。

$$V_{k+1}(s) = \sum_a \pi(s,a) \sum_{k'} P^a_{ss'} \left[r^a_{ss'} + \gamma V_k(s') \right]$$

TD(λ)学習

- より一般的なTD学習
- nステップ後での評価を比較して価値関数を更新する.
- nステップ後までの報酬を考慮にいれると，最終的に得られる報酬は次のようになる.

$$R_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n V^\pi(s_{t+n})$$

- TD(λ)学習　：　次のようなnステップの報酬の重みつき平均値を考える

$$R_t^\lambda = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} R_t^{(n)}$$

正規化のための項

- λ＝1：　モンテカルロ法
- λ＝0：　1ステップのみのバックアップ

モンテカルロ法 vs. TD

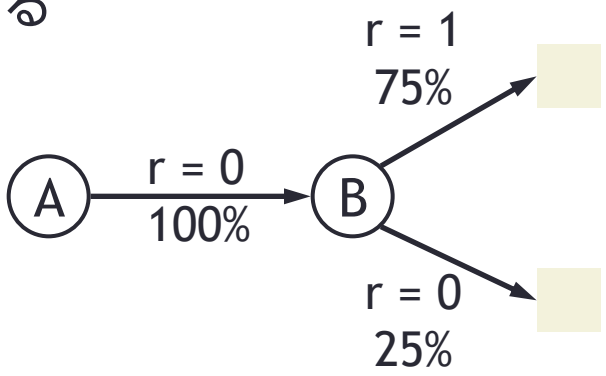
- 8つのエピソードの例:

A – 0, B – 0	B – 1	B – 1	B - 1
B – 1	B – 1	B – 1	B – 0

- モンテカルロ法とTD法はともに V(B) = 3/4　となる

- モンテカルロ法: V(A) = 0
  - 学習データの誤差を最小にする値に収束する

- TD法: V(A) = 3/4
  - マルコフ過程の ML 評価に収束する



Q値の導入

- $r(s,a)$ と $s_{t+1} = \delta(s_t, a)$ を学習するには？
- 直接的に良い　状態ー行動　のペアを表す関数を得る
- どの状態でどの行動をとるかの基準値　Q(s,a)　を導入する
- Q(s,a) 値：行動aを状態sでとったあとで最適な方策に従ったときの期待値
- Q(s,a) が与えられれば、$r(s,a)$と　$s_{t+1} = \delta(s_t, a)$を知らなくても、最適な方策を実行できる
- 次のようにすればよい

$$\pi^*(s) = \arg\max_a Q(s,a)$$

$$V^*(s) = \max_a Q(s,a)$$

Q値の最適値は？

- Q(s,a) 値：行動aを状態sでとったとり、そのあとで最適な方策に従ったときの期待値

•行動の選択基準

$$\pi^*(s) = \arg\max_a Q(s,a)$$

$$V^*(s) = \max_a Q(s,a)$$

•Bellman方程式に基づく制約

$$Q(s_t, a_t) = r_{t+1} + \gamma \sum_{s_{t+1}} P^a_{s_t s_{t+1}} \max_{a'} Q(s_t, a')$$

Exploration / Exploitation

TD(λ) や Q学習でよく用いられる例

1. ある確率1－εで最良の行動a_t^*を選択し、εでランダムな行動を選択する（ε greedy法）

$$a_t = a_t^* = \arg\max_{a_t} Q(a_t)$$

2. Boltzmann探索

時刻tに行動aが選ばれる確率は以下の式に従う。
ただしTは焼きなましの温度である。

$$P_{a_t} = \frac{e^{\frac{Q(a_t)}{T}}}{\sum_b e^{\frac{Q(b)}{T}}}$$

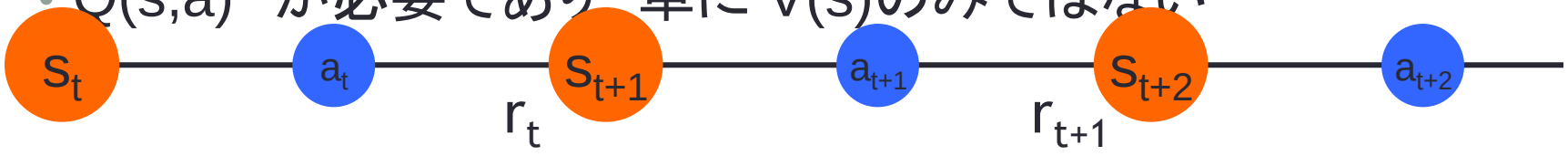
- Q値を学んでいるとき(off-policy learning)には、現在の方策に単に従わないことが重要.
- 局所解に陥る可能性があるからである。つまり、まだ見ぬより良い解があるかもしれない。
- 新しいものにときどき挑戦するのがいい。

強化学習の方法

- V値の最適値を学習するのではなく、直接Q値の最適値を学習する
  - Q(s,a) 値：行動aを状態sでとったとり、そのあとで最適な方策に従ったときの期待値
  - 最適なQ値は $V(s) = \max_{a'} Q(s, a')$ を満たす
- Active Learning (Off-Policy Learning)
  - どのような方策に従っているかは関知しない.
  - 代わりに最良のQ値を利用する.
  - つまり学習される方策と実際に実行されている方策は異なる可能性がある.
  - 例： Q学習 - Watkins
- Passive Learning (On-Policy learning)
  - ある方策を実行することにより得られる経験に基づいて更新を行う.
  - この実行中の方策は必ずしも安定したものではない.
  - 最適値に収束する
  - 例： Sarsa - Sutton

Sarsa（State action reward state action)

- Q(s,a)　が必要であり，単に V(s)のみではない



$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]$$

- 制御法
  - ランダムな方策から開始する
  - 各ステップで Q と　π を更新する
  - 行動が選択されるのを待って，その行動からQ値をバックアップする

Q学習

- Q-learning: off-policy
  - Q は直接 Q* (Bellman方程式の解) を近似する
  - 続く方策には独立である
  - 必要なのは、各 (s,a) の組を更新することだけである

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

次状態に基づくQ値のノイズのあるサンプル値

  - TD学習のように各行動のあとで更新を実行する.
  - 直接的に報酬 r_{t+1} を観察することに注意せよ.

- Cf. Sarsa

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]$$

Q学習のアルゴリズム

$Q(s,a)$ を任意に初期化;

repeat (全エピソードについて){

s を初期化;

while (s が終端状態ではない){

Q から導かれる方策に従い s での行動 a を選択する;

行動 a を取り，報酬と次状態 r, s' を観測する;

全ての a' に対して

$Q(s',a')$ のテーブルを検索し，最大値 $\max_{a'} Q(s',a')$ を探す;

$Q(s,a) \Leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$;

$s \Leftarrow s'$;

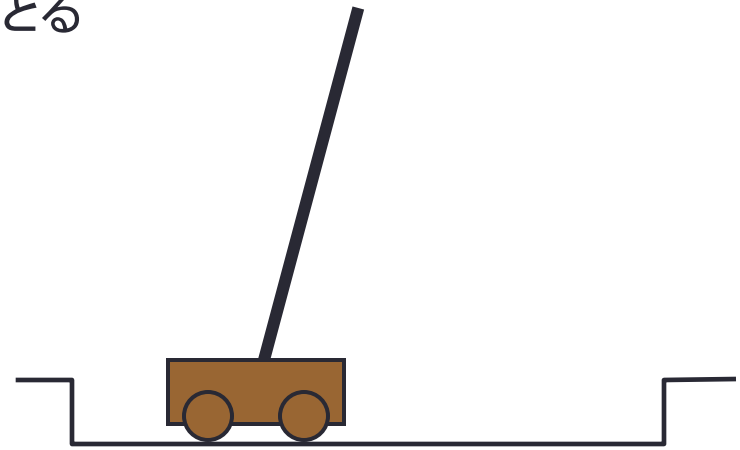
}

}

- Q値を直接学習するのでモデルを必要としない
- Q値の表現に |S|x|A| サイズのテーブルのみが必要
- 直接的にQ値を用いるExplore/exploitの戦略が可能
  E.g. Boltzmann exploration.

状態の表現法

- pole-balancingの目標
  - 車を左右に動かしながら、poleのバランスをとる
- 状態表現
  - 車の位置、速度
  - Poleの速度、角速度
- *Markov* 過程か?
  - より詳細な情報が必要
  - Poleの曲がり具合、センサー、温度。。。
- 解答
  - 4つの状態変数の粗い離散値
    - left, center, right
  - 完全にはMarkov過程ではないがうまくいく



関数近似：Q学習

- すべての状態、行動についてQ(s,a)の値を記憶するには膨大なメモリが必要となる
  - 例：将棋では　101^{70}くらい
- Q(s,a)を線形に近似する [Baird, 1995]

$$Q_{\mathbf{w}}(s,a) = \mathbf{w} \cdot \mathbf{x}(s,a)$$

重みベクトル　　特徴ベクトル

- すると、重みパラメータは次の更新式となる

$$w_i \leftarrow w_i + \alpha\left[(r_t + \gamma \max_{a'} Q_{\mathbf{w}}(s_{t+1}, a')) - Q_{\mathbf{w}}(s_t, a_t)\right] x_i(s_t, a_t)$$

利得の期待値の最大値（≒目標値）　Q関数の値

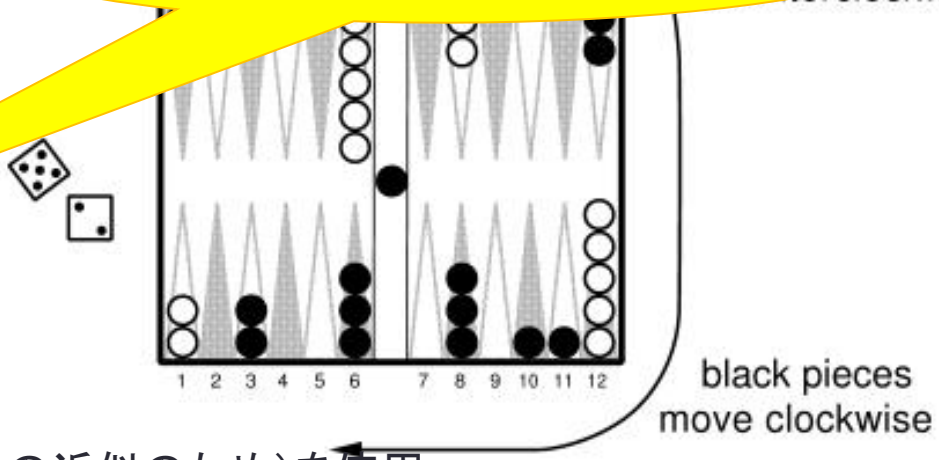
報酬の設計

- 迷路のロボット
  - エピソードのタスクで脱出すると+1，各ステップでは0 、割引はしない
- チェス
  - GOOD: 勝と+1，負けると-1
  - BAD:　敵の駒をとると+0.25
    - 悪い理由：　負けても高い報酬となってしまう
- 報酬とは？
  - 何をしたいかを表すものである
  - どうやって達成するかではない
- こつ

  - 正の報酬はしばしばとても“隔靴掻痒”となる
  - サブゴールを達成するような報酬が良い (ただし領域固有の知識が必要)
  - 初期の方策や初期の価値関数を工夫せよ

バックギャモン

状態空間が非常に大きい



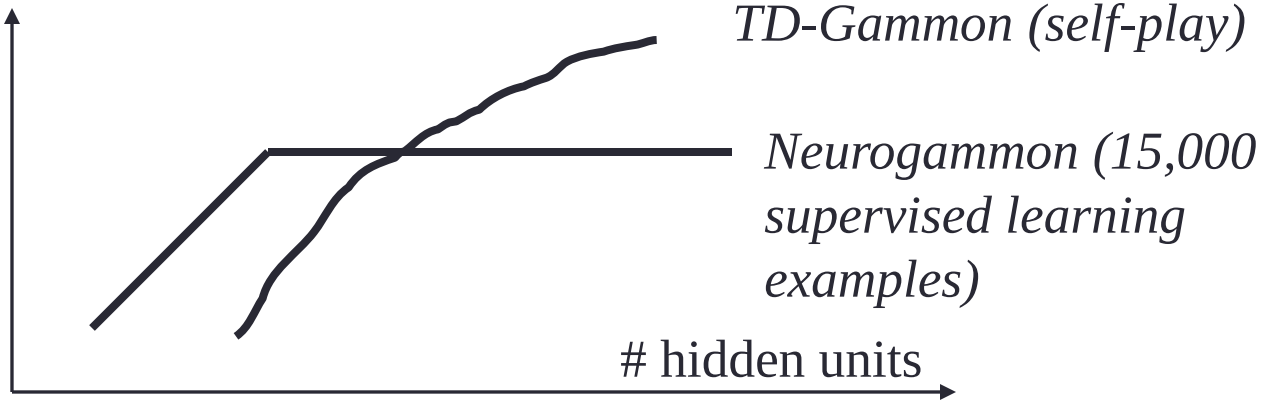
- ルール
  - 30 駒, 24 の場所
  - さいころの目が2, 5のとき
    - 2, 5を動く
  - ヒット、ブロック
  - ゲームの木の分枝数: 400
- 実装
  - TD(λ) とニューラルネット（価値関数の近似のため）を使用
  - 各位置に対する4つのバイナリ特徴 (白駒の数)
  - バックギャモンについての深い知識はない
- 結果
  - TD-Gammon 0.0: 自分自身との対戦での学習 (300,000 ゲーム)
    - 過去の最良のBG プログラムと同等の能力」 (Tesauro自身のも含めて)
    - 大量の専門家の入力や工夫した特徴量
  - TD-Gammon 1.0: 特別な特徴を付加
  - TD-Gammon 2 and 3 (2-ply and 3-ply search)
    - 1.5M ゲーム, 人間のチャンピョンとの対戦

G. Tesauro (1994). "TD-Gammon, A Self-Teaching Backgammon Program Achieves Master-level Play" in Neural Computation

TD-Gammon

- Q学習　＋　ニューラルネット（バックプロパゲーション）
- ランダムなネットから始める
- 自分自身との対戦による学習（1.5　百万ゲーム）
- 世界最高の人間プレイヤーと同レベル

TD-Gammonの性能



- Expert labeled examples are scarce, expensive & possibly wrong
- Self-play is cheap & teaches the real solution
- Hand-crafted features help