人工知能A

Topic 5: 強化学習

Topic 5: Reinforcement learning

5

強化学習

- 講義の内容
  - 動的計画法
  - 確率の復習
  - 強化学習とは
  - TD法
  - Q学習
- 目標：
  - 強化学習の概念と基本的アルゴリズムであるTD法とQ学習について学ぶ

3

動的計画法の復習

多分、アルゴリズムとデータ構造の復習

問題分割(1)

- ある問題sとして、部分問題$s_1, \ldots, s_m$を解いて最適な組み合わせを求める問題を考える。
  - 組み合わせの評価値を表す関数をJとおくと、
    $J(s_1, \ldots, s_m)$を最大化する組み合わせを求めることに相当する。
  - s_iの解の数の平均をNとすると、N^mの組合わせがある。
- もし$s_1, \ldots, s_m$を順に解き、それまでの部分問題$s_1, \ldots, s_{i-1}$の結果を利用して次の部分問題s_iを解くことができれば、計算量は縮小する。
  - s_iを部分問題s_iを解いたあとの状態と同一視する。
  - これは、hを2状態の評価値として
    $J(s_1, \ldots, s_m) = \sum_{1 \le i \le m-1} h(s_i, s_{i+1})$
    とできる特別な場合に相当。隣接する2つの組み合わせでN^2、これが$m-1$組なので、$N^2 m$となり、組み合わせの数は激減する。

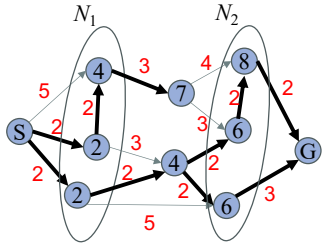
問題分割(2)

- $s_1, \ldots, s_m$を単位時間ごとに順々に解くことと考えたとき、多段決定問題 (multi-stage decision problem) という。
  - これは、s_iを状態と考えれば、時系列で遷移する状態遷移
    $s_1 \rightarrow s_2 \rightarrow \ldots \rightarrow s_m$
    と表せる
  - 多段決定問題では、初めに部分問題があるのではなく、解状態に達するための遷移を探索しながら到達する問題とも考えられる
- 注意：
  - hをコストとすれば、$J(s_1, \ldots, s_m)$を最小化する問題と考えてもよい
  - さらに、評価値とコストの組み合わせもある。このときには、コストを負の数とし、合わせて最大化する問題と考えられる。
  - 評価値の代わりに、途中あるいは最後の成功の結果得られる報酬を利用することもある。（この場合、各時点で得られる累積報酬が大きい方が評価値が高いと考える）。

動的計画法 (dynamic programming)

- 動的計画法とは、前々ページのJの性質を満たすときに、問題を部分問題に分け、それを順に解く。その際に、それまで解いた部分問題の結果を利用して次の部分問題を解く手法。
  - 例として最短経路問題：たとえばスタートSからゴールGに至る最短経路の途中のノードをnとする。
    $S \rightarrow n \rightarrow G$
    このとき部分経路$S \rightarrow n$, $n \rightarrow G$も最短経路である。
    この場合、hはノード間の最短距離（つまりコスト，c）
  - ダイクストラ法、ベルマン・フォード法による最短経路計算
  - ナップザック問題、編集距離（アルゴリズムとデータ構造A/B)。
  - その他、適用例多数。

動的計画法 (dynamic programming)



SからGへの経路はかならずN_1とN_2のどれかのノードを通る。
s_1：SからN_1の（すべての）ノードへの最短経路、
s_2：N_1のノードからN_2の最短経路、
s_3：N_2からGへの最短経路
と部分問題に分ける。

- s_1を求め、その結果に基づいてs_2を求め、その結果に基づいてs_3を求める。
- 以下で述べる強化学習では、$s_1, \ldots, s_m, \ldots$を時間ごとの行為の結果と考え、ある変数の値を、時刻tの結果と評価値を用いて、t+1の行為を行い、その結果と評価を使って次の結果を得る．．．これを繰り返して実際の値に近づける。

動的計画法のアルゴリズム

- 以降、s_iを$s_1, \ldots, s_i$を部分問題を解いた後の状態と考え、初期状態s_0を加える。つまり、$s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \ldots \rightarrow s_m$と書ける。
  - $\tilde{S}_0 = \{s_0\}$ (ただしスタートが複数状態のこともある）、$\tilde{S}_{i-1}$の要素から到達できるs_iの状態の集合を$\tilde{S}_i$と表す。

for (i = 1 to m) {
  foreach ($s_i \in \tilde{S}_i$) { // このforeachの部分がN^2
    $F(s_i) = \max_{s_{i-1} \in \tilde{S}_{i-1}} [F_{i-1}(s_{i-1}) + h(s_{i-1}, s_i)]$
    この最大値を与えるs_{i-1}を$\hat{s}_{i-1}(s_i)$とおく
  } } // end foreach and for

$F(s_m)$を最大とするs_mをs_m^*とおき、$J^* = F(s_m^*)$とする。
for ($i = m-1$ to 0) { // backward calculation
  $s_i = \hat{s}_i(s_{i+1}^*)$
}
return $(s_0^*, s_1^*, s_2^*, \ldots, s_m^*)$ and J^*

動的計画法のアルゴリズム（参考）

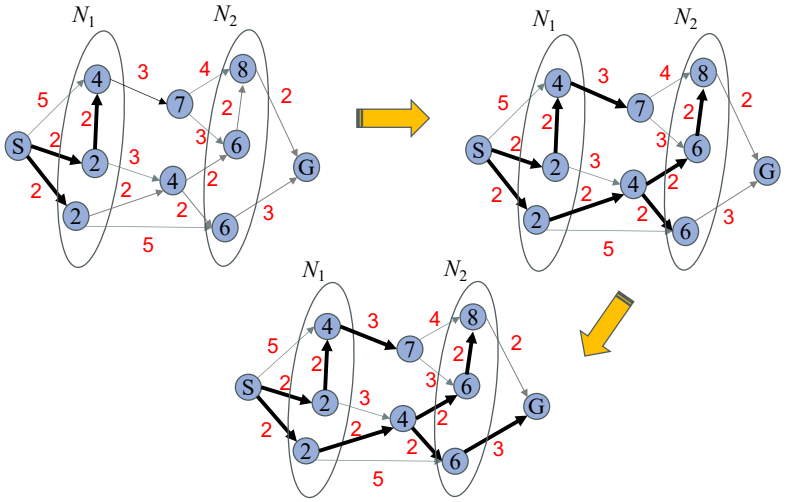
$$J^* = F(s_m^*) = \max_{s_m \in \tilde{S}_m} F_m(s_m)$$
$$= \max_{s_m \in \tilde{S}_m} [\max_{s_{m-1} \in \tilde{S}_{m-1}} [F_{m-1}(s_{m-1}) + h(s_{m-1}, s_m)]]$$
$$= \max_{s_m \in \tilde{S}_m} [\max_{s_{m-1} \in \tilde{S}_{m-1}} [\max_{s_{m-2} \in \tilde{S}_{m-2}} [F_{m-2}(s_{m-2}) + h(s_{m-2}, s_{m-1})] + h(s_{m-1}, s_m)]]$$

……

$$= \max_{(s_0, s_1, \ldots, s_m) \in \tilde{S}_0 \times \cdots \times \tilde{S}_m} \sum_{i=1}^{m} h(s_{i-1}, s_i)$$

これは「スライド：問題解決(1)」の式と同じ（つまりこの形式で書ける場合には、$s_0, s_1, \ldots$と順に状態を求め、逆向きに最適なものを選択すればよいことを示している）。

動的計画法 (dynamic programming)



事前課題5-1（済）

- 以下のアルゴリズムについて調査・確認し、その内容を確かなものにすること。
  - ダイクストラ法、ベルマン・フォード法による最短経路計算
  - ナップザック問題
  - 編集距離の定義と動的計画法による求め方

12 確率の復習

基礎なので省略の可能性もあり

確率の復習(1)

- 有限な標本空間 をΩ。その部分集合Eを事象 (event)と言う。
- 確率Pの定義（離散を想定）：

(1) 任意のイベントEに対し、$0 \leq P(E) \leq 1$

(2) $P(\Omega)=1$

(3) $E_1, E_2, \ldots \subset \Omega$が排反事象 (集合としてdisjoint, $E_i \cap E_j=\phi$)のとき、
$P(E_1 \cup E_2 \cup \ldots) = \sum_i P(E_i)$

をみたすとき、PはΩ上の確率とよぶ。

- 条件付き確率
  - 事象A, Bが共に起こる確率を「同時確率」といい、$P(A, B)$ と表す。
    これは、$P(A \cap B)$であり、これから $P(A, B) = P(B, A)$
  - Bが起った上でAが起こる確率を「条件付き確率」といい、
    $P(A|B) = P(A, B)/P(B)$
    と定義する。

確率の復習(2)

- $E_1 \cup \cdots \cup E_n = \Omega$かつ排反とする。このとき、
  $P(A) = \sum_{i=1}^{n} P(A, E_i)$
  である。これを加法定理とよぶ。
- 期待値$E[f(A)]$ の定義：事象A とΩ上の関数
  $f: \Omega \longrightarrow \boldsymbol{R}$ (実数の集合)
  に対し、
  $E[f(A)] = \sum_{x \in A} f(x)P(x)$
  と定義する。
  $f(x) = x$のときAの期待値といい、$E(A)$と表す。
- 条件付き確率の期待値を
  $E(A|B) = \sum_{x \in A} xP(x|B)$
  と定義する。

確率の復習(3)

- $P(A, B) = P(B, A)$ より

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)}$$

- もし、$\Omega = \bigcup_{n=1}^{N} H_n$ かつdisjointであるとき、B=H_iとおいて

$$P(H_i|A) = \frac{P(A|H_i)P(H_i)}{\sum_{n=1}^{N} P(H_n)P(A|H_n)}$$

- これらを（両方とも）ベイズの定理という。

16 マルコフ決定過程と強化学習の基本事項

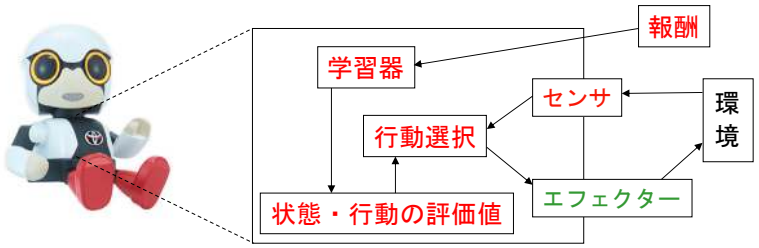
強化学習とは

- たとえば未知の世界（火星など）にエージェントが行き、行動するとする。はじめは、地図はないので、障害物に当たったり、丘や溝などに手こずるかも知れない。しかし、試行錯誤をして、それぞれの位置でどう動くべきか（動いてはいけないか）を学習し、体験を通じて新しい知識を獲得し、徐々に賢く動作できるようにしたい。
- チェスなどでは、それぞれの場面での手が良いか悪いかは簡単に決められない。ただ最終的に勝った（負けた）という何らかのフィードバックを受けて、それまでの手は良かったかどうかを判断する。
- 教師有り学習のように正負の例は与えられない。ただ、何らかの良し悪しを認知して、選択した行為を評価する。

強化学習とは

- 強化学習 (reinforcement learning)とは、エージェントが自ら行動し、その結果に応じて報酬 (reward) を得て (認知して)、最適な行為 (行動) を獲得 (強化) する学習である。
  - エージェントには予め知識（どう行動すべきかという知識）は与えられていない。ただし環境 (地図に相当するモデル) が与えられることもある。また現在の状態 (環境の一部か全体) を認知できる。
  - 報酬は、行為の直後に得られる場合もあれば、チェスのように最後の勝利（成功）の結果として得られる場合もある。エージェントは報酬を認知できる。
  - 以上の状況で、(a) 環境の状態の価値、もしくは (b) 各状態でどの行為が価値があるかを学習する。
  - ここで価値とは、将来を含めて得られるであろう報酬の累積のこと。
  - エージェントは、より価値の高い状態を求めて移動を試みる。

強化学習を行うエージェント



- エージェントは自律的に行動する。環境は周囲の状況 (situation)。
- エージェントは、センサにより環境から得た情報のみで状況を判断。取得情報の制限で状況を完全には区別できないかもしれない。
- 行動ごと、あるいは目的を達成すると報酬（reward）を受け、それにより行動が強化される（たくさん報酬を受け取りたい）。

マルコフ決定過程

- 定義：環境のダイナミクスを以下のようにモデル化したものをマルコフ決定過程（Markov decision process, MDP）という。
  - $S = \{s_1, s_2, \ldots, s_n\}$ 環境の状態の集合
    $A = \{a_1, a_2, \ldots, a_l\}$エージェントの行為 (actions) の集合
  - 時刻tで環境中のある状態 $s(t) \in S$ にあり、エージェントがある行為 $a(t) \in A$を実行すると、環境は確率的に状態 $s(t+1) \in S$ へ遷移する。その状態遷移確率（条件付き確率） $P(s(t+1) \mid s(t), a(t))$ を
    $Pr(s(t),a(t), s(t+1))$ と表す。
  - 同時に得られる報酬を$r(s(t), a(t))$とする。
- マルコフ性：
  - 状態$s(t+1)$ への遷移と報酬$r(s(t), a(t))$ が、そのときの状態$s(t)$と行為$a(t)$にのみ依存し、それ以前の状態や行為とは無関係なこと。

マルコフ決定過程

- 注意事項
  - マルコフ決定過程の状態遷移確率は、
    $Pr: S \times A \times S \rightarrow [0,1]$
    なる関数とも考えられる。
  - 確率であるから、以下の式が成り立つ。
    $\sum_{s \in S} P(s \mid s(t), a(t)) = 1$
    常に$P(s(t+1) \mid s(t), a(t)) = 1$ or 0, となるとき、マルコフ決定過程は決定的 (deterministic)であると言う。状態遷移が一意に決まることである。
  - このときは、以下のように記述してもよい。
    $Pr: S \times A \rightarrow S$

強化学習の基本設計

- エージェントが「状態」または「行為」の価値 (value) を学習する。ここで価値とは、将来を含め最終的に得られるトータルな報酬。これをを最大化するような行為の選択にこれを利用する。
  - より価値の高い状態に移動するために行為を選択する（強化学習）。
    - 状態sの価値を表す関数を価値関数といい、$V(s)$ と表す。
  - 直接、ある状態（現在の状態）における行為の価値を学習することも考えられる（Q学習, Q-learning）。
    - 状態sで、行為aの価値を関数$Q(s,a)$ と表す。
    - 状態遷移は確率的なので、正確には、大きな累積報酬が得られる確率が高いということ。これは期待値が大きいものとも言える。

強化学習の基本設計

- 環境の状態から行為を選択する関数
  $\pi: S \rightarrow A$
  を政策関数 (*policy*, 方策関数とも) と呼ぶ。強化学習は、高い累積報酬を目指した政策関数を学習すること。
- 注意：
  - 報酬は「教師」と異なり、(不)正解が明確ではない。もっと大きな報酬を得られるかもしれないし、もうないかもしれない。何度も周辺を調べて、一番良さそうだと分かる（が、将来、覆ることも）。
  - 教師有り学習は高校までの授業のように、課題が与えられてその正解を求めるタイプの学習（解答例がある）。正解との一致が重要。
  - 強化学習は正解の判断はつかず、より良いものを目指すという研究タイプ（卒論・修論など）の学習。

27 導入

導入例題

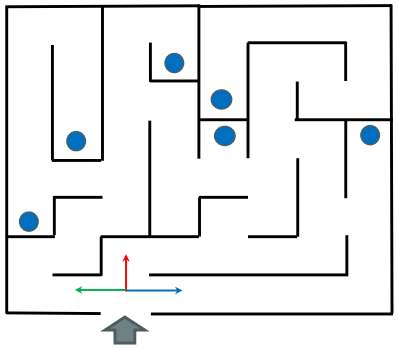
- N本腕バンディット問題（$N=10$とする）
  - N台のスロットマシン（多腕バンディット問題とも）
  - それぞれのマシンには報酬（コイン）の期待値が定義されていて、報酬の平均がその値に近づくような報酬がもらえる。
  - 各マシンには、期待値0、分散1の正規分布に従って確率的に選択された値がその期待値 として設定される。
    - つまり、期待値の高い（低い）マシンが混在する。

導入例題

- エージェントは、10のスロットマシンから期待値の大きなものを選択し、プレイをしたい。如何に選ぶか。
  - 非常に単純な問題：
    - 環境の状態は能動的（自分の行動のみが状況を変える）。
      - 腕を選択した（あるいは引いた）ときだけ状況が変化。
    - 報酬は最後に得られる（といっても、状態遷移は２段だけ）。
      - すこし工夫して、迷路を通って、それぞれのスロットマシンに近づくとすると、経路の選択時には報酬はないが、最後になって報酬が得られる問題となる（次ページ）。
    - 状態の価値を考えるなら、n番目の腕を選んだ状態をs_nとして、その（選んだ状態の）価値$V(s_n)$を求める。
    - 行為の価値を考えるなら、初期状態s_0において、n番目の腕を選ぶ行為をa_nとして、その状態における行為の価値$Q(s_0, a_n)$を求める。

スロットマシンのある迷路

やや複雑な例



赤、青、緑：どの方向に進む

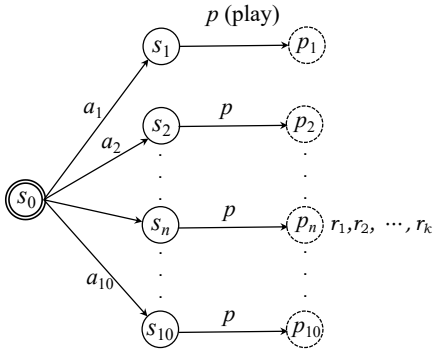
初めに間違った方向に進むと、「良い」スロットマシンに到達できなくなる。

● スロットマシン

赤、青、緑：どの方向に進む

導入例題（MDPとしての表現）

- $Q(s_0, a_n)$ や$V(s_n)$ を推定
  - 試行を何度か行い、t回目までにa_nを選択した回数をkとし、それぞれ得られた報酬をr_i $(1 \leq i \leq k)$とすると$V(s_n)$ の価値は、
    $$\frac{r_1+r_2+ \cdots +r_k}{k}$$
    と推定。この推定方法を標本平均法という。
  - 状態 s_nで行為はpだけ（一本道）で、途中の報酬は無い。$V(s_n)$も $Q(s_0, a_n)$も同じ価値となる。同様に他の状態や行為についても$(1 \leq n \leq 10)$。



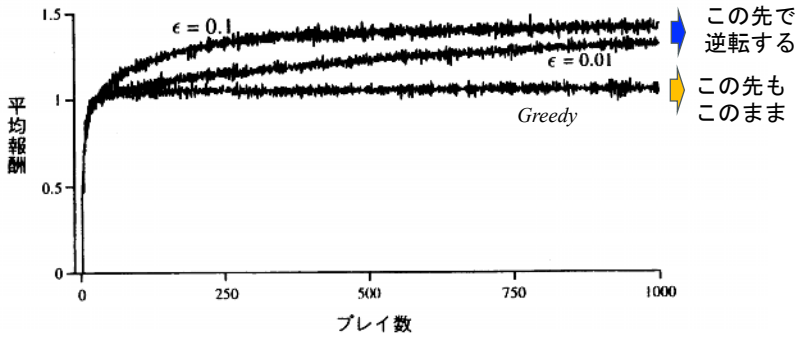
行動選択の戦略とジレンマ

- 導入問題の考察のポイント
  - ジレンマと学習能力の関係が存在する
- t 回学習した。$t+1$回目のs_0でがどれを選ぶ?
  - 価値を学習し、それを反映しながら、次の行為a_nを選択する戦略を「行動選択の問題」という。
  - それまでに学習したことを活用して（信じて）選ぶか、あるいはさらに学習の範囲を広げたり、正確さを向上させるために他の選択肢を試すかというトレードオフ。これを知識の活用か探査かのジレンマ (exploration-exploitation dilemma)という。
    - たとえば、今は、$V(a_1)$ (つまり$Q(s_0, a_1)$) が一番大きいかもしれないが、これは偶然かもしれない。他にもっと良いものがあるかも。
  - ジレンマになるのは、学習の効率（速度）と獲得報酬の間。
  - まずは、ジレンマの説明の前に、行動選択の戦略を２つ紹介。

行動選択の戦略

- Greedy 戦略 (知っている範囲で最善のもののみを選択するという意味。新しい試みはしない）。
  - すでに得られた$Q(s_0, a_j)$が最大となる行動 a_jを実行する（あるいは、$V(s_j)$が最大とな状態s_jに移動するように行動を選択する）。
- ε-greedy 戦略
  - 通常はgreedyであるが、ある確率εでランダムに他の行動aを選択する。通常は、εは0に近い値。
- Exploration-exploitation dilemmaの確認のために：
  - Greedyやε-greedyの行動選択の戦略によって得られる効率化の特徴を調べる。
  - このために、10本腕バンディット問題について価値を学習し、その結果、平均の報酬が如何に変化するかを調べてみる。

導入例題の実験結果



導入問題の考察

- Greedy戦略：
  - 期待値が最大でも必ず揃う訳ではないので−1の報酬だったとする。
  - それ以外 (初期値は0) を適当に選択し、たまたま報酬を得るとそれを選択し続ける。もしそれが適度に良いもの (期待値が0以上)であれば、統計的に平均値が0になる前に再びそろう可能性があるので、この手を取り続ける。
  - 最善の手でなくても他の試みをしないので、より良い手は見つからない。(適当なところで学習しなくなる)
- ε-greedy戦略：
  - 上記のように次善の手を一時的に採用しても、εの確率でランダムにとるので、いつかは (εが大きいほど早く) 最善の手を見つける。
  - ただし、それが最善との判定はできないので継続的にεの確率で他の手を取り続ける。これは無駄ではあるが、環境が変わったときには、新しい状況に対応はできる。

Exploration-exploitation dilemma

- $\varepsilon = 0.1$の場合は早めに最善の手を見つけるが、その後もこの確率で最善の手を採用せず、報酬は頭打ちとなる。
- $\varepsilon = 0.01$の場合は、最善の手を見つけるのは遅い。見つけると、最善の手を取らない確率は前者より小さく、得られる報酬は大きくなる。
  - 学習速度とその後の効率のトレードオフをexploration-exploitation dilemmaという。
  - 対応策としては、初めはεを大きめに、徐々に小さくすることが考えられる。
    - ソフトマックス (softmax) 法というのもある（あとで）
    - ただし、εを小さくしすぎると、環境に変化があっても、前の学習結果を更新せずに使い続けるという別のジレンマもある。

強化学習 (2)

39

時間的差分学習法

Temporal difference learning method

強化学習の問題設定の分類 (1)

- エージェントと環境の関係1 — 環境はアクセス可能か？
  - エージェントが環境の必要な部分を完全に観測できる（アクセス可能）なら、各状態を知覚・記憶し、その変化をモデル化すれば良い。
  - 観測可能性、可観測性とも言われる。
  - （一部は観測できない）部分的な観測なら、それから実際の状態を推測する必要がある（⇒ 部分観測可能MDPという。これは大学院レベル）。
  - 観測できなければ環境のモデルを予め与え、「現在はこの状態のはず」と判断させる。
  - 環境の状態が分からなければ、強化学習は成り立たない。

強化学習の問題設定の分類 (1)

- エージェントと環境の関係2 — 環境と行為のモデル
  - 行為は環境に影響を与え、その状態を変えられるか？
    - 与えられるなら能動的、与えられなければ受動的という。
    - 受動的学習者なら、環境の変化を観測し、各状態の価値を学習する（ある確率で環境が勝手に変わる）。自分の状況を評価する。
    - 能動的学習者は、価値の高い状態となるように自ら行動する。
  - 能動的なら行為が与える環境への影響［状態遷移］の知識がどこかで必要。
    - これを事前知識（プログラム・データ）として与えるか、
    - 環境への影響もモデルの一部として学習する必要がある。
    - MDPでは、状態遷移は確率的に表現されている。
    - なお、行為が環境に影響を与えなければ、行為は状態に影響しないのでQ学習はない

強化学習の問題設定の分類 (2)

- エージェントと環境の関係3 — 報酬のタイミングと意味
  - 報酬は最終状態のみか、途中や任意の状態で得られるか
    - 通常は、報酬が全ての状態で得られるわけではない。たとえばゲームなら勝敗が決まった最後だけかもしれない。
    - 宝探しなら見つけた度に報酬が得られるかもしれない。
  - 報酬の意味
    - 報酬は絶対的尺度（勝ち負けのように）か？　または
    - 途中の行為の評価（たとえば好手・悪手など。最終的に勝負を決定するわけではないが、効用のヒントにはなりうる）か？
    - ただし途中の報酬は、エージェントを混乱させる可能性もある。
      - 最後の絶対的結果を無視してしまう。
    - 負の報酬（コストや罰など）はあるか？
      - これにも罰があるなら動かないと学習することもある。

強化学習の特徴

- 試行錯誤的探索
  - 教師なし学習（風）である。予め正しい（正の例）、正しくない（負の例）は与えられない（代わりに報酬が得られる）。
  - 実際に学習しながら行為を実行し、行為の価値を学習する。知識の活用(exploit)か探査(explore)かの問題がある。（後で再び）
- 遅延報酬
  - 行為（の積み重ね）の結果として報酬を得るが、それは遅れて得られることが多い。たとえば、チェスで勝つのは最後に分かることで、途中の手が良いかどうかは、直ぐには分からないかも（教師あり学習では、良い・悪いを与えられる）。
  - 各行為ごとに報酬があることもある。ただし、エージェントは直後の報酬に惑わされること無く、その後の将来の報酬の総和（期待報酬）を大きくなるよう学習する。この場合でも、最終的な真の価値（総報酬）は遅れて分かる。

既知環境における状態の受動的学習

- 準備：
  - 既知、受動的環境 →　S, Prはすべて分かっている。最も簡単なケース。
  - 受動的なので行動は考えない。状態を観測するのみ。
    つまり、$A=\emptyset$もしくは、$A=\{Observe\}$とする。
  - $S=\{s_1, s_2, ..., s_n\}$とおいて、状態遷移行列Mを以下のように定義

$$M = \begin{pmatrix} p_{1,1} & \cdots & p_{1,n} \\ \vdots & \ddots & \vdots \\ p_{n,1} & \cdots & p_{n,n} \end{pmatrix}$$

    ここで$p_{i,j}$は状態s_iからs_jに遷移する確率とする。
- 目的：状態s_iの価値$V(s_i)$を求めたい。

既知環境における受動的学習

- 基本アルゴリズム (eは環境、state(e)は環境の状態)

Function PassiveRLAgent(e) **returns** an action
static: V, a table of estimated value // value function (utility function と呼ぶこともある)
　　N, a table of frequency for states // 何回ある状態に訪れたか
　　M, a table of transition probabilities // 状態遷移確率の表
　　percepts, a percept sequence // initially empty, 状態推移の履歴（リスト）
Add e to *percepts* // 状態を記憶
increment N[State[e]] // この状態となった回数を記録
　$V \leftarrow$ Update(V, e, *percepts*, M, N)　// eの価値を確認（再計算）
if Terminal?[e] **then** *percepts* ← the empty sequence // 最後まで来たら、次の学習の準備
return the action Observe // ここでは受動的環境なので行動は「観測」

  - 既知環境なので、状態空間Sと遷移確率Mが分かっている。
- 価値関数の学習なので、Updateの内容が本質。

既知環境における受動的学習

- Updateの主なアルゴリズム
  - 最小二乗法 (Least mean square method, LMS法)
    - 計算量は小さいが、学習の収束が遅く、多くの試行錯誤が必要。
  - 適応動的計画法 (Adaptive dynamic programming, ADP法)
    - 学習の収束は速いが、計算量が大きい。状態数が大きくなると処理時間が問題となる。
  - 時間的差分学習法 (temporal difference learning, TD学習法)
    - 動的計画法の一つ。学習の収束も比較的速く、計算量も小さい。広く使われている学習法。
  - その他、上記の改良アルゴリズムなど多数ある。
  - 本講義では、よく使われるTD学習法のみ説明する。

時間的差分学習のアルゴリズム

- 再考：ある状態sの価値$V(s)$とは何か？
  - その状態s以降に状態が受動的に推移し、最終状態になるまでに得られる報酬の総和の期待値（累積報酬期待値）のこと。
    - 確率的推移なので状態の推移も毎回変わるので、期待値を使う。
- 考え方：
  - ある状態sからs'に遷移して、そのとき報酬rを受けたとする。
  - 未来も含めた累積なので、 $V(s)$ の値はr と $V(s')$ の和と考えられる。
  - これまでの$V(s)$の値を少しだけ「rと$V(s')$ の和」に近づける。
    - 「＝」とせず「近づける」とするのは、「rと$V(s')$ の和」も変化する可能性があるから。 近づける割合を学習率といいαで表す $(0 < \alpha \leq 1)$。
    - 同様に$V(s)$ の更新はs の遷移前の状態の価値にも影響する。したがって、backwardに計算が進むイメージだが、動的計画法の考えで一時刻前の結果を利用して求める。

時間的差分学習のアルゴリズム

- TD学習法のアルゴリズム

Function TDUpdate(*V, e, percepts, M, N*) **returns** the value table *V*

if *percepts* contains more than one element **then** // *e*は最後の要素

e' ← the penultimate elements of *percepts* // 後ろから２番目の要素

i ← State[*e'*], *j* ← State[*e*];

V[*i*] ← *V*[*i*] + α(*N*[*i*])(Reward[*e'*] + *V*[*j*] – *V*[*i*])

if Terminal?[*e*] **then**

V[state[*e*]] ← (1−α(*N*[*i*]))*V*[state[*e*]] + α(*N*[*i*])Reward[*e*]
// 上の式で*V*[*j*]=0とした

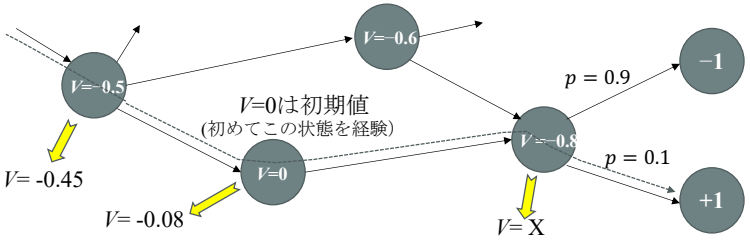
end

時間的差分学習のアルゴリズム

- TD学習法のアルゴリズムのコメント
  - $percept$は認知した環境の列。最終状態eについては平均値を計算。
  - 学習率$0 < \alpha(N[i]) \leq 1$は定数とすることが多いが、ここでは訪問回数に応じて減少する関数も想定できる。
  - e'は一つ前の状態であるが、ここを中心（現在）と考えると、現在価値$V(\text{state}[e'])$は、 赤の$V(\text{state}[e'])$ と青の
    “次の状態eの価値$V(\text{state}[e])$” ＋ “そのときの報酬”
    との差分をとり、差が小さくなるように$\alpha(N[i])$分だけ修正。
  - なお、 $V(\text{state}[e])$はまだ更新されておらず、次のタイミングで変化する。（その意味では一つ前の試行で求めた値を使っている）。
  - eが最後の状態なら、その次は無いので次の状態の価値は0と考える。

TD学習法の計算例

- 点線のように状態変化し、最後の状態で報酬 +1を得たとする。α = 0.1とする



(1) V= −0.5 + 0.1 (0-(-0.5))
= − 0.45

(2) V= 0 + 0.1× (− 0.8 − 0)
= − 0.08

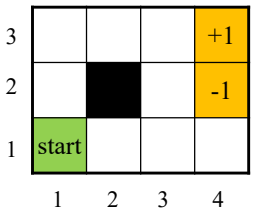
(3) 最後の状態では常に
+1を受け取るので、
その価値は1 (*V*=1)

課題５−２：上記のXの値を求めてみよ。

既知環境での受動的学習の例題

- 例題として以下を考える。

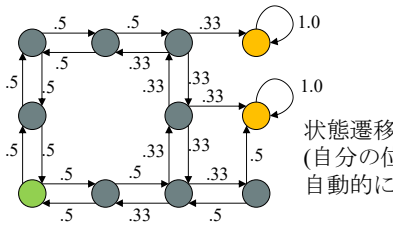
４ｘ３マスパズル



状態

正しい効用値
（この値を求めたい）

-0.0380	0.0886	0.2152	+1
-0.1646		-0.4430	-1
-0.2911	-0.4177	-0.5443	-0.7722

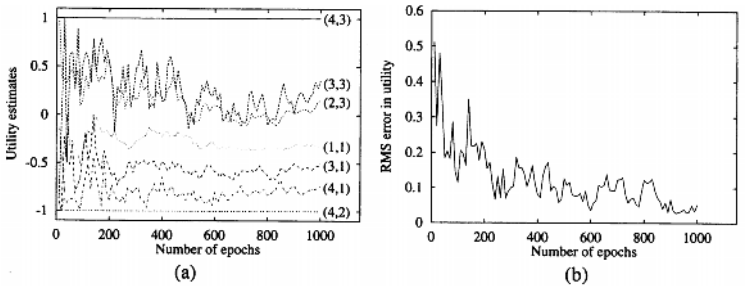


状態遷移確率
(自分の位置が
自動的に変わる)

(1, 1)から状態遷移確率に基づいて自動的に移動し、最終状態 (オレンジ) になれば報酬をもらい終了(これを 1 epochとよぶ)。これを繰り返し、各状態の価値を求める。

TD学習法の収束

- ４ｘ３マスパズルでの実験結果 (RMS = root means square)



- 時間と共に収束する(RMSは0に近づく）ことが分かる。

未知環境における受動的学習

- 既知環境の例題を見ると分かるように、環境が完全に分かっていれば、状態の計算は、行動を繰り返さなくても計算はできる（たとえば動的計画法を使う）
- 未知環境：状態空間*S*と遷移行列*M*確率は未知。ただし、
  - TD学習法では*M*は使っていない。
  - 状態*S*は、実際に行動し、認知した状態を記憶すればよい。これは*N*が対応し、*N*のサイズが徐々に増える。
  - PassiveRLAgent(*e*) で、初めて*e*を観測したときに、State[*e*]を*V*, *N*, *M*に追加する。
  - TD学習法はそのままアルゴリズムを適用できる。
  - *M*も（必要であれば）実際の観測から推測すればよい（状態遷移を数え上げて、その確率を求める。次ページ）。

未知環境における受動的学習

- Mを実際の観測系列から推測する。PassiveRLAgent(e)を以下のように拡張する。
  - たとえば、環境eからe'へ変わったとする。
  - 状態 state[e'] が初めてであれば登録。つまり、新しい状態をNとMに追加し、拡大させる。それ以外は、環境の出現数を増やす。（NはこれまでSの意味も兼ねている）
  - そのMにおいて、これが初回であればstate[e]→state[e']の遷移確率を1とする。別のepochで、状態eからe''への推移を観測したら、必要に応じてstate[e'']を登録し、state[e]→state[e']の遷移確率を0.5、state[e]→state[e'']の遷移確率も0.5とする。これを繰り返す。
    - 1 epochとは初期状態から最終状態へ至る一試行のこと
  - このように観測から得られたMを使う。初めは正しくないが、実際の確率で状態遷移するので、徐々に真の値に近づくはず。

未知環境における受動的学習

- 基本アルゴリズム (eは環境、state(e)は環境の状態)

Function UnknownPassiveRLAgent(e) **returns** an action
static: V, a table of estimated value // value function (utility function と呼ぶこともある)
N, a table of frequency for states // 何回ある状態に訪れたか
M, a table of transition probabilities // 状態遷移確率の表
percepts, a percept sequence // initially empty, 状態推移の履歴（リスト）
Add e to *percepts* // 状態を記憶
if state[e] is new **then** V[state[e]]=0, N[state[e]]=1 // Entryを増やし、初期化。
<この部分で前スライドのMの更新を行う>
increment N[State[e]] // この状態となった回数を記録
$V \leftarrow$ Update(V, e, *percepts*, M, N) // eの価値を確認（再計算）
if Terminal?[e] **then** *percepts* ← the empty sequence // 最後まで来たら、次の学習の準備
return the action Observe // ここでは受動的環境なので行動は「観測」

- 状態空間Sと遷移確率Mが不明。上記のV, N, Mの項目を動的に増やす。

未知環境における能動的学習

- 能動的環境でエージェントは行為を選択し、状態を変える。最終的な累積価値を最大にする行為（の列）を選ぶ。
- 行為aが状態を変えるので、$a \in A$ごとに状態遷移行列M^aがあるはず。前出の方法で行為aごとにM^aを学習する。
  - ある状態s_iで行為aを行うとある確率$p_{i,j}^a$で状態s_jに移動する。

$$M^a = \begin{pmatrix} p_{1,1}^a & \cdots & p_{1,n}^a \\ \vdots & \ddots & \vdots \\ p_{n,1}^a & \cdots & p_{n,n}^a \end{pmatrix}$$

  - ここで$|N| = n$とする。Nのサイズが増えることに注意。
  - あとは、モデルであるM^aを行為とその結果の観測から学習する（アクセス可能性を仮定している点に注意）

未知環境における能動的学習

- なお、TD学習法では、N (つまりSの推定モデルと訪問回数) の更新は行うが、M^aは直接は使わない。しかし、能動的学習で必要な、「次に実施すべき行為」の選択（行動戦略）に使う。

未知環境における行動選択

- 能動性：行為（行動）の選択
  - 能動的環境での強化学習では、行動しながら学習する。より価値の高い状態に移動したい。状態遷移確率と次の状態の価値から報酬期待値の高い状態に遷移する行為を選択する。
  - 現在s_iにいて、可能な行為は$a_1 \cdots a_m$とする。行為の結果は確率的に遷移するので、期待値をもとめる。それは、
    $$r_{a_j} + \sum_{k=1}^{n} p_{i,k}^{a_j} V(s_k)$$
    が最も大きい行為a_j(これをa^*とする) を「主に」選択する。これを、
    $$a^* = \arg \max_{a \in \{a_1,\ldots,a_m\}} [r_a + \sum_{k=1}^{n} p_{i,k}^{a} V(s_k)]$$
    と書く。なお、r_aはs_iで、行為aを実行したときの報酬とする。
  - ただし常に、上記の式を最大化する行為だけを選ぶと、新しいチャレンジが無く、学習は頭打ちとなる（→ジレンマ）

未知環境における能動的学習

- 基本アルゴリズム (*e*は環境、state(*e*)は環境の状態)

Function ActiveRLAgent(*e*) **returns** an action
static: *V*, a table of estimated value // value function (utility function と呼ぶこともある)
N, a table of frequency for states // 何回ある状態に訪れたか
M, a table of transition probabilities // 状態遷移確率の表
percepts, a percept sequence // initially empty, 状態推移の履歴（リスト）
Add *e* to *percepts* // 状態を記憶
if state[*e*] is new **then** *V*[state[*e*]]=0, *N*[state[*e*]]=1 // Entryを増やし、初期化。
<この部分で前スライドの$M=\{M^a\}$の更新を行う>
increment *N*[State[*e*]] // この状態となった回数を記録
V ← Update(*V, e, percepts, M, N*) // eの価値を確認（再計算）
if Terminal?[*e*] **then** *percepts* ← the empty sequence // 最後まで来たら、次の学習の準備
action ← $\arg \max_{a \in \{a_1,\ldots,a_m\}} [r_a + \sum_{k=1}^{n} p_{j,k}^{a} V(s_k)]$ <前ページの期待報酬値の計算>
return *action* // ここでは今のところの最適な行動を返す（実際には政策で決定）

時間的差分学習

- TD学習法のアルゴリズム
- // 基本的にこれまでと違いは無い

Function TDUpdate(*V, e, percepts, M, N*) **returns** the value table *V*
e', the previous state
if *percepts* contains more than one element **then** // *e*は最後の要素
e' ← the penultimate elements of *percepts* // 後ろから２番目の要素
i ← State[*e'*], *j* ← State[*e*];
V[*i*] ← *V*[*i*] + α(*N*[*i*])(Reward[*e'*] + *V*[*j*] − *V*[*i*]) // *V*[*null*]=0
if Terminal?[*e*] **then** …/* 前と同じ */
return *V;*

- なおActiveRLAgent(*e*) で、*V*の項目の追加と*M, N*が計算される。

割引報酬

- 強化学習では、報酬の合計 (累積報酬) の代わりに、割引報酬合計を価値として用いる。無限系列のため。
  - ある時刻の状態$s(t)$で、以降に行為の列$a(t), a(t+1)$ ….を実行したとき、 $s(t)$の割引報酬$V(s(t))$ を以下のように定義する。
    $$V(s(t)) = r(s(t), a(t)) + \gamma \cdot r(s(t+1), a(t+1)) + \gamma^2 \cdot r(s(t+2), a(t+2)) + \cdots$$
    $$= \sum_{n=0} \gamma^n \cdot r(s(t+n), a(t+n))$$
    ここでγ $(0 \le \gamma < 1)$ を割引率 (discount factor) という。
- 割引率の意味：
  - 割引報酬は、何時受け取れるかにより、将来得られる報酬を割り引いて考えることである。
  - 短い行為の列で報酬が得られる方が良い。
  - 割引率を導入する数学的理由は、行為の列にが無限に続く場合に、学習の収束性を確保するためでもある。

時間的差分学習（割引報酬込み）

- TD学習法のアルゴリズム (通常は割引率γを導入したものを使用)

  Function TDUpdate(*V, e, percepts, M, N*) **returns** *action* // 違いは赤の部分のみ

  e', the previous state

  if *percepts* contains more than one element **then** // *e*は最後の要素

  e' ← the penultimate elements of *percepts* // 後ろから２番目の要素

  i ← State[*e'*], *j* ← State[*e*];

  $V[i] \leftarrow V[i] + \alpha(N[i])(\text{Reward}[e'] + \gamma V[j] - V[i])$ // $V[null]=0$

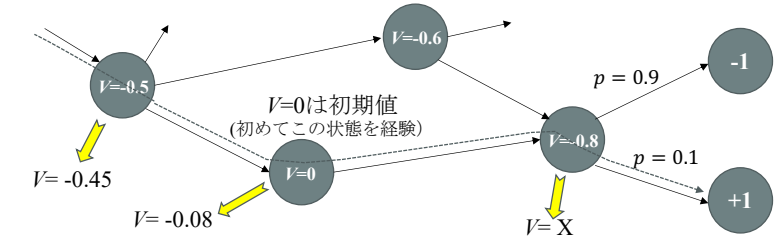
  if Terminal?[*e*] **then** …/* 前と同じ */

  return *V;*

- コメント：
  - 割引率γを導入。
  - TD学習法のアルゴリズムの核は変わらない

TD学習法の計算例 (2)

- 点線のように状態変化し、最後の状態で報酬 +1を得たとする。α = 0.1とする



(2) V= -0.5 + 0.1 (0-(-0.5)) =-0.45

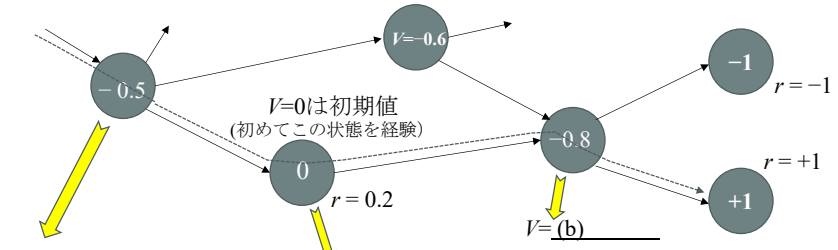
(3) V= 0 + 0.1× (-0.8-0) =-0.08

(1) 最後の状態では常に +1を受け取るので、その価値は1 (*V*=1)

課題５－３：割引率γ = 0.9を考えると上記の結果はどのように変わるか確認せよ。次スライド(a)(b) も同様に求めよ。

TD学習法の計算例

- エージェントの行動で点線のように状態変化し、最後の状態で報酬 +1を得たとする。α = 0.1とする



V= −0.5 + 0.1 (0.9*0 −(− 0.5)) = − 0.45

(1) 最後の状態では常に +1を受け取るので、その価値は1 (*V*=1)

Q学習（行動価値の学習）

Q-learning

状態ごとの行為の価値

- これまでは状態の価値を学習し、価値の高い状態へ移動することめざした（能動的環境）。
- 代わりに、ある状態でどの行為が大きな累積報酬期待値をもたらすかを直接学習する。
  - 状態s_iで、行為aの価値を$Q(s_i, a)$と表し、Q値とよぶ。
- 状態の価値とQ値の間には下記の関係がある。

$$V(s_i) = \max_{a \in A} Q(s_i, a) \qquad \text{式 (5.0)}$$

  - つまり、状態の価値は、最適な行為を選択したときに得られる価値である。前出（スライド：未知環境における行動選択）のa^*と比較すれば明らか。
- Q値を学習することをQ-learning（Q学習）と呼ぶ。

Q学習を行うエージェント

- 少し形式的に
  - エージェントの行為の集合をA、エージェントが観測しうる状態の集合をSとする。
  - ある時刻tで観測した状態を$s(t) \in S$とし、政策関数π (与えられたルール, 知識でもよい) を参照して行為$a(t)$を選択したとする。
  - 選択した行為$a(t)$を実行し、状態が$s(t+1)$ に遷移する。
  - これらを繰り返し、最終的にゴールに達成し、報酬を受けたとする。この報酬は行為の正しさを反映していると考え、それをもたらしたルールを強化する。これにより、状況ごとに選択すべき行為が強化され、その行動が起こりやすくなる。
- 状態から行動を選択する写像を政策関数と言った。

$$\pi: S \to A$$

これをπ で表す。これは学習により(良い方向に)変化する。

Q学習の利点

- 状態と行為を結ぶルール（関数）として学習できる。
  - このため、状態の遷移を表すモデル (状態遷移行列) Mを学習したり、与えたりする必要がない。
  - ただし、状態空間Sは必要。
  - 状態遷移確率もまとめてQ値として学習してしまう。
  - 特徴：
    - 割引報酬
    - 遅延報酬（これらはこれまでと同様）

Q学習 (Q-learning)

- 条件：未知なのは「環境」と「何を行動すべきか」である。
  - つまり、状態遷移確率Pと期待報酬rは未知だが、学習により状態sの割引報酬$V(s)$を最大にするような最適な政策(policy) を求めたい。
- 理想の（神様が知っている正しい）$V(s)$を$V^*(s)$と表すと、理想のQ値, $Q^*(s,a)$, は以下のように定義できる。

$$Q^*(s,a) = r(s,a) + \gamma \sum_{s' \in S} P(s,a,s') V^*(s') \qquad (5.1)$$

  - 確率的遷移なので期待値で計算する。($p^a_{s,s'}$は推定値)
  - Q値を式(5.1)を使って$Q^*(s,a)$に近づけたい。そのためには右辺の値を得て、それに近づける。

Q学習のアルゴリズム

- Q-learningのアルゴリズム (TD学習法)
  1. 任意の状態sと行為aに対して$Q(s,a)=0$と初期化する。
     - 実際には大きめの値を初期値にするとよいというtipもある。0にするとたまたま良い結果となった場合に、それを選択し続けるが、大きめの値にすると、初期段階で多くの行為を選択するから。
  2. 状態sで、ある戦略で行為aを選択し実行する。
  3. 環境から報酬$r(s,a)$ を得る。
  4. 行為sにより遷移した状態s'を観測し、
     $Q(s,a) \leftarrow Q(s,a) + \alpha \cdot (r(s,a) + \gamma \max_{a' \in A} Q(s',a') - Q(s,a))$
     と$Q(s,a)$を書き換える。（赤の部分は次ページ）
  5. $s \leftarrow s'$として2.に戻る。
  - なお、α $(0 < \alpha \leq 1)$ は学習率でQ値の更新の強さを表す。
  - 4 の式は変形すると以下のようにもかける。
    $Q(s,a) \leftarrow (1-\alpha) \cdot Q(s,a) + \alpha \cdot (r(s,a) + \gamma \max_{a' \in A} Q(s',a'))$

Q学習 (Q-learning) 続き

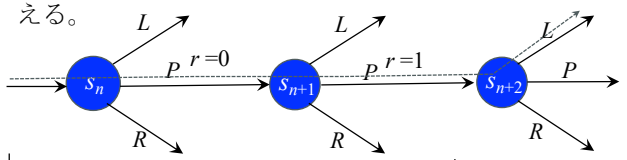
- 前の赤の項は式 (5.1) や 前のTDと比較すれば、本当は「次の状態」の価値$V(s')$ となるが、これを修正。
- Q値 $Q(s,a)$は、“状態sで行為aを実行”したときの価値であり、それは“その後、最適なpolicyを取り続けたときの利得の期待値”である。
  - 式(5.0)つまり、もしa^*を最適なpolicy π^*で選択したものとすれば、
    $Q(s,a^*) = Q(s,\pi^*(s)) = V^*(s)$
  - 最適なpolicy π^*はQ値が最大となる行為を取り出すものだから、
    $V^*(s) = Q(s,\pi^*(s)) = \max_{a \in A} Q(s,a)$
  - これから前の赤の項に置き換わり、$V(s')$を使わない式になる。
- また行動選択でも$Q(s,a)$を比較するだけなので、状態遷移確率は不要。

Q-learningの収束定理（紹介のみ）

- Q-learningの正当性を示す定理
  - Q-learningの収束定理： エージェントが十分な回数の行動を選択し、学習率αが時間とともに
    $$\sum_{t=0}^{\infty} \alpha(t) \rightarrow \infty$$
  - かつ
    $$\sum_{t=0}^{\infty} \alpha(t)^2 < \infty$$
  - のとき、Q-learningによりQ値は最適な値に収束する。(証明は略)
  - 実際には、行為の列は有限 (finite horizon)が多く、そのときも
  - この定理は成り立つ。

TD学習法によるQ値の計算例

- ロボットの行動：前進P、左に向くL、右に向くRを選択してゴールに達すると報酬20。途中に宝箱があり、それを通過しても報酬1をもらえる。



点線のように行為を選択して状態が遷移

$\alpha = 0.1, \gamma = 0.9$ とする

a	$Q(s_n,a)$
P	10
L	2
R	4

a	$Q(s_{n+1},a)$
P	8
L	6
R	5

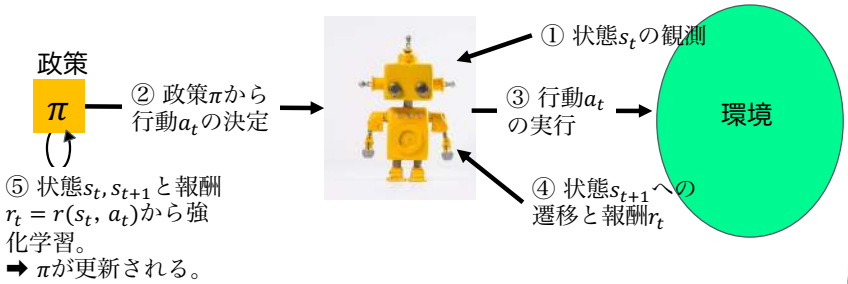
a	$Q(s_{n+2},a)$
P	6
L	8
R	3

⬅ 現在のQ関数を表す

- $Q(s_n, P)$=0.9x10+0.1(0+0.9×8)=9+0.72=9.72とQ値が更新。
- 課題５－４ ： $Q(s_{n+1},P)$の更新後の値を求めよ。

行動選択と強化学習

- 学習したVやQ値に基づいて政策（関数, policy） πを定義（学習）できる。
- と同時に学習した政策関数πから行動（列）を決める
  - エージェント行動しながら学習し、それが次の行動に影響する。これの繰り返し。



行動選択戦略（再び１）

- 学習結果と政策関数（行動選択）
- 「未知環境における行動選択」の「主な」行動選択と、Q-learningのアルゴリズム2の部分で、 πに基づいて行為を選択する部分。何を選択するか？
  →何を選択してもよい（が、ジレンマに気をつける）
  - ただし、V^*やQ値が未だ収束していない段階でもある程度多くの報酬を得るために以下の行動選択戦略をとる。（再掲）
- Greedy選択 (greedy 探索)
  - それまでのV^*やQ値に基づいて常に最適と思うものを選ぶ
- ε-greedy選択：
  - 確率εでランダムに行為を選択し、そうでないときはそれまでの段階でQ値が最大の行為を選択する。

行動選択戦略（再び２）

- ボルツマン選択（ソフトマックス, softmax）
  - たとえばQ学習であれば、行為aをexp($Q(s,a)/T$)に比例させて選択する。つまり行為a_0を

$$\frac{e^{Q(s,a_0)/T}}{\sum_{s \in S} e^{Q(s,a)/T}}$$

の確率で選択。Tの値は定数とする方法もあるが、時間とともに0に近づけることが多い。Tは温度ども言われる（温度を下げる）。
  - T=0のときは、$\varepsilon = 0$に相当する。
  - したがって、学習が十分に進んだときに他の選択肢を選ばなくなる。無駄なトライアルがなくなり、効率は上昇。
  - ただし、環境が変わり、現在の学習結果が最適で無くなったときに発見できない。（もし$\varepsilon > 0$なら、環境が変わり、別の選択肢でさらに良いものがあるときに、現在の学習結果を変えられる）

モンテカルロ法（紹介だけ）

- モンテカルロ法 (モデルを用いずに適当に動く)
  - 適当に初期状態s_0と政策関数（方策関数, policy） π (最初はランダム) を決定し、それに基づいて行動。ゴール状態s_Tに到達。
  - 実際に取った状態遷移と行動の列を考える。

$$s_0, a_0, s_1, a_1, \ldots, s_t, a_t, s_{t+1}, a_{t+1}, \ldots, a_{T-1}, s_T.$$

  - 状態s_tから最後までに得られた報酬をreward[s_t] (これはリスト) に追加。これを何回か繰り返し、 reward[s_t] の平均値を$V(s_t)$とする。この試行を繰り返す。
  - πは学習が進むにつれて変わる。
  - 実際の行動は、たとえば、 ε-greedy などによるランダム性もあるため、それまで学習した内容を常に反映するわけではない。
  - 多くのバリエーションがある。

SARSA法（紹介だけ）

- スライド「Q学習のアルゴリズム」の
  $Q(s,a) \leftarrow (1-\alpha)\cdot Q(s,a) + \alpha\cdot(r(s,a) + \gamma \max_{a' \in A} Q(s',a'))$
  の赤字の部分を修正。赤字は次では最適に行動を仮定している。
  SARSA (sarsa)は、ε-greedyなどによる実際の行動列に基づく。
- 初期状態s_0からゴールs_Tまで、実際の状態遷移と行動の列を考える。
  $s_0, a_0, s_1, a_1, \ldots, s_t, a_t, s_{t+1}, a_{t+1}, \ldots., a_{T-1}, s_T.$
- この列から、
  $Q(s_t,a_t) \leftarrow (1-\alpha)\cdot Q(s_t,a_t) + \alpha\cdot(r(s_t,a_t) + \gamma Q(s_{t+1},a_{t+1}))$
- 上記の右辺は、$s_t, a_t, r(s_t,a_t), s_{t+1}, a_{t+1}$から得られるのでSARSAという。
- モンテカルロ法風に、一連の行動後に最後から計算することもある。
  $Q(s_{T-1},a_{T-1}) \leftarrow (1-\alpha)\cdot Q(s_{T-1},a_{T-1}) + \alpha\cdot r(s_{T-1},a_{T-1})$
  $Q(s_{T-2},a_{T-2})$
  $\leftarrow (1-\alpha)\cdot Q(s_{T-2},a_{T-2}) + \alpha\cdot\left(r(s_{T-2},a_{T-2}) + \gamma Q(s_{T-2},a_{T-2})\right)$
  …

政策オン・オフ

- 政策オン：簡単には実際の行動に基づく
  - SARSA法やここで紹介したモンテカルロ法は、政策関数πで選択した実際の行動に基づき、その列に沿ってQ値などが更新される。
  - たとえば、SARSAのスライドの青の部分は、次の状態でたまたま選んだかも知れない行動a_{t+1}を使用している。
- 政策オフ：実際の行動に基づくわけではなく、最適な行動をしたとして更新する。
  - たとえば、TD法（SARSAのスライドの赤の部分）は、次の状態で最善の行動を選択したとしている。
  - 政策オンでは、次の状態で不適切な行動を取る可能性もあるため、それをも回避するようなやや注意深い行動を学習する傾向がある。ただし、どちらが良いかは、問題ごとに考える。

深層強化学習

- ここまでの方法は、状態sの価値$V(s)$や、行動価値関数$Q(s, a)$を配列や表などにまとめる計算方法である。
- しかし、状態数が大きいとすべての状態を尽くしてこれらの値を求めるには、莫大な時間がかかる。配列や表として表現するメモリも大きい。
- この配列や表に相当する部分をニューラルネットワークで表現する。
  - ネットワークの入力には、現在の状態sを与え、出力は
    - $Q(s, a)$ ：それぞれの行動に対する価値を出力する
    - $\pi(s)$: 政策（方策）関数を学習し、行動を出力

  など。非常に多くの工夫があり、現在、世界中で研究が進められている。